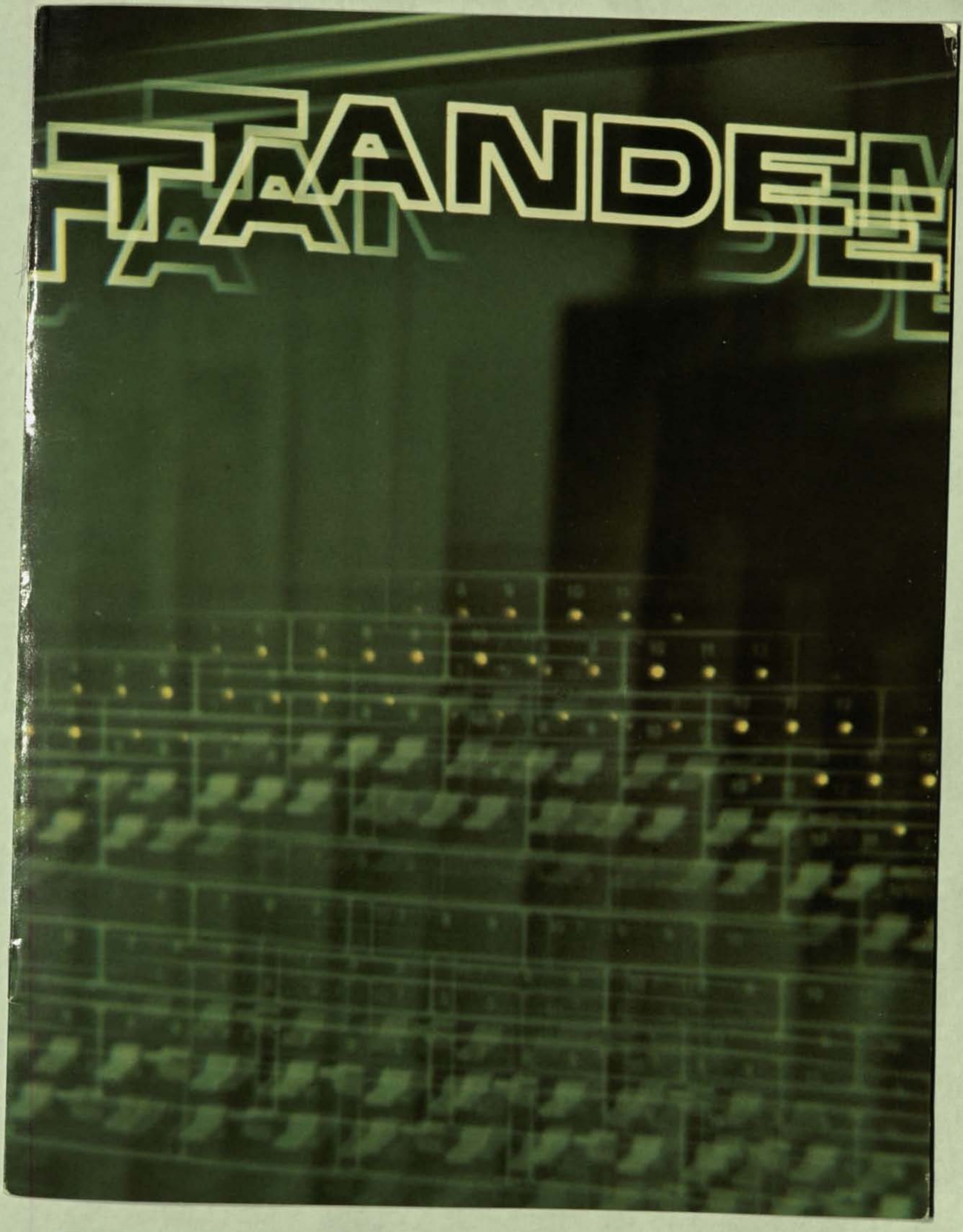
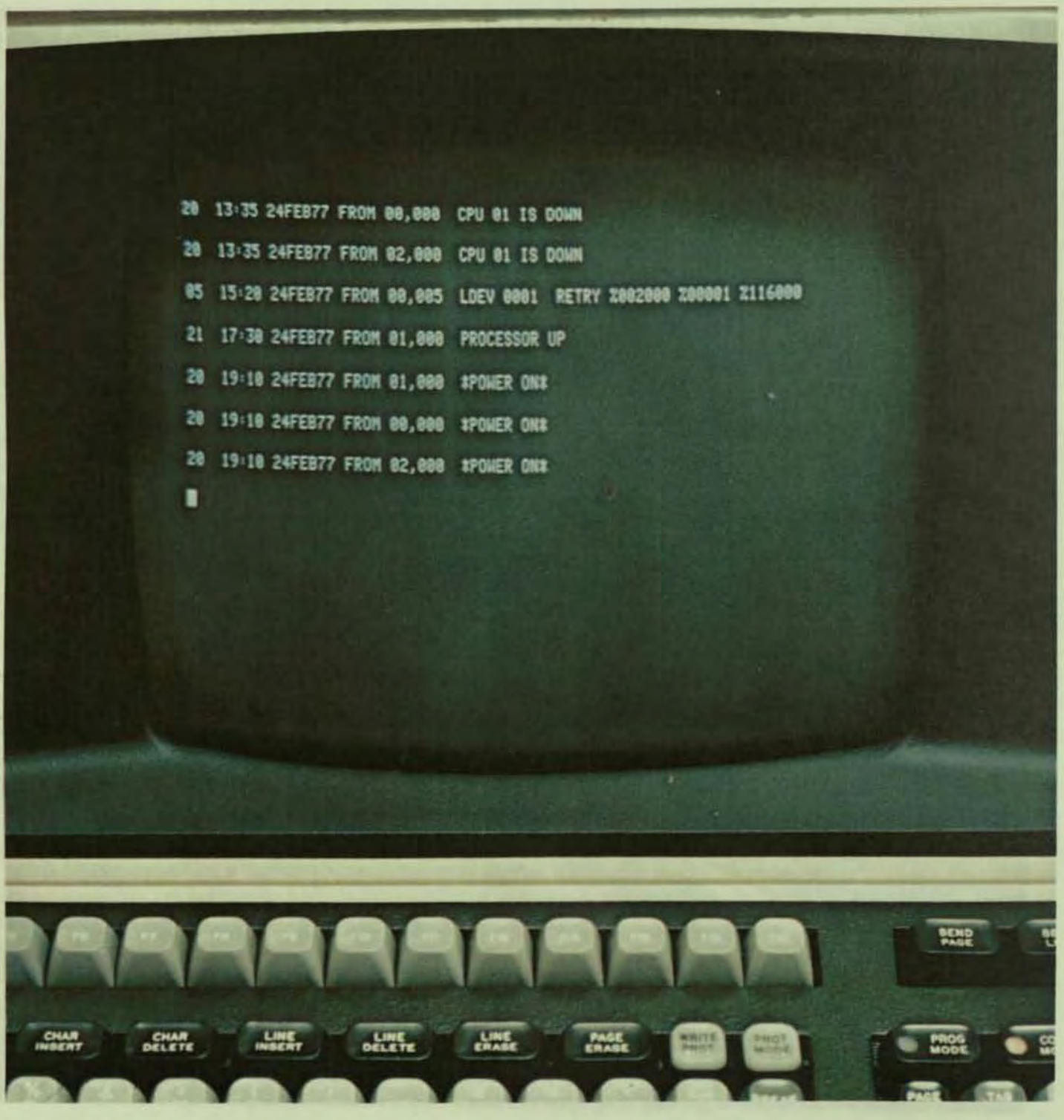

TANDEM

13:35 *Processor 1 fails, messages are printed from both processor Ø and processor 2 noting this fact. Effect on system users— none.*

15:20 *Parity error during a disc read, retry was successful. Effect on system users — none.*

17:30 *Processor 1 repaired and re-incorporated into the system. Effect on system users— none.*

19:10 *Complete system power-interrupt and restart. Effect on system users— none.*



TANDEM NonStop™ Systems.

The age of on-line computers.
It's begun in earnest now, and the reasons are sound. Dedicated, efficient, low-in-cost, today's small computers provide immediate, positive response in all types of applications.

Credit verification. Bank deposits and withdrawals. Funds transfers. Order processing and inventory control. Medical systems. Retail sales. Emergency vehicle dispatch. Theater and sports events ticketing. Hotel and motel reservations. Communications networks. Manufacturing and material control systems. Wherever there's a high volume of transactions requiring speed and accuracy; wherever there's a critical need for reliability and expandability; wherever efficient, low cost transaction processing is essential; Tandem's unique multiple-processor architecture and system software offer a new and powerful solution. And it's field-proven.

On-line means on-demand.
And there's the rub. For as reliable as the modern computer is, and it is remarkably reliable, it will on occasion fail. Which can cause lost business, or missed schedules, or unhappy customers, or costly errors, or worse. In live data base systems, a crash during processing can cause untold damage to, or even destruction of, the data base. That's a nightmare. All of which is why Tandem came into business.

The search for alternatives.
Until now, the only way an on-line computer user could protect himself against the possibility of a computer failure was to install a "back-up" system, typically a second processor strapped into the system, ready to come on-line when a failure occurred. It required special interfaces and customized system software; and there was the penalty of system inflexibility, loss of processing power, and uncertainty as to the status of transactions-in-process when a failure occurred. But aside from manual back-up systems, which went rapidly out-of-date, and were slow, inefficient and uncertain, it's all that there was. Until Tandem.

NonStop Computing.
Tandem has designed and built the first multiple processor system designed from scratch to provide non-stop processing —even during a failure—with no penalties in the speed, capacity, throughput or memory utilization of the system. With no strapped-up interfaces and no customized software, and perhaps most important, no loss of system flexibility. The Tandem NonStop System can expand from a basic two-processor system all the way to sixteen processors, without reprogramming, without customizing, and without penalty on the original investment.

NonStop Protection.
And because of its unique design, the Tandem NonStop System provides an unprecedented level of protection for both transactions and files. No module failure need result in a transaction being lost or duplicated. With Tandem, restart time and restart uncertainties are eliminated; recovery is instantaneous. No other system offers this protection or this capability.

Tandem NonStop Computers.
The one and only answer from the computer industry to the opportunities and problems of today's advanced computer applications. Data Base Systems. Distributed Systems. Data Communications Systems. Multi-Terminal Systems. All of these have grown out of the enormous capabilities of the computer itself, but as of today, only one company has dedicated itself to the design and construction of efficient, low-cost systems which meet their needs head-on. Tandem Computers. Up and running. NonStop.

Tandem four processor system, expandable to six in same cabinetry, expandable to sixteen processors with additional cabinetry.

Clockwise from upper right:

View of back of rack showing construction for a 3 processor system.

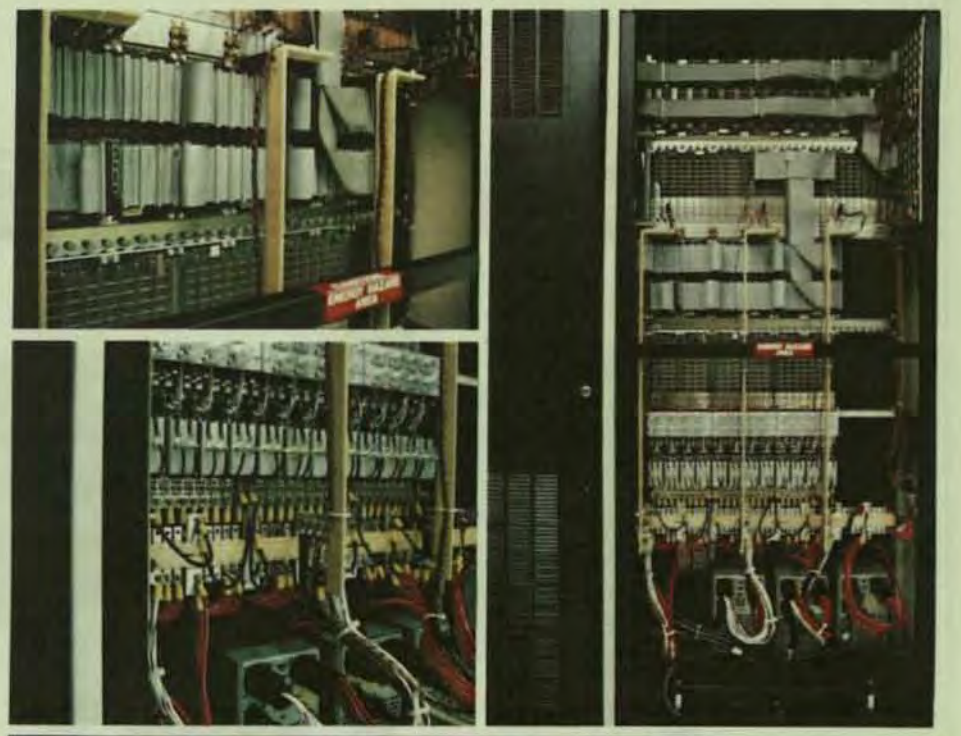
Dual independent, separately controlled interprocessor busses.

Input/output controllers power switching and load sharing circuitry.

Independent I/O channels connected to separate independent ports of the I/O controllers.

The benefits of NonStop Computing.

- Hardware and software are designed to work together continually, productively and efficiently.
- The system is designed around multiple, independent processors to provide continuous operation regardless of a failure anywhere in the system.
- A failure in one part of the system will not contaminate the rest of the system.
- On-line maintenance and replacement of failed modules are completed without shutting the system down. Operators at terminals and programs in process are unaffected by either the failure or its repair.
- The standard operating system software program is designed and tested to provide continuous operation. Single hardware failures are automatically dealt with, instantaneously. This is in marked contrast to most operating systems, which are designed to shut the system down in a failure, rather than actively respond to it.
- The system is expandable to meet changing or growing needs without hardware cost penalty, and without having to reprogram. Instead of having to invest initially for an anticipated later need, the user can start with the exact amount of computing power he needs, and add in increments, at mini prices, all the way to a full 16 processor system. Fully expanded, a Tandem system can support 2048 data communication lines, individual files of four billion bytes fully supported by the data base record manager, providing continuous real-time operation, with one of the finest packages of software available in any computer system. And it's all NonStop. From Tandem Computers.



NonStop Hardware

Multiple independent processors.
In order to insure that the loss of a processor will never completely isolate the peripheral devices attached to any controller, each Tandem controller module has two independent ports which allow it to be connected to two Tandem processor modules. In the event of a failure of that processor or I/O path, the other processor takes control automatically to insure that the system will always have a communications path to that controller.

All processor modules are interconnected through a unique, high-speed dual bus structure—Dynabus.™
Each Dynabus path is fully autonomous, operating independently of, but simultaneously with, the other bus to insure that two communications paths exist between all processor modules in the Tandem 16 NonStop system. All Dynabus interprocessor transfers are hardware controlled, independent of, but concurrent with, all normal I/O transfers and other CPU activities. Since Dynabus handles interprocessor transfers at 26 megabytes/second, this produces extremely low overhead interprocessor communications at speeds fast enough to simultaneously handle peak transaction and message loads even when the Tandem system is fully expanded to sixteen processors.

The Tandem processors are independently powerful.
Designed to increase transaction throughput and minimize system overhead, the Tandem processors separate I/O processing from interprocessor communications and CPU workloads. Each processor can handle up to 128 19.2 K Baud communications lines, a message handling capability miles ahead of other processors in their price category. The CPU segment of each processor is a pipelined microprogrammed unit designed specifically to handle business transaction data applications. With a cycle time of 100 nanoseconds, each CPU can, for example, add two 18-digit numbers in under 2.4 microseconds.

Time robbing I/O overhead is completely removed from the main processor.
Each processor contains an independent microprogrammed I/O processor which controls the 4 megabyte/second block multiplexed I/O channel. With truly large computer capability, each of these I/O channels can handle up to 32 device controllers. Each Tandem processor incorporates a powerful memory system capable of complete single-error correction and multiple-error detection, performed using semiconductor memory at a cycle speed of 500 nanoseconds. Because of the virtual memory management feature, applications programmers need not worry about many of the constraints of physical memory.

NonStop Software

Guardian controls the traffic and watches over the system.
Guardian is Tandem's complete, transaction-oriented operating system. It resides in each processor, and has the capability to respond positively to a failure anywhere in the system. With Guardian, there is no need for customized or special operating system development by the user. Since programs are geographically independent, the task of applications programming is vastly simplified for the user. Further, Guardian carries the user-assigned priorities for applications programs, handles all communications among programs and between programs and the outside world, and extends the capability to start program execution in any available processor module from any processor.

Geographical independence of programs and data is a noteworthy feature of Guardian. Programs are not only unaware of which processor is running them; they may well be running simultaneously on all processors. And programs can access any device in the system, even those not physically connected to the processor(s) running the program. Because of this geographic independence, the Tandem NonStop system can be expanded all the way to sixteen processors without any reprogramming.

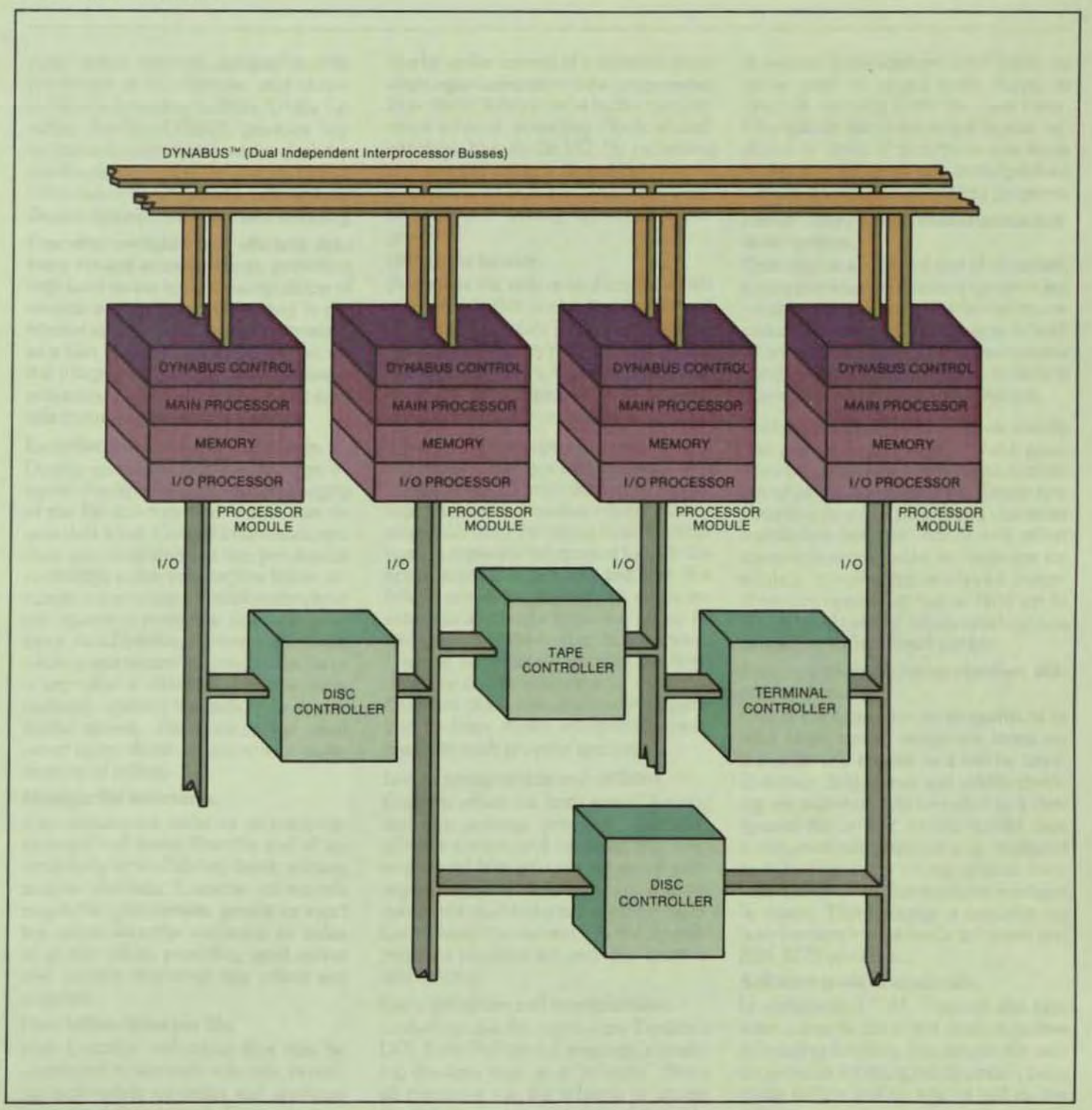
Multiprocessor message system.
Guardian automatically handles all communications between Tandem processor modules, system processes and applications programs—routing messages to the correct processor, verifying correct receipt and deciding which program receives the message in the destination processor.

Guardian provides protection, performing comprehensive data validation on all transfers. It was designed to detect and isolate any faulty module, preventing corruption of any other module by a "mad" processor. Guardian has the inherent ability to detect an error or point of nonresponse anywhere in the system, log the failure and disallow access to the faulty module without disturbing system operation. The Guardian operating system, along with the system hardware architecture, is the basis of Tandem's NonStop operation.

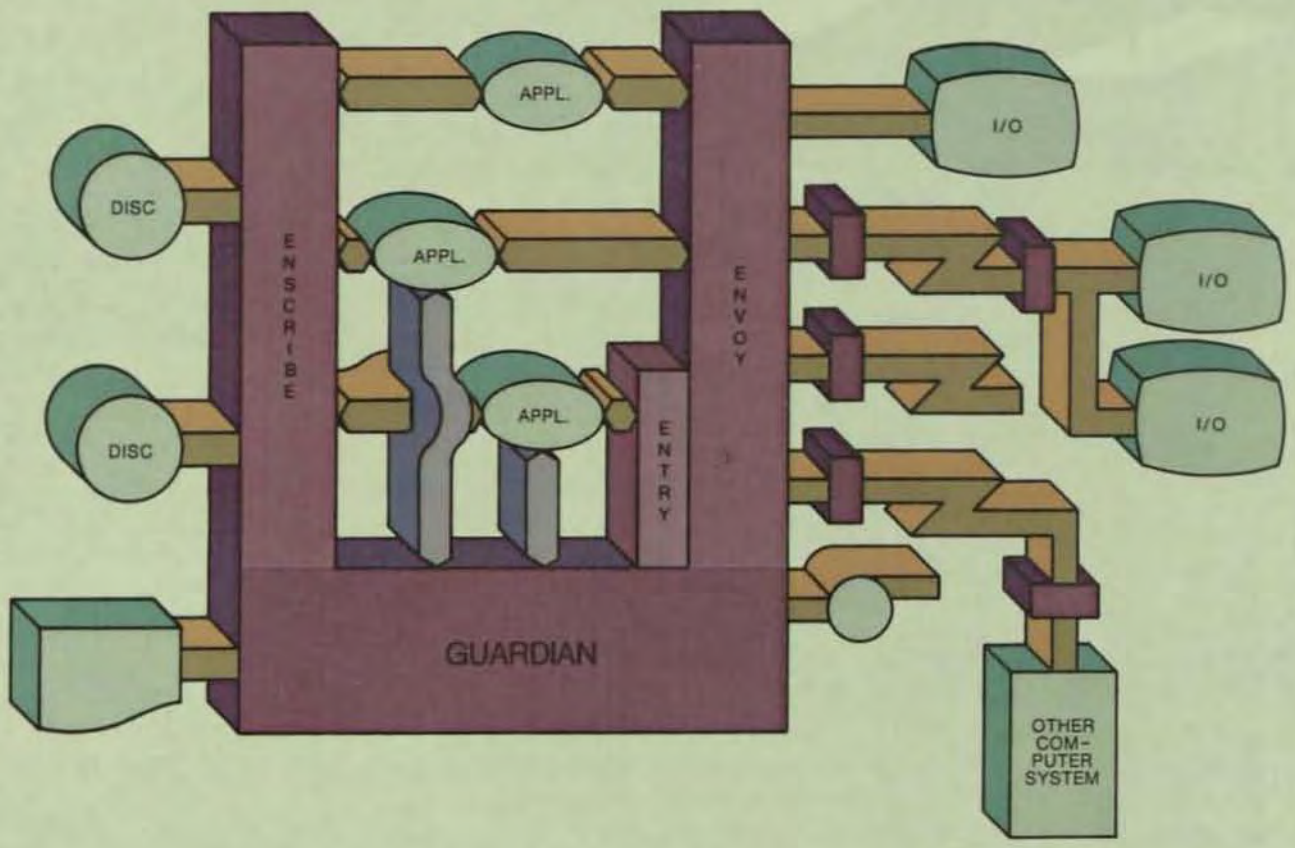
Tandem's Transaction Application Language.
T/TAL. A powerful block structured language designed for fast, flexible programming. T/TAL is self-documenting and is easy to read, modify and maintain. Developed by Tandem, T/TAL gives the programmer every vehicle to optimize the hardware potentials of its interactive, multi-processor environment.

Tandem/COBOL
The only COBOL available for multiple processor systems, Tandem/COBOL (ANSI X3.23-1974) utilizes all the capabilities of Guardian and Enscribe. Under Guardian, Tandem/COBOL features NonStop operation; shared, re-entrant

NonStop Hardware



NonStop Software



Program Development Building Blocks

Printed in U.S.A.

code; virtual memory; geographic independence of I/O devices; and checkpoint/checkmonitor facilities. Under Enscribe, Tandem/COBOL provides key-sequenced, entry-sequenced and relative file structures; logical file size up to four billion bytes; up to 255 alternate keys per file; and optional mirror data base recording.

Enscribe: versatile and efficient data base record management, providing high level access to and manipulation of records in data bases. Operating in distributed fashion across multiple processors as a part of Guardian, Enscribe ensures the integrity of the data base in case a processor, an I/O channel or a disc drive fails during transaction processing.

Enscribe protects the file structure.
During operations, Enscribe performs internal checkpointing to ensure integrity of the file structure and ensure that no user data is lost. Control information and data are maintained in two processors controlling a disc volume; if a failure occurs in one processor, Enscribe completes the operation using the alternate processor. And Enscribe maintains all indices; when a new record is added to the file or a key value is changed, Enscribe automatically updates the indices to the affected records. The programmer need never worry about assignment or maintenance of indices.

Multiple file structures.
Key-sequenced, relative or entry-sequenced—all under Enscribe and all accessible by up to 255 key fields, primary and/or alternate. Location of records may be by approximate, generic or exact key value. Enscribe maintains an index of all key values, providing rapid access and update whenever key values are supplied.

Four billion bytes per file.
With Enscribe, individual files may be partitioned in separate volumes, providing large system capacities and significant increase in throughput. Each partition can be under control of a separate processor, again transparent to the programmer. Enscribe includes a cache buffer management scheme, providing "look ahead" capability to optimize I/O. By increasing memory in the cache, throughput can be increased and there is no need to modify or recompile existing applications programs.

Mirrors to be sure.
Tandem is the only manufacturer which can provide this protection for Virtual Memory: if a failure occurs in the System Disc, Tandem's Mirror Capability prevents not only a shutdown, but prevents loss of any part of the operating system or application programs.

Enscribe offers an optional mirror volume technique whereby a disc volume's data can be physically recorded on two separate disc packs simultaneously. Reads may occur from the closest head in either pack, completely transparent to both the applications program and the user. If a failure occurs in one disc, all reads are automatically made from the other. As soon as a failed disc is restored, Enscribe automatically updates the failed device to exactly mirror the safe volume; this takes place concurrent with application updates. Again, completely transparent to both program and user.

Locks, compression and utilities.
Enscribe allows for both record locking and file locking, providing flexibility of both access and security. For key-sequenced files, an optional set of techniques provides for both data and index compression, thereby reducing the number of head movements. A file update program provides for easy file creation and loading.

Data definition and manipulation.
Under Enscribe, the user can use Tandem's DDL Data Definition Language, describing the data base as a "schema." Since all programs use the schema to access the data base, a correct view of the data is assured. It also defines which fields are to be used as access paths (keys) to retrieve records from the data base. Changes to data base record layout, additions of types of records or new fields within record types are accomplished without modification of existing programs.

Envoy: multifaceted data communications system.
Operating as an integral part of Guardian, Envoy provides the interface between applications programs and data communications networks. Envoy supports both binary synchronous and asynchronous communications, with single or multi-drop lines on either a local or remote basis.

Data is transferred from terminals directly into main memory, which means processors are not interrupted until a complete message has been received. Binary synchronous terminal polling and character translation between ASCII and other communications codes are hardware executed, minimizing overhead. Asynchronous operations run at rates up to 19.2 K baud per line; binary synchronous at rates up to 56 K baud per line.

Entry: page mode forms creation, display and access.
One of the easiest-to-use programs of its kind. Users simply design the forms on the screen to appear as it will be used. Delimiters, field names and validity checking are automatically recorded in a designated file on disc. In use, invalid data is automatically checked and displayed as a flashing entry on the screen. Individual fields in any form may be accessed by name. This package is available for both Tandem's page mode terminals and IBM 3270 terminals.

Software tools to work with.
In addition to T/TAL, Tandem also provides a source file editor and interactive debugging facilities. The source file editor provides for string manipulation, page mode editing and context as well as line editing.

Dual processor system cabinet utilizing standard 19" rack mountable modules.

Clockwise from upper right:

All system modules may be replaced on-line.

Maintenance panel shown in use.

Regularly scheduled classes are held for both programmers and customer service engineers.

Incoming inspection at the factory.

Maintenance panel facilitates on-line diagnosis and repair while the system is up and running.

Expansion, Service and Maintenance – All NonStop.

Start with what you need.
Tandem's NonStop Computer System is the only computer system on the market which allows you to start with only the computer power you need right now, yet grow as your needs grow—easily, economically and without modification to your programs, either systems or applications.
Through incremental expansion all the way up to sixteen processors, virtually unlimited system growth is assured. Each additional processor module simply increases the system's processing power, input/output capability, and available memory. Allocation of processors to primary and secondary tasks (file editing, report writing, etc.) can be pre-programmed to take place at specified times with no need for user interaction.

Service during On-Line operations.
With any system a hardware failure must be repaired. But only with Tandem can the system keep right on operating, right through the failure and right through the repair too. Tandem's customer service representative can remove and replace any failed module in your system without interrupting service. The operators at terminals and the programs in process are unaffected by either the failure or replacement of the failed module. Without this feature, no system can truly be called NonStop.

Maintenance without shut-down.
Routine maintenance, too, can be accomplished without anyone using the system being aware of or inconvenienced by the process. All programs, the data base, and system capability are maintained in operational status during the entire procedure.

The Tandem 16 NonStop System—conceived for today's and tomorrow's needs, designed for efficiency, built for reliability, and serviced without interruption. A field-proven solution.

Tandem Computers, Inc.
19333 Vallco Parkway
Cupertino, California 95014
Toll Free 800-538-9360 or
408-996-6000 in California
Telex: 352044

Tandem Computers GmBH
Bernerstrasse 50, Frankfurt 56, West Germany
5072071-73, Telex: 416247-tacu-d

TANDEM / 16

SYSTEM OVERVIEW

A. INTRODUCTION - FAILSAFE PROCESSING

B. AREAS OF POTENTIAL FAILURES IN MULTIPROCESSOR SYSTEMS

C. THE TANDEM SYSTEM
   1. TANDEM PROCESSOR (CPU)
   2. TANDEM INPUT/OUTPUT CONTROLLERS
   3. TANDEM OPERATING SYSTEM
   4. NONSTOP APPLICATION PROGRAMS

SYSTEM OVERVIEW

A. FAILSAFE PROCESSING

The Tandem/16 system is a multiprocessor computer system that provides *NONSTOP* system performance.

The factors to be considered in any multiprocessor system that is intended to continue operating in the event of failure may be defined under the headings of '*HARDWARE*' and '*SOFTWARE*'.

Under the headings of *HARDWARE*, the following points are desired:

- No non-duplicated hardware modules
- A design such that no hardware module can affect others if it fails
- Multiple paths between all modules
- No critical power supplies in the system
- Very high inter-module transfer rates (to avoid excessive overhead involved in keeping all modules updated with the current system status
- All repairs should be capable of being effected without shutting down the system

The *SOFTWARE* factors to be considered are the following:

- A true multiprocessor operating system is needed
- The operating system should change message routes (on behalf of the application programs) to avoid failed components
- The operating system should be able to re-configure itself "on the fly"
- No critical processors (system masters) should be included that can corrupt the whole system
- The failure handling and recovery should not be left to the application programmer
- The application programs should be able to be written without the programmer having decided on a particular configuration

- All the programs should be written in a high
- level language
- The application programmer should *NOT* have to be aware that he is using a multiprocessor system

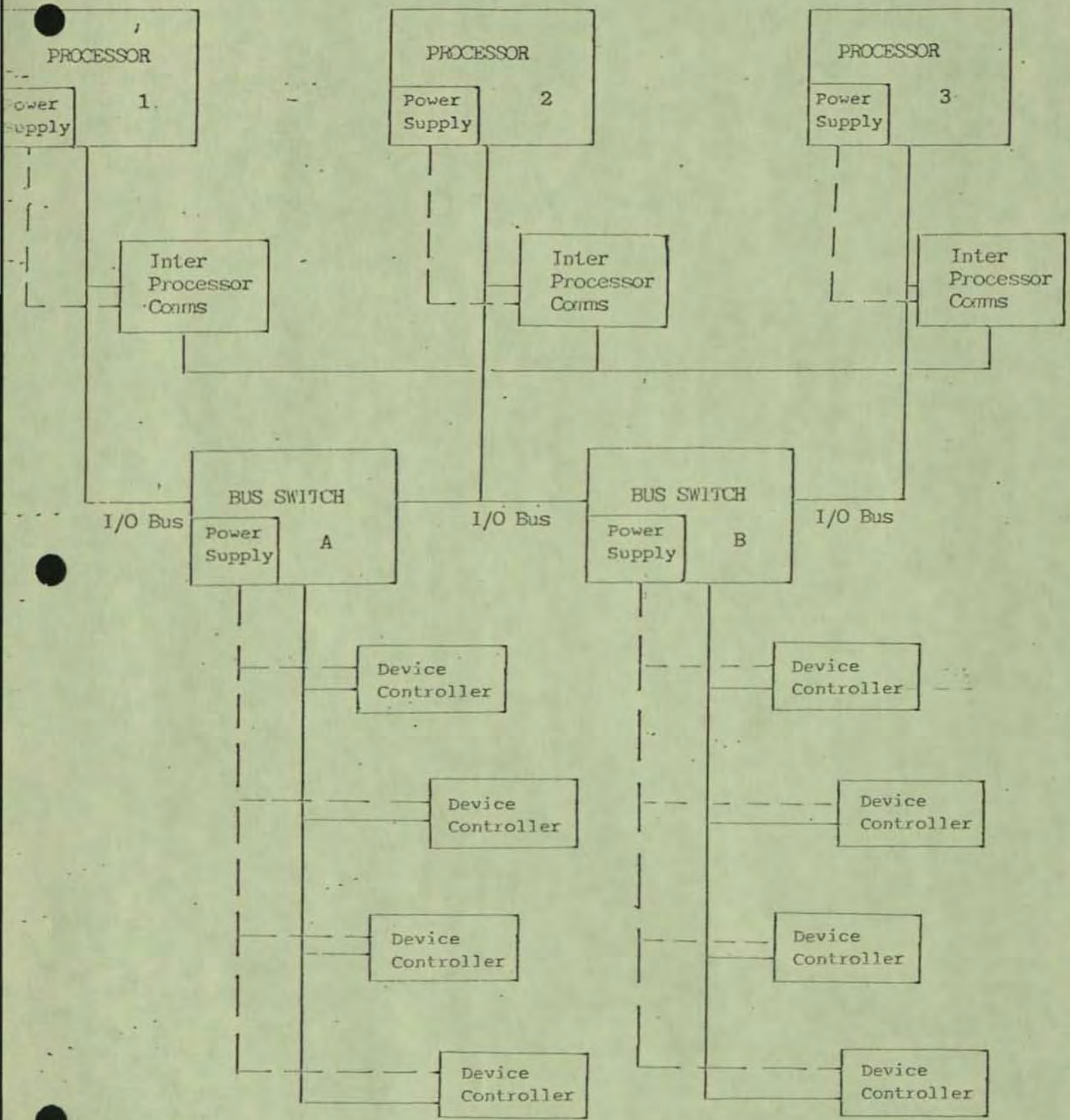
B. POTENTIAL FAILURES

- A typical system (supplied by many manufacturers) is shown in Figure A.

The potential for failure inherent in this typical design is quite alarming:

- The interprocessor bus may fail due to the failure of any of the interprocessor communications interfaces. This renders the whole system inoperative in many cases. It is true that on some systems a second bus can be added, but this will leave the problems of bus switch-over and load sharing to the application programmer
- If processor 2 has a fault (hardware or software), then the possibility exists that it can keep control of both bus switches, and hence stop the system
- If the power supply in a bus switch fails, then all of the devices attached to it are lost
- Bus switch A can fail in such a way as to corrupt Processor 2's I/O bus and thereby leave Processor 3 to handle the complete system load with only half of the peripherals
- Any device controller may fail in such a way as to corrupt all transfers to other device controllers on the same bus, thereby losing half of the peripheral devices
- Interprocessor rates are typically slow (around 1-2 Mega bits/second), thus increasing system response time and overhead
- The racks that the systems fit into normally require that the system (or a major part of it) is shut down for many failures (fans, etc.)

FIGURE A


PROCESSOR
ower
upply
1
PROCESSOR
Power
Supply
2
PROCESSOR
Power
Supply
3
Inter
Processor
Comms
Inter
Processor
Comms
Inter
Processor
Comms
BUS SWITCH
Power
Supply
A
BUS SWITCH
Power
Supply
B
I/O Bus
I/O Bus
I/O Bus
Device
Controller
Device
Controller
Device
Controller
Device
Controller
Device
Controller
Device
Controller
Device
Controller
Device
Controller

- The operating systems supplied normally consist of single processor operating systems coupled to I/O drivers for the interprocessor communications
- To obtain a satisfactory level of reliability it is usually necessary for the application programmer to code certain routines at assembler level
- It is normally required that the application programmer be fully aware of the complete system configuration
- Message re-routing to avoid failed components is the responsibility of the application programmer
- Only a very few systems can support on-line exercisers and diagnostics
- Accessing peripherals that are not connected to the same processor as the application is running in, is extremely difficult
- A *"MAD"* processor can often corrupt the entire system by generating bogus messages

C. THE TANDEM SYSTEM

All of these, and many other factors, were considered before the *TANDEM* System was designed. Solutions to all of these problems were built-in to the *TANDEM* hardware and software. Because the *TANDEM* System was designed from scratch for reliability, it has a calculated system MTBF which is significantly greater than other systems.

The *TANDEM* system pictured in Figure B depicts the general architecture of the T/16. To better explain how the T/16 provides NonStop operation a small two processor system is shown (Figure C). Additionally Figures D through P along with the associated text give a brief description of Tandem's hardware and software and finally how a simple NonStop program would perform in a *TANDEM* system.

FIGURE B

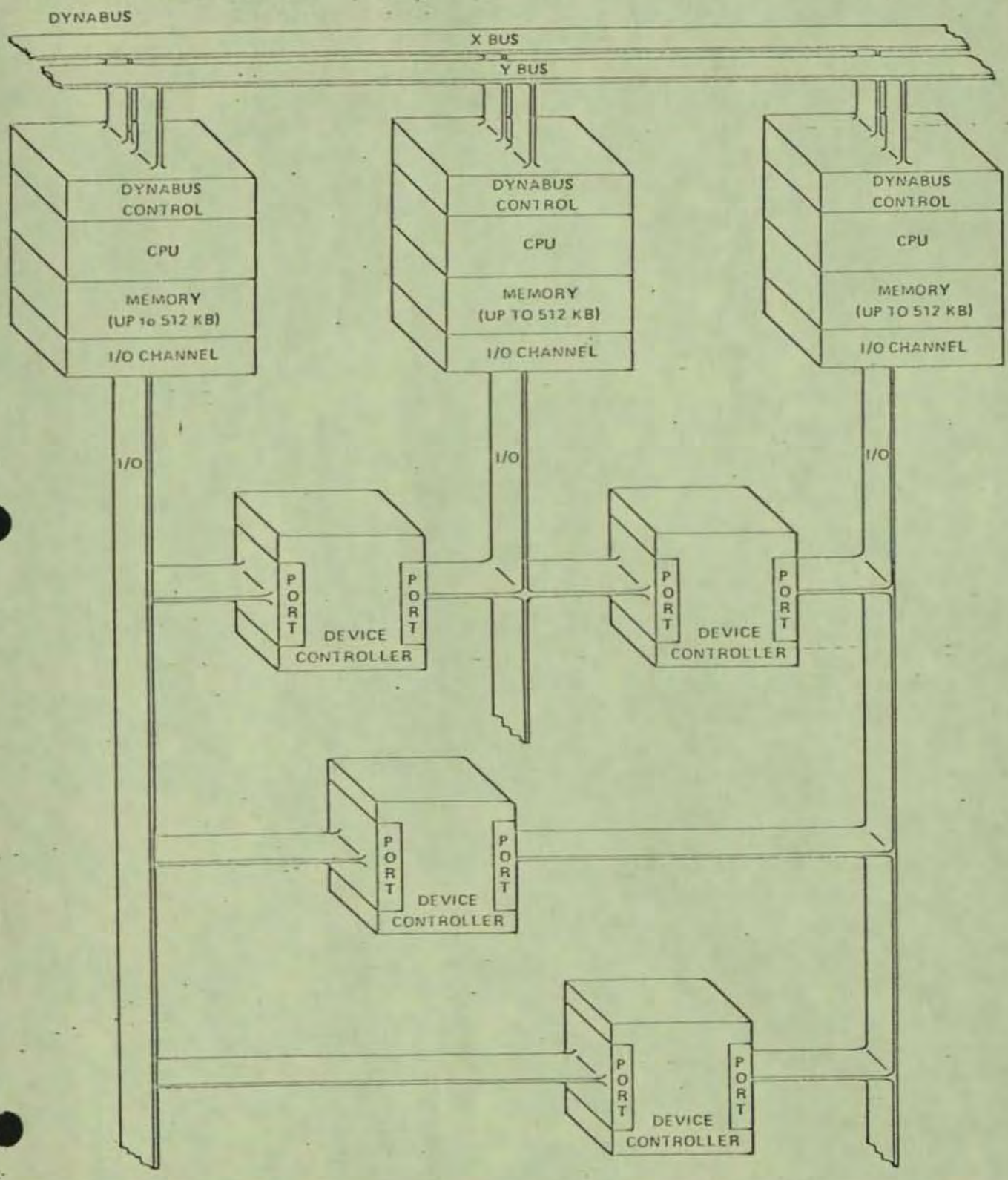
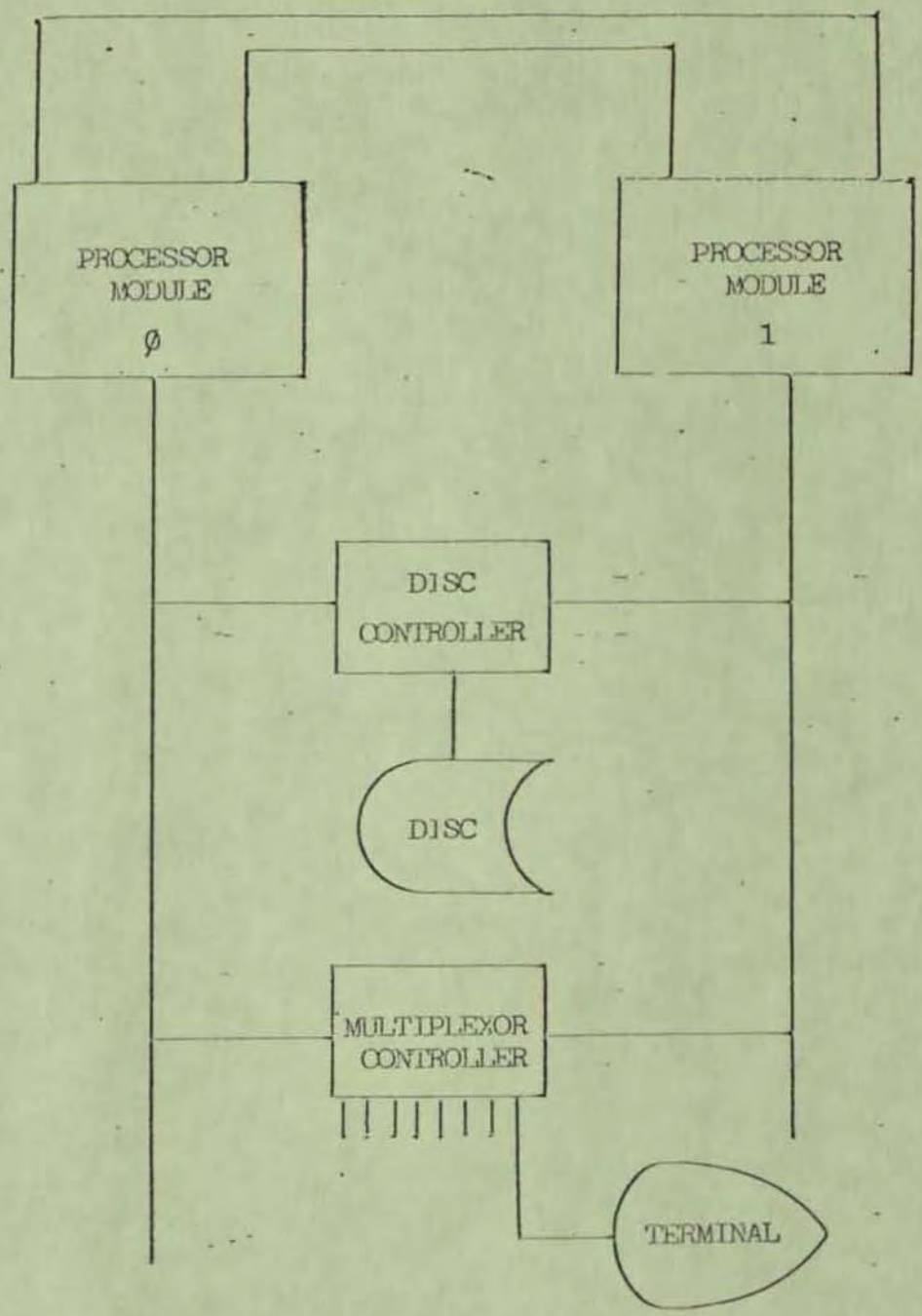

DYNABUS
X BUS
Y BUS
DYNABUS CONTROL
CPU
MEMORY (UP to 512 KB)
I/O CHANNEL
DYNABUS CONTROL
CPU
MEMORY (UP TO 512 KB)
I/O CHANNEL
DYNABUS CONTROL
CPU
MEMORY (UP TO 512 KB)
I/O CHANNEL
I/O
I/O
I/O
PORT
PORT
DEVICE CONTROLLER
PORT
PORT
DEVICE CONTROLLER
PORT
PORT
DEVICE CONTROLLER
PORT
PORT
DEVICE CONTROLLER

FIGURE C



A SMALL TANDEM CONFIGURATION

1. PROCESSOR

A processor module is comprised of the following:

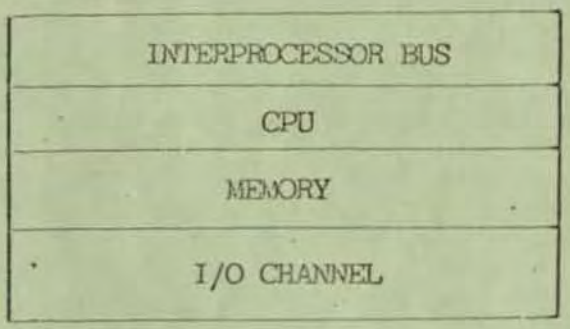


FIGURE D

. *INTERPROCESSOR BUS:*

The interprocessor bus is used for transmitting data between processor modules. Two communication paths are provided between every processor module (CPU) providing redundancy. The bandwith of the interprocessor bus is 26 megabytes/second. A 12 processor system will operate without degradation with only one bus operative.

. *CENTRAL PROCESSING UNIT (CPU):*

The CPU is a pipe lined micro processor- that is, one instruction is being executed while another is pre-fetched and decoded. The CPU executes programs by fetching and executing instructions from memory. Arithmetic computations are performed in the CPU. Additionally, a single instruction is executed to transmit data from memory, via the interprocessor bus, to memory in another processor module. Likewise, a single instruction is executed to cause data to be transferred between an I/O device and memory. Once either of these instructions has been issued the CPU is free to proceed with the other activities while the actual data transfers are taking place.

MEMORY :

Main memory can store up to 512 K bytes of information per processor. Data received from other processor modules is stored away without interferring with CPU program execution. Similarily, data is transferred between I/O devices and memory concurrently with CPU program execution (all data transmission is accomplished via DMA). Semi conductor memory cycles at 500 NANO seconds and is a 22 bit word. Six (6) bits are used for error detection and correction. We guarantee detection and correction of all single bit errors and detection of all multiple bit errors.

INPUT/OUTPUT CHANNEL:

The I/O channel is used for transmitting data between a processor module's memory and its I/O devices. The I/O has its own micro processor to facilitate I/O operations independant of each processor module. The bandwidth of the I/O bus is 4.0 megabytes with semiconductor memory.

2. INPUT/OUTPUT CONTROLLERS

The I/O Channel block multiplexes all peripherals attached to its channel. Block multiplexing is better understood with a definition of Tandem's peripheral controllers.

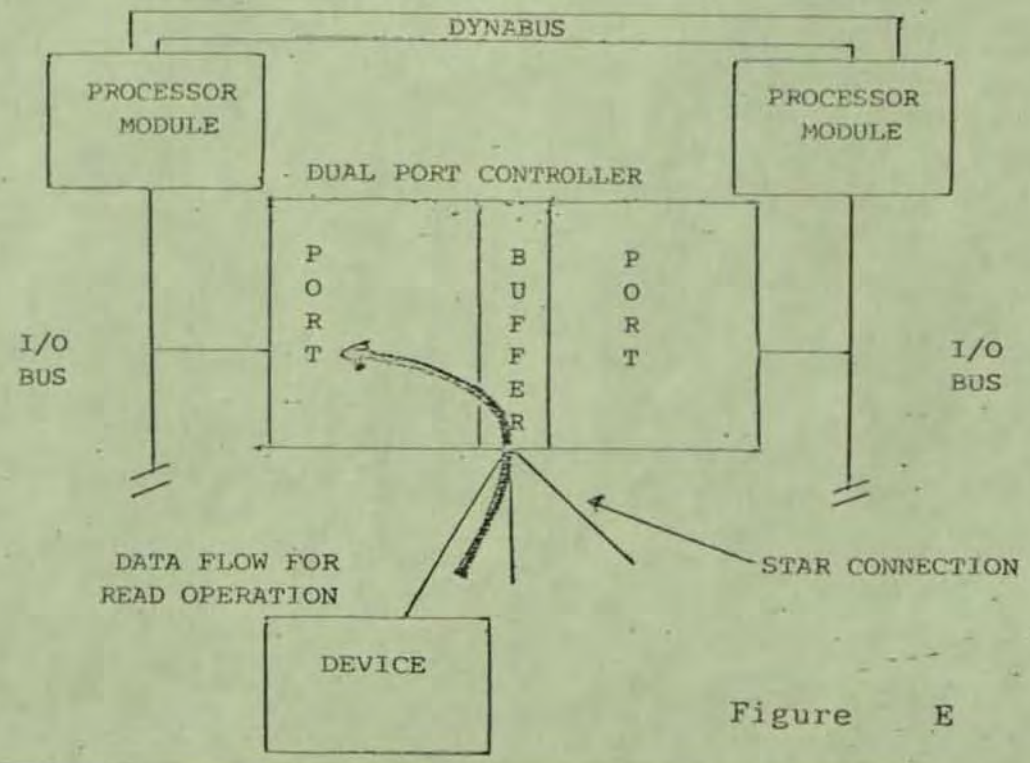


Figure E

The design of all Tandem dual port controllers is conceptually the same (as depicted in Figure E). Each controller has two ports. This allows two processors to have access (ownership) of the peripheral at any given time. The operating system controls processor ownership. When an input/output activity has been initiated (in this case a read), the I/O channel initializes the controller (executes one I/O instruction). The controller then initiates the device. The device then moves data into the controller buffer on an uninterrupted basis for the entire block (up to 4 K bytes). As the buffer begins to fill, the controller issues a 're-connect' command to the I/O channel. The I/O channel being much faster than the device, quickly empties all data to memory and 'disconnects' from the port. (Keep in mind that the controller buffer is continually being filled by the I/O device without interruption). Then the I/O channel goes on to service another controller if a 're-connect' command has been issued. This is basically how the I/O channel block multiplexes peripherals attached to it. Keep in mind that peripherals can run at a maximum speed uninterrupted for the entire length of a transfer (up to 4 K bytes).

The 10 mega byte disc controller services up to four(4) discs connected in a 'star' fashion. This controller has a 32 byte buffer. The 60 mega byte controller also has a 32 byte buffer.

The magnetic tape controller supports up to two(2) NRZI tape drives connected in star fashion. The buffer size for the tape controller is 32 bytes.

The line printer controller services up to two(2) line printers connected in star fashion. The buffer size is 32 bytes for this controller.

The asynchronous multiplexor/controller services up to 32 lines per controller. Each line on the mux is connected in star fashion and has a two byte buffer associated with each line.

3. THE OPERATING SYSTEM

The Tandem/16 operating system (T/TOS) provides the capability for multiple programs to run concurrently in the same processor module.

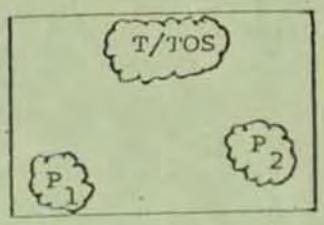


Figure F

T/TOS ensures that the programs do not interfere with each other. In fact, *A PROGRAM NEED NOT BE AWARE OF THE OTHER RUNNING PROGRAMS - AT ALL.*

The T/TOS message handler provides inter-processor communication via the interprocessor buses:

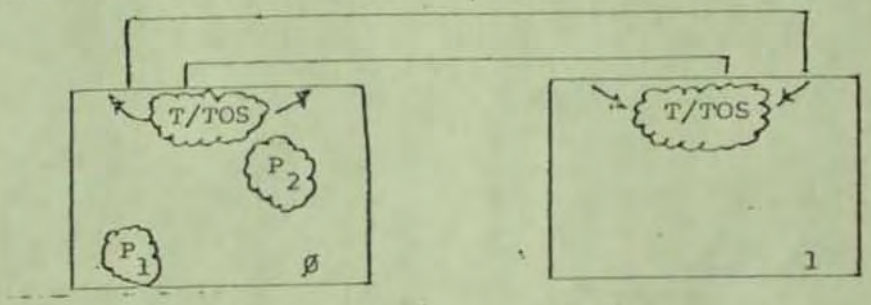


Figure G

T/TOS makes use of both buses to guarantee that a message will get to the desired processor module.

The effective result is *ONE SYSTEM* consisting of two or more separate (and redundant) hardware modules

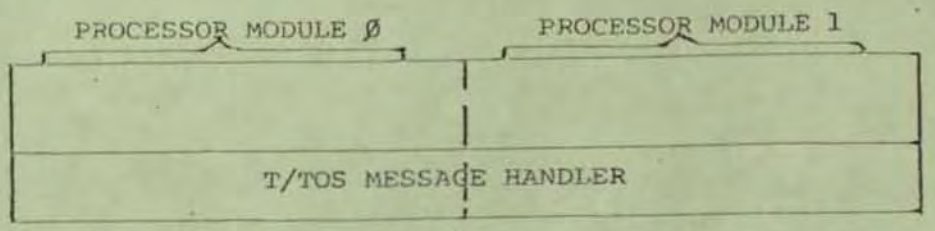


Figure H

The file system (part of T/TOS) is used by programs to communicate with each other and with I/O devices.

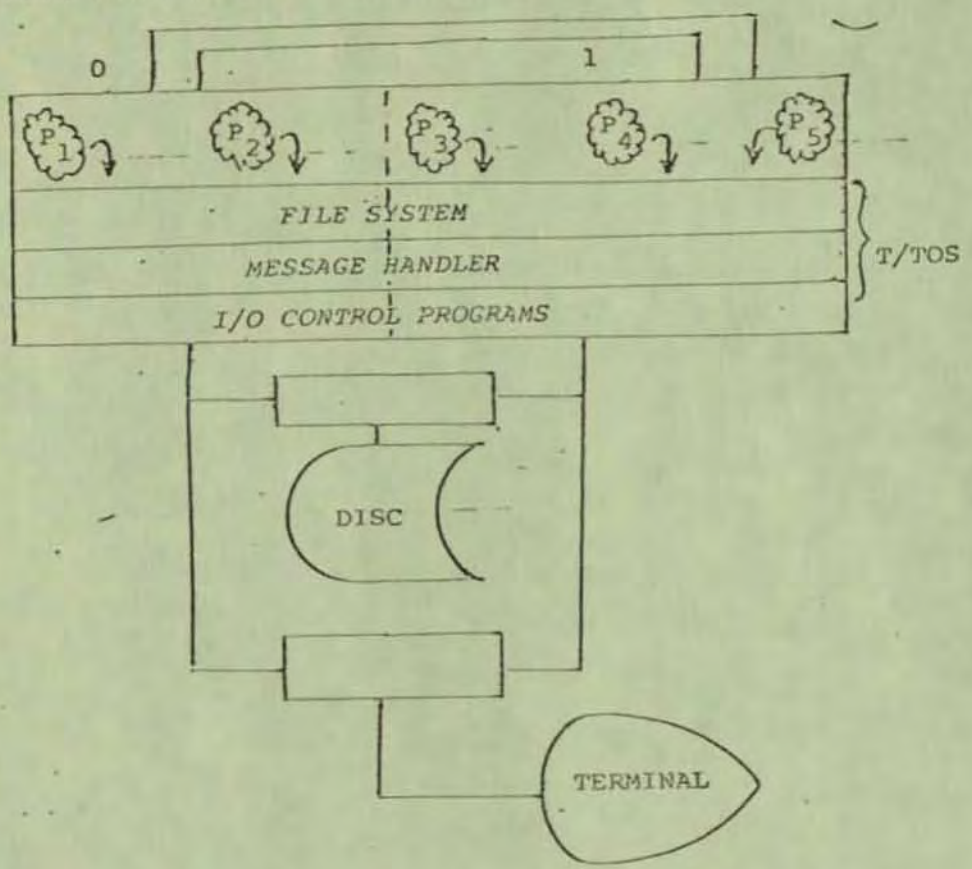


Figure I

The file system provides a single interface between a program and the outside world. Other programs and all I/O devices are accessed in a single, uniform manner. The physical locating of I/O devices and the processor module where a program is executing is transparent to application programs. Processor to processor communications are completely transparent to application programs. For program P_1 (Figure I) to send a message to Program P_3 program P_1 issues a write to file P_3. The file system determines the location of P_3 and issues the message to the T/TOS message system. P_1 is returned the status of the I/O.

As described previously, the use of dual port controllers guarantees a communication path with each I/O device even if a failure occurs:

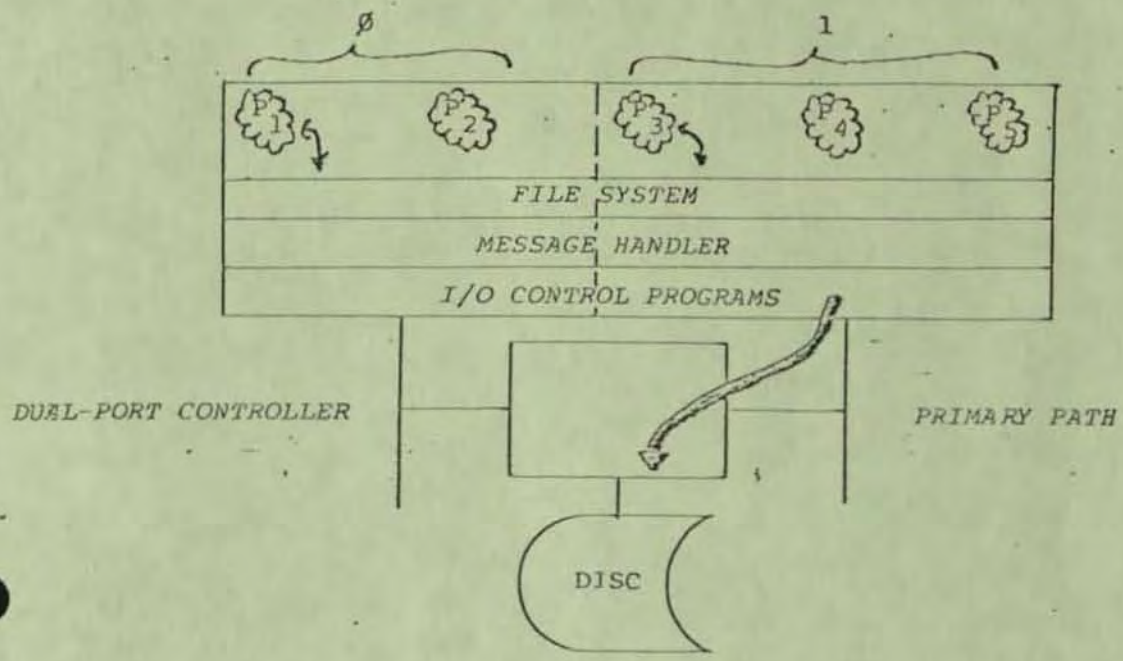


Figure J

Each device has a primary path over which communication normally occurs (Figure J).

