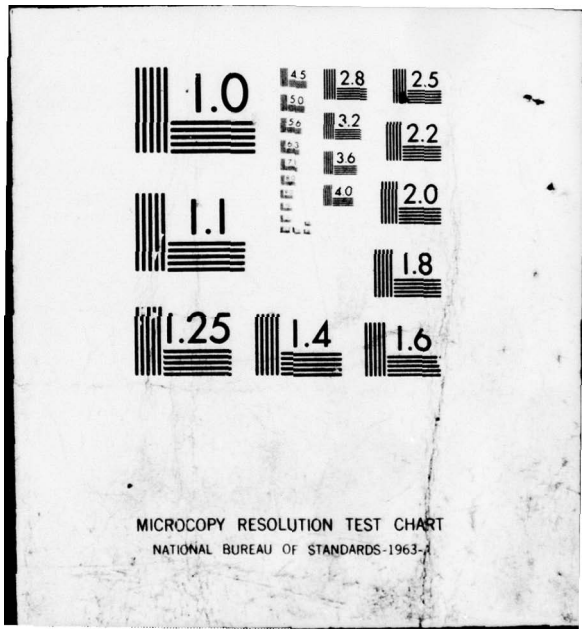
AD-A055 023 HUMAN RESOURCES RESEARCH ORGANIZATION ALEXANDRIA VA F/G 5/9
PROJECT IMPACT SOFTWARE DOCUMENTATION: VI. VOLUME 2. ZEUS PROGR--ETC(U)
AUG 72 J GARNEAU, W UNDERSHILL, D SHUFORD DAHC19-73-C-0004
UNCLASSIFIED HUMRRO-RP-D1-72-5-VOL-2 NL

1 OF 2
AD
A055 023


1.0
1.1
1.25
1.4
1.6
1.8
2.0
2.2
2.5
2.8
3.2
3.6
4.0
4.5
5.0
5.6
MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

August 1972
RP-DI-72-5

HumRRO

Project IMPACT Software Documentation: VI. Volume 2, Zeus Program Logic Descriptions

Research Product

by
Jean Garneau, William Underhill, and Doris Shuford

HumRRO Division No. 1 (System Operations)

Work Unit: IMPACT

HUMAN RESOURCES RESEARCH ORGANIZATION
300 North Washington Street • Alexandria, Virginia 22314

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER (14) RP-D1-72-5 HUMRRO-RP-D1-72-5-VOL-2	2. GOVT ACCESSION NO. 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) (6) PROJECT IMPACT SOFTWARE DOCUMENTATION: VI. VOLUME 2. ZEUS PROGRAM LOGIC DESCRIPTIONS.	5. TYPE OF REPORT & PERIOD COVERED (9) Research Product 6. PERFORMING ORG. REPORT NUMBER RP-D1-72-5
7. AUTHOR(s) (10) Jean Garneau, William Undershill, Doris Shuford	8. CONTRACT OR GRANT NUMBER(s) (15) DAHC 19-73-C-0004
9. PERFORMING ORGANIZATION NAME AND ADDRESS Human Resources Research Organization (HumRRO) 300 North Washington Street Alexandria, Virginia 22314 405 260	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS (16) CDA Proj 2Q063101D734
11. CONTROLLING OFFICE NAME AND ADDRESS (11) Office, Chief of Research and Development Department of the Army Washington, D.C. 20310	12. REPORT DATE Aug 72 13. NUMBER OF PAGES 144
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)	15. SECURITY CLASS. (of this report) (12) 146 p. Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Research performed by HumRRO Division No. 1 (System Operations), under Work Unit IMPACT.

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

CAI	Message Processing	Tasking
Cathode Ray Tube	Operating System (OS)	Time-Sharing
Command Processing	Queue Management	
Memory Management	System Support	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This research provides a description of Project IMPACT's time-sharing monitor, ZEUS, which extends the capabilities of an IBM 360 or 370 computer system. Its functions, capabilities, structure and operation are discussed. ZEUS provides capabilities needed for real-time, interactive computer applications using cathode ray tube displays. This document is intended primarily for systems programmers, and is part of a series that will completely document the IMPACT software subsystem.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

405 260

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

August 1972
RP-D1-72-5

Project IMPACT Software Documentation: VI. Volume 2, Zeus Program Logic Descriptions

Research Product

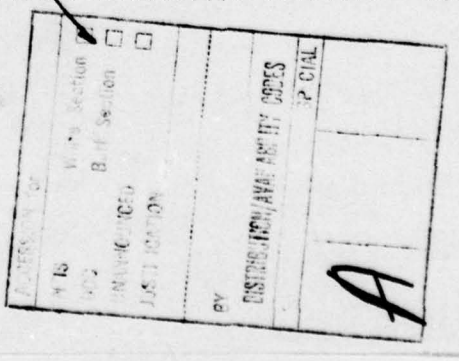
by
Jean Garneau, William Underhill, and Doris Shuford

HumRRO Division No. 1 (System Operations)

Work Unit: IMPACT

HUMAN RESOURCES RESEARCH ORGANIZATION
300 North Washington Street • Alexandria, Virginia 22314

CONTENTS



Project IMPACT Software Documentation:
VI. Volume 2,
Zeus Program Logic Descriptions

INTRODUCTION

This is the second volume in a two-volume set of documents describing Zeus, a time-sharing monitor for the IBM series 360 or 370 computer. In Volume 1, the nature, purpose, history, functions, and design concepts of Zeus are described; this document presents the program logic for Zeus subroutines and macros.

ORGANIZATION OF THIS DOCUMENT

Volume 2 is organized in five Chapters, as follows:

Chapter 1, "System Modules," describes each of the program modules in Zeus. The modules are grouped according to functions and these groups are presented in the sequence in which the functions are described in Volume 1.

Chapter 2, "Macros," describes the system macros provided by Zeus. These macros are used by Zeus subroutines and many are also suitable for application program use.

Chapter 3, "System Data Areas," identifies the structure and uses of control blocks and queue elements in Zeus.

Chapter 4, "Utilities," describes utility programs provided for the support of the Zeus system.

Chapter 5, "Procedures," lists the Job Control Language (JCL) procedures necessary for the implementation and operation of the Zeus system.

Program listings of Zeus modules and macros are available to the interested reader through HumRRO Division No. 1 (System Operations).

OVERVIEW

A brief description of how a terminal interacts with Zeus, as illustrated in Figure 1, will serve to place the Zeus components in proper perspective. In Figure 1, assume that the Zeus initialization routines have completed processing and a Terminal Task has been created for each terminal in the system. Each Terminal Task has issued a read to its terminal via OS and a Zeus macro. These functions are performed by the TERMIO component of Message and Command Processing. After initiating this operation, the Terminal Task waits for terminal input by entering the DSPTCHR component of System Support.

Eventually, the terminal initiates an input. At this time, OS notifies the Terminal Task associated with this terminal (via System Support) of the message availability. The Terminal Task, when activated, first determines if the message contains a command. If it does not it is routed to the external processor associated with this terminal.

If the message contains a command, the Terminal Task calls the Command Processing Subsystem to process it. (CMNDPR1 is the entry point to the Command Processing Subsystem.)

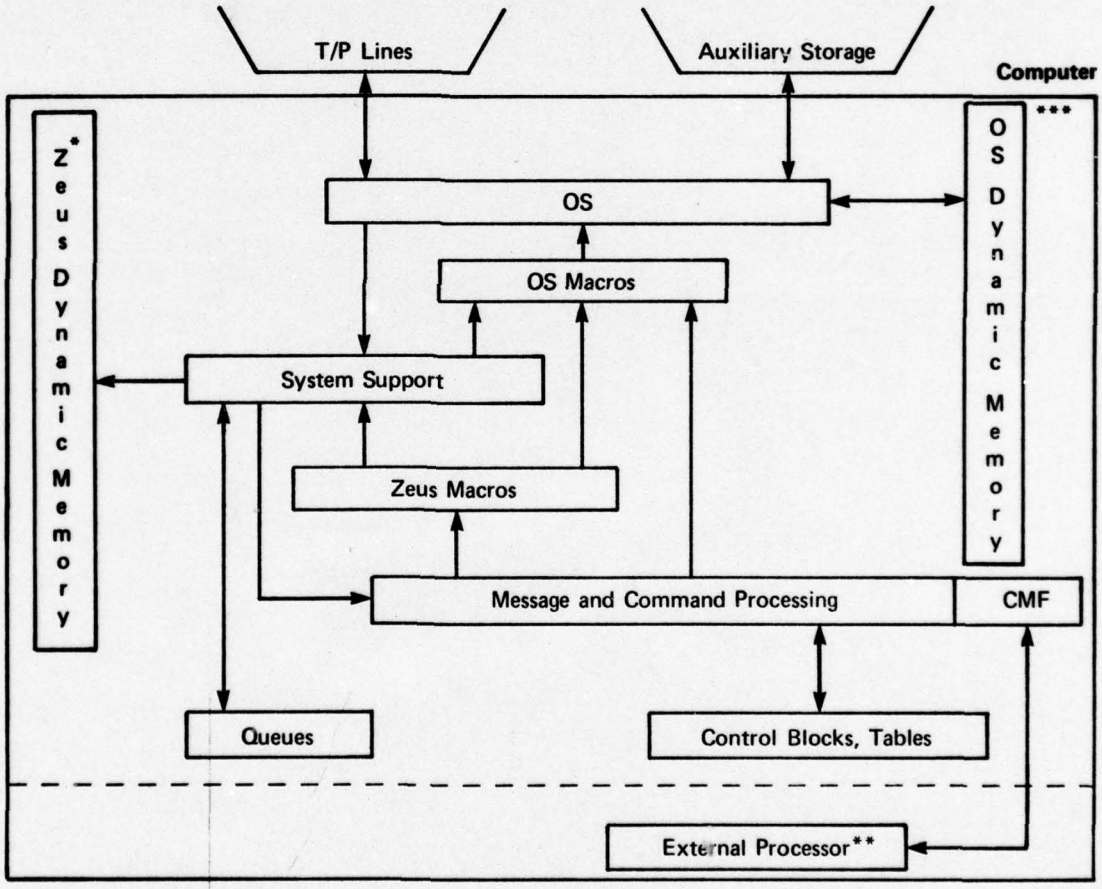
The Command Processing Subsystem interrogates the command and executes it as appropriate, utilizing Zeus macros and System Support for timer, queue, and memory management. Control Blocks are updated as necessary to maintain a state vector for the terminal.

Eventually Command Processing or the external processor will generate a request for output and subsequent input from the terminal, repeating the preceding operation.

A detailed description of this process can be found in Volume 1, Chapter 3, "Message Processing;" Chapter 4, "Command Processing;" and Chapter 5, "System Support Functions."

The main memory requirements for the major Zeus subsystems components are summarized in Figure 2. The reader may refer to the program description in subsequent chapters of this volume for detailed memory requirements for each Zeus module.

Basic Zeus Structure



*Zeus Dynamic Memory used for transient routines, save areas, and queue extensions as well as I/O buffers.

**Optional (non-Zeus) Component.

***OS Dynamic Memory used for I/O Appendages, Open/Close/EOV, etc.

Figure 1

Summary, by Component, of Zeus Main Memory Requirements

Component		Main Memory*
Message Processing		6K
Command Processing	EDITOR	3K
	DIRECTOR	3K
	Status/control	2K
	RJE	3K
System Support		8K
Buffer Pool and Transient Routine Dynamic Memory		8K
Control Blocks and Table for 9 Active Terminals		2K
OS Dynamic Memory (open, close, appendages, etc.)		5K
Total		54K

*K = 1024 bytes of memory

Figure 2

Chapter 1

SYSTEM MODULES

This Chapter presents salient items of information regarding Zeus system modules. The system modules descriptions are organized alphabetically within sections, according to the type of function performed. These sections are presented in the sequence in which the functions are described in Volume 1. The subroutines are as follows:

Section 1: Message processing
Section 2: Command processing
Section 3: System support
Section 4: Initialization

SECTION 1: MESSAGE PROCESSING

The program modules that perform message processing functions are shown in the following list:

Name	Function
CMF	I/O Communication Interface between a processor and a Terminal Task
CRTREAD	Request message from terminal
EMPTY	Request output to terminal
FULL	Process full output buffer condition
NTPIO	Initiate nonteleprocessing I/O operation
RESET	Acquire and initialize output buffer
RTERM	Terminal Task to Line Task interface for terminal input operation
RTERM1	Initiate terminal input operation
STACK	Format output buffer
TERM10	Terminal Task foundation module
WTERM	Terminal Task to Line Task interface for terminal output operation
WTERM1	Initiate terminal output operation

Zeus message processing concepts are described in Volume 1, Chapter 3. A description of each module follows.

NAME:	CMF
FUNCTION:	Read/Write Communication Interface between a processor and a Terminal Task.
INPUT:	R1 → BTAM Data Event Control Block (DECB) R13 → 18-word save area R14 → Return point address R15 → Entry point to CMF
OUTPUT:	Updated DECB, TCB, and posted Terminal Task.
PROCESS:	The CMF routine is called by a processor to initiate a read or write operation on a remote terminal. CMF first interrogates the bytes at offset 36 of the DECB for an X 'FF' to determine if a terminal has been assigned to this DECB. If a terminal has been assigned, it loads the address of its TCB from bytes of the DECB and continues to the next step. Otherwise, it locates the Zeus Vector Table, gets the address of the TCB Queque from it, and scans the Queue for an available TCB. If one is found, its address is stored in bytes 37-39 of the DECB, the DECB address is stored in the TCB, and the next step is initiated. If a TCB is not available, the CMF returns to the caller. After the TCB belonging to this DECB is known, the address of the message to be output or the buffer to be used for input is moved from the DECB to the TCB. Editing is performed on outgoing messages as is required by specific processors. Finally, an ECB in the TCB is posted complete.
ZEUS MACROS USED:	#OPENP, #CALL, #GOTO, #LPSW, #COUNT, #CLOSEP, #GENSECT, #FINDZVT
ROUTINES CALLED:	SPIE, STAE
SIZE IN BYTES:	403 bytes
EXTERNAL REFERENCES:	ZEUSSVC, SPIE, STAE
EXIT:	Address in R14
ATTRIBUTES:	Serially Reusable
WRITTEN BY:	Jean Garneau

NAME:	CRTREAD
FUNCTION:	Request input from terminal, process and, as appropriate, return data to caller.
INPUT:	R12 → TCB R13 → 18-word save area R14 → Return point address R15 → Entry point to CRTREAD
PROCESS:	CRTREAD is called by any routine that wants noncommand input from the terminal. CRTREAD first checks the current terminal output buffer to determine if it contains valid data. If it does, an output operation to the terminal is initiated by calling an output module, EMPTY. Upon return from EMPTY, or if the buffer contained no data, an input message from the terminal is requested via RTERM. Upon return from RTERM, CRTREAD waits on an ECB in its TCB for completion of the scheduled input operation. When the terminal responds, the message address is stored in the TCB and the ECB being waited on is posted by the Line Task associated with this terminal. When CRTREAD is dispatched, it checks the message for the presence of a Zeus command. If a command is located, the command processing subsystem is called. Upon return from this subsystem, a branch is made back to the first step of CRTREAD when the output buffer is checked for valid data. This loop will continue as long as the messages being input contain commands. When a message is input that does not contain a command, return is made to the original caller of CRTREAD.
ZEUS MACROS USED:	#OPENP, #COUNT, #GOTO, #WAIT, #CLOSEP, #FREEMEM
ROUTINES CALLED:	EMPTY, RTERM, CMNDPR1, STACK
SIZE IN BYTES:	368 bytes
EXTERNAL REFERENCES:	RTERM, GETSAVE, ICOUNT, EMPTY, FREEMEM
EXIT:	Address in R14
ATTRIBUTES:	REENTRANT
WRITTEN BY:	Jean Garneau

NAME:	EMPTY
FUNCTION:	Request output operation to terminal.
INPUT:	R12 → TCB R13 → 18-word save area R14 → Return point address R15 → Entry point to EMPTY
OUTPUT:	Output buffer, if any, written on terminal and deallocated.
PROCESS:	EMPTY is called to flush any valid data from the terminal output buffer. EMPTY first determines if, indeed, the buffer is available and does contain valid data. If not, it returns; otherwise it calls WTERM to schedule an output operation to the terminal. Upon return from WTERM, it waits on an ECB in the TCB for completion of the output operation. It then cleans up the TCB, frees the buffer, and returns to the caller.
ZEUS MACROS USED:	#OPENP, #GOTO, #SET, #WAIT, #COUNT, #CLOSEP, #FREEMEM
ROUTINES CALLED:	WTERM
SIZE IN BYTES:	316 bytes
EXTERNAL REFERENCES:	WTERM, FREEMEM, GETSAVE, ICOUNT
EXIT:	Address in R14
ATTRIBUTES:	REENTRANT
WRITTEN BY:	Jean Garneau

NAME:	FULL
FUNCTION:	Handle full output buffer condition
INPUT:	R12 → TCB R13 → 18-word save area R14 → Return point address R15 → Entry point to FULL
OUTPUT:	Buffer output to terminal and then deallocated
PROCESS:	FULL first inserts a comment into Block One of the buffer explaining that the output operation is caused by a "full" CRT "page." The address of this block is available in the TCB. EMPTY then calls CRTREAD which schedules the output operation, waits for a "press-send", etc. CRTREAD returns a message that is discarded by FULL. FULL then returns to the caller.
ZEUS MACROS USED:	#OPENP, #COUNT, #CLOSEP, #FREEMEM
ROUTINES CALLED:	CRTREAD
SIZE IN BYTES:	168 bytes
EXTERNAL REFERENCES:	GETSAVE, FREEMEM, ICOUNT
EXIT:	Address in R14
ATTRIBUTES:	REENTRANT
WRITTEN BY:	Jean Garneau

NAME:	NTPIO
FUNCTION:	Initiate nonteleprocessing I/O operations.
INPUT:	R1 → I/O buffer plus operation code R12 → TCB R13 → 18-word save area R14 → Return point address R15 → Entry point to NTPIO
OUTPUT:	I/O operation scheduled
PROCESS:	NTPIO, using the Basic Access Method DECB contained in the TCB, initiates either an Input or an Output operation and then returns to the caller. Register 1 at entry contains the address of the buffer to be written from or read into. If register 1 is negative, the operation requested is a write, otherwise it is a read. If the first bit of the FILECB (in the TCB) byte is on, the I/O operation involves keys, otherwise it is relative record I/O.
ZEUS MACROS USED:	#OPENP, #GOTO, #COUNT, #CLOSEP
ROUTINES CALLED:	None
SIZE IN BYTES:	356 bytes
EXTERNAL REFERENCES:	None
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable
WRITTEN BY:	Jean Garneau

NAME:	RESET
FUNCTION:	If a terminal output buffer is not currently available, RESET allocates one and formats it.
INPUT:	R1 → TCB R13 → 18-word save area R14 → Return format address R15 → Entry point to RESET
OUTPUT:	Updated TCB
PROCESS:	Reset checks SBUF1A in the TCB. If SBUF1A equals zero, a new buffer is allocated via #GETMEM. The TCB fields SBUF1A, PAGELINE, PAGEND, LINEREM, and POSREM are updated and control is returned to the caller.
ZEUS MACROS USED:	#GENSECT, #OPENP, #GETMEM, #COUNT, #GOTO, #CLOSEP, #REGS, #FREEMEM
ROUTINES CALLED:	None
SIZE IN BYTES:	264 bytes
EXTERNAL REFERENCES:	GETSAVE, ICOUNT, DSPTCHR, GETMEM, FREEMEM
EXIT:	Address in R14
ATTRIBUTES:	REENTRANT
WRITTEN BY:	Jean Garneau

NAME:	RTERM
FUNCTION:	To serve as an interface to the TP line task.
INPUT:	R12 → TCB R13 → 18-word save area R14 → Return point address R15 → Entry point to RTERM
OUTPUT:	TCB and LCB updated to reflect current status.
PROCESS:	The user specifies the address of the TCB, which points to the terminal that is to be read. RTERM increments the poll count (LPOLLC) field of the line (LCB) to which the terminal belongs. If the line is idle, RTERM posts its LCB complete and returns. If the line is busy, return is made without posting.
ZEUS MACROS USED:	#OPENP, #REGS, #GENSECT, #SET, #GOTO, #CLOSEP
ROUTINES CALLED:	None
SIZE IN BYTES:	184 bytes
EXTERNAL REFERENCES:	None
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable
WRITTEN BY:	Jean Garneau

NAME: RTERM1 (Line Task Foundation Module)

FUNCTION: To issue a read initial to the requesting line, move the message read to a buffer, and translate the message from ASCII-8 to EBCDIC.

INPUT: R1 → LCB

R15 → Entry point to RTERM1

OUTPUT: TCBECB is posted complete (Byte 1 = X'40')
AIOBUF updated (only if return code = 0)
LSTAT updated
LPOLLC updated

Error codes returned in right-most byte of TCBECB

00 = Read O.K.
04,08,0C = Read bad, try again.

PROCESS: RTERM1 is attached once for each communication line to be activated. It immediately branches to label "START". START interrogates the LCB to determine if any terminals (TCBs) require I/O servicing. If none do, the task enters idle mode by turning off the busy bit in the LCBFLGS byte of the LCB, and issuing a #WAIT on an ECB in the LCB. The task remains idle until a TCB associated with its LCB requests I/O by altering certain status fields in both the TCB and LCB and then posting the ECB that the task is waiting on.

