3

SOFTWARE COMPONENT SPECIFICATION

SYSTEM:	LEVEL 6 MOD400 OPERATING
SUBSYSTEM:	LOCAL AREA NETWORK
COMPONENT:	LACS DRIVER MEGABUS SERVICES
PLANNED RELEASE:	MOD400 4.0
SPECIFICATION REVISION NUMBER:	B
DATE:	JULY 23,1985
AUTHOR:	PETER STOPERA

This specification describes the current definition of the subject software component, and may be revised in order to incorporate design improvements.

TABLE OF CONTENTS

		PAGE
	REFERENCES	3
1	INTRODUCTION AND OVERVIEW	4
1.1	BACKGROUND	4
1.2	BASIC PURPOSE	4
1.3	BASIC STRUCTURE	4
1.4	BASIC OPERATION	7
1.4.1	DETERMINATION ROUTINE	7
1.4.1.1	DETERMINATION ROUTINE ERRORS	7
1.4.2	ISSUE IORL ROUTINE	7
1.4.2.1	ISSUE IOLD ROUTINE ERRORS	7
1.4.3	ISSUE IO ROUTINE	8
1.4.3.1	ISSUE IO ROUTINE ERRORS	8
1.4.4	INTERRUPT PROCESSING ROUTINE	8
1.4.4.1	INTERRUPT PROCESSING ROUTINE ERRORS	8
1.4.5	FLOW CONTROL ROUTINE	8
1.4.5.1	PRE LCB FLOW CONTROL ROUTINE	9
1.4.5.2	PRE LCB FLOW CONTROL ROUTINE ERRORS	9
1.4.5.2	POST LCB FLOW CONTROL ROUTINE	9
1.4.5.2.1	POST LCB FLOW CONTROL ROUTINE ERRORS	9
1.4.6	NAK'D ROUTINE	9
1.4.6.1	NAK'D ROUTINE ERRORS	9
2	EXTERNAL SPECIFICATION	9
2.1	OWNED DATA STRUCTURES	9
2.2	EXTERNAL INTERFACES	10
2.2.1	MOD400 EXECUTIVE SOFTWARE ROUITNES	10
2.2.1.1	ZXREQ	10
2.2.1.2	ZHCOMM	10
2.2.1.3	ZXD_PR	10
2.2.2	MOD400 DATA STRUCTURES IMPLEMENTED	10
2.2.3	USER INTERFACES	10
2.2.3.1	LDMS INTERFACE	10
2.2.3.2	LCB FORMAT	11
2.3	INITIALIZATION REQUIREMENTS	16
2.4	TERMINATION REQUIREMENTS	16
2.5	ENVIRONMENT	16
2.6	TIMING AND SIZE REQUIREMENTS	17
2.7	ASSEMBLY AND LINKING	17
2.8	TESTING CONSIDERATIONS	17
2.9	DOCUMENTATION CONSIDERATIONS	17
2.10	ERROR MESSAGES	17
2.10.1	INTERRUPT PROCESSING ERRORS	17

3	INTERNAL SPECIFICATION	17
3.1	OVERVIEW	17
3.2	SUBCOMPONENT DESCRIPTION	18
3.2.1	DETERMINATION ROUTINE	18
3.2.2	ISSUE IOLD ROUTINE	18
3.2.3	ISSUE IO ROUTINE	20
3.2.4	INTERRUPT PROCESSING ROUTINE	20
3.2.5	FLOW CONTROL ROUTINES	22
3.2.5.1	PRE ORDER FLOW CONTROL ROUTINE	22
3.2.5.1	POST ORDER FLOW CONTROL ROUTINE	22
3.2.6	NAK'D ROUTINE	22
4	PROCEDURAL DESIGN LANGUAGE	22
5	ISSUES	22

REFERENCES

[1] CLM User Extensions, Richard Taufman, May 14,1979.

[2] Engineering Product Specification (H/W), Local Area Controller Subsystem (LACS), Rev F, A. C. Hirtle, Oct 4, 1984.

