

Darrell Hal Smith  
1641 7th Ave.  
Longview, WA 98632  
(206) 423-9957

December 12, 1991

Mr. Richard Kratch  
1201 3rd Ave.  
Rm. 3100  
Seattle, WA. 98101-3079

cc: Dennis Lodge (415) 543-2300 (Attny)  
Bill Richards (206) 448-8070 (Seattle P-I)  
Byron Akihido (206) 464-2352 (Seattle Times)

Dear Mr. Kratch:

Enclosed with this cover letter is a report, written per your request, summarizing a major software design flaw in an electronic subsystem common to the Boeing 747-400 and 767 series aircraft. This report is recollected from my work as a member of the Boeing software verification team that analyzed the subsystem's software. Specifically, it is a summary of Boeing coordination sheet B-E32T-C90-012, which was used to notify the vender (Eldec Corp. of Bothell, WA) of design flaws and subsequent data contamination problems (documented in PSEU problem reports 8-569-258 through 8-569-261) on operational aircraft that needed to be corrected.

This subsystem is the proximity switch electronics unit (PSEU), which transmits the status of various components -- landing gear, flaps, etc., (and, to my recollection, auto-restow) to other subsystems. These other subsystems include those controlling aircraft braking, thrust reversers, etc.

Note that the specific problems were corrected for internal data contamination problems, but possible global data contamination problems were not corrected (other systems could get erroneous information) and the major architectural problems that created these specific instances were not corrected. Other such problems could exist undetected and could be introduced any time the software is revised due to the major architectural problems.

Sincerely,



Darrell H. Smith

Encl: PSEU Software Structural Faults Report  
Certificate showing team members and subsystems worked on  
B-E32T-C90-012 Coordination sheet

## PSEU SOFTWARE STRUCTURAL FAULTS REPORT

## Introduction

This report is a recollected summary of Boeing coordination sheet B-E32T-C90-012, which details major software design inadequacies in the proximity switch electronics unit (PSEU) subsystem.

In August, 1989, I joined the software support team of Boeing Commercial Airplanes, in Everett, WA., assigned to verify the software design of the PSEU subsystem for the 747-400 and 767 series aircraft. The PSEU software was written by the Eldec Corp. in Bothell, WA. for the Boeing Company. This software was already installed in the aircraft and had been under review for a considerable length of time before I joined the team.

After a few weeks of inspection, I discovered that the software design was grossly inadequate and had a basic architectural flaw that could cause false information to be reported to other subsystems that relied on the information for correct operation. It was also conceivable that the subsystem could "crash" - software slang for a catastrophic event characterized by processor lockup (non-operation), crazed operation, or false operation.

## Problem Statement

The reason for the problem was that the microprocessor being used in the subsystem utilized sophisticated and complicated mechanisms for sharing its memory amongst the various parts of the software. It was very apparent that the vendor's design engineers, as well as the previous Boeing engineers verifying the design, lacked sufficient experience and knowledge to understand and utilize this particular microprocessor. As a result, the processor's built-in random access memory (register memory) could be contaminated. This could cause a subsystem "crash" or false information about the status of equipment, such as landing gear, flap placement, or auto-restow could be sent to other systems on the aircraft.

To me, it was particularly conceivable that other aircraft systems could be led to believe that the landing gear were down in flight when they actually were not. If such false information was supplied at a high airspeed, it was conceivable that other systems would take action to slow airspeed, since landing gear deployment at high speeds could cause flight problems. At this time, I thought perhaps the auto-throttle would kick in. But, it is also conceivable that the reverse thrusters could kick in - especially if the design of these subsystems was as poorly implemented as the PSEU.

Another possibility is that software controlling the auto-thrusters, relying on false information indicating that the landing gear were down, might switch to a ground speed control mode check, see that the aircraft was going too fast, and kick in the reverse thrusters - while the aircraft was really in flight. Also, if indeed this system reports information about auto-restow or controls auto-restow, thrust reverser operation could be critically impacted.