When activated, the task first checks to see if a (terminal) output operation has caused the activation. If so, it branches to a routine (WTERM1) to take care of it/them. If not, it attempts to acquire memory for an input buffer. If not available, a #WAIT is issued to wait for the memory to become available. If the memory request is deleted during the wait, a branch is made back to START. When the memory is acquired, the address and length are saved and a read into the buffer is issued to the line. If it completes without error, the message is moved to a second buffer acquired in the same manner as the first, edited, and translated to EBCDIC. If the read completes with an error, the appropriate message is sent to the console, a write negative acknowledgement is issued, memory is freed, and a branch to START is issued. If the read completes with a negative response to poll, the negative response flag is turned on and a check is made to determine if all terminals requesting input have been polled. If not, the routine loops until all have been serviced. If all have been checked, the input buffer is freed and a branch is made to the write routine if any writes have been requested during the period. If no writes are pending, a #STIMER is issued and, when the interval completes, a branch to START is made.

ZEUS MACROS USED:	#SET, #REGS, #GENSECT, #OPEN, #GOTO, #GETMEM, #WAIT, #MOVE, #FREEMEM, #STIMER, #CALL
ROUTINES CALLED:	WTERM1
SIZE IN BYTES:	920 bytes
EXTERNAL REFERENCES:	IECTTRNS, ITIMER, GETMEM, DSPTCHR, MOVEOOO, FREEMEM, ATE, WTERM1
EXIT:	To address in R14
ATTRIBUTES:	REENTRANT
WRITTEN BY:	Jean Garneau

NAME: STACK

FUNCTION: To scan character string and format page for output to CRT.

INPUT: R0 = Maximum length of scan.
R1 → Beginning of string to be scanned.
R12 → TCB
R14 → Return point
R15 → Entry point to STACK

OUTPUT: Character string moved to output buffer and count of lines and bytes remaining in page updated.

PROCESS: The character string is scanned and the line count field in the TCB is decremented by one for each carriage return encountered or for each 84 bytes with no carriage return. The positions remaining field in the TCB is decremented by the length of the message. The scan is terminated when an ETX is found or when the scan limit (R0) is reached. The character string and prefix (if necessary) are moved to the output buffer, the address of which is in the SBUF1A field of the TCB. If the page is filled before the scan terminates, FULL and RESET are called prior to continuing the scan.

ZEUS MACROS USED: #REGS, #OPENP, #GOTO, #SET, #COUNT, #CLOSEP

ROUTINES CALLED: RESET, FULL

SIZE IN BYTES: 578 bytes

EXTERNAL REFERENCES: GETSAVE, RESET, FULL, ICOUNT, FREEMEM

EXIT: Address in R14

ATTRIBUTES: REENTRANT

WRITTEN BY: Jean Garneau

NAME: TERMIO (Terminal Task foundation module)

FUNCTION:

INPUT: R1 → Terminal Control Block

R13 → 18-word save area

R15 → TERMIO entry point address

OUTPUT: Updated DECB, TCB, LCB; I/O operation initiated and completed.

PROCESS: TERMIO is attached by the Zeus initiation routine once for each terminal to be activated. It immediately waits on a "no work" ECB in the TCB passed to it during the attach. It remains dormant until its TCB has been acquired and posted by an external processor. When the TCB is posted by the external processor, the task becomes active. It loads the external processor DECB address into a register and checks the read/write flag. If it is a write operation, a buffer is acquired, the message is moved to it and translated, if necessary, to EBCDIC. Next, the message is checked to determine if a Zeus command is present. If a command is present, the Command Processing Subsystem is called, otherwise a routine (STACK) is called which handles message traffic to the terminal. If the operation requested is read, a routine (CRTREAD) is called which handles incoming messages from terminal. Upon return from this routine, an address is available in the TCB which points to the message to be passed to the external processor. TERMIO loads this address into a register, translates the message, if necessary, moves the message to the external processor buffer, and posts the external processor's DECB. Finally, TERMIO again waits on its "no work" ECB.

ZEUS MACROS USED: #OPEN, #GENSECT, #SET, #WAIT, #GOTO, #COUNT, #LPSW, #CALL, #GETMEM, #FREEMEM

ROUTINES CALLED: MOVE, STACK, CRTREAD, EMPTY, CMNDPROC

SIZE IN BYTES: 802 bytes

EXTERNAL REFERENCES: MOVE, IECTTRNS, CMNDPROC, STACK, EMPTY, CRTREAD, DSPTCHR, LTE, FREEMEM, CTLTAB, ETL, GETMEM

EXIT: N/A

ATTRIBUTES: REENTRANT

WRITTEN BY: Jean Garneau

NAME:	WTERM
FUNCTION:	To serve as an interface between a Terminal Task and a Line Task.
INPUT:	R12 → TCB R13 → 18-word save area R14 → Return point address R15 → Entry point to WTERM
OUTPUT:	TCB and LCB updated to reflect current status.
PROCESS:	The user specifies the address of the terminal (address of the TCB) which is to be written. WTERM increments the write count field (LADDRC) of the line (LCB) to which the terminal belongs. If the line is idle, WTERM posts it complete and returns. If polling, WTERM calls RESETPL and then returns. If waiting for an interval to expire, a #TTIMER CNACEL is issued and then return is made. If waiting for memory, the memory queue element for this request is cancelled and return is made.
ZEUS MACROS USED:	#OPENP, #REGS, #SET, #GENSECT, #GOTO, #TTIMER, #FREEMEM, #CLOSEP
ROUTINES CALLED:	RESETPL
SIZE IN BYTES:	304 bytes
EXTERNAL REFERENCES:	ITIMER, RESETPL, FREEMEM
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable
WRITTEN BY:	Jean Garneau

NAME: WTERM1

FUNCTION: To issue an OS, write to a specified line.

INPUT: R12 → TCB

AIOBUFF → buffer
BYTES 1&2 = buffer length
BYTES 3&4 = message length
BYTE 5 = X'02' (STX)
BYTES 6 to n = message

OUTPUT: TCBECB posted complete (Byte 1 = X'40')
LSTAT updated
LADDRC updated

Error codes in right-most byte of TCBECB

X'00' = no errors
X'04' = message too long for buffer
X'08' = no write operation initiated
X'0C' = write completed with error

PROCESS: The message length and address are obtained from AIOBUF. The message is then translated from EBCDIC to ASCII-8, an ETX (X'03') placed in the last position of the buffer, and a STX (X'02') in the first position to the buffer. A write initial with reset is issued, and if there was an error-free completion of the operation, the necessary LCB and TCB fields are updated. The TCBECB is posted with a 0 and return is made. If there was a non-X'7F' completion, an attempt is made to write a positive acknowledgement. If unsuccessful, a message is sent to the console, the TCBECB is posted with a 12, the necessary LCB and TCB fields are updated, and return is made. If the write "ack" is successful, the message is not sent but the TCBECB is posted with a 12, the LCB and TCB fields are updated, and return is made. If there was an error in the buffer length, the TCBECB is posted with a 4. If there was a buffer address error, a message is sent to the console and the TCBECB is posted with a 0. If there was a bad write operation the TCBECB is posted with an 8.

ZEUS MACROS USED: #GENSECT, #OPENP, #GOTO, #WAIT, #SET, #CLOSEP, #REGS

ROUTINES CALLED: None

SIZE IN BYTES: 464 bytes

EXTERNAL REFERENCES: IECTTRNS, GETSAVE, ETA, DSPTCHR, FREEMEM

EXIT: To address in R14

ATTRIBUTES: REENTRANT

WRITTEN BY: Jean Garneau

SECTION 2: COMMAND PROCESSING

Zeus system modules that perform command processing functions are shown in the following list:

Name	Function
CMNDPR1	First-level command processing
CMNDPR2	Second-level command processing
SCAN	Scan and edit Zeus command strings
CON	Process console command
EOJ	Process LOG command
GCOUNT	Process counter status command
ICTBLID	Counter name table
RJE0000	First-Level Remote Job Entry (RJE)
RJE0100	Second-level RJE
RJE0200	Third-level RJE
RJE0300	Fourth-level RJE
STA	First-level interface to all status commands
WTL	Write message to system log
WTO	Write message to operator console

Zeus command syntax and purpose are described in Volume 1, Chapter 4. A description of each module follows.

NAME: CMNDPR1

FUNCTION: Process Zeus commands.

INPUT: R1 → First alpha character in command

R13 → 18-word save area

R14 → Return point address

R15 → Entry to CMNDPR1

OUTPUT: Command is executed, appropriate error message is sent or appropriate transient routine loaded and branched to.

PROCESS: First a check is made to see if a MODify is in process. If so, a return is made with a return code of 8. If not, SCAN is called and the command decoded. If it is a request for INFO, a branch is made to process INFO requests. If the command is POL, ( ), RJE, COR, or NOP a branch is made to the appropriate routines for processing these commands. For any other commands, CMNDPR2 is called to compile and insert CRT control information if necessary. Upon return, if the command was completely processed, immediate return is made. If an invalid verb was found by CMNDPR2, a branch is made to TRANLOAD to try to find the appropriate routine for processing the allegedly invalid verb. If unsuccessful, the appropriate error message is sent and return is made. If a valid command was incompletely processed by CMNDPR2, processing is completed within CMNDPR1--(CRE,COP,MOD,DEL). Return is made with a return code of 0 if processing was successfully completed and with a return code of 8 if not.

ZEUS MACROS USED: #OPENP, #GENSECT, #GOTO, #SET, #CALL, #WAIT, #LOAD, #DELETE, #CLOSEP, #REGS, #GETMEM, #FREEMEM #CONVRT, #MOVE

ROUTINES CALLED: CMNDPR2, STACK, RJE0000, SCAN, EOJ, CONSOLE, SUBMIT, MSCHD, CRTREAD, RESET.

SIZE IN BYTES: 2998 bytes

EXTERNAL REFERENCES: GETSAVE, DSPTCHR, IECTTRNS, FREEMEM, MOVE000, LDR000, GETMEM, SCAN, CMNDPR2, NTPIO, ATE, EMPTY, RESET, STACK, ETA, CRTREAD, SCANKEY, IFILE, CMNDPR1, SUBMIT, MSCHD, RJE0000, STATUS, EOJ, CONSOLE, LTE, CNVT000.

EXIT: Address in R14

ATTRIBUTES: REENTRANT

WRITTEN BY: Jean Garneau

NAME: CMNDPR2

FUNCTION: Called by CMNDPR1 to complete processing of some Zeus commands and to compile and partially complete processing of others.

INPUT: R1 → Command left parenthesis

R13 → 18-word save area

R14 → Return point address

R15 → Entry point to CMNDPR2

AQWORK → Work area

OUTPUT: Updated buffer and TCB

PROCESS: The appropriate second-level subroutine is branched to, depending on the command being processed. The second-level subroutines decode the commands and turn on the appropriate bits in the TCB to indicate which command is being processed and whether the processing has been completed. Third-level routines are branched to if necessary to determine file and record, and to identify literals. Fourth-level routines are branched to, if necessary, to process literals and record keys. Fifth-level routines are branched to get next available buffer location. Each level of routines returns to the next higher level, eventually getting to the first-level routines which insert the appropriate prefix, set the return code, and return to CMNDPR1.

ZEUS MACROS USED: #OPENP, #REGS, #GENSECT, #GOTO, #CALL, #SET, #CLOSEP, #FINDZVT, #FREEMEM, #CONVRT, #MOVE

ROUTINES CALLED: RESET, EMPTY

SIZE IN BYTES: 2183 bytes

EXTERNAL REFERENCES: GETMEM, FREEMEM, ZEUSSVC, CNVTOOO, EMPTY, RESET, DFILE, RFILE, FFILE, IFILE, QFILE, SCANKEY

EXIT: Address in R14

ATTRIBUTES: REENTRANT

WRITTEN BY: Jean Garneau

NAME:	SCAN
FUNCTION:	To determine if a character string contains a Zeus command and if so to edit it.
INPUT:	R0 → Area in which to return SCANned command. R1 → Beginning of character string R12 → TCB R13 → 18-word save area R14 → Return address R15 → Entry point to SCAN
OUTPUT:	Edited command in specified area, R1 → command terminator.
PROCESS:	The character string is scanned for the first nonblank character. If this character is not a left parenthesis, return is made. If it is a left parenthesis, the scan continues. If there are blanks, these are removed unless part of a literal. A literal is enclosed in balanced single quotes and is left as is unless it contains two consecutive single quotes, in which case one is removed. If unbalanced command delimiters or quotes are encountered, or the maximum length is exceeded, the appropriate message is sent to the terminal and the command is ignored.
ZEUS MACROS USED:	#OPENP, #GENSECT, #COUNT, #GOTO, #CLOSEP, #REGS
ROUTINES CALLED:	STACK
SIZE IN BYTES:	726 bytes
EXTERNAL REFERENCES:	GETSAVE, ICOUNT, FREEMEM, STACK
EXIT:	Address in R14
ATTRIBUTES:	REENTRANT
WRITTEN BY:	Jean Garneau

NAME: CON

FUNCTION: Formats and outputs to the requesting terminal a copy of the contents of the OS operator console buffers and related information.

INPUT:

R11 → ZVT

R12 → TCB

R13 → 18-word save area

R14 → Return address

R15 → Entry point to CON

OUTPUT: Formatted output buffer

PROCESS: CON is a transient module invoked by CMNDPR1 as a result of a CON command. It stacks the following output for the requesting terminal:

(1) Contents of the console input buffer.

(2) Contents of the console output buffers.

(3) List of outstanding replies by number and requesting task RB name.

(4) List of tasks in dispatching priority order giving jobname, stepname, and active RB name.

ZEUS MACROS USED: #REGS, #OPENP, #GOTO, #CALL, #CLOSEP, #GENSECT

ROUTINES CALLED: STACK

SIZE IN BYTES: 543 bytes

EXTERNAL REFERENCES: None

EXIT: Address in R14

ATTRIBUTES: REENTRANT, TRANSIENT

WRITTEN BY: Jean Garneau

NAME:	EOJ
FUNCTION:	Performs Zeus system shutdown functions.
INPUT:	R1 → (EOJ) command R11 → ZVT R12 → TCB R14 → Return address R15 → Entry point to EOJ
OUTPUT:	None
PROCESS:	EOJ is a transient module invoked by an EOJ command from CMNDPR1. EOJ scans all TCBs in the system to determine if any ports are logged on. If ports are logged on, a message is stacked for the requester's terminal indicating that EOJ failed. If all ports are logged off, EOJ forces completion of the RDRINTPR's outstanding #STIMER and returns to CMNDPR1. The RDRINTPR then handles final EOJ processing.
ZEUS MACROS USED:	#REGS, #OPENP, #GOTO, #CLOSEP, #SET, #TTIMER, #CALL, #GENSECT
ROUTINES CALLED:	STACK
SIZE IN BYTES:	186 bytes
EXTERNAL REFERENCES:	None
EXIT:	Address in R14
ATTRIBUTES:	REENTRANT, TRANSIENT
WRITTEN BY:	Jean Garneau

NAME:	GCOUNT
FUNCTION:	Formats and writes on a terminal the contents of the Zeus performance registers.
INPUT:	R11 → ZVT R13 → 18-word save area R14 → Return point address R15 → Entry point to GCOUNT
OUTPUT:	GCOUNT calls STACK once for each nonzero counter value, passing it a line of text containing a counter value and description from CSECT 'ICTBLID'.
ZEUS MACROS USED:	#OPENP, #REGS, #COUNT, #GOTO, #CLOSEP, #CALL, #FREEMEM
ROUTINES CALLED:	STACK
SIZE IN BYTES:	408 bytes
EXTERNAL REFERENCES:	ICOUNT1, ICOUNT2, GETSAVE, ICTBLID, ICTBL, ICTLGTH, STACK, FREEMEM
EXIT:	Address in R14
ATTRIBUTES:	REENTRANT, TRANSIENT
WRITTEN BY:	Jean Garneau

NAME:	ICTBLID
FUNCTION:	Driver for GCOUNT routine.
INPUT:	None
OUTPUT:	None
PROCESS:	Nonexecutable macro-operated table input to the GCOUNT routine.
ZEUS MACROS USED:	None (internal macro).
ROUTINES CALLED:	None
SIZE IN BYTES:	266 bytes
EXIT:	N/A
ATTRIBUTES:	N/A
WRITTEN BY:	Jean Garneau

NAME:	RJE0000
FUNCTION:	Performs the services requested by (RJE) 'text' commands.
INPUT:	R1 → (RJE text) R12 → TCB R13 → 18-word save area R14 → Return point address R15 → Entry point to RJE0000
OUTPUT:	R15 = 4 The input command has been replaced by a new command to be executed. R15 = 0 No new command has been created.
PROCESS:	RJE0000 is called by CMNDPR1 when an RJE command is encountered. It examines the text portion of the command to determine which function is to be performed, and branches to the appropriate routine.
ZEUS MACROS USED:	#REGS, #GENSECT, #OPENP, #GOTO, #CALL, #WAIT, #ENQ, #DEQ, #CLOSEP, #CONVRT
ROUTINES CALLED:	STACK, NTPIO
SIZE IN BYTES:	2488 bytes
EXTERNAL REFERENCES:	GETSAVE, CNVT000, ETA, STACK, RFILE, NTPIO, DSPTCHR, ENQCTRL, FREEMEM
EXIT:	Address in R14
ATTRIBUTES:	REENTRANT
WRITTEN BY:	William Underhill

NAME: RJE0100

FUNCTION: RJE0100 is a resident module that handles the OS job submission function of the Zeus RJE subsystem.

INPUT: R11 → ZVT

R13 → 18-word save area

R14 → Return point

R15 → Entry point to RJE0100

OUTPUT: Updated job submission queue

PROCESS: RJE0100 is attached as a Zeus task which waits on a #STIMER interval (currently set at 15 minutes). When the interval is completed (RJE0000 may force completion of the outstanding #STIMER in a VS system or upon encountering certain thresholds), RJE0100 loads and branches to RJE0200 which determines if any jobs have been placed in the Zeus RJE queue during the past interval. Upon return from RJE0200, a return code is checked to determine whether or not to load and execute REJ0300, which processes the job directory table created by RJE0200. Finally, RJE0100 checks a bit set by the EOJ processor (EOJ may force completion of the outstanding #STIMER). *If the bit is not set, RJE0100 reissues the #STIMER and* waits for the next interval to complete. If the bit is set, RJE0100 chains through the free queue elements in the Zeus memory subpool and returns them all to OS via a FREEMAIN macro and then issues an operator message requesting termination of Zeus.

ZEUS MACROS USED: #REGS, #GENSECT, #OPENP, #FINDZVT, #GOTO, #SET, #STIMER, #DELETE, #LOAD.

ROUTINES CALLED: None

SIZE IN BYTES: 552 bytes

EXTERNAL REFERENCES: ZEUSSVC, ITIMER, RFILE, RJEFILE, DSPTCHR, LDR0000

EXIT: N/A

ATTRIBUTES: Attached Task

WRITTEN BY: William Underhill

NAME:	RJE0200
FUNCTION:	Creates a table of RJE text file pointers from the RJE job submission queue and resets the queue to empty status.
INPUT:	R0 → Table of DECBs to be used in I/O processing. R11 → ZVT R13 → 18-word save area R14 → Return point address R15 → Entry point to RJE0200
OUTPUT:	R15 = 0 if queue was empty, or R15 = address of created table.
PROCESS:	RJE0200 #ENQs on the RJE text file to temporarily prevent updating of the submission queue by RJE0000. It then allocates a buffer and reads the submission queue record (Record 7) from the RJE text file. If the queue is empty, RJE0200 #DEQs and returns. If the queue is non-empty, a buffer equal to the size of the active part of the queue is allocated and the queue information is copied to the buffer. The queue size is then set to zero, the record is rewritten and RJE0200 #DEQs from the file and returns.
ZEUS MACROS USED:	#REGS, #GENSECT, #OPENP, #ENQ, #WAIT, #GOTO, #DEQ, #FREEMEN, #CLOSEP, #GETMEM
ROUTINES CALLED:	None
SIZE IN BYTES:	474 bytes
EXTERNAL REFERENCES:	None
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable, TRANSIENT
WRITTEN BY:	William Underhill

NAME: RJE0300

FUNCTION: Reads, formats, and writes the submitted RJE file records into a partioned data set and starts an OS reader on the PDS member.

INPUT:
R0 → Table of DECBs to be used for I/O
R1 → Table of submitted RJE file record numbers
R11 → ZVT
R13 → 18-word save area
R14 → Return point address
R15 → Entry point to RJE0300

OUTPUT: Batch jobs submitted to OS/VS Reader

PROCESS: For each record given in the input table, RJE0300 does the following:

(1) Translates record to EBCDIC
(2) Scans for carriage return character
(3) Moves line to 80-character output buffer and pads with blanks
(4) If line is a job or exec card, writes line on *operator console*
(5) Writes output buffer to PDS when full. When all records have been processed, RJE0300 calls the MSCHD routine with a "Start Reader" command specifying the PDS member just created.

