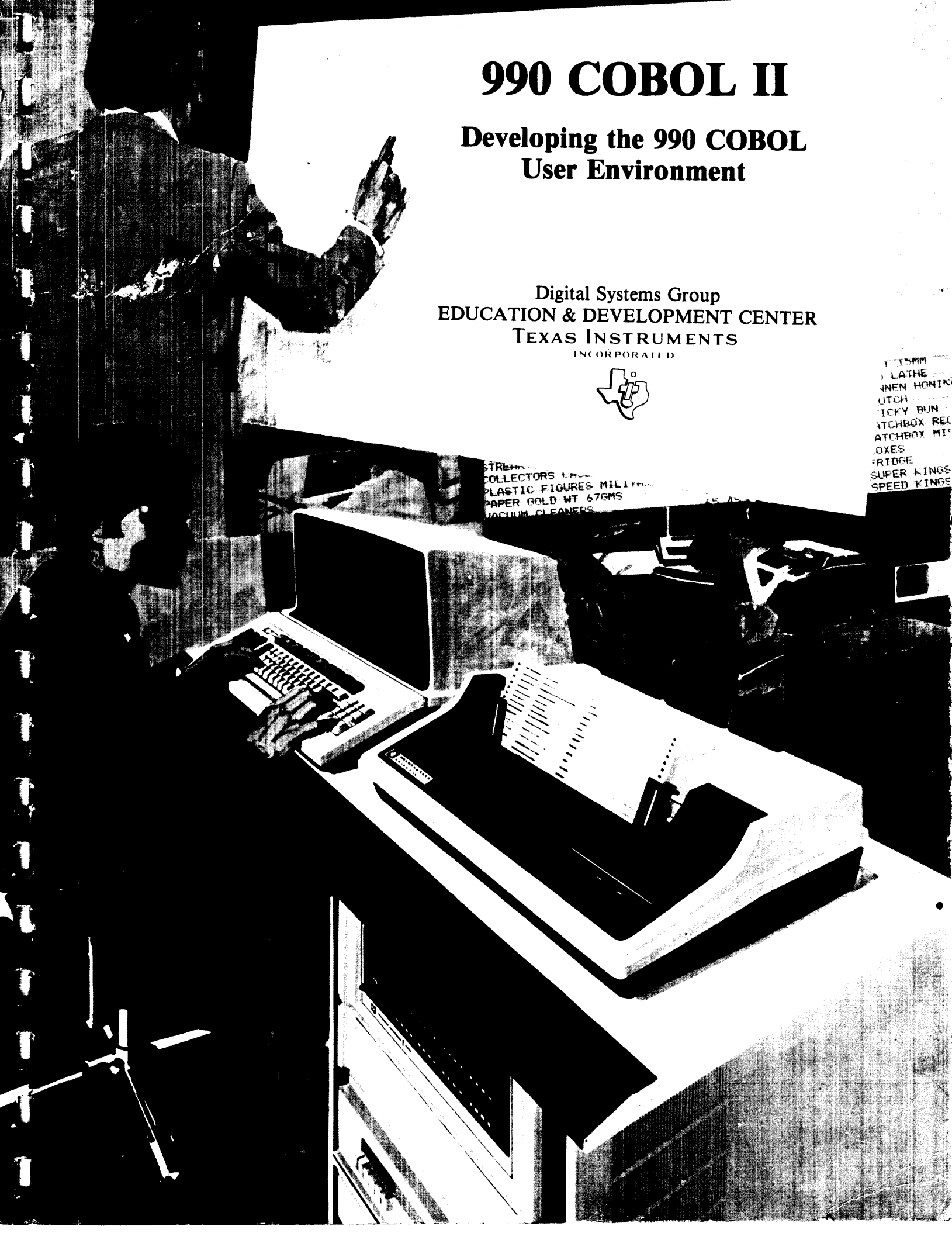
990 COBOL II

Developing the 990 COBOL User Environment

Digital Systems Group
EDUCATION & DEVELOPMENT CENTER
TEXAS INSTRUMENTS
INCORPORATED



C O B O L II

This student guide is not a self-contained document; it is designed to support live instruction.

ORIGINAL ISSUE JULY 11, 1979

REVISED MARCH 24, 1980

THIS STUDENT GUIDE SUPPORTS

990 COBOL RELEASE 3.2

DX10 RELEASE 3.3

Revisions to this student guide are planned so that the most current product releases are addressed. However, TI products may be revised before it is possible for this student guide to be updated. as a result, there might be slight differences between your system and the description of products in this student guide. Therefore, reference should always be made to the most current reference manuals.

990 COBOL II

This course is intended for the system analyst or design programmer who must know the capabilities of the DX10 operating system and COBOL's interrelationships for proper system design.

The student participating in this course must have successfully completed the course entitled 990 COBOL I or be experienced in the use of 990 COBOL and the DX10 operating system.

The format of this courses uses live instruction with worksheets to test the student's knowledge. Lab exercises comprise about 50 percent of the class time. Successful completion of this course implies that the user has mastered the material presented for discussion and has sucessfully completed the worksheets and lab exercises.

The course materials that are provided to the student include:

* COBOL II Student Guide

* Volume V of the DX10 Reference Manuals
  part no. 946250-9705

Optional materials that will be provided as needed include:

* 3780 Communication Emulator Student Guide

* DX10 3780/2780 Emulator User's Guide
  part no. 946289-9701

* 3270 Communication Emulator Student Guide

* DX10 3270 Interactive Communication Software (ICS) User's Guide
  part no. 2250954-9701

Additional reference material includes:

* Volume II of the DX10 Reference Manuals
  part no. 946289-9702

* Volume III of the DX10 Reference Manuals
  part no. 946289-9703

* Link Editor Reference Manual
  part no. 949617-9701

The five day agenda for ths course is:

Monday	A.M.	COBOL with Reentrant Procedures Segmentation and Overlays Key Index Files
	P.M.	Lab Exercise
Tuesday	A.M.	System Command Interpreter
	P.M.	Lab Exercise
Wednesday	A.M.	System Customization Lab Exercise
	P.M.	Lab Exercise Batch Command Streams Lab Exercise
Thursday	A.M.	System Generation System Backup and COBOL Installation DX5 COBOL
	P.M.	Lab Exercise
Friday	A.M.	Communication Emulators (self-paced)

TABLE of CONTENTS

MODULE 5 SYSTEM CUSTOMIZATION

MODULE 6 BATCH COMMAND STREAMS

MODULE 7 SYSTEM GENERATION

MODULE 8 SYSTEM BACKUP AND COBOL INSTALLATION

MODULE 9 DX5 COBOL

APPENDIXES

MODULE 1

COBOL WITH REENTRANT PROCEDURES

OBJECTIVES

* Describe the advantages of using shared procedures.
* Specifying the DX10 Memory Mapping features and structure.
* Write Link Control files to install reentrant procedures.

1.1 A MULTI-PROGRAMMED SEGMENTED ENVIRONMENT

In the multi-programming environment, segmentation is highly desirable to increase the throughput time. Segmentation also offers an additional technique to reduce the memory requirements. Figure 1-1 illustrates a simple multi-programming environment with two tasks. Each task has three segments in its address space, but both require the same payroll routine and system library. In most cases, each user would get a separate copy of the payroll routine and system library bound into his address space. If both payroll and system library routines are shareable, it is not necessary to have two separate copies.

A reentrant routine is one which permits multiple calls and executions before prior executions are complete. In order to accomplish this, parameter addresses should be used by indexing and indirect reference rather than by planting them into instructions within the subroutine. Temporary storage access within the program should be by indexed addresses. The index may be set by the calling program in order to take care of multiple calls. It then serves as a stack pointer for the temporary storage.

MULTI-PROGRAMMED ADDRESS SPACE

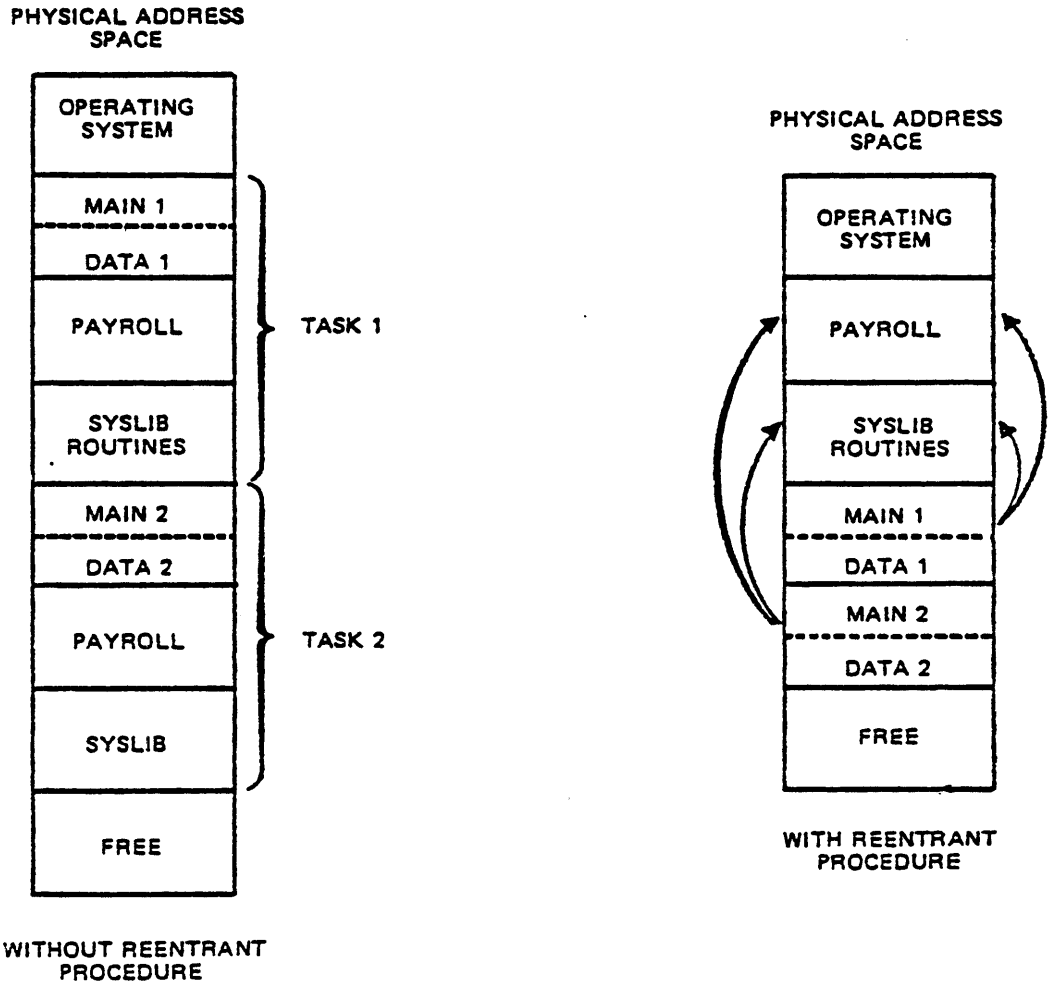


Figure 1-1 Multi-Programmed Address Space

* A reeentrant routine permits multiple calls and executions before prior executions are complete.

* A reentrant routine reduces the throughput time.

* A reentrant routine also offers an additional technique to reduce the memory requirements.

Figure 1-2 illustrates a memory requirement reduction by sharing a BASIC interpretive language processor. Should 10 jobs be running BASIC programs at the same time, the BASIC interpreter could then require 34,000 bytes plus 5,000 bytes data segment for the tables and variables. Total memory required for the 10 tasks would be 10 X (34,000 + 5,000) = 390,000 bytes. By sharing a single copy of the BASIC interpreter segment and using separate copies of the data segment, actual memory required can be reduced to 34,000 + 50,000 = 84,000 bytes (a 78% reduction).

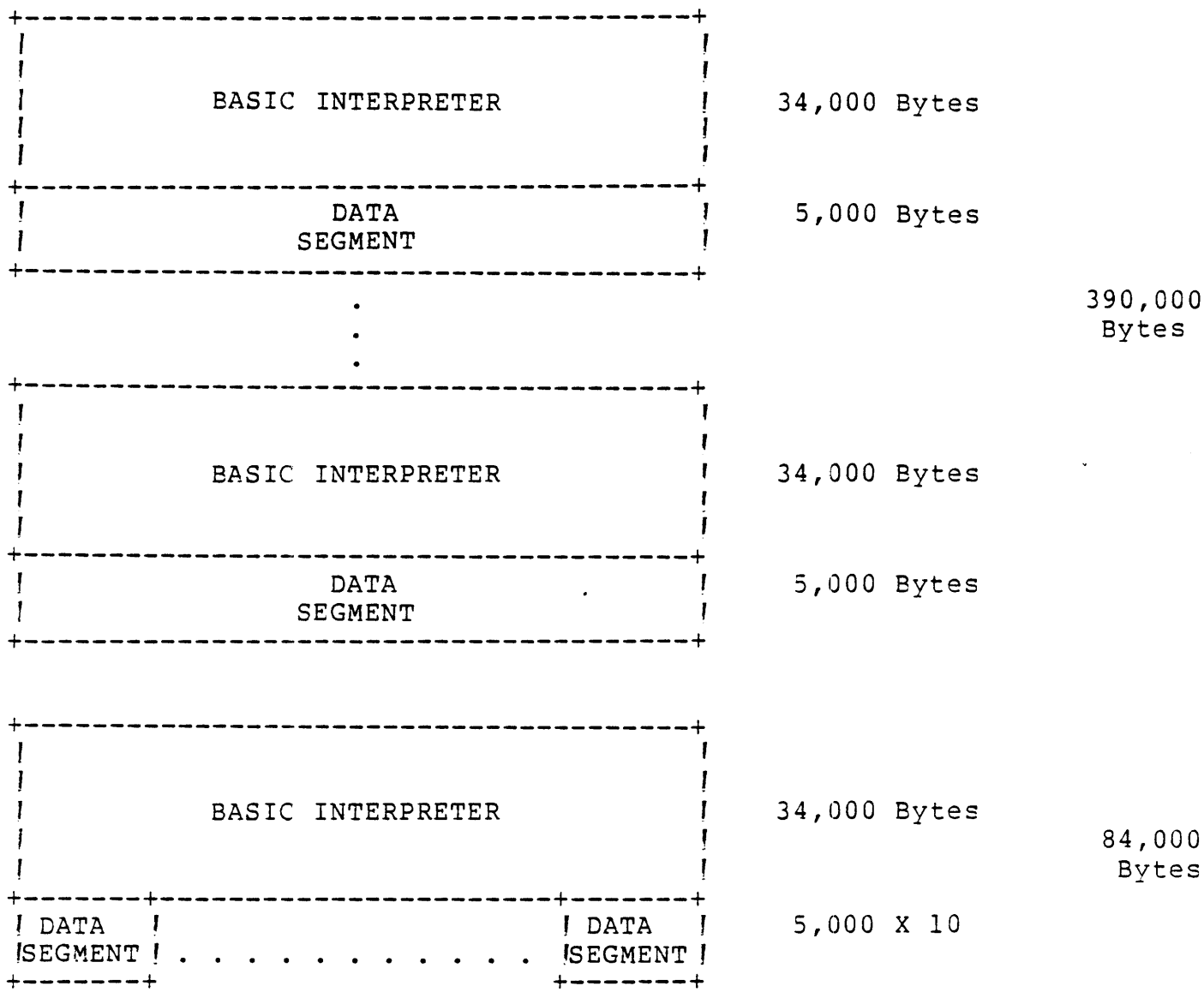


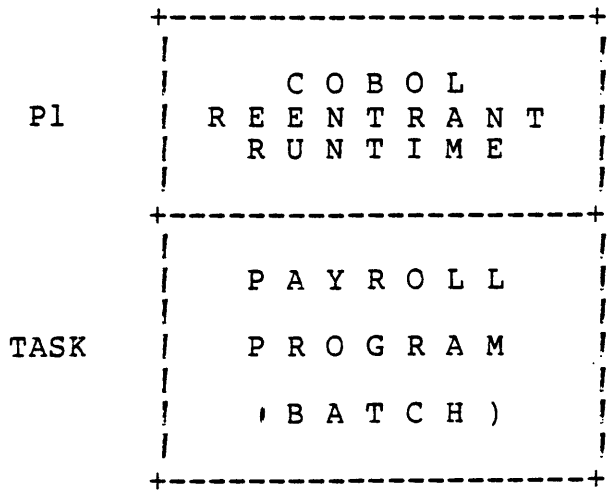
Figure 1-2 Saving Memory Through Segmentation

Up to this point we have assumed that it is possible to produce physically segmented memories. Well, lets look at the DX10 memory mapping features.

1.2 USE OF MAPPING

The TI 990/10 and 990/12 have a mapping scheme for memory which may be used to divide programs into two or three sections. Under DX10 the first and the second sections, which are optional, are called PROCEDURES and they can be shared by more than one run unit. The third section is called a TASK and it is the unique (not shared) portion of a any program. Therefore, a program operating under DX10 may consist of one or two procedures and one task. These sections are referred to by the names: procedure one (P1), procedure two (P2), and task. The following narrative will help exhibit the mapping structure.

A SMALL BUT RAPIDLY GROWING ELECTRONICS FIRM



IMPLEMENTS THEIR PAYROLL "SYSTEM" ON A 990

Figure 1-3

The DX10 operating system contains facilities which allow several tasks to share a procedure.

- LATER -

THE FIRM HAS GROWN SUCH THAT THEIR PAYROLL FILE

UPDATES AND ENQUIRES ARE DONE INTERACTIVELY

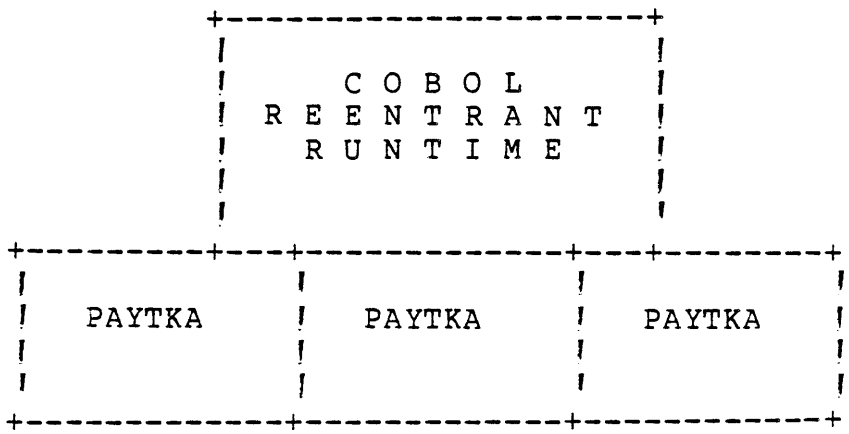


Figure 1-4

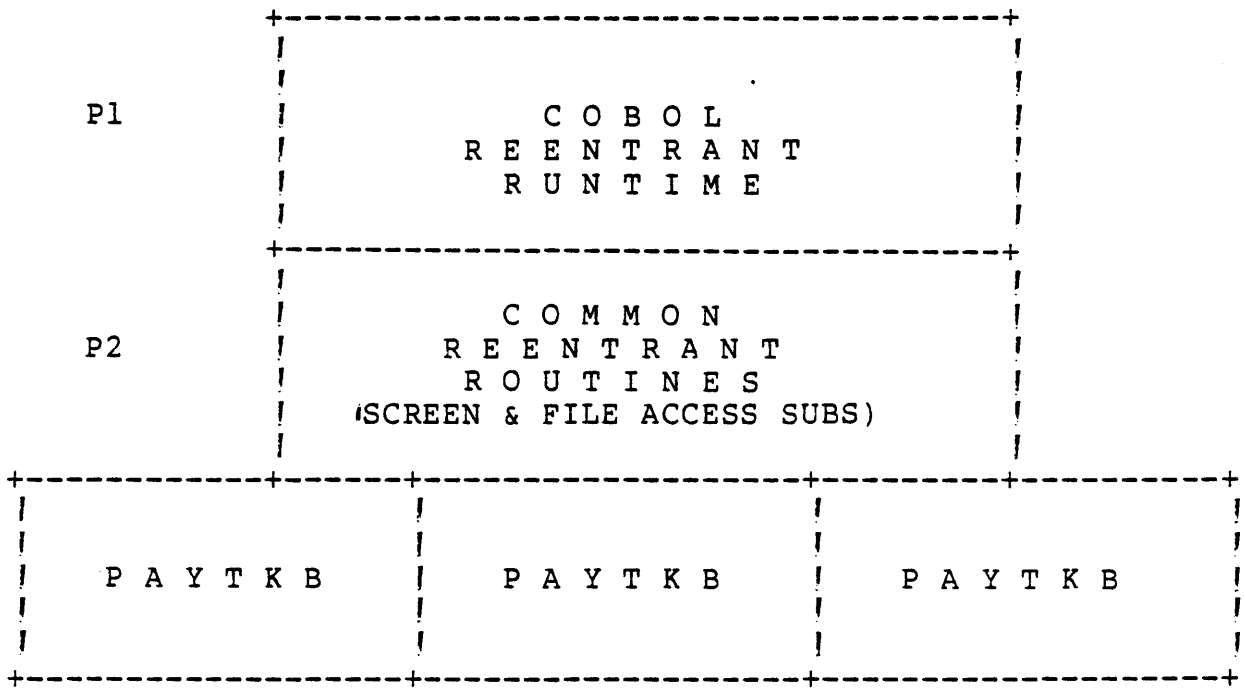
The DX10 operating system also has facilities which allow subroutines mapped into P1 and/or P2 to be shared by multiple tasks so that P1 & P2 are said to be reentrant procedures.

AS THE FIRM GROWS

MORE APPLICATIONS ARE ADDED TO THEIR 990

REQUIRING MORE EFFICIENT UTILIZATION

OF THEIR COMPUTER



THESE TASKS PERFORM THE SAME UPDATES AND ENQUIRIES

AS THE PREVIOUS TASKS

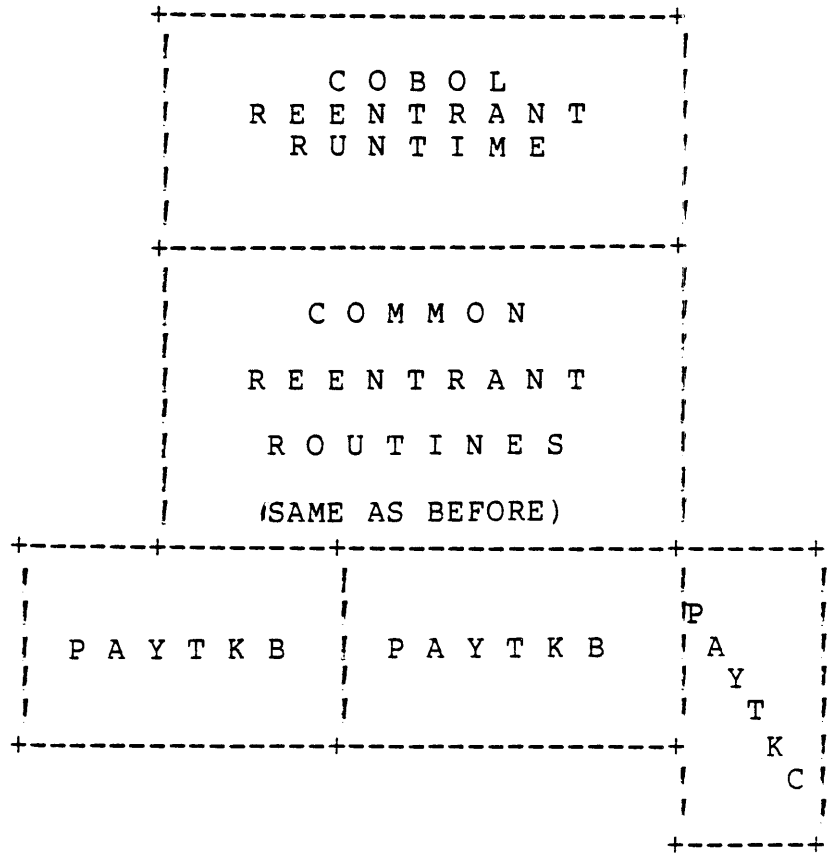
BUT

THIS TASK USES SIGNIFICANTLY LESS MEMORY

Figure 1-5

WHEN THEY ADD NEW INTERACTIVE PAYROLL APPLICATIONS

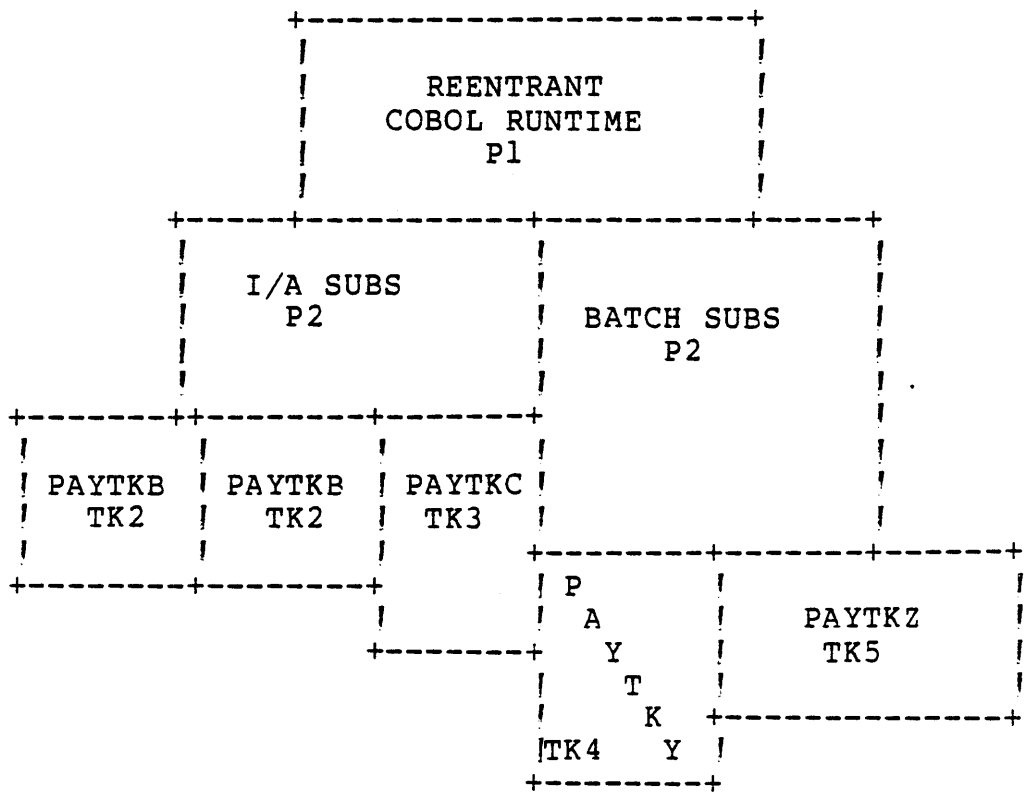
(E.G. SPECIAL MIS-TYPE ENQUIRES NOT PREVIOUSLY SUPPORTED)



THE NEW TASK (PAYTKC) IS A NEW I/A PAYROLL APPLICATION

Figure 1-6

TO FURTHER FILL IN THE PAYROLL PROCESSING PICTURE



OF COURSE, MOST PROCESSING WILL BE BATCH!

Figure 1-7

Figure 1-8 shows the memory requirements for part of the firm's payroll system.

P1: is the COBOL reentrant runtime - 16.4KB

P2a: is the reentrant I/A Screen and File Management routines - 10KB

P2b: is the reentrant Batch Report Writer and File Management routines - 25KB

Various I/A and Batch tasks ranging from 10 to 20 KB long.

WHAT SAVINGS CAN BE GAINED BY THIS SCHEME?

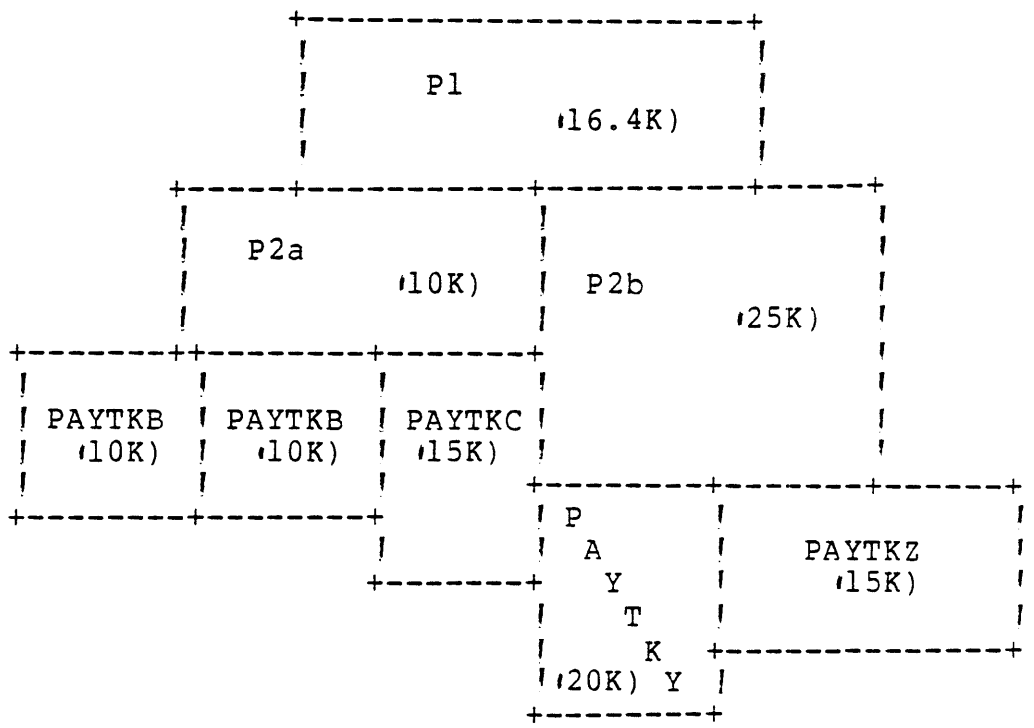


Figure 1-8

	PAYTKB	PAYTKC	PAYTKY	PAYTKZ
P1	16.4KB	16.4KB	16.4KB	16.4KB
P2a	10KB	10KB	----	----
P2b	----	----	25KB	25KB
TASK	20KB	15KB	20KB	15KB
TOTAL	46.4KB	41.4KB	61.4KB	56.4KB
TOTAL is 205.6KB				

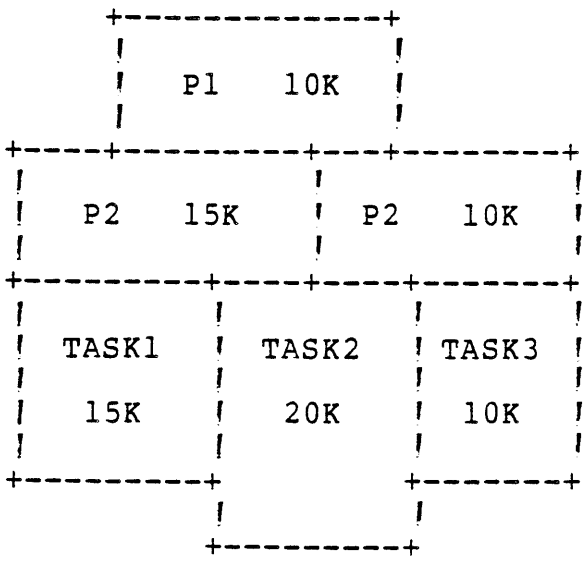
Figure 1-9 Whithout Sharing Procedures

	PAYTKB	PAYTKC	PAYTKY	PAYTKZ
P1	16.4KB	----	----	----
P2	10KB	----	----	----
P2	----	----	25KB	----
TASK	20KB	15KB	20KB	15KB
TOTAL	46.4KB	15KB	45KB	15KB
TOTAL is 121.4KB				

Figure 1-10 With Shared Procedures

WORKSHEET

By considering the following structure, what savings can be gained in memory by sharing P1 and P2?



1.3 COMPILER OUTPUT

The COBOL Compiler puts out PSEG AND DSEG tags for use by the Link Editor. The PSEG or Program Segment Directive contains reentrant code and the DSEG or Data Segment Directive contains the non-reentrant portion. Since the segments are tagged, the Link Editor is able to separate the procedure portion from the task portion in each subroutine. (Refer to Figure 1-11.)

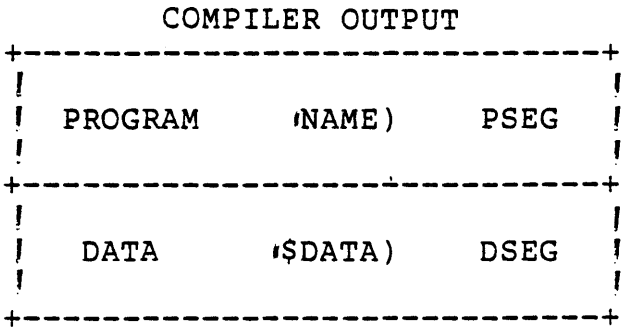


Figure 1-11

When compiler produced object modules are linked, the PSEGs are allocated before the DSEGs. Figure 1-12 shows a portion of a link map.

Note that the first $DATA, or DSEG module, is allocated after the last PSEG, which in this example is PAYR07. The $DATA or DSEG modules are said to float to the end of the task. Thus a task with no reentrant procedures would be allocated in memory by the Link Editor as shown in Figure 1-12.

S A M P L E L I N K M A P

PROCEDURE 1, RCOBOL ORIGIN = 0000

MODULE	NO	ORIGIN	LENGTH
CRTIM	1	0000	34C9

PHASE 0, PAYTKA ORIGIN = 34E0

MODULE	NO	ORIGIN	LENGTH
CXCBL	2	34E0	9BD8
$DATA	2	6BFE	04F0
C$MAIN	3	40B8	0010
PAYRO1	4	40C8	0276
$DATA	4	70EE	0051
PAYRO2	5	433E	0204
$DATA	5	7140	0078
PAYRO3	6	4542	0900
$DATA	6	71B8	014A
PAYRO4	7	4E42	052E
$DATA	7	7302	03EA
PAYRO5	8	5370	0A1A
$DATA	8	76EC	031C
PAYRO6	9	5D8A	0428
$DATA	9	7A08	041C
PAYRO7	10	61C2	0A4C
$DATA	10	7E24	0374

CORRESPONDING "MEMORY MAP"

MEMORY ORIGIN	MODULE NAME	MODULE LENGTH	NO
0000	CRTIM	34C9	1
34E0	CXCBL	0BD8	2
40B8	C$MAIN	0010	3
40C8	PAYRO1	0276	4
433E	PAYRO2	0204	5
4542	PAYRO3	0900	6
4E42	PAYRO4	052E	7
5370	PAYRO5	0A1A	8
5D8A	PAYRO6	0428	9
61B2	PAYRO7	0A4C	10
6BFE	$DATA	04F0	2
70EE	$DATA	0052	4
7140	$DATA	0078	5
71B8	$DATA	014A	6
7302	$DATA	03EA	7
76EC	$DATA	031C	8
7A08	$DATA	041C	9
7E24	$DATA	0374	10

Figure 1-12 Example Link Edit

1.4 XCP AND XCPF

XCP or XCPF allows multiple tasks to be executed sharing a common COBOL runtime as shown in Figure 1-13.

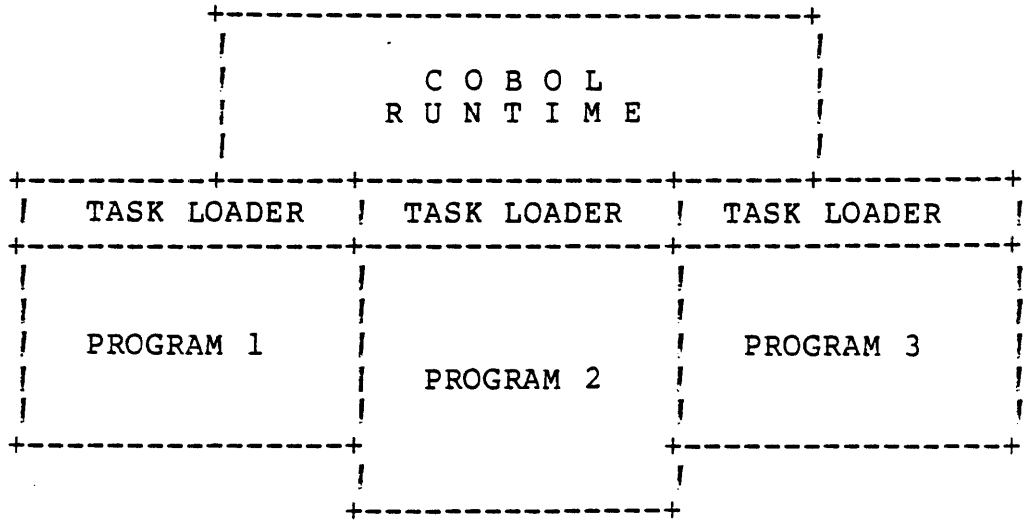


Figure 1-13

XCP or XCPF consists of the COBOL runtime as P1 and a task loader as the task attached to P1. The task loader checks the size of the user's object module; does a dynamic get memory for the amount of memory required; loads the COBOL program into the allocated memory; and starts execution of the loaded program.

1.5 SHARING ONLY THE RUNTIME

XCP and XCPF do share the COBOL runtime, but do not allow the user to use P2 or overlays with a COBOL program. In addition, the overhead for the task loader may be significant with large programs.