### Boeing and FAA Reluctance to Investigate the Problem

I was going to include these concerns in a report, but the project engineer, William Whitton, who is also the FAA's designated engineering representative (DER), said he would not allow it. He said the PSEU is a non-critical system (thrust reversers were thought to have little impact on flight stability). The software support manager, Tim Hendrickson, supported my efforts and called for further investigation with other staff members. As a result of these efforts, four instances of design flaws in operating aircraft attributable to my findings substantiated my concerns. A diluted report focusing on technical problems within the PSEU only was allowed to be reported - WITHOUT ALLOWING US TO REPORT ABOUT POSSIBLE CONSEQUENCES TO OTHER SUBSYSTEM BEHAVIOR.

The report that was submitted was B-E32T-C90-012. Also, after examining a few other software subsystems, I noticed a common theme underlying the 747-400 software: the designs looked hastily contrived by inexperienced software engineers. My professional opinion is that time and money factors have overridden safety concerns in the software of 747-400 subsystems.

### Problem Characteristics and Correct Design Architecture

Operational errors in flight, due to the nature of the problems I detected, would be random and often non-repeatable problems that are particularly hard to isolate in test laboratory environments. And, in fact, error reports from aircraft did show a history of such seemingly random errors, many of which were eventually classified as single event anomalies and never corrected.

Such problems can usually be eliminated only by careful design procedures, which seem to be lacking in the application of software design for the 747-400. Design applications that grossly disregarded Boeing software design specifications calling for careful design procedures were placed in operational aircraft. Seven such violations were reported in B-E32T-C90-12.

This disregard to public safety and pressure from project engineers and managers to overlook problems was part of the reason I chose to terminate my relationship with the Boeing Company, in June 1990, after I had fulfilled a one-year employment obligation.

## Summary of B-E32T-C90-012

This coordination sheet describes three subtasks: investigate how the microprocessor manufacturer intended the device's memory be utilized, how the software vendor actually utilized the memory, and identify problems with the vendor's utilization.

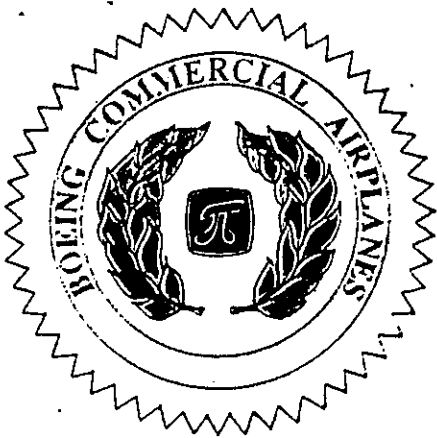
The findings of these tasks indicated that the vendor's design is grossly inadequate and is in violation of several Boeing software standards. Several possible mechanisms that can cause errors were identified, and four specific occurrences of these mechanisms were causing errors in aircraft operation, which substantiated the concerns.

The recommendations to correct the problems were that the software design should be rewritten to comply with Boeing standards and that the management of the processor's memory be included as a separate design item with formal design, code, and verification processes. It had been totally ignored and was not under any design control.

The four specific error problems were corrected for problems within the PSEU - but ignored for problems the PSEU could cause with other subsystems (such as thrust reverser activation), even though B-E32T-C90-012 advised Boeing and Eldec about the concern. The report identifies these problems, lists 7 violations of Boeing software standards that are universal throughout the design, describes how the processors memory was intended to be used by the manufacturer (INTEL), describes the type of things that would cause problems, and lists the design characteristics in the PSEU that match these descriptions. The main problem being that memory could be corrupted as various activities are interrupted during real-time (in flight) activity. The more things going on, the greater chance for false data at random.

In B-E32T-C90-012, it was noted in bold and underlined:

**"NOTE: THIS PROBLEM WITH REGISTER MEMORY UTILIZATION IS A VERY REAL AND SERIOUS IMPEDIMENT TO THE CORRECT OPERATION OF THE PSEU FOR BOTH CURRENT AND FUTURE RELEASES."** The only really adequate solution was a redesign with competent engineers.