ZEUS MACROS USED: #REGS, #GENSECT, #OPENP, #GETMEM, #WAIT, #GOTO, #CALL, #CLOSEP

ROUTINES CALLED: MSCHD

SIZE IN BYTES: 1,197 bytes

EXTERNAL REFERENCES: None

EXIT: Address in R14

ATTRIBUTES: Serially reusable, TRANSIENT

WRITTEN BY: William Underhill

NAME: STA

FUNCTION: Formats and outputs the contents of specified counters or memory locations.

INPUT:
R1 → (STAx)
R11 → ZVT
R12 → TCB
R14 → Return address
R15 → Entry address to STA

OUTPUT:
R15 = 0 Normal return
R15 ≠ if error encountered
Formatted Buffer

PROCESS: STA is a transient module loaded by CMNDPR1 in response to a STA command. The fourth character of the command determines the service to be performed as follows:

- (STAC) Outputs to the requesting terminal the contents of the Zeus system performance counters via the GCOUNT subroutine.
- (STAM) Outputs a specified portion of contiguous main memory locations in hex character and EBCDIC format.
- (STAZ) Outputs the Zeus Vector Table (ZVT) in (STAM) format.
- (STAT) Outputs the requesting terminals TCB in (STAM) format.
- (STAP) Outputs a list of log on names for the active ports on the system.

ZEUS MACROS USED: #REGS, #GENSECT, #OPENP, #GOTO, #CALL, #CLOSEP

ROUTINES CALLED: GCOUNT, CHTB, STACK

SIZE IN BYTES: 630 bytes

EXTERNAL REFERENCES: None

EXIT: Address in R14

ATTRIBUTES: REENTRANT, TRANSIENT

WRITTEN BY: Jean Garneau

NAME:	WTL
FUNCTION:	Sends messages to the WRITE-TO-PROGRAMMER log.
INPUT:	R1 → (WTL 'message') R13 → 18-word save area R14 → Return point address R15 → Entry point to WTL
OUTPUT:	R15 = 0 normal return R15 ≠ if syntax error in command Output message to Zeus SYSOUT listing
PROCESS:	The WTL process is identical to WTO except that, when the OS WTO macro is executed, the MCSFLAGS field indicates ROUTCDE = (11).
ZEUS MACROS USED:	#REGS, #GOTO
ROUTINES CALLED:	None
SIZE IN BYTES:	228 bytes
EXTERNAL REFERENCES:	None
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable, TRANSIENT
WRITTEN BY:	William Underhill

NAME:	WTO
FUNCTION:	Sends messages to the operator console
INPUT:	R1 → (WTO 'message') R11 → ZVT R13 → 18-word save area R14 → Return point R15 → Entry point to WTO
OUTPUT:	R15 = normal return R15 ≠ if syntax error in command Output to operator console
PROCESS:	WTO is a transient routine invoked by CMNDPR1 when a WTO command is encountered. WTO scans the message contained in apostrophes, builds the list form of an OS WTO command, and executes it.
ZEUS MACROS USED:	#REGS, #GOTO
ROUTINES CALLED:	None
SIZE IN BYTES:	208 bytes
EXTERNAL REFERENCES:	None
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable, TRANSIENT
WRITTEN BY:	William Underhill

SECTION 3: SYSTEM SUPPORT

The program modules that perform system support functions are listed as follows:

Name	Function
ATCHOO	Create Zeus Task
CHTB	Convert hexadecimal to binary
CNVT000	Convert numeric character to binary
CNTXCTL	Transfer control to 1 of 3 count modules
DSPTCHR	Schedule all Zeus tasks
ENQCTRL	Provide serialized access to any resource
FREEMEM	Release main memory
GETMEM	Acquire main memory
GETSAVE	Acquire a register save area
ICOUNT	Manipulate Zeus counter
ICOUNT1	Manipulate Zeus counter
ICOUNT2	Manipulate Zeus counter
ITIMER	Manipulate Zeus timer queue
LDR0000	First-level transient-load module
LDR0100	Second-level transient-load module
LDR0200	Third-level transient-load module
MOVE000	Move data within main memory
MOVE010	Move data and convert to hex within main memory
MSCHD	Interface to the Master Scheduler
PROB	Return task to problem program state
QUEMGR	Manipulate queue(s)
RCVRMGT	Process resource interlock condition
SCANKEY	Hash key
SUPV	Place task in supervisor state with protect key 0
TABLES	Translate tables (not executable module)

System support concepts are described in Volume 1, Chapter 5. A description of each subroutine follows.

NAME:	ATCHOO
FUNCTION:	Create A Zeus Task
INPUT:	R0 → Address of routine to attach R14 → Return address R15 → Entry address to ATCHOO
OUTPUT:	Attached Task
PROCESS:	ATTCHOO is used via the #ATTACH macro to create a Zeus task. After making itself dispatchable, ATCHOO BALs to the specified routine. When this routine issues a #WAIT, the DSPTCHR will return control to ATCHOO which will, in turn, return control to the task that issued the #ATTACH. If the attached task returns, it will enter a subroutine of ATCHOO, which will then call the DSPTCHR with a special code so that the DSPTCHR will never again dispatch this task.
ZEUS MACROS USED:	#REGS, #COUNT, #GETMEM, #WAIT, #FREMEM
ROUTINES CALLED:	None
SIZE IN BYTES:	150 bytes
EXTERNAL REFERENCES:	ICOUNT, GETMEM, DSPTCHR, FREEMEM
EXIT:	Address in R14
ATTRIBUTES:	Reentrant
WRITTEN BY:	J. Garneau

NAME: CHTB

FUNCTION: Converts a hexadecimal character string to its equivalent binary value.

INPUT:
R0 = # of characters to scan (1-8)
R1 → Start of character string
R11 → ZVT
R13 → 18-word save area
R14 → Return point
R15 → Entry point to CHTB

OUTPUT:
R0 = Binary value of field
R1 → Input character at which processing terminated
R15 = 0 normal return
= 4 if scan limit (R0) is invalid

PROCESS: CHTB scans the input string from low to high order positions. Each character must be an EBCDIC 0-9 or A-F. The character is converted to a binary value (0-15) and added to 16 times the previous cumulative value. Processing terminates when a non 0-9, A-F character is encountered or the scan limit is reached.

ZEUS MACROS USED: None

ROUTINES CALLED: None

SIZE IN BYTES: 148 bytes

EXTERNAL REFERENCES: None

EXIT: Address in R14

ATTRIBUTES: Serially reusable, TRANSIENT

WRITTEN BY: William Underhill

NAME:	CNVT000
FUNCTION:	Converts numeric character fields to binary.
INPUT:	R1 → Start of field R13 → 18-word save area R14 → Return address R15 → Entry point to CNVT000
OUTPUT:	R0 = Value of Field R1 → Nonnumeric which terminated the field R15 = 0 if R0 is valid ≠ 0 if an error occurred
PROCESS:	CNVT000 scans the input field for the first non 0-9 character, packs the resultant field, converts it to binary and returns to the caller.
ZEUS MACROS USED:	#REGS
ROUTINES CALLED:	None
SIZE IN BYTES:	86 bytes
EXTERNAL REFERENCES:	None
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable
WRITTEN BY:	J. Garneau

NAME: CNTXCTL

FUNCTION: To transfer control to ICOUNT, ICOUNT1 or ICOUNT2

INPUT: R15 = base

1(R14) = code
- Ø - go to ICOUNT
- 1 - go to ICOUNT1
- 2 - go to ICOUNT2

OUTPUT: N/A

PROCESS: The one-byte field at R14+1 is checked to determine to which routine control is to be transferred. The address of the appropriate routine is loaded into R15 and branched to.

ZEUS MACROS USED: #GOTO, #REGS

ROUTINES CALLED: None

SIZE IN BYTES: 52 bytes

EXTERNAL REFERENCES: ICOUNT1, ICOUNT2

EXIT: To address in R15

ATTRIBUTES: Serially reusable

WRITTEN BY: J. Garneau

NAME: DSPTCHR

FUNCTION: Distribute CPU cycles to all tasks active within the ZEUS partition or region.

INPUT: R1 → ECB address list. Last entry in list has sign bit on

R13 → 18-word save area

R14 → Caller return point address

R15 → Entry point of DSPTCHR

OUTPUT: ECB address list added to DSPTCHR stack if copy does not already exist there. First byte of first ECB address modified to reflect status of list.

RØ = ECB posted,

R15 = Relative offset of posted ECB.

PROCESS: Upon entry, the DSPTCHR first stores the callers' registers in the callers' save area, then checks the second bit of the control byte (first byte of the first ECB address in the passed list). If the bit is on, the list is already in the DSPTCHR stack. If the bit is off, the list is added to the stack and this bit is turned on.

The add operation is accomplished by scanning the list for a sign bit in an ECB address word, checking for adequate space in the stack, and then moving the list into the stack area. An entry is made in an associated stack that contains a count of the ECBs belonging to the task and the address of the task's source ECB list.

After the dispatcher is certain the ECB list is contained in the stack, it moves the first ECB from the source list to the first word of the caller's save area. The caller's save area address is then stored in the first word of the source ECB list, and the third bit of the control byte is set on to indicate that the caller is "dispatchable."

Dispatching is done on a "round robin" basis. Each entry to the DSPTCHR causes all ECBs in the stack to be checked for completion. The search always begins with the ECB following the last one dispatched. If the end of stack is reached before all ECBs are checked, the search is reinitialized to the top of the stack and continues until they are all checked.

When an ECB is posted complete the DSPTCHR searches the stack to locate it. It then retrieves this ECBs register save area, puts the address of the completed ECB in register 0, its offset into R15, reloads registers 0 to 14, and then branches to the address contained in register 14.

When the entire list has been searched without locating any completed ECBs, an OS WAIT macro is issued to the DSPTCHR ECB stack.

ZEUS MACROS USED:	#GOTO, #SET, #COUNT
ROUTINES CALLED:	None
SIZE IN BYTES:	948 bytes
EXTERNAL REFERENCES:	ICOUNT
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable
WRITTEN BY:	J. Garneau

NAME:	ENQCTRL
FUNCTION:	Provide serialized access to any resource.
INPUT:	R1 → Resource Queue Element (RQE) R1 > 0 IF ENQ Request < 0 IF DEQ Request R13 → 18-word save area R14 → Return address R15 → Entry address
OUTPUT:	R15 = 0 if caller was given control of the resource R15 ≠ 0 if the requested resource is currently in use
PROCESS:	#ENQ - The input RQE is added to a fifo ordered queue maintained by ENQCTRL. The queue is ordered by the second word of the RQE, which has been loaded with the resource name as specified in the #ENQ macro. R15 is set depending upon whether or not a previous RQE with the same sort field (name) existed. #DEQ - The specified RQE is removed from the queue. If the caller was first in the queue for the given resource (i.e., had control of the resource), and if the next queue entry specifies the same resource name, the second entry is given control. This is done by posting the ECB whose address is the third word of the RQE.
ZEUS MACROS USED:	#FINDZVT, #REGS, #GENSECT, #GOTO, #QUE, #QFB
ROUTINES CALLED:	None
SIZE IN BYTES:	219 bytes
EXTERNAL REFERENCES:	ZEUSSVC, QUEMGR
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable
WRITTEN BY:	Jean Garneau

NAME:	FREEMEM
FUNCTION:	To return memory to the ZEUS pool or to the OS pool
INPUT:	R1 → Memory to be freed R1 - 4 = Size of memory to be freed R14 → Return address R15 → Entry address
OUTPUT:	Specified amount of memory at specified address freed
PROCESS:	The amount and address of the memory to be freed are obtained. If the memory belongs to the OS pool, a FREEMAIN is issued. If it belongs to the ZEUS pool, the associated queue elements are updated. If the memory to be freed is not on a double word boundary, an error message is sent to the console prior to completing the processing of the request. Next, the core queue is checked to see if any routines are waiting for memory. If so, the requests are satisfied, if possible, prior to return.
ZEUS MACROS USED:	#REGS, #GENSECT, #FINDZVT, #GOTO, #COUNT
ROUTINES CALLED:	None
SIZE IN BYTES:	576 bytes
EXTERNAL REFERENCES:	ZEUSSVC, ICOUNT2, GETMEM
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable
WRITTEN BY:	William Underhill

NAME:	GETMEM
FUNCTION:	To obtain memory for use by ZEUS routines
INPUT:	R1 → CQE R0 = Amount requested
OUTPUT:	R14 → Return address R15 → Entry address
PROCESS:	The size of the memory request is obtained. The ZPOOLA field of the ZVT is checked to see if the ZEUS pool is available. If so, the requested memory is gotten from the ZEUS pool. If unavailable the memory is gotten from the OS pool. If memory cannot be gotten--because none is available, a pool is exhausted, a limit reached, or because of a boundary error--an attempt is made to free a transient area to satisfy the request. If the memory is obtained, it is marked with the pool ID and the requester's ECB is posted.
ZEUS MACROS USED:	#GOTO, #REGS, #GENSECT, #FINDZVT, #COUNT, #CALL
ROUTINES CALLED:	QADD, LDR0200
SIZE IN BYTES:	573 bytes
EXTERNAL REFERENCES:	ZEUSSVC, ICOUNT, ICOUNT2, SUPV, LDR0200, QADD, ZMEM, PROB
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable
WRITTEN BY:	William Underhill

NAME:	GETSAVE
FUNCTION:	Acquire a save area, waiting for memory to become free if necessary.
INPUT:	R0 → Terminal control block (TCB) of active task R1 → Size of requested save area in bytes R11 → TCB R13 → 18-word save area R14 → Return address R15 → Entry address to GETSAVE
OUTPUT:	R1 → Address of gotten memory
PROCESS:	GETSAVE is used in conjunction with the #OPENP macro when TYPE=RENT and TCBREG=n are specified. It handles and restores registers in a nonstandard manner. GETSAVE executes a #GETMEM macro for the amount of memory specified in R1, using a reserved save area (pointed to by the TCB) in order to wait for memory to become available if necessary.
ZEUS MACROS USED:	#GENSECT, #REGS, #COUNT, #GETMEM
ROUTINES CALLED:	None
SIZE IN BYTES:	104 bytes
EXTERNAL REFERENCES:	ICOUNT, GETMEM, DSPTCHR
EXIT:	Address in R14
ATTRIBUTES:	Serially reusable
WRITTEN BY:	Jean Garneau

NAME:	ICOUNT
FUNCTION:	Increments the specified ZEUS system performance counter by 1.
INPUT:	R14 → A one-byte field containing the counter number and return address - 4 R15 → Entry address
OUTPUT:	R15 = Current value of counter
PROCESS:	The counter number is checked against the range of available counters. If invalid, counter zero is used as an overflow counter. ICOUNT increments the specified counter by 1 and returns to the caller.
ZEUS MACROS USED:	#REGS, #GOTO
ROUTINES CALLED:	None
SIZE IN BYTES:	224 bytes
EXTERNAL REFERENCES:	ICOUNT, ICOUNT2, ICOUNT1
EXIT:	Address in R14 + 4
ATTRIBUTES:	Serially reusable
WRITTEN BY:	Jean Garneau

NAME:	ICOUNT1
FUNCTION:	Adds a given number to the specified ZEUS counter.
INPUT:	R1 = Value to be added to counter R14 → One-byte counter number and return address - 4 R15 → Entry address
OUTPUT:	R15 = Current value of counter
PROCESS:	Same as entry point ICOUNT, except that the value of R1 is added to the counter rather than adding a 1.
ZEUS MACROS USED:	None
ROUTINES CALLED:	None
SIZE IN BYTES:	20 bytes
EXTERNAL REFERENCES:	ICOUNT
EXIT:	Branches to ICOUNT
ATTRIBUTES:	Serially reusable
WRITTEN BY:	Jean Garneau

NAME:	ICOUNT2
FUNCTION:	Loads a given value into the specified ZEUS counter.
INPUT: OUTPUT:	Same as ICOUNT1
PROCESS:	Same as ICOUNT1, except that the value in R1 is loaded into the counter rather than added.
ZEUS MACROS USED:	None
ROUTINES CALLED:	None
SIZE IN BYTES:	20 bytes
EXTERNAL REFERENCES:	ICOUNT
EXIT:	Branches to ICOUNT
ATTRIBUTES:	Serially reusable
WRITTEN BY:	Jean Garneau

NAME: ITIMER

FUNCTION: To set, test, or cancel a timer interval, or force a previously set timer interval to 0.

INPUT: R1 = Timer Queue Element (TQE)

R0 = One of following codes:

0 - Add TQE to queue (#STIMER)

4 - Return interval remaining for this TQE (#TTIMER remaining).

8 - Remove TQE from queue (#TTIMER cancel)

12 - Force TQE completion by making current interval 0 (#TTIMER complete)

R13 → 18-word save area

R14 → Return address

R15 → Entry address ITIMER

OUTPUT: Timer Queue processed as specified.

PROCESS: Upon initial entry, which is via an #ATTACH to create the task, a branch is made to ICOMP2. The initial branch instruction is NOPed. The address of the timer queue address word (QAW) is saved and R12, the base for IEXIT is saved. The following routines (which are documented below) are branch and linked to by this task which never terminates: ILAPSD, IPOST, IRESET. The completion bit in the ECB is turned off and IEXIT is BALed to to process any queued exits. The entire process is continuously repeated.

Upon any entry subsequent to the initial one ICOMP2 is not branched to. The TQE pointer and request type are saved. The address of the QAW is obtained. The following routines, (which are documented below), are BALed to: IDSBL, ILAPSD, IENBL. Upon return ITEST is branched to if the call was via the #TTIMER macro instruction. If the call was via #STIMER, IADD and IPOST are BALed to. Upon return, the completion bit in the ECB is turned off and IRESET and IEXIT are BALed to. Return is made to the address in R14.

IEXIT: A test is made for any queued exits. If there are none, return is made. If any, a message is sent to the operator and the program terminates with an ABEND I,DUMP.

ITEST: If the request is for the interval remaining IPOST is BALed to, the completion bit in the ECB is turned off, IRESET and IEXIT are BALed to, and return is made via

R14. If the request is to cancel or force to completion, the interval is set to 0, the ECB or EXIT address is saved, type = cancel is indicated, IPOST is branch and linked to, the ECB or EXIT address is restored, the ECB completion bit is turned off, IRESET and IEXIT are BALed to and return is made.

IDSBL: The system mask is set to disable interrupts.

IENBL: The system mask is set to enable interrupts.

ILAPSD: This routine determines the amount of time that has elapsed since the interval timer was activated and returns this value in R6. If the interval timer is still active when this routine is called, it will be deactivated and the time left will be used in the computation.

IPOST: The timer queue is gone through and each TQE with an interval remaining of 0 is dequeued. If the completion address is that of an exit address, it is queued on the exit queue; if it is an ECB address, the ECB is posted; if it is zero, no action is initiated, and the process continues. If the remaining interval is greater than 0, the next TQE is checked. When the end of the queue is reached, return is made via R11.

IADD: This routine puts a TQE into the timer queue. The contents of R6 (elapsed time) are added to the TQE value (in timer units).

IRESET: This routine obtains the address of the first TQE, if any on queue, saves the current interval, resets the interval timer (via STIMER macro), turns on the low order bit of the QAW, and returns.

ICOMP: This routine is entered from OS when the interval timer completes. The registers are saved, and the low order bit of the QAW is set to 0 to show that the timer is down. The time of day in timer units is obtained (via TIME macro) and saved. The timer ECB is posted complete to allow the timer to be restarted. The registers are restored, and return is made.

ZEUS MACROS USED: #REGS, #OPENP, #GOTO, #SET, #CLOSEP, #CALL, #COUNT, #WAIT

ROUTINES CALLED: QADD, PROB, SUPV

SIZE IN BYTES: 664 bytes

EXTERNAL REFERENCES: ICOUNT, SUPV, PROB, QADD, DSPTCHR

EXIT: To address in R14

ATTRIBUTES: Serially reusable

WRITTEN BY: Jean A. Garneau

NAME: LDR0000

FUNCTION: Called by the #LOAD macro to obtain the memory address of a transient module if available. Also called by #DELETE macro to relinquish control of a module.

INPUT: R0 → 8-byte name of module requested

R1 = Address of three-word load request block (LRB)

= 0 if no LRB specified with #LOAD macro

= 1 if called by #DELETE macro

R9 → TABENTRY Table

R13→ 18-word save area

R14→ Return address

R15→ Entry address

OUTPUT: R15= Address of module if called by #LOAD

= 0 if module not available

= 4 normal return from #DELETE processing

= -1 if module not available and load request block was added to the queue of modules to be loaded. Indicates that a #WAIT must be issued.

PROCESS: #LOAD - LDR0000 seeks the requested name in a transient module table. If the module is available in memory, its responsibility count is incremented and the address is returned to the caller. If the module exists but is not available, the request is queued only if an LRB was specified with the #LOAD macro. A BLDL macro is issued on the ZEUS transient module PDS if the module is not found in the name table (i.e., for the first load). #DELETE - The module is found in the transient module table and its responsibility count is decremented.

