BACKGROUND

This discussion is a summary of the main points set forth in a lecture on operating systems delivered October 23rd and 24th, 1973 in Santa Barbara.

The illustrations are copied from the same masters used to generate the transparencies employed in the lecture.

The content of the lecture was intended for the individual not familiar with operating system principles.

The chosen approach is an attempt to be qualitative rather than quantitative. I am more concerned with principle than with detail. A simplified illustration which illuminates an ideal will be chosen over a complex explanation which is more accurate. Hence this document will not be useful as an operation manual.

When faced with the problem of how best to explain a complex entity, I can identify two basic approaches.

1. Start with some detail of the entity, and working from there, indicate how other parts relate to that detail, and then how those parts relate to still other parts, and so forth, until the interrelationships of all the parts within the entity have been clearly demonstrated.

   An example might be to imagine explaining a sewing machine to one unfamiliar with its workings or its use. The explainer might carefully point out how all the gears, levers, and cams interract with each other, to produce the desired sequence of motions.

   This would explain how the machine worked, but without knowing why the device was built in the first place, the listener could never deduce what real purpose the machine is meant to serve.

2. Start with the global problem, for example, the concept of stitch-making, and how to do it faster.

   From here the next step is to determine, in the broadest terms, the essential characteristics which must be exhibited by any solution, for example: the entire lower thread supply must pass through a loop of the upper thread supply (formed on the lower side) with each stitch.

   Now, the particular implementation of the essential characteristics (the complex entity being investigated) may be examined, a detail at a time, in the light of how successfully that detail contributes to the achievement of some characteristic which is essential to the solution of the original problem.

The first approach is addressed to the solution, while the second is concerned with the problem. It is my belief that an investigation of the problem is necessary in order to make meaningful an examination of the solution. Hence, the second approach is the one I will use in discussing the MCP.

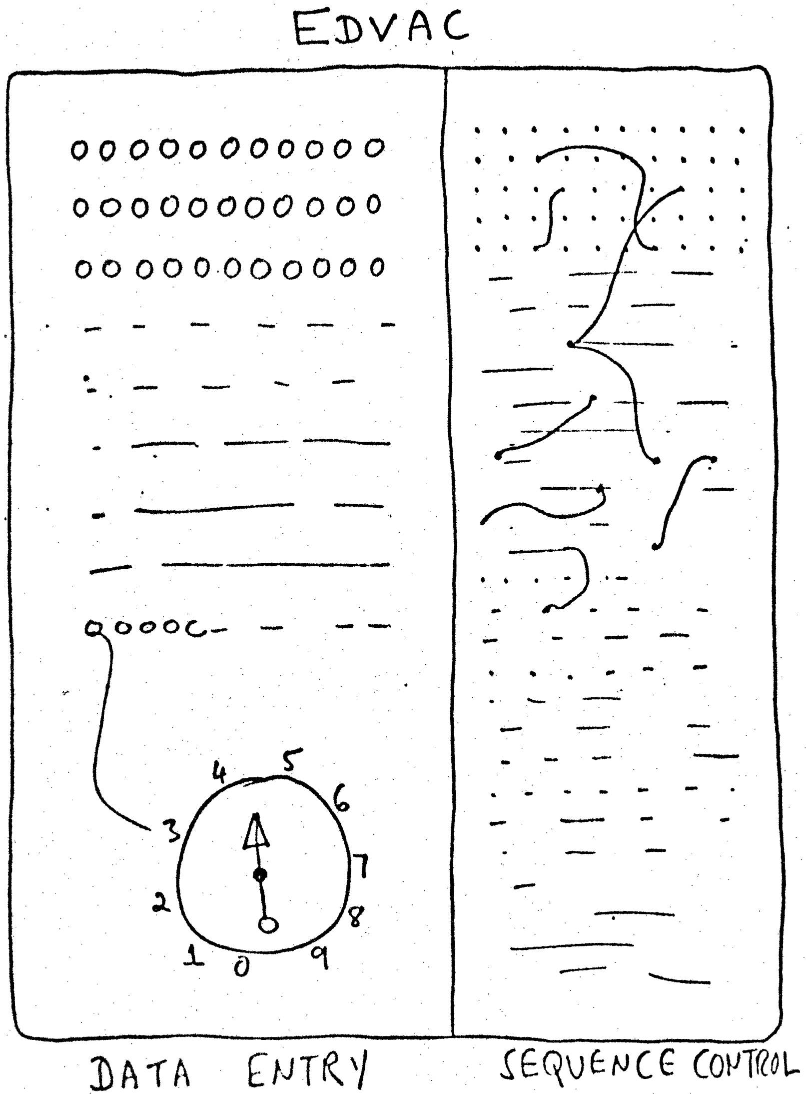
The first step is to ask why MCP's exist at all. What is the motivation behind them?

Second, we will try to see how the B1700 MCP is a current representative of a long development with its roots in the earliest history of computers.

By the turn of the century, the Burroughs adding machine had appeared. The essence of this invention, for our discussion here, is that the fundamental characteristics of some example of human behavior (adding numbers) had been identified and mapped onto a mechanical entity. The startling result was that when an adding machine added together abstract numbers, it came up with the same answers as a real person adding real numbers, only it did it much faster.

Generalizing from this example, we may state this principle:

> If an example of complex behavior can be described in terms of some simple, essential characteristics -- then that behavior can be mechanically mimicked.



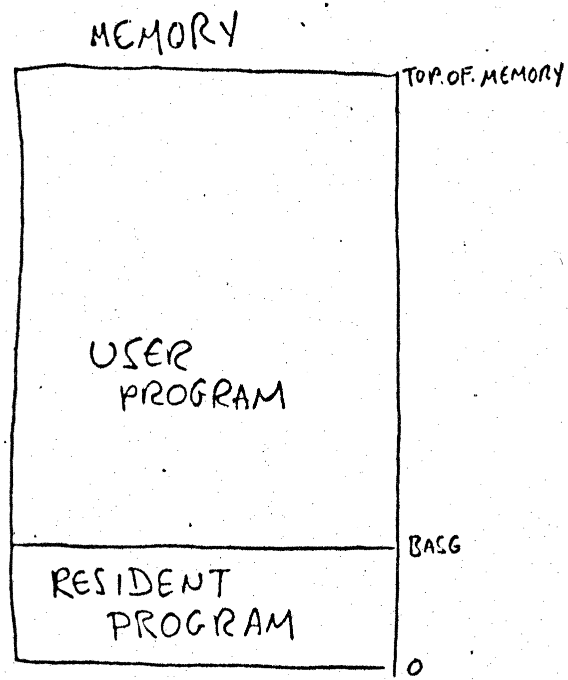
Moving up to the 1940's we find another instance of this principle being put to use.

A machine called "EDVAC" sports a bank of rotary switches for data input, but also a mammonth plug board which allows a person to abstractly represent the sequence of steps he would take in performing a computation using the numbers entered into the switches. This gave rise to the concept of a "program."

Setting up a program involved wiring up the plug board, which was slow, difficult, and error-prone.

It was soon discovered that if some of the rotary switches were set aside, and were used to represent the execution sequence, then the plug board could be wired permanently to interpret these sequencing instructions, and program set up became much simpler and more orderly. A person could now enter his whole "program," data and instructions, by adjusting the settings of the rotary switch bank.

The next example of behavior to become automated was the setting of the switches themselves. It was found that a disproportionate amount of time was being spent (in setting up the program, as against the amount of time it took to run it. The rotary switches were replaced with a memory of some sort, which could be quickly filled from an input device by pushing a "LOAD" button on the console.



RULES:

1. USER PROGRAM KNOWS HE WILL START AT "BASE"
2. USER PROGRAM MUST "BRANCH TO ZERO" WHEN DONE.
3. RESIDENT PROGRAM: (AT LOCATION ZERO)

READ A CARD;
FILL MEMORY, STARTING AT "BASE" FROM STATED DEVICE;
BRANCH TO BASE;

Time passes, and hardware speeds up by several orders of magnitude. Also new high-speed input devices start appearing. Operator intervention at the beginning of each program, plus the restriction of always having to load from the same device starts to hurt. It would be very convenient to be able to switch automatically to the next program, and to select the load source as part of this process.

So somebody said: "Why not write a little program to do this, which would always be resident." And so the first Operating System was born.

An important side effect of this event is the rigid restriction which is placed on the concept "program." It is this imposition of rules which makes a program able to be dealt with mechanically. We will see this process recurring repeatedly throughout the history of OP Systems.

OPERATING SYSTEM:

A PROGRAM WHICH MANAGES

1. THE DEMANDS
2. THE RESOURCES

OF A COMPUTER SYSTEM.

We can now develop a preliminary, working condition of what an Operating System is.

As fast efficient storage devices became more and more common, huge data motion became the rule rather than the exception. It was soon realized that most programs repeat basically the same code for all the "housekeeping" associated with input/output processing. So the same realization occurs. If pools of data, and their handling, can be rigidly restricted so as to become simple enough to be "explainable" to a computer system -- then this burden could be moved over to the system (i.e. the OP System) with a tremendous jump in efficiency.

Thus we have concepts arising such as: "FILE", "READ" and "WRITE."

Time passes, and as processor speeds increase more and more, we find that the processor spedns an ever increasing amount of its time waiting for some I/O operation to complete. It is pointed out that if there were another program able to run, then it could use the processor while the first program is waiting for its I/O. Two programs could also keep more of the system resources busy. So Multiprogramming is born, and becomes a duty of the OP System.

OPERATING SYSTEM

1. A PROGRAM

   WHICH MANAGES:

   A. RESOURCES

   B. DEMANDS

   OF A COMPUTER SYSTEM.

2. PROVIDES A FAMILY OF COMMONLY NEEDED FUNCTIONS AND SERVICES.

As more and more duties passed from the program to the OP System, it became a receptacle of commonly invoked functions which could be called on by any program. So we can now update on working definition.

OPERATING SYSTEM

1. A PROGRAM

   WHICH MANAGES:

   A. RESOURCES

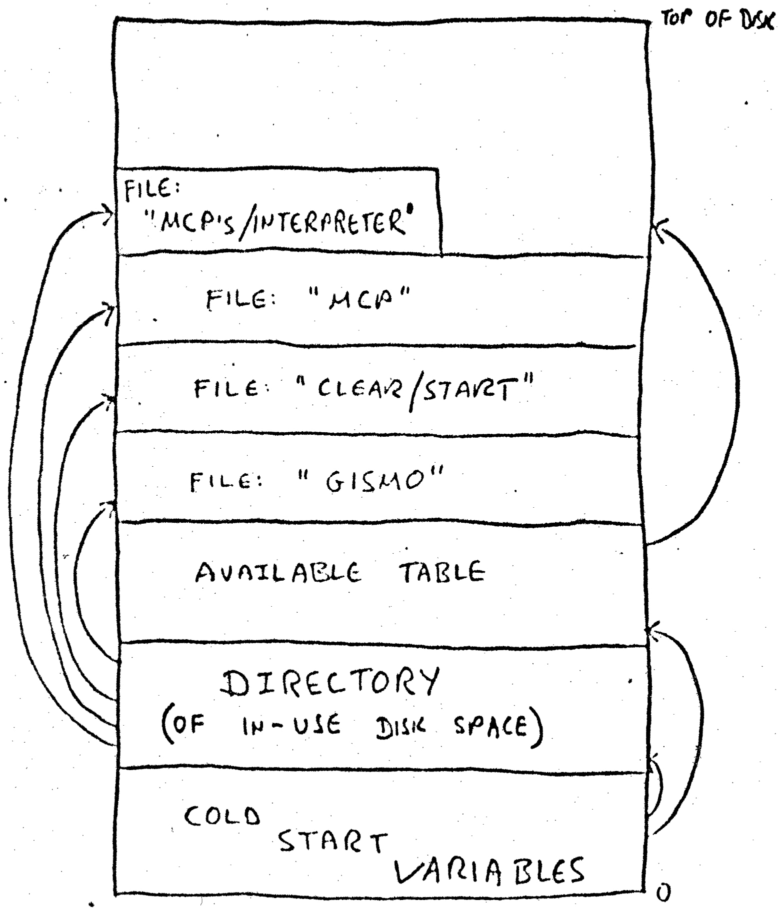
   B. DEMANDS

   OF A COMPUTER SYSTEM.

2. PROVIDES A FAMILY OF COMMONLY NEEDED FUNCTIONS AND SERVICES.

The B1700 MCP fulfills this definition, so let us take a close look at it and find out how.

B1700 SYSTEM DISK



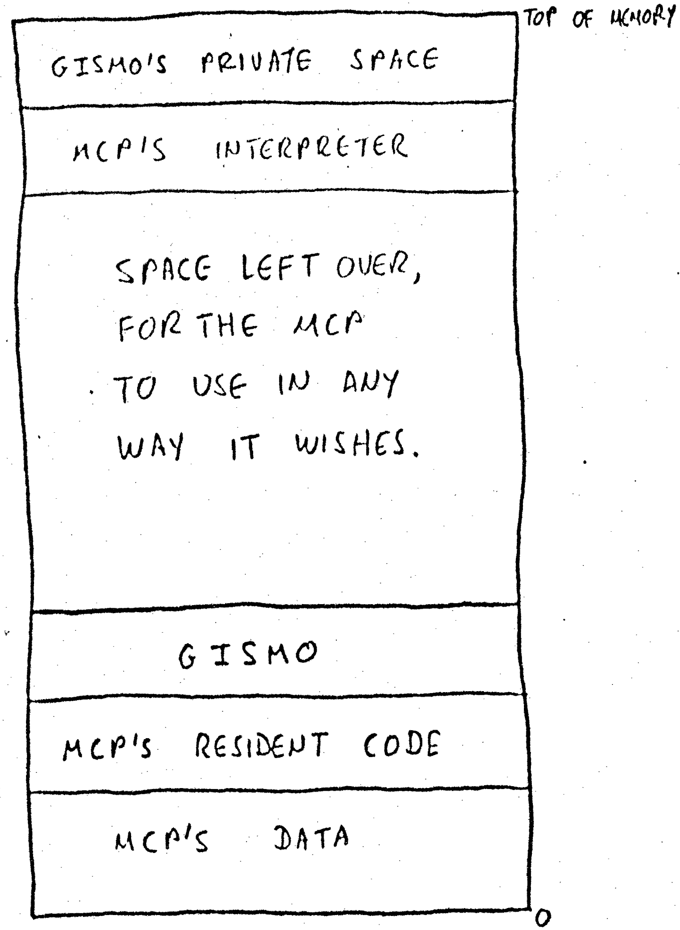
WHAT COLD START DOES.

If the MCP is a Program, how does it get started?

COLD START is a stand-alone (does not run under the MCP) program which "sets-up" the disk for MCP operation by:

1. Initializing a bunch of system values and pointers.
2. Setting up a couple of tables which describe how disk is laid out.
3. Loading several files to the disk (usually from tape).

B1700 MAIN MEMORY



WHAT CLEAR/START DOES.

After Cold Start, another stand alone program "CLEAR START" is run which sets up memory, and turns control over to the MCP.

OPERATING SYSTEM

1. A PROGRAM

   WHICH MANAGES:

   A. RESOURCES

   B. DEMANDS

   OF A COMPUTER SYSTEM.

2. PROVIDES A FAMILY OF COMMONLY NEEDED FUNCTIONS AND SERVICES.

Referring back to our definition, let us now discuss which RESOURCES one available to the B1700 MCP, and see how it manages them.

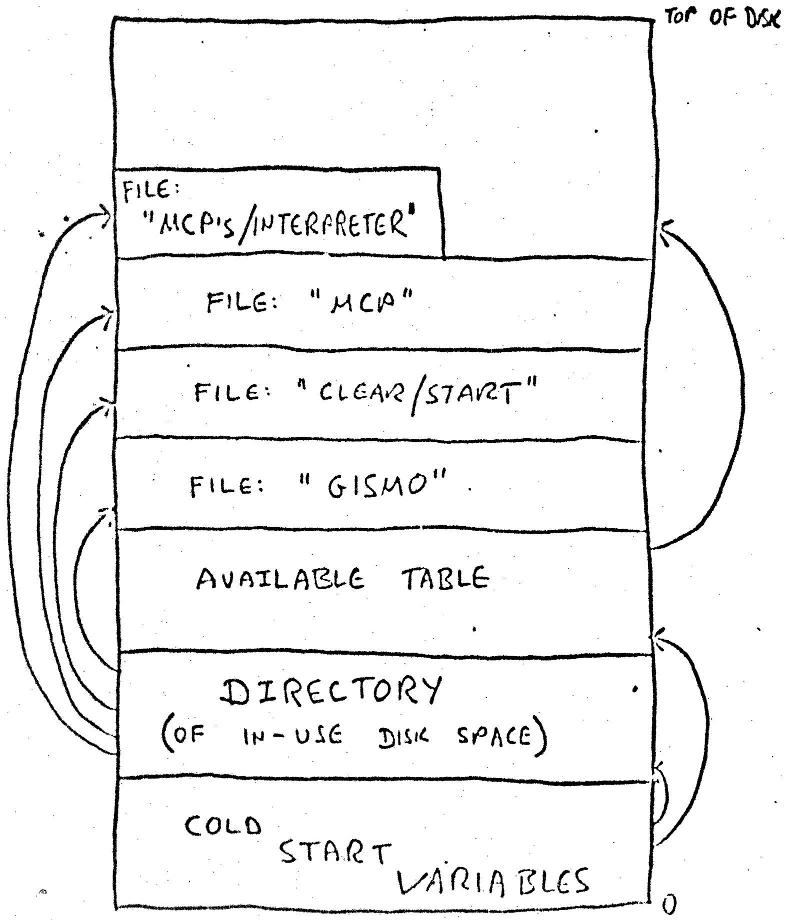
RESOURCES TO BE MANAGED

1. DISK
2. MAIN MEMORY
3. M - MEMORY
4. PERIPHERALS
5. PROCESSOR
6. MCP (AS A RESOURCE).

VIS.

Each of these will be discussed.

B1700 SYSTEM DISK

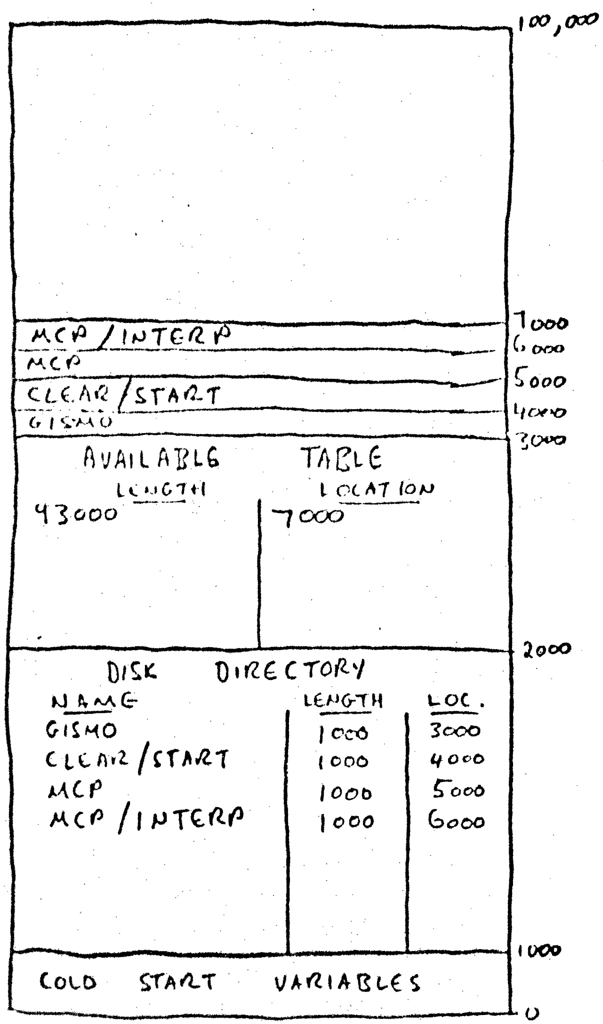


WHAT COLD START DOES.

RESOURCE 1 - DISK

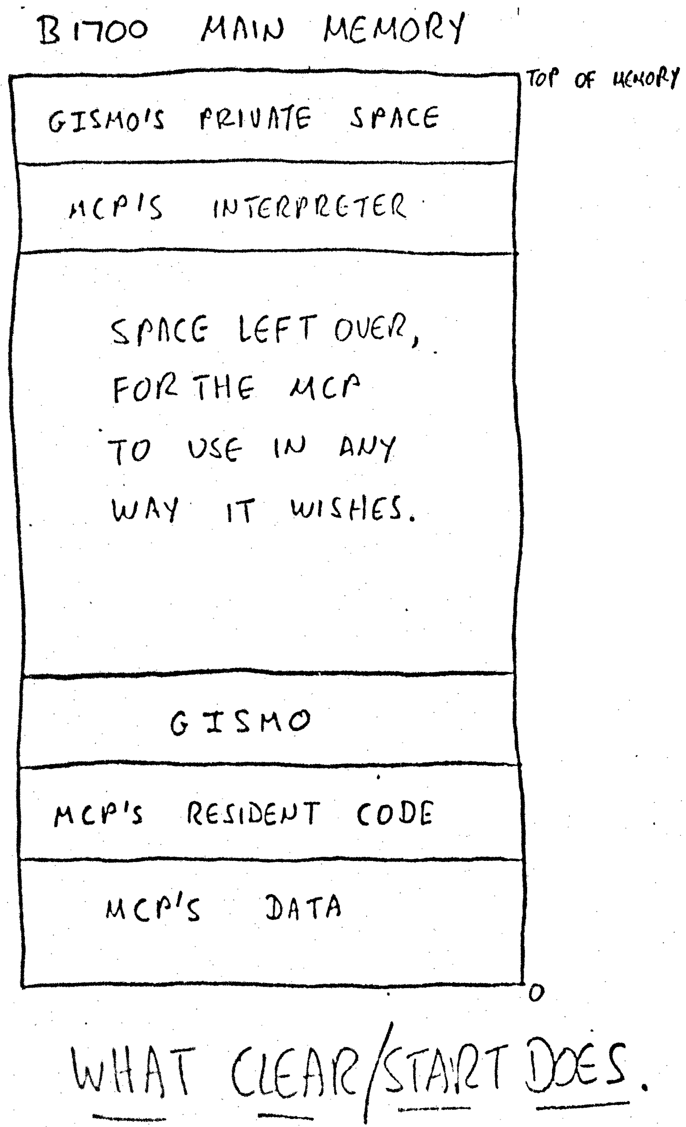
Here's how we said the disk looked right after cold-start.

B1700 DISK, RIGHT AFTER COLD-START.



Here's the same thing, but with a closer look at the Directory and the available table.

The available table starts out as one (1) entry describing a single area of available disk. Each time a file is deleted which is not bounded by available space, a new available entry will be created. This would happen for example if "CLEAR/START" were deleted.



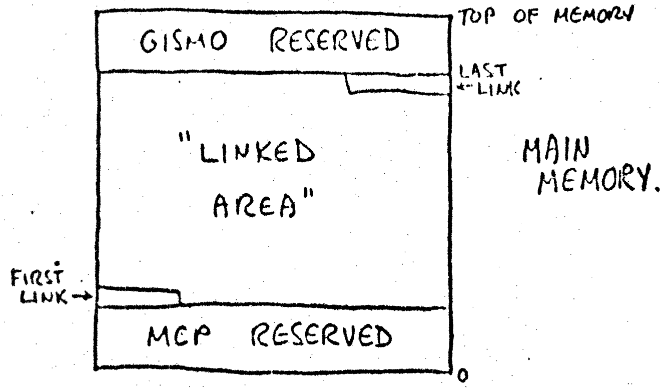
RESOURCE 2 ----- MEMORY

Clear-Start brings in from disk, or creates, everything which must be resident.

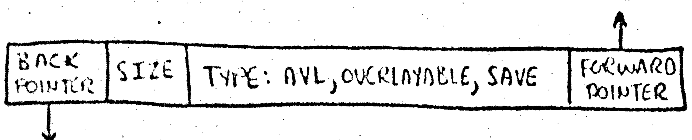
All phases of MCP operation require blocks of memory, of all different sizes, usually temporarily.

So the space left over is handled by demand.

MEMORY ALLOCATION

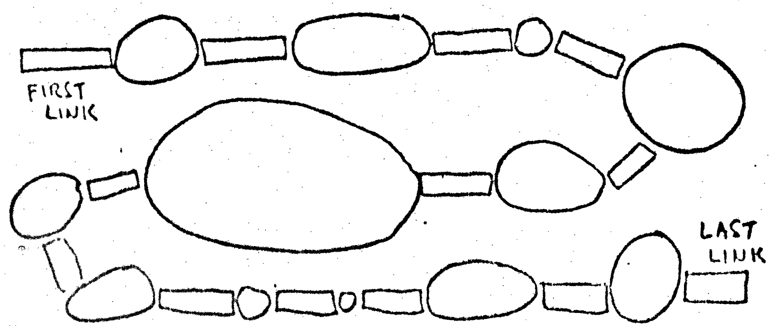


This area is referred to as the linked area.



MEMORY LINK

MEMORY IS A STRING OF BLOCKS, OF ANY SIZE, EACH PRECDED BY A DESCRIPTIVE "LINK".



MEMORY TYPES

1. AVAILABLE.

THE REQUESTOR OF THIS BLOCK HAS RELINQUISHED ALL INTEREST IN THIS BLOCK, AND IT IS FREE TO BE REUSED.

VIS.

2. OVERLAYABLE.

THE REQUESTOR OF THIS BLOCK STILL HAS AN INTEREST IN IT, BUT IT IS NOT CRITICALLY IMPORTANT, AND IF THE SPACE IS REASSIGNED, THEN THE REQUESTOR WILL GET MORE SPACE AND REPLENISH THE INFORMATION WHEN HE NEEDS IT.

3. SAVE.

THIS SPACE MAY NOT BE REASSIGNED UNDER ANY CIRCUMSTANCES, UNTIL IT IS MADE EXPLICITLY "AVAILABLE".

MEMORY ALLOCATION

SEARCH 1: LOOK FOR AVL SPACE.

IF THAT FAILS THEN ..

SEARCH 2: STARTING FROM "LEFT-OFF PTR", LOOK FOR COMBINATION OF AVL AND OVERLAYABLE.

SIZE	100	100	100	1000	50	50	40	100	100	100	100
TYPE	AVL	SAVE	AVL	OVLY	AVL	OVLY	OVLY	SAVE	OVLY	AVL	SAVE

↑ "LEFT-OFF PTR" (between OVLY 1000 and AVL 50)

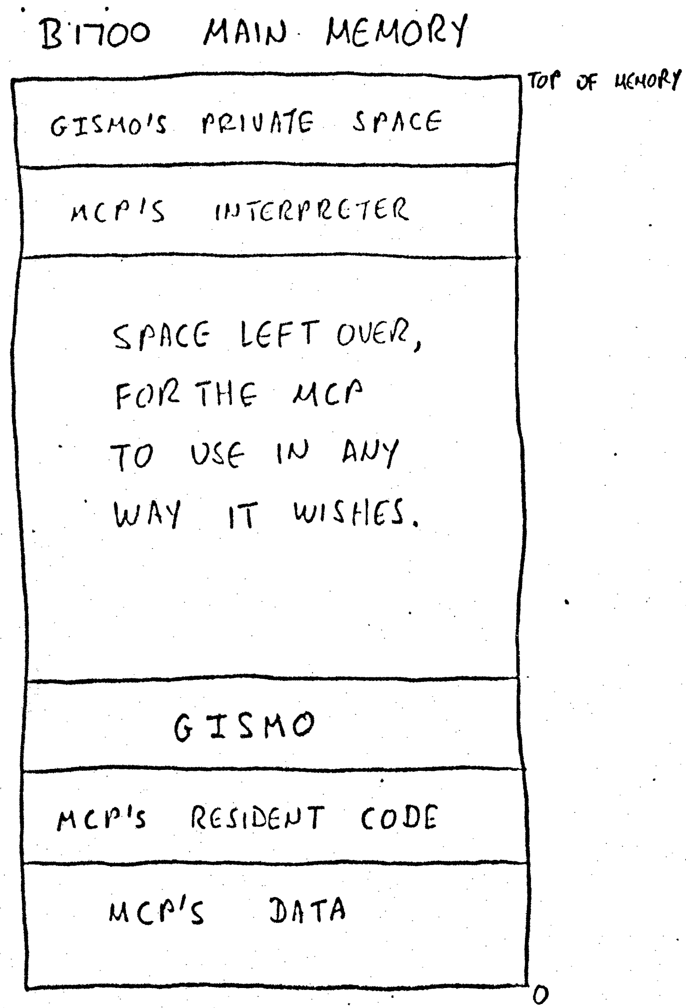
If Search 1 fails, and a piece of memory has to be overlayed in order to grant a memory request, then the "LEFT OFF POINTER" will be adjusted to point just beyond that piece of memory.

Thus, the next time this happens, we will start looking from where we "left off" last time, which ensures that no part of the linked area will be allocated more frequently than any other.

Let the picture represent a memory where the vertical lines are the links.

If a request is made for 150 bits:

1. Search 1 fails.
2. Starting at "left off" we could allocate [illegible] bits by combining "AVL" and "OVLY" space.
3. But then we encounter a "SAVE" space, so we must pass over it, and start from scratch again.
4. By combining the next two spaces, we find we can allocate the needed 150 bits and have 50 left over.
5. The LEFT-OFF POINTER will be adjusted to point between the 150 bit area just allocated and the now 50 bit "AVL" area which was left.
6. Note "LEFT OFF" automatically returns to the beginning when it drops off the end.

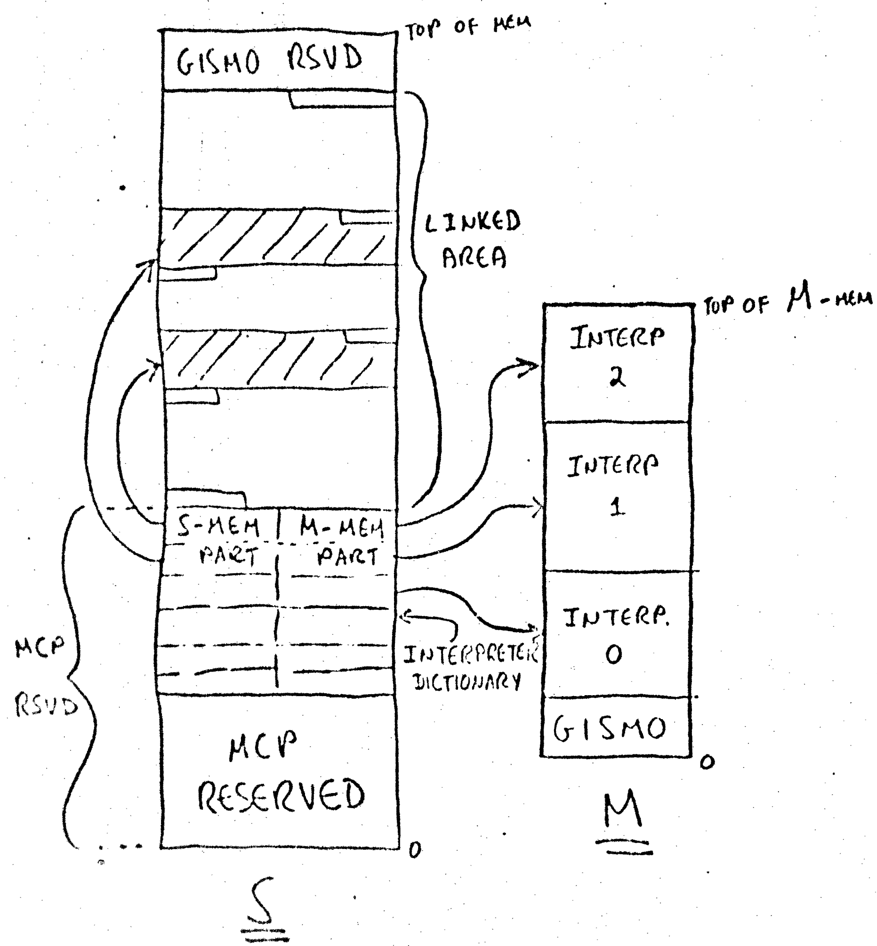


WHAT CLEAR/START DOES.

RESOURCE 3 ----- MEMORY

This picture is over-simplified. Another field which resides in MCP reserved space in Low Memory is a table called the "Interpreter Dictionary."

M-MEMORY ALLOCATION



Duties of the interpreter dictionary:

1. Keep track of which interpreters are active in the system.
2. Keep track of how M-Memory has been allocated (if the system has any).

Interpreters are written so that the most used part is at the beginning and the least used part at the end.

Thus, if all the active interpreters can't fit entirely into M-Memory, but it is possible to get all the "beginnings" in, and leave all the "ends" in the S-Memory then, since execution will be predominantly out of M-Memory, the interpreters should run at about the same speed as they would if they were entirely in M-Memory.

MANAGEMENT OF PERIPHERALS.

TYPE	WHO	P-C-U	LABELLED?	LOC	READY?
CARD RDR		0-0-0			
TAPE		0-1-0			
TAPE		0-1-1			
TAPE		0-1-2			
TAPE		0-1-3			

IOAT

RESOURCE 4 ----- PERIPHERALS

Clear start sends messages to all the possible PORT-CHANNEL-UNIT permutations and those that respond indicate their device type.

This allows a table to be built with an entry for every discovered unit. (I/O Assignment Table).

Disk does not have an entry. It is assumed to be present, and available to any number of users at the same time.

Input units may be "labelled" allowing files to be requested "by name."

Some of the INFO in the IOAT states:

What the device is.
Who is using it.
Whether it is labelled,
and where the label is.
Whether the device is ready.

PROCESSOR ALLOCATION

BY PRIORITY

HIGH

1. SOFT I/O
2. MCP
3. RUNNING PROGRAMS
   (BY PRIORITY)

LOW

RESOURCE 5 ----- PROCESSOR

Simple priority scheme.

Any higher level function may demand the processor from any lower level function.

RESOURCES TO BE MANAGED

1. DISK
2. MAIN MEMORY
3. M-MEMORY
4. PERIPHERALS
5. PROCESSOR
6. MCP (AS A RESOURCE).

[illegible]

We have now touched briefly on all of the resources.

7 Except #6, "THE MCP AS A RESOURCE" which is deferred until later.

DEMAND MANAGEMENT

1. I/O OPERATIONS.
2. PERIPHERAL MAINTENANCE.
3. CONTROL DEVICES.
4. PROGRAM WAITING INITIATION (BOJ).
5. RUNNING PROGRAMS.

8

The second major duty of an operating system is demand management.

HOW THE MCP DOES AN I/O

1. THE MCP (VIA GISMO) SUBMITS A REQUEST TO THE I/O MACHINERY FOR AN I/O OPERATION TO BE PERFORMED.

2. THE FORM WHICH THE REQUEST TAKES IS CALLED AN <u>I/O DESCRIPTOR</u> WHICH IS A FIELD CONTAINING ALL THE INFORMATION WHICH THE I/O MACHINERY WILL NEED IN ORDER TO PERFORM THE REQUESTED OPERATION.

3. FOR THE MOST PART, THE I/O MACHINERY WILL BE ABLE TO GO ABOUT ITS BUSINESS INDEPENDENTLY OF THE PROCESSOR. BUT IF IT NEEDS ATTENTION (AS, FOR EXAMPLE, IN MOVING INFORMATION INTO OR OUT OF MAIN MEMORY), THEN IT MAY SET A FLAG IN THE PROCESSOR WHICH WILL CAUSE "SOFT/IO" (A MICRO-PROGRAM) TO COME TO ITS AID.

4. WHEN THE OPERATION IS COMPLETE, THE IO/MACHINE WILL:
   1. STORE A MESSAGE DESCRIBING THE RESULTS OF THE OPERATION IN THE IO DESCRIPTOR
   2. PLACE THE ADDRESS OF THE I/O DESCRIPTOR AT LOCATION ZERO IN MAIN MEMORY
   3. TURN ON THE INTERRUPT FLAG IN THE PROCESSOR.

I/O Operations as a Demand.

The MCP starts an I/O, and then goes about its business.

When the I/O "comes complete," it demands that its completion be acknowledged.

A great amount of MCP strategy is centered around this seemingly simple event.

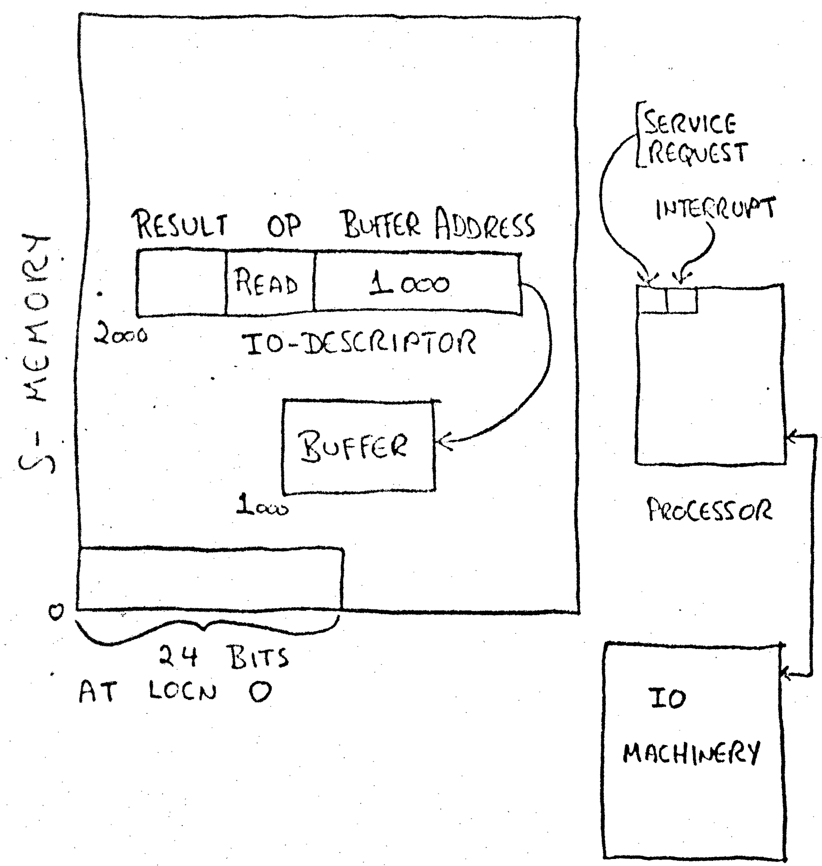
5. GISMO WILL DETECT THAT THE INTERRUPT FLAG HAS BEEN SET, AND WILL FETCH THE I/O DESCRIPTOR ADDRESS FROM LOCATION ZERO SAVING IT CAREFULLY IN THE "INTERRUPT STACK".

6. WHEN THE MCP REACHES A POINT IN ITS EXECUTION WHERE IT IS CONVENIENT TO DO SO, IT WILL INQUIRE OF GISMO IF THERE ARE ANY INTERRUPTS STACKED UP, AND UPON FINDING ONE, WILL TAKE THE ACTIONS REQUIRED BY THE I/O OPERATION HAVING COMPLETED.

7. FOR EXAMPLE, IF THE I/O WAS FOR A CARD BEING READ BY A PROGRAM, THE PROGRAM WOULD HAVE BEEN MARKED "WAITING" WHILE THE I/O WAS IN PROCESS, BUT NOW, UPON THE I/O'S COMPLETION, WOULD BE MARKED "READY TO RUN."

YUP.

MCP I/O HANDLING



These are the basic components involved in the I/O process.

MCP MAIN LOOP

```
DO FOREVER
  IF ANY INTERRUPTS
  THEN CALL IO.COMPLETE(X)
   |
   |
   OTHER STUFF
   |
   |
END
```

We now have enough to establish the most general structure of the MCP. As we proceed from here we will be filling in more and more of the details of this structure.

MANAGEMENT OF PERIPHERALS.

TYPE	WHO.	P-C-U	LABELLED?	LOC	READY?
CARD RDR		0-0-0			
TAPE		0-1-0			
TAPE		0-1-1			
TAPE		0-1-2			
TAPE		0-1-3			

IOAT

DEMAND 2 ----- Peripheral Maintenance

Every unit represented in the IOAT which is not assigned is the subject of an active I/O operation called a "test-and-wait." This I/O operation will only complete when the unit changes state in some way. When this happens:

1. The interrupt indications will be stored by the I/O machinery.
2. GISMO will discover the interrupt and stack it.
3. The MCP will pick it up in its Main Loop, and control will be passed to the routines responsible for acknowledging the interrupt, which includes any necessary maintenance to the IOAT to reflect the unit's change of state.

CONTROL DEVICE

A CONTROL DEVICE IS A MEANS OF GETTING THE MCP'S ATTENTION.

THE "SPO" IS A DEDICATED CONTROL DEVICE. FOR EXAMPLE ....

1. PUSH "INPUT REQUEST" BUTTON
2. TYPE IN "TD"
3. PUSH "END OF MESSAGE" BUTTON
4. STEP 3 COMPLETED AN I/O OPERATION, WHICH SET THE INTERRUPT FLAG, AND CAUSED THE FOLLOWING CHAIN OF MCP PROCEDURE CALLS:

IO.COMPLETE
↓
CONTROL.LANGUAGE.PROCESSOR (MSG)
↓
TD.ROUTINE
↓
SPOUT ("TIME = <TIME OF DAY>, DATE: <TODAYS DATE>")
↓
INITIATE I/O (SPO, "TIME ---- ")

XXXXXXXXXXXXX

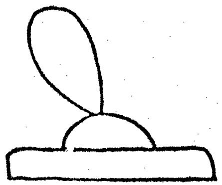
<u>DEMAND 3</u> ----- Control Devices

CARD READER AS CONTROL DEVICE.

CONSIDER IMAGINARY 2 POSITION SWITCH:

BELONGS TO MCP

POS. 1



ASSIGNED TO PROGRAM

POS 2.

NOW IF THE CARD LAST READ HAS:

1. A "?" IN COLUMN 1, THEN THE SWITCH IS SET TO POS 1.

2. "DATA <FILE-NAME>" ANYWHERE ON IT, THEN SWITCH IS SET TO POS 2, AND ASSIGNED TO THE PROGRAM LOOKING FOR THAT <FILE-NAME>

The card reader is not a dedicated control device, but must be shared between the MCP and any programs which wish to use it.

This device was arbitrarily chosen. Conventions could be established to make any input device a control dev e.

DEMAND MANAGEMENT

1. I/O OPERATIONS.
2. PERIPHERAL MAINTENANCE.
3. CONTROL DEVICES.
4. PROGRAM WAITING INITIATION (BOJ).
5. RUNNING PROGRAMS.

In our list of demands we have now worked our way up to number 4. So now we will discuss the whole process of program initiation.

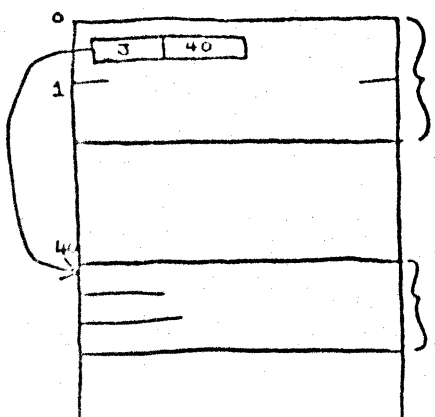
PROGRAM'S LIFE CYCLE

2 STAGES:

1. SCHEDULING.
2. BEGINNING-OF-JOB (BOJ).

PROGRAM MUST:

1. BE ON DISK, & RECORDED IN DIRECTORY
2. ADHERE TO RIGID REQUIREMENTS AS TO ITS FORMAT.



PROGRAM PARAMETER BLOCK; SEGMENTS 0+1

PPB CONTAINS LENGTH/LOCATION POINTER TO FPBs

FILE PARAMETER BLOCKS; 3 AT LOCATION 40 (1 SEGMENT EACH)

LAYOUT OF PROGRAM FILE.

PROGRAM PARAMETER BLOCK
AND
FILE PARAMETER BLOCKS

IN ANY PROGRAM FILE, THE PPB AND THE FPB's ARE THE REQUIRED FIELDS IN WHICH THE PROGRAMMER DESCRIBES HIS PROGRAM TO THE MCP IN A RIGIDLY RESTRICTED WAY.

YUP

COMMAND MCP TO EXECUTE A PROGRAM

CONSIDER A PROGRAM CALLED "DONALD/DUCK" WHICH WILL

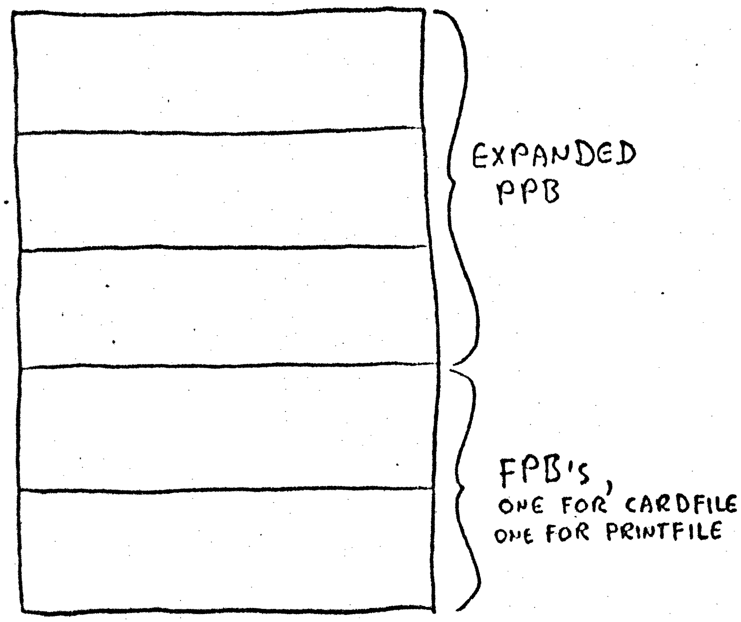
1. REQUEST A CARD DECK LABELLED DUCK/SOUP.
2. REQUEST A LINE-PRINTER.
3. LIST THE CARDS ON THE PRINTER.
4. TERMINATE.

TYPE IN "EXECUTE DONALD/DUCK"

WHAT HAPPENS NEXT?

Given a program formatted under the state constraints, let us follow the process of its execution.

SCHEDULING



A WORKING COPY OF THE DESCRIPTIVE PARTS OF THE PROGRAM FILE BUILT ON DISK BY THE MCP.

1. Scheduling consists of all the preparation for running a job which can be accomplished without actually committing resources.

2. A working description for each execution of a program allows the description to be changed (if desired) for any given execution, without changing the "Mother Copy."

BOJ

1. THE SCHEDULE ENTRY IS THE INPUT WHICH GUIDES THE SEQUENCE OF STEPS BOJ WILL TAKE.

2. PRIME DUTY:
   ALLOCATE MEMORY FOR, AND SET UP A CENTRAL STRUCTURE, IN MEMORY, WHICH WILL ALLOW THE PROGRAM TO BE EXECUTED BY THE MCP.

3. THIS STRUCTURE IS CALLED THE "RUN STRUCTURE".

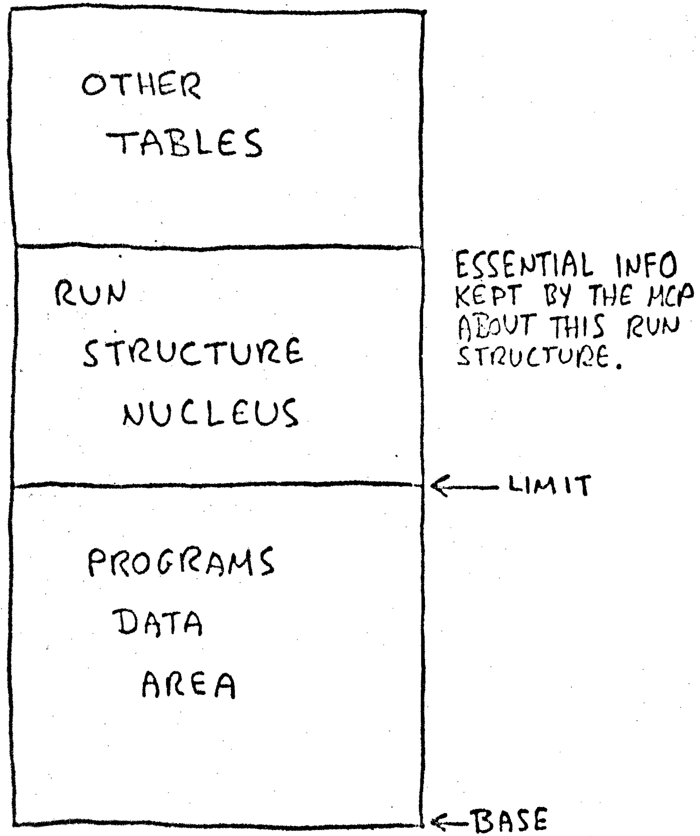
4. IN ADDITION TO THE RUN STRUCTURE, IT IS NORMALLY REQUIRED THAT SEVERAL OTHER SUPPORTING STRUCTURES ALSO BE ESTABLISHED.