* Since you are loading an object program, instead of a memory image from a program file, the load process will take a few seconds longer.

* The 65KB memory address space is reduced by the small (200-300 bytes) memory requirements of this task loader.

Thus, the Execute COBOL Task (XCT) and Execute COBOL Task in Foreground (XCTF) were developed to eliminate the restrictions of XCP and XCPF. Since XCT and XCTF are identical in function,

except that XCT cannot directly do I/O to the station on which it is exeucted, the following discussion will use XCTF in text and examples. Use of XCT or XCTF implies that the user has used the Link Editor to link his COBOL modules with the COBOL runtime modules. The COBOL runtime contains three modules:

* RCBPRC - Reentrant runtime intrepreter (CRTIM).

* RCBTSK - Initial task module (CXCBL).

* RCBMPD - Module placed before the first COBOL module to be executed (C$MAIN).

The modules RCBPRC amd RCBMPD are both reentrant while RCBTSK is not reentrant and must be placed in the task segment. Thus, the simple linking of one COBOL program is shown in Figure 1-14.

```
FORMAT IMAGE
PROC CBLRT
INCLUDE  .S$SYSLIB.RCBPRC
TASK  PAYTK4
INCLUDE  .S$SYSLIB.RCBTSK
INCLUDE  .S$SYSLIB.RCBMPD
INCLUDE  .COBOL.PAYTK4
END
```

Figure 1-14 Link Control File for One Cobol Program

Assume that the program file to be used is named .COBOL.PROG and that the link edit shown in Figure 1-14 was performed to .COBOL.PROG. After linking, the program file has a copy of the reentrant COBOL runtime linked and installed as a procedure number 1 under the name CBLRT. A map of the program file is shown in Figure 1-15.

```
    FILE MAP OF .COBOL.PROG
        TODAY IS 16:06:09 TUESDAY, JAN 22, 1980.

TASKS:
 ID          NAME     LENGTH LOAD   PRI     SPMRD  OVLY   P1/SAME P2/SAME
 01          PAYTK4    11DE  3600    04     NNNYN         01/Y
PROCEDURES:
 ID          NAME     LENGTH LOAD           RES  D
 01          CBLRT     35F6  0000            N   N
OVERLAYS:
 ID          NAME     LENGTH LOAD           MAP  D  OVLY
```

Figure 1-15 Program File Map of .COBOL.PROG

Figure 1-16 shows why the RCBMPD is required in front of your main COBOL program when you make up your Link Control stream. All references within the COBOL runtime environment are direct refererences. RCBMPD, the reentrant COBOL runtime program, transfers control to RCBTSK, the COBOL nonreentrant runtime program; which then transfers control to your program via RCBMPD.

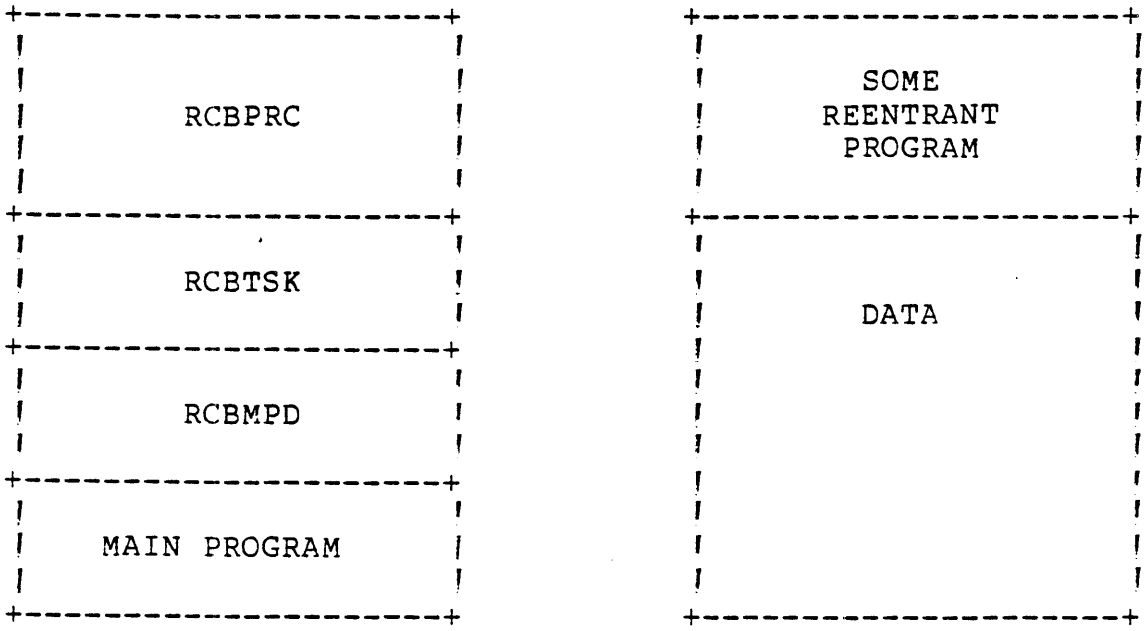


Figure 1-16

Another way to look at this situation is as in Figure 1-16. This figure depicts a COBOL program (DATA) in relation to the COBOL runtime interpreter, which is after all, just a large reentrant program.

QUESTION?

HOW WOULD YOU HANDLE THE FOLLOWING PROBLEM:

GIVEN:

* Direct referencing required in COBOL runtime environment.

* Our mapping HW can accomodate three program segments.

* Programs split into program 'reentrant code) and data 'non-reentrant) segments. 'eg. PSEGs and DSEGs).

PROBLEM:

How to share both P1 and P2 procedures with several, possibly different tasks simultaneously!

HINT:

The Link Editor can move all the DSEGs down into the non-reentrant 'task) segment. What would that do?

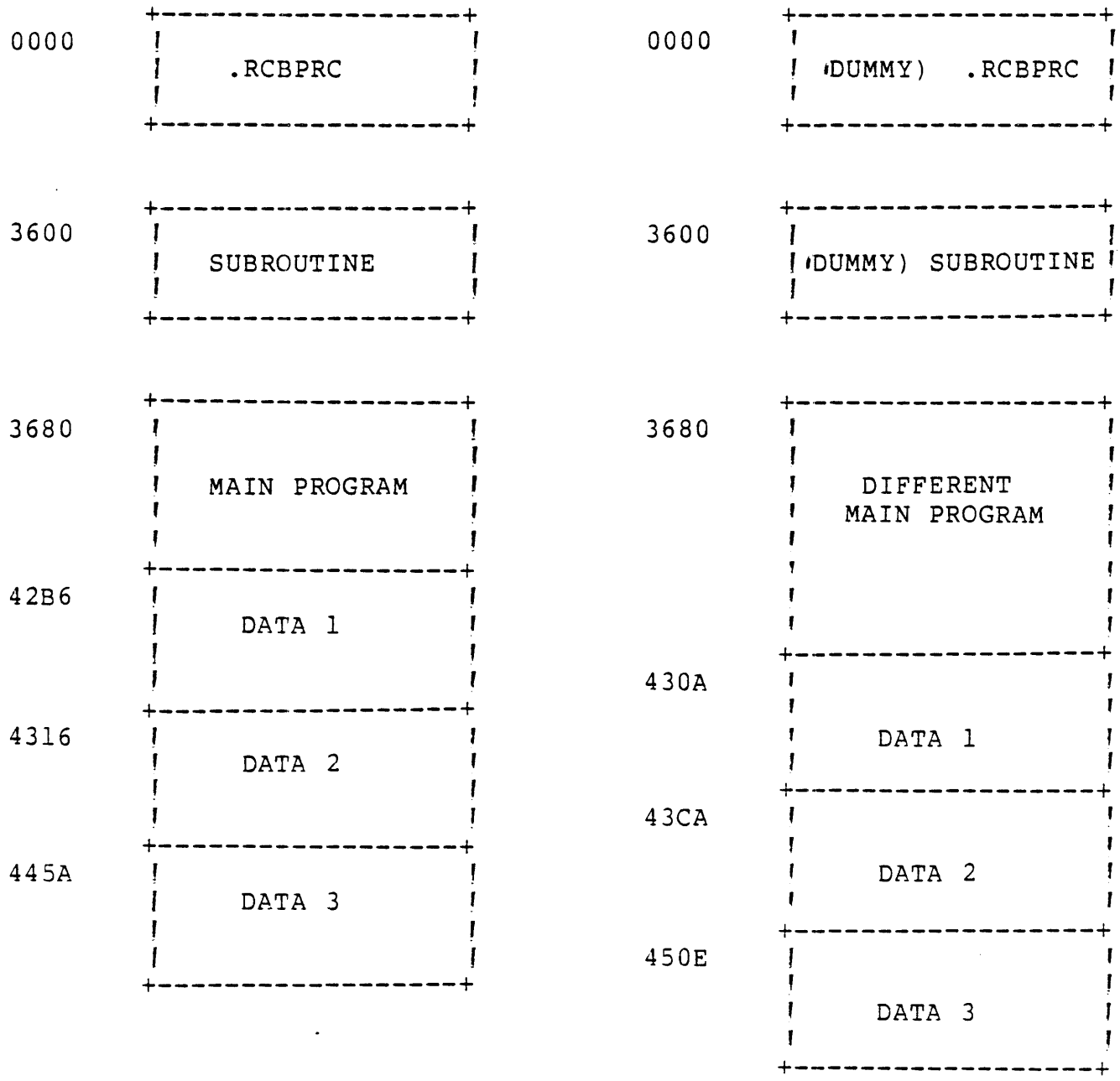

0000
.RCBPRC
3600
SUBROUTINE
3680
MAIN PROGRAM
42B6
DATA 1
4316
DATA 2
445A
DATA 3
0000
(DUMMY) .RCBPRC
3600
(DUMMY) SUBROUTINE
3680
DIFFERENT MAIN PROGRAM
430A
DATA 1
43CA
DATA 2
450E
DATA 3

Figure 1-16

The solution to this problem is the Link Editor ALLOCATE command. The ALLOCATE verb allows users to share COBOL source language programs as procedures. The ALLOCATE verb is used in the task section of the link control file, after a TASK or PHASE 0 command and before a PHASE 1 or a LOAD command, if any are used. It should normally be placed immediately following the INCLUDE .S$SYSLIB.RCBTSK statement. Its appearance causes all data sections associated with previously allocated executable code sections to be allocated immediately. That is, all $DATA sections associated with program sections, in either P1 or P2, are allocated space wherever the ALLOCATE command occurs in the link control file. However, the procedure cannot call a subroutine included in the link edit after the ALLOCATE. Figure 1-17 will help illustrate what the ALLOCATE verb does by contrasting the same link edit stream with and without an ALLOCATE verb.

```
FORMAT IMAGE
PROC RCOBOL
DUMMY
INCLUDE .S$SYSLIB RCBPRC
PROCEDURE PSUB
INCLUDE .COBOL.PAYSCN
TASK PAYTKB
INCLUDE .S$SYSLIB.RCBTSK
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE .COBOL.PAYTKB
END


PROCEDURE 1, RCOBOL ORIGIN = 0000

MODULE    NO       ORIGIN     LENGTH

CRTIM      1        0000       35F5


PROCEDURE 2, PSUB   ORIGIN = 3600

MODULE    NO       ORIGIN     LENGTH

PAYSCN     2        3600       0074
$DATA      2        4316       0030


PHASE 0, PAYTKB  ORIGIN = 3680

MODULE    NO       ORIGIN     LENGTH

CXCBL      3        3680       0C26
$DATA      3        4346       04EC
C$MAIN     4        42A6       0010
PAYTKB     5        42B6       0060
$DATA      5        4832       007E
```

```
FORMAT IMAGE
PROC RCOBOL
DUMMY
INCLUDE .S$SYSLIB.RCBPRC
PROCEDURE PSUB
INCLUDE .COBOL.PAYSCN
TASK PAYTKB
INCLUDE .S$SYSLIB.RCBTSK
ALLOCATE
INCLUDE .S$SYSLIB.RCBMPD
INCLUDE .COBOL.PAYTKB
END


PROCEDURE 1, RCOBOL ORIGIN=0000

MODULE    NO       ORIGIN   LENGTH

CRTIM      1        0000     35F5


PROCEDURE 2, PSUB ORIGIN = 3600

MODULE    NO        ORIGIN   LENGTH

PAYSCN     2        3600      0074
$DATA      2        42A6      0030


PHASE 0, PAYTKB  ORIGIN = 3680

MODULE    NO       ORIGIN   LENGTH

CXCBL      3       3680      0C26
$DATA      3       42D6      0CEC


        (POST ALLOCATE)

 PHASE 0, PAYTKB  ORIGIN = 47C2

 MODULE    NO       ORIGIN   LENGTH

 C$MAIN     4        47C2     0010
 PAYTB2     5        47D2     0060
 $DATA      5        4832     007E
```

Figure 1-17 Use of the ALLOCATE Verb

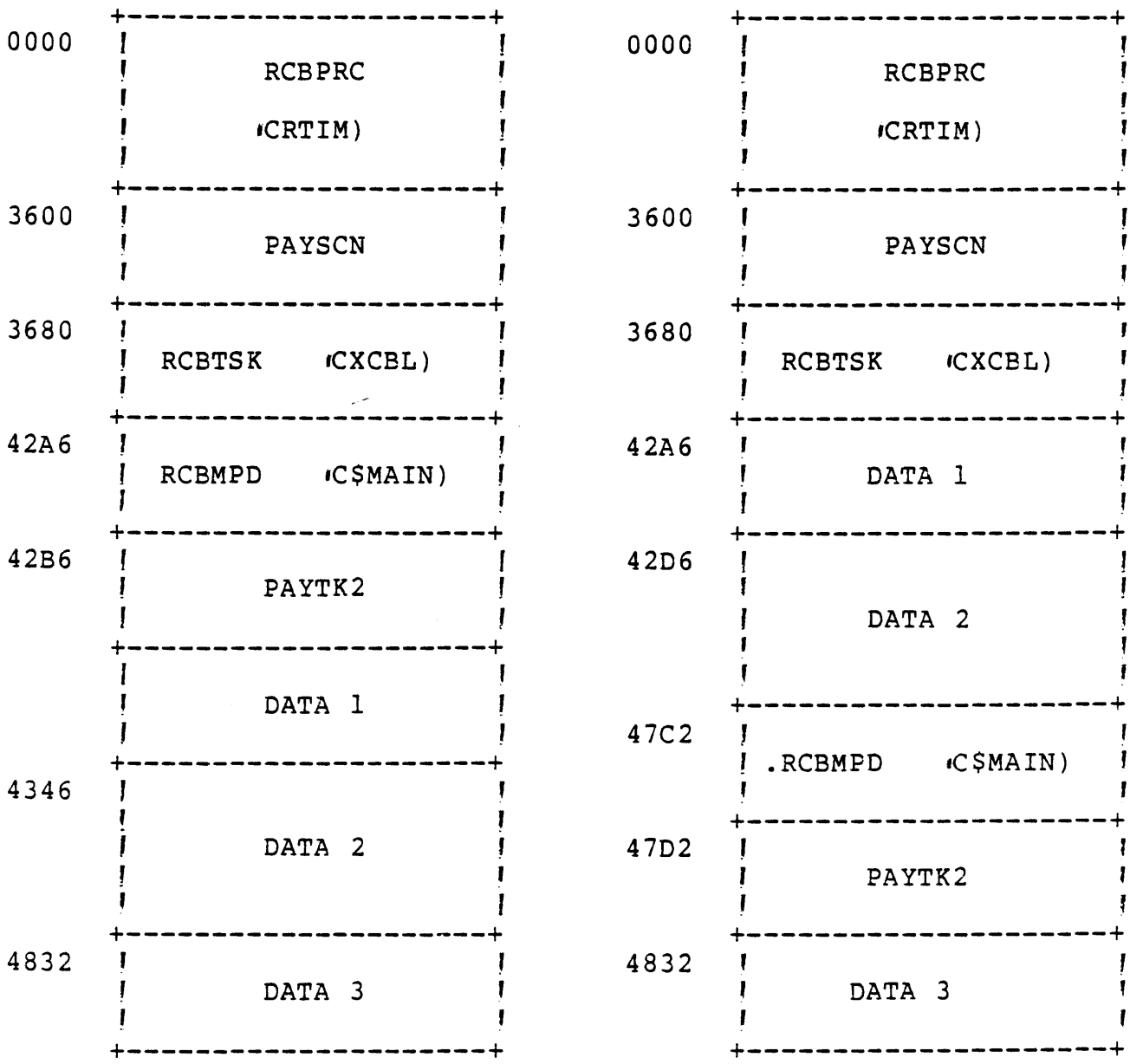


Figure 1-18 Memory Allocation With and Without the Allocate Verb

This allocation of the $DATA modules is critical in achieving the task structure as shown in Figure 1-19.

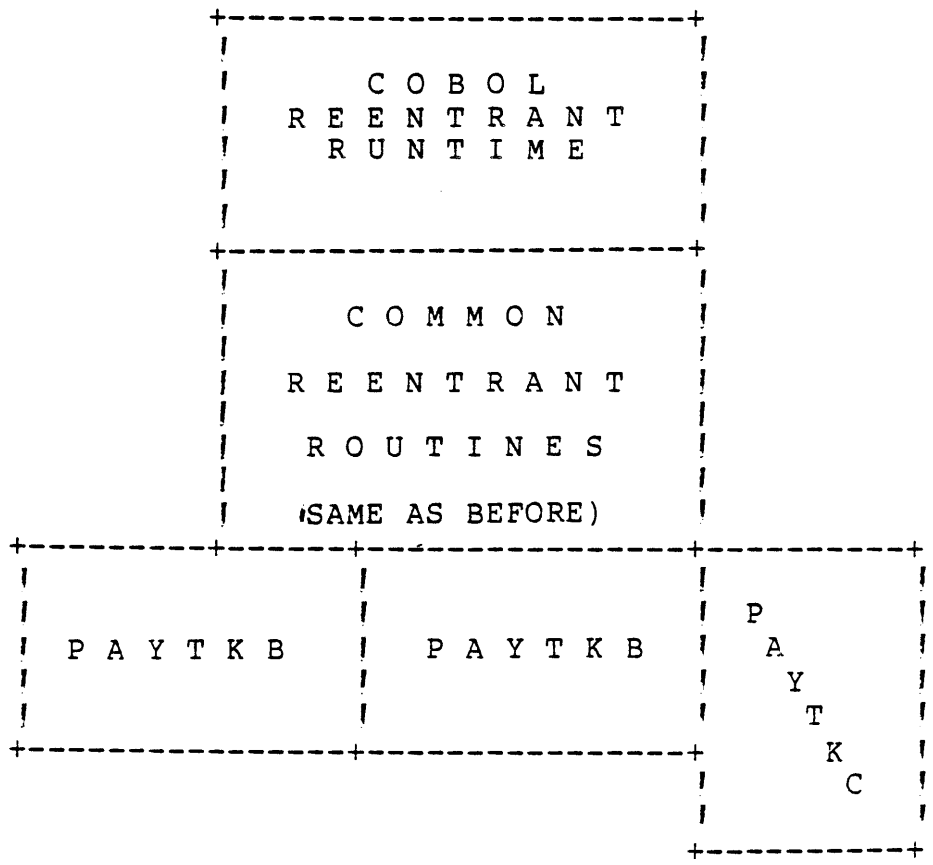


Figure 1-19

Figure 1-20 shows the link control stream for PAYTKC, which is not the same task as PAYTKB, but shares the same P1 and P2. PAYTKB must have been previously link edited.

```
     FORMAT IMAGE
PROC RCOBOL
DUMMY
INCLUDE .S$SYSLIB.RCBPRC
PROC PSUBS
DUMMY
INCL .COBOL.PAYIO1
INCL .COBOL.PAYIO2
INCL .COBL. PAYFIN
INCL .COBOL.PAYCMN
INCL .COBOL.PAYSCN
TASK PAYTKC
INCL .S$SYSLIB.RCBTSK
ALLOCATE
INCL .S$SYSLIB.RCBMPD
INCL .COBOL.PAYTKC
END
```

Figure 1-20 Link Control Stream for PAYTKC

PSUBS contains a DUMMY statement because PSUBS was already installed on the program file by the Link Editor as shown in Figure 1-21 which shows the link maps for both PAYTKB and PAYTKC. Note that .PAYTKC is larger than .PAYTKB, but the the $DATAs of PSUBS remain in the same positions in the task due to the ALLOCATE verb. Note also that DATA 10 is not in the same position, because of the increased size of .PAYTKC.

```
PROCEDURE 1, RCOBOL ORIGIN=0000
MODULE    NO      ORIGIN   LENGTH

CRTIM      1        0000     34C9


PROCEDURE 2, PSUBS  ORIGIN=34E0
MODULE    NO      ORIGIN    LENGTH
PAYIO1     2        34E0      0204
$DATA      2        6978      0078
PAYIO2     3        36E4      0900
$DATA      3        69F0      014A
PAYFIN     4        3FE4      052E
$DATA      4        6B3A      03EA
PAYFOU     5        3412      0A1A
$DATA      5        6F24      031C
PAYCMN     6        4F2C      0428
$DATA      6        7240      041C
PAYSCN     7        5354      0A4C
$DATA      7        765C      0374


PHASE 0, PAYTKB      ORIGIN=5DA0
MODULE    NO      ORIGIN   LENGTH

CXCBL      8        5DA0     0BD8
$DATA      8        79D0     04F0


-------- POST ALLOCATE --------

PHASE 0, PAYTKB      ORIGIN=7EC0
MODULE    NO      ORIGIN   LENGTH

C$MAIN     9        7EC0     0010
PAYTKB    10        7ED0     0276
$DATA     10        8146     0052
```

```
PROCEDURE 1, RCOBOL ORIGIN=0000
MODULE    NO      ORIGIN   LENGTH

CRTIM      1        0000     34C9


PROCEDURE 2, PSUBS ORIGIN=34E0
MODULE    NO      ORIGIN    LENGTH
PAYIO1     2        34E0      0204
$DATA      2        6978      0078
PAYIO2     3        36E4      0900
$DATA      3        69F0      014A
PAYFIN     4        3FE4      052E
$DATA      4        6B3A      03EA
PAYFOU     5        4512      0A1A
$DATA      5        6F2C      031C
PAYCMN     6        4F2C      0428
$DATA      6        7240      041C
PAYSCN     7        5354      0A4C
$DATA      7        765C      0374


PHASE 0, PAYTKC      ORIGIN=5DA0
MODULE    NO      ORIGIN   LENGTH

CXCBL      8        5DA0     0BD8
$DATA      8        79D0     04F0


-------- POST ALLOCATE -------

PHASE 0, PAYTKC      ORIGIN=7EC0
MODULE    NO      ORIGIN   LENGTH

C$MAIN     9        7EC0     0010
PAYTKC    10        7ED0     1476
$DATA     10        9346     07D6
```

Figure 1-21 Link Maps for PAYTKB and PAYTKC

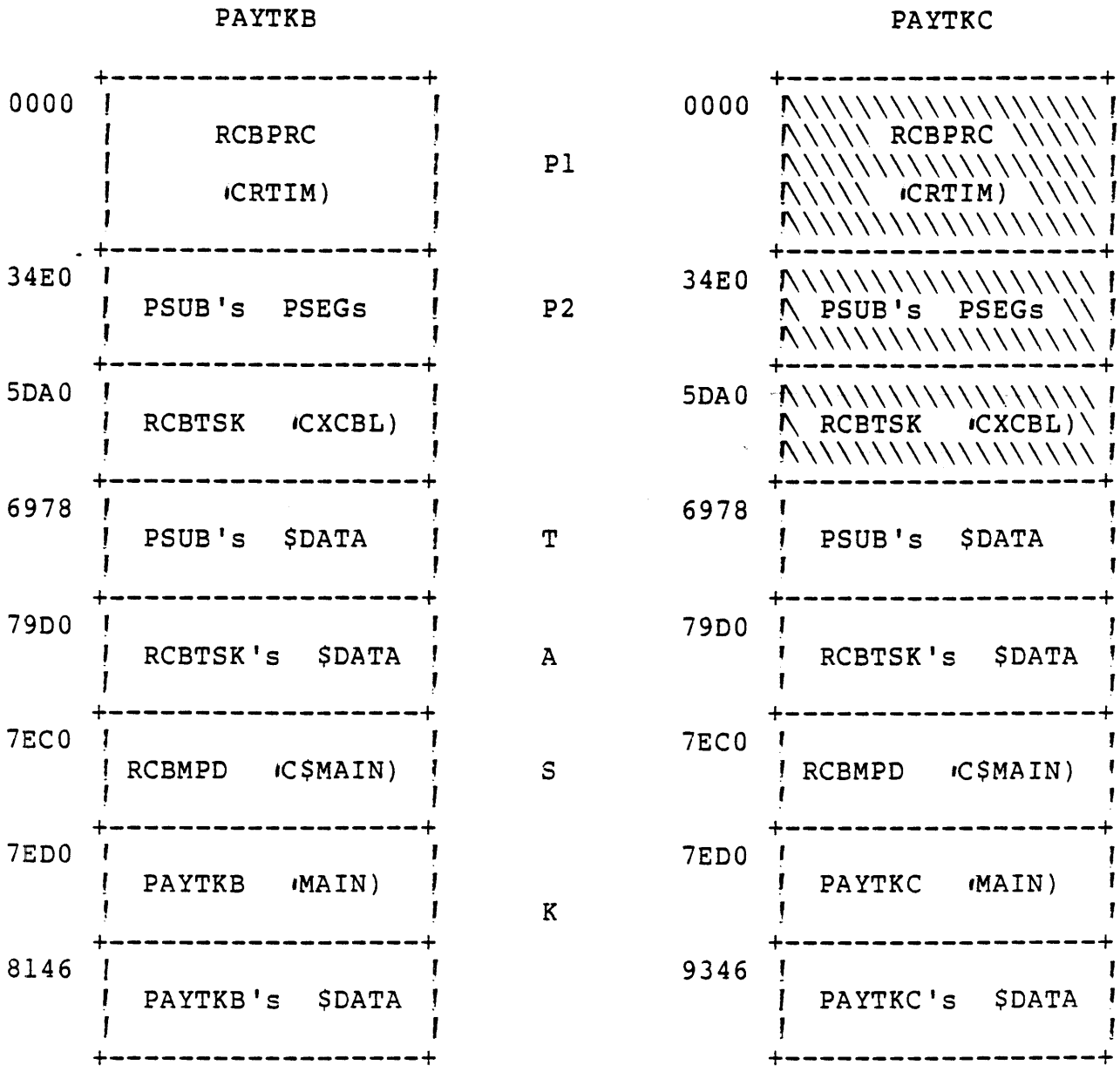

PAYTKB
PAYTKC
0000
RCBPRC
(CRTIM)
P1
34E0
PSUB's PSEGs
P2
5DA0
RCBTSK (CXCBL)
6978
PSUB's $DATA
T
79D0
RCBTSK's $DATA
A
7EC0
RCBMPD (C$MAIN)
S
7ED0
PAYTKB (MAIN)
K
8146
PAYTKB's $DATA
0000
RCBPRC
(CRTIM)
34E0
PSUB's PSEGs
5DA0
RCBTSK (CXCBL)
6978
PSUB's $DATA
79D0
RCBTSK's $DATA
7EC0
RCBMPD (C$MAIN)
7ED0
PAYTKC (MAIN)
9346
PAYTKC's $DATA

Figure 1-22

Another way to look at this structure is execute a Map Program File (MPF) command. Figure 1-23 shows the MPF's output and the corresponding logical strucutre of the firm's payroll system. Note that PAYTKA is a seperate program that shares only the COBOL runtime.

```
 FILE MAP OF .COBOL.PROG
   TODAY IS 10:02:08 TUESDAY, JAN 22, 1980

ASKS:
 ID  NAME   LENGTH LOAD  PRI  SPMDP  OVLY P1/SAME P2/SAME  INSTALL
 01 PAYTKA  3CB8   34E0   04  NNNYN       10/N           01/20/80
 02 PAYTKB  23F8   5DA0   04  NNNYN       10/N    01/Y   01/20/80
 03 PAYTKC  3D7C   5DA0   04  NNNYN       10/N    01/Y   01/21/80
 04 PAYTKY  4E28   96A0   04  NNNYN       10/N    02/Y   01/21/80
 05 PAYTKZ  3A84   06A0   04  NNNYN       10/N    02/Y   01/22/80
ROCDURES:
 ID  NAME   LENGTH LOAD      RES  D
 01  PSUBS  28C0   34E0       N   N
 02  PSUBB  61A8   34E0       N   N
VERLAYS:
 ID  NAME   LENGTH LOAD      MAP  D  OVLY                  INSTALL
```

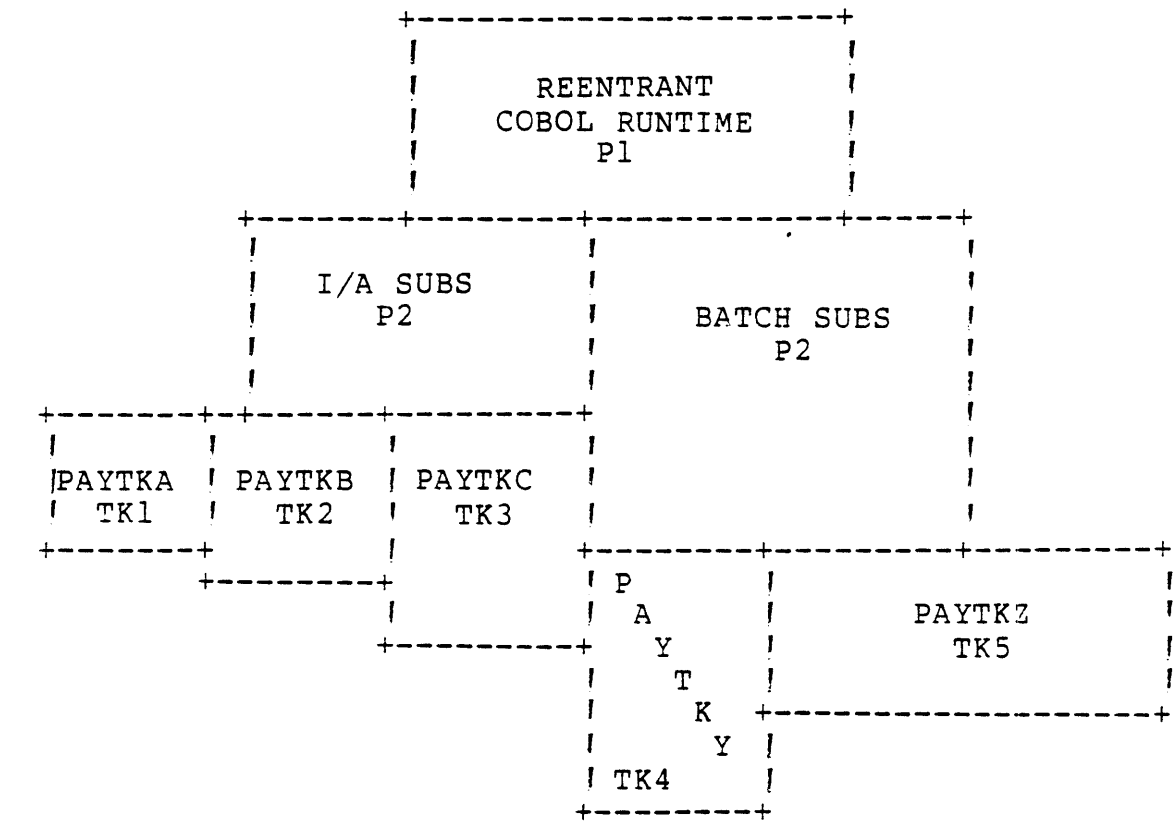


Figure 1-23 Program File Map for .COBOL.PROG

1.6 SUMMARY

Using the ALLOCATE verb it is possible to construct run units whose first procedure consists of the truly reentrant portion of the COBOL runtime system, called .S$SYSLIB.RCBPRC, and whose second procedure is a set of non-reentrant COBOL and/or assembly language runtimes. The $DATA sections for the routines can be forced to loaded immediately after the non-reentrant portion of the COBOL runtime system, called .S$SYSLIB.RCBTSK, by using the ALLOCATE verb. Even though the tasks associated with the two different run units may be different, the $DATA areas are located in identical locations, thereby allowing the direct references in the second procedure to be completed successfully. Structures as shown in Figure 1-23 can be built by adhering to the following rules for COBOL:

* Sharing of .S$SYSLIB.RCBPRC with multiple tasks or multiple executions of the same task does not require the ALLOCATE verb since RCBPRC is truly reentrant and does not address a $DATA directly. RCBPRC could also be in P2.

* The shared procedure must always contain the same modules. If a module is changed in P1 or P2, the entire group of tasks must be link edited again to recreate the original structure.

* Modifications may be made to modules which occur after the ALLOCATE command without requiring a relink of all tasks.

* P1 and/or P2 must always be installed by the first module linked and DUMMY must be used after this first installation.

* In COBOL, RCBTSK must always be the first module in the TASK or PHASE 0 portion of the link edit. ALLOCATE is usually put immediately after RCBTSK, but does not have to go there. Caution should be exercised when using ALLOCATE in a different place with COBOL.

* In debugging complex task structures, it is best to keep a full set of memory maps produced by the Link Editor. If problems arise, check the addresses generated for the $DATAs in the procedures.

1.7 LINKING vs. NOT LINKING

A COBOL program may be executed without being linked first under certain conditions. The program may not call any subroutines and may not contain overlays. The first

restriction while requiring the use of the link editor does not require you to install the linked output on a program file. There are three distinct advantages to linking and installing the linked output on a program file. They are:

1. Linked programs load faster than unlinked programs since the link editor converts object code into image format which is executable code. A task loader is not required as when using XCP.

2. COBOL programs, when linked into two sharable procedure areas and a nonsharable task are, require less memory at runtime when multiple executions of the program occur at different stations at the same time. If less memory is required, the necessity to roll programs in and out of memory will be reduced.

3. Unlinked programs all run at the same priority as the COBOL runtime interpreter (task >8A in .SSDS). Linked COBOL programs may each be assigned a individual priority. This allows the user to finetune a program by assigning a priority that will improve execution.

It is advisable to link all programs of one type (e.g., payroll, A/P, A/R, etc.) into the same program file. This will allow for the maximization of shared code.

WORKSHEET 1

Write link control files to install the COBOL RUNTIME as procedure 1, user specified library (ULIB) as procedure 2, and attach two tasks (TASK1, TASK2) to both P1 and P2. ULIB consists of three sub-programs (INSUB, ADDNUM, OUTSUB). TASK1 and TASK2 consist of one program called PROG1 and PROG2 respectively. All of the programs are stored in the directory TI.COBOL.OBJ.

MODULE 2

SEGMENTATION AND OVERLAYS

OBJECTIVES

* Use the COBOL segmentation feature to share available memory among units of the same program.
* Use link edited overlays to load large programs into a limited memory space.
* Write and execute link control streams to create overlaid tasks.

2.1 SHARING MEMORY USING COBOL SEGMENTATION

Since available main memory is frequently an important consideration to the minicomputer applications programmer, such a person should become familiar with methods of overlapping usage of this resource, TI 990 COBOL provides two powerful techniques for sharing main memory, COBOL segmentation and link edited overlays

COBOL segmentation is a syntactical division of the PROCEDURE DIVISION of a single COBOL program into shareable units, called segments. Link edited overlays, on the other hand, divide the task in question into shareable units which consist of separately compilable, whole COBOL programs. Several important differences exist between the two approaches as shown in Figure 2-1.

In general, segmentation is more nearly transportable between different COBOL systems but lacks the flexibility of modular program development provided by link edited overlays.

NOTE

A programmer may legally use both sharing techniques in the same task. That is, a link edited, overlaid program unit may also be segmented.

SEGMENTATION	LINK EDITED OVERLAYS
* SUBSET OF REGULAR ANSI STANDARDS.	* NOT AN ANSI FEATURE. IMPLEMENATION VARIES FROM ONE OPERATING SYSTEM TO ANOTHER.
* COMMUNICATION BETWEEN SEGMENTS IS VIA ANY NON-CALL CONTROL VERB OR MECHANISM SUCH AS GO TO, PERFORM, AND NORMAL CONSECUTIVE STATEMENT EXECUTION.	* COMMUNICATIONS BETWEEN SHAREABLE UNITS IS VIA CALL VERB (WITH AUTOMATIC OVERLAY LOADER ASSISTANCE).
* SHAREABLE UNITS MAY RESIDE ON DISK-BASED PROGRAM FILE (XCT) OR RELATIVE RECORD FILE (XCP). ARE LOADED BY EITHER AUTOMATIC OVER-LAY LOADER (XCT) OR COBOL RUNTIME LOADER, C$XLDR (XCP).	* SHAREABLE UNITS RESIDE ON DISK-BASED PROGRAM FILE.
	* LOAD OVERLAY SVC CALL BY AUTOMATIC OVERLAY LOADER LOADS THEM INTO MEMORY AS NEEDED.

Figure 2-1

Using segmentation effectively is very analogous to using link edited overlays. The user must seek out ways to divide the progam unit into more or less independent pieces. The term independent is used here to mean that, within each piece, a high percentage of all control references are internal, and the piece performs one or at most a few easily discernable logical functions such as update a log file, generate a single report, etc.

