

792'

I/O CPU IN THE ANPT PROJECT

**ORGANIZATION AND INTEGRATION
OF THE I/O CPU IN THE
ANPT PROJECT**

**BY
MICHAEL EDWARD BRETT, B. Sc.**

**A PROJECT REPORT
Submitted to the School of Graduate Studies
in Partial Fulfilment of the Requirements
for the Degree
Master of Engineering**

**McMaster University
November 1976**

MASTER OF ENGINEERING (1976)
(Engineering Physics)

McMaster University
Hamilton, Ontario

TITLE: The Organization and Integration of the I/O CPU in
the ANPT Project

AUTHOR: Michael Edward Brett, B. Sc. (Brock University)

SUPERVISOR: Mr. S. J. Lipton, Litton Systems (Canada) Limited

NUMBER OF PAGES: iv, 74

ABSTRACT

The ANPT (Air Navigation Procedures Trainer) is a navigation simulator being developed by Litton Systems (Canada) Limited. The ANPT design is based on the use of two ECLIPSE S/200 minicomputers to supply the background monitoring necessary for the system. This report deals with the implementation of the software for the processor that will control the navigation simulation hardware of the ANPT. Two major sections of the implementation are covered: the organization phase which details the modules needed to control the hardware and to communicate with the other processor, and the integration phase in which the various modules are linked together with each other and with the hardware in order to obtain a cycling system. The problems that could be encountered during system integration will be discussed along with possible solutions to these problems.

ACKNOWLEDGEMENTS

The author would like to express his appreciation to Litton Systems (Canada) Limited for its participation in the Industrial Internship Program of the Department of Engineering Physics, under whose auspices this project was conducted.

This report would not have been possible without the invaluable assistance of my industrial supervisors, S. J. Lipton and R. Territo, who combined to make this M. Eng. project a very enlightening and rewarding experience. Special thanks are also due to many other people at Litton especially D. Russenberger, R. Taylor, G. Hercezg, F. Venezia, K. Tipper and last but certainly not least to Daisy Hurst.

I would be seriously amiss if I did not acknowledge the contributions of Dr. T. J. Kennett, G. Cormick, K. Chin and the Engineering Physics Department of McMaster University who all helped furnish me with the necessary background to complete this report.

TABLE OF CONTENTS

	Page
ABSTRACT	ii
ACKNOWLEDGMENTS	iii
TABLE OF CONTENTS	iv
LIST OF FIGURES	v
CHAPTER	
1 INTRODUCTION	1
2 ORGANIZATION OF THE I/O CPU	5
3 INTEGRATION OF THE I/O CPU	14
4 CONCLUSIONS	18
APPENDIX A SAMPLE MODULE DOCUMENTATION	20
APPENDIX B LITTON SYSTEMS RELEASE NOTICE	71
BIBLIOGRAPHY	73
FOOTNOTES	74

LIST OF FIGURES

Figure		Page
1	ANPT Functional Block Diagram	3
2	Structure of the I/O CPU	6
3	Student's Display	8
4	Students Instrument Dial Panel	11

CHAPTER 1

INTRODUCTION

In the past decade, the cost of conventional computer systems has skyrocketed and many small computer users have not been able to afford these installations. The development of comparatively low cost minicomputers has changed this situation. Minicomputer systems, such as that based on the ECLIPSE Central Processing Unit of Data General have the computing power of an IBM System 370 without its million dollar price tag. These systems do not require either the special environment or continual maintenance of their larger cousins and are much more adaptable in their hardware configuration.

As the cost of these small processing units continues to drop and their sophistication increases, uses for computers are now being found in areas where the capital outlay for computer hardware was formerly prohibitive. Minicomputers and microprocessors have found their way into assembly lines and even down to the hobbiest level.

Industry has been quick to recognize the potential of minicomputers in simulation systems where their inherent versatility and small size make them the ideal choice for providing a real time monitoring system.

The ANPT (Air Navigation Procedures Trainers), a training system for student navigators that will be used by the Canadian Armed Forces, is an example of one such simulator based on a minicomputer system. This report is concerned with the organization and integration phases of the software modules which will control the navigation hardware of the ANPT. It attempts to give an insight into the level of sophistication of this system as well as some idea of the steps involved in obtaining a fully generated system.

The purpose of the ANPT is to provide ground training of student navigators in modern navigation principles and procedures to complement airborne training, at various difficulty levels ranging from ab initio (basic) training to refresher training.

This trainer will be installed in a building at Canadian Forces Air Navigation School (CFANS) at Canadian Forces Base (CFB) Winnipeg, in space provided as Government Furnished Facilities by the Department of National Defense (DND). The system design is based on the use of a digital computer central processing subsystem to provide dynamic simulation models with input/output devices, interfaces displays and control panels for effective student/instructor interaction in carrying out the simulated exercises.¹

Litton Systems (Canada) Limited, with its broad experience and expertise in airborne navigation and simulation systems, has been contracted by the Department of National Defense to implement the ANPT.

The computer subsystem consists of two Data General (DG) ECLIPSE Central Processing Units (CPU) connected by a DG Multiprocessor Communication Adapter (MCA). The MCA will be used as an interprocessor link for data transfer and processor-processor synchronization.

One CPU will be designated as the "Model CPU". This processor will run various aircraft, environment and navigation models that will simulate actual flight conditions. The aircraft model will provide the appropriate flight and aircraft parameters for a selected aircraft type including the speed and altitude range, and the turn and fuel consumption parameters. Included with these features will be the provision for simulated aircraft failures. The environmental models will be comprised of meteorological, celestial navigation and magnetic variation simulation modules. The navigation module will simulate a ground reference navigation facility including; ADF, VOR, TACAN, OMEGA and LORAN A and C, all with the appropriate visual/aural reception and malfunctions. The bulk of the programs in this CPU will be written in Fortran IV and will be compiled using DG Fortran V.

The second CPU, designated as the "I/O CPU", will handle most I/O operations between the computer subsystem and the ANPT and DG hardware. The ANPT simulates airborne navigation exercises for up to 16 student navigators simultaneously. Each student will have a CONRAC Model SNA/17R CRT display, a student Navigation Control Panel and an Interphone Panel. Four instructor consoles allow the monitoring and modifying of the student exercises by the instructors while two master consoles oversee the overall exercise. Since this hardware is of Litton design and implementation, the software device controllers are not available in the operating system supplied by Data General and require custom software drivers and handlers. This hardware will simulate navigation aids for the students such as magnetic and gyro compasses, altimeter, airspeed indicators and well as supply other navigation information on the CRT units. Each student will have a keyboard to communicate with the processing system.

Actions of the students and instructors at their consoles will generate interrupts in the I/O CPU which will activate the appropriate handlers. These handlers will be responsible for queueing up tasks in both processor. All modules in the I/O CPU will be written in DG ECLIPSE Assemblers.

The I/O CPU will run under DG RTOS (Real Time Operating System). RTOS is an entirely core resident operating system which is tailored by the user to the particular needs of a system environment. The rationale for using RTOS in this processor will be highlighted in the next chapter.

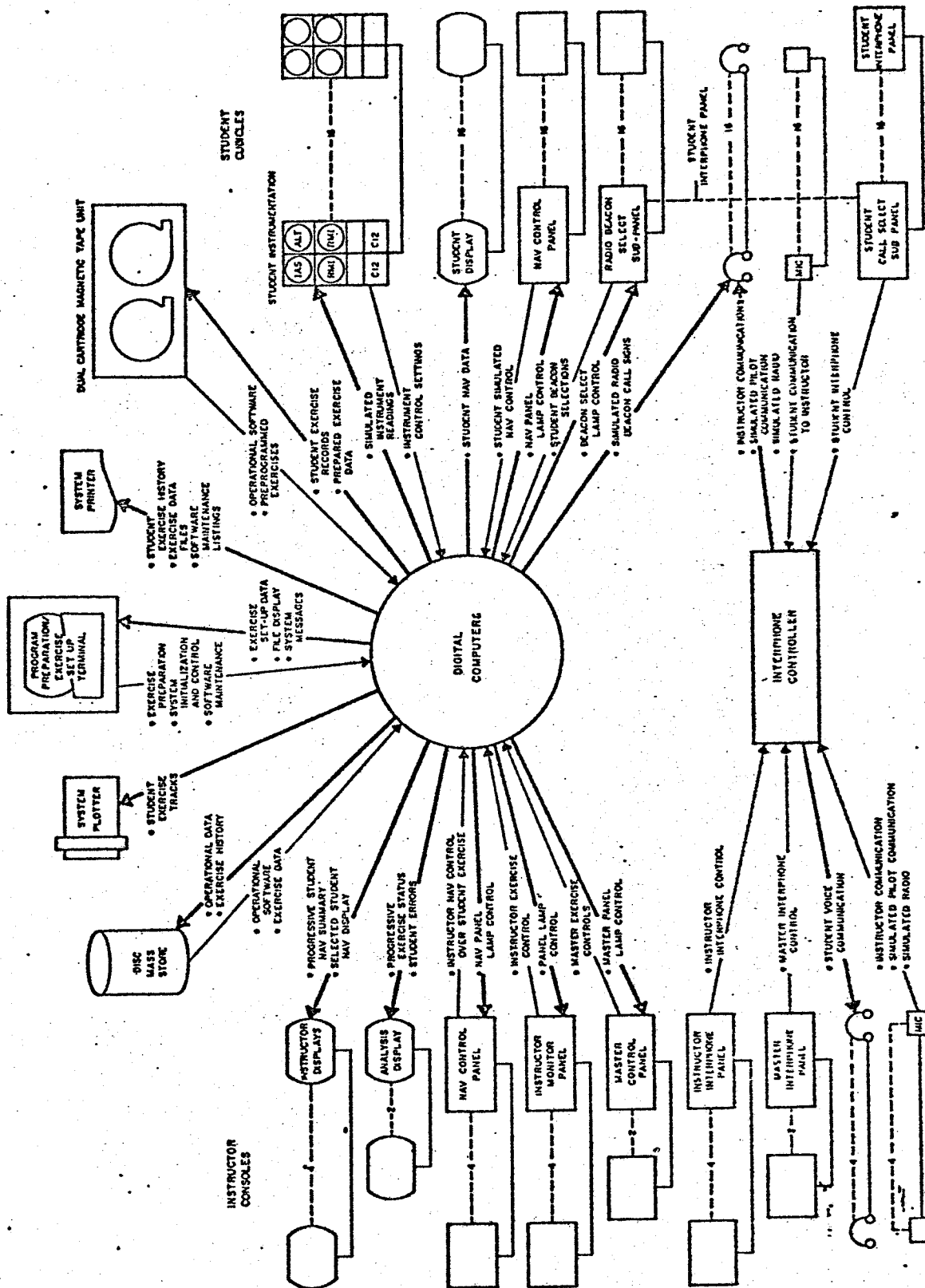


Figure 1

ANPT Functional Block Diagram

The Model CPU, on the other hand, will be configured using DG RDOS (Real Time Disc Operating System). RDOS combines the advantages of a disk operating system with the fast response provided by a core only real time operating system. The modular structure of the RDOS multi-task allows it to be tailored by the user at program load time to include only those real time features (task control logic) which will be needed in the exercise. This tailoring promotes efficient core utilization for each multi-task user program supported by RDOS. The address space in the Model CPU will not be sufficient to contain all the programs which are necessary for the operation of the system. The most efficient manner of handling this core limitation is to use the RDOS facility of overlaying. Program segments are swapped in and out of core by use of the disc. It is this facility that necessitated using RDOS in this processor and in turn required dedicating the disc to the Model CPU. The I/O CPU design is such that it does not require disc access and can use the more limited facilities of RTOS and benefit from its smaller system overhead.

The ANPT functional block diagram, shown in Figure 1 outlines the configuration of hardware in the system. The remainder of this report will deal with the generation of the I/O CPU, the CPU responsible for handling all the navigation hardware in Figure 1 during the game phase of an exercise.

CHAPTER 2

ORGANIZATION OF THE I/O CPU

The I/O CPU is divided into a number of inter-related tasks and subroutines supervised by RTOS. A simplified plan of the basic structure of the I/O CPU is shown in Figure 2. This plan arose out of the hardware requirements detailed in Figure 1 and the basic support structure of RTOS.

The Real Time Operating System (RTOS) for the DG family computer consists primarily of a small, general purpose multi-tasking monitor designed to control a wide variety of real time input/output devices. RTOS is entirely core-resident, highly modular and largely re-entrant, and allows for the straight forward addition of special device handlers.

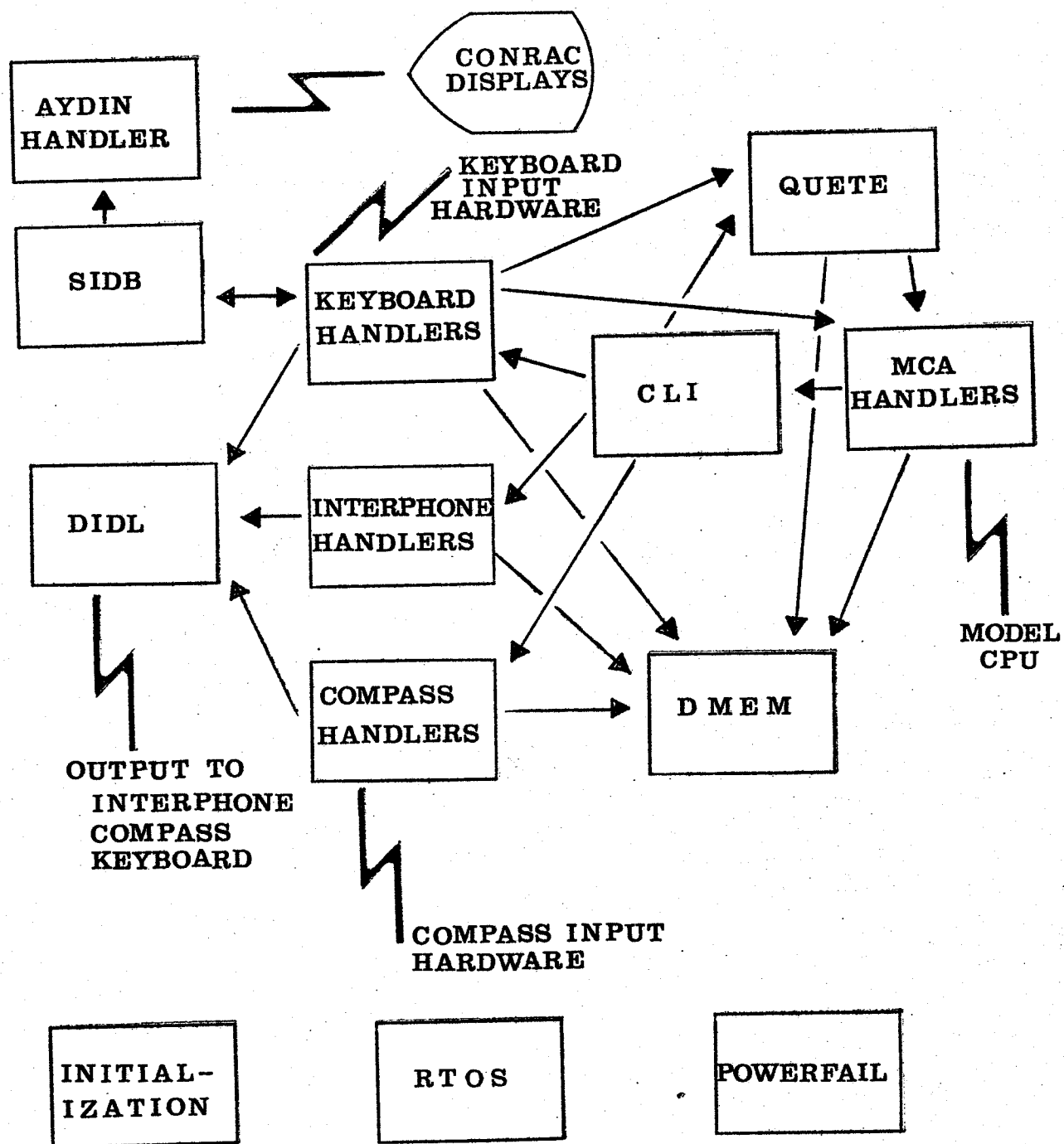
User programs are relieved from the details of I/O timing, data buffering, priority handling, and task scheduling by RTOS. In addition users are provided with a parallel processing capability plus inter-task communication and synchronization facilities. Communication with the RTOS monitor takes place through a small set of RTOS system and task calls.

A task, the basic logical unit controlled by RTOS, is a logically complete, asynchronous locus of control through a program. A task demands use of system resources (including CPU control) through the control logic of RTOS. Many tasks may be assigned to operate asynchronously in a single re-entrant sequence of instructions, and each task may be assigned a unique priority and identification number.

Due to the serial nature of a computer, tasks which appear to be executing their operations in parallel are in actuality executing these operations in short, serial segments. It is necessary then for RTOS to maintain certain status information (primarily active registers) concerning all tasks which are not currently in control of the Central Processing Unit (CPU). This information is retained in an information structure called the Task Control Block (TCB).

The I/O CPU exists in a multi-tasking environment where CPU control is allocated by the RTOS Task Scheduler to the highest priority task that is ready to perform or continue performing its function in real time. Rescheduling of CPU control occurs after hardware interrupt or calls to RTOS. The rescheduling facility of RTOS is vital to the ANPT, since certain tasks in the I/O CPU are raised to the ready state (ready to execute, but not having CPU control) by the Real Time Clock (RTC) and these tasks must gain control over lower priority (less important in time consideration) tasks that might be running at the time.