5. MAJOR BOJ SETUP CHORES:

   A. DATA
   B. CODE
   C. INTERPRETER
   D. FILES
   E. PROGRAM LINKING

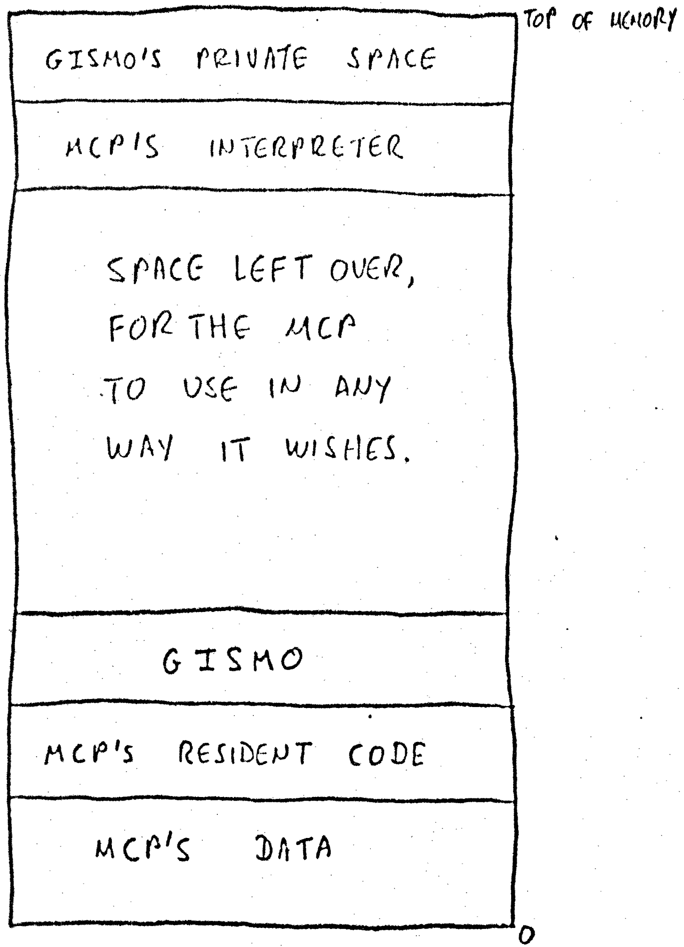
Beginning of job code actually commits resources to the program.

RUN STRUCTURE



The description in the schedule determines the characteristics of a particular run-structure.

B1700 MAIN MEMORY



WHAT CLEAR/START DOES.

Note that the MCP is a program, and as such has a run-structure. In this case the run-structure nucleus is included in Gisma's private space, and "OTHER TABLES" (which are optional for any program) are not used.

VIRTUAL MEMORY

PROBLEM: HOW TO MAKE A BIG PIECE OF INFORMATION FIT IN A SMALL SPACE.

EXAMPLE: THE TOTAL SIZE OF ALL THE MCP'S CODE IS ABOUT 200K BYTES. YET WE GUARANTEE THAT THE MCP WILL RUN IF WE CAN DEDICATE 5K BYTES TO CODE.

HOW DOES IT WORK?

AS IT IS GENERATED, THE CODE IS BROKEN INTO CONVENIENT SIZE CHUNKS.

THESE CHUNKS ARE CALLED CODE SEGMENTS.

REFERENCE TO A PARTICULAR INSTRUCTION WITHIN THE CODE-FILE MUST CONSIST OF 2 PARTS:

1. THE SEGMENT NUMBER
2. THE LOCATION WITHIN THE SEGMENT.

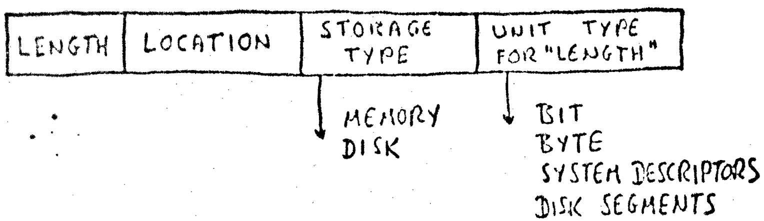
NOW THE SPACE NEEDED IS:

1. ENOUGH FOR THE SEGMENT IN USE
2. ENOUGH FOR A TABLE CONTAINING POINTERS TO ALL THE ABSENT SEGMENTS, SO THEY CAN BE BROUGHT INTO MEMORY IF REFERENCED.

At this point we must side track for a moment, and develop some supporting concepts, before we can proceed with BOJ processing directly.

SYSTEM DESCRIPTOR

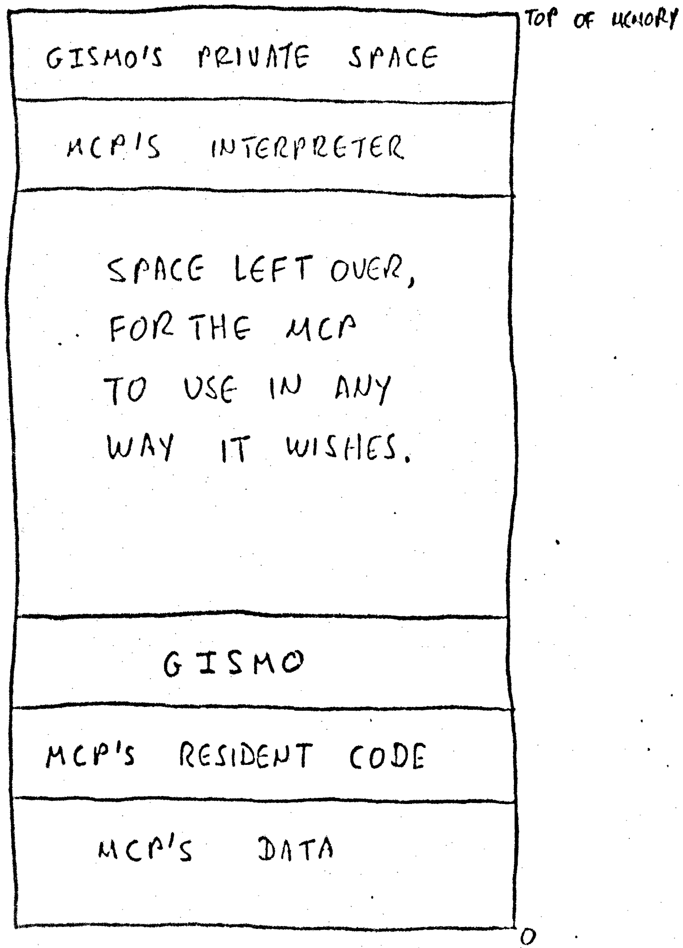
A STANDARD WAY OF DESCRIBING A PIECE OF INFORMATION



The table of pointers just mentioned is called a "segment dictionary", and it represents one of the most common requirements of the MCP -- a "standard pointer" which is able to represent any piece of information, and be fully general with respect to length, location, and format.

The segment dictionary, then, is a list of System Descriptors.

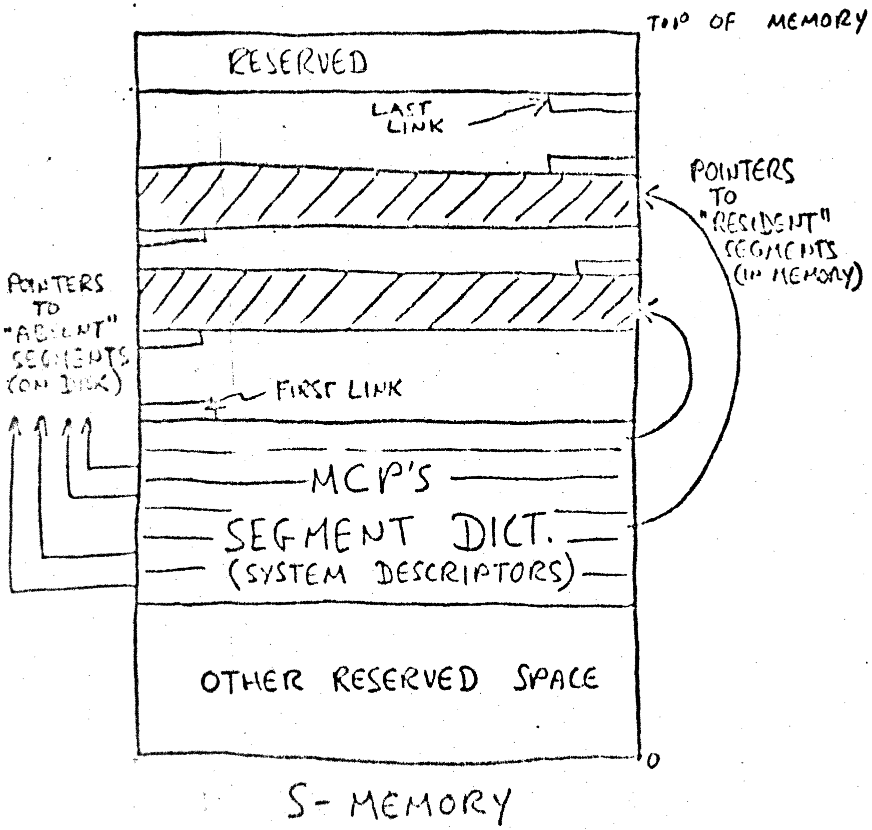
B1700 MAIN MEMORY



This picture again is an over-simplification. The MCP Segment Dictionary also resides in low end of memory along with other reserved fields like Gisma and resident code.

WHAT CLEAR/START DOES.

MCP's CODE SEGMENTS AND SEGMENT DICTIONARY.



In the coarse of MCP execution, if an absent segment is referenced, the MCP's <u>interpreter</u> will perform the necessary procedure calls to bring the missing segment into main memory, so we can say that for the main-line MCP code, it "seems" as if all segments are always present.

Now, we can say that what we mean by <u>virtual</u> is that <u>functionally</u> the MCP behaves as if it really did have 200KB of memory dedicated to its code.

BOJ

1. THE SCHEDULE ENTRY IS THE INPUT WHICH GUIDES THE SEQUENCE OF STEPS BOJ WILL TAKE.

2. PRIME DUTY:
   ALLOCATE MEMORY FOR, AND SET UP A CENTRAL STRUCTURE, IN MEMORY, WHICH WILL ALLOW THE PROGRAM TO BE EXECUTED BY THE MCP.

3. THIS STRUCTURE IS CALLED THE "RUN STRUCTURE".

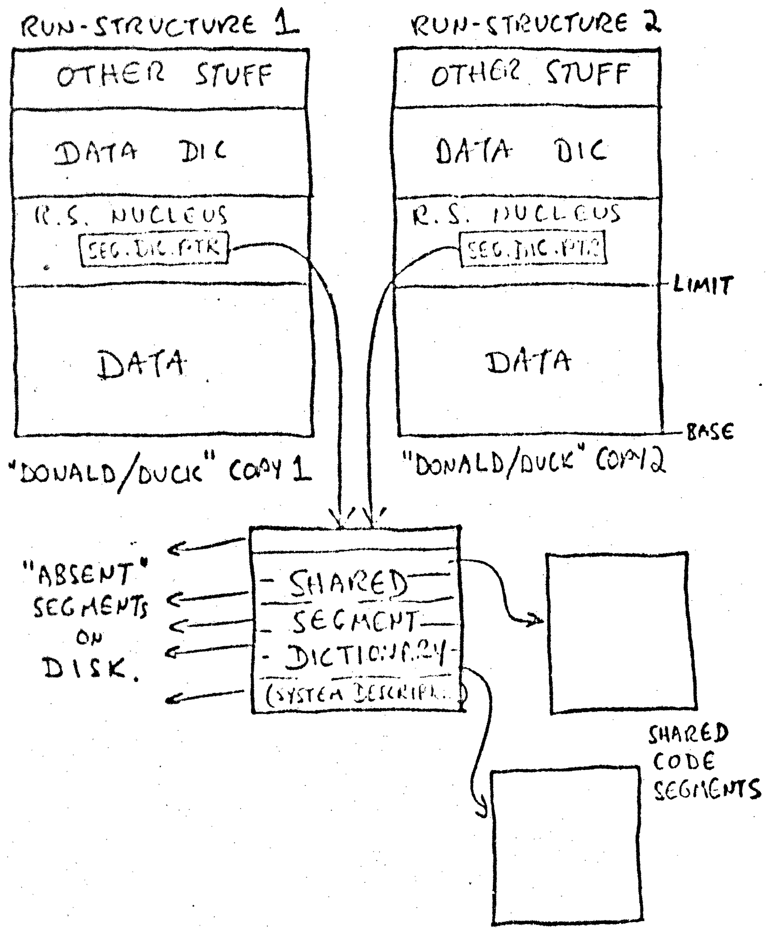
4. IN ADDITION TO THE RUN STRUCTURE, IT IS NORMALLY REQUIRED THAT SEVERAL OTHER SUPPORTING STRUCTURES ALSO BE ESTABLISHED.

5. <u>MAJOR BOJ SETUP CHORES:</u>

   A. DATA
   B. CODE
   C. INTERPRETER
   D. FILES
   E. PROGRAM LINKING

Back to BOJ chores.

CODE MANAGEMENT



Frequently the amount of data directly referenced by a program will exceed available memory.

So, the MCP provides a virtual management scheme for a program's data, if the program requests it (such a request would be part of the description in the Program Parameter Block).

There is a B1700 software rule which says that a program may only do memory writes between that programs base and limit. Thus, if an absent data segment is referenced, it must be brought into memory between the programs base and limit. Also, a program might wish to have just some of its data handled in this way, and the rest "resident" -- always present and able to be accessed without delay. In this case, the resident data also must be present between base and limit.

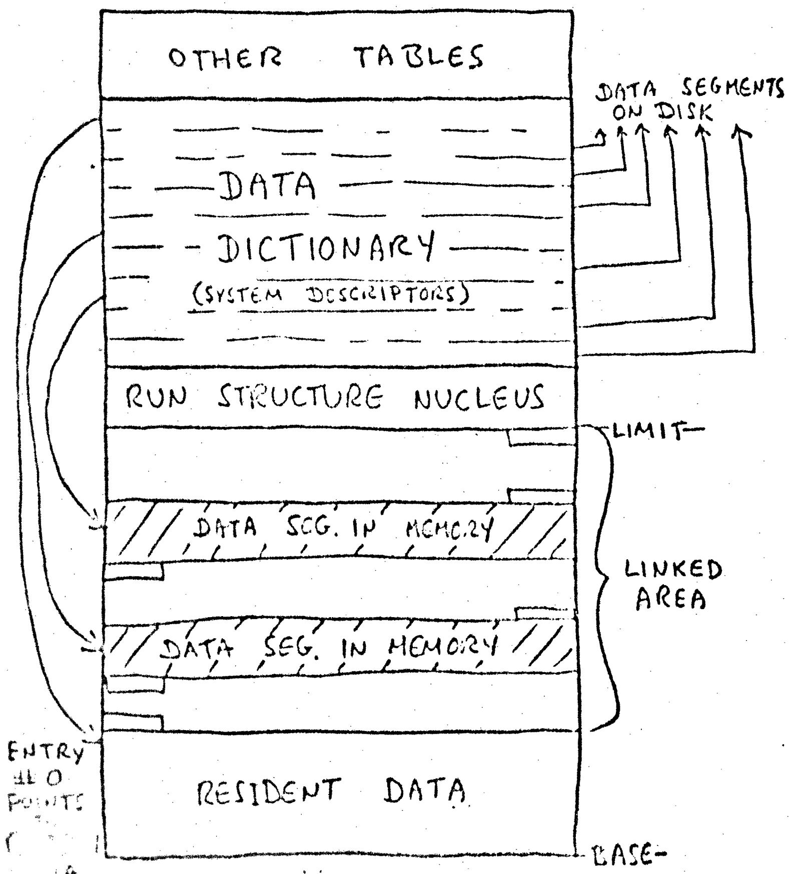
So, the MCP will set up (if requested) between a programs base and limit;

1) a space for resident data,
2) a "linked area" (just like the MCP's) for data segments.

Note: When one data segment is "overlayed" by another, the first must be saved by being written back to the disk. This disk address is another of the things kept in the memory link.

A "Data Dictionary" of system descriptors is part of the Program File, and is pointed to in the Program Parameter Block. This becomes part of the Run Structure.

PROGRAM DATA MANAGMENT



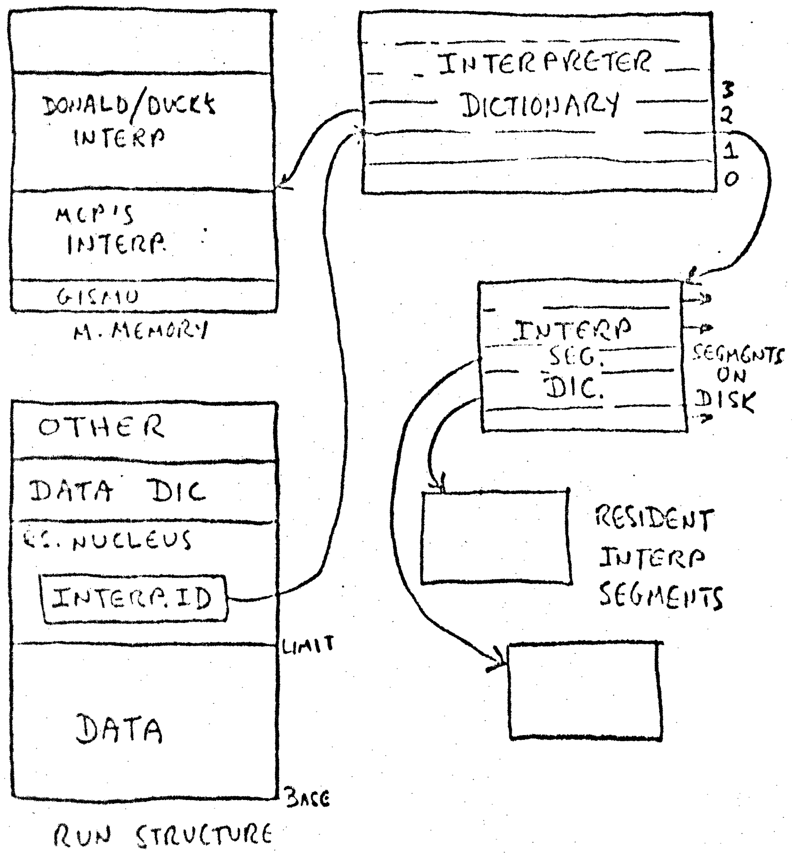
RUN STRUCTURE

The code segment dictionary is also in the program file.

Three important distinctions between Code and Data:

1) Code is "Read Only" so it need not be in the programs "Write Space" (Base — Limit).
2) Hence the code may be outside the run structure, allowing it to be shared with another copy of the same program (this is called reentrance).
3) Also, it need not be written back when overlayed.

INTERPRETER MANAGEMENT.

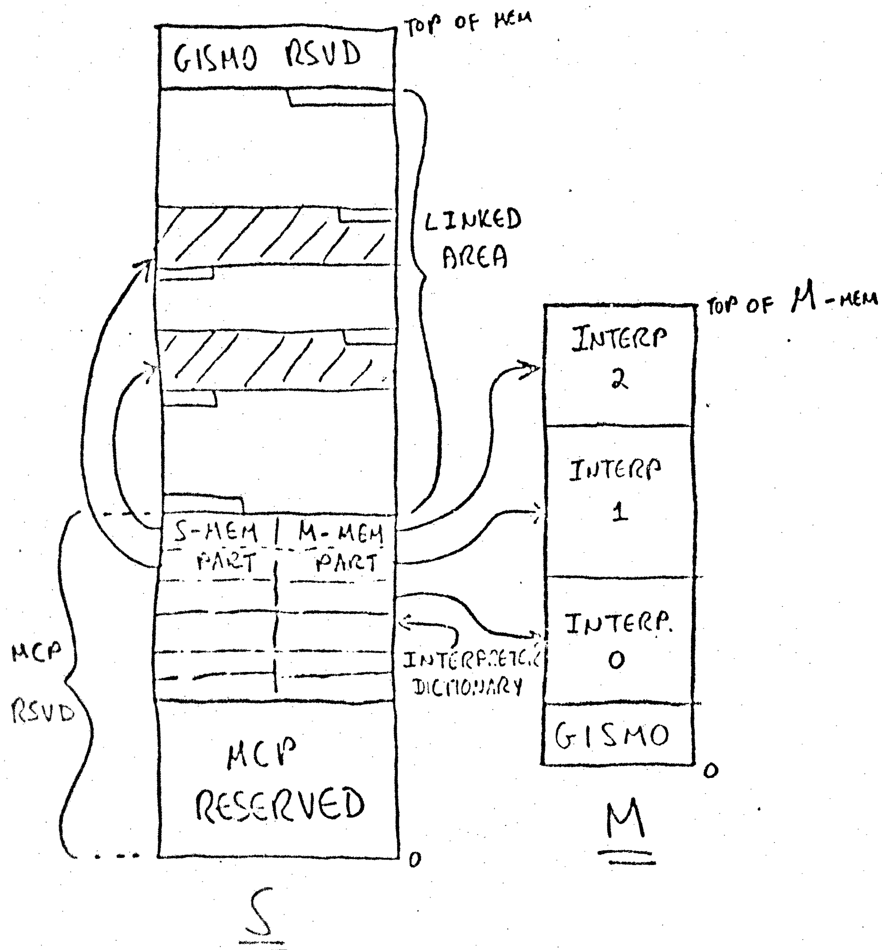


The name of the interpreter is in the PPB.

Interpreters may also be shared.

Remembering that an interpreter may be split between S-Memory and M-Memory, it is possible for some of the S-Memory part to be resident and the rest segmented.

M - MEMORY ALLOCATION



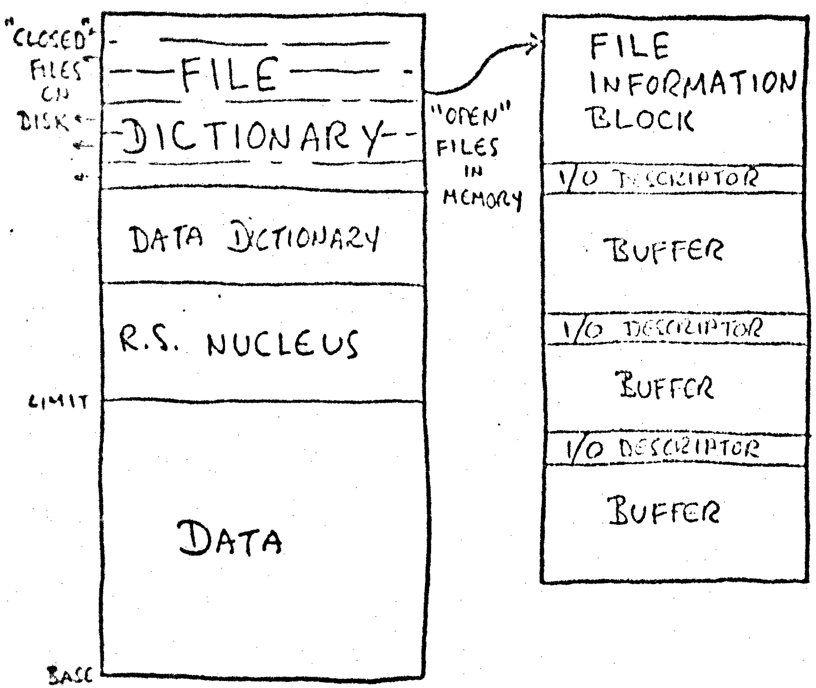
In this picture (which we've already seen) I left out the Interpreter Segment Dictionary.

The Interpreter Dictionary is the same one as in the last picture, and it resides in MCP reserved memory.

Note: The Interpreter Dictionary has one entry for each active interpreter.

Each interpreter may have an Interpreter Segment Dictionary, with one entry for each segment.

FILE MANAGEMENT



RUN STRUCTURE

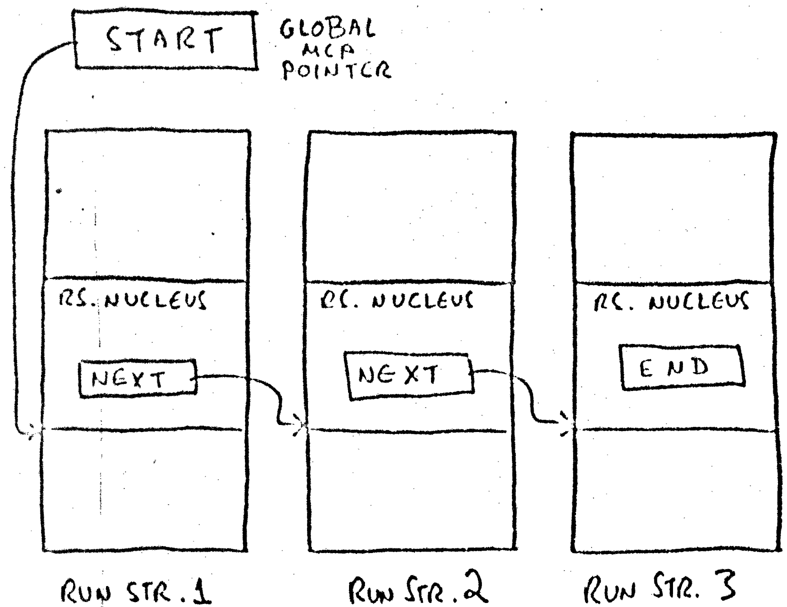
The file dictionary is built at BOJ, with one entry for each FPB in the schedule entry.

Open builds on FIB, I/O descriptors, and Buffers.

The FIB is a run-time rendition of the FPB.

It is approximately true to say that when a file is "CLOSED," its File Dictionary entry points to the corresponding FPB in the schedule entry, on disk, while if it is "OPEN," the File Dictionary entry points to an FIB in memory.

PROGRAM LINKING



The final step of BOJ is to link the programs into a chain of which all the currently running programs are members.

Linking is by priority, "START" pointing to the highest priority program.

BOJ

1. THE SCHEDULE ENTRY IS THE INPUT WHICH GUIDES THE SEQUENCE OF STEPS BOJ WILL TAKE.

2. PRIME DUTY:
   ALLOCATE MEMORY FOR, AND SET UP A CENTRAL STRUCTURE, IN MEMORY, WHICH WILL ALLOW THE PROGRAM TO BE EXECUTED BY THE MCP.

3. THIS STRUCTURE IS CALLED THE "RUN STRUCTURE".

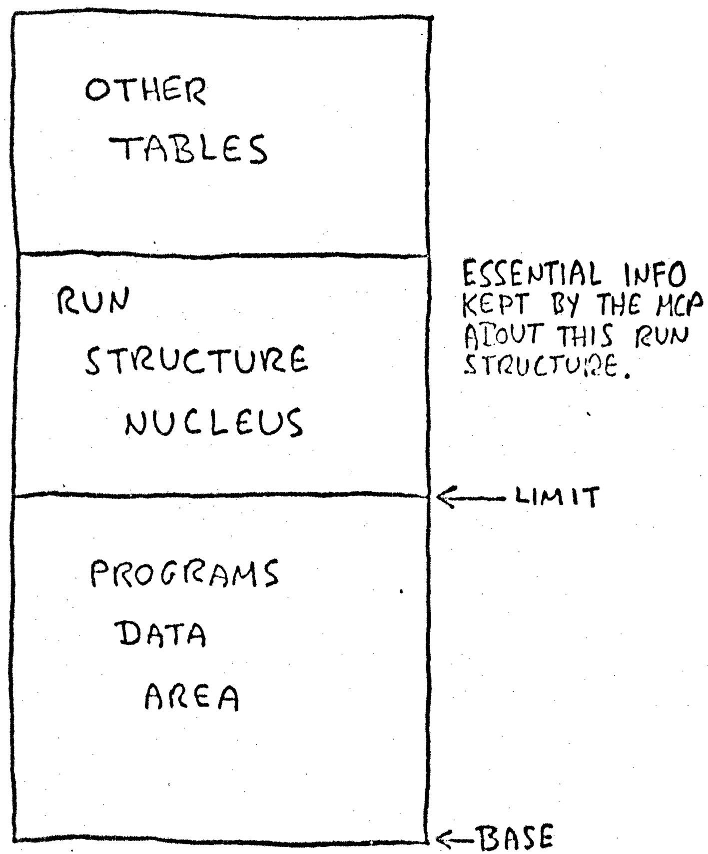
4. IN ADDITION TO THE RUN STRUCTURE, IT IS NORMALLY REQUIRED THAT SEVERAL OTHER SUPPORTING STRUCTURES ALSO BE ESTABLISHED.

5. <u>MAJOR BOJ SETUP CHORES:</u>

   A. DATA
   B. CODE
   C. INTERPRETER
   D. FILES
   E. PROGRAM LINKING

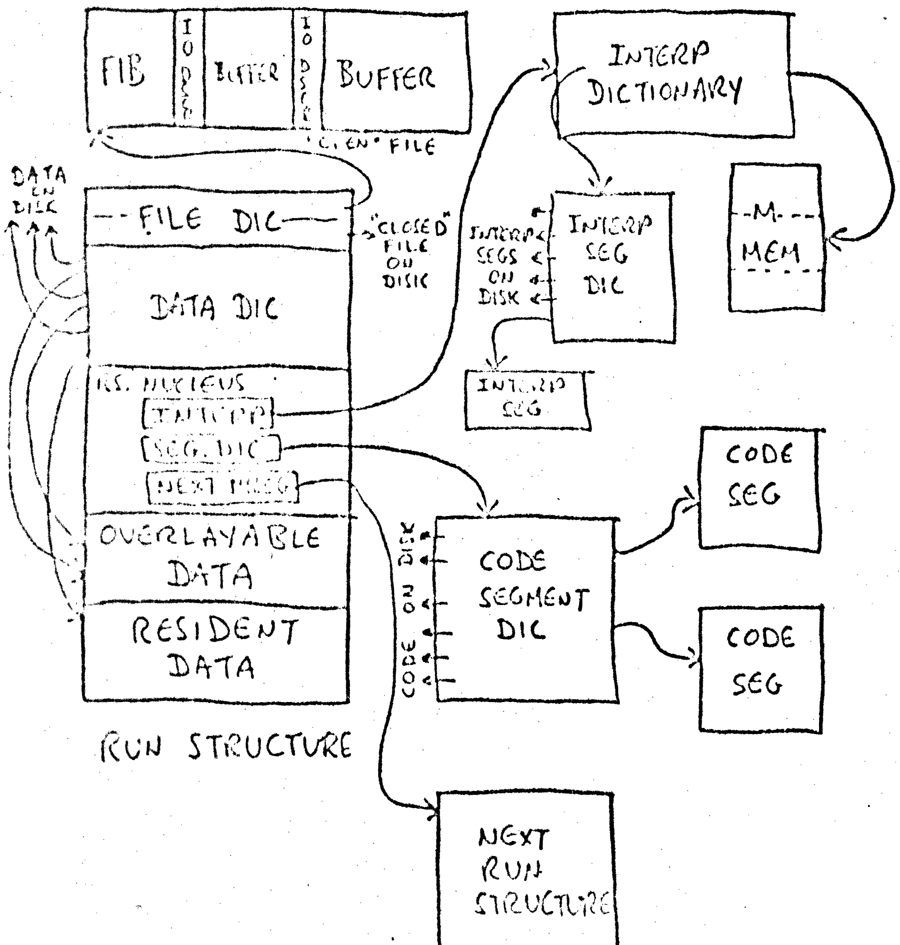
So much for BOJ chores.

RUN STRUCTURE



Here's how we started......

Run Structure And Related Info:



and here's how we wound up.

PROGRAM STATES

1. NOT.QUEUED ~ PROGRAM IS RUNNING OR CURRENTLY BEING SERVICED.
2. READY.QUEUE ~ PROGRAM IS REQUESTING THE USE OF A PROCESSOR.
3. COMMUNICATE.QUEUE ~ PROGRAM IS REQUESTING ATTENTION OF THE MCP.
4. WAIT.QUEUE ~ PROGRAM IS TEMPORARILY STOPPED, WAITING COMPLETION OF SOME I/O OPERATION.

A PROGRAMS CURRENT STATE IS REFLECTED IN A FIELD IN "RUN STRUCTURE NUCLEUS".

Now the program is "ready to run," ie it is demanding the use of the processor.

During its life-time it passes in and out of these four basic states.

When BOJ finishes, the program will be in the "ready-Q."

REINSTATE / COMMUNICATE

1. SUPPOSE THE MCP WISHES TO ALLOW PROGRAM "X" TO RUN.

2. THE MCP EXECUTES SOME CODE WHICH CAUSES THE MCP'S INTERPRETER TO CALL GISMO IN A SPECIAL WAY, PASSING A POINTER TO X'S RUN STRUCTURE.

3. GISMO PASSES CONTROL TO X'S INTERPRETER, INDICATING THAT IT IS X WHICH IS TO RUN.

4. WHEN X REQUIRES SOME SERVICE OF THE MCP, OR IF X IS INTERRUPTED DURING EXECUTION, X'S INTERPRETER WILL BUILD A FIXED FORMAT MESSAGE TO THE MCP, WHICH IS PLACED IN A SPECIAL LOCATION IN THE RUN STRUCTURE NUCLEUS, AND WHICH EXPLAINS WHAT HAPPENED. THE INTERPRETER THEN RETURNS CONTROL TO GISMO.

5. GISMO PASSES CONTROL TO THE MCP'S INTERPRETER, INDICATING THAT IT IS THE MCP WHICH IS TO RUN.

6. THE MCP STARTS TO RUN JUST AFTER THE POINT FROM WHICH IT CALLED GISMO IN #2.

7. THE MCP EXAMINES, AND ACTS ON THE MESSAGE WHICH WAS PASSED BACK.

Here's what happens when the MCP discovers a Job in the ready Q:

<u>Note:</u> If the MCP determines that Program "X" must wait for some event to happen before X can proceed (e.g. tape "T" to be mounted), then X will be put in the "wait Q."

When the event happens, X will be transferred to the Communicate Q, where X will again be discovered and serviced by the MCP. If all went well on this second try, then X will be put in the Ready Q, and we start all over again.

WHAT GISMO DOES

GISMO IS A RESIDENT MICRO PROGRAM WITH THESE DUTIES.

1. REINSTATE AND COMMUNICATE

GISMO HANDLES THE SMOOTH TRANSITION OF CONTROL OF THE PROCESSOR BETWEEN THE MCP AND THE PROGRAMS.

2. SOFT I/O

GISMO IS IMMEDIATELY INFORMED WHENEVER THE SERVICE REQUEST FLAG IN THE PROCESSOR IS TURNED ON BY SOME PART OF THE I/O MACHINERY. GISMO CALLS SOFT I/O, WHICH GRANTS THE REQUEST AND THEN RETURNS TO GISMO WHICH RETURNS THE PROCESSOR TO WHOEVER WAS USING IT AT THE TIME THE FLAG WAS SET ON

3. INTERRUPTS

IF THE INTERRUPT FLAG IN THE PROCESSOR IS TURNED ON, THEN GISMO IS INFORMED IMM-DIATELY, AND IF THE MCP IS RUNNING, THEN THE INTERRUPT WILL BE "STACKED" ELSE, IF A PROGRAM IS RUNNING, A MESSAGE EXPLAINING THIS WILL BE BUILT, AND CONTROL RETURNED TO THE MCP.

We can now look more closely at GISMO's duties.

WHAT GISMO DOES

GISMO IS A RESIDENT MICRO PROGRAM WITH THESE DUTIES.

1. REINSTATE AND COMMUNICATE

GISMO HANDLES THE SMOOTH TRANSITION OF CONTROL OF THE PROCESSOR BETWEEN THE MCP AND THE PROGRAMS.

2. SOFT I/O

GISMO IS IMMEDIATELY INFORMED WHENEVER THE SERVICE REQUEST FLAG IN THE PROCESSOR IS TURNED ON BY SOME PART OF THE I/O MACHINERY. GISMO CALLS SOFT I/O, WHICH GRANTS THE REQUEST AND THEN RETURNS TO GISMO WHICH RETURNS THE PROCESSOR TO WHOEVER WAS USING IT AT THE TIME THE FLAG WAS SET ON

3. INTERRUPTS

IF THE INTERRUPT FLAG IN THE PROCESSOR IS TURNED ON, THEN GISMO IS INFORMED IMM-DIATELY, AND IF THE MCP IS RUNNING, THEN THE INTERRUPT WILL BE "STACKED" ELSE, IF A PROGRAM IS RUNNING, A MESSAGE EXPLAINING THIS WILL BE BUILT, AND CONTROL RETURNED TO THE MCP.

We can now look more closely at GISMO's duties.

MCP MAIN LOOP

```
DO FOREVER
. IF ANY INTERRUPTS
  THEN CALL IO.COMPLETE(X)
   |
   |
   .
  OTHER STUFF
   |
   |
   |
END
```

So far, this has been our working model of the MCP.

MCP MAIN LOOP

```
DO FOREVER
    HANDLE INTERRUPTS
    EMPTY COMMUNICATE QUEUE
    IF ANYBODY SCHEDULED
       THEN CALL BOJ
    EMPTY THE READY QUEUE
END
```

We can now expand this.....

DEMAND MANAGEMENT

1. I/O OPERATIONS.
2. PERIPHERAL MAINTENANCE.
3. CONTROL DEVICES.
4. PROGRAM WAITING INITIATION (BOJ).
5. RUNNING PROGRAMS.

This completes our investigations of Demand Management, and of this MCP.

MCPI

Hardware Lecture Notes

I would like to answer two basic questions about MCPI in this lecture.

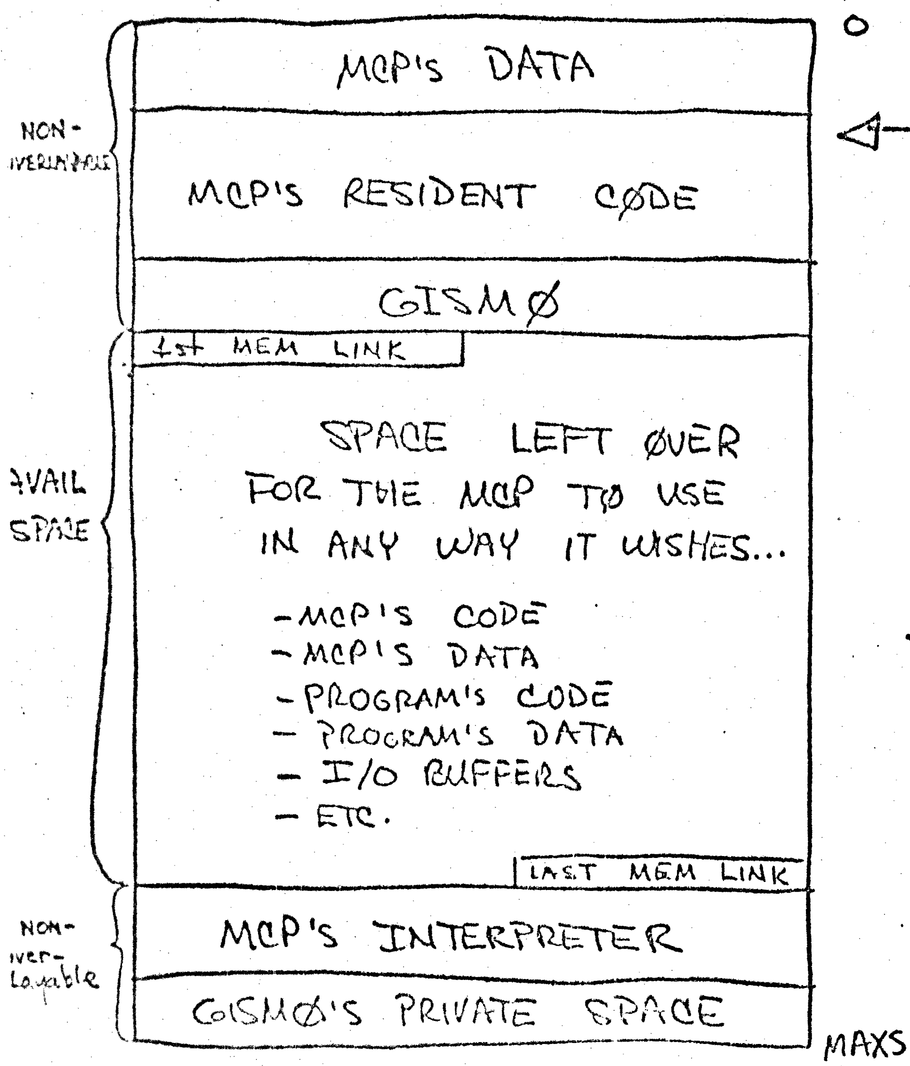
① Why was MCPI developed?

② How were the design goals achieved?

I'm assuming you've heard Peter's lecture and are quite knowledgeable on MCPII and operating systems in general.

In order to answer question #1 we need to examine in more detail what goes on when MCPII runs a program.

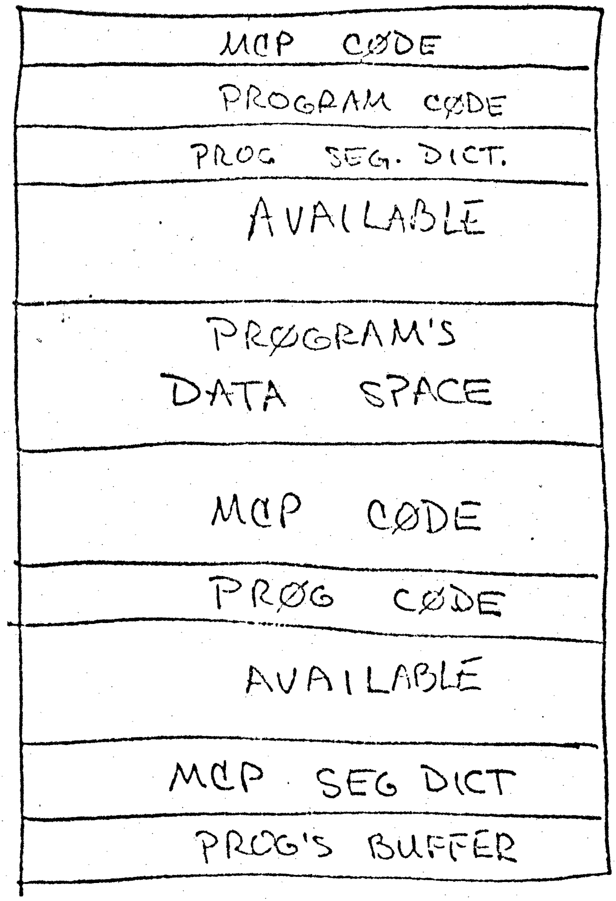
MCP II MEMORY LAYOUT



Here we see MCP II's memory layout just after CLEAR/START. As you can see there are two types of space

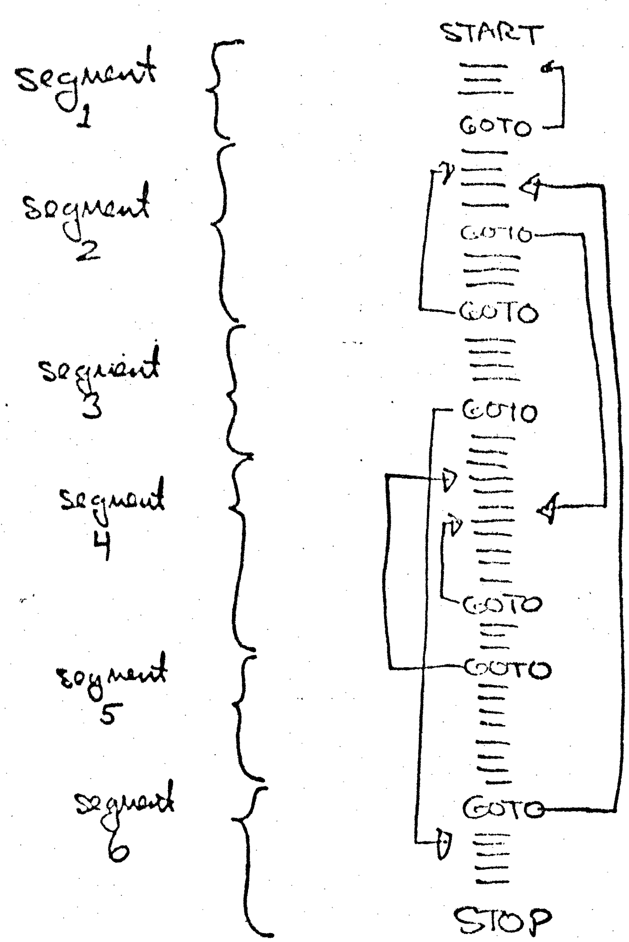
- non-overlayable space (ie resident space)
- overlayable space (or available space)

— It is in the available space that MCP II must find room to run a program.



MCP II'S AVAIL. SPACE

- If we look more closely at MCP II's overlayable space, we can see that it is composed of many different types of things, which are managed by means of memory links. (not shown for convenience)
- Usually there are more things contending for memory space than there is available memory. Consequently the MCP is constantly having to overlay things already in memory for things that are needed right now.
- Memory is said to be "over-committed"
- Notice that the MCP's own code is contending for memory along with the program's code.



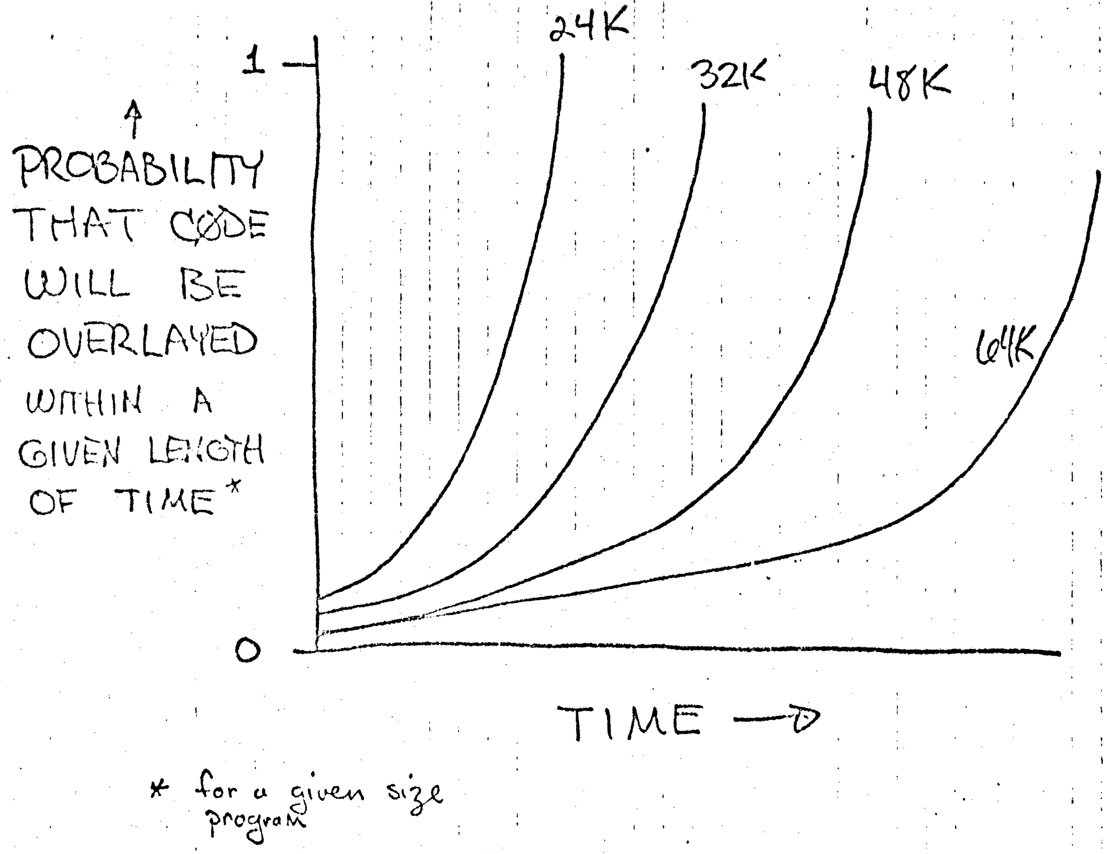
→ If we look at what happens when a program executes, we can see why the MCP must continually overlay memory.

- Assume our example program, (at left), is 2 or 3 times the size of the MCP's Available Area. Thus it is impossible to get the whole program in memory at one time.

- When the program starts segment #1 is in memory. Subsequently as the program executes the "locus of execution" continually crosses segment boundrys. When this occurs, the MCP is notified and it determines if the next required segment is in memory or not.

If it is, the MCP merely releases control to that segment of code. If it isn't however then the MCP brings the required segment into memory and releases control to it.

→ When the MCP brings in a new segment it may have to overlay a segment already in memory.

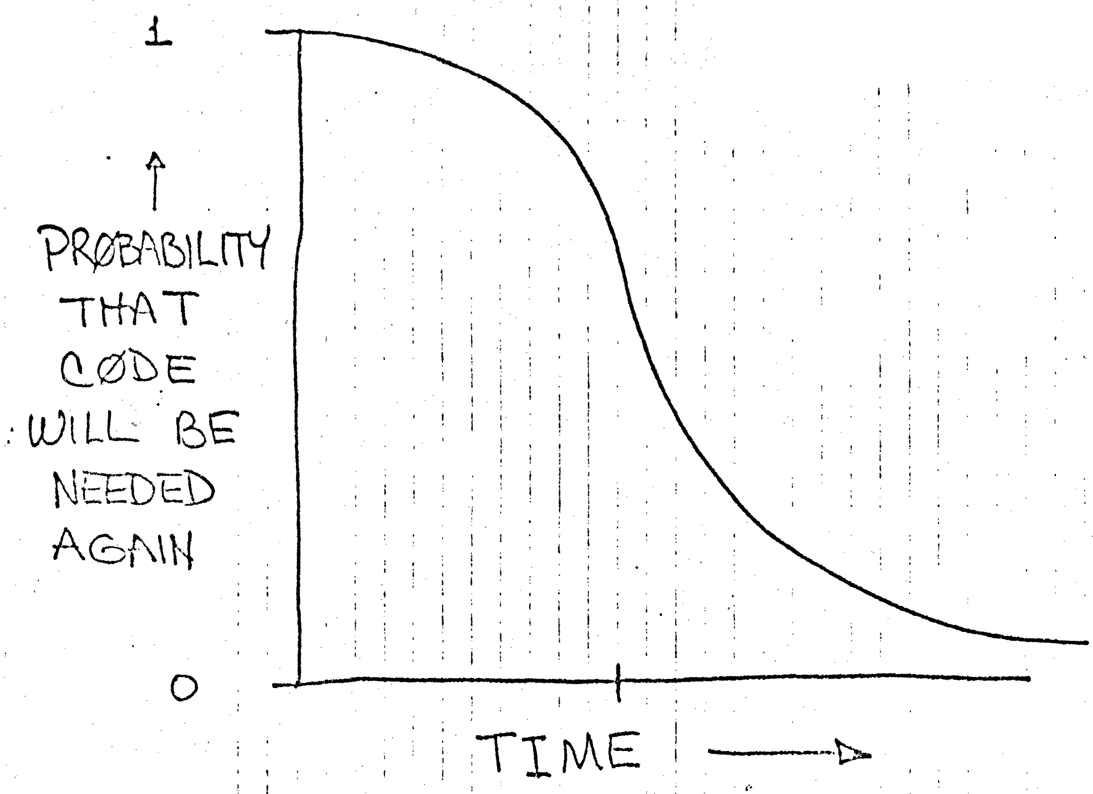


- One of the most critical factors affecting the number of overlays occuring in a given length of time is the size of available memory.

