'8008'
MONITOR Routines

SCelbi Computer Consulting, Inc.
1322 Rear - Boston Post Road
Milford, Ct. 06460
'8008' MONITOR ROUTINES

AUTHOR: ROBERT FINDLEY

© COPYRIGHT 1975
SCELBI COMPUTER CONSULTING, INC.
1322 REAR - BOSTON POST ROAD
MILFORD, CT. 06460
- ALL RIGHTS RESERVED -

IMPORTANT NOTICE

OTHER THAN USING THE PROGRAM DETAILED HEREIN ON THE PURCHASER'S INDIVIDUAL COMPUTER SYSTEM, NO PART OF THIS PUBLICATION MAY BE REPRODUCED, TRANSMITTED, STORED IN A RETRIEVAL SYSTEM, OR OTHERWISE DUPLICATED IN ANY FORM OR BY ANY MEANS ELECTRONIC, MECHANICAL, PHOTOCOPYING, RECORDING, OR OTHERWISE, WITHOUT THE PRIOR EXPRESS WRITTEN CONSENT OF THE COPYRIGHT OWNER.

THE INFORMATION IN THIS MANUAL HAS BEEN CAREFULLY REVIEWED AND IS BELIEVED TO BE ENTIRELY RELIABLE. HOWEVER, NO RESPONSIBILITY IS ASSUMED FOR INACCURACIES OR FOR 'THE SUCCESS OR FAILURE OF VARIOUS APPLICATIONS' TO WHICH THE INFORMATION CONTAINED HEREIN MIGHT BE APPLIED.
ERRATA FOR THE
'
8008' MONITOR ROUTINES

OOPS!! IN GENERATING THE LISTING USED FOR THIS MANUAL, AN INSTRUCTION WAS LEFT OUT OF THE "INSPCL" SUBROUTINE, ON PAGES 16 AND 49, WHICH SETS THE INPUT BUFFER POINTER TO PAGE 000. THIS ERROR MAY BE CORRECTED WITHOUT RE-ASSEMBLING THE LISTING IN THE BACK OF THE BOOK BY SIMPLY HAVING THE OPERATOR INPUT ROUTINE, DESCRIBED ON PAGE 5, SET THE VALUE OF REGISTER H TO ZERO BEFORE RETURNING TO THE CALLING PROGRAM. THIS WILL NOT AFFECT THE OPERATION OF THE OTHER ROUTINE (CDIN) WHICH ALSO CALLS THE OPERATOR INPUT ROUTINE, SINCE THEY BOTH INPUT CHARACTERS TO THE INPUT BUFFER WHICH IS LOCATED ON PAGE 000.

IF THE PROGRAM IS TO BE RE-ASSEMBLED, TO BE ORIGINED ON A DIFFERENT PAGE OR TO MAKE REVISIONS TO THE PROGRAM, THE "INSPCL" SUBROUTINE SHOULD INCLUDE AN "LHI 000" INSTRUCTION AS THE SECOND INSTRUCTION OF THE SUBROUTINE. THE REVISED LISTING SHOULD APPEAR AS follows:

INSPCL, LLI 340
LHI 000
LPIN, CAL RCV
...
INTRODUCTION

THE MONITOR PROGRAM IS A PROGRAM WHICH ENABLES THE COMPUTER OPERATOR TO UTILIZE A COMPUTER SYSTEM WITH GREATER EFFICIENCY AND EFFECTIVENESS, BY TAKING ADVANTAGE OF THE INHERENT POWER OF THE COMPUTER. BASICALLY, THE MONITOR PROGRAM ALLOWS THE OPERATOR TO CONTROL THE COMPUTER BY DIRECTING IT TO EXECUTE PROGRAMS STORED IN MEMORY, OPERATE PERIPHERAL DEVICES FOR STORING AND RETRIEVING PROGRAMS AND DATA, AND EXAMINE AND/OR MODIFY MEMORY LOCATIONS, EITHER ONE AT A TIME OR IN BLOCKS. THE PROGRAMMER WILL FIND ITS ABILITY TO INTERRUPT A PROGRAM BEING DEBUGGED AT VARIOUS POINTS AND EXAMINE THE CONTENTS OF MEMORY LOCATIONS AND "CPU REGISTERS AND STATUS FLAGS" AT THAT POINT IN THE PROGRAM IS A FUNCTION THAT IS AS POWERFUL A DEBUGGING TOOL AS A GOOD OSCILLOSCOPE IS FOR THE HARDWARE TROUBLESHOOTER.