The CONRAC displays, which are controlled by the Aydin Display Generator will be the first piece of the instrument suite available for system use. The Aydin Display Handler has the task



STRUCTURE OF THE I/O CPU

FIGURE 2

of controlling the 23 CRT units in the system (16 student, 4 instructor, 2 master screens and 1 set-up terminal). The display handler supplies a software driver for communication to the screens and attaches an interrupt routine for the Aydins to the priority interrupt structure of RTOS. As in the case of all non-DG hardware, the Aydins will be identified as a user device at run-time rather than at system generation, in order to avoid the problems of linking into the RTIOS (Real Time Input/Output System) of Data General which allows a device to use the writing and reading routines of RTOS. Due to the special nature of the ANPT hardware, these I/O routines would involve considerably more system overhead than the use of custom data transfer routines.

A typical student display is shown in Figure 3. It consists of two sections; the permanent screen format and the values generated during the exercise. The permanent screen format is stored in core and is accessed during the exercise set-up, in the case of a powerfail or when a student selects a new instrument. The non-permanent image is maintained in the Screen Image Data Base (SIDB). This data reflects pertinent information such as the student's mission time, airspeed, altitude and values from the simulated navigation instruments such as LORAN or LATLONG. The data base is updated by messages from the Model CPU by the interprocessor link or from tasks within the I/O CPU.

In the ECLIPSE line of computers, when power is turned off and then on again, core memory is unaltered. However, when power is restored, the state of the accumulators, program counter and device flags are indeterminate. The power fail option, supplied as a standard component of the CPU, provides a "fail soft" capability in the event of power loss.

In the event of power failure, there is a delay of one to two milliseconds before the processor shuts down. Power levels are constantly monitored and in event of a power drop, the delay before power shutdown is used to store all volatile information in a safe area. A prolonged power failure will also destroy all the screen images as well as cause a disc shut-down. When power is restored, the powerfail routine brings about an orderly resumption of the exercise. The student, instructor and master screens must be restored from the Screen Image Data Base and the permanent format. All nav-aid devices such as the altimeter, compasses and keyboard lights are also volatile and will have to be restored from their Device Dedicated Locations (DIDL) after a prolonged power failure.

These Device Dedicated Locations (DIDL) are used by the nav-aid hardware to periodically update the instruments. To make a change in an instrument, its DIDL location in the I/O CPU must be altered by the appropriate handlers. At fixed intervals (normally 1 second), a data channel assess will be requested and the data from the DIDL locations will be steered by a hardware controller to the appropriate devices.

The I/O CPU is an unmapped unit which limits its maximum memory size to 32K words of

[illegible]

Figure 3 Student's Display

core. In a multi-tasking environment, there is a requirement for a task to be able to allocate buffer area for its exclusive use while that task is running. The limited core availability excludes the possibility of allocating a static buffer for each possible task; therefore a means is needed for allocating dynamic memory. The Dynamic Memory Handler enables tasks to acquire and release blocks of contiguous by maintaining a map of the dynamic memory core area and allocating tasks memory in contiguous blocks. Memory requests, that are not immediately fillable because of insufficient contiguous blocks available, cause the requesting tasks to be suspended until such a time as sufficient dynamic memory becomes available. The filling of these queued request is done on a priority basis.

One of the major users of dynamic memory is the module that controls the Multiprocessor Communications Adapter (MCA); the hardware that is responsible for communications between the two CPUs. The MCA driver is responsible for initializing the MCA receiver to accept messages from the other processor and the MCA transmitter to send a message to the other CPU. The driver handles 3 types of messages; a message which will go on the screen of associated display devices, a message which will go to the associated analogue device, a message for the driver itself. Upon reception of the message, the MCA dispatcher will analyze the message. If it is either a screen or analogue message, the Dispatcher will attach a task to service it. If the message is for the driver, the Dispatcher awaits the reception of a message which is of specified length and destined for a specified address. This allows the transmitting CPU to load data into the Device Interface Dedicated Locations (DIDL) directly. The Dispatcher is also responsible for preparing messages from a task to be sent to the MCA driver. If a message is greater than the reception area in the other CPU, the Dispatcher will send a pre-message which will inform the other CPU to obtain sufficient core area to receive the large message. The MCA modules uses dynamic memory to receive messages and it informs the task receiving the message of its responsibility of releasing the dynamic memory.

Once a message is received and it is not a pre-message or a dedicated message, the Command Line Interpreter (CLI) must decode the header of the message and determine the message disposition. It must activate the appropriate tasks within the I/O CPU. These tasks could include the keyboard or the compass handlers or be responsible for changes in the Screen Image Data Base. The number of tasks that are activated for the CLI depends on core availability for TCBs. Incoming messages are queued in order for handling so that messages retain chronological sequence when they are passed to the destination tasks.

The compass handlers are responsible for the I/O support of the simulated Radio Magnetic Indicators (RMI), Airspeed Indicators and Pressure Altimeters. Each student will have two RMI systems displaying compass and bearing indication for VOR, ADF and TACAN (ground based radio navigation system networks) and two C-12 compass controllers to handle the RMI systems.

The C-12 compass operates in either directional gyro (DG) or magnetically slaved (MAG) mode, selectable by the student from the student instrument panels (shown in Figure 4). When one of the compass input devices signals an interrupt, the compass handler is responsible for putting the input into the proper DIDL location. The hardware interface supplies along with the actual compass information, the DIDL location for which the information is destined. The input handler is also responsible for sending the information to the Model CPU via the MCA Transmitter Dispatcher and putting the relevant information in a common area for use by the Compass Output Handler.

The Compass Output Handler accepts data from the Compass Input Handler and the Model CLI and displays the result on the compass output devices in a realistic manner. When new information is to be displayed on any of the Nav Instruments, the output handler supplies updates to the DIDL location (location accessed by hardware to update a certain device) at an appropriate rate of change. The output handler is also responsible for the synchronization for the C-12 compass in MAG mode and resetting the reference in DG mode.

The keyboard input handler accepts input from the student, instructor and master console keyboards. The hardware gives the octal identifier of the keyboard and it is the responsibility of the handler to use this information to move the inputted character to the proper fixed address in memory. The keyboard output handler receives data from the CLI indicating which keyboard lights are to lit and/or unlit on the different keyboards and performs the appropriate action.

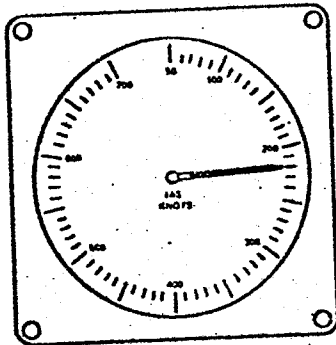
The student, instructor and master keyboard handlers accept data from the keyboard input handlers and updates the appropriate CRT and SIDB locations. As an input field is completed, the keyboard handlers will pass the information entered to the appropriate simulation modules via the MCA and into reserve locations in the I/O CPU. When a student wishes to change information in a certain field on the screen, the keyboard handler will reverse the intensity of the field until the student has completed the change and pressed the appropriate terminating key.

Some messages from the keyboard handler to the Model CPU must be sent on student mission time. Student mission time is governed by one of three clock rates: one hour exercise time in 48 minutes real time, one hour exercise time in one hour real time, one hour exercise time in 72 minutes real time. Student mission clocks can also be frozen from the instructor console. Since exercise time can vary from student to student, special handling is required to send messages on student time. Changes in aircraft parameter, for example, are entered by the student along with a time to implement the change. The module QUETE is responsible for sending these queued messages and also for deleting messages for an instrument when a handler queues a revised message for that same instrument.

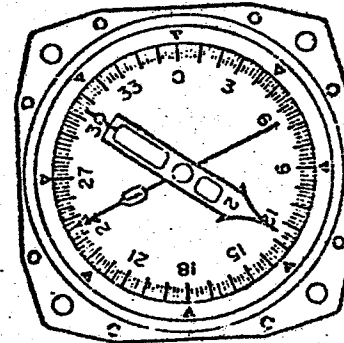
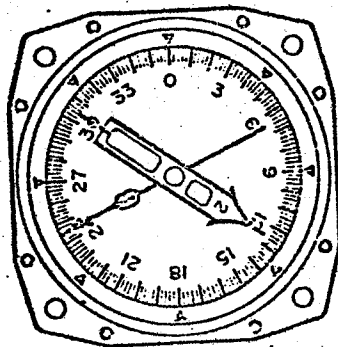
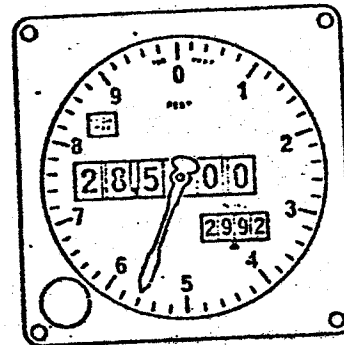
The interphone hardware, besides providing a means of student instructor communication during the exercise, is used for reception of simulated radio signals from ground based navi-

NAVIGATION INSTRUMENT DIALS

INDICATED AIRSPEED



ALTIMETER



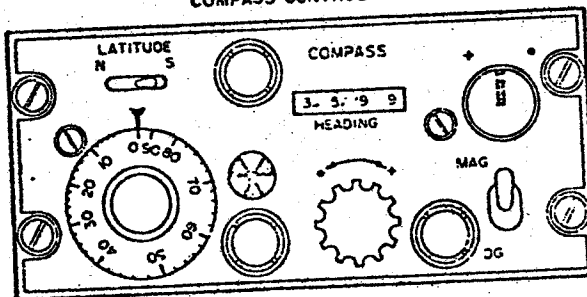
ADF 1

VOR 1 TACAN

ADF 2

VOR 2 TACAN

COMPASS CONTROL 1



COMPASS CONTROL 2

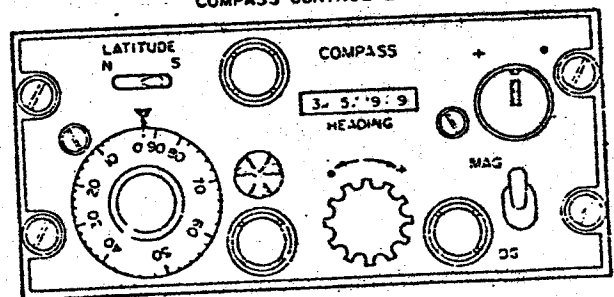


Figure 4 Students Instrument Dial Panel

gation stations. The Student Interphone panel provides for the selection of the audio tones for ADF, VOR and TACAN. When the model processor requests an interphone transmission, the I/O CPU interphone handler transmits ASCII characters to a tone generator which produces the Morse Code equivalent. in the interphone.

When the pre-game phase of an exercise is completed (the defining of the exercise scenario by the instructors using the facilities of the Model CPU), the runtime I/O CPU system requires extensive initialization before the exercise can actually begin.

All the peripherals in the system except the disc are connected to an I/O Bus Unit that is separate from either of the two processor. This unit is switchable to either CPU by program control. During pre-game the unit is controlled by the Model CPU to allow access to the line printer and certain other peripherals. At the end of pre-game, the unit is returned to a neutral position. The first act of the initialization routine in the I/O CPU is to gain control of the I/O Bus. Next the I/O CPU awaits a message from the Model processor that contains the flight instrument suite for the current exercise. This information is used to reserve sufficient core for the Screen Image Data Base (SIDB) and set up certain pointers to the student device area within it. The size of the SIDB is dependent on the number of instruments that the students are allowed to use in an exercise and thus is not known a priori. Since the remaining unused core in the I/O CPU, after the SIDB is allocated, is used as dynamic memory, the pointers for the Dynamic Memory Handler are not initialized until the set-up of SIDB is completed. The initialization module itself resides in an area of core that is destined to become dynamic memory during the game.

The initialization module is also responsible for queuing up all those tasks that run throughout the exercise. These include the Quiet routine, 2 MCA routines, the master, instructor and student keyboard handlers and the compass input and output handlers.

All devices that are controlled by the I/O CPU during the game must be linked into the RTOS structure as user devices. This initializing process requires identifying to the system a Device Control Table (DCT) for each device. The DCT will be used by RTOS to service an interrupt from that particular device.

Due to the complexity and inter-relation of different modules within the ANPT, extensive documentation is required for even the smallest module. A system module, within the ANPT structure requires the production and approval of 5 separate documents; a Design File Memo (DFM), a Computer Program Specification Document (CPPS), a Program Implementation Document (PID), the code itself, and a Software Test Document (STD).

The DFM is a general outline of the requirements for a proposed software module. Its primary function is to solicit comments and criticisms on the planned program from the ANPT staff before any further development is carried out.

The CPPS establishes the specification for the computer program. It outlines the functional analysis of a computer program along with its interfacing requirements, design constraints and application documents.

The PID is a detailed step-by-step implementation plan for a computer program. It includes a functional description of all sections of a computer program along with required storage allocation and program interrupts. The most vital parts of a PID are the computer program functional flow diagram (flowchart) and the logic of subroutine reference. The subroutine reference section outlines the initialization requirements, calling sequence and any other information pertinent to the integration of the program into the run-time system.

The code itself must be relocatable and constructed to be free of global conflicts with the system as a whole. Further, the code should be located in NREL memory (memory in locations greater than 4000) and the use of the scarce ZREL memory (memory in locations less than 4000, which is directly addressable from anywhere in core by single word instructions) is restricted to very special circumstances.

The STD must show that the program executes in the manner specified in the CPPS and PID and meets the requirements of the DFM. While not an exhaustive stand-alone test, the STD must show that the module code is of a level of readiness to be integrated into the system.

Appendix A traces one module, DMEM (Dynamic Memory Allocation) in the transition from a DFM to a completed and approved module, ready for system integration. The documentation presented in the Appendix is representative of all the modules in the I/O CPU.

Stand-alone testing cannot show the effects of interaction between the various modules in the I/O CPU, this linkage testing must be left to the integration phase. The integration of the system modules must proceed in an orderly manner to minimize complications in debugging the module linkages. This integration plan of action comprises the contents of the next chapter.

CHAPTER 3

INTEGRATION OF THE I/O CPU

The integration phase of the ANPT project is divided into 3 phases. Part 1 is a step-by-step integration of the software modules of the Model Computer. Part 2 is the integration of the software and available hardware of the I/O CPU. These two phases of integration are carried out simultaneously and independently. Part 3 is the integration of the Model and I/O processors with the total ANPT equipment suite. Start date of the integration of the ANPT project is 1 September 1976 and target for total completion is 15 February 1977 (total completion is defined as a system that will cycle with reasonable errors).

The I/O CPU integration involves establishing interfaces with each of the I/O devices on an individual basis. It is anticipated that the total completion of this part will not be accomplished until after the entire equipment suite is available for testing. The end product of this phase of the integration is a level of operational readiness that will allow a smooth transition into phase 3 integration on 1 December 1976.

In any system as complex as the ANPT project, the linking of all system modules into a workable unit represents a formidable task. In the case of the I/O CPU, integration must begin before all the nav-aid instruments are available. Without this hardware, testing of the overall system operation is seriously curtailed. The effect of the sixteen compass systems and all the instructor, student and master consoles on the I/O processor cannot be fully assessed until after all the hardware is made available.

Two pieces of hardware are available for the beginning of integration. The MCA link was delivered with the ECLIPSE processors and one CONRAC CRT with the accompanying AYDIN display generator was installed in mid August 1976. The testing of the associated drivers and handlers was completed for these devices before phase 2 integration was begun.

Some hardware integration can be accomplished before the actual hardware is delivered. The keyboard of an available ADDS CONSUL can be employed as a substitute for the student instructor keyboards. The keyboard handlers are then a logical focal point for the beginning of the integration process. The student keyboard handlers link into the Screen Image Data Base controller which in turn initiates a screen update by calling the AYDIN Display Handler. The keyboard handler is also responsible for sending the inputted message to the Model CPU via the MCA. This message is built in memory obtained from the Dynamic Memory Handler and then passed to the MCA Handler for transmission. During the integration of the keyboard handler, QUETE will be added to

the system and its linkages into DMEM and MCA will be tested. With the three keyboard handlers functional, a major portion of the system linkage will be accomplished. However the other phase of the integration process can not be completed until all 22 keyboards are in the system and active. Until then, the system degradation that will result when all the students key in at once, will remain a matter for speculation.

Concurrent with the keyboard integration will be the linking of the initialization module into the system. Once properly functioning, this routine will bring the system to an operating level by initializing the SIDB and the dynamic memory parameters and queuing up all permanent tasks and in so doing will provide the necessary initialization to allow the testing and integration of the other modules in the system.

A method of simulating the message capabilities of the MODEL CPU must be constructed in order to accomplish integration of the CLI into the I/O system. The linkage of the CLI in the various routines such as the keyboard and Compass Handlers is of the utmost importance to the overall system operation. Since the MCA receives messages in dynamic memory, it is vital that the CLI messages be decoded and acted upon as soon as possible in order to avoid locking out dynamic memory. It must be determined during the link-up if the CLI is capable of handling a large influx of messages in a short time without seriously overtaxing the system and whether the number of TCB (Task Control Buffers) attached for its use by initialization is sufficient to efficiently handle all messages.