If a failue occurs in the primary path (Figure K) the *FILE SYSTEM AUTOMATICALLY* re-routes communication to the effected I/O device via the alternate path.

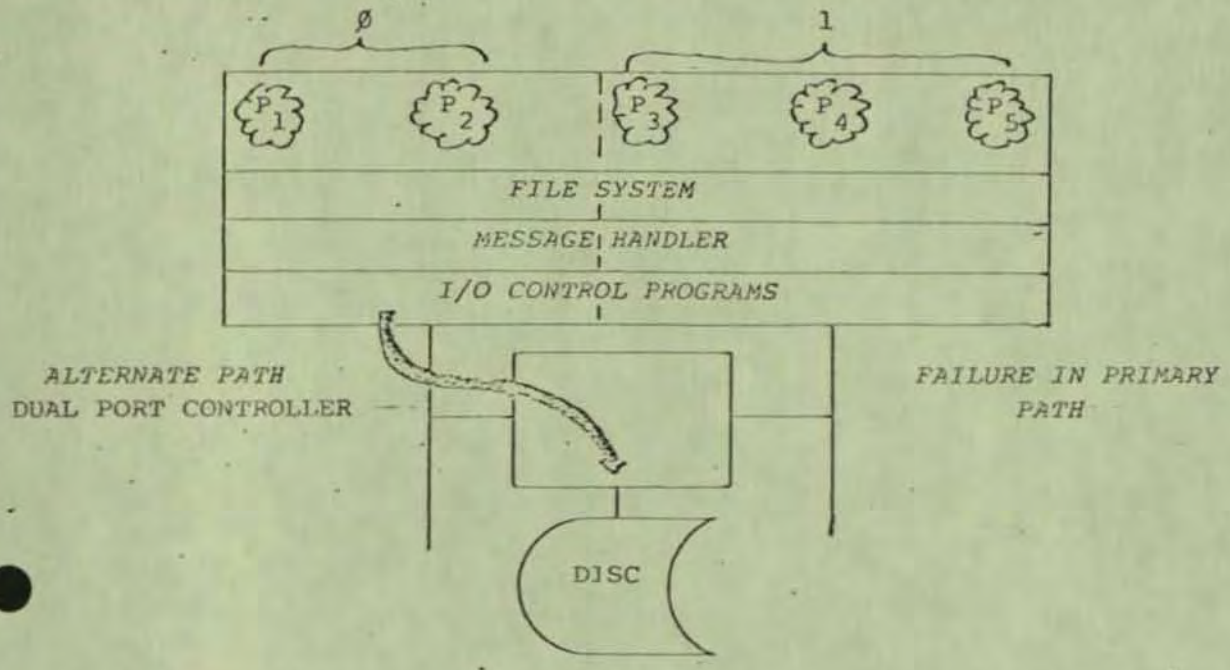


Figure K

All communication to the device occurs via the alternate path until the primary path is restored. The restoration process is performed while the system is online. Once the faulted hardware is replaced then re-integration of the new component is accomplished via one(1) command at the operator console.

The effective result is:

PROGRAMS RUNNING IN INDEPENDENT AND REDUNDANT HARDWARE MODULES THAT COMMUNICATE WITH EACH OTHER AND WITH ANY I/O DEVICE. The hardware provides two paths between processor modules and to I/O devices. The operating system guaranteees that if a single path is available, communication will occur.

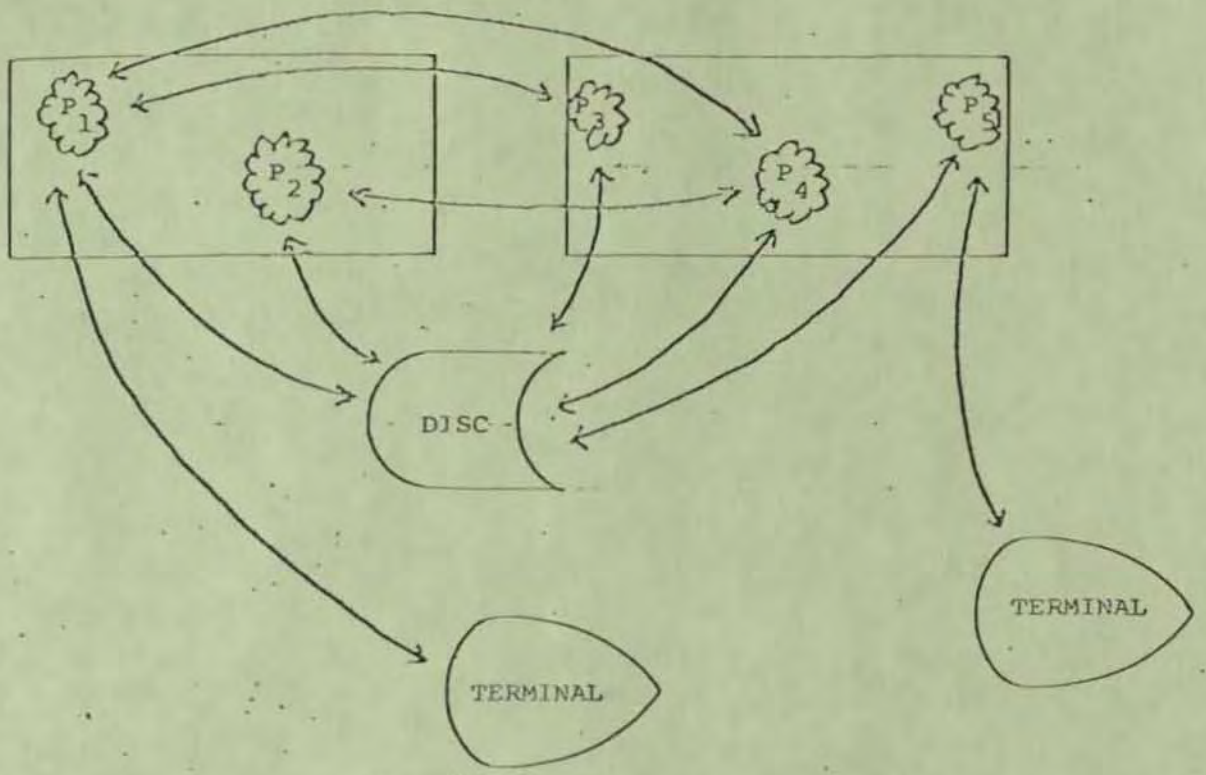


Figure L

The applications have only one common interface - the T/Tos file system. All the application needs to do is specify a file number and the operating system does the rest (takes care of identifying the communication path or paths and initiates the I/O).

The operating system provides another service:

While application programs are executing, T/Tos in each processor module periodically sends its status to all other processor modules in the system. This can be refered to as *"BROADCASTING"*.

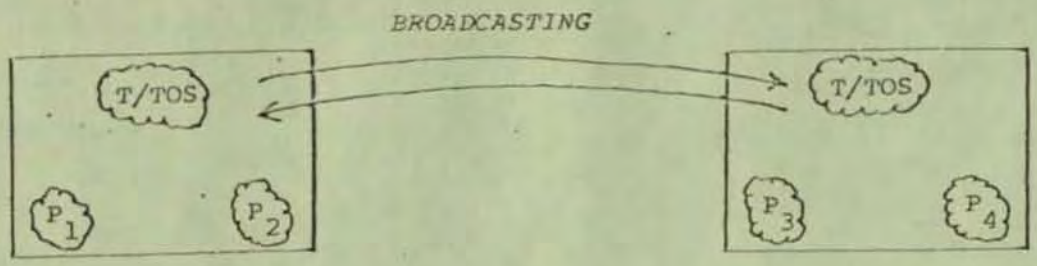


Figure II-M

If T/TOS in a processor module fails to broadcast its status message then T/TOS in the remaining module(s) (after a series of low level status checking) will declare the non broadcasting module inoperable. T/TOS in the remaining module(s) sends a *"CPU DOWN"* message to interrested programs in its processor module. The message is sent to the program through the file system (Figure N).

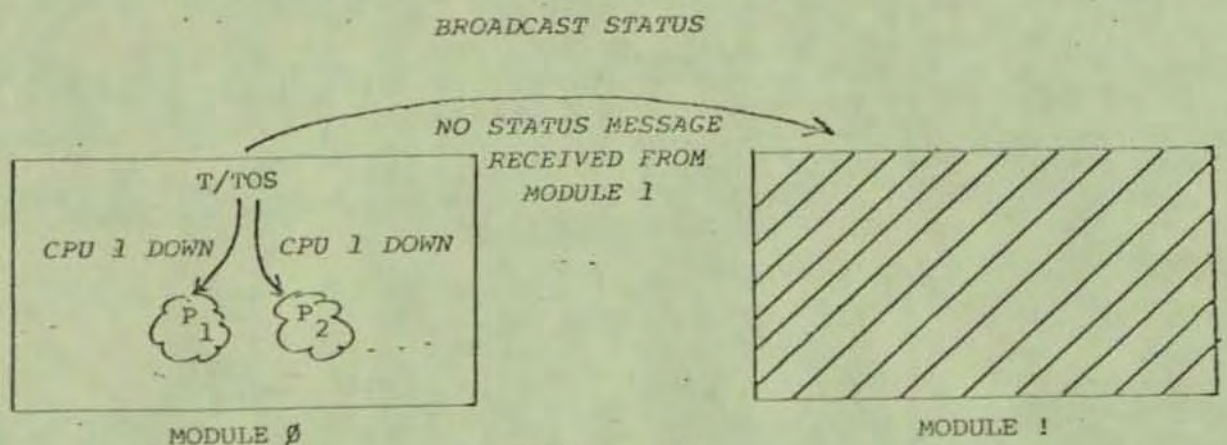


Figure N

This means that a program running in one processor module can find out if another processor module fails (and, therefore, knows of any *'STOPPED'* programs in the failed processor module). Additionally a *"CPU DOWN"* message is issued to the operator console(s).

4. NONSTOP APPLICATIONS

A nonstop application consists of a primary program running in one processor module (called "A") and a backup program running in another module (called "A'") as in Figure 0.

The primary program, while operable, performs all of the application's work. At critical points in the application (such as prior to altering a disc file), the primary program sends a message containing *"CHECKPOINTING"* information to the backup program. This is accomplished with one statement of code - - *CHECKPOINT*.

While the primary program is operable, the responsibility of the backup program is to accept the checkpointing messages and be ready to take over the application if the primary program becomes inoperable.

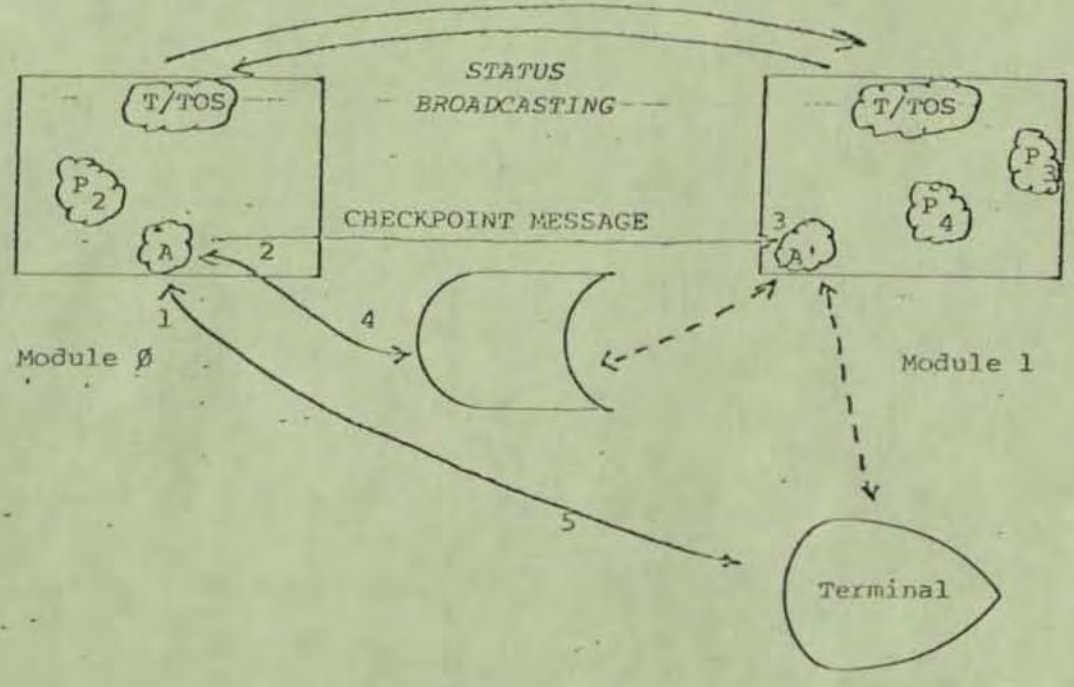


Figure 0

The Program: A

1. READ (A record from the terminal)
2. READ (A record from the disc)
3. WRITE (The old disc record from the disc)
   CHECK POINT *
4. WRITE (The updated record to disc)
5. WRITE (The result on the terminal)

* A TANDEM FAILSAFE UTILITY

The Program: A'

1. READ (The checkpoint message from A)
2. HAS SYSTEM ANNOUNCED PROCESSOR FAILURE?
3. IF ANSWER TO STEP 2 IS YES THEN TAKEOVER OTHERWISE GO TO STEP 1.

If processor '0' should fail:

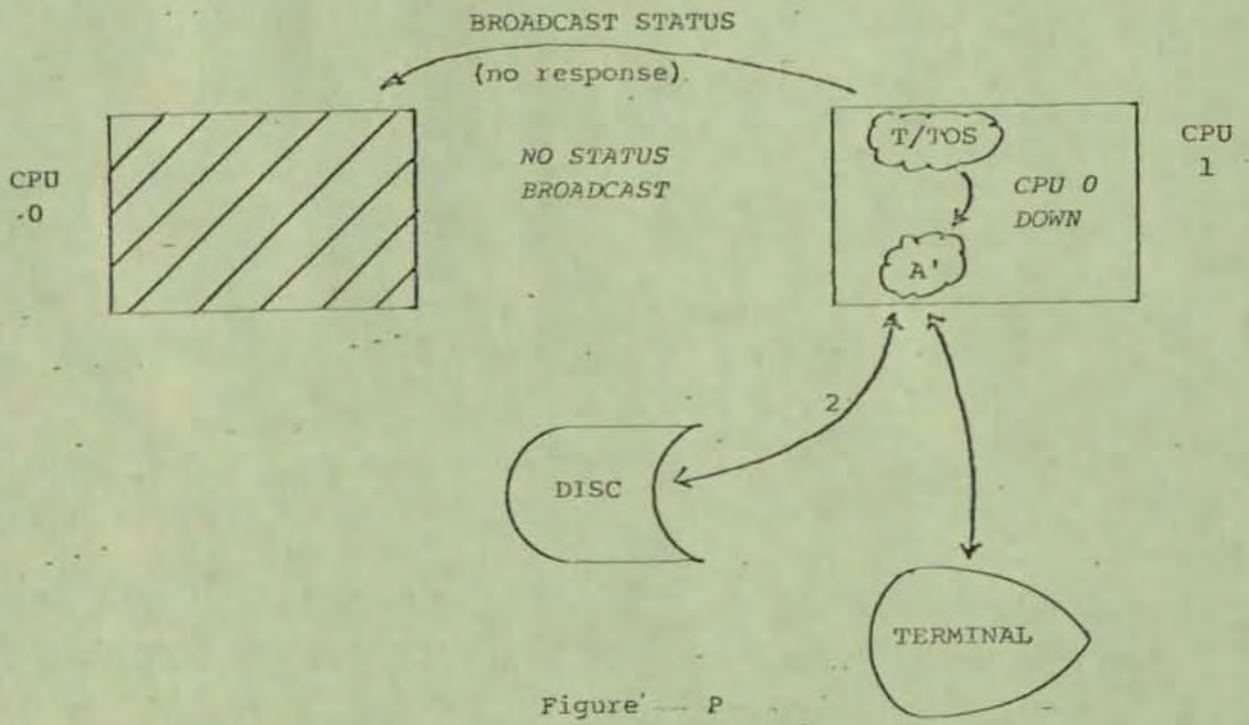


Figure P

Program A' Action

1. READ (The *CPU 0 DOWN* Message)

2. WRITE (To the disc using the last checkpoint message to restore the record)

Then continuing with the same program as "A"

READ (A record from the terminal)

READ (A record from the disc)

.

.

.

.

Except that there is no backup for A' at this time, so no checkpoint message is sent.

When CPU 0 is repaired, A *"CPU 0 UP"* message is sent by T/TOS to A'. A' then starts A running in CPU 0. (And becomes the backup)

In addition to providing the failsafe capabilities previously mentioned Tandem also provides *FAILSAFE UTILITIES*. These utilities provide considerable ease in developing failsafe applications. As an example (refer to sample program) the only differences between a *NON* - failsafe and a failsafe program are in two areas. The first would be to add *"CALL START"* as the first executable statement in the program. The second step would be to add the statement *"CALL CHECKPOINT"* at various points in the program where data is considered to be sensitive to a failure.

The *"CALL START"* invokes a utility that will create a backup program in an alternate processor. This will guarantee that if a failure should occur with the primary program or processor then the backup will automatically take over processing.

The *"CALL CHECKPOINT"* updates the status of the backup program. All variable data and program pointers are sent to the backup processor and program each time a *"CALL CHECKPOINT"* is invoked. In the event of a failure to the primary the backup will resume from the point of the last *"CALL CHECKPOINT"* that the primary issued.

SAMPLE PROGRAM

NON-FAILSAFE PROGRAM

```
        DATA DECLARATIONS


INPUT:

        READ        (RECORD FROM TERMINAL)
        READ        (RECORD FROM DISC)
        WRITE       (UPDATE RECORD ON DISC)
        WRITE       (RECORD TO TERMINAL)
        GO TO INPUT     (READ ANOTHER RECORD FROM TERMINAL)
```

FAILSAFE PROGRAM

DATA DECLARATIONS AS ABOVE

CALL START (TANDEM FAILSAFE UTILITY)

INPUT:

READ	*(RECORD FROM TERMINAL)*
READ	*(RECORD FROM DISC)*
CALL CHECKPOINT	*(TANDEM FAILSAFE UTILITY)*
WRITE	*(RECORD TO DISC - UPDATE)*
WRITE	*(RECORD TO TERMINAL)*
GO TO INPUT	*(READ ANOTHER RECORD FROM TERMINAL)*

If a failure should occur previous to the *"CALL CHECKPOINT"* then executions in the backup will occur from the first *"READ"* to the terminal. If a failure should occur after the *"CALL CHECKPOINT"* then execution in the backup will occur with the *"WRITE"* of the update record to the disc.

"CALL CHECKPOINT" can be inserted anywhere in an application program. This gives the programmer all the flexibility required to design applications with the desired level of FAILSAFE.

342

TANDEM

END USER PRICE LIST

NonStop™ SYSTEMS AND SYSTEM MODULES

TANDEM COMPUTERS, INC.

Tandem Computers Inc. was organized in 1974 to create a high volume transaction-oriented data processing system substantially more reliable than any previous system. By creating a new computer architecture and software system totally oriented toward fail safe operation, Tandem's architects, engineers and programmers achieved the "NonStop" computer.

"NonStop" means that processing operations continue even though a module fails anywhere within the system. The NonStop computer can be repaired without halting processing operations . . . any module can be removed and replaced without system interruption. No module failure can contaminate programs or data in other modules. In addition, the NonStop system can be expanded in-place without system interruption or modification to hardware or software.

For such applications as electronic funds transfer, point-of-sale terminal support and data communications systems, Tandem's NonStop computer combines fail safe operation with speed, high computing power, and greatly simplified programming. The simplest Tandem computer can be easily expanded to accommodate highly complex data networks with a level of reliability never before approached by any commercial computing system.

TANDEM 16 NonStop SYSTEM END USER PRICE LIST

This document provides pricing information on both TANDEM 16 NonStop SYSTEMS and on the individual Tandem 16 System Modules. The Tandem 16 user may elect to purchase one of Tandem's standard systems, which he can order by Product Identification Number, and may augment this system by selecting additional System Modules from the Price List.

Or he may elect to configure his system in total from the System Module Price List. In general, if a standard system can be selected it will result in a lower cost to the buyer.

All items, both Systems and System Modules, should be ordered using the Product Identification Number shown. Information is provided for each item as described below.

PRODUCT IDENTIFICATION NUMBER

This number identifies the System or the System Module. All products used in association with the Tandem 16 Processor are identified using T16/ as the initial characters in the identification number.

Standard Systems are identified by a 3-digit number following T16/. Each number is significant in describing the system.

Individual System Modules are identified by a four digit number. The first two digits are always meaningful in describing the module. In some cases, the third and fourth digit are also significant, but usually they will be sequential numbers assigned to products as introduced.

A key to the significant digits is shown on Page 2.

DESCRIPTION

Contains essential features and specifications of the package or Module. More complete information is provided in other Tandem literature.

PRICE

The purchase price of the System or Module in dollars. All prices are FOB Cupertino, California and are exclusive of Federal, State or local taxes. All prices and specifications are subject to change without notice.

PORTION DISCOUNTED

The portion of the purchase price to be used in computing the End User Volume discount.

MAINTENANCE

Monthly rates for standard one-shift maintenance service on the System or Module provides complete service, including parts and labor, as well as routine preventative maintenance for an 8-hour day, five days per week.

INSTALLATION

The one-time charge for Installation and testing for a Module being added to an existing system.

MODULE NUMBERING SYSTEM T16/ABCD

The first digit will be used to define the class of Module being described.

A	MODULE CLASS
0	Specials
1	Processors
2	Memory
3	Peripheral Controllers
4	Random Access Devices
5	Other Peripherals
6	Communications Controllers, Terminals, and Modems
7	Packaging
8	Miscellaneous
9	Software

The second digit will be used to further define the type of module within the class.

A = 1 = PROCESSOR

A	B	
1	1	T16 Processor – Core (17 bit)
	2	Unused
	3	Unused
	4	T16 Processor – Semi (22 bit)
	5-9	Unused – Would be assigned as new memories are made available.

A = 2 = MEMORY

A	B	
2	0	ROM
	1	Core (17 bit)
	2	Unused
	3	Unused
	4	Semi (22 bit)
	5-9	Unused

A = 3 = PERIPHERAL CONTROLLERS

A	B	
3	0	Unused
	1	Disk Controllers
	2	Tape Controllers
	3	Printer/Card Reader Controllers
	4	Universal Controller
	5-9	Unused

A = 4 = RANDOM ACCESS DEVICES

A	B	
4	0	Unused
	1	Moving Head Disks
	2	Fixed Head Disks
	3-9	Unused

A=5= OTHER PERIPHERALS

A	B	
5	0	Unused
	1	NRZI Tape Drives
	2	PE Tape Drives
	3	Card Readers
	4	Card Punches
	5	Line Printers
	6	Paper Tape
	7–9	Unused

A=6= COMMUNICATIONS CONTROLLERS, TERMINALS AND MODEMS

A	B	
6	0	Unused
	1	SDLC Controllers
	2	Synchronous Controllers
	3	Asynchronous Controllers
	4	Glass TTY Terminals
	5	Page Mode Terminals
	6	Modems
	7	Hard Copy Terminals
	8–9	Unused

A=7= PACKAGING

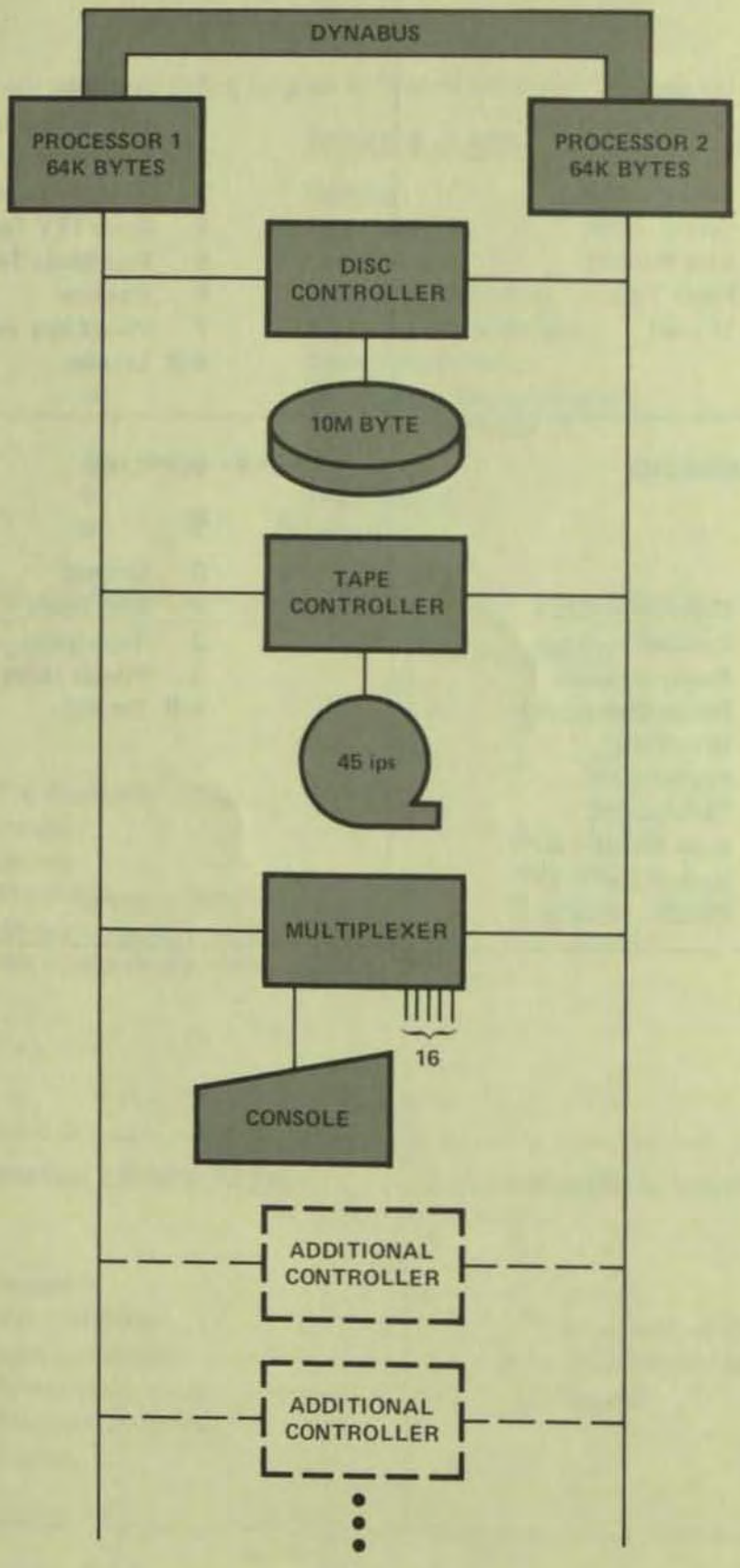
A	B	
7		
	1	Cabinets – C.P.
	2	Cabinets – Other
	3	Power Supplies
	4	Power Distribution (Bus Bars)
	5	Patch Panels
	6	Terminators
	7	Back Planes – CPU
	8	Back Planes – I/O
	9	Panels – Display

A=8= SUPPLIES

A	B	
8	0	Unused
	1	Disk Items
	2	Tape Items
	3	Printer Items
	4–9	Unused

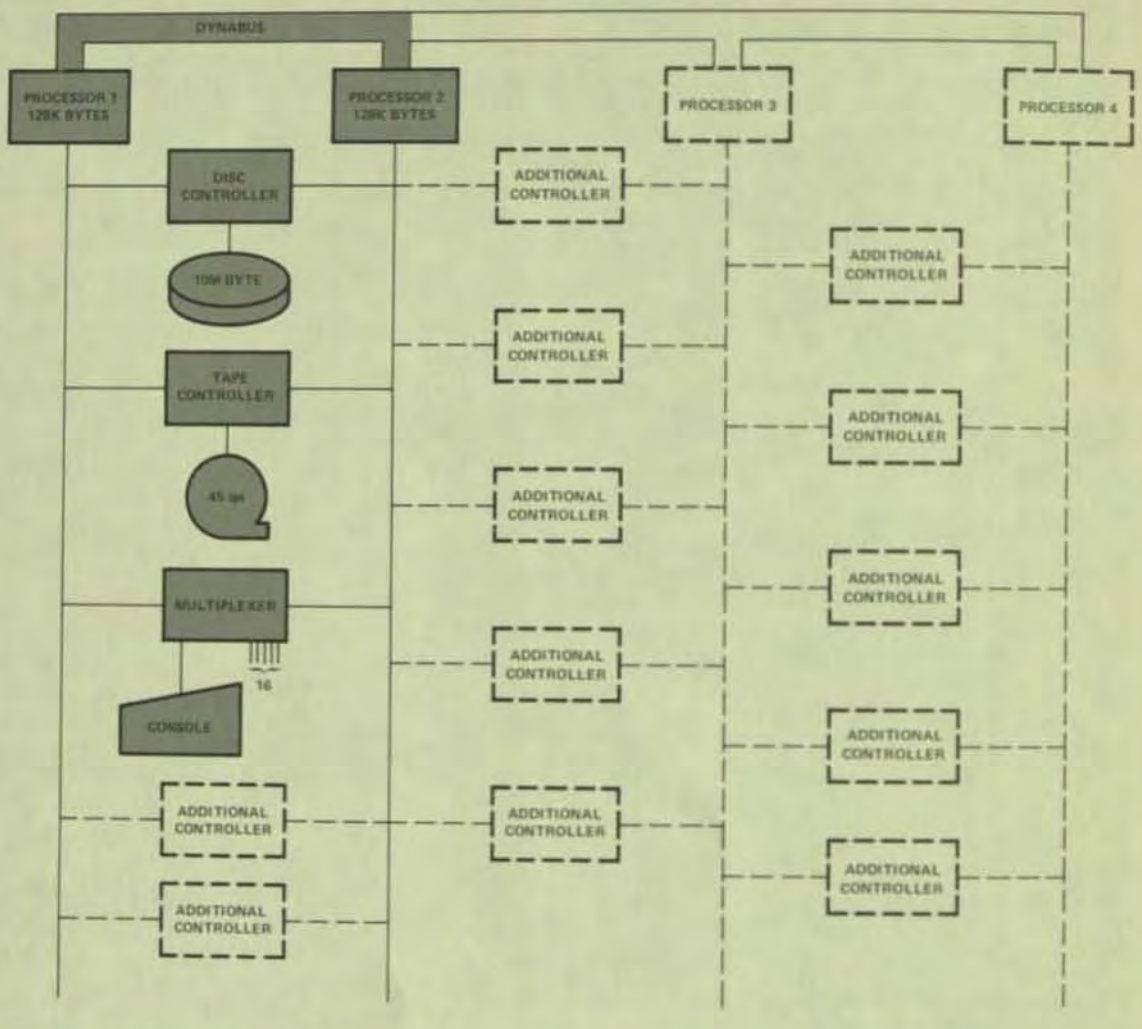
PACKAGED SYSTEMS

SYSTEM T16/212



PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/212	CONSISTING OF: 1. TWO PROCESSOR SYSTEM; each processor includes power-fail/auto-restart, memory mapping, bootstrap loader, interval timer, DMA and 64K bytes of high speed (800 Nsec) core memory with parity. Each processor may be expanded to 128K bytes of core memory with parity. Dual DYNABUS(TM) redundant inter-processor link (20M bytes/sec). 2 block multiplexed I/O channels (2.5M bytes/sec each). 17 port multiplexer, with dual independent I/O channel connections, for hardwired or modem connected asynchronous terminals (50-19.2K baud per port). Disc controller, with dual independent I/O channel connections, which can control up to four (4) star connected disc drives. Magnetic Tape Controller, with dual independent I/O channel connections, which can control up to two (2) magnetic tape drives. Four (4) unassigned I/O slots for system expansion. System Cabinet. 2. CABINET CONTAINING 45 ips, 800 BPI NRZI MAGNETIC TAPE. 3. 10M BYTE TOP LOADING PEDESTAL DISC UNIT COMPRISING ONE (1) 5M BYTE FIXED PLATTER AND ONE (1) 5M BYTE REMOVABLE PLATTER. 4. CONSOLE, 30 CPS HARD-COPY 132 COLUMN CAPABLE OF PRINTING 6-PLY REPORTS.	$65,300	$50,100	–	$458.
T16/242	IDENTICAL WITH T16/212 SYSTEM WITH THE FOLLOWING EXCEPTIONS: Use of 22 bit (500 Nsec) semiconductor memory, using error detection/correction, instead of core memory. Uninterruptible power supplies are provided for the semiconductor memories. I/O channel speed increased to 4M bytes/sec, per channel.	$67,300	$52,100	–	$628.

SYSTEM
T16/214



PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/214	CONSISTING OF: 1. TWO PROCESSORS; each processor includes power-fail/auto-restart, memory mapping bootstrap loader, interval timer, DMA, and 128K bytes of high speed (800 nsec) core memory with parity. The cabinet provides capability to expand the system to up to four (4) processors each with 192K bytes of core memory with parity. Dual DYNABUS(TM) redundant interprocessor link (20M bytes/sec). Two (2) block multiplexed I/O channels (2.5M bytes/sec. each). Expandable to four (4) by the addition of further processors.	$79,500	$64,300	–	$518.

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
	17 port multiplexer, with dual independent I/O channel connections, for hardwired or modem connected asynchronous terminals (50–19.2K baud per port). Disc controller, with dual independent I/O channel connections which can control up to four (4) star connected disc drives. Magnetic Tape Controller with dual independent I/O channel connections which can control up to two (2) star connected magnetic tape drives. Twelve (12) unassigned I/O slots for system expansion. System Cabinet. 2. CABINET CONTAINING 45 ips, 800 BPI NRZI MAGNETIC TAPE. 3. 10M BYTE TOP LOADING PEDESTAL DISC UNIT COMPRISING ONE (1) 5M BYTE FIXED PLATTER AND ONE (1) 5M BYTE REMOVABLE PLATTER. 4. CONSOLE, 30 CPS HARD-COPY 132 COLUMN, CAPABLE OF PRINTING 6-PLY REPORTS.				
T16/244	IDENTICAL WITH T16/214 SYSTEM WITH THE FOLLOWING EXCEPTIONS: Use of 22 bit (500 Nsec) semiconductor memory, using automatic error detection/correction, instead of core memory. Uninterruptible power supplies are provided for the semiconductor memories. I/O channel speed increased to 4M bytes/sec. per channel.	$83,500	$68,300	—	$858.

CORE MEMORY PROCESSORS

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/1102	GENERAL PURPOSE PROCESSOR CONSISTING OF: Two (2) pipelined microprocessors, one for programs and one for I/O. Complete DMA only I/O system (2.5M bytes/sec). Virtual memory control. Memory mapping and protection for up to 256K bytes of main memory. Hardware MPY/DIV. Power-fail/auto-restart. Bootstrap Loader. Interval Timer. Control Panel. Dual interprocessor message hardware. Provision for up to 32 I/O controllers. 122 instructions, including string manipulation and double word arithmetic. 64K bytes of core memory arranged as 17 bit words (one parity and 16 data bits per word). Memory cycle time of 800 Nsecs.	$18,500	$18,500	$ 60.	$106.
T16/1104	AS 1102 BUT WITH 128K BYTES OF CORE MEMORY	$26,500	$26,500	$ 65.	$136.
T16/1106	AS 1102 BUT WITH 192K BYTES OF CORE MEMORY	$34,500	$34,500	$ 70.	$166.
T16/1108	AS 1102 BUT WITH 256K BYTES OF CORE MEMORY	$42,500	$42,500	$ 75.	$196.

SEMICONDUCTOR MEMORY PROCESSORS

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/1402	GENERAL PURPOSE PROCESSOR CONSISTING OF: Two pipelined microprocessors, one for programs and one for I/O. Complete DMA only I/O system (4.0M bytes/sec). Virtual Memory control. Memory mapping and protection for up to 512K bytes of main memory. Hardware MPY/DIV. Power-fail/auto-restart. Bootstrap Loader. Interval Timer. Control Panel. Dual interprocessor message hardware. Provision for up to 32 I/O controllers. 122 instructions, including string manipulation and double word arithmetic. 64K bytes of semiconductor memory arranged as 22 bit words (16 data bits and 6 error detection/correction bits) *all* single bit errors are corrected and *all* double bit errors are detected. Memory cycle time of 500 nsecs. Uninterruptible power supply for semiconductor memory.	$19,500	$19,500	$ 80.	$191.
T16/1403	AS 1402 BUT WITH 96K BYTES OF SEMICONDUCTOR MEMORY.	$26,000	$26,000	$ 90.	$227.
T16/1404	AS 1402 BUT WITH 128K BYTES OF SEMICONDUCTOR MEMORY.	$28,500	$28,500	$100.	$306.
T16/1406	AS 1402 BUT WITH 192K BYTES OF SEMICONDUCTOR MEMORY.	$37,500	$37,500	$140.	$421.
T16/1408	AS 1402 BUT WITH 256K BYTES OF SEMICONDUCTOR MEMORY.	$46,500	$46,500	$160.	$536.
T16/1410	AS 1402 BUT WITH 320K BYTES OF SEMICONDUCTOR MEMORY.	$55,500	$55,500	$180.	$651.
T16/1412	AS 1402 BUT WITH 384K BYTES OF SEMICONDUCTOR MEMORY.	$64,500	$64,500	$200.	$766.

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/1414	AS 1402 BUT WITH 448K BYTES OF SEMICONDUCTOR MEMORY.	$73,500	$73,500	$220.	$881.
T16/1416	AS 1402 BUT WITH 512K BYTES OF SEMICONDUCTOR MEMORY.	$82,500	$82,500	$240.	$996.

MEMORY MODULES

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/2102	Core Memory module, consists of a 64K byte memory plane with a read access time of 500 ns and a cycle time of 800 ns. The module is arranged as 32K words of 17 bits (16 data bits one parity bit) up to 4 of the modules may be controlled by a single processor.	$ 8,000	$ 8,000	$ 60.	$ 30.
T16/2401	Semiconductor memory module, consists of 32K byte memory module with a cycle time of 500 nsecs. The module is arranged as 16K words of 22 bits, 16 data bits and 6 error detection/correction bits. The module enables detection and correction of all single bit errors and detection of *all* double bit errors. Up to 8 of these modules may be controlled by a single processor.	$ 6,500	$ 6,500	$ 60.	$ 36.
T16/2402	Semiconductor memory module, consists of 64K byte memory module with a cycle time of 500 nsecs. The module is arranged as 32K words of 22 bits, 16 data bits and 6 error detection/correction bits. The module enables detection and correction of all single bit errors and detection of *all* double bit errors. Up to 8 of these modules may be controlled by a single processor.	$ 9,000	$ 9,000	$ 60.	$115.

TERMINAL SUBSYSTEMS

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/6301	ASYNCHRONOUS CONTROLLER, incorporates two (2) independent I/O channel connections. Each is capable of providing a complete path between processor and terminals. The controller may be powered from either of the processors to which it is connected; in the event of a power failure, the controller will automatically draw its power from the second processor. This controller will control two (2) terminal lines, each of which may be either hardwired or modem connected. The speed of each line is programmable between 50 and 19200 baud. This controller may be augmented by up to two (2) 6302's.	$ 1,700	$ 1,400	$100.	$ 15.
T16/6302	ASYNCHRONOUS EXTENSION BOARD Provides control for up to 15 asynchronous lines. The speed of each line is programmable between 50 and 19200 baud. Each line may be hardwired or modem connected. This controller requires a T16/6301 as a prerequisite.	$ 3,100	$ 3,100	$ 60.	$ 18.
T16/6701	HARD COPY TERMINAL, 30 cps 132 column printer. 20 MA current loop interface compatible with 6301/6302. Includes 25' device cable.	$ 2,500	–	$120.	$ 31.
T16/6702	As 6701 except that RS232C interface is used.	$ 2,600	–	$120.	$ 31.
T16/6401	CRT TERMINAL, 24 lines 80 characters per line. Speed switch selectable 110 – 19200 baud. Compatible with 6301/6302 or may be used via modems. Includes 25' device cable.	$ 1,500	–	$120.	$ 15
T16/6511	CRT TERMINAL, may be used either in character or page mode. Includes local editing and function keys. Data formats include protected, unprotected, full bright, half bright, full bright reverse, half bright reverse, blinking, etc. Speed 110-9600 baud. May be used hardwired or via modems. Includes 25' device cable.	$ 2,400	–	$120.	$ 30.
T16/6552	As 6501, but using polling protocol for multi-drop use.	$ 2,700	–	$120.	$ 33.

DISC SUBSYSTEMS

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/3101	DISC CONTROLLER, incorporates two (2) independent I/O channel connections. Each is capable of providing a complete path between processor and disc. The controller may be powered from either of the processors to which it is connected. In the event of a power failure, the controller will automatically draw its power from the second processor. The controller can control up to four (4) disc drives, each has a separate connection to the controller (i.e., not daisy chained).	$ 4,400	$ 4,400	$ 60.	$ 21.
T16/4101	MOVING HEAD DISC, 10M BYTE Consists of a pedestal mounted disc comprising one (1) fixed platter and one (1) removable top loading platter. Seek Time = 30 msecs average Latency = 12.5 msecs average Includes 25' device cable.	$ 8,000	$ 4,000	$160.	$ 63.
T16/3102	DISC CONTROLLER, (used for discs whose capacity exceeds 10M byte). Incorporates two (2) independent I/O channel connections. Each is capable of providing a complete path between processor and disc. The controller may be powered from either of the processors to which it is connected, in the event of a power failure, the controller will automatically draw its power from the second processor. The controller can control up to four (4) discs, each has a separate connection to the controller (i.e., not daisy chained).	$ 8,800	$ 8,800	$120.	$ 43.
T16/4102	Moving Head Disc, 40M byte. Consists of a pedestal mounted disc using a 5 platter removable disc pack. Seek Time 30 msecs Latency (Avg.) 8.3 msecs	$12,000	$ 6,000	$160	$119.
T16/4103	Moving Head Disc, 80M bytes. Consists of a pedestal mounted disc using a 5 platter removable disc pack. Seek Time 30 msecs Latency (Avg.) 8.3 msecs	$15,000	$ 8,000	$160	$119.

MAGNETIC TAPE & PUNCHED CARD SUBSYSTEMS

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/3201	MAGNETIC TAPE CONTROLLER, incorporates two (2) independent I/O channel connections. Each is capable of providing a complete path between processor and magnetic tape drive. The controller may be powered from either of the processors to which it is connected; in the event of a power failure, the controller will automatically draw its power from the second processor. The controller can control up to two (2) magnetic tape drives, each has a separate connection to the controller (i.e., not daisy chained).	$ 3,000	$ 3,000	$ 60.	$ 20.
T16/5101	MAGNETIC TAPE DRIVE, cabinet containing 45 ips, 9-track, IBM compatible NRZI 800 BPI drive. Rewind speed 300 ips. Power-fail/auto-restart. Includes full size cabinet and 25' device cable.	$ 6,950	$ 3,000	$240.	$ 46.
T16/3303	CARD READER CONTROLLER incorporates two (2) independent I/O channel connections. Each is capable of providing a complete path between processor and card reader. The controller may be powered from either of the processors to which it is connected, in the event of a power failure, the controller will automatically draw its power from the second processor.	$ 1,800	$ 1,000	$ 60.	$ 17.
T16/3304	AS T16/3303 BUT CAN CONTROL TWO (2) CARD READERS.	$ 2,600	$ 2,000	$ 60.	$ 17.
T16/3305	CARD READER/LINE PRINTER CONTROLLER incorporates two (2) independent I/O channel connections. Each is capable of providing a complete path between processor and card reader/line printer. The controller may be powered from either of the processors to which it is connected, in the event of a power failure, the controller will automatically draw its power from the second processor. The controller can control one card reader and one line printer.	$ 2,600	$ 2,000	$ 60.	$ 17.
T16/5301	CARD READER, reads card at 600 cards per minute. Uses standard 80 column cards.	$ 4,800	$ 2,500	$140.	$ 40.

LINEPRINTER SUBSYSTEMS

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/3301	LINE PRINTER CONTROLLER, incorporates two (2) independent I/O channel connections. Each is capable of providing a complete path between processor and line printer. The controller may be powered from either of the processors to which it is connected, in the event of a power failure, the controller will automatically draw its power from the second processor.	$ 1,800	$ 1,000	$ 60.	$ 17.
T16/3302	LINE PRINTER CONTROLLER, AS T16/3301 BUT CAN CONTROL TWO (2) LINE PRINTERS.	$ 2,600	$ 2,000	$ 60.	$ 17.
T16/5501	LINE PRINTER, 120 column speed of 120-300 lpm dependent upon number of characters being printed. 64 character ASCII set. Includes 25' device cable.	$ 5,600	$ 2,600	$120.	$ 48.
T16/5502	LINE PRINTER, 132 column 300 lpm drum printer. Includes VFU and paper receptacle. 64 character ASCII set. 12 channel VFU. Includes 25' device cable.	$11,500	$ 6,000	$180.	$133.
T16/5503	LINE PRINTER, 132 column 600 lpm drum printer. Includes VFU and paper receptacle. 64 character ASCII set. 12 channel VFU. Includes 25' device cable.	$14,000	$ 6,000	$180.	$154.
T16/5504	LINE PRINTER, 132 column 900 lpm drum printer. Includes VFU and paper receptacle. 64 character ASCII set. 12 channel VFU. Includes 25' device cable.	$21,000	$ 8,000	$180.	$154.
T16/5505	LINE PRINTER, 132 column 1,500 lpm printer. Includes VFU and powered paper stacker. 64 character ASCII set. 12 channel VFU. Includes 25' device cable.	$42,000	$12,000	$200.	$165.

GENERAL CONTROLLERS

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/3401	UNIVERSAL INTERFACE, incorporates two (2) independent I/O channel connections. Each is capable of providing a complete path between processor and I/O devices. The controller may be powered from either of the processors to which it is connected; in the event of a power failure, the controller will automatically draw its power from the second processor. This controller will control either a line printer, card reader, or any other device having a 16 line parallel interface. The line drivers and receivers are TTL level.	$ 1,800	$ 1,000	$ 60.	$ 17.
T16/3402	As 3401, but using differential line drivers/receivers.	$ 1,800	$ 1,000	$ 60.	$ 17.
T16/3403	As 3401, but has capability to control two (2) I/O devices (both using TTL level line drivers/receivers).	$ 2,600	$ 2,000	$ 60.	$ 17.
T16/3404	As 3403, but using differential line drivers/receivers.	$ 2,600	$ 2,000	$ 60.	$ 17.
T16/3405	As 3403, but using TTL level driver/receivers for one (1) I/O device and differential level for the other.	$ 2,600	$ 2,000	$ 60.	$ 17.

PACKAGING

PRODUCT IDENTIFICATION NUMBER	DESCRIPTION	LIST PRICE	DISCOUNTABLE PORTION	INSTALLATION	MONTHLY MAINTENANCE
T16/7101	SYSTEM CABINET. Contains provision for up to eight (8) processors and 32 I/O controllers. Includes DYNABUS (TM) redundant interprocessor priority resolution circuitry and data paths. Provides I/O controller power switching and processor I/O channel wiring. Up to two (2) system cabinets may be combined into each system.	$ 5,500	$ 2,500	$320.	$ 30.
T16/7301	POWER MODULE. Supplies power for I/O controllers (only required on large configurations). Includes connections to power distribution and switchover systems.	$ 1,900	$ 1,900	$320.	$ 34.

NOTES:

NOTES:

NOTES:

TANDEM COMPUTERS, INC.
20605 VALLEY GREEN DR.
CUPERTINO, CALIFORNIA 95014
(408) 255-4800

TANDEM 16
SYSTEM INTRODUCTION

TANDEM
COMPUTERS, INC.

The Tandem 16 System, described in this document, represents a major departure from existing mini/midi computer architecture. For the first time, a complete system has been designed to meet the growing demand for on-line, transaction processing systems with "fail-safe" capability. Using standard Tandem 16 hardware and software modules, the user may build a system to match necessary requirements exactly, both in throughput and in system reliability. No special or custom designed hardware or software is necessary. Equally important, the Tandem 16 can grow to meet increasing throughput demands with no change in operating system or applications software and without loss of the system during expansion.

One Tandem 16 processor module is a powerful computer. Two or more Tandem 16 processor modules connected together by Tandem's high-speed interprocessor bus structure (DYNABUS) provide an extremely powerful multiprocessor system. This capability coupled with the Tandem 16's unique *NONSTOP*™ or "fail-safe" features make it the ideal choice for systems requiring economy today plus assured power to meet tomorrow's increased demands.

The Tandem 16 Computer System fills three important and interrelated needs: it is a computer system where applications run *NONSTOP* regardless of a module failure, it provides a computer system that can support high transaction rates to large on-line data bases, and it provides a system that is easily adaptable to any application.

The basic design philosophy of the Tandem 16 Computer System is that no single module failure will stop or contaminate the system. This assurance of *NONSTOP* operation is sometimes called "fail-safe" when no loss of throughput occurs as a result of a failure or "fail-soft" when some slowdown occurs but full processing capabilities are maintained. The Tandem 16 can provide both "fail-safe" and "fail-soft' modes of operation.

- PROCESSOR MODULES

  A single Tandem 16 Computer System may contain from two to sixteen processor modules; each module contains a micro-programmed central processing unit, its own memory (up to 512k bytes), and its own micro-programmed input/output channel. Each processor module is fully capable of operating independently of all other processor modules yet can be configured to back up other processor modules.

- INTERPROCESSOR BUSES (DYNABUS)

  Each processor module is connected to all other processor modules via redundant high speed interprocessor buses. Programs running in one processor module communicate with programs running in other processor modules by means of these buses. Each interprocessor bus is fully autonomous, operating independently of (but simultaneously with) the other bus. The use of two buses assures that two paths exist between all processor modules in the system.

- INPUT/OUTPUT CHANNEL

  Input/output devices (i.e, magnetic tape units, disc drives, terminals, etc.) are interfaced to the computer system via dual-port i/o controllers. Dual-port means that each i/o controller is connected to the input/output channels of two processor modules. This provides two paths of communication to each input/output device. A dual-port controller is "owned" by (i.e., will accept commands from) only one processor module. But in case of a failure, the other processor module can take control programmatically. The dual-port controllers are designed so that the number of components common to both paths are at a minimum.

- TANDEM 16/TRANSACTION OPERATING SYSTEM (T/TOS)

  Overseeing system operation is the Tandem 16 Operating System. The operating system provides the multiprocessing (parallel processing in separate processor modules), multiprogramming (interleaved processing in one processor module), and NONSTOP capabilities of the Tandem 16 Computer System. A copy of T/TOS resides in each processor module.

The operating system automatically schedules application programs for execution according to an application-assigned priority, provides memory management functions (automatic overlaying, swapping to disc, etc.), and gives application programs the capability to start programs executing in any processor module from any processor module.

- FILE SYSTEM

As part of the operating system, the Tandem 16 File System handles all data transfers between programs and the outside world. The file system provides the capability for any program running in the system to communicate with any other program running in the system as well as any input/output device connected to the system; programmers need not be aware of the physical location of the device/program. Up to 4,094 bytes can be transferred in one file system operation.

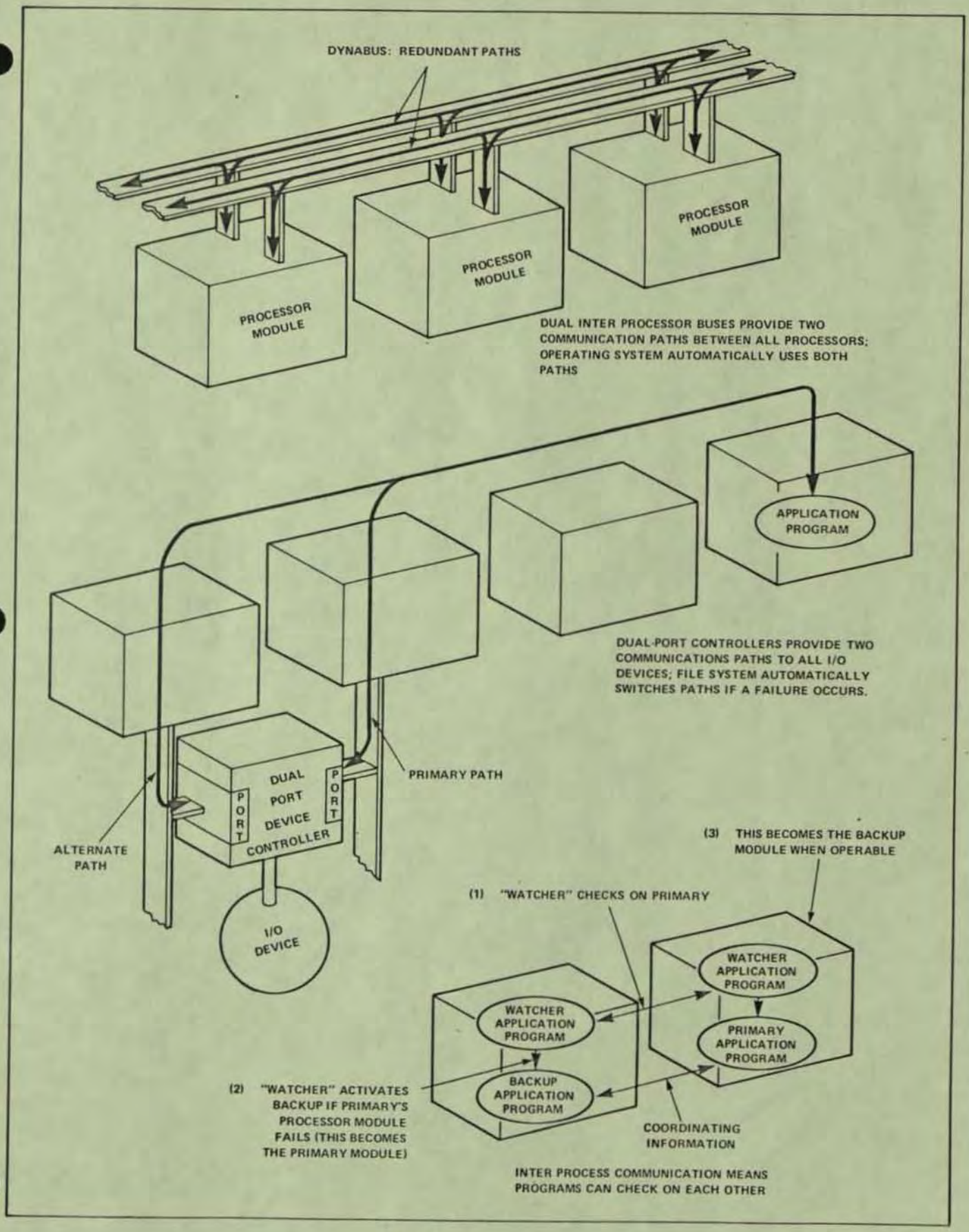
FAILURE TOLERANT

The ability of the Tandem 16 Computer System to provide an environment where applications can continue to run regardless of a module failure is due to its unique hardware and software design:

- DYNABUS connected to each processor module provide two paths between all processor modules; if a bus fails, the operating system automatically routes all interprogram communication through the other bus.

- Dual-port controllers ensure that there are two communication paths to each input/output device; if a path to an input/output device fails, the file system automatically switches control of the i/o device to the alternate processor module.

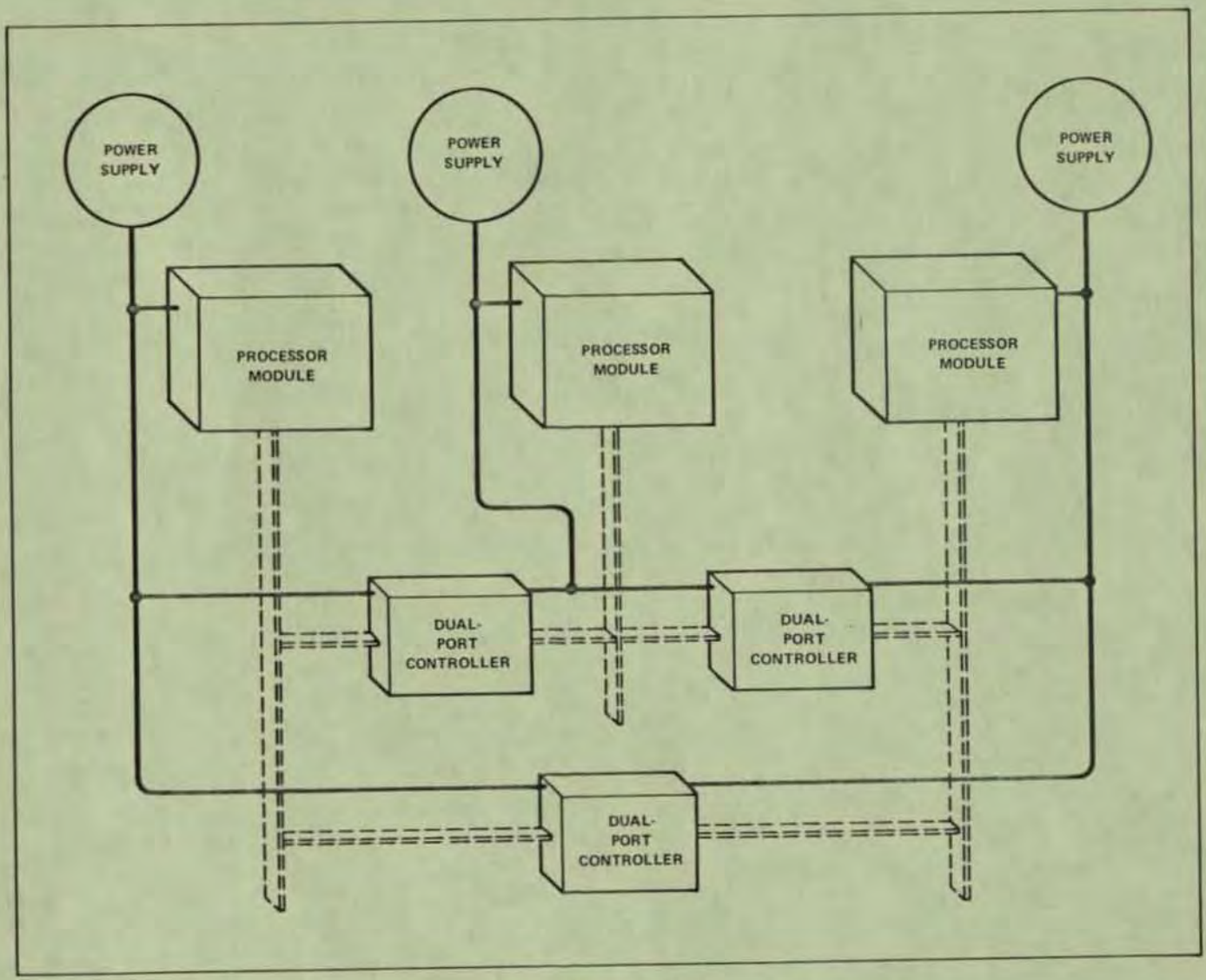
- Application programs can communicate with each other permitting a program running in one processor module to check on the progress of a program running in another processor module; if a processor module fails, its responsibilities can be programmatically switched to another module.



Failure Tolerent

- Because interprocessor communication is provided by means of the redundant interprocessor buses, no shared memory is required. This eliminates a point where a single failure could stop a system and also prevents one malfunctioning processor from contaminating any memory but its own. Likewise, the use of dual-port i/o controllers eliminates the need for bus switching devices (again, where a single failure could stop a system).

- Power is distributed in the system in such a manner that if a power supply fails, a backup module is available. The dual-port controllers receive power from two sources: the same supplies as their associated processor modules. If a supply fails, causing a processor module to become inoperative, the alternate processor module can take control.



Power Distribution

- Each processor module has the capability to save its current operating state in the event a power failure occurs and resume its operations when power is restored.

- If an uncorrectable error is found in memory, the operating system determines if the associated area is critical to system operation. If it is not, the area is flagged as bad and not used again until the memory is

repaired. (Typically, the memory would be repaired during system preventive maintenance. However, the the associated processor module could be taken off line to repair the memory, leaving the remainder of the system operable.)

- Critical portions of the operating system are main memory resident; this assures their availability in the event a disc failure occurs.

- The cooling system for the Tandem 16 is designed in such a way that if a failure occurs, ample cooling is still available. In addition, fan modules can be replaced while the system is running (without interfering with system operation).

- Any operational module in the system (e.g., processor, i/o controller, power supply, fan, etc.) can be removed from the system and replaced on-line without stopping operation of other system modules.

- Because the Tandem 16 is modular in organization, it can be configured for any degree of *NONSTOP*ability desired.