ZEUS MACROS USED: #REGS, #OPENP, #GOTO, #CALL, #CLOSEP, #COUNT, #QUE, #QFB

ROUTINES CALLED: QADD

SIZE IN BYTES: 1740 bytes

EXTERNAL REFERENCES: ICOUNT, TFILE, LRBQ, QUEMGR, LDRECB

EXIT: Address in R14

ATTRIBUTES: SERIALLY REUSABLE

WRITTEN BY: W. Underhill

NAME:	LDR0100
FUNCTION:	Loads requested transient modules into memory and posts the associated requesting tasks.
INPUT:	R9 → TABENTRY Table R13 → 18-word save area R15 → Entry address
OUTPUT:	Loaded Modules
PROCESS:	LDR0100 is attached during Zeus initialization and immediately waits on a "no-work" ECB. When the no-work ECB is posted, LDR0100 de-queues the first LRB on the LRB chain, allocates memory for the module, issues a point macro on the transient file PDS, reads the module into memory, and posts the associated ECB with the module's address. The module's length and TTR are found in the transient module table. LDR0100 repeats this process for each LRB on the queue. When finished it waits on its no-work ECB.
ZEUS MACROS USED:	#REGS, #OPENP, #WAIT, #GOTO, #COUNT, #QUE, #QFB
ROUTINES CALLED:	None
SIZE IN BYTES:	472 bytes
EXTERNAL REFERENCES:	ICOUNT, TFILE, DSPTCHR, GETMEM, FREEMEM, QUEMGR
EXIT:	N/A
ATTRIBUTES:	Attached Task
WRITTEN BY:	William Underhill

NAME: LDR0200

FUNCTION: Frees the memory being occupied by one or all transient modules that are not currently in use.

INPUT: R1 = 0 Free all unused modules

= n>0 Free the least recently used module of size n or greater.

R9 → TABENTRY Table

R13 → 18-word save area

R14 → Return address

R15 → Entry address

OUTPUT: R15 = 0 If memory freed

≠ 0 If no memory could be freed.

PROCESS: LDR0200 scans the transient module table for modules with a zero responsibility count. If R1 = 0 the memory occupied by each such module is freed. If R1 = n>0 each unused module's size is checked. When the table scan terminates, the least recently used module (as indicated by a time stamp field) of sufficient size (if any) is freed.

ZEUS MACROS USED: #REGS, #GOTO, #FREEMEM

ROUTINES CALLED: NONE

SIZE IN BYTES: 254 bytes

EXTERNAL REFERENCES: NUMMOD, MODTAB, FREEMEM

EXIT: Address in R14

ATTRIBUTES: SERIALLY REUSABLE

WRITTEN BY: W. Underhill

NAME:	MOVE000
FUNCTION:	MOVE000 moves data (1 to 32767 bytes) from one location to another.
INPUT:	R0 → Source Area R1 → Target Area R14 → 2-byte length field, R14 + 2 → return address R15 → Entry address to MOVE000
OUTPUT:	R1 = Address of last input byte + 1 R15 = 0 Normal return R15 ≠ 0 if input length <=0
PROCESS:	Data are moved, left to right, for the length specified.
ZEUS MACROS USED:	#REGS, #GOTO
ROUTINES CALLED:	NONE
SIZE IN BYTES:	100 bytes
EXTERNAL REFERENCES:	NONE
EXIT:	Address in R14 + 2
ATTRIBUTES:	SERIALLY REUSABLE
WRITTEN BY:	J. Garneau

NAME: MOVE010

FUNCTION: MOVE010 converts each input byte to two output hex character bytes.

INPUT:
R0 → Source Area
R1 → Target Area
R14 → 2-byte length field
R15 + 2 → return address
R15 → Entry address to MOVE010

OUTPUT:
R1 = Address of last input byte + 1
R15 = 0 normal return,
R15 = 4 if input length <2

PROCESS: Each input byte is converted to its hexidecimal two-byte equivalent. These bytes are stored in the target area. The length specified is that of the target area. Consequently, the number of source area bytes converted is equal to one-half the length (in bytes) value specified. The length is rounded down, if necessary, to an even number before conversion.

ZEUS MACROS USED: #REGS

ROUTINES CALLED: None

SIZE IN BYTES: 156 bytes

EXTERNAL REFERENCES: None

EXIT: Address in R14 + 2

ATTRIBUTES: Serially Reusable

WRITTEN BY: Jean Garneau

NAME: MSCHD

FUNCTION: MSCHD passes commands to the OS or VS Master Scheduler.

INPUT: R1 → Command
R13 → 18-word save area
R14 → Return address
R15 → Entry address to MSCHD

OUTPUT: Command sent to OS or VS.

PROCESS: The input string is scanned for an asterisk or ETX character, which defines the end of the string. MSCHD then enters supervisor state and protect key 0 and issues SVC 34 to pass the command to the Master Scheduler. If the sign bit is on in R1 at entry, the command is also written on the operator's console.

(NOTE: Command must be of format '(//y*' where y is an acceptable OS or VS command.)

ZEUS MACROS USED: #REGS, #OPENP, #LPSW, #CLOSEP

ROUTINES CALLED: None

SIZE IN BYTES: 326 bytes

EXTERNAL REFERENCES: None

EXIT: Address in R14

ATTRIBUTES: Serially reusable, TRANSIENT

WRITTEN BY: Jean Garneau

NAME:	PROB
FUNCTION:	Returns the Zeus system to problem program state and restores its assigned protect key.
INPUT:	R13 = Base R14 → Return address R15 → Entry address to PROB
OUTPUT:	Task in Problem State
ZEUS MACROS USED:	#GOTO, #LPSW, #REGS
ROUTINES CALLED:	None
SIZE IN BYTES:	73 bytes
EXTERNAL REFERENCES:	ZEUSSVC
EXIT:	Address in R14
ATTRIBUTES:	SERIALLY REUSABLE
WRITTEN BY:	W. Underhill

NAME: QUEMGR

FUNCTION: QUEMGR is used to add/insert, delete, or locate an element on a queue.

INPUT:
R0 → Queue Foundation Block (QFB)

R1 → Element or, if ID specified, 4-character ID.

R14 → Argument and return address (-4)

R15 → Entry point to QUEMGR

OUTPUT:
R1 → Element deleted or located
Element chained onto queue if ADD, removed from queue if DELETE.

R15 = Count of elements with identical ordering strings TYPE = (FIFO,ORDERED...) specified otherwise R15 not relevant.

PROCESS: The type of QFB is saved. A check is made to determine if the element to be processed is to go to either the top or bottom of an ordered queue. If so, the argument byte is saved. If the element is to be added to an unordered queue, a branch is made to this routine. The address of the last element on the queue is obtained from the QFB. The address of the previous element is obtained from the next address field of the last element. The address of the new element is stored in the last element field of the QFB and the previous last element address is stored in the new last element link word.

If the element is to be added to an ordered queue, bits 1 and 2 of the type field of the QFB are used to set the mask bits of a branch instruction that tests for a high or low compare condition. The queue to which the element is to be added is searched until equal IDs are found at which time it is determined whether it is a FIFO or LIFO operation. If FIFO, the element is added to the bottom of the queue of elements with the same ID; if LIFO, it is added to the top. In either case, the address chains are updated by storing the address of the next element in the current element and the address of the last element on the queue in the QFB.

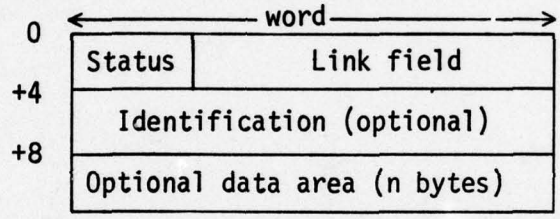
If the element is to be located or deleted, a check is made to determine if the search is to be made on ID. If so, the proper pointer is adjusted and the search is conducted in the same manner as that for addition to an ordered queue. When the correct element is found, the chain address is updated by storing the address of the next element in the next address field of the previous element. The last element field of the QFB is updated if necessary.

If the search is to be made by element rather than ID, the queue is searched until the correct element is found, at which time it is deleted by updating the next element address field of the previous element and the last element field of the QFB if necessary.

Bits in the argument field and their meanings are as follows:

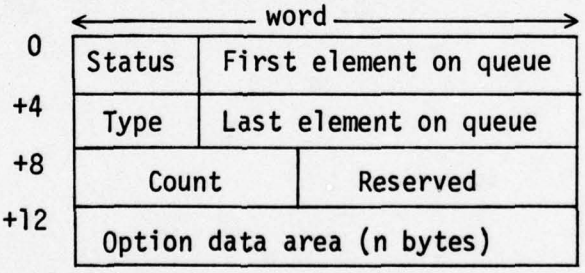
Bits	Argument
1...	delete/locate ID equal
.1..	delete/locate ID low
..1.	delete/locate ID high
11..	delete/locate ID equal or low
1.1.	delete/locate ID equal or high
...1	delete/locate/add to top of queue
.... 1...	delete/locate/add to bottom of queue
.... .1..	delete
.... ..1.	locate
....1	ID specified

The structure of elements in all queues is as follows:



For details on individual fields see element descriptions by type in Chapter 3, Section 2.

The structure of all QFBs are as follows:



For details on the STATUS and TYPE fields, see Chapter 3, Section 1.

ZEUS MACROS USED: #REGS, #GOTO, #SET

ROUTINES CALLED: None

SIZE IN BYTES:	354 bytes
EXTERNAL REFERENCES:	None
EXIT:	To address in R14 + 4
ATTRIBUTES:	Serially reusable
WRITTEN BY:	Jean Garneau

NAME:	RCVRMGT
FUNCTION:	RCVRMGT attempts to diagnose and correct memory request interlocks.
INPUT:	R13 → 18-word save area R14 → return address R15 → entry address of RCVRMGT
OUTPUT:	None
PROCESS:	RCVRMGT is attached as a Zeus task. It normally waits on an interval timer (currently set at 1 second). At the completion of each interval, when RCVRMGT gains control, it checks the memory wait queue. If the queue is empty, it reissues the #STIMER macro and waits for the next interval to complete. If the queue is non-empty, the first entry is checked for a X'FF' mark. If absent, RCVRMGT marks the entry with X'FF' and #STIMERS. If the mark is present, the queue entry has been on the queue since the last interval expired and a memory interlock is assumed to have occurred. RCVRMGT sets a bit indicating that the OS memory pool is to be used rather than the Zeus subpool and calls FREEMEM to attempt to allocate memory to the waiting tasks.
ZEUS MACROS USED:	#OPENP, #REGS, #GENSECT, #FINDZVT, #STIMER, #GOTO, #COUNT, #SET
ROUTINES CALLED:	None
SIZE IN BYTES:	316 bytes
EXTERNAL REFERENCES:	ZEUSSVC, ITIMER, ICOUNT, DSPTCHR, LDR0200, FREEMEM.
EXIT:	N/A
ATTRIBUTES:	Serially reusable
WRITTEN BY:	Jean Garneau

NAME:	SCANKEY
FUNCTION:	To hash a character string into a relative record number.
INPUT:	R1 → beginning of character string. R13 → 18-word save area. R14 → Return address. R15 → Entry address to SCANKEY.
OUTPUT:	R15 = relative record number. R1 → Edited character string used in hash operation.
PROCESS:	The input character string is scanned until an ETX is encountered or for a maximum of 40 bytes. All blanks and special characters are removed. The input character string is replaced with the edited character string (the KEY). The first four bytes of the key are logically added to the second four bytes and this total is logically added to the third four bytes. This number is made positive and divided by 1193. The result is the relative record number that is loaded into R15.
ZEUS MACROS USED:	#REGS, #OPENP, #CLOSEP
ROUTINES CALLED:	None
SIZE IN BYTES:	286 bytes
EXTERNAL REFERENCES:	None
EXIT:	Address in R14
ATTRIBUTES:	Serially Reusable, TRANSIENT
WRITTEN BY:	D. Shuford

NAME:	SUPV
FUNCTION:	Places the task in supervisor state, protect key zero
INPUT:	R13 → 18-word save area R14 → Return address R15 → Entry point
OUTPUT:	None
ZEUS MACROS USED:	#REGS, #GOTO, #LPSW
ROUTINES CALLED:	None
SIZE IN BYTES:	56 bytes
EXTERNAL REFERENCES:	ZEUSSVC
EXIT:	Address in R14
ATTRIBUTES:	Serially Reusable
WRITTEN BY:	W. Underhill

NAME: TABLES, ENTRY→ETL, LTE, ATE, ETA

FUNCTION: To provide translate tables for use by Zeus routines.

The tables provided are:

ETL	EBCDIC to 1050 line code.
LTE	1050 line code to EBCDIC.
ATE	ASCII to EBCDIC.
ETA	EBCDIC to ASCII
CTLTAB	

The tables have been modified as follows:

ETL

Upper case letters changed to lower case.

Carriage return changed to null.

Horizontal tab changed to restore.

Vertical tab changed to bypass.

Home changed to end of addressing.

End of transmission changed to end of block.

End of test changed to end of block.

Dot (.) changed to space.

Back slash changed to reader stop.

Up arrow changed to punch on.

Left arrow changed to punch off.

LTE

Lower case letters changed to upper case.

End of addressing changed to null.

End of block changed to end of text.

Space changed to dot (.).

Idle changed to idle (ASCII).

Bypass changed to idle.

Restore changed to idle.

Null changed to carriage return.

Reader stop changed to back slash.

Punch on changed to up arrow.

Punch off changed to left arrow.

All modifications (LTE) are made to both upper and lower case control characters.

INPUT:	N/A
OUTPUT:	N/A
PROCESS:	N/A
ZEUS MACROS USED:	None
ROUTINES CALLED:	None
SIZE IN BYTES:	1,024 bytes
EXTERNAL REFERENCES:	None
EXIT:	N/A
ATTRIBUTES:	Read Only
WRITTEN BY:	Doris Shuford

SECTION 4: INITIALIZATION

The Zeus initialization programs are presented in the following list:

Name	Function
INIT0	Open files, load external processors, and link to INIT1
INIT1	Process Zeus parameter cards, attach tasks, create control blocks, and transfer control to INIT2
INIT2	Create Zeus main memory pool and return control to INIT0

The initialization process is described in Volume 1, Chapter 6. A description of each subroutine follows.

NAME: INITO

FUNCTION: First-level Zeus initiation.

INPUT: R1 → External Processor Name
R13 → 18-word save area
R14 → Return point address
R15 → Entry point address to INITO

OUTPUT: Open files, and loaded external processor(s).

PROCESS: INITO first interrogates the parm field as passed via register 1. If a name is present, it is assumed to be an external processor name and an attempt is made to load it. If no external processor is requested, a dummy external processor is loaded. INITO then obtains the address of the ZVT and from it gets a list of DCBs to open, which it does next. Finally, it links to INIT1. Upon return, INITO initializes the external processor(s).

ZEUS MACROS USED: #OPENP, #GENSECT, #GOTO, #LPSW, #COUNT, #CLOSEP, #FINDZVT, #SET, #WAIT

ROUTINES CALLED: None

SIZE IN BYTES: 1,704 bytes

EXTERNAL REFERENCES: ICOUNT2, DDMOSNO1, GETSAVE, ZEUSSVC

EXIT: Address in R14

ATTRIBUTES: REENTRANT

WRITTEN BY: J. Garneau

NAME: INIT1

FUNCTION: Read Zeus parameter cards and create control blocks, and ATTACH tasks which will be required for operation.

INPUT: R1 → parameter list

R13 → 18-word save area

R14 → return address

R15 → Entry point to INIT1

Zeus parameter cards

OUTPUT: The following tasks ATTACHed:

ZITIMER - interval timer (1)
ZRJE0100 - reader interpreter (1)
ZWTR000 - writer (1)
ZRCVRMGT - recovery management (1)
ZLDR0100 - transient loader (1)
ZLINEIO - line (1 per line)
ZTERMIO - terminal (1 per terminal)

The following control blocks created:

TCB - Task control block (1 per terminal)
LCB - Line control block (1 per line)
PAL - Polling/Addressing list (1 per line)

PROCESS: SYSIN is OPENed and the required tasks ATTACHed. The parameter cards are read and the amount of main storage to be RESERVED, ALLOCATED, or LIMITED is determined. This value is stored in the ZPOOLA field of the ZVT. The line and terminals that are to be used are determined and the necessary control blocks created. When all parameter cards have been processed SYSIN is CLOSED and control is transferred to INIT2.

ZEUS MACROS USED: #OPENP, #GENSECT, #REGS, #FINDZVT, #WAIT, #LPSW, #GOTO, #ATTACH, #SET, #GETMEM

ROUTINES CALLED: None

SIZE IN BYTES: 1,832 bytes

EXTERNAL REFERENCES: None

EXIT: INIT2

ATTRIBUTES: Serially reusable

WRITTEN BY: Jean Garneau

NAME: INIT2

FUNCTION: Determines the type of memory allocation scheme to be used by Zeus.

INPUT: R1 → parameter list

R13 → 18-word save area

R14 → Return address

R15 → Entry point to INIT2

OUTPUT: ZPOOLA field of ZVT updated.

PROCESS: ZPOOLA word of the ZVT is interrogated. If ALLOCATE was specified, memory is acquired and structured into a Zeus Pool. If RESERVE was specified, memory is reserved for OS and what remains is structured into a Zeus Pool. If LIMIT or no parameters was specified, no Zeus Pool is structured. The ZPOOLA word of the ZVT before and after INIT2 is as follows:

Before:

Byte 1	bit 0	ALLOCATE
	1	RESERVE
	2	LIMIT
Bytes 2-4		value

After:

Byte 1	bit 0	No Zeus Pool
	bit 1	Limit specified
Bytes 2-4		Address of Zeus pool or limit value or zero.

ZEUS MACROS USED: #OPENP, #REGS, #FINDZVT, #GOTO, #CLOSEP, #SET

ROUTINES CALLED: None

SIZE IN BYTES: 396 bytes

EXTERNAL REFERENCES: ZEUSSVC

EXIT: Address in R14

ATTRIBUTES: Serially reusable

WRITTEN BY: Jean Garneau

Chapter 2

MACROS

Several macros have been created to reduce the amount of detail coding required to use Zeus functions. A number of these macros can be considered general purpose since they may be used in any environment. Some may be used after careful consideration of the effect of execution, and some may be used only within a system built on Zeus components and architecture.

A summary listing of available macros is contained in Table 1. Column 3 of the table indicates the level of utility of the individual macros. The symbols and notations used to represent macro syntax are presented in Figure 3. A description of each macro follows.

Table 1

Macros

Name	Function	Use[a]
#ATTACH	Create a Zeus Task	3
#CALL	Call a subroutine	1
#CLOSEP	Close program and return to caller	1
#CONVRT	Convert data from one format to another	1
#COUNT	Accumulate statistics	2
#DELETE	Delete Transient Module	3
#DEQ	Release resource from exclusive control	2
#ENQ	Acquire resource for exclusive control	2
#FINDZVT	Locate Vector Table	2
#FREEMEM	Release main memory	2
#GENSECT	Generate Label Map	3
#GETMEM	Acquire main memory	2
#GOTO	Execute conditional branch	1
#LDREG	Load register	1
#LOAD	Load transient module	3
#LPSW	Load program status word	2
#MOVE	Move data with/without translate	1
#OPENP	Open program	1
#QFB	Generate queue foundation block	1
#QUE	Process Queue	1
#REGS	Register symbol generation	1
#SET	Set bit(s) on/off/invert	1
#STIMER	Set interval timer	2
#TTIMER	Test interval timer	2
#WAIT	Wait for event completion	3
#ZEUSSVC	Execute Zeus SVC	2

[a] 1 = General Use
2 = Restricted Use
3 = Zeus only

Macro Coding Key

Symbol	Meaning
Upper-Case Characters	Code exactly as shown.
lower-case characters	Replace as instructed.
[ ]	Optional argument.
{ }	Select one option from stack. An underlined option will be defaulted if the argument is omitted. If no option is underlined, the argument cannot be omitted unless the entire argument is enclosed within brackets.
address	Any address valid in an 'LA' instruction; a symbol valid in an 'A' or 'V' type address constant; or (r), $2 \leq r \leq 12$, given the address is available in the register specified.
address-sym	Any symbol valid within an 'A' or 'V' type address constant.
(r)	General purpose register 0-15 enclosed within parentheses.
register	General purpose register 2-12, parentheses optional.
char	One or more alphanumeric characters.

Figure 3

#ATTACH -- Create a Zeus Task

The #ATTACH macro provides a facility whereby one task can create another task.

[label]	#ATTACH	{EPADDR=address / name—char / (r)} [,PARAM=address]

label

specifies the address of the first executable instruction generated by the macro.

EPADDR=address

the address of a word in main storage where the address of the routine being attached is stored.

name

code a character string from 1 to 8 characters which identifies the routine to be attached. The name must satisfy the requirements of a V-type address constant.

(r)

replace r with a register that has previously been loaded with the address of the routine being attached.

PARAM=address

The address that will be passed to the attached task via register 1. If omitted, register 1 will be passed unaltered.

Registers Altered: 0, 14 and 15
External References: ATCHOO
Written by: J. Garneau

#CALL -- Call subroutine

The #CALL macro provides a mechanism for transferring control to another module.

[label]	#CALL	{EPADDR=address name-char [, RESOLVE=LOCAL] (r)}

label

specifies the address of the first executable instruction generated by the macro.