SMALLER MEMORY ⟹ more overlays per unit time
" " ⟹ the shorter the length of time a segment of code will stay in memory before being overlayed.

- This can be expressed as a probability (see figure)

- The shape of the curves here is not important, only that after some length of time the probability of being overlayed goes to 1.



- Similarly, for an *average* program the probability that a piece of code will be needed in memory stays near 1 for some length of time and then drops to zero.

- This is easy to understand. When a piece of code is brought into memory it is executed for some length of time, and then is not needed as the locus of execution passes to some new segment. Averaging over many programs and many segments produces a curve like the above.

- In fact what is being said here could apply to small groups of segments as well as individual segments.

- If we compare the graphs on the previous two pages we can see that in small memory a piece of code, on the average, may be overlayed before the probability that it is still needed has dropped to some small value. If this occurs then the MCP "in all probability" will have to bring it back into memory. In fact it may have to bring it back into memory many times.

- Now when the MCP does an overlay the program is not executing, and in fact on many occasions the time spent overlaying can and does exceed the time spent executing.

BUZZ WØRDS

① <u>WØRKING SET</u>

- Those code segments that are necessary to run a program "efficiently". That is,

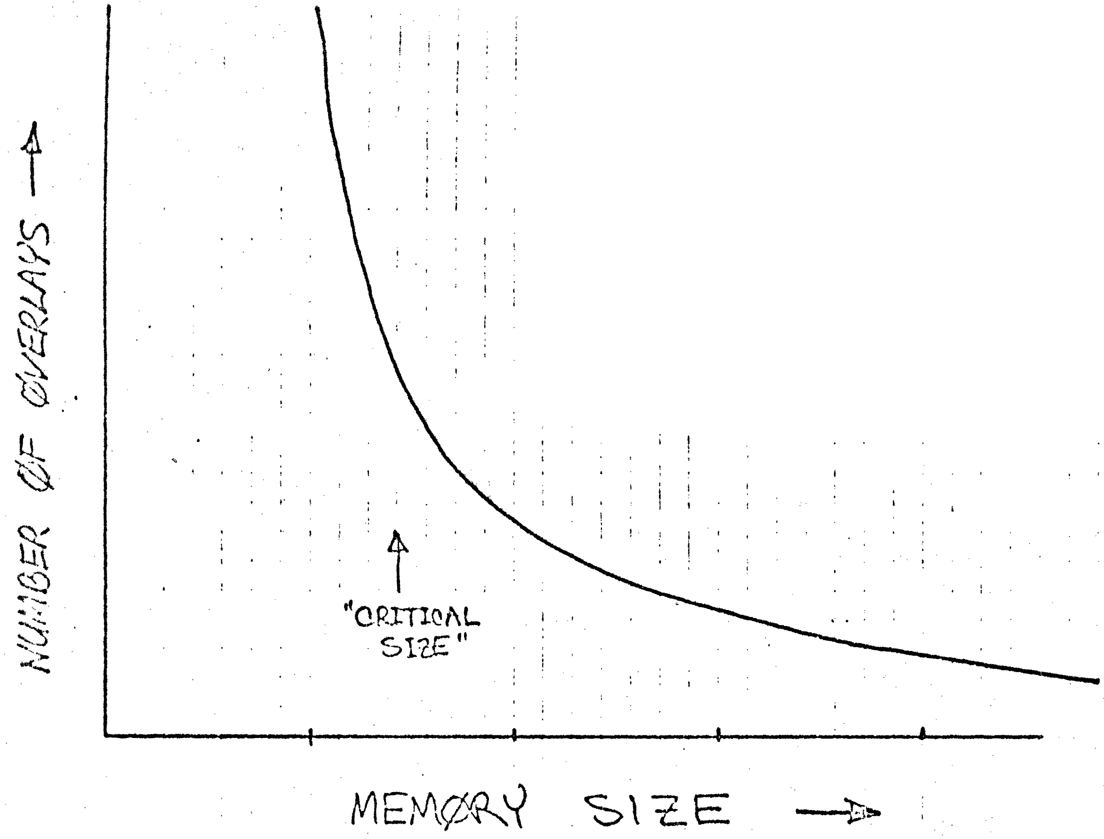
OVERLAY.TIME << EXECUTIØN.TIME

② <u>THRASHING</u>

- occurs when overlay time becomes "unacceptable".

- Two words are used alot in software circles to express the ideas we've been discussing, to wit

- If we can keep a programs working set in available memory then thrashing won't occur. If however the amount of memory on a machi is somewhat small compaired to the size of the programs we want to run, then we will have thrashing.



- In fact for a program with a working set of a given size, if we "squeeze" memory ever smaller then at a certain point known as the "critical size" the number of overlays will approach infinity.
- At that point we say the system has begun thrashing.
- Leaving theoretical matters and getting back to MCP II, it was found empirically that the critical size for MCP II with only small to medium size programs was on the order of 32-48K bytes.

MARKETING DEMANDED

1. 16 K MINIMUM SYSTEM
2. EXECUTIØN TIMES ØN THE ØRDER ØF THØSE FØR SYSTEM/3 WITH SIMILAR MEMØRY.

- Now it turned out that Marketing wanted a machine to compete against the small end of the SYSTEM/3 market. A SYSTEM/3 with 12 K bytes of memory was the smallest configuration we had to compete against.
- Since SYSTEM/3 had no operating system to speak of, our marketing organization was willing to concede 4K bytes and allow our minimum system to be 16K bytes.
- MCP II remember required about 32K bytes to run even small programs well.

MCP II

SPACE REQUIRED

MCP II'S INTERP	7000 Bytes
GISMØ	3400
DATA & STACKS	4000
RESIDENT CØDE	1700
MISC.	1000
	17,100 Bytes

- Not to mention:
  - Program's Interpreter
  - Buffer space
  - Program's Data Space
  - Program's R.S. NUCLEUS
  - Program's CØDE

- To see why MCP II needed so much space lets add up a few numbers
- As we can see (at left) the resident space itself is quite large, and it is only in 32K that the available space approaches 50% of total memory

c

EXAMPLE RUN TIMES

MCP II

	64K	32K	24K	16K
COMPILE SAV001	5:24	7:30	∞	∞
EXECUTE SAV001	0:55	0:55	1:50	∞

SYSTEM/3	12K
COMPILE	4:59
EXECUTE	1:45

- In addition, the execution times themselves weren't too impressive due to the fact that the system began thrashing in the 32K byte region.
- "SAV001" is a small RPG program, and as you can see it begins to thrash 24-32K region.
- The RPG compiler, a large program, begins thrashing in the 32-64K region

MCP II
DESIGN GOALS

① 16 – 48K Systems

② No thrashing for small programs in 16K.

– Consequently it was decided to construct a new MCP that would be able to run small programs in 16K bytes efficiently

– In order to accomplish this goal a number of diverse ~~the~~ techniques were used.

- We are now ready to answer the 2nd question we posed at the beginning of this lecture: How did we achieve the design goals of MCP I.

- We started out by trying to cut down the size of MCP II as much as possible. We did this by limiting its scope and power.

  ① No multi-programming

  ② Limit the number and type of peripherals available.

  ③ Limit the types of functions the MCP will perform for a program
    - no data management
    - no data communications
    - no variable length record files
    - plus numerous other limitations

  ④ Make MCP's interpreter smaller by excluding those functions the MCP doesn't use or need (such as the "sort" ops)

  ⑤ Make GISMØ smaller for the same reason.

MCP I

SPACE	REQUIRED
MCPI'S INTERP	4000
GISMØ	1750
DATA & STACKS	4000
RESIDENT CØDE	1700
MISC.	1000
	12,450

- Looking at the new figures for resident code we can see that it is possible to run in 16K with about 4K left as available space.
- This is enough to run the MCP but still not enough to run a program too. Even a small program requires about 8K bytes all to itself.

How a Program Executes

BØJ

OPEN FILE 1
OPEN FILE 2
OPEN FILE 3

}

10,000 READS & WRITES
+
2000 CØDE ØVERLAYS

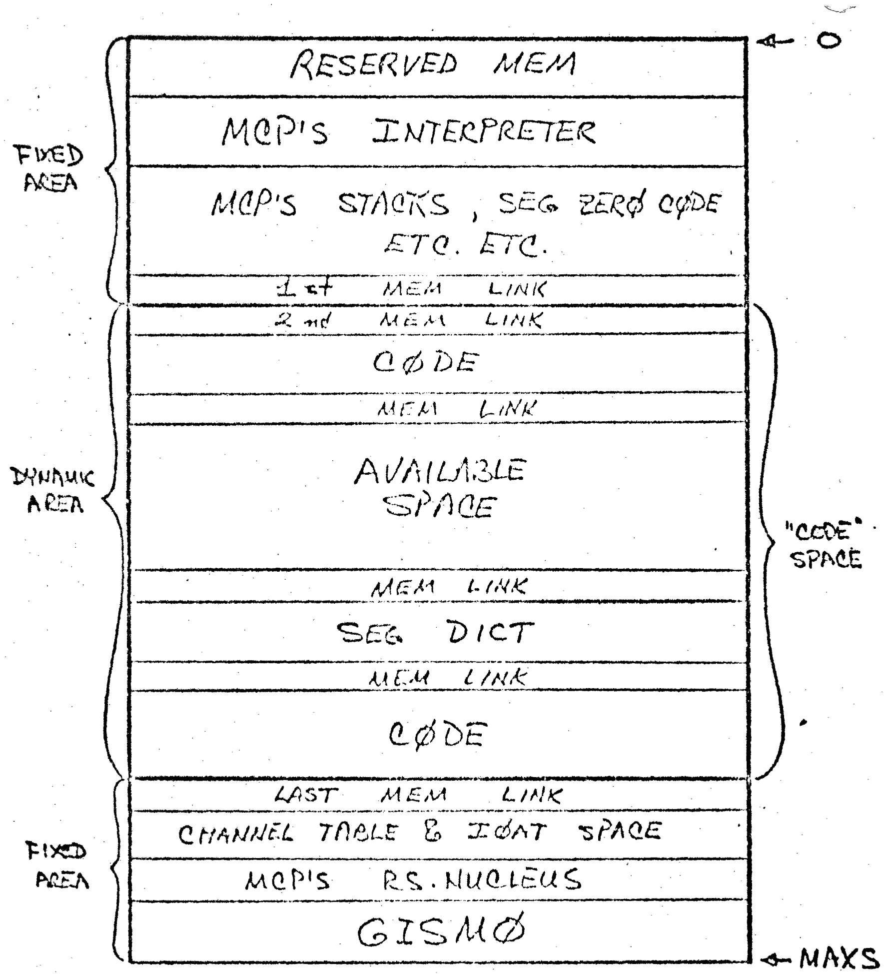
}

CLØSE FILE 1
CLØSE FILE 2
CLØSE FILE 3

EØJ

- To understand what we did next we have to look again at what happens when a program executes.

- As we know the MCP performs certain "functions" for a program, but the frequency with which various functions are performed is by no means equal. In fact, Reads, Writes, and Code Overlays constitute the vast majority of communicates to the MCP, by typical RPG and CØBØL programs.



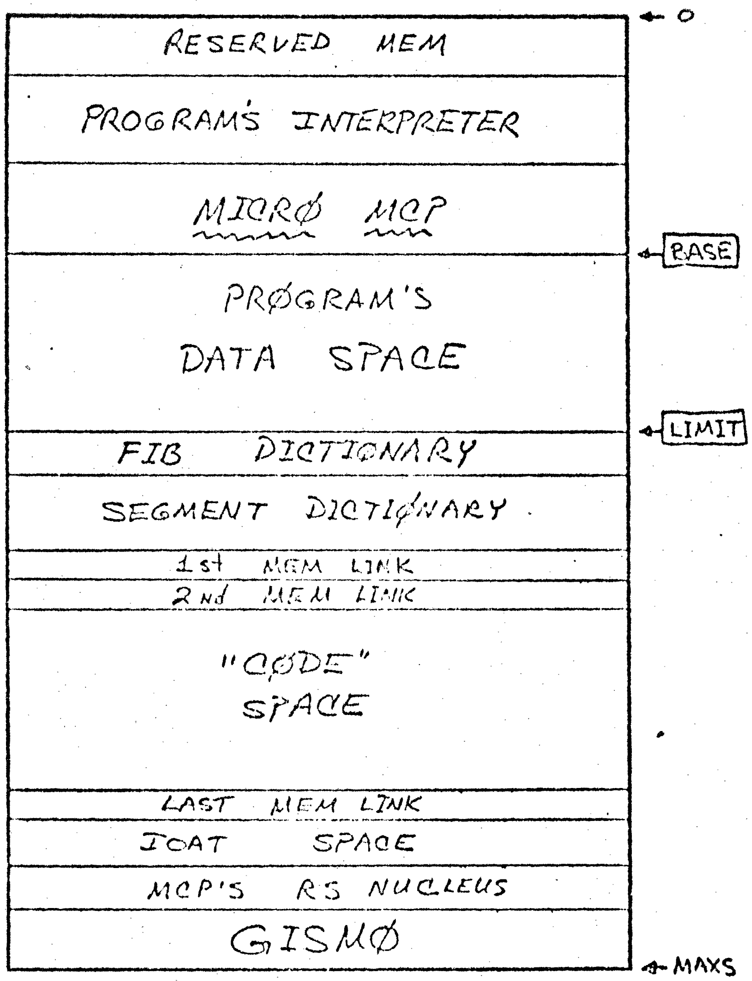
MCPI MEMØRY LAYØUT

- We decided on this basis to "get rid of" most of the MCP when it was not needed, and to create a small micro-coded module to perform the very frequent Reads, Writes, and Code overlays. (R, W & CØ's)

- We thus decided to do two things

  ① "SWAP" out to disk all of the MAIN MCP not needed to do R, W, & CØ's

  ② Create a "MICRO/MCP" to be resident with the program and perform its R, W, & CØ's

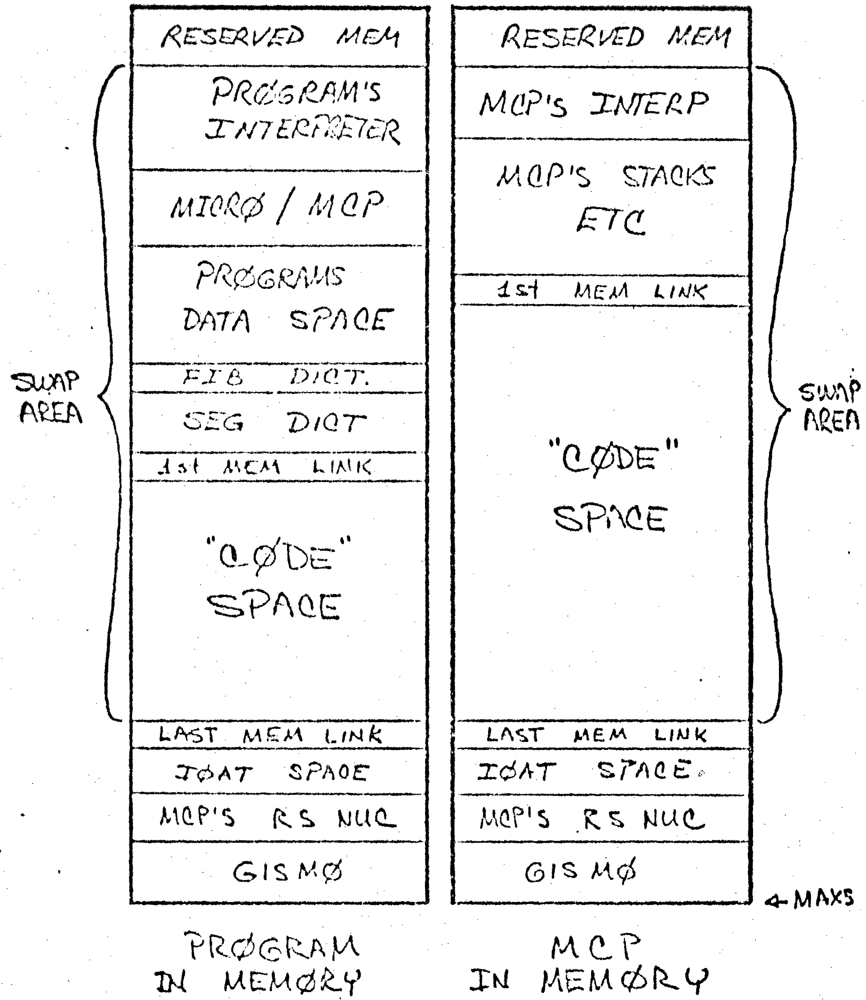


PRØGRAM'S MEMØRY LAYØUT

- Notice that by micro coding the MICRO/MCP that we obtain two advantages:

  ① Micro code is faster than the interpreted SDL code to perform the R, W, & CØ's

  ② Micro code does not require an interpreter, hence we can also swap out the MCP's interpreter along with the MCP.



- We thus ended up with a system where the program and MAIN MCP are never in memory at the same time.

- A certain amount of space is always resident to hold GISMØ (who does the actual swapping), and for passing information etc.

- Space is reserved on disk to hold the "swap images"

- The amount of space required to hold non-swapable items now drops from about 12K to 4K. That leaves about 12K on a 16K size system for the program its interpreter and the MICRØ/MCP.

EXAMPLE RUN TIMES

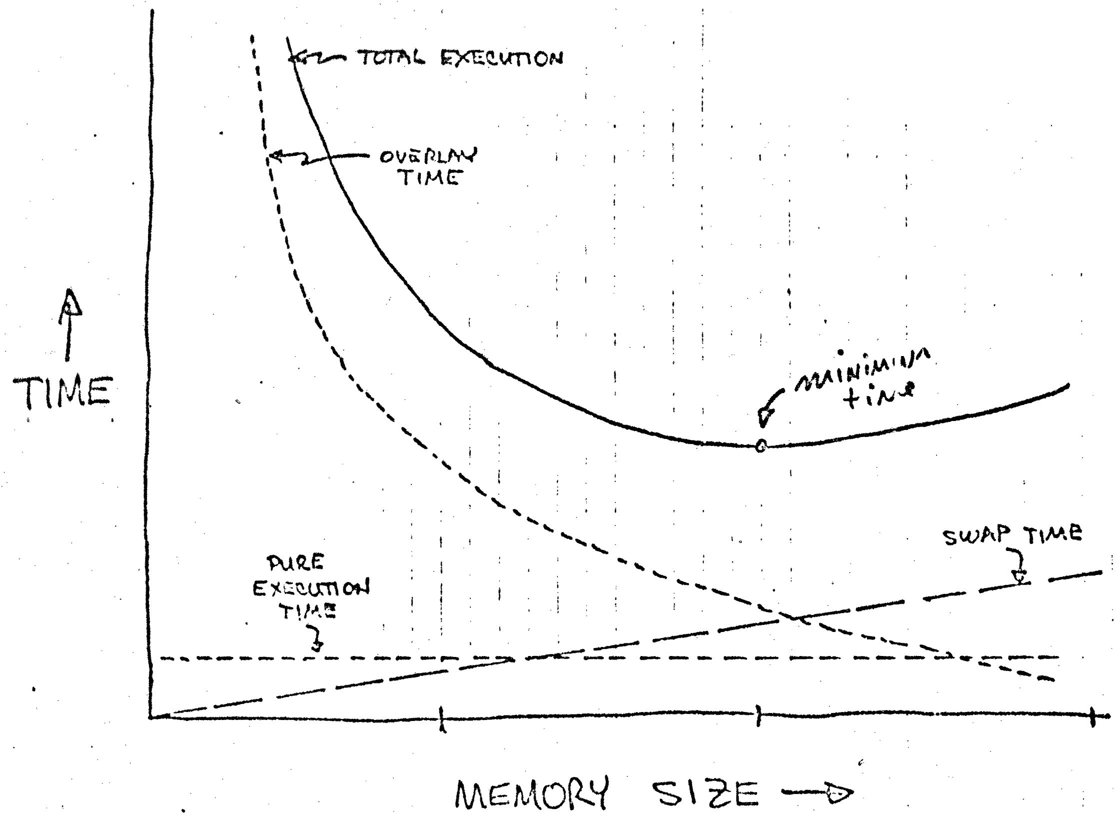
MCP I vs MCP II

		64K	32K	24K	16K
MCP II	COMPILE SAV001	5:24	7:30	∞	∞
	EXECUTE SAV001	0:55	0:55	1:50	∞
MCP I	COMPILE SAV001	—	5:01	4:47	6:30
	EXECUTE SAV001	—	1:32	1:22	1:27

SYSTEM/3 in 12K

compile	4:59
execute	1:45

- It turns out that 12K is enough to run even large programs although thrashing has begun
- Small to medium size programs run quite well.
- In general MCP I will run the same program as MCP II 20% — 30% faster in 8K less memory.



- A curious thing about MCPI is that it runs programs slower in very large memory systems!

- The reason for this becomes clear when you remember that the time needed to do a "SWAP" increases with increasing memory size.

- The time needed to do a swap by the way is about 1/2 sec in 16K to about 2 sec in 64K

EMULATION

WHAT?

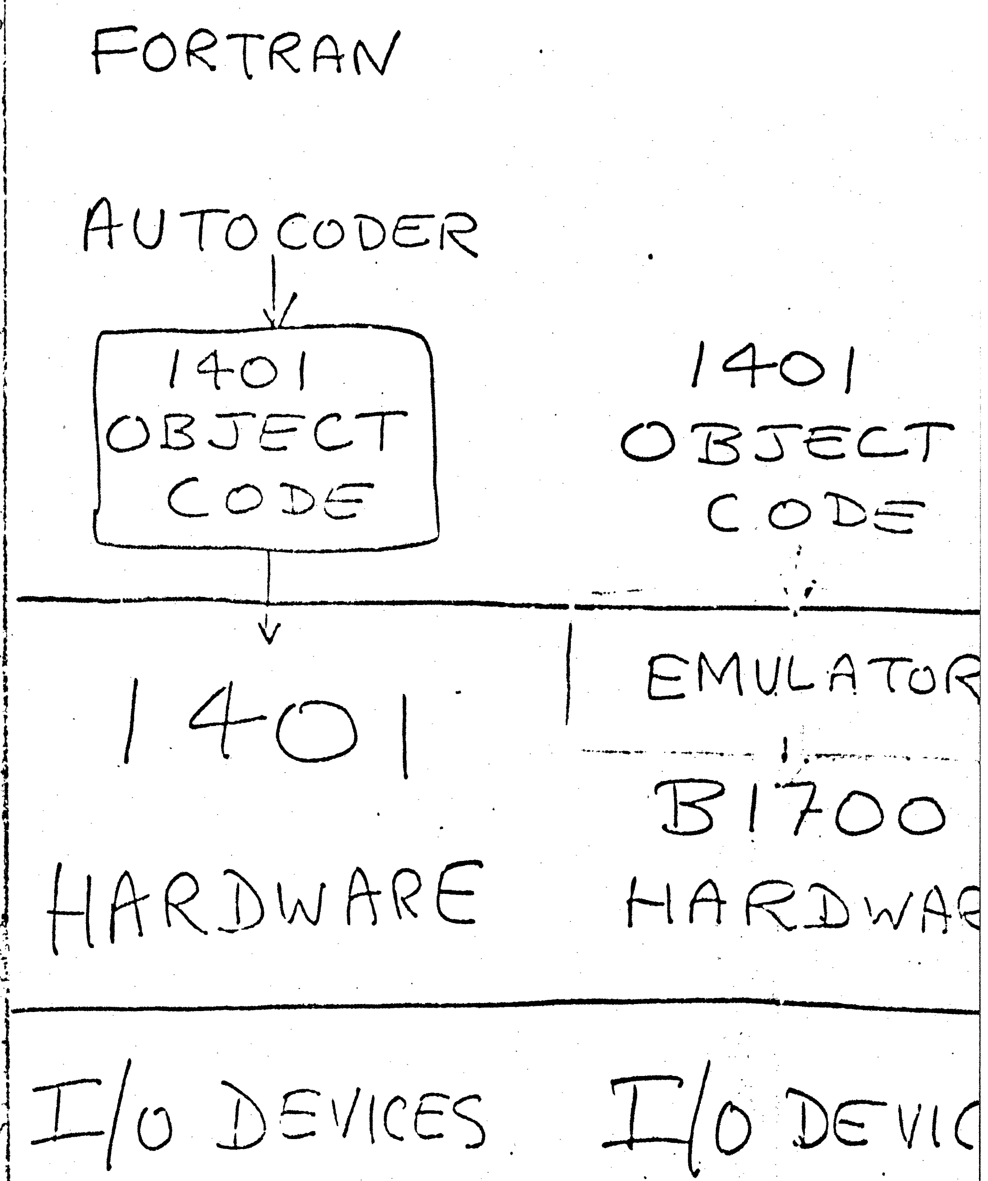
WHY?

HOW

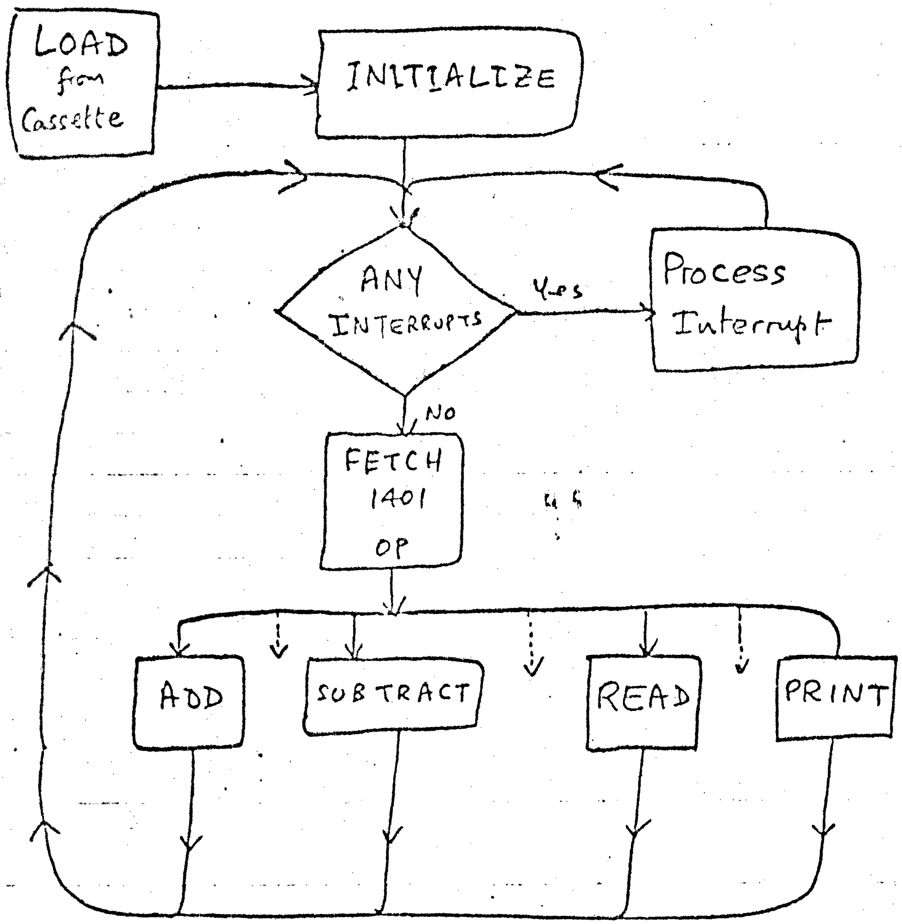
(2A)

WHY ?

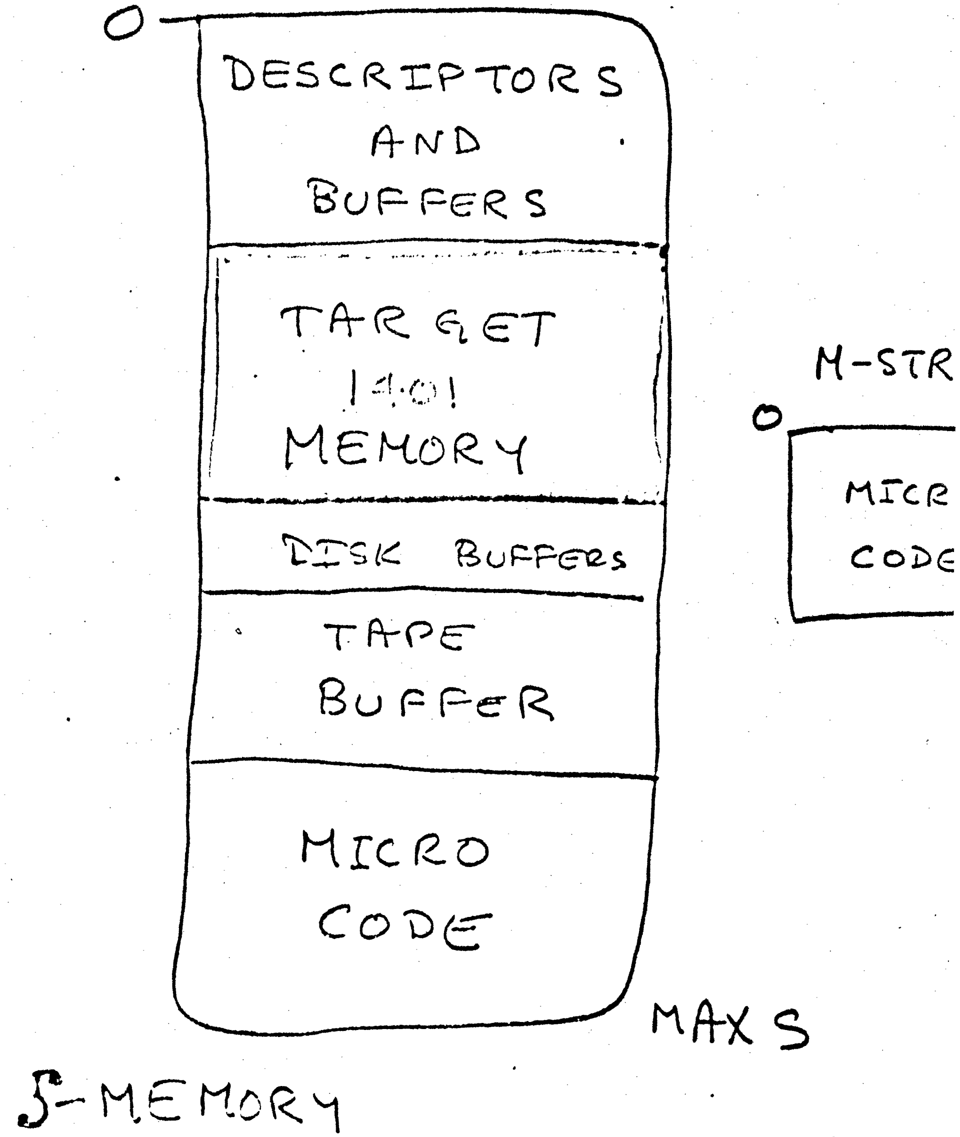
1. NO COPY OF ORIGINAL SOURCE
2. SOURCE INCOMPLETE (due to object deck patching)
3. NO DOCUMENTATION of ORIGINAL
4. ORIGINAL PROGRAMMER NO LONGER AVAILABLE
5. TAKES TOO MUCH TIME
6. TOO EXPENSIVE
7. NOT SUITABLE for REPROGRAMMIN



EMULATOR - FLOW of CONTROL



EMULATOR LAYOUT



IMPROVEMENTS OFFERED

1. FASTER PROCESSOR
2. OVERLAPPED I/O
3. OPERATOR CONTROL
4. Measurement e.g:- count 1401 OPS
5. PORTABILITY of 1401 programs
6. EASE of USE

OPERATIONAL CONVENIENCES

1. NO RED LIGHTS
2. INVISIBLE TRACE
3. INVISIBLE DUMP
4. ALTER/DISPLAY TARGET MEMORY
5. ADDRESS STOP
6. TAPE ASSIGNMENT
7. IGNORE < device >

SIMILARITIES to MCP

(7)

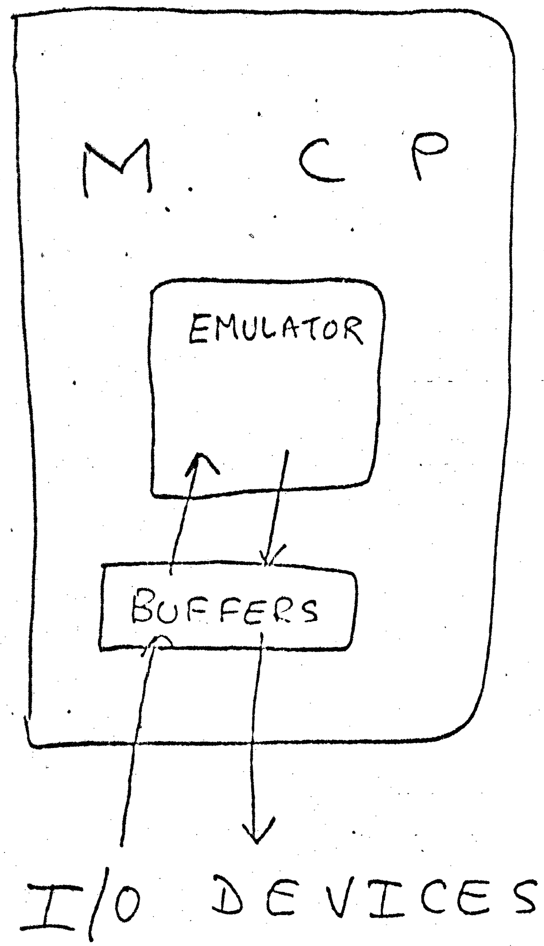
1. I/O HANDLING
2. MAIN LOOP
3. CONTROL DEVICES
4. CONTROL LANGUAGE PROCESSOR

HOW TO USE AN EMULATOR

(8)

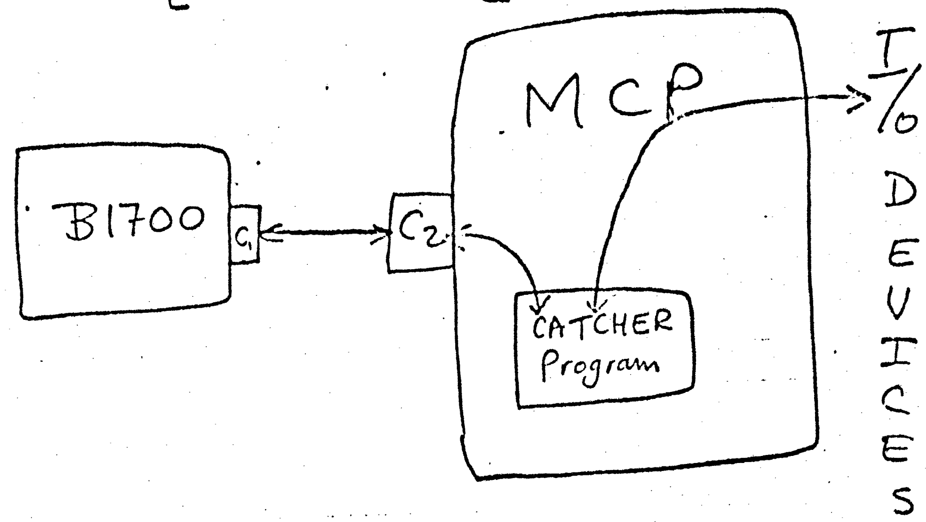
1. LOAD CASSETTE
2. LOAD EMULATOR
3. INSERT 1401 OBJECT PROG.

OPERATION UNDER B1700 MCP



EMULATOR VEHICLE

[E M V]



MEDIUM SYSTEMS COMPUTER

B2500 — B4700

Firmware

WHAT IS FIRMWARE?

* INTERPRETERS
* EMULATORS
* CLEAR START
* I/O DRIVERS

WHAT IS INTERPRETATION?

* SUBROUTINES PERFORMING A FUNCTION INDICATED BY STATEMENTS IN SOME LANGUAGE.

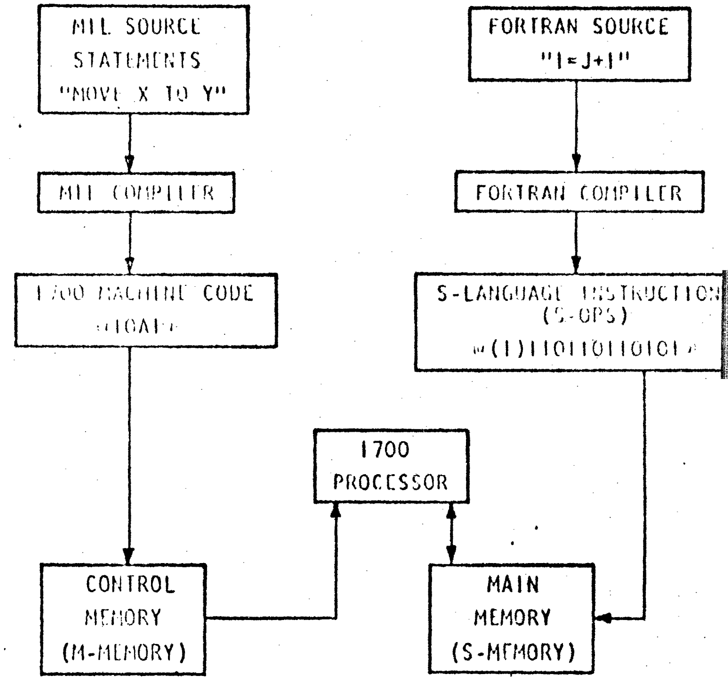
WHY DO WE INTERPRET?

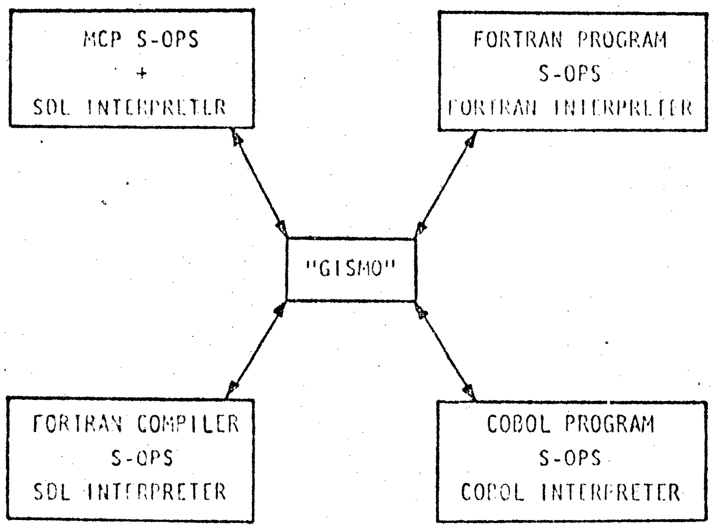
* HARDWARE COST
* FLEXIBILITY

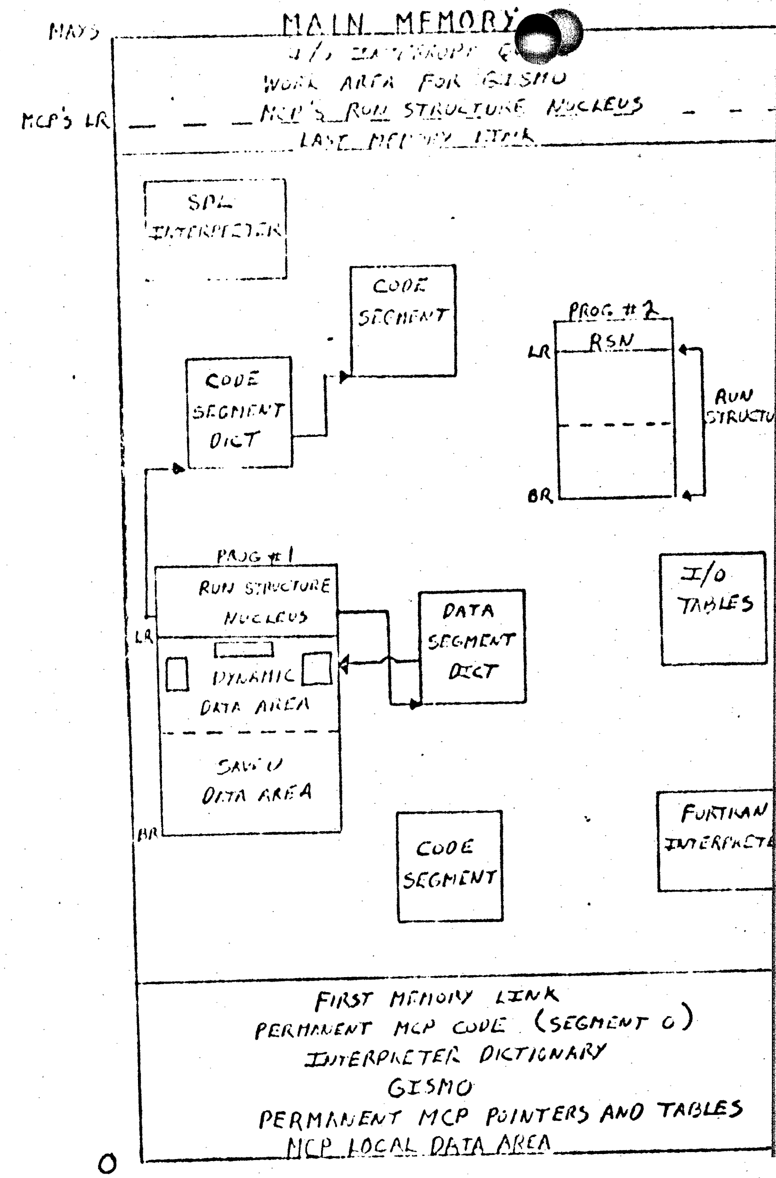
WHAT IS DYNAMIC MICROPROGRAMMING?

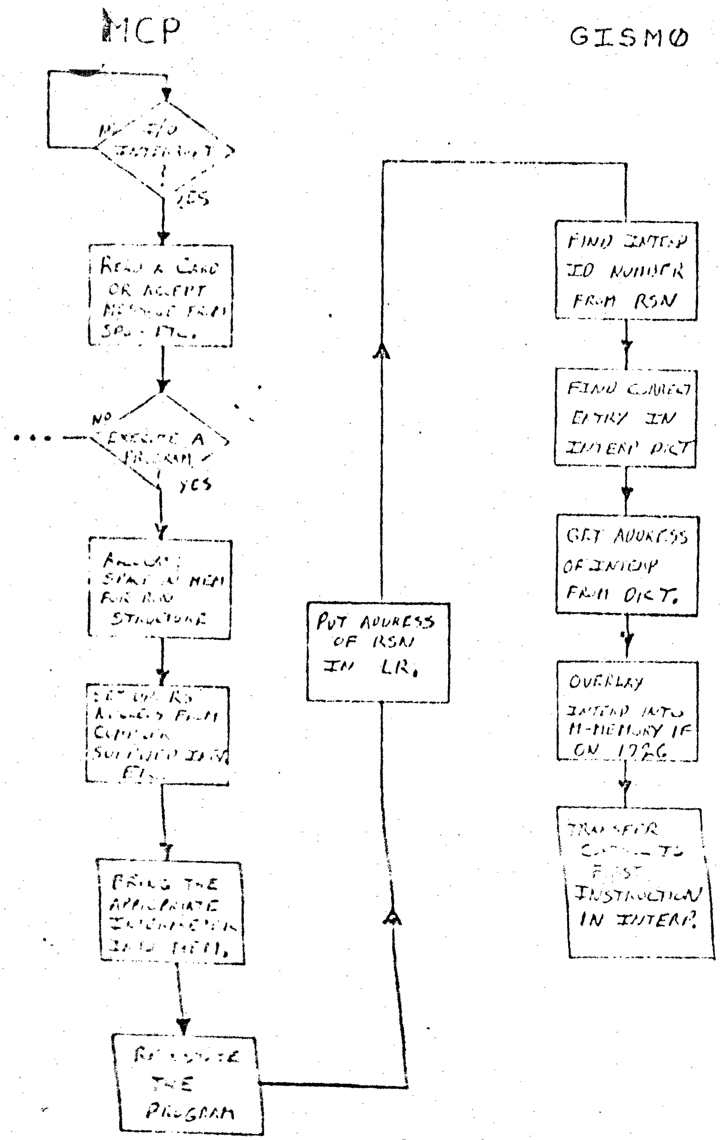
* ALLOWING MULTIPLE INTERPRETERS TO TIME SHARE THE RESOURCES OF THE SYSTEM IN REAL TIME.

HOW IS INTERPRETATION ACCOMPLISHED ON THE 1700?

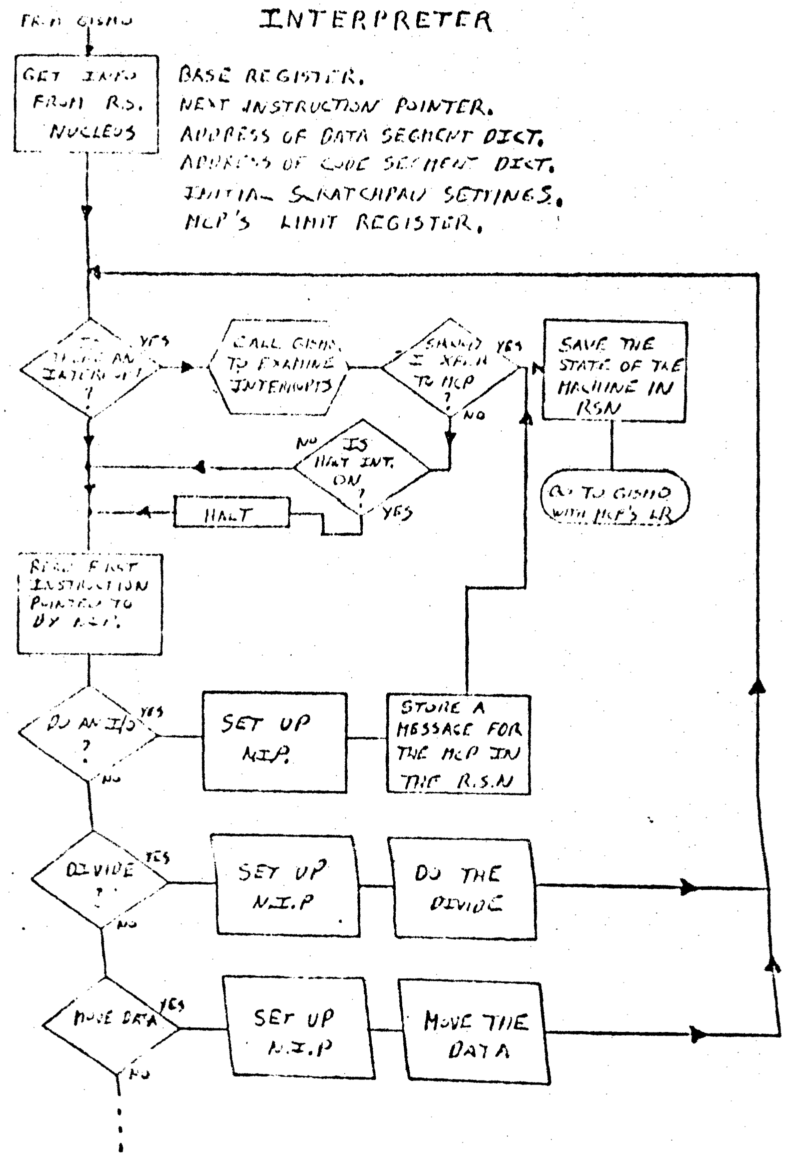



MCP S-OPS
+
SDL INTERPRETER
FORTRAN PROGRAM
S-OPS
FORTRAN INTERPRETER
"GISMO"
FORTRAN COMPILER
S-OPS
SDL INTERPRETER
COBOL PROGRAM
S-OPS
COBOL INTERPRETER

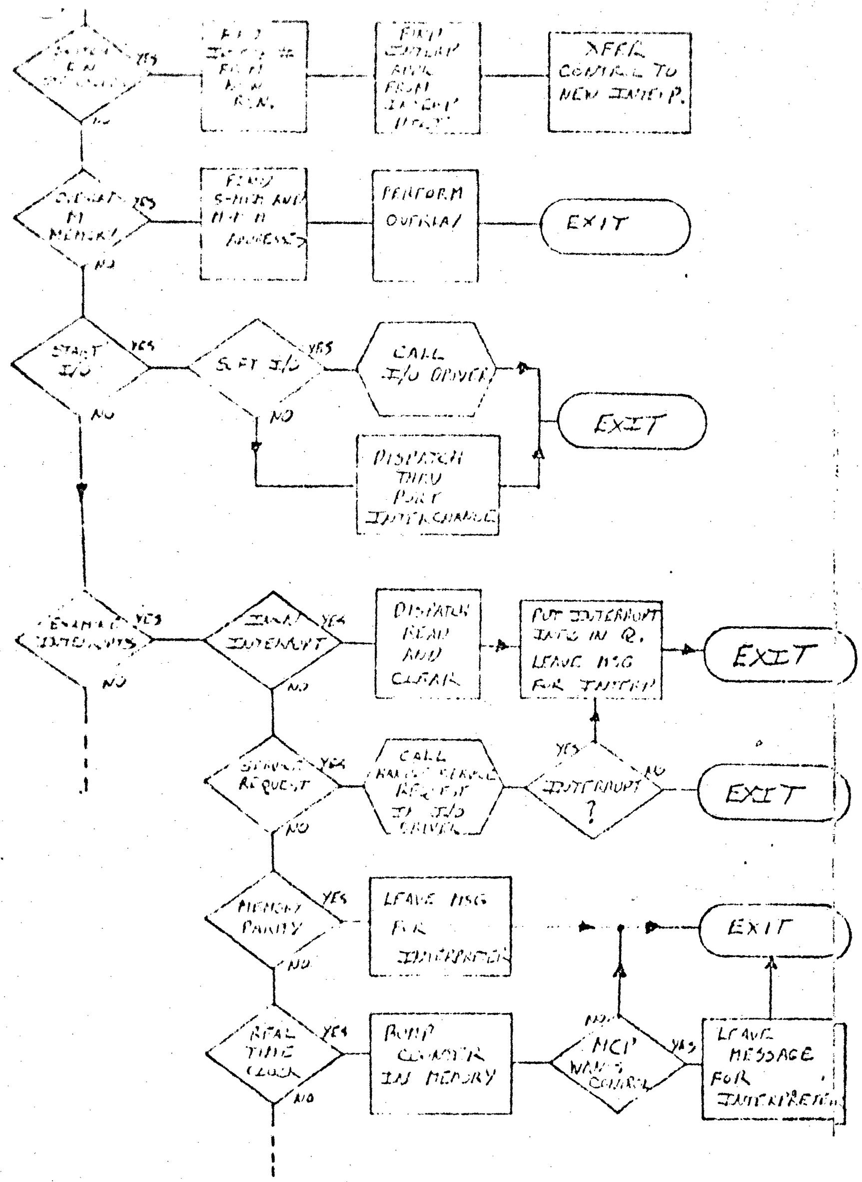

MAIN MEMORY
WORK AREA FOR GISMO
MCP'S LR
MCP'S RUN STRUCTURE NUCLEUS
CODE SEGMENT
CODE SEGMENT DICT
PROG #2
LR
RSN
BR
RUN
PROG #1
RUN STRUCTURE NUCLEUS
LR
DYNAMIC DATA AREA
SAVED DATA AREA
BR
DATA SEGMENT DICT
I/O TABLES
CODE SEGMENT
FIRST MEMORY LINK
PERMANENT MCP CODE (SEGMENT 0)
INTERPRETER DICTIONARY
GISMO
PERMANENT MCP POINTERS AND TABLES
MCP LOCAL DATA AREA
0



RSN = RUN STRUCTURE NUCLEUS



GISMØ


YES
NO
XFER CONTROL TO NEW INTERP.
PERFORM OVERLAY
EXIT
START I/O
CALL I/O DRIVER
DISPATCH THRU PORT INTERCHANGE
EXIT
DISPATCH READ AND CLEAR
EXIT
INTERRUPT ?
EXIT
LEAVE MSG FOR INTERPRETER
MEMORY PARITY
EXIT
REAL TIME CLOCK
BUMP COUNTER IN MEMORY
MCP WANTS CONTROL
LEAVE MESSAGE FOR INTERPRETER

5.1 GISMO INTERFACE

When first given control from GISMO, you may assume:

1. CA ← CB ← BR ← CD ← 0
2. LR, TOPM, MBR, A will be initialized properly
3. TAS (if 26 then TAS else L) ← Interp Segment Dictionary Ptr.

