SLAC-PUB-2511
May 1980
(M)

LIMS - THE LINK INTER-TASK MESSAGE SYSTEM FOR THE PEP MODCOMP COMPUTER NETWORK*

Anthony Gromme and Alex Hunter
Stanford Linear Accelerator Center
Stanford University, Stanford, California 94305

LIMS, written for the Modcomp Max IV and Max III operating systems, provides inter-task transmission of messages, that is, physical movement of information, as opposed to information occupying memory shared among tasks. Source and destination tasks may be in the same cpu or in different cpu's. LIMS was written entirely in assembler language, and consists of the following: 1) code running at task level to service the LIMS rex calls and remote I/O rex calls; 2) code running at I/O service interrupt level to handle the 4821 link hardware and to step through the link protocol; 3) a resident link support task to perform certain functions that cannot be done at interrupt level; 4) a non-resident task to fill remote cpu's; and 5) numerous changes to Max IV and Max III source modules and additions to sysgen data structures. In this description, wherever Max IV and Max III differ, the information for Max III is in square brackets.

In general, LIMS was designed for speed and efficiency, though the resulting code may be hard to modify.

No attempt was made to make LIMS compatible with other existing link software, e.g., Maxnet or LBL RTSG software. Conversion will be in one step rather than gradual.

* Work supported by the Department of Energy under contract number DE-AC03-76SF00515.

(Submitted for Publication)

To see just what LIMS does, the reader should at this point refer to the attached description of the LIMS rex call parameters.

Each complete message transmission requires the sending task to issue a "send" rex and the receiving task to issue a "get" rex. Of course, the "get" may occur before or after the "send".

The concept of "virtual circuit" is absent, and there is no "open" rex. No attempt was made to allow abstract coding of the destination.

For each "send" rex, the caller specifies, as part of the rex parameters, the destination cpu number (16 bits), the destination task name (32 bits), and the destination subqueue number (16 bits). For each "get" rex, the caller may select messages by specifying, as part of the rex parameters, the source cpu number (-1 means "any"), the source task name (-1 means "any"), and the subqueue number (-1 means "any").

The subqueue number is any arbitrary 16-bit quantity, whose meaning is established merely by "agreement" between the source and destination tasks. Using a variety of subqueue numbers requires no additional node space or buffer space.

Cpu number Ø always means intra-cpu transmission (same cpu). In all other respects except speed, intra-cpu transmission and inter-cpu transmission appear identical to the user.

No specific provision is made for transmission more than one cpu away from the source. The necessary buffering mechanism already partly exists, however, and the 16-bit cpu number would provide a natural carrier for additional routing information.

If insufficient node space or buffer space is available to service a LIMS rex, or if the parameters are bad, then the request is not satisfied or queued, and completion is posted with bits identifying the error. At present, there is no fixed limit enforced on the number of requests a single task may queue. A limitation could be based on task priority, such that for a given priority (set at sysgen time) tasks running at or below that priority would always leave a certain minimum amount of space available for use by tasks running at higher priorities.

Multiple "send's" may be queued as well as multiple "get's". Each "send" and "get" requires an array in the callers operand space, into which the completion information will be asynchronously posted. Completion information for the "get" rex always includes source identification sufficient for sending a reply.

"Get" and "send" may each be queued or not queued. If the caller selects the "test" option in a "get" rex, and at that instant no matching incoming message is queued, the "get" rex does nothing at all except post the completion information by which the caller can tell that there was no message. If the "get" rex does not specify "test", and no matching incoming message is queued, a node representing the "get" will be queued, and the "get" will not complete until a matching message arrives. Similarly, if a "send" specifies "test", and at that instant the destination task has no matching (non-test) "get" node queued, the "send" will do nothing at all except post its completion information.