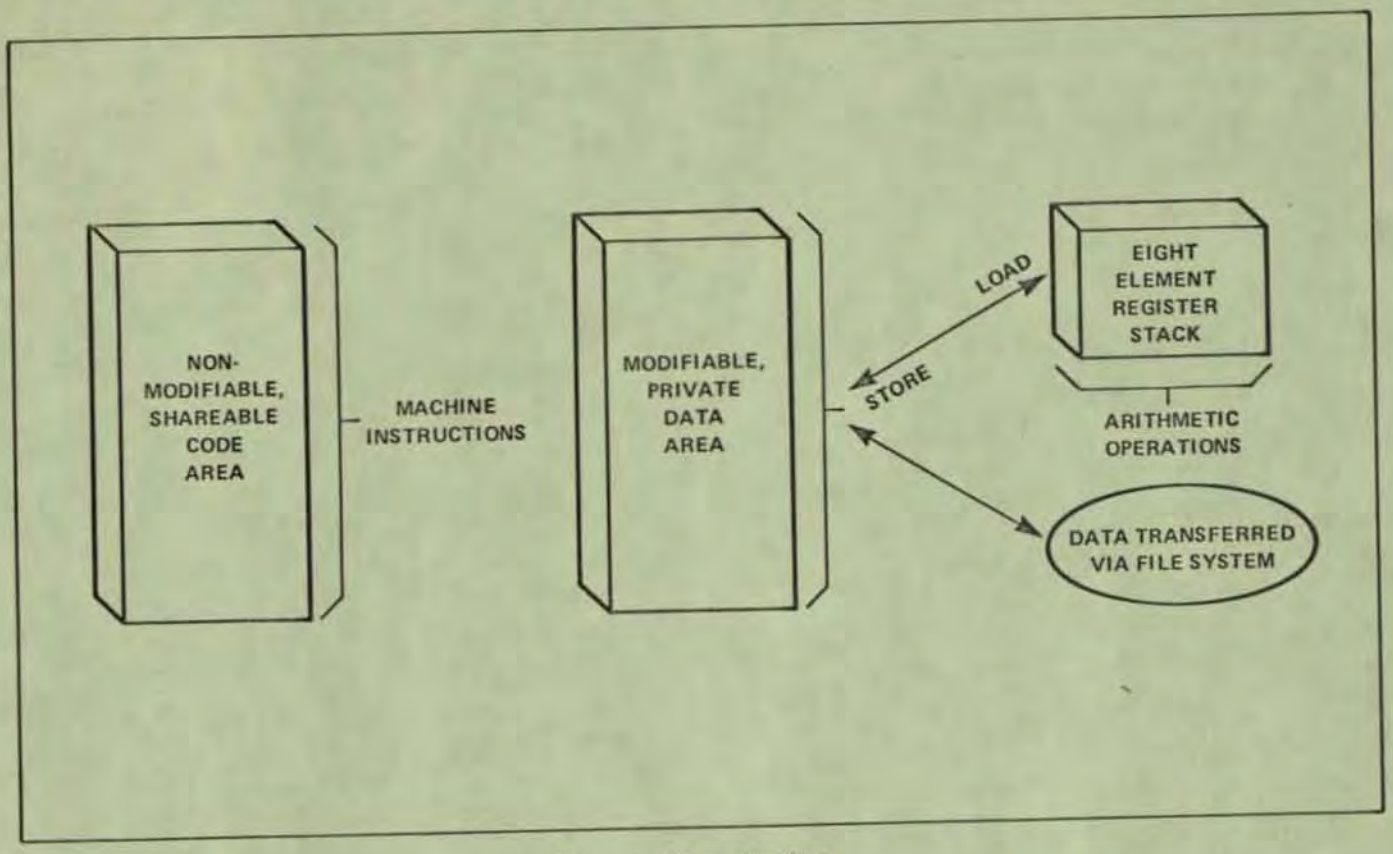
TRANSACTION ORIENTED

A number of features provide the high throughput rates attainable by the Tandem 16.

- Program Organization

  Tandem 16 programs while executing in memory are physically separated into two parts: a code part containing machine instructions and program constants and a data part containing program variables. The code part of a program is actually read-only storage (i.e., there are no machine instructions for writing into the code area).

  The fact that the code part contains pure code and cannot be modified means that it can be shared by a number of users. In fact, operating system library routines that are executed on behalf of application programs are shared by all application programs running in a given processor module (i.e., only one copy need reside in memory).



Program Organization

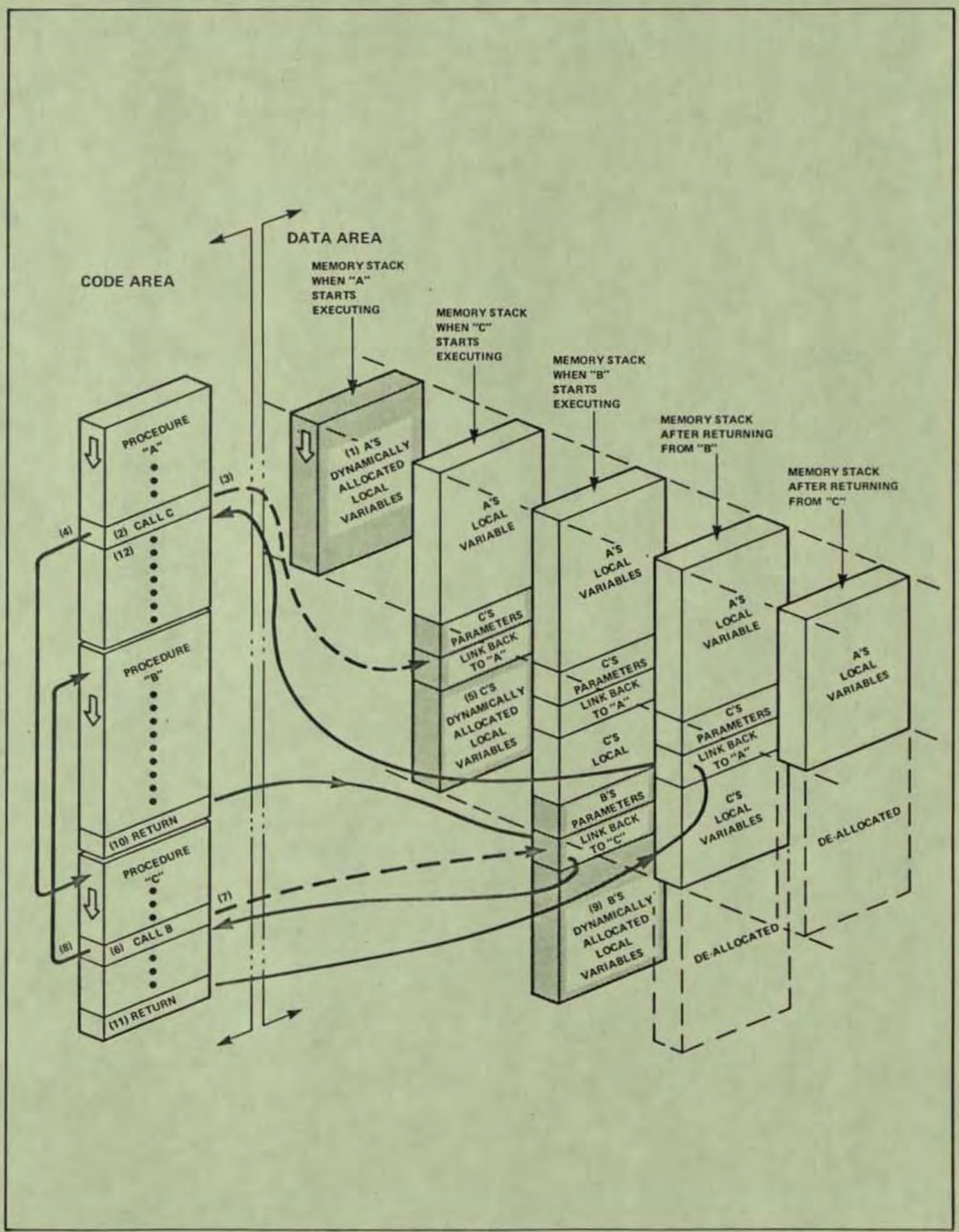
– Procedure Oriented

Programs are functionally separated into blocks of machine instructions called procedures. A procedure, like a program, has its own private data area (actually in the program's data area). The real power of procedures is that they can be called into execution from any point in a program (including other procedures and themselves); the hardware automatically saves the calling environment when a procedure starts executing and restores the calling environment when the procedure finishes. A programmer can write procedures that receive parameter information (arguments), perform computations using the parameters, then return results to the caller (the machine instructions for passing parameters and returning results are generated automatically for programmers using Tandem's Transaction Application Language – T/TAL).

Operating system and file system functions are actually invoked by calling procedures that are part of the operating system (special machine instructions exist that call operating system procedures as efficiently as an application program's own procedures).

– Memory Stack

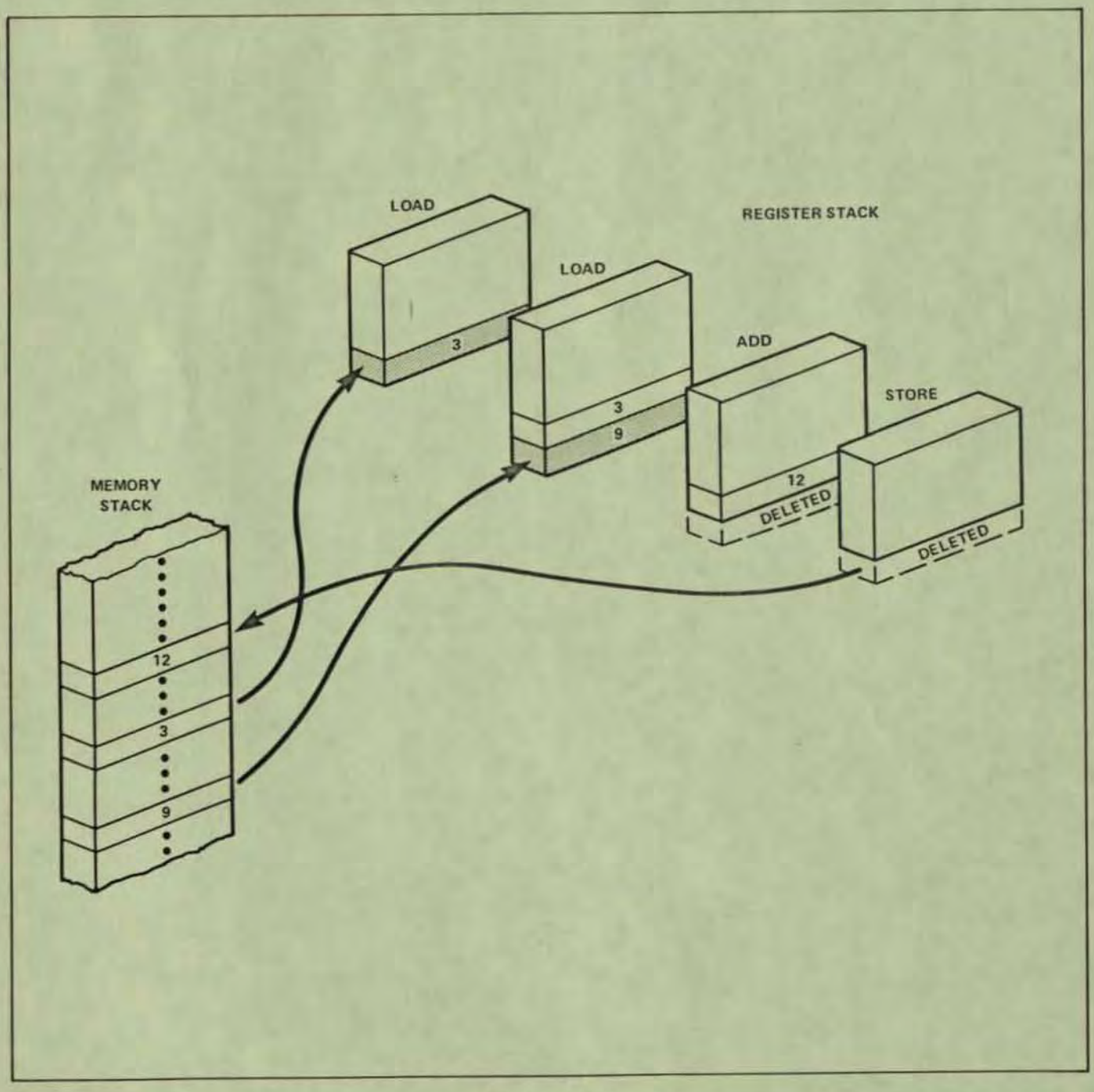
Data areas for programs are organized in main memory as stacks. A stack is a storage allocation method where the last item added is the first item removed. The CPU has registers that automatically keep track of the last area allocated in a stack. The use of the stack means that data areas for a procedure's private variables are allocated dynamically (when the procedure is called into execution), keeping the amount of memory space required by a program to a dynamic minimum. The stack also provides the mechanism for passing parameters to procedures and saving and restoring the calling environment (this applies to calling both an application's own procedures and operating system procedures).



Memory Stack

– Register Stack

Each central processor contains an eight element register stack. The register stack provides a highly efficient means of executing arithmetic operations; operands are loaded into the stack, arithmetic operations are performed, the operands are deleted, and a result is left on the stack. An add of two 16-bit numbers typically takes 500 nanoseconds; storing the result into the memory stack typically takes 900 nanoseconds. The use of the register stack is transparent to programmers using Tandem's Transaction Application Language (T/TAL). T/TAL automatically generates the machine instructions for efficiently using the register stack. T/TAL, however, does provide the capability of using the register stack explicitly.



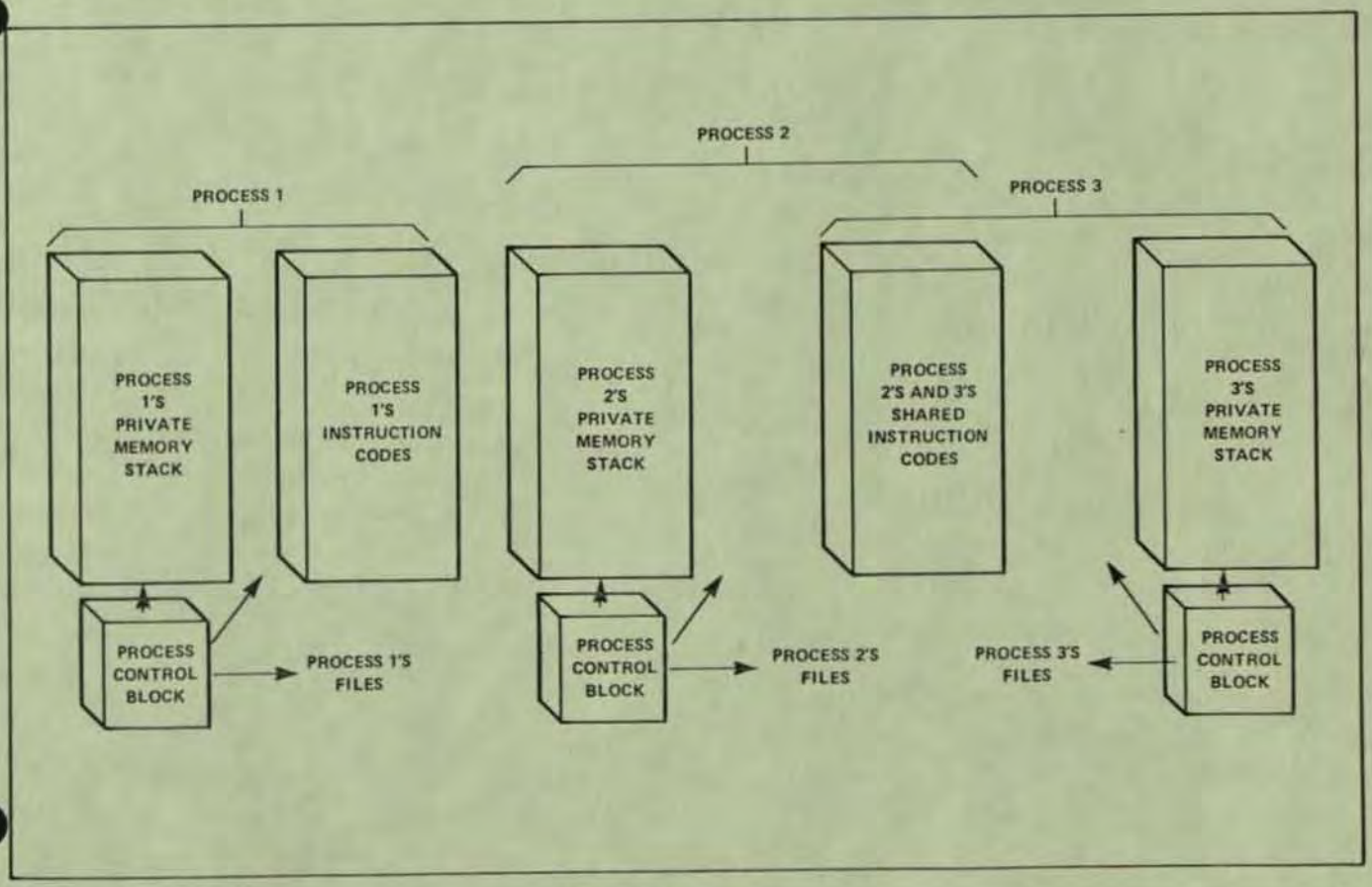
Register Stack

Process Structure

Programs (both application and system programs) are run by the Tandem 16 Operating System on the basis of processes. Two definitions: the term "program" is used to identify a static group of instruction codes and initialized data (like the output of a compiler), the term "process" indicates the dynamically changing states of an executing program. The same program can be executing concurrently a number of times; each execution is a different process.

A process consists of:

- An area in memory containing the instruction codes to be executed (this area may be shared by other processes).
- An area in memory containing a memory stack that is private to the process. (Even if other processes use the same code area, each has a private data area.)
- A process identification number (process id) assigned by the operating system when the program is first called for execution (the process id indicates the processor number where the program is executing and the number of the process in that processor).
- A process control block (PCB), identified by a process id, that is used by the operating system to control process execution. The PCB contains pointers to the process's code and data areas, retains the current state of the process in the event the process is suspended, and defines the system resources needed by the process.



Process Structure

The process structure provides the mechanism for executing programs in a multiprogramming (interleaved processing) environment. An operating system function called the "dispatcher" assigns processor time to the various processes present in the system. A process, when ready to execute, is placed in a ready list according to its priority number. When one process completes executing or is suspended (while i/o occurs), the highest priority process ready for execution is given control of the processor (a "ready" high priority process automatically causes any lower priority process to be suspended).

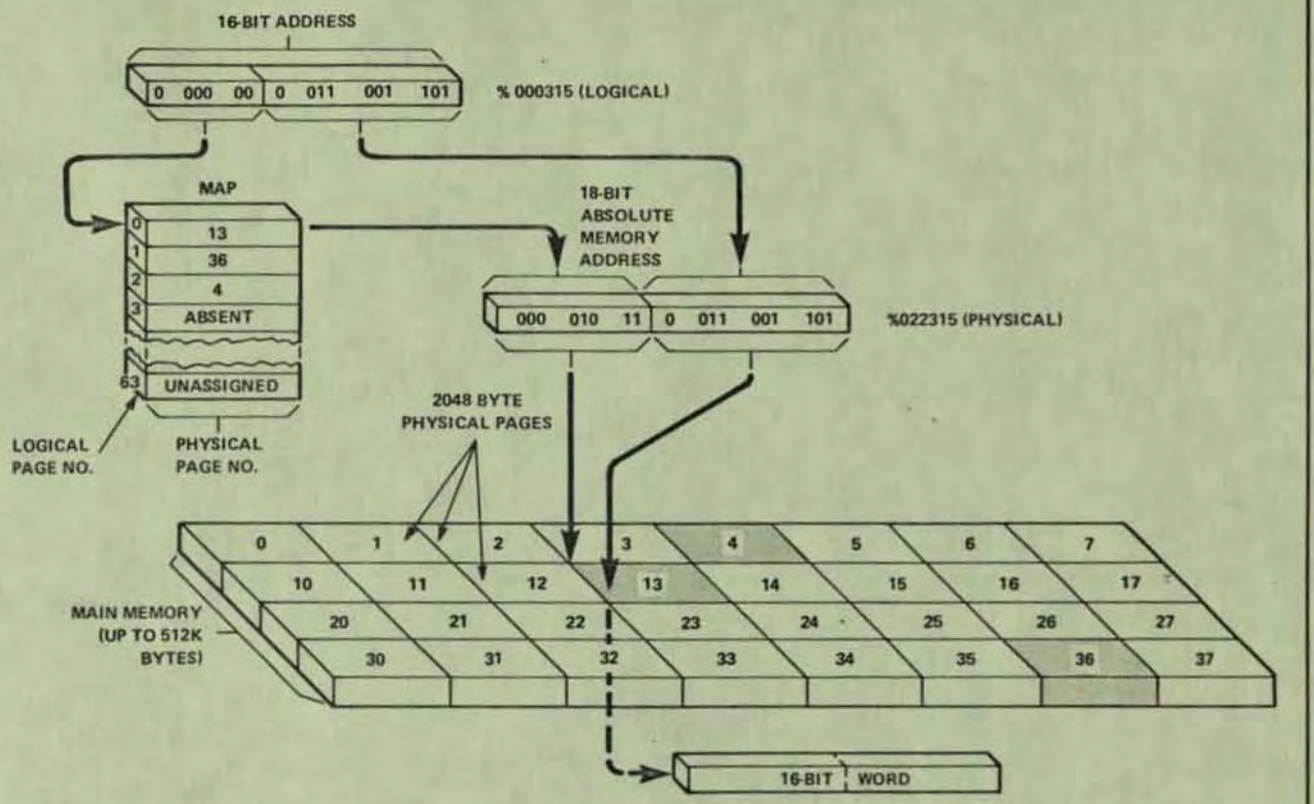
The use of the process structure permits each program to execute as though it has sole use of a processor module and peripherals attached to the system.

- Memory Mapping

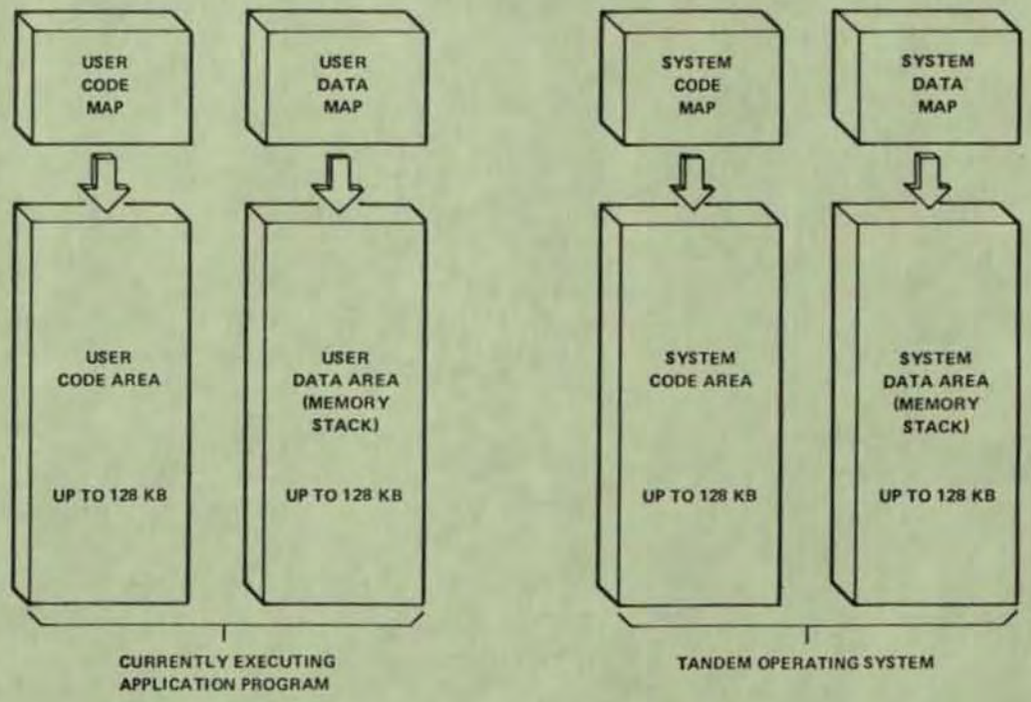
Main memory space is arranged in 2048-byte pages. A memory mapping scheme associates the 16-bit addresses used in a program with the physical pages in memory. The maximum number of pages allotted a process is 128 pages: 64 code pages and 64 data pages. The pages as addressed in a program are referred to as logical pages, the actual pages in memory are called physical pages. The first page in a program's code space and data space are referred to as logical page 0. Because of the mapping scheme, a program's memory pages need not be located contiguously.

Four separate memory maps are provided: the system code map points to the area where the operating system's instruction codes are located, the system data map points to the operating system's memory stack, the user code map points to the currently executing application program's code space, and the user data map points to the currently executing application program's data space. Because actual memory locations are associated with entries in a map register, pages assigned to a particular program can reside in non-contiguous locations. The user code and data maps are shared by all user processes present in a processor module. The operating system dynamically loads the user maps from a table indicated by the appropriate process control block just prior to dispatching that process for execution. The system maps are used only by the operating system and are never modified (except for entries of non-resident pages).

Because all program addresses are relative to logical pages rather than actual memory locations, all Tandem 16 programs (code and data) are inherently relocatable (i.e., can be positioned anywhere in memory). This means that the operating system does not require a special function for relocating programs in memory; program addresses need not be readjusted each time the operating system dispatches a process for execution.



*FOUR MAPS SEPARATE CODE FROM DATA; PROTECT THE OPERATING SYSTEM FROM APPLICATION PROGRAMS



Memory Mapping

- Virtual Memory

The memory management function of the Tandem 16 Operating system automatically brings memory pages to main memory from the system disc as required by the currently executing program. Because all Tandem 16 programs are inherently relocatable and because a program's pages need not be in contiguous locations, a page can be swapped from disc into any available page in memory (the memory manager just makes the appropriate map register entry). In fact, a process can execute with only two pages in main memory: the code page containing the current instruction and the data page referenced by that instruction.

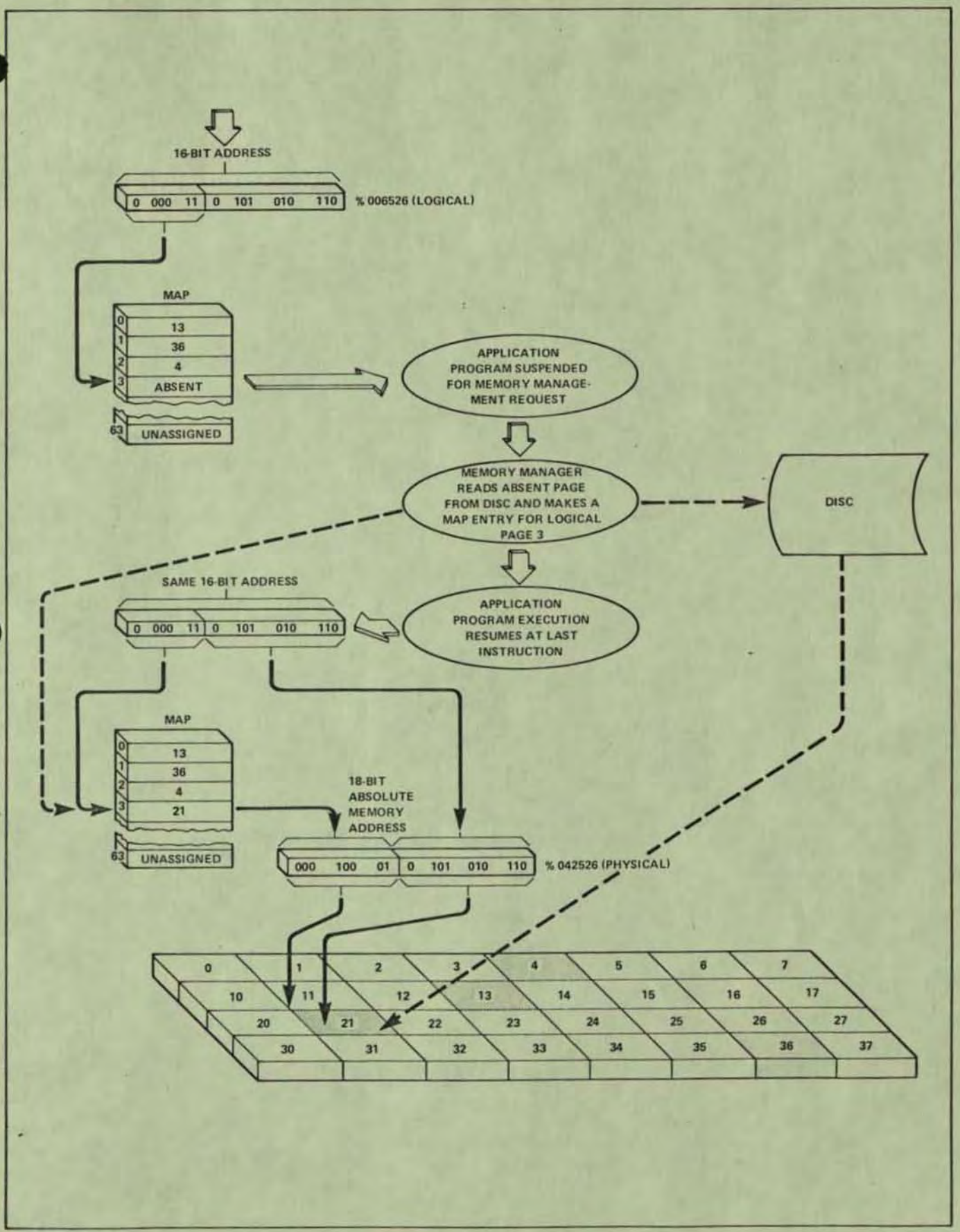
A number of features are incorporated into the processor module's hardware that aid the memory manager in reducing the amount of swapping that occurs.

– Because code is non-modifiable, code pages are never swapped out to disc.

– Because code is sharable by multiple programs, only one copy of a program need exist in memory; if a needed code page is already in memory it need not be swapped in.

– A "dirty" bit is associated with each data map entry; only if a data page has been modified is it ever swapped out.

– History bits are associated with each map entry to record access and overlays occurring with a particular page. The memory manager also maintains a list of maps (each process has a separate map) active in a processor module. When memory space is needed for an overlay, the memory manager selects the map that is at the head of the map list then selects the least accessed page in that map for overlaying. The memory manager ensures that processes are selected on an equal basis for potential overlays by putting the last map selected for overlay at the tail of the map list.



Virtual Memory

- High Performance Processor Modules

The central processing unit's micro-instruction cycle time is 100 nanoseconds; microinstructions are 32 bits in length; 6 general purpose registers are available to the micro-processor. The memory cycle time for accessing a 16-bit word in semi-conductor memory is 500 nanoseconds, including accessing the map registers and, if necessary, error correction. The cycle time for core memory is 800 nanoseconds (including mapping and parity checking). The next instruction to be executed is prefetched while the current instruction is being executed.

The instruction set contains 122 instructions; each instruction is 16 bits in length. Twelve memory reference instructions can directly address any of five data areas in memory: global, local, system global, parameters, and sublocal. Two instructions are provided for reading constant information from a program's code area. The data and code memory reference instructions can use the contents of the direct memory location as an indirect reference to another location; any of these instructions can be indexed (three index registers are provided).

Instructions are provided for string moves, scans, and compares (both eight-bit and sixteen-bit quantities).

- High Performance Inter-processor Buses

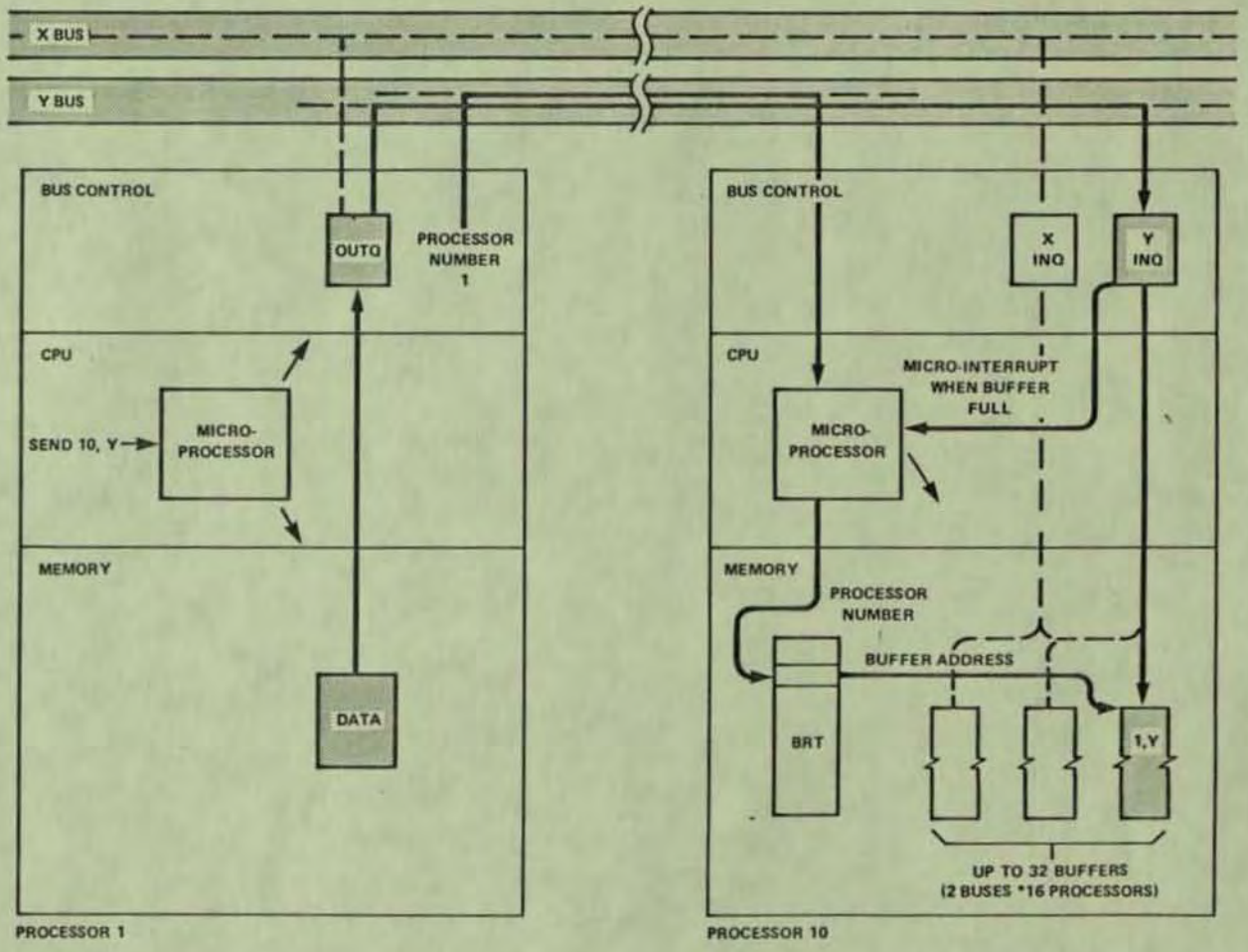
Data is transferred over each interprocessor bus at a 10 megabyte per second rate. Each bus is capable of transferring data between all processor modules concurrently on a multiplexed basis. Data transfers can also be occurring simultaneously on both buses. The operating system is designed to keep both buses operating at peak efficiency.

One hardware instruction (SEND) is used to transmit blocks of 1 to 32,767 words to a designated processor module over a designated bus.

Data, as far as the hardware is concerned, comes into a processor module unsolicited (i.e., there is no corresponding "RECEIVE" instruction).

Data is actually sent across a bus in "packets" of 16 words (15 data words plus one checksum word); each processor module contains two high-speed 16-word buffers (one for each bus) for receiving the incoming information. These buffers are designated INQ X (for the X bus) and INQ Y (for the Y bus). Transfers into the buffers occur simultaneously with microprogram execution; when a buffer fills, the microprogram is interrupted and a special microroutine stores the block in memory.

A table (called the Bus Receive Table – BRT) is maintained in each processor module's memory to direct the incoming data to a specified location in memory. The table contains a maximum of 32 entries (corresponding to the two buses from each of 16 processor modules). Each entry specifies a buffer address where the incoming data is to be stored and the number of words expected. When the expected number of words has been received the currently executing program is interrupted.



Interprocessor Buses

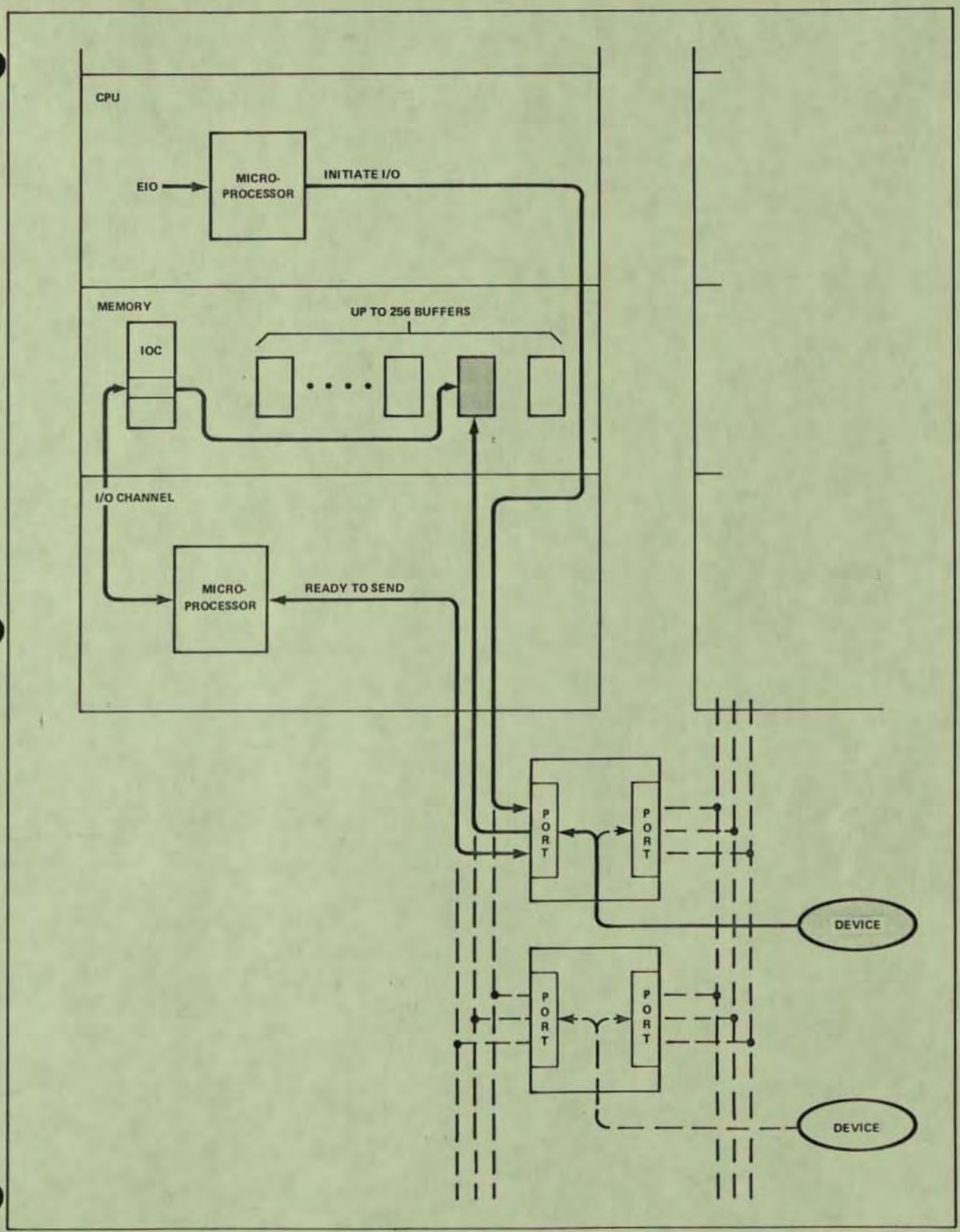
- High Performance Input/Output Channel

  Input/output transfers occur directly between memory and i/o devices concurrently with program execution. A single i/o operation is capable of transferring data in blocks of from one to 4094 bytes.

  One hardware instruction (EIO) is used to initiate input/output operations. Once i/o is initiated, a special microprocessor contained in the i/o channel controls the transfer of data between an i/o device and memory.

  A table (called the i/o control table – IOC) is maintained in each processor module's memory to permit the channel's microprocessor to control the operation on a device basis. The table contains up to 256 entries corresponding to the 256 possible devices on a channel; each entry contains a buffer address (in the system data area) and a count of the number of bytes to be transferred. Data transfer occurs simultaneously with CPU program execution; CPU execution is suspended only when both the channel and CPU need to access memory at the same time. When the number of bytes indicated in the IOC have been transferred, the currently executing program is interrupted.

  Data is buffered by each dual-port controller so that data is transferred in bursts over a channel at memory speed (the number of bytes in a "burst" is dependent on a controller's buffer size). Controllers are designed so that they signal the channel prior to actually emptying (during a write operation) or filling (during a read operation) their buffer. This gives the channel ample time to respond thereby reducing the possibility of data overrun conditions. Each i/o channel is capable of communicating with a maximum of 256 input/output devices (32 controllers with up to eight units per controller); all 256 devices can be transferring simultaneously (with "bursts" from one device being interleaved with "bursts" from others).



Input/Output Channel

- System Integrity

  Each processor module contains a number of features that assure system integrity:

  – A checksum word is generated by the transmitting processor module and checked by the receiving processor for every 15 words transferred over an interprocessor bus.

  – One parity bit is associated with each 16-bit word transmitted over an i/o channel.

  – An interval timer is provided; the operating system and the file system use the timers to notify the application program in the event a data transfer does not complete.

  – When the semiconductor memory is used, six error correction bits are generated and stored with each 16-bit word in memory; circuitry is provided that corrects all single bit errors and detects ALL double bit errors. With core memory, a parity bit is generated for each 16-bit word.

  – The addressing and count information controlling an i/o transfer is kept in the controlling processor module. This prevents a controller from contaminating more than one processor module because of a failure of an address or word count register.

  – The file system protects against an input/output device from erroneously writing into memory (in the IOC table, either the device's count field is set to zero or its write only bit is set).

  – Each entry in the registers that are used for memory mapping has an associated parity bit.

  – Because the memory mapping scheme provides separate system/user maps, operating system areas can only be accessed by operating system programs; application programs cannot inadvertently destroy the operating system.

  – Two hardware modes of processor operation are provided: privileged and nonprivileged. Certain critical operations (such as initiating input/output transfers or accessing system tables from application programs) can be performed only while in privileged mode. Typically, only the operating system operates in privileged mode; privileged operations are performed on behalf of application programs through calls to operating system procedures. Application programs running in nonprivileged mode cannot inadvertently become privileged.

- Efficient File System

  The file system is designed so that an application program can execute concurrently with its own input/output. Additionally, the system can be configured so that while an application program runs in one processor module, its file system operations can be occurring simultaneously in another module.

- Access to Large Data Bases

  On-line access to large data bases is possible for two reasons: the large number of peripheral devices that can be attached to the system (a 16 processor system can directly address 2048 input/output devices) and an application program can communicate with ANY device connected to the system.

- Large Number of On-line Terminals

  Any two processor modules can handle up to 128 data communication lines; each line is capable of communicating with one single-drop terminal or many multi-drop terminals.

- **Fast Response**

Fast response to on-line inquiries is possible because application programs are scheduled for execution by the operating system according to a priority assigned when a program is first readied for execution.

Programs that require precedence over other programs can be scheduled with a higher priority. Additionally, a program can dynamically change its own execution priority, permitting it to compensate for any unusual load conditions that may arise.

Frequently used and critical portions of the operating system are main memory resident (i.e., are never swapped out). This enables operating system functions to be called into execution without having to wait for a page to be swapped in from disc. Additionally, part or all of an application program can be made main memory resident for applications requiring a guaranteed response time.

The area in memory where instruction codes are stored (i.e., referenced by the user and system code maps) cannot be modified. This means that, if a program is suspended for some reason, the memory area can be overwritten immediately; it's not necessary to write this area out to disc. A program's data area is only written out to disc if it has been changed since last loaded into memory. Nonmodifiable code provides an additional benefit: when one program is suspended for some reason, another program can use the same code. This saves time because the code area need not be swapped into main memory and saves space because only one copy of a program need reside in memory.

EASILY ADAPTABLE

The Tandem 16 Computer System's software has been specifically engineered so that applications can be defined and implemented with a minimum of time and expense.

- Many of the responsibilities normally handled by applications programs with other systems are taken care of automatically by the Tandem 16's operating system (T/TOS).

  – The virtual memory scheme incorporated into the Tandem 16 enables programmers to concentrate fully on the intended application. Programs or portions of programs are swapped between memory and disc automatically by the operating system; the swapping is invisible to the application program.

  – The multiprogramming, multiprocessing features of the Tandem 16 operating system permit programs to be written without regard for other programs running in the system and without regard for the processor module in which a program is eventually run.

  – Programmers can write programs that communicate with input/output devices and other programs without actually knowing where the devices are connected to the system or where other programs are physically executing the system.

  – Using the file system, the hardware aspect of input/output transfers is transparent to application programmers; completion interrupts and system dependent error conditions are handled automatically.

  – Transfers over the interprocessor buses are handled entirely by the operating system; the bus operation is completely invisible to application programmers using the file system.

  – For application dependent error recovery routines (such as not ready on a magnetic tape), the file system provides an error number that specifically describes any errors encountered during an input/output operation.

- User/System Interface

  Application programs and programmers typically make use of the Tandem 16 Computer System through these means:

  – Command Interpreter Program (COMINT)

  Programmer/operator control (versus program control) over the system is by means of a Tandem-supplied program called the Command Interpreter (COMINT). The Command Interpreter performs its functions by conversing with a user through an on-line terminal device; the command interpreter prompts the user for a command, the user enters a command, the command interpreter executes the command, then prompts the user for another command. (Typically, the operating system is configured so that the Command Interpreter program initially executes on one of the console devices. From this point, COMINT can be run on other terminals connected to the system.)

  Functions that the Command Interpreter performs are: obtaining and altering the current operational status of the system, obtaining information about disc files, creating and purging disc files, informing the operating system that a disc volume is to be mounted or dismounted, and running programs (application programs and Tandem-supplied programs such as EDIT, TAL, TOSYSGEN, or COMINT).

For example to obtain the operational status of the system, the operator enters the following command:

```
:STATUS
└— Command Interpreter prompt
```

The command interpreter displays system status information on the terminal.

Entering a command name not known by the Command Interpreter results in an attempt to run a program by that name. Parameter information can also be passed to the program at that time. For example, to run a program called SORT and specify two file names (INFILE and OUTFILE) to the program, the following could be entered:

```
:SORT INFILE, OUTFILE
```

– Editor Program (EDIT)

The EDIT or program is used by applications programmers to enter and/or modify source language programs through an interactive terminal.

The editor program is typically run through use of the command interpreter program:

```
:EDIT MYSRCE
```

(MYSRCE is a text file to be accessed by the editor)

Editor functions are invoked interactively by issuing commands to the editor program via an online terminal:

```
& LIST ALL OUT $LP
└— editor prompt
```

(lists the entire contents of the current text file on the system line printer)

The EDIT program also has the capability to communicate directly with application programs running in the system. This permits specialized text formatting programs to be written and permits applications programs to take advantage of the editor's text manipulation capabilities.

– Tandem/Transaction Application Language Compiler Program (TAL)

The T/TAL Compiler program is used to prepare ready-to-run programs from source programs written in Tandem's Transaction Application Language (T/TAL). T/TAL is a high-level language, designed especially for transaction processing. A typical statement written using T/TAL that compares two character strings might be:

```
IF INARRAY = "$RECEIVE" THEN . . . .;
```

Or to call a procedure for execution

```
CALL COMPUTETAX(AMOUNT,RATE,TAX);
```

(The name assigned to the procedure is COMPUTETAX; AMOUNT and RATE are parameters; TAX is the result)

The T/TAL compiler program is typically run through use of the Command Interpreter program (but could be run through an application program):

```
:TAL MYSRCE,MYOBJECT, $LP
```

(MYSRCE is the file containing T/TAL source statements, MYOBJECT is the disc file where the ready-to-run object program is stored by the compiler, $LP indicates where the compiler listing is to be sent)

– File System

The file system provides access to all input/output devices in a uniform manner. File system operations are invoked through calls to library procedures that are part of the operating system. Page mode, multi-drop, and conversational mode terminals, other devices, discs and portions of discs, and other programs are accessed as files. This permits programs to be written without regard for the actual physical location of the device or program to be accessed. File names are assigned to devices when the system is configured, to disc files when created, and to other programs when they are run. As far as the application program is concerned, interprogram communication appears identical to input/output with physical devices.

A typical T/TAL statement to write to a page mode terminal might be:

```
CALL WRITE (TERMFILE, BUFFER, 600, NUMWRITTEN, TENSECONDS);
```

(WRITE is a file system procedure; TERMFILE is a file number assigned to a terminal by the file system; BUFFER is an array in the user's memory stack containing the data to be written on the terminal; 72 is the number of bytes to be written; NUMWRITTEN is the actual number written on the terminal, TENSECONDS is a timeout value assigned to completing the write)

– Tandem 16 System Generator Program (TOSYSGEN)

A Tandem 16 Computer System is fully configured and ready-to-run when installed on the computer site. A Tandem-supplied system configuration program (TOSYSGEN) is available for tailoring the operating system to suit a particular application. Input/output devices can be added or reassigned, i/o device characteristics are configured, space is assigned for system buffers, programs can be designated that automatically start executing (i.e., without using the Command Interpreter) when the operating system is loaded into the system.

To configure a system, the EDIT program is used to "fill in the blanks" in a disc file that is supplied with the system. Then the TOSYSGEN program is run. It reads the configuration file and generates a fully configured operating system (usually on a disc) that is ready to be loaded into the system.

SUMMARY OF TANDEM 16 FEATURES

Tandem 16 Computer System (fail-safe, transaction oriented, easily adaptable)

- Two to sixteen processor modules
- Dual, high-speed interprocessor buses
- High-speed, burst-multiplexed input/output channel
- Multiprocessing, multiprogramming, transaction oriented, fail-safe operating system (T/TOS)
- Virtual file system

Processor Modules

- Microprogrammed (100 nanosecond cycle time)
- Sixteen bit data paths and memory addressing
- Up to 512k bytes memory per processor module.
- Memory mapping (four separate maps: system data, system code, user data, user code)
- Up to 128k bytes addressable through each map
- 500 nanosecond semiconductor memory access time (including mapping and error correction)
- 800 nanosecond core memory access time (including mapping and parity checking)
- 122 instructions (including string manipulation and doubleword arithmetic)
- Stack architecture (memory stack and register stack)
- Procedure oriented hardware
- Memory references: direct or indirect with or without indexing to global, local, procedure parameters, top of stack, or system global data areas and to core area. Word, doubleword, byte addressing
- No machine instruction can write into code area (non-modifiable code)
- All programs are inherently re-entrant and relocatable
- I/O, bus receive, and instruction execution can occur concurrently
- Next instruction prefected while current instruction executes
- Hardware power fail/auto restart
- Hardware multiply/divide
- Maximum of five microseconds to call operating system procedures

Interprocessor Buses

- Two paths between each processor module
- 10 megabyte transfer for each bus
- Packet-multiplexed transfers between any number of processor modules
- Block transfers of 1 to 32,767 bytes
- Both buses used simultaneously

Input/Output Channels

- Dual-port Controllers
- Data transfers occur at memory speed (4 megabytes per second with semiconductor memory)
- Up to 256 devices per processor module
- Burst-multiplexed transfers from any number of devices
- Block transfers of 1 to 4,094 bytes

T/TOS Operating System

- Multiprocessing
- Multiprogramming
- Geographical independence
- Process structure
- Memory management function makes virtual memory invisible to users
- Processes scheduled for execution according to priority (0-255)
- Sharable code areas handled automatically
- Up to 256 processes per processor module

File System

- Two paths to each i/o device; file system automatically switches paths
- All devices and programs are accessed as files
- Geographic independence
- Timeout associated with each i/o operation

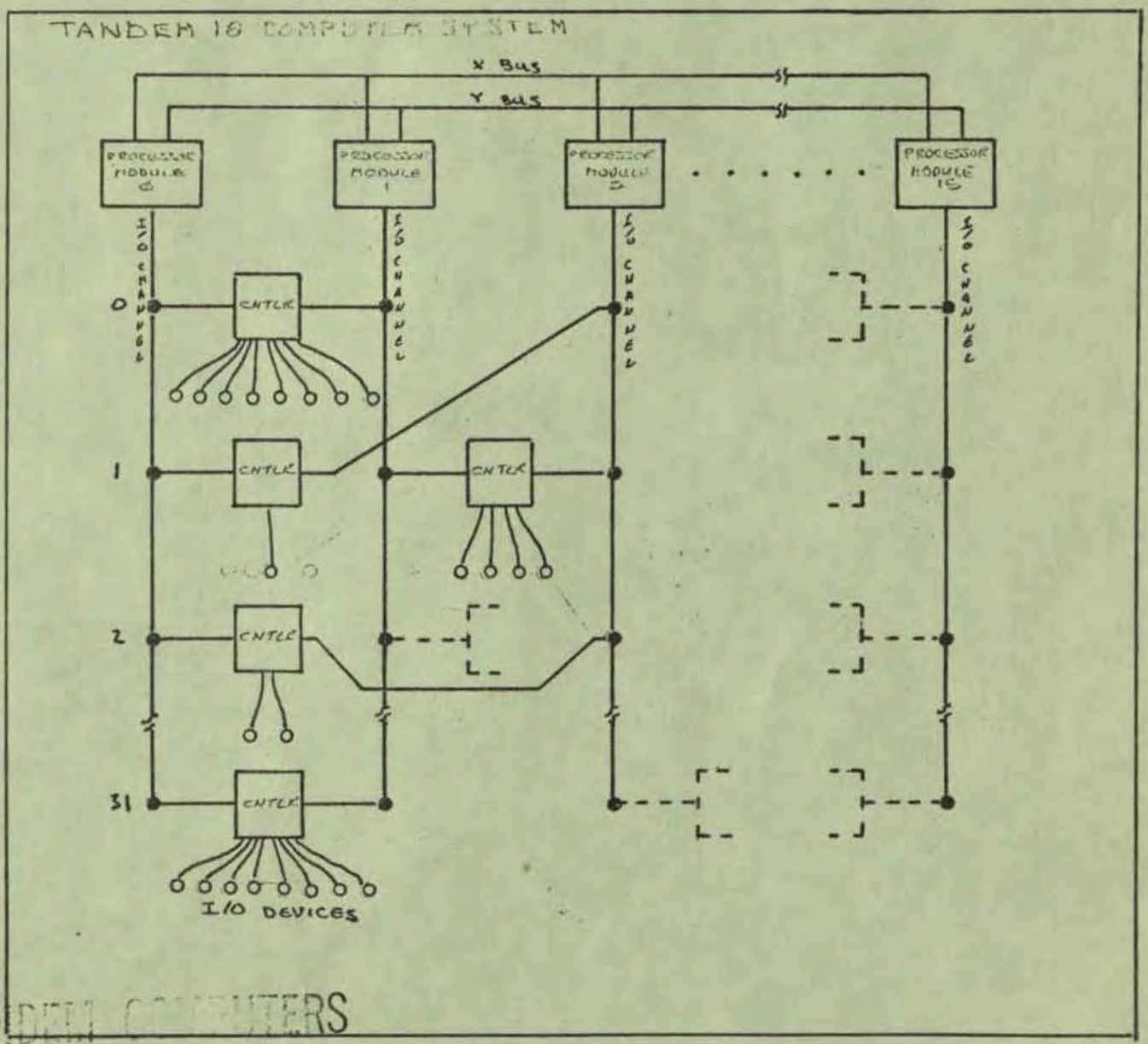
For more information regarding the Tandem 16 Computer System, refer to the following manuals:

- Tandem 16 System Manual (detailed description of system from a hardware and software standpoint and information on making an application "fail-safe")
- Tandem 16 Programming Manual (describes T/TAL, using the file system, using the process control procedures, using the Tandem-supplied utility procedures, picking up the parameter message from the Command Interpreter, and using the DEBUG feature)
- Tandem 16 Text Editor Manual (describes the EDIT commands and how to communicate with the editor from another process)
- Tandem 16 System Generation Manual (describes how to configure a system and how to run the Tandem-supplied TOSYSGEN program)
- Tandem 16 Operation Manual (describes how to run the Command Interpreter program, explains the Command Interpreter command set, how to remove and mount disc volumes, how to "down" system modules for maintenance or repair, how to run programs).

SYSTEM STRUCTURE

A Tandem 16 Computer System consists of two to sixteen processor modules. Each processor module is capable of operating autonomously, but can communicate with other processor module by means of either of two interprocessor buses.

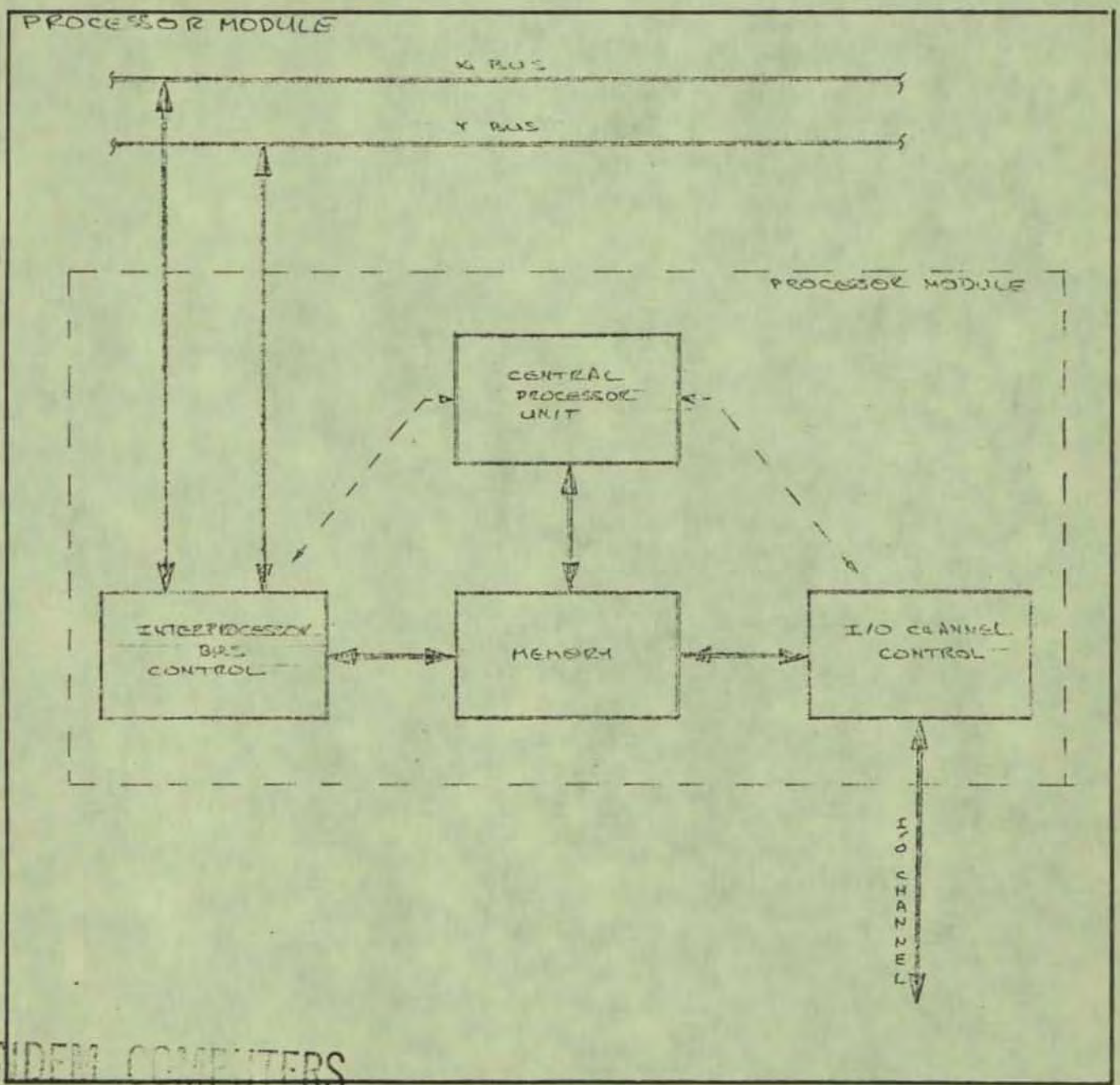
Data is transferred between input/output devices and a processor module by means of a simple but very fast input/output channel. Each processor module has one i/o channel. I/O devices are interfaced to an i/o channel by means of dual-port controllers. Each dual-port controller is connected to the input/output channel of two processor modules.



PROCESSOR MODULE

Functionally, a processor module can be separated into four parts:

* Interprocessor Bus
* Central Processing Unit
* Memory
* Input/output Channel



CPU

The CPU executes programs by fetching and execting machine instructions from memory. The instruction set consists of 123 instructions, including arithmetic operations, logical operations, bit manipulation, block (multiple element) moves/compars/scans, interprocessor bus send, and input/outut instructions.

The CPU provides for orderly interruption of executing programs to points in the operating system called interrupt handlers. Interrupts are caused by events such as power failure, uncorrectable memory error, interval clock, completion of bus transmission, or completion of input/output transfer.

Two modes of program exection are provided: privileged mode and non-privileged mode. Privileged mode permits so-called privileged instructions such as EIO (execute input/output) or SEND (send data over an interprocessor bus) to be executed. Normally only the Tandem 16 operating system (T/TOS) executes in privileged mode. Any attempt by a non-privileged program to execute a privileged instruction results in an interrupt to the operating system.

The basic unit of information is the 16-bit word. All Tandem 16 machine instructions are 16-bits in length as are logical memory addresses. This means that a program can consist of 128k words (65,536 words of code -- machine instructions --, 65,536 words of data). Data elements can be addressed on half-word (byte), word, and double-word boundaries. Three index registers are provided for element indexing; on level of indirect addressing is permitted.