THERE ARE SEVERAL FACTORS WHICH DETERMINE THE ABILITY TO OPERATE A COMPUTER SYSTEM 'EFFECTIVELY.' ONE OF THESE FACTORS IS TO BE ABLE TO CONTROL ITS OPERATION FROM A SINGLE LOCATION. THE MOST COMMON METHOD IS TO CONTROL THE COMPUTER FROM ITS 'FRONT PANEL'. THIS IS NORMALLY A MYRIAD OF SWITCHES AND LAMPS WHICH ENABLE THE OPERATOR TO LOAD AND EXAMINE MEMORY LOCATIONS, EXECUTE PROGRAMS STORED IN MEMORY AND, IN SOME OF THE MORE SOPHISTICATED FRONT PANELS, PERFORM SEVERAL PROGRAM DEBUGGING FUNCTIONS. USING THE FRONT PANEL TO OPERATE THE COMPUTER IS AN EXCELLENT WAY TO INTRODUCE THE BEGINNER TO THE BASICS OF THE COMPUTER'S OPERATION, BECAUSE IT GIVES HIM FIRST-HAND EXPERIENCE IN THE CONCEPTS OF LOADING MEMORY WITH A PROGRAM, STEPPING THROUGH THE PROGRAM AND SEEING HOW THE COMPUTER PROGRESSES FROM ONE INSTRUCTION TO ANOTHER. THAT'S FINE FOR THE BEGINNER! BUT ONCE THE 'THRILL' OF WATCHING THE COMPUTER STEP THROUGH ONE OR TWO PROGRAMS IS GONE (ESPECIALLY SINCE THEY HAD TO BE LOADED SEVERAL TIMES TO GET THEM IN CORRECTLY), EVEN THE BEGINNER FINDS OPERATING THROUGH THE FRONT PANEL SLOW, CUMBERSOME AND OFTEN ANNOYING.

AN ALTERNATIVE METHOD IS TO HAVE THE COMPUTER AID IN THESE BASIC FUNCTIONS BY PROGRAMMING IT TO UTILIZE A MORE CONVENIENT 'CONTROL' DEVICE, NAMELY A KEYBOARD AND DISPLAY DEVICE. THE KEYBOARD ENTRY IS BY FAR A FASTER AND MORE ACCURATE MEANS OF ENTERING MEMORY ADDRESSES AND DATA THAN THAT OF TOGGING THEM IN THROUGH THE FRONT PANEL SWITCHES AND DISPLAYING THE INFORMATION AS OCTAL DIGITS ON AN ALPHANUMERIC DISPLAY. WHETHER IT BE A TTY PRINTER OR VIDEO DISPLAY, IS MUCH EASIER TO READ THAN DECODING THE BINARY PRESENTATION OF MEMORY ADDRESS AND CONTENTS ON THE FRONT PANEL INDICATORS. MAKING USE OF THESE DEVICES IMPROVES THE SYSTEM FROM THE 'HUMAN ENGINEERING' STANDPOINT, SINCE THEY GIVE THE OPERATOR A FORM OF COMMUNICATION WITH THE COMPUTER THAT IS MORE CONVENTIONAL THAN FLIPPING SWITCHES AND WATCHING LIGHTS. THIS BRINGS UP THE SECOND FACTOR IN OPERATING AN EFFECTIVE COMPUTER SYSTEM: THAT FACTOR IS USING A COMPUTER PROGRAM TO PERFORM AS MANY OF THE TASKS AS POSSIBLE WHICH THE COMPUTER IS CAPABLE OF PERFORMING FASTER AND MORE ACCURATELY THAN THE OPERATOR COULD EVER DREAM OF PERFORMING.