[3] Engineering Product Specification, LAN Software, R. Dhondy, Aug. 16, 1985.

[4] LAN S/W Component Specification, System Management, D. O'Shaughnessy Aug. 16, 1985.

[5] LAN S/W Component Specification, LACS Driver Interface Services, P. Stopera, Aug. 16, 1985.

[6] LAN S/W Component Specification, LACS Driver Megabus Services, P. Stopera, Aug. 16, 1985.

[7] Lan S/W Component Specification, Configuration Requirments L. Vivaldi, Aug. 16, 1985.

[8] LAN S/W Component Specification, LACS Link Layer Protocol, H. King Aug. 16, 1985.

[9] L6 LAN Data Structures, P. Stopera - Aug. 16, 1985

1 INTRODUCTION AND OVERVIEW

1.1 BACKGROUND

The lacs driver megabus services (ldms) is a component of the lacs driver in the lan subsystem. The ldms is the lacs driver's interface to the lacs via the l6 megabus. The ldms is used by the lacs driver's layer servers (ls) when they wish to send io or iold requests to the lacs.

1.2 BASIC PURPOSE

The ldms has several purposes, they are: to issue iold's to the lacs, to process interrupts from the lacs, to issue io's to the lacs, to maintain flow control at the controller level and to process nak'd io's.

1.3 BASIC STRUCTURE

Figure 1 shows the relation of the ldms to the other components of the lacs driver. Figure 2 shows the subcomponents of the ldms. The following is a brief description of the functions of each subcomponent of the ldis:

determination routine - This routine determines whether to call the issue io or issue iold routine. The determination is made from the input parameters. The routine is also responsible for returning to the calling routine.

issue iold routine - This routine will issue an iold to the lacs. The buffer address and range in the iold represent the lan control block (lcb). The lcb is supplied as an input parameter when the ls calls the ldms. The routine is also responsible in queuing lcbs on the llt, checking controller states, calling the flow control routine, and handling naks by the megabus.

issue io routine - This routine will issue an io to the lacs. The function to be performed is supplied as an input parameter when the ls calls the ldms. The routine is also responsible for checking controller states and handling naks by the megabus.

interrupt processing routine - This routine is responsible for handling interrupts from the lacs. The interrupts will be a result of the lacs completing a lcb. The routine will dequeue completed lcbs off the llt queue, call the flow control routine, and depending on how the lcb is set up the routine will lnj to the ls which issued the lcb to the ldms, or perform a task request, or release the lcb's memory.

flow control routine - This routine is responsible for maintaining the count of lcbs outstanding at a controller. The routine will queue lcbs if the controller limits are exceeded, and dequeue and issue lcb via a call to the issue iold routine when the limits recede.