One of the most difficult chores in integration involves the implementation of the Compass Output Handlers. This module is responsible for taking data from the CLI and simulating a realistic motion of the compasses. When the actual integration of the compass handlers with the hardware begins, the compass motion will be carefully studied to determine the most realistic motion with the least CPU intervention. The updating of the 32 compasses in the ANPT equipment suite could place a serious strain on the system especially if the hardware response is appreciably more rapid than the real instrument and thus will require that the CPU continually move it in small steps. The Writeable Control Store option of the ECLIPSE line may be used to relieve some of the burden of these updates.

The Writeable Control Store (WCS) is an extension of the microprogrammed control logic of the computer's central processing unit. WCS gives users access to the microprogrammed logic of the CPU and thus allows them to implement specialized instructions. WCS consists of 256 56 bit words of high speed random-access semi-conductor memory (RAM) in the CPU's control store. Information placed in the RAM defines the execution of sixteen two accumulator instructions. Under microprogrammed control, instructions are defined that cause particular control signals to be generated in the system. These signals could be used to control much of the compass

handling since the WCS instructions would be designed specifically for the update task, instead of being the more conventional and generalized ECLIPSE instructions which must be first fetched from the comparatively slow core memory and then gated through much more control logic before the actual instruction is executed. If, even with WCS incorporated into the Compass Handlers, the updating overhead becomes impractical, damping of the actual hardware may become necessary.

Probably the most important consideration in a system that is partially driven by hardware interrupts is the very real problem of system lock-up and lock-out. With rescheduling of tasks occurring on every interrupt, low priority tasks may not get system control for any reasonable length of time over extended periods. In the case of the low priority compass handlers, this may result in a rather stilted motion of the compasses, thus violating a contractual commitment to move them realistically. If a noticeable lock-out occurs for any module, the priority structure of the I/O CPU must be carefully reviewed. Another possible effect of an improperly organized priority structure is the lock-up of data basis or system resources by the suspension of a low priority task. Dynamic memory could become badly fragmented by low priority tasks being suspended for long periods of time and not releasing dynamic memory back to the pool. If the MCA handler is not able to obtain sufficient memory to receive a message, the MCA transmitter on the Model CPU will be locked, attempting to send the message over and over again. Dynamic memory in the Model CPU may also become scarce as more and more messages become queued for transmission. To remedy these sort of problems, tasks which originally earmarked as having a high priority may be reduced in priority so that a less important task may be run and release the resources it has captured. The final cycling system will be a carefully tuned balance between the overall task priority setup and the need for the quick release of system resources.

Even with all possible optimizing of the priority structure, there may still be insufficient core for the I/O CPU to function properly. Dynamic memory could become either totally allocated or so badly fragmented that large requests could not be fulfilled. High priority task would be delayed waiting to acquire memory and low priority task would become totally locked out in their attempts to obtain their required core. Some memory could be obtained by storing certain items such as the permanent instrument screen masks on the disc, but this would require calls to the Model CPU for a disc access when a student changes an instrument or in the case of powerfail. The only solution if the memory problem becomes extremely critical would be to extend the memory capabilities of the ECLIPSE by employing a memory mapping unit which would increase the systems capacity up to a maximum of 128K. As mapped RTOS does not currently exist, a need for more core would necessitate writing an extension of the RTOS operating system to handle the mapping unit. Since the mapping unit checks each address before it accesses it to see if it is in range of the calling partition, the cycle time of the system will be slowed. The capital outlay and

the time spent in implementing the mapping unit make it a last resort in obtaining a cycling system.

If memory space does become limited, the Symbolic Debugger will not be loaded into the system during integration. The Symbolic Debugger interfaces with user routines, allowing the user to monitor and correct his program during execution. The Symbolic Debugger provides up to eight active breakpoints within a user routine. Accumulators, carry and memory can be examined and modified during execution using simple debugger commands issued from the console. The loss of the Symbolic Debugger, due to core limitations during integration will slow the testing of the various module linkages considerably.

Another important factor to be considered in the integration of the I/O CPU involves the Model CPU. Certain modules in the Model Processor have to be written in DG Assembler. These modules include an MCA Handler, a Dynamic Memory Request Handler and TIMERT, a routine to update student and instrument clocks. The integration of these three routines into the Model CPU will require the efforts of the programmers of the I/O CPU and any difficulties in this process will slow down the integration of the I/O Processor. Special care must be taken that the Assembler routines properly link in with the Fortran structure of the Model CPU.

The integration picture in the I/O CPU is really not as grim as presented so far. None of the difficulties mentioned here has been encountered to date (8 November 1976). This static core requirements for all modules and data basis appears to be less than 75% of the total capacity of the system and no major inter-program incompatibilities have been encountered so far. The response of the I/O CPU have been well within acceptable limits as none of the modules has even remotely taxed the system capabilities. The major obstacle in integration is the omni-present lack of nav-aid and keyboard hardware and this looms as the major deterrent in the path of completion of phase 2 integration by 1 December 1976. This problem has been further compounded when the delivery data of a complete student cubicle, originally scheduled for 1 October 1976, was postponed to mid November 1976.

CHAPTER 4

CONCLUSIONS

The ANPT concept presented here is that of a real-time operating system providing a training environment for student navigators. It demands a computer that is flexible in design and can operate at a rate sufficient to handle 16 students simultaneously without any system degradation noticeable to the student. The ECLIPSE is the logical choice for the computer system. Based on DG published benchmarks and instruction execution times, the ECLIPSE is faster than most general purpose computers currently available. A direct memory channel is incorporated for increasing the throughput of I/O operations. Memory interleaving also substantially reduces effective cycle time. In the system provided for the ECLIPSE, consecutive memory locations can be loaded on separate memory modules. This allows the CPU to access the next instruction while executing the current one.

The structure of the I/O CPU can now be seen as the logical outgrowth of the hardware devices of the ANPT equipment suite and the Date General Real Time Operating System (RTOS). Heavy reliance is placed on RTOS both in its facilities for handling multi-tasking through priority rescheduling and sync words, and in the area of response to hardware interrupts. The handling of the NAV-AID hardware of the ANPT presents some rather special problems because of the unique nature of the compasses and keyboards. Considerable effort has been spent to protect data bases and system resources from the effects of task rescheduling.

The integration phase represents the most formidable task in the generation of the I/O CPU system. Task scheduling and priority, properly sequencing events, debugging module linkage and incompatibilities all present problems that must be conquered during integration. All sources of system degradation from software sources must be tracked down and corrected by whatever means are possible. Interfacing with the hardware and analyzing the effects of hardware induced system degradations will be especially difficult without a full suite of student and instructor cubicles. Only with this hardware can the full system load of 16 student and 4 instructors be applied to the I/O CPU with the resulting strain on the processor resources.

In overview, the ANPT represents yet another of the growing number of applications of minicomputers to the field of training simulator. These systems are highly complex units, requiring state of the art technology and the efforts of many highly skilled professionals during development. Once functional however, these system can be operated by personnel with only rudimentary knowledge of the actual internal operations of the system. With the increasing trend of industry to the micro-processor and minicomputer philosophy, systems such as the ANPT will become more

and more commonplace in the years to come.

As a final thought, it should be emphasized that even with all the sophistication and the almost frightening speed of present computers, it is not likely that the use of computers will replace human teaching. Computer - assisted instruction will vastly alter the teacher's function as it will force a renewed emphasis on the creative motivating aspects which are even today the essence of good teaching. Progress in computer-assisted instruction will be primarily a function of the instructional computer programs themselves. The trend will be to programs which will be highly responsive to the student's progress. The human element will continue to be a vital part of any educational process to provide the understanding an inflexible machine cannot.

APPENDIX A

SAMPLE MODULE DOCUMENTATION



LITTON SYSTEMS (CANADA) LIMITED

TORONTO OFFICE

INTER-OFFICE
CORRESPONDENCE

O: ANPT Design File

FROM: R. Territo

DESIGN FILE MEMO NO. 1015

DATE: 14 MAY, 1976

SUBJECT:

DYNAMIC MEMORY ALLOCATION

In a multi-tasking environment there is a requirement for a task to be able to allocate buffer area for its exclusive use while the task is running. Limited core availability excludes the possibility of allocating a static buffer of maximum length for every possible task; therefore a method of dividing available memory amongst active tasks is required.

A solution is outlined here:

The initialization module determines the amount of unused memory available after allocating the screen image data base and builds a list of contiguous words in which each bit represents one page of memory (256 words). Each word in this list then defines 4096 words of memory. The total list will be at maximum 4 or 5 words long. Each request for dynamic memory (GETMAIN) will scan the list for the minimum amount of available space to satisfy the request. The requesting task is then passed the start address of its area. When the task no longer needs the buffer or part of the buffer it passes back the start address and number of pages to be returned to the pool (FREEMAIN). Some precautions must be taken to ensure that the GETMAIN and FREEMAIN are not interrupted while operating on the "bit map". The GETMAIN must be smart enough not to allocate a single page from the middle of a large area of available memory thus fragmenting it. Also, the condition of "lock-out" must be recognized and procedures defined to avoid it.

R. Territo



25 Cityview Drive, Rexdale, Ontario, Canada M9W 5A7

28711

PAGE

22

DOCUMENT
TYPE

SOFTWARE DOCUMENT

COMPUTER PROGRAM PERFORMANCE SPECIFICATION
DYNAMIC MEMORY ALLOCATION (DMEM)

REV	ECO	CHANGE DATE	CHANGE DESCRIPTION	APPROVAL

LSL RELEASE: DRN

DATE

LSL RELEASE: DRN

DATE _____

APPROVALS

PREPARED

Michael E. Brett

SYSTEMS ENGINEER

R.J. Taylor

~~PROJECT ENGINEER~~

S.J. Lipton

SOFTWARE ENGINEER

R. Territo

RELEASE / APPROVAL

S. J. Lindor

THIS DOCUMENT CONTAINS INFORMATION PROPRIETARY TO LITTON SYSTEMS (CANADA) LIMITED. ANY REPRODUCTION, DISCLOSURE OR USE OF THIS DOCUMENT IS EXPRESSLY PROHIBITED EXCEPT AS LITTON SYSTEMS (CANADA) LIMITED MAY OTHERWISE AGREE IN WRITING.



CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

1. SCOPE

1.1 Identification

This specification establishes the performance, development and verification requirements for the DYNAMIC MEMORY ALLOCATION computer program referred to as DMEM. The DMEM is a part of the 'ANPT Software Package' for use in the Air Navigation Procedures Trainer.

1.2 Functional Summary

This computer program will enable tasks to acquire and free blocks of dynamic contiguous memory in order to use available core more effectively.

1.3 Assumptions and Constraints

T.B.D.



LITTON SYSTEMS (CANADA) LIMITED
Litton 25 Cityview Drive, Rexdale, Ontario, Canada M9W 5A7

NUMBER

28711

REV

PAGE

24

CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

2. APPLICABLE DOCUMENTS

2.1 Government Documents

None

2.2 L.S.L. Documents

T.B.D.

ANY RESTRICTIVE AND/OR OTHER PROTECTIVE NOTICES, IF ANY, ON THE SHEET FOR WHICH THIS SHEET SERVES AS A CONTINUATION ARE HEREBY INCORPORATED HERE ON.



CODE
IDENT

09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

3. REQUIREMENTS

3.1. Computer Program Definition

In a multi-tasking environment, there is a requirement for a task to be able to allocate buffer area for its exclusive use while the task is running. Limited core availability excludes the possibility of allocating a static buffer for each possible task; therefore this program provides a means for allocating and releasing dynamic memory.

3.2. Interface Descriptions

3.2.1 Equipment Interface Requirements

None

3.2.2 Interfacing Computer Program Requirements

The initialization module must determine the amount of unused memory available after allocating the screen image data base. The module must build a list of contiguous words, called the bit map, in which each bit represents one block of memory (32 words). The initialization module then defines the location of the first word of the bit map, the number of available blocks of contiguous dynamic memory and the starting address of the dynamic memory as global symbols for the external use of GETMAIN (Get Memory) and FREEMAIN (Free Memory).

GETMAIN and FREEMAIN require a sync word to be set to a non-zero value by the initialization routine. The sync word will be used by the .XMT and .REC commands in the subprogram to give a task sole



CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

control of that subprogram in order to maintain the integrity of the bit mask.

WGETMAIN (Wait Until Enough Memory for GETMAIN) requires that its sync word be set to zero by the initialization routine. This sync word will be used to queue up tasks for GETMAIN calls.

Each subprogram (GETMAIN and FREEMAIN and WGETMAIN) will be a complete logical unit, labelled and commented. The label associated with each subprogram shall be declared as an entry point for external use.

3.2.3

Timing and Sequencing Requirements

None

3.3

Detailed Functional Analysis

Dynamic memory is divided into 32 word blocks. Each block is represented by one bit in the bit map and the condition of the bit determines whether the block is allocated or free (1 = allocated, 0 = free.)

When GETMAIN is called, the number of blocks needed by the task are passed in AC0. GETMAIN scans the memory bit map for sufficient contiguous memory to satisfy the request. If the search is successful, the bit map is updated and control is returned to the calling task with the starting address of the allocated blocks in AC2. If insufficient contiguous memory is available, control is returned to the calling task with an illegal address in AC2 (177777). When FREEMAIN is called, the number of blocks to be freed is passed in AC1 and the starting address of the core to be freed is passed in AC0. FREEMAIN then resets the appropriate bits in the memory masks and returns control to the calling task. No error messages are returned to the calling task.

ANY RESTRICTIVE AND/OR OTHER PROTECTIVE NOTICES, IF ANY, ON THE SHEET FOR WHICH THIS SHEET SERVES AS A CONTINUATION ARE HEREBY INCORPORATED HERE ON.



CODE
IDENT

09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

In order to maintain the integrity of the bit mask during execution of GETMAIN or FREEMAIN, the bit map is protected by .REC and .XMT task calls.

When WGETMAIN is called, the number of blocks needed by the task are passed in AC0. WGETMAIN then calls GETMAIN. If, on the return from GETMAIN, AC2 contains a valid address control is returned to the calling task. If, however, on return from GETMAIN, an invalid address is contained in AC2 (insufficient contiguous memory available to satisfy request) this task will be suspended until such a time as some dynamic memory is released. These suspended tasks will be activated on a priority basis. Once a task is re-activated it again follows the WGETMAIN flow, therefore control is only returned to the calling task when its memory request has been filled.

3.4

Adaptation Data

N/A

3.5

Design Constraints

(a) Since this program will use ECLIPSE unique instructions it will be necessary to assemble the module using the MACRO assembler.

(b) The module will be supplied on magnetic tape cartridge and a copy on punched paper tape.

(c) The module uses the facilities of the RDOS/RTOS operating system and therefore cannot run stand-alone.

CODE 09691
IDENT

DOCUMENT
TYPE

SOFTWARE DOCUMENT

TITLE

ANPT PROGRAM IMPLEMENTATION DOCUMENT
DYNAMIC MEMORY ALLOCATION (DMEM)

REVISIONS

REV	ECO	CHANGE DATE	CHANGE DESCRIPTION	APPROVAL

LSL RELEASE: DRN DATE

PREPARED

M. Brett

M. Brett.

APPROVALS

SYSTEMS ENGINEER

R.J. Taylor

INEER 

~~PROJECT ENGINEER~~

~~S. J. Winston~~

~~REFUSE~~ APPROVAL

S. J. Dipton

SOFTWARE ENGINEER

R. Territo

ENGINEER
R. Terry

THIS DOCUMENT CONTAINS INFORMATION PROPRIETARY TO LITTON SYSTEMS (CANADA) LIMITED. ANY REPRODUCTION, DISCLOSURE OR USE OF THIS DOCUMENT IS EXPRESSLY PROHIBITED EXCEPT AS LITTON SYSTEMS (CANADA) LIMITED MAY OTHERWISE AGREE IN WRITING.



CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

1. SCOPE

1.1 Identification

This specification establishes the description and logic of the DYNAMIC MEMORY ALLOCATION computer program referred to as DMEM. The DMEM is part of the 'ANPT Software Package' for use in the Air Navigation Procedures Trainer.

1.2 Functional Summary

This computer program will enable tasks to acquire and free 32 word blocks of dynamic contiguous memory in order to use available core more effectively. The program maintains a memory bit map of dynamic core.

The bit map is updated when a request is filled. When tasks free dynamic memory, the appropriate bits in the map are reset.



LITTON SYSTEMS (CANADA) LIMITED
Litton 25 Cityview Drive, Rexdale, Ontario, Canada M9W 5A7

NUMBER

28707

REV

PAGE

30

CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

2. APPLICATION DOCUMENTS

2.1 Government Documents

None

2.2 L.S.L. Documents

Specification # 28711

'Dynamic Memory Allocation (DMEM)'



CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

3. REQUIREMENTS

3.1 Function Allocation Description

DMEM consists of three subroutines; GETMAIN, the memory allocation subroutine, FREEMAIN, the memory de-allocation subroutine and WGETMAIN, the queued memory allocation subroutine.

3.2 Functional Description

3.2.1 Subroutine GETMAIN

The number of blocks of required memory are passed to the subroutine in AC0. Control is passed to GETMAIN by a PSHJ GETMAIN command. GETMAIN will scan the memory map to find if sufficient contiguous memory is available to satisfy the request. If the search is successful the bit map is updated and control is returned to the calling task with the starting address of the available core in AC2. If insufficient contiguous memory is available, control is returned to the calling task with 177777 (an illegal address) in AC2.