June 1990  
CERTIFICATE OF OUTSTANDING PERFORMANCE  
PRESENTED TO

EDWARD CUMMINGS  
LYNETTE CURRIER  
MICHAEL DELK  
FRANCOIS DESPLANCHES  
DENIS GUNDERSON  
DONALD HAM  
GEORGE HAMASAKI

VALARIE JENSEN  
JAMES KENNY  
LYNN LABONTE  
ED MARVIN  
REZA MINAEI  
KENNETH MORGAN  
TERENCE OLSON  
PHILLIP PHUNG

LAWRENCE POIRIER  
MANUEL SANTOS  
JULE SCOTT  
DERRELL SMITH  
KIM STINDT  
ROYLE WELLS  
WILLIAM WHITTON

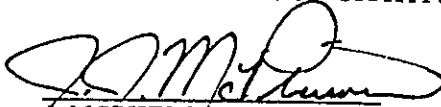
THIS CERTIFICATE IS PRESENTED TO YOU IN RECOGNITION OF YOUR OUTSTANDING WORK PERFORMANCE, PROFESSIONALISM AND DEDICATION ON THE 747-400 PROGRAM. YOU WERE ASKED TO GO ABOVE AND BEYOND THE CALL OF DUTY, WHICH YOU DID WITHOUT HESITATION. THIS WAS ACCOMPLISHED WHILE ALSO SUPPORTING 747 AND 767 SUSTAINING TASKS.

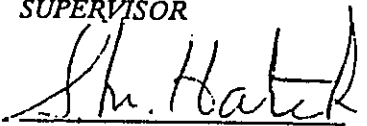
YOUR ROLE IN DEVELOPMENT OF THE 747-400 INCLUDED THE FOLLOWING SYSTEMS:


- TIRE PRESSURE INDICATION SYSTEM
- PROXIMITY SWITCH ELECTRONICS
- BRAKE SYSTEM CONTROL
- RUDDER TRIM CONTROL
- BRAKE TEMPERATURE INDICATION
- HYDRAULIC CONTROL/INDICATION
- PRIMARY FLIGHT CONTROL SURFACE INDICATION


OTHER ACTIVITIES INCLUDED DEVELOPMENT OF SCHEMATICS, DIAGRAMS AND FUNCTIONAL TESTS; VENDOR COORDINATION; VERIFICATION/VALIDATION TESTING, ELECTRICAL ANALYSIS AND CERTIFICATION; SOFTWARE ENGINEERING SUPPORT; AND AIRPLANE TROUBLE-SHOOTING FOR THE FACTORY AND FLIGHT LINE.

PLEASE ACCEPT OUR GRATITUDE AND APPRECIATION FOR A JOB WELL DONE.

  
J. MGPHERSON  
ELECTRICAL POWER  
SUPERVISOR

  
S. M. HATCH  
CHIEF DESIGN ENGINEER  
ELECTRICAL SYSTEMS

  
R. W. SUTTON  
ELECTRICAL SYSTEMS  
MANAGER

  
R. A. DAVIS  
DIRECTOR OF ENGRG.  
EVERETT DIVISION

PRIDE IN EXCELLENCE

## COORDINATION SHEET

B-E32T-C90-012  
March 5, 1990

TO:	J.J. McPherson	B-G16T	OT-07
cc:	D.P. Gunderson	B-G16T	OT-07
	W.S. Whitton	B-G16T	OT-07

3 Enclosures.

SUBJECT: PSEU 8096 Register Memory Use

### Purpose:

The purpose of this portion of Task 24, 8097 Register Overlay Analysis, was to see how register memory usage is controlled by Intel software utilities, determine how the PSEU software utilizes these controls, and identify any problems with this utilization.

### Summary of Findings:

The PSEU software design is grossly inadequate in addressing the utilization and management of on-chip register memory. The design is in violation of several Boeing software standards. Several possible mechanisms that can cause errors to be introduced in the software implementation were identified. Four specific instances of software errors causing data contamination problems substantiating these concerns were discovered by Lynette Carrier during independent analysis.