EPADDR=address

The address of a word in main storage where the address of the routine being called is stored.

name-char

code a character string from 1 to 8 characters which identifies the routine to be called. The name must satisfy the requirements of a V-type address constant.

RESOLVE=LOCAL

code this parameter to override the effect of the 'RESOLVE=#CALL' parameter of the #GENSECT macro. The override is in effect during the expansion of this macro only. If the #GENSECT macro option has not been selected during this assembly, 'RESOLVE=LOCAL' is the default. For additional details see the #GENSECT macro description in this document.

(r)

replace r with a register that has previously been loaded with the address of the routine being called.

Registers Altered: 14, 15
Written by: J. Garneau

#CLOSEP -- Close Program/Subroutine and Return to Caller

The #CLOSEP macro release save areas,[1] restores all (0-15) registers, and returns to the caller.

[label]	#CLOSEP	[RC={value-char / (r)}] [,RET={yes / no}] [,SAREA=address]

label

specifies the label of the first executable instruction generated by the macro.

RC={value / (r)}

a value (0 to 4095) or a register that contains a value ($\pm 2^{32-1}$) that will be loaded into register 15 before return is made. If omitted, return is made with register 15 unaltered.

RET={YES / NO}

If YES is coded or the parameter omitted, an unconditional branch to the address contained in register 14 will be made after SAVE areas have been freed, and registers 0-15 are restored. If NO is coded, no branch will be made.

SAREA=address

The address of the save area from which registers 0-15 should be loaded. Registers are loaded, 14 through 12, starting with the third

[1]Save areas dynamically acquired by the #OPENP macro will be freed automatically by #CLOSEP. The size of the save area freed in such a case is determined by the #OPENP, which physically precedes this #CLOSEP in the assembly not the #OPENP, which may possibly precede it "logically."

word of the save area. If this parameter is omitted, the address of the save area from which the registers are to be loaded is assumed to be stored in a word that is addressed by adding 4 to the current contents of register 13 (note: this is standard OS convention and is compatible with the #OPENP macro).

Registers Altered: See 'RC=' option
Written by: J. Garneau

#CONVRT -- Convert Data From One Format to Another

The #CONVRT macro instruction is used to reduce numeric character strings of unknown length to binary values or to expand binary values to "edited" numeric character strings.

Convert to binary requires a character string address and a scan limit value. The character string is scanned left to right (within the limit imposed) for a nonblank character. The scan continues from that point until either a nonnumeric character is encountered or until the scan limit value equals zero. The character string so delimited is then used as input to the conversion routine.

Upon return, register 15 will equal zero, register 0 will contain the converted number, and register 1 will contain the address of the character that terminated the scan. If during the scan or conversion it is determined that the string is less than 1 character or greater than 15 characters in length, or the resultant value is larger than 2,147,483,647, return will be made with register 15 equal to 4, 8, or 12 respectively.

Convert to character converts a binary value to an EBCDIC character string equivalent. Upon return, general register 1 points to a '+' or '-' followed by a string of digits equivalent to the (input) binary value. General register 0 contains a count of the number of digits in the string. Nonsignificant zeroes are edited out except when the input value is 0. In such a case, register 1 will point to '+0' and register 0 will equal 1.

[label]	#CONVRT	{ [data-address],TO-BIN [,limit] [,(r)] } { [data-register],TO-CHA }

address

is the address in main storage of the data to be converted. If this argument is omitted, general purpose register 1 is assumed to contain the address.

register

specifies the general purpose register that contains the number to be converted. If this argument is omitted, register 0 is assumed to contain the number.

TO-BIN

convert from character to binary

TO-CHA

convert from binary to character

limit

specifies a number equal to or less than 4095 (or a register, r, which contains a number) to limit the scan. If omitted, a limit of 4095 is assumed. If the limit is contained in a register, the register, upon return, will be reduced by an amount equal to the number of bytes scanned.

Registers Altered: 0, 1, 14, 15
External References: CNVT000 or CNVT010
Written by: J. Garneau

#COUNT -- Accumulate Statistics

The #COUNT routine is used to add, subtract, or load data into a series of full word counters. Upon return from a #COUNT, register 15 will contain a number equivalent to the current value of the specified counter.

[label]	#COUNT	counter-char [,{LOAD / ADD}={value-char / (r)}]

label

specifies the address of the first executable instruction generated by the macro.

counter-char

code a number from 1 to 255 to specify the counter to be affected. If a counter does not exist that can be associated with this number, a special counter (counter 0) will be assumed.

{LOAD / ADD}={value-char / (r)}

code a number from 1 to 4095 or replace r with a register that has previously been loaded with a number ($\pm 2^{-31}$). The number is loaded into the specified counter or is added to the current value of the specified counter depending on which keyword is coded. If both these keywords are omitted, the specified counter will be incremented by one (1).

Registers Altered: 1, 14, 15
External References: ICOUNT or ICOUNT1 or ICOUNT2 or CNTXCTL
Written by: J. Garneau

#DELETE -- Delete Transient Module

The #DELETE macro provides a facility whereby a task can delete a module that had previously been loaded with a #LOAD macro.

[label]	#DELETE	EP={address / 'char'}

label

The address of the first executable instruction generated by the macro.

EP={address / 'char'}

The address of an 8-byte character string (padded with blanks if necessary) or a 1- to 8-byte character string enclosed in single quotes that identifies the module to be deleted.

Registers Altered: 0, 1, 14, 15
External References: LDR0000
Written by: J. Garneau

#DEQ -- Release Resource From Exclusion Control

The #DEQ macro enables a task to release a resource that it has exclusive control of.

[label]	#DEQ	[RQE=address]

label

The address of the first executable instruction generated by the macro.

RQE=address

The address of the RQE that was specified in the #ENG macro instruction issued to gain control of the specified resource. If omitted, register 1 is assumed to contain the address of the RQE.

Registers Altered: 1, 14, 15
External References: ENQCTRL
Written by: J. Garneau

#ENQ -- Request Exclusive Control of Resource

The #ENQ macro provides a facility whereby a task may request exclusive control of a resource.

[label]	#ENQ	[NAME={address / 'char'}][,ECB=address][RQE=address][,WAIT=address]

label

The address of the first executable instruction generated by the macro.

NAME={address / 'char'}

The address of a 4-byte character string (padded with blanks if necessary) or a 1- to 4-byte character string enclosed in single quotes. If omitted, the character string is assumed to have been stored in bytes 4-8 of the RQE.

ECB=address

The address of the ECB to be posted complete when exclusive control of the resource is acquired. If omitted, the ECB address is assumed to have been stored in bytes 8-12 of the RQE.

RQE=address

The address of the Resource Queue Element used to process the #ENQ request. The RQE must be at least three words long. If omitted, Register 1 is assumed to contain the address of the RQE.

WAIT=address

The address of the ECBLIST that contains the address of the ECB that will be posted when the task gains exclusive control of the

requested resource. If omitted, no automatic wait is generated within the #ENQ macro. It is possible for the task to #WAIT on this list at any latter point so long as the integrity of the RQE is honored.

Registers Altered: 0, 1, 14, 15
External References: ENQCTRL
Written by: J. Garneau

#FINDZVT -- Locate the Zeus Vector Table

The #FINDZVT macro returns to the caller with the address of the ZVT (Zeus Vector Table) in register 15.

[label]	#FINDZVT	

label

specifies the address of first executable instruction generated by the macro.

Registers Altered: 0, 1, 14, 15
External References: ZEUSSVC
Written by: J. Garneau

#FREEMEM -- Free Main Memory

The #FREEMEM macro provides a means for freeing main memory previously acquired via a #GETMEM macro.

[label]	#FREEMEM	[address]

label

The address of the first executable instruction generated by the macro.

address

The address of the memory segment to be freed. The memory segment must have previously been allocated with a #GETMEM macro. If omitted, Register 1 is assumed to contain the memory segment address.

Registers Altered: 1, 14, 15
External References: FREEMEM
Written by: J. Garneau

#GENSECT -- Generate DSECT/CSECT

The GENSECT macro provides a means of generating DSECTS (or CSECTS) which map Zeus blocks and tables.

	#GENSECT	type [,BASE=register] [,RESOLVE=#CALL]

type

code LCB to generate a Line Control Block DSECT, TCB to generate a Task Control Block DSECT, ZVT to generate a Zeus Vector Table DSECT, or DSKREC to generate a Text File Header DSECT. In each case the name of the DSECT will be its type (e.g., LCB DSECT).

BASE=register

code any register notation valid in a 'USING' instruction. If omitted, no base register will be assigned to the DSECT generated.

RESOLVE=#CALL

code in a ZVT generation request to cause the addresses of routines referenced by a #CALL macro to be obtained from the ZVT. If the parameter is coded, a valid base register must be assigned and maintained throughout the module, since all Zeus macros that generate requests to subroutines do so via (internal) #CALL macros.

Note:

Only one DSECT of each type may be generated during any assembly. If more than one of a type is coded, its expansion will be bypassed and a message issued. Base register reassignment will be executed, however, if the 'BASE=...' parameter is coded.

Written by: J. Garneau

#GETMEM -- Allocate Main Memory

The #GETMEM macro is used to dynamically allocate main memory. Memory is allocated on a "first come, first served" basis for all contending tasks in the system. If memory is not immediately available, the requesting task can wait for it to become available. The address of the memory segment allocated is returned in register 1.

[label]	#GETMEM	{address / (r) / char}	[CQE=address] [,WAIT=address] [,ECB=address]

label

The address of the first executable instruction generated by the macro.

{address / (r) / char}

The amount of main memory requested, in bytes. Code one of the following:

(1) The address of a word that contains the amount required.
(2) A register enclosed in parentheses that contains the amount required.
(3) A numeric character string equaling the amount required.

CQE=address

The address of the Core Queue Element associated with the request. If omitted, and memory is not immediately available, the request will not be queued.

WAIT=address

The address of the ECBLIST that contains the address of the ECB that will be posted when memory becomes available. If omitted,

no automatic wait is generated.[1] It is possible for the task to wait at a later point so long as the integrity of the CQE is honored.

ECB=address

The address of the ECB to be posted when memory becomes available. If omitted, the address of the ECB is assumed to have been stored in bytes 8-12 of the CQE.

Registers Altered: 0, 1, 14, 15
External References: GETMEM
Written by: Jean Garneau

[1]Upon return from a #GETMEM without the 'WAIT' option, register 15 will equal zero if the memory requested was available. Register 1 will contain the address of the requested segment. If memory is not available, register 15 ≠ zero. If a CQE was available and memory was not available, the request was queued. When the memory becomes available, the ECB addressed by the CQE will be posted complete and the address of the memory segment will be stored in the rightmost three bytes of the ECB.

#GOTO -- Executive Conditional Branch

The #GOTO macro instruction is used to alter the instruction execution flow based on the evaluation of an expression.

[label]	#GOTO	address[,(expression)][,RET=register]

label

The address of the first executable instruction generated by the macro.

address

The address of the instruction to be branched to if the expression is evaluated to be true.

expression

The expression to be evaluated to determine if the branch will take place. The expression has one basic form as follows:

(operand A, operator, operand B)

Operands appear exactly as they would in standard assembler comparison instructions except for the following:

(1) Decimal operation (i.e. CP) is not supported.
(2) If the content of a register is to be evaluated the register must be enclosed in parentheses.
(3) Literals are not indicated by a preceding equal (=) sign except when the literal is an 'A' type address constant (e.g., =A(X)).
(4) A register may be evaluated against zero by coding a single 0 as the righthand operand.
(5) Bits within a byte may be evaluated by coding one or more bit numbers separated by commas and enclosed within parentheses. Bits are numbered within a byte left to right 1 through 8.

operator

Code one of the following

EQ	equal
GE	greater than or equal
LE	less than or equal
LT	less than
GT	greater than
Bit-on	Equal if specified bit(s) on
Bit-off	Equal if specified bit(s) off
Bit-any	Equal if any bit(s) specified on

Examples

Expression	Is true if
(A, GT, B)	A greater than B.
(A(6), GE, B)	First 6 bytes of A greater than or equal to first 6 bytes of B.
(B(3), LT, C 'CAT')	First 3 bytes of B less than string 'CAT'
(5(10,7), EQ, 10C 'A')	Ten bytes addressed +5 from register 7 equal 'A's.
((5), EQ (4))	Register 5 equals register 4.
((5), LE, H '500') or ((5), LE, F '500')	Register 5 is less than or equal to 500.
((R4), eq, 0)	Register 4 is equal to zero.
(A, Bit-on, 2)	Bit 2 of byte addressed by A is on
(A+6, Bit-off, (1, 4, 5))	Bits 1, 4, and 5 of byte addressed by A+6 are off.
(0(R2), Bit-any, (1,2,3,4))	Any one or all of bits 1 through 4 as addressed by register 2 are on.

If the expression is omitted, the branch is made unconditionally.

RET=register

The register to be used as a return register in a 'BAL' instruction. If omitted, register 14 is assumed. A 'BAL' instruction is generated only if the expression is omitted and two or more addresses are coded. Addresses must be separated by commas with the entire list enclosed in parentheses. A single 'BAL' can be generated by coding the second address as an asterisk.

Examples:

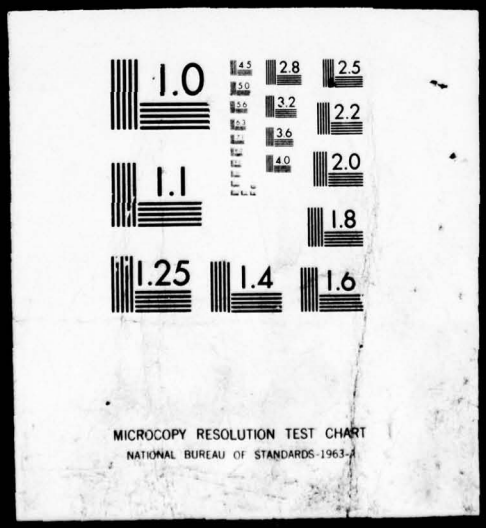
```
#GOTO A                 unconditional branch to A
#GOTO (A,*)             'BAL' to A, R14 = return register
#GOTO (A,*),RET=3       'BAL' to A, R3 = return register
#GOTO (A,B,C)           'BAL' to A, then B, then C,
                        R14 = return register
```

Registers Altered: See 'RET=' parameter.
Written by: Jean Garneau

AD-A055 023 HUMAN RESOURCES RESEARCH ORGANIZATION ALEXANDRIA VA F/G 5/9
PROJECT IMPACT SOFTWARE DOCUMENTATION: VI. VOLUME 2. ZEUS PROGR--ETC(U)
AUG 72 J GARNEAU, W UNDERSHILL, D SHUFORD DAHC19-73-C-0004
UNCLASSIFIED HUMRRO-RP-D1-72-5-VOL-2 NL

2 OF 2
AD
A055023

END
DATE
FILMED
7-78
DDC



#LDREG -- Load Register

The #LDREG macro provides a facility whereby other macros may decode parameter values and load appropriate registers with a minimum of effort. The #LDREG macro is intended to be used only within other macro descriptions.

	#LDREG	(register[,option]),{address / (r)}[,'message']

register

the register from 0-15 which is to be loaded.

option

code one of the following or omit:

LA Load address
L Load full word
LH Load half word
LN Load address and complement

For effect of omission of 'option,' see following parameter definition.

{address / (r)}

The value that is to be loaded into the register. If address is coded and option is omitted, 'LA' is assumed; if (r) is coded and option is omitted, 'LF' is assumed. If (r) is coded and 'LA' is the option, the effect in instruction executed is 'LA register,0(r).'

'message'

Code a message to be output (via a MNOTE statement) if the source data parameter is not available. If omitted, no message is generated.

Registers Altered: Any
Written by: J. Garneau

#LOAD -- Load Transient Module

The #LOAD macro provides a facility whereby a task may request that a nonresident module be made available for use. If a copy of the module is currently available in main memory it will be used, otherwise the module will be loaded from an external storage device. The address of the module is returned in register 15. If register 15 equals 0, the module was not found or could not be loaded because of I/O problems.

[label]	#LOAD	EP={address / 'char'} [,LRB=address [,ECB=address][,WAIT=address]]

label

The address of the first executable instruction generated by the macro.

EP={address / 'char'}

The address of an 8-byte character string (padded with blanks if necessary) or a 1- to 8-byte character string enclosed in single quotes that identifies the module to be loaded.

LRB=address

The address of the Load Request Block used to process the #LOAD request. The LRB must be (at least) three words long.

WAIT=address

The address of the ECBLIST that contains the address of the ECB that will be posted complete when the loaded module is made available to the task.

If the WAIT keyword is omitted, no automatic wait is generated within the #LOAD macro. However, the task may issue a standard #WAIT at any later point so long as the integrity of the LRB is honored. In this case, upon return from the #LOAD, register 15 is negative if the request was successful (i.e., LRB was added to Load Request Queue), and zero if the request failed (i.e., module could not be located in library).

ECB=address

The address of the ECB to be posted complete when the module becomes available to this task. If omitted, and the WAIT keyword is coded, the ECB address is assumed to have been stored in bytes 8 through 12 of the LRB.

Registers Altered: 0, 1, 14, 15
External References: LDR0000
Written by: W. Underhill

#LPSW -- Load Program Status Word

The #LPSW macro will change the state of a program from supervisor to problem or from problem to supervisor, setting protect keys as required.

[label]	#LPSW	[state,KEY={address / (r)}]

label

specifies the label of the first executable instruction generated by the macro.

state

code SUPV to load a supervisor state psw or PROB to load a problem state psw.

KEY={address / (r)}

The address of a byte in main storage, or a register, in which the protect key from the TCB can be stored after a change to Supervisor state or fetched from during the change to problem state.

Registers Altered: 0, 1, 14, 15, also see 'KEY=' parameter
External References: ZEUSSVC
Written by: J. Garneau

#MOVE -- Move Data

The #MOVE macro allows data segments from 1 to 32,767 bytes to be moved, supports execution time length generation, and provides automatic EBCDIC to Hexadecimal conversion. Register 1, upon return from a #MOVE, will point to the last input character moved +1. CPU cycles and memory are optimized for each macro expansion.

[label]	#MOVE	[TO=address] [,FROM=address] [TRNSLAT={EBC/HEX}]

TO=address

The address in main storage where the data will be moved. This address can take any format acceptable within the first operand of a 'MVC' instruction with the following additions:

(1) any length from 1 to 32,767 is valid
(2) the length may be loaded into a register and that register specified in the TO address length field, e.g. TO= 0((4),5)

If this parameter is omitted, register 1 is assumed to contain the address and register 14 the length.

FROM=address

The address in main storage from where the data are to be moved. This address can take any format acceptable within the second operand of a 'MVC' instruction. If omitted, register 0 is assumed to contain the from address.

TRNSLAT={EBC/HEX}

If EBC is specified, or if the parameter is omitted, data are moved unmodified. If HEX is specified, each input data byte is converted to 2 output "HEX" bytes, that is, the actual number of input data bytes moved is one-half the number of characters specified as the move length.

Registers Altered: 0, 1, 14, 15
External References: MOVE000 or MOVE010
Written by: Jean Garneau

#OPENP -- Open Program/Subroutine and Initialize for Execution.

The #OPENP macro saves callers registers, assigns base registers, acquires and initializes program save areas.

[label]	#OPENP	[register] [,TYPE={REUS | RENT}] [,EXTEND=val-char] [,TCB=register]

label

Specifies the name to be assigned to the CSECT statement generated by the #OPENP macro. If omitted, the label 'MAIN' is assumed. The inclusion of any CSECT statement(s) prior to the #OPENP overrides the generation of the CSECT within the #OPENP.

register

Specifies the one or more registers (if more than one they must be separated by commas and the entire list enclosed in parentheses) that will become the program base registers. Registers are loaded and assigned, one by one, to contiguous 4095-byte segments of the program. If the parameter is omitted and the #OPENP is the first in the program, register 12 is assumed. If the parameter is omitted and the #OPENP is not the first in the program, no attempt to reassign a base register will be made; however all remaining parameters will be assembled normally.

TYPE={REUS | RENT}

Specifies dynamic or static save area generation. If 'RENT' is coded, the save area will be acquired at execution time out of available storage within the partition or region assigned to the program. If 'REUS' is coded or the parameter is omitted, an in-line save area will be generated. In either case, the address of the save area is loaded into register 13 and the save area is set to zeros (x'00').

EXTEND=val-char

Specifies the number of words that the save area will be extended. The address of the first byte of the first word of the extended

area is 72 (decimal) plus the contents of register 13. If this parameter is omitted, the standard OS 18-word save area is generated.

TCB=register

Code a register that contains the address of a Zeus TCB. If this parameter is selected and TYPE=RENT is coded, the save area that is acquired will be obtained via a #GETMEM Zeus macro (if memory is not available it will be waited for); if omitted, an OS GETMAIN will be executed.

External References: GETSAVE if TCB=r and TYPE=RENT are coded
Written by: J. Garneau

#QFB -- Generate Queue Foundation Block

The #QFB macro provides a facility for describing and generating a queue foundation block.

[label]	#QFB	[QTYPE=(option,[...option])][,EXTEND=val-char]

label

The label of the first byte of the first word of the QFB generated within the macro.