3.2.2 Subroutine FREEMAIN

When FREEMAIN is called by a PSHJ FREEMAIN, the number of blocks to be freed is in AC1, and the starting address of the released core in AC0. FREEMAIN then resets the appropriate bits in the memory map and returns control to the calling task. No messages are returned to the calling task.

3.2.3 Subroutine WGETMAIN

When WGETMAIN is called, the number of blocks needed by the task are passed in AC0. WGETMAIN then calls GETMAIN. If on the return from GETMAIN, AC2 contains a valid address control is returned



CODE
IDENT

09691

DOCUMENT
TYPE

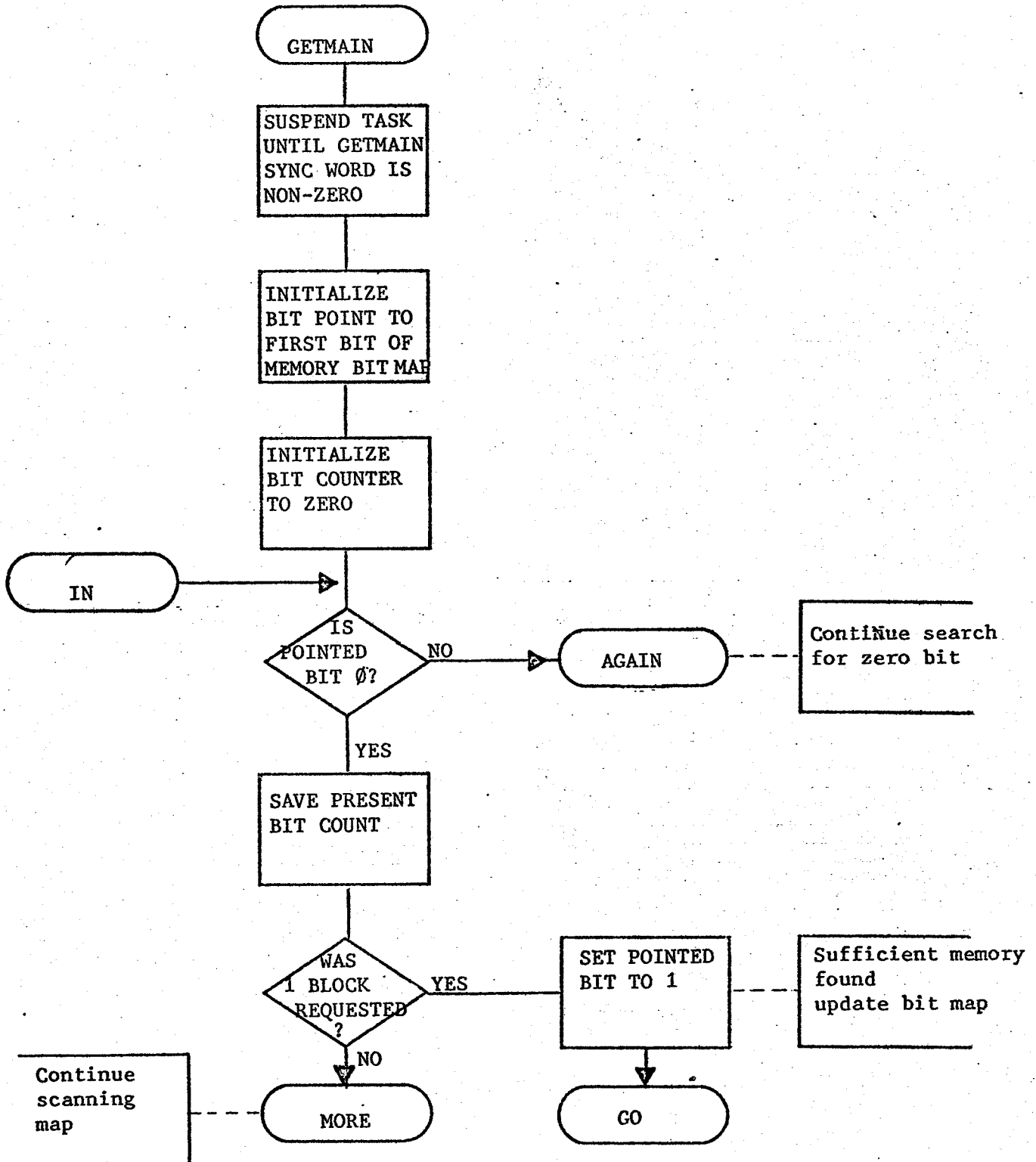
SOFTWARE DOCUMENT

to the calling task. If, however, on return from GETMAIN, an invalid address is contained in AC2 (insufficient contiguous memory available to satisfy request) this task will be suspended until such a time as some dynamic memory is released. These suspended tasks will be activated on a priority bases. Once a task is re-activated it agains follows the WGETMAIN flow, therefore control is only returned to the calling task when its memory request has been filled.

3.3

Storage Allocation

DMEM requires approximately 190 words of NREL core. No ZREL memory is required. Of these 190 words, GETMAIN requires 120 words, FREEMAIN requires 40 words and WGETMAIN requires 30 words. GETMAIN, FREEMAIN and WGETMAIN require 1 sync word each and GETMAIN and FREEMAIN share 3 control words. The sync and control words are defined in the initialization routine.

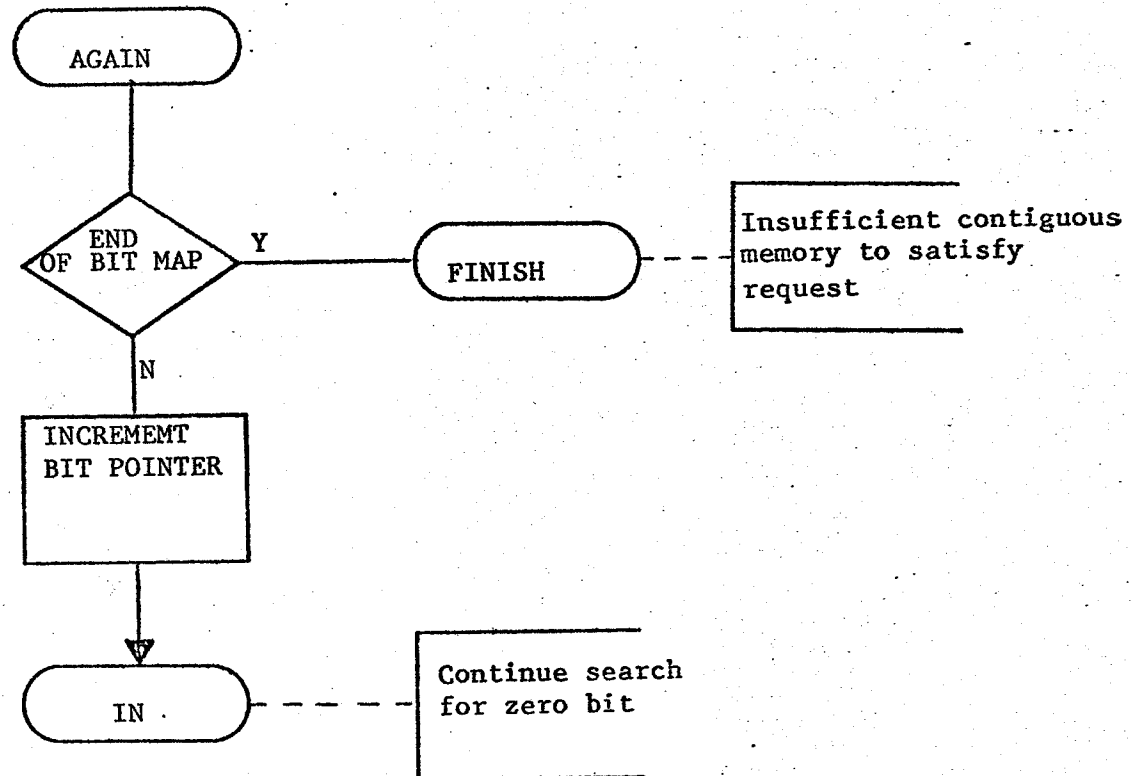
3.4 Computer Program Functional Flow Diagram