### Recommendations:

Register memory use, especially the overlayable segment (OSEG), is a separate architecture in itself. It should be a separate design item with formal design, code, and verification processes. Consequently, the software architecture must be enhanced to correct the deficiency. This can be accomplished in two phases.

During the first phase, for the clean up of the Version 5 documentation, a description of the architecture and utilization of the register memory should be added to the SWDD. The rationale and techniques for assembling, compiling, and linking with register memory should also be added to the SWID (see the enclosed "PSEU Violation of Boeing Software Standards" for specific requirements of what the documentation requires).

During the second phase, for the next part number roll, register memory overlay usage should be completely reevaluated. The requirement to use it should be challenged and approved at the CDR. Boeing would provide a standard for designing software with register memory overlay, including architectural requirements, design methodologies, verification, etc. The register memory overlay usage would be audited for conformance to the standard. Note: to properly utilize memory overlay would require code and timing changes in the software. Task 29 has been identified to determine if the advantages gained by using memory overlay are outweighed by the penalties its usage causes.

#### Method:

The first step was to analyze Intel documentation and generate a document providing a comprehensive description of assembler, PLM compiler, and linker controls that effect register memory (see the enclosed "MCS96 Register Memory Controls Summary").

The second step was to identify any possible problems associated with utilizing and managing register memory, including special design constraints and possible error mechanisms (see the enclosed document, "PSEU Register Memory Problem Description"). This document also recommends three alternative approaches to develop a structured discipline for PSEU software register memory use.

The third step was to analyze the software, looking for specific error instances. During this phase it was determined that the software design of the PSEU inadequately addresses the utilization and management of on-chip register memory and is in violation of several Boeing software standards (see the enclosed "PSEU Violations of Boeing Software Standards").

As part of the third step, independent analysis performed by Lynnette Courier, resulting in the discovery of four specific errors in the software causing data contamination problems, substantiated these concerns (see PSEU problem reports 8-569-258 through 8-569-261). A concern for possible data contamination has been identified. Five modules, spn, spn\_direct, sgn, sgn\_direct, and srs, define local absolute variables that encompass the same address space as srf\_buffer, a 4 byte global array with a starting address of FCH. As the software now stands, the local variables cannot be contaminated. It might be possible, though, for the local modules to contaminate the global variable. This concern should be transmitted to Eldec.

Prepared by \_\_\_\_\_  
D. H. Smith  
B-E32T OT-23  
Ph 266-6520

Concurred by \_\_\_\_\_  
R. E. Wells  
B-E32T OT-23  
Ph 342-6606

Concurred by \_\_\_\_\_  
T. W. Hendrickson  
B-E32T OT-23  
Ph 342-9697



## PSEU Violations of Boeing Software Standard

D6-35071-1

p 17, section 1.1 : "Goals of the standard are to ensure software with good design, uniformity in development and production, and documentation visibility".

Register memory architecture and utilization is not visible in the documentation. It needs to be a separate design item to provide visibility.

p 29, section 2.3.6 : "A memory map of the software shall be developed to provide memory organization visibility and to assist in debug and test operations".

A memory map of register memory, particularly overlay segments and variables, is not provided.

p 33, section 2.5.1.f : "Special design constraints. All constraints imposed by unique demands of the function shall be identified".

Functional descriptions do not include details of register memory use by the function.

p 39, section 2.7 : "The SWID shall include the following elements, as a minimum: ...

- c. Assembly/compilation procedures.
- d. Link/edit procedures.
- e. Load map."

These items do not address the aspects of register memory utilization.

D6-35071-2

p 20, section 5.1.b : "Describe any conventions established to avoid problems associated with the following, if applicable, to the component being described: ... (5) shared data..."

Overlayable memory is by its vary nature, shared data, and since it is, conventions to avoid problems associated with its utilization shall be described in the SWDD.

p 21, section 6.1 : Module descriptions: " Specify implementation constraints (e.g., memory usage, time requirements, etc.).