SINCE THE PROGRAM WILL BE OCCUPIED SPACE IN MEMORY, IT IS NECESSARY TO EVALUATE THE TYPE OF FUNCTIONS IT IS TO PERFORM AND CHOOSE THE ONES WHICH WILL BE OF GREATEST IMPORTANCE TO THE OPERATOR. FIRST, THE FUNCTIONS OF THE FRONT PANEL SHOULD BE REPLACED. ONE OF THESE FUNCTIONS IS THE EXAMINATION AND MODIFICATION OF MEMORY CONTENTS, FOR LOADING AND REVISING PROGRAMS AND DATA IN MEMORY. AN EXPANSION OF THIS WILL ALSO BE PROGRAMMED, THAT OF DISPLAYING A LARGE BLOCK OF MEMORY AT ONE TIME. THIS IS QUITE VALUABLE FOR CHECKING THAT A PROGRAM HAS BEEN LOADED CORRECTLY AND, IN DEBUGGING, TO EXAMINE LARGE DATA STORAGE AREAS.
THE NEXT FUNCTION THAT WOULD GENERALLY FOLLOW WOULD BE TO DIRECT
THE OPERATION OF A STORAGE DEVICE TO STORE AND RETRIEVE THE CONTENTS OF
A BLOCK OF MEMORY FOR SAVING PROGRAMS OR DATA. THIS WILL SAVE A LOT OF
TIME IN THAT A LARGE PROGRAM WOULD NOT HAVE TO BE ENTERED THROUGH THE
KEYBOARD EVERY TIME IT IS DESIRED TO USE IT. INSTEAD, IT CAN BE READ
FROM THE BULK STORAGE DEVICE DIRECTLY INTO MEMORY TAKING ADVANTAGE OF
ITS SPEED AND ACCURACY, AS OPPOSED TO KEYBOARD ENTRY. THIS PORTION OF
THE PROGRAM WILL HAVE TO BE CUSTOMIZED TO THE USER'S SPECIFIC STORAGE
DEVICE, AS WILL BE DESCRIBED LATER.

NOW THAT THE ABILITY TO ENTER, MODIFY AND STORE A PROGRAM HAS BEEN
ESTABLISHED, THE NEXT LOGICAL PROGRESSION WOULD BE TO ENABLE THE OPERA-
TOR TO START EXECUTION OF A PROGRAM FROM THE KEYBOARD. AT THIS POINT, A
REQUIREMENT FOR DEBUGGING PROGRAMS MUST BE CONSIDERED.

IN THE PROCESS OF DEBUGGING A PROGRAM, IT MAY BE DESIRED TO SET THE
INITIAL VALUES OF SPECIFIC CPU REGISTERS BEFORE JUMPING TO THE START OF
A ROUTINE BEING WORKED ON. THIS CAN BE ACCOMPLISHED BY USING A SEPARATE
FUNCTION TO SET UP THE VALUES TO BE PLACED IN THE CPU REGISTERS AT THE
TIME THE PROGRAM IS ENTERED, VIA THE 'GO TO' FUNCTION.

AS A COMPLIMENTARY FUNCTION OF GO TO, THE MONITOR SHOULD BE ABLE TO
SET A 'BREAKPOINT.' A BREAKPOINT IS A POINT IN A PROGRAM AT WHICH THE
PROGRAMMER DESIRES TO STOP EXECUTION AND CHECK THE PROGRESS OF THE PRO-
GRAMS OPERATION. THE BREAKPOINT FUNCTION REPLACES THE INSTRUCTION AT
THE POINT IN QUESTION WITH A JUMP TO THE BREAKPOINT ROUTINE. WHEN THE
BREAKPOINT IS REACHED, THE COMPUTER RETURNS CONTROL TO THE MONITOR WHERE
THE BREAKPOINT ROUTINE WILL SAVE THE CONTENTS OF THE CPU REGISTERS AND
THE STATUS FLAGS IN A TABLE IN MEMORY WHICH THE PROGRAMMER MAY REFER TO
IN CHECKING THE OPERATION OF THE PROGRAM.