Memory

Although the logical address space of a program is limited by the 16-bit address, the physical size of memory can extend to 256k words. This permits multiple programs and critical parts of the operating system to reside in main memory. The conversion of the 16-bit logical address to the 18-bits required to address 256k words of accomplished through a mapping scheme. Four maps are provided; each map consists of 64 entries, one map entry corresponds to 1024 words of memory (called a page of memory). The four maps provide separate access to user code and user data (a user is defined as the currently executing application program) and to system code and system data (the system is defined as the Tandem 16 Operating System).

Data is transferred between memory and other processor module functions in 16-bit words. If a semiconductor memory is used, six error correction bits are appended to each 16-bit word when a word is stored. A parity bit is appended if a core memory is used. All data is verified for accuracy when read from memory. The use of the six error correction bits in the semiconductor memory permits all single bit errors to be corrected and all other errors to be detected. Any error occurring in a core memory is considered uncorrectable. Any

time an error is detected (whether correctable or uncorrectable) causes an interrupt to an operating system program (where appropriate action is taken).

Interprocessor Bus

The interprocessor bus is used to transfer data from one processor module to another. All data, while being transferred, is verified for accuracy. Data is transferred between modules in the form of 16-word packets. The interprocessor bus is capable of interleaving packets between all 16 processors; both buses can be in use simultaneously. A single machine instruction is used to send information to other processor modules. Data received from other modules is directed to main memory locations by a software configured table (also in memory). Receipt of data by a processor module occurs concurrently with machine instruction execution. Only when a bus transfer is completed is the currently executing program interrupted.

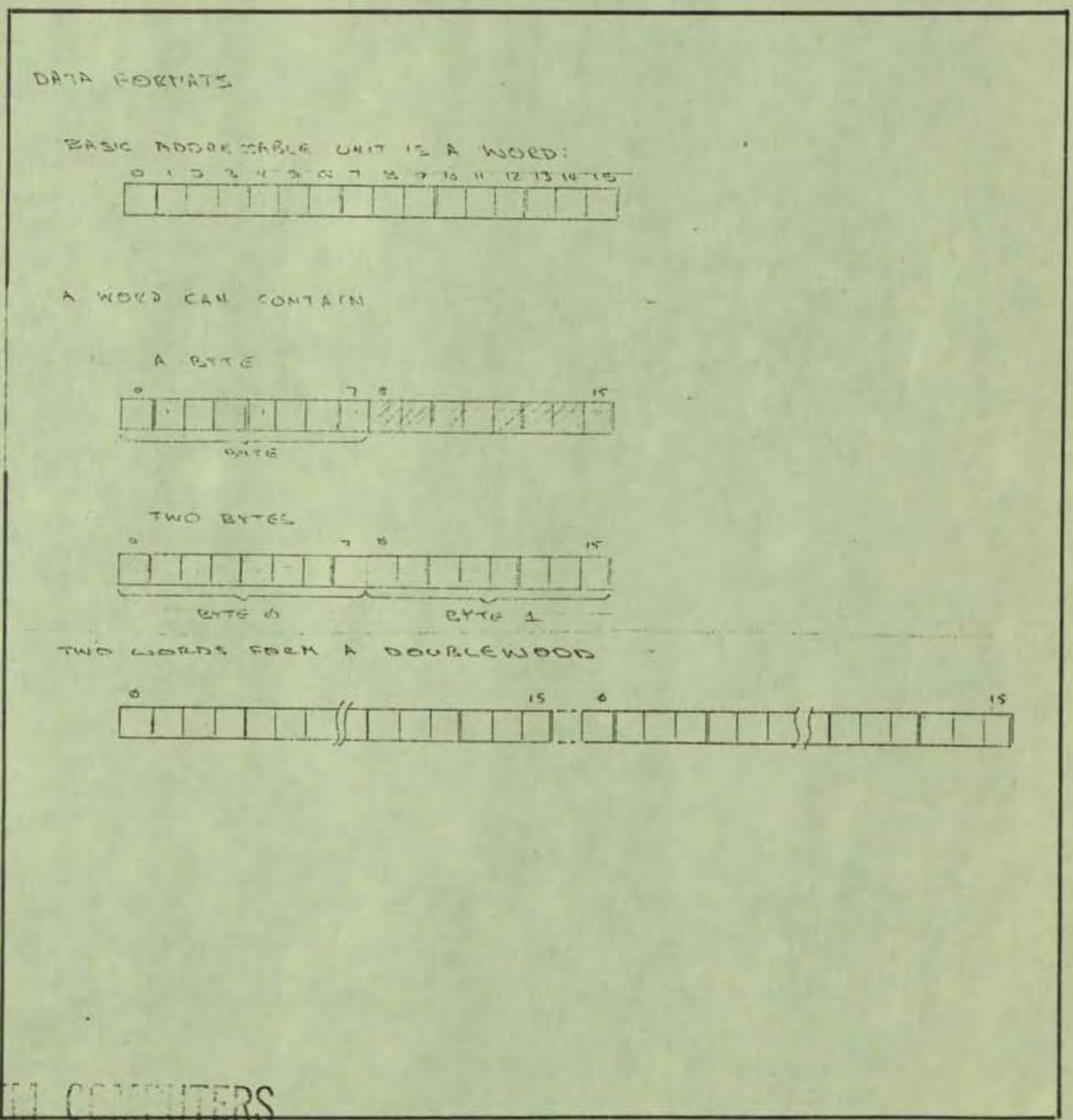
Input/Output Channel

Each processor module has on simple but very fast I/O cahnnel capable of transferring data between i/o devices and memory at full memory speed. I/O operations are initated by initializing an entry in a table in memory, then executing an EIO instruction. Once an i/o operation is initiated, data transfers occurs concurrently with software program execution. The only time the software program is affected is when both the i/o channel and the CPU need to access memory at the same time. If this occurs, the software program is momentarily suspended while a word is transferred between memory and the i/o channel (the action is invisible to the executing program). When an i/o operation completes, the currently executing program is interrupted and an operating system program executes.

Each channel is capable of addressing 256 i/o devices; 32 controllers with a maximum of eight devices each. I/O transfers can be occurring concurrently with any number of devices.

DATA FORMATS AND ADDRESSES

As previously stated, the basic unit of information used in the Tandem 16 is the 16-bit word. The Tandem 16 instruction set permits individual access to and operations on single or multiple bits (bit fields) in a word, access of 8-bit quantities (bytes), 16-bit words, and 32-bit quantities double words).



Bits

The individual bits in a word are numbered from zero (0) through fifteen (15), from left to right:

```
                          1 1 1 1 1 1
bit: 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
```

The following notation is used in this manual (and in Tandem Application Language -- T/TAL) to describe bit fields.

```
bit<left bit:right bit>
```

For example, to indicate a field starting with bit four and extending through bit 15 the following notation would be used:

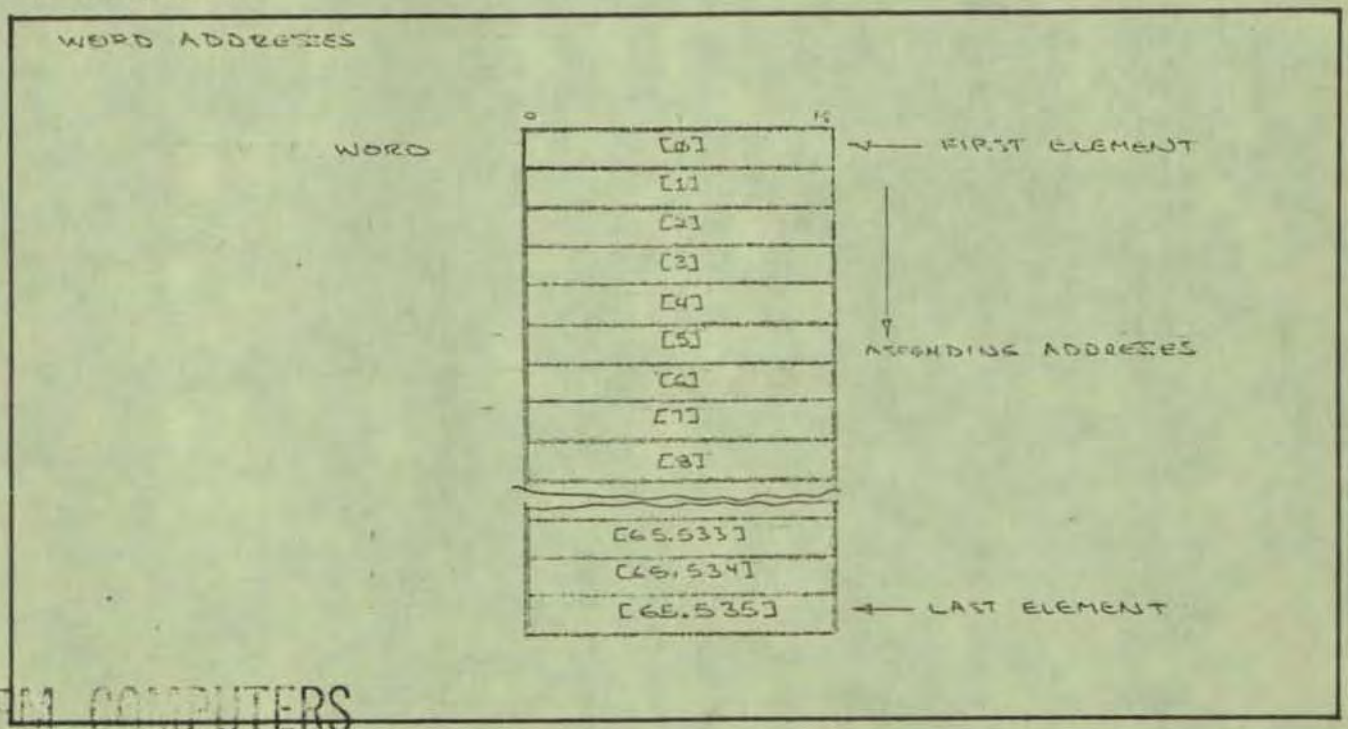
```
bit.<4:15>
```

Or to indicate just bit 0 (zero) the following is used:

```
bit.<0>
```

Words

The 16-bit word defines the Tandem 16's machine instruction length and its logical addressing range. The 16-bit word is the basic addressable unit stored in memory. The first word in logical memory is addressed as word[0], the last addressable location is word[65,565].



Note: In this manual and in T/TAL, a number surrounded by brackets is used to denote an individual element in a block of elements. For example, to indicated to fourth element in a word block, the following notoation is used:

```
word[4]
```

When operating on block of words (or any elements), the first element is the lowest numerical element; the last element is the highest numerical element. The following notation is also used to denote a block of elements:

```
words[first element:last element]
```

For example, to indicate the second through twentieth words in a block, the following notation is used:

```
words[4:20]
```

A number of instructions are provided for explicitly referencing words. These are:

LOAD: Load Word
STOR: Store Word
LWP: Load word from Program (code) area
NSTO: Non-destructive Store Word

Two instructions are available for operating on blocks of words. They are

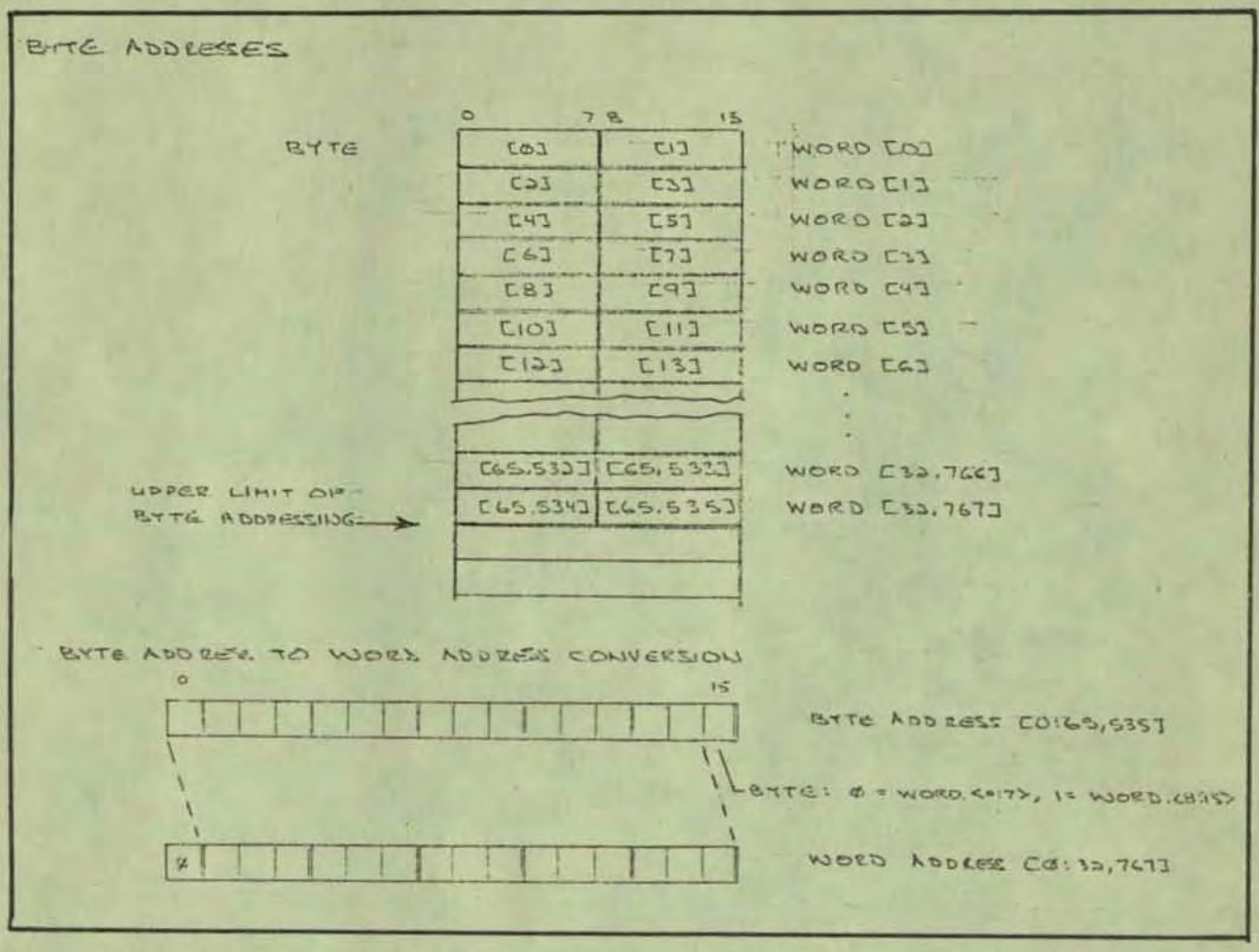
MOVW: Move words (move a block of words from one memory location to another)
COMW: Compare words (compare one block of words with another)

Note that when addressing a block of words (as in the MOVW instruction), the leftmost word (i.e., lowest numerical element) is addressed.

Bytes

The 16-bit word has the capability of storing either one or two bytes. When a single byte is stored in a word, the byte occupies bits.<0:7> (left half); bits.<8:15> are set to zero. When two or more bytes are stored as a block (group) of bytes, two bytes are stored per word. The leftmost byte, bits.<0:7> is address as byte[0], the other byte is addressed as byte[1].

The 16-bit address provides for element addressing of 65,536 bytes. Byte locations are addressed starting at byte[0] and extend through byte[65,535]. Because two bytes can be packed per word, only the first 32,768 words of logical memory are availble for byte addressing. The CPU converts a byte address to a word and element address as follows:



A number of instructions are provided for explicitly referencing bytes. These are:

LDB: Load Byte
STB: Store Byte
LBP: Load Byte from Program (code) area
BTST: Byte Test

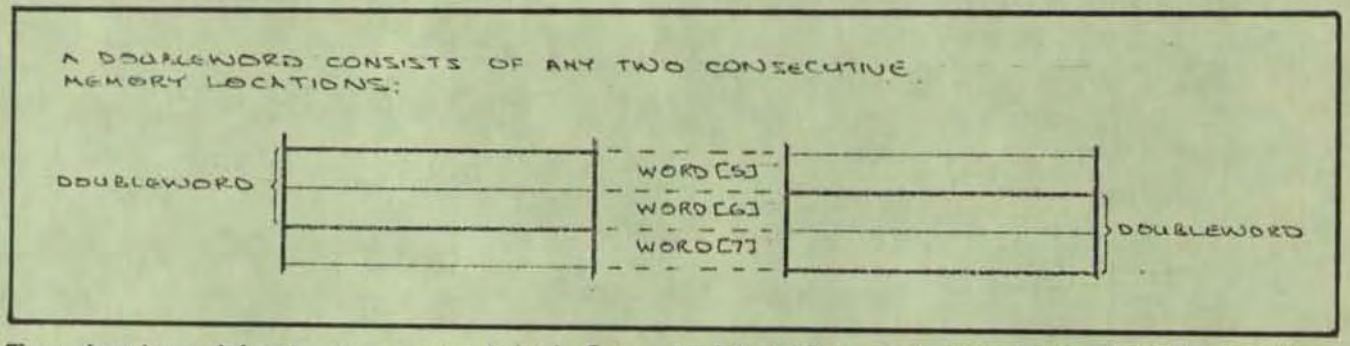
A number of instructions are available for operating on blocks of bytes. They are

MOVB: Move Bytes (move a block of bytes from one memory location to another)
COMB: Compare Bytes (compare one block of byte with another)
SBW: SCAN bytes While (look for a partical byte in a block of bytes)
SBU: SCAN bytes Until

When addressing a block of bytes, the lowest numerical element is addressed.

Doublewords

Two 16-bit words can be addressed as a single 32-bit element. The 16-bit address provides for addressing of doubleword elements in all 65,536 words of logical memory.



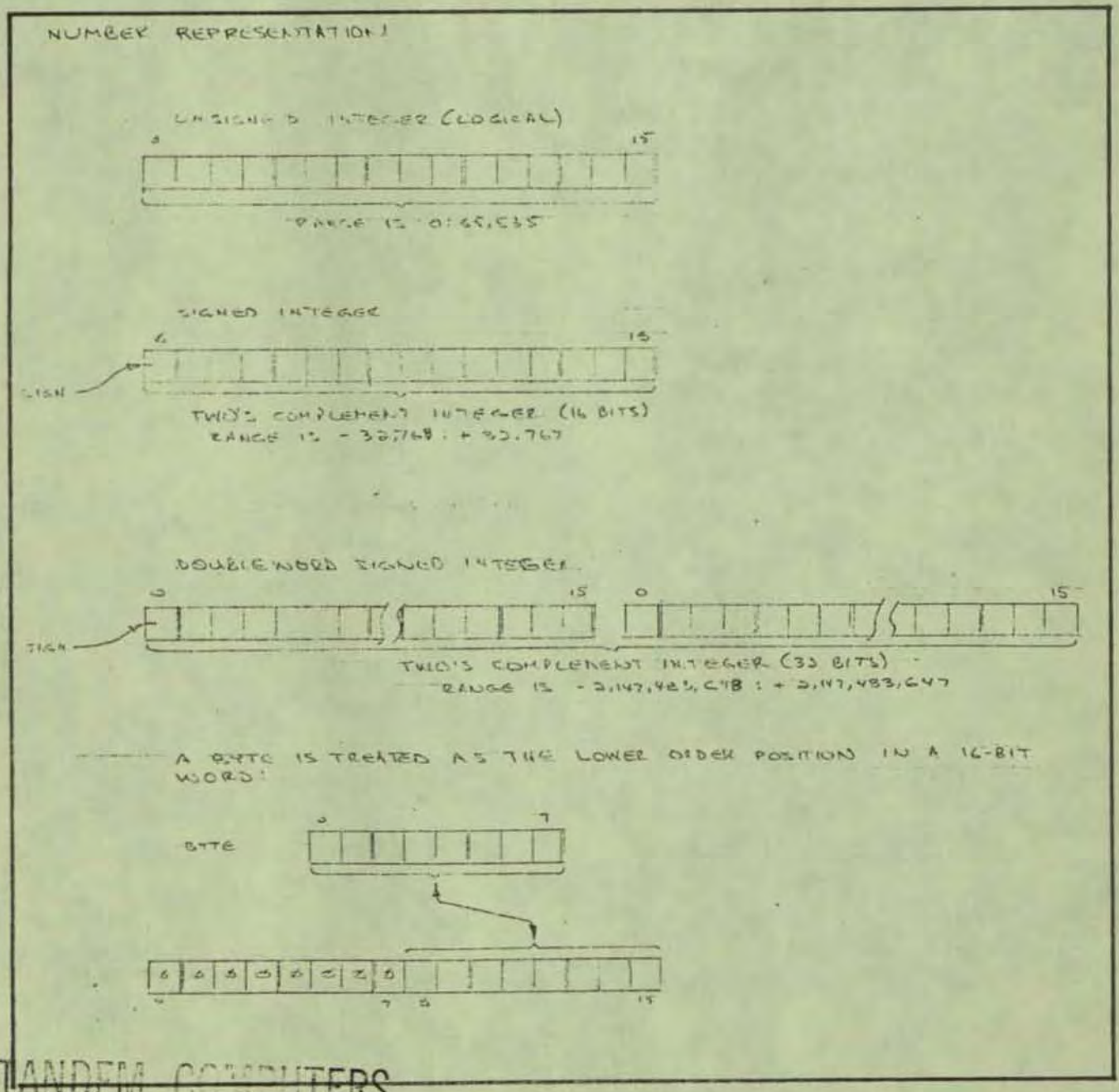
Two instructions are provided for explicitly referencing doublewords. They are:

LDD: Load Double
STD: Store Double

NUMBER REPRESENTATION

The Tandem 16 can treat an operand as either integer (signed) or logical (unsigned) number. Integers can be represented in either 16 bits (a word) or 32 bits (doubleword). Representation of logical numbers is restricted to 16-bit quantities.

Byte operands are treated as the right-half (i.e., bits.<8:15>) of a 16-bit operand (the left-half is set to zero). Normal 16-bit integer or logical arithmetic is performed.



Additionally, three hardware indicators are subject to change as the result of an computation or comparison. They are:

* Condition Code Indicator (CC) -- generally, indicates if the result of a computation was a negative value, zero, or a positive value. (The condition code can be checked by one of the branch on condition code instructions and alter program execution sequence accordingly.)

* Carry -- indicates that a carry out of the high order bit position occurred.

* Overflow -- indicates that the result of a computation could not be represented in the available number of bits in the data format.

Logicals

Logical operands represent positive values. The range of numbers that can be represented by a 16-bit logical operand is 0 through 65,535.

Some of the instructions for logical operands are:

LADD: Logical Add
LSUB: Logical Subtract
LMPY: Logical Multiply
LDIV: Logical Divide
LNEG: Logical Negate
LCMP: Logical Compare
LADI: Logical Add Immediate

The results obtained from a logical add and subtract are identical to that obtained by integer add and subtract except that logical add and subtract do not set the Overflow indicator. The 16-bit result, the condition code setting,and the Carry indicator setting are the same. Logical divide (LDIV), however, sets the Overflow indicator if the quotient cannot be represented in 16 bits.

Integers

Integers represent either positive or negative values. The range of numbers that can be represented by a 16-bit integer is -32,768 to +32,767; by a 32-bit integer is -2,147,483,648 to + 2,147,483,647. Positive numbers are represented in true binary notation. Negative numbers are represented in two's complement notation with the sign bit (bit.<0>) set to one. The two's complement of a number is obtained by inverting each bit position in the number then adding a one in bit.<15>.

If the result of an integer computation can not be represented within the number of bits indicated by the operand type (i.e., 15 bits for a single word integer, 31 bits for a double word integer), an Overflow occurs. Overflow also occurs if a divide operation is attempted with

divisor of 0 (zero). An Overflow condition causes an interrupt to an operating system Overflow trap handler.

Some of the instructions for 16-bit integer operands are:

IADD: Integer Add
ISUB: Integer Subtract
IMPY: Integer Multiply
IDIV: Integer Divide
ICMP: Integer Compare
ADDI: Add immediate operand

Some of the instructions for 32-bit integer operands are:

DADD: Double Add
DSUB: Double Subtract
DCMP: Double Compare
DTST: Double Test

INSTRUCTIONS

The Tandem 16 instruction set consists of the 123 instructions described in detail in section 3. These instuctions are grouped functionally as follows:

* 16-bit arithmetic on top of register stack

IADD: Integer Add
LADD: Logical Add
ISUB: Integer Subtract
LSUB: Logical Subtract
IMPY: Integer Multiply
LMPY: Logical Multiply
IDIV: Integer Divide
LDIV: Logical Divide
INEG: Integer Negate
LNEG: Logical Negate
ICMP: Integer Compare
LCMP: Logical Compare
CMPI: Integer Compare Immediate
ADDI: Integer Add Immediate
LADI: Logcial Add Immediate

* 32-bit Integer Arithmetic on top of register stack

DADD: Double Add
DSUB: Double Subtract
DNEG: Double Negate
DCMP: Double Compare
DTST: Double Test
MOND: (load) Minus One Double
ZERD: (load) Zero Double
ONED: (load) One Double

* 16-bit Integer Arithmetic in register stack element (index register)

ADRA: Add a Register to A
SBRA: Subtract a Register from A
ADAR: Add A to a Register
SBAR: Subtract A from a Register
ADXI: Add Immediate to Index

* Register Stack Manipulation

EXCH: Exchange A with B
DXCH: Double Exchange
DDUP: Double Duplicate
STAR: Store A in a Register
NSAR: Non-destructive Store A in a Register
LDRA: Load A from a Register
LDI: Load Immediate
LDXI: Load Index Immediate
LDLI: Load Left Immediate

* Boolean Operations

LAND: Logical AND
LOR: Logical OR
XOR: Exclusive OR
NOT: NOT
ORRI: OR Right Immediate
ORLI: OR Left Immediate
ANRI: AND Right Immediate
ANLI: AND Left Immediate

* Bit Shift and Deposit

DPF: Deposit Field
LLS: Logical Left Shift
DLLS: Double Logical Left
DLRS: Double Logical Right Shift
ALS: Arithmetic Left Shift
DALS: Double Arithmetic Left Shift

Shift
LRS: Logical Right Shift
ARS: Arithmetic Right Shift
DARS: Double Arithmetic Right Shift

* Byte Test: BTST

* Memory <--> Register Stack

LDX: Load Index
NSTO: Non-destructive Store
LOAD: Load Word
STOR: Store Word
LDB: Load Byte
STB: Store Byte
LDD: Load Double
STD: Store Double
LADR: Load Address of Variable
ADM: Add to Memory
PUSH: Push Registers to Memory
POP: Pop Memory to Registers
LWP: Load Word from Program (code)
LBP: Load Byte from Program (code)

* Branching

BIC: Branch if Carry
BUN: Branch Unconditionally
BOX: Branch on Index
BGTR: Branch if CC Greater
BEQL: Branch if CC Equal
BGEQ: Branch if CC Greater or Equal
BLSS: Branch if CC Less
BAZ: Branch if A Zero
BNEQ: Branch if CC Not Equal
BANZ: Branch if A Not Zero
BLEQ: Branch if CC Less or Equal
BNOV: Branch if no Overflow
BNOC: Branch if no Carry
BFI: Branch Forward Indirect

* Moves/Compares/Scans

MOVW: Move Words
MOVB: Move Bytes
COMW: Compare Words
COMB: Compare Bytes
SBW: Scan Bytes While
SBU: Scan Bytes Until

* Program Register Control

SETL: Set L Register
SETS: Set S Register
SETE: Set E Register
SETP: Set P Register
RDE: Read E Register
RDP: Read P Register
STRP: Set Register Pointer
ADDS: Add to S Register
CCL: Set CC Less
CCE: Set CC Equal
CCG: Set CC Greater

* Routine Calls/Returns

PCAL: Procedure Call
SCAL: System Procedure Call
DPCL: Dynamic Procedure Call
EXIT: Exit from Procedure
DXIT: Debug Exit (privileged)
BSUB: Branch to Subprocedure
RSUB: Return from Subprocedure

* Interrupt (privileged)

RIR: Reset INT Register
XMSK: Exchange MASK Register
IXIT: Exit from Interrupt Handler
DISP: Dispatch

* Bus: SEND (privileged)

* Input/Output (privileged)

  EIO: Execute I/O
  IIO: Interrogate I/O
  HIIO: High Priority Interrogate I/O

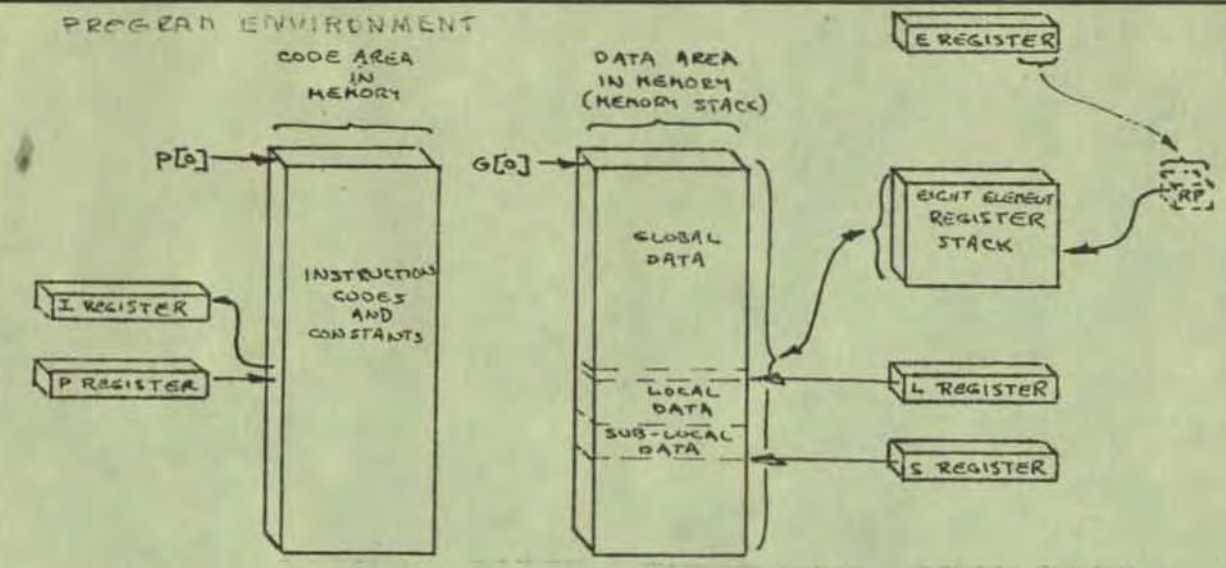
* Map Register Control (privileged)

  SMAP: Set Map
  RMAP: Read Map
  AMAP: Age Map

* Miscellaneous

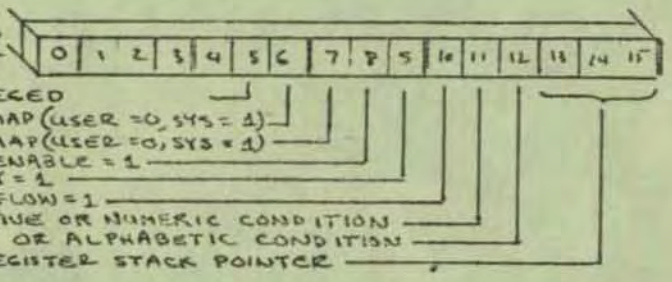
  RSW: Read Switch Register
  SSW: Set Switch Register
  NOP: No Operation
  HALT: Halt (privileged)
  RMD: Read Memory Data

PROGRAM ENVIRONMENT



DEFINITIONS:

E REGISTER: ENVIRONMENT



I REGISTER: Current Instruction Register

P REGISTER: Program Counter; Address of Current Instruction + 1 (relative to P[0])

P[0]: First element in the code area

G[0]: First element in the data area

GLOBAL DATA: Data Area Accessible from any Point in a Program

LOCAL DATA: Data Area Accessible only from currently executing Procedure

SUB-LOCAL DATA: Data Area Accessible only from currently executing Subprocedure

L REGISTER: Local Data Pointer: G[0] Relative Address of First Element in the Local Data Area. Also indicates the location in the memory stack of the link (i.e., stack marker) back to the calling Procedure

S REGISTER: Top of Stack: G[0] Relative Address of the last active element in the memory stack

REGISTER STACK: Eight-Element Register Stack where Arithmetic Operations are Performed. Three elements can also be used for Indexing

RP: Register Stack Pointer: Indicates the Top Element in the Register Stack

A Tandem 16 program in memory consists of instruction codes in a

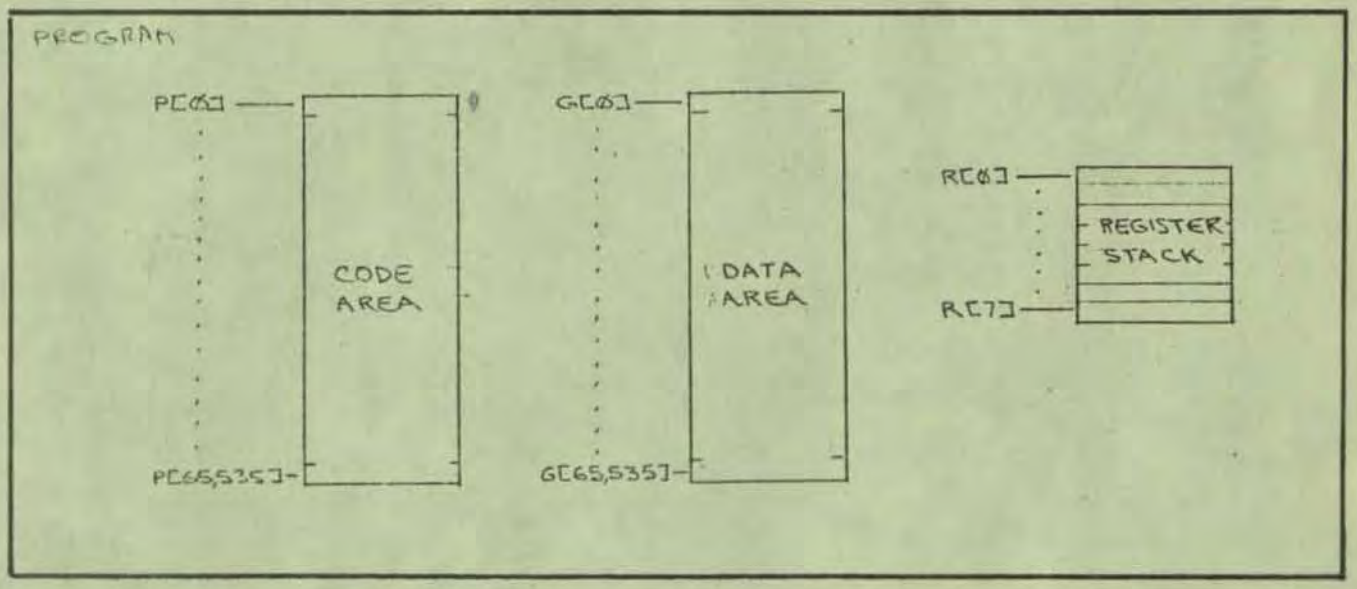
* CODE area

that manipulate variable data in a

* DATA area

using the CPU's eight-element

* Register Stack



CODE AREA: Information in the code area consists of instruction codes and constants. No instruction exists for writing into the code area, therefore the code area cannot be modified.

The code area consists of up to 65,536 16-bit words. Words in a code area are numbere consecutively from P[0] (program, element 0) through P[65,535].

DATA AREA: The data area contains a program's temporary storage locations (i.e., variables). Data in this area consists of single element items, address pointers, and multiple element items (arrays). Input/output transfers (which are performed on behalf of application programs by the Tandem 16 File Manager) are via arrays in a program's data area.

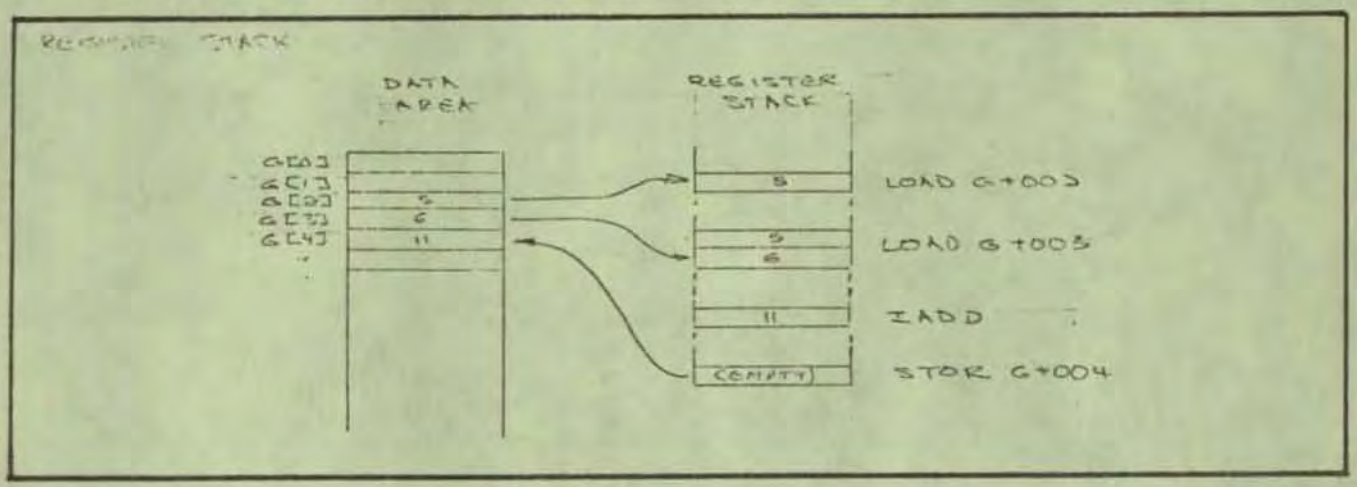
Part of the data area is used for dynamic allocation of storage when procedures are invoked; this area is referred to as the "memory stack".

The data area consists of up to 65,536 16-bit words. Words in a data area are numbered consecutively from G[0] (global data, element 0)

through G[65,535]. The "memory stack" portion of the data area is limited to the lower 32,768 words (i.e., G[0 : 32,767]).

REGISTER STACK: Arithmetic computations and comparisons are performed in the register stack. To perform a computation, the operands are first loaded into the register stack using an instruction such as LOAD, an instruction is then executed performing the desired arithmetic, the result then stored back into memory using an instruction such as STOR. Grouped together to form a program, the operation might look like this:

```
LOAD G + 002 ! load data element G[2] into register stack
LOAD G + 003 ! load data element G[3] into register stack
IADD         ! integer add
STOR G + 004 ! store the result from the register stack into G[4]
```



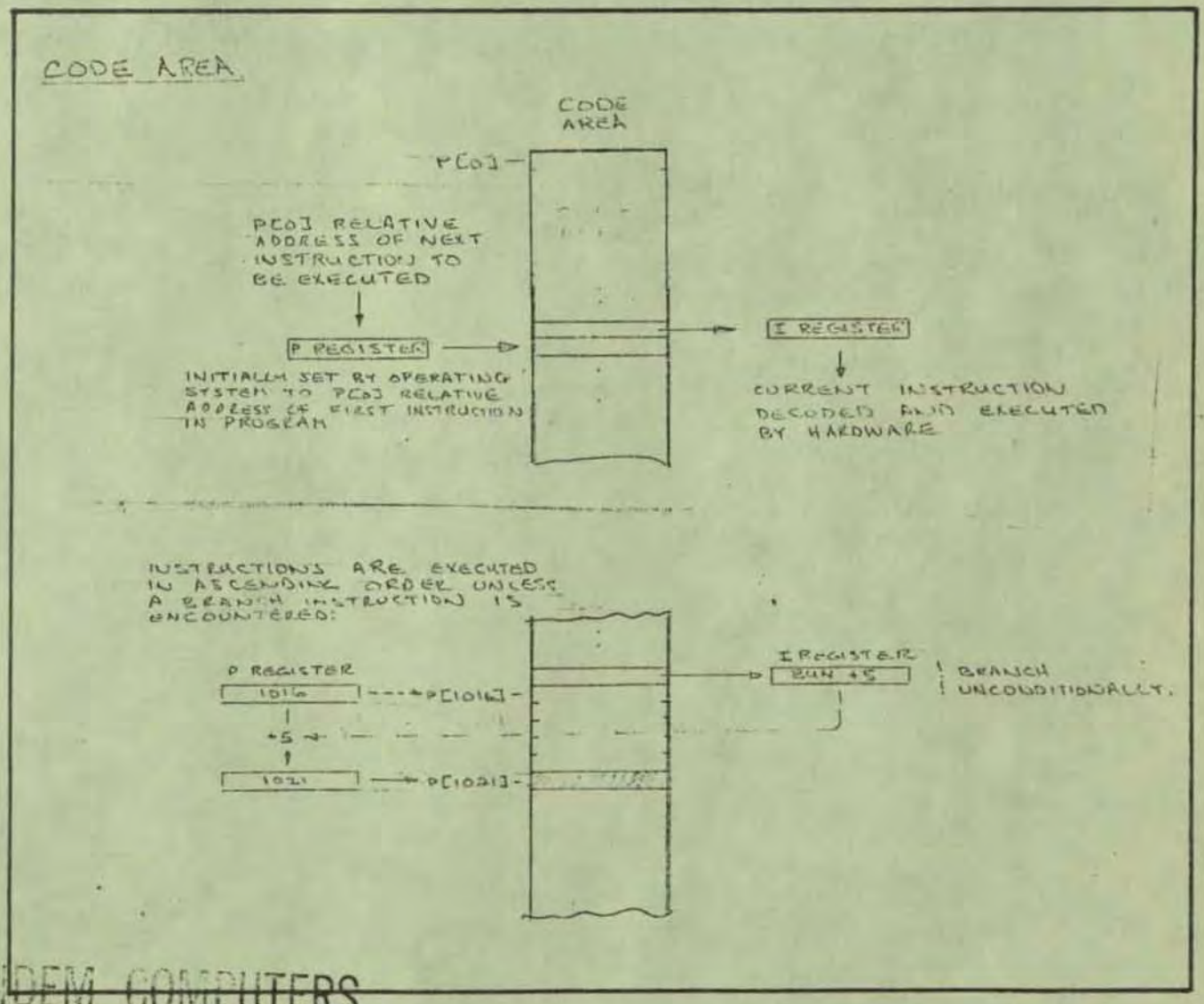
The register stack consists of eight 16-bit registers: the registers are defined as R[0] (register stack, element zero) through R[7]. Three elements of the register stack double as index register (registers R[5:7]) to provide element indexing within the data code areas.

Code Area

The code area contains instruction code and program constants. Two registers are associated with the Code Area:

P Register: The P (for program) register is the program counter. It contains the 16-bit P[0] relative address of the current instruction plus one. P is incremented by one for each instruction executed so that instructions are fetched (and executed) from ascending memory locations. To alter the sequence of program execution, the P Register is given a new setting when a branch is taken, a procedure or subprocedure is called, or an interrupt occurs.

I Register: The I (for instruction) register contains the machine instruction currently being executed. This 16-bit register is always filled with the instruction in the code area pointed to by the P Register.



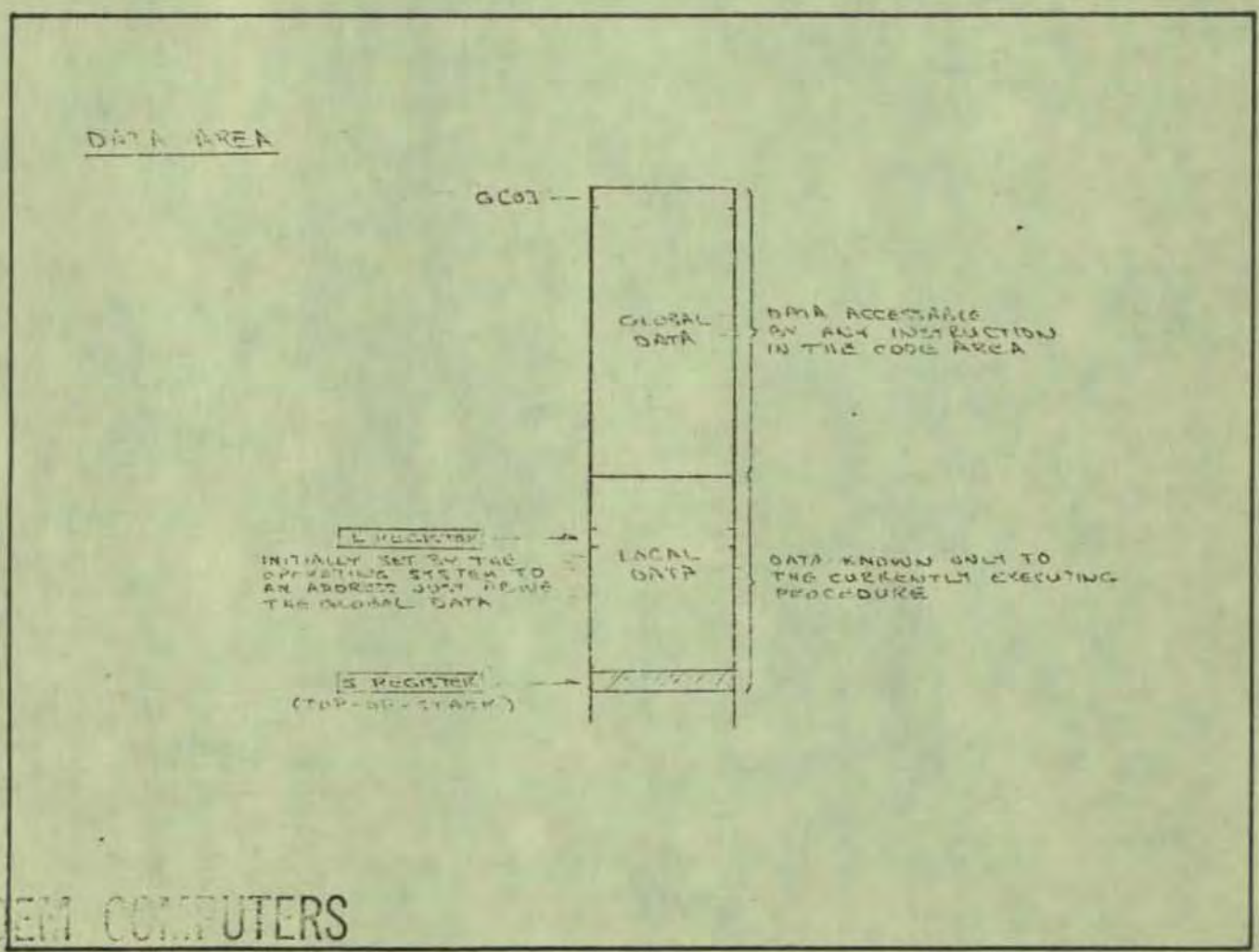
Data Area

The data area contains a program's variable data. The data area is logically separated into three areas (two of which are defined by registers):

GLobal Area: Data within the global area is directly accessable by any instruction in the program. The first word in the global area is defined as G[0].

Local Area: Data within the local area is known only (i.e., direclty accessable) to the currently executing procedure. The local area is defined by the 16-bit L Register. The L (for local) register contains the G[0] relative address of the current stack marker and the beginning of this area.

Top-of-stack (or sublocal) area: Data in the top-of-stack area is known only to the currently executing procedure. The top-of-stack location is defined by the 16-bit S Register. The S (for stack) register contains the G[0] relative of the top-of-stack element.



Addressing: Instructions that reference memory locations contain a 9-bit field for directly addressing the three logical areas that comprise the data area. Five addressing modes are provided to directly access these areas. They are:

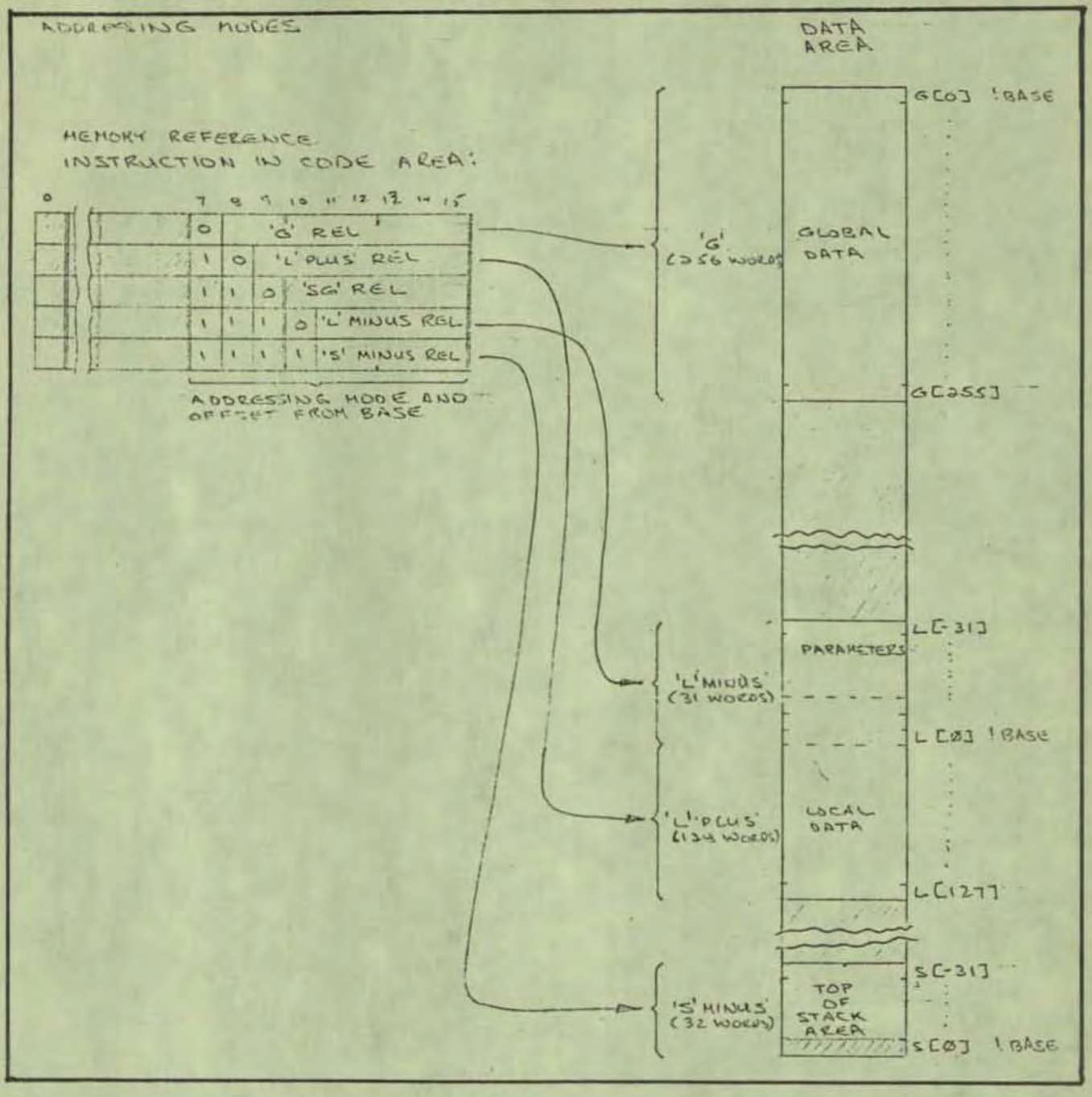
* 'G' Relative Mode: This mode accesses the first 256 locations in the global area (G[0:255]). Bits I.<8:15> are a positive offset from G[0].

* 'L' Plus Relative: This mode accesses the first 128 words of a procedure's local data area (L[0:127]). Bits I.<9:15> are a positive offset from the current L Register setting

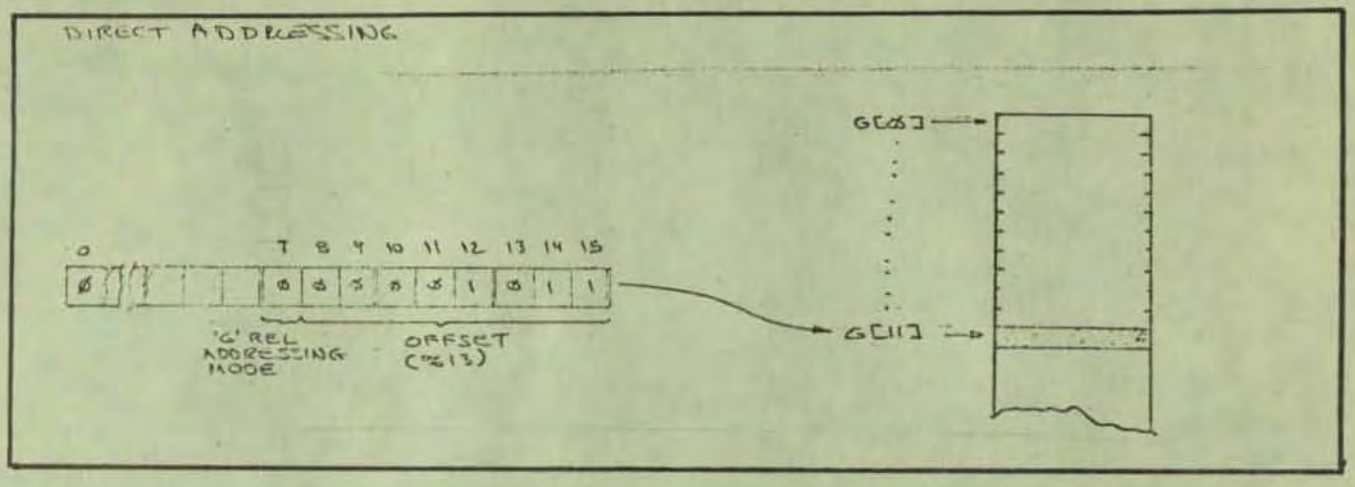
* 'SG' Relative Mode: This mode is usable only by programs running in privileged mode. It provides direct access to the first 64 locations of the operating system's data area (SG[0:63]). Bits I.<10:15> are a positive offset from SG[0].

* 'L' Minus Relative: This mode accesses the data just below the local data area. This area is used to for passing parameters to the procedure having the current L Register setting. Bits I.<11:15> are a negative offset from the current L register setting and provide access to the 31 words in the stack below the local area (L[-31:0]).

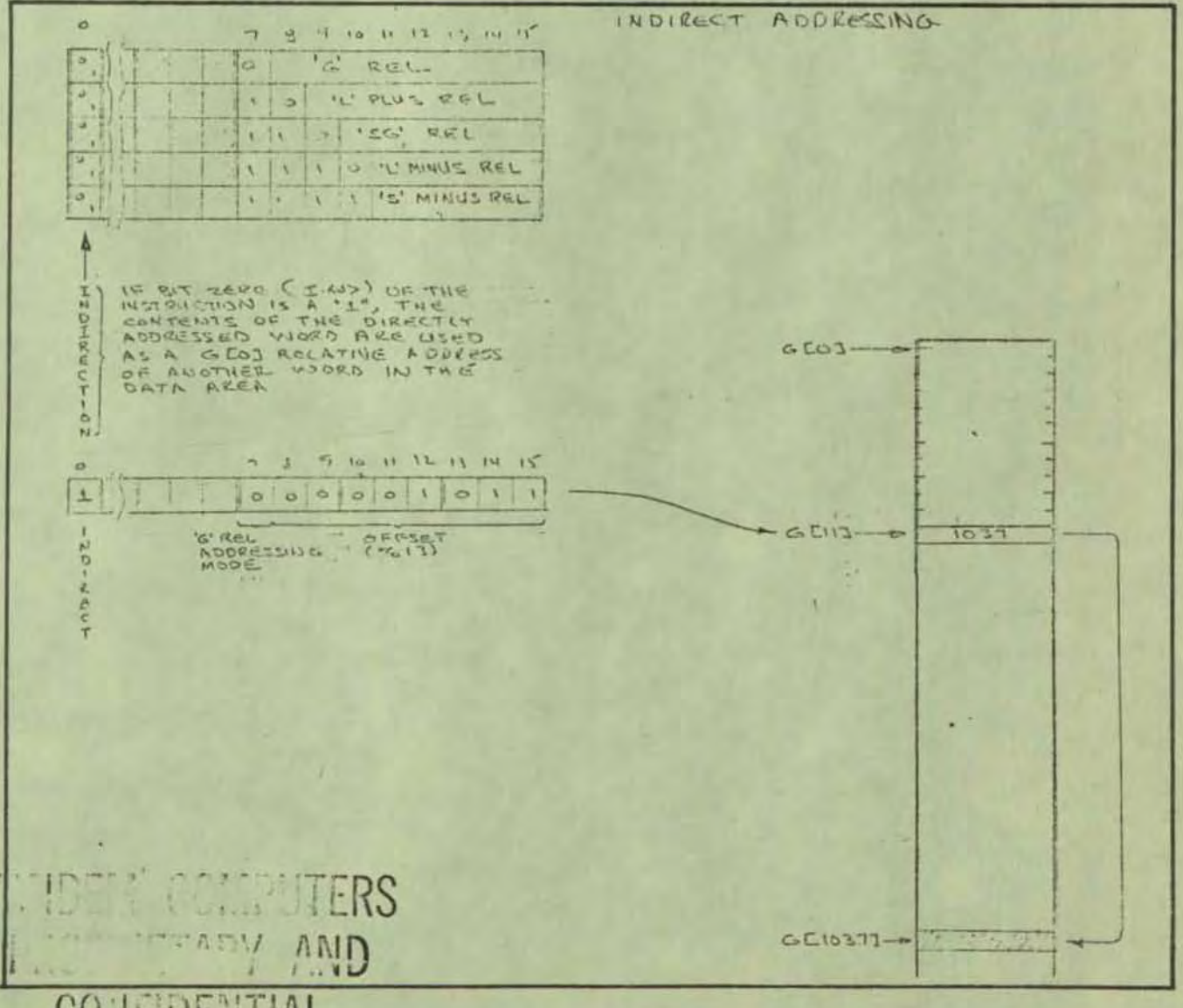
* 'S' Minus Relative: This mode accesses the data in the top-of-stack area. This area is used for a subprocedure's sublocal data and for temporary storage of the register stack contents by the PUSH and POP instructions. Bits I.<11:15> are a negative offset from the current S register setting and provide access to the 31 words in the top-of-stack area (S[-31:0]).



Any directly addressable location can be used as an address pointer (indirect reference) to another memory location. Bit I.<0> specifies whether a memory location is to be accessed directly or used as an address. If I.<0> is a "0" then I.<7:15> specify the actual location (offset from the appropriate base register) of the word to be accessed.



If I.<0> is a "1" then the directly addressed word is an address pointer and is used as the G[0] relative address of the actual memory location wanted. This means that, through indirection, any location in the data area can be accessed (G[0:65,535]).