LAN L6 SOFTWARE ARCHITECTURE

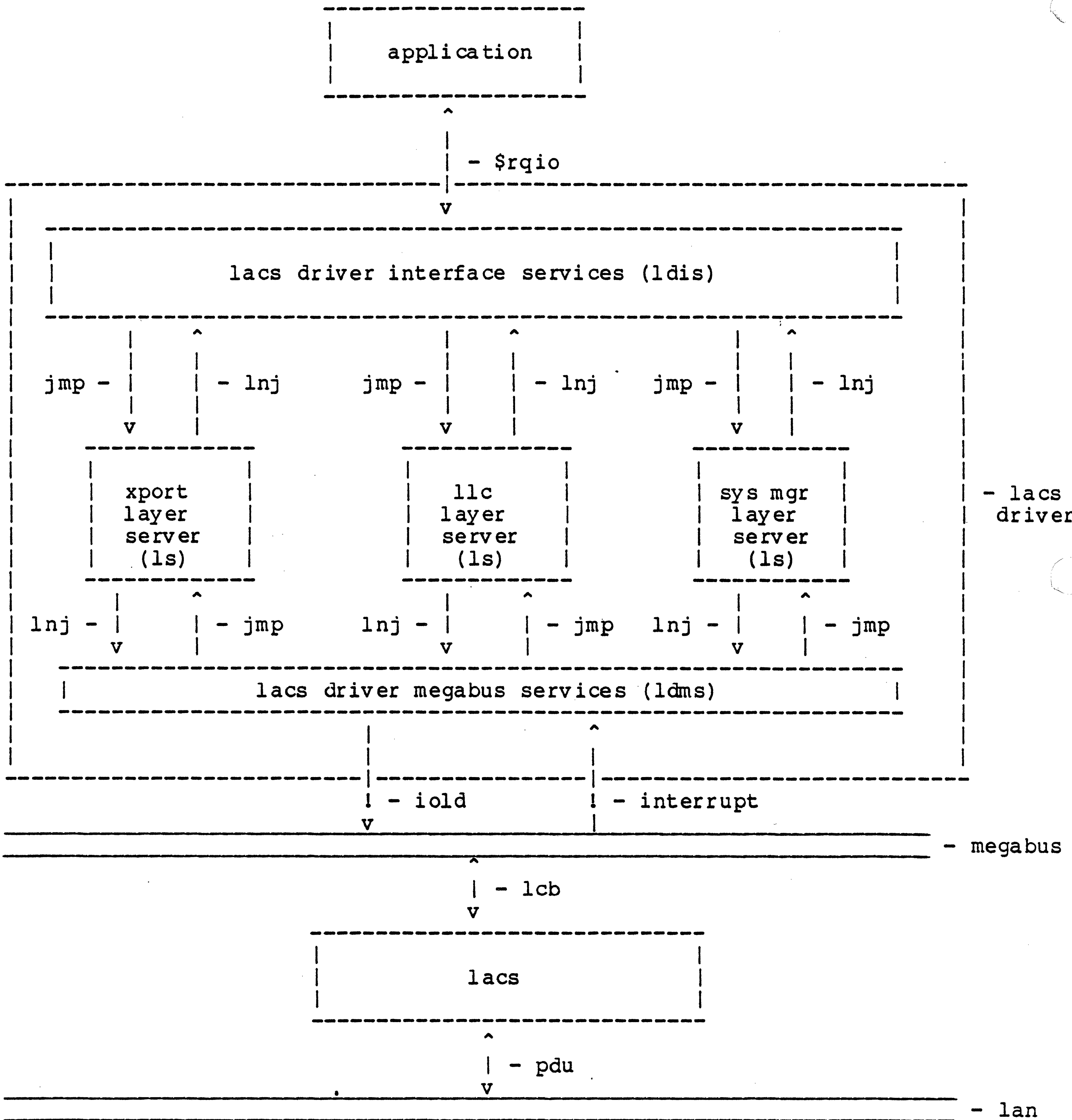

figure 1

LACS DRIVER MEGABUS SERVICES SUBCOMPONENTS

```
 ----------------------------------------
|                                        |
|        determination routine           |
|                                        |
 ----------------------------------------
|                                        |
|          issue iold routine            |
|                                        |
|----------------------------------------|
|                                        |
|           issue io routine             |
|                                        |
|----------------------------------------|
|                                        |
|             nak'd routine              |
|                                        |
|----------------------------------------|
|                                        |
|     interrupt processing routine       |
|                                        |
|----------------------------------------|
|                                        |
|         flow control routine           |
|                                        |
 ----------------------------------------
```

figure 2

nak'd routine - This routine is responsible for processing a io or iold which was nak'd an excessive amount of times during processing of the issue io or issue iold routines.

1.4 BASIC OPERATION

1.4.1 DETERMINATION ROUTINE

The determination routine will be invoked via a lnj from a routine wishing to issue a io or iold to the controller. The routine will save the return address. The routine will then validate the input parameters, returning to the calling routine if any of the parameters are invalid. Then using the input parameters the routine determines whether to call the issue iold or issue io routine. The call to the issue routine is a lnj, the issue routines are expected to return to the determination routine. Upon return from a issue routine the determination routine will retrieve the saved return address (of the is), then will return to the caller with a status which was set up by the issue routine.