DATE

ISSUE

REV

PROGRAM
GETMAINMODULE
DMEMPAGE
1

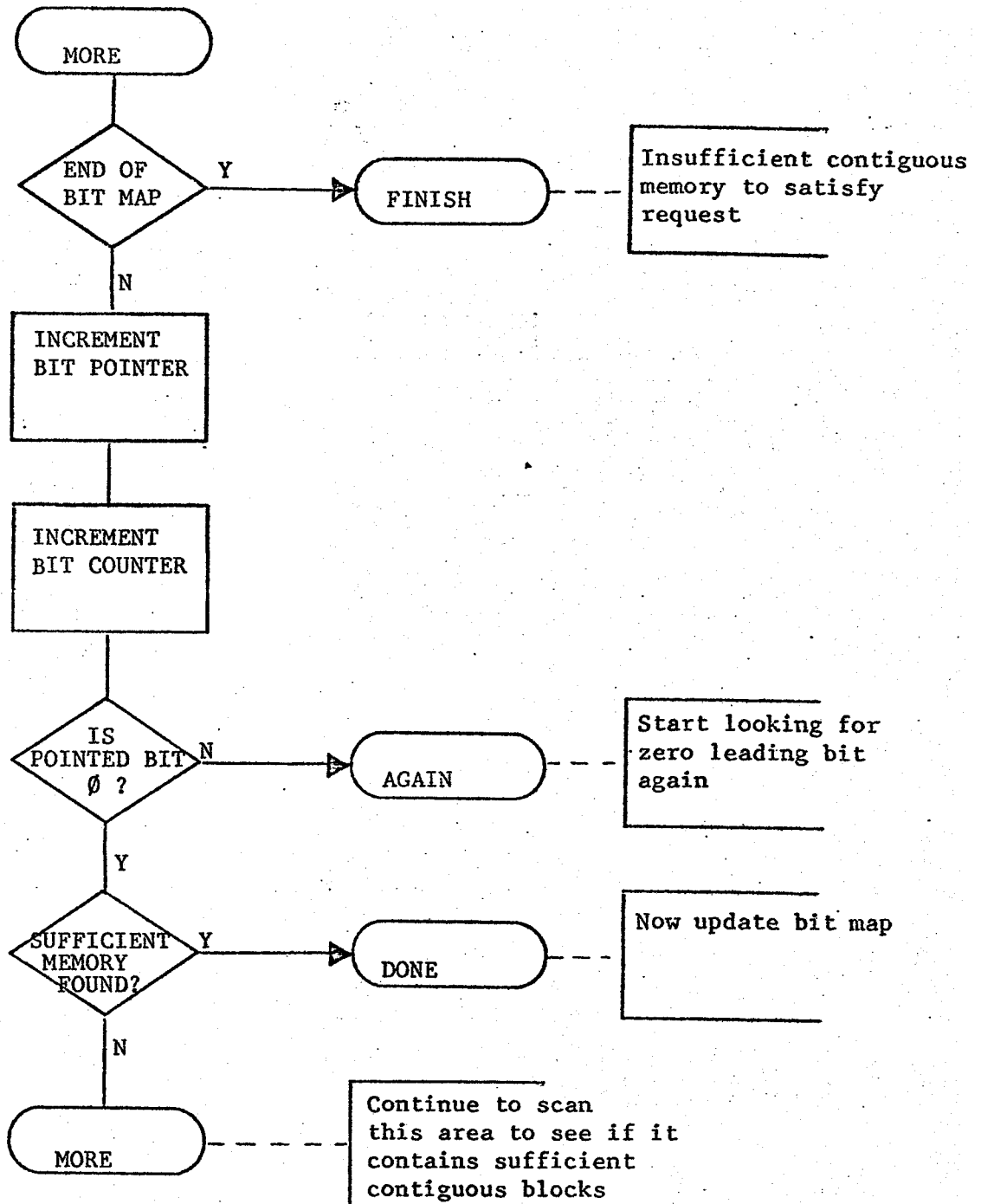


DATE

ISSUE

REV

PROGRAM
GETMAINMODULE
DMEMPAGE
2



DATE

ISSUE

REV

PROGRAM

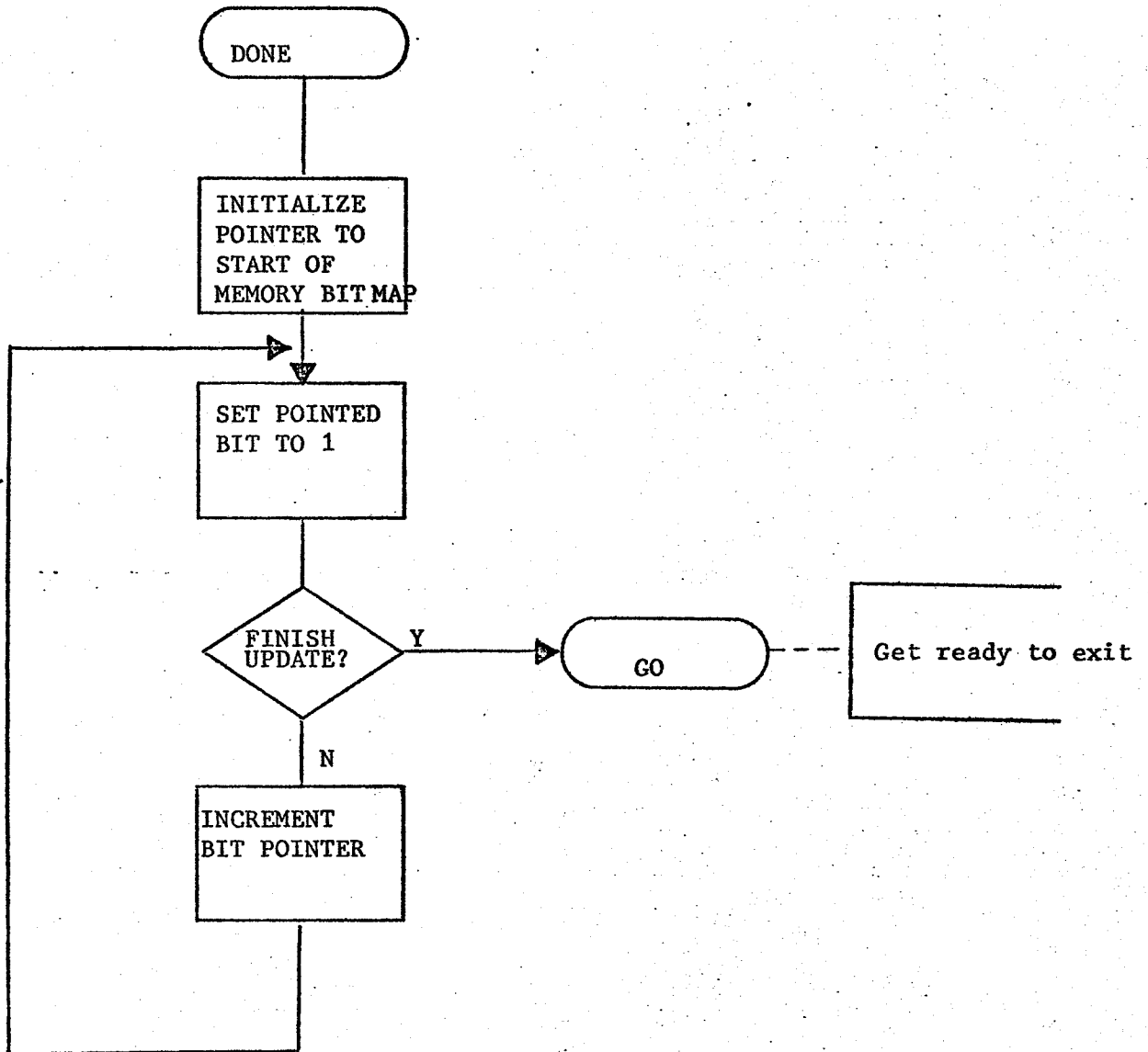
GETMAIN

MODULE

DMEM

PAGE

3

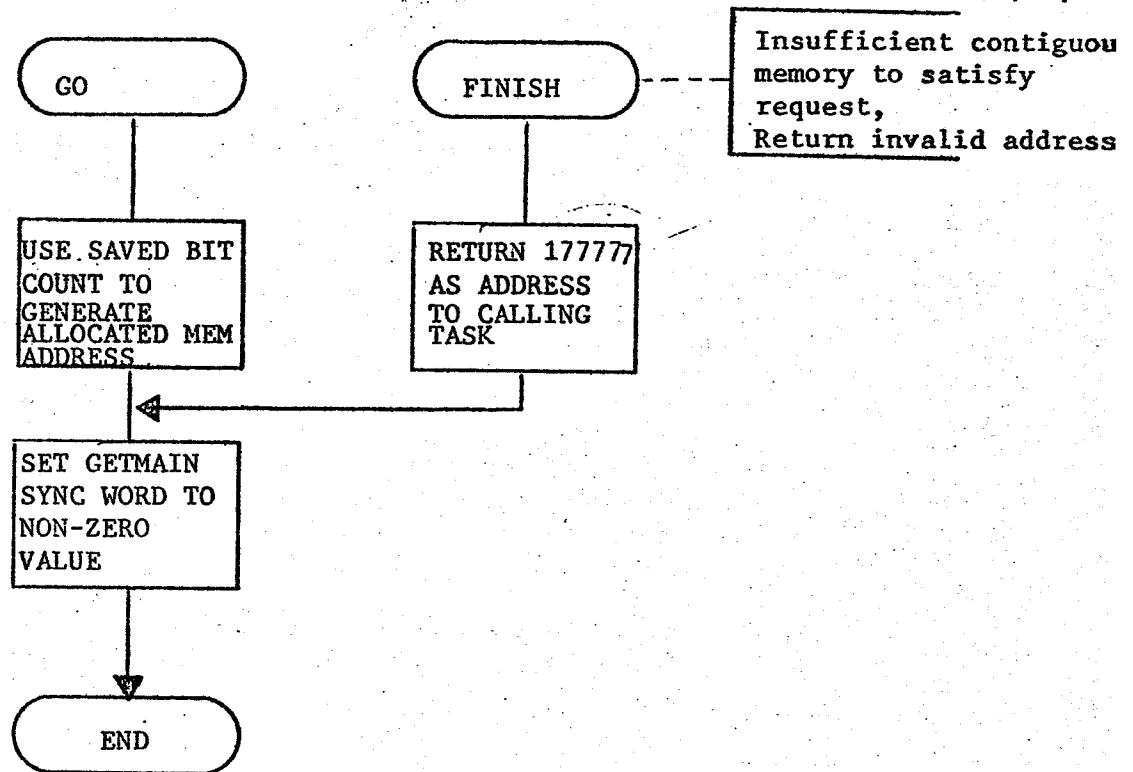


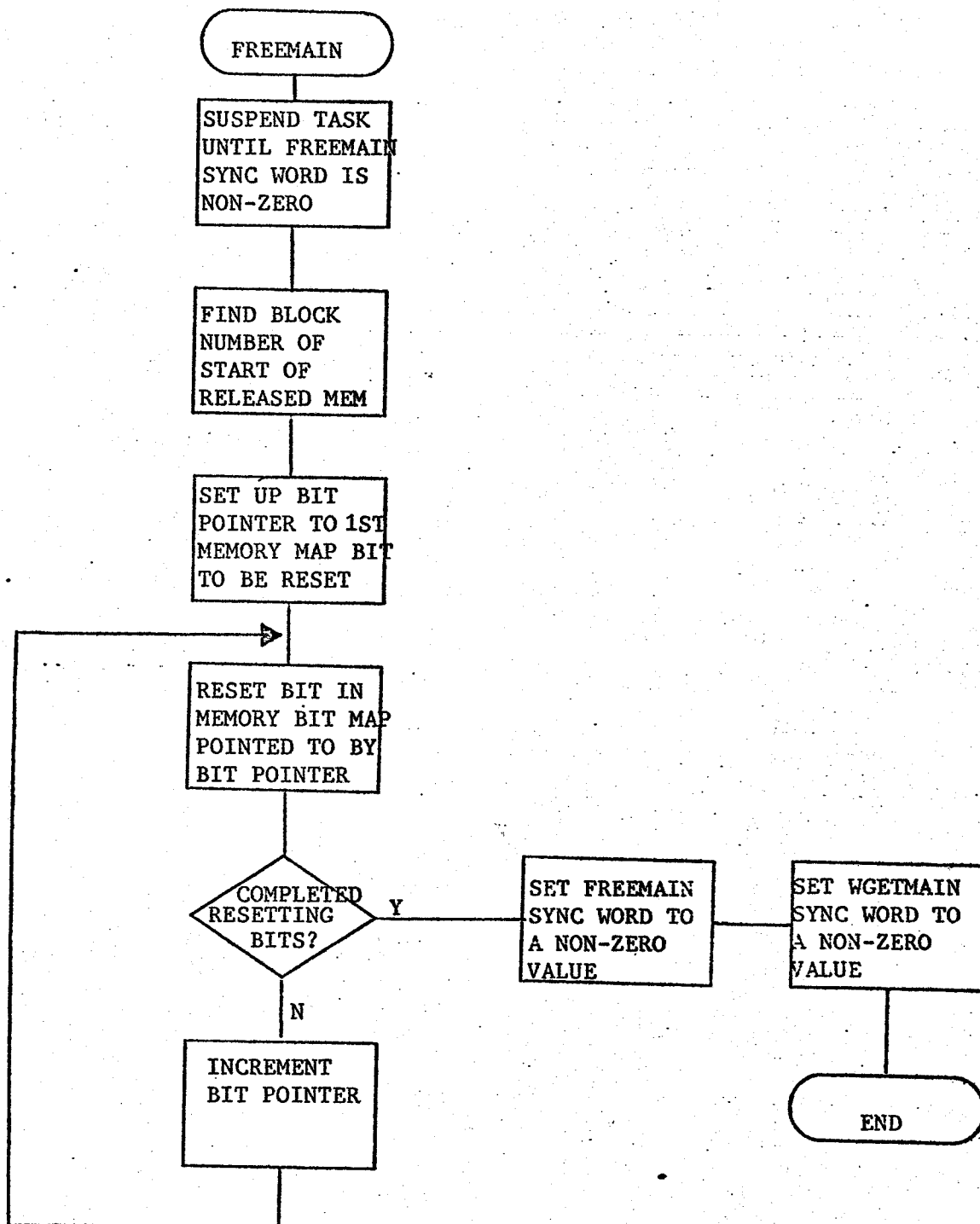
DATE

ISSUE

REV

PROGRAM
GETMAINMODULE
DMEMPAGE
4



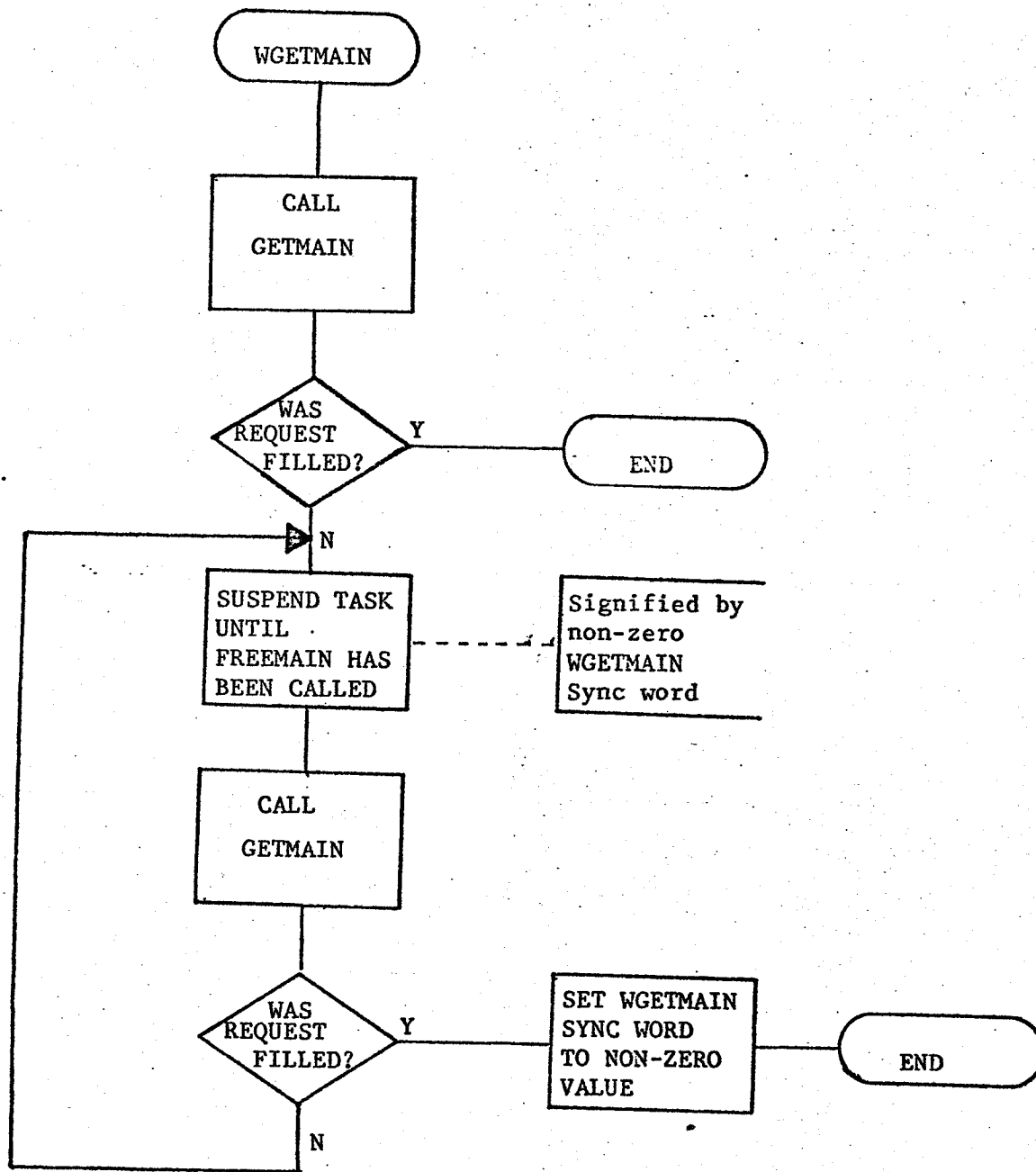


DATE

ISSUE

REV

PROGRAM
FREEMAINMODULE
DNEMPAGE
6



DATE

ISSUE

REV

PROGRAM
WGETMAINMODULE
DMEMPAGE
7



CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

3.4.1

Program Interrupts

In order to maintain the integrity of the bit mask during execution of GETMAIN and FREEMAIN, the bit map is protected by .REC and .XMT system task calls.

.REC ensures that once a task has gained control of a subroutine it will have exclusive use of that subroutine until the subroutine issues a .XMT. Any other task attempting to access a subroutine that is in use will be suspended until such a time as the subroutine is available to that task (i.e., when a subroutine is freed, it is made available to tasks on a priority basis).

WGETMAIN uses a .REC and .XMT commands to queue up tasks that have unfulfilled memory requests. Control is not returned to these calling tasks until their memory requests are fulfilled. These queued tasks will exist in a suspended state and will be activated in a priority basis.

3.4.2

Logic of Subroutine Reference

3.4.2.1

Initialization

Since core requirements will vary from exercise to exercise, the amount of dynamic memory available is not known a priori. After the initialization routine has determined the fixed core requirements, it must generate a memory bit map of available dynamic memory. Each bit in the map will represent a 32 word block. The initialization routine will zero all bits in the bit map (a zero in a bit indicates that the corresponding block is free) and pass the location of the first word in the bit map in DMEN1. DMEN2 will be loaded with the total number of 32 word blocks of available dynamic memory and DMEN3 will contain the location of the first word of dynamic memory. The address of

CODE
IDENT

09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

the first location of dynamic memory must be such that it is divisible by 40 (octal). DMEN1, DMEN2 and DMEN3 will be defined as entry points by the initialization routine.

GETMAIN and FREEMAIN each require a sync word to be set to a non-zero value by the initialization routine which further must define the sync words (SYNØ2, SYNØ3) as entry points.

WGETMAIN requires a sync word to be set to zero by the initialization routine which further must define the sync words (SYNØ7) as an entry point.

3.4.2

Logic of Subroutine Reference

3.4.2.1

Referencing GETMAIN

INPUT:

ACØ: Number of 32 word blocks required

CALL:

PSHJ GETMAIN

OUTPUT:

AC2: Starting address of allocated memory

or

177777 (Insufficient contiguous memory available to satisfy request)

ACØ, AC1, AC3 and carry destroyed.

GETMAIN requires the use of one location on the stack of the calling task. The calling task must define GETMAIN as an external normal. The action of the .REC command as specified in 3.4.1 will be transparent to the calling task. No priority is associated uniquely with GETMAIN, rather it assumes the priority of the calling task.

ANY RESTRICTIVE AND/OR OTHER PROTECTIVE NOTICES, IF ANY, ON THE SHEET FOR WHICH THIS SHEET SERVES AS A CONTINUATION ARE HEREBY INCORPORATED HERE ON.



CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

3.4.2.2

Referencing FREEMAIN

INPUT:

AC0: Address of start of released memory

AC1: Number of blocks to be released

CALL

PSHJ FREEMAIN

OUTPUT:

No messages returned

AC0, AC1, AC2, AC3 and carry destroyed.

FREEMAIN requires the use of three locations on the stack of the calling task. The calling task must define FREEMAIN as an external normal. The action of the .REC command as specified in 3.4.1 will be transparent to the calling task. No priority is associated uniquely with FREEMAIN rather it assumes the priority of the calling task.

3.4.2.3

Referencing WGETMAIN

INPUT:

AC0: Number of 32 word blocks required.

CALL:

PSHJ WGETMAIN

OUTPUT:

AC2: Starting address of allocated memory

AC0, 1, 3 and carry destroyed.

ANY RESTRICTIVE AND/OR OTHER PROTECTIVE NOTICES, IF ANY, ON THE SHEET FOR WHICH THIS SHEET SERVES AS A CONTINUATION ARE HEREBY INCORPORATED HERE ON.



CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

WGETMAIN requires the use of 3 locations on the stack of the calling task. The calling task must define WGETMAIN as an external normal. The action of the .REC command as specified in 3.4.1 will be transparent to the calling task. No priority is associated uniquely with WGETMAIN rather it assumes the priority of the calling task.

3.4.3

Special Control Features

None

4.

DESIGN CONSIDERATIONS

(a) Since this module will use ECLIPSE unique instructions, it will be necessary to assemble the module using the MACRO assembler.

(b) This module uses the facilities of the RDOS/RTOS operating system and therefore cannot run stand-alone.

ANY RESTRICTIVE AND/OR OTHER PROTECTIVE NOTICES, IF ANY, ON THE SHEET FOR WHICH THIS SHEET SERVES AS A CONTINUATION ARE HEREBY INCORPORATED HERE ON.

```
01
02
03
04
05
06
08
09
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
```

```
*****
;
; 01 TITLE
; TITL DMEM          ; DYNAMIC MEMORY REQUEST HANDLER
;
*****
;
; 02 IDENTIFICATION
;
; PURPOSE: THIS PROGRAM ALLOWS TASKS TO OBTAIN
;          AND RELEASE DYNAMIC MEMORY
;
; AUTHOR:  MICHAEL BRETT
;
; PROGRAM LEVEL:  OPERATING SYSTEM
;
; ORIGINAL ISSUE DATE:
;
; ISSUE:  1
;
; DATE OF ISSUE:
;
*****
;
; 03 DESCRIPTION
;
; REFERENCES:
;
; CPPS: 28711
; PID:  28707
;
; NARRATIVE:
;
; WHEN A TASK CALLS GETMAIN, THE DYNAMIC MEMORY BIT
; MAP IS SEARCHED FOR SUFFICIENT CONTIGUOUS MEMORY
; TO SATISFY REQUEST. IF SEARCH IS SUCCESSFUL, THE
; THE BIT MAP IS UPDATED AND THE STARTING ADDRESS OF
; THE ALLOCATED MEMORY IS RETURNED TO THE CALLING
; TASK. IF INSUFFICIENT CONTIGUOUS MEMORY IS FOUND
; TO EXIST, AN INVALID ADDRESS IS RETURNED TO THE
; CALLING TASK.
; WHEN A TASK CALLS FREEMAIN, THE ARGUMENTS IT PASSES
; ARE USED TO RELEASE (RESET) THE APPROPRIATE BITS
; IN THE BIT MASK.
; WHEN A TASK CALLS WGETMAIN, GETMAIN IS CALLED. IF
; SUFFICIENT MEMORY IS AVAILABLE, THE STARTING ADDRESS
; IS RETURNED. IF NOT, THE ROUTINE WILL BE QUEUED
; UP TO RUN AGAIN WHEN MORE MEMORY IS AVAILABLE. THE
; ROUTINE IS EXITED WHEN SUFFICIENT MEMORY IS FOUND.
;
; PERIPHERALS:
;
; NONE
;
; SIZE:  190 WORDS
;
```

```

01 ;*****
02 ;
03 ; 04 CALLING SEQUENCE
04 ;
05 ; ^^^^^GETMAIN^^^^^
06 ;
07 ;         AC0: # OF 32 WORD BLOCKS REQUESTED
08 ;         CALL: PSHJ GETMAIN
09 ;         RETURN: AC2: STARTING ADDRESS OF ALLOCATED MEMORY
10 ;                   OR
11 ;                   177777 INSUFFICIENT MEMORY TO
12 ;                   SATISFY REQUEST
13 ;                   AC0,1,3 AND CARRY DESTROYED
14 ;
15 ; ^^^^^FREEMAIN^^^^^
16 ;
17 ;         AC0: ADDRESS OF START OF RELEASED MEMORY
18 ;         AC1: # OF BLOCKS TO BE RELEASED
19 ;         CALL: PSHJ FREEMAIN
20 ;         RETURN: NO MESSAGES RETURNED
21 ;                   AC0,1,2,3 AND CARRY DESTROYED
22 ;
23 ; ^^^^^WGETMAIN^^^^^
24 ;
25 ;         AC0: # OF 32 WORD BLOCKS REQUESTED
26 ;         CALL: PSHJ WGETMAIN
27 ;         RETURN: AC2: STARTING ADDRESS OF ALLOCATED MEMORY
28 ;                   AC0,1,3 AND CARRY DESTROYED
29 ;
30 ;
31 ;*****
32 ;
33 ; 05 SUBROUTINES
34 ;
35 ;         NONE
36 ;
37 ;*****
38 ;
39 ; 06 EXTERNAL DATA
40 ;
41 ;.EXTN SYN02          ; GETMAIN SYNC WORD
42 ;.EXTN SYN03          ; FREEMAIN SYNC WORD
43 ;.EXTN SYN07          ; WGETMAIN SYNC WORD
44 ;.EXTN DMEN1          ; LOCATION OF START OF BIT MAP
45 ;.EXTN DMEN2          ; TOTAL # OF BLOCKS OF AVAIL. MEM.
46 ;.EXTN .XMT           ; .XMT TASK CALL
47 ;.EXTN .REC           ; .REC TASK CALL
48 ;.EXTN DMEN3          ; START OF DYNAMIC MEMORY
49 ;
50 ;*****
51 ;
52 ; 07 LISTING
53 ;
54 ;
55 ;
56 ;         ; NO .ZREL MEMORY USED
57 ;
58 ;
59 ;.NREL