Details of fast RAM utilization including implementation constraints (i.e., fast RAM controls and reasons for use, data mapping, etc) shall be included in module descriptions.

p 27, section 4.2 : SWID Implementation Procedures: "Link Edit Procedure. This section provides detailed instructions for linking modules comprising the program into an executable program. Required command files and command streams shall be identified."

Details of fast RAM utilization including linkage controls and call hierarchies shall be included in the SWID.

## PSEU Register Memory Problem Description

### 1.0 Overview.

The 8096 CPU contains high speed on-chip RAM called Register Memory. Because this memory is of limited size, the Intel software development environment allows variables from different procedures and even variables from different blocks within a procedure to be assigned to the same memory location within this RAM (Register Overlay).

The use of Register Memory requires highly disciplined design, coding, and verification processes to prevent data corruption because the linker, assembler, and PLM compiler controls that effect register memory provide a high degree of flexibility. A disciplined structure must be provided for designing with register memory and to provide assistance for the analysis of the PSEU software.

### 1.1 Applicable Documents.

"MCS96 Register Memory Controls Summary" (attached).

### 1.2 Register Memory Utilization as an Architecture.

Register Memory use, especially the Overlayable Segment (OSEG), is a separate architecture in itself. It should be a separate design item with formal design, code, and verification processes - under control. Command directive files for assemblers, compilers, and linkers should also be in this formal process to guarantee compliance and reliability. This includes memory management directives such as that for overlayable registers (e.g., call hierarchies). Each particular logical substructure should be a separate entity in the process (e.g., program memory architecture, RSEG architecture, OSEG architecture, stack architecture, etc.)

### 1.3 Development Environment Problems.

Assembler controls are simple, providing a solid, maintainable platform. The PLM Compiler, however, because it tries to overlay as much as possible without having much knowledge about modules outside of the one being compiled, has complex controls, making the situation complicated and more susceptible to errors. Complicating this further, the linker will not allow overlaying unless the REGOVERLAY control is specified, which causes the compiler to overlay all such modules unless a call relationship can show that particular modules should not be overlaid. The call relationship is in itself complicated, showing relationships explicitly and by deduction. Call relationships parallel the actual code design and are supposed to be specified during design and never changed there after.

## **2.0 Perceived Problems.**

### **2.1 Interrupts.**

The most salient error mechanism is that an interrupt routine may call a procedure (directly or indirectly) that accesses a variable overlaid with another variable from a routine suspended by the interrupt request. Software operation, by analogy, becoming a game of Russian roulette with the possibility of critical data being corrupted in a non-repeatable, non-traceable fashion (i.e., at random).

### **2.3 Sneaky Calls.**

Sneaky calls (variable calls, lookup table calls, pseudo multi-tasking, etc) are undocumented as far as the compiler and linker are concerned and cannot be used by either to establish a call relationship unless explicitly defined. These are hard to document and control, even when an attempt is made to do so. Sneaky calls abound in the software as it exists.

### **2.4 Mixing Assembler and PLM.**

Mixing these two languages within one software program increases the potential tremendously for one of the above mechanisms to cause a problem. This is especially so if the use of the two languages at a procedure and module basis is not structured according to sound software discipline with appropriate documentation.

### **2.5 Libraries.**

Procedures in library modules, including intrinsics, must have been compiled as REENTRANT in order to prevent overlay.

### 3.0 Recommendations.

Three alternative approaches to develop a structured discipline for PSEU software Register Memory usage can be perceived. These alternatives are as follows:

#### 3.1 Pure Standard Approach.

The Pure Standard Approach would entail Boeing providing a standard for designing software with Register Memory, including architectural requirements, design methodologies, verification, etc. The burden would then be on the vendor to insure that the product conforms to this standard. This is the most comprehensive and desirable approach. It would have the most up front cost impact on the PSEU software but would be the most effectual for long term development and verification processes.