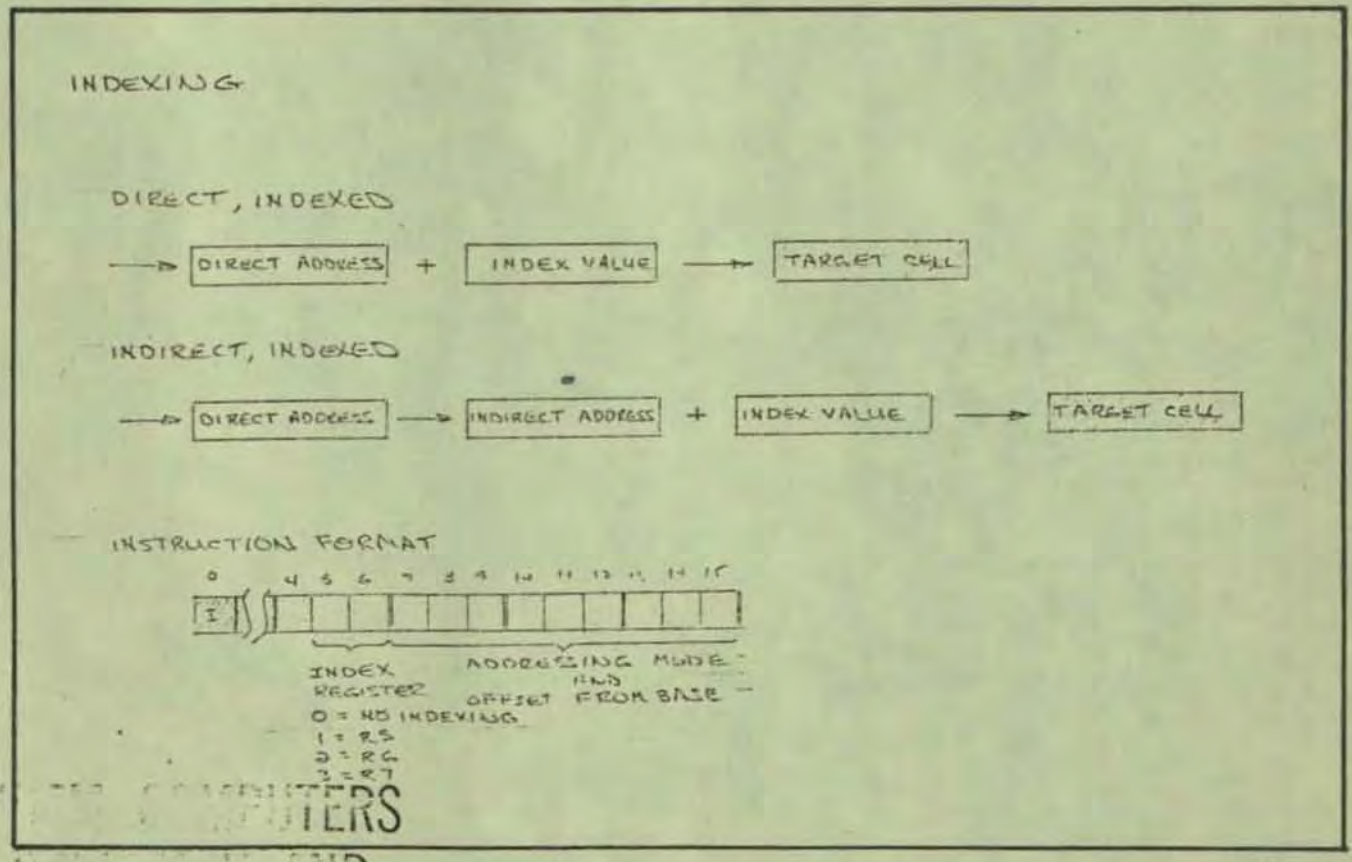
Indexing: Three registers in the register stack (R[5:7]) can be used for indexing. Each instruction that references the data area contains a two-bit field for specifying the index register to be used. The field corresponds to register stack elements as follows:

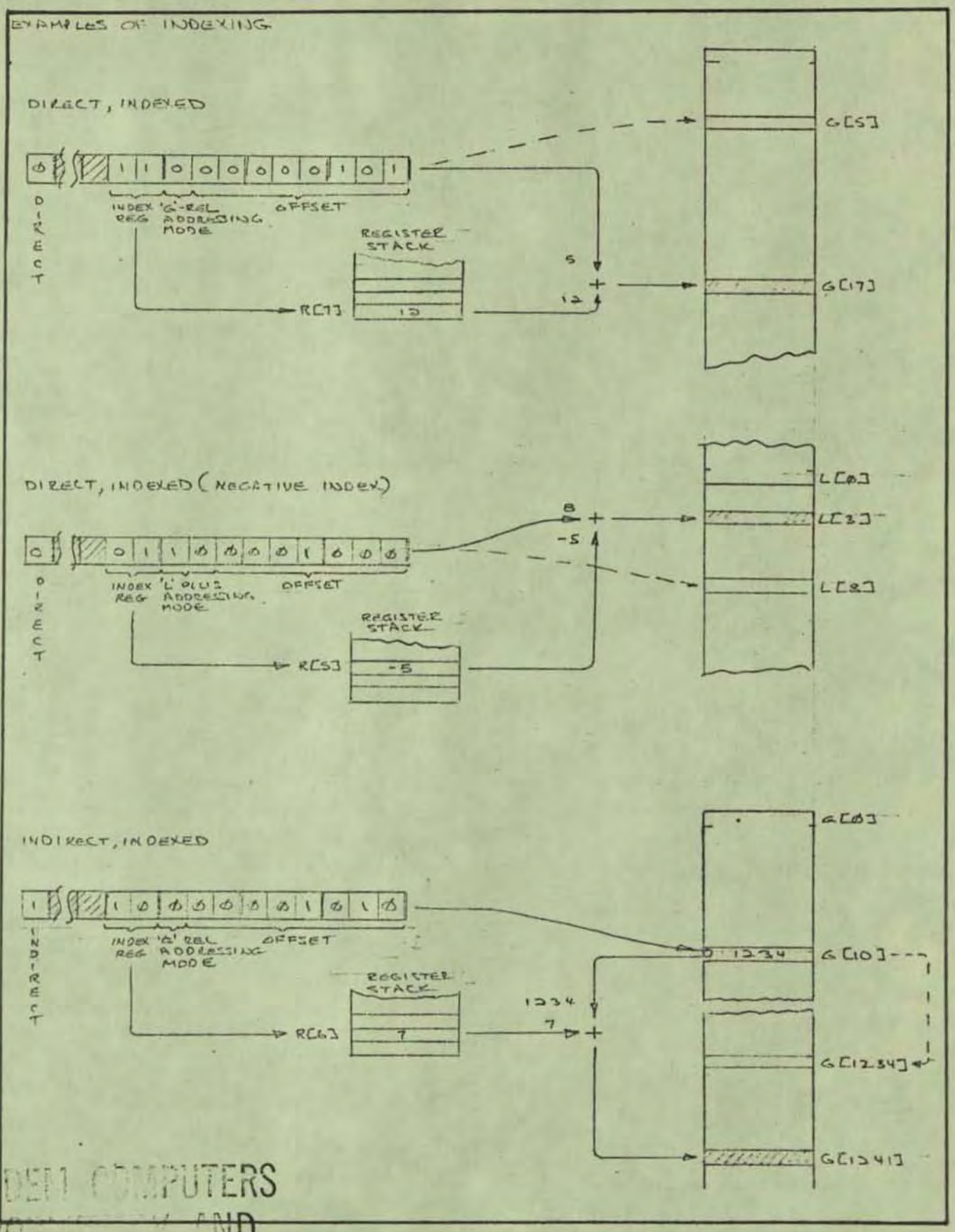
I.<5:6> VALUE	INDEX REGISTER
0	no indexing
1	R[5]
2	R[6]
3	R[7]

When indexing is specified, the address is calculated in the normal manner (i.e., direct or indirect). Then the value contained in the index register is added to the address to provide the address of the target cell. An index register can contain values from zero though 65,535 to provide DIRECT addressing to any location in the data area.

A number of instructions in the instruction set deal with indexing and index registers. They are:

ADRA:	Add Register to A	ADXI:	Add to Index Immediate
SBRA:	Subtract Register from A	LDXI:	Load Index Immediate
ADAR:	Add A to a Register	LDX:	Load Index
SBAR:	Subtract A from a Register	BOX:	Branch on Index





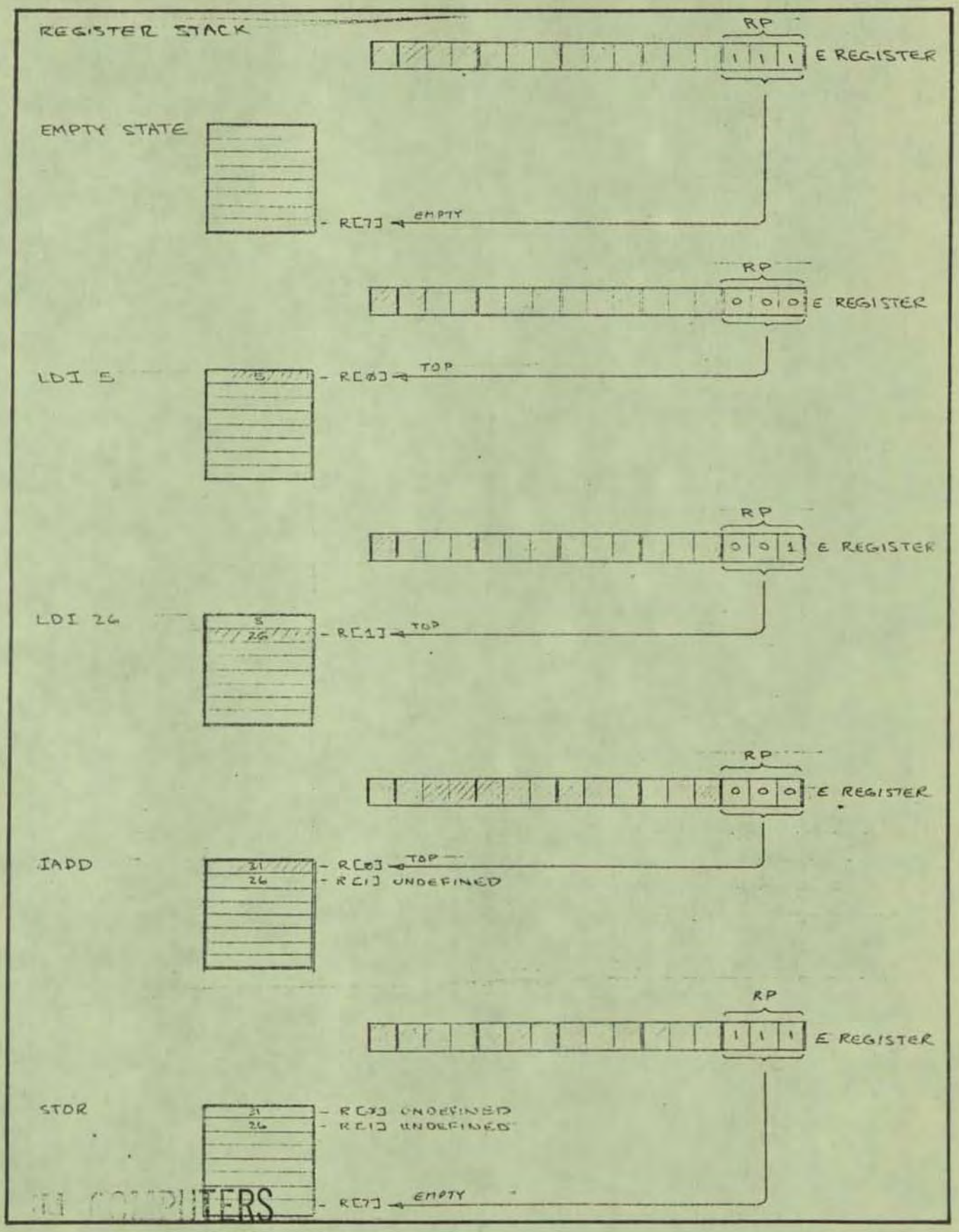
Register Stack

The register stack is where arithmetic computations are performed, and (except for COMW and COMB) where comparisons are made. The register stack consists of eight 16-bit registers (designated R[0:7]) and the register stack pointer (RP). The register stack pointer is a three-bit field in the E (environment) register. RP is the register number of the top element in the register stack.

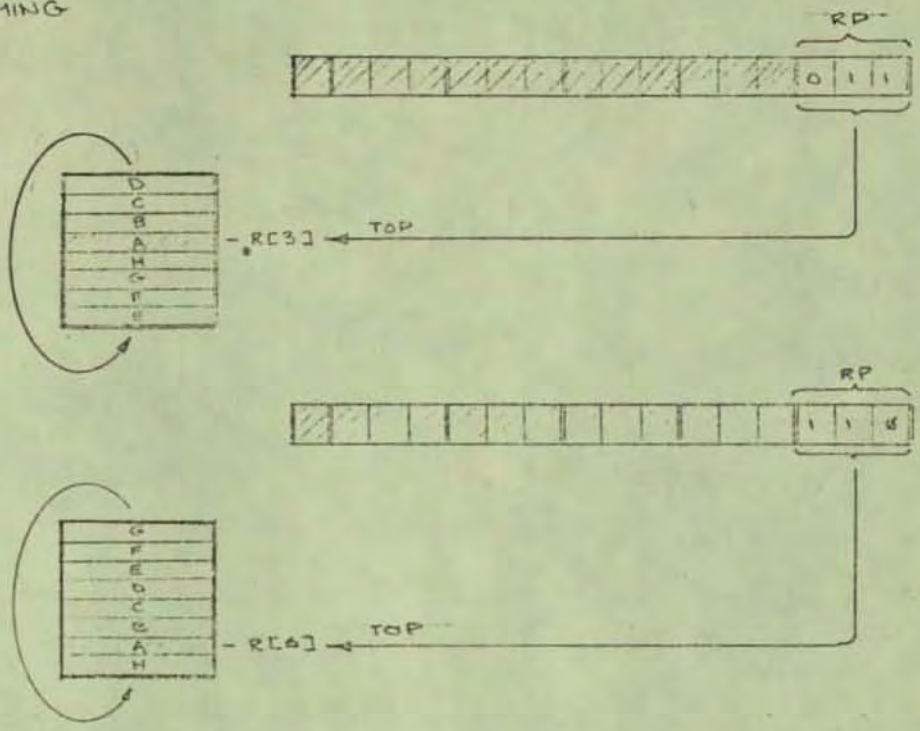
Usually, elements in the stack are addressed implicitly. That is, an instruction operates on the top element (or elements) without being aware of the actual register(s) involved. The RP setting is incremented when operands are loaded into the register stack, decremented when arithmetic is performed or results are stored.

The empty state of the register stack is defined as RP = 7. The full state is also RP = 7. There is no protection against rolling RP over from 7 to 0.

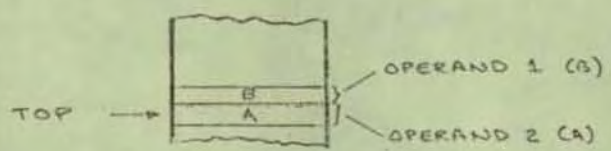
The elements in the register stack are named as to their relative location with the current top element. The top element is designated "A", the second from the top "B", and so on through "H".



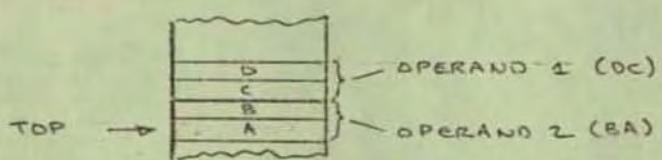
REGISTER NAMING



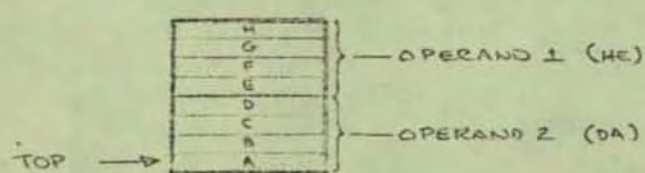
16-BIT OPERANDS



32-BIT OPERANDS

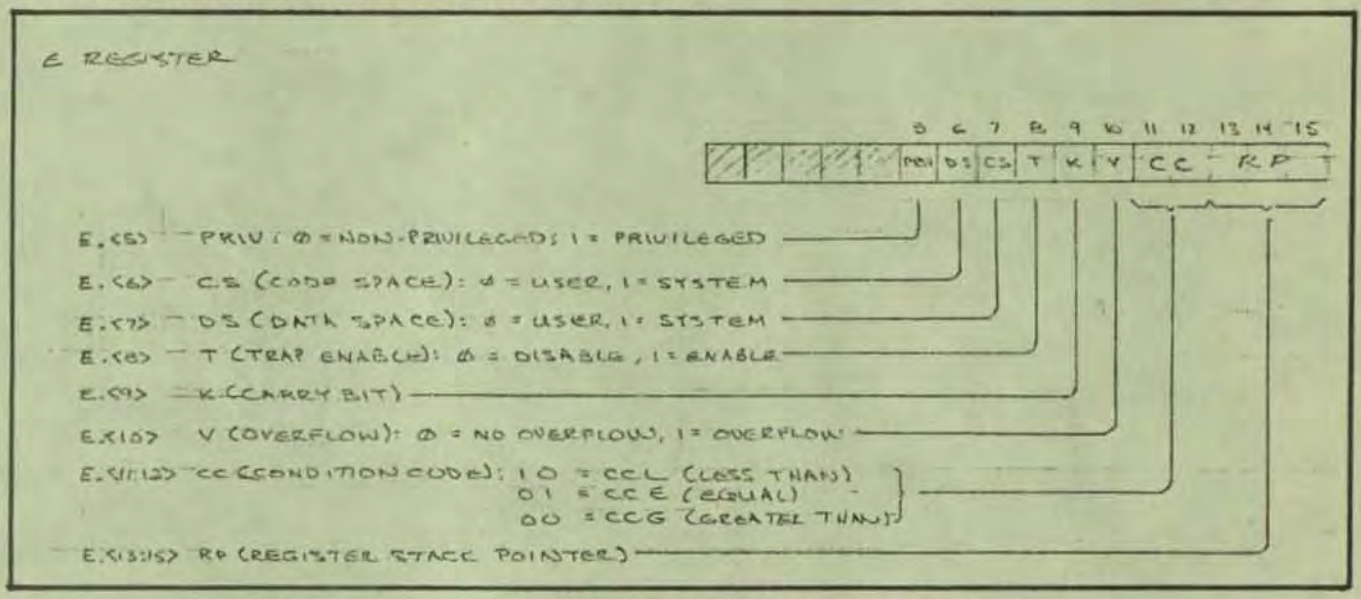


64-BIT OPERANDS



Environment Register

The 16-bit E (for environment) Register maintains the CPU state of the currently executing program. The individual bits and bit fields of the E-Register are continually referenced and updated by the CPU hardware. The E Register contents are used (along with the contents of the P and L registers) by the hardware to save the executing state of a program when a procedure is invoked or when an interrupt occurs. The E-Register is automatically restored to its previous state when the procedure or interrupt finishes.



The bits in the E Register have the following meanings:

E.<5> = Privileged Mode (PRIV): The PRIV bit when a "1" means that the program is currently executing in the privileged mode. A number of operations that have the potential for adversly affecting the processor module if misued are designated privileged operations and can be executed only by programs executing in privileged mode. Some examples of privileged operations are: sending data over an interprocessor bus (SEND), initiating input/output operations, calling privileged procedures, accessing system tables. Normally, only the operating system executes in privileged mode: privileged operations are performed on behalf of application programs by the operating system.

Non-privileged programs can become privileged only by calling procedures designated as callable. (Callable procedures execute in privileged mode, but can be called by non-privileged procedures.) If a non-privileged procedure calls a callable procedure, the non-privileged state is restored when returning to the non-privileged procedure. In general, those instructions designated privileged can be executed only if the PRIV bit in the E Register is a "1". (Exectuting a privileged instruction or calling a privileged procedure when PRIV is a "0" results in a trap to an operating system trap handler).

The PRIV bit is set to a "1" as a result of any the following conditions:

* Cold load or reset

* An interrupt occurring

* Calling a procedure designated "Callable"

E.<6> = Data Space (DS): This bit specifies the data area to be accessed when a data reference is made. DS when "0" specifies the user data area; "1" specifies the system data area. (Programs executing in privileged mode can make explicit system data references regardless of the state of the DS bit.)

The DS bit is set to a "1" as a result of either of the following events occurring:

* Cold Load

* An interrupt occurring

E.<7> = Code Space (CS): This bit specifies the code area to be accessed when an instruction or code area constant is fetched. CS when "0" specifies the user code area; "1" specifies the system code area.

The CS bit is set to a "1" as a result of any of the following events occurring:

* Cold Load

* An interrupt occurring

* Invoking a System Procedure (SCAL)

E.<8> = Trap Enable (T): The T bit specifies whether or not a trap (see Interrupts) is to occur if an overflow occurs or a divide with a divisor of zero is attempted. If T is a "1" and V (E.<07>) becomes a "1", an interrupt to the overflow interrupt handler in the operating system occurs (see the Tandem 16 Programming Manual for possible recovery procedures).

Generally, the T bit is under control of the operating system. However, application programs can control T if it is desired to handle overflow conditions locally. The SETE instruction is used.

E.<9> = Carry (K): The K bit when "1" indicates that a carry out of the high order bit position occurred when executing an arithmetic instruction on a 16- or 32-bit operand. The state of the K bit reflects the last arithmetic type instruction executed. The state of the K bit is also altered as the result of executing a scan instruction (SBW or SBU).

Two instructions are available for branching on the state of the carry bit. They are:

BIC: Branch if carry BNOC: Branch if no carry

E.<10> = Overflow (V): The V bit if a "1" indicates that an overflow condition occurred or a divide with a divisor of zero was attempted. Overflow is generally associated with arithmetic operations on 16- and 32-bit operands: overflow also occurs in a LDIV instruction if the quotient can't be represented in 16 bits.

An instruction (BNOV: Branch if no oveflow) is availble for branching on the state of the overflow bit.

E.<11:12> = Condition Code (CC): This two-bit field forms the condition code. The condition code generally reflects the outcome of a computation, comparison, bus transfer, or input/output operation. (The condition code is also used by the Tandem 16 File Manager to reflect the outcome of operations).

The two bits that form the condition code are designated:

N = negative or numeric (E.<11>) and

Z = zero or alphabetic (E.<12>)

The condition code has three states. They are:

```
CCL = less than    = 10  (N = 1, Z = 0)
CCE = equal to     = 01  (N = 0, Z = 1)
CCG = greater than = 00  (N = 0, Z = 0)
```

More specifically, the condition code is set as follows:

	COMPUTATION	COMPARISON	BYTE TEST	BUS OR I/O
CCL:	operand < 0	opr1 < opr2	ASCII numeric	Error
CCE:	operand = 0	opr1 = opr2	ASCII alpha	Normal
CCG:	operand > 0	opr1 > opr2	ASCII special	Abnormal

For computation, the condition code reflects the value in a data area location, the top of the register stack, or in an index register. The location reflected by the condition code depends on the last instruction executed (see section three for particulars). For example, a simple program to add two numbers then store the result affects the condition code as follows:

```
! memory location G[2] contains the value "5"
! memory location G[3] contains the value "- 5"
LOAD G + 002
   sets the condition code to CCG (5 on the top of the register
   stack)
LOAD G + 003
   sets the condition code to CCL (-5 on the top of the register
   stack)
```

IADD
sets the condition code to CCE (0 on the top of the register stack)
STOR G + 004
does not change the condition code

For comparisons, the condition code is set according to the value of the operands being compared. In the table above, opr1 refers to the first element loaded onto the register stack (i.e., the second element from the top of the stack), opr2 refers to the top element in the register stack. When two arrays are compared by a COMW or COMW instruction, opr1 refers to the element in the destination array, opr2 refers to the source array.

For byte test, the condition code is set according to bits.<8:15> of the operand on the top of the register stack when either a BTST (byte test), LDB (load byte), or a LBP (load byte from program) instruction is executed. A condition code of CCL indicates that an ASCII numerical character (i.e., "0, 1, ..., 9") is on the top of the regsiter stack. CCE indicates an ASCII alphabetical character (i.e., "a, b, ..., z" or "A, B, ..., Z") CCG indicates an ASCII special character (i.e., not numerical and not alphabetical).

For condition code setting result from inteprocessor bus communication see the interprocessor bus description elsewhere in this section and see the description of the SEND instruction in section three. For input/output, see the input/output channel descrition in this section and the EIO, IIO, and HIIO instructions in section three.

A number of instructions are available for branching on condition code values. They are:

BGTR: Branch if CCG
BEQL: Branch if CCE
BGEQ: Branch if CCE or CCG
BNEQ: Branch if CCL or CCG
BLEQ: Branch if CCL or CCE

Three instructions are available for explicitly setting the condition code. They are:

CCL: Set CCL
CCE: Set CCE
CCG: Set CCG

E.<13:15> = Register Stack Pointer (RP): This three-bit field defines the current top element of the register stack. The value of RP is implicitly changed by instructions that operate on values on the top of the register stack. RP is generally incremented as instructions are executed to load operands into the register stack, decremented when computations are performed or results stored.

RP is set to an initial value of seven (indicating an empty register stack) when any of the following events occur:

* An interrupt occurring

* A procedure is invoked

RP is set to 0 because of a cold load.

An instruction, STRP, is available for explicitly setting the RP value.

SETE Instruction: The SETE instruction is used to alter the E register contents. E.<8:15> can be set to any value desired; E.<0:7> are either cleared or left unchanged. This prevents non-privileged programs from becoming privileged and programs from becoming privileged and/or accessing system data. A similar mechanism is used to restore the E Register contents when a procedure finishes.

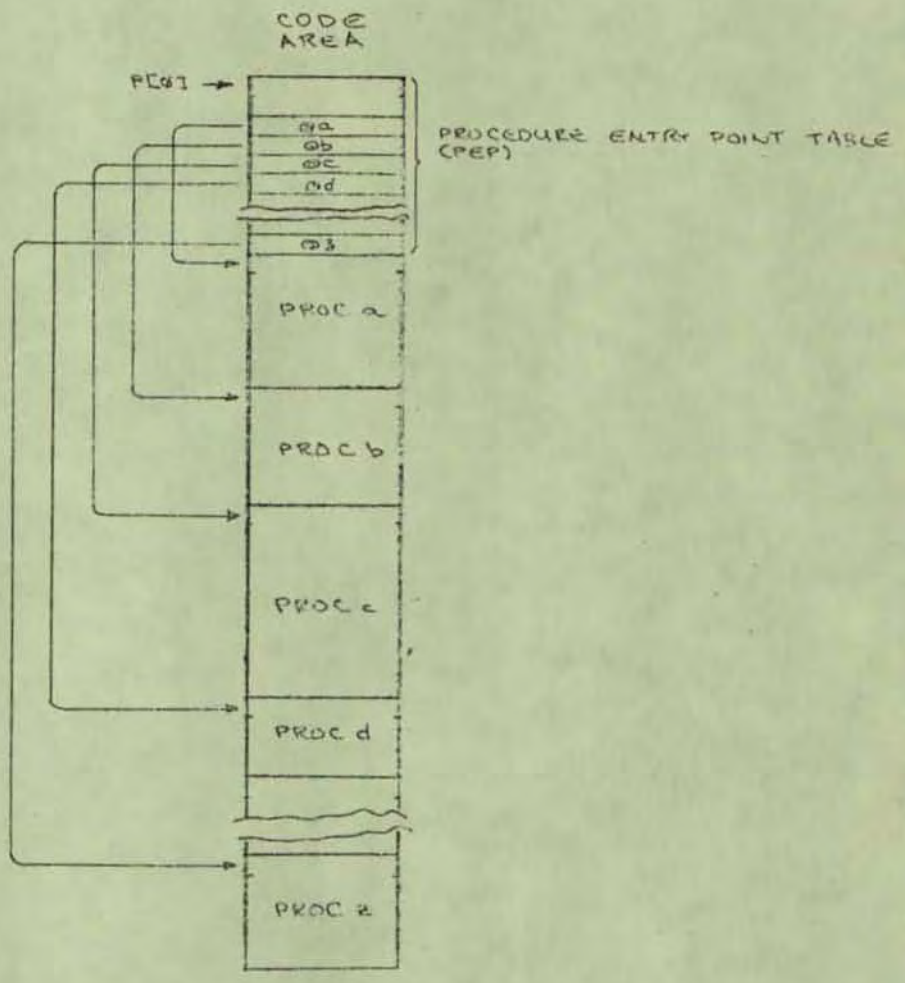
Procedures

When a program is written, the programmer separates the funtion to be performed into executable blocks of instructions called procedures. A procedure can be written to perform an operation as simple as adding two numbers or as complex as locating an entry in a data base. The power of procedures is in the following four items:

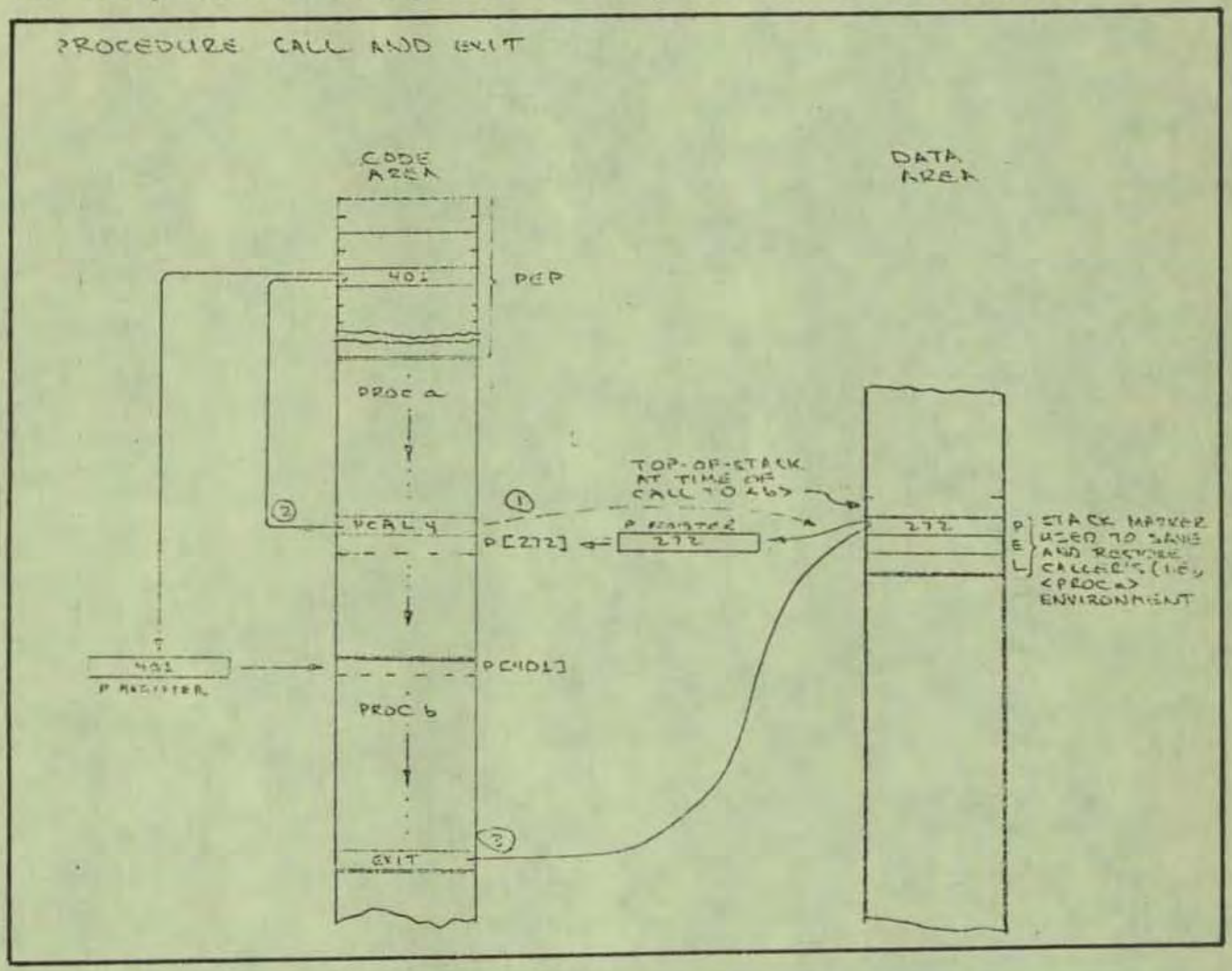
* A procedure can be called into execution (invoked) from any point in a program
* The caller's environment is automatically saved by the hardware when a procedure is called, restored by the hardware when the procedure finishes.
* When a procedure is called into execution it is allocated a storage area called a local data area. The local data area is known only to the executing procedure and is separate from other procedure's local data areas (much like a program's data area is known only to the executing program).
* Parameters (or arguements) can be passed to a procedure for evaluation. The parameters can be actual operands or can be addresses of operands.

The code area is comprised of all the procedures in the program and the procedure entry point table. (The procedure entry point table is a list of addresses of each procedure in the program.)

PROCEDURES



Procedure Call and Exit: Procedures are invoked using the PCAL (procedure call) instruction. The PCAL instruction contains a nine-bit field which is a positive offset from P[0] into the procedure entry point table. The field in the PCAL instruction points to the PEP entry that in turn points to the procedure to be executed.



When a procedure is called, the caller's environment is saved in a three-word stack marker. The stack marker is placed in the data area at the current top-of-stack location.

A procedure returns to the caller by executing an EXIT instruction. The EXIT instruction uses the information saved in the stack marker to restore the caller's environment.

Specifically, the steps involved when a procedure is called into execution are

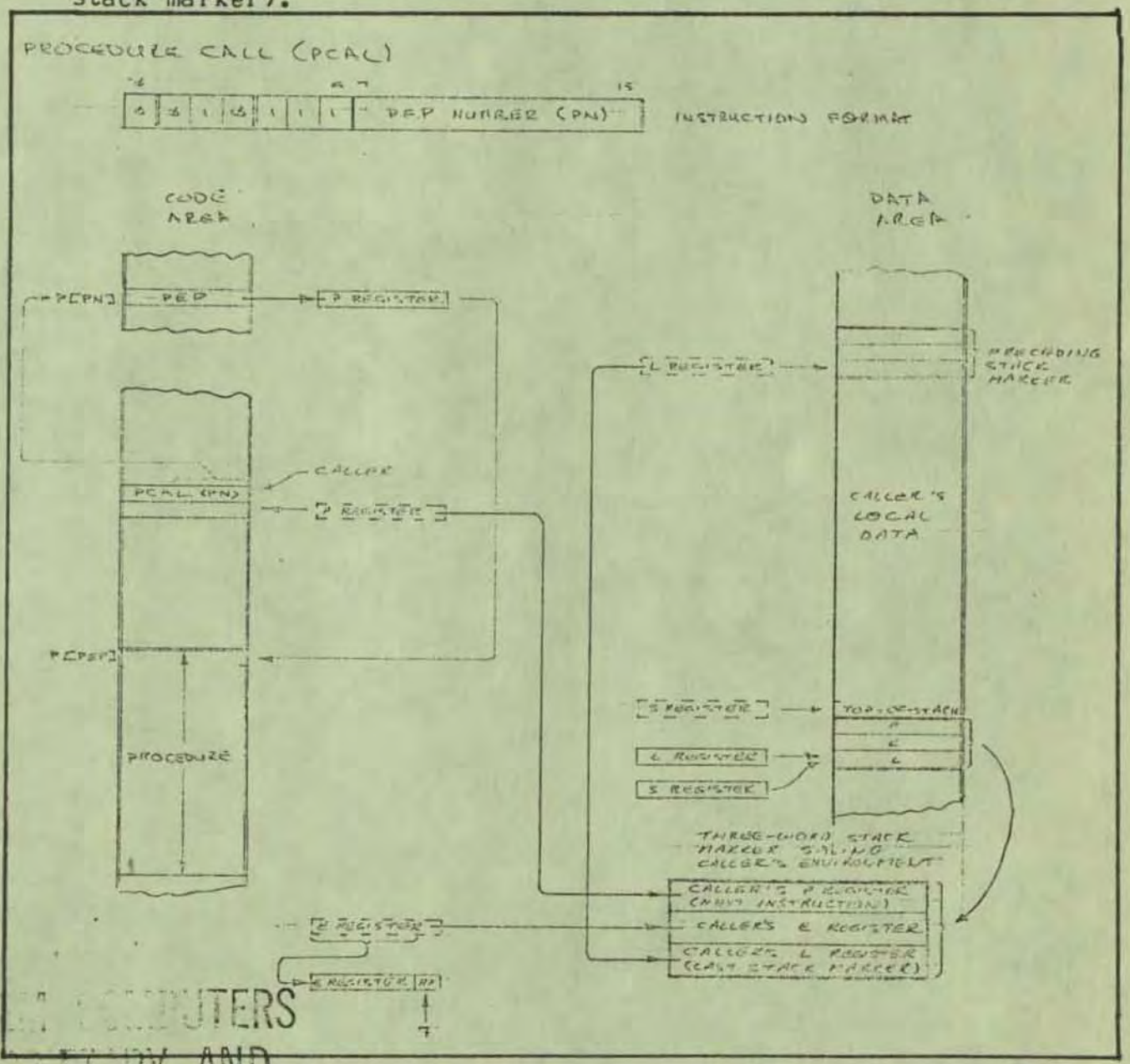
1. The current environment is saved in a three-word stack marker. The first word of the stack marker is stored in the top-of-stack location pointed to by the address in the S Register plus one. The stack marker contains the following information:

   * The current P Register setting (the next instruction after the PCAL)

   * The current E Register setting

   * The current L Register (the beginning of the caller's local data area)

   The S and L Registers are then loaded with the G[0] relative address of the new top-of-stack location (the third word of the stack marker).



2. The P[0] relative address of the procedure to be executed is obtained from the PEP table entry pointed to by the nine-bit field in the PCAL instruction. This address is put in the P (program counter) register so that the next instruction executed will be the first instruction of the called procedure.

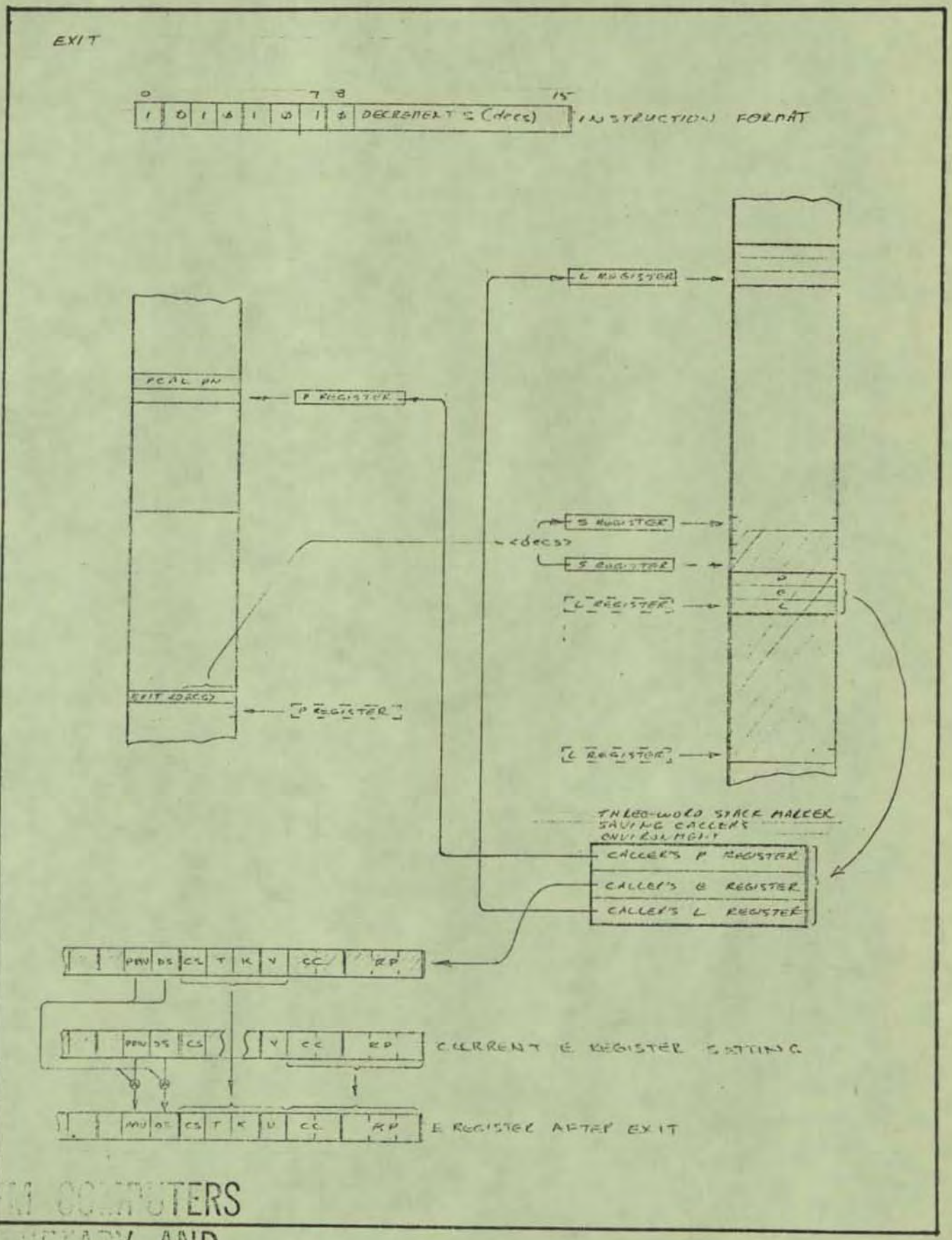
3. The instructions comprising the procedure are executed. The last instruction executed is an EXIT instruction.

4. The EXIT instruction uses the three-word stack marker to restore the caller's environment. Specifically

   * The P Register is set with the P Register value saved in the stack marker at L[-2]. The next instruction executed is the one following the PCAL instruction.

   * The E Register is restored from parts (see drawing) of the E Register value saved in the register stack at L[-1].

   * The L Register is restored from the L Register value saved in the stack marker at L[0]. The L Register now points to the caller's local data area.



5. The instruction following the PCAL instruction executes.

To protect the system from misuse, procedures are classified according to who is elegible to call them. The catagories are:

* Non-privileged — These procedures are callable by any procedure in the program. They execute in the same mode (i.e., PRIV or non-PRIV) as the caller.

* Callable -- These procedures are also callable by any procedure in the program but execute in PRIV mode (i.e., E.<5> = "1"). The callers mode is restored when a callable procedure exits.

* Privileged -- These procedures are callable only by procedures currently executing in PRIV mode. An attempt by a non-privileged procedure to call a privileged procedure results in an intruction failure trap toan operating system trap handler.

The mechanisms provide the needed protection, the first is the procedure entry point table, the other is the way the E register is restored when exiting a procedure.

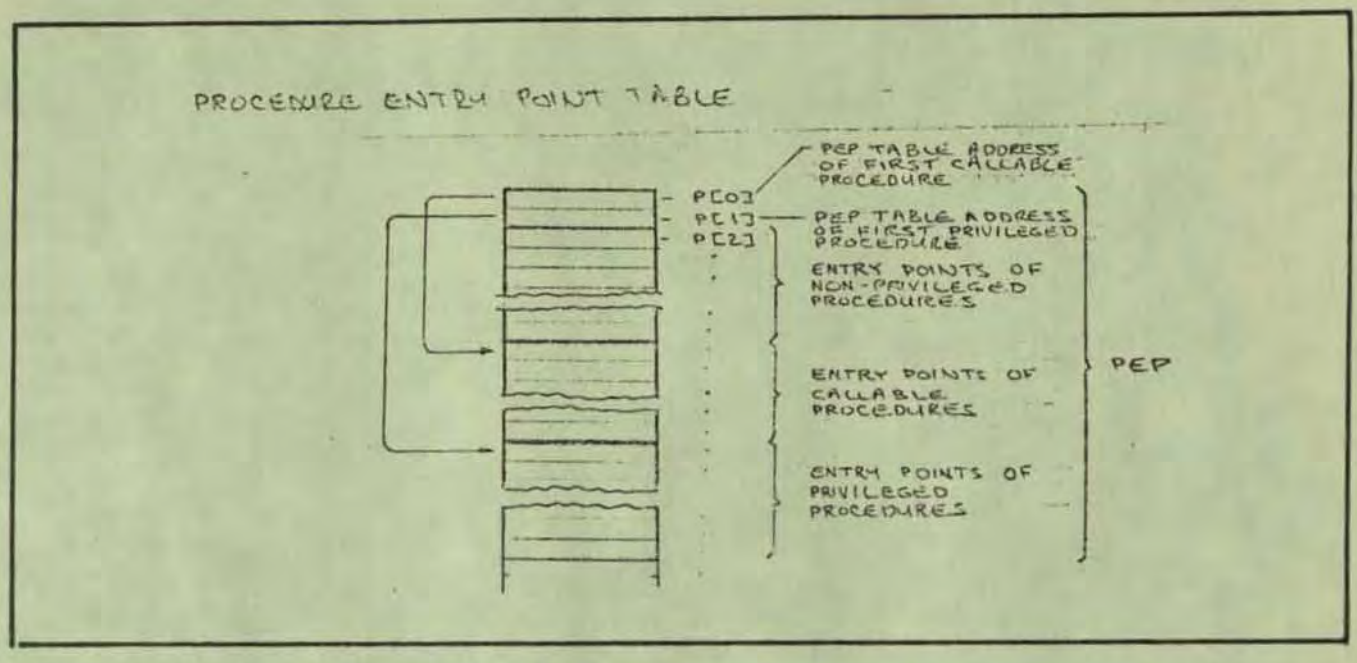
Two mechanisms provide the needed protection, the first is the procedure entry point table, the other is the way the E register is restored when exiting a procedure.

The first two words in the PEP table describe the procedures in the code area. P[0] is the PEP address of the first "callable" procedure entered in the PEP table. P[1] is the PEP address of the first "privileged" procedure entered in the PEP table. Any time a procedure is called, the following check takes place.

* the PCAL PEP address (i.e., I.<7:15>) is compared with P[0]; if the PEP table address is less than the value of P[0] then the called procedure is non-privileged.

* If PEP table address is equal to or greater than P[0], then the called procedure executes in PRIV mode and E.<5> is set to "1".

* Then, if the calling procedure is not executing in PRIV mode, the PEP address is compared with P[1]. If the PEP address is equal to or greater than P[1] an atempt is being made by a non-privileged procedure to call a privileged procedure. If this is the case, an instruction failure trap occurs (and the program is probably aborted).



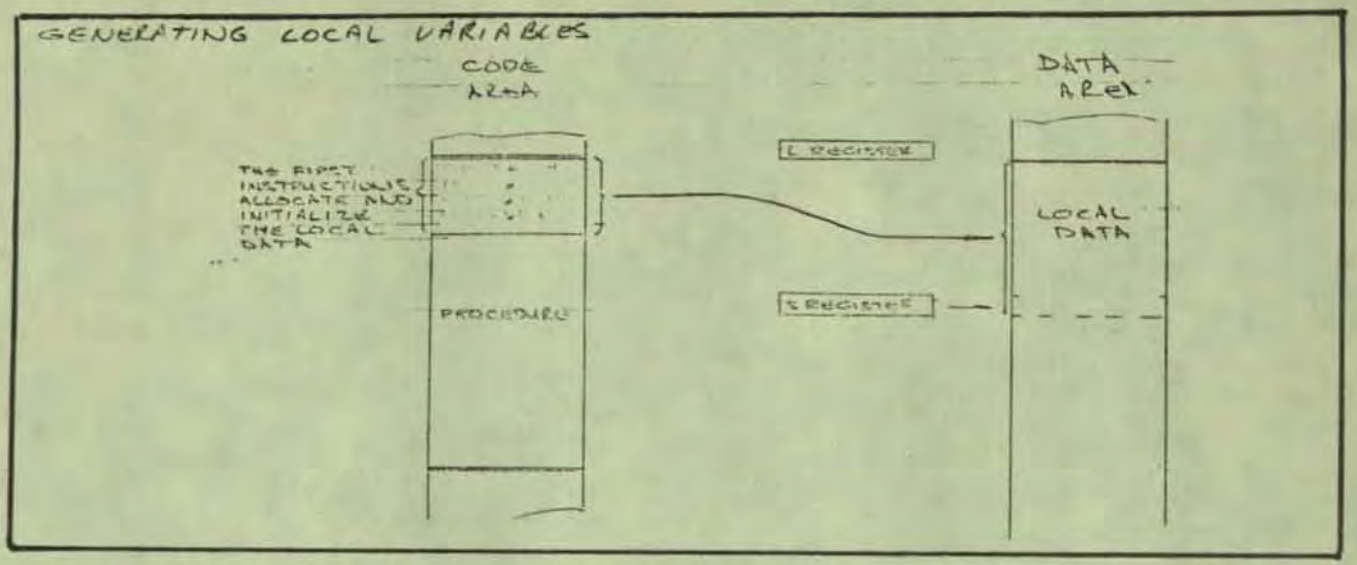
The E Register is restored from its current setting and the value saved in the stack marker as follows:

* The PRIV and DS (data space) bits are restored by ANDing the current E Register settings with the settings in the stack marker. This prevents non-priviled procedures from inavertantly (or intentionally) becoming privileged (or accessing system data) by altering the stack marker.

* The CS (code space), T (traps enable), K (carry), V (overflow) bits are restored from the stack marker. (CS is restored from the stack marker so that procedures in user code can be executed by the operating system by means of a DPCL (dynamic procedure call) instruction.)

* The CC (condition code) and RP (register pointer) are retained from the current setting.

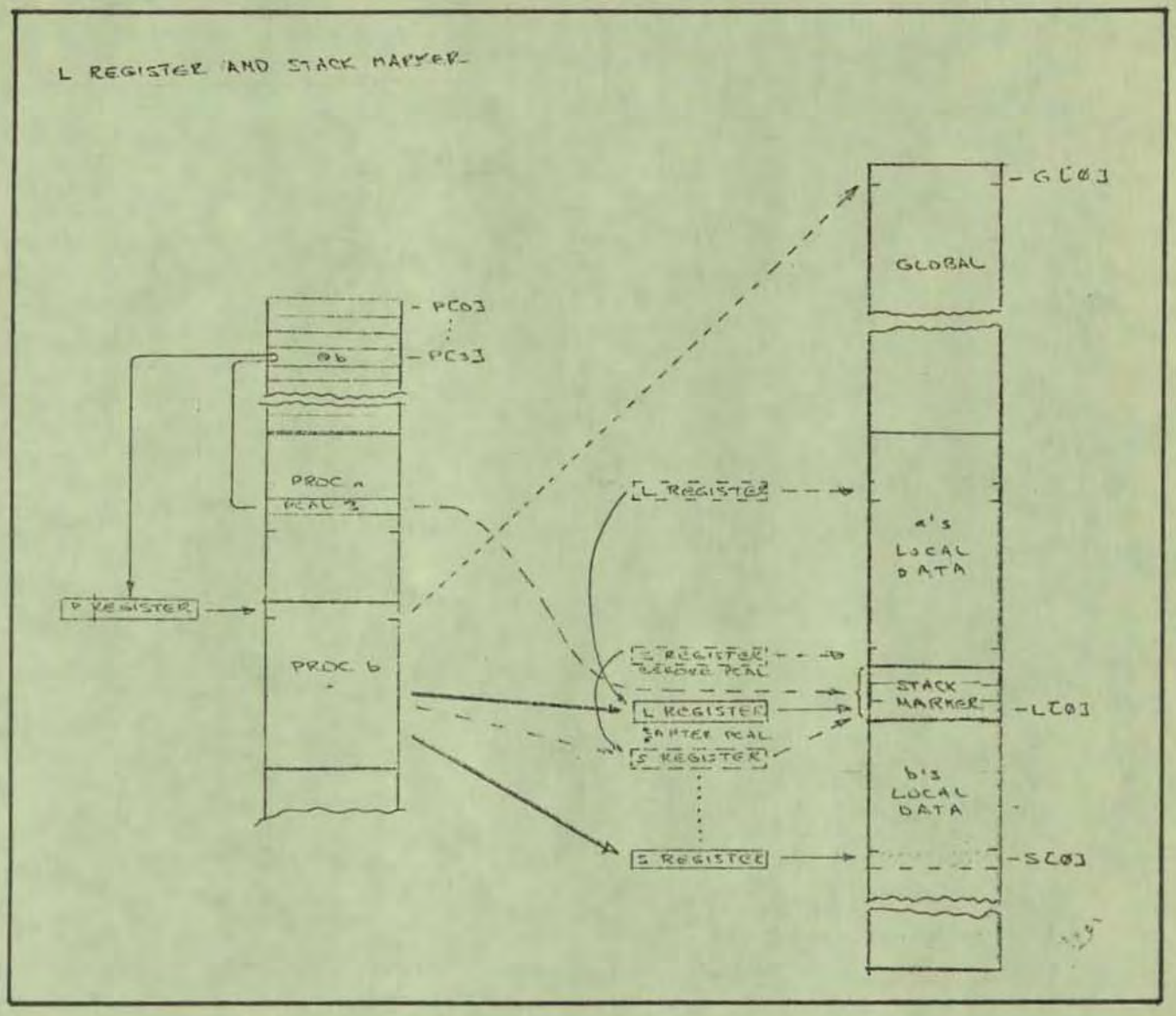
Local Data: Unlike the global data area which exists at all time (and a copy resides in the object file on disc), the local data area for a procedure exists only while the procedure is actually executing. The local variables are generated and initialized by instructions at the start of a procedure's code. Thus a procedure can be called any number of times (and in fact can call itself) and each call generates a fresh copy of the procedue's local data area.



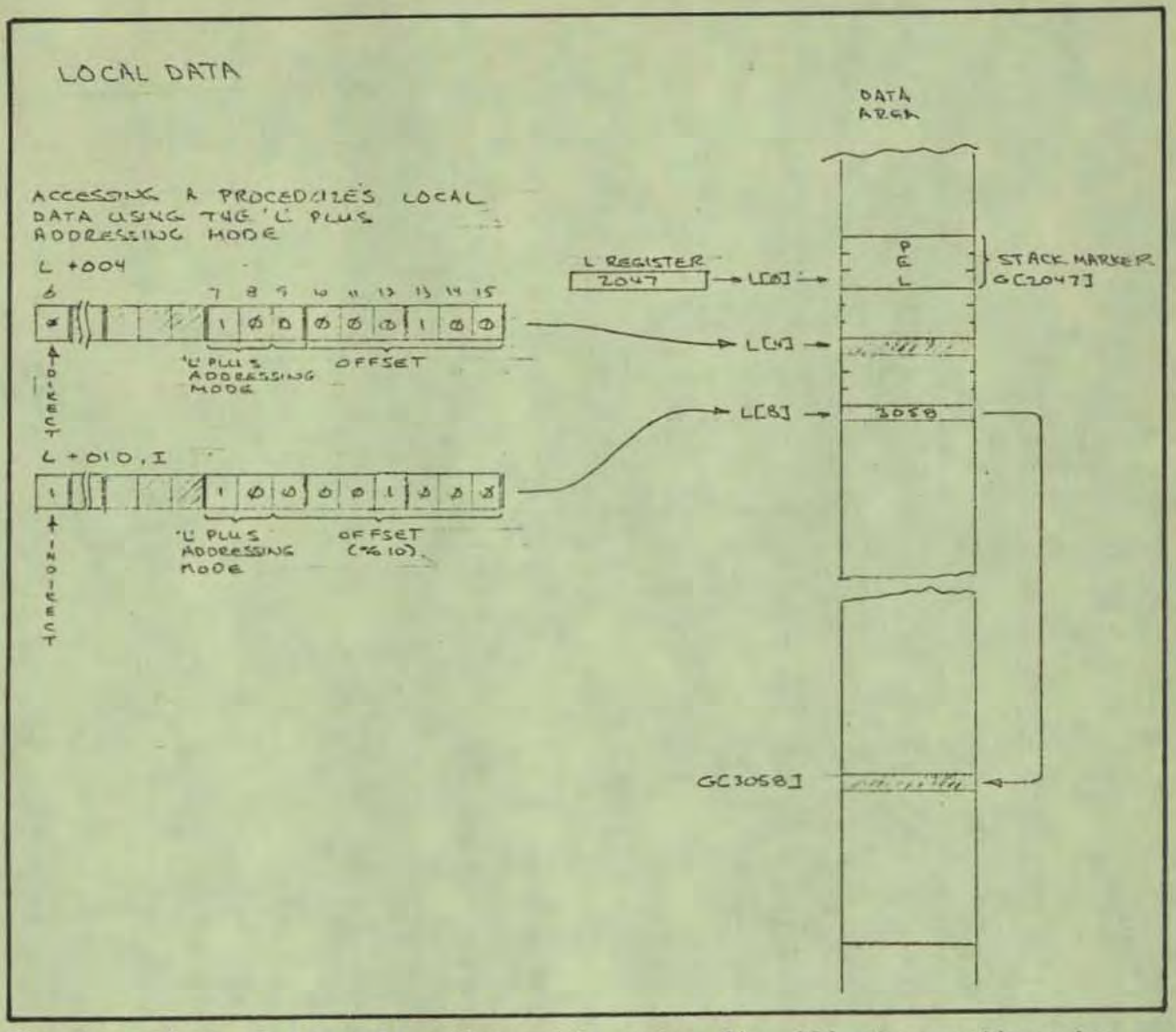
Data in the local area is addressed relative to the current L Register setting using the 'L' plus addressing mode. As shown in the drawing below, when a procedure is called into execution, the L Register is given a new (higher) setting above the preceding procedure's local data area. The L Register and S Register, at the beginning of procedure execution, points to the current top-of-stack location (which is the caller's stack marker). All local addressing is done relative to this point.

The first few instructions in the procedure generate and initialize any local storage needed. After the local storage is generated, the S Register is updated (using an ADDS or SETS instruction) so that it points to the upper limit of the local storage area. (Remember, that when a procedure is called,the stack marker is placed at the current S Register setting plus one. In this manner, a procedure's local data is always retained when it calls another procedure.)

In summary, the way the a stack marker (which saves the caller's enviroment) points to the preceding stack marker, the L Register gets a new setting when a procedure is invoked, and the local data areas are allocated and generated relative to the current L Register setting, results in a memory stack that keeps retains the envorinments of procedures in the order executed. Each time a procedure EXITs, the preceding procedure's environment is restored.



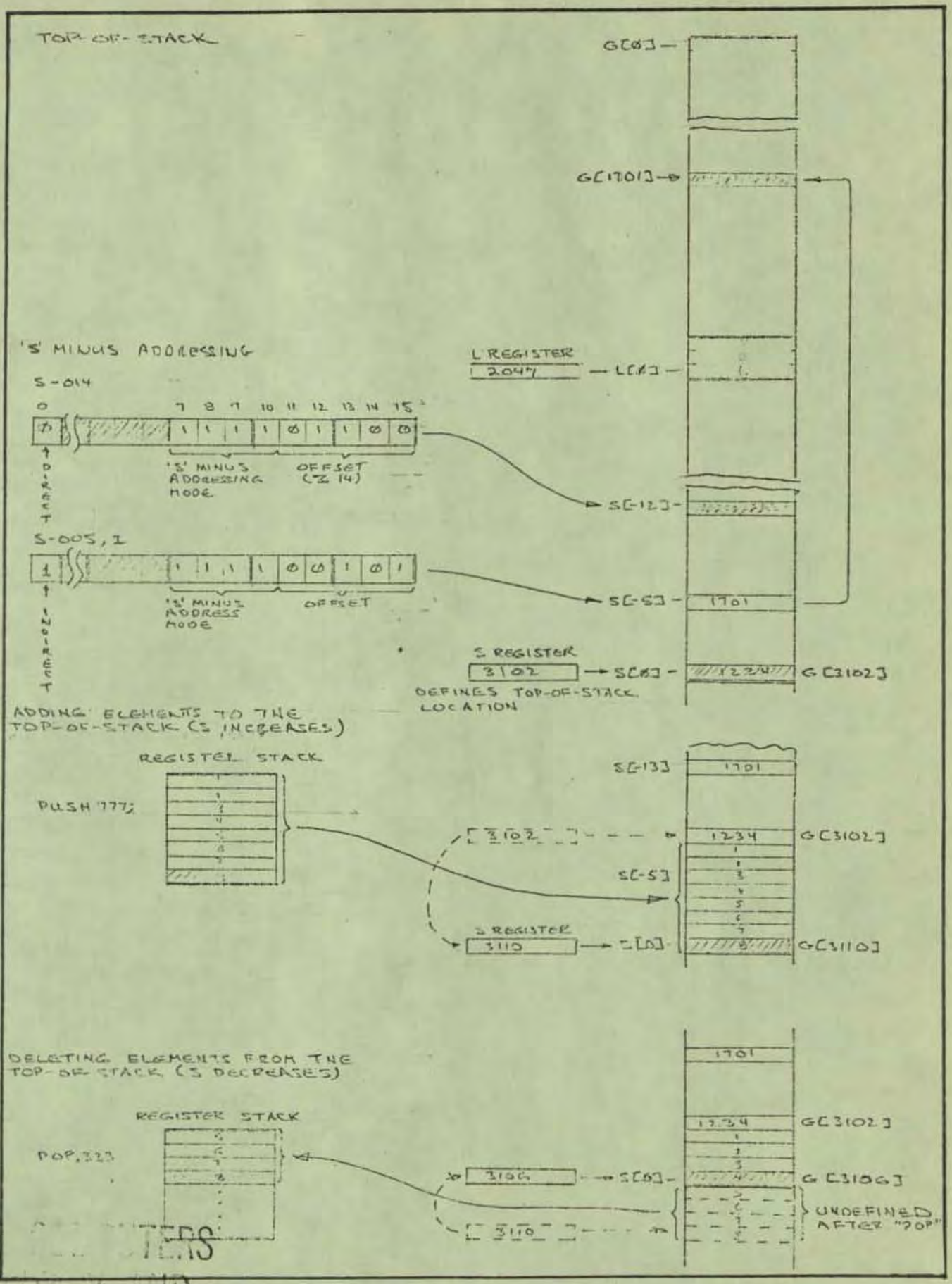
The 'L' plus addressing mode is used to access data in the local data area. As illustrated, this mode can access local data directly, or can use the direct address as an address pointer (indexing is also possible).



The top-of-stack area is addressable using the 'S' minus addressing mode. This provides direct access to the 31 locations below the current S Register setting (i.e., S[-31:0]). 'S' minus addressing is used to access a subprocedures sublocal stoage area (a subprocedure is similar to a procedure but has limited local storage).