```

; DYNAMIC MEMORY ALLOCATION GETMAIN

```

00003 DMEM
01
02
03
04
05
06
07
08
09 00000'111000
10 00001'162070
11 077777
12 00003'077777
13
14 00004'126070
15 077777
16 00006'044503
17 00007'050503
18 00010'152400
19 00011'050503
20 00012'024504
21 00013'136070
22 077777
23 00015'131310
24
25
26 00016'000407
27 00017'014472
28 00020'000402
29 00021'000457
30
31 00022'010472
32 00023'175422
33 00024'151400
34 00025'156210
35 00026'000771
36 00027'024463
37 00030'044463
38 00031'024463
39 00032'044463
40 00033'014460
41 00034'000403
42 00035'156010
43 00036'000433
44 00037'014452
45 00040'000402
46 00041'000437
47 00042'175422
48
49 00043'151400
50 00044'010450
51 00045'156210
52 00046'000751
53 00047'014444
54
55
56 00059'000767
57 00051'000401
58
59
60 00052'152400 DONE:

```

GETMAIN:

```

MOV 0,2
ELEF 0,SYN02
.REC
ELDA 1,DMEN2
STA 1,TEMP1
STA 2,TEMP2
SUB 2,2
STA 2,LOOP
LDA 1,.4
ELDA 3,DMEN1
DLSH 1,2
JMP IN
DSZ TEMP1
JMP .+2
JMP FINISH
ISZ LOOP
INCZ 3,3,SZC
INC 2,2
SZB 2,3
JMP AGAIN
LDA 1,TEMP2
STA 1,TEMP3
LDA 1,LOOP
STA 1,LOC
DSZ TEMP3
JMP MORE
BTO 2,3
JMP GO
DSZ TEMP1
JMP .+2
JMP FINISH
INCZ 3,3,SZC
INC 2,2
ISZ LOOP
SZB 2,3
JMP AGAIN
DSZ TEMP3
JMP MORE
JMP DONE
SUB 2,2

```

AGAIN:

MORE:

DONE:

```

; SAVE AC0
; AC0 = ADDRESS OF GETMAIN SYNC WORD
; SUSPEND TASK IF SYNC WORD 0
; SYNC WORD = 0 IF GETMAIN IN USE
; DMEN2= # OF BITS IN BIT MASK
; TEMP1 USED FOR SEARCH TERMINATION
; TEMP2 IS # OF BLOCKS REQUESTED
; LOOP IS USED FOR BIT COUNTING
; AC1=4, USED TO INITIAL. BIT POINTER
; DMEN1 = FIRST WORD OF BIT MAP
; SHIFT AC2,3 4 LEFT
; AC2,3 NOW CONTAINS BIT POINTER TO
; START OF MEMORY MASK
; START SEARCH
; END OF BIT MAP ?
; NO, CONTINUE SEARCH
; YES, RETURN INVALID ADDRESS TO CALL
; SINCE INSUFFICIENT CONTIGUOUS MEM. AVAILABLE
; INCREMENT BIT COUNTER
; INCREMENT LOW ORDER WORD
; OVERFLOW FROM AC3, AC2=AC2+1
; SKIP IF POINTED BIT = 0
; CONTINUE SEARCH FOR LEADING ZERO
; TEMP2 = # OF BLOCKS REQUESTED
; TEMP3=COUNTER FOR FREE BLOCKS
; START OF CURRENT FREE AREA
; KEEP IT
; HAS ONLY 1 BLOCK BEEN REQUESTED
; NO MULTIPLE BLOCKS, CONTINUE SEARCH
; YES UPDATE MASK
; JUMP TO EXIT ROUTINE
; END OF BIT MAP
; NO CONTINUE SEARCH
; INSUFFICIENT MEMORY AVAILABLE
; INCREMENT LOW BIT POINTER
; AND SKIP IF NO OVERFLOW
; OVERFLOW, INC HIGH ORDER POINTER
; INCREMENT BIT COUNTER
; SKIP IF BIT=0
; NON-ZERO, START SEARCH AGAIN
; OK, BIT FREE
; HAS SUFFICIENT MEMORY FOUND
; NO, CONTINUE IN LOOP
; SUFFICIENT CONTIGUOUS MEMORY HAS BEEN
; FOUND TO SATISFY REQUEST
; UPDATE BIT MASK
; RE-INITIALIZE BIT POINTER

```

```

0004 DMEN
01 00053'136070      ELDA 3,DMEN1      ; GET ADDRESS OF START OF MAP
02      000014'
03 00055'024441      LDA 1,.4
04 00056'131310      DLSH 1,2      ; BIT POINTER INITIALIZED
05 00057'020436      LDA 0,LOC      ; BIT # OF ALLOCATED MEMORY
06 00060'117022      ADDZ 0,3,SZC    ; ADD BASE ADDRESS + COUNTER TO GET
07                                     ; ABSOLUTE BIT ADDRESS
08 00061'151400      INC 2,2        ; IF OVERFLOW, INCREMENT AC2
09
10 00062'156010      UPDATE: BTO 2,3      ; SET BIT TO 1
11 00063'014427      DSZ TEMP2          ; FINISHED UPDATE ?
12 00064'000402      JMP .+2           ; NO
13 00065'000404      JMP CO           ; YES, GET READY TO LEAVE
14 00066'175422      INCZ 3,3,SZC      ; INCREMENT BIT POINTER
15 00067'151400      INC 2,2
16 00070'000772      JMP UPDATE        ; CONTINUE IN LOOP
17 00071'020424      GO: LDA 0,LOC      ; GET BIT COUNT OF START OF ALLOCATED MEMORY
18 00072'101120      MOVZL 0,0        ; MOVE AC0
19 00073'101410      HXL 1,0          ; 5 LEFT TO GET DISPLACEMENT
20                                     ; OF ALLOCATED MEMORY FROM START OF DYN. MEM.
21 00074'132070      ELDA 2,DMEN3      ; DMEN3 = START OF DYN. MEM.
22      077777
23
24 00076'143000      ADD 2,0          ; AC0=ABSOLUTE ADDRESS OF START OF
25                                     ; ALLOCATED MEMORY
26 00077'000402      JMP .+2           ; SKIP OVER NO MEMORY ROUTINE
27 00100'102000      FINISH: ADC 0,0    ; AC0=177777
28 00101'126000      ADC 1,1          ; SET AC1 TO NON-ZERO FOR SYNC
29 00102'111000      MOV 0,2          ; SAVE AC0
30 00103'162070      ELEF 0,SYN02     ; AC0 = ADDRESS OF GETMAIN SYNC WORD
31      000002'
32 00105'077777      .XMT             ; PUT NON-ZERO MESSAGE INTO SYNC
33 00106'000401      JMP .+1          ; TO FREE GETMAIN FOR OTHER TASKS
34 00107'141000      MOV 2,0          ; RESTORE AC0
35 00110'117710      POPJ            ; RETURN TO TASK
36                                     ; STORAGE FOR GETMAIN
37 00111'000000      TEMP1: 0
38 00112'000000      TEMP2: 0
39 00113'000000      TEMP3: 0
40 00114'000000      LOOP: 0
41 00115'000000      LOC: 0
42 00116'000004      .4: 4

```

```

10003 DMEM
01
02
03
04 ; MEMORY DE-ALLOCATION FREEMAIN
05
06 .ENT FREEMAIN
07
08
09 FREEMAIN:
10 00117'107110 PSH 0,1 ; SAVE AC0,1
11 00120'162070 ELEF 0,SYN03 ; LOAD ADDRESS OF FREEMAIN SYNC WORD
12 077777
13 00122'000003' .REC ; IF FREEMAIN NOT IN USE (SYN NON-ZERO)
14 ; CONTINUE EXECUTION
15 ; IF FREEMAIN IN USE , SUSPEND TASK
16 ; UNTIL FREEMAIN RELEASED
17 00123'123210 POP 1,0 ; GET AC0,1 BACK
18 00124'044435 STA 1,FREED ; FREED = # OF BLOCKS RELEASED
19 00125'126070 ELDA 1,DMEN3 ; LOAD ADD. OF START OF DYN. MEM.
20 000075'
21 00127'122400 SUB 1,0 ; GET RELATIVE ADDRESS FROM ST. OF DY. MEM.
22 00130'101510 HXR 1,0 ; SHIFT AC0 5 RIGHT TO
23 00131'101220 MOVZR 0,0 ; GET BLOCK NUMBER OF FIRST RELEASED BLOCK
24 00132'136070 ELDA 3,DMEN1 ; LOAD ADDRESS OF START OF BIT MAP
25 000054'
26 00134'024762 LDA 1,.4 ; AC1 = 4
27 00135'152400 SUB 2,2 ; AC2 = 0
28 00136'131310 DLSH 1,2 ; GET POINTER TO FIRST BIT OF MAP
29 00137'117022 ADDZ 0,3, SZC ; GET PTER TO FIRST BIT TO BE FREED
30 00140'151400 INC 2,2 ; INC AC2 IF OVERFLOW FROM AC3
31 00141'156110 REDO: BTZ 2,3 ; SET BIT TO ZERO
32 00142'014417 DSZ FREED ; FINISHED UPDATING BIT MAP ?
33 00143'000402 JMP .+2 ; NO
34 00144'000404 JMP OUT ; YES
35 00145'175422 INCZ 3,3, SZC ; INCREMENT AC3
36 00146'151400 INC 2,2 ; INC 2 IF OVERFLOW FROM AC3
37 00147'000772 JMP REDO ; CONTINUE CLEARING BITS
38 00150'162070 OUT: ELEF 0,SYN03 ; LOAD ADDRESS OF GETMAIN SYNC WORD
39 000121'
40 00152'000105' .XMT ; PUT A NON-ZERO WORD INTO SYNC
41 00153'000401 JMP .+1 ; TO FREE FREEMAIN FOR OTHER TASKS
42 00154'162070 ELEF 0,SYN07 ; GET WGETMAIN SYNC
43 077777
44 00156'000152' .XMT ; TELL WGETMAIN THAT FREEMAIN WAS
45 ; CALLED
46 00157'000401 JMP .+1 ; DON'T CARE IF MESSAGE IN USE
47 00160'117710 POPJ ; RETURN TO CALLING TASK
48 ; STORAGE FOR FREEMAIN
49 00161'000000 FREED: 0
50 00162'000003 .5: 5
51

```

10006 BTEM

; QUEUED MEMORY ALLOCATION WCETMAIN

```

01
02
03      .ENT WCETMAIN
04
05
06      WCETMAIN:
07 00163'103110      PSH 0,0      ; SAVE # OF BLOCKS
08 00164'102670      PSHJ GETMAIN ; CALL GETMAIN
09      077613
10
11 00166'151113      MOVL# 2,2,SNC ; AC2 CONTAINS ADDRESS ON RETURN
12 00167'000421      JMP AHEAD    ; VALID ADDRESS RETURNED ?
13
14
15 00170'162070 HERE: ELEF 0,SYN07 ; YES, WE CAN LEAVE
16      000155'      ; NO, WE MUST SUSPEND UNTIL A
17 00172'000122'      .REC        ; FREEMAIN IS EXECUTED
18
19 00173'103210      POP 0,0      ; GET ADDRESS OF QUEUE SYNC WORD
20 00174'103110      PSH 0,0      ; SUSPEND TASK UNTIL FREEMAIN
21 00175'162670      PSHJ GETMAIN ; IS EXECUTED
22      077602
23 00177'151112      MOVL# 2,2,SZC ; GET # OF BLOCKS BACK
24 00200'000770      JMP HERE     ; SAVE IT , WE MAY NEED IT AGAIN
25 00201'153110      PSH 2,2      ; TRY AGAIN FOR MEMORY
26 00202'145000      MOV 2,1
27 00203'162070      ELEF 0,SYN07 ; HAVE WE GOT IT ?
28      000171'      ; NO SUSPEND TILL A FREEMAIN
29 00205'000156'      .XMT        ; SAVE ALLOCATED ADDRESS
30
31 00206'000401      JMP .+1       ; AC1 GETS NON-ZERO VALUE
32 00207'153210      POP 2,2      ; GET ADDRESS OF QUEUE WORD
33 00210'103210 AHEAD: POP 0,0    ; TRANSMIT TO ALLOW OTHER
34 00211'117710      POPJ        ; QUEUE TASKS TO RUN
35      .END                    ; DON'T CARE ABOUT ERROR RETURN
                                ; GET ADDRESS BACK
                                ; CLEAR UP USER STACK
                                ; RETURN TO CALL

```

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS

0007 DMEM

AGAIN 000017'		3/27	3/35	3/52		
AHEAD 000210'		6/12	6/33			
DMEN1 000133'	XN	2/44	3/21	4/01	5/24	
DMEN2 000095'	XN	2/45	3/14			
DMEN3 000126'	XN	2/48	4/21	5/19		
DONE 000052'		3/57	3/60			
FINIS 000100'		3/29	3/46	4/27		
FREED 000161'		5/18	5/32	5/49		
FREEN 000117'	EN	5/05	5/09			
GETMA 000090'	EN	3/05	3/38	6/08	6/21	
GO 000071'		3/43	4/13	4/17		
HERE 000170'		6/15	6/24			
IN 000025'		3/26	3/34			
LOC 000115'		3/39	4/05	4/17	4/41	
LOOP 000114'		3/19	3/31	3/38	3/50	4/40
MORE 000037'		3/41	3/44	3/56		
OUT 000150'		5/34	5/38			
REDO 000141'		5/31	5/37			
SYN02 000104'	XN	2/41	3/10	4/30		
SYN03 000151'	XN	2/42	5/11	5/38		
SYN07 000204'	XN	2/43	5/42	6/15	6/27	
TEMP1 000111'		3/16	3/27	3/44	4/37	
TEMP2 000112'		3/17	3/36	4/11	4/38	
TEMP3 000113'		3/37	3/40	3/53	4/39	
UPDAT 000062'		4/10	4/16			
WCETM 000163'	EN	6/03	6/06			
.4 000116'		3/20	4/03	4/42	5/26	
.5 000162'		5/50				
.REC 000172'	XN	2/47	3/12	5/13	6/17	
.XMT 000205'	XN	2/46	4/32	5/40	5/44	6/29



LITTON SYSTEMS (CANADA) LIMITED
Litton 25 Cityview Drive, Rexdale, Ontario, Canada M9W 5A7

NUMBER

28746

REV

PAGE

51

CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

TITLE

ANPT MODULE TEST DOCUMENT

DYNAMIC MEMORY ALLOCATION MODULE TEST

REVISIONS

REV	ECO	CHANGE DATE	CHANGE DESCRIPTION	APPROVAL
			<p>copy</p> <p>REFERENCE ONLY</p>	

LSL RELEASE: DRN 36938 DATE 12-8-76

APPROVALS

PREPARED

M. Brett

M. Brett

SYSTEMS ENGINEER

R.J. Taylor

R.J. Taylor

PROJECT ENGINEER

S.J. Lipton

S.J. Lipton

SOFTWARE ENGINEER

R. Territo

R. Territo

RELEASE APPROVAL

S.J. Lipton

S.J. Lipton

THIS DOCUMENT CONTAINS INFORMATION
PROPRIETARY TO LITTON SYSTEMS (CANADA)
LIMITED. ANY REPRODUCTION, DISCLOSURE
OR USE OF THIS DOCUMENT IS EXPRESSLY
PROHIBITED EXCEPT AS LITTON SYSTEMS
(CANADA) LIMITED MAY OTHERWISE AGREE
IN WRITING.



CODE
IDENT

09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

1. SCOPE

1.1 Identification

This document establishes the description of the DYNAMIC MEMORY ALLOCATION MODULE TEST computer program referred to as DMENT.

1.2 Functional Summary

This computer program will run a series of tests on the GETMAIN, FREEMAIN and WGETMAIN subroutines (the DYNAMIC MEMORY ALLOCATION MODULE (DMEM) and output the results of these tests.



LITTON SYSTEMS (CANADA) LIMITED
Litton 25 Cityview Drive Rexdale, Ontario, Canada M9W 5A7

NUMBER

28746

REV

PAGE

53

CODE
IDENT

09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

2. APPLICABLE DOCUMENTS

2.1 Government Documents

None

2.2 L.S.L. Documents

C.P.P.S. # 28711

P.I.D. # 28707

Software #

ANY RESTRICTIVE AND/OR OTHER PROTECTIVE NOTICES, IF ANY, ON THE SHEET FOR WHICH THIS SHEET SERVES AS A CONTINUATION ARE HEREBY INCORPORATED HERE ON.



CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

3. TEST REQUIREMENTS

3.1 GETMAIN Test

This test must show that GETMAIN properly processes:

- (a) single block requests
- (b) multiple block requests
- (c) allocating blocks out of a fragmented bit map
- (d) requests for more contiguous memory than is currently available and returns the proper allocated starting address in each case.

3.2 FREEMAIN Test

This test must show that FREEMAIN properly resets bits in the memory bit map as specified by the arguments passed to it by the test routine.

3.3 WGETMAIN Test

This test must show that WGETMAIN acquires memory when memory is available and suspends the calling task until sufficient contiguous memory becomes available to satisfy the request. It must satisfy Requirements 3. (a), (b), and (c) of GETMAIN.



CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

4. DYNAMIC MEMORY REQUEST HANDLER TEST

4.1 Test Description

The test module initializes all the external data and sync words needed by the DMEM module as well as acquiring a stack. It then issues a series of lettered requests to GETMAIN, FREEMAIN and WGETMAIN and prints out the responses of the subroutine.

The memory map used in the test is 16 bits long and the start of dynamic memory was set at location 50000. All outputted values are in octal.

4.2 Test Output Explanation

GETMAIN : A is a GETMAIN request for 1 block from a clear bit map. The bit map and the allocated starting address are consistent with the request (Test Requirement 3.1(a)).

GETMAIN : B is a GETMAIN request for 5 blocks. The allocated starting address is consistent with the bit map. (Test Requirement 3.1 (b)).

FREEMAIN : A is a release of the first requested block. The proper bit in the memory map is updated (Test Requirement 3.2).

GETMAIN : C is a GETMAIN request for 4 blocks from a fragmented bit map. The allocated starting address and the updated bit are consistent with the expected result (Test Requirements 3.1 (b),(c)).

GETMAIN : D is a GETMAIN request for 10 blocks (octal) from a bit map where there are insufficient contiguous blocks to satisfy request. An invalid starting address is returned and the bit map is unchanged (Test Requirement 3.1 (d)).

CODE
IDENT

09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

A WGETMAIN is now issued for 10 blocks (octal). At the time the request was issued, there was insufficient contiguous memory available so WGETMAIN suspends itself.

The WGETMAIN test task becomes active and releases the C request. FREEMAIN: C is a release of 4 blocks (Test Requirement 3.2). After FREEMAIN releases the 4 blocks, it signals WGETMAIN that the blocks were released. Since the task that called WGETMAIN is of higher priority than the WGETMAIN test task, WGETMAIN is re-activated immediately and since sufficient contiguous memory is now available for that request, the request is filled. The output of the WGETMAIN information is suspended until the previous FREEMAIN information is outputted. The bit map outputted by FREEMAIN: C reflects the bit map resulting from the WGETMAIN request, since control was passed immediately to WGETMAIN, before the FREEMAIN information was printed. This shows that WGETMAIN was properly queued to run upon a FREEMAIN access.

The GETMAIN: D request shows that WGETMAIN was able to obtain 10 blocks (octal) with the proper starting address and bit map (Requirement 3.3).

GETMAIN: E is a WGETMAIN request for 3 blocks from a bit map where sufficient contiguous memory to fill the request is available. No queuing of the task occurs and the proper address and bit map are returned (Test Requirement 3.3).

FREEMAIN: D is a release request for 10 blocks that were obtained by a WGETMAIN. The proper bits were set in the bit mask (Test Requirement 3.2).

4.3

Test Output

ANY RESTRICTIVE AND/OR OTHER PROTECTIVE NOTICES, IF ANY, ON THE SHEET FOR WHICH THIS SHEET SERVES AS A CONTINUATION ARE HEREBY INCORPORATED HERE ON.

RTOS REV 4.00
DYNAMIC MEMORY TEST

GETMAIN: A
BLOCKS NEEDED : 000001
UPDATED BIT MAP: 100000
ALLOCATED STARTING ADDRESS: 005000

GETMAIN: B
BLOCKS NEEDED : 000005
UPDATED BIT MAP: 176000
ALLOCATED STARTING ADDRESS: 005040

FREEMAIN: A
BLOCKS RELEASED : 000001
UPDATED BIT MAP: 076000

GETMAIN: C
BLOCKS NEEDED : 000004
UPDATED BIT MAP: 077700
ALLOCATED STARTING ADDRESS: 005300

FREEMAIN: B
BLOCKS RELEASED : 000005
UPDATED BIT MAP: 001700

GETMAIN: D
BLOCKS NEEDED : 000010
UPDATED BIT MAP: 001700
ALLOCATED STARTING ADDRESS: 177777

FREEMAIN: C
BLOCKS RELEASED : 000004
UPDATED BIT MAP: 177400

*** Actual Bit Map is 000000 ***
*** See Test Documentation ***

GETMAIN: D
BLOCKS NEEDED : 000010
UPDATED BIT MAP: 177400
ALLOCATED STARTING ADDRESS: 005000

GETMAIN: E
BLOCKS NEEDED : 000003
UPDATED BIT MAP: 177740
ALLOCATED STARTING ADDRESS: 005400

FREEMAIN: D
BLOCKS RELEASED : 000010
UPDATED BIT MAP: 000340
END OF TEST



CODE
IDENT 09691

DOCUMENT
TYPE

SOFTWARE DOCUMENT

4.4

Test Program

See following pages.

ANY RESTRICTIVE AND/OR OTHER PROTECTIVE NOTICES, IF ANY, ON THE SHEET FOR WHICH THIS SHEET SERVES AS A CONTINUATION ARE HEREBY INCORPORATED HERE ON.

0001 DMENT MACRO REV 04.00

15:12:29 07/22/76

```

01 ;*****
02 ; NAME: DMENT.SR
03 ;
04 ; DESCRIPTION: DYNAMIC MEMORY MODULE TEST
05 ;
06 ; REVISION HISTORY:
07 ; REV. DATE
08 ;
09 ; 00 07/14/76
10 ;
11 ; AUTHOR: MICHAEL BRETT
12 ;
13 ;*****
14
15 .TITL DMENT
16
17 ; DMEM.RB MUST BE INCLUDED AT RLDR TIME
18
19 ; NO ZREL MEMORY IS REQUIRED FOR THIS PROGRAM
20
21 ; MINIMUM ENVIROMENT
22 ; RDOS/RTOS WITH CSTAK
23 ; ECLIPSE WITH A TTO
24
25
26 000001 .TXIM 1
27 001402 .CONN TASK,3*400+2
28 .ENT START
29 .ENT SYN02,DMEN1,DMEN2,DMEN3,SYN03,SYN07
30 .EXTN GETMAIN,FREEMAIN,WCETMAIN
31 .EXTN .KILAD,.KILL,.PRI,.TASK,CSTAK,.REC,.XMT
32
33 .NREL
34
35
36 ; MASTER TEST ROUTINE
37
38
39 000000'102000 START: ADC 0,0 ; SET AC0=-1
40 000001'040523 STA 0,SYN02 ; INITIALIZE GETMAIN SYNC
41 000002'040523 STA 0,SYN03 ; INITIALIZE FREEMAIN SYNC
42 000003'040524 STA 0,MSYNC ; WRITE PROTECTION SYNC
43 000004'102400 SUB 0,0
44 000005'040521 STA 0,SYN07 ; INITIALIZE WCETMAIN SYNC
45 000006'040522 STA 0,BMP ; INITIALIZE BIT MAP TO ZERO
46 000007'162470 ELEF 0,BMP ; GET ADDRESS OF BIT MAP
47 0000120
48 000011'040510 STA 0,DMEN1 ; STORE IT IN DMEN1
49 000012'006017 .SYSTM
50 000013'021052 .GCHN ; GET A FREE CHANNEL
51 000014'009764 JMP START ; NO FREE CHANNEL
52 000015'050514 STA 2,CHNUM ; STORE CHANNEL NUMBER
53 000016'122470 ELDA 0,TTOD ; GET TTO NAME
54 0000464
55 000020'126400 SUB 1,1
56 000021'006017 .SYSTM
57 000022'014077 .OPEN 77 ; OPEN CHANNEL FOR TTO
58 000023'000462 JMP END ; ERROR RETURN
59 000024'122470 ELDA 0,M1 ; GET MESSAGE POINTER
60 0000442

```

```

0002 DMENT
01 00026'030503 LDA 2,CHNUM ; GET CHANNEL #
02 00027'006017 .SYSTM ; WRITE HEADER
03 00030'017077 .WRL 77
04 00031'063077 HALT ; THIS SHOULD NOT HAPPEN
05
06 00032'106070 EJSR GSTAK ; GET A STACK FROM RTOS/RDCS
07 077777
08 00034'102520 SUBZL 0,0 ; AC0=1
09 00035'077777 .PRI ; MAKE THIS TASK PRIORITY 1
10
11
12
13
14 ; START OF TESTS
15 ; THE TEST FORMAT IS TO
16 ; LOAD THE ADDRESS OF THE
17 ; REQUEST STORAGE AND THEN TO CALL THE
18 ; APPROPRIATE HANDLING ROUTINE
19 00036'162470 ELEF 0,A ; GET ADDRESS OF 'A' REQUEST
20 000302
21 00040'004472 JSR PT1 ; GET HANDLER
22
23 00041'162470 ELEF 0,B ; GET ADDRESS OF 'B' REQUEST
24 000302
25 00043'004467 JSR PT1 ; GET HANDLER
26
27 00044'162470 ELEF 0,A ; GET ADDRESS OF 'A' REQUEST
28 000274
29 00046'106470 EJSR PT2 ; RELEASE HANDLER
30 000217
31
32 00050'162470 ELEF 0,C ; GET ADDRESS OF 'C' REQUEST
33 000276
34 00052'004460 JSR PT1 ; GET HANDLER
35
36 00053'162470 ELEF 0,B ; GET ADDRESS OF 'B' REQUEST
37 000270
38 00055'106470 EJSR PT2 ; RELEASE HANDLER
39 000210
40
41 00057'162470 ELEF 0,D ; GET ADDRESS OF 'D' REQUEST
42 000272
43 00061'004451 JSR PT1 ; GET HANDLER
44
45 ; CONVERT PSHJ GETMAIN TO
46 ; PSHJ WCETMAIN IN PT1
47 00062'162070 ELEF 0,WCETMAIN ; GET ADDRESS OF WCETMAIN
48 077777
49 00064'142470 ESTA 0,INSERT+1 ; CHANGE PSHJ COMMAND
50 000052
51
52 ; ALL CALLS TO PT1
53 ; WILL NOW BE CALLS TO
54 ; WCETMAIN INSTEAD OF GETMAIN DIRECTLY
55 00066'162070 ELEF 0,2 ; AC0=2
56 000002
57 00070'166470 ELEF 1,TASK1 ; GET ADDRESS OF SECOND TASK
58 000416
59 00072'077777 .TASK ; CREATE A WCETMAIN TASK OF LOWER PRIORITY
60 ; THAN MAIN

```

```

0903 DMENT
01 00073'063077      HALT                ; SHOULD NOT GET HERE
02
03 00074'162470      ELEF 0,D              ; GET ADDRESS OF 'D' REQUEST
04      000255
05 00076'004434      JSR PT1              ; GET HANDLER
06
07 00077'162470      ELEF 0,E              ; GET ADDRESS OF 'E' REQUEST
08      000255
09 00101'004431      JSR PT1              ; GET HANDLER
10
11 00102'162470      ELEF 0,D              ; GET ADDRESS OF 'D' REQUEST
12      000247
13 00104'004562      JSR PT2              ; RELEASE HANDLER
14
15 00105'122470 END:  ELDA 0,T6            ; GET MESSAGE POINTER
16      000351
17 00107'030422      LDA 2,CHNUM          ; GET CHANNEL NUMBER
18 00110'006017      .SYSTM              ; WRITE OUT TRAILER MESSAGE
19 00111'017077      .WRL 77
20 00112'000404      JMP .+4
21 00113'006017      .SYSTM              ; CLOSE THE CHANNEL
22 00114'014477      .CLOSE 77
23 00115'000401      JMP .+1
24 00116'006017      .SYSTM              ; STOP THIS PROGRAM
25 00117'004400      .RTN                ; RETURN TO CLI
26 00120'063077      HALT                ; RDOS/RTOS COMPATIBLE
27
28
29
30 00121'000000 DMEN1: 0                  ; CONSTANT STORAGE
31 00122'000020 DMEN2: 20                ; FOR ADDRESS OF MAP
32 00123'005000 DMEN3: 5000              ; FOR # OF BLOCKS
33 00124'000000 SYN02: 0                  ; START OF DYNAMIC MEMORY
34 00125'000000 SYN03: 0                  ; GETMAIN SYNC
35 00126'000000 SYN07: 0                  ; FREEMAIN SYNC
36 00127'000000 MSYNC: 0                  ; WGETMAIN SYNC
37 00130'000000 BMP: 0                    ; WRITE PROTECTION SYNC
38 00131'000000 CHNUM: 0                  ; BIT MAP
39
40
41
42
43 00132'054460 PT1:  STA 3,BACK          ; GETMAIN HANDLER
44 00133'040460      STA 0,TEMP          ; SAVE RETURN ADDRESS
45 00134'034457      LDA 3,TEMP          ; FOR ARGUMENT PASSING
46 00135'021401      LDA 0,1,3          ; GET BASE ADDRESS
47 00136'102270 INSERT: PSHJ GETMAIN      ; GET # OF BLOCKS NEEDED
48      077777                          ; CALL GETMAIN
49 00140'153110      PSH 2,2              ; SAVE AC2
50 00141'162470      ELEF 0,MSYNC        ; GET MASTER SYNC
51      077765
52 00143'077777      .REC                ; RECEIVE IT SO PT2 CAN FINISH
53 00144'126000      ADC 1,1              ; SEND IT BACK OUT
54 00145'077777      .XMT
55 00146'063077      HALT
56
57
58
59 00147'153210      POP 2,2              ; GET AC2 BACK
60 00150'034443      LDA 3,TEMP          ; GET STORAGE BASE ADD.

```

```

0004 DIENT
01 00151'051402 STA 2,2,3 ; STORE STARTING ADDRESS
02 ; NOW INSERT LETTER INTO
03 ; OUTPUT STRING
04 00152'030442 LDA 2,NUM1 ; GET BYTE COUNT IN MESSAGE
05 00153'122470 ELDA 0,T1 ; GET MESSAGE POINTER
06 000204 ;
07 00155'113000 ADD 0,2 ; GET ABSOLUTE BYTE COUNT
08 00156'025400 LDA 1,0,3 ; GET ASCII FOR LETTER
09 00157'147010 STB 2,1 ; STORE LETTER IN STRING
10 ; NOW OUTPUT STRING
11 00160'024435 LDA 1,NUMB1 ; # OF BYTES TO BE OUTPUTTED
12 00161'030750 LDA 2,CHNUM ; GET CHANNEL NUMBER
13 00162'090017 .SYSTEM ; WRITE OUT MESSAGE
14 00163'016477 .WRS 77
15 00164'090721 JMP END ; ERROR RETURN
16 00165'034426 LDA 3,TEMP ; GET STORAGE BASE ADD.
17 00166'021401 LDA 0,1,3 ; GET # OF BLOCKS REQUESTED
18 00167'004427 JSR CODE ; OUTPUT IT
19 ; NOW WRITE ABOUT MAP
20 00170'122470 ELDA 0,T2 ; GET BYTE POINTER TO MESSAGE
21 000210 ;
22 00172'030737 LDA 2,CHNUM ; GET CHANNEL NUMBER
23 00173'006017 .SYSTEM ; WRITE MESSAGE
24 00174'017077 .WRL 77
25 00175'090710 JMP END ; ERROR RETURN
26 00176'020732 LDA 0,BMP ; GET BIT MAP
27 00177'004417 JSR CODE ; OUTPUT IT
28 ; WRITE ABOUT ALLOCATED ADD.
29 00200'122470 ELDA 0,T3 ; GET POINTER TO MESSAGE
30 000213 ;
31 00202'030727 LDA 2,CHNUM ; GET CHANNEL NUMBER
32 00203'006017 .SYSTEM ; WRITE OUT MESSAGE
33 00204'017077 .WRL 77
34 00205'090700 JMP END ; ERROR RETURN
35 00206'034405 LDA 3,TEMP ; GET STORAGE BASE ADD.
36 00207'021402 LDA 0,2,3 ; GET ALLOCATED ADDRESS
37 ;
38 00210'004406 JSR CODE ; OUTPUT IT
39 00211'002401 JMP @BACK ; RETURN TO MAINLINE
40 ; HANDLER STORAGE
41 00212'000000 BACK: 0
42 00213'000000 TEMP: 0
43 00214'000013 NUM1: 13
44 00215'000040 NUMB1: 40
45 ;
46 ;
47 ;
48 ;
49 ;
50 00216'034447 CODE: STA 3,RETURN ; SAVE RETURN ADDRESS
51 00217'176470 ELEF 3,STORE ; GET ADDRESS OF STORAGE AREA
52 000031 ;
53 00221'175120 MOVZL 3,3 ; MAKE IT A BYTE POINTER
54 00222'024425 LDA 1,.60
55 00223'101102 MOVL 0,0,SZC ; IS IT A 0
56 00224'125400 INC 1,1 ; NO MAKE AC1 ASCII FOR 1
57 00225'167010 STB 3,1 ; STORE THE BYTE
58 00226'030420 LDA 2,P5
59 00227'030421 STA 2,COUNT ; COUNT FOR LOOP COUNT
60 00230'024417 LDA 1,.60 ; GET ASCII BASE