"Get" and "send" may each include one free wait or include no wait. If "quick" is specified, no wait is included with the rex, and control is returned immediately to the caller. If "quick" is not specified, one wait is automatically done before control is returned to the caller, unless the request is already completed or any other resumption has occurred. Other events besides message transaction completion may resume the calling task, and the caller must in all cases test the completion information to determine why he was resumed, and execute additional wait's as needed. At the beginning of processing each rex, word Ø of the caller's completion information is set to "busy" (#ØØØ1). The caller should regard the request as completed only when the low-order bit of this word is reset to Ø. The meanings of the bits in this word are listed in the attached description of the LIMS rex parameters.

Completion is posted to the non-test sender when his buffer has been vacated; if the sender specifies no buffering, posting of completion to the sender tells him that reception is complete at the other end; if the sender specifies (allows) buffering, the sender has no automatic way of knowing that reception is complete, short of a reply programmed explicitly as a separate message.

Buffering is an option selected by the sender. For each "send" rex, the data will be buffered if and only if the sender specifically allows buffering and does not specify "test", and the destination task does not at that instant have a matching "get" already queued; otherwise, the data will be transmitted directly. Inter-cpu buffering is always done in the destination cpu.

For an unbuffered "send", data transmission is delayed until the destination task has issued a matching "get". This is true in the inter-cpu case as well as in the intra-cpu case.

In order to allow unbuffered variable-length data transmission, the inter-cpu link protocol always includes sending a fixed-length header separately from any data. For each inter-cpu message, all routing and descriptive information is transmitted in the header, which may or may not be followed by the message contents.

A more complete description of the link protocol is attached. The protocol steps are coded in assembler language in a handler edited into the resident system at sysgen time. Service interrupts are routed through the PDT directly to the handler, as with standard Modcomp handlers. Everything necessary for proceeding from each protocol step to the next (if any) is done directly at service interrupt level, including acquisition and release of buffer memory.

The message "write" protocol sequence also takes care of node purge requests from dying tasks, and remote I/O (that is, I/O to or from a device attached to a cpu other than that in which the requesting task is running).

Remote I/O always consists of two complete link message transactions, between which the link is available for any use. The first message transmits the I/O parameters, and, if the operation is a write, the data is buffered. If the operation is a read, buffer memory for the data is acquired at this time. After the actual I/O is done, the second message transmits UFT completion information, and transmits the data if the operation was a read.

As in the RTSG system, in order to get I/O done remotely, the user must assign (in the cpu where the request is originating) the logical file to the link device. Then the logical file name (specifically, the last in any file-to-file assign chain), say XYZ, is transmitted as part of the I/O parameters, and, in the cpu servicing the request, a dummy file through which the actual I/O will be done is internally (automatically) assigned to XYZ. It may be necessary for the user to globally assign XYZ in the servicing cpu.

The main I/O module (IO.IV) [M3IOS] was modified so that I/O to the link device is not queued by the standard Modcomp routines, but instead the request is diverted to the link handler.

Load modules can be fetched via remote I/O across a link into a Max III system, but not into a Max IV system. This limitation may be corrected in the future.

Separate protocol sequences are included for clearing the line upon timeout, and for filling remote cpu's; see the attached protocol description.

Link PDT's must be in the untimed PDT chain. A timeout routine, driven every 250 milliseconds by the "low clock" interrupt (level #E), scans the link PDT's as well as the activation request queue. The timeout routine decrements the timer word in each active link PDT, and initiates the timeout sequence if the timer runs out.

The message node size is 25 [21] words. A fixed-size pool of message nodes is assembled in map 0 [resident SYSBLOCK] at sysgen time. Standard Modcomp I/O nodes are never used for messages. The Modcomp ROLLER [checkpoint task] knows nothing about message nodes,

and therefore the ROLLER [checkpoint task] and LIMS are incompatible. This incompatibility will probably be corrected in the future.

Message nodes contain only pointers and routing and descriptive information, no message data. For buffering data, pages are acquired as needed from the system's free page chain [foreground dynamic memory pool]. These buffer pages remain unmapped except by the IOP, and the program addresses them by LDAM/STAM instructions. All Modcomp source modules that refer to the free page chain [foreground memory pool] were altered to allow manipulation of the chain at service interrupt level. Each buffer page contains at most one message. This inefficiency will probably be alleviated later.