THESE FUNCTIONS ARE A GOOD BASE FOR SETTING UP A MONITOR PROGRAM,
SINCE THEY PROVIDE THE OPERATOR WITH AN ASSORTMENT OF FUNCTIONS WHICH
ARE COMMON TO THE OPERATION OF ANY COMPUTER SYSTEM. FROM THIS BASE, THE
MONITOR CAN BE EXPANDED TO INCLUDE OPERATIONS OF SPECIFIC APPLICATION TO
ONE'S OWN SET UP. SEVERAL POSSIBILITIES ARE PRESENTED AS PART OF THIS
MONITOR PROGRAM. THESE FUNCTIONS INCLUDE FILLING A BLOCK OF MEMORY WITH
A SPECIFIC DATA VALUE, SEARCHING MEMORY FOR A DATA PATTERN AND SHIFTING
BLOCKS OF DATA FROM ONE SECTION OF MEMORY TO ANOTHER.

THE PURPOSE OF THE MANUAL IS TO PRESENT THE READER WITH A MONITOR
PROGRAM WHICH CAN BE USED AS IS, OR MODIFIED OR EXPANDED TO CREATE A
REAL "OPERATING SYSTEM" FOR ONE'S OWN COMPUTER SYSTEM. THE MONITOR PRO-
GRAM CAN BE AN INVALUABLE ASSET TO ANY COMPUTER SYSTEM. ITS ABILITY TO
PERFORM MANY OF THE REQUIRED 'CONVENIENCE' FUNCTIONS NEEDED TO CONTROL A
COMPUTER SYSTEM ALONG WITH THE POWER IT AFFORDS THE PROGRAMMER IN DEBUG-
GING PROGRAMS MAKES IT A 'MUST' FOR THE SERIOUS COMPUTER OWNER.
THE BASIC FUNCTIONS AND CAPABILITIES OF A "MONITOR" PROGRAM

GENERALLY, A MONITOR PROGRAM CONSISTS OF A VARIETY OF COMMANDS WHICH ENABLE THE COMPUTER OPERATOR TO CONTROL THE OPERATION OF THE COMPUTER AND ITS RELATED PERIPHERAL DEVICES. THIS IS ACHIEVED BY ENTERING COMMANDS ON A KEYBOARD DEVICE WHICH DIRECT THE COMPUTER TO DISPLAY AND/OR MODIFY THE CONTENTS OF MEMORY LOCATIONS, PERFORM DATA STORAGE AND RETRIEVAL ON AVAILABLE 'BULK' STORAGE PERIPHERALS AND EXECUTE OTHER PROGRAMS WHICH ARE STORED IN THE COMPUTER'S MEMORY. THE MEMORY ADDRESS, OR ADDRESSES, AFFECTED BY THE COMMAND IS GENERALLY SPECIFIED IN THE COMMAND INPUT. THE NUMBER OF DIFFERENT COMMANDS ONE SETS UP IN A MONITOR PROGRAM WILL DEPEND ON THE AMOUNT OF MEMORY DESIRED TO DEDICATE TO THE MONITOR PROGRAM, SINCE IT MUST RESIDE IN MEMORY, AND ON THE NUMBER OF PERIPHERALS IT IS DESIRED TO CONTROL WITH THE MONITOR.

THE SPECIFIC I/O (INPUT/OUTPUT) DEVICES USED TO OPERATE THE MONITOR PROGRAM WILL NATURALLY VARY FROM ONE SYSTEM TO ANOTHER. FOR THIS REASON THE I/O PORTION OF THE MONITOR IS SET UP TO CALL 'USER-PROVIDED' I/O DRIVER ROUTINES TO PERFORM THE ACTUAL INPUTTING AND OUTPUTTING OF COMMANDS AND DATA IN RESPONSE TO THE COMMANDS. THE REQUIREMENTS OF THE I/O DRIVERS WILL BE DESCRIBED IN THE NEXT SECTION. THIS APPROACH ENABLES THE READER TO "CUSTOMIZE" THE MONITOR PROGRAM TO THE SPECIFIC DEVICES AVAILABLE ON ONE'S COMPUTER SYSTEM WITHOUT CHANGING THE INSTRUCTIONS OF THE MONITOR PROGRAM PRESENTED HEREIN.