The independent pieces, or units which share memory, are called segments. Each segment in turn consists of one or more sections. Segments are identified by an integer number between 0 and 127. The formal structure of a segment is as follows:

```
section-name SECTION segment-number.
paragraph-name.  sentence ...  ...  ...
```

The notation used here is the same as that used in the COBOL manual. A typical segment then might be coded as follows:

```
        TEST-SEQ-OUTPUT  SECTION 60.

        DO-OUTPUT.

            OPEN OUTPUT SEQ-FILE

            MOVE QUOTES TO SEQ-REC.

            PERFORM POST-ELAPED-TIME.

            PERFORM SEQ-OUTPUT-LOOP LIMIT TIMES.

            PERFORM POST-ELAPSED-TIME

            CLOSE SEQ-FILE.

        TEST-SEQ-INPUT  SECTION 60.

        DO-INPUT.

            OPEN INPUT  SEQ-FILE

            PERFORM POST-ELAPSED-TIME.

            PERFORM SEQ-OUTPUT-LOOP LIMIT TIMES.

            PERFORM POST-ELAPSED-TIME.

            CLOSE SEQ-FILE.

            GO TO END-OF-60.

        SEQ-PERFORMS  SECTION 60.

        SEQ-OUTPUT-LOOP.

            WRITE SEQ-REC.

        SEQ-INPUT-LOOP.

            READ SEQ-FILE RECORD.

        END-OF-60.
```

Notice that in this example, the segment was divided into three SECTIONs. We could have just as easily structured the segment as one section with more paragraphs. For example, it could also have been coded:

```
TEST-I-O  SECTION 60.

TEST-SEQ-OUTPUT.

    OPEN OUTPUT SEQ-FILE.

    MOVE QUOTES TO SEQ-REC.

    PERFORM POST-ELAPSED-TIME.

    PERFORM SEQ-OUTPUT-LOOP LIMIT TIMES.

    PERFORM POST-ELAPSED-TIME.

    CLOSE SEQ-FILE.

TEST-SEQ-INPUT.

    OPEN INPUT SEQ-FILE.

    PERFORM POST-ELAPSED-TIME.

    PERFORM SEQ-OUTPUT-LOOP LIMIT TIMES.

    PERFORM POST-ELAPSED-TIME.

    CLOSE SEQ-FILE.

    GO TO END-OF-60.

SEQ-OUTPUT-LOOP,

    WRITE SEQ-REC.

SEQ-INPUT-LOOP.

    READ  SEQ-FILE RECORD.

END-OF-60.
```

This second configuration of the source program would be logically identical to the first and would also have identical sharing attributes as the first. So, while at least one SECTION label must be used to identify the segment, additional SECTION labels with the same segment-number could just as easily be paragraph names.

Remember segments are the unit of sharing, not sections. All sections of a single segment must be grouped together in the source program.

Just as in the case with overlays, at least one segment must remain resident at all times. This segment is called the root,

dependent, or fixed segment. The root does not share main memory with any other segment. All the other segments are called independent segments and one at most will be in memory at one time. Independent segments then share a single piece of memory. Whenever one independent segment references another, the referenced segment is copied into memory before execution procedes. Needless to say, one does not want to go back and forth between segments too often. (Refer to Figure 2-2.)

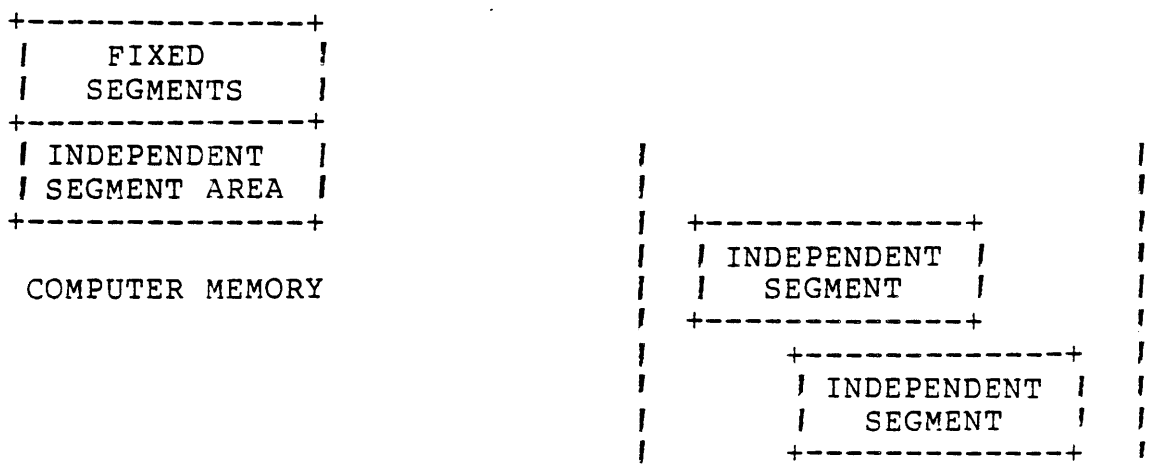


Figure 2-2

Segment numbers are assigned as follows:

* If one wants the segment to be in the root, assign a number between 0 and 49, inclusive.

* If one wants the segment to be independent, i.e., shareable, assign a number between 50 and 127, inclusive.

* Sections without segment numbers are assigned to segment number 0.

Segment numbers should be assigned using the following additional guidelines:

* Logic Requirements - Segments which must be available for reference at all times, or which are referred to frequently, are normally classified as being one of the fixed segments. Segments which are used less frequently are normally classified as being one of the independent segments, depending on logic requirements.

* Frequency of Use - Generally, the more frequently a

segment is referred to, the lower its segment number; the less frequently it is referred to, the higher its segment number.

* Relationship Between Sections - Sections which frequently communicate with one another should be given the same segment numbers.

When segmentation is used, the entire PROCEDURE DIVISION must be in sections. Since the DECLARATIVES must consist of segments, using DECLARATIVES forces the entire program to be in sections.

NOTE

Segments in the DECLARATIVES always belong to the root. Therefore, they must have segment-numbers less than 50.

Figure 2-3 shows an example of a segmented COBOL program.

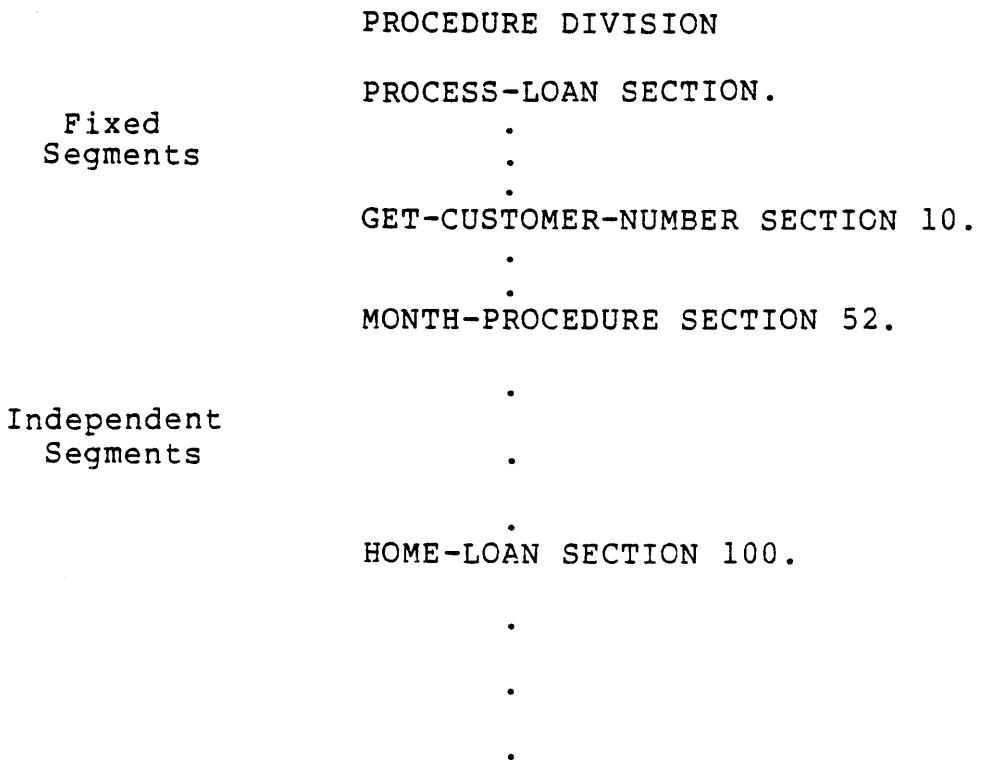


Figure 2-3

2.2 LINK EDITOR OVERLAYS

When memory space is at a premium, the user may find it advantageous to use overlays in his programs. Programs that do not use overlays are loaded in their entirety into the system and remain in memory while execution continues. Programs that use overlays conserve memory space since each overlay is resident in memory only when it is called. The total memory space required by the program is that memory which is required to hold the root portion and the longest overlay path.

For example, consider the structure shown in Figure 2-4. If the length of each path is given as in Figure 2-4, this program would require hexadecimal 118E bytes of memory.

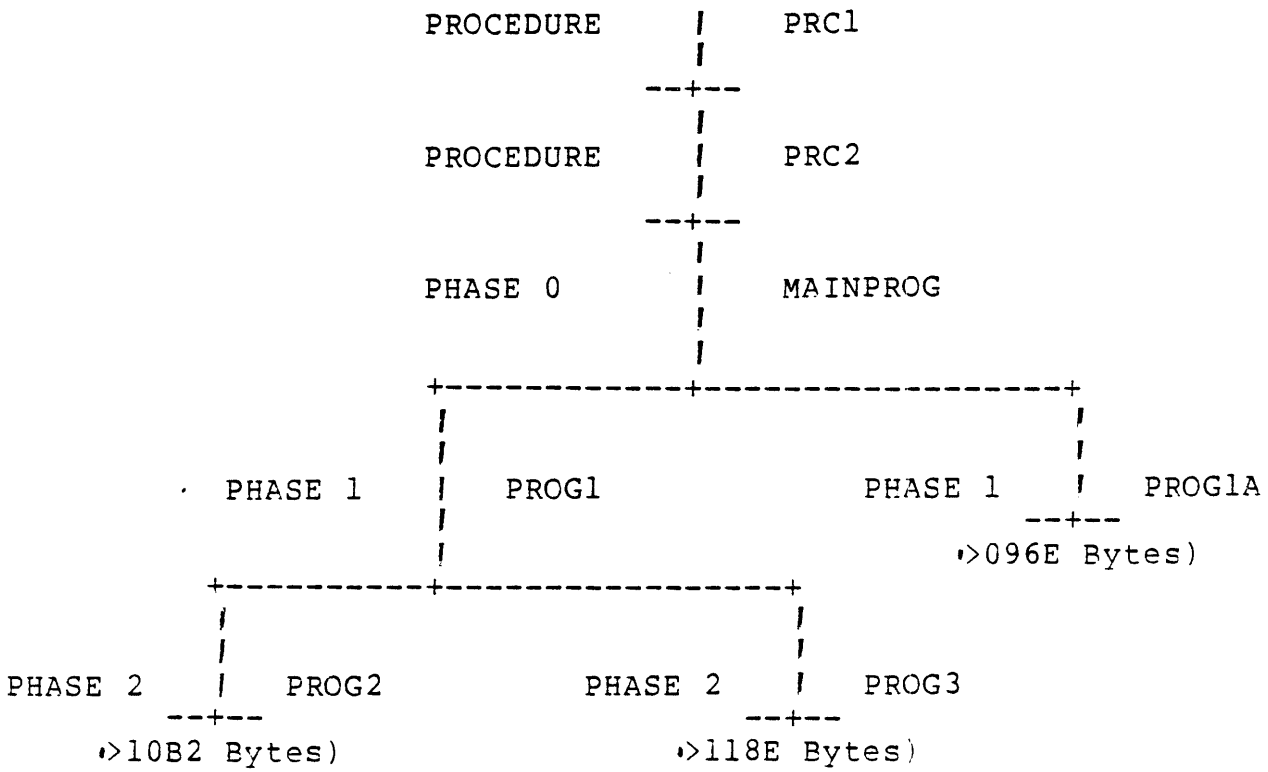


Figure 2-4

2.3 STRUCTURE CONSIDERATIONS

The structure of an overlaid program is dependent upon the relationships between the phases in the program. Phases that do not have to be in memory at the same time can overlay each other. These phases are considered to be independent in that they do not reference each other directly or indirectly. Independent phases can be assigned the same load address and can be loaded only when referenced. When a specific phase is called, all phases in its

path must be in memory.

Figure 2-5 shows an example of the link control file necessary to construct the overlay structure given in Figure 2-4.

```
PROC  PRC1
      .
      .
PROC  PRC2
      .
      .
PHASE 0,  MAINPROG
      .
      .
PHASE 1,  PROG1
      .
      .
PHASE 2,  PROG2
      .
      .
PHASE 2,  PROG3
      .
      .
PHASE 1,  PROG1A
      .
      .
END
```

Figure 2-5

Figure 2-6 shows an example of an overlay that has been installed on a program file. This assumes that a copy of the procedure RCOBOL, which in this example is the COBOL runtime, is already resident in the system program file.

* DUMMY COMMAND causes the Link Editor to suppress the linked output for the procedure, task, or phase in which it appears.

* PHASE COMMAND specifies the level and name of the overlay phase. Phases at level 1 or higher are disk resident overlays and are loaded into memory when called by a phase already in memory. Phase 0 is always a memory resident phase. (Corresponds to a module designated by a TASK command.)

* LOAD COMMAND causes the Automatic Overlay Manager to be included in the linked output. The LOAD command is only applicable when IMAGE format is being used and should appear in the root phase. The random library .S$SYSLIB is also required with the LOAD command.

```
LIBRARY  .S$SYSLIB
FORMAT  IMAGE, REPLACE
PROC  RCOBOL
DUMMY
INCL   (RCBPRC)
PHASE 0,  CLCK
INCL   (RCBTSK)
LOAD
INCL   (RCBMPD)
INCL  TI.COBOL.OBJ.CLCKMAIN
PHASE 1,  CLCK1
INCL  TI.COBOL.OBJ.CLCKSUB1
PHASE 2,  CLCK2
INCL  TI.COBOL.OBJ.CLCKSUB2
PHASE 1,  CLCK3
INCL  TI.COBOL.OBJ.CLCKSUB3
END
```

Figure 2-6 Link Control Stream to Install an Overlaid Task

Figure 2-7 shows the map of the program file after execution of the link control file specified in Figure 2-6.

```
  FILE MAP OF TI.COBOL.PROGF
    TODAY IS 15:32:26 TUESDAY, JAN 22, 1980

TASKS: MAXIMUM POSSIBLE =  10
 ID    NAME   LENGTH LOAD PRI  S P M R D E O C  OVLY  P1/SAME P2/SAME
 01    CLCK    15D2  3620   4        R          03   10/N

PROCEDURES: MAXIMUM POSSIBLE =   5
 ID    NAME   LENGTH LOAD     M D E W C

OVERLAYS: MAXIMUM POSSIBLE =  10
 ID    NAME   LENGTH LOAD   MAP  D  OVLY
 01    CLCK1   0140  49F0
 02    CLCK2   0202  4C00            01
 03    CLCK3   01C0  49F0            02
```

Figure 2-7

2.4 PARTIAL LINKS

A partial link is useful when a very large or complicated structure is being defined as may be the case with a large overlay structure. A partial link enables you to link various modules that are to be included as part of the executable output. References that occur within a functional grouping will be resolved. The PARTIAL command requires that a TASK or a PHASE command be included in the link control file. The output from the partial link may be either normal (ASCII) or compressed object. The ouput is not executable and must be linked again before execution.

Figure 2-8 shows an example of how a partial link would be included as part of a larger link control file.

```
PARTIAL
TASK  PRTPHS
INCL  TI.COBOL.OBJ.RE1O
INCL  TI.COBOL.OBJ.IS1O
END
```

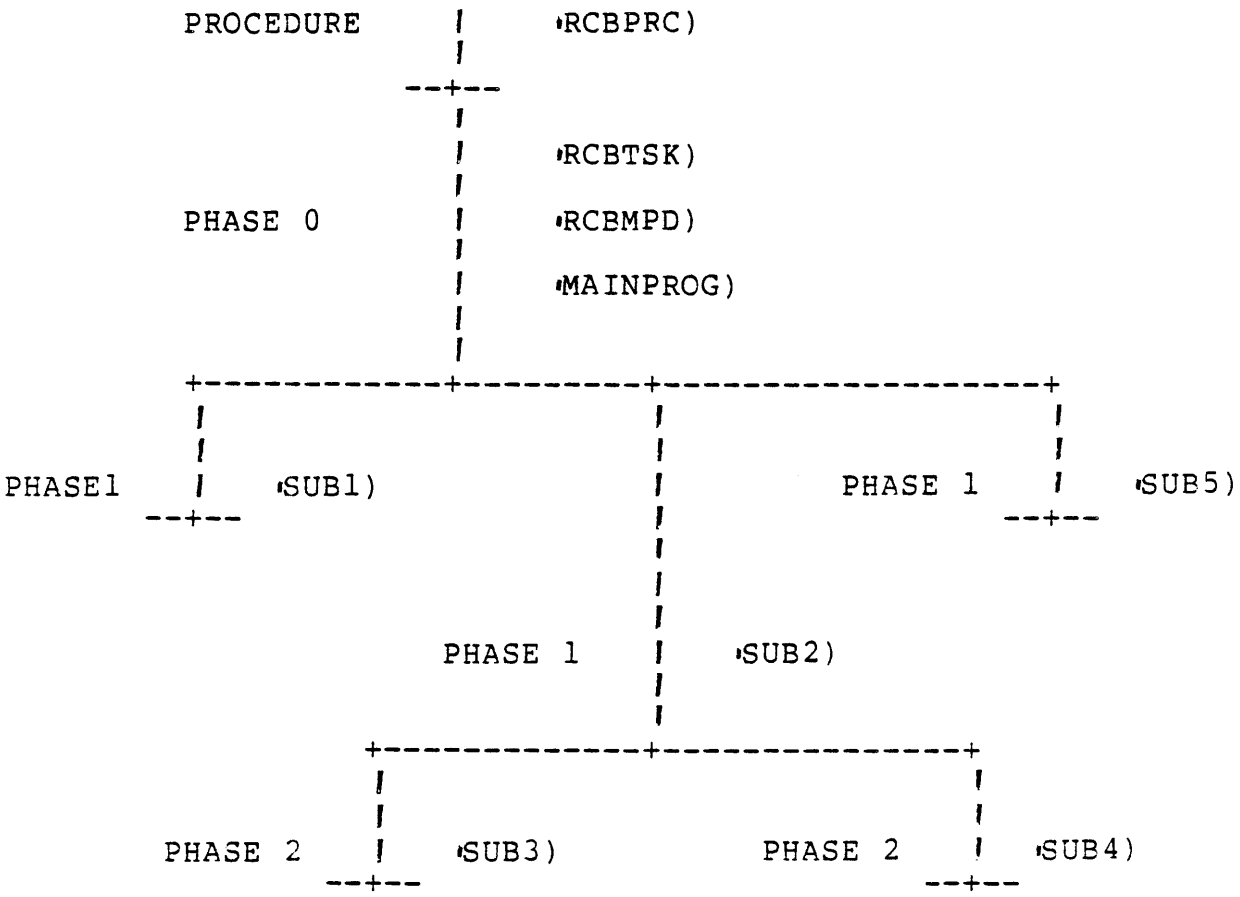
```
[] XLE

   EXECUTE LINK EDITOR
        CONTROL ACCESS NAME: TI.LCF.PARTL
    LINK OUTPUT ACCESS NAME: TI.COBOL.OBJ.PLOBJ
        LISTING ACCESS NAME: TI.LMAP.PARTL
                 PRINT WIDTH: 80
```

```
FORMAT IMAGE, REPLACE
LIBR  .S$SYSLIB
LIBR  TI.COBOL.OBJ
PROC  RCOBOL
DUMMY
INCL   (RDBPRC)
PHASE 0,  STOCK
INCL   (RCBTSK)
LOAD
INCL   (RCBMPD)
INCL   (STOCKO)
PHASE 1,  UPDATE
INCL   (PLOBJ)
PHASE 1,  FILMNG
INCL   (RWFILE)
END
```

Figure 2-8

WORKSHEET

Write a link control file to construct the following overlay structure. Choose your own procedure, task, and overlay names. Parentheses) denote modules to be included.



LAB EXERCISE

1) Create a directory for yourself, if you do not already have one, under TI.COBOL. Allow yourself room for about 20 entries. You may wish to create subdirectories under this directory.

```
[] CFDIR

   CREATE DIRECTORY FILE
                        PATHNAME: TI.COBOL.JONES
                     MAX ENTRIES: 20
   DEFAULT PHYSICAL RECORD SIZE:
```

2) Create a key index file that has a record format that corresponds to the sample data listed below. Use the Create Key Indexed File (CFKEY) command to create the your file. The part number is the primary key and the description is a secondary key. Use the Copy Sequential File to KIF (CSK) command to copy the sample data to your file. The sample data has been stored in TI.DATA.COBOL.PARTSFL.

Part Number Pos.1-5	Description Pos.6-25	Cost Pos.26-30	Quantity Pos.31-35	Reorder Level Pos.36-40
43219	Handle	00500	10090	00400
34583	Wrench	04500	00980	00080
10051	Saw	04000	01650	00200
19768	Hammer	02000	02000	00500
32154	Level	02500	00900	00030
61532	Pliers	01500	02500	00500
34599	Wrench	05000	01700	00100
29984	Screw Driver	01000	04500	00500
82314	Nails	00002	00100	00050

3) You must write a COBOL program that will perform 3 functions:

   a. receive inventory

   b. issue inventory

   c. print a report that lists those items which need to be reordered

You will be provided with subroutines that perform these functions. You need only code a program that will accept a part number and the quantity from the screen and drive the subroutines. You should display a menu of the 3 functions as well as screens similar to these.

```
+-----------------------------------------+
|                                         |
|              STOCK ISSUE                |
|                                         |
|     PART NUMBER _____                   |
|                                         |
|     DESCRIPTION xxxxxxxxxxxxxxxxxxx     |
|                                         |
|     QUANTITY    _____                   |
|                                         |
|   *Parts Issued*   *Parts NOT Issued*)  |
|                                         |
+-----------------------------------------+


+-----------------------------------------+
|                                         |
|          STOCK RECEIPT                  |
|                                         |
|    PART NUMBER _____                    |
|                                         |
|    DESCRIPTION xxxxxxxxxxxxxxxxxxx      |
|                                         |
|    QUANTITY    _____                    |
|                                         |
| *Parts Received*   *Parts NOT Received*)|
|                                         |
+-----------------------------------------+
```

The subroutines, their function, and required parameters are:

SUBROUTINE	FUNCTION	PARAMETERS
RDINV	reads inventory file (must be key index file you created)	PART NUMBER DESCRIPTION STATUS
RECPT	updates file to reflect receipt of inventory	PART NUMBER QUANTITY STATUS
ISSUE	updates file to reflect inventory that has been issued	PART NUMBER QUANTITY STATUS
REORD	generates a report of all items in the inventory that should be reordered	----------
SSTAT	displays the current status of a given part number	PART NUMBER

SSTAT does not have to be called by your program but must be linked since it is called by RECPT an ISSUE. The STATUS parameter indicates whether the operation performed was successful or not. A value of zero will be returned to your program if the operation completed successfully. You must assign the synonym "PIF" to your inventory file and the synonym "RR" to a sequential file or device that will receive the reorder report. Your program should not use the lower half of the screen since this area will be used by the subroutines. The object modules for these subroutines are stored under TI.DATA.COBOL.

4) Once you have tested your program without the subroutines, you must link and execute the COBOL runtime, your program, and the subroutines 3 different ways.

a. The subroutines should be linked so that they are entrant. Test this with someone else. Two or more people should be able to link their task so that they share a common procedure that contains the subroutines.

b. Link using overlays where the subroutines called by your program are each a separate overlay.

c. Link using overlays where 2 levels are required. Hint - RECPT and ISSUE can both call SSTAT into memory.

5) Using the compiler listing and the link map, you should be able to determine the amount of memory required for each of these three methods.

WORKSHEET

This worksheet requires an understanding of the material to be tested in the previous exercise. It is presented here as an aid to completing section 4 of that exercise.

Diagram the three structures that you are going to create with the Link Editor.

MODULE 3

KEY INDEX FILES

OBJECTIVES

* Utilize KIF structure in application programs.
* Specify the advantages and disadvantages of KIF and DBMS.
* Estimate the disk allocation required for a key index file.
* Describe the physical structure of a key index files.

3.1 KEY INDEX FILES

When using key index files (KIF), the user should be aware of the requirements and/or limitations imposed by not only the system but also ANSI standards. KIF logic is an option that is selected at when a system generation is performed. COBOL requires this logic to support indexed file structures. There are some limitations imposed on DX10 KIF when using COBOL due to the standards set forth by ANSI. According to the ANSI X3.23-1974 COBOL standard:

* Duplicate primary keys are not allowed.

* Alternate or secondary keys may not overlap the primary key.

* Secondary keys may overlap each other but they may not overlap the leftmost character.

* All keys that have been defined for a file must be defined in the COBOL program regardless of whether or not they will be used by that application.

Release 3.3.0 and all later releases of DX10 support either sequential or hashed placement of keys in a KIF file. Earlier releases support only hashed placement. Sequential placement is generally faster if the user has loaded the data in sequence. It may become necessary to offload this data and reload it whenever file fragmentation offsets this benefit. This module will use examples implemented on a system with sequential placement. Appendix A contains similar information for an implementation using hashed placement.

With sequential placement in a KIF file, all reads to the file are performed with the read by key supervisor call. This implies that it is not any faster to perform a read by primary key than it is to perform a read using a secondary key. This is true for all KIF files using sequential placement and running under DX10 release 3.3.0 or later.

One consideration that may be important on a system with a restricted amount of memory, is the memory requirements for KIF logic. KIF logic, when included, will require approximately 2K words of additional memory in the operating system plus an additional 180 words for every disk drive that is defined.

It should also be noted here that defining a KIF file causes two entries to be used in the directory in which it was defined. This is due to the amount of overhead that is required to store the file attributes in the directory.

```
DIRECTORY LISTING OF:  TI.COBOL
  MAX # OF ENTRIES: 11       # OF ENTRIES AVAILABLE: 7

DIRECTORY  ALIAS  ENTRIES    LAST UPDATE          CREATION
 CLASS       *       23   03/19/80  09:42:01  03/15/80  12:10:38

FILE       ALIAS  RECORDS    LAST UPDATE       FMT  TYPE   BLK  PROTECT
 ACCT        *       11   03/16/80  10:24:25  BS   N SEQ  YES
 BATCH       *     2027   02/18/80  11:37:14  BS   N SEQ  YES  WRT DEL
 SALES       *      113   03/15/80  14:01:25  BS   N SEQ  YES
12:02:26 THURSDAY, JUN 19, 1980
```

After creating a KIF file the directory appears as:

```
DIRECTORY LISTING OF:  TI.COBOL
  MAX # OF ENTRIES: 11       # OF ENTRIES AVAILABLE 5

DIRECTORY  ALIAS  ENTRIES    LAST UPDATE          CREATION
 CLASS       *       23   03/19/80  09:42:01  03/15/80  12:10:38

FILE       ALIAS  RECORDS    LAST UPDATE       FMT  TYPE   BLK  PROTECT
 ACCT        *       11   03/16/80  10:24:25  BS   N SEQ  YES
 BATCH       *     2027   02/18/80  11:37:14  BS   N SEQ  YES  WRT DEL
 PMFL        *      394   03/19/80  12:04:07  BS   N KEY  YES
 SALES       *      113   03/15/80  14:01:25  BS   N SEQ  YES
12:04:43 THURSDAY, JUN 19,1980
```

3.2 KIF vs. DBMS

There are advantages and disadvantages to using either key index files or a data base management system for the 990 which should be weighed in light of the specific needs at each installation. There are a couple of factors that may strongly influence any decision regarding a change from KIF to DBMS. One, is this a new application or is it tied to other applications, and two, is there a large demand to insert records or portions of a record an unpredictable or variable number of times.

The following points illustrate the advantages and benefits to be derived from implementing a traditional file structure using KIF.

* Sequential Keys -- easy access of data in a specific sequence.

* Multiple Keys -- can define up to 14 different keys and these keys may overlap (except primary key).

* Blank Suppression -- compression of records to conserve the required disk storage.

* Variable Length Records -- can have records that vary in length within the same key index file.

* Utilities -- file utilities to ease maintainence efforts (CSK, CKS, MD).

* ANSI COBOL specifies a standard -- while the actual file may not be transportable, the file structure as implemented in the application program is more nearly transportable on computers whose compilers conform to these standards.

* Ease of Training -- less training required when working with KIF which has come to be known as a conventional file structure.

When implementing a data base management system, there are some distinct advantages to be gained.

* Variable Length Records -- can have records that vary in length within the same DBMS file.

* Ordering of Data -- ordering of data within the record in a manner that is not available with a conventional file.

* Faster Updates -- updating of items is in general, faster than with KIF.

* Hashed Keys -- hashing of keys makes the insertion of records much faster. (No index, table, or balance trees to maintain.)

* Query -- simplifies inquiries to the file and may also reduce the effort of report writing.

* Data Independence -- user need not know the actual data structure to access a given field. Unlike a traditional file, you do not have to know a fields position in relation to the rest of the record.

* Security -- if security is an installed feature of the DBMS, you can go beyond the file level (which is probably the only security available, if any, in a conventional file) and assign security to the lowest data element in the file.

* Logging -- logging may be an optionally installed feature with DBMS which provides an easier means of maintaining an audit trail and ensuring data integrity.

* Limited Redundancy of Data -- it is not necessary to duplicate fields so frequently in different files.

* Utilities -- several utililites are available to make maintenance of the data base an easier job.

* Estimating File Size -- it is far easier to determine the actual amount of disk storage that will be required for a given file. The estimate is also very accurate.

PK

F1 F2 F3 F3 F3 F3 ??

* IS INSERTION A PROBLEM?

* WITH DBMS CAN INSERT A VARIABLE NUMBER OF FIELDS INTO THE RECORD.

* WITH KIF CAN SOLVE PROBLEM WITH AN INDEXING SCHEME BUT IT IS VERY DIFFICULT TO THEN DETERMINE HOW MANY RECORDS TO ACCESS.

Figure 3-1

3.3 ESTIMATING KIF FILE SIZE

When a key index file is created using the sequential placement scheme, the maximum size of the file can be calculated fairly accurately. The user must know the value of a number of parameters before an estimate of the file size can be made. The required parameters are:

* Physical record size

* Average blank-suppressed logical record size

* The size of each key

* ADU size of the disk which will contain the file

* Maximum number of logical records

* Will the input data be sorted when loaded?

The accuracy of the user's estimate is dependent upon the accuracy of these parameters. The most difficult parameter to estimate is the average logical record size. Since KIF blank suppresses all logical records, the user must be able to estimate the actual number of characters that will be stored.

3.3.1 Disk Organization.

KIF allocates disk space to meet three distinct requirements. The first area is used for prelogging. Should an error occur, such as a power failure, the prelog area is used to restore the file to its original state when it is next opened. The second area contains the nodes of the balanced trees or B-Trees. This is the beginning of the indexing structure for each key. The third area contains physical records that are used to store the actual data records and additional B-Trees nodes.

The size of the prelog area may be determined as follows:

$$(18 * K) + 3 = \text{NPR for prelog}$$

where:

K = number of keys
NPR = number of physical records

The space required for B-Trees may be determined as follows:

$$\left\lceil \frac{PRS - 20}{KS + 6} \right\rceil = X$$

$$\left\lceil \frac{\#LR}{X} \right\rceil + SPLIT * \left\lceil \frac{\#LR}{X} \right\rceil = \text{NPR for B-Trees}$$

where:

PRS = physical record size
KS = key size
#LR = maximum number of logical records
SPLIT = 0.1 if input sorted, else 0.25
| | and | | mean round down or up to the nearest integer

This calculation must be performed for each key that has been defined.

The area to be used for data records and additional B-Tree space is determined as follows:

$$\left\lfloor \frac{PRS - 16}{LRS + 6} \right\rfloor = X$$

$$\left\lfloor \frac{\#LR}{\lceil X \rceil} \right\rfloor = \text{NPR for data}$$

where:

LRS = average blank suppressed logical record size (if there is only one key in the file, do not count the key as part of the record)

The total number of physical records is then:

$$\text{NPR data} + \text{NPR prelog} + \sum_{i=1}^{K} \text{NPR B-Tree } i = \text{NPR total}$$

3.4 ADDITIONAL NOTES

As stated earlier there are 18K + 3 physical records at the beginning of the file for logging, where K is the number of keys. Records are written to this area before being updated to prevent a loss of data should an error occur before the update is complete. Based upon the maximum number of records to be loaded, an allocation is made for the nodes of the B-Trees. The remaining space will contain the data. If the file grows beyond its initial size, additional disk allocations are made to contain B-Tree nodes and data.

Within each node of the B-Tree there are a few words of overhead and several pairs of keys and pointers. At higher levels of nodes, the key value indicates the largest key value that resides in the node to which it points in the next lower level. At the lowest level of nodes, the pointer for a key indicates the hash bucket that the logical record is in.

It is permissable to load records for which one or more of the secondary keys has not been given a value. The user may then rewrite the record at a later time and give the key a value by accessing the record using the primary key or a secondary key that has a value.

Keys may be up to 100 characters in length, however it is more efficient in terms of disk storage if the key is not defined as being a large number of characters. Also, when fewer keys are defined, less overhead is required.

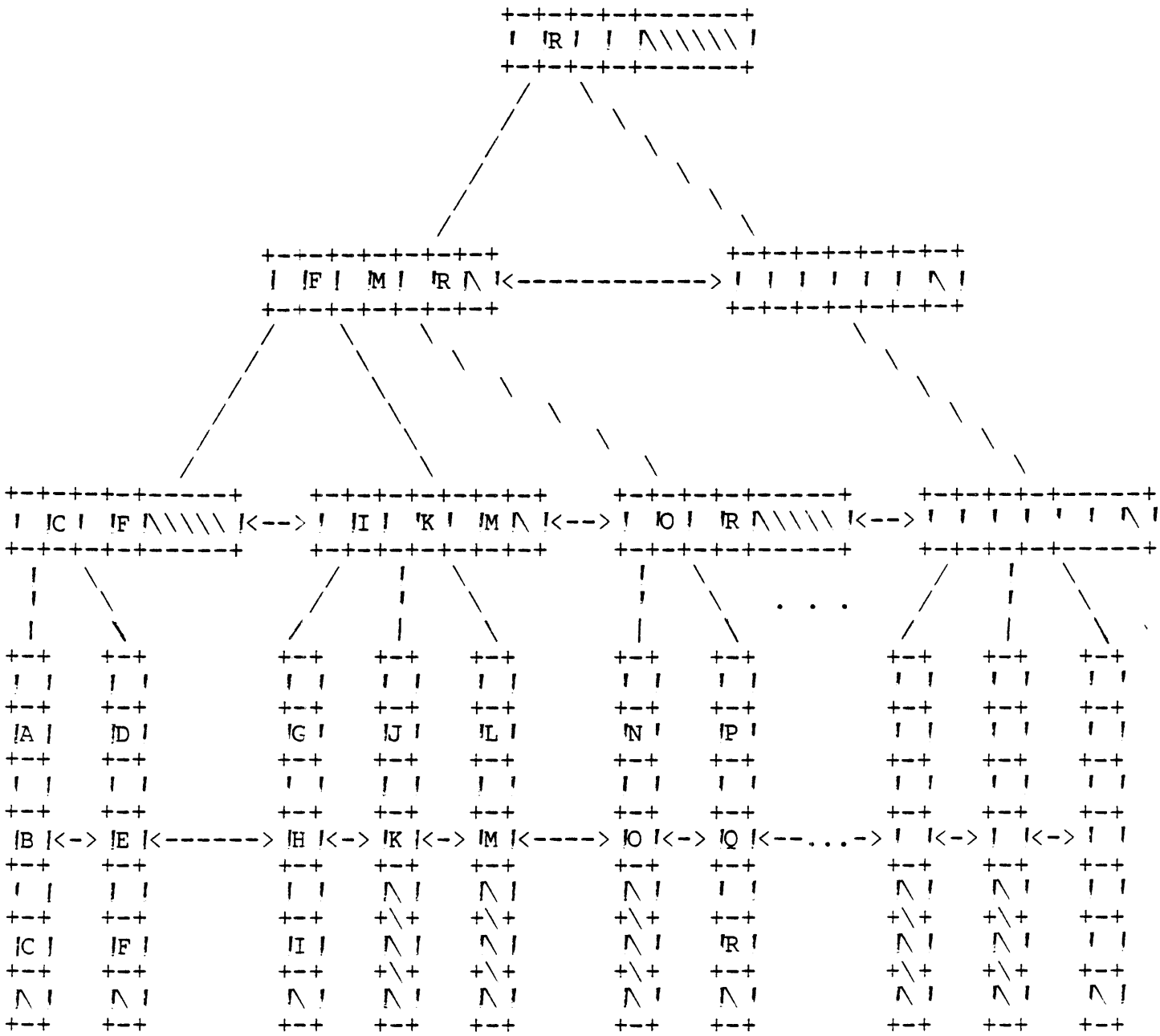


Figure 3-2 Key Index File B-Tree

MODULE 4

SYSTEM COMMAND INTERPRETER

OBJECTIVES

* Read and interpret existing SCI commands.
* Write user-defined SCI commands.
* Utilize SCI primitives to improve performance.