1.4.1.1 DETERMINATION ROUTINE ERRORS

Errors reported by the ldms determination routine are:

1. Invalid input parameters when the routine is called.

1.4.2 ISSUE IOLD ROUTINE

This routine is invoked via a lnj from the determination routine. The routine saves the return address. The routine will call the flow control (fc) routine, if the fc routine return with an error status, the issue iold routine will return to the determination routine with the error message from the fc routine. The routine will next check the controller state, if the controller is in the Otherwise, the routine will set up the registers for the iold, using information in the lcb, the nak'd retry count is reset, and the iold is issued. If the iold is nak'd the routine will set the controller into a nak'd state, increment the retry count, then try the iold again. If the iold is nak'd repeatedly, the nak'd routine will be called. If a nak'd iold is successful the controller is placed out of the nak'd state and a successful return to the determination routine is performed. If the iold is successful a return to the determination routine is performed.

1.4.2.1 ISSUE IOLD ROUTINE ERRORS

Errors reported by the issue iold routine are:

1. The flow control routine return with an error.

2. iold is nak'd.

1.4.3 ISSUE IO ROUTINE

The issue io routine is invoked via a inj from the determination routine. The routine saves the return address. The routine will issue the io using the supplied input parameters. If the io is nak'd, the routine will retry the io a set number of times until the io is successful or the limit of retrying the io is reached. If the retry limit was exceeded the routine will call the nak'd routine. Otherwise, a successful return to the caller (determination routine) is done.

Note, if a input id io is sent to a controller which is in a active state (i.e. ready to recieve iold's) the controller will hang, therefore the state of the controller must be checked before sending an input id.

1.4.3.1 ISSUE IO ROUTINE ERRORS

Errors reported by the issue io routine are:

1. Io is nak'd.

1.4.4 INTERRUPT PROCESSING ROUTINE

The interrupt processing routine is invoked via a interrupt from the lacs. The lacs interrupts when it has completed a lcb. The routine will retrieve the interrupt control word (word 0 of the interrupt level tcb). From the icw bits the routine will obtain the pointer to the lit in which the completed lcb is queued. The queue of lcb's is searched until a completed lcb is found. When found the lcb is dequeued from the active lcb queue. The pointer to the next lcb in the queue is saved, and the flow control routine is called. Upon return from the fc routine, the interrupt processing routine will perform the function specified in the l6 portion of the lcb. The options are: performing a task request, performing a inj to a layer server routine, or nothing. After the routine performs the specified request, the pointer to the next lcb in the queue is retrieved, then the list is searched until another completed lcb is found or the end of the queue is reached. If another completed lcb is found, the routine will repeat the above steps until the end of queue is reached. If the end of the queue is reached, the level will be exited via a lev instruction.

1.4.4.1 INTERRUPT PROCESSING ROUTINE ERRORS

Errors reported by the interrupt processing routine are:

1. icw = 0.

1.4.5 FLOW CONTROL ROUTINE

There are 2 flow control routines called the pre lcb flow control routine and post lcb flow control routine.

1.4.5.1 PRE LCB FLOW CONTROL ROUTINE

The pre lcb fc routine is invoked via a call from the issue iold routine. The routine will test if a lcb can be sent to this controller, this check is done through counts kept in a lan data sturcture. If the controller has sufficent resources the pre lcb fc routine will increment the iold count, then return to the issue iold routine with an successful status. If an iold can not be sent to the controller, the routine will queue the lcb on a wait queue and return to the issue iold routine with an error status.

1.4.5.1.1 PRE LCB FLOW CONTROL ROUTINE ERRORS

Errors reported by the pre lcb flow control routine are:

1. Lcb queued because of flow control.

1.4.5.2 POST LCB FLOW CONTROL ROUTINE

The post lcb fc routine is invoked via a call from the interrupt processing routine after a completed lcb has been found. The routine will decrement the iold count for the controller, then if there are any lcb waiting to the sent the routine will dequeue the head lcb off the wait queue, call the ldms determintion routine, then return to the interrupt processing routine. Otherwise, if there are no lcb waiting the routine will return to the interrupt processing routine with after decremting the controller iold count.