A pool of 200 message nodes occupies 5K of map 0 space. Moving the pool out of map 0, and addressing the nodes by means of LDAM/STAM instructions, would require a laborious rewrite of the entire LIMS system. Another alternative is to permanently claim a hardware map (other than map 0) for exclusive use by LIMS, and address message nodes and buffers through this map. This approach would in fact speed up memory-to-memory data transfer.

The process of moving data memory-to-memory is driven sometimes by the source task and sometimes by the destination task. During each such move, the taskmaster is kept blocked continuously. This delay will probably be alleviated in the future.

If an inter-cpu buffered message is being transmitted but transmission is not yet completed when the destination task issues a matching "get" rex specifying "quick", then there is no direct way the destination task's rex can move the data from the system buffer into

the task's memory. Moving the data at service interrupt level would hold up that interrupt too long. To solve this problem, we modified the taskmaster (and all other Max IV source modules that refer to TCBRIF) to allow a software interrupt (an "asynchronous processing element"). This approach allows the data to be moved at task level with the destination task's map image loaded into hardware registers.

Attached to each TCB are five queues of message nodes: 1) unsatisfied "get" requests; 2) pending unbuffered inter-cpu "send" requests; 3) incoming messages for which the task has not yet issued a "get"; 4) messages requiring memory-to-memory data transfer to be done by software interrupt; 5) I/O requests being serviced by a remote cpu. In Max IV, eleven words starting at LOXITE in the TCB's loader extension contain the roots of these queues, plus a count of message nodes which the task has queued on link PDT's.

In certain steps of the link protocol, it is necessary at service interrupt level to locate a task, which may or may not exist at that instant. To avoid running the TCB chain [RCB list] at interrupt level, we added a separate list of task names and TCB [TWA] addresses, which can be searched at interrupt level. When a task is "born", but before it receives control, its name and TCB [TWA] address are added to this list, effectively making the task eligible to receive messages. Also at this time any nodes destined for that task are moved from the activation queue to the task's queue of incoming messages, and each such node's activation request bit is reset. When a task is "dying", its name and TCB [TWA] address are removed from the list. Also at this time message nodes related to the dying task are

purged throughout the network. Any still unprocessed incoming message nodes that still have the activation request bit set are moved to the activation request queue to cause reactivation of the task.

Functions that cannot be done at interrupt level but rather must be done by a task are done by LK, the resident link support task. Several special queues belong to LK: the activation request queue, the remote I/O request queue, and the fill request flags. Whenever LK is resumed, LK services as many requests as it can. LK scans the activation request queue, and, for each node whose activation request bit is still set, LK issues an activate rex, resets the request bit in the node, and sets a timeout counter in the node. Nodes in the activation request queue which time out will be purged from the queue.

LK has a pool of UFT's for servicing remote I/O requests. LK searches this pool for any requests whose actual I/O has completed, and, for each, sends the reply back to the requesting cpu. LK then initiates the actual I/O for as many requests from the remote I/O request queue as there are available UFT's.

If any cpu needs to be filled, LK sends the fill request flags as a message to LKN, the non-resident fill task. LKN will service as many fill requests independently and concurrently as it can.

If an event freezing a trace table has occurred, LK sends the contents of the trace table as a message to LKTCP [LKT], the non-resident trace outputting task.

As a sysgen option, a trace routine and table may be included which will trace each link service interrupt. This trace is turned on or off for each link by a bit in the PDT. OC commands to turn the trace on or off have not yet been coded.

4821 Inter-cpu Link Protocol

The link protocol design was based on the following considerations: 1) Each hardware link is half-duplex bidirectional, and is symmetrical with respect to its two ends. 2) The maximum amount of unsolicited information that can be sent across a 4821 link is six bits, namely, the five more-or-less programmable 4821 input status bits, plus the bit that causes the external service interrupt. 3) The protocol sequences should allow unbuffered variable-length data transmission. 4) There should be a "write" sequence and a "read" sequence, each of which can be initiated from either end of the link.

In the following description, what is referred to as the "sending cpu" may be at either end of the link, depending on momentary context.