THE MONITOR PROGRAM PRESENTED IN THIS MANUAL IS CAPABLE OF PERFORMING THE FUNCTIONS MENTIONED WHILE OPERATING IN AN '8008' BASED MINICOMPUTER SYSTEM WITH AT LEAST 1.5K BYTES OF MEMORY. IF A SHORTER VERSION IS DESIRED, THE FUNCTIONS DEEMED LESS VALUABLE TO THE USER CAN BE DELETED. EACH FUNCTION AND ITS ASSOCIATED ROUTINE(S) IS EXPLAINED IN DETAIL TO ENABLE THE READER TO UNDERSTAND THE OPERATION OF THE PROGRAM. MANY OF THE ROUTINES DESCRIBED MAY BE APPLICABLE TO OTHER TYPES OF FUNCTIONS WHICH ONE MAY DESIRE TO INCLUDE IN ONE'S MONITOR PROGRAM. OR, THEY MAY BE UTILIZED IN DEVELOPING OTHER PROGRAMS. AS EACH ROUTINE IS PRESENTED A DETAILED, HIGHLY COMMENTED LISTING IS PROVIDED. A COMPLETE ASSEMBLED LISTING OF THE MONITOR PROGRAM IS THEN PRESENTED TO WHICH THE READER MAY ADD THE CUSTOM I/O DRIVER ROUTINES AND IMPLEMENT THE MONITOR PROGRAM ON AN '8008' BASED SYSTEM. (READERS THAT DESIRE TO IMPLEMENT THIS PROGRAM ON OTHER TYPES OF SYSTEMS SHOULD FIND THE INFORMATION CONTAINED IN THIS MANUAL OF CONSIDERABLE VALUE. FOR EXAMPLE, IMPLEMENTING SUCH A PROGRAM ON AN '8080' BASED SYSTEM WOULD REQUIRE THE MERE TRANSLATION OF THE SOURCE LISTING TO THE EQUIVALENT '8080' INSTRUCTIONS.)

I/O (INPUT/OUTPUT) CONSIDERATIONS FOR THE MONITOR PROGRAM

BEFORE DISCUSSING THE ACTUAL ROUTINES WHICH MAKE UP THE MONITOR PROGRAM, IT IS NECESSARY TO MENTION SEVERAL POINTS ABOUT THE CHARACTER SET USED AND DESCRIBE THE REQUIREMENTS FOR THE I/O PROGRAMMING.

THE CHARACTER CODE USED BY THE MONITOR PROGRAM FOR ENTERING COMMANDS AND OUTPUTTING CHARACTERS TO THE DISPLAY DEVICE IS ASSUMED TO BE "ASCII" ENCODED CHARACTERS. THE "ASCII" CHARACTER SET CONSIST OF A 7-BIT CODE WHICH IS CAPABLE OF DEFINING UP TO 128 "CHARACTERS." THE MONITOR PROGRAM DESCRIBED HEREIN UTILIZES A SUBSET OF THIS CODE CONSISTING OF 31 DIFFERENT CHARACTERS - 15 "UPPER CASE" LETTERS OF THE ALPHABET,
The numerals 0 - 9, and several symbols and punctuation marks, often, when communicating with an ASCII encoded I/O device, an 8th bit is added to the seven bit ASCII code. This 8th bit is often referred to as the "parity" bit because it can be used to serve as an error detecting bit. Many I/O devices are designed to operate with eight bits of information, regardless of whether or not "parity" error checking methods are being utilized. The monitor program described herein assumes that the "parity" position is always in a logic one state. The "ASCII" character codes used by the monitor are presented below along with the codes for other "ASCII" characters generally provided by "ASCII" encoded devices. For I/O devices which do not operate with the "ASCII" character set, the problem of code conversion is easily taken care of by programming the I/O driver to make the necessary conversion between the ASCII code defined here to the code utilized by the device.