#### 3.2 Conformed Standard Approach.

The Conformed Standard Approach is identical to the Pure Standard Approach except that the standard would be written to conform to discerned PSEU software methodologies in place. This would have the positive impact of reducing up front costs but could adversely effect the utility of the standard in regards to achieving the goal of having resultant software operate correctly.

#### 3.3 Apologetic Approach.

The Apologetic Approach would require the vendor to provide after-the-fact documentation detailing the requirements, design, and implementation of all aspects of the software which involve Register Memory. This would include philosophies (for lack of a better term) of use, guidelines, architecture of Register Memory variables, memory maps, utility designs and operation (linker, loader). Boeing would then review and critique this and the vendor would respond to the critique.

This approach has many critical drawbacks, the major being that without a standard to work with, the documentation cannot be effectively critiqued. It would also give too much discretion to the vendor and make it very hard to get required changes to obtain a software product that works correctly. NOTE: THIS PROBLEM WITH REGISTER MEMORY UTILIZATION IS A VERY REAL AND SERIOUS IMPEDIMENT TO THE CORRECT OPERATION OF THE PSEU FOR BOTH CURRENT AND FUTURE RELEASES.

## MCS96 Register Memory Controls Summary

### 1.0 Overview.

Assembler, PLM compiler, and linker controls that effect register memory provide a high degree of flexibility. This document is an attempt to provide a comprehensive description of these controls and their effects to assist in the analysis of the PSEU software.

### 1.1 Applicable Manuals.

Intel "ASM96 / R & L" manual which includes "MCS-96 Macro Assembler Users Guide for DOS Systems" and "MCS-96 Utilities Users Guide for DOS Systems".

Intel "PL/M-96 Users Guide".

### 1.2 Register Memory.

Register Memory is RAM memory located within the 8097, hence it is faster to access than external memory, yet is of limited size. Register segments are portions of this memory, of which there are two types, the non-overlayable Register Segment (RSEG), and the Overlayable Register Segment (OSEG).

### 1.3 Register and Segment Overlay.

The OSEG is provided to overcome size limitations by allowing variables from procedures that are not simultaneously active to be assigned to the same memory location (Register Overlay). At the linker level, this is implemented by overlaying the OSEG of individual modules (Segment Overlay).

### 1.4 Overlay Criteria.

Two modules can be overlaid only if during program execution no chance exists that a procedure in one may be active simultaneously with a procedure in the other (e.g., direct or indirect calls, interrupts).

After a source module is assembled or compiled, the resulting object module may or may not contain an OSEG, as determined by respective assembler and compiler controls. Assembler controls are simple - variables are explicitly declared in either the OSEG or RSEG. The compiler, however, because it tries to overlay as much as possible without having much knowledge about modules outside of the one being compiled, has a much more complex set of controls.

The linker, being the last processing stage, has final overlay authority. It will not overlay a module unless the REGOVERLAY control is specified. Then it will overlay all such modules unless a calls relationship shows that particular modules should not be overlaid. The linker produces an executable program containing a single absolute module merging all input modules (including libraries).

## 2.0 ASM96 Assembler.

The RSEG and OSEG assembler directives are used to assign variables to the non-overlayable and overlayable register segments, respectively, by placing variable declarations after the appropriate directive.

## 3.0 PLM96 Compiler.

The compiler looks only at the module being compiled. If an item is not declared PUBLIC or EXTERNAL, its scope is limited to the module it's declared in.

### 3.1 Register Memory Variable Declarations.

Variables may be explicitly declared either FAST or SLOW to be permanently placed in or outside Register Memory, respectively. Global variables and fast variables are statically allocated, non-fast procedure variables are dynamically allocated on the stack, which achieves greatest possible compression of non-fast memory requirements.

If space remains after allocating all FAST variables, the compiler chooses more-often repeated variables declared without either suffix and places them in Register Memory. To insure that a variable is not placed in Register Memory, explicitly declare it SLOW. The FAST/SLOW suffix may not be specified for based variables, nor with the AT or DATA attributes.