TO CALL GISMO on a 14

1. TAS ← return address
2. A ← address (GISMO)

TO CALL GISMO on a 26

1. TAS ← return address,
MBR.TOPM,
L,
T
2. L ← address (GISMO)
3. T ← 0
4. TRANSFER CONTROL

PRIOR TO CALLING GISMO

1. X ← SWAPPER VALUE
2. L ←, T ← } PARAMETERS TO GISMO (VARIES UPON SWAPPER VALUE)

RETURNING FROM GISMO

1. (ON 26 ONLY) T, L ← TAS
2. T, L = RETURN PARAMETERS

SWAPPER VALUES

VALUE	MEANING	STATE SAVED	GISMO INPUT L	T	GISMO OUTPUT L	T	* ISSUED BY
0	COMMUNICATE ~~CHECK FOR I/Ø~~	X	—	—	—	—	ALL
1	not used						
2	OVERLAY M.MEM		INTERP ID	–	–	–	MCP
3	DISPATCH I/Ø		REF. ADR.	PORT CHAN	–	–	MCP
4	FETCH INTERRUPT		VARIANT*	–	RESULT DESC	PORT CHAN	MCP
5	EXAMINE INTERRUPTS		–	–	SOFT INTERRUPT CODE	–	ALL
6	MICRO MCP RETURN	X	Q. ID	RSN TO BE PUT IN READY.Q.	–	–	M.MCP
7	SEND MSG (I.Q.) TO MCP		MSG ADDR	(12 BITS) HEADER	–	–	M.MCP
8	not used						
9	not used						
10	ENABLE/DISABLE INTERRUPTS		0=E 1=D	—	—	—	HI. PRI. MCP
11	START MCP	X	—	—	-	-	SYSTEM INIT
12	PUT.IN.GISMO.TRACE.TABLE		(8 BIT) KEY	DATA	-	-	ALL
13	not used						
14	COMMUNICATE WITH GISMO		MSG ADR	MSG LENGTH	-	-	ALL

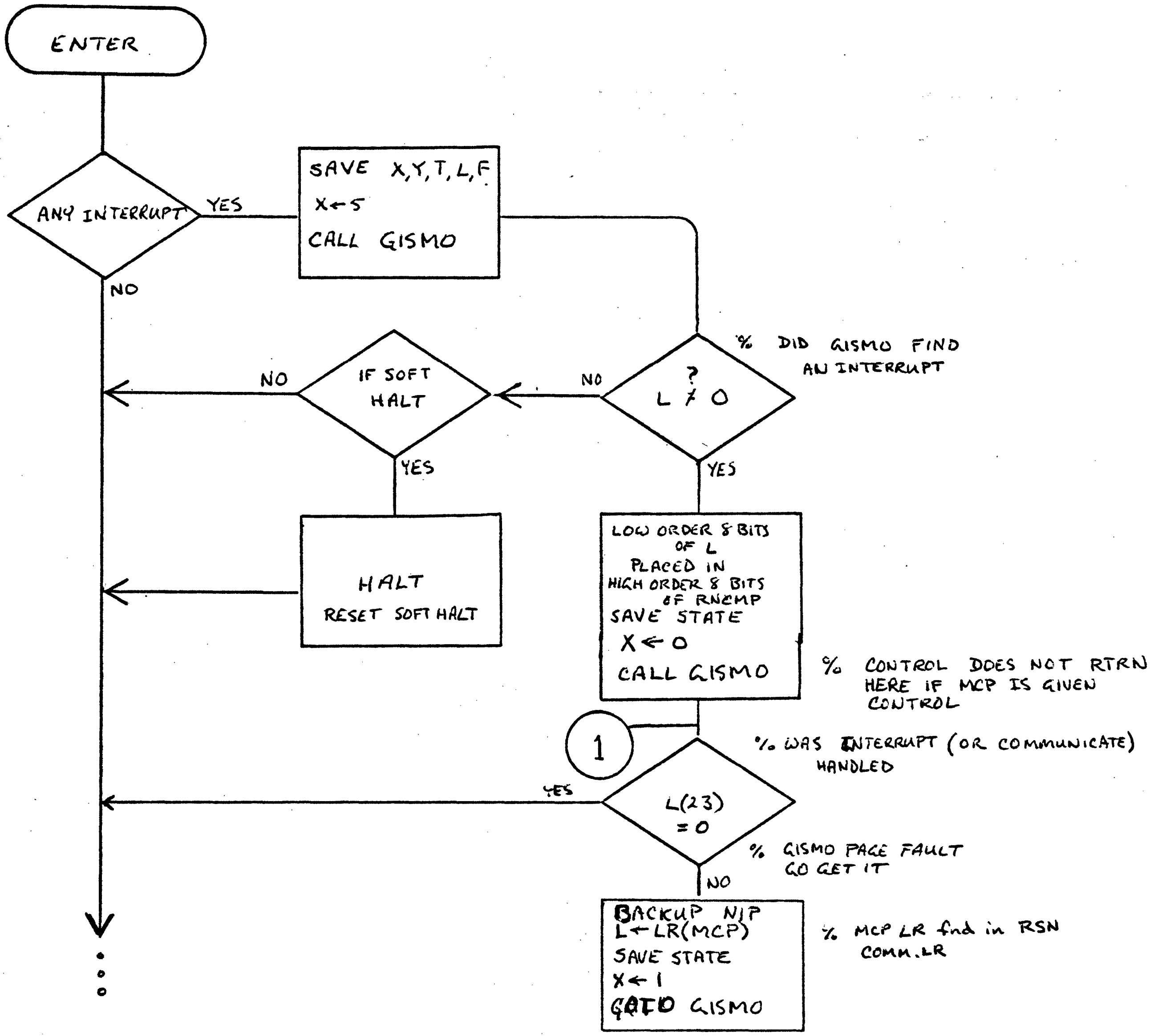
* 0 ⇒ any non hi-pri interrupt in Q
F-F ⇒ any hi-pri interrupt in Q
≠0 ⇒ addr of specific interrupt.

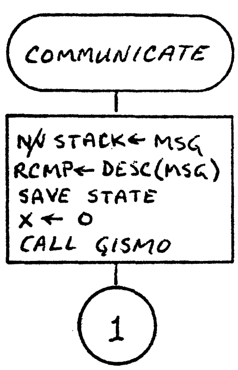
GISMO CODE

GISMO INPUT X	L	T		GISMO OUTPUT L	T
			INITIAL CONTROL FROM GISMO. LR, TOPM, MBR, A, (CA, CB, BR) = 0; TAS ← INTERP.SEG.DIC.PTR (26); L ← INTERP.SEG.DIC.PTR (14); (RA OUT.OF.BNDS, W.OVRRD) ← 0		
0			CHECK FOR I/O	(23) PAGE FAULT	
~~1~~	SWAP ~~LR~~		~~COMMUNICATE (SWAP PROGRAMS)~~	SEE ABOVE	
2	A.REG		OVERLAY *		
3	A.REG A(DSCPTR)	B REG P&CH	DISPATCH.I/O *	RESULT DESC.	
4	A.REG		DISPATCH.READ.&.CLEAR *	RESULT DESC	P&CH
5			CHECK.FOR.INTERRUPTS	SOFT INTERRUPT CODE	
6			DISPATCH.IO.&.WAIT ǂ		
10	0=E 1=D		DISABLE.ENABLE.INTERRUPTS		
12			PUT.GISMO.TRACE.TABLE ǂ		
13	A.REG		DISPATCH.AND.READ *	RESULT DESC	P&CH
14	ABS ADDR OF(MSG)	LENGTH & TYPE OF(MSG)	COMMUNICATE.WITH.GISMO		

* PRIVILEGED INST.

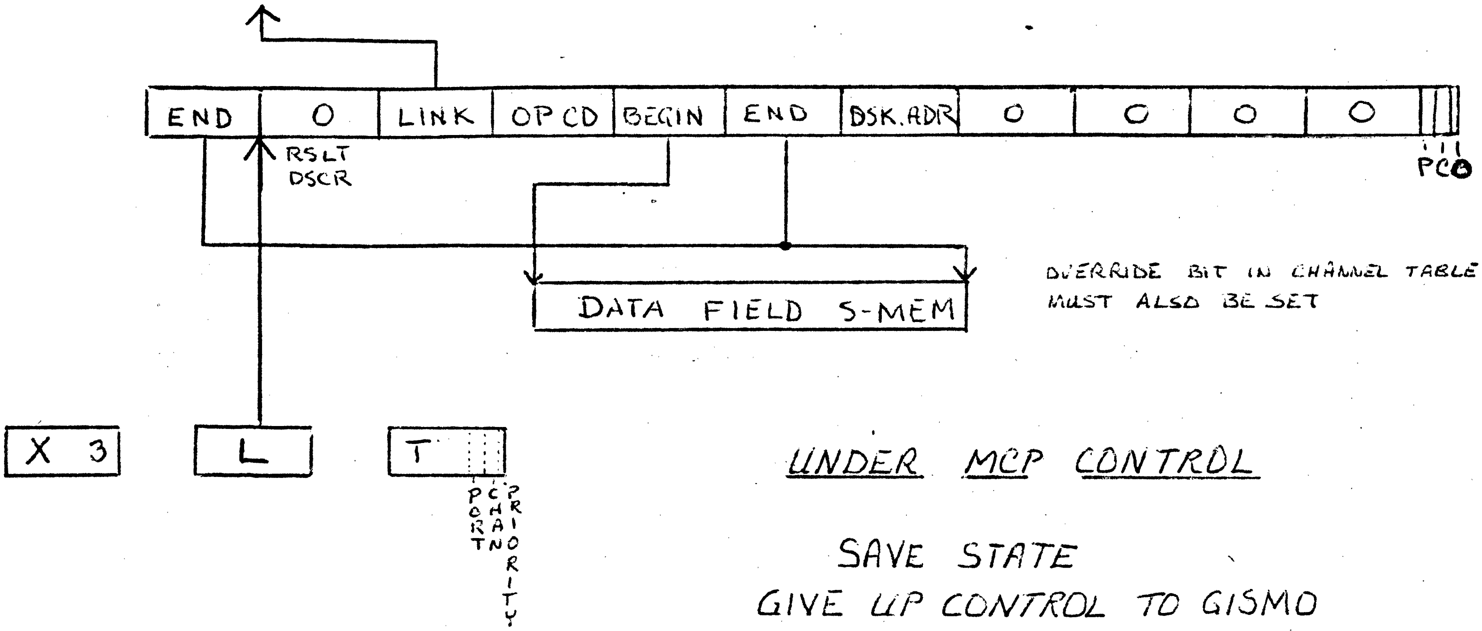
ǂ NOT ISSUED BY SDL/INTERP





GISMO RETURNS

```
CALL GISMO
   ON 1714
        [TAS ← RETURN.ADDRESS]
        A ← ADDR(GISMO)
   ON 1726
        [TAS ← RETURN.ADDRESS]
         TAS ← MBR.TOPM
         TAS ← L  }
                  } CONTENTS DEPENDS ON GISMO CODE
         TAS ← T  }
         L ← ADDR(GISMO)
         T ← 0
         XFER.CONTROL
   RETURN.ADDRESS
         T ← TAS
         L ← TAS
```



UNDER MCP CONTROL

SAVE STATE
GIVE UP CONTROL TO GISMO

READ (DISK)	BEFORE	SEND	AFTER	WRITE	BEFORE	SEND	AFTER
CLR&TEST	1	150003	1	CLR &TEST	1	150003	1
XMIT OP(1)	1	250000	2	XMIT OP(1)	1	250040	2
XMIT OP(2)	2	250000	3	XMIT OP(2)	2	250000	3
XMIT OP(3)	3	250000	4	X OP(3)	3	250000	4
XMIT FA(1)	4	250000	5	X FA(1)	4	250000	5
XMIT FA(2)	5	250000	6	X FA(2)	5	250000	6
XMIT FA(3)	6	250000	7	X FA(3)	6	250000	14
XMIT REF(1)	7	2500XX	8	X DA(X)	14	2500XX	14
XMIT REF(2)	8	2500XX	9	X ETX	14	150006	7
XMIT REF(3)	9	2500XX	10	X REF(1)	7	2500XX	8
RCV REF(1)	11	450000	12	X REF(2)	8	2500XX	9
RCV REF(2)	12	450000	13	X REF(3)	9	2500XX	10
RCV REF(3)	13	450000	15	R REF(1)	18	450000	19
RCV DA(X)	15	450000	15,17	R REF(2)	19	450000	20
RCV ETX	17	450000	21	R REF(3)	20	450000	21
RCV RD(1)	21	450000	22	R RD(1)	21	450000	22
RCV RD(2)	22	450000	23	R RD(2)	22	450000	23
RCV RD(3)	23	450000	1	R RD(3)	23	450000	1

GISMO

I) MESSAGE QUEUES ~~OR~~ INTERRUPT QUEUE

- MAINTAINED BY GISMO'S MESSAGE QUEUES ROUTINES.
- CONTAINS MESSAGES FOR MCP'S ~~NOT~~ WHICH ARE NOT DIRECTLY RELATED TO RS.
  - USUALLY IO COMPLETE MESSAGES.
- AN EVENT ASSOCIATED WITH EACH OF TWO QUEUES.
- MMQ CONTAINS MESSAGES FOR MICRO MCP.
- SMQ CONTAINS MESSAGES FOR SDL MCP.

1) ENTER.MESSAGE (MESSAGE, MESSAGE.Q.ID)
   NORMALLY CALLED BY SOFT.IO, OCCASSIONALLY BY MCP.
   A) ENTERS THE MESSAGE IN THE Q.
   B) CAUSE THE EVENT ASSOCIATED WITH THE Q.

2) REMOVE.MESSAGE
   - FUNCTION WHICH RETURNS "NULL" OR MESSAGE.
   - MCP'S CALL IT; IT ~~RE~~ OPERATES ON THE CALLING MCP'S QUEUE.
   A) IF Q EMPTY THEN RETURN ~~RE~~ NULL.
   B) REMOVE MESSAGE.
   C) IF Q NOW EMPTY ~~THEN~~ THEN RESET EVENT.
   D) RETURN MESSAGE.

II) INTERRUPT HANDLER

- ROUTINE IS CALLED BY INTERP, SCHEDULER OR OTHER MIL PROG WHEN ANY.INTERRUPT FLAG SET IN CP.
- PRIMARY ROUTE TO SOFT.IO ROUTINES.
- RETURNS A VALUE TO CALLER WHICH TELLS CALLER TO EITHER CONTINUE RUNNING OR GIVE UP CP.

STEPS TO DECIDE IF CALLER TO GIVE UP:
   A) IF EITHER MCP IS RUNNING THEN RETURN NO
   B) IF EITHER MCP IS IN READY Q THEN RETURN YES
   C) RETURN NO

NOTE: NOT COVERED ARE TIME AND HIGH PRIORITY INTERRUPTS FOR SORTER.

GISMO (CONTINUED)

III) SOFT.IO, AT COMPLETION OF I/O.
- WILL PUT MESSAGE IN Q ~~IF~~ REQUESTED
BY MCP OR EXCEPTION ON COMPLETION.
-"WHICH Q" SPECIFIED BY BIT IN IO DESC.
A) CAUSE COMPLETION EVENT FOR DESC.

B) IF INTERRUPT REQUESTED OR EXCEPTION THEN
ENTER.MESSAGE (IO COMPLETE,
MMQ OR SMQ DEPENDING ON IOD)

IV STATE SWITCH

- GISMO HAS ALWAYS CONTAINED THE FINAL STEP IN SWITCHING RUN STRUCTURES.
- TO THIS IS ADDED THE SDL MCP'S PROCEDURE "M.REINSTATE" WHOSE FUNCTIONS ARE:
  - MAINTAIN PROCESSOR TIME CLOCK FOR RS
  - WHEN IN CONTENTION MODE ON 1720, DECIDE WHICH ~~[illegible]~~ PAGE TO USE FOR OVERLAY OF THIS INTERP.

RUN STRUCTURE QUEUES.

- EVERY RUN STRUCTURE (WITH MINOR EXCEPTIONS) IN THE MACHINE IS IN ONE OF FOUR QUEUES:

  SDL COMMUNICATE QUEUE (~~SCQ~~ SCQ)
  M MCP COMMUNICATE QUEUE (MCQ)
  READY Q
  WAIT Q

- MOVEMENT FROM ONE QUEUE TO ANOTHER IS DONE BY GISMO'S (MS) RSQUEUE ROUTINES.

- SCQ/MCQ: EACH HAS AN EVENT ASSOCIATED WITH IT. AN RS IS PUT IN THE Q WHEN IT REQUESTS A FUNCTION OF THE MCP.

- READY. Q: CONTAINS ~~JOBS~~ RS-S READY TO RUN, ORDERED BY PRIORITY.

- WAIT. Q: CONTAINS RS-S WHO ARE WAITING FOR SOMETHING:
  - USERS ARE PUT HERE BY AN MCP WHEN MCP CANNOT CONTINUE PROCESSING COMMUNICATE UNTIL SOME RESOURCE IS AVAILABLE.
    FOR EXAMPLE:
    - FOR FILE OPEN, NO DEVICE FILE NOT PRES, NO DISK.
    - FOR LOGICAL I/O, NEXT PHYSICAL IO NOT COMPLETED YET.

  - MCPS GO HERE WHEN THERE IS NOTHING TO DO OR WHEN MCP CANNOT CONTINUE.
    FOR EXAMPLE
    - NO MESSAGES OR COMMUNICATES.
    - WAITING FOR MCP CODE.
    - WAITING FOR I/O DURING SOME FUNCTION (WAITING FOR READ OF FPB DURING FILE OPEN, FOR EXAMPLE).

RUN STRUCTURE QUEUES, (CONTINUED)

V GISMO: RUN STRUCTURE ROUTINES (MS)

1) RS.Q.IN (RS, Q.ID)
- CALLED BY MCP, CAUSE, COMM HANDLER.

A) PUT RS IN Q.
B) IF Q IS MCQ OR SCQ THEN CAUSE EVENT ASSOCIATED WITH Q.

2) RS.Q.OUT (Q-ID)
- A FUNCTION WHICH RETURNS "NULL" OR RS.

A) IF NO RS IN Q RETURN NULL
B) ~~REMO~~ IF Q IS READY Q THEN MAINTAIN ROUND ROBIN FOR EQUAL PRIORITY RS-S.
C) REMOVE RS.
D) IF Q IS MCQ OR SCQ AND NOW EMPTY, RESET EVENT

NOTE RS.Q.OUT (READY.Q) ONLY DONE BY SCHEDULER. RS.Q.OUT (MCQ) ONLY BY M.MCP, RS.Q.OUT (SCQ) ONLY BY S.MCP.
RS.Q.OUT (WAIT.Q) NEVER (SEE CAUSE)

3) CAUSE (EVENT)
- MOVES RS-S FROM WAIT Q TO ONE OF SCQ, MCQ, READY Q. LINKS THRU LIST OF ALL RS-S IS WAIT Q, FINDING IF THEY ARE WAITING FOR THE EVENT. IF THEY ARE THEN DOES RS.Q.IN TO THE RS.NEXT.Q (SEE M.WAIT, HW6.PROG)

V GISMO: RUN STRUCTURE ROUTINES (MS)

4) M.WAIT (CLOCK, EVENT LIST)

- A COMMUNICATE HANDLED BY GISMO.
- MEANS WHEREBY AN MCP GIVES UP CP AND IS PUT IN WAIT Q. WHEN EITHER THE TIME EXPIRES OR AN EVENT IS CAUSED, WILL BE MOVED TO READY.Q.

5) HANG.PROG (RS, RS.NEXT.Q, CLOCK, EVENT LIST)

- MEANS WHEREBY USER RS PUT IN WAIT Q BY AN MCP.
- RS.NEXT.Q SPECIFIES WHICH Q (MCQ, 3CQ, READY.Q) RS IS TO GO INTO NEXT (AT CAUSE TIME).

NOTE OTHER FUNCTIONS OF MS NOT COVERED ~~ARE~~ INCLUDE OVERLAY M MEMORY, "USE" ROUTINE FOR SORTER

VI GISMO: MICRO SCHEDULER

- ALL RS-S GIVE UP CP VIA A COMMUNICATE.
- THE COMMUNICATE HANDLER OF THE MICRO SCHEDULER DISPOSES OF THE RS THEN GOES TO THE SCHEDULE LOOP
- THE SCHEDULE LOOP ~~WAIT~~ IDLES CALLING SOFT.IO UNTIL SOME RS IS IN THE READY.Q

VI GISMO: MICRO SCHEDULER

1) COMMUNICATE HANDLER

A) IF COMMUNICATE HANDLED BY GISMO
(FOR EXAMPLE, M.WAIT) THEN
CALL APPROPRIATE ROUTINE
GO SCHEDULE

B) IF RS WAS INTERRUPTED ~~BY~~
BY GISMO'S INTERRUPT HANDLER THEN
RS.Q.IN (RQ)
GO SCHEDULE

C) IF COMMUNICATE IS HANDLED BY
MICRO MCP (READ, WRITE, SEEK) THEN
RS.Q.IN (MCQ)
ELSE RS.Q.IN (SCQ)
GO SCHEDULE

2) SCHEDULER
- PRIMARY "IDLE" LOOP IN MACHINE

A) IF ANY.INTERRUPT THEN
CALL SOFT.IO

B) IF RS := RS.Q.OUT (RQ) THEN
GO TO STATE SWITCH TO RS

C) GO TO (A)

MICRO MCP

- HANDLES "NORMAL" CASE OF LOGICAL I/O.
- PASSES RS-S TO SDL MCP (THRU SCQ) WHEN EXCEPTIONAL CONDITION ARISES, SUCH AS NEW DISK AREA TO BE ALLOCATED.
- A SIMPLIFIED FLOW OF UNBLOCKED SERIAL I/O:

```
A) IF  RS:= RS.Q.OUT (MCQ) NEQ NULL
       CALL   COMM
       GO  A
B) IF  MESS := REMOVE.MESSAGE NEQ NULL
       CALL   MESS
       GO  A
C) M.WAIT (MCQ, MMQ)
   GO  A

COMM
   A) FWD FIB AND SET UP STATE
   B) IF  CURRENT DESC NOT COMPLETE
            REQUEST INTERRUPT
            HANG.PROG (RS, MCQ, 0, DESC EVENT)
            EXIT.
   C) BLOCK/DEBLOCK
   D) INITIATE I/O
   E) EXIT

MESS
   A) IF  EXCEPTION THEN
            ENTER.MESSAGE IN SMQ
            EXIT
   B) CAUSE (DESC EVENT)
   C) EXIT
```

OPERATING INSTRUCTIONS

FOR

GISMO TRACE

JUNE 7, 1974

GISMO AND INTERPRETER TRACE TABLE

THE 4.1 INITIALIZER AND GISMO CAN BE MADE TO ALLOCATE AND MAINTAIN AN AREA IN MEMORY TO BE USED FOR "TRACING" VARIOUS THINGS. THE INITIALIZER WILL ALLOCATE AN AREA ABOVE THE MCP-S LIMIT REGISTER WHEN REQUESTED AT CLEAR START TIME. GISMO WILL MAKE ENTRIES IN THIS TABLE IN AN "END-AROUND" FASHION. TWO KINDS OF ENTRIES CAN BE MADE: ENTRIES GENERATED BY GISMO, WHICH ARE INTENDED TO GIVE A TRACE OF THE LAST "N" THINGS DONE BY GISMO, AND ENTRIES GENERATED BY INTERPRETERS, WHICH MAY REFLECT INTERPRETER OR S-PROGRAM INFORMATION. "N" IS DETERMINED BY THE SIZE OF THE AREA ALLOCATED, AS DESCRIBED BELOW.

TWO OPTIONAL SEGMENTS EXIST IN GISMO FOR THIS FUNCTION. THE FIRST CONTAINS THE CODE NECESSARY TO MAKE ENTRIES GENERATED EXTERNAL TO GISMO, AND IS AVAILABLE IN THE STANDARD GISMO. THE SECOND CONTAINS THE CODE TO TRACE GISMO ITSELF, AND IS AVAILABLE ONLY IN THE DEBUG VERSION.

AT CLEAR/START TIME, WHEN SWITCHING FROM TAPE TO RUN MODE, THE BR REGISTER IS RESERVED FOR INFORMATION PERTINENT TO TRACING. BR IS USED AS FOLLOWS (BITS NUMBERED LEFT TO RIGHT, 0-23):

BITS	CONTAINS
0-7	THE MOST SIGNIFICANT 8 BITS OF A 16 BIT VALUE (LEAST SIG 8 ARE 0) WHICH SPECIFIES THE SIZE OF THE TRACE TABLE TO BE ALLOCATED.
8-23	A MASK USED WHILE TRACING GISMO THAT SPECIFIES WHAT I/O CHANNELS ARE TO BE TRACED: BIT 8 SET SPECIFIES CHANNEL 0, BIT 9 IS CHANNEL 1 AND SO ON. BIT 23 SPECIFYS NON-DRIVER FUNCTIONS OF GISMO.

NOTE THAT THE TABLE SIZE IS MODULAR IN 256 BIT (8 ENTRY) HUNKS, UP TO A MAXIMUM OF 2047 ENTRIES (65K BITS, 8K BYTES).

SYSTEM/INITIALIZER WILL ALLOCATE SPACE FOR THE TABLE AND WILL PLACE THE TABLE'S ABSOLUTE ADDRESS IN HINTS AT @21F@. THE TABLE AREA IS INITIALIZED AS FOLLOWS:

```
        ------------------------------------->
        |   16   | 8 | 16 | 8 | 32   | 32 |
        ------------------------------------->
        012...15|  |     |    |      |-----|--ENTRY
           |    |  |     |    |-----UNUSED
    CHANNEL--|  |  |     |
     MASK       |  |     |---CURRENT INDEX.POINTS TO
                |  |         THE OLDEST ENTRY
    HINTS POINTS-|  |
       HERE        |---TABLE SIZE
                       (FROM BR)
```

NOTE THAT THE FIRST 32 BIT ENTRY IS USED FOR MAINTENANCE INFORMATION, AND IS NEVER OVERWRITTEN. THE FIRST ENTRY IS MADE AT BASE+32. SUCCESSIVE ENTRIES ARE MADE UNTIL THE END OF THE TABLE IS REACHED; THEN WRAP-AROUND TO BASE+32 OCCURS.

THE MCPII DUMP ANALYZER KNOWS ABOUT THIS TABLE AND IF IT IS PRESENT, IT IS PRINTED IN SORTED ORDER, WITH THE FIRST ENTRY PRINTED BEING THE OLDEST.

WHEN BR IS NOT LOADED DURING CLEAR/START, THE INITIALIZER WILL DISCARD THE OPTIONAL SEGMENTS. WHEN ONLY THE TABLE SIZE PORTION OF BR IS LOADED, THE SEGMENT FOR EXTERNAL ENTRIES IS KEPT AND THE GISMO TRACE SEGMENT (IF A DEBUG VERSION) IS DISCARDED. WHEN BOTH THE TABLE SIZE AND THE CHANNEL MASK ARE LOADED, BOTH ARE KEPT (THIS WILL RESULT IN AN ERROR HALT IF ITS NOT A DEBUG VERSION OF GISMO).

FOR MAKING EXTERNAL ENTRYS FROM INTERPRETERS, THIS PROCEDURE IS USED:

-THE X REGISTER IS LOADED WITH A "SWAPPER" CODE OF 12 (@C@)

-THE LE AND LF REGISTERS ARE LOADED WITH AN 8 BIT "KEY"

-THE T REGISTER IS LOADED WITH 24 BITS OF STUFF.

-IF ITS A 1726, L AND T THEN GO IN THE STACK.

DURING THE ENTRY ROUTINE, X, Y, FA, FB ARE CLUBBERED AND CP IS SET TO 24. T AND L ARE NOT CLOBBERED

EACH 32 BIT ENTRY SHOULD BE THOUGHT OF AS CONSISTING OF AN 8 BIT KEY AND 24 DATA BITS. DUMP ANALYZER PRINTS THE ENTRIES THIS WAY. CONSEQUENTLY THE VALUES IN L HAVE BEEN (ARBITRARILY) RESERVED:

01-1F	MCP I
20-3F	MCP II
40-4F	SDL INTERP
50-7F	OTHER INTERPS
80-BF	OTHER (?)
C0-FF	GISMO.

TRACING GISMO

FOR THE DEBUG VERSION OF GISMO, CODE HAS BEEN ADDED AT KEY POINTS TO MAKE ENTRIES IN THE TRACE TABLE. WHEN THIS CODE IS DISCARDED BY THE INITIALIZER (WHEN THE CHANNEL MASK IN BR IS ZERO) THE PENALTY IN TIME IS NEGLIGIBLE. WHEN IT IS KEPT, IT CAUSES GISMO TO BE BIGGER (BY APPROX 350 BYTES) AND TO RUN SOMEWHAT SLOWER (3-8%).

THIS CODE IS KEPT WHEN THE CHANNEL MASK PART OF BR IS SET NEQ ZERO. THIS MASK SPECIFIES WHAT CHANNELS ARE TO BE TRACED, 0 THRU 14 (SEE ABOVE). THE RIGHT MOST BIT CONTROLS NON-DRIVER GISMO ENTRIES, SUCH AS FOR SWITCH RUN STRUCTURES OR ENHANCED I/O.

WHEN THE GISMO TRACE SEGMENT IS KEPT, THE INTERP.TRACE SEGMENT IS RETAINED AS WELL, SINCE THE SAME TABLE ENTER ROUTINES ARE USED.

THE CURRENT "KEY" VALUES AND DATA ENTRIES FOR GISMO ARE DESCRIBED BELOW. SINCE THIS VERSION OF GISMO IS FOR INTERNAL DEBUGGING ONLY, THE "KEY" VALUES AND DATA MAY CHANGE IF REQUIRED. THE FOLLOWING LIST IS CURRENT AS OF JUNE 5, 1974.

"KEY" VALUES

C0 THRU FF ARE RESERVED FOR GISMO. THE KEY IS BROKEN INTO TWO FOUR BIT FIELDS WHERE THE FIRST FOUR BITS TAKE ON THESE VALUES:

- C USED FOR TEMPORARY "ONE SHOT" ENTRIES.
- D DRIVER ENTRIES, MORE OR LESS PERMANENT.
- E NON-DRIVER ENTRIES, MORE OR LESS PERMANENT.
- F "DISPATCH"S.

THE NON-DRIVER ENTRIES, E0-EF

E0 SWITCH.RUN.STRUCTURES

DATA IS THE LIMIT REGISTER WE ARE SWITCHING TO.

THIS ENTRY WILL SHOW WHAT PROCESS WAS RUNNING, WHERE

ENHANCED.IO IS CONSIDERED PART OF SOME USER PROCESS. YOU CAN TELL, FOR EXAMPLE, WHETHER ENHANCED.IO OR THE MCP ISSUED SOME PARTICULAR DISPATCH BY WORKING BACK FROM THE DISPATCH ENTRY TO THE PREVIOUS E0 ENTRY (ALSO, SEE KEY E5, E7)

E1 OVERLAY.M.MEMORY

DATA IS S.MEMORY ADDRESS FOR THE OVERLAY.

E2 EXAMINE.INTERRUPTS
DATA IS:

LEFTMOST 12 BITS:	ARE THE POINTERS FOR THE INTERUPT QUEUE
RIGHTMOST 12 BITS:	ARE THE VALUE TO BE RETURNED TO THE INTERPRETER

NOTE: THIS ENTRY IS NOT USUALLY MADE. IT CAN ONLY BE "TURNED ON" VIA PATCHING THE OBJECT CODE.

E3 DISPATCH.READ.AND.CLEAR

DATA IS REFERENCE ADDRESS (NOT REF.ADR + 24). ENTRY IS MADE ONLY IF AN INTERRUPT WAS IN THE QUEUE WHEN DISPATCH.READ.AND.CLEAR WAS CALLED.

NOTE: CURRENTLY DISPATCH.READ (BUT DON'T CLEAR) SHOWS UP AS AN E3 KEY. THIS WILL BE CHANGED TO A KEY E8 SOMETIME IN THE FUTURE.

E4 A NON-COMMUNICATE REQUEST MADE TO THE MCP.

DATA IS THE 24 BITS AT THE PROGRAM'S LIMIT REGISTER

E5 A COMMUNICATE.
DATA IS :

LEFT 8 BITS:	ARE MIX NUMBER
NEXT 8 BITS:	ARE FILE NUMBER
RIGHT 8 BITS:	ARE COMMUNICATE VERB.

THIS ENTRY IS FOR ALL COMMUNICATES. IF IT IS FOLLOWED BY AN E0 KEY, THEN IT IS A COMMUNICATE FOR THE MCP, OR AN I/O REQUEST WHICH ENHANCED I/O COULD NOT HANDLE. IF IT IS AN IO REQUEST WHICH ENHANCED I/O CAN HANDLE, THEN A E7 KEY WILL FOLLOW. THERE MAY BE OTHER ENTRIES BETWEEN THE E5 AND THE E0 OR

E7 (FOR EXAMPLE, DRIVER ENTRIES). YOU CAN TELL WHO ISSUED THE DISPATCH, MCP OR ENHANCED I/O. FOR USER I/O'S BY LOOKING FOR THE PREVIOUS E KEYS.

E6 NOT USED PRESENTLY TIMER INT. - VALUE IS TIMER CONTENTS

E7 ENHANCED I/O IS RETURNING TO USER, HAVING SUCCESSFULLY COMPLETED AN I/O REQUEST.DATA IS USER'S LIMIT REGISTER.

E8 RESERVED (SEE E3) DISPATCH.READ.AND.DONT.CLEAR

E9-EF NOT USED PRESENTLY

BEFORE LOOKING AT DRIVER ENTRIES, LET'S LOOK AT KEY F, WHICH MAKES USE OF THE KEY VALUE IN A DIFFERENT WAY

F0-FF

THIS ENTRY SHOWS A "DISPATCH" FROM EITHER THE MCP, ENHANCED I/O, OR A PSEUDO DISPATCH GENERATED BY A STATUS COUNT ONE SERVICE REQUEST FROM DCC-2 OR DISKPACK. THE SECOND FOUR BIT FIELD IN THE KEY SHOWS THE CHANNEL NUMBER.

THE 24 DATA BITS ARE THE FIRST 24 BITS IN THE CORRESPONDING CHANNEL TABLE ENTRY, PRIOR TO ANY CHANGES CAUSED BY THIS DISPATCH.

THIS ENTRY DOES NOT OCCUR FOR "TIMER" DISPATCHES.

AN ENTRY CAUSED BY A STATUS COUNT ONE SERVICE REQUEST WILL IMMEDIATELY AFTER THE ENTRY FOR THE SERVICE REQUEST (SEE BELOW, KEYS D6 AND D8). THE OTHER TWO WILL NOT OCCUR IMMEDIATELY AFTER A SERVICE REQUEST ENTRY.

YOU CAN'T TELL IF A DISPATCH WAS FROM THE MCP OR ENHANCED.IO UNLESS THE NON-DRIVER ENTRIES ARE PRESENT (SEE ABOVE, KEYS E0 AND E7).

IF THE CHANNEL TABLE ENTRY SHOWS THE BUSY BIT (BIT 0) SET AND THE OVERRIDE BIT (BIT 4) RESET, THEN THE I/O WILL NOT BE INITIATED TO THE CONTROL AT THIS TIME. IF THE BUSY BIT IS RESET OR THE OVERRIDE BIT IS SET THEN THE I/O WILL BE INITTIATED AND THE NEXT ENTRY SHOULD BE A D3.

THE DRIVER ENTRIES ARE:

D0-D1 NOT USED

D2 DATA TRANSFER.

THIS ENTRY IS MADE ANYTIME THE DATA TRANSFER ROUTINE IS CALLED, AND AT ITS EXIT THE END.ADR IS NEQ TO THE A.ADR (B.ADR FOR REVERSE). THE DATA IS E.ADR MINUS A.ADR (B.ADR MINUS E.ADR FOR REVERSE).

THE LAST D2 ENTRY FOR ANY I/O APPEARS TWICE IF THE CONTROL IS THE TYPE WHICH RETURNS TO STATUS 10 BETWEEN THE LAST DATA TRANSFER AND THE RESULT DESCRIPTOR (DISKPACK IS ONE OF THESE).

D3 AT I/O INITIATE TO THE CONTROL. DATA IS THE REFERENCE ADDRESS SENT TO THE CONTROL.

NOTE ENTRIES D3, D4, AND D5 ARE ALL MADE BY THE DRIVER AT THE SAME TIME, WHICH IS AFTER THE OP, FILE ADR, FIRST DATA BUFFER (IF ANY) AND THE REF.ADR HAVE BEEN SENT TO THE CONTROL. THE CONTROL WILL HAVE GONE TO 10 (OR BEYOND) BY THIS TIME. BECAUSE OF THE WAY IT IS CODED, IF DATA IS SENT BETWEEN STATUS 6 (END FILE.ADR) AND STATUS 7 (BEGIN REF.ADR) THE D2 ENTRY WILL APPEAR AFTER THE D3, D4 AND D5 ENTRIES. ALSO, NOTE THAT THE ENTRIES ARE NOT IN THE SAME ORDER AS THEY ARE SENT TO THE CONTROL, WHICH IS OP, FILE.ADR, REF.ADR, BUT GO REF.ADR (D3), OP (D4) AND FILE.ADR (D5).

AN OBJECT CODE PATCH CAN BE EASILY MADE WHICH SUPPRESSES D3, D4, AND D5 ENTRIES FOR TEST, TEST AND WAIT, PAUSE.

D3, D4, AND D5 ARE NOT ENTERED FOR A STOP OP.

IF A D3 IMMEDIATELY FOLLOWS A D9 ENTRY OR A D6-DB ENTRY, THE INITIATE WAS FOR A LINKED OP.

D4 THE 24 BIT OP CODE SENT TO THE CONTROL. IT MAY BE DIFFERENT THAN THE OP IN THE I/O DESCRIPTOR; FOR EXAMPLE, DM SEARCH OP WILL BE ENTERED AS DISK READ. (SEE D3)

D5 THE FILE ADDRESS (SEE D3)

D6 TRANSFER TO DRIVER FOR SERVICE REQUEST
DATA IS:

LEFTMOST 4 BITS:	CHANNEL NUMBER TO BE SERVICED NEXT
RIGHTMOST 20 BITS:	THE MASK RETURNED BY TEST SERVICE REQUEST

THIS ENTRY IS ONLY MADE IF A CHANNEL THAT IS BEING TRACED HAS ITS BIT SET IN THE MASK.

SERVICE REQUEST ENTRIES USE TWO KEYS, D6 AND D8. D6 IS USED WHEN THE SERVICE REQUEST WAS DETECTED OUTSIDE THE DRIVER; A D6 KEY, THEN, SHOWS AN ENTRY TO THE DRIVER. D8 IS USED WHEN THE DRIVER HAS FINISHED SERVICING A CHANNEL AND DISCOVERS THAT SERVICE REQUEST IS STILL SET; A D8 KEY SHOWS AN "ITERATION".

D7 POCKET SELECT INFORMATION HAS JUST BEEN SENT TO AND ACKNOWLEDGED BY THE READER.SORTER.

DATA IS :

LEFTMOST 8 BITS:	POCKET VALUE SENT TO CONTROL
RIGHTMOST 16 BITS:	INFO SENT BACK FROM CONTROL AFTER POCKET SELECT.

D8 A RESULT DESCRIPTOR HAS JUST BEEN RECEIVED FROM THE CONTROL. THE DATA IS THE REFFRENCE ADDRESS. THIS ENTRY IS MADE AT THE SAME TIME AS A D9 IS MADE, WHICH SHOWS THE RESULT DESCRIPTOR.

THE ENTRIES ARE MADE PRIOR TO THE DETECTION OF A SECOND.OP.COMPLETE OFF CONDITION. AN EXCEPTION TO THIS IS THAT THE TWO ENTRIES ARE NOT MADE AT THE TERMINATION OF ANY PAUSE OP.

AN OBJECT CODE PATCH CAN BE MADE TO INHIBIT ENTRIES D8 AND D9 FOR RESULT DESCRIPTORS WHICH HAVE THE SECOND.OP.COMPLETE BIT OFF.

D8 AND D9 ARE ENTERED FOR A STOP OP (ALTHOUGH D3, D4

AND D5 ARE NOT).

IF A D9 IS IMMEDIATELY FOLLOWED BY A D3, IT SHOWS THAT LINKING TOOK PLACE AND AN UNLOCKED DESCRIPTOR WAS FOUND. IF A D9 IS FOLLOWED BY SOMETHING ELSE (D8, D6, DB, FX, EX ARE ALL POSSIBLE) IT SHOWS THAT EITHER LINKING WAS NOT DONE OR THERE WERE NO UNLOCKED DESCRIPTORS.

D9 A RESULT DESCRIPTOR (SEE D8)

DA DRIVE THRU GAP PERMISSION WAS JUST SENT TO THE CONTROL

DATA IS THE LINKED I/O DESCRIPTOR'S OP CODE.