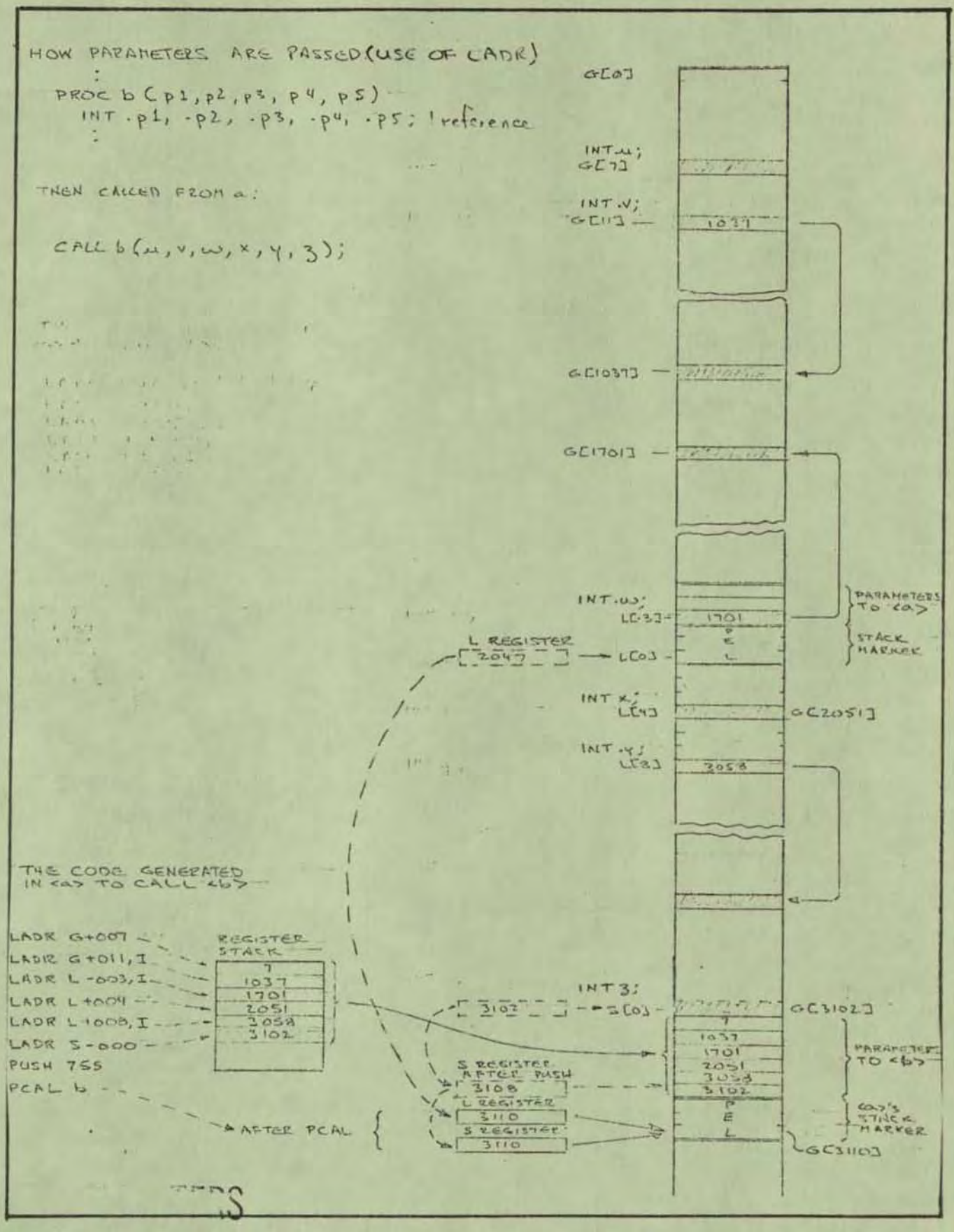
The top-of-stack area can be addressed implicitly using a PUSH instruction. The PUSH instruction is used for temporary storage of the register stack contents, usually prior to calling a procedure. When the PUSH instruction is executed, the S Register setting is incremented by the number of words pushed. The POP instruction is used to restore the register stack, then decrement the S Register setting accordingly. A part of a program to save the register stack, invoke a procedure, then restore the register stack might look like:

```
PUSH ! saves the register stack
PCAL ! calls a procedure
POP  ! restores the register stack.
```

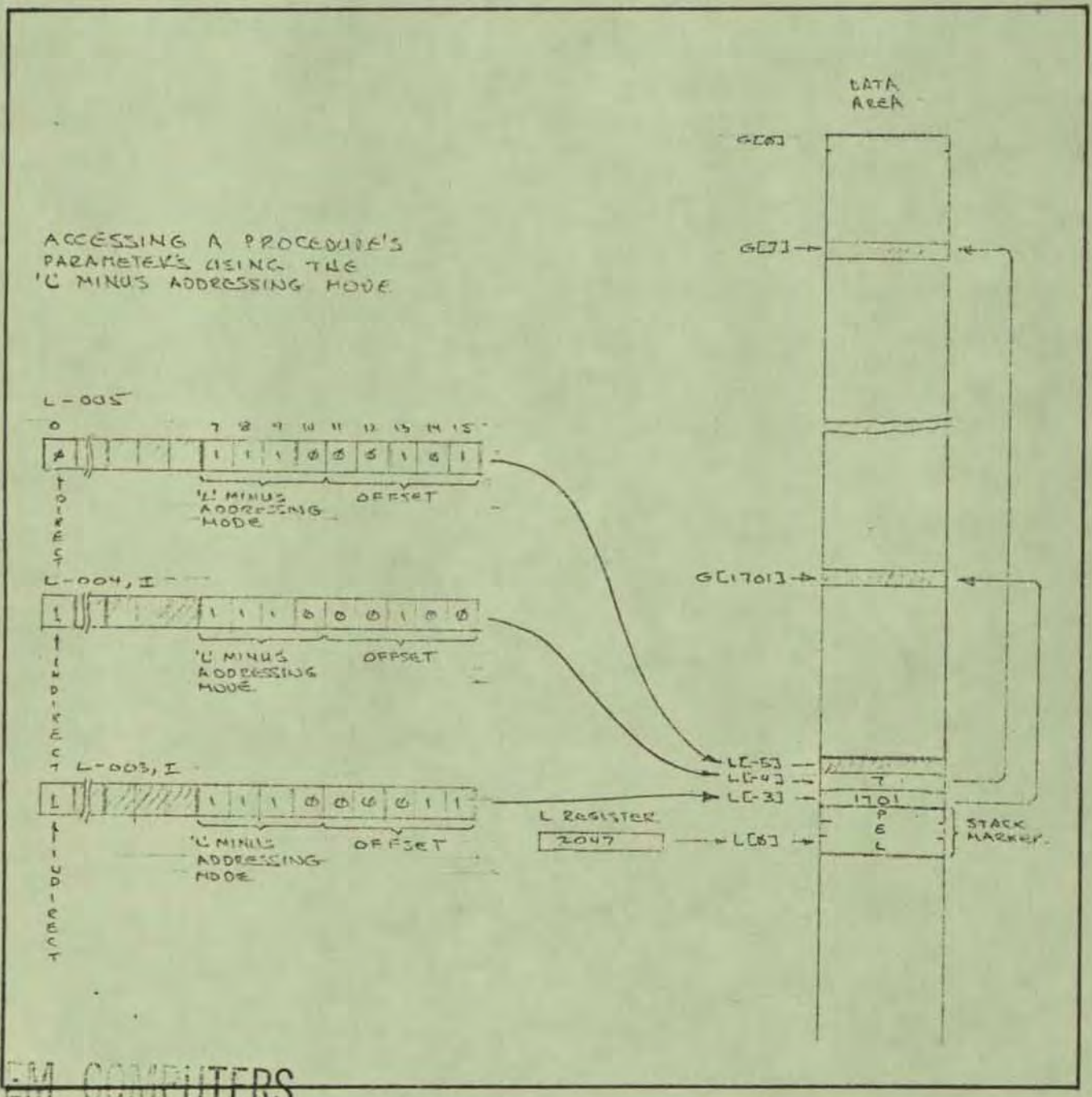


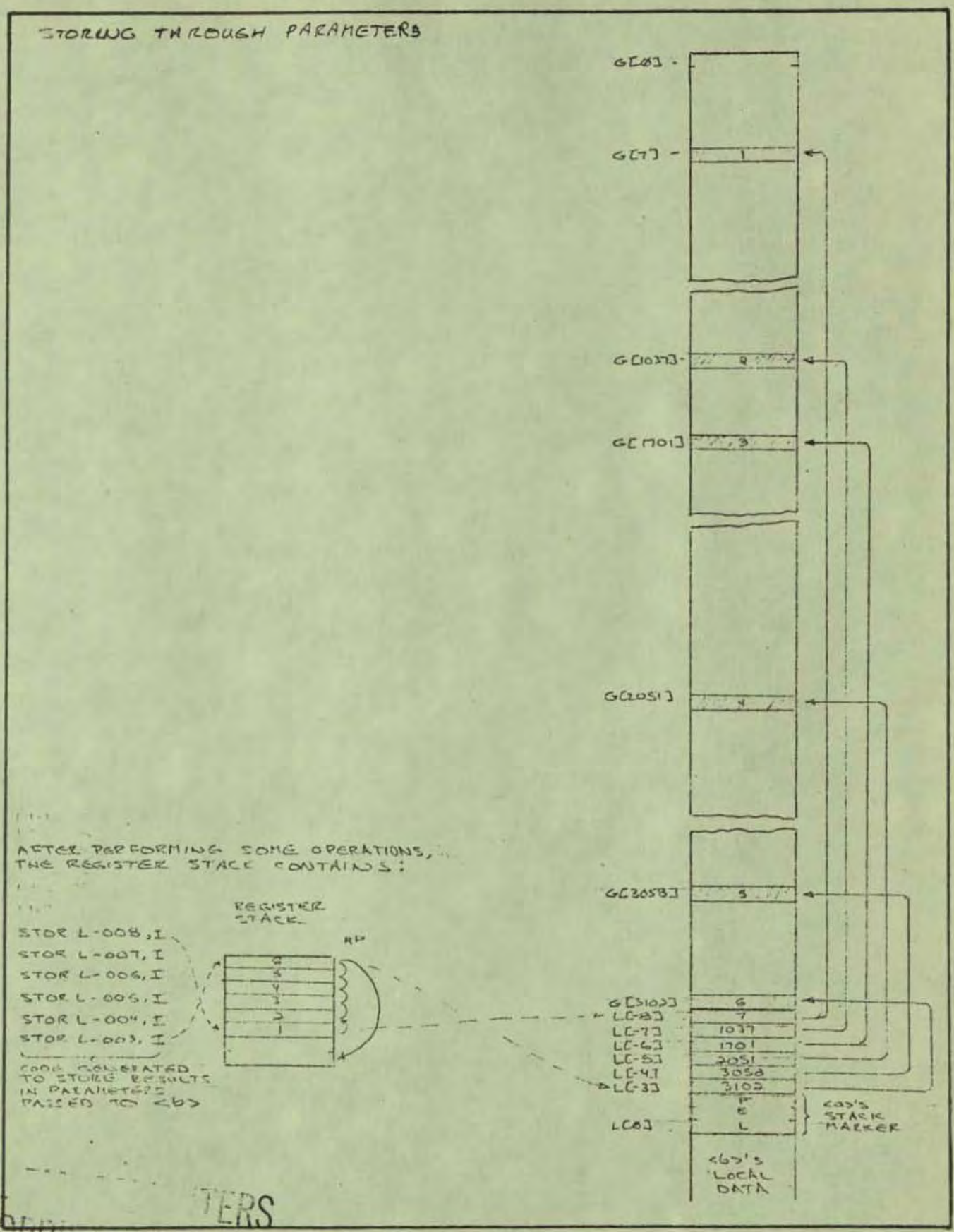
Parameters: Parameters are passed to a procedure in the area just below the caller's stack marker. Naturally, there must be coordination between the caller and the called when passing parameters. The caller must know the order in which a procedure expects parameters, whether a parameter is to be an actual operand (called a "value" parameter) or an address pointer (called a "reference" parameter).

Before the caller invokes a procedure, the parameters are prepared in the register stack. The actual operands (for value parameters) and the addresses of operands (for reference parameters) are loaded into the register stack in the order required by the procedure being called. The address of a reference parameter is obtained by executing an LADR (load address) instruction (the LADR instruction provides the G[0] relative address of a variable). The parameters prepared in the register stack are load on the top of the memory stack by executing a PUSH instruction (which increments the S Register accordingly).

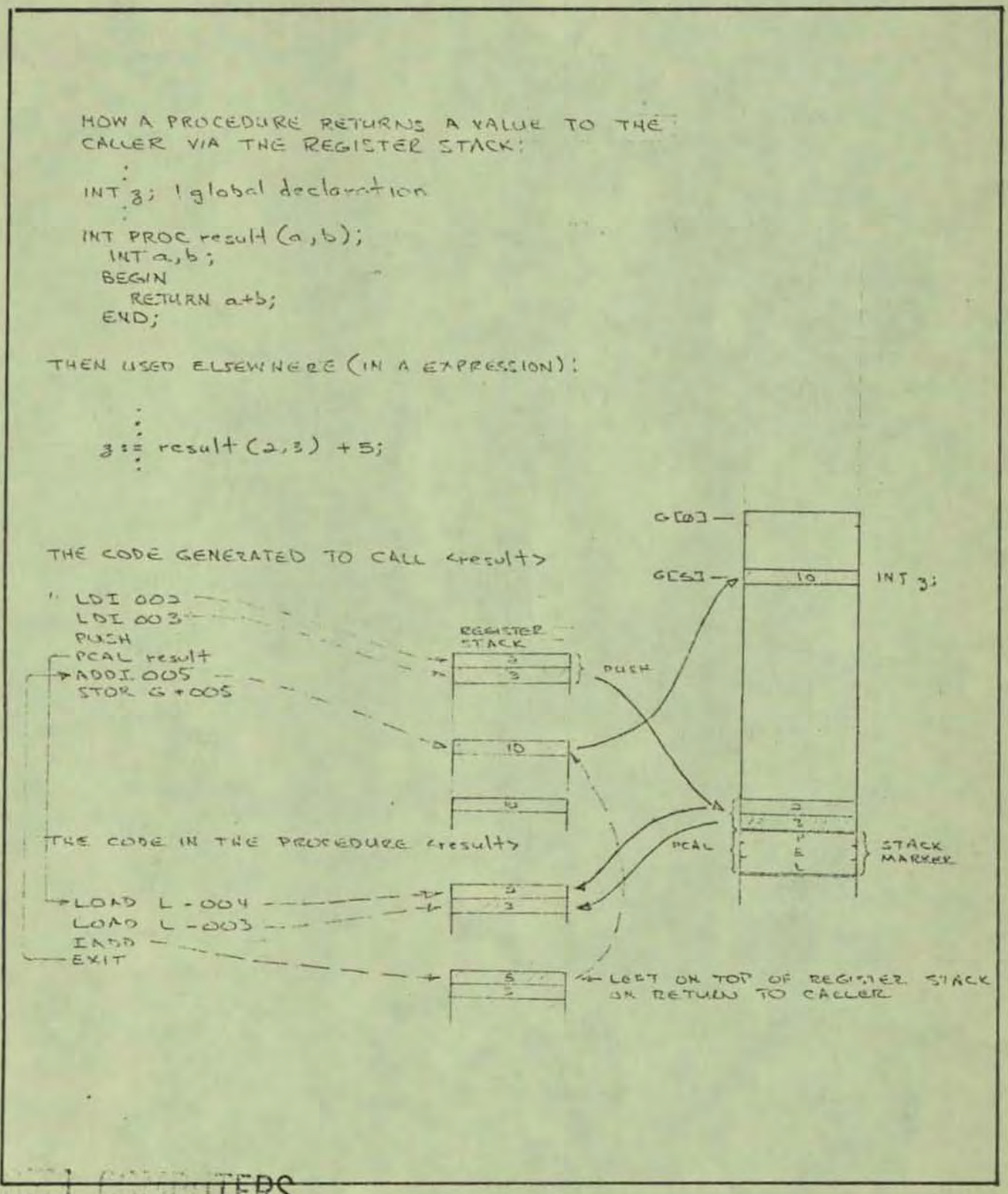


After the parameters are loaded onto the memory stack, a PCAL instruction is executed. PCAL causes the caller's three-word stack marker to be placed at the current S regster setting plus one (just above the parameters) and the L Register and S Register to be given a new setting. The parameters are now accessed by the called procedure using the 'L' minus addressing mode. This mode provides access to the 31 locations just below the current L register setting (L[-31:0]); L[-2:0] are used by the caller's stack marker. If value parameters are passed, they are accessed directly. If reference parameters are passed, they are accessed indirectly. Indexing in either mode is permitted.





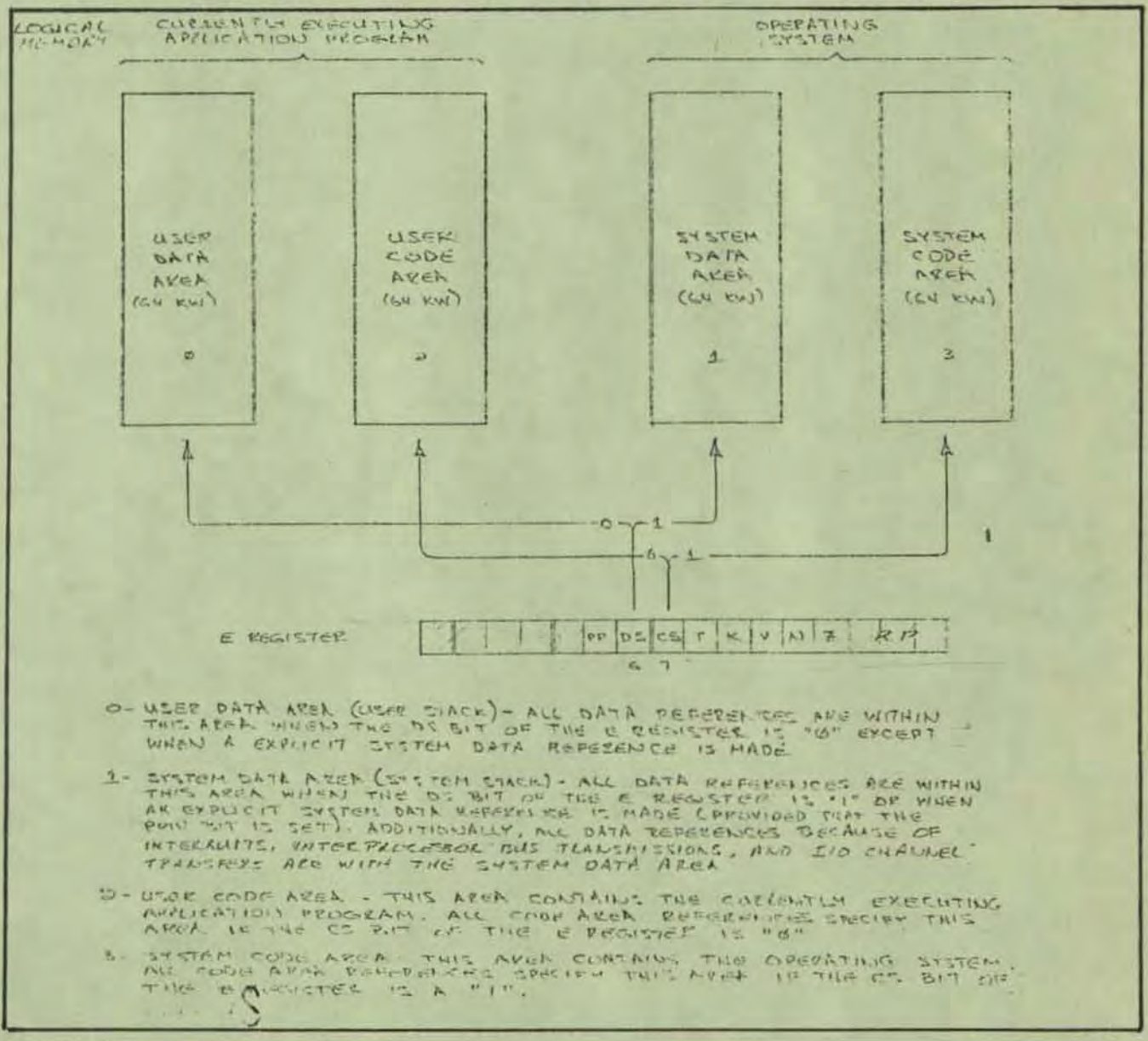
Procedures can be written that return a value to the caller via the register stack. This is shown in the following illustration:



LOGICAL MEMORY

Logical memory is separated into two areas: the user area and the system area. The user area is defined as the currently active application program; the system area is defined as the Tandem 16 operating system (T/TOS). (A number of application programs can be running in a processor module, but only one is active.)

Two bits in the E Register, the DS and CS bits, define the current environment. The DS (data space) bit defines where data references are made. Generally, this indicates whether processor is executing on behalf of a user (DS = "0") or the operating system (DS = "1"). The CS (code space) bit defines whether instructions are executed from the user area (CS = "0") or the system area (CS = "1"). Operating system procedures that are called on behalf of a user, execute with DS set to "0" (indicating user data) and CS set to "1" (indicating system code).

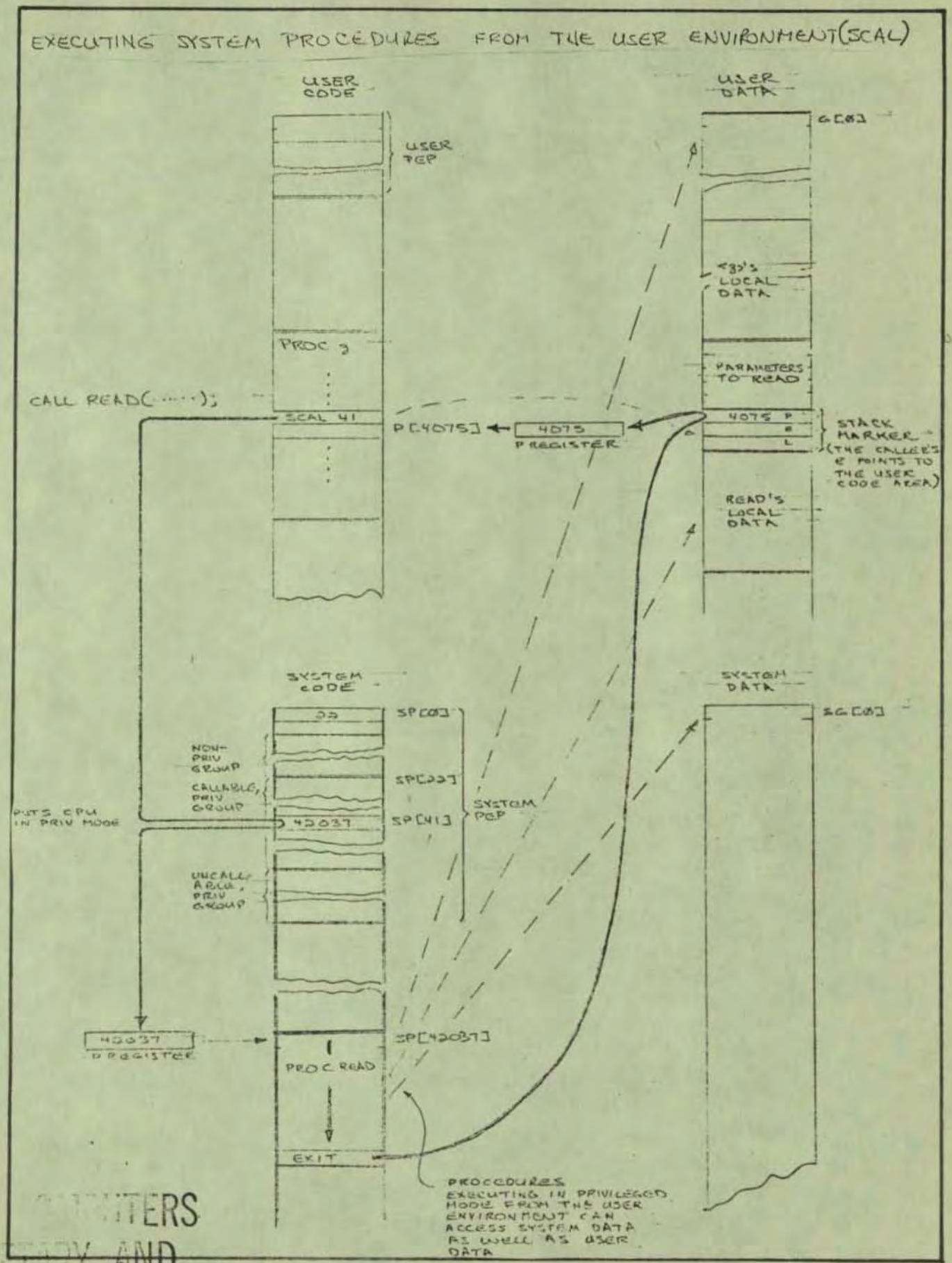


CALLING OPERATING SYSTEM PROCEDURES FROM THE USER ENVIRONMENT

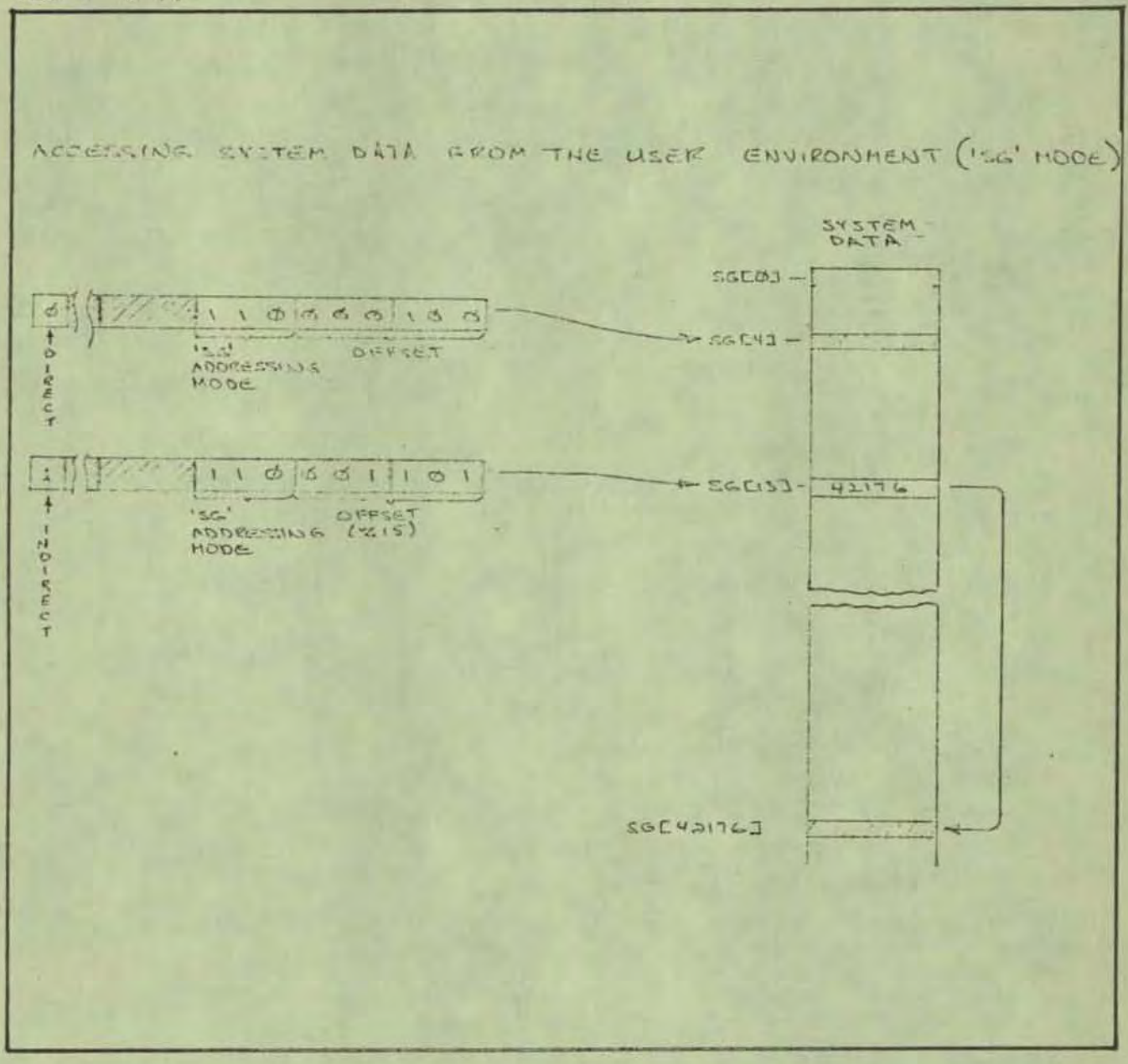
Two features permits the operating system procedures to be called and executed as efficienlty as a program's own procedures; the SCAL (system procedure call) instruction and the 'SG' relative addressing mode.

When an application program calls an operating system procedure, an SCAL instruction is executed. Executing an SCAL instruction places a three-word stack marker on the top of the user stack and moves L and S in the same manner as a PCAL instruction (i.e., allocates new local storage). However, the the system Procedure Entry Point table (PEP) is used to determine the procedure's starting address and the CS bit in the E Register is set to "1" so that instructions will be executed from the system code area. Note that the DS bit remains a "0" so that the system procedure uses the user data area for its local storage.

When the system procedure finishes, the usual EXIT instruction is executed. The CS bit is restored from the stack marker so that the next instruction is executed from the user code area.



If the system procedure must access the system data area it is designated "callable" (so that it can be called the non-privileged application program) and executes in privileged mode. Executing in privileged mode permits the procedure to make use of the "SG" addressing mode. The 'SG' mode provides access to the system data area (and, therefore any system tables) while in the user environment (DS = "0").



Executing in privileged mode while in the user environment also means that data can be moved (using a MOVW or MOVB) instruction between the user data area and the system data area. This is the method used by system procedures when transferring data between the user data stack and the File System.

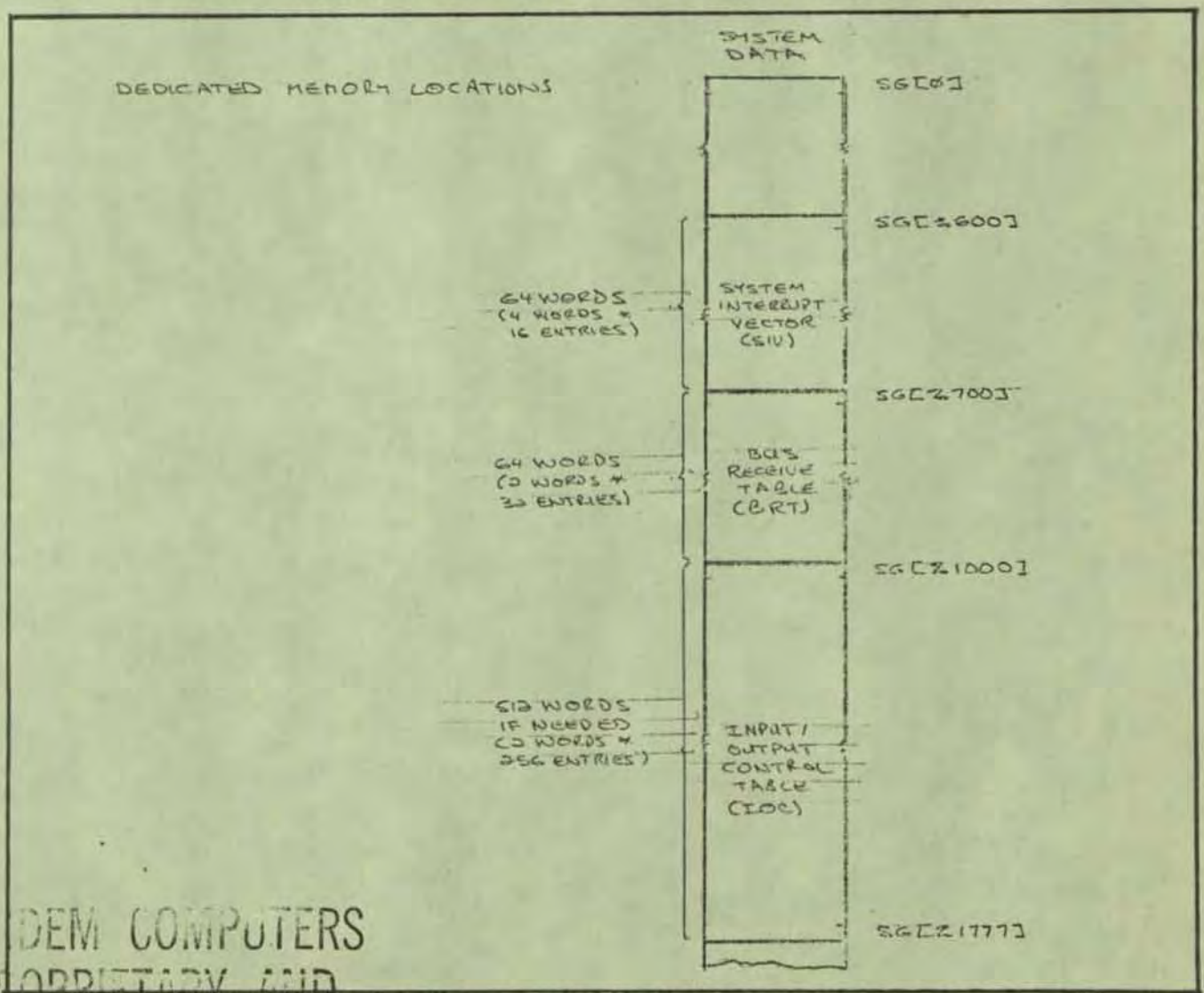
SYSTEM TABLES

The following locations in the system data area are tables known to the hardware.

SG[%600:%677] is the System Interrupt Vector. This table contains 16 four-word entries; each entry defines the executing environment forone of the 16 operating system interrupt handlers (see Interrupts).

SG[%700:%777] is the Bus Receive Table. This table contains 32 entries corresponding to two buses from each of 16 processors. Each entry decribes the number of words expected and where the system data location where the data is to be stored (see Interprocessor Bus).

SG[%1000:%1777] is the I/O Control Table. This table contains up to 256 entries corresponding to the 32 controllers with up to eight units each that can be connected to an i/o channel. Each entry describes the number of bytes to be transferred and the system data location where the data transfer takes place (see Input/Output Channel).

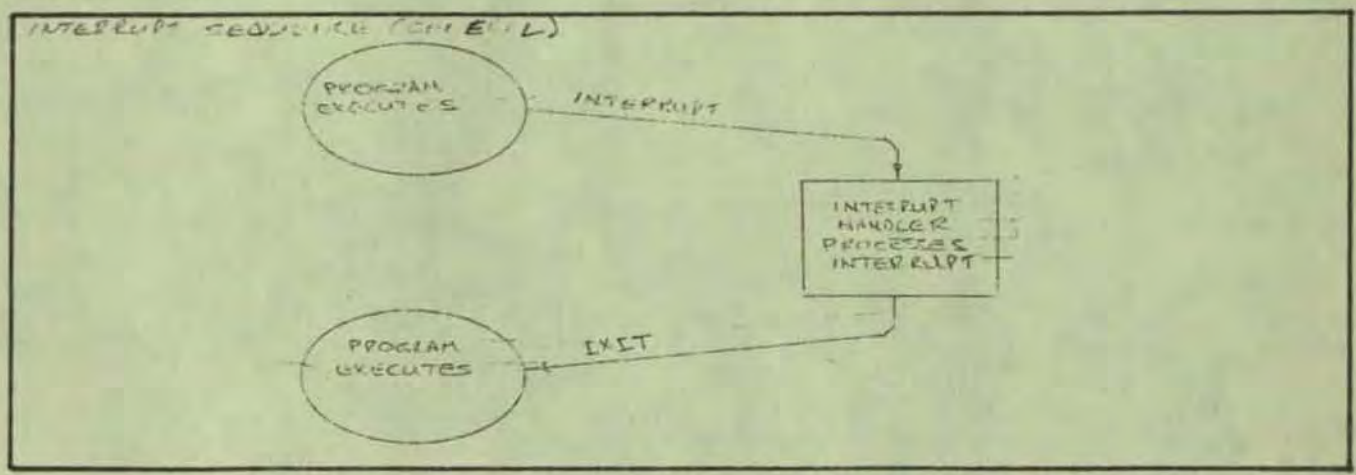


INTERRUPTS

The interrupt system causes a transfer of control to a specific location in the operating system (called an interrupt handler) upon the occurrence of any of the following events:

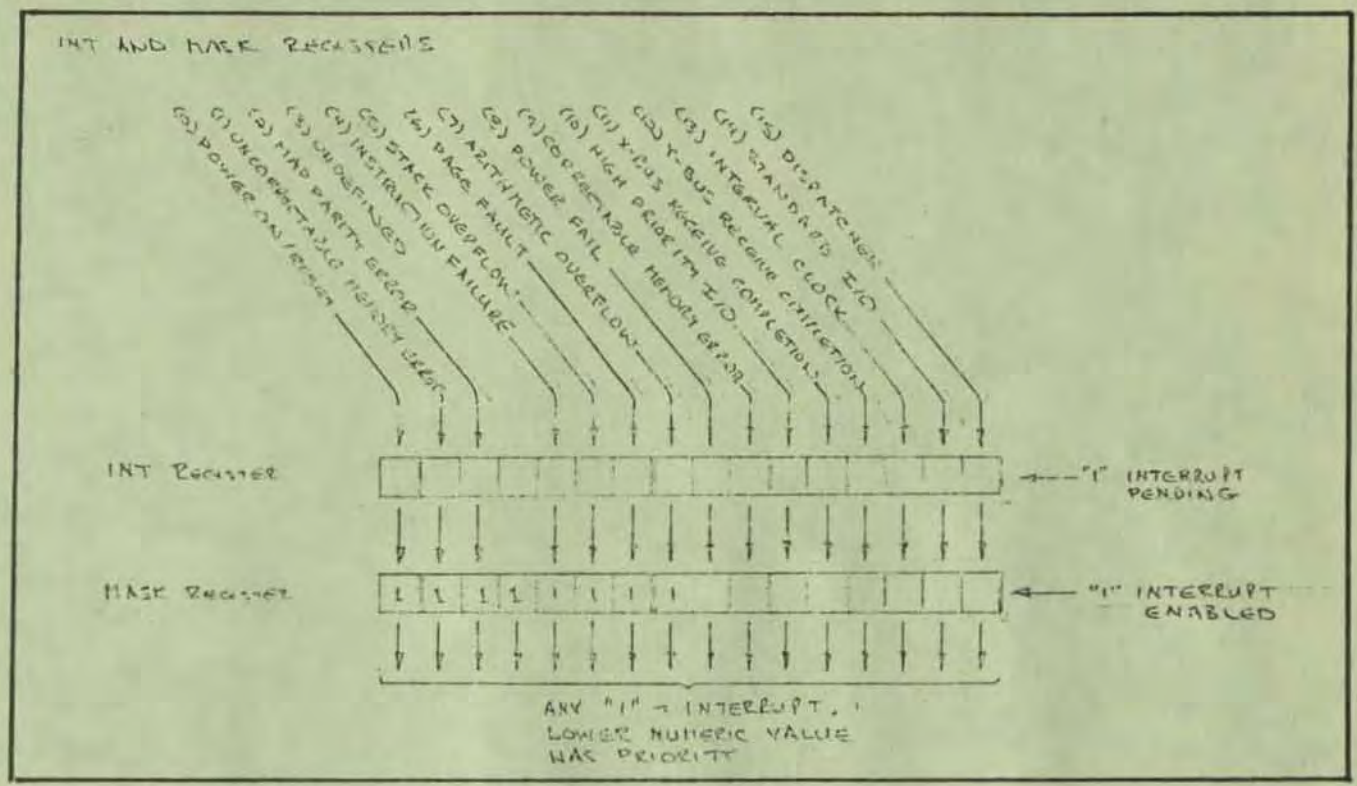
(0) Power On / Reset
(1) Uncorrectable Memory Error
(2) Map Parity Error
(3) undefined
(4) Instruction Failure
(5) Stack Overflow
(5) Page Fault
(7) Arithmetic Overflow or Divide by Zero
(8) Power Fail
(9) Correctable Memory Error
(10) High-priority I/O
(11) X-bus Receive Completion
(12) Y-bus Receive Completion
(13) Interval Clock
(14) Standard I/O
(15) Dispatcher

Generally, when an interrupt occurs the interrupted environment is saved in an interrupt stack marker. An operating system interrupt handler executes to process the particular interrupt. Then an IXIT (interrupt exit) instruction is executed to restore the interrupted environment.



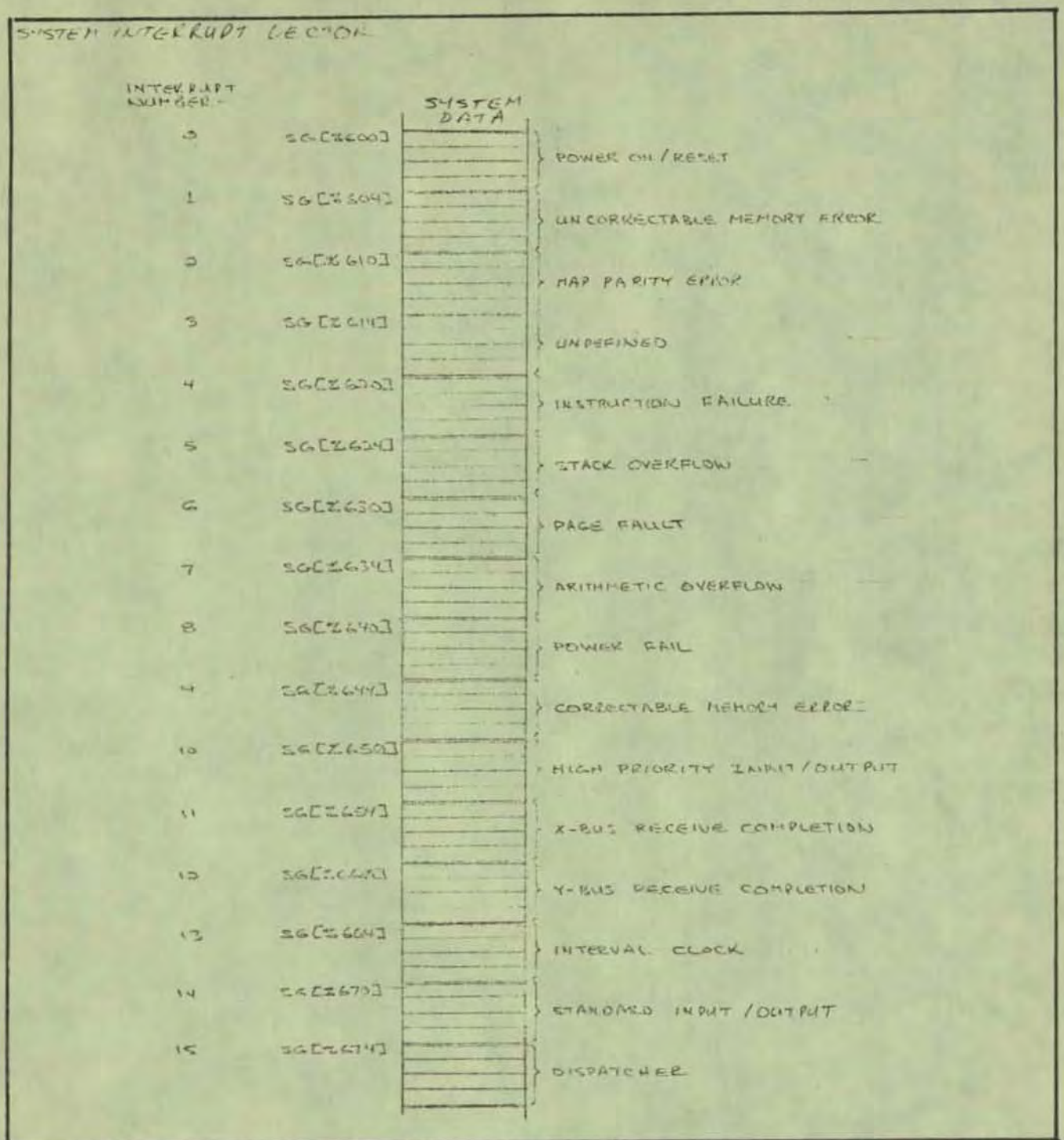
Two registers are associated with interrupts: the 16-bit INT register and the 16-bit MASK register. Each bit the INT and MASK registers corresponds to one of the 16 possible interrupt conditions. The INT register is used to flag that an interrupt condition occurred (interrupt pending); the MASK register is used by the operating system to control when a particular interrupt type is allowed. MASK.<0:7> are always set to "1"s so that interrupt types zero through seven

cannot be prevented.



An interrupt occurs when an interrupt condition is detected, setting an INT register bit, and the corresponding bit in the MASK register is a "1". Interrupts normally occur at the end of an instruction except that interrupt types one through six may prematurely terminate an instruction. Also, if an interrupt condition occurs and the corresponding MASK bit is a "0", the interrupt is deferred until the MASK bit is set to a "1". Additionally, if two or more interrupt conditions occur simultaneously, and both have the correspond MASK register bits set, the interrupt type with the lowest number takes precedence (the other is deferred until the interupt handler finishes executing).

Each event has a corresponding entry in the System Interrupt Vector (SIV). The SIV, which is initialized by the operating system, defines the executing environment for each of the 16 operating system interrupt handlers. The SIV begins at system data location %600 and contains 16 4-word entries:



Specifically, the SIV entry for interrupt type i is:

SG [%600 + 4i : %603 + 4i]

Each entry in the system interrupt vector contains the following information:

* The address in the system data area for an interrupt handler's local storage (stack). These local storage areas are shared by interrupt handlers that can't be active at the same time.

* A mask value for masking off unwanted interrupts while an interrupt handler executes. The operating system usually masks off all interrupt types except zero through seven (which can't be masked off) and types eight (power fail) and ten (high priority i/o completion).

  The MASKi value in the SIV entry is ANDed with the current MASK register setting to derive an new setting. This permits nesting of interrupts of different types.

* The system code address of the interrupt handler program

* A location where an interrupt related parameter may be returned by the hardware.

When an interrupt occurs, the interrupted environment is saved in an interrupt stack marker. The interrupt stack marker is located at L1 in the interrrupt control stack. The interrupt contains the following information:

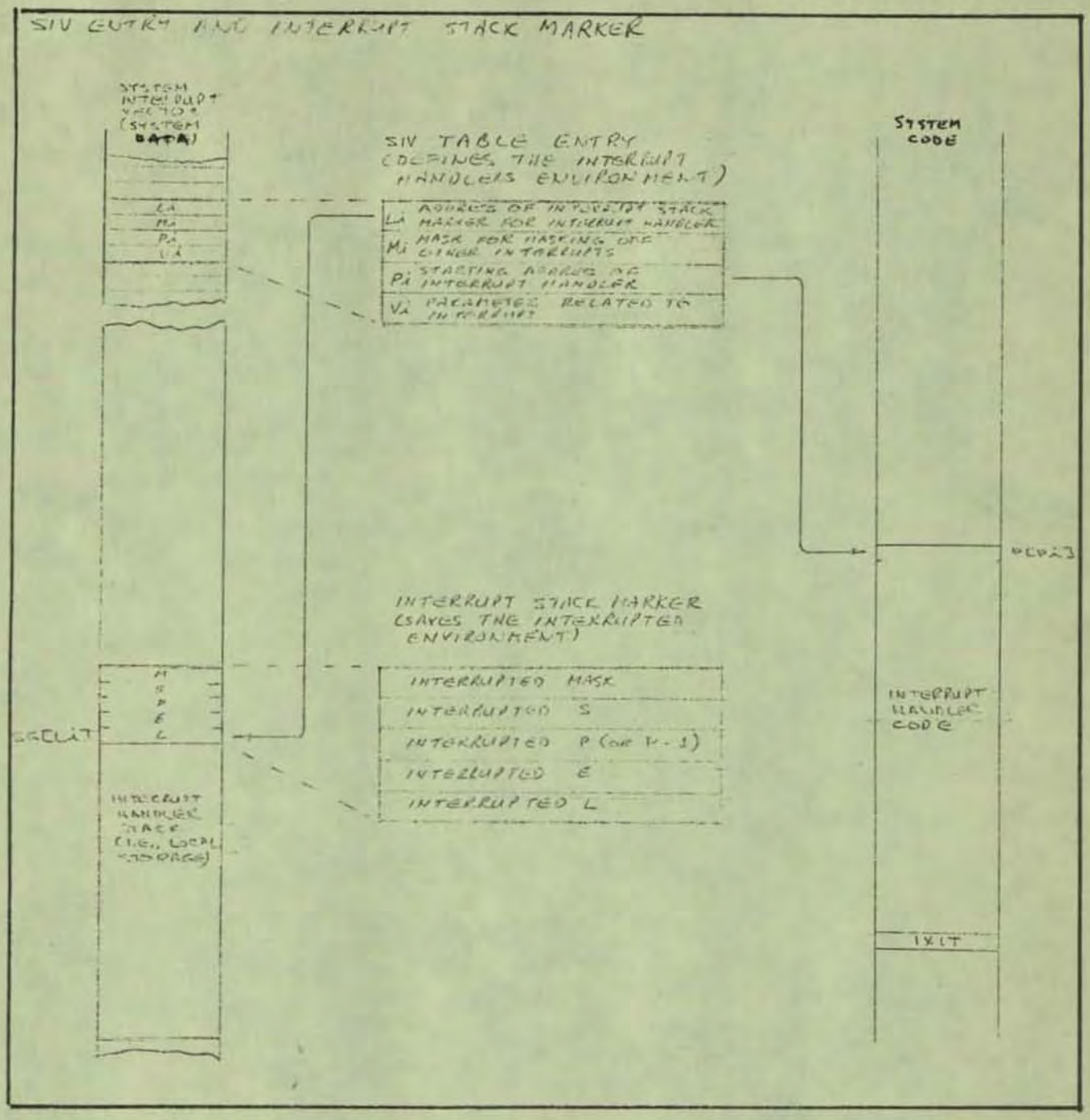
* The MASK register setting at the time of the interrupt.

* The S Register setting at the time of the interrupt.

* The P Register setting at the time of the interrupt.

* The E Register setting at the time of the interrupt.

* The L Register setting at the time of the interrupt.



Interrupt Sequence

Specifically, an interrupt is defined as (i is the interrupt type):

```
Vi := interrupt parameter;          ! for interrupts 1, 2, 6, 9, 11, 12
e := E;
E := %3407;                         ! PRIV, DS, CS, RP=7
SG[Li-4:Li] := (MASK,S,P,e,L);      ! interrupt stack marker
S := L := Li;
P := Pi;
MASK := MASK LAND MASKi;
```

An example is shown in the following illustration:

1. An interrupt condition occurs (in this example, the interval clock).

2. The current instruction completes executing (note that some other interrupts may terminate an instruction prematurely)

3. The PRIV (privileged mode), DS (data space), and CS (code space) bits in the E Register are set. This defines the interrupt handler executing environment.

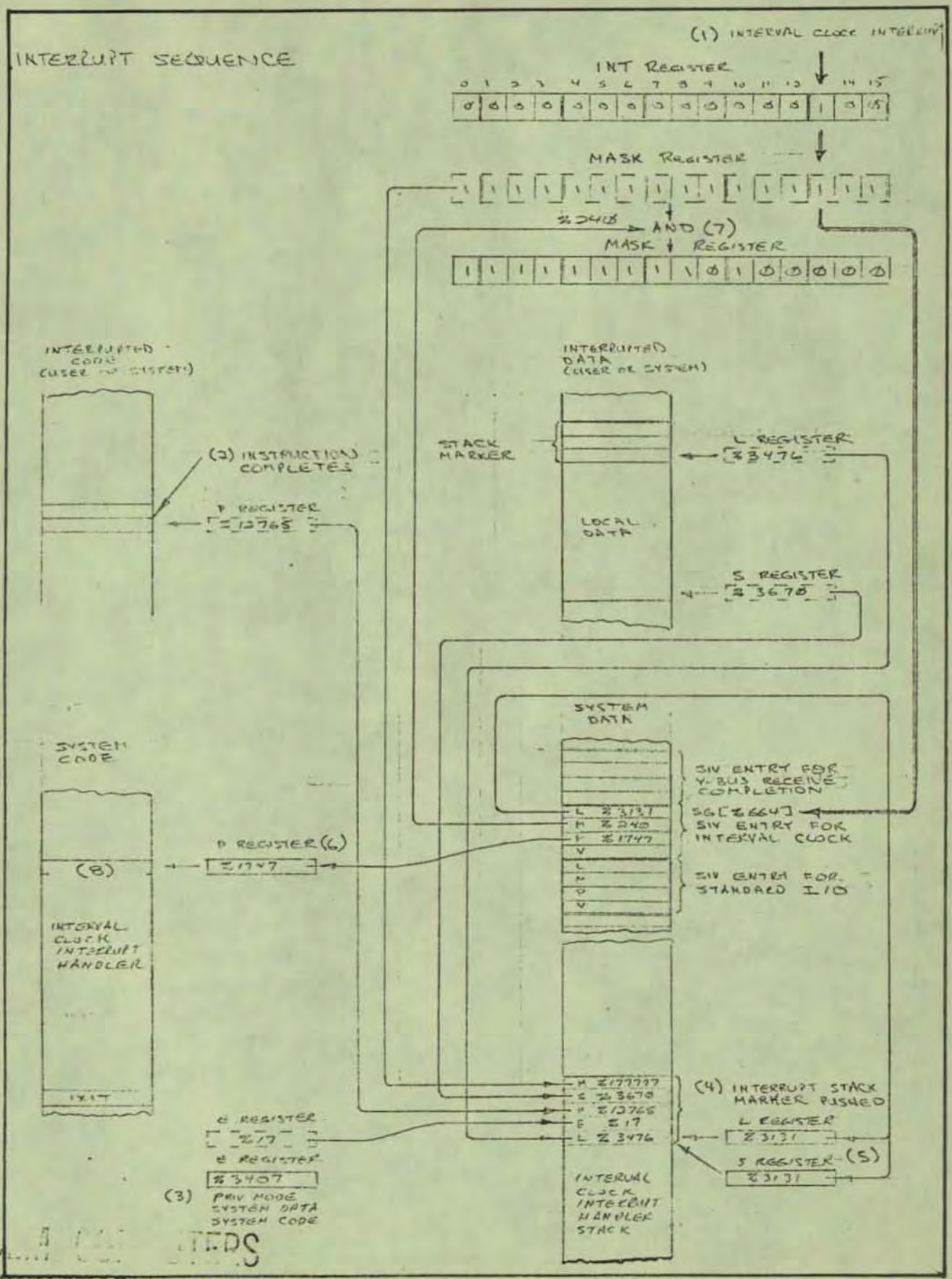
4. The interrupted environment (included the current MASK and S register settings) are saved in the location pointed to by Li in the SIV entry for the interval clock (i.e., SG[%664]).

5. The L and S Register are set with the address of the interrupt handler's local data area. This is the value Li in the SIV entry for the interval clock.

6. The P register is set with the address of the first instruction in the interval clock interrupt handler. The is the value Pi in the SIV entry for the interval clock.

7. The MASKi value in the SIV entry is ANDed with the current MASK register setting to derive a new MASK register setting.

8. The first instruction of the interval clock interrupt handler executes.

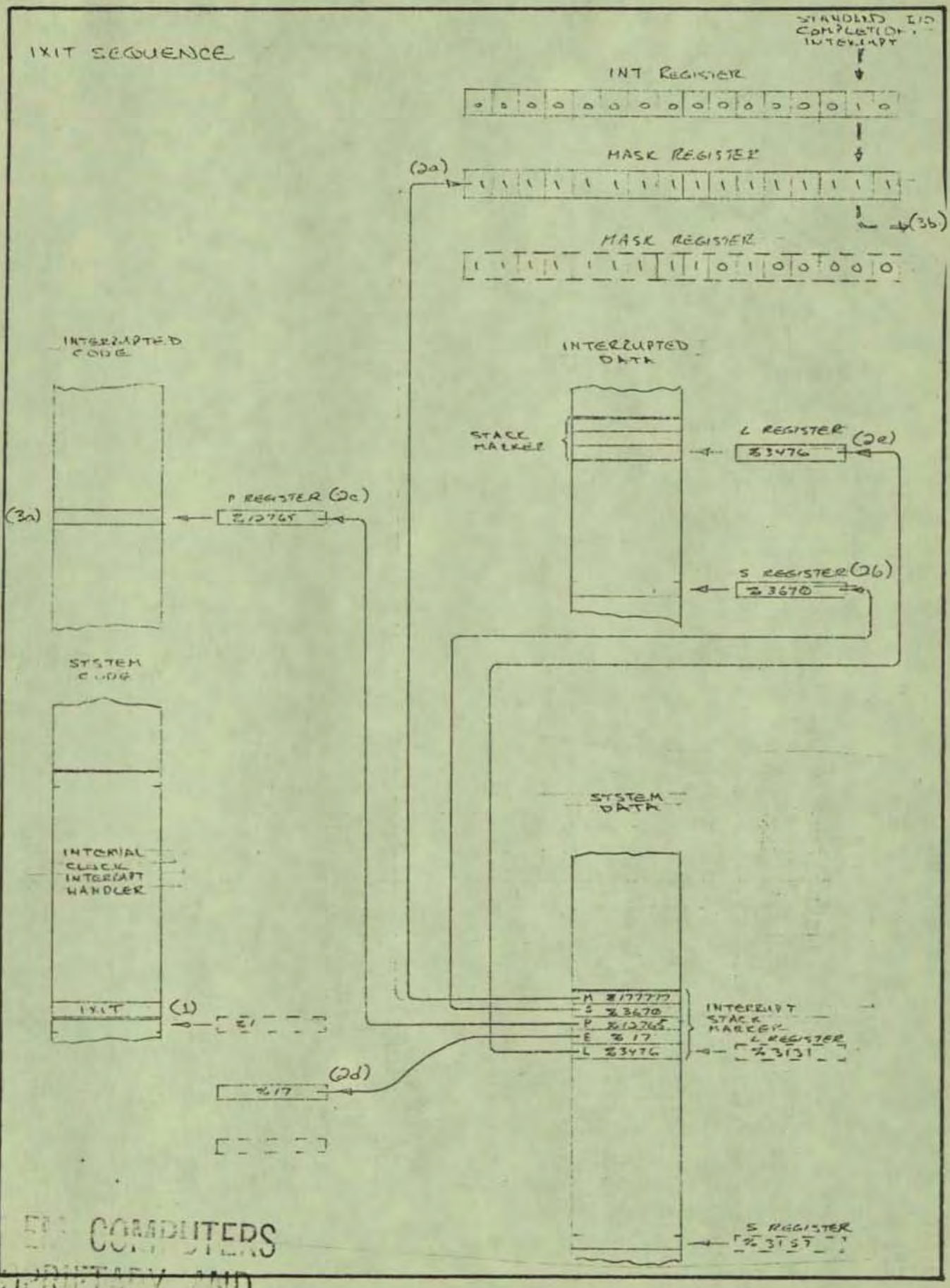


9. The interrupt handler runs to completion (unless interrupt by an unMASKed interupt type). Then an IXIT instruction is executed to return to the interrupted program.

10. The interrupted environment saved in the interrupt stack marker (at L[0]) is restored; The MASK, S, P, E, and L registers are returned to their pre-interrupt value.

11a. If no interrupt is pending when the IXIT instruction finishes, program execution resumes at the point of interruption.

11b. If an interrupt is pending, the interrupt sequence is repeated from step 1 using the appropriate SIV entry to set up the interrupt handlers environment.

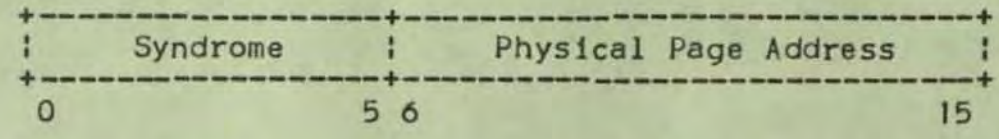


Interrupt Types

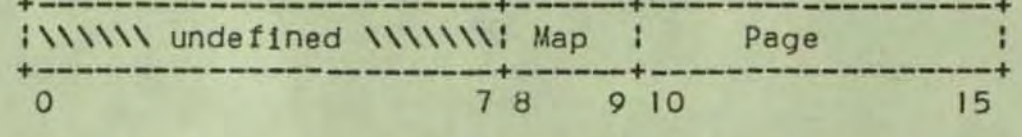
Following are descriptions of each of the individual interrupts:

(0) Power On (SIV entry at SG[%600]): This interrupt occurs when power is applied following a power failure when memory is in a valid state. There is no parameter for this interrupt.

(1) Uncorrectable Memory Error (SIV entry at SG[%604]): this interrupt occurs when a memory word is read and contains an error which cannot be corrected. The parameter contains the physical address of the page at fault and the 6 syndrome bits generated by the error correction circuitry. The format of the parameter is:



(2) Map Parity Error (SIV entry at SG[%610]): This interrupt occurs when a memory map word is read and itself contains a parity error. The interrupt parameter contains the address of the map entry which was in error. The format of the parameter is as follows:



(4) Instruction Failure (SIV entry at SG[620]): This interrupt occurs when an unimplemented instruction is executed or a privileged instruction is executed by a program which is not in privileged mode. There is no parameter for this interrupt.

(5) Stack Overflow (SIV entry at SG[%624]): This interrupt occurs when S exceeds %77777 following a SETS, PCAL, SCAL, ADDS, BSUB, or PUSH instruction. There is no parameter.

(6) Page Fault (SIV entry at SG[%630]): This interrupt occurs upon access to memory via a map entry whose absent bitis set to 1. The parameter is the same as provided on a map parityinterrupt.

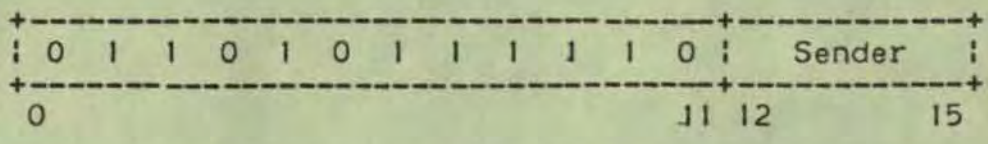
(7) Arithmetic Overflow (SIV entry at SG[%634]): This interrupt occurs when the V and T bits in the E register are simultaneously set to 1. There is no parameter.

(8) Power Fail (SIV entry at SG[%640]): This interrupt occurs at least.5 msec before the power goes. There is no parameter.

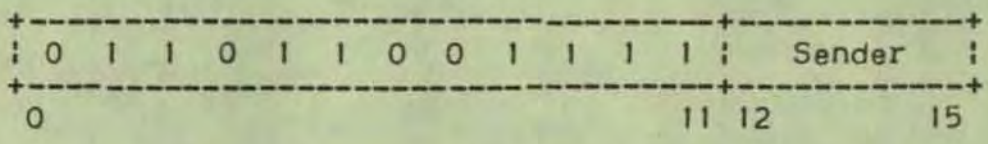
(9) Correctable Memory Error (SIV entry at SG[%644]): This interrupt occurs when a memory error occurs but can be corrected. The parameter is the same as that provided on an uncorrectable memory error.