4.1 SYSTEM COMMAND INTERPRETER

The interface between the user and the sytem is a very powerful tool called the System Command Interpreter (SCI). This highly flexible language enables the user to tailor the system functions, capabilities, and resources to the needs of his specific environment. SCI can be executed either in batch mode or from an SCI procedure. SCI consists of SCI primitives, existing SCI commands, and SCI menus. A procedure, or PROC, is an SCI command procedure.

In this section, we discuss how a user can write SCI command procedures unique to an application or modify existing commands. A command procedure is a series of instructions written in the SCI language which define a command to SCI. Since SCI is an interpretive language, SCI interprets one statement at a time.

All SCI statements have the following format:

[blank(s)] <operator> blank(s) [<keyword list>]

where:

operator -- SCI primitive or an existing SCI command.

keyword list -- Required prompts or parameters (must be seperated by commas).

[ ] -- Denotes an optional string.

< > -- Denotes a string that must be supplied by the user.

4.2 KEYWORD LIST

A keyword list has the form:

<Keyword> = [*]<Type> [(<Default Value>)]

where:

Keyword -- Any string that the user supplies for the purpose of prompting the operator. (prompts are displayed when the PROC is executing in foreground.)

* -- Indicates that this is an optional keyword and that the operator need not reply to this prompt.

Type -- Indicates the type of response that is valid for this keyword.

Default Value -- A value that will be displayed as the default response to the keyword.

The following keyword types are valid:

* STRING -- any character string (quotation marks, parentheses, and commas are not permitted) or a character string enclosed in quotation marks (quotation marks are not permitted).

* ACNM -- a file pathname or a device name.

* INT -- a hexadecimal or decimal integer expression.

* YESNO -- any alphabetic string beginning with an Y or an N.

* NAME -- an alphanumeric string which begins with an alphabetic character.

* If no keyword type is given, the keyword will not be displayed.

Whenever you desire to use the reply to a keyword as a parameter in the PROC, the ampersand (&) is required to properly evaluate the value of the keyword. Since synonyms may be used whenever a reply is made to a keyword, this is a common occurence with the type ACNM, the use of the symbol @ allows synonyms to be evaluated. If you wish to evaluate a keyword which may contain a synonym, use the format @&<keyword>.

```
.PROC  TEST (TEST SCI PROCEDURE) = 4,
INPUT PATHNAME = ACNM (@$PATH)
.SYN $PATH = @&INPUT PATHNAME
SF  FILE PATHNAME = @$PATH
.EOP
```

In this example, note the use of the symbols @ and & to evaluate synonyms and keywords. Also note the commands that begin with a period, these are SCI primitives. Inclusion of the Show File (SF) command demonstrates how you may include any existing SCI command in an SCI procedure.

4.3 SCI PRIMITIVES

SCI primitives perform predefined operations. They may be executed in an SCI procedure, in a batch stream, or interactively as an SCI command. All SCI primitives begin with a period.

The following is a list of the SCI primitives and their parameters:

```
PRIMITIVE
 COMMAND                          PARAMETERS

 .PROC     <name> [(<full name>)] [=<int>,] [<keyword list>]
 .EOP
 .DATA     <acnm> [,EXTEND= Y/N] [,SUBSTITUTION= Y/N] [,REPLACE= Y/N]
 .EOD
 .SYN      <name>="<string>"
 .EVAL     <name>=<int expression>
 .BID      TASK=<int/name> [,LUNO=<int>] [,CODE=<int>] [,PARMS=(<string>)]
 .QBID     TASK=<int/name> [,LUNO=<int>] [,CODE=<int>] [,PARMS=(<string>)]
 .DBID     TASK=<int/name> [,LUNO=<int>] [,CODE=<int>] [,PARMS=(<string>)]
 .OVLY     OVLY=<int/name> [,LUNO=<int>] [,CODE=<int>] [,PARMS=(<string>)]
 .SPLIT    LIST=<string>, FIRST=<name> [,REST=<name>]
 .IF       "<string>", <relational operator>, "<string>"
 .ELSE
 .ENDIF
 .LOOP
 .UNTIL    "<string>", <relational operator>, "<string>"
 .WHILE    "<string>", <relational operator>, "<string>"
 .REPEAT
 .EXIT
 .USE      [<acnm>] [<,acnm>]
 .MENU     <menu name>
 .STOP     TEXT = "<string>" [,CODE = <int>]
 .SHOW     <acnm>
 MSG       TEXT = "<string>" [,REPLY = <name>]        (not a primitive)
 .OPTION   [,PROMPT="<string>"] [,MENU="<name>"] [,PRIMITIVES= Y/N]
```

4.4 DEFINE PROCEDURE

.PROC begins the definition of an SCI procedure. It has the following format:

```
.PROC <name> [(<full name>)] [=<integer>] [,<keyword list>]
```

where:

name -- Mnemonic used to call the procedure.

full name -- Full name of the command which will be displayed when the command is called.

integer -- Integer value in the range of 0 to 7 which indicates the privledge level of the command.

4.5 END OF PROCEDURE

.EOP indicates the end of the procedure definition. It has the following format:

```
.EOP
```

.PROC and .EOP are required when defining the PROC interactively or from a batch stream. If the Text Editor is used, they may be omitted.

4.6 ASSIGN SYNONYM

.SYN assigns a value to a synonym. It has the following format:

```
.SYN <name> = "<value>"
```

where:

name -- The synonym name, which may be any character string.

value -- String, character string, variable or a concatentated expression.

This is the only command used in the Assign Synonym (AS) procedure. The @ symbol will allow you to access the value of the synonym by placing it in front of the synonym name.

SYNONYM	VALUE
C	TI.COBOL
S	C.SOURCE
O	.OBJECT

@@S.MAIN is equivalent to TI.COBOL.SOURCE.MAIN

@C@O.MAIN is equivalent to TI.COBOL.OBJECT.MAIN

4.7 CONDITIONAL PRIMITIVES

The primitives .IF, .ELSE, and .ENDIF provide for the conditional execution of an SCI command stream. Their format is as follows:

```
.IF <operand 1>, <relation>, <operand 2>
            'SCI command stream 1)
.ELSE
            'SCI command stream 2)
.ENDIF
```

where:

relation -- EQ, NE, GT, GE, LT, LE

The .ELSE primitive is not required when using .IF however, every .IF must be terminated by a .ENDIF. The .IF statements may be nested up to a maximum of 32 levels deep.

```
* THIS PROCEDURE IS CALLED "EXP"
*
EXP 'EXAMPLE PROC) = 5,                    !PRIVLEDGE LEVEL 5
INPUT PATHNAME = ACNM '@$PATH),
DISPLAY OR PRINT? = STRING 'DISPLAY)
.SYN $PATH = @&INPUT PATHNAME              !ASSIGN SYNONYM $PATH
*
* IS THE FILE TO BE DISPLAYED OR PRINTED?
*
.IF &DISPLAY OR PRINT?, GE, P
   PF  FILE PATHNAME = @$PATH,             !PERFORM PRINT FILE
       ANSI FORMAT = N,
       LISTING DEVICE = LP01,
       DELETE = N,
       NUMBER OF LINES = ""
.ELSE
   SF  FILE PATHNAME = @$PATH              !PERFORM SHOW FILE
.ENDIF
```

The .IF determines whether or not the response to the keyword DISPLAY OR PRINT? is greater than or equal to P. If it is, the the file will be printed, else the file will be displayed.

Note that comments may be included by placing an asterisk in the first position. Comments may also be placed on a command line by using an exclamation point preceeded by one or more blanks.

Note also that the synonym $PATH begins with a dollar sign. If a synonym has no value, as will be the case when this PROC is first executed, the synonym itself will be displayed. The use of the dollar sign will prevent this from happening.

The operator keys in:

```
[] EXP
```

and the following appears:

```
EXAMPLE PROC
        INPUT PATHNAME:
    DISPLAY OR PRINT?: DISPLAY
```

4.8 WRITING MESSAGES

The command MSG may be used to communicate with the operator from an SCI procedure. While the MSG command is not an SCI primitive, it is very useful when defining a PROC. This command may also be used to accept a response to the message. It has the format:

```
MSG TEXT = "<string>" [,REPLY = <name>]
```

where:

string -- Any character string that is to be displayed.

name -- A synonym that will be assigned to the response if a reply was requested.

When the MSG command is executed, the message will be displayed at the bottom of the screen. The procedure will then halt execution until the operator responds to the message. (Striking the RETURN key will cause the procedure to continue.)

4.9 EVALUATING NUMERIC EXPRESSIONS

The primitive .EVAL may be used to evaluate a numeric expression. The results of the expression will be converted to a decimal, ASCII string and stored as the value of a synonym. The format is:

```
.EVAL <name> = "<value>"
```

where:

name -- The synonym to which the value of the expression is assigned.

value -- The numeric expression.

```
* *
*  EXAMPLE SCI PROC USING ".EVAL" AND "MSG"
* *
.PROC FP (FIND PRODUCT) = 1,
ENTER MULTIPLICAND = INT,
ENTER MULTIPLIER = INT
.USE
.SYN A = &ENTER MULTIPLICAND
.SYN B = &ENTER MULTIPLIER
.SYN HEX = ""
.IF @A, GE, >                    !
   .SYN HEX = YES                !
.ENDIF                           !CHECK TO SEE IF EITHER OF
.IF @B, GE, >                    !THE NUMBERS ENTERED WAS A
   .SYN HEX = YES                !HEXADECIMAL VALUE
.ENDIF                           !
.EVAL P = @A * @B
.IF @HEX, EQ, YES
   MSG T = "HEX NUMBERS CONVERTED TO DECIMAL"
.ENDIF
MSG TEXT = "THE PRODUCT IS @P"
.SYN A = ""
.SYN B = ""
.EOP
```

The command FP accepts two integer values and calculates the product. If either of the values entered is a hexadecimal number then the message HEX NUMBERS CONVERTED TO DECIMAL is displayed. Remember, .EVAL evaluates the expression and stores it as ASCII decimal.

Notice that the synonym is evaluated within the message text. Also, the synonyms A and B were assigned null values at the end of the procedure so that synonym table overflow may be avoided.

WORKSHEET

Write a procedure that will execute a program which updates an inventory file. You should prompt the operator to determine which operation is to be executed. The options which can be selected are a receipt, an issue, or the printing of a reorder report. Assume that there are 3 separate programs to perform each of these operations. The task names are RECPT, ISSUE, and REORD. Assume all the programs are stored under TI.INV.OBJ.

If the reorder option is selected, you should perform a print file on TI.INV.REORD and display a completion message when the printing is complete.

You should validate the option selected and display an error message where appropriate.

The keywords for the Execute COBOL Program Foreground (XCPF) command are:

```
     OBJECT ACCESS NAME:
              DEBUG MODE: NO
    MESSAGE ACCESS NAME:
                SWITCHES: 00000000
          FUNCTION KEYS: NO
```

The keywords for the Print File (PF) command are:

```
          FILE PATHNAME(S):
              ANSI FORMAT?: NO
            LISTING DEVICE:
    DELETE AFTER PRINTING?: NO
      NUMBER OF LINES/PAGE:
```

4.10 ITERATIVE LOOPS

The primitives .LOOP, .UNTIL, .WHILE, and .REPEAT may be used to create a loop within the procedure. The format for their use is:

```
.LOOP

.UNTIL "<operand 1>", <relation>, "<operand 2>"

.WHILE "<operand 1>", <relation>, "<operand 2>"

.REPEAT
```

where:

relation -- EQ, NE, GT, GE, LT, LE

The loop must be initiated by a .LOOP statement and terminated by a .REPEAT statement. The loop must contain at least one .WHILE or .UNTIL but may contain more than one. The statements between the .LOOP and .REPEAT will be continuously executed as long as the condition in the .WHILE is true or they will be executed until the condition in a .UNTIL becomes true. As soon as the loop is terminated by one of the conditions, the statement following the .REPEAT will be executed. If either of the conditional statements causes the loop to terminate, the branch out of the loop is immediate, that is, no statements between the conditional and .REPEAT will be executed.

```
.LOOP
   .
   .       SCI statemetns
   .
.UNTIL  or  .WHILE
   .
   .       SCI statements
   .
.REPEAT
```

```
* *
*  EXAMPLE SCI PROC USING ".EVAL", "MSG", AND ".LOOP"
* *
.PROC FP (FIND PRODUCT) = 1,
ENTER MULTIPLICAND = INT,
ENTER MULTIPLIER = INT
.USE
.SYN A = &ENTER MULTIPLICAND
.SYN B = &ENTER MULTIPLIER
.SYN HEX = "", AGAIN = ""
.IF @A, GE, >                    !
   .SYN HEX = YES                !
.ENDIF                           !CHECK TO SEE IF EITHER OF
.IF @B, GE, >                    !THE NUMBERS ENTERED WAS A
   .SYN HEX = YES                !HEXADECIMAL VALUE
.ENDIF                           !
.EVAL P = @A * @B
.IF @HEX, EQ, YES
   MSG T = "HEX NUMBERS CONVERTED TO DECIMAL"
.ENDIF
MSG TEXT = "THE PRODUCT IS @P"
* *
*  COMPUTE THE SQUARE OF THE PRODUCT
*  IF REQUESTED, CONTINUE TO COMPUTE THE SQUARE
* *
.LOOP
   MSG TEXT = "FIND THE SQUARE? (Y/N)", REPLY = AGAIN
.WHILE @AGAIN, GE, Y
   .EVAL S = @P * @P
   MSG TEXT = "THE SQUARE OF @P IS @S"
   .EVAL P = @S
.REPEAT
.SYN A="", B=""
.SYN HEX="", AGAIN=""
.EOP
```

This extension to the previous example shows a method for using a loop. As long as the operator responds in the affirmative to the message FIND THE SQUARE?, the procedure will continue to perform that calculation. The last .EVAL sets the synonym P equal to the synonym S. The same operation could have been performed using a .SYN statement.

4.11 EXIT FROM A PROCEDURE

The .EXIT statement allows you to terminate the execution of a command procedure. If a .EXIT is executed, the PROC is terminated immediately; no other commands will be executed. It has the format:

```
.EXIT
```

```
EXP (EXAMPLE PROC) = 5,          ! PRIVLEDGE LEVEL 5
INPUT PATHNAME = ACNM (@$PATH),
   .
   .
   .
.IF @DP, NE, >3C
   .
   .
   .
.EXIT
.ELSE
   .
   .
   .
.ENDIF
```

4.12 DISPLAYING A FILE

The primitive .SHOW can be used to display the contents of a specified file. It has the format:

```
.SHOW <acnm>
```

where:

acnm -- A valid pathname.

```
SF (SHOW FILE),
FILE PATHNAME = *ACNM (@$SF$P)
.SYN  $SF$P = "&FILE PATHNAME"
.IF  "&FILE PATHNAME", NE, ""
.SHOW @&FILE PATHNAME
.ENDIF
```

4.13 TERMINATING SCI

The primitive .STOP will cause execution of SCI to be terminated. It has the following format:

```
.STOP [TEXT = "<string>" [,CODE = <int>]]
```

where:

TEXT -- A string which may be passed back to the foreground terminal local file in place of the normal batch stream message.

CODE -- May be used to set the synonym $$BC.

The .STOP command may be used in any application of SCI. However, the parameters TEXT and CODE, only have meaning when this primitive is used in a batch stream. Batch streams are discussed in Module 6.

4.14 SPECIFYING AN SCI PROCEDURE LIBRARY

SCI normally searches the system procedure library, .S$PROC, whenever it is given a command. The user may specify that an alternate directory or library be used to find the command. The primitive .USE may be used for this purpose. It has the following format:

```
.USE [<acnm>] [,<acnm>]
```

where:

acnm -- The valid pathname of a directory to be used as the SCI procedure library. The default is .S$PROC.

This command is especially useful in the application environment. It enables the user to place commands in different libraries and still use them within the same command procedure. This may also be used to make the system more secure from unauthorized use of a command.

If two directories are specified, SCI will search the first directory for the command to be executed. If it is not found, the second directory will then be searched. This feature would allow placing user-defined PROCs for a given application in one directory while allowing the use of system PROCs within these commands.

```
.USE TI.GNLEDGER, .S$PROC
```

4.15 BUILDING A DATA FILE

Data may be copied directly into a file through the use of the .DATA and .EOD primitives. All statements between these two commands will be placed in a specified file. The format is:

```
.DATA <pathname> [,EXTEND = <YES/NO>]
                 [,SUBSTITUTION = <YES/NO>]
                 [,REPLACE = <YES/NO>]
  .
  .
  .
.EOD
```

where:

pathname -- A valid pathname that indicates where the file is to be stored.

EXTEND -- This indicates whether or not the file should be opened extended. This allows you to concatenate several files together or to append additional records to the end of a file. The default is NO.

SUBSTITUTION -- This allows the user to use synonyms and keywords in the text which will be evaluated and the values substituted before being written to the file. The default is NO.

REPLACE -- Specifies whether the data stream is to replace an existing file. The default is YES.

This is a useful command for building files from within the procedure. This is especially useful when you wish to build a temporary file, add to, or substitute in an existing file.

```
EMPINFO 'QUERY ON EMPLOYEE INFO) = 4,
PASSWORD = STRING,
EMPLOYEE NUMBER = INT
.IF @$$ST, LT, 06
   MSG TEXT = "PRIVLEDGED COMMAND, FOR ACCOUNTING ONLY"
   .EXIT
.ENDIF
.IF @$$ST, GT, 08
   MSG TEXT = "PRIVLEDGED COMMAND, FOR ACCOUNTING ONLY"
   .EXIT
.ENDIF
* *
*  APPEND QUALIFICATION TO QUERY STATEMENT
* *
.DATA  TI.EMP.QEISRC, EXTEND=YES, SUBSTITUTION=YES
   WHERE EMPN EQ &EMPLOYEE NUMBER
   BY KEY BY LIST
.EOD
* *
*  BID THE QUERY TASK
* *
.BID TASK=>C0, LUNO=>10, PARMS='3, 4, @$MR$, &PASSWORD, 60,
     80, .LIST@$$ST, R, N, N, N, N, N, N, TI.QEI@$$ST,
     .LIST@$$ST, .TEMPQ@$$ST, .QUERYLIB.ERRMSG, , @$MT$)
* *
*  SHOW THE QUERY OUTPUT
* *
.SHOW .LIST@$$ST
* *
*  DELETE TEMPORARY LISTING AND WORK FILES
* *
DF  PATHNAME = .LIST@$$ST
DF  PATHNAME = .TEMPQ@$$ST
```

This procedure accepts a password and an employee number that will be used by the QUERY 990 processor. A query file already exists which will cause information to be retrieved from a DBMS employee file. A statement that indicates which employee is desired must be appended to the end of the query file. Note, this PROC will only execute from stations 6, 7, and 8. The synonym $$ST, which contains the station number, has been concatenated with the file names to give unique temporary files.

WORKSHEET

Write a PROC that will accept a task name from the screen. That name should then be used as the file name for source, object and listing files when compiling the program. It will also be the task name in the link control file. Build the link control file in your procedure. Use the synonym $$LU to retrieve the luno from the AL (assign luno) command. Use $$LU where required. Your PROC should then compile, link, and execute the task.

The format for the Execute COBOL Compiler Foreground (XCCF) command is:

```
      SOURCE ACCESS NAME:
      OBJECT ACCESS NAME:
     LISTING ACCESS NAME:
                 OPTIONS:
             PRINT WIDTH: 80
               PAGE SIZE: 55
PROGRAM SIZE (LINES): 1000
```

The format for the Execute Linkage Editor (XLE) command is:

```
         CONTROL ACCESS NAME:
LINKED OUPUT ACCESS NAME:
         LISTING ACCESS NAME:
                 PRINT WIDTH: 80
```

The format of the Execute COBOL Task Foreground (XCTF) command is:

```
       PROGRAM FILE LUNO:
         TASK ID OR NAME:
              DEBUG MODE: NO
MESSAGE ACCESS NAME:
                SWITCHES: 0000000
          FUNCTION KEYS: NO
```

The format of the Assign LUNO (AL) command is:

```
             LUNO:
      ACCESS NAME:
PROGRAM FILE?: NO
```

The format of the Release LUNO (RL) command is:

```
LUNO:
```

A suggested sequence of events is as follows:

1. Prompt for the task name -- command name and keywords.
2. Compile the program -- XCC or XCCF.
3. Build link control file -- .DATA through .EOD.
4. Execute the link editor -- XLE.
5. Assign a luno to your program file -- AL.
6. Execute the task -- XCT or XCTF.
7. Release the luno -- RL.

If any commands are executed in background, such as the Link Editor, a WAIT command should be executed after that command. This will prevent the next foregound command from being executed before a required background task has terminated.

4.16 THE .SPLIT PRIMITIVE

The .SPLIT primitive assigns the first term from a value list to a synonym and assigns the rest of the list to another synonym. It has the following format:

```
.SPLIT LIST = <list>, FIRST = <name 1> [,REST = <name 2>]
```

where:

LIST -- Defines the value list.

FIRST -- Defines a synonym given to first term.

REST -- Defines a synonym given to the remainder of the value list.

```
ADST 'APPEND DAILY SALES TOTALS) = 2,
DAILY SALES FILE 'S) = *NAME '@$DSF),
OUTPUT FILE = NAME
.SYN $DSF = &DAILY SALES FILE
.SYN OF = &OUTPUT FILE
.SPLIT LIST = @$DSF,
       FIRST = DAY,
       REST = $DSF
.SHOW TI.SALES.@DAY
MSG TEXT = "DOES TOTAL BALANCE WITH SUMMARY SHEET? 'Y/N)",
    REPLY = BAL
.IF @BAL, EQ, Y
   AF  INPUT ACCESS NAME = TI.SALES.@DAY,
       OUTPUT PATHNAME = TI.SALES.@OF
.ENDIF
.IF @$DSF, NE, ""
   ADST  OUTPUT FILE = @OF
.ENDIF
.SYN $DSF="", OF="", DAY="", BAL=""
```

This example accepts the names of several files that will be displayed one at a time. If the operator responds in the affirmative to the message DOES TOTAL BALANCE WITH SUMMARY SHEET?, the file currently displayed will be appended to a common file. The PROC then calls itself to display and append the next file.

Instead of calling itself again, the PROC could also have been written to use the .LOOP primitive. A third method of writting this PROC allows the user to key in all of the file names at the same time. Each file name is seperated by a comma as in the first method, however, the keyword type is ACNM. This gives the added

advantage of allowing SCI to determine that a valid file name was entered. Commas are not usually permitted with type ACNM, however, they are permitted as delimiters if the keyword type has been enclosed in parentheses. The Copy/Concatenate (CC) command is an example of this usage.

```
CC (COPY/CONCATENATE),
INPUT ACCESS NAME (S)=(ACNM),
OUTPUT ACCESS NAME=ACNM,
REPLACE?=YESNO (NO),
MAXIMUM RECORD LENGTH = *INT
.BID   TASK=>34,CODE=2,
PARMS=(('&INPUT ACCESS NAME),
@&OUTPUT ACCESS NAME,
NO,&REPLACE,NO,&MAXIMUM RECORD LENGTH)
```

4.17 BIDDING A TASK OR AN OVERLAY

Most of the existing SCI commands as well as many that the user will write himself, cause a task or an overlay to be bid from a program file and executed. The primitives .BID, .QBID, .DBID, and .OVLY may be used for this purpose. When used in place of an existing SCI command they will cause execution to be considerably faster since the SCI command does not have to be evaluated. .BID bids a task to be executed in foreground and .QBID will cause the task to be executed in background. .DBID bids a task and then places it in a suspended state (state >6) for the purpose of debugging it, e.g. setting breakpoints etc. As such, this is not a suitable command for use with COBOL applications. The command .OVLY is used by many of the existing SCI commands since they are disk resident features of the operating system. A good example of this is the text editor (XE), which is installed as overlay >3 (E$EDIT) on the system program file.

The format for these primitives is:

```
.BID TASK=<int/name> [,LUNO=<int>] [,CODE=<int>]
     [,PARMS=<(string)>]

.QBID TASK=<int/name> [,LUNO=<int>] [,CODE=<int>]
      [,PARMS=<(string)>]

.DBID TASK=<int/name> [,LUNO=<int>] [,CODE=<int>]
      [,PARMS=<(string)>]

.OVLY OVLY=<int/name> [,LUNO=<int>] [,CODE=<int>]
      [,PARMS=<(string)>]
```

where:

TASK -- A hexadecimal number for the task or overlay id or

the name of that task or overlay.

LUNO -- A hexedicimal number that is the luno assigned to the program file from which this task or overlay is being bid. If this parameter is omitted, the system program file, .S$PROGA, is assumed. This program file should be used only for software supplied by Texas Instruments.

CODE -- A value in the range of 0 to 255 that may be accessed as a binary value by the task or overlay.

PARMS -- A string of characters or parameters, seperated by commas, that may be accessed by the task or overlay.

```
UPINV = 5
* *
*  ASSIGN A LUNO TO THE PROGRAM FILE
* *
.OVLY  OVLY=>1B, LUNO=0, PARMS='6,0,TI.COBOL.PROGF,Y,$AL$L,Y)
* *
*  EXECUTE COBOL TASK IN FOREGROUND
* *
.BID  TASK=>02, LUNO=@$AL$L, PARMS=',N,DUMY,00000000,N)
* *
*  RELEASE THE LUNO
* *
.OVLY  OVLY=>1B, LUNO=0, PARMS='30,@$AL$L)
```

This example causes the task on the program file TI.COBOL.PROGF, whose id is >2, to be executed. First a luno must be assigned to that program file and then the task is bid to execute in the foreground. When the task is terminated, the assigned luno will be released. The synonym ALL was used instead of $$LU. This is the synonym normally assigned by this processor.

Notice that .PROC and .EOP where not used. They are not required unless the PROC is being defined interactively. Also, since no keywords were defined, the operator will key in the command and then the procedure will begin execution.

WORKSHEET

Modify the procedure of the previous worksheet so that all of the functions are implemented using primitives. It is always a good practice to include comments in your procedure, however, this becomes a real necessity when using primitives to bid a task or an overlay.

The procedures XCC and XCCF are written as:

```
XCC 'EXECUTE COBOL COMPILER <VERSION 3.2.0 79173>) =2,
SOURCE ACCESS NAME = ACNM '"@$$S"),
OBJECT ACCESS NAME = ACNM '"@$XCC$OB"),
LISTING ACCESS NAME = ACNM '"@$XCC$L"),
OPTIONS = *STRING '"@$XCC$O"),
PRINT WIDTH = INT '80),
PAGE SIZE = INT '55),
PROGRAM SIZE 'LINES) = INT '1000)
      .
      .
      .                              ! '.BID should be used for XCCF)
.QBID TASK = >87, LUNO = >10,
PARMS = '@&SOURCE ACCESS NAME, @&OBJECT ACCESS NAME,
@&LISTING ACCESS NAME, "&OPTIONS",&PRINT WIDTH,
&PAGE SIZE,@MEMORY,&PROGRAM SIZE 'LINES))
      .
      .
      .
```

The procedures AL and RL are written as:

```
AL 'ASSIGN LUNO),                          RL 'RELEASE LUNO),
LUNO=*INT,                                 LUNO=INT
ACCESS NAME=ACNM,                          .OVLY  OVLY=>1B,LUNO=0,
PROGRAM FILE?=YESNO 'NO)                   PARMS= '30,$LUNO)
.IF    &LUNO, EQ, ""
.OVLY  OVLY=>1B,LUNO=0,
PARMS= '6,0,@&ACCESS NAME,Y,$AL$L,&PROGRAM FILE?)
.SYN   $$LU="@$AL$L"
.ELSE
.OVLY  OVLY=>1B,LUNO=0,
PARMS= '6,&LUNO,@&ACCESS NAME,N,$AL$L,&PROGRAM FILE?)
.ENDIF
.SYN   $AL$L=""
```

The procedure XLE is written as:

```
XLE  'EXECUTE LINKAGE EDITOR) =2,
CONTROL ACCESS NAME = ACNM '"@$XLE$C"),
LINKED OUTPUT ACCESS NAME = *ACNM '"@$XLE$OB"),
LISTING ACCESS NAME = *ACNM '"@$XLE$L"),
PRINT WIDTH = INT '80)
      .
      .
      .
.QBID TASK = >86, LUNO = >10,
PARMS = '@&CONTROL ACCESS NAME,
@&LINKED OUTPUT ACCESS NAME,
@&LISTING ACCESS NAME,
4096,&PRINT WIDTH)
      .
      .
      .
```

The procedures XCT and XCTF are written as:

```
XCT  'EXECUTE COBOL TASK <VERSION: 3.2.0 79173>),
PROGRAM FILE LUNO = INT '"@$XCT$P"),
TASK ID OR NAME = STRING '"@$XCT$T"),
DEBUG MODE = YESNO 'NO),
MESSAGE ACCESS NAME = *ACNM '"@$XCT$L"),
SWITCHES = *STRING '"00000000"),
FUNCTION KEYS = YESNO 'NO)
       .
       .
       .                    ! '.BID should be used for XCTF)
.QBID  TASK = &TASK ID OR NAME, LUNO = "&PROGRAM FILE LUNO",
PARMS = ',"&DEBUG MODE",@&MESSAGE ACCESS NAME,
"&SWITCHES",&FUNCTION KEYS)
       .
       .
       .
```

4.18 MODIFYING THE SCI INTERFACE

The .OPTION primitive allows the user to modify some of the characteristics of the SCI interface. The user may change the command prompt, the main menu, and disallow the use of primitives. In a user environment these features are useful for customizing a system as well as helping to make the system more secure from unauthorized use.

The format of the .OPTION primitive is:

```
.OPTION [,PROMPT=<"string">] [,MENU=<"name">]
        [,PRIMITIVES=Y/N]
```

where:

PROMPT -- An alternate command prompt may be supplied. This string of characters, which must be less than 50, is displayed in place of the default prompt whenever the terminal is in command mode. On a 911 VDT, the prompt appears on the lower left part of the screen.

MENU -- A menu name may be entered here. This menu is displayed as the main menu. The default main menu is stored in .S$PROC.M$LC. The user may supply a different main menu without using this primitive if the file name is M$LC. The next paragraph in this module explains the use of menus.

PRIMITIVES -- The use of primitives may be disallowed by specifying NO to this option. This will prevent the use of primitives by users not authorized to exercise their functions. Primitives may be disallowed from the primary level only, that is from a batch stream or interactively while SCI is in the command or menu display cycle. They may still be used from a PROC. The default is YES.

The following example demonstrates the use of the prompt and menu options. This segment of code might be installed in the PROC M$00 so that when a specific user logs on to the system a new command prompt and main menu will be established for that user. The use of M$00 is discussed in Module 5.

```
M$00
   .
   .
   .
.IF @$$UI, EQ, BCM013
.OPTION PROMPT = "Command? ", MENU = YRMENU
.ENDIF
   .
   .
   .
```

When user BCM013 logs on the system the text Command? is displayed on the terminal and the contents of the file M$YRMENU from a specified procedure directory is displayed as the main menu.

The next example demonstrates the use of the menu and primitives options to control these features at terminals ST09 and greater.

```
   .
   .
   .
.IF @$$ST, GE, 09
.USE TI.GNLEDGER.PROC, TI.PAYROLL.PROC
.OPTION MENU = USER, PRIMITIVES = NO
.ENDIF
   .
   .
   .
```

The main menu and all accessible PROCs must be stored in one or the other of the two directories specified by the .USE command. This and disallowing the use of primitives from these stations will enhance security on the system.

4.19 DISPLAYING A MENU

You can cause menus to be displayed that contain either commands to perform some function or still other menu names. The .MENU primitive will cause the specified menu to be displayed the next time SCI is in the menu display cycle, i.e., just before it prompts for the next command. You may also cause menus to be displayed by using a slash */) in front of the menu name. All menus must be in the procedure directory and must be a file name of the format M$name where "name" is the menu name. "dddd" in the next example represents the directory name. Recall that the default procedure directory is .S$PROC unless changed by a .USE command.

```
MENUS
.MENU JAPPL


dddd.M$JAPPL


   *** JOE'S PART SHOP ***

      APPLICATION MENUS


   /GEN  -- GENERAL LEDGER APPLICATIONS
   /PAY  -- PAYROLL APPLICATIONS
   /INV  -- INVENTORY APPLICATIONS
   /MISC -- MISCELLANEOUS UTILITIES


dddd.M$PAY


   SELECT ONE OF THE FOLLOWING PAYROLL APPLICATIONS


      /HOUR -- HOURLY PAYROLL REPORTS
      /SAL  -- SALARIED PAYROLL REPORTS
      /EOM  -- MONTHLY PAYROLL REPORTS
      /QURT -- QUARTERLY PAYROLL REPORTS
      /YEAR -- YEARLY PAYROLL REPORTS


dddd.M$HOUR


   HOURLY PAYROLL REPORTS


      /GER    -- GROSS EARNINGS REPORT
      /DREG   -- DEDUCTION REGISTER
      /PFM    -- PAYROLL FILES MAINTENANCE
      /CHECKS -- PRINT PAYROLL CHECKS
      /CKREG  -- CHECK REGISTER
      /SSR    -- FICA SUMMARY
      /CR     -- CREDIT UNION TRANSFERS
```

```
dddd.M$DREG


   DEDUCTION REGISTER


      DRTOT -- DEDUCTION REGISTER TOTALS
      DRUP  -- UPDATE MASTER FILE W/REGISTER CALCULATIONS
      PRDR  -- PRINT DEDUCTION REGISTER


dddd.PRDR

PRDR (PRINT DEDUCTION REGISTER) = 3,
OUTPUT DEVICE = ACNM (LP02)
    .
    .
    .
```

LAB EXERCISE

Write a procedure that will accept a number between -5 and +5, compute the cube of that number, and then display the result. You must test for the upper and lower bounds of the allowable range. If the boundaries are violated, display an error message and accept a new value. The PROC may not terminate until a value within the allowable range has been entered.

LAB EXERCISE

Write a procedure that will print multiple copies of a given file and optionally delete that file when the printing is complete. You will have to prompt for the pathname of the file, the number of copies, and whether or not to delete the file.

LAB EXERCISE

Write a procedure that may be used to save a file while that file is being text edited. If this is an existing file, then store the new version using the existing file name. If this is a new file, prompt for the file pathname and then store the file. Your procedure should then reenter the text editor for that file. The commands XE, QE, and QE$1 may be used to accomplish this.

The procedure for the Text Editor (XE) command is:

```
XE (INITIATE TEXT EDITOR),
FILE ACCESS NAME = *ACNM (@$XE$)
        .
        .
        .
.OVLY   OVLY=3,  LUNO=0, CODE=0,
        PARMS= (@&FILE ACCESS NAME,@$MR$,@$MRM$,@$XEM$,@$MT$)
```

The procedure for the Quit Edit (QE) command is:

```
QE (QUIT EDIT),
ABORT? = YESNO (NO)
.IF  "&ABORT", LT, "Y"
QE$1
.ELSE
.OVLY  OVLY=4,CODE = 9,PARMS= (Y,,,)
.ENDIF
.SYN  $XE$S="",  $XE$SC="",  $XE$EC="",  $XEM$=""
```

The procedure for the QE$1 command is written:

```
QE$1 (QUIT EDIT),
OUTPUT FILE ACCESS NAME=ACNM (@$XE$),
REPLACE?=YESNO (NO),
MOD LIST ACCESS NAME=*ACNM
.OVLY OVLY=4, LUNO=0, CODE=9,
      PARMS= (N,"@&OUTPUT FILE ACCESS NAME",&REPLACE,
         "@&MOD LIST ACCESS NAME")
.SYN  $XE$ = "&OUTPUT FILE ACCESS NAME"
.SYN  $$S =  "&OUTPUT FILE ACCESS NAME"
```

MODULE 5

SYSTEM CUSTOMIZATION

OBJECTIVES

* Modify existing SCI commands.
* Permanently modify the status of terminals.
* Utilize the news file, login and logoff, main menu, and completion code features of SCI.

5.1 MODIFYING EXISTING SCI

Many of the existing SCI commands display default parameters to be used in the execution of the procedure. Once you understand how the procedure has been written, it is usually not very difficult to modify the procedure to fit the needs of the user's installation. An example of an existing command that the user may wish to modify is the Initialize System (IS) command. The format of the IS command is:

```
IS (INITIALIZE THE SYSTEM) = 4
IDT
.IF @$$CC,NE,0
.EXIT
.ENDIF
ISL
.BID TASK=3,PARMS=(P,  01 )
.BID TASK=3,PARMS=(P,  02 )
.BID TASK=3,PARMS=(P,  03 )
.BID TASK=3,PARMS=(P,  04 )
.BID TASK=3,PARMS=(P,  09 )
.BID TASK=3,PARMS=(P,  0A )
.BID TASK=3,PARMS=(P,  0B )
.BID TASK=3,PARMS=(P,  0D )
.BID TASK=3,PARMS=(P,  0E )
.BID TASK=3,PARMS=(P,  0F )
.BID TASK=3,PARMS=(P,  010)
.OVLY  OVLY=>23, CODE=4, PARMS=("WARMSTART PROCEDURE COMPLETE",)
```

You may wish to modify this command to perform a number of chores for you such as installing certain volumes at the beginning of the day, modifying the terminal status of each station, or assigning certain lunos that are required by application programs executing in the system.