DB SERVICE REQUEST DETECTED IN DRIVER (SEE D6)

TABLE OF CONTENTS:

ALPHABETIC INDEX:

G I S M O   T R A C E   A R E A

FILE / LOADER, RUNNING NORMALLY

→ CHANNEL 2: 96 COL
→ CHANNEL 8: PACK
→ CHANNEL 15: NON-DRIVER STUFF

CHANNEL MASK: 0010000010000001
TABLE SIZE: 32512 BITS
CURRENT POINTER ADDRESS: 0F0420

TAG	DATA	TAG	DATA	TAG	DATA	TAG	DATA	TAG	DATA	TAG	DATA	TAG	DATA	TAG	DATA
R.A D3	0ED5B0	OP D4	900003	F.A D5	000000	SR DB	800120	R.A D8	0ED5B0	RD D9	E2801E	LINKS R.A D3	0EED55	OP D4	400000
F.A D5	000B60	SR DB	800100	DATA D2	0005A0	END EIO E7	0F5503	BEGIN EIO E5	010001	TIMER E6	09A807	SR D6	800100	DATA (NO) D2	0005A0
R.A D8	0EED55	RD D9	800080 NO LINK	MCP E0	0F7080	DISP.R E3	0EC08C	TIMER E6	09A808	RA D3	0ED1D8	OP D4	900001	FA D5	000000
SR DB	800100	RA D8	0ED1D8	RD D9	E2801E LINK	RA D3	0ED3C4	OP D4	900002	FA D5	000000	SR DB	800100	RA D8	0ED3C4
RD D9	E2801E LINK	RA D3	0ED5B0	OP D4	900003	D5	000000	DB	800120	D8	0ED5B0	D9	E2801E	SR D6	200004
DATA D2	000280	RA D8	0F0300	RD D9	800080 LINK	D3	0F0690	D4	078000	D5	FFFFFF	DISP.R E3	0F0300	DISP F2	C10000
USER E0	0F5503	BEGIN EIO E5	010001	E6	09A809	D3	0ED1D8	D4	900001	D5	000000	DB	800100	D8	0ED1D8
D9	E2801E	D3	0ED3C4	D4	900002	D5	000000	DB	800120	D8	0ED3C4	D9	E2801E	D3	0ED5B0
D4	900003	D5	000000	DB	800120	D8	0ED5B0	D9	E2801E	D6	200004	D2	000280	D8	0F0690
D9	800080	D3	0F0A20	D4	078000	D5	FFFFFF	F2	C10000	END EIO E7	0F5503	E6	09A80A	D3	0ED1D8
D4	900001	D5	000000	DB	800100	D8	0ED1D8	D9	E2801E	D3	0ED3C4	D4	900002	D5	000000
DB	800100	D8	0ED3C4	D9	E2801E	D3	0ED5B0	D4	900003	D5	000000	DB	800100	D8	0ED5B0
D9	E2801E	E5	010001	E6	09A808	D3	0ED1D8	D4	900001	D5	000000	DB	800100	D8	0ED1D8
D9	E2801E	D3	0ED3C4	D4	900002	D5	000000	DB	800100	D8	0ED3C4	D9	E2801E	D3	0ED5B0
D4	900003	D5	000000	DB	800120	D8	0ED5B0	D9	E2801E	D6	200004	D2	000280	D8	0F0A20
D9	800080	D3	0EFBE0	D4	078000	D5	FFFFFF	F2	C10000	E7	0F5503	E5	010001	E6	09A80C
D3	0ED1D8	D4	900001	D5	000000	DB	800100	D8	0ED1D8	D9	E2801E	D3	0ED3C4	D4	900002
D5	000000	DB	800120	D8	0ED3C4	D9	E2801E	D3	0ED5B0	D4	900003	D5	000000	DB	800120
D8	0ED5B0	D9	E2801E	E6	09A80D	D3	0ED1D8	D4	900001	D5	000000	DB	800100	D8	0ED1D8
D9	E2801E	D3	0ED3C4	D4	900002	D5	000000	DB	800120	D8	0ED3C4	D9	E2801E	D3	0ED5B0
D4	900003	D5	000000	DB	800120	D8	0ED5B0	D9	E2801E	D6	200004	D2	000280	D8	0EFBE0
D9	800080	D3	0EFF70	D4	078000	D5	FFFFFF	F2	C10000	E7	0F5503	E5	010001	E6	09A80E
D3	0ED1D8	D4	900001	D5	000000	DB	800100	D8	0ED1D8	D9	E2801E	D3	0ED3C4	D4	900002
D5	000000	DB	800100	D8	0ED3C4	D9	E2801E	D3	0ED5B0	D4	900003	D5	000000	DB	800100
D8	0ED5B0	D9	E2801E	E6	09A80F	D3	0ED1D8	D4	900001	D5	000000	DB	800120	D8	0ED1D8
D9	E2801E	D3	0ED3C4	D4	900002	D5	000000	DB	800120	D8	0ED3C4	D9	E2801E	D3	0ED5B0
D4	900003	D5	000000	DB	800120	D8	0ED5B0	D9	E2801E	DB	200004	D2	000280	D8	0EFF70
D9	800080	D3	0F0300	D4	078000	D5	FFFFFF	F2	C10000	E7	0F5503	E5	010102	F8	118100
D3	0ED1D8	D4	900001	D5	000000	DB	800100	D8	0ED1D8	D9	E2801E	D3	0ED3C4	D4	900002
D5	000000	DB	800120	D8	0ED3C4	D9	E2801E	D3	0ED5B0	D4	900003	D5	000000	DB	800120
D8	0ED5B0	D9	E2801E	D3	0EED55	D4	400000	D5	000861	DB	800100	D2	0005A0	E7	0F5503
E5	010001	D6	800100	D2	0005A0	D8	0EED55	D9	800080	E6	09A810	D3	0ED1D8	D4	900001
D5	000000	DB	800100	D8	0ED1D8	D9	E2801E	D3	0ED3C4	D4	900002	D5	000000	DB	800100
D8	0ED3C4	D9	E2801E	D3	0ED5B0	D4	900003	D5	000000	DB	800100	D8	0ED5B0	D9	E2801E
DB	200[illegible]4	D2	000280	D8	0F0300	D9	800080	D3	0F0690	D4	078000	D5	FFFFFF	[illegible]2	C10000
E7	0F5[illegible]	E6	09A811	D3	0ED1D8	D4	900001								
D5	000000	D9	800100	D8	0ED1D8	D9	E2801E	D3	0ED3C4	D4	900002	D5	000000	DB	800100

E7 0ECE79	E5 010001	E7 0ECE79	E5 010001	E7 0ECE79	E5 010001	E7 0ECE79	E5 010001
E7 0ECE79	E5 010001	F8 118100	D3 0E5B74	D4 900002	D5 000000	DB 800120	D8 0E5B74
D9 E0801E	D3 0CEF3D	D4 000001	D5 001864	E7 0ECE79	E5 010001	E0 0EF080	E0 0DD83E
D6 800100	D2 0005A0	DB 800120	D2 000B40	DB 800120	D2 0010E0	DB 800120	D2 001680
DB 80012	[illegible]	DB 800120	D2 0021C0	DB 800120	D2 002760	DB 800120	D2 002D00
DB 80012				20	D2 003840	D8 0CEF3D	D9 800080
E4 3C000	① DISKPACK: NO DATA TRANSFER			)1	E7 0DD83E	D3 0E5B74	D4 900002
D5 00000				)1	E7 0DD83E	E5 030001	E7 0DD83E
E5 03000				)1	E7 0DD83E	E5 030001	E7 0DD83E
E5 03000				'4	D4 900002	D5 000000	DB 800100
D8 0E5B74	D9 E0801E	D3 0D6A53	D4 000000	D5 00BF5E	E7 0DD83E	E5 030001	E0 0EF080
D6 800100	D2 0005A0	DB 800100	D2 000B40	DB 800100	D2 0010E0	DB 800100	D2 001680
DB 800100	D2 001C20	E0 0E2CC0	E5 020001	E7 0E2CC0	E5 020001	E7 0E2CC0	E5 020001
E7 0E2CC0	E5 020001	E7 0E2CC0	E5 020001	E7 0E2CC0	E5 020001	E7 0E2CC0	E5 020001
E7 0E2CC0	E5 020001	E7 0E2CC0	E5 020001	F8 918100	E7 0E2CC0	E5 020001	D6 800100
D2 0021C0	DB 800100	D2 002760	DB 800100	D2 002D00	E0 0EF080	E0 0ECE79	E5 010001
E7 0ECE79	E5 010001	E7 0ECE79	E5 010001	E7 0ECE79	E5 010001	E7 0ECE79	E5 010001
E7 0ECE79	E5 010001	E7 0ECE79	E5 010001	E7 0ECE79	D6 800100	D2 0032A0	DB 800100
D2 003840	DB 800100	D2 003840	D8 0D6A53	D9 810080	D3 0E5B74	D4 900002	D5 000000
DB 800100	D8 0E5B74	D9 E0801E	D3 0D2CC8	D4 000003	D5 00190E	E4 3C0000	E0 0EF080
E3 0D6A53	E0 0ECE79	E5 010001	E7 0ECE79	D6 800100	D2 0005A0	DB 800100	D2 000B40
DB 800100	D2 0010E0	DB 800100	D2 001680	DB 800100	D2 001C20	DB 800120	D2 0021C0
DB 800120	D2 002760	DB 800120	D2 002D00	DB 800120	D2 0032A0	DB 800120	D2 003840
DB 800120	D2 003840	D8 0D2CC8	D9 800080	E4 3C0000	E0 0EF080	E3 0D2CC8	E0 0ECE79
E5 010001	F8 118100	D3 0E5B74	D4 900002	D5 000000	DB 800120	D8 0E5B74	D9 E0801E
D3 0CEF3D	D4 000001	D5 00186E	E7 0ECE79	E5 010001	E0 0EF080	D6 800120	D2 0005A0
DB 800120	D2 000B40	DB 800120	D2 0010E0	DB 800120	D2 001680	DB 800120	D2 001C20
DB 800120	D2 0021C0	DB 800120	D2 002760	DB 800120	D2 002D00	DB 800120	D2 0032A0
DB 800120	D2 003840	DB 800120	D2 003840	D8 0CEF3D	D9 800080	E0 0ECE79	E5 010001
E7 0ECE79	E5 010001	E7 0ECE79	E5 010001	E7 0ECE79	E5 010001	E7 0ECE79	E5 010001

1

E7 0ECE79	E5 010001	E7 0ECE79	E5 010001	E7 0ECE79	E5 010001	E7 0ECE79	E5 010001
F8 118100	D3 0E5B74	D4 900002	D5 000000	DB 800100	D8 0E5B74	D9 E0801E	D3 0CEF3D
D4 000001	D5 001878	E7 0ECE79	E5 010001	E0 0EF080	E0 0DD83E	E5 030001	E7 0DD83E
E5 030001	E7 0DD83E	E5 030001	E7 0DD83E	E5 030001	E7 0DD83E	E5 030001	E7 0DD83E
E5 030001	E7 0DD83E	D6 800100	D2 0005A0	DB 800120	D2 000B40	DB 800120	D2 0010E0
DB 800120	D2 001680	DB 800120	D2 001C20	DB 800120	D2 0021C0	DB 800120	D2 002760
DB 800120	D2 002D00	DB 800120	D2 0032A0	DB 800120	D2 003840	DB 800120	D2 003840
D8 0CEF3D	D9 800080	E4 3C0000	E0 0EF080	E3 0CEF3D	D3 0E5B74	D4 900002	D5 000000
DB 800100	D8 0E5B74	D9 E0801E	E0 0DD83E	E5 030001	E7 0DD83E	E5 030001	E7 0DD83E
E5 030001	F8 118100	D3 0E5B74	D4 900002	D5 000000	DB 800100	D8 0E5B74	D9 E0801E
C3 0D6A53	D4 000000	D5 00BF68	E7 0DD83E	D6 800100	D8 0D6A53	D9 800080	E5 030001
F8 118100	D3 0E5B74	D4 900002	D5 000000	DB 800100	D8 0E5B74	D9 E0801E	D3 0D6A53
D4 000000	D5 00BF72	E7 0DD83E					

(2) MT UPRIGHTS: AT TIME OF DUMP, TAPE CONTROL IS IN STATUS 10_8

D8 600040	D8 0ED71C	D9 C1CA30	D3 0EDAF4	D4 900004	D5 0EE0FD	D8 600040	D8 0EDAF4
D9 E0CC30	D3 0ED530	D4 900001	D5 0EDF4D	D8 600040	D8 0ED530	D9 E0CC30	D3 0ED71C
D4 880002	D5 0EDFDD	D8 600040	D8 0ED71C	D9 C1CA30	D3 0ED908	D4 C0A003	D5 0EE06D
D6 600040	D8 0ED908	D9 C04080	D3 0EDAF4	D4 900004	D5 0EE0FD	D8 600040	D8 0EDAF4
D9 E0CC30	E3 0ED908	D3 0ED530	D4 900001	D5 0EDF4D	D8 600060	D8 0ED530	D9 E0CC30
D3 0ED71C	D4 880002	D5 0EDFDD	D8 600060	D8 0ED71C	D9 C1CA30	D3 0EDAF4	D4 900004
D5 0EE0F				000F62	E3 000F62	D3 0ED530	D4 900001
D5 0EDF4				0ED71C	D4 880002	D5 0EDFDD	D8 600060
D8 0ED71				0EE06D	DA 008003	E3 000F62	E3 0EC5D0
D6 600040	DA 008003	D2 000640	D8 600040	D2 000860	D8 0D5378	D9 800080	D3 0D5FE8
D4 008003	D5 0EE06D	DA 008003	D6 600040	DA 008003	D2 000640	E0 0E8C81	E5 010701
E0 0F7080	D6 600060	D2 000860	D8 0D5FE8	D9 800080	D3 0D6C58	D4 008003	D5 0EE06D
DA 008003	D6 600040	DA 008003	D2 000640	D6 600040	D2 000860	D8 0D6C58	D9 800080
D3 0D78C8	D4 008003	D5 0EE06D	DA 008003	D6 600040	DA 008003	D2 000640	D8 600060
D2 000860	D8 0D78C8	D9 800080	D3 0D8538	D4 008003	D5 0EE06D	DA 008003	E3 000F62
D6 600040	DA 008003	D2 000640	D6 600040	D2 000860	D8 0D8538	D9 800080	D3 0D91A8
D4 008003	D5 0EE06D	DA 008003	D6 600040	DA 008003	D2 000640	D6 600040	D2 000860
D8 0D91A8	D9 800080	D3 0D9E18	D4 008003	D5 0EE06D	DA 008003	D6 600040	DA 008003
D2 000640	D6 600040	D2 000860	D8 0D9E18	D9 800080	D3 0DAA88	D4 008003	D5 0EE06D
DA 008003	D6 600060	DA 008003	D2 000640	D6 600040	D2 000860	D8 0DAA88	D9 800080
D3 0D86F8	D4 008003	D5 0EE06D	DA 008003	D6 600040	DA 008003	D2 000640	D6 600040
D2 000860	D8 0D86F8	D9 800080	D3 0DC368	D4 008003	D5 0EE06D	DA 008003	D6 600040
DA 008003	D2 000640	D6 600040	D2 000860	D8 0DC368	D9 800080	D3 0DCFD8	D4 008003
D5 0EE06D	DA 008003	E3 0E9326	D6 600040	DA 008003	D2 000640	D6 600040	D2 000860
D8 0DCFD8	D9 800080	D3 0DDC48	D4 008003	D5 0EE06D	DA 008003	E0 0E8C81	E5 010701
E7 0E8C81	E5 010701	F6 918200	E7 0E8C81	E5 010701	E7 0E8C81	D6 600040	DA 008003
D2 000640	E5 010701	F6 D18200	E7 0E8C81	D6 600060	D2 000860	D8 0DDC48	D9 800080
D3 0DE888	D4 008003	D5 0EE06D	DA 008003	E5 010701	E7 0E8C81	E5 010701	F6 D18200
E7 0E8C81	E5 010701	E7 0E8C81	E5 010701	F6 D18200	E7 0E8C81	D6 600060	DA 008003
D2 000640	D6 600040	D2 000860	D8 0DE888	D9 800080	D3 0DF528	D4 008003	D5 0EE06D
DA 008003	E5 010701	E7 0E8C81	E5 010701	F6 D18200	E7 0E8C81	E5 010701	E7 0E8C81
E5 010701	F6 D18200	E7 0E8C81	D6 600060	DA 008003	D2 000640	E5 010701	E7 0E8C81
D6 600040	D2 000860	D8 0DF528	D9 800080	D3 0E0198	D4 008003	D5 0EE06D	E5 010701
F6 D18200	E7 0E8C81	E5 010701	E7 0E8C81	E5 010701	F6 D18200	E7 0E8C81	D6 600040
D2 000640	E5 010701	E7 0E8C81	D6 600040	D2 000860	D8 0E0198	D9 800080	D3 0EDAF4
D4 900004	D5 0EE0FD	D8 600060	D8 0EDAF4	D9 E0CC30	D3 0ED530	D4 900001	D5 0EDF4D
D8 600060	D8 0ED530	D9 E0CC30	D3 0ED71C	D4 880002	D5 0EDFDD	D8 600060	D8 0ED71C
D9 C1CA30	D3 0D5378	D4 008003	D5 0EE06D	DA 008003	E5 010701	F6 918200	E7 0E8C81
E5 010701	E7 0E8C81	E5 010701	F6 D18200	E7 0E8C81	E5 010701	E7 0E8C81	E5 010701
F6 D18200	E7 0E8C81	D6 600040	DA 008003	D2 000640	D6 600040	D2 000860	D8 0D5378
D9 800080	D3 0D5FE8	D4 008003	D5 0EE06D	DA 008003	E5 010701	E7 0E8C81	E5 010701
F6 D18200	E7 0E8C81	E5 010701	E7 0E8C81	E5 010701	F6 D18200	E7 0E8C81	D6 600060
DA 008003	D2 000640	E5 010701	E7 0E8C81	D6 600040	D2 000860	E5 010701	F6 D18200
E7 0E8C81	E5 010701	E7 0E8C81	E5 010701	F6 D18200	E7 0E8C81	E5 010701	E7 0E8C81
E5 010701	F6 D18200	E7 0E8C81	E5 010701	E0 0F7080	E3 000F62	E3 000F62	E3 000F62
E3 0EC5D0	E3 0EC5D0	E3 000F62	E3 000F62	E3 000F62	E3 000F62	E3 000F62	E3 000F62
E3 0EC5D0	E3 000872	E3 000F62	E3 000982	E3 000F62	E3 0EC5D0		

③ READER/SORTER: DUMP TAKEN BY HALTING SYSTEM WHILE READING CHECKS.

D3 808000	D2 FFFCC8	D8 D02020	D3 057AB5	D4 080000	D5 808000	E0 0688C5	E5 010005
E0 077C80	D6 D02000	D2 000320	D8 057AB5	D9 800000	E3 057AB5	E0 0688C5	E5 010004
E0 077080	FD 0B0000	D7 3E0080	D3 057AB5	D4 E00000	D5 808000	D2 FFFCC8	D6 D02000
D3 057AB5	D4 080000	D5 808000	E0 0688C5	E5 010005	E0 077080	D6 D02000	D2 000320
D8 057AB5	[illegible]	[illegible]	[illegible]	[illegible]	E0 077080	FD 0B0000	D7 3E0080
D3 057AB5	[illegible]	[illegible]	[illegible]	[illegible]	D3 057AB5	D4 080000	D5 808000
E0 0688C5	[illegible]	[illegible]	[illegible]	[illegible]	D8 057AB5	D9 800000	E3 057AB5
E0 0688C5	E5 010004	[illegible]	[illegible]	[illegible]	D3 057AB5	D4 E00000	D5 808000
D2 FFFCC8	D6 D02000	D3 057AB5	D4 080000	D5 808000	E0 0688C5	E5 010005	E0 077080
D6 D02000	D2 000320	D8 057AB5	D9 800000	E3 057AB5	E0 0688C5	E5 010004	E0 077080
FD 0B0000	D7 3E0080	D3 057AB5	D4 E00000	D5 807000	D2 FFFCC8	D6 D02000	D3 057AB5
D4 080000	D5 807000	E0 0688C5	D6 D02020	D2 000320	D8 057AB5	D9 800000	E4 3B0000
E0 077080	E3 057AB5	E0 0688C5	E5 010004	E0 077080	FD 0B0000	D7 3E0080	D3 057AB5
D4 E00000	D5 807000	D2 FFFCC8	D6 D02020	D3 057AB5	D4 080000	D5 807000	E0 0688C5
E5 010005	E0 077080	DB D02000	D2 000320	D8 057AB5	D9 800000	E3 057AB5	E0 0688C5
E5 010004	E0 077080	FD 0B0000	D7 3E0080	D3 057AB5	D4 E00000	D5 807000	D2 FFFCC8
E0 0688C5	D6 D02000	D3 057AB5	D4 080000	D5 807000	E5 010005	E0 077080	D6 D02000
D2 000320	D8 057AB5	D9 800000	E3 057AB5	E0 0688C5	E5 010004	E0 077080	FD 0B0000
D7 3E0080	D3 057AB5	D4 E00000	D5 807000	D2 FFFCC8	E0 0688C5	D6 D02020	D3 057AB5
D4 080000	D5 807000	E5 010005	E0 077080	D6 D02000	D2 000320	D8 057AB5	D9 800000
E3 057AB5	E0 0688C5	E5 010004	E0 077080	FD 0B0000	D7 3E0080	D3 057AB5	D4 E00000
D5 807000	D2 FFFCC8	D6 D02000	D3 057AB5	D4 080000	D5 807000	E0 0688C5	E5 010005
E0 077080	D6 D02000	D2 000320	D8 057AB5	D9 800000	E3 057AB5	E0 0688C5	E5 010004
E0 077080	FD 0B0000	D7 3E0080	D3 057AB5	D4 E00000	D5 807000	D2 FFFCC8	D6 D02000
D3 057AB5	D4 080000	D5 807000	E0 0688C5	E5 010005	E0 077080	D6 D02000	D2 000320
D8 057AB5	D9 800000	E3 057AB5	E0 0688C5	E5 010004	E0 077080	FD 0B0000	D7 3E0080
D3 057AB5	D4 E00000	D5 807000	D2 FFFCC8	D6 D02020	D3 057AB5	D4 080000	D5 807000
E0 0688C5	E5 010005	E0 077080	D6 D02000	D2 000320	D8 057AB5	D9 800000	E3 057AB5
E0 0688C5	E5 010004	E0 077080	FD 0B0000	D7 3E0080	D3 057AB5	D4 E00000	D5 807000
D2 FFFCC8	D6 D02000	D3 057AB5	D4 080000	D5 807000	E0 0688C5	E5 010005	E0 077080
D6 D02000	D2 000320	D8 057AB5	D9 800000	E3 057AB5	E0 0688C5	E5 010004	E0 077080
FD 0B0000	D7 3E0080	D3 057AB5	D4 E00000	D5 807000	D2 0020B8	D6 D02020	D3 057AB5
D4 080000	D5 807000	E0 0688C5	E5 010005	E0 077080	D6 D02000	D2 000320	D8 057AB5

D9 800000	E3 057AB5	E0 0688C5	E5 010004	E0 077080	FD 0B0000	D7 3E0080	D3 057AB5
D4 E00000	D5 806200	D2 FFFCC8	D6 D02000	D3 057AB5	D4 180000	D5 806200	E0 0688C5
D6 D02000	D2 000320	D8 057AB5	D9 800000	E4 3B0000	E0 077080	E3 057AB5	E0 0688C5
E5 010004	E0 077080	FD 0B0000	D7 3E0080	D3 057AB5	D4 E00000	D5 806200	D2 FFFCC8
D6 D02020	D3 057AB5	D4 180000	D5 806200	E0 0688C5	E5 010005	E0 077080	D6 D02000
D2 000320	D8 057AB5	D9 800000	E3 057AB5	E0 0688C5	E5 010004	E0 077080	FD 0B0000
D7 3E0080	D3 057AB5	D4 E00000	D5 806200	D2 FFFCC8	E0 0688C5	D6 D02020	D3 057AB5
D4 180000	D5 806200	E5 010005	E0 077080	D6 D02000	D2 000320	D8 057AB5	D9 C00201
E3 057AB5	E0 0688C5	E5 010004	E0 077080	FD 2B0000	D7 3E0080	E0 0688C5	E5 010005
E0 077080	E0 0688C5	E5 010005	E0 077080	E0 0688C5	E5 010005	E0 077080	E0 0688C5
E5 010005	E0 077080	E0 0688C5	E5 010005	E0 077080	FD CB0000	D3 06DAC0	D4 900000
D5 000000	DB D02000	D8 06DAC0					
D9 800094	D3 057AB5	D4 080000	D5 80A080				

Burroughs Corporation INTER-OFFICE CORRESPONDENCE

CORPORATE UNIT	LOCATION	DEPT.
Computer Systems Group	Santa Barbara Plant	Software Activity

TO:

NAME	DATE
Members of Software Activity	

FROM	DEPT. & LOCATION
E. R. Bauerle, Section Manager	Operating Systems

SUBJECT:

C.C.
E. Munsch
L. Trautwein
J. Trost
J. Swensson
S. Bhagwan
J. Macker

On Tuesday, Wednesday and Thursday, August 20 through August 22, there will be classes on the micro-coded SOFT.IO routines. The classes will be held in the new conference room and will begin at 1:30 P.M. each day. Each class will last about three hours; the content of each days class is presented below.

SOFT.IO, if anyone does not know, is the micro-coding in the system which receives I/O requests, usually from the MCP, and makes corresponding requests upon the hard I/O Subsystem. The class will be organized along the same lines. Discussion will be presented covering (1) requests from the MCP (et. al.) to SOFT.IO and (2) requests from SOFT.IO to the Subsystem. There will be no discussion of the MCP at this time.

The approximate schedule is as follows:

Tuesday - General discussion of SOFT.IO
- A. Requests upon SOFT.IO
- B. Requests upon the Subsystem by SOFT.IO
- C. The SPO

Wednesday - Disk and Tape
- A. Disk Requests to SOFT.IO
- B. Tape Requests to SOFT.IO
- C. Exchange Requests to SOFT.IO
- D. Disk Requests, SOFT.IO to Subsystem
- E. Tape Requests, SOFT.IO to Subsystem
- F. Exchange Requests, SOFT.IO to Subsystem

Thursday - Reader-Sorter, Data Comm, Trace
- A. Sorter Requests to SOFT.IO
- B. Data Comm Requests to SOFT.IO
- C. Sorter Requests, SOFT.IO to Subsystem
- D. Data Comm Requests, SOFT.IO to Subsystem
- E. Trace

The class is intended to be detailed. Hopefully, however, the structure presented above will allow you to leave the class whenever the level of detail exceeds your level of interest. You may leave and return as much as you like; please do so quietly. It is recommended that no one leave during the first day, unless they have no intention of returning.

E. R. Bauerle, Operating Systems

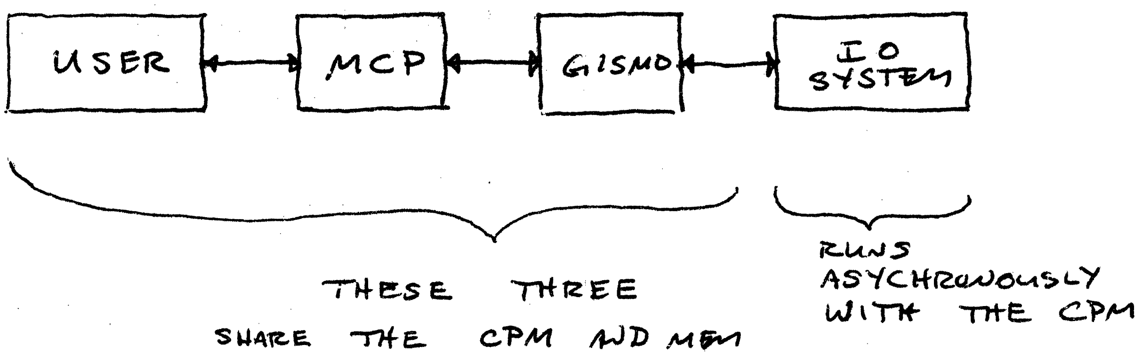
THE HARDWARE IS CONNECTED

THIS REQUIRES MICRO-CODED ROUTINES TO

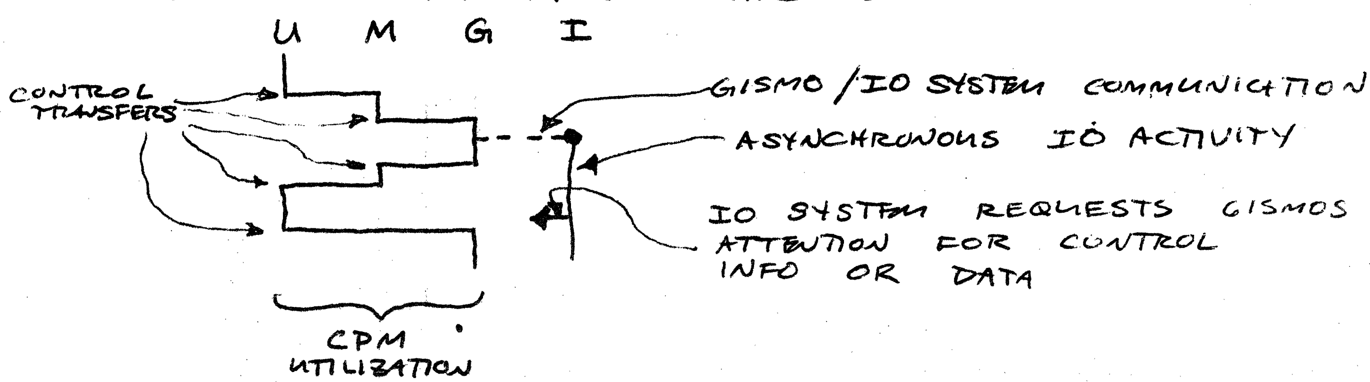
- COMMUNICATE CONTROL INFORMATION TO/FROM THE I/O SYSTEM, SUCH AS OPCODE, MEDIA ADDRESS, RESULT DESCRIPTOR
- PROVIDE FOR DATA TRANSFER BETWEEN MEMORY AND THE I/O SYSTEM.

THESE ROUTINES ARE THE SOFT IO ROUTINES IN GISMO. GISMO SERVES OTHER FUNCTIONS; THIS CLASS IS PRIMARILY CONCERNED WITH ITS SOFT IO ROUTINES

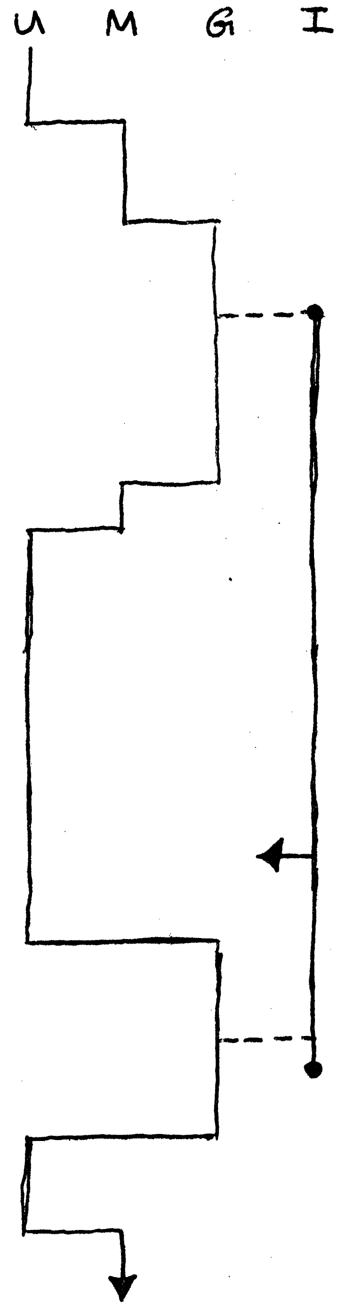
ANOTHER WAY OF LOOKING AT IT IS THAT GISMO INTERFACES BETWEEN IO REQUESTORS AND THE IO SYSTEM. NORMALLY, USERS MAKE REQUESTS TO THE MCP (OR, IN ITS PLACE, ENHANCED-IO), WHO IN TURN SENDS THEM TO GISMO. THUS



WE CAN CHART ACTIVITY ON THE SYSTEM BY



A CHART OF BUFFERED I/O ACTIVITY (WRITE): ②



USER FORMATS AN OUTPUT RECORD IN HIS DATA AREA.

USER REQUESTS THAT IT BE WRITTEN

MCP TRANSFERS THE DATA FROM THE USER DATA AREA TO THE BUFFER AREA, AND REQUESTS GISMO TO INITIATE IO

GISMO INITIATES THE CONTROL, SENDING WHATEVER CONTROL INFO IS REQUIRED (OP CODE, AT LEAST) AND SOME OR ALL OF THE DATA TO BE WRITTEN

THE CONTROL INITIATES THE PERIPHERAL TO BEGIN THE WRITE OPERATION.

GISMO RETURNS CONTROL TO THE MCP

THE MCP RETURNS TO THE USER

THE USER BEGINS FORMATTING THE NEXT OUTPUT RECORD.

ASYNCHRONOUS ACTIVITY

THE PERIPHERAL COMPLETES THE OPERATION. THE CONTROL THEN INDICATES THAT IT DESIRES COMMUNICATION WITH GISMO (CALLED SERVICE REQUEST)

CONTROL IS TRANSFERRED FROM THE USER (INTERPRETER) TO GISMO.

THE IO SYSTEM (CONTROL) COMMUNICATES THE RESULT DESCRIPTOR TO GISMO, THEN BECOMES INACTIVE.

GISMO RETURNS TO THE USER

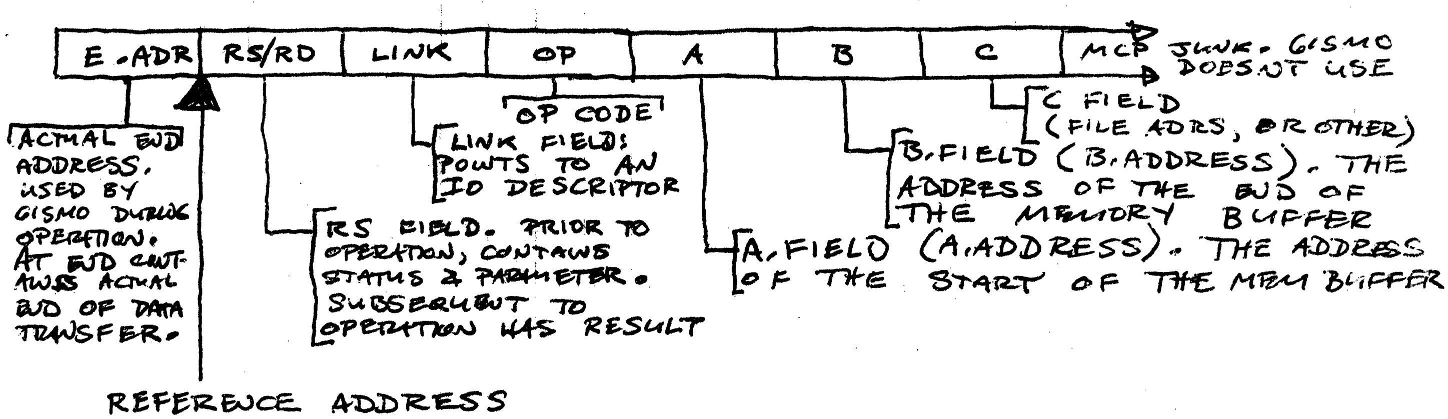
THE USER REQUESTS THAT THE NEXT RECORD BE WRITTEN

ETC

THE IO SYSTEM ALLOWS FOR 15 CHANNELS (CONTROLS) AND SO THE I COLUMN ABOVE MAY CONSIST OF 15 IN PROCESS I/O OPERATIONS AT ONCE. AT ANY GIVEN MOMENT, THOUGH, COMMUNICATION BETWEEN GISMO AND THE IO SYSTEM INVOLVES COMMUNICATION WITH 1 CONTROL ONLY, AS IS SHOWN ABOVE.

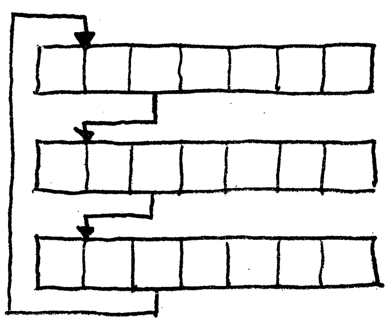
GISMO IS REQUESTED TO DO AN IO OPERATION BY DRAWING ITS ATTENTION TO AN IO DESCRIPTOR. THIS DESCRIPTOR PROVIDES GISMO WITH MOST OF THE INFORMATION NECESSARY TO DO AN IO. ITS FORMAT IS:

IO DESCRIPTOR: SEVEN 24 BIT FIELDS



AN IO DESCRIPTOR IS USUALLY ~~FOUND~~ LOCATED BY ITS REFERENCE ADDRESS; IE, THE ADDRESS OF THE RS FIELD.

DESCRIPTORS MAY BE LINKED TOGETHER



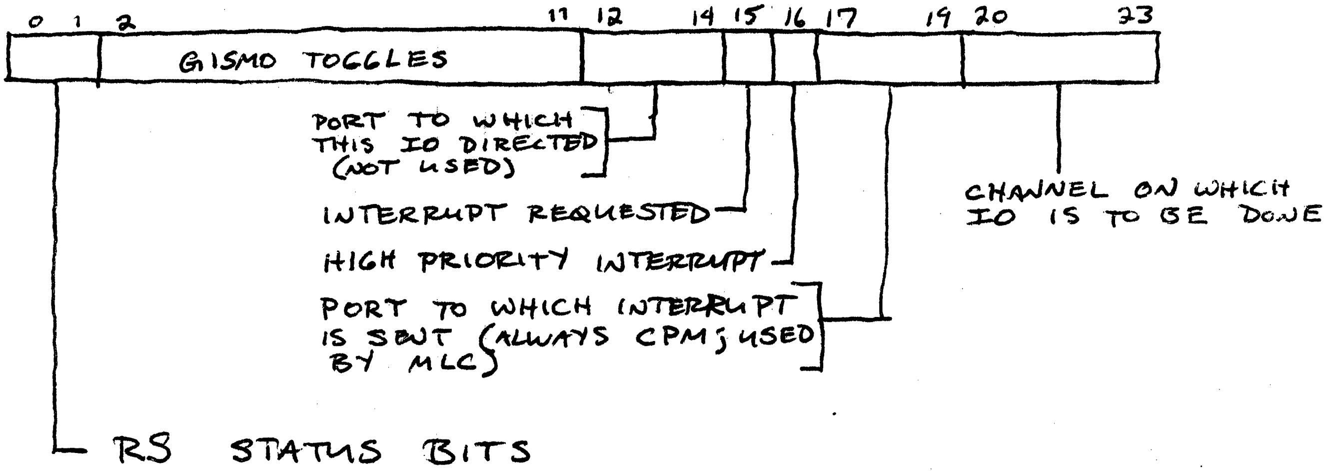
IN THIS CASE, THE LINK FIELD OF EACH DESCRIPTOR CONTAINS THE REFERENCE ADDRESS OF THE NEXT DESCRIPTOR.

A LINK FIELD MAY NOT BE ZERO; THE DESCRIPTOR MAY BE LINKED TO ITSELF.

GISMO'S ATTENTION MAY BE DRAWN TO AN IO DESCRIPTOR IN TWO WAYS; DIRECTLY, VIA AN MCP (OR ENHANCED IO) DISPATCH OPERATION OR INDIRECTLY, VIA LINKING, AT THE COMPLETION OF AN IO TO THE NEXT DESCRIPTOR IN THE CHAIN. IN THE FIRST CASE, THE ~~IO~~ CHANNEL FOR THE OPERATION AND THE DESCRIPTOR'S REFERENCE ADDRESS ARE PARAMETERS. IN THE SECOND, THE CHANNEL IS THE ONE JUST TERMINATED AND THE REFERENCE ADDRESS IS FROM THE JUST TERMINATED DESCRIPTOR'S LINK FIELD.

THE REFERENCE ADDRESS POINTS TO THE RS FIELD. THIS FIELD'S FORMAT IS:

RS FIELD (24 BITS)



RS STATUS BITS

- 00 READY TO BE EXECUTED
- 01 IO CURRENTLY IN PROCESS
- 10 LOCKED / IO COMPLETE
- 11 IO COMPLETE WITH EXCEPTION,

THE LEFTMOST BIT OF A RESULT DESCRIPTOR IS ALWAYS SET. CONSEQUENTLY, STORING THE RESULT DESC LOCKS THE RS; THE MCP MAY LOCK AN RS AS WELL.

GISMO WILL ONLY INITIATE A READY (RS BITS = 00) DESCRIPTOR, AT INITIATION TIME GISMO SETS THE STATUS TO 01 (IN PROCESS).

THE "GISMO TOGGLES" AREA IS USED BY GISMO WHEN THE DESCRIPTOR IS IN PROCESS TO SAVE INFORMATION INTERESTING TO ITSELF ABOUT THE IO.

WHEN LINKING, IF GISMO ENCOUNTERS A LOCKED DESCRIPTOR FOR SIMPLE DEVICES IT WILL STOP PROCESSING IOS ON THAT CHAIN. TO START INITIATION AGAIN, A DISPATCH MUST BE DONE.

A TABLE IS ALLOCATED IN MEMORY AND INITIALIZED BY THE MCP ~~[illegible]~~ IN WHICH IS KEPT INFORMATION PERTINENT TO CHANNELS, CALLED THE CHANNEL TABLE. IT IS INDEXED BY CHANNEL. EACH ENTRY IS 48 BITS.

CHANNEL TABLE ENTRY

0–15	16–23	24–47
TOGS	EXCH LINK	ADDRESS

TOGS (bits 0–15):

0	1	2	3	4	5	6	7	8	9	10	11	12–15
B	P	EI	T	Ø	EL	P	I	K	TP	-	-	TYPE

FOR	
A	BUSY: SET/RESET BY GISMO. SET MEANS CONTROL IS BUSY.
T,D	PENDING: SET/RESET BY GISMO. SET MEANS LINK THROUGH HEAD OF QUE
S	EXCEPTION IDLE: SET/RESET BY GISMO. SET MEANS INHIBIT DISPATCHS DUE TO AN EXCEPTION ON THE CHANNEL
T,TD	TIMER: SET BY MCP, NEVER CHANGES. AT TIMER INTERVAL INTERRUPT (100 MIL) ISSUE DISPATCH ON THIS CHANNEL. MECHANIZES TEST & WAIT FOR TAPE AND DISK
A	OVERRIDE: SET BY MCP/RESET BY GISMO. ~~R~~SET MEANS AT DISPATCH TIME RESET BUSY, PENDING, EXCEPTION. IDLE.
T,D	EXCHANGE LINK: SET BY MCP, NEVER CHANGES. SET MEANS THE EXCHANGE LINK FIELD CONTAINS THE PORT/CHANNEL THAT IS ~~OF THE~~ NEXT ON ~~THE~~ EXCHANGE.
DC-1 DATA COM	PAUSE: SET BY MCP, NEVER CHANGES. SET MEANS TO PAUSE WHEN A LOCKED DESCRIPTOR OR PAUSE OP IS ENCOUNTERED WHILE LINKING. RESET MEANS STOP FOR THESE.
A	INITIALIZED: SET BY MCP, NEVER CHANGES. THIS ENTRY IS SET UP
T,D	KEEP LINKING: SET BY MCP, NEVER CHANGES. WHEN AN EXCEPTION OCCURS, OR A LOCKED OR IN PROCESS DESCRIPTOR IS ENCOUNTERED WHILE LINKING, LINK TO THE NEXT DESC.
DC-1	TO PAUSE: SET/RESET BY GISMO. UPON LINKING TO HEAD OF Q, IF SET THEN RESET AND PAUSE ELSE STOP.
	TYPE: USED BY MCP - ANALYZER
T,D	EXCH LINK: MAY POINT TO NEXT ON EXCHANGE
T,D	ADDRESS: IF KEEP LINKING SET THEN POINTS TO HEAD OF QUE.

"FOR" LEGEND: A = ALL DEVICES, T = ALL TAPE DEVICES, D = ALL DISK DEVICES, S = ALL DEVICES EXCEPT DISKS AND TAPE, TD = DC-2 AND DISKPACK.

THE FOLLOWING IS A PSEUDO LISTING OF GISMO. SOME ROUTINES ARE PRIMARILY CONCERNED WITH INTERFACE BETWEEN IO REQUESTORS AND GISMO; SOME WITH THE INTERFACE BETWEEN GISMO AND THE I/O SUBSYSTEM.
EACH ROUTINE HAS A KEY IN THE UPPER RIGHT CORNER. THE FIRST LETTER IS THE SEQUENCE LETTER. THE SECOND INDICATES WHETHER THIS VERSION IS COMPLETE; R IS INCOMPLETE, Z IS COMPLETE. THE WHOLE LISTING WILL BE COMPLETE AFTER COVERING DATA-COMM AND READER SORTER.

THIS FIRST VERSION SHOWS THE ~~[illegible]~~ ROUTINES NECESSARY TO COPE WITH SIMPLE DEVICES.

A - R

```
DISPATCH    (MCP-GISMO)
    GET THE APPROPRIATE CHANNEL TABLE ENTRY
    IF ITS NOT INITIALIZED THEN BYPASS USUAL RULES
    IF EXCEPTION, OVERRIDE THEN
        RESET BUSY, OVERRIDE, EXCEPTION IDLE
    IF BUSY THEN
        EXIT
    GO TO RESET CHANNEL
```

B - R

```
RESET CHANNEL   (GISMO - IO SYSTEM)
    IF PORT NEQ 7 THEN    % NOT SOFT IO)
        MISSING DEVICE ERROR
    IF CHANNEL EQL 15 THEN   (HI RESOLUTION TIMER)
        MISSING DEVICE ERROR
    SET UP CHANNEL, REFERENCE ADDRESS
    GET CONTROL'S STATUS AND ID    (CA-RC)
    IF ALL ZERO THEN
        MISSING DEVICE ERROR
    IF STATUS NEQ 1 THEN
        CLEAR CONTROL          (CA-RC)
    GO TO ANALYZE ID
```

```
ANALYZE ID     (GISMO-GISMO)                                  C-Z
    SET UP LOCAL TOGGLES BY DECODING CONTROL
    ID. TOGGLES SAVED IN RS FIELD (EVENTUALLY)
    OF IO DESCRIPTOR. TOGGLES ARE ALL.DISK,
    ALL.TAPE, DATA COMM, READER.SORTER
```

```
GET A DESCRIPTOR   (MCP-GISMO)                                D-R
    FETCH RS FIELD FROM DESC.
    IF NOT READY THEN (NOT 00)
    BEGIN
        RESET BUSY BIT IN CHANNEL TABLE ENTRY
        GO TO NEXT.SERVICE
    END
    LOCK RS (01) AND SAVE TOGGLES
```

```
                                                              E-R
ANALYZE OP CODE    (MCP-GISMO)
    IF OP CODE IS STOP OP THEN
    BEGIN
        FAKE UP RESULT DESC
        SET DONT.LINK TOGGLE
        GO TO ANALYZE.RESULT
    END
    SET E.ADR (IN DESC) TO A.ADR
```

```
EXECUTE DESCRIPTOR   (GISMO-CONTROL)                          F-R

    TRANSFER OUT OP CODE          (3⊗ CA-RC) (STATUS 1,2,3)
    TRANSFER OUT FILE ADDRESS     (3⊗ CA-RC) (STATUS 4,5,6)
    CALL IO.DATA ROUTINE (MAY OR MAY NOT MOVE DATA)
    TRANSFER OUT REFERENCE ADDRESS (3⊗ CA-RC) (STATUS 7,8,9
        (NOTE: CONTROL IS NOW "BUSY", IN STATUS 10)
    GO TO NEXT.SERVICE
```

```
NEXT.SERVICE   (GISMO - CONTROL)                                G-R
    (NOTE: ARRIVE HERE WHEN INTERPRETER DETECTS
     SERVICE REQUEST AND TRANSFERS TO GISMO)
    (ALSO: PRIOR TO EXITING, GISMO WILL RETURN TO
     THIS POINT TO SEE IF ANY OTHER CONTROLS
     WANT SERVICE)

    GET SERVICE.REQUEST MASK  (CA-RC, ALL CONTROLS)
    IF MASK ALL ZEROS THEN
        EXIT
    SELECT AND CONVERT LEFTMOST BIT (HIGHEST CHANNEL)
    TRANSFER IN REFERENCE.ADDRESS (3⊗CA-RC, STATUS 11,12,13 / 18,19,20)
    CALL IO-DATA ROUTINE (MAY OR MAY NOT MOVE DATA)
    GET CONTROL STATUS  (CA-RC)
    IF STATUS EQL 7 THEN
    BEGIN
        TRANSFER OUT REFERENCE.ADDRESS (3⊗CA-RC)(STATUS 7,8,9)
            (NOTE: AT THIS POINT CONTROL IS "BUSY", STATUS 10)
        GO TO NEXT.SERVICE
    END
    (CONTROL IN STATUS 21)
    TRANSFER IN RESULT DESCRIPTOR (3⊗CA-RC, STATUS 21,22,23)
    (CONTROL IN STATUS 1, INITIAL STATE)
```

```
ANALYZE RESULT   (MCP-GISMO)                                    H-R
    IF EXCEPTION (IN RESULT) OR WANT INTERRUPT (RS FIELD) THEN
    BEGIN
        FORMAT AND ENTER INTERRUPT IN INTERRUP QUEUE
        IF EXCEPTION THEN
        BEGIN
            SET EXCEPTION.IDLE BIT IN CHANNEL TABLE ENTRY
            ~~RESET BUSY~~ BIT
        END
    END
    IF EXCEPTION OR DONT.LINK THEN
    BEGIN
        RESET BUSY BIT IN CHANNEL TABLE ENTRY
        GO TO NEXT.SERVICE
    END
    USING REFERENCE ADDRESS FROM LINK FIELD, ~~GO TO GE~~
```

IO.DATA (GISMO-CONTROL) I-Z

(NOTE: THIS ROUTINE IS USED TO MOVE DATA TO AND FROM THE CONTROL AND MEMORY. IF THE CONTROL IS NOT INTERESTED IN MOVING DATA (NOT IN STATUS 14 OR 15) THIS ROUTINE EXITS.)

```
GET CONTROL STATUS (CA-RC)
IF STATUS EQL 14 THEN
    SET OUTPUT
IF STATUS EQL 15 THEN
    SET INPUT
IF STATUS NEQ 14 AND STATUS NEQ 15 THEN
    EXIT
CALCULATE SPACE REMAINING BETWEEN E.ADR AND B.ADR
    (OR A.ADR AND E.ADR, IF REVERSE) IN MEMORY BUFFER.
```

(NOTE: THE NEXT PART OF THE ROUTINE SETS A VARIETY OF TOGGLES AND PARAMETERS, DEPENDING ON DEVICE TYPE, WHICH CONTROL THE DATA TRANSFER. THESE INCLUDE BYTE SIZE OF DATA IN CA-RC CYCLE (6, 8, 12, 16, 24), INDICATION OF MULTI-BUFFER DEVICE, INDICATION OF DEVICES THAT ACCEPT TERMINATE COMMAND. THE ACTUAL LOOPS FOR INPUT AND OUTPUT ARE QUITE SIMILAR AND ARE SHOWN BELOW, MERGED)

(NOTE: THE FOLLOWING CODE DOES NOT APPLY TO 5N DISK, DISKPACK OR MODEL 4 TAPE. THE CODE FOR THESE IS NOT IN THE PSEUDO LISTING)

INPUT – OUTPUT LOOP (INPUT / OUTPUT)

```
    GET DATA FROM CONTROL (CA-RC) OR READ DATA FROM MEMORY
    WRITE DATA TO MEMORY OR SEND DATA TO CONTROL (CA-RC)
    IF CONTROL'S BUFFER IS EMPTY (STATUS 16 OR 17)
        GO TO CHECK.TERMINATE
    IF MEMORY BUFFER NOT FULL YET (E.ADR ≠ B.ADR) THEN
        GO TO INPUT-OUTPUT LOOP
    IF TERMINATE TYPE DEVICE THEN
        GO TO TERMINATE
    GO TO EMPTY (INPUT) OR BLANK.REST (OUTPUT)
```

CHECK.TERMINATE

    IF MEMORY BUFFER FULL THEN
        IF TERMINATE TYPE DEVICE THEN
            GO TO TERMINATE
    GO TO WRAP.UP

TERMINATE

    SEND TERMINATE DATA COMMAND TO CONTROL (CA-RC)
    GO TO WRAP UP

EMPTY-BLANK.REST

    GET DATA FROM CONTROL (INPUT (CA-RC)) OR SEND DUMMY DATA TO CONTROL (OUTPUT (CA-RC))
    IF CONTROL'S BUFFER NOT EMPTY THEN (STATUS STILL 14,15)
        GO TO EMPTY-BLANK.REST

WRAP.UP

    UPDATE E.ADR FIELD OF DESCRIPTOR
    EXIT

UTILITY ROUTINES (GISMO-CONTROL) J-Z

SEVERAL UTILITY ROUTINES ARE USED TO SEND TO AND RECEIVE FROM THE CONTROL, INCLUDING:

TRANSFER.OUT: SENDS A 3 BYTE (24 BITS) ITEM TO THE CONTROL, WITH 3 CA-RC CYCLES. USED TO SEND OUT THE OP.CODE, FILE ADDRESS AND REFERENCE.ADDRESS

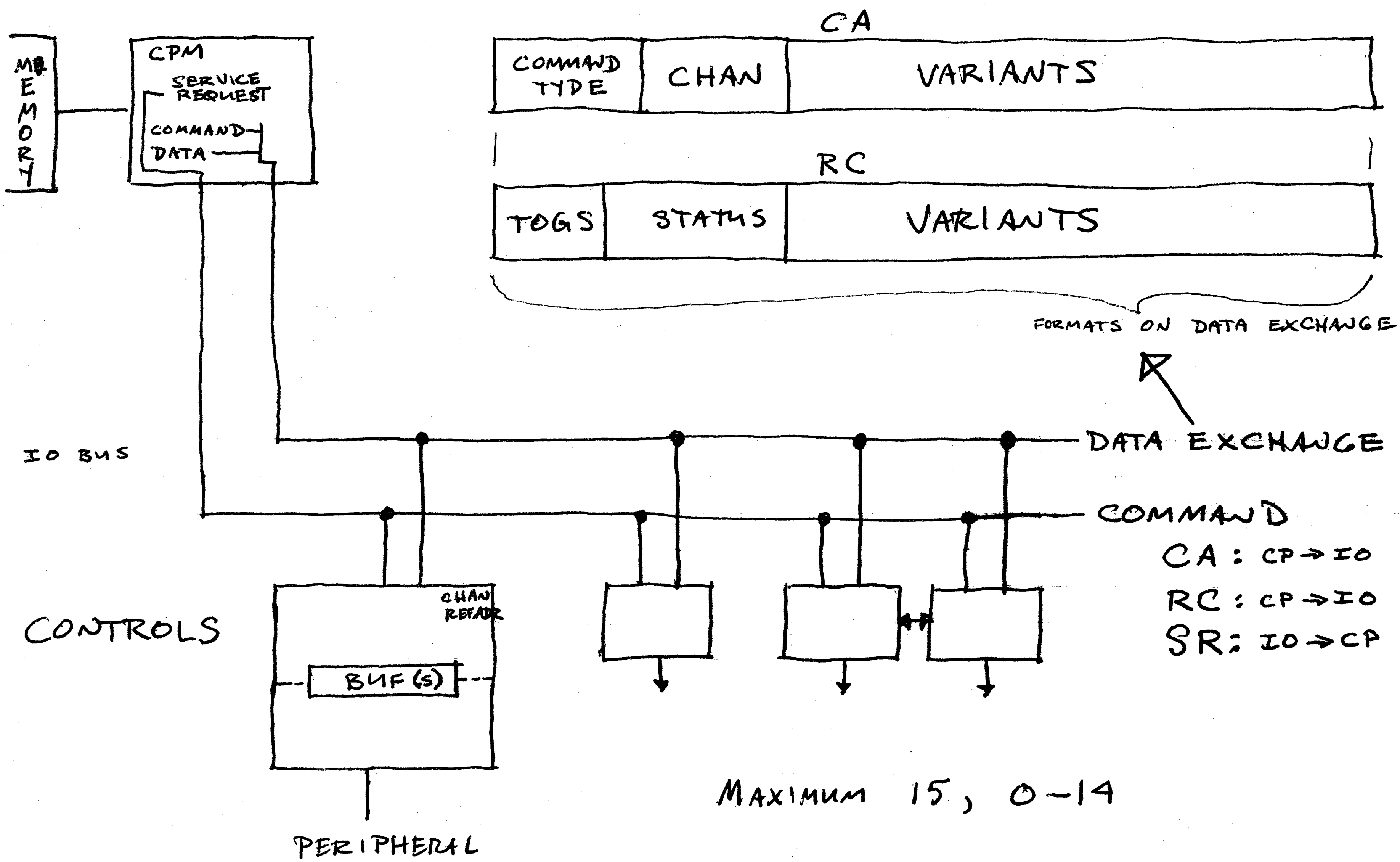
TRANSFER.IN: RECEIVES A 3 BYTE (24 BITS) ITEM FROM THE CONTROL, WITH 3 CA-RC CYCLES. USED TO BRING IN ~~THE REFERENCE~~ ADDRESS AND RESULT

GET.SERVICE.REQUEST MASK
    USES ONE CA-RC CYCLE TO GET THE MASK

GET STATUS
    USES ONE CA-RC CYCLE TO TEST CONTROL'S STATUS

GET.BYTE.FROM.CHANNEL
    USES ONE CA-RC CYCLE TO RECEIVE BYTE. USED FOR ~~[illegible]~~ GETTING BYTE COUNT, POCKET SELECT RESULT, ETC

SEND BYTE TO CHANNEL
    USES ONE CA-RC TO SEND BYTE. USED FOR SENDING



I/O SUBSYSTEM

MOVE <REGISTER> TO COMMAND → CA CYCLE

MOVE DATA TO <REGISTER> → RC CYCLE

COMMAND ACTIVATE - RESPONSE COMPLETE FORMATS

TYPES	CA	TYPE	CHAN	VARIANT	RC	TOG (0 1 2 3)	STATUS (7 8)	VARIANT (23)
TEST STATUS		0001	CHAN	0 ———— 01		—	STATUS	DEVICE ID | X
CLEAR & TEST STATUS		0001	CHAN	0 ———— 011		—	STATUS	DEVICE ID | X
TEST SERVICE REQUEST		0001	—	0 ———— 0101		—	— 1	SERVICE REQ MASK
TERMINATE DATA		0001	CHAN	0 ———— 0110		—	STATUS	——
TRANSFER OUT A		0010	CHAN	1 OR 2 DATA BYTES		—	STATUS	——
TRANSFER IN		0100	CHAN	——		—	STATUS	1 OR 2 DATA BYTES 24 DATA BITS
TRANSFER OUT B		0011	CHAN			24 DATA BITS		

TOGS AT RC TIME

TAPE: BIT 0 = ODD BYTE, BIT 1 = DRIVE THRU GAP, BIT 2 = REVERSE

SORTER: BIT 0 = NONE BIT 1 = POCK SELECT, BIT 2 = REVERSE

OTHER: BIT 2 = REVERSE

ADDITIONAL SPECIAL TYPES

- TRANSFER IN BYTE COUNT (MTC-4, PACK, 5N DISK)
- TRANSFER OUT BYTE COUNT (" " " , PAPER TAPE)
- ODD, CHAR COUNT (MTC-2)
- DRIVE THRU GAP (MTC-2)

STATUS VALUES: REFLECT CONTROL STATE PRIOR TO THIS CA-RC TRANSACTION.