Say the cpu's at each end of a link are called A and B. Then the "write" sequence, as well as the "read" sequence, looks like the following:

First approximation:

Cpu A sends cpu B a fixed-length header describing the message to follow.

The data is transmitted, either directly into the destination task's memory or into system buffer memory, or else this step is skipped.

An acknowledgement is transmitted describing the outcome.

Second approximation:

1. Cpu A sends an interrupt (called "request") to cpu B with status bits that say, "I want to send you a header."

2. B prepares to input the header, and sends an interrupt (called "grant") back to A with status bits that say the header may now be sent.

3. A sends the header.

4. B decodes the header and decides whether or not the associated data should be transmitted now, and notifies A of that decision via status bits.

5. "Grant" is sent and the data is transmitted, unless B decided to skip this step.

6. "Grant" is sent and the acknowledgement, consisting of a count of words actually received plus other completion status bits, is transmitted.

7. Another sequence begins, or else an interrupt is sent saying, "I got the acknowledgement."

The "read" and "write" data transmission sequences differ mainly as follows: in the "write" sequence, header and data are transmitted in the same direction; in the "read" sequence, header and data are transmitted in opposite directions.

Transmission of a message always begins with the "write" sequence. If (at step 4 above) there is a matching "get" queued by

the destination task in cpu B, then the data is transmitted without buffering, and the acknowledgement goes from B to A, completing the transaction. If there is no matching "get" queued, then there are two cases: If the sender requested buffering, B acquires buffer memory, the data is transmitted into the buffer, and the acknowledgement goes from B to A, just as in the preceding case. As far as the source task in cpu A is concerned, the transaction is complete at this time. If, however, the sender did not request buffering, then data transmission is delayed until a later time, but a node is created and attached to the destination TCB in cpu B, and an acknowledgement goes from B to A. The message is now represented by two nodes: one in A (source) and one in B (destination). Later, when the destination task in cpu B issues a "get" rex that matches the latter node, the "read" sequence takes place: B sends a header to A, the data is transmitted from A to B, the acknowledgement goes from B to A, and only now is the transaction complete (for the source task as well as the destination task). If the source task has disappeared in the meantime, then no data is transmitted, and the acknowledgement goes from A to B. Between the write and read sequences, the link is available for any use.

When either end of a link times out, the timeout recovery sequence is initiated. In this sequence, each end sends the other a summary description of the state it was in when the timeout occurred. If the timeout recovery sequence completes successfully, each end knows whether the interrupted (timed-out) sequence appeared complete or incomplete at the other end.

The fill sequence can be initiated by a task in the sending cpu, or by pushing halt, master clear, fill, and run on the control panel of the cpu to be filled. If the cpu to be filled is a Modcomp IV, then only the latter method may be used. If the fill sequence is initiated by a task in the sending cpu, then the sending cpu will issue a mode command to force an automatic master clear, fill, run sequence in the Modcomp II being filled. In any case, the fill operation will be overseen by LKN, the non-resident fill task. LKN requires, as input, a file containing the verbatim memory image of the system to be loaded. A separate processor, named IMAGE, has been written to convert any sysgen link-edit output (normally read by SAL) into a memory image readable by LKN. In this image, word location Ø must contain the initial execution entry address, and word location #2Ø must contain the highest address to be loaded (size - 1). Further restrictions must also be followed; see the modified version of Max IV module CS.INI [Max III module M3CLD]. For use by the startup code in the cpu being filled, LKN will patch the cpu number into word location #2E of the image.

In the first step of the fill protocol, after the "fill me" interrupt has been received from the cpu being filled, the sending cpu issues a mode command to slow the transmission rate, and sends the first stage of the bootstrap program, constructed by LKN. As part of the rex completion information, the settings of the sense switches on the control panel of the cpu being filled are returned to LKN. Subsequent protocol steps merely transfer data blocks, though in fact the first of these is the second stage of the bootstrap program,

constructed by LKN. LKN transmits the image in blocks of 4Ø96 words, repeating the "fill-write" rex for each block, and accumulates a checksum of the entire image. The bootstrap program also accumulates a checksum as it reads the blocks, and this checksum is returned to LKN after the last block as part of the rex completion information. If the checksums do not match, LKN will complain. Since the link handler does not return to a quiescent state between servicing "fill-write" rex's, a timeout failure will occur if LKN is delayed excessively.