The following example will cause a TTY device, such as a Model 820 KSR, to be assigned a priveledge level of three with no login required. Three 911 VDTs are to require login, with a maximum privilege level of 4 on two of the VDTs and 7 on the third. A global luno has been assigned to a user's program file that is used in the execution of the application tasks. The luno has been delete protected to ensure that it is assigned throughout the day. Since the same volumes are used every day for most processing, these volumes will be installed at this time as well.

```
IS 'INITIALIZE THE SYSTEM) = 4,
INSTALL DEFAULT VOLUME? = *YESNO 'YES)
IDT
.IF @$$CC,NE,0
.EXIT
.ENDIF
ISL
*
*===[INSTALL THE DEFAULT VOLUMES]==================================
*
.IF "&INSTALL DEFAULT VOLUMES?", GE, "Y"
IV  UN="DS02", VN="TI"
IV  UN="DS03", VN="APPL"
.ENDIF
*
*===[INITIALIZE THE TERMINALS]=====================================
*
MTS TN="ST01", NS="ON", NM="TTY", LR="NO", UPC="3", DM="TTY"
MTS TN="ST02", NS="ON", NM="VDT", LR="YES", UPC="4"
MTS TN="ST03", NS="ON", NM="VDT", LR="YES", UPC="4"
MTS TN="ST04", NS="ON", NM="VDT", LR="YES", UPC="7"
*
*====[ASSIGN GLOBAL LUNO TO APPLICATION PROGRAM FILE AND PROTECT]===
*
AGL LUNO=>99, AN="APPL.PROGF", PF="YES"
MLP PROTECT="P", LUNO=>99
*
.BID TASK=3,PARMS='P,  01 )
.BID TASK=3,PARMS='P,  02 )
.BID TASK=3,PARMS='P,  03 )
.BID TASK=3,PARMS='P,  04 )
.BID TASK=3,PARMS='P,  09 )
.BID TASK=3,PARMS='P,  0A )
.BID TASK=3,PARMS='P,  0B )
.BID TASK=3,PARMS='P,  0D )
.BID TASK=3,PARMS='P,  0E )
.BID TASK=3,PARMS='P,  0F )
.BID TASK=3,PARMS='P,  010)
.OVLY OVLY=>23, CODE=4, PARMS='"WARMSTART PROCEDURE COMPLETE",)
```

In the last example several commands such as the Modify Terminal Status 'MTS) and Install Volume 'IV) commands were used to perform functions from the IS command. Whenever an SCI command is used as part of another command, the writer usually supplies all the required keywords to that command. The keywords may be abbreviated, however, and all the keywords may not be required.

If the keyword is optional, such as DEFAULT MODE 'TTY/VDT) in the MTS command, it is not necessary to include it unless you do not wish the default parameter to be used. Remember, optional keywords were designated by placing an asterisk in front of the keyword type when the procedure was defined.

If the keyword is used, it may be abbreviated but a few guidelines should be followed.

* The abbreviation must begin with the same character as the original keyword.

* Any characters used in the abbreviation must also appear in the original keyword and in the same order.

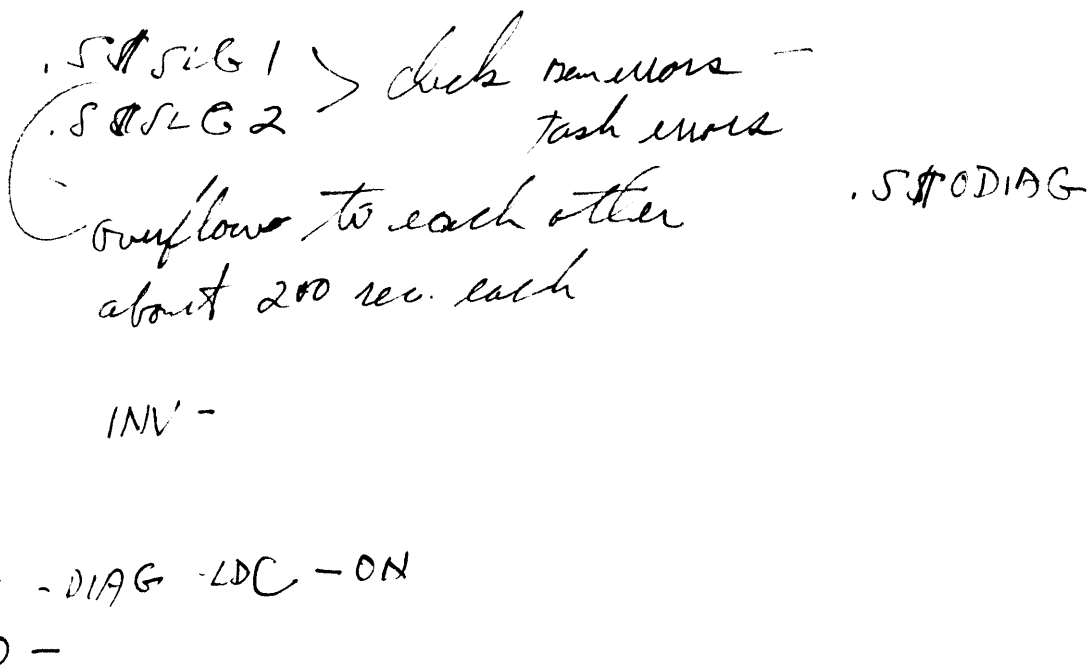
* Digits in the original keyword must be used in the abbreviation and in the same order.

* Characters which follow a special character in the original keyword are ignored.

* Use the first character that follows a blank in the original keyword as part of the abbreviation.

* Be careful that an abbreviation could not be taken for more than one keyword.

If a command is used without supplying the keywords and their values, the command and all keywords will be displayed at the station. The Initialize Date and Time (IDT) and Initialize System Log (ISL) commands are used in this manner by the IS command.



Another command that is easily modified is the IDT (initialize date and time) command. It is written as follows:

```
IDT (INITIALIZE DATE AND TIME) =4,
YEAR = INT,
MONTH = STRING,
DAY = INT,
HOUR = INT,
MINUTE = INT
.SYN   $IDT$ = "&MONTH"
.SYN   $YEAR$ = "&YEAR"
IDT$1 M="&MONTH"
.BID  TASK = >19,
      PARMS = (20,@$YEAR$,@$IDT$,&DAY,&HOUR,&MINUTE)
.SYN   $IDT$ = ""
.SYN   $YEAR$ = ""
```

This command could be modified so that the user does not have to enter the year. The procedure would then have to be modified once a year to keep it current.

```
IDT (INITIALIZE DATE AND TIME) =4,
MONTH = STRING,
DAY = INT,
HOUR = INT,
MINUTE = INT
.SYN   $IDT$ = "&MONTH"
IDT$1 M="&MONTH"
.BID  TASK = >19,
      PARMS = (20,1980,@$IDT$,&DAY,&HOUR,&MINUTE)
.SYN   $IDT$ = ""
```

WORKSHEET

Modify the command procedure Execute COBOL Compiler (XCC) so that the user enters a file name a directory name, and the number of lines in the program. These should be the only prompts or keywords that appear on the screen. The parameter OPTIONS should always default to a cross reference listing. PRINT WIDTH and PAGE SIZE should always be 80 and 55 respectivley. The source, object, and listing access names should always be ddd.SRC.nnn, or ddd.OBJ.nnn, or ddd.LST.nnn where "ddd" is the directory name the user entered and "nnn" is the file name.

```
XCC (EXECUTE COBOL COMPILER <VERSION: 3.2.0 79173>) =2,
SOURCE ACCESS NAME = ACNM ("@$$S"),
OBJECT ACCESS NAME = ACNM ("@$XCC$OB"),
LISTING ACCESS NAME = ACNM ("@$XCC$L"),
OPTIONS = *STRING ("@$XCC$O"),
PRINT WIDTH = INT (80),
PAGE SIZE = INT (55),
PROGRAM SIZE (LINES) = INT (1000)
.SYN MEMX = "&PROGRAM SIZE (LINES)"
.EVAL MEMORY = "@MEMX / 500 * 7168"
.IF "@MEMORY" ,GT, "30840"
.EVAL MEMORY = "30840"
.ENDIF
.IF "@MEMORY" ,LT, "7168"
.EVAL MEMORY = "6144"
.ENDIF
.SYN  $XCC$OB = "&OBJECT ACCESS NAME"
.IF  "@&SOURCE ACCESS NAME", EQ, "@&OBJECT ACCESS NAME"
MSG T="ERROR: SOURCE AND OBJECT ARE SAME NAME"
.EXIT
.ENDIF
.IF  "@&SOURCE ACCESS NAME", EQ, "@&LISTING ACCESS NAME"
MSG T="ERROR: SOURCE AND LISTING ARE SAME NAME"
.EXIT
.ENDIF
.QBID TASK = >87, LUNO = >10,
PARMS = (@&SOURCE ACCESS NAME, @&OBJECT ACCESS NAME,
@&LISTING ACCESS NAME, "&OPTIONS",&PRINT WIDTH,
&PAGE SIZE,@MEMORY,&PROGRAM SIZE (LINES))
.SYN  $XCC$O = "&OPTIONS", $$S = "&SOURCE ACCES NAME",
$XCC$L = "&LISTING ACCESS NAME", MEMORY = "", MEMX = ""
```

One command procedure that the user may wish to modify or add to is Q$SYN. This command is called by the Quit (Q) command when you sign off a terminal. Its function is to delete many of the system synonyms so that synonym table overflow does not occur.

Some synonyms assigned by some of the utilities and some of the software are not included. From time to time the user may find a synonym that was not deleted by either the procedure that assigned it or by Q$SYN. You could add such synonyms to this procedure or any that are created by your own procedures. It is not always desireable to delete all synonyms created by a procedure at the termination of the procedure, but rather leave them so that they may be accessed by other command proceudres. $$LU is a good example of a synonym created by the AL command. It is available to other procedures that may be invoked but should be deleted eventually to conserve storage in the synonym table. It will be deleted by Q$SYN.

Note that the synonym table is stored in .S$TCALIB when a user signs off provided that the user was logged in with a user id. Each record in this file is 864 bytes of which 804 bytes are available to store synonyms and their values.

```
Q$SYN
.SYN   $$OB="", $$RI="", $$VN="", $PF$D="", $SF$P="",
$AA$P="", $IT$T="", $LLR$P="", $XCP$L="",
$SPI$A="", $SPI$I="", $SPI$T="", $XCT$L="",
$SRF$F="",  $SRF$P="",  $SRF$R="", $LD$P="",
$XE$ = "", $$BT="", $XD$D="", $XLE$OB="",
$$LU="",   $$S="",  $BD$D="",  $BD$S="",
$$LD="",  $CD$M="",  $CD$C="",  $$PF="",
$$ML="",  $$OP="",  $XLE$C="",  $XLE$L="",
$XMA$O="",  $XMA$L="",  $XMA$E="", $XMA$OP="",
$XFC$O="",  $XFC$L="",  $XFC$E="", $XFC$OP="",
$XCC$L="",  $XCC$O="",  $XCC$OB="", $XCP$O="",
$XFT$P="",  $XFT$T="",  $XCT$P="", $XCT$T="",
$CFK$L="", $CFK$PN="", $CFK$LRL="", $CFK$KN="",
$CFK$PRL="", $CFK$KS="", $CFK$IA="", $CFK$M="",
$CFK$SA="", $CFK$MS="", $RPG$PL="", $RPGU="",
$RPG$P="", $RPG$L="", $RPG$B="", $RPG$S="" ,
$XMA$OPL="", $XMA$MC="",
```

For the following software, you may wish to add:

SORT/MERGE	XSMFNM, XSMSEQ, XSMNXT
~~DBMS 990~~	~~DBPSW, P$SC~~
Misc.	$$BC, ~~EC~~

Do not use

LAB EXERCISE

Write a procedure that will accept a directory name and then use that directory name to store from 1 to 10 files. The files are currently in the directory TI.COBOL. There are currently 4 files in this directory with the names TCTEMP1 through TCTEMP4. Use the command procedure UNIQUE to give unique names to the files as they are copied to your directory. UNIQUE will generate names with the format .S$TMPxn where "x" is your station number and "n" is the Nth file to be copied. Set up a loop to increment the input file name and prompt whether or not you wish to continue with the next file to be copied.

LAB EXERCISE

Modify the previous lab exercise so that up to 100 files may be copied from any directory. Also modify the output filename to be something other than .S$TEMPxn. Be sure UNIQUE is in your own directory before making any modifications to it.

LAB EXERCISE

Modify the Create Key Indexed File (CFKEY) command so that the starting position of the key defaults to the next available position after the previous key that was created. For example, if the primary key begins in position 1 and has a length of 5, the next key that is defined should show position 6 as a default value for START POSITION.

Also modify the default for MODIFIABLE so that NO will be the the default for the primary key and YES will be the default for all other keys. Be careful with all synonym assignments.

Be sure that you have copied all necessary commands into your directory before attempting any modifications. Execute a Map Key File (MKF) command to verify that the file was created with the desired attributes. A suggested test of your PROC would be to create a file with the following attributes.

KEY	START COLUMN	LENGTH	MODIFIABLE	DUPLICATES ALLOWED
1	1	5	N	N
2	6	20	Y	Y
3	26	10	Y	N

5.2 MODIFYING THE TERMINAL STATUS

The Modify Terminal Status (MTS) command allows the user to modify certain conditions associated with each station in a configuration. The attributes associated with a station that the user may modify include:

* On-line/off-line status
* VDT/TTY mode
* Login required
* Privilege code

A series of MTS commands is typically included in the IS command as was done in a previous example. This enables the user to tailor the system to meet his requirements. One drawback to this method is that unauthorized users can gain access to a system that is not physically locked by performing an IPL on the system. The user may then bid SCI before the IS command is executed.

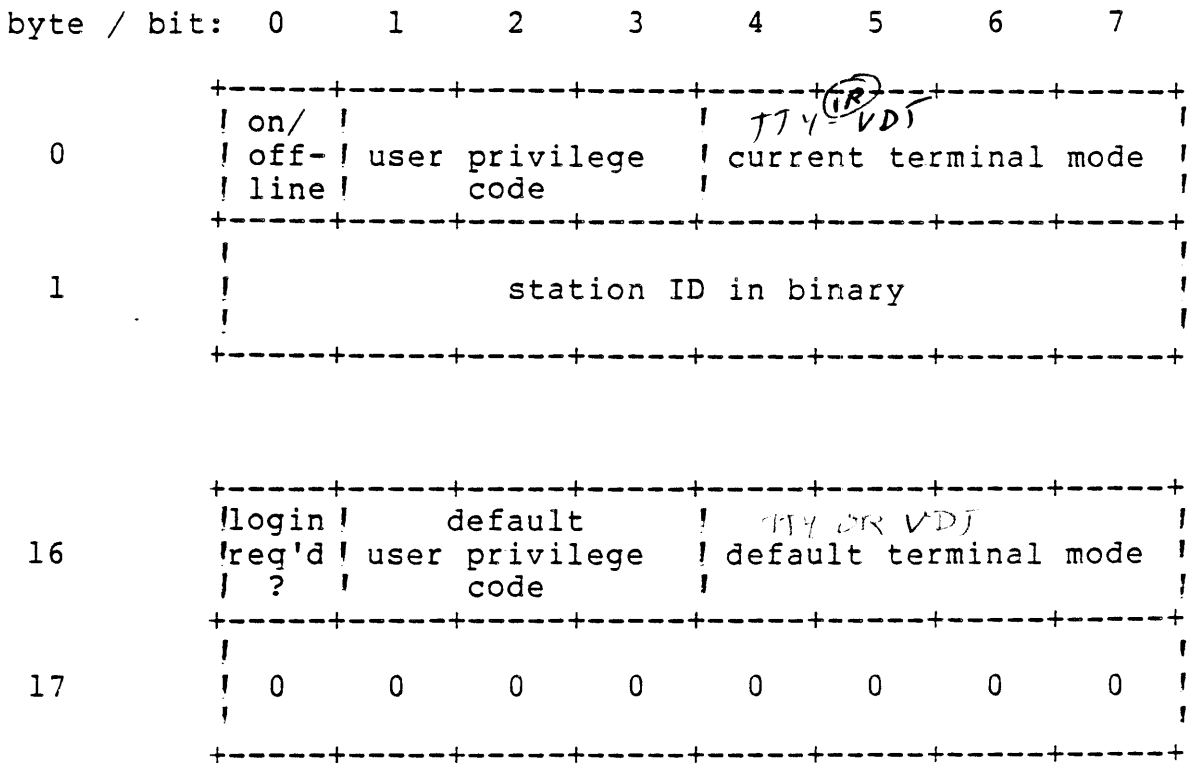
A solution to this problem is to modify the terminal status data on the system disk. This will not only prevent unauthorized use of the system, but will significantly speed up the execution of the IS command where several terminals are involved.

Terminal status information is maintained in the System Communications Area (SCA), a procedure shared by the active SCI tasks. This procedure resides on the system program file (.S$PROGA) and contains entries for stations ST01 through ST39.

The initial status for all stations is set equivalent to the following MTS responses:

```
MODIFY TERMINAL STATUS
            TERMINAL NAME: STxx
      NEW STATUS (ON/OFF): ON
       NEW MODE (TTY/VDT): TTY
          LOGIN REQUIRED?: NO
      USER PRIVILEGE CODE: 7
  DEFAULT MODE (TTY/VDT): VDT
```

There is a 16 word area allocated in the SCA for each of the 39 stations SCI is capable of supporting. The information affected by the MTS command is contained in two of these 16 words in the following manner:



Byte 0, bit 0:

0 = on-line
1 = off-line

Stations marked as off-line cannot access SCI and can only be used as physical I/O devices by the DX10 operating system. In addition, stations marked off-line are not listed by the List Terminal Status (LTS) command. It may be desireable to mark all stations not physically generated in the system as being off-line to remove them from the LTS listing. Remember that the SCA will contain entries for stations ST01 through ST39 regardless of the number of stations actually specified during SYSGEN. All stations are initially marked as being on-line.

Byte 0, bits 1-3:

0-7 = User Privilege Code

This field contains the user privilege code associated with this station when login is not required and, therefore, a user ID and associated privilege code are not available to SCI.

Byte 0, bits 4-7:

>1 = TTY mode - BIT 7 ON.
>F = VDT mode — ALL ON

This field describes the mode of the station to SCI which determines how menus are to be displayed, etc.

Byte 1:

>01 - >27 = Station ID in binary

The station ID in binary (>01 through >27) is maintained here and must be preserved when bits in the preceding byte (byte 0) are modified with the Modify Program Image (MPI) command.

Byte 16, bit 0:

0 = LOGIN not required
1 = LOGIN required

Login required indicates that a user must supply a valid User ID and optionally a password to access SCI. All stations are initially marked such that login is not required.

Byte 16, bits 1-3:

0-7 = Default User Privilege Code

The default user privilege code is used to establish a maximum user privilege code for this station regardless of the privilege code associated with a user ID at login or the user privilege code assigned via the MTS command to the station. All stations are initially given a default privilege code value of 7. This value can only be modified by using the MPI command.

Byte 16, bits 4-7:

>1 = Default TTY mode — BIT 7 ON
>F = Default VDT mode — ALL ON

This defines an upper limit on the capabilities of the physical device associated with this station id. Stations marked in VDT mode can be operated in either VDT or TTY mode. Stations marked in TTY mode may be operated only in TTY mode. An INVALID MODE CHANGE error will be received when attempting to set NEW MODE = VDT with the MTS command if the default mode has been established as TTY. Actual TTY devices should be altered to indicate the default mode as TTY.

BYTE 17 - ALWAYS 0

The fields that you might wish to permanently alter on the system disk are:

* On/Off-line status (from on to off for devices not physically present in the system).

* LOGIN required (from no to yes to insure system integrity).

* User Privilege Code (from 7 to 1,2,...,6 as appropriate).

* Current Terminal Mode (from TTY to VDT for all VDT stations).

* Default User Privilege Code (from 7 to 1,2,...,6 as appropriate). This field is not accessible by the MTS command.

* Default Mode (from VDT to TTY for any physical TTY device). This field is not accessible by the MTS command.

5.3 MODIFYING THE SYSTEM DISK

Modifying the system disk is accomplished through the Modify Program Image (MPI) command as follows

```
    MODIFY PROGRAM IMAGE
            PROGRAM FILE: .S$PROGA
     OUTPUT ACCESS NAME:
            MODULE TYPE: PR     (or PROCEDURE)
MODULE NAME OR ID: SCA    (or 02 -- verify with MPF)
             ADDRESS: >xxxx -
    VERIFICATION DATA: >yyyy
                DATA: >zzzz
            CHECKSUM: >zzzz
```

where:

xxxx -- the address of the station entrys (byte 0 or byte 16) to be modified

yyyy -- the current disk data

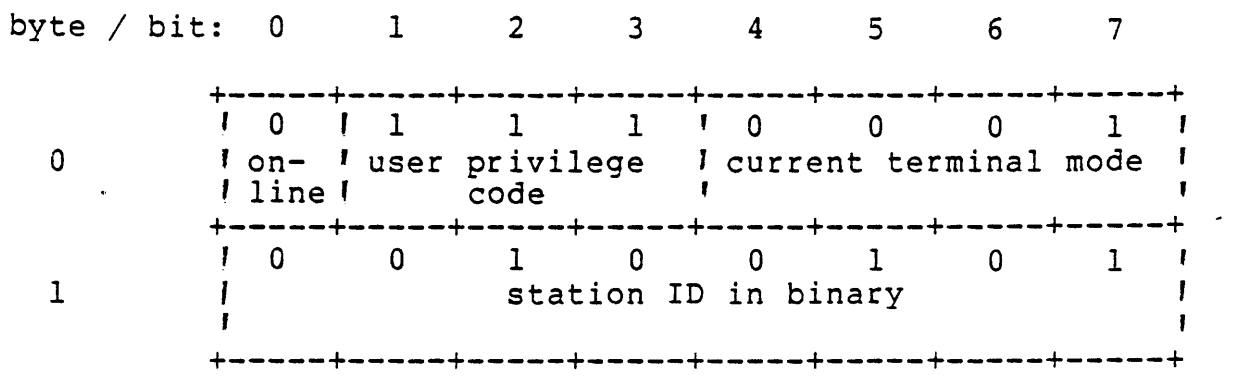
zzzz -- the value to be written to the disk (CHECKSUM is optional)

The address of the station entry can be computed from the address of ST39 which is >0020. Since each station entry requires 16 words, the address of ST38 would be >0040. Each station's entry address can then be computed until then address of the entry for ST01 is computed as >04E0.

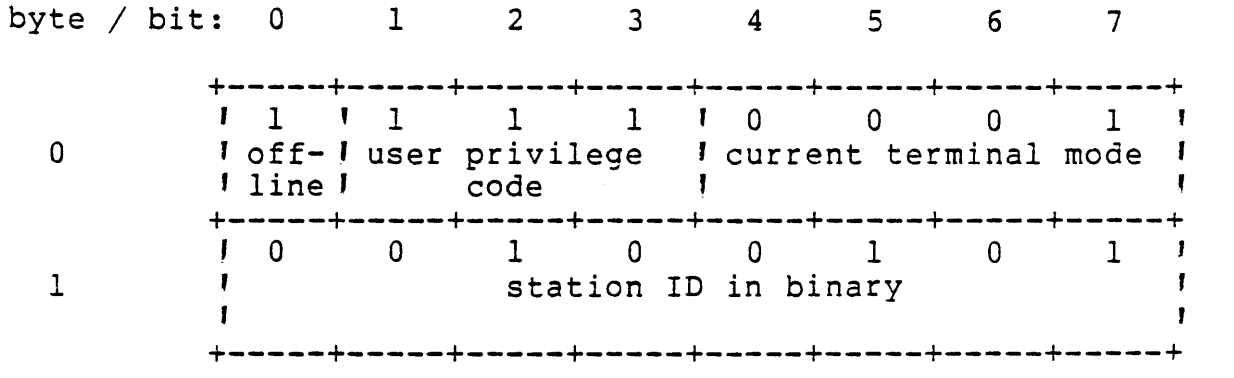
TMPI

This example shows how you would modify ST37 to be off-line.

```
MODIFY PROGRAM IMAGE
            PROGRAM FILE: .S$PROGA
      OUTPUT ACCESS NAME:
             MODULE TYPE: PR
       MODULE NAME OR ID: SCA
                 ADDRESS: >0060
       VERIFICATION DATA: >7125
                    DATA: >F125
                CHECKSUM:
```



Becomes:



The following will be displayed:

```
VERIFICATION DATA
0060   7125
CURRENT IMAGE
0060   7125 5354 3337 0000 0000 0000 0000 0000
NEW IMAGE:   CHECKSUM = F125
0060   F125 5354 3337 0000 0000 0000 0000 0000
```

This example shows how you would modify ST04 to come up in VDT mode with login required.

```
MODIFY PROGRAM IMAGE
            PROGRAM FILE: .S$PROGA
      OUTPUT ACCESS NAME:
             MODULE TYPE: PR
       MODULE NAME OR ID: SCA
                 ADDRESS: >0480 -ST04
       VERIFICATION DATA: >7104
                    DATA: >7F04          (change to VDT mode)
                CHECKSUM:
```

byte / bit:	0	1	2	3	4	5	6	7
0	0 on-line	1 user privilege code			0	0	0	1 current terminal mode
1	0	0	0	0 station ID in binary			0	0

Becomes:

byte / bit:	0	1	2	3	4	5	6	7
0	0 on-line	1 user privilege code			1	1	1	1 current terminal mode
1	0	0	0	0 station ID in binary			0	0

The following will be displayed:

```
VERIFICATION DATA
0480   7104
CURRENT IMAGE
0480   7104 5354 3034 0000 0000 0000 0000 0000
NEW IMAGE:   CHECKSUM = 7F04
0480   7F04 5354 3034 0000 0000 0000 0000 0000
```

This example shows how you would modify ST04 to come up in VDT mode with login required.

```
MODIFY PROGRAM IMAGE
           PROGRAM FILE: .S$PROGA
      OUTPUT ACCESS NAME:
             MODULE TYPE: PR
       MODULE NAME OR ID: SCA
                 ADDRESS: >0060+16
       VERIFICATION DATA: >7F00
                    DATA: >FF00                    (require login)
                CHECKSUM:
```

byte / bit:	0	1	2	3	4	5	6	7
16	0 w/out login	1 default user privilege code			1	1	1	1 default terminal mode
17	0	0	0	0	0	0	0	0

Becomes:

byte / bit:	0	1	2	3	4	5	6	7
16	1 login req'd	1 default user privilege code			1	1	1	1 default terminal mode
17	0	0	0	0	0	0	0	0

The following will be displayed:

```
VERIFICATION DATA
0490   7F00
CURRENT IMAGE
0490   7F00 0000 0000 0000 0000 0000 0000 0000
NEW IMAGE:   CHECKSUM = FF00
0490   FF00 0000 0000 0000 0000 0000 0000 0000
```

5.4 NEWS FILE

SCI has a facilitiy for displaying messages to the user on an as needed basis. This facility is called the news file and is a sequential file stored in .S$NEWS. The user may create or modify this file with the Text Editor. The news file, if it exists, will be displayed each time a user bids SCI or logs on.

5.5 MAIN MENU

The main menu is the first menu displayed if the terminal is in VDT mode. It will be displayed every time the terminal is in the menu display cycle provided no other menus have been called for. The main menu shipped with a system is stored in .S$PROC.M$LC. If you wish to change the main menu to something other than the one that has been supplied by Texas Instruments, you may use the menu option of the .OPTION primitive as explained in Module 4. You may also wish to have a main menu that is system wide. If so, you should first create a new menu with the Text Editor and store it in a temporary file. To replace the main menu, all stations on the system but one must be either off, in TTY mode, or displaying a main menu other than the one supplied with the system. The Copy/Concatenate (CC) command may be used to copy your new menu to .S$PROC.M$LC.

5.6 STARTUP AND SIGNOFF TASKS

You may wish to have a certain procedure performed every time someone either bids or quits SCI. An example of such a task is a user written task that will compute the amount of time that a user has been logged onto the system. You can write procedures to perform these startup and signoff tasks and store them in the PROC library. The command procedures must be called M$00 for the startup procedure and M$01 for the signoff procedure.

5.7 SEQUENCE OF EVENTS

The sequence of events for SCI when login is required is:

1. Accept and validate user ID and optional passcode.
2. Retrieve user synonym table and set certain system synonyms e.g., $$ST, $$UI, $$CC, $$MO, ME, etc.
3. Display news file if .S$NEWS exists.
4. Execute M$00 if it exists.
5. Display main menu (M$LC or menu specified by .OPTION) if

in VDT mode.

At signoff:

6. Copy synonym table to .S$TCALIB.

7. Execute M$01 if it exists.

When login is not required the same sequence of events occurs except for steps 1 and 6 and the first part of step 2.

5.8 COMPLETION CODES

Most of the existing command processors such as the assembler, the various compilers, and the link editor, set a completion code that may be accessed through the sysnonym $$CC. If your procedure or batch stream executes any of the processors, it is a good practice to check the value of $$CC after their completion. This can be accomplished with the .IF statement. The status of the completion code should be one of the following:

>0000	No warnings or errors
>4000	Warnings
>8000	Errors

COBOL users may also set the right most byte of the condition code from their program. Execution of a STOP literal statement in a COBOL program will cause the literal or code created by the user to be set in the condition code.

>0033	Implies normal completion with a user code of 33.

5.9 SCI MODE

The synonym $$MO contains a two digit hexadecimal code for the current mode of a station.

>00	Batch mode
>01	TTY mode
>0F	VDT mode

```
CLE  (COMPILE, LINK, AND EXECUTE) = 5,
FILE NAME = ACNM
.USE
.SYN CS = "TI.SRC", CO = "TI.OBJ", CL = "TI.LST"
*
* === COMPILE PROGRAM ===
*
XCCF  SOURCE = @CS.&FILE NAME,
      OBJECT = @CO.&FILE NAME,
      LISTING = @CL.&FILE NAME
*
* === IF NO WARNINGS OR ERRORS, BUILD LINK CONTROL FILE ===
*
.IF @$$CC, EQ, 0
   .DATA TI.LCF.&FILE NAME, SUBSTITUTION = YES
      LIBRARY  .S$SYSLIB
      FORMAT   IMAGE,REPLACE
      PROC     RCOBOL
      DUMMY
      INCLUDE  (RCBPRC)
      TASK     &FILE NAME
      INCLUDE  (RCBTSK)
      INCLUDE  (RCBMPD)
      INCLUDE  @CO.&FILE NAME
      END
   .EOD
*
* === EXECUTE LINK EDITOR ===
*
   XLE  CONTROL = TI.LCF.&FILE NAME,
        LINKED  = TI.PROGF,
        LISTING = TI.LMAP.&FILE NAME
   WAIT
*
* === IF NO WARNINGS OR ERRORS, EXECUTE PROGRAM ===
*
   .IF @$$CC, EQ, 0
      AL  LUNO = "",
          ACCESS = TI.PROGF,
          PROG FILE = YES
      XCTF  PROG FILE LUNO = @$$LU,
            TASK ID OR NAME = &FILE NAME
      RL  LUNO = @$$LU
   .ENDIF
.ENDIF
.SYN  CS="", CO="", CL=""
Q$SYN
```

LAB EXERCISE

Write a procedure that will link a COBOL object program as a DX10 task. The procedure must create the link control file, execute the link editor, and optionally create the program file if it does not already exist. The keywords that should be used by your procedure and their meaning are:

OBJECT ACCESS NAME -- pathname of the COBOL object program

APPLICATION NAME -- name to be given to the task

PROGRAM FILE NAME -- name of the program file that is to receive the linked output

1st LINK TO THIS OUTPUT FILE -- has this program file been been used before, if not, then you will have to create the program file

LIBRARY ACCESS NAME -- library to be searched in an effort to resolve references (optional keyword)

LISTING ACCESS NAME -- file or device to which the link map is to be sent

Your procedure should create a temporary file to contain the link control file and then delete this file when the link editor has terminated. Use $$ST to create a unique file name. Be sure that you erase any synonyms that are generated by your procedure.

MODULE 6

BATCH COMMAND STREAMS

OBJECTIVES

* Write and execute a batch stream.
* Use all applicable SCI functions in a batch stream.

6.1 BATCH STREAMS

The user can commmunicate with SCI in background using commands that are in a batch stream. SCI in the batch stream does not interact with the terminal when a batch stream is being executed. This implies that all commands in the batch stream must be in the proper format. That is, all commands must include required keywords and responses.

The batch feature is very useful when executing a very long task or a series of tasks. Operator intervention is not required except to place the batch stream into execution. Once the batch stream has begun execution, the foreground of the station is available to the user.

6.2 EXECUTE BATCH

The XB command is used to place a batch stream into execution.

```
[] XB

EXECUTE BATCH
      INPUT ACCESS NAME: (FILE CREATED w/ 'XE' - COMMANDS)
    LISTING ACCESS NAME:
```

where:

INPUT ACCESS NAME -- sequential media containing the batch commands

LISTING ACCESS NAME -- file or device to be used for listing the batch stream and any messages that are generated by it

Commands are entered in a batch stream in the same manner that they are entered in an SCI procedure. If a command is to be entered, it must include the required keywords and a valid response. You may abbreviate keywords but must follow the same rules that where required for abbreviating keywords in a PROC. All primitives are valid for inclusion in a batch stream.

Certain commands may not be used in a batch stream. They are:

* All Debug and Text Editor commands
* Activate Task (AT) command
* Kill Background Task (KBT) command
* Show Background Status (STS) command

* Modify Synonym (MS) command

* Execute GEN990 (XGEN) command

* ~~Modify Volume Information (MVI) command~~

6.3 KILL BACKGROUND TASK

The KBT command allows you to terminate the execution of any background activitiy at the station.

[] KBT

6.4 SHOW BACKGROUND STATUS

The SBS command will display a message that describes the status of background activity at the station.

[] SBS

SHOW BACKGROUND STATUS

AWAITING COROUTINE ACTIVATION

Vol 2 - MAS [redacted] MESG Explanations

6.5 WAITING FOR BACKGROUND TERMINATION

The WAIT command will lockout the foreground of a terminal while a background task is executing. After starting the execution of background task, simply key in the command WAIT. This command may be aborted to free up the foreground by striking the CMD key on a 911 VDT or by striking CTRL and X on a TTY device. A message similar to those displayed by the SBS command will then appear and the foreground becomes available.

```
[] WAIT

--WAITING FOR BACKGROUND TASK TO COMPLETE--



== FOREGROUND COMMAND EXECUTING ==
```

6.6 BEGIN AND END BATCH

Two commands may be used to begin and terminate the batch stream. The BATCH command will cause the user and station IDs to be listed at the beginning of the batch stream listing as well as the input and listing access names from the XB command. The date and time will also be listed. You may optionally include the synonym table by including the keyword LS = YES. The purpose of this command is to delete all the system synonyms from the synonym table before execution of the batch stream begins so that synonym table overflow does not occur.

The EBATCH command will cause the date and time to be placed at the end of the batch stream listing. The synonym table may be included in the listing by using the same keyword LS = YES. If the keyword TEXT = "xxxxx" is used, the user supplied text is supplied in place of the usual batch stream completion message.

Recall that the primitive .STOP may be used to send a message to the foreground terminal local file from a batch stream as well as setting the synonym $$BC. It may be placed anywhere in the batch stream or used in place of the EBATCH command.

6.7 ERROR COUNT

The Error Count (EC) command may be useful when working with a very lengthy batch stream. This command will check the value of the condition code ($$CC) and if it is greater than 0, increment the synonym EC by one. If placed after each batch stream command, the user may check for successful completion of that command. The total number of errors may be displayed at the end of the batch stream by using .STOP. This will eliminate the need to check the batch stream listing if no errors were encountered since error messages become a part of the listing.

If the user has set the condition code from a COBOL program, you may wish to modify this PROC since it assumes that any value for $$CC greater than zero is an error.

Use of primitives to bid a task or a system utility does not allow the condition code to be set. If the user needs to check the condition code after performing a specific function, the task or utility should be bid by using a PROC and not a primitive.

```
BATCH
.SYN CS=TI.COBOL.SRC
.SYN CO=TI.COBOL.OBJ
.SYN CL=TI.COBOL.LST
XCC  SOURCE=CS.MAIN,
     OBJECT=CO.MAIN,
     LISTING=CL.MAIN
EC
XCC  SOURCE=CS.SUB1,
     OBJECT=CO.SUB1,
     LISTING=CL.SUB1
EC
XCC  SOURCE=CS.SUB2,
     OBJECT=CO.SUB2,
     LISTING=CL.SUB2
EC
.IF @$E$C, GT, 0
   .STOP TEXT="CANNOT LINK -- @$E$C ERRORS IN COMPILE PHASE"
.ENDIF
.DATA  TI.COBOL.LCF.PROG1
     LIBRARY   .S$SYSLIB
     LIBRARY   TI.COBOL.OBJ
     FORMAT    IMAGE, REPLACE
     PROC      RCOBOL
     DUMMY
     INCL      'RCBPRC)
     PROC      SUBS
     INCL      'SUB1)  'SUB2)
     TASK      MAIN
     INCL      'RCBTSK)
     ALLOCATE
     INCL      'RCBMPD)
     INCL      'MAIN)
     END
.EOD
XLE  CONTROL ACCESS NAME=TI.COBOL.LCF.PROG1,
     LINKED OUTPUT=TI.COBOL.PROGF,
     LISTING ACCESS NAME=TI.COBOL.LMAP.PROG1
EC
EBATCH  TEXT="@$E$C ERRORS ENCOUNTERED"
```

Figure 6-1

6.8 CREATING A KEY FILE

To create a key indexed file in batch mode, the batch stream must include a CFKEY command followed by 1 to 14 KEY commands and an ENDKEY command. The keywords for the CFKEY command are:

```
PATHNAME                    *required)
LOGICAL RECORD LENGTH       *required)
PHYSICAL RECORD LENGTH
INITIAL ALLOCATION
SECONDARY ALLOCATION
MAXIMUM SIZE                *required)
```

The keywords for the KEY command are:

```
START POSITION      *required)
KEY LENGTH          *required)
DUPLICATES?
MODIFIABLE?
```

The ENDKEY command does not have any keywords.