COUNTS	MEANS
0	CONTROL NOT PRESENT
1	CLEARED (INITIAL) STATE
1,2,3	READY TO RECEIVE OP CODE BYTES 1,2,3
4,5,6	READY TO RECEIVE FILE ADDRESS BYTES 1,2,3
7,8,9	READY TO RECEIVE REFERENCE-ADR BYTES 1,2,3
10	BUSY (DOING OPERATION). USUALLY GOES TO 11 OR 18 AND RAISES SERVICE REQUEST
11,12,13	READY TO SEND REFERENCE ADDRESS, BYTES 1,2,3. USUALLY IMPLIES DATA TO FOLLOW
14	READY TO RECEIVE DATA (OUTPUT)
15	READY TO SEND DATA (INPUT)
16	END OF BUFER (READY TO SEND OR RECEIVE LAST BYTE); MORE TO COME.
17	END OF BUFFER; LAST BUFFER
18,19,20	READY TO SEND REFERENCE ADDRESS, BYTES 1,2,3. IMPLIES RESULT DESC TO FOLLOW.
21,22,23	READY TO SEND RESULT DESC BYTES 1,2,3

CONTROLS GO FROM STATUS 23 BACK TO 1.

CONTROLS DO NOT SEQUENCE STRAIGHT THRU COUNTS. RATHER STATUS COUNT IS USED TO TELL SOFT.IO WHAT IS REQUIRED NEXT.

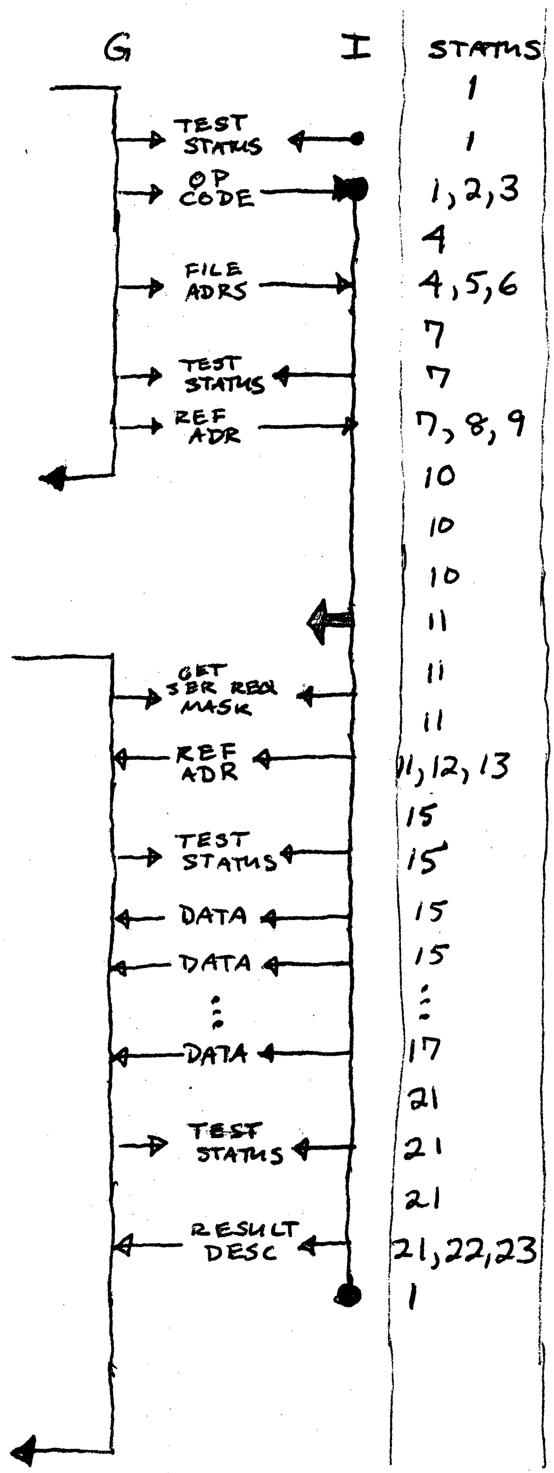
LEGAL TRANSITIONS: THOSE COUNTS GROUPED TOGETHER ABOVE (IE, 1,2,3) INDICATE A REQUIRED SEQUENCE (IE, 2 MUST FOLLOW 1, 3 MUST FOLLOW 2).

OTHERS ARE

COUNT	MAY GO NEXT TO	COUNT	MAY GO NEXT TO
0	1	14	16,17
3	4	15	16,17
6	7,14	16	7
9	10	17	7,21
10	11,18	20	21
13	14,15,16,17	23	1

TYPICAL SET OF GISMO-CONTROL TRANSACTIONS FOR SINGLE BUFFER DEVICES:

READ:



GISMO REQUESTED TO INITIATE A CA-RC CYCLE

THREE CA-RC CYCLES. CONTROL BEGINS.

THREE CA-RC CYCLES

A CA-RC CYCLE. CONTROL INDICATES READY FOR REFERENCE ADDRESS

THREE CA-RC CYCLES.

CONTROL INITIATES PERIPHERAL. FILLS ITS BUFFER FROM THE PERIPHERAL. WHEN DONE, RAISES SERVICE REQUEST.

CA-RC CYCLE. CONTROL SETS ITS BIT IN MASK

THREE CA-RC CYCLES. CONTROL PROVIDES REFERENCE ADDRESS.

A CA-RC CYCLE. CONTROL INDICATES DATA TRANSFER. DATA IS MOVED, ONE BYTE PER CA-RC CYCLE

CONTROL INDICATES LAST BYTE

A CA-RC CYCLE. CONTROL INDICATES READY TO SEND RD.

THREE CA-RC CYCLES. CONTROL SENDS RESULT DESC.

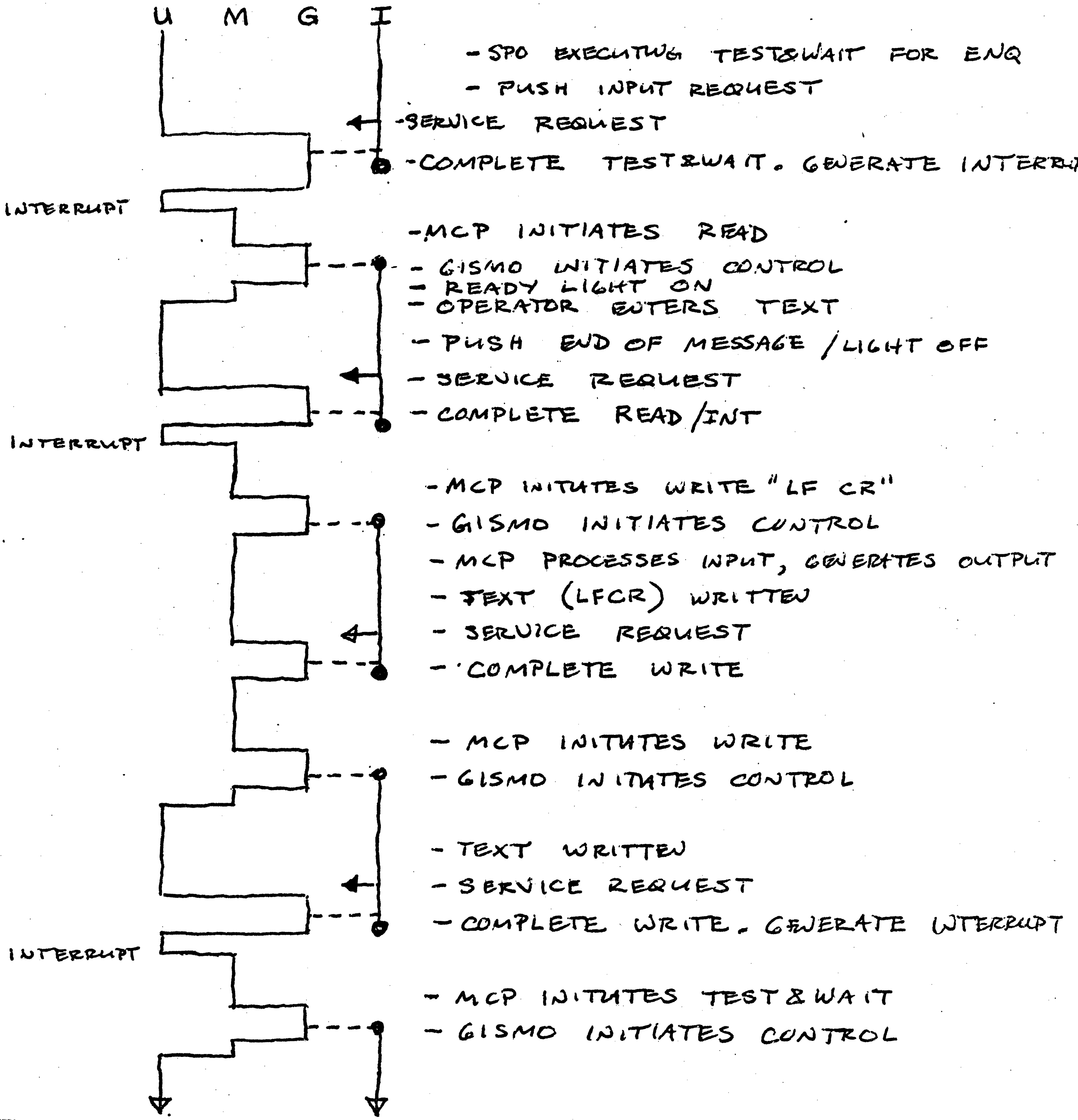
CONTROL IS FINISHED AND RETURNS TO STATUS 1

GISMO PROCESSES RESULT, MAY LINK TO NEXT IO OR RETURN TO MCP/USER.

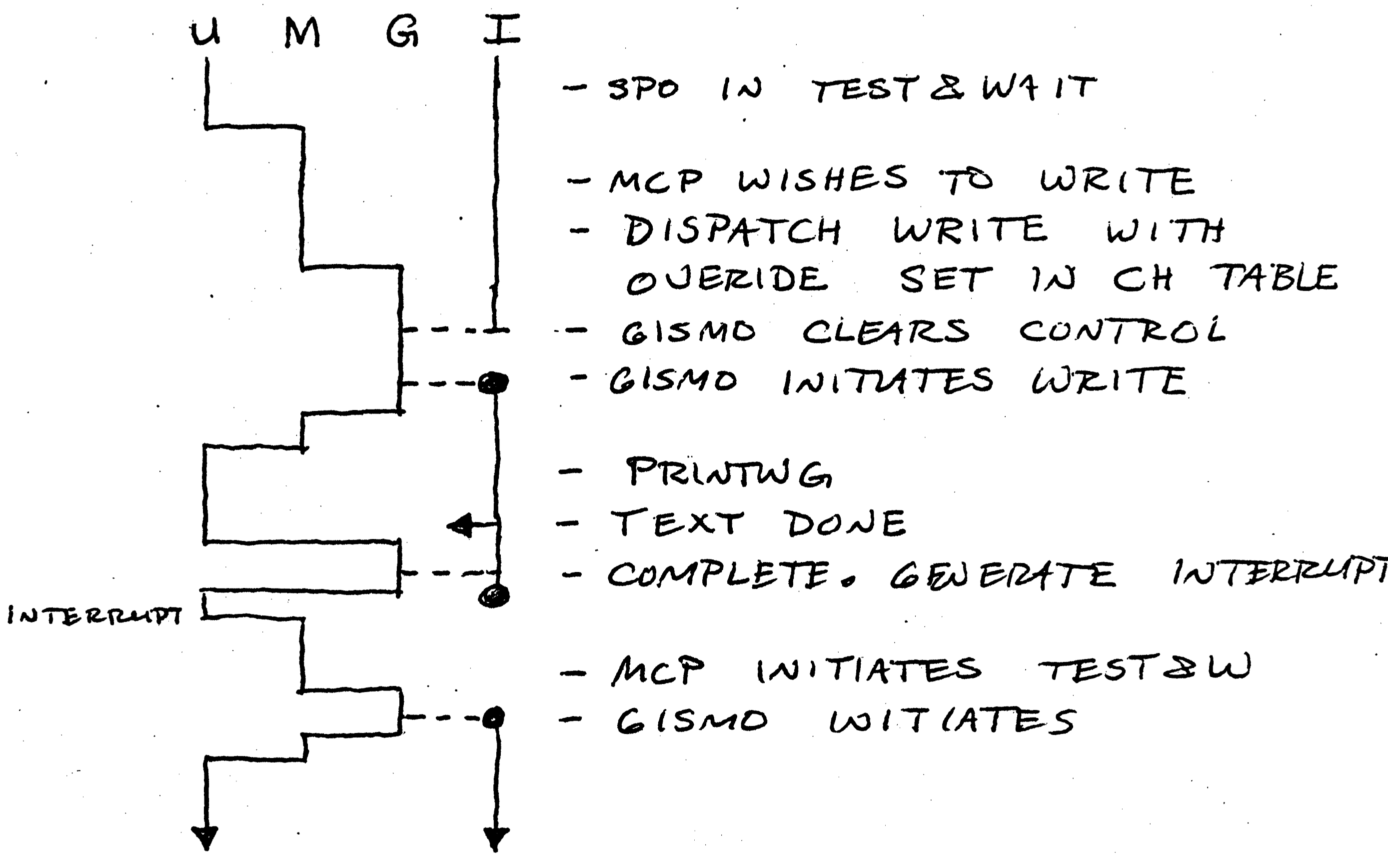
WRITE:

G ↔ I	STATUS	
	1	GISMO REQUESTED TO INITIATE.
TEST STATUS	1	A CA-RC CYCLE.
OP CODE	1,2,3	THREE CA-RC CYCLES. CONTROL BEGINS
	4	
FILE ADRS	4,5,6	THREE CA-RC CYCLES.
	14	
TEST STATUS	14	A CA-RC CYCLE. CONTROL INDICATES DATA TRANSFER.
DATA	14	DATA IS SENT TO CONTROL, ONE BYTE PER CA-RC CYCLE.
DATA	14	
⋮	14	
DATA	17	CONTROL INDICATES LAST BYTE
	7	
TEST STATUS	7	A CA-RC CYCLE. CONTROL INDICATES READY FOR REFERENCE ADDRESS
REF ADR	7,8,9	THREE CA-RC CYCLES. CONTROL RECEIVES AND SAVES REF ADR
	10	CONTROL "BUSY". INITIATES PERIPHERAL, SENDS THE DATA FROM ITS BUFFER. WHEN PERIPHERAL NOTIFIES THAT IT IS DONE, CONTROL RAISES
	18	SERVICE REQUEST
GET SER REQ MASK	18	A CA-RC CYCLE. CONTROL SETS ITS BIT IN MASK
	18	
REF ADR	18,19,20	THREE CA-RC CYCLES. CONTROL PROVIDES REFERENCE ADDRESS.
	21	
TEST STATUS	21	A CA-RC CYCLE. CONTROL INDICATES READY TO SEND RESULT DESC.
RESULT DESC	21,22,23	THREE CA-RC CYCLES, CONTROL SENDS RESULT DESC
	1	CONTROL IS FINISHED AND RETURNS TO STATUS 1.
	1	
		GISMO PROCESSES RESULT. MAY LINK TO NEXT IO OR RETURN TO MCP/USER

SPO: TYPICAL CASE OF INPUT



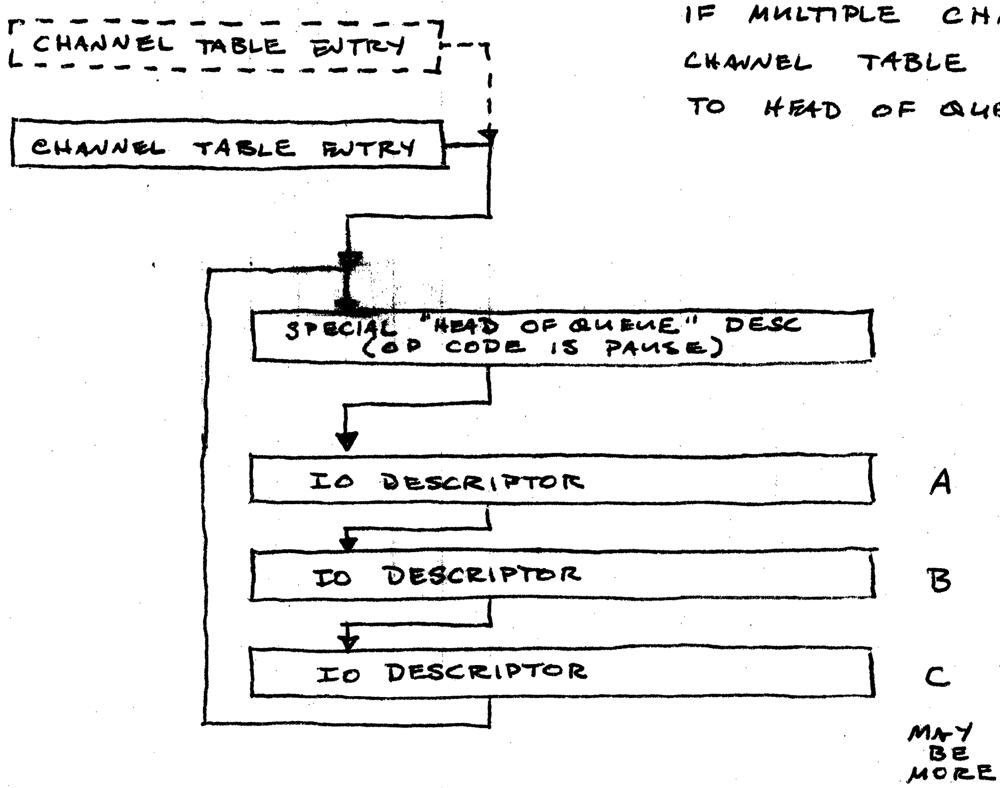
SPO: TYPICAL OUTPUT (UNSOLICITED)



NOTE: CANCELLATION OF IN PROCESS TEST & WAIT DURING DISPATCH

DISK QUEUE: ALL IO DESCRIPTORS FOR ALL DISK CHANNELS IN THE SYSTEM ARE IN THE SAME QUEUE.

IF MULTIPLE CHANNELS, EACH CHANNEL TABLE ENTRY POINTS TO HEAD OF QUEUE DESCRIPTOR.



DISK: MCP-GISMO - NEW STUFF

1) IF AN EXCEPTION OCCURS, DO NOT STOP, BUT INSTEAD, KEEP LINKING TO NEXT DESCRIPTOR IN CHAIN.
   SPECIFIED BY "KEEP-LINKING" BIT IN CH TABLE ENTRY.
   (NOTE: SUPPRESSES SETTING "EXCEPTION.IDLE" BIT)
2) IF A DESCRIPTOR WHICH IS NOT READY FOR EXECUTION IS ENCOUNTERED (RS STATUS BITS 01, 10, 11) LINK TO NEXT DESCRIPTOR IN CHAIN.
   SPECIFIED BY "KEEP, LINKING"
3) SINCE WE KEEP LINKING AS IN (1), (2), WE NEED TO KNOW WHEN WE'VE BEEN THROUGH THE WHOLE QUEUE; THEN WE CAN STOP.
   SPECIAL "HEAD OF QUEUE" DESCRIPTOR, POINTED TO BY ADDRESS FIELD OF CH TABLE ENTRIES FOR DISK. "KEEP LINKING" SPECIFIES THIS, AND MEANS THAT AT DISPATCH TIME (INITIATE REQUEST), WE DISCARD THE REFERENCE ADDRESS SENT BY THE MCP AND USE THE ADDRESS FROM THE CHANNEL TABLE ENTRY.

4) IN ORDER TO COPE WITH DISPATCHS ARRIVING IN AN ORDER DIFFERENT THAN THE ORDER THAT THE DESCRIPTORS ARE LINKED IN THE QUEUE, WE MUST BE ABLE TO OVERRIDE (3) AND NOT STOP UPON ENCOUNTERING HEAD OF QUEUE. (FOR EXAMPLE, IF A, B AND C ARE INITIALLY LOCKED, THEN B IS UNLOCKED AND DISPATCHED, WE WILL LINK TO AND INITIATE B. IF DURING THE TIME B IS IN PROCESS, A IS UNLOCKED AND DISPATCHED, AT COMPLETION OF B WE MUST LINK PAST C (LOCKED) AND THE HEAD IN ORDER TO FIND AND INITIATE A.

 ADD BIT IN CHANNEL TABLE (SET AT DISPATCH) CALLED "PENDING". IF WE ENCOUNTER THE HEAD OF QUEUE AND IT IS RESET, THEN WE STOP. IF IT IS SET THEN WE RESET IT AND LINK TO NEXT DESC.

5) ALL IO DESCRIPTORS FOR DIFFERENT CHANNELS IN SAME Q.

 ~~FOR~~ UPON ENCOUNTERING A READY DESCRIPTOR, CHECK CHANNEL FIELD OF RS. IF WRONG THEN LINK TO NEXT.

NEW GISMO CODE FOR MCP-GISMO DISK.

- DISPATCH
  - SET PENDING BIT IN CHANNEL TABLE ENTRY
  - IF KEEP-LINKING THEN USE REF.ADR FROM CH TABLE.
- GET A DESCRIPTOR
  - IF RS BITS NOT READY (00) THEN
    - LINK TO NEXT DESCRIPTOR
- ANALYZE OP CODE
  - IF CHANNEL FROM RS NEQ CHANNEL WE ARE ON THEN
    - UNLOCK (00) RS BITS AND LINK TO NEXT DESC
  - IF SPECIAL DESC (PAUSE OP - HEAD OF QUEUE) THEN
    - IF PENDING BIT THEN
      - RESET PENDING BIT
      - LINK TO NEXT DESC
    - GO TO NEXT-SERVICE
- ANALYZE RESULT
  - IF KEEP-LINKING THEN
    - IF EXCEPTION THEN SUPPRESS SETTING EXCEPTION-IDLE
    - LINK TO NEXT DESCRIPTOR

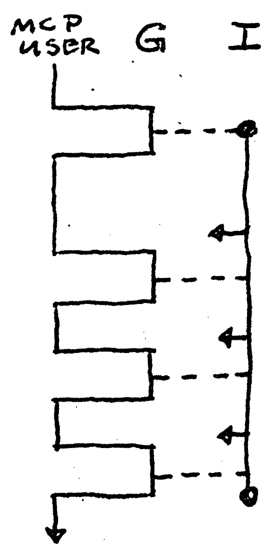
DISK: GISMO - CONTROL

CONTROLS	TRANSFER WIDTH	EXCH	ARM	NOTIFY SEEK COMP	T&W	PAUSE
HPT	16	YES	N	—	—	—
DC 1	16	N	YES	N	N	YES
DC 2	16	N	YES	YES	YES	N
PACK	24	N	YES	YES	YES	N
5N HPT	24	N	N	—	—	—

ALL HAVE SOME NUMBER OF 180 CHARACTER BUFFERS (LONGER FOR DIAGNOSTIC STUFF).
ALL ARE MULTIPLE SERVICE REQUEST TYPES.
ALL ALLOW (REQUIRE) ~~USE~~ GISMO TO TERMINATE.

DATA WIDTH AND TERMINATE SEQUENCE HANDLED BY IO.DATA ROUTINE. "24" TYPES ARE OPTIONAL GISMO SEGMENT.

HPT, INCLUDING 5N: TYPICAL READ



IO REQUEST
GISMO INITIATES
LATENCY TIME
SERVICE REQUEST
180 CHARS DATA
SER REQ
180 CHARS DATA
SER REQ
≤180 CHARS DATA: GISMO SENDS "TERMINATE" COMMAN
RESULT DESC

DISK: GISMO - CONTROL

ARM MOVERS: (IF ARM IS IN CORRECT CYLINDER, OP PROCEEDS AS IN HPT CHART)

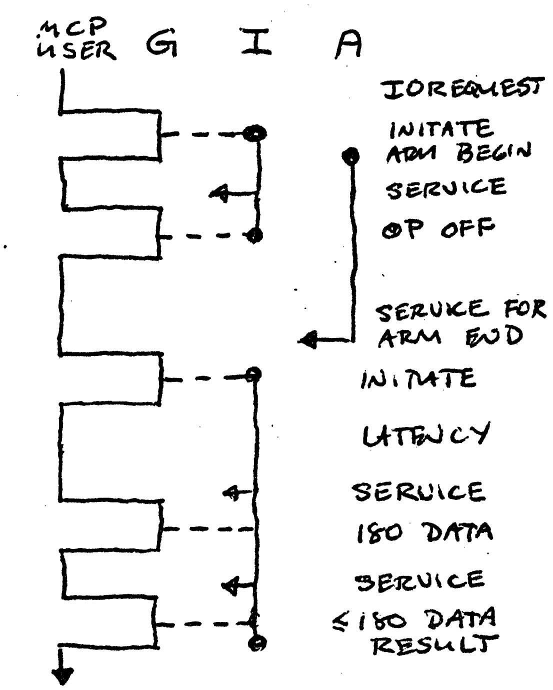
ALL USE SUPRESSION OF ~~B~~ SECOND OP COMPLETE BIT TO INDICATE SEEKING, ~~OR~~ WRONG CYLINDER ~~OR~~ ~~FEST 2 WAIT COMPLETION~~

SEEKING: ARM IN MOTION

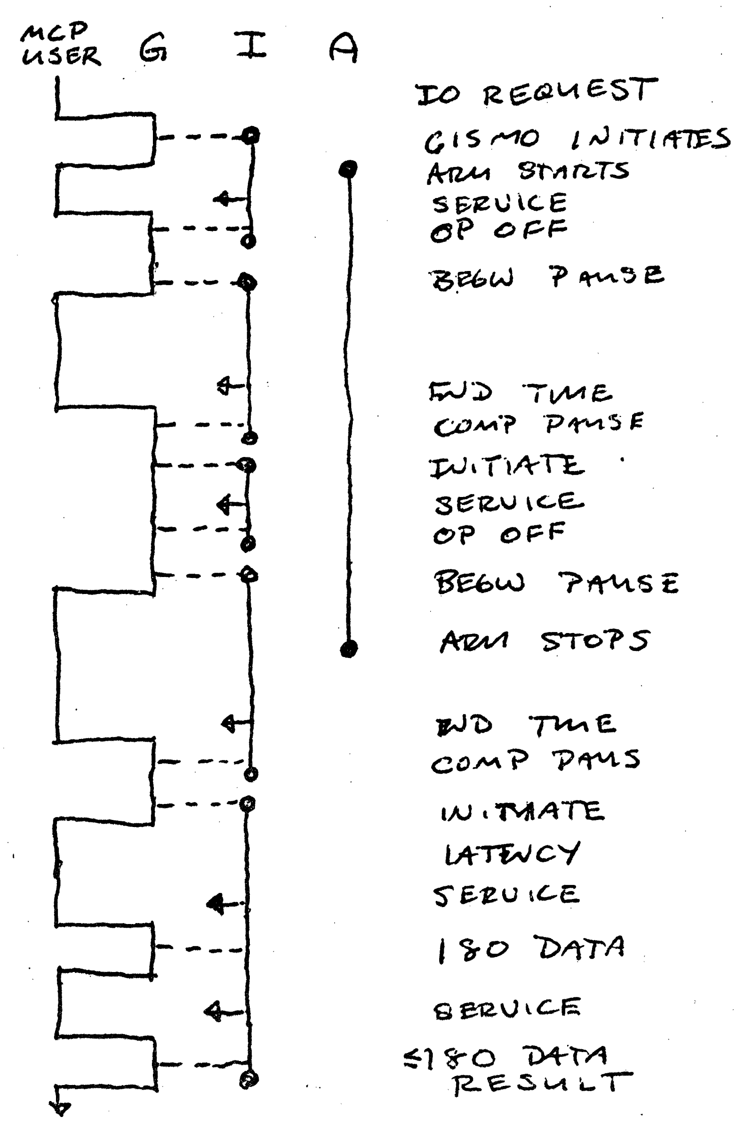
WRONG CYLINDER: WE JUST SOUGHT, BUT THIS IO IS NOT FOR THAT TRACK (CYLINDER)

ALL ARE INITATED BY STANDARD WAY
DC-2, PACK NOTIFY GISMO OF SEEK COMPLETE
DC-1 DOES NOT NOTIFY

DC-2, PACK: READ



DC-1: READ



DISK: GISMO - CONTROL

NEW RULES FOR GISMO

ANALZE OP CODE
- IF SPECIAL.DESCRIPTOR THEN
  - IF PAUSE BIT IN CH TABLE THEN
    - RESET PAUSE BIT
    - CAUSE EXECUTION OF PAUSE OP, LEAVING SPECIAL DESC READY (00).

NEXT.SERVICE
- GET STATUS
- IF STATUS EOL 1 THEN
  - SET BUSY, PENDING IN CH TABLE
  - GET REF.ADR FROM TABLE
  - GO TO RESET CHANNEL
- TRANSFER IN REFERENCE ADDRESS
- IF RS BITS NOT 01 THEN (END OF PAUSE)
  - TRANSFER IN RD AND DISCARD
  - LINK TO NEXT DESCRIPTOR

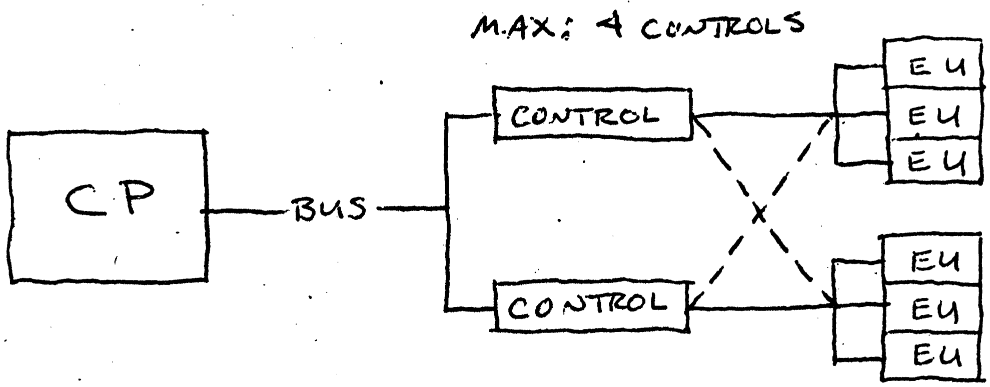
ANALYZE RESULT
- IF SECOND.OP.OFF THEN
  - IF DC-1 THEN
    - SET PAUSE BIT IN CH TABLE
  - UNLOCK RS BITS
  - LINK TO NEXT DESCRIPTOR

DC-2 AND DISKPACK WILL SUPPRESS SECOND OP COMPLETE BIT IN RESULT DESC IF OP IS TEST & WAIT AND CONDITION IS NOT MET.

NEW RULE FOR TEST & WAITS
- AT TIMER INTERRUPT TIME INDEX THROUGH CHANNEL TABLE LOOKING FOR ENTRY WITH BUSY OFF AND TIMER ON.
  - IF FOUND THEN DISPATCH, USING REF.ADR FROM TABLE

NOTE: DEVICES WHICH HAVE TIMER SET (DC-2, PACK, ALL TAPE) ARE A SUBSET OF DEVICES WITH KEEP.LINKING SET (ALL DISK, ALL TAPE).

EXCHANGE: DISK



--- INDICATES PATH ADDED BY EXCHANGE

EXCH   MCP-GISMO

MCP WILL INITIALIZE CHANNEL TABLE INDICATING PRESCENCE OF AND POSSIBLE PATHS ON EXCHANGE. SUBSEQUENT TO INITIALIZATION, MCP WILL DIRECT ALL DISPATCHS TO LOWEST CHANNEL ON EXCHANGE (CALLED PRIMARY CHANNEL) AND GISMO WILL PROVIDE ALTERNATE PATH SELECTION WHEN PRIMARY CHANNEL BUSY. MCP ALSO SETS CHANNEL IN RS FIELD OF ALL DESCRIPTORS FOR EXCH TO PRIMARY CHANNEL.

PRESCENCE AND PATHS ARE SPECIFIED BY TWO FIELDS IN CHANNEL TABLE ~~ENTRY~~ ENTRY(S):

EXCHANGE LINK BIT: WHEN SET, THE EXCHANGE LINK FIELD OF THIS ENTRY CONTAINS PORT*/CHANNEL OF NEXT ON EXCHANGE. LAST ENTRY IN EXCHANGE HAS THIS BIT RESET.

EXCHANGE LINK FIELD: CONTAINS PORT*/CHANNEL OF NEXT ON EXCHANGE.

* ALWAYS 7, (SOFT.IO)

NEW GISMO STUFF: MCP-GISMO

DISPATCH
- IF CHANNEL BUSY THEN
  - IF EXCH.LINK.BIT THEN
    - ITERATE THRU DISPATCH, USING CHANNEL FROM EXCH LINK FIELD.

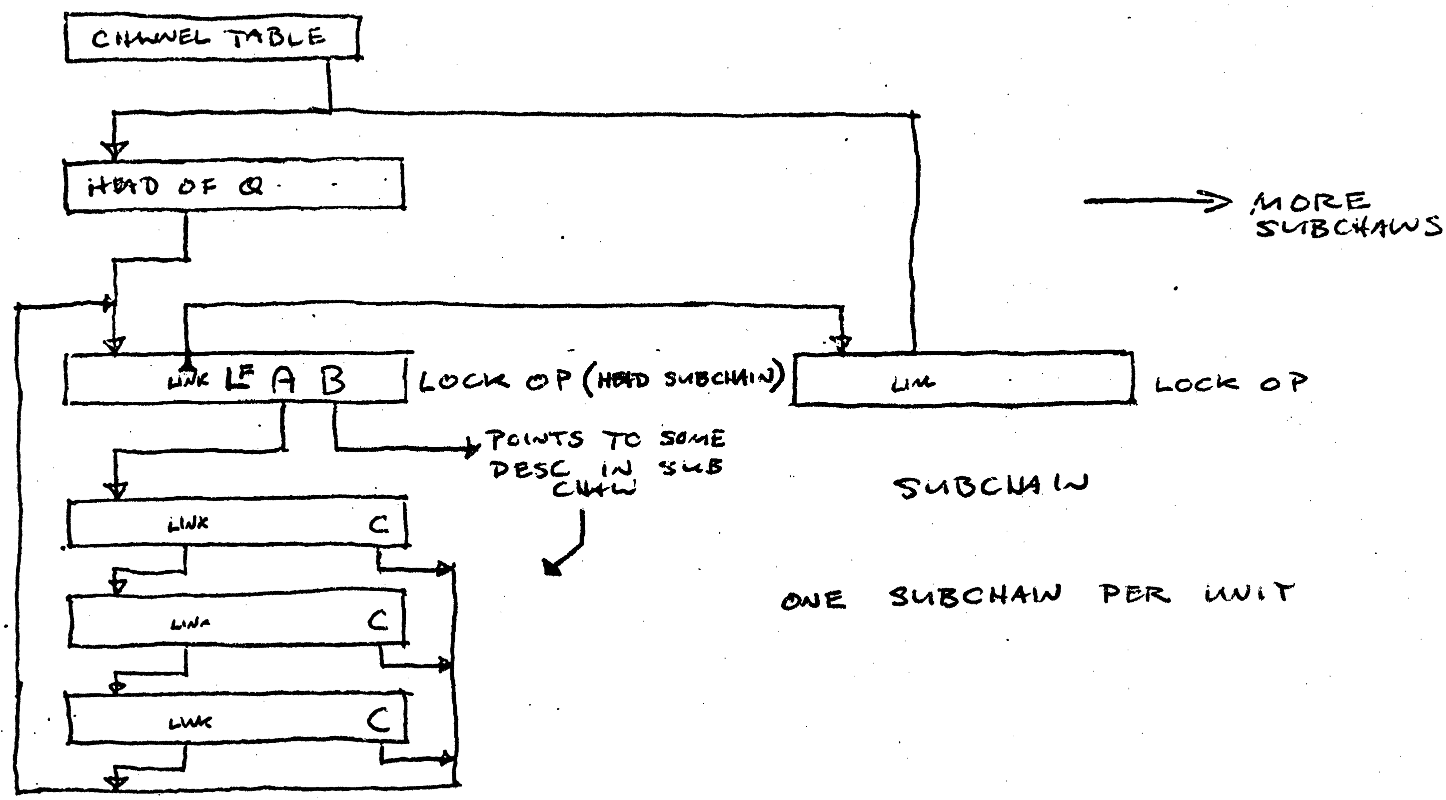
ANALYZE OP CODE
    IF CHANNEL WE ARE ON IS NOT CHANNEL IN RS FIELD
        LINK THROUGH EXCHANGE, STARTING AT CHANNEL
        TABLE ENTRY FROM RS (PRIMARY) TO SEE
        IF CHANNEL WE ARE ON IS IN EXCHANGE
        IF NOT IN EXCHANGE THEN
            UNLOCK RS (00)
            LINK TO NEXT DESCRIPTOR

EXCHANGE: DISK   GISMO - IO SYSTEM

GISMO DOES NOT KEEP TRACK OF WHAT UNITS ARE CURRENTLY IN PROCESS. IF AN IO IS IN PROCESS THROUGH ONE CHANNEL AND AN OP IS INITIATED THROUGH THE OTHER CHANNEL TO THE SAME UNIT, THE CONTROL WILL TERMINATE THE SECOND INITIATION AND RETURN A RESULT WITH THE SECOND OP COMPLETE BIT SUPPRESSED (EU BUSY). GISMO'S RESPONSE TO THIS INDICATION WILL BE TO LINK TO THE NEXT DESCRIPTOR, ALLOWING THE OTHER CONTROL, WHEN IT IS FINISHED ITS PRESENT OP, TO LINK TO AND RE-INITIATE THIS DESCRIPTOR. KEEPING IN MIND THE PROBLEM REFERRED TO ON PAGE 19, ITEM (4), THE FOLLOWING CODE IN GISMO IS ADDED:

NEW GISMO STUFF: GISMO - IO SYSTEM
    ANALYZE RESULT
        IF SECOND OP OFF THEN
            IF CHANNEL IS IN EXCHANGE THEN
                (EU BUSY) LINK THROUGH CHANNEL TABLE USING EXCH LINKS, UNTIL AN ENTRY WITH THE BUSY BIT SET IS FOUND. IN THAT ENTRY, SET THE PENDING BIT.
                UNLOCK RS
                LINK TO NEXT DESCRIPTOR.

TAPE CHAIN



ALLOWS SUBCHAINS AS WHOLES TO BE HANDLED IN ANY ORDER.
ALLOWS LINKING WITHIN SUBCHAIN IN SERIAL ORDER.

SHARES SOME OF STUFF IMPLEMENTED FOR DISK
CH TABLE: PENDING, EXCH, KEEP LINKING, TIMER, EXCH LINK, ADR

NEW OP CODE: LOCK. INDICATES HEAD OF SUBCHAIN

ADDITIONAL GISMO TOGGLE IN RS FIELD: LOCKED STATE

TAPE: GISMO - CONTROL

CONTROL WILL:

SET A TOGGLE INDICATING REVERSE IN EVERY STATUS REPORT AFTER RECEIVING OP CODE.

SET A TOGGLE AND ACCEPT A SPECIAL COMMAND FOR HANDLING SINGLE BYTE OF DATA IN OR OUT.

SUPRESS THE SECOND OP COMPLETE BIT IF CONDITIONS ARE NOT MET ON A TEST & WAIT.

NEW GISMO RULES

```
GET, DESC
    IF NOT READY THEN % RS BITS)
        IF NOT LOCKED, STATE THEN
            LINK TO NEXT DESCRIPTOR
        IF RS IS 10 OR 11 THEN
            EXIT SUBCHAN USING C.ADR
            SET B,ADR IN LOCK OP (HEAD SUBCHAN)
                TO POINT TO NOT READY DESC
            UNLOCK LOCK OP AND RESET LOCKED STATE
            LINK TO NEXT DESC (NEXT LOCK OP)
        IF RS IS 01 THEN (AT LOCK OP)
            SET B,ADR TO A,ADR OF LOCK OP
            UNLOCK LOCK OP AND RESET LOCKED STATE
            LINK TO NEXT DESCRIPTOR
ANLYZE OP
    IF OP EQL LOCK OP THEN
        SET LOCKED STATE
        USE B,ADR TO LINK TO NEXT DESC

EXECUTE DESC
    IF REVERSE TOG (IN STATUS REPORT RC) THEN
        SET E,ADR TO B,ADR
```

ANALYZE RESULT
IF SECOND OP OFF THEN (A TEST & WAIT: CONDITION NOT MET)
UNLOCK RS
EXIT SUBCHAN USING C,ADR
SET B,ADR TO REF,ADR OF THAT DESC
UNLOCK LOCK OP RS AND RESET LOCKED STATE
LINK TO NEXT DESC.
IF EXCEPTION THEN
~~UNLK~~ EXIT SUBCHAN USING C,ADR
SET B,ADR TO REF,ADR OF THAT DESC
UNLOCK LOCK RS AND RESET LOCKED STATE
LINK TO NEXT DESC

IO,DATA
COPE WITH ODD BYTE TRANSFERS AND REVERSE OPERATIONS.
COPE WITH SPACE OP CODE,

TAPE EXCHANGE: SAME SORT OF PICTURE AS DISK EXCHANGE.
MCP INITIALIZES CHANNEL TABLE, THEN GISMO PROVIDES ALTERNATE PATH SELECTION (SAME AS DISK).

NO POSSIBILITY OF UNIT CONFLICT (AS EU BUSY ON DISK), SINCE ALL OPS FOR ANY UNIT ARE GROUPED TOGETHER IN ONE SUBCHAN; WHEN A CONTROL IS EXECUTING IN A SUBCHAN, THE LOCK OP DESC (HEAD OF SUBCHAN) IS MARKED IN PROCESS (01) AND THE OTHER CONTROL, IF INITIATED, WILL NOT ENTER SUBCHAN THAT IS NOT READY (00).

THE FOLLOWING IS A PSEUDO LISTING WHICH REPLACES THE ONE ON PAGES 6-7-8. PAGES 9-10 ARE THE SAME, AND ARE NOT DUPLICATED HERE. THE SAME KEY RULES AS ON PAGE 6 APPLY.