### 3.2 Compiler Controls for Register Memory.

#### 3.2.1 Enable Register Overlay (REGOVERLAY).

REGOVERLAY

DEFAULT: NOREGOVERLAY

To enable or inhibit overlaying FAST variables from different procedures, use the REGOVERLAY or NOREGOVERLAY controls, respectively. These form a primary compiler control which can be specified once within a module and cannot be changed.

### 3.2.2 Specify Number of Fast Registers (FAST).

FAST(0 - 220)

DEFAULT: FAST(ALL)

To specify the number of Fast Registers used for an application, use the FAST() primary compiler control.

To specify that the entire application uses Register Memory, use FAST(ALL). The compiler then assumes that all variables, including EXTERNAL variables, are FAST.

If FAST(ALL) is not specified, any EXTERNAL variable not explicitly declared FAST EXTERNAL is assumed not to be FAST. Beware, an EXTERNAL SLOW can match a PUBLIC FAST; however, the reverse causes a link-time error.

### 3.3 Compiler Overlay Criteria.

The compiler tries to overlay as much as possible (e.g., variables of disjoint DO blocks are always overlaid; the compiler tries to overlay any pair of procedures that can never be active simultaneously).

But, the main problem with overlaying is for the compiler to know which procedures might be active simultaneously. Since the compiler looks only at the module being compiled and cannot efficiently determine what other modules are doing, it ignores possible sneaky calls by assuming that they do not exist. If the CALL's do occur, errors may be introduced.

For this reason, if calls possibly occur outside the module, the compiler expects procedures to be declared INDIRECTLY\_CALLABLE or INTERRUPT\_CALLABLE.



### **3.4 Procedure Declarations.**

#### **3.4.1 Without Fast Variables (REENTRANT).**

A procedure that cannot have FAST variables must be declared REENTRANT. All variables will be SLOW (affecting stack size), no variable may be declared FAST. Procedures that must be declared REENTRANT include: any procedure that may be interrupted and that is also activated from an interrupt procedure (including interrupting an interrupt procedure); any procedure that calls itself (direct recursion); any procedure that calls itself through an intermediary call (indirect recursion).

#### **3.4.2 Without Overlaying (INTERRUPT\_CALLABLE).**

A procedure that cannot be overlaid must be declared INTERRUPT\_CALLABLE. Any procedure called (directly or indirectly) from an interrupt procedure in another module should be declared INTERRUPT\_CALLABLE. These procedures can only be PUBLIC or procedures at the module level whose address is taken (any other case causes a compilation warning).

#### **3.4.3 Overlaying Only by Non-calling Procedures (INDIRECTLY\_CALLABLE).**

A procedure that will not be overlaid with procedures that may call it is declared INDIRECTLY\_CALLABLE. No procedures that call external procedures or make indirect calls will be allowed to overlay it. Any procedure that may be called in sneaky ways should be declared INDIRECTLY\_CALLABLE. These procedures can only be PUBLIC or procedures at the module level whose address is taken (any other case causes a compilation warning).

#### **3.4.4 Interrupt Procedures.**

Interrupt service procedures are declared with INTERRUPT N. These procedures must be untyped without arguments. They may be declared PUBLIC, INDIRECTLY\_CALLABLE, INTERRUPT\_CALLABLE, and/or REENTRANT.

#### 4.0 RL96 Linker.

#### 4.1 Linker Overlay Criteria

The linker will not overlay a module unless the REGOVERLAY control is specified with that module. Modules so specified will then be overlaid with all other specified modules unless a "calls" relationship can show that particular modules should not be overlaid with each other. RL96 controls are entered on the RL96 invocation line and may be plural, separated by blanks.

#### 4.2 Order of Allocation.

- 1) All absolute segments.
- 2) Relocatable OSEG of modules specified by REGOVERLAY.
- 3) Relocatable RSEG and OSEG not yet allocated.
- 4) Relocatable CODE, DATA, etc.