(10) High-priority I/O Completion (SIV Entry at SG[%644]): This interrupt occurs when a device which is connected to the high-priority interrupt poll line requires servicing. There is no parameter.

(11) X-Bus Receive Completion (SIV entry at SG[%654]): This interrupt occurs when a transmission is received on the X-bus. The parameter has the following format:



(12) Y-Bus Completion (SIV entry at SG[%660]): This interrupt occurs when a transmission is received on the Y-bus. The parameter has the following format:



(13) Interval Clock (SIV entry at SG[%664]): This interrupt occurs every 10 milliseconds. There is no parameter.

(14) Standard I/O Completion (SIV entry at SG[%670]): This interrupt occurs when a device which is connected to the standard interrupt poll line requires servicing. There is no parameter.

(15) Dispatcher (SIV entry at SG[%674]): This interrupt occurs when a DISP instruction is executed. There is no parameter.

For a description of the action taken by each operating system interrupt handler, see the Operating System Overview in section four.

When an interrupt occurs, further interrupts of the same type are disabled. To permit future interrupts of the same type, the appropriate INT register bit must be cleared.

The four following interrupts types must be cleared explicitly by executing a RIR (reset interrupt register) instruction:

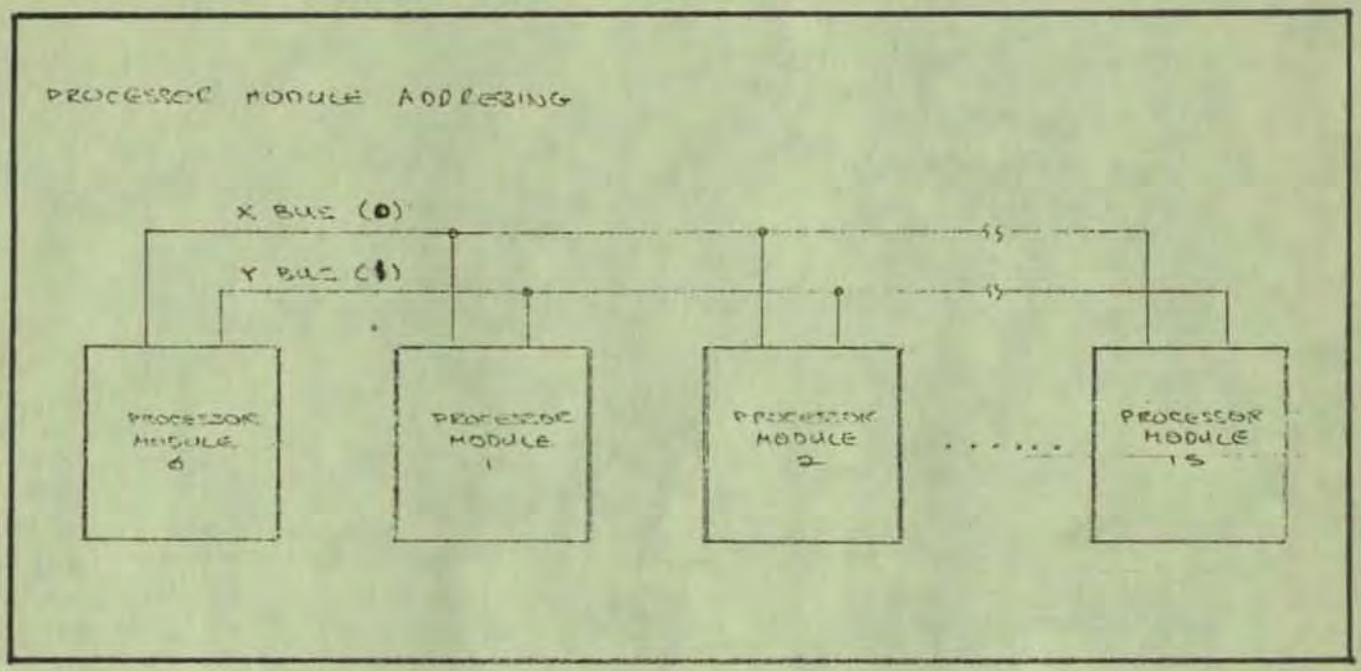
X-bus Receive Completion
Y-bus Receive Completion
Interval Clock
Dispatcher

The High-priority I/O and Standard I/O interrupts are cleared implicitly by clearing the device. Power Fail is effectively cleared by power up. Instruction Failure and Stack Overflow occur only during the violating instruction and need not be cleared. Arithmetic Overflow cannot be cleared by RIR, but persists if the offending program resumes with the condition still valid. Uncorrectable Memory

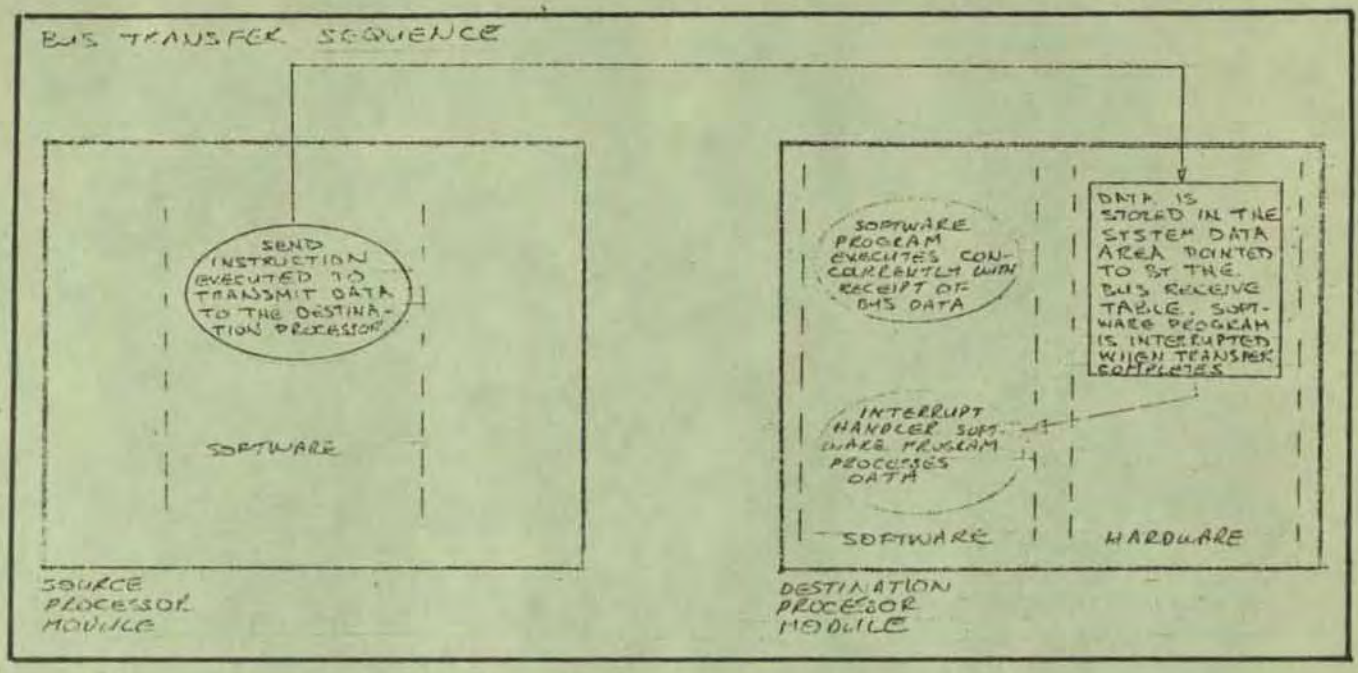
Error, Map Parity Error, Page Fault, and Correctable Memory Error interrupts are automatically cleared when the interrupt takes place.

INTERPROCESSOR BUS

A Tandem 16 Computer System has two interprocessor buses, designated X-bus and Y-bus. Each processor module in the system is connected to both buses and is capable of communicating with any other processor module (including itself) over either bus.



With any given interprocessor bus transfer, one processor module is the source (and initiator), the other is the destination (and receiver). For a processor to receive data over an interprocessor bus, the operating system first configures an entry in a table known as the Bus Receive Table (BRT). Each BRT entry contains the address where the incoming data is stored and the number of words expected. To initiate a transfer, a SEND instruction is executed in the source processor module; the SEND instruction specifies the bus to be used for the transfer, the destination processor module, the number of words to be sent, and the location in memory of the data to be sent. While the source processor module is executing the SEND instruction and sending data over the bus, the bus control harwdare in the destination processor module is storing the data away according to the appropriate BRT entry (this occurs concurrently with program execution). When the destination processor module receives the expected number of words, the currently executing program is interrupted and the bus transfer is completed.



Bus Receive Table (BRT)

The BRT contains 32 two-word entries: corresponding to two buses from each of 16 processor modules possible in a system. Specifically, the entry for bus b and processor module p begins at location:

SG[%700 + %40b + 2p]

b = 0 (X bus) or 1 (Y bus)
p = 0:15

Each entry in the BRT contains an address in the System Data area where the incoming data is to be stored and a count of the number of words expected.

Additionally, if a processor wants to receive data over a designated bus, the corresponding bit in the interrupt MASK register must be a "1." The bits are:

X-bus Receive Enable = MASK.<11>
Y-bus Receive Enable = MASK.<12>

SEND Instruction

The SEND instruction requires four parameter words in the register stack:

* D.<0> specifies the bus (0 = X bus, 1 = Y bus)
  D.<12:15> specify the destination processor module

* C.<0:15> is a value that is subtracted from 32,768 to derive the number of 0.8 microsecond units allotted to completing a single

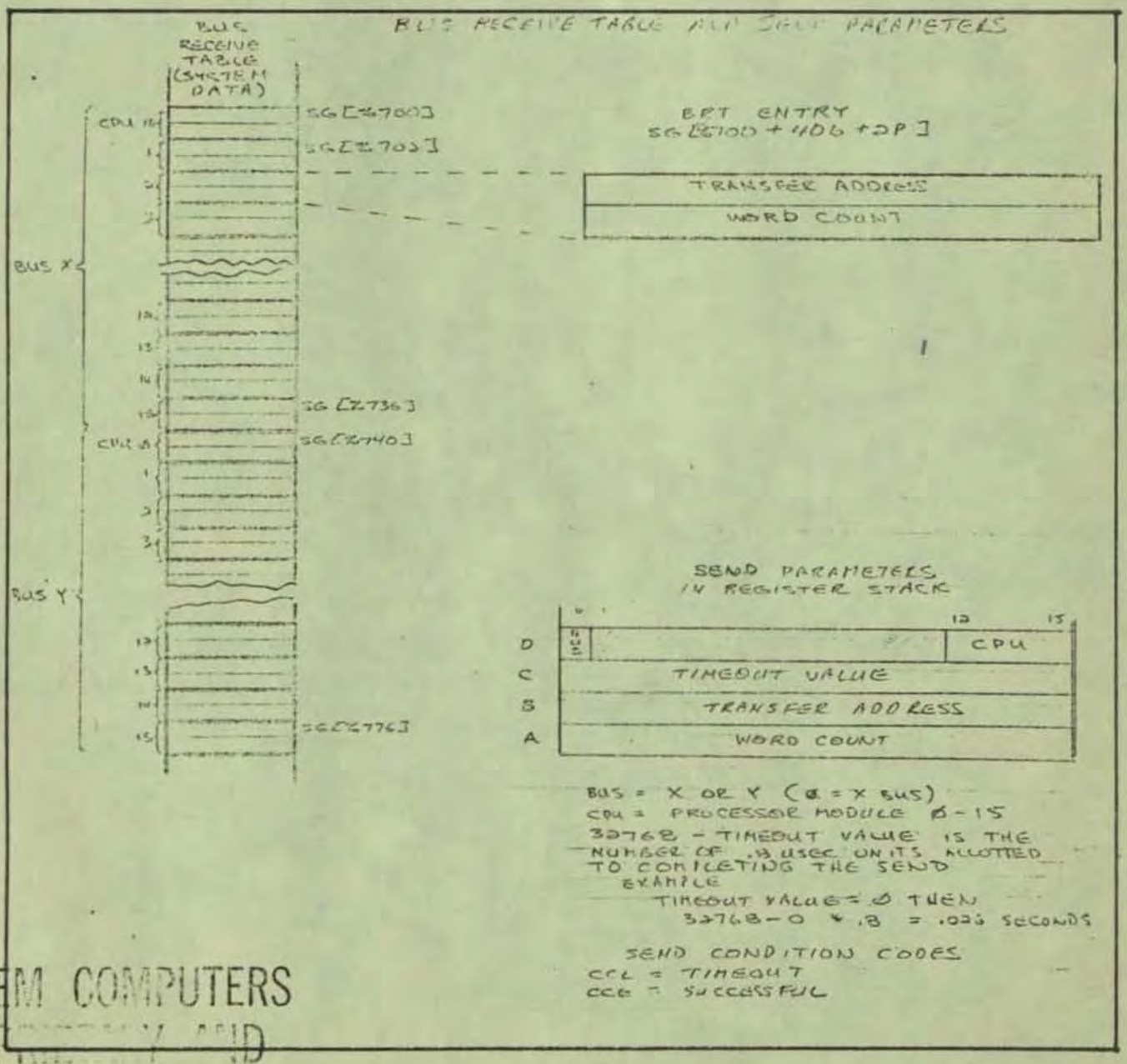
packet (15 word) transfer. The timeout period is restarted for each packet transferred

* B.<0:15> is the address in the System Data area where the data to be transferred is located.

* A.<0:15> is a count of the number of words to be transferred. This value must match the number expected by the BRT in the destination processor module if the transfer is to complete successfully

* As a result of executing the SEND instruction, the condition code is set to either of two values:

  CCL = Packet Timeout
  CCE = Successful



Bus Transfer Sequence

As previously stated, there must be coordination between source processor module and the destination module in regards to the number of words to be transferred. The operating system accomplishes this by preceding each transfer with a separate transfer (i.e., SEND) of a predetermined number of words. The initial transfer consists of 14 words of control information. In general, this control information tells the operating system in the destination module to expect a specified number of words over a specified bus. In the following illustration, assume that the inital transfer has taken place and that the operating system in the destination module has configured the appropriate BRT entry for receiving 200 words.

1. A SEND instruction is executed in the source processor module (processor module 1). The SEND parameters specify:

   * X-Bus to Processor Module 3 (D)

   * A packet timeout value of 0 (meaning that a timeout occurs if a single packet transfer takes longer than 26 milliseconds) (C)

   * System Data address SG[1234] as the data to be transmitted is located (B)

   * A count of 200 words to be transmitted

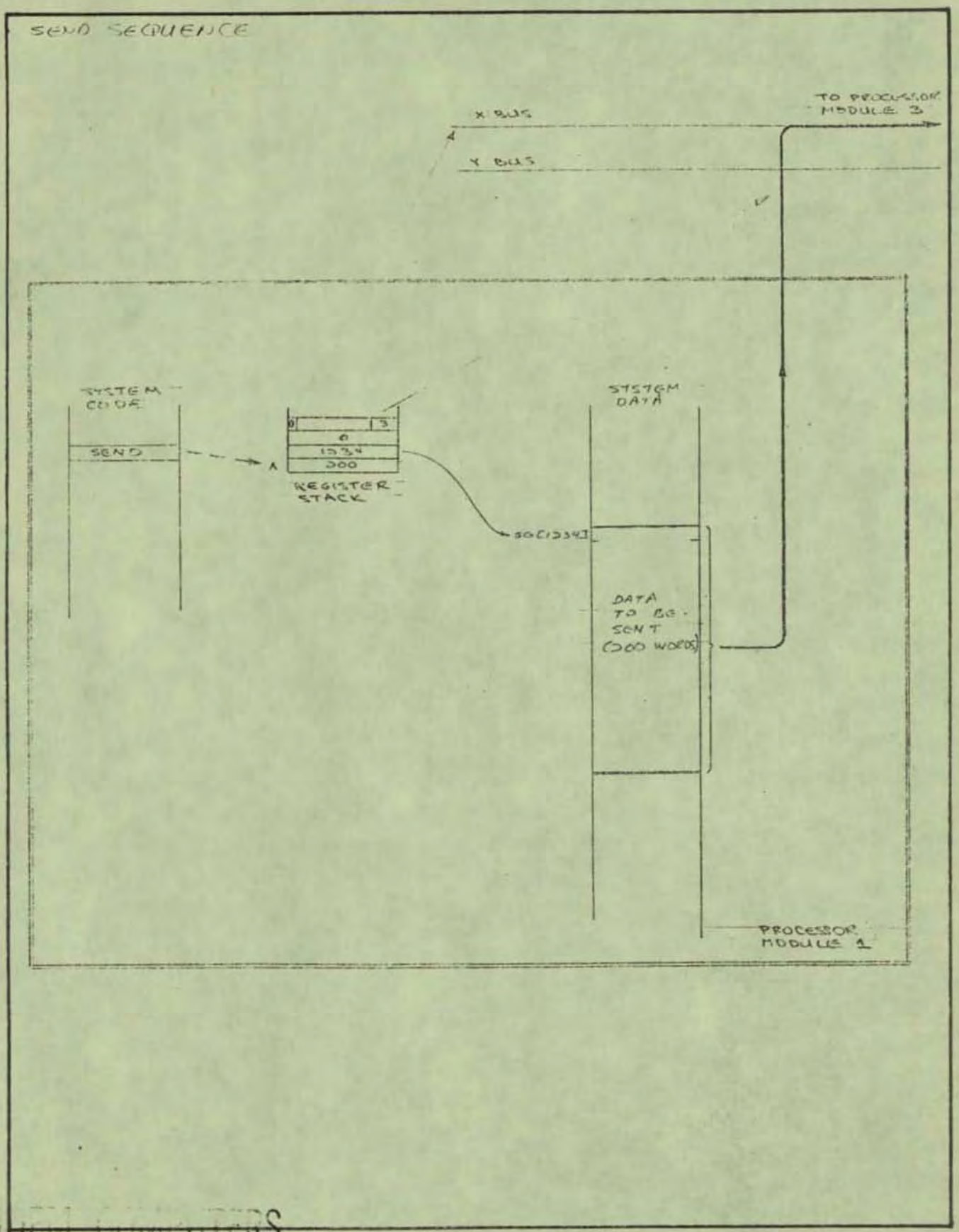
   The SEND instruction transmits the 200 words to processor module 3 via the X-bus then completes. The parameters are deleted from the register stack and the condition code is set to CCE (indicating a successful operation).

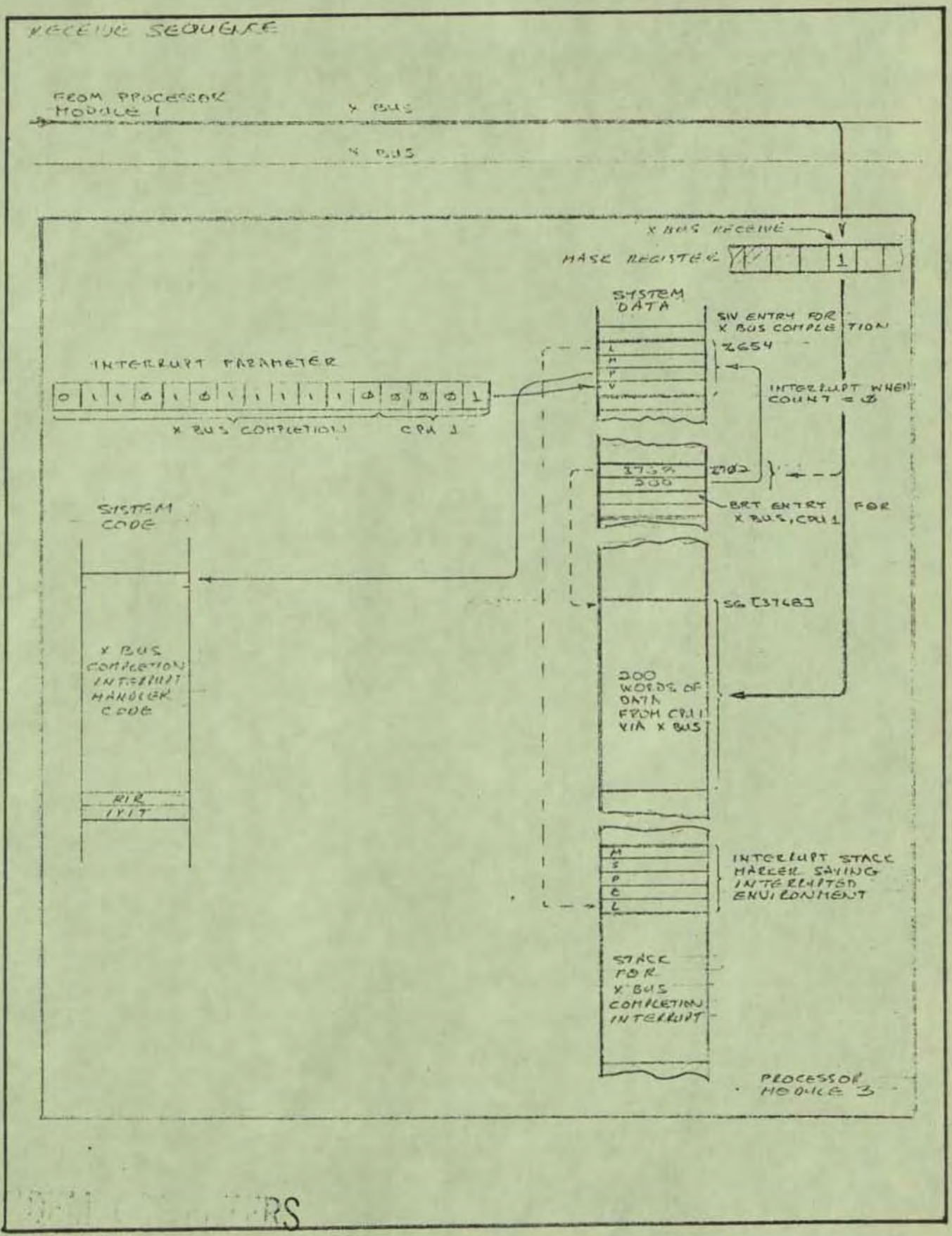
2. Processor Module 3, which has been previously readied for this transfer, has MASK.<11> set to a "1" to enable receipt of data over the X-bus and has the BRT entry for X-bus, processor module 1 configured as follows:

   * The transfer address where the incoming data is to be stored, system data location SG[3768]

   * The count of the number of words expected, 200.

3. The data, as received is stored away as indicated by the BRT entry. As the data is stored, the transfer address is incremented accordingly and the count is decremented accordingly.





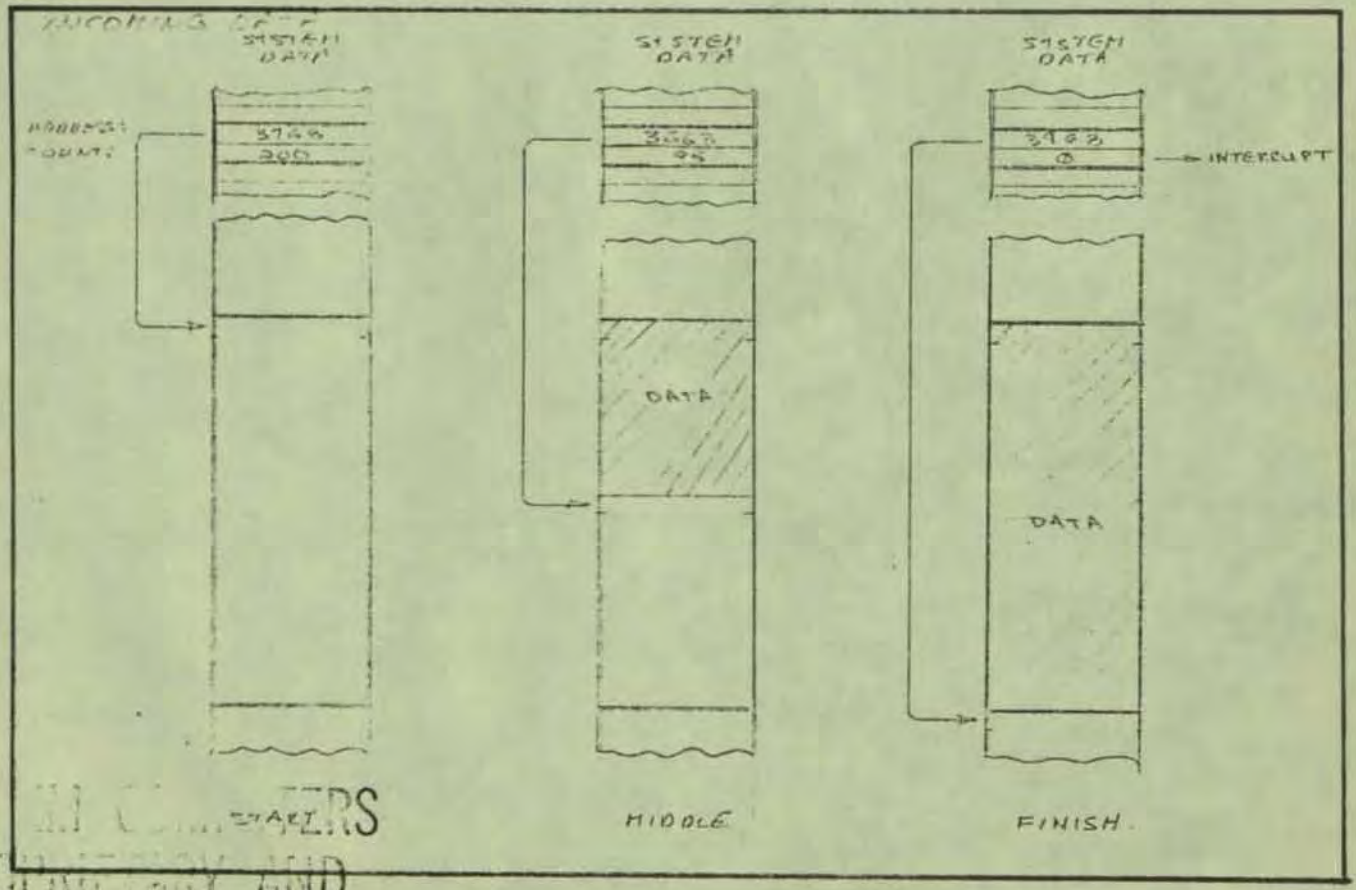
4. When the count in the BRT entry reaches zero, 200 words have been received. At this point an interrupt through the SIV (System Interrupt Vector) from X-bus completion occurs. There is a parameter associated with this type of interrupt, it contains the processor module number of the source processor module.

   Because, INT.<11> in the interrupt register is now set, further data transmissions to this processor module over the X-bus are rejected. Additionally, the MASKi word in the SIV entry for the X-bus masks off further interrupts in the MASK.<11> position. (Remember that bus completion interrupts remain pending, even though an interrupt has occurred, until reset by a RIR instruction.)

6. Prior to executing the IXIT (interrupt exit) instruction to return to the interrupted program, the interrupt handler executes a RIR (reset interrupt register) for INT.<11> (X-bus recieve). This partially enables data transfers over the X-bus into this processor module.

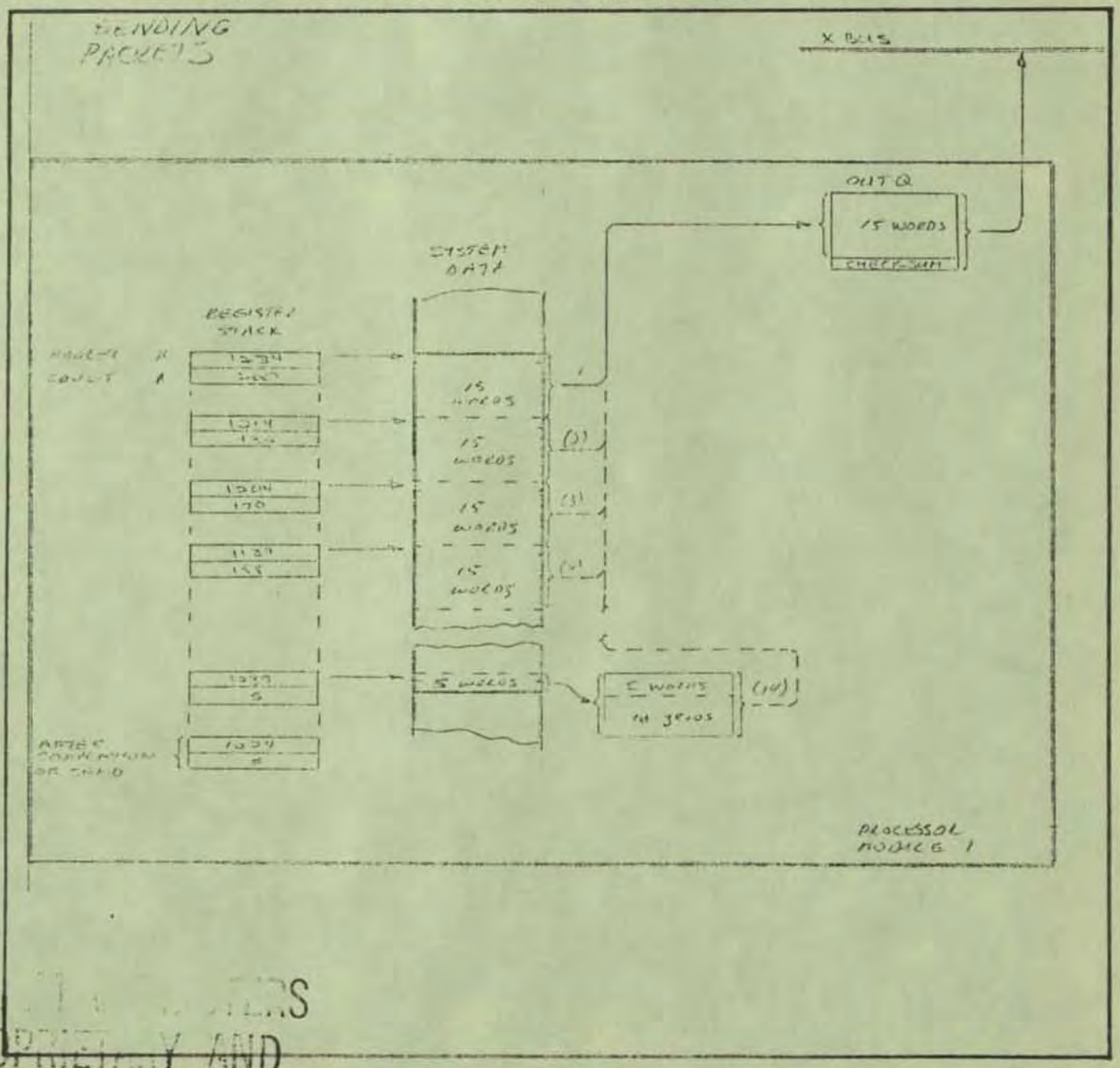
7. When the IXIT instruction executes, the previous MASK register setting is restored. Processor module 3 is now enabled for receiving data over the X-bus.

The following illustration shows the relationship between the transfer address and count in the BRT entry and the incoming data in the transfer location.



OUTQ, INQ and Packets

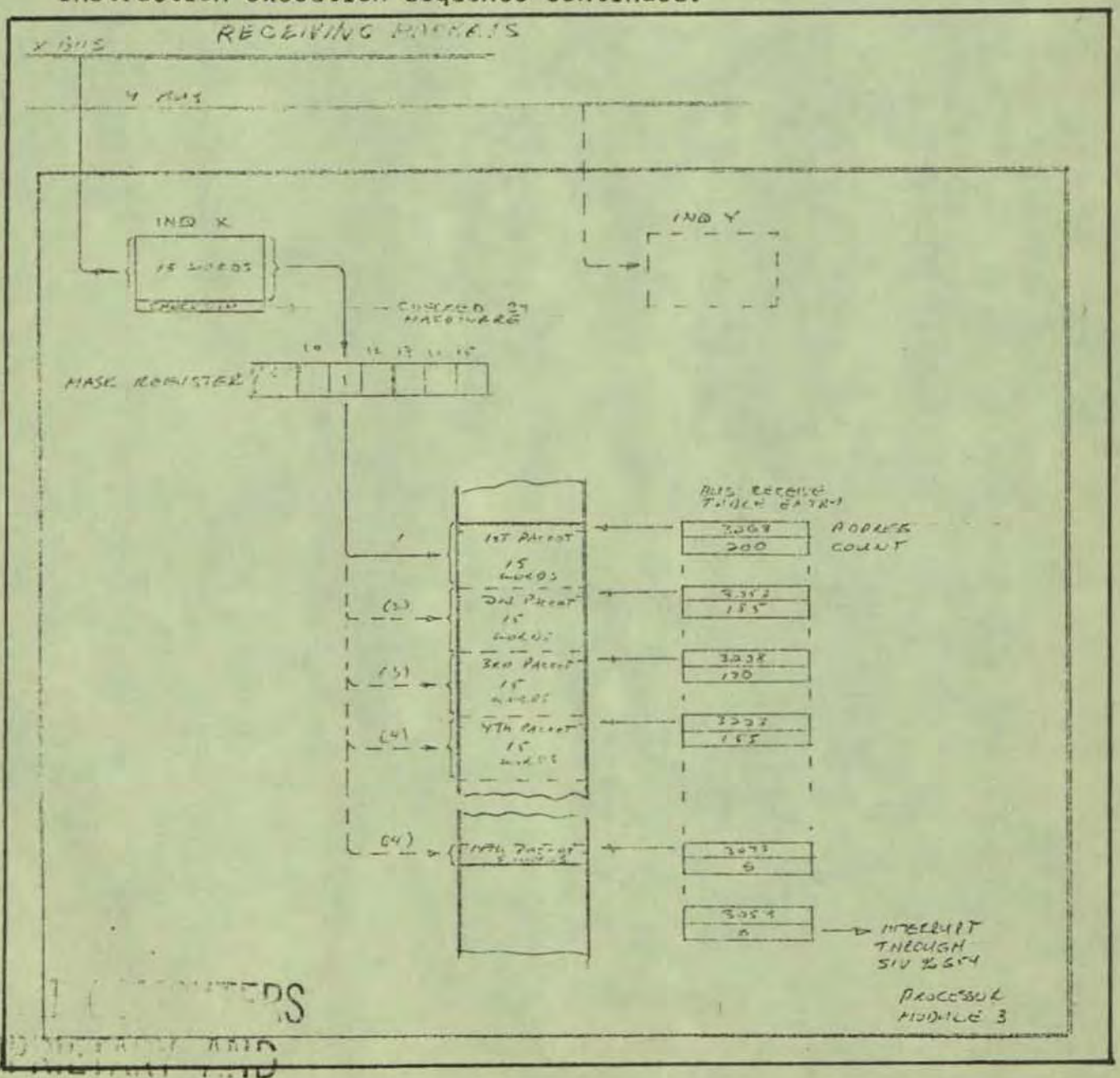
The interprocessor buses are significantly faster than memory, therefore each processor has buffered interface to the bus, consisting of a 16-word output buffer (called the OUTQ) and two 16-word input buffers, one for each bus (called INQ X and INQ Y). Data is transmitted over a bus in the form of 16-word packets. The SEND instruction fills the output buffer with 15 words, appends an odd-parity checksum, and signals the system bus controller that it has a packet ready for transmission. After the 16-word packet is transmitted, execution of the SEND instruction resumes at the point where it left off. If the last packet of the block is less than 15 words, the remaining words are filled in with zeroes. The instruction terminates when the last packet is transmitted.



When either of the INQ X or INQ Y buffers in the destination processor module is filled and the corresponding MASK register bit is a "1", a microinterrupt occurs. The action taken by the processor module during the microinterrupt (which is transparent to the executing program) is:

* The count in the BRT entry is checked. If the count indicates that data is expected, 15 words (or less if the count is less) are read into memoryat the location specified. The transfer address and count are then updated.

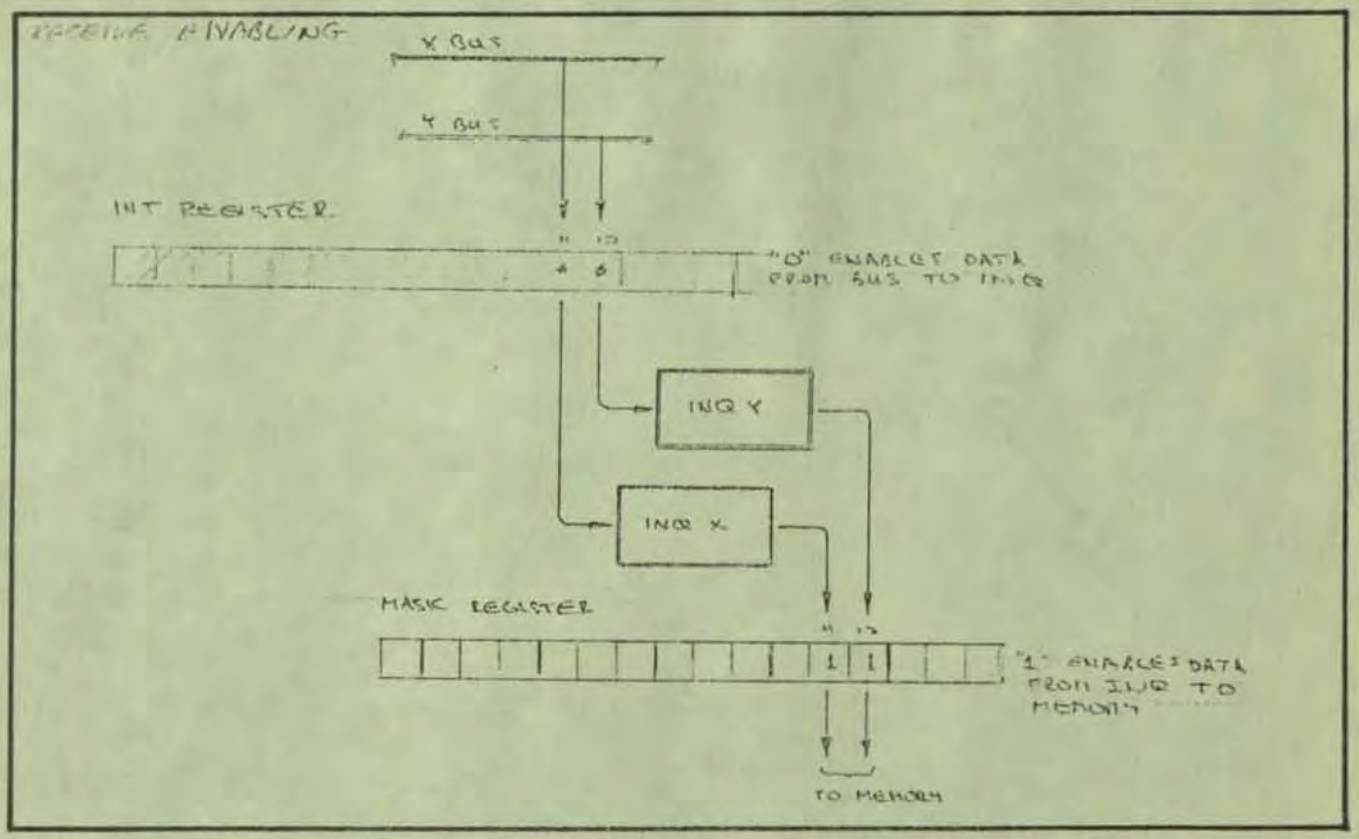
* The checksum of the packet is checked. If the checksum is valid andthe count still exceeds zero, the INQ is marked empty (permitting further transmissions to take place) and the normal instruction execution sequence continues.



* If the count is zero or a checksum error is detected, the INT register bit associated with the bus over which the transmission took place is set, and an interrupt occurs. In the case of a transmission error, the countword is set to -%400 before the actual interrupt occurs. When a normal receive completes, the count word will contain a value between -14 and 0.After an interrupt, no further transfers to the processor over thebus which caused the interrupt are permitted until the bus is clearedwith an RIR instruction.

INT and MASK Registers

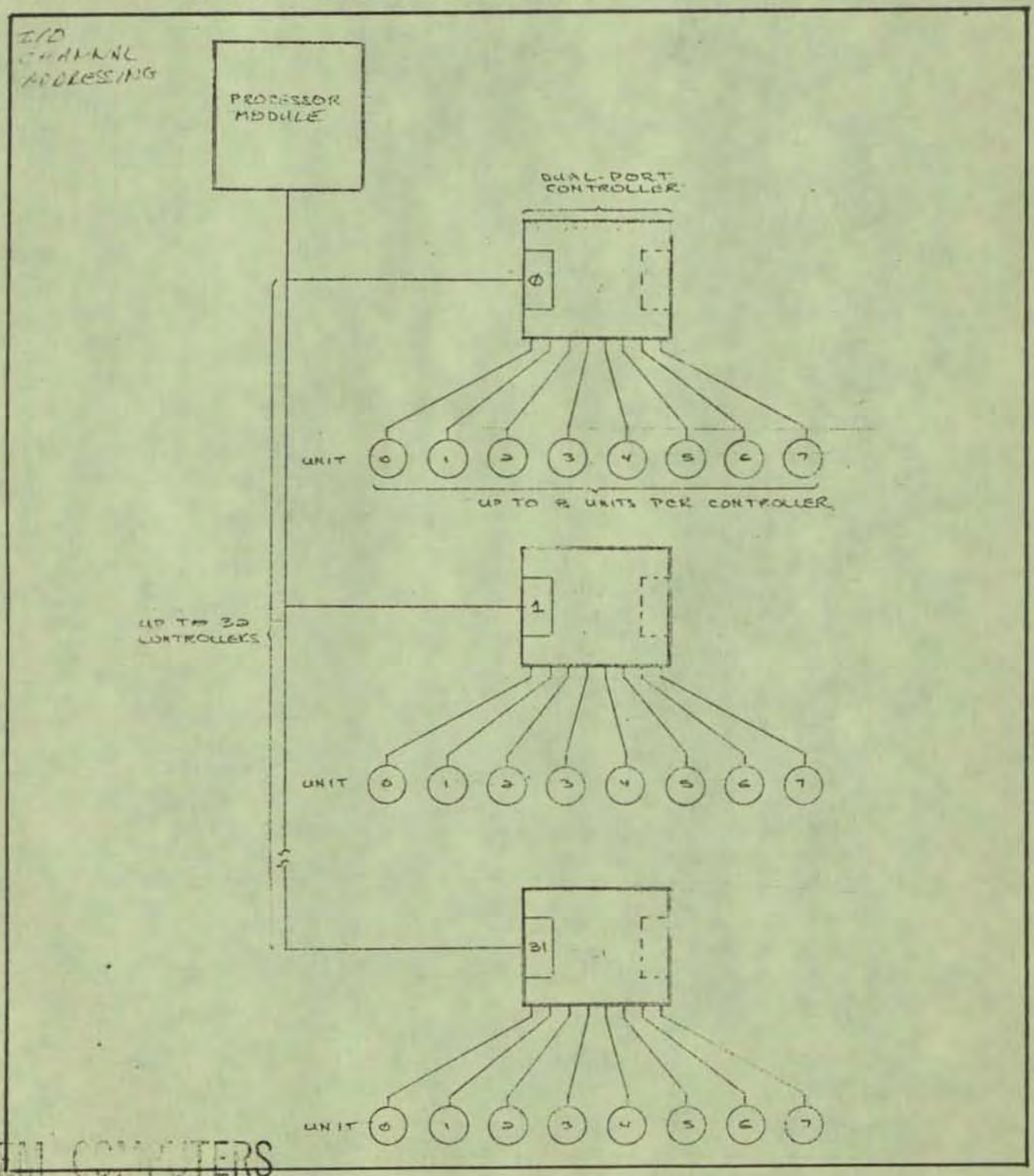
These registers have a direct bearing on the ability of a processor module to accept data over an interprocessor bus. A shown in the following illustration, data can only be input to an INQ if the correspond bit in the INT register is set to "0"; data can only be transferred from an INQ into memory if the corresponding bit in the MASK register is a "1".



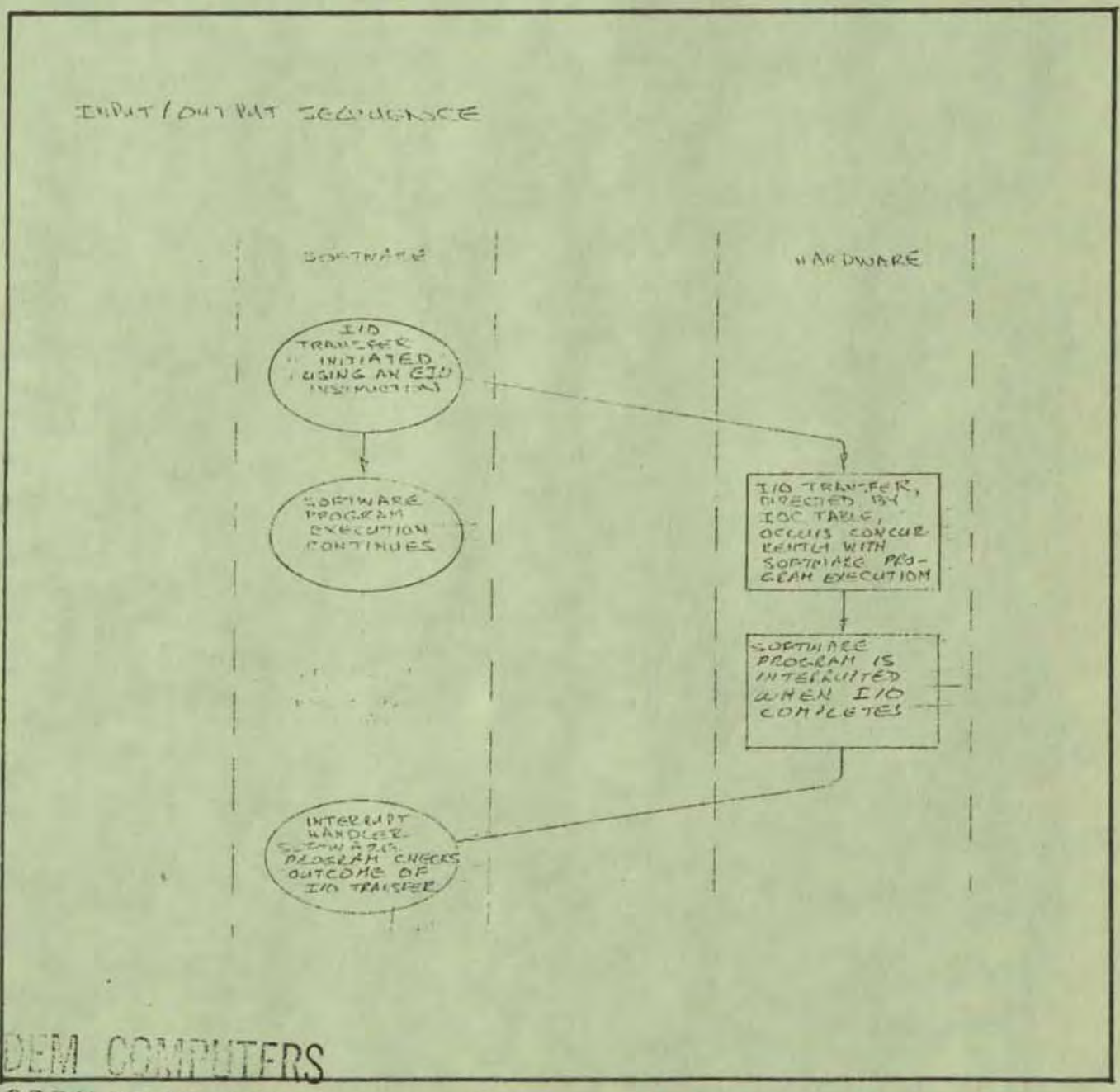
This also affects the indication a source processor module receives when a packet timeout occurs: If the INT register bit in the destination module is a "1", no information transfer has taken place. If the.INT register bit in the destination module is a "0" and the MASK register bit is also a "0", one packet (i.e., 15 words) of information is transferred before the timeout occurs.

INPUT/OUTPUT CHANNEL

Each processor module has a single block-mutliplexed input/output channel through which all input/output takes place. Up to thirty-two device controllers may be connected to an i/o channel; each device controller may control up to eight units.



The operating system performs input/output operations by configuring an entry in a system table called the I/O Control table (IOC). The IOC contains 256 entries; one entry for each device that can possibly communicate over the i/o channel. Each entry contains the address where the data transfer takes place and a count of the number of bytes to be transferred. Once the entry corresponding to the device is configured, an EIO (execute i/o) instruction is executed to initiate the i/o transfer; the actual data transfer is performed concurrently with program execution. When the transfer completes, an interrupt to an operating system interrupt handler takes place. In the interrupt handler, an IIO (interrogate i/o) instruction is executed to check the outcome of the operation.



I/O Control Table (IOC)

The data which is transferred between memory and a specific unit is determined by an entry in the I/O Control Table (IOC). This table resides at a fixed location of the system data area and contains a two-word entry for every possible unit which may be defined. Specifically, the two-word entry for controller c, unit u begins at location:

SG[%1000 + %20c +2u]

The first word contains the data transfer address. This is a word address within the system data area.

The second word contains the following information:

* <p> word[1].<0> is the protect bit. When set to "1", only output transfers are permitted to this unit.

* <ch err> Word[1].<1:3> is set with the channel error code at the end of an operation. The possible values are:

  0 -- no error
  1 -- protect violation
  2 -- pad in violation
  3 -- data parity violation
  4 -- timeout
  5 -- map absent bit detected
  6 -- map parity error
  7 -- uncorrectable memory error

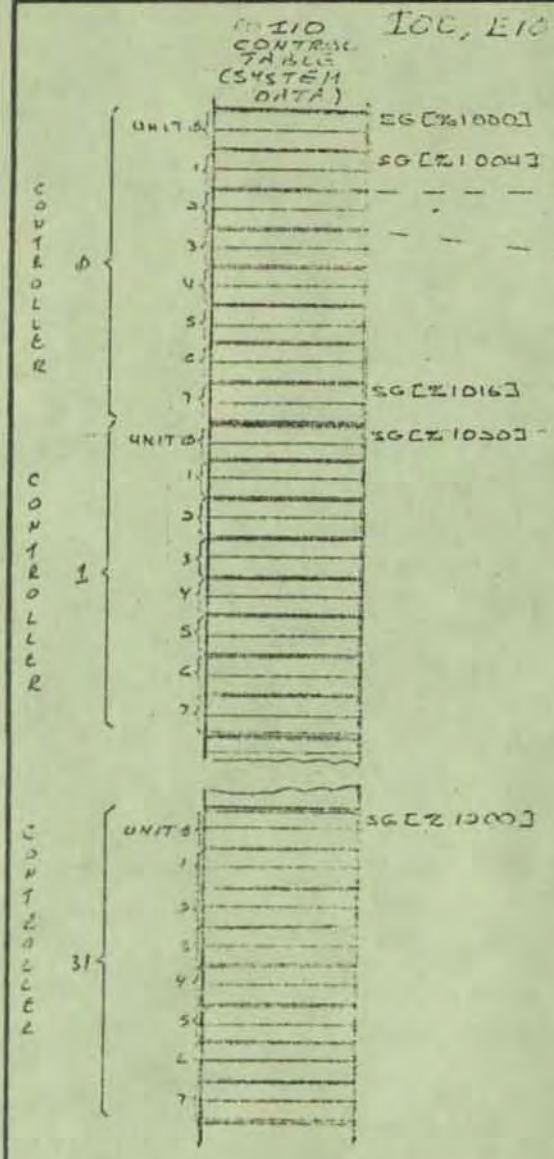
  In case of a simultaneous occurrence of multiple errors, the highest numbered error will be designated.

* <byte count> Word[1].<4:15> contain the number of bytes to be transferred in the current operation.

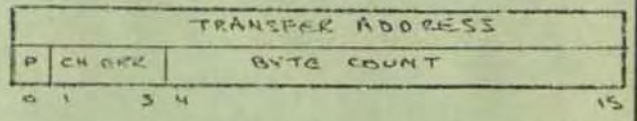
To prevent erroneous data transfers, the operating system sets the second word in IOC entry to zero where transfers are not expected. If a device should attempt to transfer data and the <byte count> is zero, the i/o channel will abort the device, causing an interrupt to occur. In such a case, the <ch err> remains zero, but the status returned by the device as a result of an IIO or HIIO reflects the abort.

EIO Instruction

To perform an I/O operation, the IOC entry for the unit must first be correctly initialized. An EIO instruction is then executed, specifying the controller, unit, command, and other parameter information. These are placed in B and A of the register stack.

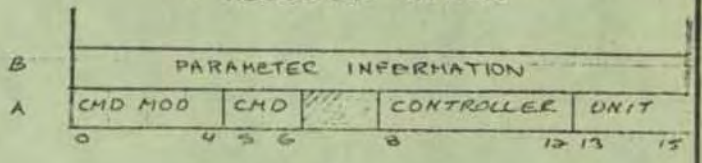


IOC TABLE ENTRY
SG[%1000 + %20C + 2u]



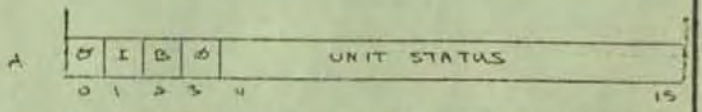
P = PROTECT BIT (1 = OUTPUT ONLY)
CH ERR = CHANNEL ERROR
0 = NO ERROR
1 = PROTECT VIOLATION
2 = PAD IN VIOLATION
3 = DATA PARITY VIOLATION
4 = TIMEOUT
5 = MAP ABSENT BIT DETECTED
6 = MAP PARITY ERROR
7 = UNCORRECTABLE MEMORY ERROR

EIO PARAMETERS IN R1 REGISTER STACK



CMD = COMMAND (A.<5:6>)
0 = SENSE
1 = WRITE
2 = READ
3 = CONTROL
CMD MOD = COMMAND MODIFIER (A.<0:4>) IS DEVICE DEPENDENT EXCEPT:
0 = COLD LOAD
%17 = TAKE OWNERSHIP & CLEAR DEVICE

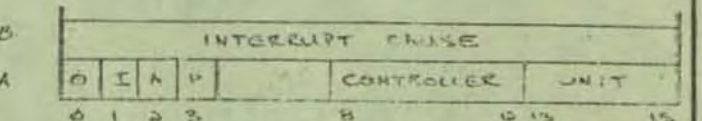
DEVICE STATUS RETURNED IN REGISTER STACK FROM EIO



O = OWNERSHIP (1 = OWNED BY OTHER PORT)
I = INTERRUPT PENDING (1 = DEVICE IS SIGNALLING INTERRUPT)
B = BUSY DEVICE

EIO CONDITION CODES
CCL = CHANNEL ERROR: A = = -1
CCE = DEVICE STATUS.<0:3> = 0
CCG = DEVICE STATUS.<0:3> <> 0

STATUS RETURNED IN REGISTER STACK FROM IIO



O & I ARE AS DESCRIBED ABOVE
A = DATA TRANSFER ABORTED
P = PARITY ERROR

IIO CONDITION CODES
CCL = CHANNEL ERROR DURING IIO
CCE = STATUS BITS.<0:3> = 0
CCG = STATUS BITS.<0:3> <> 0

The parameters to the EIO instruction have the following meaning:

* <parameter information> is a device dependent parameter that is sent to the specified device.

* Command bits A.<0:5> specify the command the device is to perform. The <cmd> bits, A.<4:5>, specify the general type of command:

  0 -- sense
  1 -- write
  2 -- read
  3 -- control

  <cmd mod> bits, A.<0:3>, modify the command, alllowing up to 16 device dependent commands.

  Three configurations of the field A.<0:5> are reserved:

  %02 instructs the controller/unit to perform a cold load operation

  %77 instructs the controller to set its ownership bit to the channel issuing the command. The controller logic is cleared

  %73 instructs the controller to set is port disbale bit and cease responding to the channel issuing the command.

* <controller no.> A.<8:12> -- specifies one of 32 controllers.

* <unit no.> A.<13:15> -- specifies one of 8 units.

The EIO instruction replaces the two parameter words by a single word containing the device status, and sets the condition code according to the outcome of the instruction. The condition code settings are as follows:

CCL: Channel error (service in, timeout, or parity). A is set to -1.
CCE: Device status bits <0:3> = 0.
CCG: Device status bits <0:3> <> 0.

The Device Status bits returned have the following meaning:

* 0 <ownership> A.<0> is a "1" if the device is owned by other port. No data will be transferred.

* I <interrupt pending> A.<1> if a "1" indicates the device is signalling interrupt. No data will be transferred.

* B <busy device> A.<2> indicates that the device is already executing an i/o transfer. No data will be transferred because of this EIO.

* P <parity error> A.<3> if a "1" indicates that the controller detected a parity error during execution of the EIO instruction. No data will be transferred.

IIO and HIIO Instructions

Following an EIO which initiates an operation, an interrupt occurs when the operation completes. At this point, an IIO instruction (or HIIO if the interrupt was a high-priority I/O interrupt) must be executed to determine the cause of the interrupt. When the IIO or HIIO is executed, the highest priority device with an interrupt pending returns it address, and status pertaining to the interrupt.

* O <ownership> A.<0>, if a "1" indicates that the controller is owned by the alternate port.

* I <interupt pending> A.<1>, if a "1" indicates that the device has an interrupt pending (this should be "0").

* A <aborted> A.<2>, if a "1" indicates that the data transfer was aborted.

* P <parity error>, A.<3>, if a "1" indicates that a parity error was detected during the data transfer sequence.

* <controller>, A.<8:12>, is the controller number associated with the interrupt.

* <unit>, A.<13:15>, is the specific unit associated with the interrupt.

* PON <power on>, B.<0>, if a "1" indicates that power was just restored to the controller and the controller logic is in a reset or arbitrary state. The channel deasserting its "RESET" signal first is the one interrupted.

* <status>, B.<1:15>, is related to the particular controller/unit that is interrupting.

Following execution of an IIO or an HIIO instruction, the condition code is set as follows:

CCL: Channel Error (parity error or timeout)
CCE: No error
CCG: A.<0:3> <> 0 (error)

Input/Output Sequence

A typical data transfer sequence over the input/output channel is illustrated in the following illustration. The sequence is as follows:

1. Instructions are executed to configure the IOC entry for the controller/unit combination where the transfer is to take place. In this case, the IOC entry is at SG[%1034] for controller 1, unit 6.

2. The EIO parameters are loaded onto the register stack.

3. An EIO instruction is executed. The parameter information is sent to controller 1, unit 6.

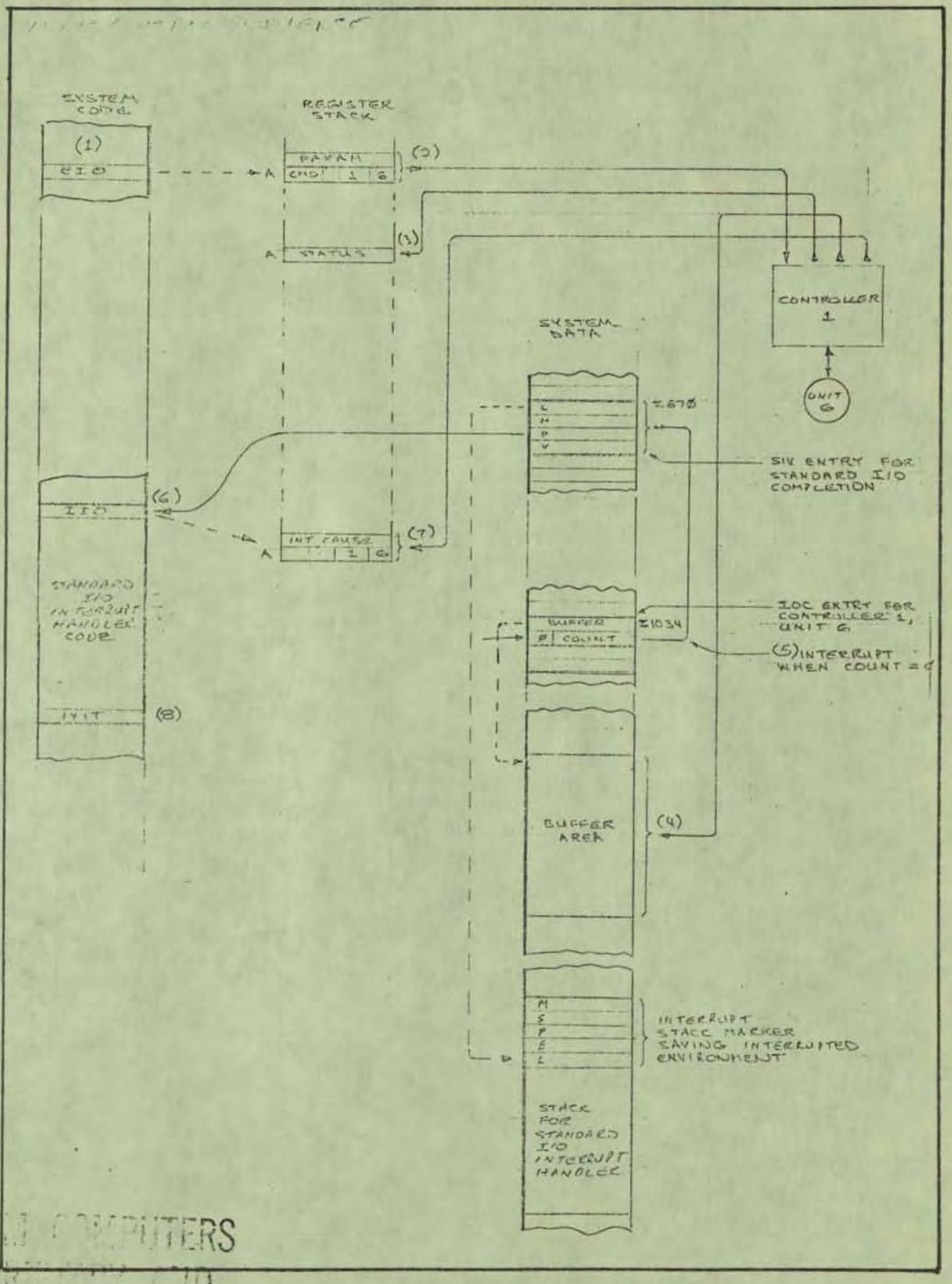
4. To indicate its outcome, the EIO instruction returns a status word to the top of the register stack and sets the condition code. These are checked by subsequent instructions.