The commands must be entered in order and must all be present.

```
BATCH  LS=Y
*
* === CREATE MASTER INVENTORY FILE ===
*
CFKEY  PN="TI.INV.MSTR", LRL=80, MS=3500
KEY  SP=1, KL=5, DUP=N, MOD=N
KEY  SP=6, KL=20, DUP=Y, MOD=Y
KEY  SP=48, KL=6, DUP=N, MOD=Y
ENDKEY
*
* === MAP KEY FILE TO VERIFY ATTRIBUTES ===
* === DETERMINE FILE ALLOCATION OF NEWLY CREATED FILE ===
*
MKF  PN="TI.INV.MSTR"
MD  PN="TI.INV.MSTR", SF=N
EBATCH
```

Figure 6-2

PROC CAN BUILD A BATCH STREAM - 'XB'

LAB EXERCISE

Write and execute a batch stream that will compile the program that you wrote for the lab exercise in Module II. The batch stream should then build the link control files required by that exercise and execute the Link Editor for each control file. Be sure to check for succesful completion of the Link Editor after each link.

6.9 SUMMARY OF USER WRITTEN SCI

SCI procedures and batch streams should be written to simplify the execution of application programs and various system tasks. Two basic rules should be kept in mind when writing SCI procedures and batch streams.

1. Release synonyms as soon as possible.
2. Use SCI primitives whenever possible.

The first rule will help prevent synonym table overflow. The synonym table is a fixed length and may overflow if the user requires the use of many synonyms. The user should always release synonyms as soon as they are no longer needed. The following examples delete their synonyms after the COBOL program has been executed. If the user written SCI does calls any system SCI procedures, then it would be a good idea to also call Q$SYN to delete any system synonyms.

The second rule will improve the execution speed of your procedure or batch stream. When SCI calls a procedure, it must locate the procedure file, open the file, execute the instructions, and close the file. All of this is time consuming. Figures 6-3 and 6-4 show the same SCI procedure written with and without SCI primitives. Timings were run from the point when the SCI command was entered until the first application program input was requested. When not using primitives, the procedure took approximately 11.2 seconds while the second example which used primitives took only about 6.5 seconds. Table 6-1 shows a detailed comparison of timings.

```
DDA  'DEMAND DEPOSIT ACCOUNTING DEMO)
AS   SYN = INDEX1, VAL = .DATA.DDA.INDEX1
AS   SYN = MAIN, VAL = .DATA.DDA.MAIN
AS   SYN = PEOPLE, VAL = .DATA.DDA.PEOPLE
AS   SYN = SYSFILE, VAL = .DATA.DDA.SYSFILE
AS   SYN = THINGS, VAL = .DATA.DDA.THINGS
AS   SYN = STOPS, VAL = .DATA.DDA.STOPS
AS   SYN = PRTFIL, VAL = .DATA.DDA.PRTFIL
AS   SYN = PRTFL2, VAL = .DATA.DDA.PRTFL2
XCTF  P = >51, TASK = >12, DEBUG = NO, MESS = DUMY
.SYN  INDEX1 = "", MAIN = "", PEOPLE = "", SYSFIL = ""
.SYN  THINGS = "", STOPS = "", PRTFIL = "", PRTFL2 = ""
Q$SYN
```

Figure 6-3 Using SCI Command Procedures

```
DDA  (DEMAND DEPOSIT ACCOUNTING DEMO)
.SYN   INDEX1 = .DATA.DDA.INDEX1
.SYN   MAIN = .DATA.DDA.MAIN
.SYN   PEOPLE = .DATA.DDA.PEOPLE
.SYN   SYSFIL = .DATA.DDA.SYSFIL
.SYN   THINGS = .DATA.DDA.THINGS
.SYN   STOPS = .DATA.DDA.STOPS
.SYN   PRTFIL = .DATA.DDA.PRTFIL
.SYN   PRTFL2 = .DATA.DDA.PRTFL2
.BID   TASK = >12, LUNO = >51, PARMS = (,NO,DUMY)
.SYN   INDEX1 = "", MAIN = "", PEOPLE = "", SYSFIL = ""
.SYN   THINGS = "", STOPS = "", PRTFIL = "", PRTFL2 = ""
```

Figure 6-4 Using Primitives

Table 6-1

		PROCEDURES		PRIMITIVES	
STEP	PROCEDURE FUNCTIONS	STEP TIME	CUMULATIVE TIME	STEP TIME	CUMULATIVE TIME
1	Assign DDA & Process Start	0.33	0.33	0.33	0.33
2	Process 8 Synonyms	3.70	4.03	0.33	0.66
3	Bid COBOL Program	2.90	6.93	1.50	2.16
4	Begin COBOL Processing (includes file and runtime processing)	4.30	11.23	4.30	6.46

MODULE 7

SYSTEM GENERATION

OBJECTIVES

* Describe the function of the system task scheduler.
* Perform a system generation for a given configuration.
* Assemble, link, patch, test, and install a generated system.
* Modify an existing system configuration.

7.1 DX10 TASK SCHEDULER

The DX10 Operating System requires that each task have a defined priority level. There are 132 priority levels:

Highest	0	DX10 internal use
	R1-R127	Real-time priorities IRJ. COMMAND OR MTE
Lowest	1,2,3	Interactive and batch mode
	4	Floating priority

Level zero is intended for the most critical system functions and is reserved for DX10 internal use only.

Real-time priorities provide the user with the capability to supercede all but the most important system tasks. For applications which require an expedious access to the CPU, DX10 will delay some routine maintenance of system duties in an effort to schedule real-time tasks.

Priorities one, two, three and four are designed to satisfy the requirements of most installations. Priority level one gives quick response for programs which interact with the users terminal, while level two is adequate for programs requiring multiple-disk accesses. Priority four automatically switches priority levels between levels one and two as the program executes.

Priority level three is for batch streams and tasks not requiring user interaction. At this level tasks access the CPU only when no higher priority tasks (interactive, real time or system) are waiting for execution.

DX10 always schedules the highest priority task waiting for execution.

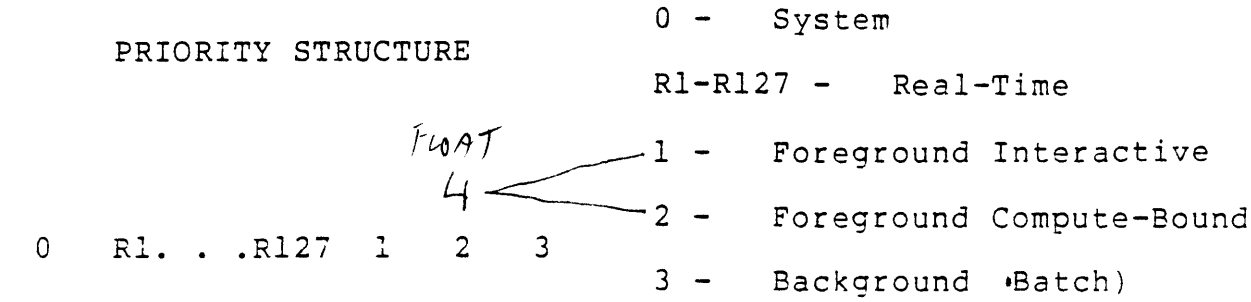


Figure 7-1

Four SYSGEN parameters determine how the scheduler works:

1. TIME SLICING - YES/NO

If the time slicing option is selected, then CPU time for a given priority level will be allocated on a round robin basis among all the active tasks on that queue.

If time slicing is not chosen, then the first task on a queue will be allocated CPU time until it terminates, is suspended, or an external event causes a rescheduling.

2. LENGTH OF TIME SLICE

A multiple of 50 msec intervals.

3. TASK SENTRY - YES/NO

Task Sentry is a SYSGEN option which guards against CPU lock out. DX10 always executes the highest priority task in the system; therefore, it is possible to lock out lower priority tasks for seconds at a time.

When a task remains compute bound at any priority for a specified number of 50 millisecond intervals, the Task Sentry will lower the priority of the task by one. This lowering process continues for as long as the task remains compute bound or until the task reaches priority three. When the task does suspend, the task sentry will restore the task to its proper priority.

4. SENTRY TIME

A multiple of 50 msec intervals.

The scheduling may be affected by these events:

* PREEMPTIVE BIDDING - A higher priority task always gets the CPU when it becomes active.

* The executing task suspends.

* A time delayed task is due to be activated.

* The priority of the executing task is lowered by the Task Sentry (if Task Sentry is active in the system).

* A task completes a time slice (if the time slice option was included in the system).

Table 7-1 Interacting Factors in Scheduling

TIME SLICE	TASK SENTRY: YES	TASK SENTRY: NO
YES	round robin on queue task bumped when sentry time up	round robin on queue when queue is exhausted, next queue is called
NO	first task on queue gets CPU until sentry time up, then bumped	highest task hogs CPU as long as it wants

7.2 SYSTEM GENERATION

DX10 is a modular operating system which can be tailored for each user's needs. Customizing the system causes resources to be utilized more efficienctly.

7.2.1 Customized System Generation.

Tailoring the operating system provides the following benefits:

* Reduces disk and memory space requirements by eliminating unnecessry modules, such as Device Service Routines (DSRs) for equipment not included in the system.

* Eliminates replication of DSRs for multiple installations of a device type.

* Adds DSRs for non-standard devices.

* Adds user defined routines as operating system Supervisor Calls (SVCs).

* Adds user defined routines as Extended Operation Processors (XOPs).

* Adjusts operating system parameters for best efficiency in a given installation.

7.3 GENERATING A DX10 OPERATING SYSTEM

System creation includes the following steps:

1. XGEN Generate
2. ALGS Assemble and Link
3. PGS Patch the Generated System
4. TGS Test the Generated System
5. Perform an IPL and checkout system
6. IGS Install the Generated System

When performing a system generation on a new disk, the following steps must be performed initially:

1. INV Initialize the New Volume
2. CFDIR Create the Directory .S$SYSGEN

7.4 GENERATE

XGEN is an interactive program that, by prompting for information, will allow a user to generate a new DX10 system. The Sysgen utility uses prompting and tutorial displays to guide the user through the generation process. It creates a configuration file and a source file which must be assembled and linked to the rest of the system. Sysgen can be performed in a short period of time.

Requirements include information on the device configuration and programs to be included, such as an interrupt decoder. This information should be gathered before beginning the XGEN process.

7.4.1 XGEN Prompts.

DATA DISC
Disk drive which contains the standard DX10 object modules in .S$SYSGEN. The default is DS01.

TARGET DISC
Disk drive onto which created files are to be placed. The default is DS01.

GT TIME: TI.

INPUT — RTM

Name of the system to be modified. If you are creating a new system, leave this field blank.

OUTPUT — RTM

Name of the system being created. Name should be 1 to 5 characters with the first character being alphabetic.

LINE

Power frequency for the location of the system. USA = 60 hertz; Europe = 50 hertz. The default is 60 hertz.

TIME SLICING ENABLED: YES

If the system has time slicing then a task will execute for one time slice value before being suspended and another task is allowed to execute. The default is YES.

TIME SLICE VALUE = 1

If Time Slicing is enabled then the length of the time slice must be designated. Its value is multiple of system time units (50 Msec). The default is 1 system time unit.

TASK SENTRY ENABLED = NO

The Task Sentry will reduce the priority of a task that has executed for a given period of time. Without the sentry, a task could theoretically maintain control of the CPU. The default is NO.

TASK SENTRY VALUE

1100 - BASE
1 FORE
1 BACK
300 / FORE
350 / BACK
200 / DRIVE

The length of time, in system time units, that a task may execute before its priority is lowered. This is applicable only if Task Sentry is enabled. The default is 60 system time units.

— THEN BUMP UP MORE (SAFETY)

TABLE

Size of the sytem table containing system log messages, Intertask Communication Areas, buffered SVC blocks, and many system built tables. A memory estimator is available in the Sec 1 Appendix to Volume V of the DX10 reference manuals that is very useful with this prompt. No default value is given.

COMMON = 0 NONE USING

The user may optionally include his own object code that defines a common area in memory for use by user tasks. There is no default value. DEFAULT = (NONE)

INTERRUPT DECODER — DEFAULT = (STANDARD)

The user may supply the object code of his own interrupt decoder. There is no default value. DEFAULT = NONE

FILE MANAGEMENT TASKS = 2

The number of file managers to be included in the system. The default value is sufficient for nearly all systems unless a large number of disk drives will be in use. The default is 2.

CLOCK = 5 — EITHER OR 15 — use 5 - IF NO GOOD GOING WAY UP = 15

This is the interrupt level of the system clock. The default is 5.

ID

The user may have a startup task executed every time an IPL is performed. The task must be installed on the system program file, .S$PROGA. If this option is desired, enter the installed ID of the task. There is no default value. NONE

OVERLAYS = 2 — USE FOR ONLY COMMON

The number of system overlay areas. Each overlay area requires 400 words. MEM Adding overlays may reduce the number of disk accesses for system tasks. The default is 2.

SYSLOG = 6

Maximum number of system log messages that may be queued before being sent to the log file. Area for this is allocated in the sytem table. If the value supplied is too small, messages may be lost. If the value supplied is too large, the table may overflow. The default is 6.

BUFFER MANAGEMENT = 1024

Size, in bytes, of the file buffer. This should be at least as large as the largest physical record or at least one ADU. The default is 1024 bytes = 1K

I/O BUFFERS

Area used for special devices and initiate I/O calls. This value will be added to the sytem table area. Size is given in bytes. The default is 0 bytes.

TEXAS @ 512

INTERTASK — FIND OUT HOW BYTES MUCH REQUIRED

Size of the area within the sytem table area which is dedicated for Intertask Communication, such as that generated by GETDATA and PUTDATA SVC calls. This may be required for SORT/MERGE and the 3780 communication emulator. It is required for TIFORM. The default is 100 bytes.

2 BYTES / PER

KIF = YES

Include logic to support key index files? Requires approximately 2K words in the system root. This is necessary for COBOL. The default is YES.

SEQUENTIAL PLACEMENT = YES

This determines whether the sequential or hashed placement algorithm should be used. The default is YES.

COUNTRY CODE = US

This parameter identifies the country where the system is installed. A list of the available country codes may be found by entering a question (?) mark twice. The default is US.

POWER FAIL = NO — WE DONT HAVE

If the system has a backup power supply, 911 VDTs may be included in the power recovery feature. The default is NO.

SCI BACKGROUND = 2 — MINIMUM 3

This is the maximum number of SCI tasks that can be executing in the background at one time. If a request is made for an additional background task, the task will be queued. Each background task beyond the first requires 350 words of system table area. The default is 2.

SCI FOREGROUND (FORE GROUND / PER STATION)

This is the maximum number of SCI tasks that can be executing in the foreground at one time. An error will be generated if an attempt is made to exceed this number. A good practice is to have one foreground task available for each terminal that is being used for program development or the execution of application tasks. Each foreground task beyond the first will require 300 words of system table area. The default is 8.

BREAKPOINT = 5 — 16 C TEXAS

The maximum number of Breakpoints used in debugging a task that may be present in the system at one time. The default is 16.

CARD 1

LDC

The interrupt level for the half of the expander card used for expansion chassis 1 to 4. Interrupts within those chassis will be given distinctive interrupt positions. Strike RETURN to indicate no expansion chassis.

CARD 2

The interrupt level for the half of the expander card used for expansion chassis 5 to 7.

DEVICE

At this point, each device to be included in the system must be specified. The prompts for each device are different and based on the characteristics of the device. Each device must be specified seperately even though multiple devices of one device type are to be included. The hardware configuration of each device must be known before the device may be defined. To begin the definition of a device, enter the appropriate device mnemonic as given below:

* CR Card Reader
* LP Line Printer
* K820 820 Terminal
* ASR 733 ASR

*	KSR	KSR
*	VDT	911 or 913 CRT (CRT may also be entered)
*	MT	Mag Tape
*	DK	Floppy Diskette (single sided/density)
*	COM	Communications
*	SD	Nonstandard Device

The following pages explain some of the parameters that must be defined for a few of the more common devices.

K820

CRU

What is the CRU address of the device. This is needed for I/O operations. Determined by the hardware configuration. Should be given on the top of the chassis. The default is >00.

ACCESS TYPE

A record oriented device has exclusive access only during an I/O operation. A file oriented device has exclusive access from open to close. The default is RECORD.

TIME OUT

Time given, in seconds, for an I/O operation. If operation has not occurred during the allotted time, the system assumes an error has occurred with the device and aborts the operation. A time out is generally not asigned to a keyboard device. The default is 0.

CHARACTER QUEUE

The number of unsolicited characters entered at the keyboard which will be buffered between I/O requests. Any characters entered over this maximum will be lost. The default is 6.

INTERRUPT

Interrupt level assigned to the device. This is determined by the hardware configuration. Multiple devices may share the same interrrupt level. A decoder will determine which device generated the interrupt. The default is 6.

VDT

CRU

The default is >100.

ACCESS TYPE
The default is RECORD.

TIME OUT
The default is 0.

CRT TYPE
Indicate whether this is a 911 VDT or a 913 VDT. The default is 911.

3270 CRU ADDRESS
If this device is to be used with the 3270 communication emulator, then the CRU address of the emulator must be given here. Strike RETURN if this option is not to be used.

CHARACTER QUEUE
The default is 6.

INTERRUPT
The default is 10.

EXPANSION CHASSIS
Indicate which expansion chassis this device is located on. This prompt is only displayed if the interrupt level entered was the same as that assigned to the expander card.

EXPANSION POSITION
Indicate the interrupt level in the expansion chassis that is assigned to the device. This prompt is only displayed if the interrupt level entered was the same as that assigned to the expander card.

LP

CRU
The default is >60.

ACCESS TYPE
The default is FILE.

TIME OUT
The default is 30 seconds.

PRINT MODE
Indicate whether this is a serial or a parallel line printer. The 810 is serial; the 2230 and 2260 are parallel. The default is SERIAL.

EXTENDED
If the line printer has the extended character set, then the logic for those characters needs to be included for the device. The default is NO.

3270 CRU ACRESS
The default is NO.

INTERRUPT
The default is 14.

DS = DSK

TILINE
Indicate the TILINE address for this device. The default value is device dependent.

DRIVES
Indicate the number of disk drives supported by this controller. From 1 to 4 drives can be suppororted by a single controller. Each will have the same TILINE address and interrupt level. The default is 1.

DEFAULT RECORD SIZE DX10 = 2F8 -
Size of the default physical record. Should be equal to the size of an ADU on the system disk. The default is 864 bytes.

SVC
User-defined supervisor calls may be included with the operating system and then accessed through XOP level 15. Information on SVC structure may be found in Volume V of the DX10 reference manuals. This prompt is requesting the beginning label of the SVC code. Strike RETURN for none.

XOP TEXAS C/S
User-defined Extended Operations (XOPs) may be installed at XOP Levels 0 through 14. Volume V of the DX10 reference manuals contains the information needed to create an XOP. If a user-defined XOP is to be included, indicate here the level, else strike RETURN.

7.5 GEN990 COMMANDS

GEN990 operates in two modes:

1. Generate Mode = WHEN ANS PROMPTS
2. Command Mode

The generate mode will be used in responding to the prompts described earlier. The command mode will be used as needed and is entered by issuing a command. The commands are given in Table 7-2.

Table 7-2 GEN990 Commands

COMMAND	SHORT FORM	RESULT
WHAT	W, ?	Provide an explanation of the given parameter. If none is is specified, an explanation will be given of the parameter just prompted.
LIST	L, DS- GIVES LIST IF DRIVES	Print out value of a GEN990 parameter. If ALL is indicated, all defined parameter values are printed. GEN990 requests a listing device.
CHANGE	C	Change the value of a given parameter. The original value of the parameter is displayed first.
DELETE	D	Delete a device. This may only be used after a change command was issued, and the device name displayed.
GENERATE	G	Return to generate mode.
STOP	-	Terminate GEN990.
HELP	-	Abort whatever action is in process and return to command mode.
SAVE	-	Save the previously defined parameters, putting them on the output file. Useful when a sysgen cannot be completed at that time.
BUILD	B	Build the configuration, the source, and the link control files, and terminate GEN990.

7.6 ASSEMBLE AND LINK GENERATED SYSTEM

GEN990 will build the following:

1. Configuration File

2. Source File

3. Link Edit Control Stream

4. The Batch Stream necessary to complete the System Generation

Once GEN990 has reached successful completion, it is necessary to assemble and link the operating system. This is done with the Assemble and Link Generated System (ALGS) command as shown in Figure 7-2.

```
[] ALGS

ASSEMBLE AND LINK GENERATED SYSTEM

          TARGET DISK:  Disk unit which contains the GEN990
                        output files.
          SYSTEM NAME:  Name of output given in GEN990.
     D$DATA LISTING:    File name for Macro Assembler listing.
      BATCH LISTING:    File name for batch SCI listing.
```

Figure 7-2 ALGS Command

```
[] ALGS

ASSEMBLE AND LINK GENERATED SYSTEM

          TARGET DISK: DS01
          SYSTEM NAME: SYS1
     D$DATA LISTING: .S$SYSGEN.SYS1.D$LIST
      BATCH LISTING: .S$SYSGEN.SYS1.BTCHLIST
```

Figure 7-3 Example ALGS Command

As the ALGS process will execute in the background, you may wish to issue a WAIT command. ALGS will normally take about 30 minutes. The following message will be displayed upon successful completion.

```
*** ALGS NORMAL TERMINATION ***
```

7.7 PATCH GENERATED SYSTEM

Once ALGS has sucessfully completed, the operating system must be patched before an IPL may be performed. This is done with the Patch Generated System (PGS) command as shown in Figure 7-4.

```
[] PGS

PATCH GENERATED SYSTEM
        TARGET DISK:  Disk containing all of the files
                      output by GEN990 and ALGS.
        SYSTEM NAME:  Name of output given in GEN990.
      BATCH LISTING:  File name for batch SCI listing.
```

Figure 7-4 PGS Command

```
[] PGS

PATCH GENERATED SYSTEM
        TARGET DISK: DS01
        SYSTEM NAME: SYS1
      BATCH LISTING: .S$SYSGEN.SYS1.PGSLIST
```

Figure 7-5 Example PGS Command

PGS executes in the background, as did ALGS. Issue a WAIT command. PGS should take about 5 minutes.

PATCH STREAM ERROR CT=0

7.8 TEST GENERATED SYSTEM

The system is now ready to be tested. Execute the Test Generated System (TGS) command and perform and IPL to start the system checkout procedure. Should the system fail to work properly, performing another IPL will cause the original system image to be reloaded.

```
[] TGS

TEST GENERATED SYSTEM
        TARGET DISK:  Disk unit specified as the target disk
                      during GEN990, ALGS, and PGS.
        SYSTEM NAME:  Name of output given in GEN990.
```

Figure 7-6 TGS Command

```
[] TGS

TEST GENERATED SYSTEM
        TARGET DISK: DS01
        SYSTEM NAME: SYS1
```

Figure 7-7 Example TGS Command

7.8.1 System Checkout.

TGS will set the new operating system up so that it may be loaded, by performing an IPL, and tested. Follows steps similar to these:

1. Perform an IPL.
2. Bid SCI from a terminal and initialize the system.
3. Bid SCI at each terminal. Do not exceed the foreground limit specified in GEN990.
4. Execute an SCI command from each terminal. Send a file to each disk and tape drive on the system as well as the line printers, ASRs, and KSRs.

Check all Devices

Should any of these steps fail, the system is not fully operational. Check the hardware configuration, does it reflect the device descriptions given GEN990? If the new system does not function properly, another IPL will load the original system.

7.9 INSTALL GENERATED SYSTEM

Once the system checks out, it is ready to be installed as the primary operating system. This may be done with the Install Generated System (IGS) command. The IGS command modifies the volume information on the specified disk pack to select a new operating system as the primary system.

```
[] IGS

INSTALL GENERATED SYSTEM
        TARGET DISK:  Disk unit specified as the target disk
                      during GEN990, ALGS, PGS, and TGS.
        SYSTEM NAME:  Name of output given in GEN990.
```

Figure 7-8 IGS Command

```
[] IGS

INSTALL GENERATED SYSTEM
        TARGET DISK: DS01
        SYSTEM NAME: SYS1
```

Figure 7-9 Example IGS Command

The system is now ready to be fully operational. Each IPL will cause the new system to be loaded into memory.

7.10 SYSTEM UPKEEP

SCI provides several commands for maintaining your system. Included among these are the following:

* List Device Configuration (LDC)
* Show Memory Status (SMS)
* Show Memory Map (SMM)

The List Device Configuration (LDC) command causes the devices that have been included in the system configuration to be listed. Figure 7-10 shows an example of this command. Volume II of the DX10 reference manuals contains a description of the output generated by the LDC command.

```
[] LDC

LIST DEVICE CONFIGURATION
       LISTING ACCESS NAME:
```

DEVICE	TYPE	ADDRESS	INT	CHAS	POS	STATE	CODE
CM01	COMM	0020	6			ON	N
CS01	CASSETTE	0000	6			ON	N
CS02	CASSETTE	0000	6			ON	N
DK01	DISK	01C0	3			ON	N
DK02	DISK	01C0	3			ON	N
DS01	DISK	F800	15			ON	N
DS02	DISK	F800	15			ON	N
DS03	DISK	F820	7			ON	N
EM01	NON-STANDARD	0540	10	1	6	ON	N
LP01	PRINTER	0060	14			ON	N
LP02	PRINTER	0460	10	1	15	ON	N
MT01	MAG TAPE	F880	12			ON	N
ST01	KEYBOARD	0000	6			ON	N
ST02	VDT	0580	10	1	7	ON	N
ST03	VDT	05A0	10	1	14	ON	N
ST04	VDT	0480	10	1	3	ON	N
ST05	VDT	04A0	10	1	11	ON	N
ST06	VDT	05C0	10	1	8	ON	N
ST07	VDT	05E0	10	1	9	ON	N
ST08	VDT	04C0	10	1	4	ON	N
ST09	VDT	04E0	10	1	10	ON	N
ST10	KEYBOARD	0440	10	1	2	ON	N
TM01	NON-STANDARD	0500	10	1	12	ON	N

Figure 7-10 LDC Command

The sysgen dialog that produced the previous configuration example may be found in Appendix B.

The Show Memory Status (SMS) command is useful in finding the size of the operating system, in seeing how much memory the system recognizes it has, in determining the efficiency of the system table area, and in finding the Foreground/Background limit for SCI tasks. The output of the SMS command, which is shown in Figure 7-11, is explained in Volume II of the DX10 reference manuals.

```
[] SMS

SHOW MEMORY STATUS
    LISTING ACCESS NAME:


                    SHOW MEMORY STATUS

                    TOTAL MEMORY SIZE =  176 K WORDS
DX10 OS SIZE = 43.5 K WORDS              USER AREA = 132.5 K WORDS

                    SYSTEM TABLE AREA = 10616 WORDS
CURRENT USAGE =  4927 WORDS              LARGEST AREA USED =  8087 WORDS

                    SCI INFORMATION
FOREGROUND LIMIT = 10 TERMINALS          CURRENTLY ACTIVE =  6 TERMINALS
BACKGROUND ACTIVE LIMIT =  3 TASKS       ACTIVE/WAITING =  2/ 0 TASKS
```

Figure 7-11 SMS Command

The Show Memory Map (SMM) command also displays information about system memory and the size of the operating system. This command also gives the user information on the amount of memory that is in use, the usage of the system disk, and the usage of the CPU. A graphic representation of physical memory is displayed that indicates the installed and run IDs of any tasks or procedures that are active in the system. Buffers in memory are also identified. The SMM command does not have any prompts.

```
[] SMM
```

LAB EXERCISE

Perform a partial sysgen but do not include all of the devices that are currently installed on the system. Terminate the sysgen being sure to save the configuration.

Execute the XGEN command again and complete the previous sysgen.

Assemble and link the generated system. Patch and test the generated system and determine that the newly created system is functional. Do not install the generated system. Perform another IPL to reload the primary system. Don't forget to use the IS command.

MODULE 8

SYSTEM BACKUP AND COBOL INSTALLATION

8.1 SYSTEM BACKUP

The user is responsible for backing up the system disk and data files either on disk or on magnetic tape. The backup copies can be made using the Copy Directory (CD), Backup Directory (BD), Restore Directory (RD), or the Disk Copy/Restore (DCOPY) command.

The type of media (disk or tape) being used determines the command that is used to perform the copy. Copy Directory copies one hierarchy (disk) to another hierarchy (disk). Backup Directory copies a hierarchy (disk) to a sequential medium (magnetic tape). Restore Directory copies a sequential medium (magnetic tape) to a hierarchy (disk).

DCOPY can be used to copy from disk to disk, from disk to tape, or from tape to disk.

8.1.1 Copying from Disk to Disk Using Copy Directory.

The Copy Directory (CD) command allows users to copy a set of files under one directory on a disk to another directory on a disk. The contents of the source directory do not change as a result of the copy operation. Files and directories that do not exist in the destination directory are automatically created by Copy Directory. The CD command creates the topmost directory equal in size to the topmost directory being copied.

The CD command copies all files and aliases in a directory, except files named .S$ROLLA and .S$CRASH, unless the user limits the CD command with directives and options. Directives are supplied to the CD command in the form of a control file as described in the Volume II of the DX10 reference manuals. The options are described in Table 8-1.

8.1.1.1 CD Command Format.

```
[] CD
COPY DIRECTORY
          INPUT PATHNAME:
         OUTPUT PATHNAME:
   CONTROL ACCESS NAME:
   LISTING ACCESS NAME:
                OPTIONS: ADD,ALIAS,
```

The user should respond to the CD prompts as follows:

INPUT PATHNAME: Enter the pathname identifying the topmost directory of the set of files to be copied. Optionally, the pathname identifies a single file when only one file is being copied.

OUTPUT PATHNAME: Enter the pathname that identifies the directory to which DX10 copies the file or set of files identified by the input pathname.

CAUTION

The output pathname MUST specify a directory. For example, to copy .MY.FILES to .YOUR.FILES, specify .MY.FILES as the input pathname and .YOUR as the output pathname. Do not use .YOUR.FILES for the output pathname as the resulting file pathname would be .YOUR.FILES.FILES.

CONTROL ACCESS NAME: Press RETURN to specify that no control file is to be used.

LISTING ACCESS NAME: Enter the device name of the file pathname to which DX10 should list a summary of the results of the copy operation. Press RETURN to have the listing displayed at the terminal.

OPTIONS: Enter one or more of the following options separated by commas:

ADD
ALIASES
NOALIASES
REPLACE

Press RETURN to accept the default options of ADD,ALIASES. See Table 8-1 for a description of the available options.

NOTE

Although a control file, a master pathname, and a copy pathname are all optional responses to prompting messages, a control file must be specified if either of the other responses is

not given. Both a master pathname and a copy pathname must be provided if a control file is not specified.

8.1.1.2 CD Command Example.

The following example shows the use of the CD command to copy the contents of the directory named VOL1.MYFILES to the directory named VOL2.HISFILES:

```
[] CD
COPY DIRECTORY
          INPUT PATHNAME:  VOL1.MYFILES
         OUTPUT PATHNAME:  VOL2.HISFILES
    CONTROL ACCESS NAME:
    LISTING ACCESS NAME:
                 OPTIONS:  ADD,ALIASES,REPLACE
```

The pathnames specified for the CD, BD, RD, VC, and VB commands cannot start with disk unit names (e.g., DS02.ABCFILES). Use volume names such as MYDSC.ABCFILES.

Table 8-1 CD, RD, BD Options

Option	Purpose
ADD	Files are to be copied unless a file with the same name and at the same level already exists in the destination directory.
REPLACE	Files are to be copied even if a file with the same name and at the same level already exists on the destination directory. The existing file is deleted and replaced with the file being copied.
ALIASES	All aliases associated with a file being copied are to be copied unless an alias already exists with the same name and at the same level in the destination directory.
NOALIASES	No aliases are to be copied.
BLOCK	BLOCK specifies that records will be grouped in 9600-byte blocks. Each block will be written as a physical record.
NOBLOCK	NOBLOCK specifies that each record will be written seperately without blocking.

MULTI — MULTI specifies that the directory spans more than one magnetic tape volume.

When multiple tapes must be used and the end of the tape is encountered, the following message is displayed at the terminal:

MOUNT TAPE X. TYPE TO QUIT, Y TO CONTINUE.

where X is the number of the next volume. The command can be terminated by entering a $. Control then returns to SCI without finishing the process. Otherwise, it is necessary to mount the next tape. When the tape is ready, enter Y. If the tape is not ready or if an error is received, such as a wrong volume number, the error is displayed at the terminal and the tape prompt is reprinted. The prompt will be reprinted until the user enters $ or the tape is in the ready position. When Y is entered and the tape is in the ready position, the command continues processing.

NOMULTI — Multiple tapes will not be used.

8.1.2 Copying from Disk to Tape Using Backup Directory.

The Backup Directory (BD) command allows users to copy a set of files under a directory to a sequential file or to a magnetic tape device, excluding cassette tapes, in a format that allows later restoration of the backup copy by a Restore Directory (RD) command. The Verify Backup (VB) command can be used to verify a copy made by Backup Directory.

Backup Directory copies all files and aliases in a directory, except the files named .S$ROLLA and .S$CRASH, unless the user limits the BD command with directives and options. Directives are supplied in the form of a control file as described in Volume II of the DX10 reference manuals. Options are described in Table 8-1.

8.1.2.1 BD Command Format.

```
[] BD
BACKUP DIRECTORY
      DIRECTORY PATHNAME:
  SEQUENTIAL ACCESS NAME: MT01, Ø2 etc.
     CONTROL ACCESS NAME:
     LISTING ACCESS NAME:
                 OPTIONS: ALIASES,NOBLOCK, BLOCK
```

The user should respond to the BD prompts as follows:

DIRECTORY PATHNAME:	Enter the pathname identifying the topmost directory of the set of files to be copied. Optionally, the pathname identifies a single file when only one file is being copied.
SEQUENTIAL ACCESS NAME:	Enter the name of the device or a pathname identifying the sequential file to which DX10 should backup the directory.
CONTROL ACCESS NAME:	Enter the device name or file pathname from which DX10 reads control directives to control the copy operation.
LISTING ACCESS NAME:	Enter the device name of the file pathname to which a summary of the results of the backup operation are to be listed. Press RETURN to have the listing displayed at the terminal.
OPTIONS:	Enter one or more of the following options, separated by commas, to specify whether files and aliases being copied are to replace files and aliases of the same name on the destination directory: ALIASES NOALIASES BLOCK NOBLOCK Press RETURN to accept the default options of ALIASES,NOBLOCK. Refer to Table 8-1 for an explanation of the options.

8.1.2.2 BD Command Example.

The following example shows the use of the BD command to copy the contents of the directory named .SAMPLE to the magnetic tape mounted on the device named MT01:

```
[] BD
BACKUP DIRECTORY
       DIRECTORY PATHNAME: .SAMPLE
   SEQUENTIAL ACCESS NAME: MT01
      CONTROL ACCESS NAME:
      LISTING ACCESS NAME:
                  OPTIONS: ALIASES,NOBLOCK
```

8.1.3 Copying from Tape to Disk Using Restore Directory.

The Restore Directory (RD) command restores a set of files from a sequential file or magnetic tape to a directory on a disk volume. The options are described in Table 8-1.

8.1.3.1 RD Command Format.

```
[] RD
RESTORE DIRECTORY
   SEQUENTIAL ACCESS NAME:
       DIRECTORY PATHNAME:
     LISTING ACCESS NAME:
                 OPTIONS: ADD
```

The meaning of the RD prompts is the same as in the BD command.

8.1.3.2 RD Command Example.

The following example shows the use of the the use of the RD command to copy the contents of the magnetic tape mounted on the device named MT01 to a directory on disk named .SAMPLE:

```
[] RD
RESTORE DIRECTORY
   SEQUENTIAL ACCESS NAME: MT01
       DIRECTORY PATHNAME: .SAMPLE
     LISTING ACCESS NAME:
                 OPTIONS: ADD
```

8.2 USE OF THE MODIFY VOLUME INFORMATION COMMAND

If a system disk has been backed up using any of the previously described commands, the Modify Volume Information (MVI) command must be used before an Initial Program Load (IPL) can be performed using the backup disk. This step is not required if the backup was made using the DCOPY command.