1.4.5.2.1 POST LCB FLOW CONTROL ROUTINE ERRORS

Errors which are reported by the post fc routine are:

1. The ldms returns with an error.

1.4.6 NAK'D ROUTINE

The nak'd routine is invoked via a call from the issue io or issue iold routines when an iold has been nak'd. If the iold was issued (i.e. the iold went through on subsquent tries) the routine will issue all iold queued on the nak'd queue off the controller directory, then return to the calling routine. If th eiold was not issued, the routine will call the sm ls. Upon return from the sm ls the routine will return to the calling routine.

1.4.6.1 NAK'D ROUTINE ERRORS

Errors which may occur in processing nak'd io orders are:

1. Sm ls returns with an error.

2 EXTERNAL SPECIFICATIONS

2.1 OWNED DATA STRUCTURES

The following pages define the data structures owned and used by the lan subsystem:

2.2 EXTERNAL INTERFACES

2.2.1 MOD400 EXECUTIVE SOFTWARE ROUTINES

2.2.1.1 ZXREQ - Request task

```
entry:      lnj  $b5,zxreq

input:      $b4 = address of task request block
            $b5 = return address

output:     $r1 = 0 - task request was queued successfully
            $r1 > 0 - task request was not queued
            $b4 = address of task request block

modifies:   $r1,$r2,$r3,$b1,$b2,$b3

function:   Request a normal task with a supplied request block
            pointer.
```

2.2.1.2 ZHCOMM - Null address

```
function:   Will load the null address when referenced i.e. ldb
            $b5,<zhcomm  will load $b5 with the null address.
```

2.2.1.3 ZXD_PR - Dequeue and post IRB

```
entry:      lnj  $b5,zxd_pr

input:      $r2 = completion status for request
            $b5 = return address

output:     $r1 = 0 - request was dequeued and posted
            $r1 > 0 - no request on queue exist
```

2.2.2 MOD400 DATA STRUCTURES IMPLEMENTED

The following system owned data structures are referenced by the ldms:

Task Control Block (TCB)
System Control Block (SCB)

2.2.3 USER INTERFACES

2.2.3.1 LDMS INTERFACE (MSIIOR)

```
call:       lnj $b5,msiior
```

```
input:      $b1 = a(lcb)
            $r1 = function code
                0009 - output lcb
                000x - reset/halt
                000x - load/dump
                0001 - strar io
                0026 - input id
            $b1 = a(lcb) if function code = 9
            $b1 = a(lit) if function code <> 9
            $b5 = a(return)

output:     $r1 = status
                0000 - io or iold was successful
                0001 - io or iold was nak'd
                0002 - iold not performed, lcb queued because of
                flow control
                ffff - invlaid function code
            $r2 = hardward id if function code = input id on input
            $b1 = a(lit) or a(lcb)

modifies:   $r2, $r1
```

2.2.3.2 LCB FORMAT

```
cb_pri

     input:    mbz
     output:   na

cb_ncb

     input:    mbz
     output:   na

cb_rct

     input:    address of caller's rct
     output:   same as input

cb_lit

     input:    address of the lit in which this lcb will be
               queued
     output:   same as input

cb_frw

     input:    bit 0-3 - 9
               bit 4-7 - mbz (iorb major function code ?)
               bit 8-f - xx
     output:   same as input
```

cb_itp

```
    input:     address of the post processing  routine, or trb,
               on null
    output:    same as input
```

cb_ind

```
    input:     indicators
                 bit 7 - cb_itp points to a trb when set
                 bit 6 - sm lcb when set
                 all other bits mbz
    output:    same as input
```

cb_icw

```
    input:     bit 0-5 - mbz
               bit 6-9 - cpu number to interrupt
               bit a-f - level to interrupt the cpu
    output:    same as input
```