THIS LISTING REFLECTS THE ADDITION OF DISK, TAPE AND EXCHANGES. A FEW ADDITIONS HAVE ALSO BEEN MADE IN ANTICIPATION OF ~~[illegible]~~ DATA-COMM AND READER SORTER; THE BULK OF THAT CODE IS PRESENTED LATER.

```
DISPATCH TIME: HERE FROM S-OP (OR E.IO) (MCP-GISMO)          A - Z
    GET CHANNEL TABLE ENTRY
    IF NOT INITIALIZED THEN BYPASS RULES
    IF EXCEPTION + OVERRIDE THEN
        RESET BUSY, PENDING, EX.IDLE, OVERRIDE
    IF EXCEPTION.IDLE THEN
 EXCH.LINK
        IF EXCH THEN
            GET PORT/CHANNEL FROM EXCH-LINK
            GO TO DISPATCH TIME
        EXIT
    SET BUSY, PENDING
    IF IT WAS ALREADY BUSY GO TO EXCH.LINK
    IF KEEP LINKING THEN
        GET REF.ADR FROM TABLE (POINTS TO HEAD OF QUEUE)
```

```
RESET CHANNEL   (GISMO-IO SYSTEM)                            B - Z
    IF PORT NEQ 7 THEN "MISSING DEVICE" ERROR EXIT
    IF CHANNEL EQL 15 THEN "MISSING DEVICE" ERROR EXIT
    SET UP CHANNEL, REF.ADR IN SCRATCHPADS
    GET CONTROL STATUS & ID  %(CA-RC)
    IF WHOLE RC IS ZERO THEN "MISSING DEVICE" ERROR EXIT
    IF POCKET SELECT TOG THEN GO TO HANDLE.POCKET
    IF STATUS NEQ 1 THEN CLEAR CONTROL (CA-RC)
```

```
ANALYZE ID   (GISMO-GISMO)                                   C - Z
    SET UP LOCAL TOGGLES BY DECODING DEVICE ID.
    THEY WILL BE SAVED DURING EXECUTION IN
    RS FIELD OF DESCRIPTOR.
```

```
GET A DESCRIPTOR
    FETCH RS BITS
    IF NOT READY (00) THEN
        IF DISK TOG THEN
  LINK:     LINK TO NEXT DESCRIPTOR
        IF TAPE TOG THEN
            IF NOT LOCKED.STATE THEN  (LOOKING AT LOCK)
                GO TO LINK
            IF RS BITS 10 OR 11 THEN  (A LOCKED DESC)
                EXIT SUBCHAN USING C.FIELD
                SET B.ADR OF LOCK TO LOCKED.DESC
                UNLOCK LOCK DESC AND RESET LOCKED.STATE
                USING REF ADR OF LOCK GO TO LINK
            IF RS BITS 01 THEN  (LINKED BACK TO LOCK)
                SET B.ADR OF LOCK.OP TO A.ADR (HEAD SUBCHAN)
                UNLOCK LOCK DESC AND RESET LOCKED.STATE
                USING REF ADR OF LOCK GO TO LINK
```

```
        (A NOT READY DESC, ALL DEVICES BUT TAPE, DISK)
        FAKE UP PAUSE OP
        GO TO CHECK FOR PAUSE
    (THE DESCRIPTOR WAS READY (00))

    LOCK DESCRIPTOR (SET RS TO 01) AND SAVE TOGGLES
```

```
ANALYZE OP CODE
    IF  STOP.OP.CODE  THEN
        FAKE UP RD
        SET DONT.LINK
        GO TO ANALYZE RESULT
    IF  PAUSE OP CODE THEN (HEAD OF QUEUE: TAPE, DISK)
        UNLOCK RS BITS
        GO TO CHECK FOR PAUSE
    IF  DISK.TOG THEN
        IF CHANNEL WE ARE ON IS NOT CHANNEL IN RS THEN
            IF CHANNEL WE ARE ON IS NOT IN EXCHANGE THEN
                UNLOCK RS BITS
                GO TO LINK
    IF  TAPE.TOG THEN
        IF OP CODE IS LOCK OP THEN
            SET LOCKED STATE
            USING B.ADR, LINK TO NEXT I/O
    IF  DATA.COMM TOG THEN
        IF OP IS WRITE AUTO POLL THEN
            GO TO SKIP.E.ADR
    SET E.ADR TO A.ADR
SKIP-E.ADR
    GO TO EXECUTE DESC
```

EA-Z

```
                                                (NON DISK, TAPE)
CHECK FOR PAUSE: WE HAVE ENCOUNTERED NOT READY DESC OR PAUS
    IF  ~~SF~~ PAUSE BIT (DATA COMM) THEN
        FAKE UP PAUSE OP AND GO TO EXECUTE.DESC
    IF  TO.PAUSE THEN (DC-1) THEN
        RESET TO.PAUSE
        FAKE UP PAUSE OP AND GO TO EXECUTE.DESC
    IF  PENDING THEN
        RESET PENDING
        IF KEEP LINKING THEN (DISK, TAPE)
            LINK TO NEXT DESCRIPTOR
    RESET BUSY
    GO TO NEXT SERVICE
```

EXECUTE DESCRIPTOR F - Z

```
    TRANSFER OUT OP CODE      (CA-RC (3))
    TRANSFER OUT FILE ADR (C.ADR)  (CA-RC (3))
    IF REVERSE.TOG THEN (IN RC OF LAST CA-RC)
        SET E.ADR TO B.ADR (END OF BUFFER)
    CALL IO.DATA  (MAY OR MAY NOT MOVE DATA)
    (CONTROL ALWAYS IN STATUS 7)
    TRANSFER OUT REF.ADR  (CA-RC (3)) (CONTROL IN 10, BUSY)
    GO TO NEXT.SERVICE
```

G - Z

```
NEXT SERVICE: HERE FROM INTERP OR "ITERATE" GISMO
    GET SERVICE REQUEST MASK
    IF NONE THEN RESET SERVICE REQ THEN EXIT
    SELECT HIGHEST CHANNEL
    GET STATUS
    IF STATUS EQL 1 THEN (DC-2, PACK; SEEK COMPLETE)
        SET BUSY, PENDING
        USING REF.ADR FROM TABLE, GOTO RESET CHANNEL
    (CONTROL IN STATUS 11 OR 18)
    TRANSFER IN REF.ADR  (CA-RC (3))
    IF RS BITS NEQ 01 THEN  (END OF PAUSE)
        TRANSFER IN RD AND DISCARD  (CA-RC(3))
        IF DISK.TOG THEN (WAS A HEAD OF Q)
            LINK TO NEXT I/O (GO TO RESET CHANNEL)
        USING THIS REF.ADR, GO TO RESET CHANNEL (NOT READY)
    CALL IO.DATA (MAY OR MAY NOT MOVE DATA)
    GET STATUS (7, 10, 21)
    IF STATUS EQL 7 THEN
        TRANSFER OUT REF ADR
        GO TO NEXT.SERVICE
    IF STATUS NEQ 21 THEN
        GO TO NEXT.SERVICE
    TRANSFER IN RESULT DESC (CA-RC(3))
```

ANALYZE RESULT

```
    IF   SECOND.OP.OFF THEN  CALL  SECOND.OP.COMP.OFF
    IF   DATA.COMM.TOG  THEN
         (LONG RECORD MAY BE DETECTED IN IO.DATA AND TOC
          SET IN RS)
         MAYBE SET "NO ETX" AND EXCEPTION
    STORE RESULT IN RS FIELD OF IO DESC
    IF   EXCEPTION(RD) OR WANT.INTERRUPT (RS) THEN
         FORMAT AND ENTER INTERRUPT IN QUE
         IF EXCEPTION AND NOT KEEP.LINKING THEN
              SET EXCEPTION.IDLE
    IF   DONT.LINK  THEN
         RESET BUSY, PENDING
         GO TO NEXT SERVICE
    IF   TAPE.TOG AND EXCEPTION THEN
         EXIT SUBCHAN USING C.ADR
         SET B.ADR OF LOCK.OP TO EXCEPTION DESC
         UNLOCK RS OF LOCK AND RESET LOCKED STATE
         USING LINK FROM LOCK OP, LINK TO NEXT I/O
    IF   NOT DISK.TOG AND EXCEPTION THEN
         GO TO NEXT.SERVICE
    (NO EXCEPTION OR DISK AND EXCEPTION)
    LINK TO NEXT DESC
```

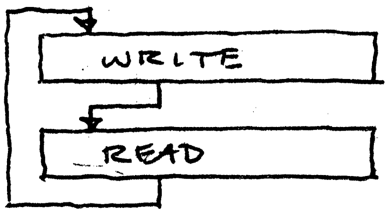
(NOTE: THE FOLLOWING ROUTINE DOES NOT CONTAIN DATA COMM OR READER SORTER. SEE PAGE A1)

SECOND.OP.COMPLETE.OFF HH-R

```
  IF   DISK.TOG  THEN
       IF   SPECIAL.RD THEN  SET SECOND.OP.COMP THEN EXIT
       IF   TEST&WAIT  THEN  GO TO NEXT.ONE
       IF   DC-1 THEN  SET TO.PAUSE
       IF   EXCH THEN
            FWD BUSY CONTROL (CH TABLE ENTRY) AND SET ITS PENDING
NEXT.ONE
       REMOVE RETURN ADR
       UNLOCK RS
       USING LINK, GO TO RESET CHANNEL (LINK TO NEXT)
  IF   TAPE.TOG THEN
       UNLOCK RS FIELD
       EXIT SUBCHAN USING C.ADR
       SET B.ADR OF LOCK DESC TO POINT TO THIS DESC
       UNLOCK LOCK DESC AND RESET LOCKED.STATE
```

DATA COMM – SINGLE LINE CONTROL

ALL DATA COMM I/O ON REQUESTOR SIDE HANDLED BY SPECIAL ROUTINES DISTINCT FROM MCP (HERE CALLED HANDLER). THIS HANDLER ALLOCATES ITS OWN IO DESCRIPTORS.
TYPICALLY, USAGE INVOLVES A WRITE DESCRIPTOR LINKED TO A READ, WHICH IS LINKED TO THE WRITE.



IF THE HANDLER WISHES TO WRITE A MESSAGE, IT WILL FOLLOW THESE STEPS:

HANDLER

1) PLACE A SELECT SEQUENCE IN THE DATA AREA OF THE WRITE.
2) UNLOCK BOTH (AND FIX RS) (TO REQUEST INTERRUPT ON READ)
3) DISPATCH THE WRITE

GISMO

4) INITIATE THE WRITE.
5) IF AT COMPLETION OF THE WRITE THERE IS EXCEPTION, GENERATE INTERRUPT AND STOP LINKING.
6) IF NO EXCEPTION, LINK TO AND INITIATE READ.
7) IF AT COMPLETION OF READ, THERE IS EXCEPTION, GENERATE INTERRUPT AND STOP LINKING.
8) IF NO EXCEPTION, GENERATE INTERRUPT THEN LINK. THE WRITE IS LOCKED, SO INITIATE PAUSE. AT TERMINATION OF PAUSE, RE-EXAMINE WRITE; IF UNLOCKED, INITIATE IT. IF LOCKED, PAUSE AGAIN.

HANDLER	GISMO
9) AT SUCCESSFUL TERMINATION OF READ, HANDLER LOOKS AT DATA FOR "ACK"KNOWLEDGMENT OF REQUEST BY TERMINAL. IF SO:	
10) PLACE TEXT FOR WRITE INTO WRITE AREA.	
11) UNLOCK READ DESC, REQUEST INTERRUPT	
12) UNLOCK WRITE DESC	
	13) AT THE COMPLETION OF SOME PAUSE, WE SEE THE WRITE UNLOCKED SO WE INITIATE IT.
	14) AT COMPLETION OF WRITE, IF EXCEPTION THEN GENERATE INTERRUPT AND STOP LINKING.
	15) IF NO EXCEPTION THEN LINK TO AND INITIATE READ
	16) AT COMPLETION OF READ, GENERATE INTERRUPT. IF EXCEPTION THEN STOP LINKING. IF NO EXCEPTION THEN LINK TO WRITE. IT IS LOCKED, SO INITIATE PAUSING, AS IN (8).
17) AT SUCCESSFUL TERMINATION OF READ, HANDLER LOOKS AT DATA FOR "ACK"NOWLEDGEMENT BY TERMINAL OF RECEIPT OF MESSAGE.	

AT THIS POINT, GISMO IS PAUSING ON THE WRITE. TO TERMINATE THIS, THE HANDLER MAY CHANGE THE WRITE OP TO A STOP OP AND UNLOCK THE RS.

IF THE HANDLER WISHES TO READ FROM THE TERMINAL, IT FOLLOWS THE SAME STEPS AS OUTLINED ABOVE, ONLY THE DATA BEING DIFFERENT:

- FIRST WRITE: CONTROL CHARS "ANY INPUT?"
- FIRST READ: CONTROL CHAR "NO" OR MESSAGE TEXT
- SECOND WRITE: ACKNOWLEDGE RECEIPT (CONTROL CHARS)
- SECOND READ: END OF TRANS (CONTROL CHARS).

TO PROVIDE THE ABOVE, NOTHING NEW NEED BE ADDED TO GISMO. WE USE EXISTING LINKING MECHANISMS ET AL:

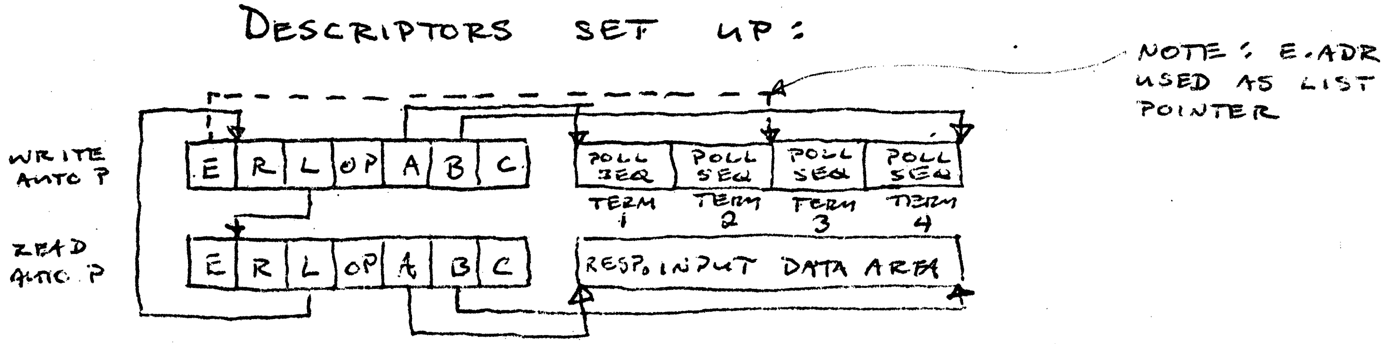
PAUSE BIT SET IN CHANNEL TABLE ENTRY (PAUSE ON LOCKED)
EXCEPTION OVERRIDE TO CANCEL IN PROCESS IO.
(AS, FOR EXAMPLE, WHEN TERMINAL DOESNT KNOW HOW TO SAY "NO" TO INPUT REQUEST, THE READ STAYS ACTIVE UNTIL TIMEOUT OCCURS (OR INDEFINITLY)).
STOP OPS AND EXCEPTIONS TERMINATE LINKING.
STOP OPS TO TERMINATE OR INHIBIT PAUSING.
INTERRUPTS GENERATED WHEN REQUESTED AND ON EXCEPTIONS.

AT LEVEL 4.2 GISMO WILL DETECT AND REPORT AN ERROR CONDITION INDICATING MORE DATA RECEIVED FROM CONTROL THAN WOULD FIT IN MEMORY BUFFER.

AN ADDITIONAL FUNCTION IS PROVIDED: THE ABILITY TO CONTINUALLY POLL A SERIES OF TERMINALS LOOKING FOR ONE THAT HAS INPUT. A POLL OPERATION CONSISTS OF SENDING A TERMINAL ADDRESS AND INQUIRY CHARACTERS (WRITE) TO THE CONTROL, THEN THE CONTROL (FROM THE TERMINAL) SENDING BACK (READ) A NEGATIVE RESPONSE (NAK) OR A MESSAGE FROM THE TERMINAL.

THE HANDLER WILL SET UP THE DESCRIPTORS AS SHOWN BELOW. TWO SPECIAL OPS ARE USED, WRITE AUTO POLL AND READ AUTO POLL.

DESCRIPTORS SET UP:



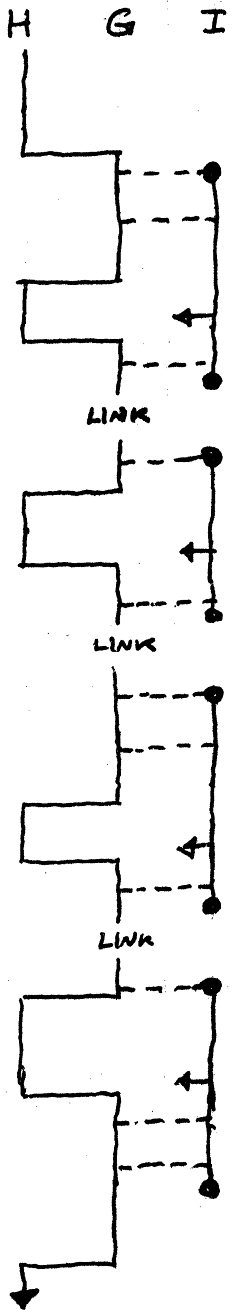
- THE CONTROL WILL ONLY ACCEPT THE NUMBER OF CHARACTERS ON A WRITE AUTO POLL AS DEFINED IN THE OP CODE (LENGTH OF ONE POLL SEQUENCE), THEN TERMINATE THE DATA TRANSFER.
- GISMO WILL, AT TERMINATION, STORE THE UPDATED DATA ADDRESS IN THE E,ADR FIELD (NOW POINTS TO NEXT POLL SEQUENCE).
- CONTROL WILL ALWAYS SUPRESS SECOND OP COMPLETE BIT FOR WRITE AUTO POLL.
- GISMO WILL RESPOND BY UNLOCKING THE RS BITS AND LINKING TO THE NEXT DESCRIPTOR.
- CONTROL WILL SET EXCEPTION AND SECOND OP COMPLETE BIT ON READ AUTO POLL IF MESSAGE RECEIVED.
- GISMO WILL GENERATE INTERRUPT AND TERMINATE LINKING
- CONTROL WILL SUPPRESS SECOND OP COMPLETE BIT ON READ AUTO POLL IF RESPONSE IS NEGATIVE (NO INPUT)
- GISMO WILL UNLOCK RS BITS AND LINK TO NEXT DESCRIPTOR (WRITE AUTO POLL)

GISMO MUST NOT SET THE E,ADR TO THE A,ADR AT INITIATE TIME AS IT USUALLY DOES; THE HANDLER MUST, PRIOR TO DISPATCH, SET THE E,ADR TO THE BEGGINING OF A POLL SEQUENCE (NEED NOT BE THE FIRST ONE IN THE BUFFER).

GISMO MUST NOTICE WHEN E,ADR (LIST POINTER) REACHES B,ADR (END LIST) AND RESET IT TO A,ADR (BEGIN LIST).

CONTROL TELLS GISMO WHICH SECOND OP COMPLETE SUPPRESSION BY SETTING BITS 22,23 IN RESULT TO 01 FOR WRITE AUTO POLL AND TO 00 FOR READ AUTO POLL.

AUTO POLL SEQUENCE



SET E.ADR TO BEGINNING OF A POLL SEQ
REQUEST INITIATE (DISPATCH)
GISMO INITIATES WRITE AUTO POLL
CONTROL ACCEPTS FIRST POLL SEQ.
GISMO UPDATES E.ADR (LIST POINTER)
SERVICE REQUEST
CONTROL SUPPRESSES SECOND OP COMPLETE
GISMO UNLOCKS RS AND LINKS
GISMO INITIATES READ AUTO POLL
(NO INPUT THIS TERMINAL)
SERVICE REQUEST
CONTROL SUPPRESSES SECOND OP COMPLETE
GISMO UNLOCKS RS AND LINKS
GISMO INITIATES WRITE AUTO POLL
CONTROL ACCEPTS SECOND POLL SEQUENCE
GISMO UPDATES E.ADR
SERVICE REQUEST
CONTROL SUPPRESSES SECOND OP COMPLETE
GISMO UNLOCKS RS AND LINKS
GISMO INITIATES READ AUTO POLL
(INPUT RECEIVED BY CONTROL FROM TERMINAL)
SERVICE REQUEST
DATA SENT TO GISMO (AND INTO MEMORY)
CONTROL SETS EXCEPTION AND SECOND OP COMPLETE
GISMO GENERATES INTERRUPT AND
TERMINATES LINKING.

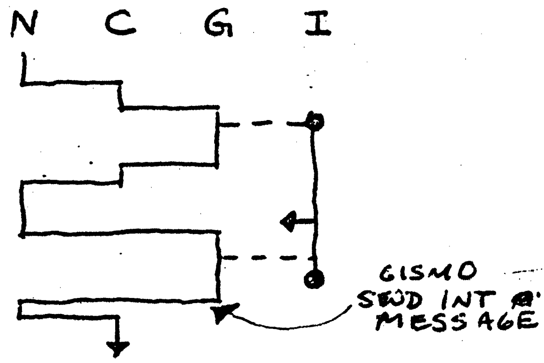
NEW GISMO CODE
- LONG RECORD
  - DETECTED IN IO.DATA AND BIT SET IN RS TOGGLES.
  - ANALYZE RESULT
    - IF RS.TOG THEN SET EXCEP AND "NO ETX" IN RD
- READ AUTO POLL
  - SECOND OP OFF (SEE PSEUDO LISTING)
- WRITE AUTO POLL
  - ANALYZE OP CODE
    - IF.WRITE.AUTO.POLL THEN
      - DONT SET E.ADR TO A.ADR

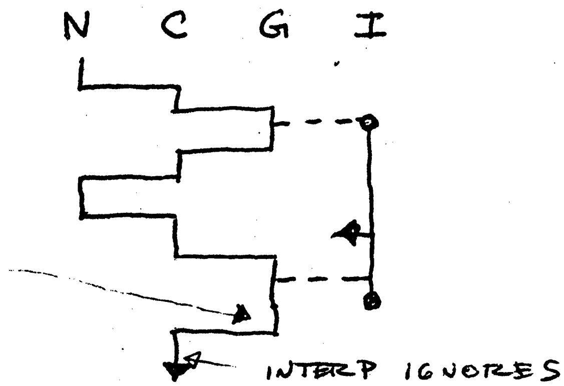
INTERRUPTS.

GISMO GENERATES THEM AND PASSES INDICATOR BACK TO INTERPRETER. INTERPRETER TRANSFERS TO MCP.

INTERRUPTS MAY BE DISABLED (ONLY IN CONTROL STATE). THIS MEANS GISMO WILL ALWAYS SEND BACK "NO-INTERRUPT".

IF SDL INTERP IS RUNNING MCP (CONTROL STATE) IT WILL NOT RECURSE TO ITSELF.



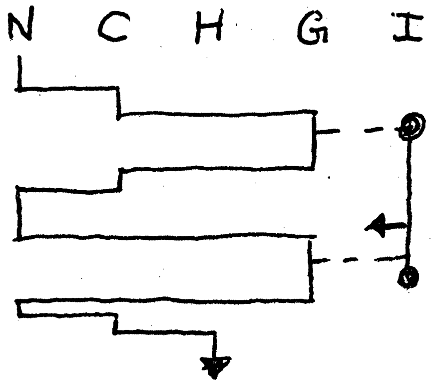


FOR REAL TIME DEVICE (READER SORTER) NEED TO SEIZE CONTROL FROM ANYONE WHO'S RUNNING.

ADD HI PRIORITY BIT TO RS AS INTERRUPT REQUEST QUALIFIER.

ADD KNOWLEDGE IN SDL INTERP AS TO LOCATION OF SPECIAL MCP HI PRIORITY INTERRUPT HANDLER.

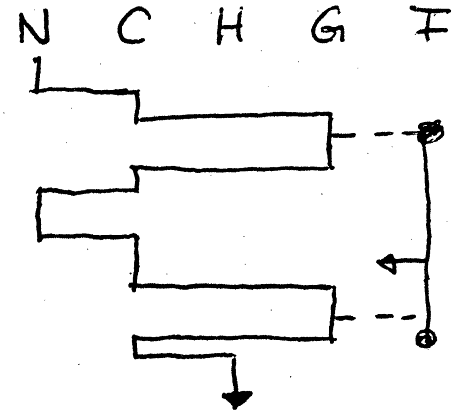
NOTE: TRAP TO HI PRI AUTOMATICALLY DISABLES INTERRUPTS.



GISMO SENDS HI PRI INDICATOR.

NORMAL STATE INTERP GOES TO CONTROL STATE.

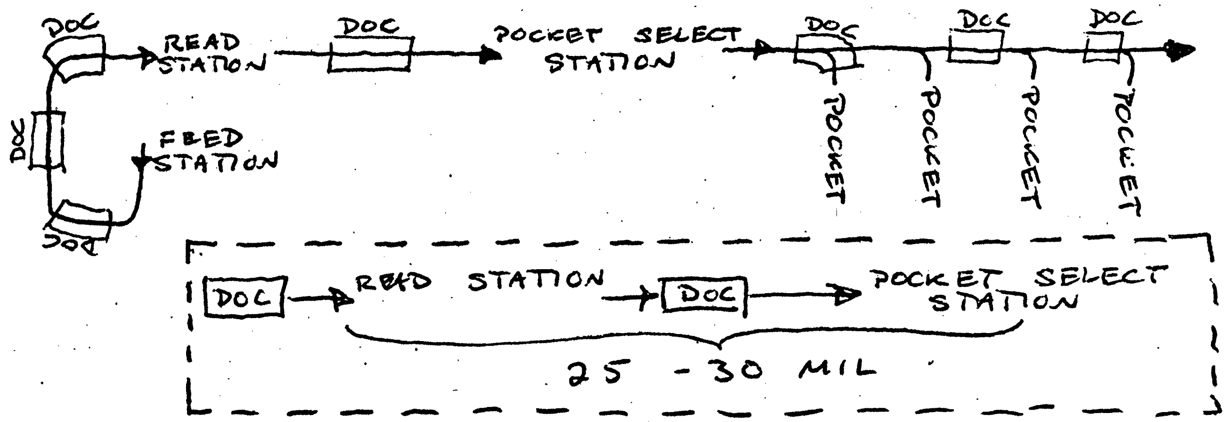
CONTROL TRAPS TO HI PRI ROUTINE



GISMO SENDS HI PRI INDICATOR.

CONTROL STATE TRAPS TO HI PRI ROUTINE

READER SORTER



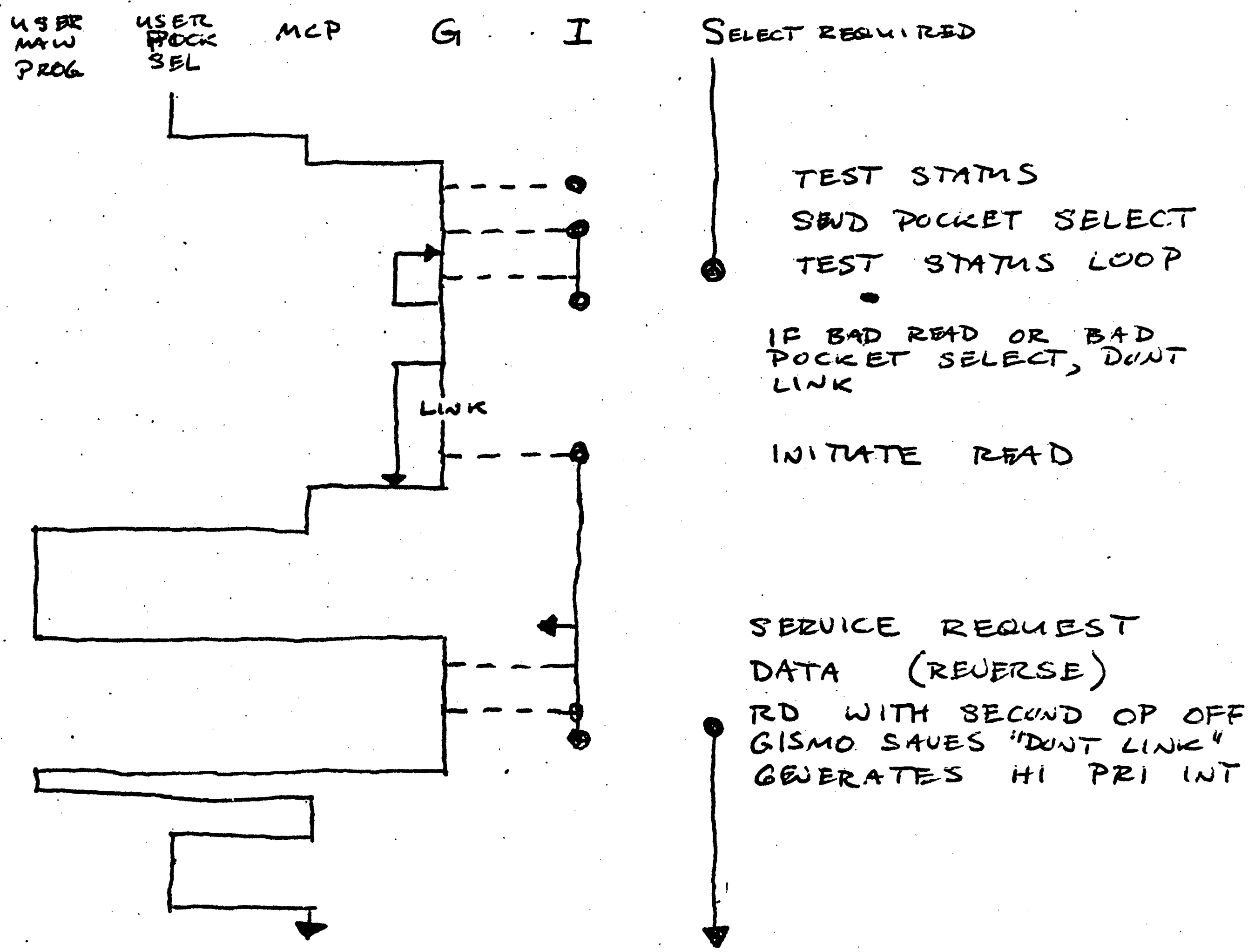
REAL TIME DEVICE
- SYSTEM MUST RESPOND IN FIXED AMOUNT OF TIME
- USER CODED POCKET SELECT ROUTINE RUNS INDEPENDENTLY OF MAIN PROGRAM.

USUALLY ON HIGHEST CHANNEL.
USE HI PRIORITY INTERRUPTS.

ONCE SORTER IS RUNNING (FLOW MODE) ANY READ INITIATE MUST ALSO COMMUNICATE POCKET SELECT INFORMATION FOR THE PREVIOUS DOCUMENT.

NEW GISMO RULES
- RESET CHANNEL
  - DETECT AND HANDLE POCKET SELECT REQUEST
- SECOND OP OFF (EXITS BACK TO ANALYZE RESULT)
  - SET SECOND OP COMPLETE
  - REMEMBER CERTAIN ERRORS WHICH OCCUR ON READ AND IMPLY NO LINKING AFTER POCKET SELECT
- ALWAYS READS REVERSE (IO, DATA)

NEW CONTROL STUFF
- STUFF REQUIRED FOR POCKET SELECT TRANSACTION.
- SUPRESS BIT 17; DECODE ERROR CONDITIONS AND ISOLATE "NO LINKING" CONDITION



EXCEPTION CONDITIONS

NO POCKET SELECT, NO LINK

- CONTROL: SET BIT 17, EXCEPTION, SUPPRESS POCKET SELECT REQUEST

POCKET SELECT BUT NO LINK

- CONTROL SUPPRESS BIT 17, SET NO LINK BIT.

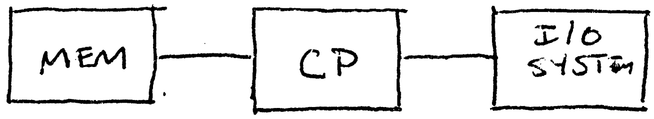
NOTE: THE FOLLOWING TWO ROUTINES COMPLETE THE PSEUDO LISTING FOR THIS CLASS (PAGES 28-32) "SECOND.OP.COMPLETE.OFF" REPLACES THE VERSION ON PAGE 32, AND "HANDLE.POCKET.SELECT" IS ADDITIONAL - Z

```
SECOND.OP.COMPLETE.OFF
    IF DISK.TOG THEN
        IF SPECIALORD THEN SET SECOND.OP.COMP THEN EXIT
        IF TEST&WAIT THEN GO TO NEXT.ONE
        IF DC-1 THEN SET TO.PAUSE
        IF EXCH THEN
            FWD BUSY CONTROL (CH TABLE ENTRY) AND SET ITS PENDING
NEXT.ONE
        REMOVE RETURN ADR
        UNLOCK RS
        USING LINK, GO TO RESET CHANNEL (LINK TO NEXT)
    IF TAPE.TOG THEN
        UNLOCK RS FIELD
        EXIT SUBCHAN USING C.ADR
        GET B.ADR OF LOCK DESC TO POINT TO THIS DESC
        UNLOCK LOCK DESC AND RESET LOCKED.STATE
        LINK TO NEXT DESC (FROM LOCK OP).

    IF READER.SORTER TOG THEN
        MOVE BIT 23 OF RESULT (DONT LINK AFTER POCK)
            TO BIT 16 OF C.FIELD (PLACE TO REMBER)
        SET SECOND.OP.COMPLETE BIT IN RD
        SET DONTOLINK
        EXIT
    IF DATA.COMM.TOG THEN
        IF RD BITS 22,23 EQL 00 THEN (READ AUTO POLL)
            UNLOCK RS
            EXIT
        IF RD BITS 22,23 EQL 01 THEN (WRITE AUTO POLL)
            UNLOCK RS
            IF E.ADR EQL B.ADR THEN
                SET E.ADR TO A.ADR
            EXIT
```

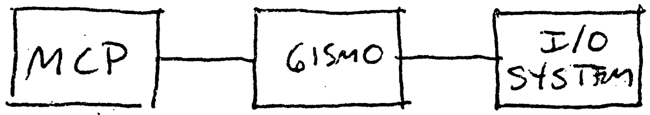
HANDLE POCKET SELECT

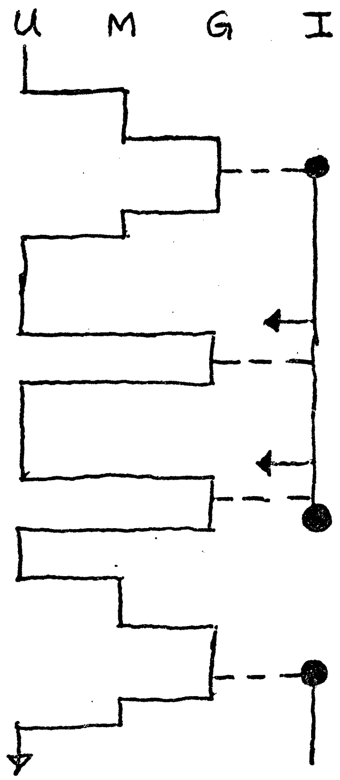
```
    FETCH POCKET SELECT INFO (FIRST 8 BITS OF C.FIELD)
    SEND IT TO CONTROL (CA-RC) (TRNSFER OUT)
LOOP
    GET.STATUS
    IF POCKET SELECT TOG THEN GO TO LOOP
    GET A BYTE FROM CHANNEL (CA-RC) (TRNSFER IN)
    STORE BYTE IN FIRST 8 OF C.FIELD
    GET "HAD BAD READ" BIT (BIT 16 OF C.FIELD)
    IF HAD BAD READ OR BAD POCKET SELECT THEN
        GO TO NEXT.SERVICE
    USE LINK FIELD OF THIS DESC AS REF.ADR
    GO TO RESET.CHANNEL
```

HARD



SOFT





USER : IO REQUEST

MCP : UNBLOCK PREV BUFF, INITIATE.

GISMO INITIATE CONTROL

USER PROCESS REC

IO WANTS SERVICE

GISMO

USER

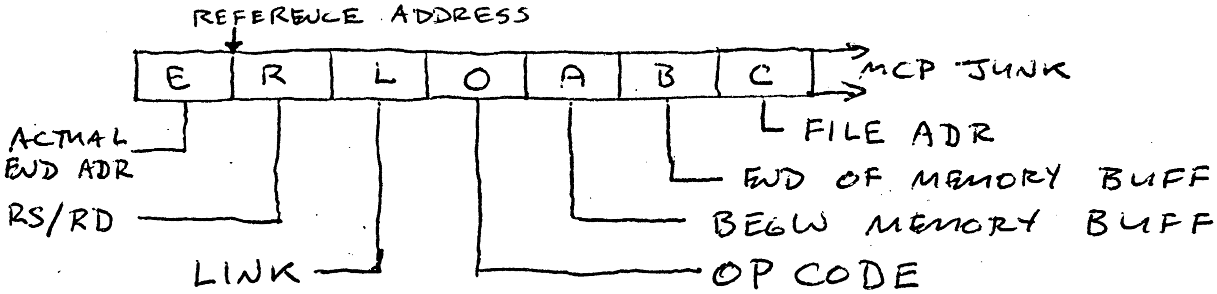
IO WANTS SERVICE

TERMINATE IO

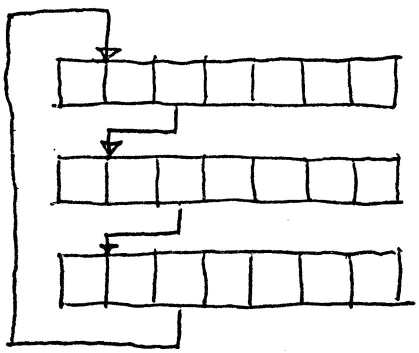
USER

IOREQUEST

IO DESCRIPTOR; 7 24 BIT FIELDS



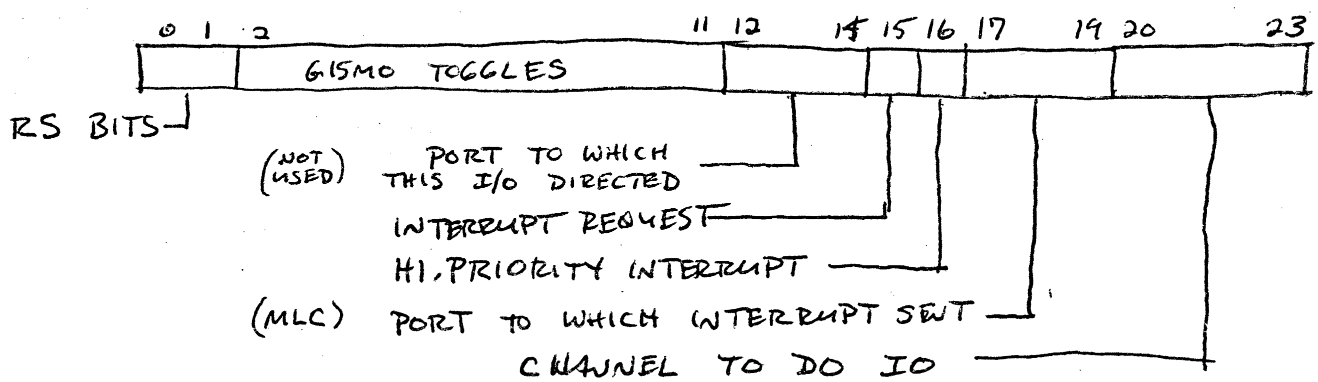
LINKED DESCRIPTORS



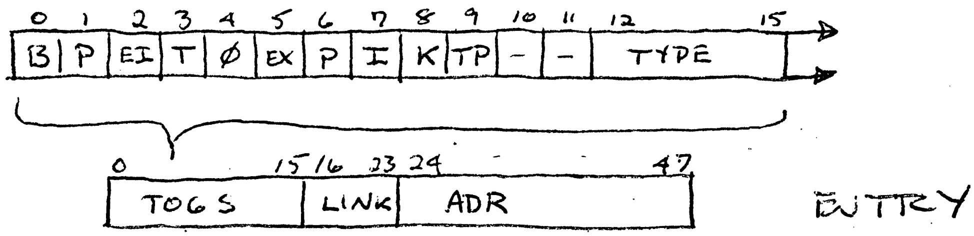
RS STATUS BITS

00 READY TO EXECUTE
01 IN PROCESS
10 LOCKED / IO COMPLETE
11 COMPLETE WITH EXCEPTION

RS FIELD



CHANNEL TABLE ENTRIES 48 BITS



B	BUSY: CONTROL IS ACTIVE
P	PENDING: LINK THRU PAUSE
EI	EXCEPTION IDLE: DISPATCH INHIBITED
T	ISSUE TIMER DISPATCH
Ø	EXCEPTION OVERRIDE
EX	LINK FIELD HAS P/C NEXT ON EXCH
P	SEND OUT PAUSE
I	INITIALIZED
K	KEEP LINKING ON EXCEPTION AND LOCKED DESCRIPTOR
TP	PAUSE THIS TIME

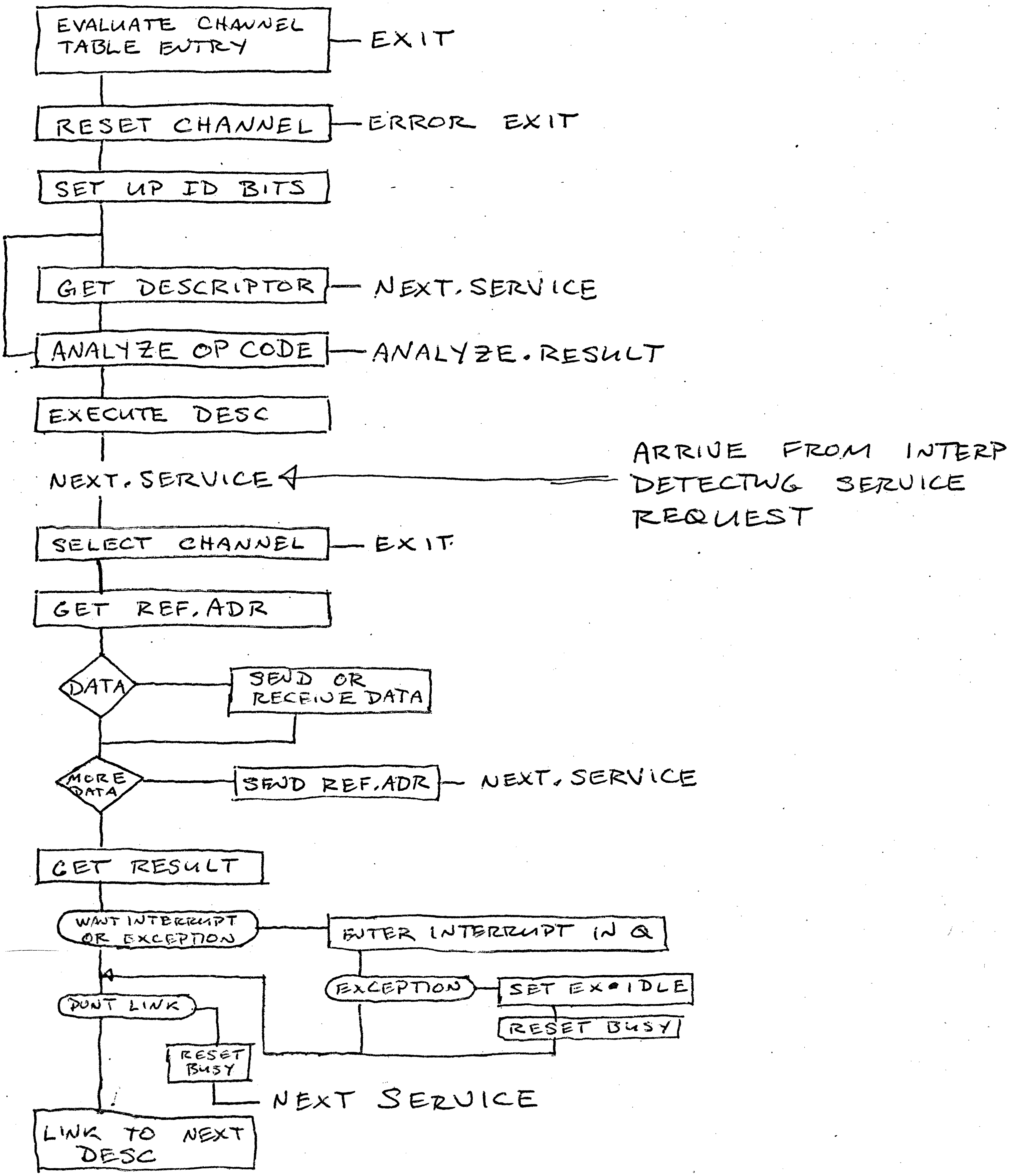
TYPE MCP-ANALYZER

LINK PORT/CHANNEL

ADR REFERENCE ADDRESS OF PAUSE OP AT HEAD OF CHAIN

R

MCP COMM THRU S-OP TO GISMO:
DISPATCH P/C, REF.ADR



A - R

DISPATCH TIME
GET CHANNEL TABLE ENTRY
IF NOT INITIALIZED THEN BYPASS RULES
IF EXCEPTION.OVERRIDE THEN
RESET BUSY, OVERRIDE, EXCEPTION IDLE
IF NOT BUSY THEN
GO DO.IT

EXIT
DO.IT

RESET CHANNEL AB - R

IF PORT NEQ 7 THEN
MISSING DEVICE
IF CHANNEL EQL 15 THEN
MISSING DEVICE
SET UP CHANNEL, REF.ADR
GET CONTROL STATUS & ID
IF STATUS EQL 0 THEN
MISSING DEVICE

IF STATUS NEQ 1 THEN
CLEAR CONTROL

ANALYZE ID C - Z
SET UP LOCAL TOGGLES BY
DECODING DEVICE ID FROM CONTROL

D-R

```
GET.A.DESC
    FETCH RS
    IF NOT READY THEN
    BEGIN
        GO TO NEXT SERVICE
    END
    LOCK RS % 01
```

E-R

```
ANALYZE OP
    IF STOP.CODE THEN
    BEGIN
        FAKE UP RD
        SET DONT.LINK
        GO TO ANALYZE.RESULT
    END
    SET E.ADR TO .A.ADR
```

F-R

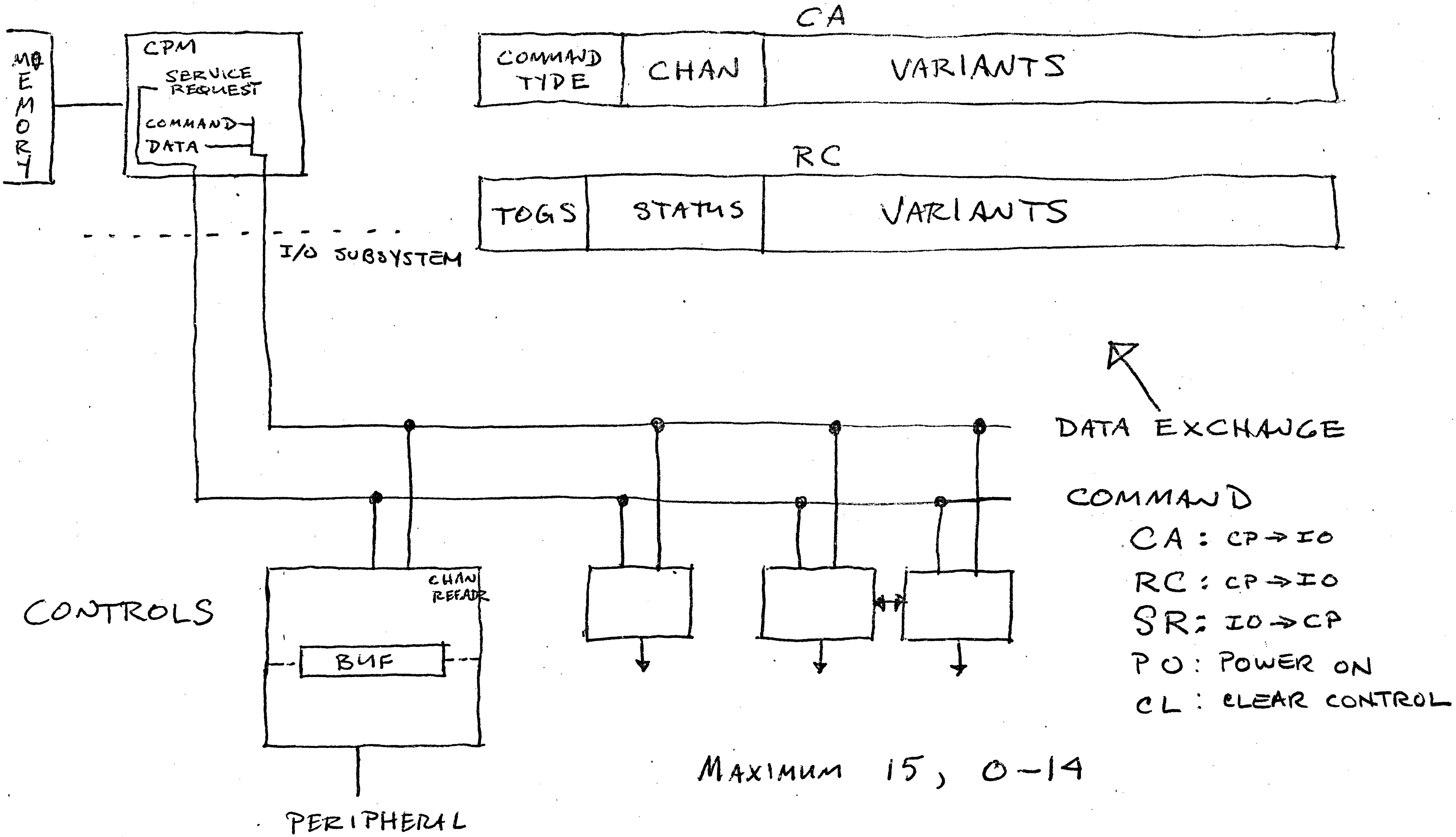
```
EXECUTE DESC
    TRANSFER OUT OP
    TRANSFER OUT FA (FILE ADDRESS)
    CALL IO.DATA
    TRANSFER OUT REF.ADR
    GO TO NEXT.SERVICE
```

```
NEXT SERVICE                                                    G-R
    % ARRIVE HERE IF INTERP DETECTS SERVICE REQUEST
    GET SERVICE REQUEST MASK
    IF NONE THEN EXIT
    SELECT LEFTMOST CHANNEL

    TRANSFER IN REF.ADR
    CALL IO.DATA
    GET CONTROL.STATUS
    IF STATUS EQL 7 THEN
    BEGIN
        TRANSFER OUT REF.ADR
        GO TO NEXT SERVICE
    END
    TRANSFER IN RESULT DESC
```

```
ANALYZE RESULT                                                  H-R
    IF EXCEPTION OR WANT.INTERRUPT THEN
    BEGIN
        ENTER INTERRUPT IN Q
        IF EXCEPTION THEN
        BEGIN
            SET EX.IDLE IN CH TABLE
            RESET BUSY
        END
    END
    IF DONT.LINK THEN
    BEGIN
        RESET BUSY
        GO NEXT.SERVICE
    END
    IF EXCEPTION THEN
        GO NEXT.SERVICE
    GO LINK TO NEXT DESC
```

```
IO.DATA
   % CALLED AT INITIATE AND SERVICE REQ

   GET CONTROL STATUS
   IF STATUS EQL 14 THEN
   BEGIN
       SET OUTPUT
   END ELSE
   BEGIN
       IF STATUS NEQ 15 THEN
           EXIT
       SET INPUT
   END
   CALCULATE REMAINING SPACE IN MEMORY BUFF
   CASE ON DEVICE TYPE
       SET UP TOGGLES AND INITIAL STATE.
           TERMINATE
           MULTI-TERMINATE
           BYTE SIZE
   IF NO SPACE LEFT THEN GO EMPTY
INPUT/OUTPUT
    GET DATA FROM CONTROL / READ DATA FROM MEMORY
    WRITE TO MEMORY / SEND TO CONTROL
    IF CONTROL IS DONE THEN
        GO CHECK TERMINATE
    IF BUFFER NOT FULL THEN GO TO INPUT/OUTPUT
    IF TERMINATE THEN GO TO TERMINATE
    GO TO EMPTY / BLANK.REST
TERMINATE
    SEND TERMINATE.DATA TO CONTROL
    GO TO WRAP.UP
EMPTY/BLANK.REST
     GET DATA FROM CONTROL / SEND DUMMY DATA
                                 TO CONTROL
     IF CONTROL NOT DONE THEN
         GO TO EMPTY
WRAP.UP
     UPDATE ENDING ADR IN MEMORY
     EXIT
```

```
SET.UP.TRANSFER OUT
    % 24 BITS IN T REG TO CONTROL
    BUILD TRANSFER OUT COMMAND USING CHANNEL
        SAVED
    OR IN FIRST 8 BITS FROM T
    MOVE TO COMMAND % CA
    MOVE DATA TO L % RC
    OR IN SECOND 8 BITS FROM T
    MOVE TO COMMAND
    MOVE DATA TO L
    OR IN THIRD 8 BITS FROM T
    MOVE TO COMMAND
    MOVE DATA TO L
    EXIT
SET.UP.TRANSFER.IN
    % ACCUMULATE 24 BITS FROM CONTROL IN T
    BUILD TRANSFER IN COMMAND USING SAVED CHANNEL
    MOVE TO COMMAND % CA
    MOVE DATA TO L % RC
    EXTRACT 8 BITS FROM RIGHT END, ACCUMULATE
    MOVE TO COMMAND
    MOVE DATA TO L
    EXTRACT AND ACCUMULATE
    MOVE TO COMMAND
    MOVE DATA TO L
    EXTRACT AND ACCUMULATE
    EXIT
GET.SERVICE.MASK
    BUILD TRANSFER IN SERVICE REQUEST
    MOVE TO COMMAND % CA
    MOVE DATA TO T % RC
    EXIT
GET.STATUS % TEST
    BUILD TEST.STATUS COMMAND, USING SAVED CHAN
    MOVE TO COMMAND % CA
    MOVE DATA TO T % RC
    EXIT
```



I/O SUBSYSTEM

COMMAND ACTIVATE – RESPONSE COMPLETE FORMATS

TYPES	CA TYPE	CA CHAN	CA VARIANT	RC TOG (0 1 2 3)	RC STATUS (3 – 7)	RC VARIANT (8 – 23)
TEST STATUS	0001	CHAN	0 ——— 01	—	STATUS	DEVICE ID | X
CLEAR & TEST STATUS	0001	CHAN	0 ——— 011	—	STATUS	DEVICE ID | X
TEST SERVICE REQUEST	0001	~~CHAN~~	0 ——— 0101	—	~~STATUS~~	SERVICE REQ MASK 15 ← 0
TERMINATE DATA	0001	CHAN	0 ——— 0110	—	STATUS	———
TRANSFER OUT A	0010	CHAN	1 OR 2 DATA BYTES	—	STATUS	———
TRANSFER IN	0100	CHAN	———	—	STATUS	1 OR 2 DATA BYTES
				24 DATA BITS		
TRANSFER OUT B	0011	CHAN	———	24 DATA BITS		

TOGS AT RC TIME

TAPE: BIT 0 = ODD BYTE. BIT 1 = DRIVE THRU GAP. BIT 2 = REVERSE

SORTER: BIT 0 = NONE BIT 1 = POCK SELECT, BIT 2 = REVERSE

OTHER: BIT 2 = REVERSE

ADDITIONAL SPECIAL TYPES

TRANSFER IN BYTE COUNT (MTC-4, PACK, 5N DISK)

TRANSFER OUT BYTE COUNT (" " ", PAPER TAPE)

ODD CHAR COUNT (MTC-2)

DRIVE THRU GAP (MTC-2)

STATUS VALUES: REFLECT CONTROL STATE PRIOR TO THIS CA-RC TRANSACTION.