It may happen that A and B each initiate a transmission or timeout sequence at the same instant. If the link protocol were completely symmetrical with respect to the two ends, there would be a deadlock. I chose the following asymmetry: When simultaneous requests occur, one end (called "slave") will always back down and defer its own request, and the other end (called "master") never backs down. Master and slave are indicated for each link by bits in the PDT's at each end.

The behavior of the 4821 input status bits can be inferred only from direct experimentation with the link hardware. In particular, input status bits 5, 11, and 15 are latched at the receiving end of the link. Once set to 1, they remain set until the receiving (sensing) cpu issues a transfer initiate or mode command; they cannot be reset by the sending cpu. Input status bits 12 and 13, on the other hand, are latched at the sending end, and can be reset only by the sending cpu, not by the receiving (sensing) cpu. Input status bit 13, indicating "other end is or was busy", goes to Ø only when the sending cpu issues a mode command after becoming not busy.

After issuing a mode command, the program must delay at least 8 microseconds before issuing a transfer initiate or another mode command; otherwise, the input status bits sensed at the other end may be garbled.

Simultaneous occurrence of internal and external service interrupts will often be followed by a vacuous service interrupt with neither the external nor the internal flag set (i.e., neither of input status bits 9 and 10).

We modified the 4821 hardware as follows:

If bit 15 (previously unused) in the transfer initiate command is set to 1 and bit 10 is reset to 0, then the other end's input status bit 15 (usually indicating "fill me") will not be set; bit 15 in the transfer initiate command has no effect if bit 10 is a 1. This modification allows sending the "grant" and initiating input simultaneously, with just one command.

If bit 13 (previously unused) in the mode command is set to 1, then the mode command will not reset any of the input status bits that that end senses. If bit 13 in the mode command is reset to 0, then the mode command behaves as in the unmodified controller, namely it will reset that end's input status bits 1, 2, 4, 5, 6, 11, and 15, as will a transfer initiate. Transfer initiate, mode, and no-op commands have no effect whatever if issued when the controller is in the busy state.

The link protocol assigns the following meanings to the 4821 input status bits:

bits						meaning
5	9	11	12	13	15	
1	1	Ø	Ø	Ø	Ø	request to send header
x	1	1	Ø	1	Ø	grant
x	1	Ø	Ø	1	Ø	grant with exception
Ø	1	Ø	Ø	Ø	Ø	ack-ack if no request follows; error flag if in place of grant; ready flag if immediately after completion of fill
x	1	x	Ø	1	1	"fill me"
x	1	x	1	Ø	x	"I timed out"
x	1	1	1	1	Ø	grant during timeout recovery sequence

The link protocol never uses input status bits Ø, 6, or 14.

LIMS Rex Parameters

"Get message" rex

(rex code #60; service entry M$LKRD)

R2 (if R8 bit 1 is set to 1) points to an array containing a list of address/wordcount pairs describing the scatter-read input buffer segments as follows:

word 0: number of segments (1 <= number of segments <= 4)

word 1: address of 1st segment of buffer

word 2: wordcount of 1st segment (1 <= wordcount <= 8192)

[Max III: 1 <= wordcount <= 16384]

etc.