cb_fsf

```
    input:     function specific function code
                 function codes for read lcbs are:
                         0012 - ci read
                         0022 - co read
                         0042 - co expideted read
                 function codes for write lcbs are:
                         0011 - ci write
                         0021 - co write
                         0042 - co expideted write
                 function codes for event lcbs are:
                         001e - sap event
                         002e - connection event
                         004e - sm event
    output:    same as input
```

cb_cts

```
    input:     mbz
    output:    bit 0-7 - rfu and mbz
               bit 8 -   invalid function code when set
               bit 9 -   ram memory exausted when set
               bit a -   ram location non-existent when set
               bit b -   ram parity error when set
               bit c -   level 6 memory yellow when set
               bit d -   level 6 memory non-existent when set
               bit e -   level 6 bus parity error when set
               bit f -   level 6 memeory red when set
```

cb_fss

```
input:    mbz
output:   function specific status
            0001 -  sap not active
            0002 -  lack of resources
            0004 -  controller unavailable
            0008 -  sm layer instance error
            0020 -  sap already active
            0040 -  sap already deactivated
            0080 -  recieve buffer too small
            0100 -  illegal logical address
            0200 -  invalid lcb
            0400 -  write credit violations
            0800 -  read credit violations
```

cb_cbs

```
input:    mbz
output:   bit 0 -    lcb is complete when set
          bit 1 -    lcb not processed when set
          bit 2-f - rfu and mbz
```

cb_abs

```
input:    mbz
output:   actual buffer size if cb_fss = 0080, otherwise
          same as input
```

cb_lsa

```
input:    logical local address for cl operations
output:   same as input
```

cb_lra

```
input:    logical remote address for cl write operation, mbz
          for event and read operations
output:   logical remote address for cl read operations,
          otherwise same as input
```

cb_trg

```
input:    total byte range
output:   same as input
```

cb_bct

```
input:    number of buffers
output:   same as input
```

cb_ad1

```
input:    buffer #1 address
output:   same as input
```

cb_rg1

```
input:    buffer #1 range
output:   same as input
```

cb_rs1

```
input:    mbz
output:   buffer #1 residual range
```

cb_ad2

```
input:    buffer #2 address
output:   same as input
```

cb_rg2

```
input:    buffer #2 range
output:   same as input
```

cb_rs2

```
input:    mbz
output:   buffer #2 residual range
```

cb_ad3

```
input:    buffer #3 address
output:   same as input
```

cb_rg3

```
input:    buffer #3 range
output:   same as input
```

cb_rs3

```
input:    mbz
output:   buffer #3 residual range
```

cb_ad4

```
input:    buffer #4 address
output:   same as input
```

cb_rg4

```
input:    buffer #4 range
output:   same as input
```

cb_rs4

    input: mbz
    output: buffer #4 residual range

cb_ad5

    input: buffer #5 address
    output: same as input

cb_rg5

    input: buffer #5 range
    output: same as input

cb_rs5

    input: mbz
    output: buffer #5 residual range

cb_ad6

    input: buffer #6 address
    output: same as input

cb_rg6

    input: buffer #6 range
    output: same as input

cb_rs6

    input: mbz
    output: buffer #6 residual range

cb_ad7

    input: buffer #7 address
    output: same as input

cb_rg7

    input: buffer #7 range
    output: same as input

cb_rs7

    input: mbz
    output: buffer #7 residual range

cb_ad8

    input: buffer #8 address
    output: same as input

cb_rg8

input: buffer #8 range
output: same as input

cb_rs8

input: mbz
output: buffer #8 residual range

The rest of the lcb fields are function specific fields, there defintion can be found in the ldls, llc ls or the cl4 ls or the sm ls.

2.3 INITIALIZATION REQUIREMENTS

The clm process will load the ldms into system memory, and configure at least one interrupt level for the ldms to execute under. The ldms bound unit will be loaded into memory by clm via a task request. At this time the ldms will begin executing it's initialization code. The ldms lst code consists fo performing the following:

1. Posting the task request, using a call to the zxd_pr routine.

2. Since the levels the ldms interrupt processing code executes under will only be invoked as a result of an interrupt, the fixed to level bit (mt_fix) in the tcb's first indicators word (t_ind) must be set.