The following functions may be performed by the MVI command:

* L - List
* C - Change
* Q - Quit

The files and information that may be designated by the MVI command are:

* S - System Image
* O - System Overlay File
* P - System Program File

* L - System Loader File

* V - Volume Name

The user should respond to the MVI command as shown in the following example.

```
[] MVI

MODIFY VOLUME INFORMATION
CONTROL ACCESS NAME: ME→(SEQ FILE) IF FROM BATCH STREAM

MVI
DISK? DS02     'enter disk drive or volume name to be modifed)
COMMAND 'L/C/Q)? C
WHICH ITEM 'S,O,P,L,V)? S
PRIMARY: SYS1        'enter name of primary system)
SECONDARY
SELECT: P
COMMAND 'L/C/Q)? C
WHICH ITEM 'S,O,P,L,V)? O
PRIMARY: S$OVLYA     'entered by user)
SECONDARY:
SELECT: P
COMMAND 'L/C/Q)? C
WHICH ITEM 'S,O,P,L,V)? P
PRIMARY: S$PROGA     'entered by user)
SECONDARY:
SELECT: P
COMMAND 'L/C/Q)? C
WHICH ITEM 'S,O,P,L,V)? L
PRIMARY: S$LOADER    'entered by user)
SECONDARY:
SELECT: P
```

CHANGES VOL NAME

The user should verify the entries that were made by listing the information.

```
COMMAND 'L/C/Q)? L
                PRIMARY    SECONDARY    SELECT
 SYSTEM IMAGE: SYS1                        P
 PROGRAM FILE: S$PROGA                     P
 OVERLAY FILE: S$OVLYA                     P
  LOADER FILE: S$LOADER                    P
     WCS FILE:                             P
   DIAGNOSTIC:             --------        P
  VOLUME NAME: TICOBOL                     N
COMMAND 'L/C/Q)? Q
MVI TERMINATED:
```

8.3 CREATING SYSTEM FILES

If this backup is to be used as a system disk, two additional system files must be present on the disk. They are .S$ROLLA and .S$CRASH. These files will not be copied by any of the previous backup procedures that have been described. They must be created with the Create System Files (CSF) command. The CSF command appears as:

```
[] CSF

CREATE SYSTEM FILES
        VOLUME NAME:
        MEMORY SIZE: 64K
```

The user should supply the name of the backup volume and the amount of memory that is installed on his system. It is very important that these files be created to conform to the actual memory size.

8.4 USING DCOPY

The Disk Copy/Restore (DCOPY) command copies and optionally verifies disks used in the operating system. The copy is from disk to magnetic tape, magnetic tape to disk, or disk to disk. The copy is performed track by track with no disk compression. Disk Copy requires that the destination disk be error free and of the same type as the source disk when the copy is from disk to disk.

NOTE

One or more system files on the system disk are updated when DCOPY is terminated. Therefore, when making a copy of the system disk, verify the copy before terminating DCOPY.

Disk Copy/Restore is faster than Copy Directory, Backup Directory, and Restore Directory because no disk compression is performed.

The following is an example of a DCOPY operation which transfers data from disk to disk:

```
[] DCOPY

DISK COPY/RESTORE

ANSWER (Y/N) QUESTIONS WITH Y FOR YES
OR ANY OTHER CHARACTER EXCEPT $ FOR NO

RESPOND ANYTIME WITH $ TO RESTART

LISTING DEVICE NAME LP01
VERIFY ONLY? (Y/N) N
DEFAULT? (Y/N) N — use DEFAULT
COPY WITHOUT VERIFY? (Y/N) N
PAUSE ON ERROR? (Y/N) Y
FORCED WRITE AFTER READ ERROR (Y/N) Y
USE ADU MAPS FOR CONTROL? (Y/N) (Y) - DISK W/ BAD TRACKS
MASTER DEVICE DS02
     VOLUME ANYTHING
COPY DEVICE DS03
     VOLUME SCRATCH
MOUNT DESIRED VOLUMES. TYPE CR WHEN READY
COPY AND VERIFY COMPLETE
QUIT (Y/N) Y
SYSTEM DISK READY? (Y/N) Y
DISK COPY TERMINATED
```

For a more complete description of the DCOPY prompts, refer to Volume II of the DX10 reference manuals.

While DCOPY is generally faster that CD, BD, or RD, there are several advantages in using the directory utility commands instead of Disk Copy/Restore. Disk fragmentation can be eliminated if the copy is made with Copy Directory, Backup Directory and Restore Directory; no disk compression is done when DCOPY is used to make the copy. When the CD, BD, or RD command is being used to make a copy, other activities can be going on at the same time; no other activity may be taking place on the disks involved, while a Disk Copy/Restore is executing. DCOPY requires that the destination disk be of the same type as the source disk; Copy Directory does not have that requirement.

8.4.1 Backing Up a System Disk on Disk.

To backup a system on disk using Copy Directory, perform the following steps:

1. Place the backup disk in a secondary disk drive.

2. Use the Initialize New Volume (INV) command to initialize the disk in the secondary drive.

3. Use the Copy Directory (CD) command to copy the contents

of the system disk to the disk in the secondary drive.

4. Use the Create System Files (CSF) command to create the system roll and crash files on the disk in the secondary drive.

5. Use the Modify Volume Information (MVI) command to establish the name of the primary system, the system overlay file, program file, and loader file.

6. Use the Unload Volume (UV) command to unload the volume in the secondary disk drive.

7. Remove the backup disk from the secondary drive.

Optionally, the backup copy may be perform using the DCOPY command as as described previously.

8.4.2 Backing Up a System Disk on Tape.

The user who has only one disk drive faces a special problem in backing up the system: the system disk must be copied to another disk. This can be done using magnetic tape and the Disk Copy/Restore (DCOPY) command. The procedure to backup a disk with only one disk drive is as follows:

1. Mount the tape in the tape drive.

2. Press RESET, then LOAD on the tape drive. The READY light will come on when the tape is in position.

3. Use the DCOPY command to copy the system disk to magnetic tape.

4. Use the DCOPY command again to copy the magnetic tape to the backup disk. After the DCOPY command has been entered and the initial prompts have been answered, DCOPY responds with this message: 'MOUNT DESIRED VOLUMES. TYPE CR WHEN READY.' At this point, remove the system disk from the disk drive and mount the backup disk. Press RETURN when the backup disk is ready.

5. After the copy, DCOPY prompts with: 'SYSTEM DISK READY? (Y/N)'. Before answering this prompt, remove the backup disk and replace the system disk. Then enter a 'Y' in response to the prompt.

CAUTION

Responding with a Y to the 'SYSTEM DISK READY? (Y/N)' prompt with any disk in the drive other than the system disk used to begin

execution DCOPY may result in a system crash, destruction of data on the disk, or both.

To backup a system on magnetic tape using Backup Directory, perform the following steps:

1. Mount the backup tape in the tape drive.

2. Press RESET, then LOAD on the tape drive. The READY light comes on when the tape is in position.

3. Use the Backup Directory (BD) command to copy the contents of the system disk to the tape.

4. Use the Assign LUNO (AL) command to assign a LUNO to the magnetic tape drive.

5. Use the Rewind LUNO (RWL) command to rewind the tape.

6. Use the Verify Backup (VB) command to verify the copy, if desired.

7. Press UNLOAD on the tape drive.

8. Remove the tape from the tape drive.

NOTE

The operating system cannot be loaded directly from tape. A minimal system should be kept on disk to allow the Restore Directory command or the Disk Copy/Restore command to be used to copy the contents of the tape to disk. If the copy to tape was made with Backup Directory, the copy back to disk must be made using Restore Directory. If the copy to tape was made by the Disk Copy/Restore (DCOPY) command, the copy back to disk must be made by DCOPY.

8.4.3 Restoring a System from Magnetic Tape.

To restore the system disk using Restore Directory, perform the following steps:

1. Place the disk with the minimal system on it on DS01. Then initialize the system.

2. Mount the tape in the magnetic tape drive.

3. Press RESET, then LOAD on the tape drive. The READY light will come on when the tape is in position.

4. Place the backup system disk in a secondary disk drive.

5. Use the Initialize New Volume (INV) command to initialize the disk in the secondary drive.

6. Use the Restore Directory (RD) command to copy the contents of the magnetic tape to the backup disk in the secondary drive.

7. Use the Create System Files (CSF) command to create the system roll and crash files.

8. Use the Modify Volume Information (MVI) command to establish the name of the primary system, and the system overlay file, the program file, and the loader file.

9. Press UNLOAD on the tape drive.

10. Remove the tape from the tape drive.

8.4.4 Backing Up a Data Disk.

To make a backup copy of a data disk, follow the steps outlined above for backing up a system disk and omit using the Create System Files (CSF) command or the Modify Volume Information (MVI) command. The CSF and MVI commands need to be used only if the disk is going to be used as a system disk.

8.4.5 Verifying a Directory Copy.

The Verify Copy (VC) command can be used to verify a copy made by Copy Directory. The VC command compares a set of files under a master directory against a set of files under a copy directory to determine which files in each set match. The VC command detects matches by comparing the file type, file use, file name, and file contents of files at corresponding levels of each set of files. The results of the verify operation are listed at a device or are copied to a file specified by the user.

8.4.5.1 VC Command Example.

The following example shows the use of the VC command to compare the set of files under the directory named VOL1.MYFILES against the set of files under the directory named VOL1.HISFILES:

```
[] VC
VERIFY COPY
          MASTER PATHNAME:  VOL1.MYFILES
            COPY PATHNAME:  VOL1.HISFILES
     CONTROL ACCESS NAME:
     LISTING ACCESS NAME:  LP01
```

8.4.6 Verifying a Backup or Restore Copy.

The output of a Backup Directory (BD) command or Restore Directory (RD) command can be verified by using the Verify Backup (VB) command. The VB command compares a set of files on a sequential file or on magnetic tape to a set of files under a given directory on a disk file to see which files in each set match. Verify Backup detects matches by comparing the file type, file use, file name, and contents of files at a device or a file specified by the user. The results of the verify operation are listed at a device or copied to a file specified by the user.

8.4.6.1 VB Command Example.

The results of the following example are that the set of files under the directory named VOL1.MYFILES are compared against the set of files under the directory named VOL1.HISFILES. The summary result of the verify operation will be written to the line printer.

```
[] VB
VERIFY BACKUP
     SEQUENTIAL ACCESS NAME: VOL1.MYFILES
         DIRECTORY PATHNAME: VOL1.HISFILES
        LISTING ACCESS NAME: LP01
              MULTI-VOLUME?:
```

8.5 COBOL INSTALLATION

The COBOL disk COBOLINS contains both the object files and the batch stream necessary to install COBOL on DX10 version 3.X. A disk map of COBOLINS showing the contents of this disk, the batch stream listings, and problem notification are contained in the product documentation package.

. COBOLINS

The installation batch stream is stored as the file COBOLINS.INSTALL on the disk volume COBOLINS. In order to install the COBOL Compiler and runtime support on a system, install this disk, assign the synonym DSC and execute the batch stream. The synonym DSC must be assigned to the disk on which COBOL is to be installed.

1. Put the disk with volume name COBOLINS in disk drive DS02 on a functioning 3.X system and load it. Be sure to leave the write/protect on as you will not be writing out to the disk.

2. To install the disk, execute the command:
   IV UNIT=DS02, VOLUME=COBOLINS

3. To assign the synonym, execute the command:
   AS S=DSC, V=DS01

4. To execute the batch stream, execute the command:
   XB Input=COBOLINS.INSTALL, LIST=.LISTING

The IV command installs the volume COBOLINS on the secondary disk unit DS02. The Assign Synonym (AS) command causes COBOL to be installed on DS01. The XB command accepts commands from the file COBOLINS.INSTALL and executes them. These commands will:

1. Automatically delete any previously installed version of COBOL.

2. Install COBOL and its overlays.

3. Apply any patches required for proper execution of the processor on your system.

4. The output generated by the batch stream output will be placed in a file called .LISTING which may be examined with a Show File (SF) or Print File (PF) command to insure that COBOL was properly installed. A 0027 error on a DF command, a 285F or 3158 error on a DT command, a 0026 error on a CFDIR command, or a 0001 error on a RL command, are normal and should be ignored.

It will normally take 3 to 5 minutes for this batch stream to execute. During this time you may wish to check the status of the batch stream for completion and proper execution. Executing the SCI commands SBS or WAIT should be used for this purpose.

If you are installing COBOL from your system disk you must first copy the installation media to your system disk. Assign the sysnonym COBOLINS to the directory .COBOLINS and procede as previously outlined.

8.5.1 Removing COBOL Software from a System.

If you are generating a target system, it may be necessary to remove the COBOL software from the system disk. Executing the following commands will achieve this result. It is never necessary to have language software on a system that is not going to be used for development. If all of the applications have been linked and installed in program files, this is sufficient to execute the applications.

```
AS S=DSC, V=DS01

DT P=DSC.S$SDS$, T=COBOL
DT P=DSC.S$SDS$, T=RCOBOL
DP P=DSC.S$SDS$, T=RCOBOL
DP P=DSC.S$PROGA, P=RCOBOL
DF P=DSC.S$SYSLIB.RCBPRC
DF P=DSC.S$SYSLIB.RCBMPD
DF P=DSC.S$SYSLIB.RCBTSK
DD P=DSC.S$SYSLIB.S$SUBS
DF P=DSC.S$PROC.XCC
DF P=DSC.S$PROC.XCCF
DF P=DSC.S$PROC.XCP
DF P=DSC.S$PROC.XCPF
DF P=DSC.S$PROC.XCT
DF P=DSC.S$PROC.XCTF
```

The last four PROCs should not be deleted from .S$PROC if they will be called by any user-written PROCs.

8.5.2 Installing COBOL from Magnetic Tape.

In order to install COBOL from a magnetic tape, the user must first create a directory on either the system disk or a secondary disk. A Restore Directory (RD) is then issued to move the contents of the magnetic tape to the directory. The same sequence as previously described is then executed, except that the disk is already installed.

8.5.3 Verifying the Operation of COBOL.

To test COBOL, a sample program is provided on the disk. The program must be compiled and executed.

1. XCCF SOURCE=COBOLINS.TESTCASE, OBJECT=.TSTOBJ, LIST=ME

2. XCPF OBJECT=.TSTOBJ, DEBUG MODE?=NO

MODULE 9

DX5 COBOL

OBJECTIVES

* Describe the differences between DX10 COBOL and DX5 COBOL.
* Explain changes necessary to execute a DX10 COBOL application on a DX5 system.

9.1 INTRODUCTION

DX5 is an operating system which provides an applications oriented runtime only environment. No support is given for the development of COBOL application programs. DX5 COBOL is upwardly compatible with DX10 COBOL, with few or no modifications necessary. DX10 COBOL programs may be executed on DX5 systems if modifications are made to adapt them to the DX5 system requirements.

9.2 DIFFERENCES FROM DX10

There are three major differences between DX10 and DX5 COBOL.

1. DX5 is an unmapped system.

The memory available on systems running DX5 is 64kb. This memory is divided between the memory resident portion of the operating system, the user's application program, and the system overlay area. (Refer to Figure 9-1.) The maximum size of a COBOL program is limited by the amount of memory that is required by the memory resident portion of the operating system and the COBOL runtime. When you IPL the system, it responds with the amount of memory available to the user. Program size can be computed while developing on the DX10 system during the link edit phase. The user will then know if there is enough memory available to load the program, before actually trying to do so. If the amount of memory available is not enough to support a COBOL program, the user will have to incorporate programming techniques, such as overlays and segmentation, into his programs.

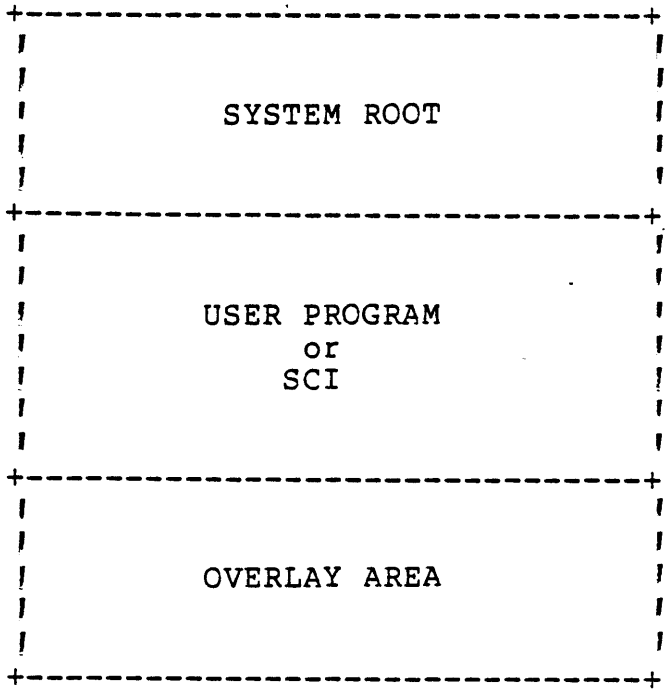


Figure 9-1 DX5 Memory

2. DX5 is a single-tasking operating system.

Since DX5 is a single-tasking operatins system, it does not support intertask communication. This capability requires a multi-tasking operating system. TIFORM990, SORT/MERGE, and DBMS990 all require intertask communication if these utilites are called under program control. Therefore these utilities cannot be executed from program control under DX5.

NOTE

SORT/MERGE cannot be called from within a program. SORT/MERGE is supported as a stand-alone task.

Because DX5 is a single-user station, the UNIT clause on the ACCEPT statement is not supported. All I/O is directed to the system console. COBOL programs that interface through the intertask communication area cannot be supported under DX5.

3. Program chaining under DX5 functions differently.

DX5 cannot support one task bidding another, with the bid task returning control to the calling task, once the bid task has terminated. Therefore, program chaining, using the C$CBID

subroutine, works differently on a DX5 system than on a DX10 system. Under DX5, the bid task cannot return control back to the bidding task. The bid task can, however, rebid the bidding task, placing it back into execution at the beginning of is executable code. Under DX5 the bidding task using the C$CBID subroutine must have a value of 8 through 12 in the FLAGS data item of the calling sequence, otherwise an error is returned.

9.3 DEVELOPMENT STEPS

DX5 is a non-development system. Development of DX5 COBOL application programs must be done on a DX10 system. The development steps are:

1. Develop source code using the DX10 Text Editor.
2. Compile the source code using the DX10 XCC or XCCF commands.
3. If the application program requires system routines, link the program using the DX5 libraries.
4. Use dual/sided, double/density, flexible diskettes to transport application to the DX5 system.

As can be seen, the development process is the same up to the linking step, as long as program size is not a factor.

9.3.1 Linking for DX5.

Because the output of the link editor will be transported to the DX5 system, the user must ensure that the files being transported have the correct blocking factor size. The easiest way to do this is create a directory with a 288 byte physical record size. All files to be moved to the DX5 system should be output to this directory.

NOTE

Make sure the pathnames being used on the DX5 system do not exceed the three level restriction.

To link the DX5 object code produced by the DX5 compiler the user must change three include statements in the link control stream. These are the three modules that make up the COBOL runtime. The three statements that must be modified are:

```
replace  .S$SYSLIB.RCBPRC   with  .DX5.S$SYSLIB.RCBPRC
replace  .S$SYSLIB.RCBTSK   with  .DX5.S$SYSLIB.RCBTSK
replace  .S$SYSLIB.RCBMPD   with  .DX5.S$SYSLIB.RCBMPD
```

NOTE

It is assumed that the DX5 COBOL object is installed on the DX10 system disk under the directory .DX5.

These three include statements will bring in the necessary DX5 system routines rather than the corresponding DX10 routines.

```
FORMAT IMAGE
PROC COBOLRTM
INCLUDE .DX5.S$SYSLIB.RCBPRC
TASK CBLTASK1
INCLUDE .DX5.S$SYSLIB.RCBTSK
INCLUDE .DX5.S$SYSLIB.RCBMPD
INCLUDE TI.MPROG1
INCLUDE TI.SPROG1
END
```

Figure 9-2

If a LOAD command is used in the link control stream, you must be sure to include a LIBRARY directive for the DX5 system library. The LOAD command will then pull in the DX5 overlay manager from the DX5 system library.

```
FORMAT IMAGE
PROC COBOLRTM
LIBRARY .DX5.S$SYSLIB
     .
     .
     .
LOAD
     .
     .
     .
```

Figure 9-3

9.4 DX5 COBOL EXECUTION

For user convience, the same commands used to execute COBOL tasks and programs under DX10 are used with DX5. Because the DX5 system is a single-tasking system, executing with the XCT/XCTF or XCP/XCPF commands has basically the same effect. Executing either in foreground or background mode places the task into execution in a manner similar to the foreground mode of the DX10 system. When the XCT or XCTF command is entered the system will prompt for TASK ID or NAME. The user must enter the TASK ID for the task to be executed. DX5 does not support mapping the name to the ID.

9.5 MODIFYING DX10 PROGRAMS TO RUN UNDER DX5

There may be a few changes that need to be made. Most of these changes are a result of the differences between DX10 and DX5 COBOL. The areas that need to be addressed are:

* If TIFORM990 or DBMS990 are called from the COBOL program they must be removed. DX5 does not support either of these utilities.

* If SORT/MERGE is called from a program, the call must be removed and the sort operation performed external to the program.

* If program chaining is being performed, the method of chaining must be examined to see which must be performed; modification or deletion.

  - If a nonreturning (to the calling program) chain is being performed, then simply modify the FLAGS in the data item of the calling sequence.

  - If return to the calling program (after execution of the called program terminates) is to be performed, then this application must be redesigned to fit the single-tasking environment of DX5.

  - After the necessary modifications have been made, the user should recompile the source code then estimate the program size to ensure it will fit in the memory available on the DX5 system.

* The user should create directories that will correspond to the blocking buffers established during the DX5 sysgen. Usually these will be 288 bytes.

The user should now relink the object code produced by the compiler, making sure to use the DX5 include statements in the link control stream. Next, copy the created directory containing the generated files to media compatible with the DX5 system. Finally, execute the COBOL application on the DX5 system.

APPENDIX A

INVENTORY SUBROUTINES

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  RDINV.
* *
*  THIS SUBROUTINE IS USED TO READ THE INVENTORY FILE.
* *
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  TI-990-10.
 OBJECT-COMPUTER.  TI-990-10.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT STOCK-FILE, ASSIGN TO RANDOM, "PIF";
         ORGANIZATION IS INDEXED;
         ACCESS IS DYNAMIC;
         RECORD KEY IS PART;
         ALTERNATE RECORD KEY IS DESC WITH DUPLICATES;
         FILE STATUS IS FILE-STATUS.
 DATA DIVISION.
 FILE SECTION.
 FD  STOCK-FILE
     LABEL RECORD IS STANDARD.
 01  STOCK-RECORD.
     03  PART                  PIC X(5).
     03  DESC                  PIC X(20).
     03  COST                  PIC 999V99.
     03  QUANTITY-ON-HAND      PIC 9(5).
     03  REORDER-LEVEL         PIC 9(5).
 WORKING-STORAGE SECTION.
 01  FILE-STATUS               PIC XX.
 01  TAG                       PIC X.
 01  BLANK-LINE                PIC X(80) VALUE SPACES.
 LINKAGE SECTION.
 01  PART-NO                   PIC 9(5).
 01  DESCRIPTION               PIC X(20).
 01  STATIS                    PIC 9.
 PROCEDURE DIVISION USING PART-NO, DESCRIPTION, STATIS.
 DECLARATIVES.
 FILE-ERRORS SECTION 0.
     USE AFTER ERROR PROCEDURE ON STOCK-FILE.
 CHECK-ERRORS.
     IF FILE-STATUS = 99 DISPLAY "RECORD UNAVAILABLE "
         LINE 23, ACCEPT TAG POSITION 0, GO TO READ-RECORD.
     IF FILE-STATUS = 23 DISPLAY "INVALID PART NUMBER "
             LINE 23,
         ELSE DISPLAY "ERROR -- FILE STATUS = "
             LINE 23, DISPLAY FILE-STATUS POSITION 0.
     ACCEPT TAG POSITION 0.
     CLOSE STOCK-FILE.
 EXIT-ERRORS.
     EXIT PROGRAM.
 END DECLARATIVES.
 MAIN-ROUTINE SECTION 1.
 SET-KEY.
     MOVE 1 TO STATIS.
     MOVE SPACES TO DESCRIPTION.
```

```
     OPEN INPUT STOCK-FILE.
     MOVE PART-NO TO PART.
 READ-RECORD.
     DISPLAY BLANK-LINE LINE 23.
     READ STOCK-FILE RECORD KEY IS PART.
     MOVE 0 TO STATIS.
     MOVE DESC TO DESCRIPTION.
     CLOSE STOCK-FILE.
 EXIT-ROUTINE.
     EXIT PROGRAM.




 IDENTIFICATION DIVISION.
 PROGRAM-ID.  RECPT.
* *
*  THIS SUBROUTINE ADDS TO THE QUANTITY ON HAND.
* *
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  TI-990-10.
 OBJECT-COMPUTER.  TI-990-10.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT STOCK-FILE, ASSIGN TO RANDOM, "PIF";
         ORGANIZATION IS INDEXED;
         ACCESS IS DYNAMIC;
         RECORD KEY IS PART;
         ALTERNATE RECORD KEY IS DESC WITH DUPLICATES;
         FILE STATUS IS FILE-STATUS.
 DATA DIVISION.
 FILE SECTION.
 FD  STOCK-FILE
     LABEL RECORD IS STANDARD.
 01  STOCK-RECORD.
     03  PART                 PIC X(5).
     03  DESC                 PIC X(20).
     03  COST                 PIC 999V99.
     03  QUANTITY-ON-HAND     PIC 9(5).
     03  REORDER-LEVEL        PIC 9(5).
 WORKING-STORAGE SECTION.
 01  FILE-STATUS              PIC XX.
 01  TAG                      PIC X.
 01  BLANK-LINE               PIC X(80) VALUE SPACES.
 LINKAGE SECTION.
 01  PART-NO                  PIC 9(5).
 01  QUANTITY                 PIC 9(5).
 01  STATIS                   PIC 9.
 PROCEDURE DIVISION USING PART-NO, QUANTITY, STATIS.
 DECLARATIVES.
 FILE-ERRORS SECTION 0.
     USE AFTER ERROR PROCEDURE ON STOCK-FILE.
```

```
 CHECK-ERRORS.
     IF FILE-STATUS = 99 DISPLAY "RECORD UNAVAILABLE "
         LINE 23, ELSE MOVE 0 TO TAG.
     IF TAG = 1 GO TO READ-RECORD.
     IF TAG = 2 GO TO UPDATE-RECORD.
     IF FILE-STATUS = 23 DISPLAY "INVALID PART NUMBER "
             LINE 23,
         ELSE DISPLAY "ERROR -- FILE STATUS = "
             LINE 23, DISPLAY FILE-STATUS POSITION 0.
     ACCEPT TAG POSITION 0.
     CLOSE STOCK-FILE.
 EXIT-ERRORS.
     EXIT PROGRAM.
 END DECLARATIVES.
 MAIN-ROUTINE SECTION 1.
 SET-KEY.
     MOVE 1 TO STATIS.
     OPEN I-O STOCK-FILE.
     MOVE PART-NO TO PART.
 READ-RECORD.
     DISPLAY BLANK-LINE LINE 23.
     MOVE 1 TO TAG.
     READ STOCK-FILE RECORD KEY IS PART.
     MOVE 0 TO STATIS.
     ADD QUANTITY TO QUANTITY-ON-HAND.
 UPDATE-RECORD.
     DISPLAY BLANK-LINE LINE 23.
     MOVE 2 TO TAG.
     REWRITE STOCK-RECORD.
     DISPLAY "Display Stock Status? (Y/N) "
         LINE 23.
     ACCEPT TAG POSITION 0.
     DISPLAY BLANK-LINE LINE 23.
     IF TAG = "Y" CALL "SSTAT" USING PART-NO.
     CLOSE STOCK-FILE.
 EXIT-ROUTINE.
     EXIT PROGRAM.
```

```
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  ISSUE.
* *
*  THIS SUBROUTINE ISSUES STOCK IF THE QUANTITY ON HAND
*    IS AT LEAST AS LARGE AS THE QUANTITY REQUESTED.
* *
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  TI-990-10.
 OBJECT-COMPUTER.  TI-990-10.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
```

```
    SELECT STOCK-FILE, ASSIGN TO RANDOM, "PIF";
        ORGANIZATION IS INDEXED;
        ACCESS IS DYNAMIC;
        RECORD KEY IS PART;
        ALTERNATE RECORD KEY IS DESC WITH DUPLICATES;
        FILE STATUS IS FILE-STATUS.
DATA DIVISION.
FILE SECTION.
FD  STOCK-FILE
    LABEL RECORD IS STANDARD.
01  STOCK-RECORD.
    03  PART                  PIC X(5).
    03  DESC                  PIC X(20).
    03  COST                  PIC 999V99.
    03  QUANTITY-ON-HAND      PIC 9(5).
    03  REORDER-LEVEL         PIC 9(5).
WORKING-STORAGE SECTION.
01  FILE-STATUS               PIC XX.
01  TAG                       PIC X.
01  BLANK-LINE                PIC X(80) VALUE SPACES.
LINKAGE SECTION.
01  PART-NO                   PIC 9(5).
01  QUANTITY                  PIC 9(5).
01  STATIS                    PIC 9.
PROCEDURE DIVISION USING PART-NO, QUANTITY, STATIS.
DECLARATIVES.
FILE-ERRORS SECTION 0.
    USE AFTER ERROR PROCEDURE ON STOCK-FILE.
CHECK-ERRORS.
    IF FILE-STATUS = 99 DISPLAY "RECORD UNAVAILABLE "
        LINE 23, ELSE MOVE 0 TO TAG.
    IF TAG = 1 GO TO READ-RECORD.
    IF TAG = 2 GO TO UPDATE-RECORD.
    IF FILE-STATUS = 23 DISPLAY "INVALID PART NUMBER "
            LINE 23,
        ELSE DISPLAY "ERROR -- FILE STATUS = "
            LINE 23, DISPLAY FILE-STATUS POSITION 0.
    ACCEPT TAG POSITION 0.
    CLOSE STOCK-FILE.
EXIT-ERRORS.
    EXIT PROGRAM.
END DECLARATIVES.
MAIN-ROUTINE SECTION 1.
SET-KEY.
    MOVE 1 TO STATIS.
    OPEN I-O STOCK-FILE.
    MOVE PART-NO TO PART.
READ-RECORD.
    DISPLAY BLANK-LINE LINE 23.
    MOVE 1 TO TAG.
    READ STOCK-FILE RECORD KEY IS PART.
    IF QUANTITY-ON-HAND < QUANTITY GO TO DISPLAY-STATUS.
    MOVE 0 TO STATIS.
    SUBTRACT QUANTITY FROM QUANTITY-ON-HAND.
UPDATE-RECORD.
```

```
    DISPLAY BLANK-LINE LINE 23.
    MOVE 2 TO TAG.
    REWRITE STOCK-RECORD.
DISPLAY-STATUS.
    DISPLAY "Display Stock Status? (Y/N) "
        LINE 23.
    ACCEPT TAG POSITION 0.
    DISPLAY BLANK-LINE LINE 23.
    CLOSE STOCK-FILE.
    IF TAG = "Y" CALL "SSTAT" USING PART-NO.
EXIT-ROUTINE.
    EXIT PROGRAM.




IDENTIFICATION DIVISION.
PROGRAM-ID.  REORD.
* *
*  THIS SUBROUTINE PRINTS A LISTING OF ALL ITEMS WHERE
*    THE QUANTITY ON HAND IS LESS THAN OR EQUAL TO THE
*    REORDER LEVEL.
* *
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  TI-990-10.
OBJECT-COMPUTER.  TI-990-10.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT STOCK-FILE ASSIGN TO RANDOM, "PIF";
        ORGANIZATION IS INDEXED;
        ACCESS IS SEQUENTIAL;
        RECORD KEY IS PART;
        ALTERNATE RECORD KEY IS DESC WITH DUPLICATES;
        FILE STATUS IS FILE-STATUS.
    SELECT PRINT-FILE ASSIGN TO PRINT, "RR"
        ORGANIZATION IS SEQUENTIAL;
        ACCESS IS SEQUENTIAL;
        FILE STATUS IS PRINT-STATUS.
DATA DIVISION.
FILE SECTION.
FD  STOCK-FILE
    LABEL RECORD IS STANDARD.
01  STOCK-RECORD.
    03  PART                PIC X(5).
    03  DESC                PIC X(20).
    03  COST                PIC 999V99.
    03  QUANTITY-ON-HAND    PIC 9(5).
    03  REORDER-LEVEL       PIC 9(5).
FD  PRINT-FILE
    LABEL RECORD IS OMITTED.
01  PRINT-RECORD            PIC X(80).
WORKING-STORAGE SECTION.
```

```
01  FILE-STATUS          PIC XX.
01  PRINT-STATUS         PIC XX.
01  TAG                  PIC X.
01  BLANK-LINE           PIC X(80) VALUE SPACES.
01  FLAG                 PIC 9.
01  DATE-IN.
    03  YEAR             PIC 99.
    03  MONTH            PIC 99.
    03  DAYS             PIC 99.
01  1ST-HEADING.
    03  FILLER           PIC X(31) VALUE SPACES.
    03  FILLER           PIC X(17) VALUE "I N V E N T O R Y".
    03  FILLER           PIC X(32) VALUE SPACES.
01  2ND-HEADING.
    03  FILLER           PIC X(33) VALUE SPACES.
    03  FILLER           PIC X(14) VALUE "REORDER REPORT".
    03  FILLER           PIC X(13) VALUE SPACES.
    03  DATE-O.
        05  MONTH-O      PIC Z9.
        05  FILLER       PIC X VALUE "/".
        05  DAYS-O       PIC 99.
        05  FILLER       PIC X VALUE "/".
        05  YEAR-O       PIC 99.
    03  FILLER           PIC X(12) VALUE SPACES.
01  3RD-HEADING.
    03  FILLER           PIC X(7) VALUE SPACES.
    03  FILLER           PIC X(4) VALUE "PART".
    03  FILLER           PIC X(6) VALUE SPACES.
    03  FILLER           PIC X(11) VALUE "DESCRIPTION".
    03  FILLER           PIC X(15) VALUE SPACES.
    03  FILLER           PIC X(4) VALUE "COST".
    03  FILLER           PIC X(5) VALUE SPACES.
    03  FILLER           PIC X(8) VALUE "QUANTITY".
    03  FILLER           PIC X(5) VALUE SPACES.
    03  FILLER           PIC X(7) VALUE "REORDER".
    03  FILLER           PIC X(8) VALUE SPACES.
01  4TH-HEADING.
    03  FILLER           PIC X(6) VALUE SPACES.
    03  FILLER           PIC X(6) VALUE "NUMBER".
    03  FILLER           PIC X(41) VALUE SPACES.
    03  FILLER           PIC X(7) VALUE "ON HAND".
    03  FILLER           PIC X(6) VALUE SPACES.
    03  FILLER           PIC X(5) VALUE "LEVEL".
    03  FILLER           PIC X(9) VALUE SPACES.
01  DETAIL-LINE.
    03  FILLER           PIC X(7) VALUE SPACES.
    03  PART-O           PIC 9(5).
    03  FILLER           PIC X(5) VALUE SPACES.
    03  DESC-O           PIC X(20).
    03  FILLER           PIC X(4) VALUE SPACES.
    03  COST-O           PIC ZZZ.99.
    03  FILLER           PIC X(6) VALUE SPACES.
    03  QUANTITY-ON-HAND-O  PIC ZZ,ZZ9.
    03  FILLER           PIC X(6) VALUE SPACES.
    03  REORDER-LEVEL-O     PIC ZZ,ZZ9.
```

```
    03  FILLER      PIC X(9) VALUE SPACES.
PROCEDURE DIVISION.
DECLARATIVES.
STOCK-ERRORS SECTION 0.
    USE AFTER ERROR PROCEDURE ON STOCK-FILE.
CHECK-STOCK.
    IF FILE-STATUS = 99 DISPLAY "RECORD UNAVAILABLE"
        LINE 23, ACCEPT TAG POSITION 0, GO TO READ-RECORD,
    ELSE DISPLAY "ERROR -- FILE STATUS = "
        LINE 23, DISPLAY FILE-STATUS POSITION 0.
    ACCEPT TAG POSITION 0.
    CLOSE STOCK-FILE.
    IF FLAG = 1 CLOSE PRINT-FILE.
    GO TO EXIT-ROUTINE.
PRINT-ERRORS SECTION 1.
    USE AFTER ERROR PROCEDURE ON PRINT-FILE.
CHECK-PRINT.
    IF PRINT-STATUS = 93 DISPLAY "DEVICE NOT AVAILABLE"
        LINE 23,
    ELSE DISPLAY "ERROR -- PRINT STATUS = "
        LINE 23, DISPLAY PRINT-STATUS POSITION 0.
    ACCEPT TAG POSITION 0.
    CLOSE STOCK-FILE, PRINT-FILE.
    GO TO EXIT-ROUTINE.
END DECLARATIVES.
MAIN-ROUTINE SECTION 3.
OPEN-FILES.
    OPEN INPUT STOCK-FILE.
    OPEN OUTPUT PRINT-FILE.
    MOVE 1 TO FLAG.
    MOVE LOW-VALUES TO DESC.
    START STOCK-FILE KEY > DESC.
    ACCEPT DATE-IN FROM DATE.
    MOVE MONTH TO MONTH-O.
    MOVE DAYS TO DAYS-O.
    MOVE YEAR TO YEAR-O.
PRINT-HEADINGS.
    WRITE PRINT-RECORD FROM 1ST-HEADING AFTER 2 LINES.
    WRITE PRINT-RECORD FROM 2ND-HEADING AFTER 1 LINE.
    WRITE PRINT-RECORD FROM BLANK-LINE AFTER 1 LINE.
    WRITE PRINT-RECORD FROM 3RD-HEADING AFTER 1 LINE.
    WRITE PRINT-RECORD FROM 4TH-HEADING AFTER 1 LINE.
    WRITE PRINT-RECORD FROM BLANK-LINE AFTER 1 LINE.
READ-RECORD.
    READ STOCK-FILE NEXT RECORD WITH NO LOCK
        AT END CLOSE STOCK-FILE, PRINT-FILE,
            GO TO EXIT-ROUTINE.
    IF REORDER-LEVEL < QUANTITY-ON-HAND GO TO READ-RECORD.
    MOVE PART TO PART-O.
    MOVE DESC TO DESC-O.
    MOVE COST TO COST-O.
    MOVE QUANTITY-ON-HAND TO QUANTITY-ON-HAND-O.
    MOVE REORDER-LEVEL TO REORDER-LEVEL-O.
    WRITE PRINT-RECORD FROM DETAIL-LINE AFTER 1 LINE.
    GO TO READ-RECORD.
```

```
 EXIT-ROUTINE.
     EXIT PROGRAM.