QTYPE=option

Select one from each stack or as required:

{FIFO}
{LIFO}

{NOTORDERED }
{ORDERED[-ASCENDING]}
{ORDERED-DESCENDING }

EXTEND=val-char

A numeric value from 1-4095 that defines the number of words the QFB should be extended. The extended area begins at offset 8 bytes from the first byte of the QFB. If omitted, the QFB is not extended.

Written by: J. Garneau

#QUE -- Process Queue

The #QUE macro provides a facility whereby a routine may manipulate a queue. The queue must have been originally defined with a #QFB macro.

```
[label]  #QUE  { ADD,ELEMENT=(address[,option])
               { DELETE  ,{ELEMENT={0|address}[,option]}
               { LOCATE  ,{ID={'char'|address}[,option])}  [,QFB={address|(r)}]
```

label

The label of the first executable instruction generated by the macro.

ADD

add element to queue.

LOCATE

locate element and return its address in register 1. If element cannot be located, register 1 will equal 0 when control is returned.

DELETE

locate and delete specified element. If the element is deleted, register 1 will contain the address of the element; otherwise register 1 will equal 0.

ELEMENT={0|address}

The address of the element to be ADDed, DELETEd, or LOCATEd. If 0 is coded, either the absolute top or absolute bottom element on the queue will be affected (see option parameter).

ID={'char' / address}

The address of a 4-byte field (padded with blanks if necessary) or a 1- to 4-byte character string enclosed in single quotes which identifies the element to be deleted or located. The identifier is maintained in bytes 4-7 of the element and consists of any bit combination from X'00000000' to X'FFFFFFFF.'

option

As subparameter of ELEMENT parameter only:

Code TOP or BOTTOM to ADD/DELETE/LOCATE either the absolute top (first) or bottom (last) element on the queue (see "ELEMENT" parameter). If omitted, the request is processed as specified by the QFB options field.

As subparameter of ID parameter only:

Code TOP or BOTTOM to ADD/DELETE/LOCATE either the top (first) or bottom (last) element on the queue with the specified ID. Code LT (less than), GT (greater than), or GE (greater than or equal) to ADD/DELETE/LOCATE an element relative to some other element currently on the queue. If omitted, the request is processed as specified by the QFB options field.

QFB={address / (r)}

The address of the Queue Foundation Block associated with this queue. If omitted, register 0 is assumed to contain the QFB address.

Registers Altered: 0, 1, 14, 15
External References: QUEMGR
Written by: Jean Garneau

#REGS -- Register Symbol Generation

The #REGS macro provides automatic generation of register symbols.

	#REGS	[char][,PREFIX=char]

char

Code a number representing a register to be equated, or two numbers separated by commas and enclosed in parentheses representing a range of registers to be equated. If omitted, registers 0-15 will be equated.

PREFIX=char

Code one to six characters to specify the prefix to be assigned to each register selected. If omitted, the prefix "R" will be assumed. Checking to protect against generation of duplicate symbols will be performed only if the prefix "R" is either chosen or defaulted.

Written by: Jean Garneau

#SET -- Set Bit Status

The #SET macro instruction is used to alter the status of one or more bits within a byte.

[label]	#SET	byte-address,BIT-{ON / OFF / INV},{(bit-char) / ALL}

byte-address

is the address in main storage of the byte that is to be altered.

BIT-

is the type of operation to be performed where:

ON - turns specified bit(s) on
OFF - turns specified bit(s) off
INV - Inverts status of specified bit(s)

bit-char

is a single bit number, or several bit numbers separated by commas and enclosed in parentheses. Bits are numbered within the byte, left to right, 1 to 8.

ALL

can be specified if all eight bits are to be affected. This parameter can be used in lieu of a specific request for each of the eight bits, for example, ALL instead of (1, 2, 3, 4, 5, 6, 7, 8).

Written by: J. Garneau

#STIMER -- Set Interval Timer

The #STIMER macro allows a task to set one or more interval timers. The task can wait for the specified interval to elapse or can continue processing. When the interval expires, the task can have either an ECB posted or can request that an exit routine be scheduled.

[label]	#STIMER	SEC=char[,TQE=address] [{,WAIT=address [,ECB=address] / ,EXIT=address}]

label

The address of the first executable instruction generated by the macro.

SEC=char

Code a numeric character string with 0, 1, or 2 decimal places to specify the clock interval in seconds, tenths, and hundreds.

TQE=address

The address in main memory of the Timer Queue Element associated with this request. If omitted, register 1 is assumed to contain the TQE address.

WAIT=address

The address in main memory of an ECBLIST. The ECBLIST must contain the address of the ECB that will be posted when the interval expires. If omitted no automatic wait is generated by the macro. It is possible, however, for the task to wait at any later point so long as the integrity of the TQE is honored.

ECB=address

The address of the ECB to be posted complete when the interval expires. If omitted, the ECB address is assumed to have been stored in bytes 8 through 12 of the TQE with the sign bit off.

EXIT=address

The address of a routine to be executed when the interval expires. Upon entry to the Exit Routine, register 1 will point to the originating TQE. Words 4 to n of this TQE can be used to pass information to the exit routine. The exit routine is executed asynchronously with the task which requested the original interval. Any synchronization required between the requesting task and the exit task must be handled by the tasks themselves.

Registers Altered: 0, 1, 14, 15
External References: ITIMER
Written by: J. Garneau

#TTIMER -- Test Zeus Interval Timer

The #TTIMER macro is used to test, cancel or force completion of an interval timer previously set by a #STIMER macro.

[label]	#TTIMER	[type][,TQE=address]

label

specifies the address of the first executable instruction generated by the macro.

type

code one of the following:

REMAINING	Return time remaining this interval (in hundreds of seconds) in the second word of the specified TQE. This is default type if parameter omitted.
COMPLETE	Force ECB referenced by specified TQE to be posted complete and cause TQE to be removed from timer queue.
CANCEL	Cause TQE to be removed from timer queue (ECB not posted).

TQE=address

The address in main memory of the TQE (Timer Queue Element) associated with the request. If omitted, register 1 is assumed to contain the TQE address.

Registers Altered: 0, 1, 14, 15
External References: ITIMER
Written by: J. Garneau

#WAIT -- Wait for Event Completion

The #WAIT macro instruction allows a task to wait for the completion of one or more events. The event may be resource availability, I/O, timer interval completion, memory availability, task synchronization, and so forth. Upon return from a #WAIT register 0 contains the address of the ECB posted complete and register 15 contains its offset in the LIST (i.e., ECB=1, R15=0; ECB=3, R15=8).

[label]	#WAIT	[{ecblist-address / ECBLIST=ecblist-address}] [,EPADDR=entry-address]

label

specifies the address of the first executable instruction generated by the macro.

{ecblist-address / ECBLIST=ecblist-address}

specifies the address in main memory of the ecblist that contains the addresses of one or more ECB's to be waited on. Each ECB address is 4 bytes in length. The last ECB address must have bit 0 (i.e. X'80') on. If omitted, register 1 is assumed to contain the ecblist address.

EPADDR=entry-address

The address in main memory of a word which contains the Dispatcher address. If the Dispatcher address has previously been loaded into a register, this parameter may be coded (r) where r is the register so loaded. If the parameter is omitted, a V-type address constant for the entry point of the Dispatcher.

Registers Altered: 0, 1, 14, 15
External References: DSPTCHR
Written by: J. Garneau

#ZEUSSVC -- Execute Zeus SVC

The #ZEUSSVC macro executes an SVC instruction which is stored in a "remote" CSECT. (The purpose of this is to allow the Zeus SVC to be changed with a single assembly and a relink-edit of Zeus.) Upon return, register 15 will contain the address of the Zeus incore address table and register 0 the address of the partition or region TCB.

[label]	#ZEUSSVC	

label

specifies the address of the first executable instruction generated by the macro.

Registers Altered: 0, 1, 5
External References: ZEUSSVC
Written by: J. Garneau

Chapter 3

SYSTEM DATA AREAS

The operation of Zeus requires that certain control blocks and queue elements be created and maintained. Table 2 contains a list of these areas, a brief description of each, and their primary use.

Table 2

Control Blocks

Name	Used For	Number Required
LCB: Line Control Block	Control I/O activity on a TP Line or local channel	1 per device type per line
TQE: Timer Queue Element	Support a Zeus interval timer	unlimited
CQE: Core Queue Element	Support conditioned memory request	unlimited
TCB: Terminal Control Block	Control I/O activity on a terminal device	1 per device
ZVT: Zeus Vector Table	Cross reference address table for main Zeus components	1 per Zeus
QFB: Queue Foundation Block	Anchor for each queue	unlimited
LRB: Load Request Block	Support transient module load function	unlimited

A description of each Control Block and Queue Element is presented in this Chapter, organized alphabetically within two sections—Section 1, Control Blocks and Tables and Section 2, Queue Elements.

SECTION 1: CONTROL BLOCKS AND TABLES

Descriptions of the Zeus Line Control Block (LCB), Queue Foundation Block (QFB), Terminal Control Block (TCB), and Vector Table (ZVT) are provided in this section. For each, an overview is presented first, followed by a detailed description.

LINE CONTROL BLOCK

The Line Control Block (LCB) is used by the Line Task to control I/O on a remote or local communication channel. More than one LCB may be assigned to a given channel. However, I/O on LCBs assigned to separate channels is asynchronous while I/O on LCBs assigned to the same channel is synchronous.

Synchronization of multiple LCBs on a single channel is accomplished by assigning the channel exclusively to each LCB in turn. The #ENQ/#DEQ Zeus macros are used for this purpose.

Line Control Block—Overview

Offset	Word		
0(0)	DECSDECB		
4(4)	DECTYPE	last op. DECLNGTH	
8(8)	DECONLTT	DECDCBAD	
12(C)	DECAREA		
16(10)	DECSENS0	DECSENS1	DECCOUNT
20(14)	DECCMCOD	DECENTRY	
24(18)	DECFLAGS	DECRLN	DECRESPN
28(1C)	DECTPCOD	DECERRST	DECCSWST
32(20)	DECADRPT		
36(24)	DECPOLPT		
40(28)	LDECBA		
44(2C)	LTCBC	LTCBA	
48(30)	LPOLS	LLISTA	
52(34)	LPOLLC	LADDRC	
56(38)	LCBFLGS	LCBNEXT	
60(36)			
68(44)	LTQE	LCQE	
72(48)	LLTH		

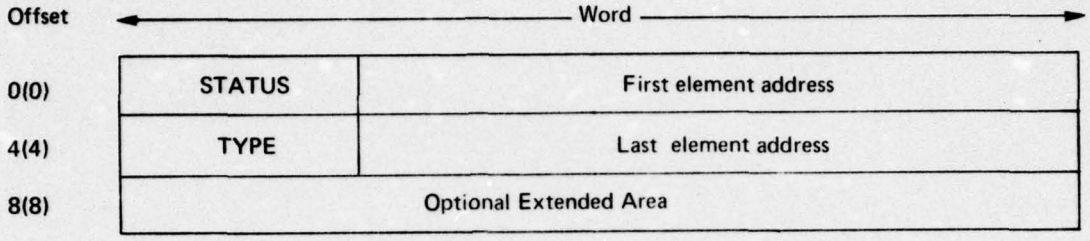
Line Control Block—Detailed Description

Offset	Bytes and Alignment	Field Name	Field Description, Contents, Meaning
0(0)	0	DECB	
0(0)	4	DECSDECB	Event control block
4(4)	1	DECTYPE	Type field
5(5)	. 1	. . . . .	Last operation
6(6)	. . 2	DECLNGTH	Length
8(8)	1	DECONLTT	Reserved for on-line test
8(8)	4	DECDCBAD	DCB address
12(C)	4	DECAREA	Area address
16(10)	1	DECSENS0	First sense byte
17(11)	. 1	DECSENS1	Second sense byte
18(12)	. . 2	DECCOUNT	Residual count
20(14)	1	DECCMCOD	Command code
20(14)	4	DECENTRY	Terminal list address
24(18)	1	DECFLAGS	Status flags
25(19)	. 1	DECRLN	Relative line number
26(1A)	. . 2	DECRESPN	Response field
28(1C)	1	DECTPCOD	TP op code
29(1D)	. 1	DECERRST	Error status
30(1E)	. . 2	DECCSWST	CSW status
32(20)	4	DECADRPT	Address of current addressing entry
36(24)	4	DECPOLPT	Address of current polling entry
40(28)	1	LDECBA	Reserved for DSPTCHR
41(29)	. 3	. . . . .	DECB address
44(2C)	1	LTCBC	Count of this line's TCBs
44(2C)	4	LTCBA	Address of first TCB on this line's queue
48(30)	1	LPOLS	Polling list control byte save area
48(30)	4	LLISTA	Line polling list address
52(34)	2	LPOLLC	Number of active entries in polling list
54(36)	. . 2	LADDRC	Number of writes pending
56(38)	1	LCBFLGS	
		1	Busy
		. 1	Poll
		. . 1	Interval
		. . . 1	Write
		. . . . 1 . . .	Memory wait
56(38)	4	LCB NEXT	Address of next LCB on queue
60(3C)	12	LTQE	Line timer queue element
60(3C)	12	LCQE	Line core queue element
72(48)	4	LLTH	LCB length

QUEUE FOUNDATION BLOCK

The head pointer to each queue is located in a Queue Foundation Block. The Queue Foundation Block is used as an anchor for all Zeus Queues. For example, the timer queue, the memory queue, and the load request queue are all anchored with their own QFBs.

Queue Foundation Block—Overview



Queue Foundation Block—Detailed Description

Offset	Bytes and Alignment	Field Name	Field Description, Contents, Meaning
0(0)	1	1	Status bytes - null queue
1(1)	. 3		Address of first element in queue
4(4)	1	TYPE	
		. 1	Ascending queue
		. . 1	Descending queue
		. . . 1	Last in/first out - LIFO
		. . . . 1 . . .	First in/first out - FIFO
		. . . . . 1 . .	Count
		X. XX	Reserved
5(5)	. 3		Address of last element in queue
8(8)	Any	Data	Optional extended data area

ZEUS TERMINAL CONTROL BLOCK

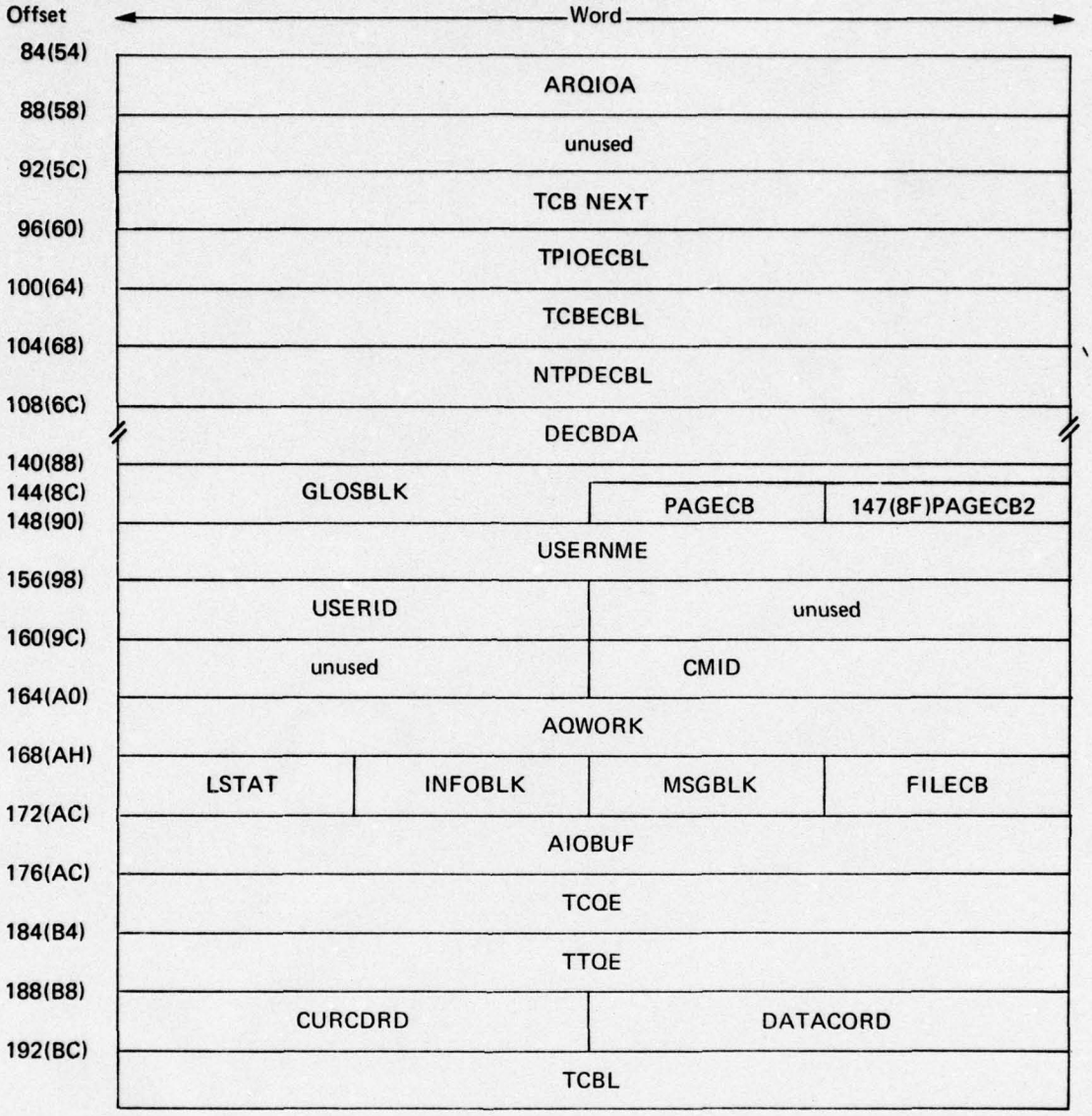
The Terminal Control Block (TCB) is used by the Terminal Task to control I/O on a (terminal) device.

Zeus Terminal Control Block—Overview

Offset	Word	
0(0)	TCBIDENT	TCBCOUNT
4(4)	TCBECB	
8(8)	TPIOECB	
12(C)	BUFSIZE	
16(10)	LINEREM	
20(14)	POSREM	
24(18)	RBLOKADR	
28(1C)	RELBLOCK	
32(20)	TCBRJE	
36(24)	SBUF1L	
40(28)	SBUF2L	
44(2C)	REQDCB	
48(30)	ACHAINSA	
32(34)	SBUF1A	
56(38)	SBUF2A	
60(3C)	PAGELINE	
64(40)	PAGEND	
68(44)	unused	
72(48)	ALCB	
76(4C)	ATAENTRY	
80(50)	ARQDECB	

(Continued)

Zeus Terminal Control Block—Overview *(Continued)*



Zeus Terminal Control Block–Detailed Description

Offset	Bytes and Alignment	Field Name	Field Description, Contents, Meaning
0(0)	3	TCBIDENT X'E3C3C2'	Characters 'TCB' to identify control block
3(3)	. . . 1	TCBCOUNT	TCB's number
4(4)	4	TCBECB	ECB for CPU synchronization
8(8)	4	TPIOECB	ECB for terminal I/O
12(C)	4	BUFSIZE F'1048'	Buffer size for terminal READ operation
16(10)	4	LINEREM	Unused lines remaining in terminal output buffer
20(14)	4	POSREM	Unused characters remaining in terminal output buffer
24(18)	4	RBLOKADR	Address in page where miscellaneous system messages are to be stored
28(1C)	4	RELBLOCK	Relative block number of text to be retrieved from or stored on direct access storage device
32(20)	4	TCBRJE	Used by RJE subsystem
36(24)	4	SBUF1L	Length of buffer one
40(28)	4	SUBF2L	Length of buffer two
44(2C)	4	REQDCB	Address of DCB associated with current request for text storage or retrieval this TCB
48(30)	4	ACHAINSA	Address of initial save area in chain currently attached to this TCB
52(34)	4	SBUF1A	Address of buffer one
56(38)	4	SBUF2A	Address of buffer two
60(3C)	4	PAGELINE	Next available position in terminal output buffer
64(40)	4	PAGEND	Last position in terminal output buffer
68(44)	4		unused
72(48)	4	ALCB	Address of Zeus Line Control Block
76(4C)	4	ATAENTRY	Address of terminal addressing entry
80(50)	4	ARQDECB	Address of requestor DECB for this terminal
84(54)	4	ARQIOA	Address of requestor I/O area
88(58)	4		unused
92(5C)	4	TCBNEXT	Address of next TCB
96(60)	4	TPIOECBL	
	1	1	
	. 3		Address (list) of ECB for terminal I/O
100(64)	4	TCBECBL	
	1	1	
	. 3		Address (list) of CPU synchronization ECB (TCB)
104(68)	4	NTPDECBL	
	1	1	
	. 3		Address (list) of non-IP DECB for I/O

(Continued)

Zeus Terminal Control Block—Detailed Description *(Continued)*

Offset	Bytes and Alignment	Field Name	Field Description, Contents, Meaning
108(6C)	4	DECBDA	Event control block
	1	. . . 1	Type field
	. 1	. 1 . . 1 . . .	Type field
	. . 2	. . . . .	Length
	4	. . . . .	DCB address
	4	. . . . .	Area address
	4	. . . . .	IOB address
	4	. . . . .	Key address
	4	. . . . .	Block reference address
140(88)	1	GLOSBLK	Relative position on terminal for glossary 'ROLL' request
	. 3	. . . . .	unused
	2	. . . . .	unused
146(8E)	. . 1	PAGECB	Page control byte
		1	Page contains valid data
		. 1	Prefix processing completed
		. . 1	Suffix processing completed
		. . . 1	Processing complete
		. . . . 1 . . .	CUR requested
		. . . . . 1 . .	ROL requested
		. . . . . . 1 .	DEL requested
		 1	MOD requested
147(8F)	. . . 1	PAGECB2	Page control byte extension
		1	COP requested
		. 1	CRE requested
		. . 1	DIS requested
		. . . 1	RES requested
		. . . . 1 . . .	LIS requested
		. . . . . 1 . .	LOC requested
		. . . . . 1 .	cursor reset required
		 1	commands will be executed
148(90)	8	USERNME	User name
156(98)	2	USERID	User identification
158(9A)	. . 2		unused
160(9C)	2		unused
162(9E)	. . 1	CMID	C'(' command id
163(9F)	. . . 1	0100 0000	space
164(A0)	4	AQWORK	Address of work space currently available
168(A4)	1	LSTAT	Line status byte
		1	TCB processing READ request
		. 1	TCB processing WRITE request
		. . X	reserved
		. . . 1	TCB waiting for terminal output
		. . . . 1 . . .	TCB waiting for terminal input
		. . . . . X . .	reserved
		. . . . . . 1 .	Terminal logged on
		 1	RJE function in process

(Continued)

Zeus Terminal Control Block—Detailed Description *(Continued)*

Offset	Bytes and Alignment	Field Name	Field Description, Contents, Meaning
169(A5)	. 1	INFOBLK	Relative position on terminal of the 'INFO' input area
170(A6)	. . 1	MSGBLK	Relative position on terminal for system message presentation
171(A7)	. . . 1	FILECB	File control byte
		1.	Locate record with key
		. X X X X X X X	reserved
172(A8)	4	AIOBUF	Address of terminal I/O buffer
176(AC)	1 2	TCQE	TCB core queue element (CQE)
176(AC)	1 2	TTQE	TCB time queue element (TQE)
188(B8)	2	CURCORD	XY cursor co-ordinates
190(BA)	. . 2	DATACORD	XY data co-ordinates
192 (BC)	4	TCBL	Length of TCB

ZEUS VECTOR TABLE

The Zeus Vector Table (ZVT) is used by all system routines to acquire data and addresses of system resources.