3. Exit the level via a lev enable instruction, leaving the p-counter at the entry point of the interrupt processing routine.

2.4 TERMINATION REQUIREMENTS

There are no termination requirements since the ldms will be active as long as the mod400 operating system is active.

2.5 ENVIRONMENT

The following items are required by the ldms for it to perform it's task:

1. Mod400 operating system.

2. Any l6 computer model except 6/10 and 6/20.

3. Lan clm.

4. A lacs attached to the l6 megabus.

5. A user of the lan subsystem to drive the lacs driver.

2.6 TIMING AND SIZE REQUIREMENTS

Currently memory usage and timing requirements are not an issue. However, the code should be be as efficient as possible (note: the interrupt processing routine executes at a low level, and care should be taken in writing the code to make it very efficient).

2.7 ASSEMBLY AND LINKING

The software will be written in Series 6 Assembly Language using a subset of the instruction set that is present on all Series 6 systems. The ldms will be linked with the lacs driver interface services module by the gcos6 mod400 linker to produce a portion of the lacs driver's bound units. The name of the module will be zqllms.

2.8 TESTING CONSIDERATIONS

Since the product is new, all functions will be tested by the developer, and software test.

2.9 DOCUMENTATION CONSIDERATIONS

The ldms source listing will include a program design language used by the developer to aid in the maintenance by future developers and also to aid in the development by the developer.

2.10 ERROR MESSAGES

The ldms will inform the calling layer server of an error by placing into $r1 the error message, then returning to the calling routine.

1. $r1 = 0001 - io or iold was repeatedly nak'd.
2. $r1 = 0002 - lcb queued because of flow control.
3. $r1 = 0000 - io or iold was successful.
4. $r1 = ffff - invalid function code.

2.10.1 INTERRUPT PROCESSING ERRORS

1. lcw = 0

3 INTERNAL SPECIFICATION

3.1 OVERVIEW

There are 2 ways to activate the ldms code, they are:

1. A layer server requests an io or iold be sent to the lacs.

2. An interrupt is generated by the lacs.

3.2 SUBCOMPONENT DESCRIPTION

3.2.1 DETERMINATION ROUTINE

The routine is invoked via a call from a ls wishing to issue an lcb. The routine requires the following input parameters:

```
$r1 = function code
      0009 - output lcb
      0001 - start i/o
      000x - load/dump
      000x - reset/halt
      0026 - input id
$b1 = a(lcb) if function code = 9
$b1 = a(lit) if function code <> 9
$b5 = return address
```

The routine supplies the following output parameters:

```
$r1 = status
      0000 - io or iold was successful
      0001 - io or iold was nak'd
      0002 - iold not performed, lcb queued because of fc
      ffff - invalid function code
$r2 = hardware id if function code = input id upon input
$b1 = a(lcb)
```

When the determination routine is invoked by a ls the routine performs the following:

1. Validates the function code, returning to the caller if the function code is no in the allowable range.

2. Saves the return address.

3. Depending on the function code, the routine calls either the issue iold routine or issue io routines.

When the determination routine is invoked by either the issue iold routine or issue io routine, the routine performs the following:

1. Retrieve the return address.

2. Jumps back to the ls.

3.2.2 ISSUE IOLD ROUTINE

The issue iold routine is called by the determination routine. The routine requires the following input parameters:

```
$r1 = function
      0009 - output lcb
      0001 - start io
      000x - load/dump
$b1 = a(lcb)
$b5 = return address
```

The routine supplies the following output parameters:

```
$r1 = status
      0000 - iold was successful
      0002 - iold not issued, lcb was queued because of fc.
      0001 - iold not issued because of success nak'd
$b1 = a(lcb)
```

The routine performs the following function:

1. Save the return address
2. Retrieve the pointer to the lit from the lcb (cb_lit)
3. Mask in the control information from the lit (li_id2) with the function code.
4. Set up the lcb length word, along with the protocol id.
5. Set the pointer to the lacs specific portion of the lcb.
6. Call flow control routine.
7. If a non zero status resulted in the call, retrieve the return address and jump back to the determination routine.
8. Clear the current iold retry count.
9. Increment the current iold count, if the current count > the maximum count call the nak'd routine. Upon return from the nak'd routine return to the caller.
10. Issue the iold.
11. If the iold was nak'd, go to #8.
12. Queue the lcb on the tail of the lit active lcb queue.
13. Return to the caller with a successful return status.