5. Meanwhile, the data transfer takes place. Data is transferred from unit 6 to the location in memory indicated by the IOC entry for controller 1, unit 6. As the data is transferred into memory, the transfer address and count word in the IOC are updated accordingly.

6. When the count word in the IOC reaches zero, indicating that the transfer is completed, the INT.<13> bit in the interrupt register is set to "1" to signal interrupt pending. If the corresponding bit in the MASK register is set, an interrupt through the SIV entry for Standard I/O (at SG[%670]) occurs. The MASKi entry in the SIV causes any further standard I/O interrupts to be deferred while the interrupt handler is active.

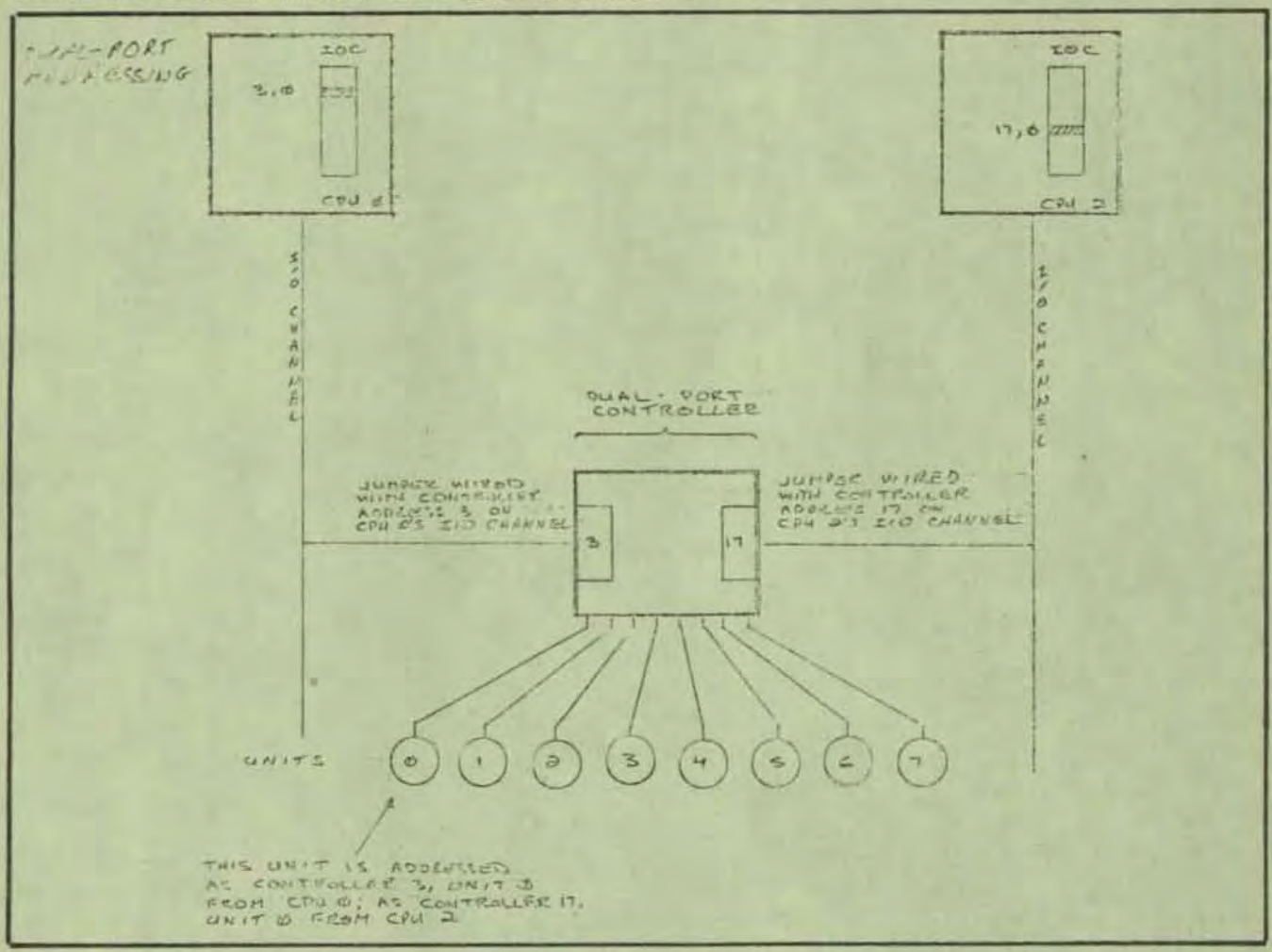
6. The interrupt handler executes an IIO instruction. Executing IIO signals the highest priority interrupting controller to stop interrupting and returns two word of status information to the top of the register stack. The status word contains the controller/unit number of the interrupting device as well as channel error status and device dependent status information.

7. When the interupt handler for standard i/o completes, an IXIT instruction is executed. IXIT restores the previous MASK register value (which may allow another standard i/o interrupt) and returns to the interrupted program.

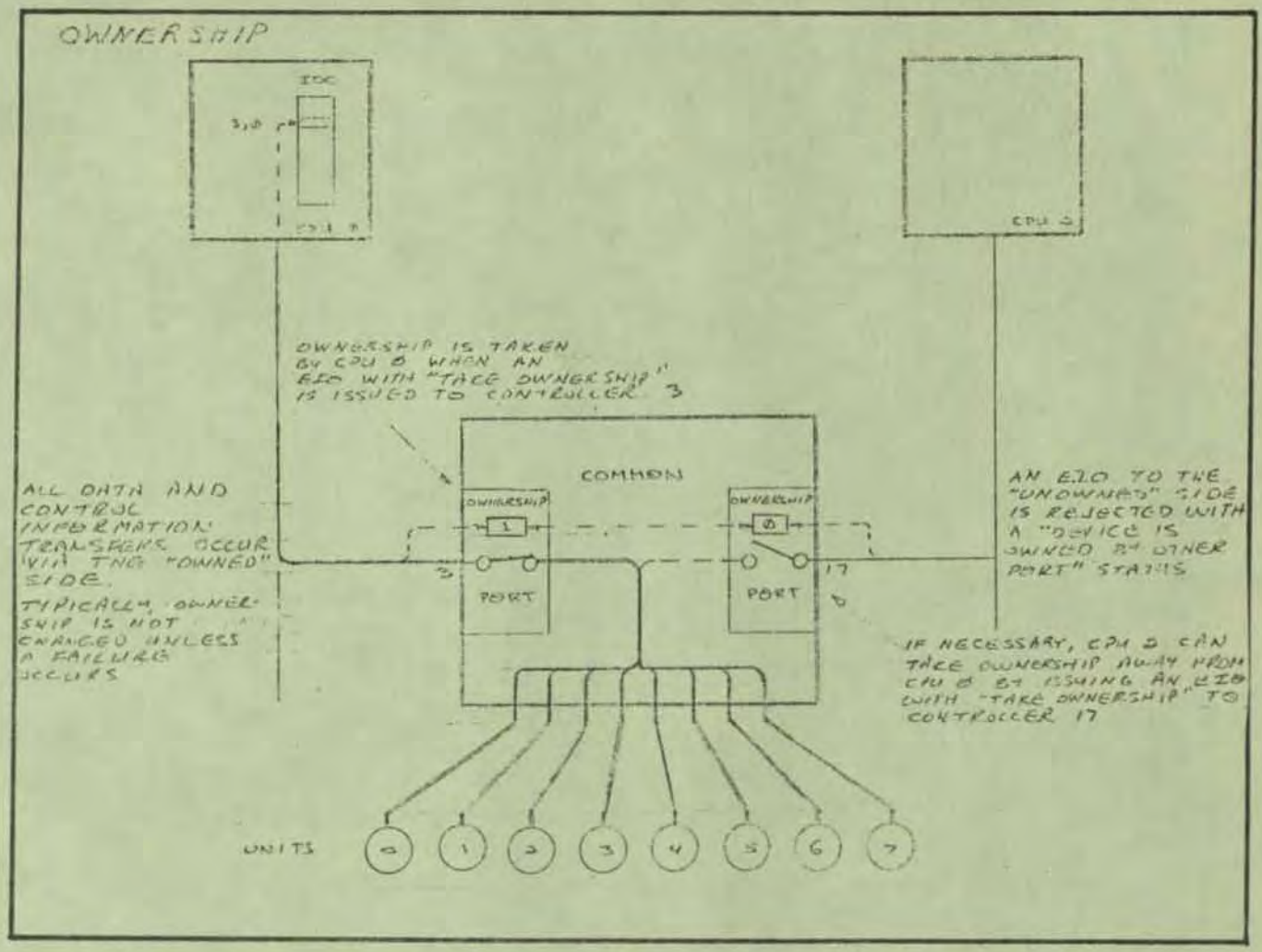


Dual-Port Controllers and Ownership

Each controller in the Tandem 16 computer system is connected to the input/output channels of two processor modules. This provides redundant communication paths to i/o devices. As shown in the following illustration, this means that a single controller (and associated units) have entries in the IOC's of two processor modules. Note also that it is actually the port that is addressed when communicating with a controller, so that both ports need not have the same channel address on both channels.



Although each controller has two ports and is fully capable of communicating through either i/o channel, only one channel is used during normal operation (the other channel, as far as a particular controller is concerned is not used). The i/o channel through which communication to a particular controller occurs is said to "own" the controller. ALL input/output transfers (i.e., control and data) occur through the channel owning the controller.



Each of the two ports in a controller contains a flag bit known as the "ownership" bit. The state of these bits determine which channel the controller will accept commands from. Initially (i.e., power on), a controller comes up in a arbitrary state: the ownership bit is pointing to either channel. An operating system configuration parameter specifies which channel is to be the primary channel for communication to a particular controller. The operating system program that controls a particular controller executes an EIO instruction to take owenership over the configured primary i/o channel.

The operating system transfers data only through the "owned" side: an attempt to communicate through the unowned side results the EIO instruction being rejected with an "ownership" error. If, during the course of a data transfer, the primary path to the controller (i.e., the primary processor module, channel, or port) becomes inoperable, the operating system program controlling the controller executes an EIO instruction over the alternate (backup) channel to take ownership. The "ownership" bits in the controller switch over to point to the other i/o channel. All subsequent data transfers now occur through this channel.

Each port also has a "disable" bit that is separate from the ownership bit. The disable bit, if a "1", prevents a controller from transmitting information through that port onto an i/o channel. The disable bit is set by an EIO with "set disable" issued to a controller. Normally, this is used by the operating system when a controller performs some unexpected action that could affect the entire channel. The disable bit is associated with a port so, if the malfunction is in one port, normal communication with the controller may still occur via the other port.

Interrupts

A controller signals an interrupt on the i/o channel when the associated transfer has completed. A controller also interrupts if it is necessary to prematurely terminate a transfer. And the channel may also interrupt on behalf of a controller if it detects a malfunction (such as a timeout).

When simultaneous interrupts occur on an i/o channel, a priority scheme determines which interrupt is handled first. There are two levels of priorities designated "rank1" and "rank2". Each rank has up to 16 controller assigned to it. Jumper wires on a controller determine the rank assigned to the controller and the relative position within a rank (positions 0:15). The interrupting controller with the lowest rank (1 or 2) and the lowest relative position within that rank is handled first.

A controller continues to interrupt until cleared. Normally, this clearing is done via an IIO or HIIO instruction.

High-Priority I/O

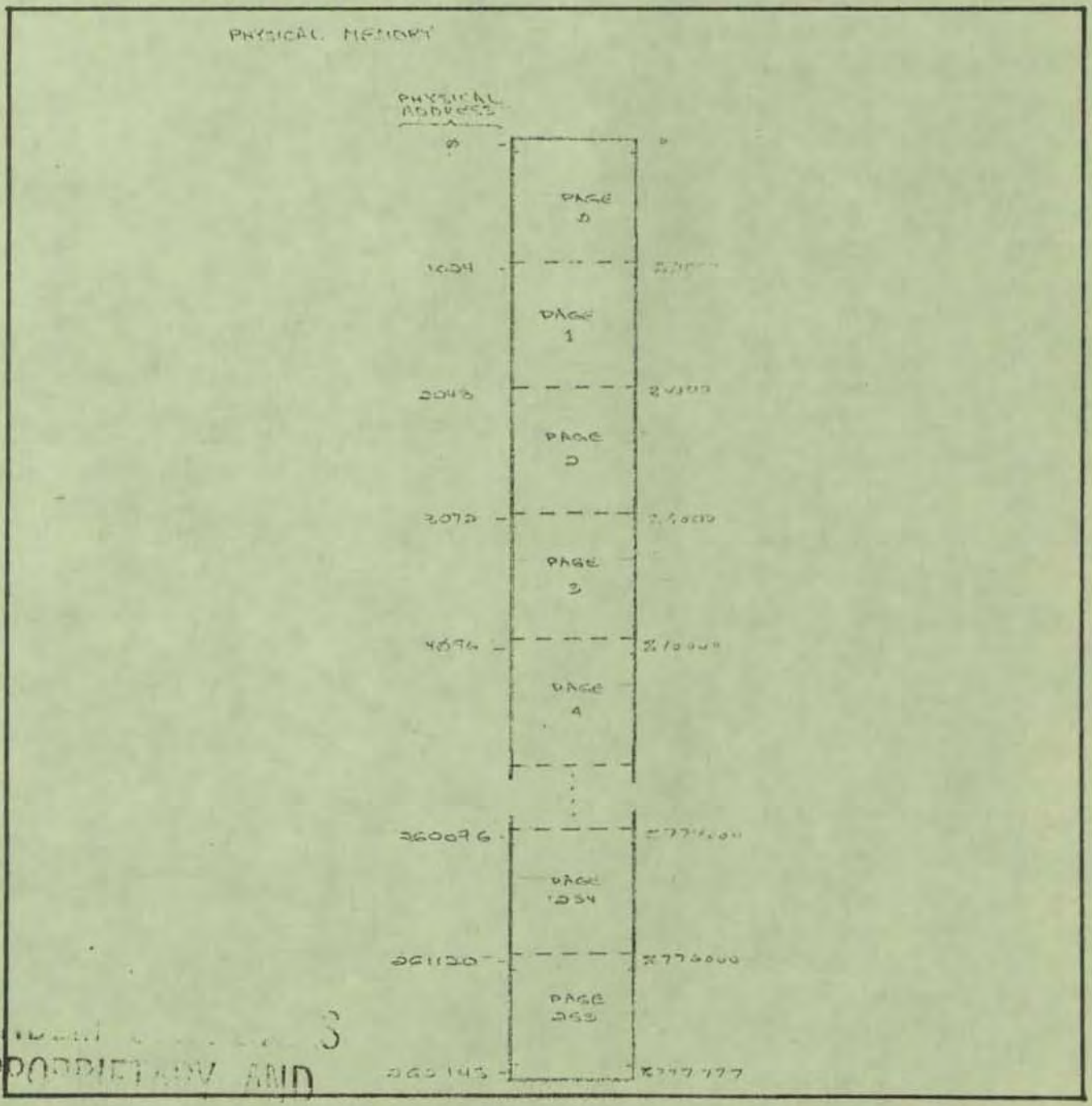
Two levels of interrupt are available on an i/o channel: standard i/o and high-priority i/o. Standard i/o is characterized by controller that interrupts through the SIV entry for standard i/o. Likewise, high-priority i/o is characterized by controllers that interrupt through the SIV entry for high-priority i/o. Whether a controller interrupts with standard or high-priority is determined by a jumper connection on a controller (all controllers under control of the Tandem 16 operating system interrupt as standard i/o).

High priority i/o is used by applications requiring an ultra-fast response time (as in a real-time environment). The operating system never masks off the high-priority interrupt position, thereby ensuring that no matter what is executing in a processor module, a high-priority interrupt will be recognized instantly.

PHYSICAL MEMORY AND MAPPING

A processor module's physical memory consists of up to 262,144 words of 16 data bits each. In addition to the 16 data bits, each word in memory has an additional parity bit (if a core memory) or six additional error correction bits.

Physical memory is logically divided into contiguous blocks of 1024 words each called pages. Pages in physical memory are numbered consecutively frompage 0, starting at physical location zero. The address range of physical memory, which is 0 through 262,143, requires 18 bits of address information.



All references to memory are made to one of the four logical address areas: user data, system data, user code, and system code. The range of addressing in any given logical address area is that of the 16-bit logical address, 0 through 65,535.

A processor module converts the 16-bit logical address to an 18-bit physical address through use of the map registers. There are four map registers, each map register corresponds to one of the four logical addressing areas.

The four maps are designated:

* Map 0 -- User data map: All data references use this map when the DS bit of the E register is "0" except when an explicit system data reference is made (SG addressing mode).

* Map 1 -- System data map: All data references use this map when the DS bit of the E register is "1" or when an explicit system data reference is made (provided that the PRIV bit is set). In addition, all memory references by either the I/O channel, the bus handling microcode, or the interrupt handling microcode specifies this map.

* Map 2 -- User code map: This map defines the active user program. All code area references use this map if the CS bit in the E register is "0".

* Map 3 - System code map: This map defines the operating system. All code area references use this map if the CS bit in the E register is "1".

Each map has 64 entries corresponding to the 64 pages possible in each logical addressing area. Each entry contains the following information:

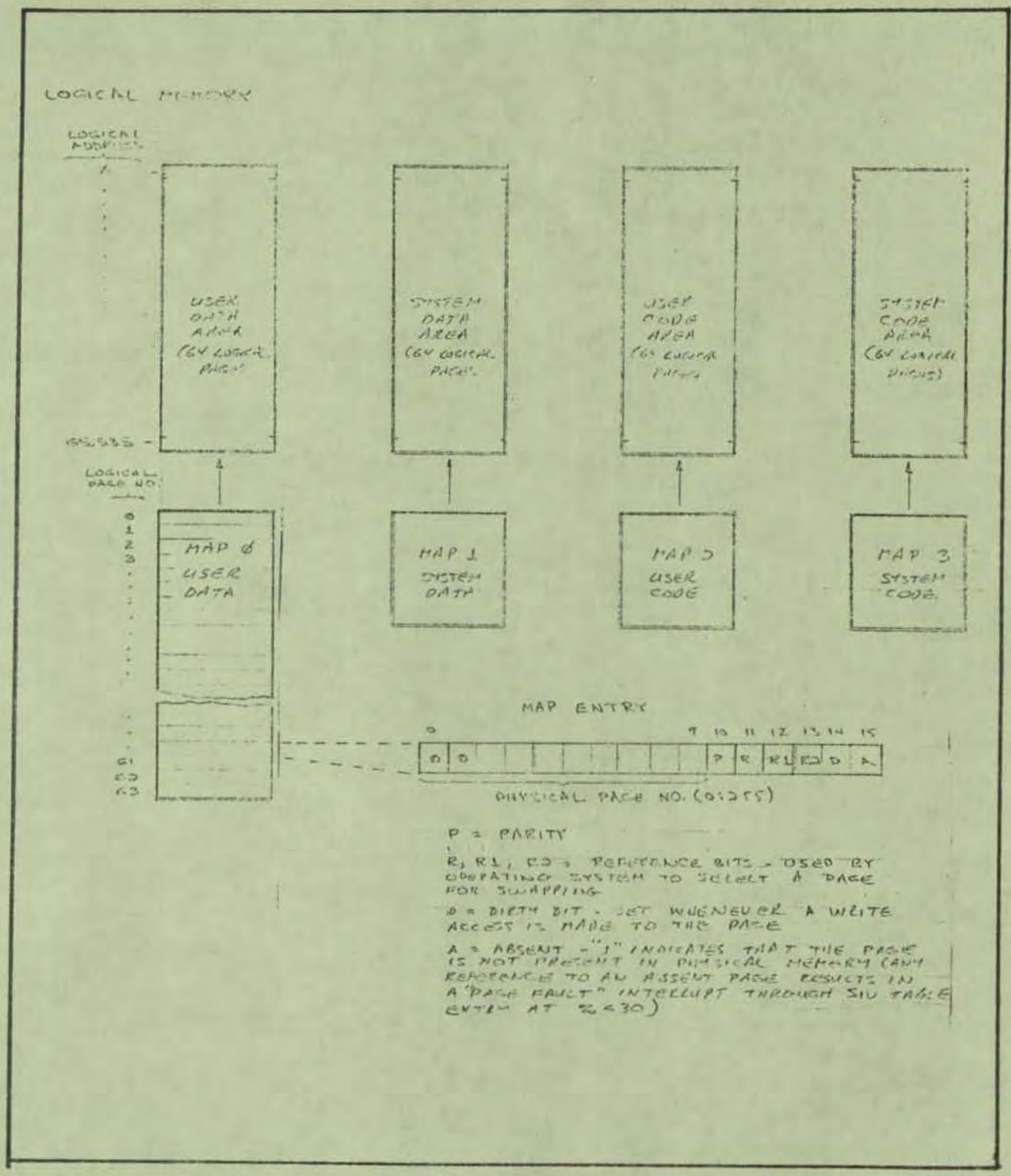
* map[logical page].<0:9> contains the physical page number (0 through 255).

* P: map[logical page].<10> contains the odd parity bit for the map entry. The parity bit is generated by the hardware when the map entry is made.

* R, R′, R′′: map[logical page].<11:13> are the reference bits. Thesebits and the dirty bit) are used by the memory manager function of the operating system to help select a page for overlay.

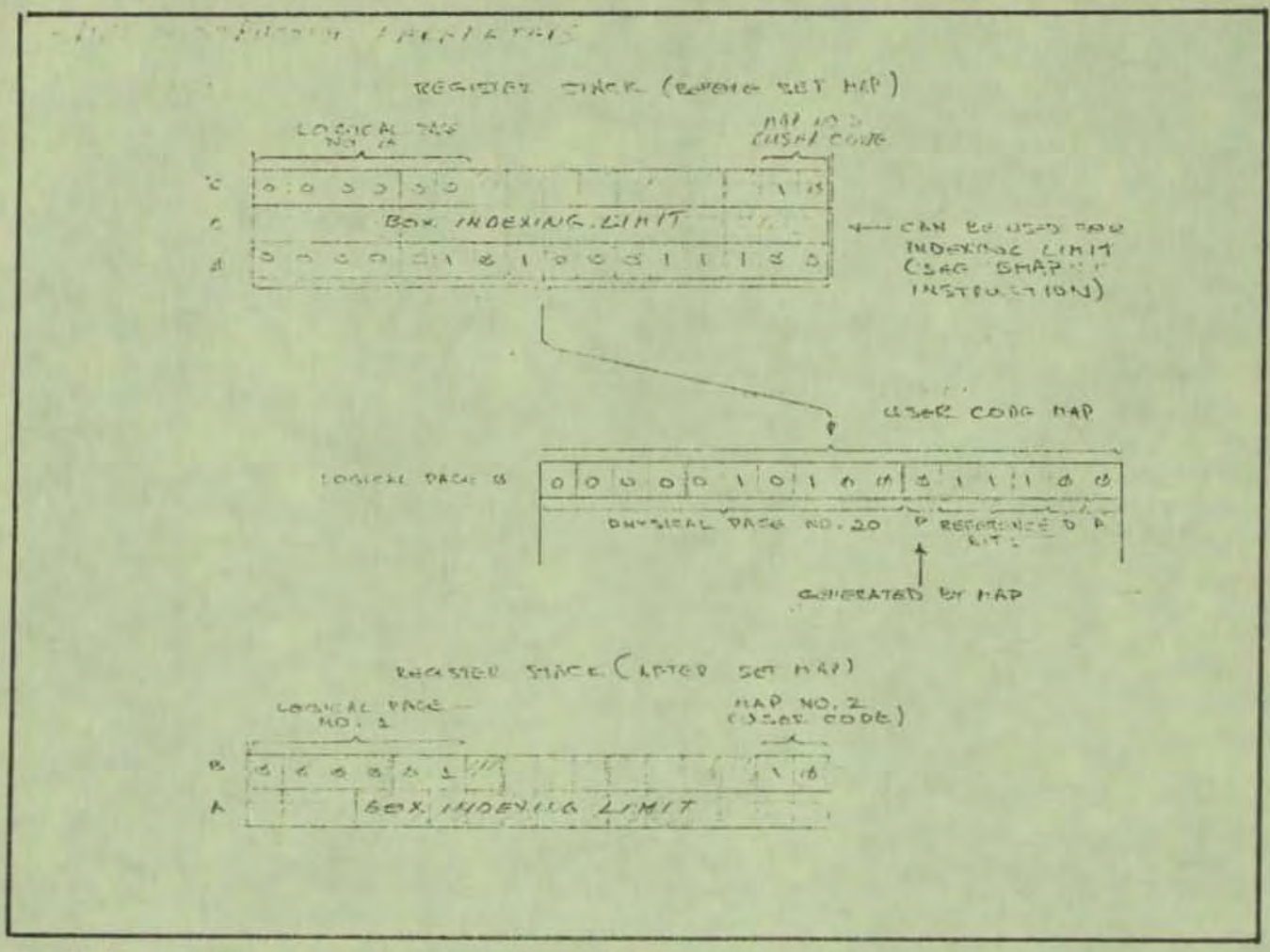
* D: map[logical page].<14> is the dirty bit. The dirty bit is set toa "1" when a write access is made to the corresponding memory page.

* A: map[logical page].<15> is the absent bit. The absent bit is initially set to a "1" by the operating system to flag a page as being absent from main memory. An access to a page with this bit set to "1" causes an interrupt to the operating system page fault interrupt handler (SIV entry at SG[%630]).

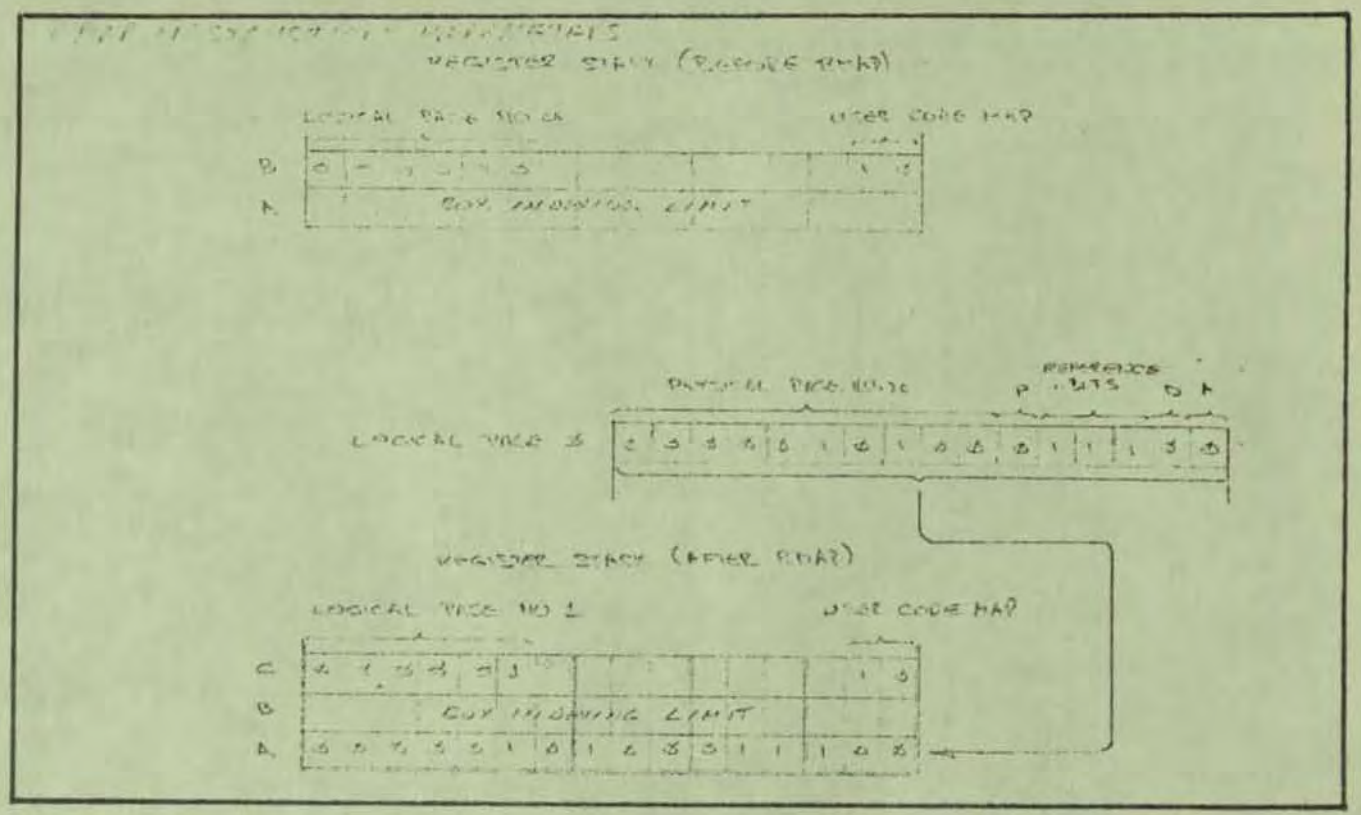


Three instructions are used in connection with the map registers:

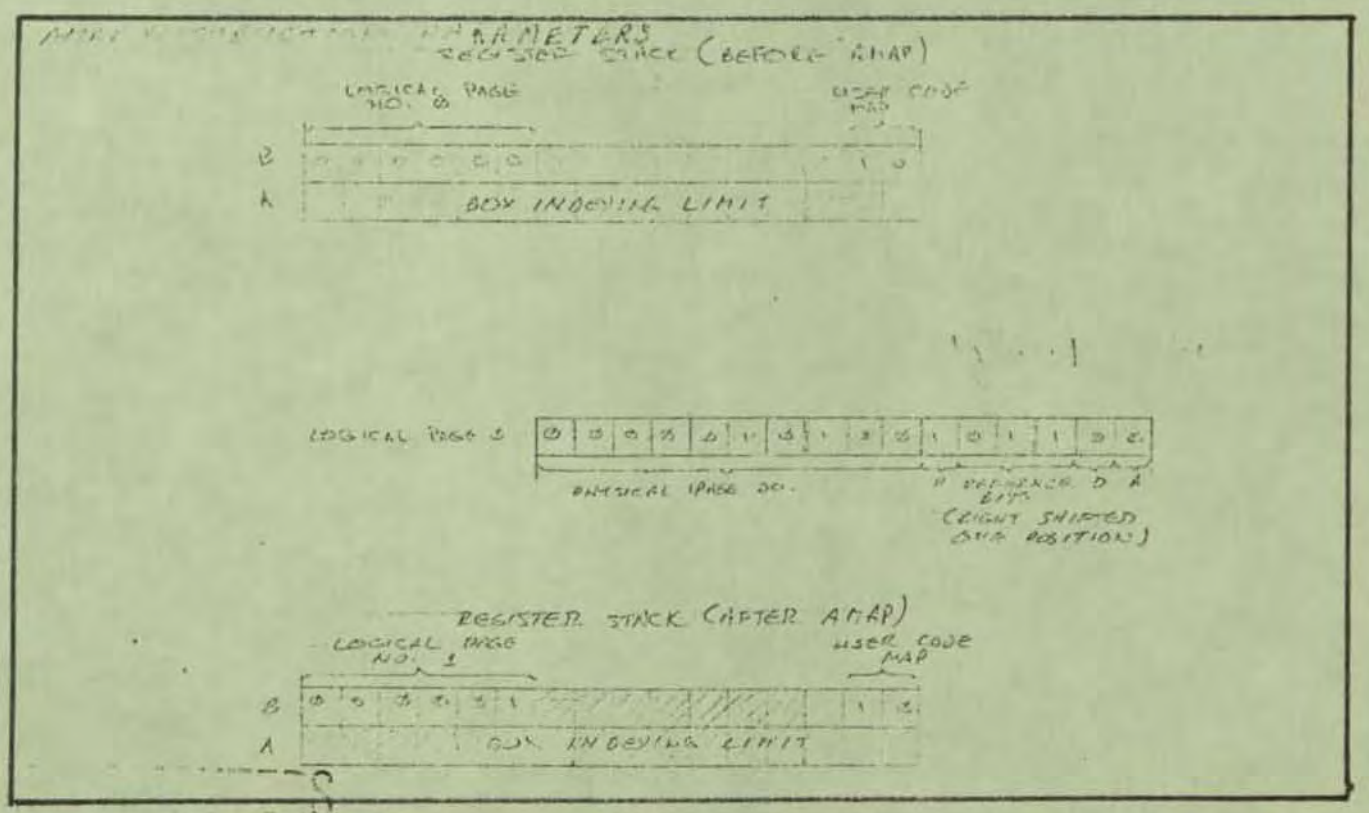
* SMAP (set map entry) -- This instruction is used by the operating system when initializing a map entry. SMAP requires two parameters in the register stack as shown below



* RMAP (read map entry) -- This instruction is used by the operating system when reading a map entry. RMAP requires one parameter and returns the designated map entry to the top of the register stack.

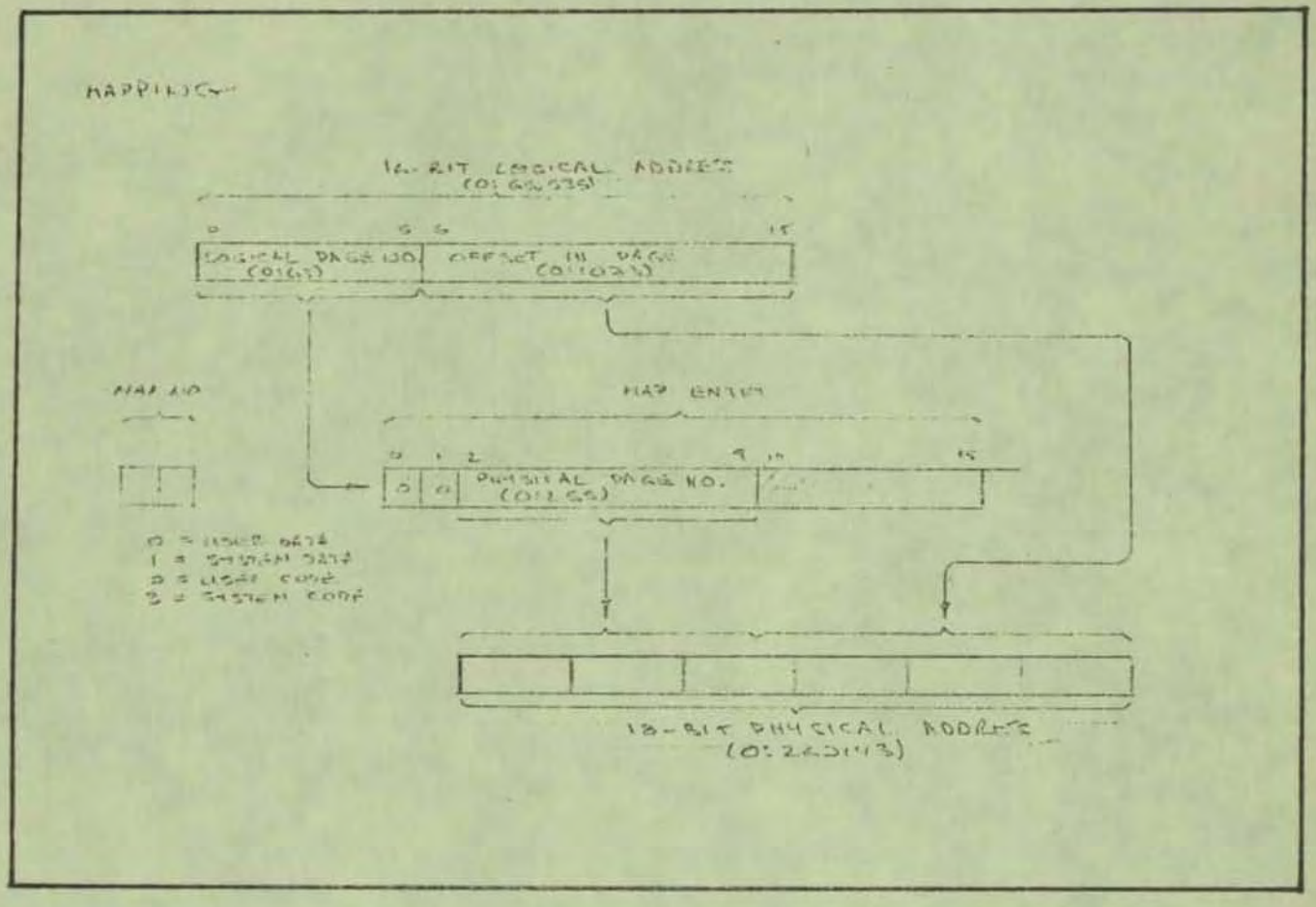


* AMAP (age a map entry) — This instruction causes the reference bits (R, R′, R″) to be shifted one position to the right. This is used by the operating system as an aid in selecting a page for overlay. AMAP requires one parameter in the register stack.



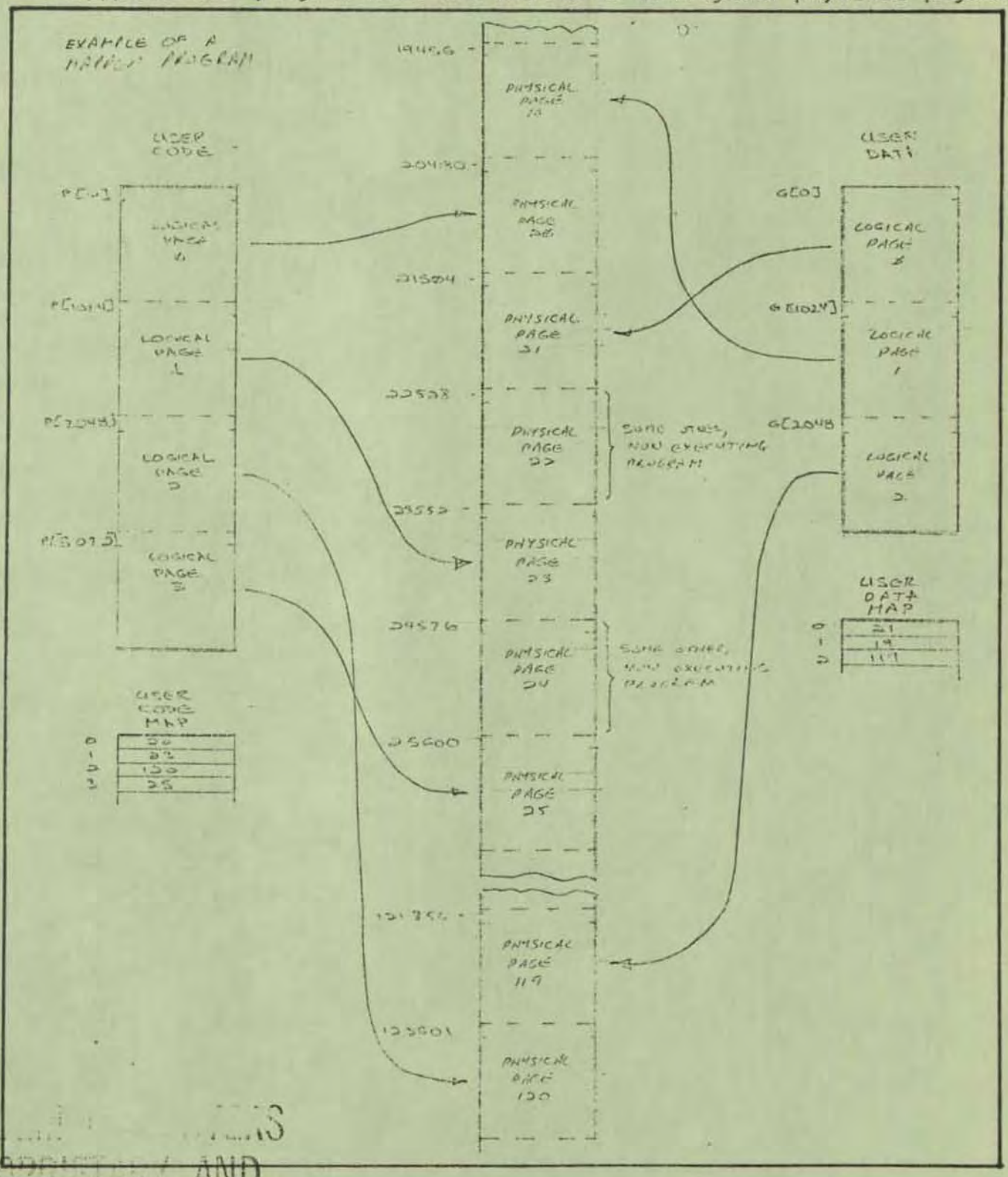
Mapping

For a given reference to a logical area, a 16-bit logical address is known. This converts to an 18-bit physical address as shown:

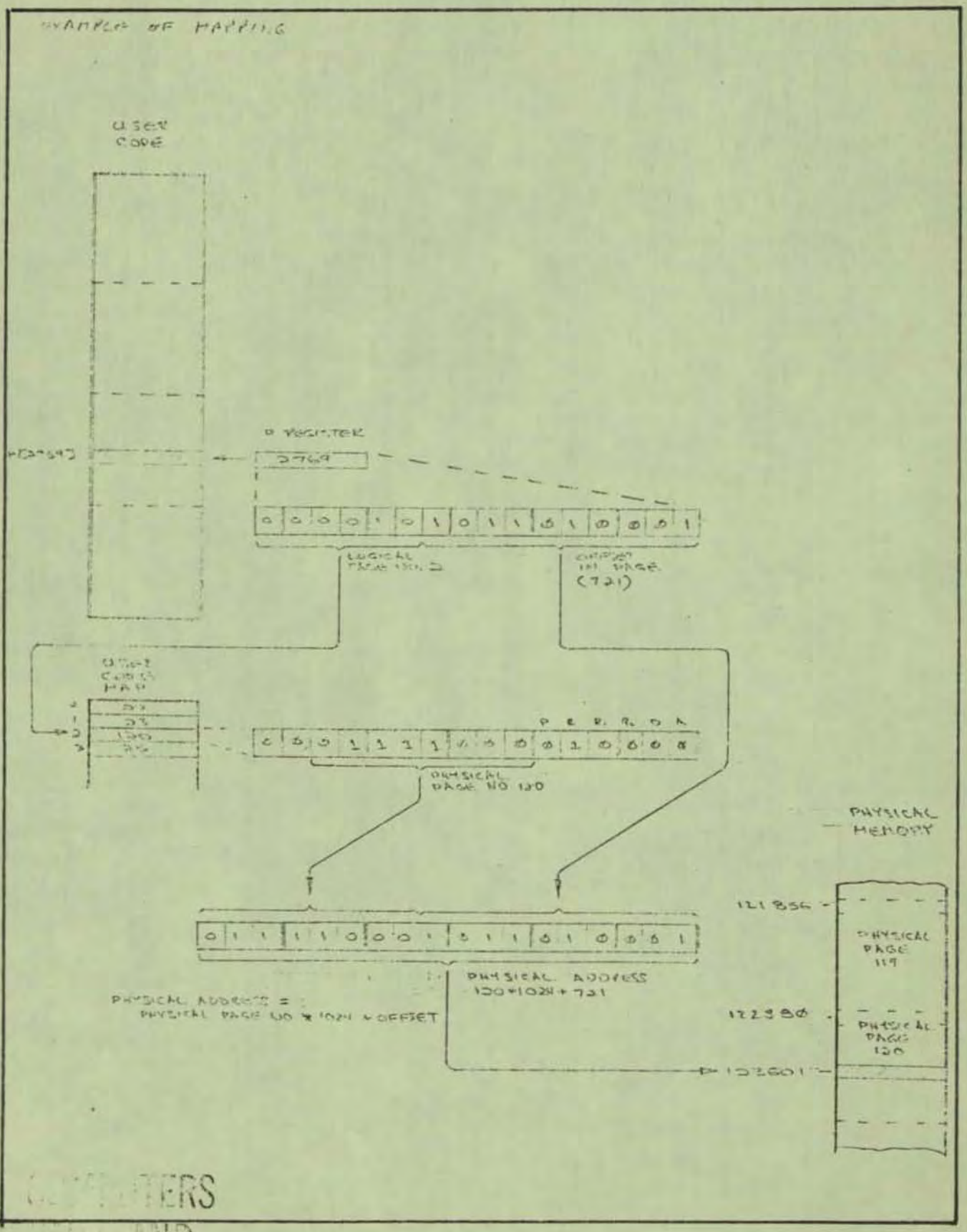


A Mapped Program

The following illustration shows an example of a mapped application program. Note, especially, that the various logical pages comprising the appplication program need not reside in contiguous physical pages.



The following illustration shows an example of mapping.



Page Fault

A page fault occurs when a reference is made to a page that does not currently reside in main memory. When a page fault is detected, an interrupt through to the operating system page fault handelr occurs.

The page fault interrupt sequence (as shown in the following illustration) is:

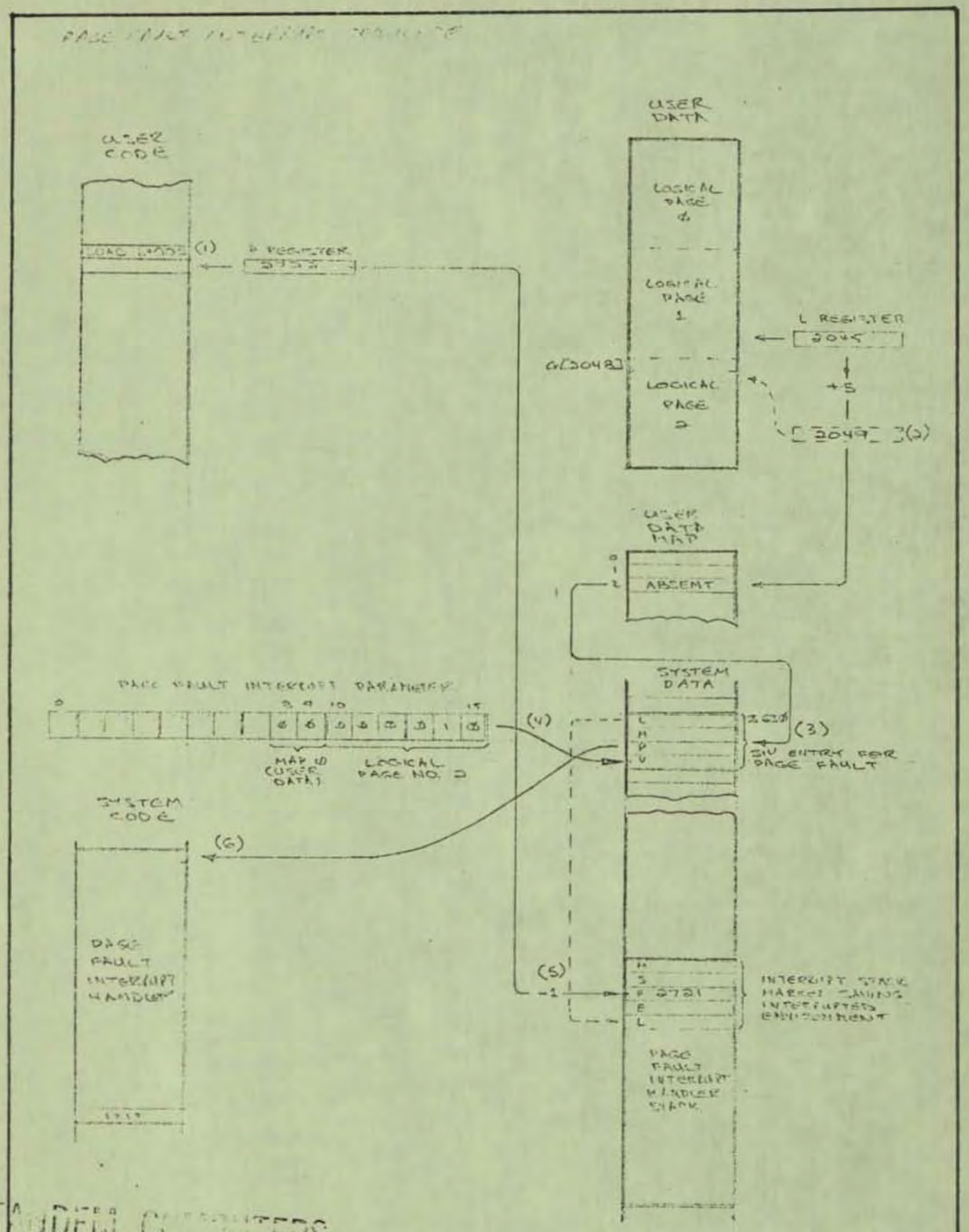
1. An address reference is made to a page that is absent from physical memory (A = "|").

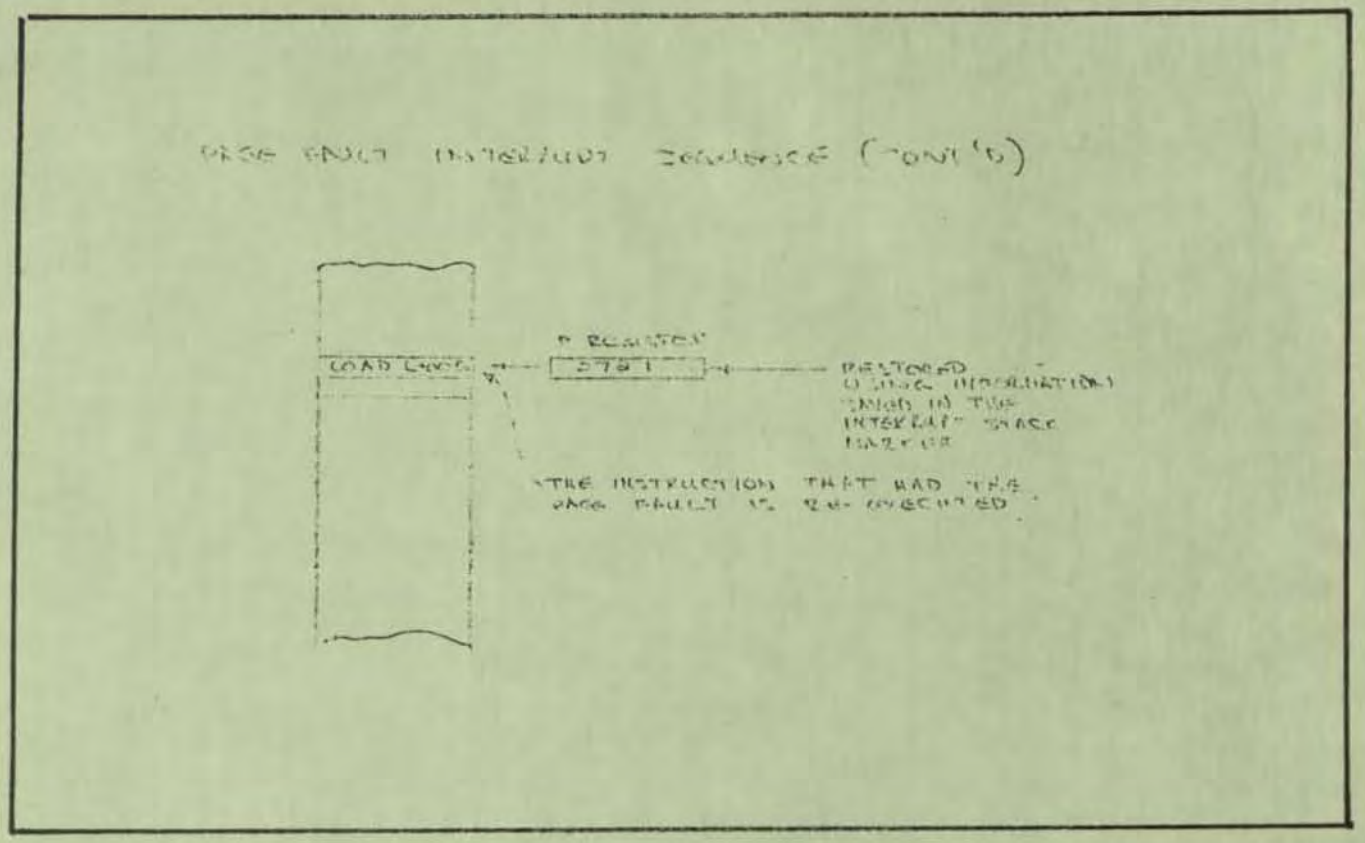
2. An interrupt through the SIV entry at SG[%630] occurs. The hardware places an interrupt parameter indicating the map number and the logical page number in the Vi location in the SIV entry. One is subtracted from the current P Register setting (so that the faulted instruction will be repeated) then the current environment is saved in the interrupt stack marker.

   The MASKi word in the SIV entry for page fault is a "0" preventing another page fault interrupt from occurring while the current page fault is being processed.

3. The page fault interrupt handler executes. An operating system program executes to read the absent page from disc. When the interrupt handler completes, an IXIT instruction is executed. The environment that had the page fault is restored.

4. Because the P setting minus one was saved in the interrupt stack marker, the instruction previously causing the page fault is re-executed.





Reference Bits

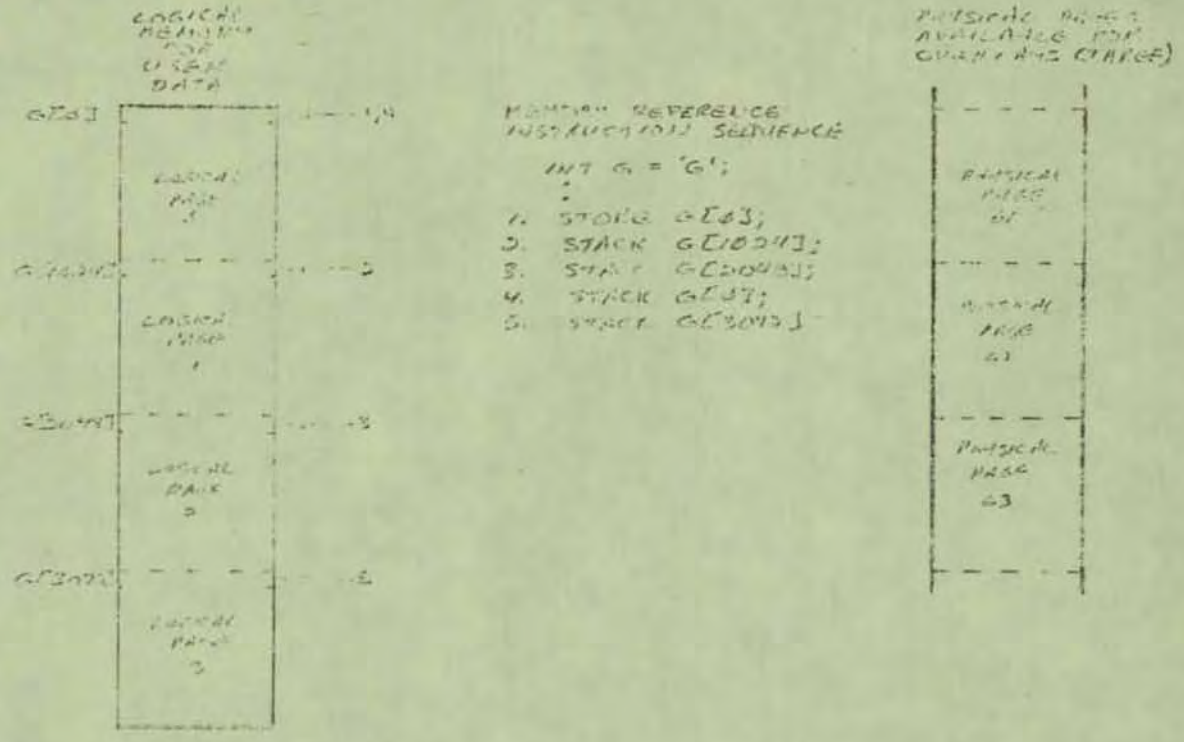
The following illustration shows how the reference bits are used by the operating system to select a page for overlay. The criteria used is: select the page that hasen't been accessed for the greatest amount of time. Note that R is set to "1" on any access to the page. R′ and R′′ are the R value shifted right one position by an AMAP instruction.

any access: "1" -> R

AMAP: "0" -> R -> R′ -> R′′

write access: "1" -> D

HOW THE OPERATING SYSTEM SELECTS PAGES FOR OVERLAYS USING THE REFERENCE BITS, DIRTY BIT, AND ABSENT BIT. THE CRITERIA IS THE NRU WITH THE LONGEST TIME SINCE LAST ACCESS



START: ALL PAGES ABSENT

USER DATA MAP

	V	U	M	D	A	LOGICAL PAGE #
	0	0	0	0	1	0
	0	0	0	0	1	1
	0	0	0	0	1	2
	0	0	0	0	1	3

1. STORE G[0];

- CAUSES PAGE FAULT INTERRUPT FOR LOGICAL PAGE 0
- EACH ENTRY IN THE MAP IS AGED
- LOGICAL PAGE 0 IS READ FROM DISK INTO PHYSICAL PAGE 61
- PHYSICAL PAGE NO. IS ENTERED INTO MAP USING SMAP

	1	1	1	0	0
	0	0	0	0	1
	0	0	0	0	1
	0	0	0	0	1

* INSTRUCTION IS REPEATED (SETTING THE DIRTY BIT)

2. STACK GROWS;

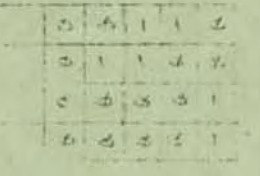
* CAUSES PAGE FAULT INTERRUPT FOR LOGICAL PAGE 2
* PAGE ENTRY IN THE MAP IS READ (RMAP)

* LOGICAL PAGE 2 IS READ FROM DISC INTO PHYSICAL PAGE 63
* PHYSICAL PAGE NO. IS ENTERED INTO MAP USING EMAP

* INSTRUCTION IS REPEATED

3. STACK GROWS;

* CAUSES PAGE FAULT INTERRUPT FOR LOGICAL PAGE 3
* PAGE ENTRY IN THE MAP IS READ (RMAP)



* LOGICAL PAGE 3 IS READ FROM DISC INTO PHYSICAL PAGE 62
* PHYSICAL PAGE NO. IS ENTERED INTO MAP USING SMAP

* INSTRUCTION IS REPEATED

4. STACK G-[03];

* THE REFERENCE BIT IS SET FOR LOGICAL PAGE 4

1	0	1	1	0	0
0	1	1	0	0	1
1	1	1	0	1	2
0	0	1	0	1	3

5. STACK G[3:07-3]

* CAUSES PAGE FAULT INTERRUPT FOR LOGICAL PAGE 2
* HOLE ENTRY IN THE MAP IS USED TO SELECT A PAGE

0	1	0	1	0	0
0	0	1	0	1	1
0	1	1	0	0	2
0	1	0	0	1	3

* BECAUSE ONLY THREE PHYSICAL PAGES ARE AVAILABLE, A PAGE MUST BE SELECTED FOR DISCARD. THE OVERLAY PAGE IS SELECTED ON THE BASIS OF THE LOWEST VALUE OF THE P,R',R",D FIELD COMPRISED IN THE MAP. IN THIS EXAMPLE, THE PHYSICAL PAGE ASSOCIATED WITH LOGICAL PAGE 1 IS CHOSEN. NOTE ALSO THAT THE DIRTY BIT FOR LOGICAL PAGE 1 IS A "0" SO THE PAGE NEED NOT BE WRITTEN OUT TO DISC
* LOGICAL PAGE 3 IS READ FROM DISC DISC INTO PHYSICAL PAGE 02
* PHYSICAL PAGE NO. IS ENTERED INTO MAP USING SMAP
* LOGICAL PAGE 1 IS MARKED ABSENT USING SMAP

0	1	0	1	0	0
0	0	0	0	1	1
0	1	1	0	0	2
1	1	1	0	0	3

* THE INSTRUCTION IS REPEATED

COLD LOAD

A processor may be initially loaded in one of two ways: from an I/O device or from another processor via one of the interprocessor buses. To perform a coldLoad operation, the SWITCH register switches are set to the eight-bit controller/unit number of the device or to a negative value if cold Loading over a bus, and the LOAD switch is pressed. The hardware then does the following:

* The system data map entry for logical page 0 and the system code map entry for logical page 63 to both set to refer to physical page 0 of memory.

* The instruction'BUN *' is placed in system code location P[%677].

* The E Register is set to %3400 (PRIV, DS,CS).

* If an I/O ColdLoad was selected, the IOC entry corresponding to the designated controller/unit in the SWITCH register is set with a transfer address of 0 and a byte count of %1600, a Cold Load command is issued to the controller/unit, and all interrupts are disabled.

* If a Bus Cold Load was selected, the BRT is set up to receive data from any processor over either bus, and all interrupts except X-bus receive and Y-bus receive are masked off.

* P is set to %176677. The BUN * in that location is executed repeatedly while the "bootstap" program is being read in.

The data read in by the bootstrap program in the Cold Load sequence loads the operating system. The bootstrap program begins executing as soon as the BUN * instruction in location %677 is overwritten. The starting conditions of the bootstrap program are:

```
P = %176677
E = %3400(PRIV, DS, CS)
System Data Map entry for logical page 0 and
  the System Code Map entry for logical page 63
  both point to physical page 0
MASK = %177400 if I/O Cold Load
     = %177430 if Bus Cold Load
```