Zeus Vector Table—Overview

Offset	Word
0(0)	ZDSPTCHR
4(4)	ZMASTER
8(8)	ZITIMER
12(C)	ZRJE0100
16(10)	TCBQ
20(14)	ZLCBQ
24(18)	ZCOREQ
28(1C)	ZTIMERQ
32(20)	ZCNVTOOO
36(24)	ATE
40(28)	ETA

(Continued)

Zeus Vector Table—Overview *(Continued)*

Offset	Word
44(2C)	TPDCB
48(30)	ZNTPDCBT
52(34)	ZWTR00
56(38)	ZLINE10
60(3C)	ZTERMIO
64(40)	LTE
68(44)	ETL
72(48)	ZCTLTAB
76(4C)	ZMOVE
80(50)	ZMSCHD
84(54)	ZCHTB
88(58)	ZCTB
92(5C)	ZCMNDPC1
96(60)	ZCMNDPC2
100(64)	ZGETSAVE
104(68)	ZGETCORE
108(6C)	ZFREECOR
112(70)	
116(74)	
120(78)	ZQUEMGR
124(7C)	ZRCVRMGT
128(80)	ZRESET
132(84)	ZRTERM
136(88)	ZSCAN
140(8C)	ZSTACK

(Continued)

Zeus Vector Table—Overview *(Continued)*

Offset	Word
144(90)	ZSTATUS
148(94)	ZPOOLA
152(98)	ZLDR0100
156(9C)	ZLDR0000
160(A0)	ZENQCTRL
164(A4)	
168(A8)	ZCNTXCTL
172(AC)	ZMOVEX
176(B0)	ZRITQE
180(B4)	ZGCOUNT
184(B8)	ZLDR0200
188(BC)	ZSUPV
192(C0)	ZPROB
196(C4)	ZMOVE000
200(C8)	ZMOVE010
204(CC)	ZWTERM
208(D0)	ZCRTREAD
212(D4)	ZNTPIO
216(D8)	ZEMPTY
220(DC)	ZFULL
224(EO)	ZZS0B001
228(E4)	ZVTL

Zeus Vector Table—Detailed Description

Offset	Bytes and Alignment	Field Name	Field Description, Contents, Meaning
0(0)	4	ZDSPTCHR	Address of dispatch subroutine
4(4)	4	ZMASTER	Address of ZEUS command interface
8(8)	4	ZITIMER	Address of interval timer task
12(C)	4	RJE0100	Address of reader/interpreter task
16(10)	4	TCBQ	Address of first TCB on queue
20(14)	4	ZLCBQ	Address of first LCB on queue
24(18)	4	ZCOREQ	Address of first CQE on queue
28(1C)	4	ZTIMERQ	Address of first TQE on queue
32(20)	4	. . . . .	unused
36(24)	4	ATE	Address of ASC11 to EBCDIC translate table
40(28)	4	ETA	Address of EBCDIC to ASC11 translate table
44(2C)	4	TPDCB	Address of TP DCB
48(30)	4	ZNTPDCBT	Address of non-TP DCB table
52(34)	4	ZWTR00	Address of writer task
56(38)	4	ZLINEIO	Address of line I/O task
60(3C)	4	ZTERMIO	Address of terminal I/O task
64(40)	4	LTE	Address of line code to EBCDIC translate table
68(44)	4	ETL	Address of EBCDIC to line code translate table
72(48)	4	ZCTLTAB	Address of ASC11 control character table
76(4C)	4	ZMOVE	Address of move subroutine
80(50)	4	ZMSCHD	Address of master schedule subroutine
84(54)	4	ZCHTB	Address of convert hex to binary subroutine
88(58)	4	ZCTB	Address of convert to binary subroutine
92(5C)	4	ZCMNDPCI	Address of command processing subroutine-part 1
96(60)	4	ZCMNDPC2	Address of command processing subroutine-part 2
100(64)	4	ZGETSAVE	Address of GETSAVE subroutine
104(68)	4	ZGETCORE	Address of GETCORE subroutine
108(6C)	4	ZFREECOR	Address of FREECORE subroutine
112(70)	8		unused
120(78)	4	ZQUEMGR	Address of QUEMGR subroutine
124(7C)	4	ZRCVRMGT	Address of recovery management task
128(80)	4	ZRESET	Address of output buffer allocation subroutine
132(84)	4	ZRTERM	Address of terminal I/O interface subroutine
136(88)	4	ZSCAN	Address of SCAN subroutine
140(8C)	4	ZSTACK	Address of STACK subroutine
144(90)	4	ZSTATUS	Address of STATUS subroutine
148(94)	4	ZPOOLA	
	1	1.	No subpool
	. 3	. . . . .	Address of Zeus subpool

(Continued)

Zeus Vector Table—Detailed Description *(Continued)*

Offset	Bytes and Alignment	Field Name	Field Description, Contents, Meaning
152(98)	4	ZLDR0100	Address of transient routine loader task
156(9C)	4	ZLDR0000	Address of transient routine queue manager
160(A0)	4	ZENQCTRL	Address of ENQ/DEQ processor
164(A4)	4		reserved
168(A8)	4	ZCNTXCTL	Count subroutine 'XCTL' control
172(AC)	4	ZMOVEX	Address of character to hex move subroutine
176(B0)	4	ZRITQE	Address of reader-interpreter QFB
180(B4)	4	ZGCOUNT	Address of routine to format and dump
184(B8)	4	ZLDR0200	Address of transient table dump routine
188(BC)	4	ZSUPV	Address of supervisor state routine
192(C0)	4	ZPROB	Address of problem state routine
196(C4)	4	ZMOVE000	Address of move routine
200(C8)	4	ZMOVE010	Address of movex routine
204(CC)	4	ZWTERM	Address of TP output interface routine
208(D0)	4	ZCRTREAD	Address of real terminal routine
212(D4)	4	ZNTPIO	Address of non-TP I/O routine
216(D8)	4	ZEMPTY	Address of TP output buffer empty routine
220(DC)	4	ZFULL	Address of TP output buffer full routine
224(E0)	4	INIT0	Address of ZEUS first level initialization routine
228(E4)	2	ZVTL	Length of ZVT

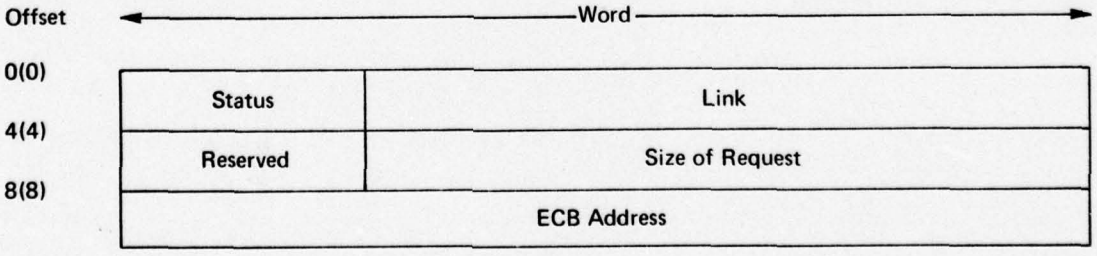
SECTION 2: QUEUE ELEMENTS

Descriptions of the Zeus Core Queue Element (CQE), Load Request Element (LRE), Resource Queue Element (RQE), and Timer Queue Element (TQE) are presented in this section.

CORE QUEUE ELEMENT

The Core Queue Element (CQE) is used, in conjunction with a conditional memory request, to queue a request for more main memory than is currently available.

Core Queue Element—Overview



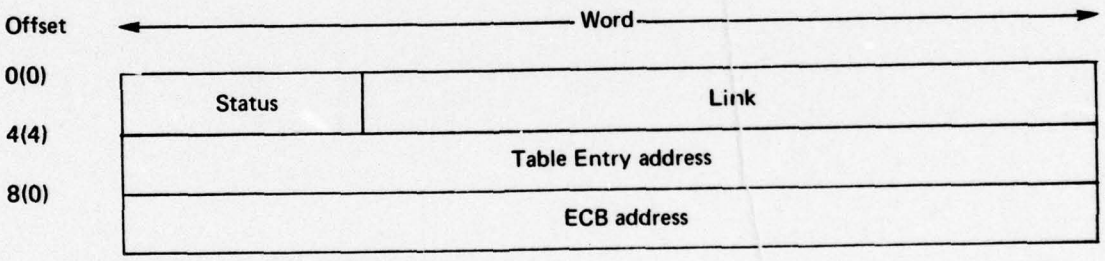
Core Queue Element—Detailed Description

Offset	Bytes and Alignment	Field Name	Field Description, Contents, Meaning
0(0)	1	Status	
		1.	Last entry on Queue
		.XXX XXXX	Reserved
	. 3	Link	Address of next CQE on core request queue. If last CQE on queue link will address previous CQE.
4(4)	1	XXXX XXXX	Reserved
	. 3	size	Amount of memory required to satisfy this request.
8(8)	4	Pose	Address of ECB which is to be posted complete when enough core becomes available to satisfy the request. When the ECB is posted complete the address of the core element obtained will be stored in its rightmost three bytes. The CQE is then removed from the Core Queue.

LOAD REQUEST ELEMENT

The Load Request Element (LRE) is used to process a load request for a transient module that is not currently in main memory.

Load Request Element—Overview



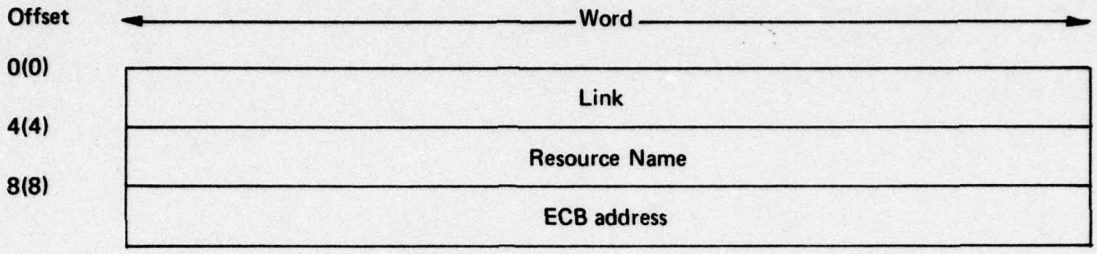
Load Request Element—Detailed Description

Offset	Bytes and Alignment	Field Name	Field Description, Content, Meaning
0(0)	1	Status	
		1.	Last Entry on Queue
		.XXX XXXX	Reserved
	. 3	Link	Address of next LRE on Load Request Queue. If last LRQ on queue link will address previous CQE
4(4)	4	Table	Address of table entry for module to be loaded
4(4)	4	Post	Address of ECB to be posted when module becomes available

RESOURCE QUEUE ELEMENT

The Resource Queue Element is used to synchronize accessing of system resources.

Resource Queue Element—Overview



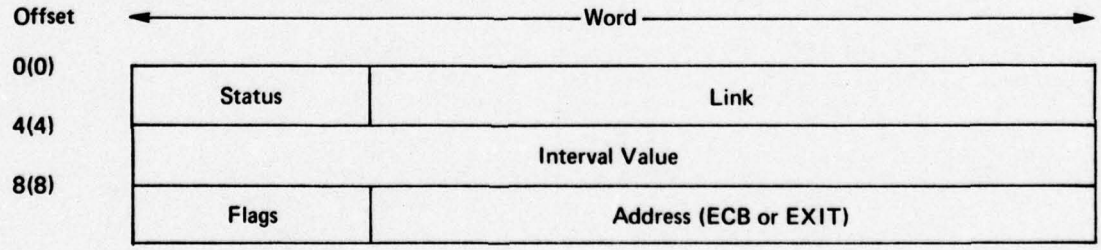
Resource Queue Element—Detailed Description

Offset	Bytes and Alignment	Field Name	Field Description, Content, Meaning
0(0)	1	Status	
		1.	Last Entry on Queue
		.XXX XXXX	Reserved
	. 3	Link	Address of next RQE on Resource Queue If last entry on queue, link points to previous entry
4(4)	4	RName	Resource name--any four characters, X'00' thru X'FF', may be used. If the name is less than 4 characters, it must be kept adjusted and padded with blanks
8(8)	4	Post	Address of Event Control Block

TIMER QUEUE ELEMENT

The Timer Queue Element (TQE) is used to create and maintain an interval timer.

Timer Queue Element—Overview



Timer Queue Element--Detailed Description

Offset	Bytes and Alignment	Field Name	Field Description, Content, Meaning
0(0)	1	Status	
		1.	Last Entry on Queue
		.XXX XXXX	Reserved
	. 3	Link	Address of next TQE on Timer Queue. If last element on the Timer Queue address of previous TQE.
4(4)	4	Time	Remaining interval this element at time of last Timer Queue 'post'. If this TQE is the first on the queue, then this interval was used to set the last 'real' clock.
8(8)	1	Post	
		1.	Exit address in next 3 bytes
		0.	ECB address in next 3 bytes
		.XXX XXXX	Reserved bits
8(8)	1 3		When the TQE interval is reduced to zero, the TQE is removed from the queue. Depending upon FLGS either routine is scheduled. This field contains either the exit address or the ECB pointer.

Chapter 4

UTILITY PROGRAMS

This chapter describes the programs necessary to implement, operate and maintain the Zeus system. A summary of the programs contained in this chapter is presented below.

Program Name	Function
BLDIFLE1	Format key file
BLDIFLE2	Add records to key file
BLDIFLE3	Replace records in key file
BLDXFILE	Format text files
BLDTFILE	Add modules to transient routines file
BLDRFILE	Format Remote Job Entry file
IMACOPY	Copy one or more files
IMAPDSMD	Update a partitioned data set
IMAPTPCH	Print/punch utility

NAME:	BLDIFLE1
FUNCTION:	To format a direct access file for input of the glossary records
INPUT:	None
OUTPUT:	A disk file of 1200 dummy records formatted for 40-byte keys
PROCESS:	The output file is opened. 1200 dummy records are written. The file is closed.
MACROS USED:	#REGS, #OPENP, #CLOSEP
ROUTINES CALLED:	None
SIZE IN BYTES:	434 bytes
EXTERNAL REFERENCES:	None
EXIT:	To address in R14
WRITTEN BY:	D. Shuford

NAME: BLDIFLE2

FUNCTION: To add records to the glossary (IFILE)

INPUT: A card file containing the QFILE record numbers of the records to be transferred to the IFILE;

The QFILE

OUTPUT: A printed listing of keys and relative record numbers of records added to the IFILE. Both the derived and actual record numbers are listed.

A disk file organized with 40-byte keys and 800-byte data records.

A punched or printed error file containing error messages and images of the input card records that caused the errors.

PROCESS: The input card file is read to determine which QFILE record is to be added to the IFILE. The QFILE record is read. The first line of the record, up to the delimiting period, carriage return characters, or the first 40 bytes is used as input to SCANKEY which removes spaces and special characters to and from the key and derives an IFILE relative record number for the record to be added. The record and key are written in the IFILE. The printed listing giving key and actual and derived relative locations is written. Any errors cause an error message to be written along with the card record that caused the error.

MACROS USED: #OPENP, #REGS, #CLOSEP

ROUTINES CALLED: SCANKEY, CTB

SIZE IN BYTES: 3,294 bytes

EXTERNAL REFERENCES: SCANKEY, CTB

EXIT: To address in R14

WRITTEN BY: D. Shuford

NAME: BLDIFLE3

FUNCTION: To update the IFILE.

INPUT: A card file containing relative record numbers of QFILE records to be transferred to the IFILE.
The QFILE.
The IFILE.

OUTPUT: The updated IFILE.
A printed listing of IFILE records added or replaced.
A printed or punched error file.

PROCESS: The files are opened. The card file is read to obtain the relative record number of the QFILE record that is to be transferred. The first 40 bytes or the characters up to the delimiter, period-carriage return, of the QFILE records are used as input to SCANKEY, which removes space and special characters to form the record key and derives a relative record number for the IFILE record.

The IFILE is read. If no record with the key being used is found, the QFILE record is written in the IFILE. If a record is found in the IFILE, with the key being used in the search, this record is checked to determine if it is "active." If so, an error message is written. If not, the QFILE record is written in place of the inactive IFILE record. A printed listing is written giving the keys and actual and derived relative record number of all records written in the IFILE.

An error file is produced for records that could not be written. A record may not be written because it was to replace an "active" record or because there was no room within the specified search limits.

MACROS USED: #OPENP, #REGS, #CLOSEP

ROUTINES CALLED: SCANKEY

SIZE IN BYTES: 3,718 bytes

EXTERNAL REFERENCES: SCANKEY

EXIT: To address in R14

WRITTEN BY: D. Shuford

NAME: BLDXFILE

FUNCTION: To allocate space for the D, F, and Q files.

INPUT: R1 PARM field

OUTPUT: A direct access file containing the number of records specified in the PARM field and of the size specified in the DCB BLKSIZE field on the DD card; a print file containing error messages.

PROCESS: The files are opened. The parm field is examined to see if the number of records has been specified. If not, an error message is printed and processing stops. If a number of records exceeding the maximum 32-bit binary number (2,147,483,647) is specified, an error message is pointed and processing stops. If a valid number of records is specified, this number of records is written. The records are formatted with the first 100 bytes set to zero and the remainder--up to 1,028 bytes--set to spaces. If errors occur while writing a record, an error message is printed and processing continues. When the last record is written, the files are closed and processing ceases.

MACROS USED: #REGS, #OPENP, #CLOSEP

ROUTINES CALLED: None

SIZE IN BYTES: 1,964 bytes

EXTERNAL REFERENCES: None

EXIT: To address in R14

WRITTEN BY: D. Shuford

NAME: BLDTFILE

FUNCTION: To load Zeus transient routines in executable form into partitioned data set.

INPUT: R1 Parm field for name of load module Linkage Editor output (SYSLMOD).

OUTPUT: The specified transient routine loaded into IMP1.TFILE.

PROCESS: The parm field is checked. If no name has been specified, or if a name exceeding eight characters is specified, an appropriate error message is printed and processing is discontinued. If a name with a valid length is specified, it is saved. The files are opened (if either file fails to open, an error message is printed and processing is discontinued). A BLDL instruction is issued for the input data set. The results are checked to make sure that the module in question does exist in executable form.

If not, the appropriate error message is printed and processing is discontinued. If so, the size is checked. If the size is eight bytes or 7,000 bytes, an error message is printed and processing discontinues. If the size is valid, the module is loaded and written into IMP1.TLOAD.

The PDS directory is updated to reflect the addition of the member with a STOW instruction. If an error occurs, the appropriate message is printed and processing discontinues. If there is no error the files are closed, a message indicating that the module has been successfully placed in the PDS is printed and processing discontinues.