R3 points to an array containing message selection information as follows:

word 0: source cpu number (0 means intra-cpu; -1 means "any")

words 1-2: source task name (radix 40) (binary -1 means "any")

word 3: subqueue number (-1 means "any")

R4 points to an array to receive completion information as follows:

word Ø: completion/error bits (listed below)

word 1: actual number of data words transmitted

word 2: (reserved)

word 3: actual source cpu number

words 4-5: actual source task name (radix 4Ø) if a message

word 6: actual subqueue number was received

R8 contains "get" rex code and option bits as follows:

bit Ø: 1 means "quick" (no wait);

Ø means give me one automatic wait

bit 1: 1 means R2 points to buffer segment list;

Ø means R14-R15 contain unsegmented buffer address and wordcount

bit 2: 1 means "test" (treat as no-op if no matching incoming message is already queued);

Ø means queue this "get" request if no matching incoming message is already queued

bits 3-8: reserved

bits 9-15: #6Ø (rex code)

R14 (if R8 bit 1 is reset to Ø) contains address of buffer

R15 (if R8 bit 1 is reset to Ø) contains maximum buffer size in words (1 <= wordcount <= 8192) [Max III: 1 <= wordcount <= 16384]

Notes for the "get" rex:

Wordcount (segment or total) must never be zero. The actual number of words transmitted into the callers buffer will never exceed the specified wordcount. Each buffer segment will be filled before the next segment is begun. Buffer segments must not overlap. [For Max III, the user must reserve two additional words for system use at the end of each chained buffer segment except the last; only privileged tasks may specify more than one buffer segment.] All parameters and buffers must be in the callers operand map [unprotected memory]. Arrays pointed to by R2 and R3 (segment list and selection information) are available for any use as soon as control is returned from the rex.

At the beginning of servicing the rex, word Ø of the completion information is unconditionally set to "busy" (#ØØØ1). The user should regard the request as completed when and only when the low-order bit of this word is reset to Ø.

The non-quick option merely implies that the equivalent of one wait rex will be included with the "get" rex (saving some overhead). Since resumption can be caused by other events than message arrival, the user must always explicitly test the completion information and issue additional wait rex's as needed. In cases where completion is posted but no message was received, the input data buffer may have been altered nonetheless.

"Send message" rex

(rex code #61; service entry M$LKWR)

R2 (if R8 bit 1 is set to 1) points to an array containing a list of address/wordcount pairs describing the gather-write output message segments as follows:

word Ø: number of segments (1 <= number of segments <= 4)

word 1: address of 1st segment of message

word 2: wordcount of 1st segment (1 <= wordcount <= 8192)

[Max III: 1 <= wordcount <= 16384]

etc.

R3 points to an array containing routing information as follows:

word Ø: destination cpu number (Ø means intra-cpu)

words 1-2: destination task name (radix 4Ø)

word 3: destination subqueue number

R4 points to an array to receive completion information as follows:

word Ø: completion/error bits (listed below)

word 1: actual number of words received at destination

R8 contains "send" rex code and option bits as follows:

bit Ø: 1 means "quick" (no wait);

Ø means give me one automatic wait

bit 1: 1 means R2 points to message segment list;
Ø means R14-R15 contain unsegmented message address and wordcount

bit 2: 1 means "test" (treat as no-op (except for possibly activating the destination task) if no matching "get" is already queued by the destination task);
Ø means queue this "send" request or buffer the data if no matching "get" is already queued by the destination task

bit 3: 1 means activate the destination task if needed;
Ø means let this "send" fail if the destination task is not active

bit 4: 1 means put this message ahead of all other messages already waiting in the destination task's queue;
Ø means let this message follow all other messages already waiting in the destination task's queue

bit 5: 1 means buffer the data if needed (ignored if bit 2 is set);
Ø means delay data transmission until the destination task is ready to receive the data

bits 6-8: reserved

bits 9-15: #61 (rex code)

R14 (if R8 bit 1 is reset to Ø) points to message

R15 (if R8 bit 1 is reset to Ø) contains message wordcount (see below)

Notes for the "send" rex:

Wordcount (segment or total) must never be zero. Segment wordcount must not exceed 8192 [16384]. Total wordcount must not exceed 32768 words. If the message is unsegmented, then its length must not exceed 8192 [16384] words. If the message may be buffered, its total length must not exceed 2036 [2018] words. Message segments must not overlap. The message and all parameters must be in the callers operand map. Arrays pointed to by R2 and R3 (segment list and routing information) are available for any use as soon as control is returned from the rex.