#### 4.3 Request Register Overlay (REGOVERLAY).

REGOVERLAY(Overlay\_Term[,...])      DEFAULT: NOREGOVERLAY

Abbreviation: OV/NOOV

The RL96 linker control REGOVERLAY specifies that overlaying is desired, which modules are to be considered for overlaying, and the constraints to apply. The request to overlay any two modules may be fully granted, partially granted, or not granted at all. A request not to overlay two modules or a request for no overlaying (i.e., NOREGOVERLAY) is fully honored.

In the REGOVERLAY control, the "calls" relationship disallows overlaying particular modules. If a module is specified without a calls relationship, the linker assumes that the module can be overlaid with any other specified module. If a relationship between two modules is not distinctly specified, or cannot be deduced (by transitivity), RL96 assumes that those modules can be overlaid. The Linker can be fooled, especially by indirect address calls.

## 5.0 PLM / Assembler Interface.

PLM96 uses the eight byte registers and addresses 1CH-23H for temporary computations. The library PLM96.LIB defines (as needed) the public symbol PLM\$REG which is used to return function results. If a procedure has both SLOW and FAST arguments, place all FAST arguments at the end of the argument list, to avoid stack use.

The easiest way to ensure compatibility between assembler and PLM is simply to write dummy procedures in PLM with the same argument list as the desired assembly subroutine and with the same attributes. Compiling with the CODE control produces a pseudoassembly listing which may be copied as the prologue and epilogue of the assembly subroutine.

### 5.1 Procedure Prologues.

interrupt routine:

```
pushf
push PLM$REG
push PLM$REG+2
push PLM$REG+4
push PLM$REG+6
```

procedure with FAST variables:

```
pop PLM$REG
pop parm n
.
.
pop parm2
pop parm1
push PLM$REG
```

procedure with local SLOW variables continues with:

```
sub sp, #total size of local SLOW variables.
```

### 5.1 Procedure Epilogues.

noninterrupt routine with no SLOW parameters:

```
ret
```

if SLOW parameters or variables are present:

```
ld PLM$REG+6, size of slow variables [SP]
add sp, #size of all parameters, variables,
and return address on the stack
br [PLM$REG+6]
```

PLM DIRECTIVES AFFECTING NON-PROCEDURE VARIABLES

LINKAGE	EXTERNAL			PUBLIC			UNSPECIFIED		
	FAST	SLOW	UNSPECIFIED	FAST	SLOW	UNSPECIFIED	FAST	SLOW	UNSPECIFIED
LOCATION FAST (CALL) (DEFAULT)	RSEG	<u>ERROR</u>	FAST	FAST	<u>ERROR</u>	FAST	FAST	<u>ERROR</u>	FAST
FAST(H)	RSEG	DSEG	SLOW (DEFAULT FOR INTERNAL)	FAST	DSEG	UNSPECIFIED	FAST	DSEG	UNSPECIFIED
NOT ENOUGH FAST SPACE									
ENOUGH FAST SPACE									
NOT ENOUGH FAST SPACE									

PLM DIRECTIVES AND PROCEDURE ATTRIBUTES AFFECTING PROCEDURE VARIABLES

LOCATION	FAST	SLOW	UNSPECIFIED
FAST (CALL) (DEFAULT)	FAST	STACK (DYNAMIC)	FAST
FAST(H)	FAST	STACK (DYNAMIC)	UNSPECIFIED
REENTRANT	WARNING AND UNDER VARIABLE		STACK (DYNAMIC)
NOT ENOUGH FAST SPACE	<u>ERROR</u>		STACK (DYNAMIC)
ENOUGH FAST SPACE	FAST		FAST
NO RETURN (DEFAULT)	RSEG		RSEG
RECURSIVE	FAST		FAST
INTERRUPT CALLABLE	RSEG		RSEG
INDIRECT CALLABLE	<u>ERROR</u>		<u>ERROR</u>
UNSPECIFIED	<u>ERROR</u>		<u>ERROR</u>

BUT MAP  
GENERATES  
WARNING  
GETS  
MESSAGE  
WHEN LIST

NOT  
USE