ZEUS MACROS USED: #CLOSEP, #OPENP, #GOTO, #REGS

ROUTINES CALLED: None

SIZE IN BYTES: 1,330 bytes

EXTERNAL REFERENCES: None

EXIT: To address in R14

WRITTEN BY: W. Underhill

NAME:	BLDRFILE
FUNCTION:	To build the RFILE
INPUT:	Parm field data to specify file size
OUTPUT:	RFILE containing the specified number of 1,124-byte records. Records 0 and 1 are special records for use by the RJE functions of Zeus.
PROCESS:	The parm field is checked. If it is missing an error message is printed and processing ceases. If the file size is <8 or if (file size -8)/8 is >1,020 an error message is printed and processing ceases. If a valid file size is specified, records 0 and 1 are formatted and written. The remainder of the file is written with records formatted with zeroes. A message is printed, upon completion, saying that the file has been formatted and giving the record number of the last record. Processing ceases.
ZEUS MACROS USED:	N/A--program written in PL/1
ROUTINES CALLED:	WTP
SIZE IN BYTES:	1,048 bytes
EXTERNAL REFERENCES:	WTP
EXIT:	To System
WRITTEN BY:	W. Underhill

NAME:	IMACOPY
FUNCTION:	To copy any data set
INPUT:	Data set(s) to be copied--up to 10
OUTPUT:	Copied data set(s)
PROCESS:	The input data set is copied to the output data set. The process is repeated for each input and output data set specified.
ZEUS MACROS USED:	#OPENP, #CLOSEP
ROUTINES CALLED:	None
SIZE IN BYTES:	946 bytes
EXTERNAL REFERENCES:	IMACOPY
EXIT:	To address in R14
WRITTEN BY:	Jean A. Garneau

NAME:	IMAPDSMD
FUNCTION:	To add/replace/delete members of a partitioned data set (PDS).
INPUT:	Parameter card indicating what operation is to be performed; member to be added or replaced.
OUTPUT:	Modified PDS.
PROCESS:	The parameter card is read and interpreted to determine what operation is to be performed; the name of the member; whether to print the entire input data; whether an identification is to start in location 73 and if so what it is; and what the increment for sequencing is to be. The specified PDS is then modified as indicated.
ZEUS MACROS USED:	#REGS, #OPENP, BSECT, RET, CLOSEP
ROUTINES CALLED:	None
SIZE IN BYTES:	3,937 bytes
EXTERNAL REFERENCES:	None
EXIT:	To address in R14
WRITTEN BY:	Jean A. Garneau

NAME:	IMAPTPCH
FUNCTION:	Utility for printing or punching sequential data sets.
INPUT:	Sequential data set. PARM field indicating operation to be performed.
OUTPUT:	Printed or punched data set.
PROCESS:	The PARM field is interrogated to determine which operation is to be performed; at which record in the data set the operation is to begin; whether any subsequent records are to be skipped; whether the entire operation is to be repeated, and if so for how many times; whether sequence numbers are to be placed in the output record (if punched) and if so, what the initial value and the increment are to be; whether printed output is to be spaced, and if so how many blank lines (0 to 3) are to be between records; whether conversion to hexadecimal or to EBCDIC is to be made. The specified operations are performed and the output data set written.
ZEUS MACROS USED:	#REGS, #OPENP, #GOTO, #BITS, #BSECT, #RET, #CLOSEP
ROUTINES CALLED:	CTB, MOVEX, MOVE
SIZE IN BYTES:	3,176 bytes
EXTERNAL REFERENCES:	CTB, MOVEX, MOVE, IECTTRNS, IMAPTPCH
EXIT:	Address in R14
WRITTEN BY:	Jean A. Garneau

Chapter 5

PROCEDURES

This chapter lists all the Job Control Language (JCL) procedures required to support the Zeus system. A summary of the procedures, along with a brief description of them, is presented in the following list:

Member Name	Description
#ASMFC	Assemble and store object module in Zeus library
#ASMFCG	Assemble and Go
#ASMFCL	Assemble and Linkedit
#ASMFCZ	Assemble and store object code in Zeus object code file
#ASMFCLG	Assemble, link, and go
#ASMFCLT	Assemble, linkedit, and store executable module in Zeus Transient Routine file
MOVE	Mode data set
COMPRESS	Compress a partitioned data set
PDSMOD	Modify a partitioned data set
RDRZ	Zeus reader
RGET	Extract data set off-line from RJE file
RPUT	Add data set off-line to RJE file
UT	Print/Punch file
ZEUS	Zeus procedure
ZOLD	Zeus Job (Old Zeus)
ZREG	Zeus Job (Regular Zeus)
ZTEST	Zeus Job (New Zeus)

```
MEMBER NAME  #ASMFC
//         PROC  NAME=TEMP
//ASM      EXEC  PGM=IEUASM,PARM='NODECK,LOAD,NOXREF',TIME=20
//SYSLIB   DD    DSN=IMP1.ZMACRO,DISP=SHR
//         DD    DSN=SYS1.MACLIB,DISP=SHR
//SYSPRINT DD    SYSOUT=A,DCB=(LRECL=121,RECFM=FBM,BLKSIZE=363,BUFNO=5)
//SYSUT1   DD    SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=SYSPRINT)
//SYSUT2   DD    SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=(SYSPRINT,SYSUT1))
//SYSUT3   DD    SPACE=(CYL,(3,3)),UNIT=(SYSDA,
//                SEP=(SYSUT2,SYSUT1,SYSLIB))
//SYSGO    DD    DISP=OLD,DSN=SYS1.GARN(&NAME)
```

```
MEMBER NAME  #ASMFCG
//ASM EXEC PGM=IEUASM,PARM='NODECK,LOAD',TIME=10
//SYSPRINT DD SYSOUT=A,DCB=(LRECL=121,RECFM=FBM,BLKSIZE=363,BUFNO=9)
//SYSLIB   DD    DSNAME=IMP1.ZMACRO,DISP=SHR
//         DD    DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1   DD SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=SYSPRINT)
//SYSUT2   DD SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=(SYSPRINT,SYSUT1))
//SYSUT3   DD    UNIT=(SYSDA,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//                SPACE=(CYL,(3,4))
//SYSGO    DD    UNIT=SYSDA,DSNAME=&LOADSET,SPACE=(CYL,(1,1)),DISP=(MOD,
//                PASS)
//GO       EXEC  PGM=LOADER,COND=(8,LT,ASM)
//SYSLIB   DD    DSNAME=HUM1.SUBRTLIB,DISP=SHR
//         DD    DSN=IMP1.SUBRTLIB,DISP=SHR
//SYSLIN   DD    DSNAME=*.ASM.SYSGO,DISP=(OLD,PASS)
//         DD    DDNAME=SYSDIN
//SYSLOUT  DD    SYSOUT=A
//SYSPRINT DD    SYSOUT=A
//SYSUDUMP DD SYSOUT=A
```

```
MEMBER NAME  #ASMFCL
//ASM     EXEC  PGM=IEUASM,PARM='LOAD,NODECK',TIME=10
//SYSPRINT DD SYSOUT=A,DCB=(LRECL=121,RECFM=FBM,BLKSIZE=363,BUFNO=9)
//SYSLIB DD    DSNAME=IMP1.ZMACRO,DISP=SHR
//        DD   DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1 DD SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=SYSPRINT)
//SYSUT2 DD SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=(SYSPRINT,SYSUT1))
//SYSUT3 DD    UNIT=(SYSDA,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//            SPACE=(CYL,(3,4))
//SYSGO  DD    UNIT=SYSDA,DSNAME=&LOADSET,SPACE=(CYL,(1,1)),DISP=(MOD,
//            PASS)
//LKED    EXEC  PGM=IEWL,PARM=(XREF,LIST,LET),COND=(8,LT,ASM)
//SYSPRINT DD SYSOUT=A,DCB=(LRECL=121,RECFM=FBM,BLKSIZE=363,BUFNO=9)
//SYSLMOD DD   DSNAME=&TEMP(PDS),UNIT=SYSDA,SPACE=(CYL,(1,1,1)),
//            DISP=(MOD,PASS)                                          0030000
//SYSLIB DD    DSNAME=SYS1.LINKLIB,DISP=SHR
//        DD   DSNAME=HUM1.SUBRTLIB,DISP=SHR
//        DD   DSN=IMP1.SUBRTLIB,DISP=SHR
//SYSLIN DD    DSNAME=&LOADSET,DISP=(OLD,DELETE)                        0024000
//        DD   DDNAME=SYSIN                                             0026000
//SYSUT1 DD    UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),                      2
//            SPACE=(CYL,(1,1))
```

```
MEMBER NAME  #ASMFCZ
//          PROC  NAME=TEMP
//ASM       EXEC  PGM=IEUASM,PARM='NODECK,LOAD,XREF',TIME=20
//SYSLIB    DD    DSN=IMP1.ZMACRO,DISP=SHR
//          DD    DSN=SYS1.MACLIB,DISP=SHR
//SYSPRINT DD   SYSOUT=A,DCB=(LRECL=121,RECFM=FBM,BLKSIZE=363,BUFNO=5)
//SYSUT1    DD    SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=SYSPRINT)
//SYSUT2    DD    SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=(SYSPRINT,SYSUT1))
//SYSUT3    DD    SPACE=(CYL,(3,3)),UNIT=(SYSDA,
//            SEP=(SYSUT2,SYSUT1,SYSLIB))
//SYSGO     DD    DISP=OLD,DSN=IMP1.ZOBJ(&NAME)
```

```
MEMBER NAME  #ASMFCLG
//ASM     EXEC  PGM=IEUASM,PARM='NODECK,LOAD',TIME=10
//SYSPRINT DD SYSOUT=A,DCB=(LRECL=121,RECFM=FBM,BLKSIZE=363,BUFNO=9)
//SYSLIB DD     DSNAME=IMP1.ZMACRO,DISP=SHR
//        DD    DSNAME=SYS1.MACLIB,DISP=SHR
//SYSUT1 DD SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=SYSPRINT)
//SYSUT2 DD SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=(SYSPRINT,SYSUT1))
//SYSUT3 DD     UNIT=(SYSDA,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//              SPACE=(CYL,(3,4))
//SYSGO  DD     DSNAME=&LOADSET,UNIT=SYSDA,SPACE=(CYL,(1,1)),
//              DISP=(MOD,PASS)                                          0018000
//LKED   EXEC  PGM=IEWL,PARM=(XREF,LET,LIST),COND=(8,LT,ASM)
//SYSPRINT DD SYSOUT=A,DCB=(LRECL=121,RECFM=FBM,BLKSIZE=363,BUFNO=9)
//SYSLMOD DD    DSNAME=&GOSET(GO),UNIT=SYSDA,SPACE=(CYL,(1,1,1)),
//              DISP=(MOD,PASS)                                          0030000
//SYSLIB DD     DSNAME=SYS1.LINKLIB,DISP=SHR
//        DD    DSNAME=HUM1.SUBRTLIB,DISP=SHR
//        DD    DSN=IMP1.SUBRTLIB,DISP=SHR
//SYSLIN DD     DSNAME=&LOADSET,DISP=(OLD,DELETE)                        0024000
//        DD    DDNAME=SYSIN                                             0026000
//SYSUT1 DD     UNIT=(SYSDA,SEP=(SYSLIN,SYSLMOD)),                      20032000
//              SPACE=(CYL,(1,1))
//GO      EXEC  PGM=*.LKED.SYSLMOD                                       0038000
//SYSPRINT  DD  SYSOUT=A
//SYSUDUMP DD SYSOUT=A
```

```
MEMBER NAME  #ASMFCLT
//          PROC NAME=TEMP
//ASM       EXEC   PGM=IEUASM,PARM=(NODECK,LOAD,XREF),TIME=10
//SYSPRINT DD  SYSOUT=A,DCB=(RECFM=FBM,LRECL=121,BLKSIZE=363,BUFNO=9)
//SYSLIB    DD DSN=IMP1.ZMACRO,DISP=SHR
//          DD DSN=SYS1.MACLIB,DISP=SHR
//SYSUT1    DD SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=SYSPRINT)
//SYSUT2    DD SPACE=(CYL,(3,3)),UNIT=(SYSDA,SEP=(SYSPRINT,SYSUT1))
//SYSUT3    DD UNIT=(SYSDA,SEP=(SYSUT2,SYSUT1,SYSLIB)),
//             SPACE=(CYL,(3,4))
//SYSGO     DD DSN=&&LOADSET,UNIT=SYSDA,SPACE=(CYL,(1,1)),
//             DISP=(MOD,PASS)
//LKED      EXEC   PGM=IEWL,PARM=(XREF,LIST,LET),COND=(0,NE,ASM)
//SYSPRINT DD  SYSOUT=A,DCB=(RECFM=FBM,LRECL=121,BLKSIZE=363,BUFNO=9)
//SYSLMOD   DD DSN=&&GOSET(&NAME),UNIT=SYSDA,SPACE=(CYL,(1,1,1)),
//             DISP=(MOD,PASS)
//SYSLIB    DD DUMMY
//SYSLIN    DD DSN=&&LOADSET,DISP=(OLD,DELETE)
//SYSUT1    DD UNIT=SYSDA,SPACE=(CYL,(1,1))
//LOAD      EXEC   PGM=TLOAD,PARM=(&NAME),COND=(0,NE,ASM)
//STEPLIB   DD DSN=IMP1.DATALIB.UNDERHIL,DISP=SHR
//INPUT     DD DSN=&&GOSET,DISP=(OLD,DELETE)
//OUTPUT    DD DSN=IMP1.TFILE,DISP=SHR
//SYSUDUMP DD SYSOUT=A
```

```
MEMBER NAME  MOVE
//        PROC  SER1=USRLIB,SER2=USRLIB,SER3=USRLIB,SER4=USRLIB
//MOVE    EXEC  PGM=IEHMOVE
//SYSPRINT DD   SYSOUT=A
//SYSUT1 DD     UNIT=2314,DISP=OLD,VOL=SER=USRLIB
//V1       DD   UNIT=2314,VOL=SER=&SER1,DISP=OLD
//V2       DD   UNIT=2314,VOL=SER=&SER2,DISP=OLD
//V3       DD   UNIT=2314,VOL=SER=&SER3,DISP=OLD
//V4       DD   UNIT=2314,VOL=SER=&SER4,DISP=OLD
```

```
MEMBER NAME  COMPRESS
//        PROC  ON=
//COMPRESS EXEC PGM=IEBCOPY
//SYSPRINT DD   SYSOUT=A
//INPUT    DD    &ON,DSN=&PDS,DISP=OLD,UNIT=2314
//SYSIN    DD   DSN=HUM1.PCSLIB(COPY),DISP=SHR
//SYSUT3 DD     UNIT=DISK,SPACE=(TRK,(1,25))
//SYSUT4 DD     UNIT=DISK,SPACE=(TRK,(1,25))
```

```
MEMBER NAME  PDSMOD
//*
//*       UPDATE PARTITIONED DATASET
//*
//        EXEC  PGM=IMAPDSMD
//SYSPRINT DD   SYSOUT=A,DCB=(LRECL=133,BLKSIZE=399,RECFM=FBA)
//SYSLIB DD     DSNAME=&PDS,DISP=OLD
```

```
MEMBER NAME  RDRZ
//        PROC  LIB='IMP1.ZSOURCE',MBR=ZREG                                  0000001
//IEFPROC EXEC PGM=IEFIRC,PARM='805030040010249070015YSDA    '               0000002
//IEFRDER DD    DSNAME=&LIB.(&MBR),DISP=SHR                                  0000003
//IEFPDSI DD    DSNAME=SYS1.PROCLIB,DISP=SHR                                 0000004
//IEFDATA DD    UNIT=SYSDA,SPACE=(80,(5,100),RLSE,CONTIG),                   20000005
//               DCB=(LRECL=80,BUFNO=2,BLKSIZE=800,RECFM=FB,BUFL=800)        0000006
//SYSABEND DD SYSOUT=A,SPACE=(CYL,(10,10))                                   0000007
```

```
MEMBER NAME  RGET
//R            EXEC   PGM=RGETPUT,PARM='OUT,&LOGON'
//STEPLIB      DD     DSN=IMP1.DATALIB.UNDERHIL,DISP=SHR
//SYSPRINT     DD     SYSOUT=A,DCB=(RECFM=F,BLKSIZE=100)
//SYSUDUMP     DD     SYSOUT=A
//RFILE        DD     DSN=IMP1.RFILE,DISP=SHR
```

```
MEMBER NAME  RPUT
//             PROC   LOGON=TEMP
//R            EXEC   PGM=RGETPUT,PARM='IN,&LOGON'
//STEPLIB      DD     DSN=IMP1.DATALIB.UNDERHIL,DISP=SHR
//SYSPRINT     DD     SYSOUT=A,DCB=(RECFM=F,BLKSIZE=100)
//SYSUDUMP     DD     SYSOUT=A
//RFILE        DD     DSN=IMP1.RFILE,DISP=SHR
```

```
MEMBER NAME  UT
//UT       EXEC  PGM=IMAPTPCH
//SYSPRINT DD    SYSOUT=A,DCB=(LRECL=133,RECFM=FBA,BLKSIZE=399)
//SYSPUNCH DD    SYSOUT=B,DCB=(LRECL=80,BLKSIZE=240,RECFM=FB)
```

```
MEMBER NAME  ZEUS
//        PROC   PROG=STUDMODE,LIB='IMP1.PROGLIB'
//ZEUS    EXEC   PGM=&PROG,TIME=1000
//STEPLIB DD     DISP=SHR,DSN=&LIB
//CRTL0   DD     UNIT=050   720 CRT LINE 1
//CRTL1   DD     UNIT=051   720 CRT LINE 2
//CRTL3   DD     UNIT=053   720 CRT LINE 4
//DUMMY   DD     UNIT=AFF=CRTL0 DUMMY TP LINE
//FILED   DD     DSNAME=CAI1.DFILEV3,DISP=SHR
//FILEF   DD     DSNAME=CAI1.FFILEV3,DISP=SHR
//FILEI   DD     DSNAME=CAI1.IFILEV1,DISP=SHR
//FILEQ   DD     DSNAME=CAI1.QFILEV3,DISP=SHR
//FILER DD DSN=IMP1.RFILE,DISP=SHR
//TFILE DD DSN=IMP1.TFILE,DISP=SHR
//SUBRFL  DD     DISP=SHR,DSN=CAI2.SRFILE
//SYS010 DD DSN=CAI1.SYS010,DISP=SHR
//SYS011 DD UNIT=2314,DSN=CAI2.SYS011,DISP=SHR
//SYS012   DD   DSNAME=CAI1.SYS012,DISP=SHR
//SYS020 DD      DISP=SHR,DSN=CAI2.SYS020
//SYS021 DD      DISP=SHR,DSN=CAI2.SYS021
//SYS022   DD    DISP=SHR,DSN=CAI2.SYS022
//LDAFLE DD      DISP=SHR,DSN=CAI2.CAILDA
//SYSLST DD SYSOUT=A,UNIT=SYSOUT
//D1050C DD      UNIT=AFF=DUMMY
//   DD   UNIT=AFF=DUMMY
//   DD   UNIT=AFF=DUMMY
//   DD   UNIT=AFF=DUMMY
//         DD     UNIT=AFF=DUMMY
//         DD     UNIT=AFF=DUMMY
//         DD     UNIT=AFF=DUMMY
//         DD     UNIT=AFF=DUMMY
//         DD     UNIT=AFF=DUMMY
//ZLINE1 DD      UNIT=AFF=CRTL0
//       DD     UNIT=AFF=CRTL1
//       DD     UNIT=AFF=CRTL3
//SYSUDUMP DD SYSOUT=F,UNIT=SYSOUT
//RJEFILE  DD   DSNAME=IMP1.RJEFILE,DISP=SHR
//*
//* THE FOLLOWING 3 DD CARDS ARE REQUIRED ONLY IF THE ZEUS WRITER
//*TASK IS TO BE MADE ACTIVE
//*
//SYSWTR1 DD DSN=SYS1.POSTFILE,DISP=SHR
//SYSWTR2 DD DSN=SYS1.WTRSPOOL,DISP=SHR,DCB=RECFM=U
//SYSWTR3 DD UNIT=00E,DCB=(RECFM=UM,BLKSIZE=133)
```

```
MEMBER NAME  ZOLD
//IMPONL JOB (253A,IMPONL,0),'OLD ZEUS',CLASS=M,MSGLEVEL=1
// EXEC ZEUS,PROG=STUDMODF,PARM=CWV210BJ
//SYSIN DD *,DCB=BLKSIZE=80
(10,11,9),(7,8,6),(4,5,3),ALLOCATE=8800
```

```
MEMBER NAME  ZREG
//IMPONL JOB (253A,IMPONL,0),'CURRENT ZEUS',CLASS=M,MSGLEVEL=1
// EXEC ZEUS,PARM=CWV210BJ,PROG=STUDMODE
//SYSIN DD *,DCB=BLKSIZE=80
(10,11,9),(7,8,6),(4,5,3),ALLOCATE=8800
```

```
MEMBER NAME  ZTEST
//IMPONL JOB (253A,IMPONL,0),'NEW ZEUS',CLASS=M,MSGLEVEL=1
// EXEC ZEUS,PROG=TESTMODE,LIB='SYS1.GARNEAU',PARM=CWV210BJ
//SYSIN DD *,DCB=BLKSIZE=80
(10,11,9),(7,8,6),(4,5,3),ALLOCATE=8800
```