```

```

; END OF GETMAIN HANDLER

```

```

; SUBROUTINE TO OUTPUT AC0

```

```

; SAVE RETURN ADDRESS
; GET ADDRESS OF STORAGE AREA

```

```

; MAKE IT A BYTE POINTER

```

```

; IS IT A 0
; NO MAKE AC1 ASCII FOR 1
; STORE THE BYTE

```

```

; COUNT FOR LOOP COUNT
; GET ASCII BASE

```

```

0005 DNET
01 00231'101100      MOVL 0,0
02 00232'175400 LOOP: INC 3,3      ; INCREMENT BYTE POINTER
03 00233'101100      MOVL 0,0      ; SHIFT AC9
04 00234'101100      MOVL 0,0      ; 3
05 00235'101100      MOVL 0,0      ; LEFT
06 00236'111000      MOV 0,2      ; GET IT INTO AC2
07 00237'153770      ANDI 7,2      ; AND IT TO 3 BITS
08      000007
09 00241'133000      ADD 1,2      ; MAKE IT ASCII
10 00242'173010      STB 3,2      ; STORE BYTE
11 00243'014405      DSZ COUNT    ; FINISHED ?
12 00244'000766      JMP LOOP     ; NO CONTINUE ON
13 00245'000410      JMP OUT      ; LEAVE
14 00246'000005 P5:   -5
15 00247'000060 .60:   60
16 00250'000000 COUNT: 0
17 00251'000000 STORE: 0
18 00252'000000      0
19 00253'000000      0
20 00254'006400      .TXT *(15)*
21 00255'162470 OUT:  ELEF 0,STORE ; GET ADDRESS OF MESSAGE
22      077773
23 00257'030652      LDA 2,CHNUM
24 00260'101120      MOVZL 0,0    ; GET BYTE POINTER
25 00261'006017      .SYSTEM
26 00262'017077      .WRL 77     ; WRITE OUT MESSAGE
27 00263'000622      JMP END
28 00264'002401      JMP @RETURN  ; RETURN TO CALL
29 00265'000000 RETURN: 0
30
31      ; FREEMAIN HANDLER
32 00266'034447 PT2:  STA 3,DONE  ; SAVE RETURN ADDRESS
33 00267'040447      STA 0,TEMP1  ; FOR ARGUMENT PASSING
34 00270'034446      LDA 3,TEMP1  ; GET ARGUMENT BASE ADD.
35 00271'021402      LDA 0,2,3    ; GET START ADD. OF REL.
36 00272'025401      LDA 1,1,3    ; # OF BLOCKS TO BE REL.
37 00273'102270      PSHJ FREEMAIN ; CALL FREEMAIN
38      077777
39
40      ; NOW INSERT INTO OUTPUT STRING
41
42
43 00275'030442      LDA 2,NUM2    ; GET BYTE COUNT IN MESSAGE
44 00276'122470      ELDA 0,T5     ; GET MESSAGE POINTER
45      000135
46 00300'113000      ADD 0,2      ; GET ABSOLUTE BYTE COUNT
47 00301'034435      LDA 3,TEMP1  ; GET ARGUMENT BASE ADDRESS
48 00302'025400      LDA 1,0,3    ; GET ASCII FOR LETTER
49 00303'147010      STB 2,1      ; STORE LETTER IN STRING
50      ; NOW OUTPUT STRING
51 00304'024434      LDA 1,NUMB2   ; GET # OF BYTES IN MESSAGE
52 00305'030624      LDA 2,CHNUM   ; GET CHANNEL #
53 00306'006017      .SYSTEM      ; WRITE OUT MESSAGE
54 00307'016477      .WRS 77
55 00310'063077      HALT          ; ERROR RETURN
56 00311'034425      LDA 3,TEMP1   ; GET BASE ADDRESS
57
58 00312'021401      LDA 0,1,3     ; GET # OF BLOCKS
59 00313'004703      JSR CODE      ; OUTPUT IT
60      ; WRITE OUT ABOUT MAP

```

```

0006 DMENT
01 00314'122470      ELDA 0,T2      ; BYTE POINTER TO MESSAGE
02      000064
03 00316'030613      LDA 2,CENUM      ; GET CHANNEL #
04 00317'006017      .SYSTM          ; WRITE MESSAGE
05 00320'017077      .WRL 77
06 00321'000402      JMP .+2
07 00322'000403      JMP .+3
08 00323'102470      EJMP END          ; ERROR RETURN
09      077561
10 00325'020603      LDA 0,BMP        ; GET BIT MAP
11 00326'004670      JSR CODE         ; OUTPUT IT
12 00327'126000      ADC 1,1          ; MAKE AC1 NON-ZERO
13 00330'162470      ELEF 0,MSYNC     ; GET ADDRESS OF MASTER SYNC
14      077576
15 00332'000145'      .XMT            ; ALLOW PT1 TO RUN
16
17
18 00333'000401      JMP .+1          ; USED TO ALLOW WGETMAIN TEST TO RUN
19 00334'002401      JMP @DONE        ; DON'T CARE IF IT GETS HERE
20
21 00335'000000 DONE: 0              ; RETURN TO MAINLINE
22 00336'000000 TEMP1: 0            ; CONSTANT STORAGE FOR FREEMAIN
23 00337'000014 NUM2: 14
24 00340'000044 NUMB2: 44
25
26
27
28
29
30 00341'000101 A:      "A
31 00342'000001      1
32 00343'000000      0
33 00344'000102 B:      "B
34 00345'000005      5
35 00346'000000      0
36 00347'000103 C:      "C
37 00350'000004      4
38 00351'000000      0
39 00352'000104 D:      "D
40 00353'000010      10
41 00354'000000      0
42 00355'000105 E:      "E
43 00356'000003      3
44 00357'000000      0
45
46
47 00360'000742 "T1:      .+1*2      ; TITLES FOR OUTPUT
48 00361'006412      .TXT *(15)<12>GETMAIN: <15><12>BLOCKS NEEDED : *
49      043503
50      052115
51      040511
52      047072
53      020040
54      006412
55      041114
56      047503
57      045523
58      020116
59      042505
60      042105

```

```

0007 DMMEM
01      042040
02      035040
03      020000
04 00401'001004"T2:      .+1*2
05 00402'052520      .TXT *UPDATED BIT MAP:  *
06      042101
07      052105
08      042040
09      041111
10      032040
11      046501
12      050072
13      020040
14      000000
15 00414'001032"T3:      .+1*2
16 00415'040514      .TXT *ALLOCATED STARTING ADDRESS:  *
17      046117
18      041501
19      052105
20      042040
21      051524
22      040522
23      052111
24      047107
25      020101
26      042104
27      051105
28      051523
29      035040
30      020000
31 00434'001072"T5:      .+1*2
32 00435'006412      .TXT *(15)<(12)>FREEMAIN:  <(15)<(12)>BLOCKS RELEASED :  *
33      043122
34      042505
35      046501
36      044516
37      035040
38      020040
39      006412
40      041114
41      047503
42      045523
43      020122
44      042514
45      042501
46      051505
47      042040
48      035040
49      000000
50 00457'001140"T6:      .+1*2
51 00460'042516      .TXT *END OF TEST<(15)<(12)>*
52      042040
53      047506
54      020124
55      042523
56      052015
57      005000
58 00467'001160"M1:      .+1*2
59 00470'042131      .TXT *DYNAMIC MEMORY TEST<(15)<(12)>*
60      047101

```


0003 DNMET

```

01      046511
02      041440
03      046505
04      046517
05      051131
06      020124
07      042523
08      052015
09      005000
10 00503'001210"TTOD:  .+1*2
11 00504'022124      .TXT *STTO*
12      052117
13      000000
14
15
16
17
18
19
20
21
22
23
24
25 00507'106070 TASK1: EJSR GSTAK
26      000033'
27 00511'102400      SUB 0,0
28 00512'142470      ESTA 0,MSYNG
29      077414
30
31 00514'162470      ELEF 0,C
32      077632      EJSR PT2
33 00516'106470
34      077547
35
36
37 00520'162470      ELEF 0,LABEL
38      000002
39 00522'077777      .KILAD
40 00523'077777 LABEL: .KILL
41
42
43
44      .END START

```

**00000 TOTAL ERRORS, 00000 PASS 1 ERRORS

```

; THIS TASK WILL TEST WGETMAIN QUEUING
; IT WILL BE ACTIVATED WHEN THE MAIN TASK
; IS SUSPENDED BY INSUFFICIENT CORE FOR
; A WGETMAIN

; DYNAMIC CORE WILL BE RELEASED BY
; THIS TASK AND WGETMAIN SHOULD
; REGAIN CONTROL

; GET A STACK FROM RTOS/RDOS

; AC0=0
; STOP PRINTOUT OF WGETMAIN UNTIL

; THIS ROUTINE IS FINISHED ITS PRINTOUT
; GET ADDRESS OF "C" REQUEST

; RELEASE THE "C" CORE

; NOW SUFFICIENT CORE IS AVAILABLE
; TO SATISFY WGETMAIN
; GET KILAD ADDRESS

; KILL THIS TASK
; NOW MAINLINE WILL REGAIN CONTROL
; AND WGETMAIN REQUEST WILL BE SATISFIED

```

0009 DMEET

A	000341'		2/19	2/27	6/30				
B	000344'		2/23	2/36	6/33				
BACK	000212'		3/43	4/39	4/41				
BMP	000130'		1/45	1/46	3/37	4/26	6/10		
C	000347'		2/32	6/36	3/31				
CHNUM	000131'		1/52	2/01	3/17	3/38	4/12	4/22	4/31
			5/23	5/52	6/03				
CODE	000216'		4/18	4/27	4/38	4/50	5/59	6/11	
COUNT	000250'		4/59	5/11	5/16				
D	000352'		2/41	3/03	3/11	6/39			
DMEN1	000121'	EN	1/29	1/48	3/30				
DMEN2	000122'	EN	1/29	3/31					
DMEN3	000123'	EN	1/29	3/32					
DONE	000335'		5/32	6/19	6/21				
E	000355'		3/07	6/42					
END	000105'		1/58	3/15	4/15	4/25	4/34	5/27	6/08
FREEM	000274'	XN	1/30	5/37					
GETMA	000137'	XN	1/30	3/47					
GSTAK	000510'	XN	1/31	2/06	8/25				
INSERT	000136'		2/49	3/47					
LABEL	000523'		8/37	8/40					
LOOP	000232'		5/02	5/12					
M1	000467'		1/59	7/58					
MSYNC	000127'		1/42	3/36	3/50	6/13	8/28		
NUM1	000214'		4/04	4/43					
NUM2	000337'		5/43	6/23					
NUMB1	000215'		4/11	4/44					
NUMB2	000340'		5/51	6/24					
OUT	000235'		5/13	5/21					
P5	000246'		4/58	5/14					
PT1	000132'		2/21	2/25	2/34	2/43	3/05	3/09	3/43
PT2	000266'		2/29	2/38	3/13	5/32	8/33		
RETUR	000265'		4/50	5/28	5/29				
START	000000'	EN	1/28	1/39	1/51	8/44			
STORE	000251'		4/51	5/17	5/21				
SYNO2	000124'	EN	1/29	1/40	3/33				
SYNO3	000125'	EN	1/29	1/41	3/34				
SYNO7	000126'	EN	1/29	1/44	3/35				
T1	000350'		4/05	6/47					
T2	000401'		4/20	6/01	7/04				
T3	000414'		4/29	7/15					
T5	000434'		5/44	7/31					
T6	000457'		3/15	7/50					
TASK	001402	NC	1/27						
TASK1	000507'		2/57	8/25					
TEMP	000213'		3/44	3/45	3/60	4/16	4/35	4/42	
TEMP1	000336'		5/33	5/34	5/47	5/56	6/22		
TTOD	000503'		1/53	8/10					
WCETM	000063'	XN	1/30	2/47					
.60	000247'		4/54	4/60	5/15				
.KILA	000522'	XN	1/31	8/39					
.KILL	000523'	XN	1/31	8/40					
.PRI	000035'	XN	1/31	2/09					
.REC	000143'	XN	1/31	3/52					
.TASK	000072'	XN	1/31	2/59					
.XMT	000332'	XN	1/31	3/54	6/15				

DEV:DMEMT.SV LOADED BY RLDR REV 05.00 AT 15:13:38 07/22/76

DMEMT
DMEM
RTOS
TXMT
BTCBM
BSYST
BINTD
BRTIN
TTYDR
RTCDR
CENIO
BIOSE
STACK

XN QTCK 006564
XN .CKUS 006140

NMAX 013051
ZMAX 000063
CSZE 000000
EST 000000
SST 000000

.RTCF 000001
.FRTC 000001
.NCHL 000012
.RTCI 000012
.SYSE 000060
.SER2 000061
.SER1 000061
.SER3 000062
.NTSK 000144
USTAD 000400
START 000440
DMEN1 000561
DMEN2 000562
DMEN3 000563
SYN02 000564
SYN03 000565
SYN07 000566
GETMA 001164
FREEM 001303
WCETM 001347
.TCBP 001376
.UFPT 004642
.HINT 004666
.ITBL 004676
.ETBL 004775
.CHTB 004776
SYST1 005210
.TMAX 005347
..YST 005517
.XMT 006050
.XMTW 006051
.REC 006052
.IXMT 006053
.PRI 006054
.KILL 006055
.KILA 006056
.TASK 006057
.FOPN 006141
NMCHK 006151
.IOST 006377

.PTSK	006421
.TOCK	006514
.INTP	006604
.INTD	006606
.BDCT	006665
.RTCS	006667
.SACS	006673
.RQUE	006773
.RTOS	007034
.TIEX	007256
.TIIN	007261
.TOIN	007273
.TISV	007301
.WCHR	007311
.TIIS	007346
.TIDT	007477
.TOSV	007540
.TOEX	007547
.TODT	007574
RTCDR	007605
TODMS	007642
TODH	007643
RTGIS	007655
TSECI	010030
.CLK	010031
.GTIM	010073
.STIM	010120
.GTDY	010156
.STDY	010200
.UCLI	010222
.UCLR	010233
..RDL	010273
..WRS	010277
..RDS	010277
..WRL	010341
.GCHR	010353
.PCHR	010355
.OPNO	010372
.OPNI	010407
.CLSO	010416
.CLSI	010420
.RSET	010445
.XIBU	010524
.PENQ	010571
.ENQB	010572
.FINP	010636
.CRIT	011023
.STOU	011044
.COSE	011046
.DIAS	011323
.NIOC	011325
.DOAS	011327
..IAC	011331
.NIOS	011333
.SKPB	011335
CSTAK	011352
STACK	011411
.KLPC	012616
.CORE	077400
TTIDC	107230
TTODC	107510
RTCDC	107605
.INTR	177777
.CKMT	177777

.RLES	177777
.TP10	177777
.CKOT	177777
.CKUS	177777
.CKMC	177777
.GTCK	177777
.GTMC	177777
.CKPK	177777
.CKDK	177777

APPENDIX B

LITTON SYSTEMS RELEASE NOTICE



LITTON SYSTEMS (CANADA) LIMITED

TORONTO OFFICE

72


INTER-OFFICE
CORRESPONDENCE

TO: M. Brett
FROM: L.A. Meikle
DATE: 1 April, 1977
SUBJECT: ANPT DOCUMENTATION

In reference to your memo of 30 March, 1977 to D. Russenberger you may accept this memo as authority to use the requested LSL ANPT documentation in the preparation of the project report you are preparing in partial fulfilment of the requirements for the Master of Engineering Degree.

LAM/ac

c.c. D. Russenberger
A.M. Philip



L.A. Meikle, Director
Engineering Administration

BIBLIOGRAPHY

- Stevens, B.R. Proposal for the Air Navigation Procedures Trainer (ANPT)
Volume 1 ANNEX "A" Detail Specification.
Rexdale: Litton Systems (Canada) Limited 1975.
- Stevens, B.R. Proposal for the Air Navigation Procedures Trainer (ANPT)
Volume 1 Technical Proposal. Rexdale: Litton Systems
(Canada) Limited 1975.
- Programmer's Reference Manual, ECLIPSE Line Computers
Rev. 04. Southboro Mass: Data General Corporation, 1975
- User's Manual, Microprogramming ECLIPSE Computer with the
WCS Features Rev. 00. Southboro Mass.:
Data General Corporation, 1975.
- Introduction to ECLIPSE - Line Real Time Operating
System Rev 00. Southboro Mass.: Data General Corporation
1975.
- ECLIPSE Real Time Operating System User's Manual Rev 00.
Southboro Mass.: Data General Corporation, 1975.
- Introduction to ECLIPSE - Line Real Time Disc Operating
System Rev 00. Southboro Mass.: Data General Corporation
1975.
- ECLIPSE - Line Real Time Disc Operating System Rev 01.
Southboro Mass.: Data General Corporation, 1975.
- User's Manual Symbolic Debugger Rev. 04.
Southboro Mass.: Data General Corporation 1975.
- Standard 5215 Display Generator Communication Programming
Manual. Washington Pa.: AYDIN Corporation, 1975.

FOOTNOTES

- ¹**B.R. Stevens, Proposal for the Air Navigation Procedure Trainer (ANPT)
Volume 1 Technical Proposal. (Rexdale, 1975), p. 2-1**