*********** states must be checked before an iold can be issued

3.2.3 ISSUE IO ROUTINE

The issue io routine is invoked through a call by the determination routine. The routine requires the following input parameters:

```
$r1 = function code
      0026 - input id
      0001 - start i/o
      000x - reset/halt
$r2 = protocol id
$b1 = a(lit)
$b5 = return address
```

The routine supplies the following output parameters:

```
$r1 = status
      0000 - io was successful
      0001 - io not issued because it was nak'd
$r2 = hardware id if $r1 = 26 on input
```

The routine performs the following function:

1. Save the return address
2. Mask the function code with the ll_id2 word in the lit, to form the control word.
3. Clear the current io retry count in the lit..
4. Increment the current io count, if the current count is greater than the maximum count, call the nak'd routine. Upon return from the nak'd routine return to the caller.
5. Issue the io.
6. If the io was nak'd go to #4.
7. Return to the caller with a successful return status.

*********** States must be checked before the io is issued

3.2.4 INTERRUPT PROCESSING ROUTINE

The interrupt processing routine is invoked via an interrupt generated when the lacs completes an lcb. The routine requires the following input parameters:

```
$iv.0 = interrupt control word (tcb word 0)
```

The routine supplies the following output parameters:

terminates it's level when complete

When the interrupt processing routine is invoked as a result of an interrupt, it performs the following.

1. Retrieve the icw from $iv.o.

2. If the icw = 0, then go to an error processing routine (tbd).

3. Retrieve the pointer to the scb, retrieve the pointer to the cd from the scb.

4. Index into the cd by the icw bits 0-3 for the pointer to the ct.

5. Index into the ct by the icw bits 4-6 for the pointer to the lt.

6. Index into the lt by the icw bits 7-9 for the pointer to the lit.

7. Retrieve the pointer to the active lcb queue from the lit.

8. Search the queue starting from the head, until a completed lcb is found, if no completed lcb is found, terminate the level through a lev instruction, make sure the p-counter is set to the start of this routine.

9. Dequeue the completed lcb, save the pointer to the next lcb on the queue, call the flow control routine.

10. Retrieve the cb_itp word.

11. If bit 7 of the cb_ind word is set, call the exec request task routine to request the task from the trb in the cb_itp word, with $b5 = return, $b4 = a(trb), and all registers saved. Restore the registers, (if an error resulted in the call what the routine will do is tbd), retrieve the pointer to the next lcb in the queue and go to #8.

12. If bit 7 of the cb_itp word is not set, lnj to the address specified in the cb_itp word, with $b5 = return address and $b1 = lcb, all other registers are saved.. Upon return restore the registers, retrieve the pointer to the next lcb and go to #8.

13. If bit 6 of the cb_itp word is set, release the memory of the lcb through an exec call (tcb). Retrieve the pointer to the next lcb in the queue and go to #8.

3.2.5 FLOW CONTROL ROUTINES

3.2.5.1 PRE LCB FLOW CONTROL ROUTINE

The pre lcb fc routine is invoked by the issue iold routine. The routine requires the following input parameters:

```
$b1 = a(lcb)
$b2 = a(rct)
$b3 = a(tt)
$b5 = return address
```

The routine requires the following output parmaeters:

```
$b1 = a(lcb)
$b2 = a(rct)
$b3 = a(tt)
$r1 = status
      0000 - ok to send lcb
      0001 - lcb queued because of flow control
```

The routine performs the following function:

1.

3.2.6 NAK'D ROUTINE

tbd

PDL

5 ISSUES

1. Controller states

2. Nak'd lcb's

3. p