COUNTS	MEANS
0	CONTROL NOT PRESENT
1	CLEARED (INITIAL) STATE
1,2,3	READY TO RECEIVE OP CODE BYTES 1,2,3
4,5,6	READY TO RECEIVE FILE ADDRESS BYTES 1,2,3
7,8,9	READY TO RECEIVE REFERENCE-ADR BYTES 1,2,3
10	BUSY (DOING OPERATION). USUALLY GOES TO 11 OR 18 AND RAISES SERVICE REQUEST
11,12,13	READY TO SEND REFERENCE ADDRESS, BYTES 1,2,3. USUALLY IMPLIES DATA TO FOLLOW
14	READY TO RECEIVE DATA (OUTPUT)
15	READY TO SEND DATA (INPUT)
16	END OF BUFER (READY TO SEND OR RECEIVE LAST BYTE); MORE TO COME.
17	END OF BUFFER; LAST BUFFER
18,19,20	READY TO SEND REFERENCE ADDRESS, BYTES 1,2,3. IMPLIES RESULT DESC TO FOLLOW.
21,22,23	READY TO SEND RESULT DESC BYTES 1,2,3

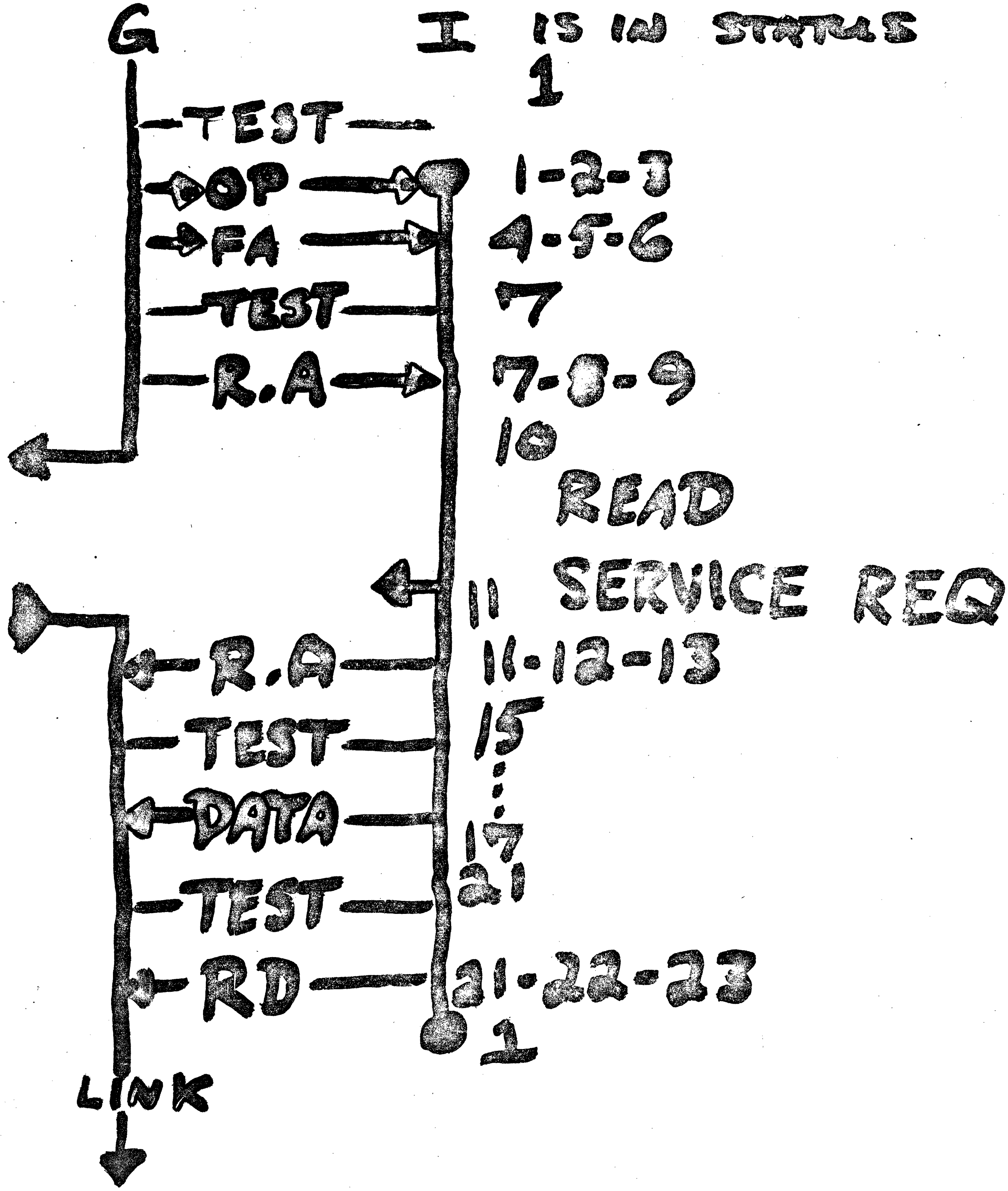
CONTROLS GO FROM STATUS 23 BACK TO 1.

CONTROLS DO NOT SEQUENCE STRAIGHT THRU COUNTS. RATHER STATUS COUNT IS USED TO TELL ~~[illegible]~~ SOFT-IO WHAT IS REQUIRED NEXT.

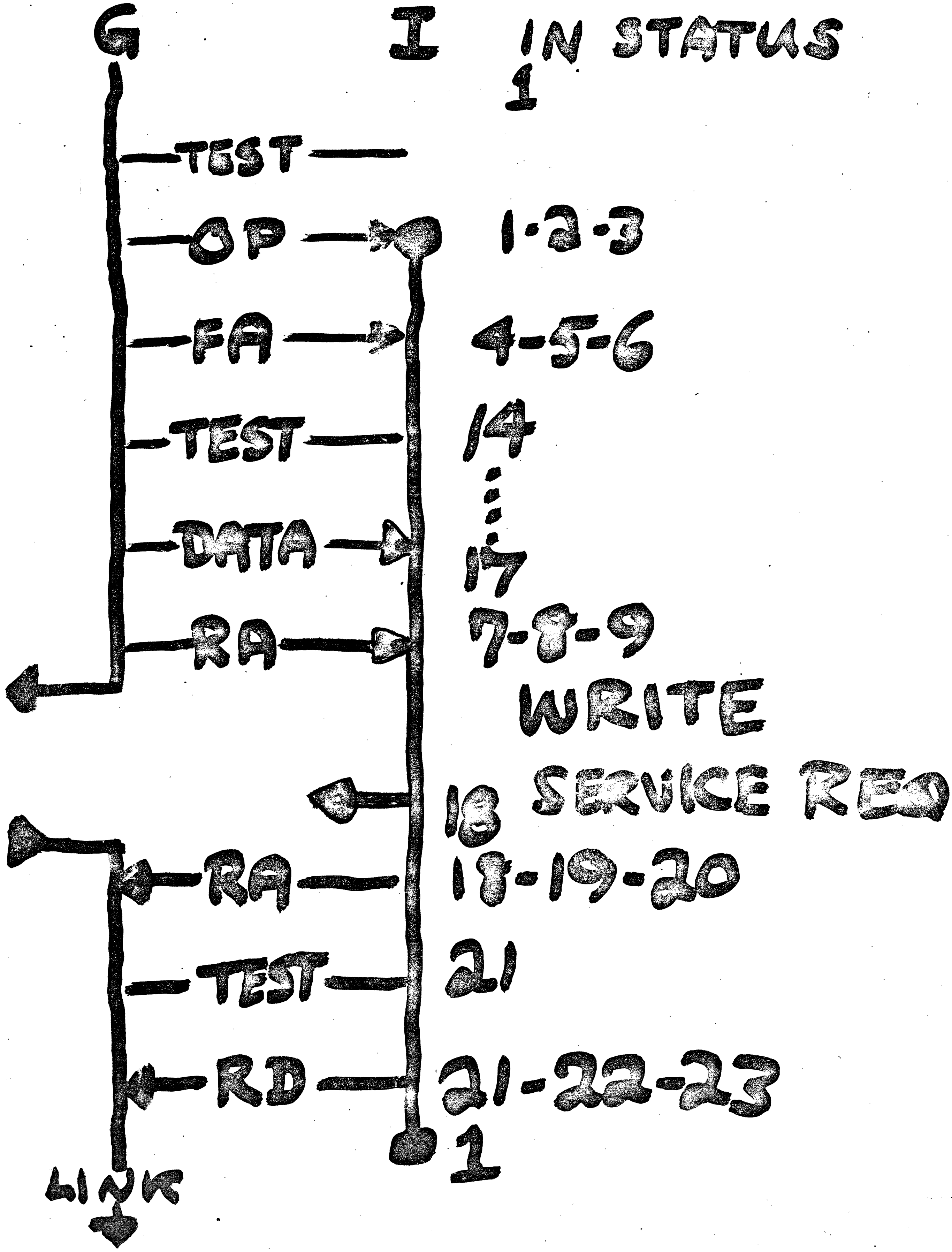
LEGAL TRANSITIONS: THOSE COUNTS GROUPED TOGETHER ABOVE (IE, 1,2,3) INDICATE ~~THAT~~ A REQUIRED SEQUENCE (IE, 2 MUST FOLLOW 1, 3 MUST FOLLOW 2).
OTHERS ARE

COUNT	MAY GO NEXT TO	COUNT	MAY GO NEXT TO
0	1	14	16,17 IF TERM 7,21
3	4	15	16,17 " " 7,21
6	7, 14	16	7
9	10	17	7, 21
10	11, 18	20	21
13	14,15,16,17	23	1

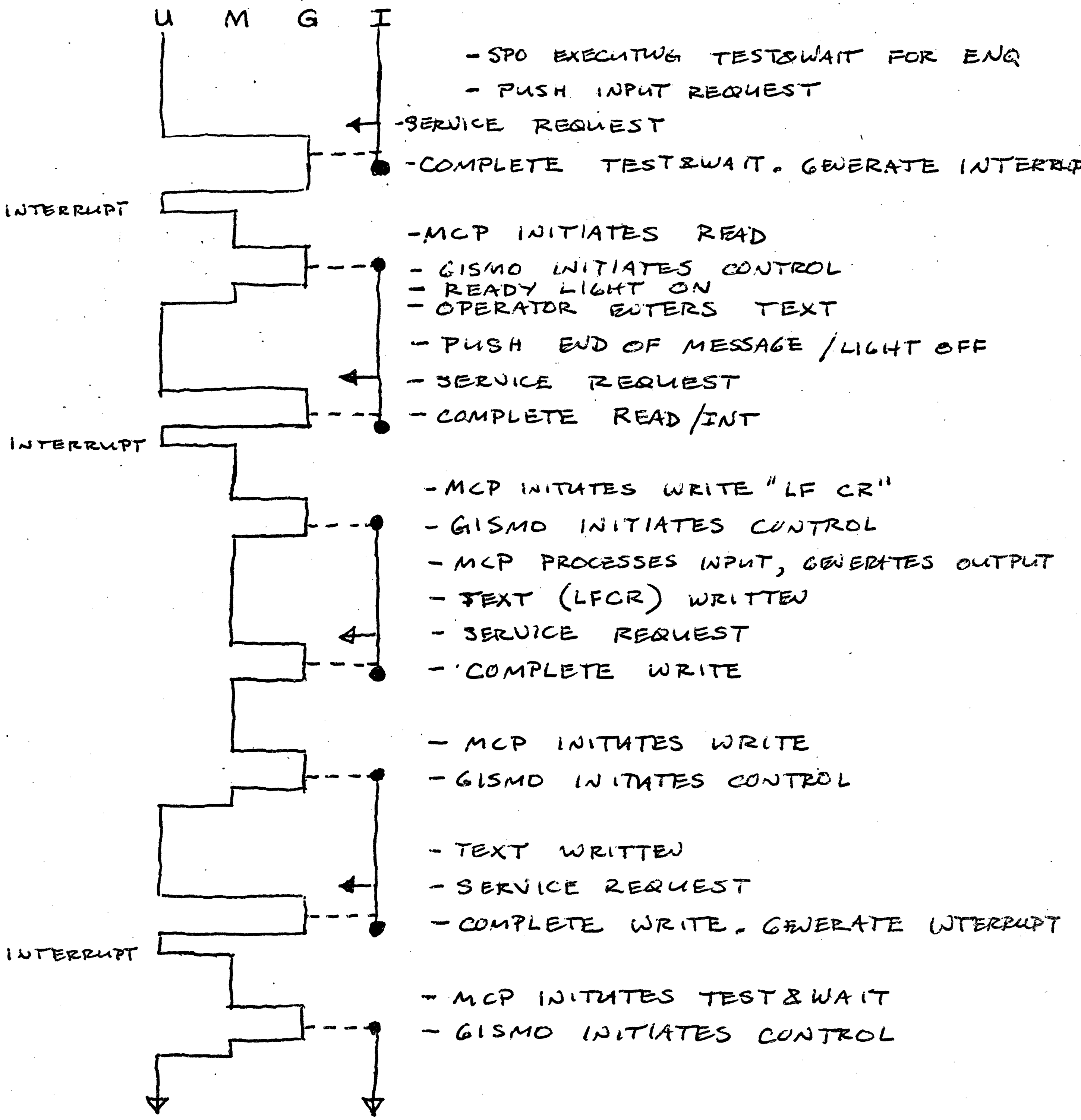
GISMO-CONTROL : READ



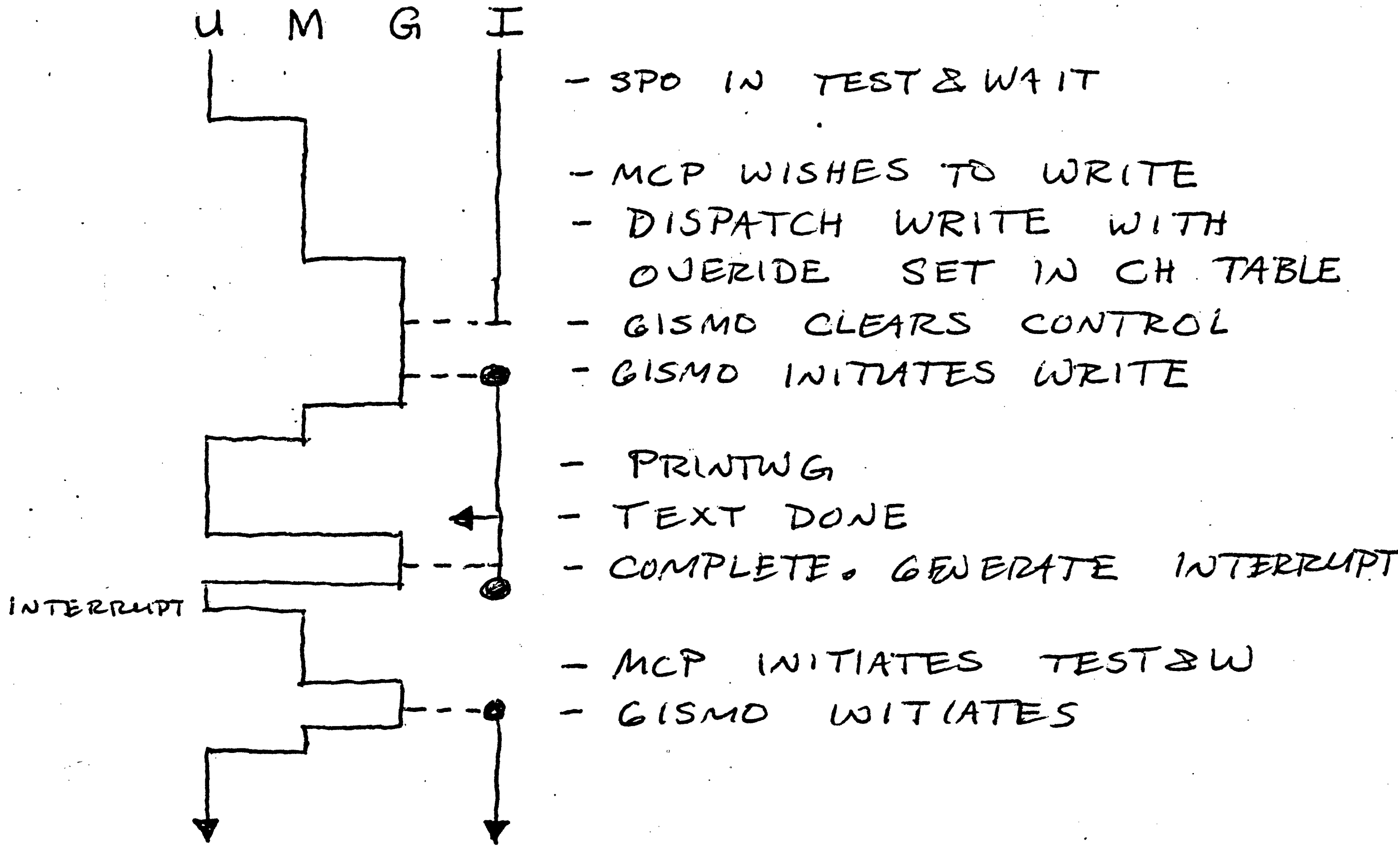
GISMO-CONTROL : WRITE



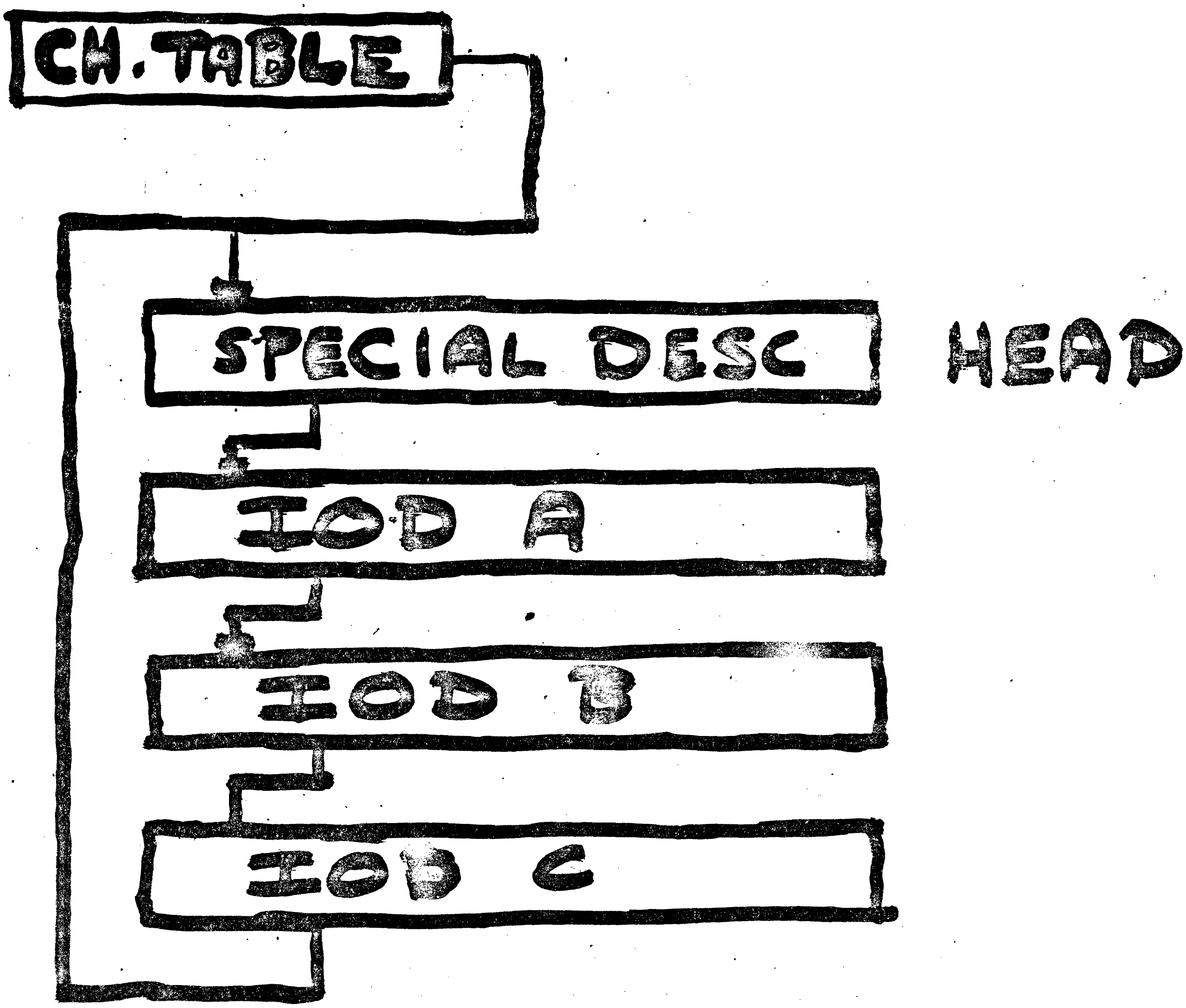
SPO: TYPICAL CASE OF INPUT



SPO: TYPICAL OUTPUT (UNSOLICITED)



DISK QUEUE



DISK: MCP - GISMO.

DISTINGUISHING CHARACTERISTICS

1) KEEP LINKING AFTER EXCEPTION.
   BIT IN CHANNEL TABLE; NO EXCEPTION IDLE.
2) LINK PAST A LOCKED DESCRIPTOR.
   SAME BIT AS (1)
3) NEED TO RECOGNIZE AND STOP AT HEAD OF QUEUE
   SPECIAL DESCRIPTOR IN DISK CHAIN, POINTED TO BY ADDRESS FIELD OF CHANNEL TABLE ENTRY. IF KEEP LINKING IS SET THEN REF.ADR FROM MCP DISCARDED AND TABLE VERSION USED.
4) NEED TO PASS THROUGH HEAD AS DEFINED BY (3) TO COPE WITH OUT OF ORDER DISPATCH.
   PENDING BIT IN CHANNEL TABLE.
5) ALL DISK DESCRIPTORS FOR VARIOUS CHANNELS IN SAME QUEUE.
   CHANNEL FIELD OF ~~R~~ RS FIELD MUST BE CORRECT.
6) ANY ORDER

NEW RULES FOR GISMO

```
DISPATCH
        SET PENDING BIT
        IF KEEP.LINK THEN USE REF.ADR FROM TABLE
ANALYZE OP CODE
        IF CHANNEL FROM RS NOT CHANNEL WE ARE ON
            LINK TO NEXT DESCRIPTOR
        IF SPECIAL DESCRIPTOR (HEAD OF Q) THEN
            IF PENDING BIT THEN
                RESET PENDING
                LINK TO NEXT DESC.
            NEXT SERVICE
ANALYZE RESULT
        IF EXCEPTION THEN
            IF NOT KEEP LINKING THEN
                SET EXCEPTION.IDLE
GET DESC
        IF ~~LOCKED OR~~ NOT READY THEN
            LINK TO NEXT DESC.
```

DISK: GISMO - CONTROL

```
NEW RULES FOR GISMO
    ANALZE OP CODE
        IF SPECIAL.DESCRIPTOR THEN
            IF PAUSE BIT IN CH TABLE THEN
               RESET PAUSE BIT
               CAUSE EXECUTION OF PAUSE OP
    NEXT.SERVICE
        GET STATUS
        IF STATUS EQL 1 THEN
            SET BUSY, PENDING IN CH TABLE
            GET REF.ADR FROM TABLE
            GO TO RESET CHANNEL
        TRANSFER IN RD
        IF RS BITS NOT 01 THEN
            TRANSFER IN RD AND DISCARD
            LINK TO NEXT DESCRIPTOR
    ANALYZE RESULT
        IF SECOND.OP.OFF THEN
            IF DC-1 THEN
                SET PAUSE BIT IN CH TABLE
            UNLOCK RS BITS
            LINK TO NEXT DESCRIPTOR
```

```
NEW RULE FOR TEST & WAITS
    AT TIMER INTERRUPT TIME INDEX THROUGH
      CHANNEL TABLE LOOKING FOR ~~CH~~ ENTRY
      WITH BUSY OFF AND TIMER ON.
      IF FOUND THEN DISPATCH, USING
         REF.ADR FROM TABLE
```

NOTE; DEVICES WHICH HAVE TIMER SET (DC-2, PACK, ALL TAPE) ARE A SUBSET OF DEVICES WITH KEEP.LINKING SET (ALL DISK, ALL TAPE).

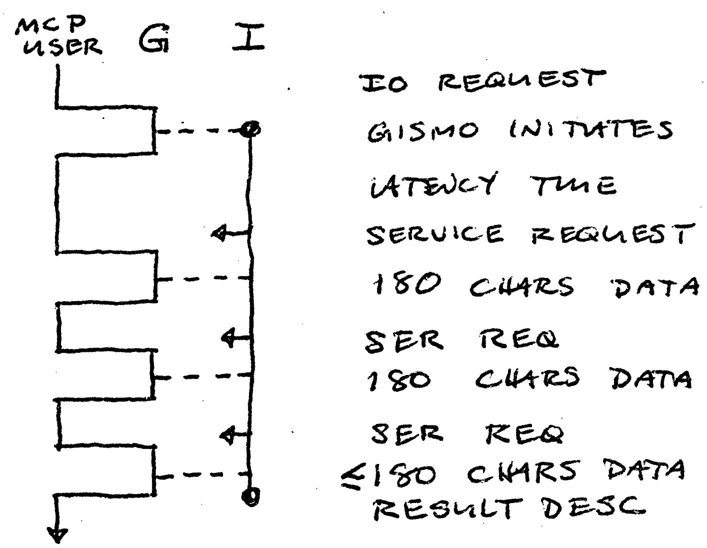
DISK: GISMO - CONTROL

CONTROLS	TRANSFER WIDTH	EXCH	ARM	NOTIFY SEEK COMP	T&W	PAUSE
HPT	16	YES	N	—	—	—
DC 1	16	N	YES	N	N	YES
DC 2	16	N	YES	YES	YES	N
PACK	24	N	YES	YES	YES	N
5N HPT	24	N	N	—	—	—

ALL HAVE SOME NUMBER OF 180 CHARACTER BUFFERS (LONGER FOR DIAGNOSTIC STUFF).
ALL ARE MULTIPLE SERVICE REQUEST TYPES.
ALL ALLOW (REQUIRE) ~~USE~~ GISMO TO TERMINATE.

DATA WIDTH AND TERMINATE SEQUENCE HANDLED BY IO.DATA ROUTINE. "24" TYPES ARE OPTIONAL GISMO SEGMENT.

HPT, INCLUDING 5N: TYPICAL READ



SLIPPAGE RESULTS IF ALL BUFFERS ARE FULL.

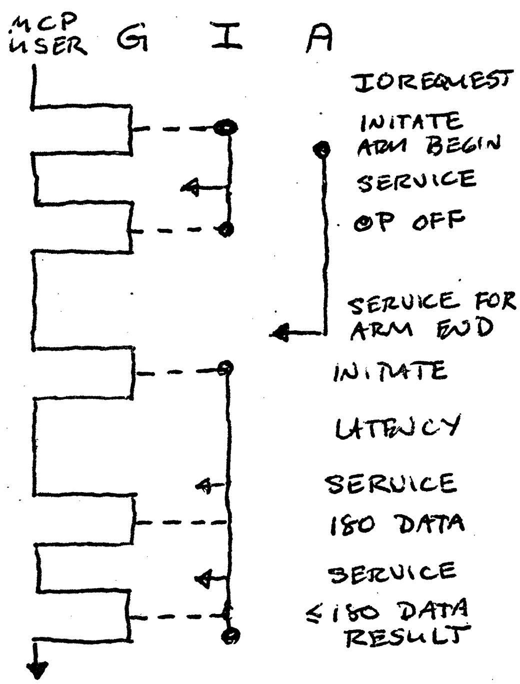
DISK: GISMO - CONTROL

ARM MOVERS.

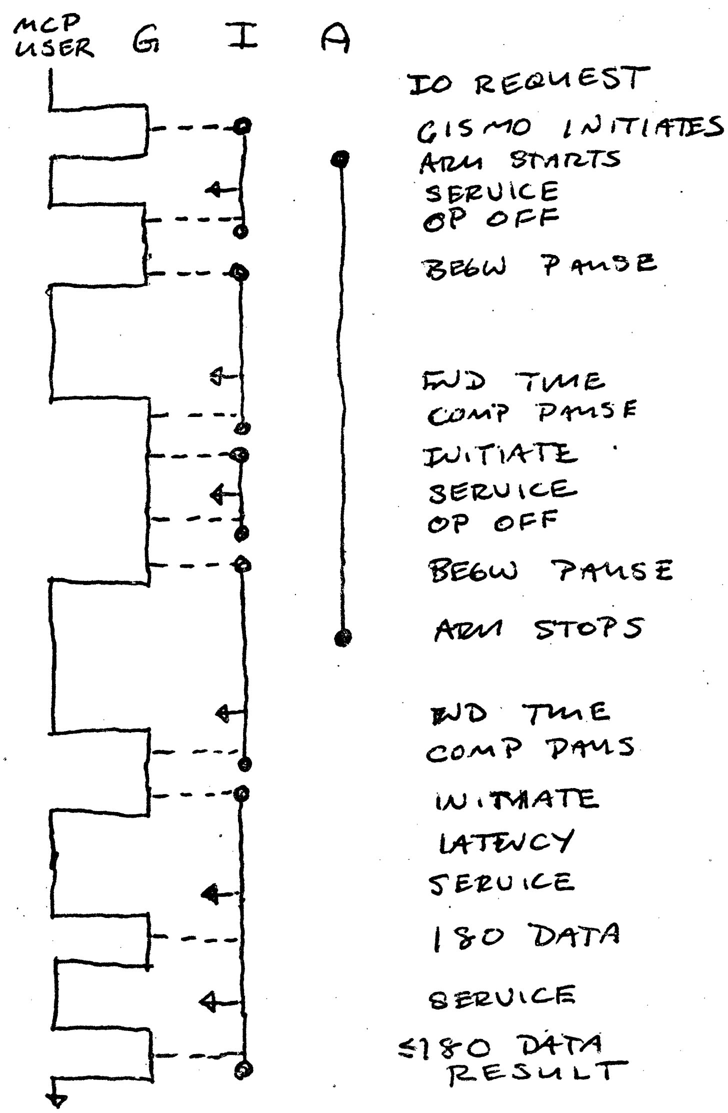
ALL USE SUPRESSION OF ~~B~~ SECOND OP COMPLETE BIT
TO INDICATE SEEKING OR WRONG CYLINDER
SEEKING: ARM IN MOTION
WRONG CYLINDER: WE JUST SOUGHT, BUT THIS
IO IS NOT FOR THAT TRACK (CYLINDER)

ALL ARE INITIATED BY STANDARD WAY
DC-2, PACK NOTIFY GISMO OF SEEK COMPLETE
DC-1 DOES NOT NOTIFY

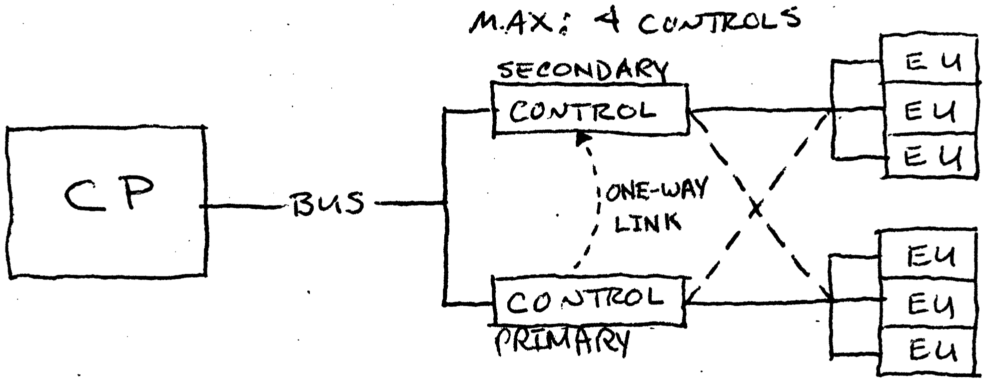
DC-2, PACK: READ



DC-1: READ



EXCHANGE: DISK



--- INDICATES PATH ADDED BY EXCHANGE

ADDS TO CHANNEL TABLE:

EXCH BIT: THE EXCH LINK FIELD POINTS TO THE NEXT CHANNEL ON THE EXCHANGE. THE LAST ENTRY FOR THE EXCHANGE HAS THIS BIT RESET

EXCH LINK: CONTAINS THE PORT/CHANNEL OF THE NEXT CHANNEL ON EXCH. CURRENTLY PORT IS ALWAYS 7.

NEW RULES FOR GISMO

DISPATCH
- IF BUSY THEN
  - IF ~~ON~~ EXCH THEN
    - ITERATE THRU DISPATCH WITH CHANNEL FROM LINK FIELD
  - EXIT

ANALYZE OP CODE
- IF CHANNEL WE ARE ON IS NOT CHANNEL ~~ON~~ IN RS LINK THROUGH EXCHANGE, IF ANY, TO SEE IF ITS SECONDARY CHANNEL. IF IT IS NOT THEN LINK TO NEXT DESCRIPTOR.

ANLYZE RESULT
- IF SECOND OP OFF THEN
  - IF EXCH THEN
    - FIND CH TABLE ENTRY FOR BUSY CONTROL AND SET ITS PENDING BIT.

```
DISPATCH TIME: HERE FROM S-OP (OR E.IO)                    A - Z
    GET CHANNEL TABLE ENTRY
    IF NOT INITULIZED THEN BYPASS RULES
    IF DISPATCH OVERRIDE THEN
        RESET BUSY, PENDING, EX.IDLE., OVERRIDE
    IF EXCEPTION.IDLE THEN
 EXCH.LINK
        IF EXCH THEN
            GET PORT/CHANNE FROM EXCH-LINK
            GO TO DISPATCH TIME
        EXIT
    SET BUSY, PENDING
    IF IT WAS ALREADY BUSY GO TO EXCH.LINK
    IF KEEP LINKING THEN
        GET REF.ADR FROM TABLE
```

```
RESET CHANNEL                                              B - Z
    IF PORT NEQ 7 THEN "MISSING DEVICE" ERROR EXIT
    IF CHANNEL EQL 15 THEN "MISSING DEVICE" ERROR EXIT
    SET UP CHANNEL, REF.ADR IN SCRATCHPADS
    GET CONTROL STATUS & ID  %( CA -RC)
    IF WHOLE RC IS ZERO THEN "MISSING DEVICE"
    IF POCKET SELECT TOG THEN GO TO HANDLE.POCKET
    IF STATUS NEQ 1 THEN CLEAR CONTROL (CA-RC)
```

```
ANALYZE ID                                                 C - Z
    SET UP LOCAL TOGGLES BY DECODING DEVICE ID.
    THEY WILL BE SAVED DURING EXECUTION IN
    RS FIELD OF DESCRIPTOR,
```

```
GET A DESCRIPTOR
    FETCH RS BITS
    IF NOT READY (00) THEN
        IF DISK TOG THEN
LINK:       LINK TO NEXT DESCRIPTOR
        IF TAPE TOG THEN
            IF NOT LOCKED.STATE THEN (LOOKING AT LOCK)
                GO TO LINK
            IF RS BITS 10 OR 11 THEN (A LOCKED DESC)
                EXIT SUBCHAN USING C.FIELD
                SET B.ADR OF LOCK TO LOCKED.DESC
                UNLOCK LOCK DESC AND RESET LOCKED.STATE
                USING REF ADR OF LOCK GO TO LINK
            IF RS BITS 01 THEN (LINKED BACK TO LOCK)
                SET B.ADR OF LOCK.OP TO A.ADR (HEAD SUBCHAN)
                UNLOCK LOCK DESC AND RESET LOCKED.STATE
                USING REF ADR OF LOCK GO TO LINK
GO TO  { IF PAUSE BIT IN CH TABLE (DATA COMM)
CHECK  {     FAKE UP PAUSE OP
FOR    {     GO TO EXECUTE DESC (TO DO PAUSE)
PAUSE
        (A NOT READY DESC, NOT TAPE OR DISK OR DATA COMM)
        GO TO NEXT.SERVICE

    (THE DESCRIPTOR WAS READY (00))

    LOCK RS BITS (01)
```

```
ANALYZE OP CODE
    IF STOP.OP.CODE THEN
        FAKE UP RD
        SET DONT.LINK
        GO TO ANALYZE RESULT
    IF PAUSE OP CODE THEN
        UNLOCK RS BITS
        GO TO CHECK FOR PAUSE
    IF DISK.TOG THEN
        IF CHANNEL WE ARE ON IS NOT CHANNEL IN RS THEN
            IF CHANNEL WE ARE ON IS NOT IN EXCHANGE THEN
                UNLOCK RS BITS
                GO TO LINK
    IF TAPE.TOG THEN
        IF OP CODE IS LOCK OP THEN
            SET LOCKED STATE
            USING B.ADR, LINK TO NEXT I/O
    IF DATA.COMM TOG THEN
        IF OP IS WRITE AUTO POLL THEN
            GO TO SKIP.E.ADR
    SET E.ADR TO A.ADR
SKIP.E.ADR
    GO TO EXECUTE DESC
```

```
                                                    (NON DISK, TAPE)
CHECK FOR PAUSE: WE HAVE ENCOUNTERED NOT READY DESC OR PAUS
    IF ~~SF~~ PAUSE BIT (DATA COMM) THEN
        FAKE UP PAUSE OP AND GO TO EXECUTE.DESC
    IF TO.PAUSE THEN (DC-1) THEN
        RESET TO.PAUSE
        FAKE UP PAUSE OP AND GO TO EXECUTE.DESC
    IF PENDING THEN
        RESET PENDING
        IF KEEP LINKING THEN (DISK, TAPE)
            LINK TO NEXT DESCRIPTOR
    RESET BUSY
    GO TO NEXT SERVICE
```

EXECUTE DESCRIPTOR F - Z

```
    TRANSFER OUT OP CODE      (CA-RC (3))
    TRNSFER OUT FILE ADR (C.ADR)  (CA-RC (3))
    IF REVERSE.TOG THEN  (IN RC OF LAST CA-RC)
        SET E.ADR TO B.ADR  (END OF BUFFER)
    CALL IO.DATA   (MAY OR MAY NOT MOVE DATA)
    (CONTROL ALWAYS IN STATUS 7)
    TRANSFER OUT REF.ADR
    GO TO NEXT.SERVICE
```

G - Z

```
NEXT SERVICE: HERE FROM INTERP OR "ITERATE" GISMO
    GET SERVICE REQUEST MASK
    IF NONE THEN RESET SERVICE REQ THEN EXIT
    SELECT HIGHEST CHANNEL
    GET STATUS
    IF STATUS EQL 1 THEN (DC-2, PACK: SEEK COMPLETE)
        SET BUSY, PENDING
        USING REF.ADR FROM TABLE, GOTO RESET CHANNEL
    (CONTROL IN STATUS 11 OR 18)
    TRANSFER IN REF.ADR  (CA-RC(3))
    IF RS BITS NEQ 01 THEN  (END OF PAUSE)
        TRANSFER IN RD AND DISCARD  (CA-RC(3))
        IF DISK.TOG THEN  (WAS A HEAD OF Q)
            LINK TO NEXT I/O (GO TO RESET CHANNEL)
        USING THIS REF.ADR, GO TO RESET CHANNEL  (NOT READY)
    CALL IO.DATA  (MAY OR MAY NOT MOVE DATA)
    GET STATUS  (7, 10, 21)
    IF STATUS EQL 7 THEN
        TRANSFER OUT REF ADR
        GO TO NEXT.SERVICE
    IF STATUS NEQ 21 THEN
        GO TO NEXT.SERVICE
    TRANSFER IN RESULT DESC  (CA-RC(3))
```

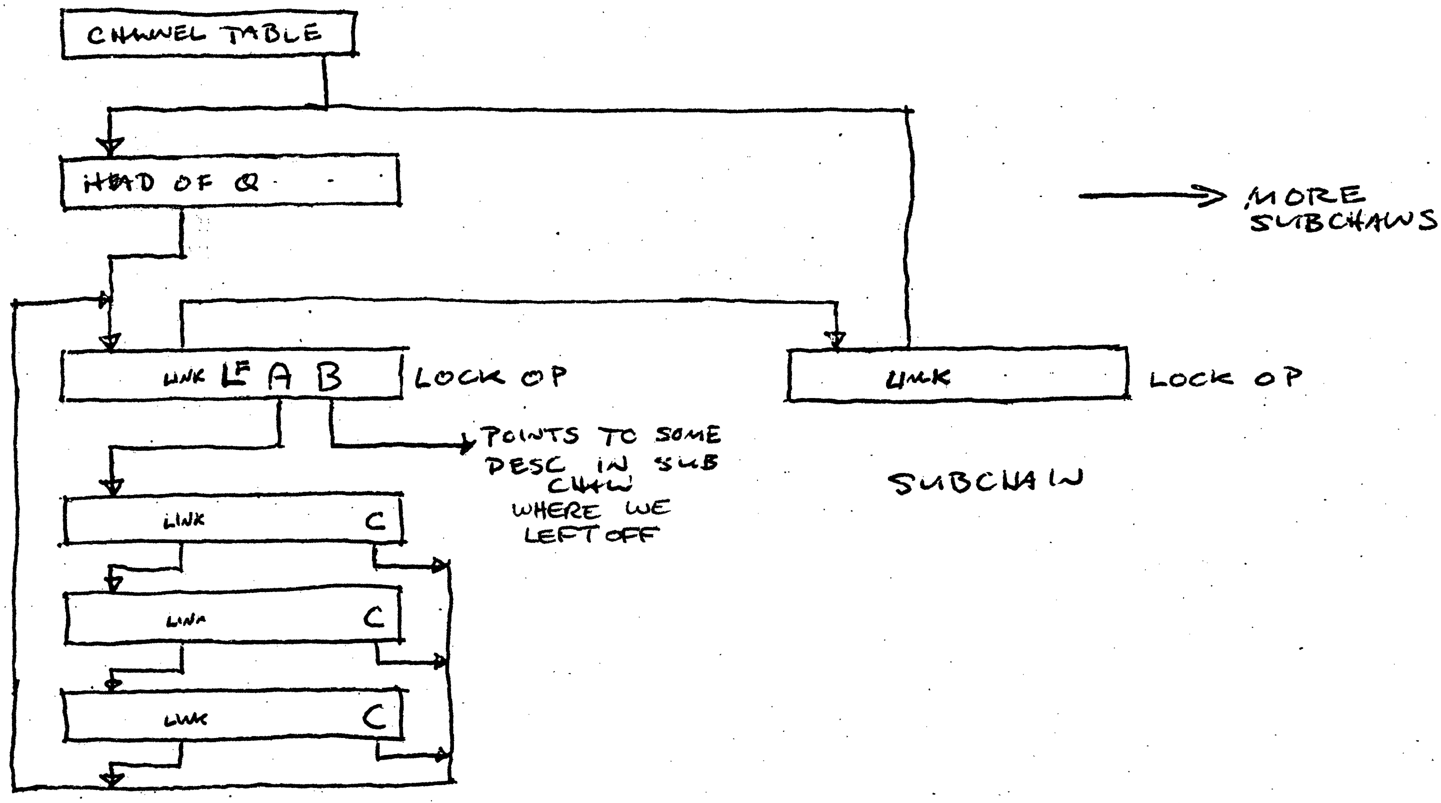
ANALYZE RESULT

```
IF SECOND.OP.OFF THEN CALL SECOND.OP.COMP.OFF
IF DATA.COMM.TOG THEN
   (LONG RECORD WAS DETECTED IN IO.DATA AND TOG
    SET IN RS)
   MAYBE SET "NO ETX" AND EXCEPTION
STORE RESULT IN RS FIELD OF IO DESC
IF EXCEPTION OR WANT.INTERRUPT (RS) THEN
   FORMAT AND ENTER INTERRUPT IN QUE
   IF EXCEPTION AND NOT KEEP.LINKING THEN
      SET EXCEPTION.IDLE
IF DONT.LINK THEN
   RESET BUSY, PENDING
   GO TO NEXT SERVICE
IF TAPE.TOG AND EXCEPTION THEN
   EXIT SUBCHAN USING C.ADR
   SET B.ADR OF LOCK.OP TO EXCEPTION DESC
   UNLOCK RS OF LOCK AND RESET LOCKED STATE
   USING LINK FROM LOCK OP, LINK TO NEXT I/O
IF NOT DISK.TOG AND EXCEPTION THEN
   GO TO NEXT.SERVICE
(NO EXCEPTION OR DISK AND EXCEPTION)
LINK TO NEXT DESC
```

SECOND.OP.COMPLETE.OFF

```
IF DISK.TOG THEN
   IF SPECIAL.RD THEN SET SECOND.OP.COMP THEN EXIT
   IF TEST&WAIT THEN GO TO NEXT.ONE
   IF DC-1 THEN SET TO.PAUSE
   IF EXCH THEN
      FWD BUSY CONTROL (CH TABLE ENTRY) AND SET ITS PENDING
NEXT.ONE REMOVE RETURN ADR
   UNLOCK RS
   USING LINK, GO TO RESET CHANNEL (LINK TO NEXT)
IF TAPE.TOG THEN
   UNLOCK RS FIELD
   EXIT SUBCHAN USING C.ADR
   SET B.ADR OF LOCK DESC TO POINT TO THIS DESC
   UNLOCK LOCK DESC AND RESET LOCKED.STATE
   LINK TO NEXT DESC (FROM LOCK OP).
```

TAPE CHAIN



ALLOWS SUBCHAINS AS WHOLES TO BE HANDLED IN ANY ORDER.

ALLOWS LINKING WITHIN SUBCHAIN IN SERIAL ORDER.

SHARES SOME OF STUFF IMPLEMENTED FOR DISK CH TABLE: PENDING, EXCH, KEEP LINKING, TIMER, EXCH-LINK, ADR

NEW OP CODE: LOCK. INDICATES HEAD OF SUBCHAIN

ADDITIONAL GISMO TOGGLE IN RS FIELD: LOCKED STATE

EACH SUBCHAIN IS A UNIT.

TAPE: GISMO - CONTROL

CONTROL WILL:

SET A TOGGLE INDICATING REVERSE IN EVERY STATUS REPORT AFTER RECEIVING OP CODE.

SET A TOGGLE AND ACCEPT A SPECIAL COMMAND FOR HANDLING SINGLE BYTE OF DATA IN OR OUT.

SUPRESS THE SECOND OP COMPLETE BIT IF CONDITIONS ARE NOT MET ON A TEST & WAIT.

NEW GISMO RULES

GET, DESC

IF NOT READY THEN (% RS BITS)

IF NOT LOCKED, STATE THEN

LINK TO NEXT DESCRIPTOR

IF RS IS 10 OR 11 THEN

EXIT SUBCHAIN USING C.ADR

SET B, ADR IN LOCK OP (HEAD SUBCHAIN)

TO POINT TO NOT READY DESC

UNLOCK LOCK OP AND RESET LOCKED STATE

LINK TO NEXT DESC (NEXT LOCK OP)

IF RS IS 01 THEN (AT LOCK OP)

SET B, ADR TO A, ADR OF LOCK OP

UNLOCK LOCK OP AND RESET LOCKED STATE

LINK TO NEXT DESCRIPTOR

ANLYZE OP

IF OP EQL LOCK OP THEN

SET LOCKED STATE

USE B, ADR TO LINK TO NEXT DESC

EXECUTE DESC

IF REVERSE TOG (IN STATUS REPORT RC) THEN

SET E, ADR TO B, ADR

ANALYZE RESULT

    IF SECOND OP OFF THEN
        UNLOCK RS
        EXIT SUBCHAIN USING C,ADR
        SET B,ADR TO REF,ADR OF THAT DESC
        UNLOCK LOCK OP RS AND RESET LOCKED STATE
        LINK TO NEXT DESC.
    IF EXCEPTION THEN
        ~~UNLK~~ EXIT SUBCHAIN USING C,ADR
        SET B,ADR TO REF,ADR OF THAT DESC
        UNLOCK LOCK RS AND RESET LOCKED STATE
        LINK TO NEXT DESC

IO,DATA

    COPE WITH ODD BYTE TRANSFERS
        AND REVERSE OPERATIONS.
    COPE WITH SPACE OP CODE,