If activation is requested, and the destination task is already active, the destination task's secondary activate bit (ACR in TCBOST) will be set. Furthermore, if the destination task exits before accepting the incoming message, the node representing the message will be moved to the resident link support task's activate queue, and the destination task will thus be reactivated.

At the beginning of servicing the rex, word 0 of the completion information is unconditionally set to "busy" (#0001). The user should regard the request as completed when and only when the low-order bit of this word is reset to 0.

The non-quick option merely implies that the equivalent of one wait rex will be included with the "send" rex (saving some overhead). Since resumption can be caused by other events than "send" completion, the user must always explicitly test the completion information and issue additional wait rex's as needed.

"Initial fill" rex

(rex code #62; service entry M$LKFF)

R2 (if R8 bit 1 is set to 1) points to an array containing a list of address/wordcount pairs describing segments of the bootstrap program, just as for the "send" rex

R3 contains the number of the cpu to be filled (not an address in memory)

R4 points to an array to receive completion information as follows:

word Ø: completion/error bits (listed below)

word 1: sense switches from cpu being filled

R8 contains "initial fill" rex code and option bits as follows:

bit Ø: 1 means "quick" (no wait);
Ø means give me one automatic wait

bit 1: 1 means R2 points to segment list;
Ø means R14-R15 have address and wordcount of bootstrap program

bits 2-8: reserved

bits 9-15: #62 (rex code)

R14 (if R8 bit 1 is reset to Ø) points to bootstrap program

R15 (if R8 bit 1 is reset to Ø) contains number of words in bootstrap program

"Fill-write" rex

(rex code #63; service entry M$LKFW)

R2 same as for "send" rex

R3 contains the number of the cpu being filled (not an address in memory)

R4 points to an array to receive completion information as follows:

word Ø: completion/error bits (listed below)

word 1: overall checksum, if last block

R8 contains "fill-write" rex code and option bits as follows:

bit Ø: 1 means "quick" (no wait);
Ø means give me one automatic wait

bit 1: 1 means R2 points to segment list;
Ø means R14-R15 contain the address and wordcount of the block to be sent

bit 2: 1 means this is the last block;
Ø means this is not the last block

bits 3-8: reserved

bits 9-15: #63 (rex code)

R14 same as for "send" rex

R15 same as for "send" rex

Bits returned in word Ø of the completion information

bit	name	meaning
Ø	AENO	no matching node was found
1	AENT	destination task was not found
2	..	unused
3	AEDE	some kind of transmission error
4	AEWE	"get" wordcount was too short; some data was lost
5	AEPE	bad parameters
6	AEME	memory protection violation
7	AEOL	this link is offline
8	AENV	insufficient nodes are available
9	AEBV	insufficient buffer memory is available
1Ø	AETO	timeout failure
11	..	unused
12	AETI	transfer initiate was done for data input (this is not an error)
13	..	unused
14	..	unused
15	AEBY	busy

"Set R:IO" rex

(rex code #72; service entry M$SRIO)

R8 contains "set R:IO" rex code and option bit as follows:

bit Ø: 1 means set the R:IO bit to 1 in word TCBOST in the caller's TCB;

Ø means reset the R:IO bit to Ø

bits 1-8: unused

bits 9-15: #72 (rex code)

Note for the "set R:IO" rex:

When the R:IO bit in the TCB is set to 1, then any I/O completion will cause the task to be "resumed" (independently of clearing the I/O hold condition). By default, R:IO = Ø, and non-link I/O completion does not cause the task to be "resumed".

"Specify activation file name" rex

(rex code #64; service entry M$LKAN)

R8 contains rex code as follows:

bits Ø-8: unused

bits 9-15: #64 (rex code)

R12 contains logical file name (radix 4Ø) for use by LIMS when activating the named task

R14-R15 contain task name (radix 4Ø)

Note on "specify activation file name" rex:

This rex merely adds the given task name and logical file name to a list. Whenever the LK task activates a task, LK searches this list. If the name of the task to be activated is present, the given logical file name is used; otherwise, a fixed default logical file name is used.