 IDENTIFICATION DIVISION.
 PROGRAM-ID.  SSTAT.
* *
*  THIS SUBROUTINE DISPLAYS THE STATUS OF A PART NUMBER.
* *
 ENVIRONMENT DIVISION.
 CONFIGURATION SECTION.
 SOURCE-COMPUTER.  TI-990-10.
 OBJECT-COMPUTER.  TI-990-10.
 INPUT-OUTPUT SECTION.
 FILE-CONTROL.
     SELECT STOCK-FILE, ASSIGN TO RANDOM, "PIF";
         ORGANIZATION IS INDEXED;
         ACCESS IS DYNAMIC;
         RECORD KEY IS PART;
         ALTERNATE RECORD KEY IS DESC WITH DUPLICATES;
         FILE STATUS IS FILE-STATUS.
 DATA DIVISION.
 FILE SECTION.
 FD  STOCK-FILE
     LABEL RECORD IS STANDARD.
 01  STOCK-RECORD.
     03  PART                  PIC X(5).
     03  DESC                  PIC X(20).
     03  COST                  PIC 999V99.
     03  QUANTITY-ON-HAND      PIC 9(5).
     03  REORDER-LEVEL         PIC 9(5).
 WORKING-STORAGE SECTION.
 01  FILE-STATUS               PIC XX.
 01  TAG                       PIC X.
 01  COST-D                    PIC ZZZ.99.
 01  QUANTITY-ON-HAND-D        PIC ZZ,ZZ9.
 01  REORDER-LEVEL-D           PIC ZZ,ZZ9.
 01  BLANK-LINE                PIC X(80) VALUE SPACES.
 01  N                         PIC 99.
 LINKAGE SECTION.
 01  PART-NO                   PIC 9(5).
 PROCEDURE DIVISION USING PART-NO.
 DECLARATIVES.
 FILE-ERRORS SECTION 0.
     USE AFTER ERROR PROCEDURE ON STOCK-FILE.
 CHECK-ERRORS.
     IF FILE-STATUS = 99 DISPLAY "RECORD UNAVAILABLE "
         LINE 23, ACCEPT TAG POSITION 0, GO TO READ-RECORD.
     IF FILE-STATUS = 23 DISPLAY "INVALID PART NUMBER "
             LINE 23,
         ELSE DISPLAY "ERROR -- FILE STATUS = "
```

```
            LINE 23.
    ACCEPT TAG POSITION 0.
EXIT-ERROR.
    EXIT PROGRAM.
END DECLARATIVES.
MAIN-ROUTINE SECTION 1.
SET-KEY.
    OPEN INPUT STOCK-FILE.
    MOVE PART-NO TO PART.
READ-RECORD.
    DISPLAY BLANK-LINE LINE 23.
    READ STOCK-FILE RECORD KEY IS PART.
    PERFORM DISPLAY-STATUS THRU DS-EXIT.
    CLOSE STOCK-FILE.
EXIT-ROUTINE.
    EXIT PROGRAM.
DISPLAY-STATUS.
    MOVE COST TO COST-D.
    MOVE QUANTITY-ON-HAND TO QUANTITY-ON-HAND-D.
    MOVE REORDER-LEVEL TO REORDER-LEVEL-D.
    DISPLAY "+---------------------------------------+"
        LINE 16, POSITION 19.
    PERFORM BOUNDARY THRU B-EXIT
        VARYING N FROM 17 BY 1 UNTIL N = 24.
    DISPLAY "+---------------------------------------+"
        LINE 24, POSITION 19.
    DISPLAY "STOCK STATUS" LINE 17, POSITION 35.
    DISPLAY "PART NUMBER" LINE 19, POSITION 22.
    DISPLAY "DESCRIPTION" POSITION 22.
    DISPLAY "COST" POSITION 22.
    DISPLAY "QUANTITY ON HAND" POSITION 22.
    DISPLAY "REORDER LEVEL" POSITION 22.
    DISPLAY PART LINE 19 POSITION 40.
    DISPLAY DESC POSITION 40.
    DISPLAY COST-D POSITION 40.
    DISPLAY QUANTITY-ON-HAND-D POSITION 40.
    DISPLAY REORDER-LEVEL-D POSITION 40.
    ACCEPT TAG.
    PERFORM CLEAR-SCREEN THRU CS-EXIT
        VARYING N FROM 16 BY 1 UNTIL N = 25.
DS-EXIT.  EXIT.
BOUNDARY.
    DISPLAY " !" LINE N, POSITION 19.
    DISPLAY " !" LINE N, POSITION 62.
B-EXIT.  EXIT.
CLEAR-SCREEN.
    DISPLAY BLANK-LINE LINE N.
CS-EXIT.  EXIT.
```

APPENDIX B

SAMPLE SYSGEN DIALOG

```
================== GEN990-AUTO SYSGEN   RELEASE 3.3.0   ==================
DATA DISC:  'DS01)
TARGET DISK:  'DS01)
INPUT:
OUTPUT: SYS1

LINE:  '60)
TIME SLICING ENABLED?  'YES)
TIME SLICE VALUE:  '1)
TASK SENTRY ENABLED?  'NO)

TABLE: ?
WHAT LENGTH IN WORDS DO YOU WANT THE SYSTEM TABLE TO BE: ?
THE SYSTEM TABLE AREA IS A MEMORY RESIDENT, RESERVED AREA THAT IS USED
TO MAINTAIN SYSTEM DATA STRUCTURES.  THE SIZE OF THIS AREA IS HIGHLY
DEPENDENT ON THE SYSTEM CONFIGURATION.  A GUIDELINE TO FOLLOW WHEN
SPECIFING THIS AREA IS AS FOLLOWS:
                      SYSTEM TABLE SIZING
       FEATURE                              ADDITIONAL WORDS
   BASE SYSTEM AREA *                          1100 WORDS
   ACTIVE FOREGROUND SCI **                     300 WORDS
   ACTIVE BACKGROUND SCI **                     350 WORDS
   ADDITIONAL PER INSTALLED DISK                200 WORDS
>>> ENTER ANY KEY TO CONTINUE <<<

     *  SYSTEM AREA FOR A SINGLE TERMINAL SYSTEM; WITH 1 FOREGROUND
        AND 1 BACKGROUND SCI ALLOWED; AND ONLY THE SYSTEM DISK INSTALLED
    **  THESE ESTIMATES ARE FOR NOMINAL LOADING.

GEN990 PROVIDES NO DEFAULT FOR THIS PARAMETER:
TABLE: 5K

COMMON:  'NONE)
INTERRUPT DECODER:  'NONE)
FILE MANAGEMENT TASKS:  '2)
CLOCK:  '5)
ID:  'NONE)
OVERLAYS:  '2)
SYSLOG:  '6)
BUFFER MANAGEMENT:  '1K)
I/O BUFFERS:  '0)
INTERTASK:  '100)  512
KIF?  'YES)
SEQUENTIAL PLACEMENT?:  'YES)
COUNTRY CODE:  'US)
POWERFAIL:  'NO)
SCI BACKGROUND:  '2)
SCI FOREGROUND:  '8) 10
BREAKPOINT:  '16)
CARD 1: 10
CARD 2:

DEVICE: K820
CRU:  '>00)
ACCESS TYPE:  'RECORD)
```

```
TIME OUT:  '0)
CHARACTER QUEUE:  '6)
INTERRUPT:  '6)

DEVICE: VDT
CRU:  '>100)  0580
ACCESS TYPE:  'RECORD)
TIME OUT:  '0)
CRT TYPE:  '911)
3270 CRU ADDRESS:  'NONE)
CHARACTER QUEUE:  '6)
INTERRUPT:  '10)
EXPANSION CHASSIS:  '1)
EXPANSION POSITION: 7

DEVICE: VDT
CRU:  '>100)  05A0
ACCESS TYPE:  'RECORD)
TIME OUT:  '0)
CRT TYPE:  '911)
3270 CRU ADDRESS:  'NONE)
CHARACTER QUEUE:  '6)
INTERRUPT:  '10)
EXPANSION CHASSIS:  '1)
EXPANSION POSITION: 14

DEVICE: VDT
CRU:  '>100)  0480
ACCESS TYPE:  'RECORD)
TIME OUT:  '0)
CRT TYPE:  '911)
3270 CRU ADDRESS:  'NONE)
CHARACTER QUEUE:  '6)
INTERRUPT:  '10)
EXPANSION CHASSIS:  '1)
EXPANSION POSITION: 3

DEVICE: VDT
CRU:  '>100)  04A0
ACCESS TYPE:  'RECORD)
TIME OUT:  '0)
CRT TYPE:  '911)
3270 CRU ADDRESS:  'NONE)
CHARACTER QUEUE:  '6)
INTERRUPT:  '10)
EXPANSION CHASSIS:  '1)
EXPANSION POSITION: 11

DEVICE: VDT
CRU:  '>100)  05C0
ACCESS TYPE:  'RECORD)
TIME OUT:  '0)
CRT TYPE:  '911)
3270 CRU ADDRESS:  'NONE)
CHARACTER QUEUE:  '6)
```

```
INTERRUPT: ‹10)
EXPANSION CHASSIS: ‹1)
EXPANSION POSITION: 8

DEVICE: VDT
CRU: ‹<100) 05E0
ACCESS TYPE: ‹RECORD)
TIME OUT: ‹0)
CRT TYPE: ‹911)
3270 CRU ADDRESS: ‹NONE)
CHARACTER QUEUE: ‹6)
INTERRUPT: ‹10)
EXPANSION CHASSIS: ‹1)
EXPANSION POSITION: 9

DEVICE: VDT
CRU: ‹>100) 04C0
ACCESS TYPE: ‹RECORD)
TIME OUT: ‹0)
CRT TYPE: ‹911)
3270 CRU ADDRESS: ‹NONE)
CHARACTER QUEUE: ‹6)
INTERRUPT: ‹10)
EXPANSION CHASSIS: ‹1)
EXPANSION POSITION: 4

DEVICE: VDT
CRU: ‹>100) 04E0
ACCESS TYPE: ‹RECORD)
TIME OUT: ‹0)
CRT TYPE: ‹911)
3270 CRU ADDRESS: ‹NONE)
CHARACTER QUEUE: ‹6)
INTERRUPT: ‹10)
EXPANSION CHASSIS: ‹1)
EXPANSION POSITION: 10

DEVICE: K820
CRU: ‹>100) 0440
ACCESS TYPE: ‹RECORD)
TIME OUT: ‹0)
CHARACTER QUEUE: ‹6)
INTERRUPT: ‹6) 10
EXPANSION CHASSIS: ‹1)
EXPANSION POSITION: 2

DEVICE: DS
TILINE: ‹F800)
DRIVES: ‹1) 2
DEFAULT RECORD SIZE: ‹864)
INTERRUPT: ‹13) 15

DEVICE: DS
TILINE: ‹F800) 0F820
DRIVES: ‹1)
```

```
DEFAULT RECORD SIZE:  •864) 288
INTERRUPT:  •13) 7

DEVICE: MT
TILINE:  •F800)
DRIVES:  •1)
INTERRUPT:  •9) 12

DEVICE: LP
CRU:  •>60)
ACCESS TYPE:  •FILE)
TIME OUT:  •30)
PRINT MODE:  •SERIAL)
EXTENDED?  •NO) Y
3270 CRU ADDRESS:  •NONE)
INTERRUPT:  •14)

DEVICE: LP
CRU:  •>60) 0460
ACCESS TYPE:  •FILE)
TIME OUT:  •30)
PRINT MODE:  •SERIAL)
EXTENDED?  •NO)  Y
3270 CRU ADDRESS:  •NONE)
INTERRUPT:  •14) 10
EXPANSION CHASSIS:  •1)
EXPANSION POSITION: 15

DEVICE: CM
CRU:  •>140) 020
COMM PACKAGE:  •3780)
BUFFER SIZE:  •0) 512
INTERRUPT:  •8) 6

DEVICE: SD
CRU:  •>20) 0500
INTERRUPT CRU BIT:  •NA)
NAME: TM
KSB:  •NONE)
DSR WORKSPACE: PDTM01
INTERRUPT ENTRY: TMINT2
PDT FILE: .S$SYSGEN.PDT$TM01
DSR FILE: .S$SYSGEN.DSR$TM2
OVERRIDE?  •YES)
INTERRUPT:  •15) 10
EXPANSION CHASSIS:  •1)
EXPANSION POSITION: 12

DEVICE: SD
CRU:  •>20) 0540
INTERRUPT CRU BIT:  •NA)
NAME: EM
KSB:  •NONE)
DSR WORKSPASE: PDEM01
INTERRUPT ENTRY: EMINT2
```

```
PDT FILE: .S$SYSGEN.PDTEM01
DSR FILE: .S$SYSGEN.DSR$EM2
OVERRIDE? (YES)
INTERRUPT: (15) 10
EXPANSION CHASSIS: (1)
EXPANSION POSITION: 6

DEVICE:

SVC:
XOP:

CONFIGURATION FILE IS COMPLETE.  DO YOU WANT TO SAVE IT? (YES)
**********  CONFIGURATION FILE SAVED  **********
********  D$DATA SOURCE FILE IS NOW BEING BUILT  **********
******  THE LINK EDIT COMMAND STREAM SOURCE FILE IS BEING BUILT  ******
********  BATCH FILE FOR SYSGEN COMPLETION IS NOW BEING BUILT *******
DO YOU NEED INSTRUCTIONS TO COMPLETE THE SYSGEN?  N
*********  GEN990 TERMINATED  **********
```

APPENDIX C

SAMPLE SOLUTIONS

MODULE 1

Worksheet 1

WITHOUT SHARED PROCEDURES

	TASK1	TASK2	TASK3
P1	10K	10K	10K
P21	15K	---	---
P22	---	10K	10K
TASK	15K	20K	10K
TOTAL	40K	40K	30K

TOTAL is 110K

WITH SHARED PROCEDURES

	TASK1	TASK2	TASK3
P1	10K	---	---
P21	15K	---	---
P22	---	10K	---
TASK	15K	20K	10K
TOTAL	40K	30K	10K

TOTAL is 80K

Savings is 30K

Worksheet 2

```
LIBRARY     TI.COBOL.OBJ
PROCEDURE   RCOBOL
DUMMY
INCLUDE     .S$SYSLIB.RCBPRC
PROCEDURE   ULIB
INCLUDE      *INSUB)
INCLUDE      *ADDNUM)
INCLUDE      *OUTSUB)
TASK        TASK1
INCLUDE     .S$SYSLIB.RCBTSK
ALLOCATE
INCLUDE     .S$SYSLIB.RCBMPD
INCLUDE      *PROG1)
END


LIBRARY     TI.COBOL.OBJ
PROCEDURE   RCOBOL
DUMMY
INCLUDE     .S$SYSLIB.RCBPRC
PROCEDURE   ULIB
DUMMY
INCLUDE      *INSUB)
INCLUDE      *ADDNUM)
INCLUDE      *OUTSUB)
TASK        TASK2
INCLUDE     .S$SYSLIB.RCBTSK
ALLOCATE
INCLUDE     .S$SYSLIB.RCBMPD
INCLUDE      *PROG2)
END
```

MODULE 2

Worksheet 1

```
LIBRARY     .S$SYSLIB
LIBRARY     TI.COBOL.OBJ
PROCEDURE   RCOBOL
DUMMY
INCLUDE      'RCBPRC)
PHASE 0,    ROOT
INCLUDE      'RCBTSK)
LOAD
INCLUDE      'RCBMPD)
INCLUDE      'MAINPROG)
PHASE 1,    BEGIN
INCLUDE      'SUB1)
PHASE 1,    MANIP
INCLUDE      'SUB2)
PHASE 2,    CALC1
INCLUDE      'SUB3)
PHASE 2,    CALC2
INCLUDE      'SUB4)
PHASE 1,    QUIT
INCLUDE      'SUB5)
END
```

MODULE 4

Worksheet 1

```
.PROC UPINV  'RECEIPT 'R), ISSUE '
```

APPENDIX C

SAMPLE SOLUTIONS

MODULE 1

Worksheet 1

WITHOUT SHARED PROCEDURES

	TASK1	TASK2	TASK3
P1	10K	10K	10K
P21	15K	---	---
P22	---	10K	10K
TASK	15K	20K	10K
TOTAL	40K	40K	30K

TOTAL is 110K

WITH SHARED PROCEDURES

	TASK1	TASK2	TASK3
P1	10K	---	---
P21	15K	---	---
P22	---	10K	---
TASK	15K	20K	10K
TOTAL	40K	30K	10K

TOTAL is 80K

Savings is 30K

Worksheet 2

```
LIBRARY     TI.COBOL.OBJ
PROCEDURE   RCOBOL
DUMMY
INCLUDE     .S$SYSLIB.RCBPRC
PROCEDURE   ULIB
INCLUDE      'INSUB)
             'ADDNUM)
```

MODULE 4

Worksheet 1

```
.PROC UPINV  'RECEIPT 'R), ISSUE 'I), REORDER REPORT 'P)) = 4,
ENTER OPTION = STRING
.SYN I = TI.INV.OBJ
.SYN OPT = &ENTER OPTION
*
* === VALIDATE OPTION ===
*
.IF @OPT, NE, R
  .IF @OPT, NE, I
     .IF @OPT, NE, P
 MSG TEXT = "INVALID OPTION"
     .ENDIF
  .ENDIF
.ENDIF
*
* === DETERMINE OPTION SELECTED ===
*
.IF @OPT, EQ, R
  .SYN PROG = RECPT
.ENDIF
.IF @OPT, EQ, I
  .SYN PROG = ISSUE
.ENDIF
.IF @OPT, EQ, P
  .SYN PROG = REORD
.ENDIF
*
* === EXECUTE PROGRAM ===
*
XCPF  OBJECT ACCESS NAME = @I.@PROG,
      DEBUG MODE = NO,                  !NOT REQUIRED
      MESSAGE ACCESS NAME = "",         !NOT REQUIRED
      SWITCHES = 00000000,              !NOT REQUIRED
      FUNCTION KEYS = NO                !NOT REQUIRED
*
* === IF OPTION = "P" EXECUTE "REORD" AND PRINT REPORT ===
*
.IF @OPT, EQ, P
   PF  FILE PATHNAME = TI.INV.REORD,
       ANSI FORMAT = NO,                !NOT REQUIRED
       LISTING DEVICE = LP01,
       DELETE AFTER PRINTING = YES,
       NUMBER OF LINES/PAGE = ""        !NOT REQUIRED
   MSG TEXT = "PRINTING OF REORDER REPORT COMPLETE"
.ENDIF
.SYN I = "", OPT = "", PROG = ""
.EOP
```

Worksheet 2

```
CLE  (COMPILE, LINK, AND EXECUTE) = 5,
FILE NAME = ACNM
.SYN CS=TI.COBOL.SRC
.SYN CO=TI.COBOL.OBJ
.SYN CL=TI.COBOL.LST
*
* === COMPILE PROGRAM ===
*
XCCF  SOURCE = @CS.&FILE NAME,
     OBJECT = @CO.&FILE NAME,
     LISTING = @CL.&FILE NAME
*
* === IF NO WARNINGS OR ERRORS, BUILD LINK CONTROL FILE ===
*
.IF @$$CC, EQ, 0
  .DATA TI.COBOL.LCF.&FILE NAME, SUBSTITUTION = YES
     LIBRARY  .S$SYSLIB
     FORMAT   IMAGE,REPLACE
     PROC     RCOBOL
     DUMMY
     INCLUDE  (RCBPRC)
     TASK     &FILE NAME
     INCLUDE  (RCBTSK)
     INCLUDE  (RCBMPD)
     INCLUDE  @CO.&FILE NAME
     END
  .EOD
*
* === EXECUTE LINK EDITOR ===
*
  XLE  CONTROL = TI.COBOL.LCF.&FILE NAME,
       LINKED  = TI.COBOL.PROGF,
       LISTING = TI.COBOL.LMAP.&FILE NAME
  WAIT
*
* === IF NO WARNINGS OR ERRORS, EXECUTE PROGRAM ===
*
  .IF @$$CC, EQ, 0
     AL  LUNO = "",
         ACCESS = TI.COBOL.PROGF,
         PROG FILE = YES,
         DISPLAY = NO
     XCTF  PROG FILE LUNO = @$$LU,
 TASK ID OR NAME = &FILE NAME
     RL  LUNO = @$$LU
  .ENDIF
.ENDIF
.SYN CS = "", CO = "", CL = ""
Q$SYN
```

Worksheet 3

```
CLE  'COMPILE, LINK, AND EXECUTE) = 5,
FILE NAME = ACNM
.SYN CS=TI.COBOL.SRC
.SYN CO=TI.COBOL.OBJ
.SYN CL=TI.COBOL.LST
.SYN F=&FILE NAME
*
* === COMPILE PROGRAM ===
*
.BID  TASK=>87, LUNO=>10, PARMS='@CS.@F, @CO.@F, @CL.@F,
     "", 80, 55, 6144, 1000)
*
* === IF NO WARNINGS OR ERRORS, BUILD LINK CONTROL FILE ===
*
.IF @$$CC, EQ, 0
  .DATA TI.COBOL.LCF.@F, SUBSTITUTION = YES
     LIBRARY   .S$SYSLIB
     FORMAT    IMAGE,REPLACE
     PROC      RCOBOL
     DUMMY
     INCLUDE   'RCBPRC)
     TASK      @F
     INCLUDE   'RCBTSK)
     INCLUDE   'RCBMPD)
     INCLUDE   @CO.@F
     END
  .EOD
*
* === EXECUTE LINK EDITOR ===
*
  .QBID  TASK=>86, LUNO=>10, PARMS='"TI.COBOL.LCF.@F",
         "TI.COBOL.PROGF", "TI.COBOL.LMAP.@F", 4096, 80)
  WAIT
*
* === IF NO WARNINGS OR ERRORS, EXECUTE PROGRAM ===
*
  .IF @$$CC, EQ, 0
     .OVLY  OVLY=>1B, LUNO=0,
  PARMS='6,0,"TI.COBOL.PROGF",Y,$AL$L,Y,N)
     .BID  TASK=@F, LUNO=@$AL$L,
 PARMS=',N,"","","")
     .OVLY  OVLY=>1B, LUNO=0, PARMS='30,@$AL$L)
  .ENDIF
.ENDIF
.SYN CS = "", CO = "", CL = "", F = ""
```

Lab Exercise 1

```
CUBE  'CUBE A NUMBER THAT IS BETWEEN -5 AND +5),
ENTER A NUMBER = INT
.SYN N = &ENTER A NUMBER
.LOOP
  .LOOP
  .WHILE @N, GT, 5
  MSG TEXT = "+NVALID NUMBER, TRY AGAIN"
  CUBE
  .EXIT
  .REPEAT
.WHILE @N, LT, -5
MSG TEXT = "-NVALID NUMBER, TRY AGAIN"
CUBE
.EXIT
.REPEAT
.EVAL CUBEN = @N * @N * @N
MSG TEXT = "THE CUBE OF @N IS @CUBEN"
.SYN N = "", CUBEN = ""
```

Lab Exercise 2

```
PR  'PRINT REPORT)=4,
FILE ACCESS NAME=ACNM,
HOW MANY COPIES?=INT,
ANSI FORMAT?=YESNO 'NO),
DELETE AFTER PRINTING?=YESNO 'NO)
.SYN FAN = "@&FILE ACCESS NAME"
.SYN HMC = "&HOW MANY COPIES"
.SYN AF = "&ANSI FORMAT"
.SYN DEL = "NO"
.LOOP
.WHILE @HMC, GE, 1
*
* === IF LAST COPY, SET DELETE PARAMETER ===
*
  .IF @HMC, EQ, 1
     .SYN DEL = "&DELETE AFTER PRINTING"
  .ENDIF
*
PF  FILE PATHNAME = @FAN,
   ANSI FORMAT = @AF,
   LISTING DEVICE = LP01,
   DELETE AFTER PRINTING = @DEL
.EVAL HMC=@HMC-1
.REPEAT
.SYN FAN="", HMC="", AF="", DEL=""
```

Lab Exercise 3

```
SAVE
.IF "@$XE$", EQ, " "
  MSG TEXT = "OUTPUT FILE PATHNAME", REPLY = "$XE$"
.ENDIF
QE$1  OUTPUT FILE ACCESS NAME = "@$XE$",
      REPLACE = YES,
      MOD LIST ACCESS NAME = ""
XE   FILE ACCESS NAME = "@$XE$"
```

MODULE 5

Worksheet 1

```
XCC  'EXECUTE COBOL COMPILER <VERSION: 3.2.0 79173>) =2,
FILE NAME = ACNM '"@$XCC$F")
DIRECTORY NAME = ACNM '"@$XCC$D"),
PROGRAM SIZE  'LINES) = INT '1000)
.SYN D=@&DIRECTORY NAME
.SYN F=&FILE NAME
.SYN S=@@D.SRC.@F
.SYN O=@@D.OBJ.@F
.SYN L=@@D.LST.@F
.SYN MEMX = "&PROGRAM SIZE  'LINES)"
.EVAL MEMORY = "@MEMX / 500 * 7168"
.IF "@MEMORY" ,GT, "30840"
.EVAL MEMORY = "30840"
.ENDIF
.IF "@MEMORY" ,LT, "7168"
.EVAL MEMORY = "6144"
.ENDIF
*
* === DELETE UNNECESSARY MESSAGES AND SYNONYM ASSIGNMENT ===
* === MODIFY .QBID PARAMETERS ===
*
.QBID TASK = >87, LUNO = >10,
PARMS =  '@@@S, @@@O, @@@L, X, 80, 55,
       @MEMORY,&PROGRAM SIZE  'LINES))
*
* === MODIFY SYNONYM ASSIGNMENTS ===
*
.SYN $XCC$D = &DIRECTORY NAME
.SYN $XCC$F = &FILE NAME
.SYN MEMORY="", MEMX="", S="", O="", L="", D="", F=""
```

Lab Exercise 1

```
CFILE (COPY FILE) = 7,
INPUT DIRECTORY NAME = ACNM,
OUTPUT DIRECTORY NAME = ACNM
.SYN IDN = @&INPUT DIRECTORY NAME
.SYN ODN = @&OUTPUT DIRECTORY NAME
.SYN FN=0
* *
*  INCREMENT INPUT FILE NUMBER
*  GENERATE UNIQUE FILE NAME AND COPY
*     AS LONG AS OPERATOR ENTERS "Y"
* *
.LOOP
  .EVAL FN = @FN+1
  .SYN FILE = TCTEMP@FN
  UNIQUE SYNONYM = F
  CC INPUT = @@IDN.@FILE,
     OUTPUT = @@ODN.@F,
     REPLACE = YES
  MSG TEXT="@@IDN.@F   COPIED TO   @@ODN.@F"
  MSG TEXT="CONTINUE (Y/N)", REPLY=ANS
.WHILE @ANS, EQ, Y
.REPEAT
.SYN IDN="", ODN="", FN="", FILE="", F="",
    $$UN$1="", $$UN$2="", ANS=""
```

Lab Exercise 2

```
UNIQUE (GENERATE UNIQUE FILENAME),
SYNONYM TO BE ASSIGNED=NAME ("@$$UN$2")
.SYN $$UN$2=&SYN
.IF ("@$$UN$1",EQ,"$$UN$1"
.EVAL $$UN$1="@$$ST*100"            !MODIFY MULTIPLIER
.ENDIF
.SYN &SYN=CFILE@$$UN$1              !MODIFY FILE NAME
.EVAL $$UN$1="@$$UN$1+1"
.IF @$$UN$1,GT,"@$$ST*100+99"       !MODIFY VALIDATION
.SYN $$UN$1=""
.ENDIF
```

Lab Exercise 3

```
CFKEY  (CREATE KEY INDEXED FILE),
PATHNAME                    = ACNM,
LOGICAL RECORD LENGTH       = INT,
PHYSICAL RECORD LENGTH      = *INT,
INITIAL ALLOCATION          = *INT,
SECONDARY ALLOCATION        = *INT,
MAXIMUM SIZE                = INT
.IF  "@$CFK$KN", NE, "$CFK$KN"
MSG  T="ERROR:  INVALID CFKEY SEQUENCE; CFKEY BEFORE ENDKEY"
.SYN   $CFK$L="", $CFK$PN="", $CFK$FLRL="",
$CFK$PRL="", $CFK$KS="", $CFK$IA="", $CFK$M="",
$CFK$SA="", $CFK$MS="", $CFK$KN=""
.EXIT
.ENDIF
.SYN   $CFK$KN = 1,
$CFK$L   = "",
$CFK$KS  = "1",
$CFK$PN  = "@&PATHNAME",
$CFK$FLRL = "&LOGICAL RECORD LENGTH",
$CFK$PRL = "&PHYSICAL RECORD LENGTH",
$CFK$IA  = "&INITIAL ALLOCATION",
$CFK$SA  = "&SECONDARY ALLOCATION",
$CFK$MS  = "&MAXIMUM SIZE",
*
* === ADD FOLLOWING SYNONYM ASSIGNMENT ===
*
$CFK$M   = "NO"
.IF  "&PHYSICAL RECORD LENGTH", EQ, ""
.SYN  $CFK$PRL = 0
.ENDIF
.IF  "&INITIAL ALLOCATION", EQ, ""
.SYN $CFK$IA = 0
.ENDIF
.IF  "&SECONDARY ALLOCATION", EQ, ""
.SYN $CFK$SA = 0
.ENDIF
.IF  "@$$MO", NE, 0
CFK$1
ENDKEY
.ENDIF
```

```
KEY  'KEY DESCRIPTION FOR KEY NUMBER @$CFK$KN),
START POSITION = INT '"@$CFK$KS"),
KEY LENGTH   = INT,
DUPLICATES? = YESNO  'NO),
*
* === MODIFY THE DEFAULT FOR THE FOLLOWING KEYWORD ===
*
MODIFIABLE? = YESNO  '@$CFK$M),
ANY MORE KEYS? = *YESNO 'YES)
.IF   "@$CFK$KN", EQ, "$CFK$KN"
MSG   T="ERROR:  INVALID CFKEY SEQUENCE; KEY BEFORE CFKEY"
.EXIT
.ENDIF
*
* === DELETED .IF STATEMENT ===
*
.SYN    $CFK$KS = "&START POSITION"
*
*  MODIFY FOLLOWING PARAMETER LIST
*  @$CFK$M BECOMES &MODIFIABLE
*
.SYN
$CFK$L="@$CFK$L, '@$CFK$KS,&KEY LENGTH,&DUPLICATES,&MODIFIABLE,Y)"
.EVAL   $CFK$KN = @$CFK$KN+1
*
* === ADD THE FOLLOWING .SYN AND .EVAL STATEMENTS ===
*
.SYN    $CFK$M = "YES"
.EVAL   $CFK$KS = "@$CFK$KS + &KEY LENGTH"
.IF     "&ANY MORE KEYS", LT, Y
.SYN   $CFK$KS = ""
.ENDIF
```

No modifications of CFK$1 or ENDKEY are required.

Lab Exercise 4

```
ACL 'AUTOMATED COBOL LINK)=5,
OBJECT ACCESS NAME=ACNM '@$$XCC$OB),
APPLICATION NAME=NAME,
PROGRAM FILE NAME=ACNM,
1st LINK TO THIS OUTPUT FILE=YESNO 'NO),
LIBRARY ACCESS NAME=*ACNM '@$ACL$LIB),
LISTING ACCESS NAME=ACNM
.SYN $ACL$AN=&APPLICATION
.SYN $XCC$OB=@&OBJECT
.SYN $ACL$LIB=@&LIBRARY
*
* === IF FIRST LINK CREATE PROGRAM FILE ===
*
.IF &1st, GE, Y
    .OVLY  OVLY=>1B, LUNO=0,
           PARMS= '14,&PROGRAM,25,10,20,85,"",YES)
.ENDIF
*
* === SET LIBRARY COMMAND ===
*
.IF @$ACL$LIB, NE, ""
    .SYN LIBRARY="LIBRARY  @$ACL$LIB"
.ELSE
    .SYN LIBRARY=";NO LIBRARY USED"
.ENDIF
*
* === BUILD LINK CONTROL FILE ===
*
.DATA  .S$ACL@$$ST, SUBSTITUTION=YES
    @LIBRARY
    FORMAT  IMAGE, REPLACE
    PROCEDURE  RCOBOL
    DUMMY
    INCLUDE  .S$SYSLIB.RCBPRC
    PROCEDURE  @$ACK$ANTION
    INCLUDE  .S$SYSLIB.RCBMPD
    INCLUDE  @$XCC$OB
    TASK  @$ACL$AN
    INCLUDE  .S$SYSLIB.RCBTSK
    END
.EOD
*
* === EXECUTE LINK EDITOR ===
*
.QBID  TASK=>86, LUNO=>10,
       PARMS= '.S$ACL@$$ST, &PROGRAM, &LISTING, 4096, 80)
*
.OVLY  OVLY=>23, CODE=10                    !WAIT COMMAND
*
* === DELETE TEMPORARY LINK CONTROL FILE ===
*
.OVLY  OVLY=>1B, LUNO=0, PARMS= '8, .S$ACL@$$ST)
.SYN  LIBRARY="", $ACL$AN=""
```

MODULE 6

Lab Exercise 1

```
BATCH
.SYN C=TI.COBOL
XCC  SOURCE=@C.SRC.STOCK, OBJECT=@C.OBJ.STOCK, LIST=@C.LST.STOCK
.IF @$$CC, GT, 0
   .STOP TEXT="ERRORS IN THE COMPILE PHASE
.ENDIF
.DATA  @C.LCF.STOCKREN, SUBSTITUTION=YES
   FORMAT      IMAGE, REPLACE, 4
   PROCEDURE   RCOBOL
   DUMMY
   INCLUDE     .S$SYSLIB.RCBPRC
   PROCEDURE   INVSUBS
   INCLUDE     @C.OBJ.RDINVO
   INCLUDE     @C.OBJ.RECPTO
   INCLUDE     @C.OBJ.ISSUEO
   INCLUDE     @C.OBJ.REORDO
   INCLUDE     @C.OBJ.SSTATO
   TASK        STOCKREN
   INCLUDE     .S$SYSLIB.RCBTSK
   ALLOCATE
   INCLUDE     .S$SYSLIB.RCBMPD
   INCLUDE     @C.OBJ.STOCKO
   END
.EOD
XLE  CONTROL=@C.LCF.STOCKREN, LINK OUT=@C.PROGF,
     LIST=@C.LMAP.STOCKREN
EC
.DATA  @C.LCF.STOCKOVL, SUBSTITUTION=YES
   FORMAT      IMAGE, REPLACE, 4
   LIBRARY     .S$SYSLIB
   PROCEDURE   RCOBOL
   DUMMY
   INCLUDE      'RCBPRC)
   PHASE 0,    STOCKOVL
   INCLUDE      'RCBTSK)
   LOAD
   INCLUDE      'RCBMPD)
   INCLUDE     @C.OBJ.STOCKO
   INCLUDE     @C.OBJ.SSTATO
   PHASE 1,    RDINV
   INCLUDE     @C.OBJ.RDINVO
   PHASE 1,    RECPT
   INCLUDE     @C.OBJ.RECPTO
   PHASE 1,    ISSUE
   INCLUDE     @C.OBJ.ISSUEO
   PHASE 1,    REORD
   INCLUDE     @C.OBJ.REORDO
   END
.EOD
```

```
XLE  CONTROL=@C.LCF.STOCKOVL, LINK OUT=@C.PROGF,
     LIST=@C.LMAP.STOCKOVL
EC
.DATA  @C.LCF.STOCKOV2, SUBSTITUTION=YES
   FORMAT      IMAGE, REPLACE, 4
   LIBRARY     .S$SYSLIB
   PROCEDURE   RCOBOL
   DUMMY
   INCLUDE      'RCBPRC)
   PHASE 0,    STOCKOV2
   INCLUDE      'RCBTSK)
   LOAD
   INCLUDE      'RCBMPD)
   INCLUDE     @C.OBJ.STOCKO
   PHASE 1,    RDINV2
   INCLUDE     @C.OBJ.RDINVO
   PHASE 1,    RCVISU
   INCLUDE     @C.OBJ.RECPTO
   INCLUDE     @C.OBJ.ISSUEO
   PHASE 2,    SSTAT2
   INCLUDE     @C.OBJ.SSTATO
   PHASE 1,    REORD2
   INCLUDE     @C.OBJ.REORDO
   END
.EOD
XLE  CONTROL=@C.LCF.STOCKOV2, LINK OUT=@C.PROGF,
     LIST=@C.LMAP.STOCKOV2
EC
.SYN C=""
Q$SYN
EBATCH  TEXT="THERE ARE @$E$C ERRORS IN THE BATCH STREAM"
```