<table>
<thead>
<tr>
<th>CHARACTERS SYMBOLIZED</th>
<th>BINARY CODE</th>
<th>OCTAL REP</th>
<th>CHARACTERS SYMBOLIZED</th>
<th>BINARY CODE</th>
<th>OCTAL REP</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11 000 001</td>
<td>301</td>
<td>!</td>
<td>10 100 001</td>
<td>241</td>
</tr>
<tr>
<td>B</td>
<td>11 000 010</td>
<td>302</td>
<td>&quot;</td>
<td>10 100 010</td>
<td>242</td>
</tr>
<tr>
<td>C</td>
<td>11 000 011</td>
<td>303</td>
<td>#</td>
<td>10 100 011</td>
<td>243</td>
</tr>
<tr>
<td>D</td>
<td>11 000 100</td>
<td>304</td>
<td>$</td>
<td>10 100 100</td>
<td>244</td>
</tr>
<tr>
<td>E</td>
<td>11 000 101</td>
<td>305</td>
<td>%</td>
<td>10 100 101</td>
<td>245</td>
</tr>
<tr>
<td>F</td>
<td>11 000 110</td>
<td>306</td>
<td>&amp;</td>
<td>10 100 110</td>
<td>246</td>
</tr>
<tr>
<td>G</td>
<td>11 000 111</td>
<td>307</td>
<td>'</td>
<td>10 100 111</td>
<td>247</td>
</tr>
<tr>
<td>H</td>
<td>11 001 000</td>
<td>310</td>
<td>(</td>
<td>10 101 000</td>
<td>250</td>
</tr>
<tr>
<td>I</td>
<td>11 001 001</td>
<td>311</td>
<td>)</td>
<td>10 101 001</td>
<td>251</td>
</tr>
<tr>
<td>J</td>
<td>11 001 010</td>
<td>312</td>
<td>*</td>
<td>10 101 010</td>
<td>252</td>
</tr>
<tr>
<td>K</td>
<td>11 001 011</td>
<td>313</td>
<td>+</td>
<td>10 101 011</td>
<td>253</td>
</tr>
<tr>
<td>L</td>
<td>11 001 100</td>
<td>314</td>
<td>-</td>
<td>10 101 100</td>
<td>254</td>
</tr>
<tr>
<td>M</td>
<td>11 001 101</td>
<td>315</td>
<td>.</td>
<td>10 101 101</td>
<td>255</td>
</tr>
<tr>
<td>N</td>
<td>11 001 110</td>
<td>316</td>
<td>/</td>
<td>10 101 110</td>
<td>256</td>
</tr>
<tr>
<td>O</td>
<td>11 001 111</td>
<td>317</td>
<td>0</td>
<td>10 101 111</td>
<td>257</td>
</tr>
<tr>
<td>P</td>
<td>11 010 000</td>
<td>320</td>
<td>1</td>
<td>10 110 000</td>
<td>260</td>
</tr>
<tr>
<td>Q</td>
<td>11 010 001</td>
<td>321</td>
<td>2</td>
<td>10 110 001</td>
<td>261</td>
</tr>
<tr>
<td>R</td>
<td>11 010 010</td>
<td>322</td>
<td>3</td>
<td>10 110 010</td>
<td>262</td>
</tr>
<tr>
<td>S</td>
<td>11 010 011</td>
<td>323</td>
<td>4</td>
<td>10 110 011</td>
<td>263</td>
</tr>
<tr>
<td>T</td>
<td>11 010 100</td>
<td>324</td>
<td>5</td>
<td>10 110 100</td>
<td>264</td>
</tr>
<tr>
<td>U</td>
<td>11 010 101</td>
<td>325</td>
<td>6</td>
<td>10 110 101</td>
<td>265</td>
</tr>
<tr>
<td>V</td>
<td>11 010 110</td>
<td>326</td>
<td>7</td>
<td>10 110 110</td>
<td>266</td>
</tr>
<tr>
<td>W</td>
<td>11 010 111</td>
<td>327</td>
<td>8</td>
<td>10 110 111</td>
<td>267</td>
</tr>
<tr>
<td>X</td>
<td>11 011 000</td>
<td>330</td>
<td>9</td>
<td>10 111 000</td>
<td>270</td>
</tr>
<tr>
<td>Y</td>
<td>11 011 001</td>
<td>331</td>
<td>:</td>
<td>10 111 001</td>
<td>271</td>
</tr>
<tr>
<td>Z</td>
<td>11 011 010</td>
<td>332</td>
<td>;</td>
<td>10 111 010</td>
<td>272</td>
</tr>
<tr>
<td>[</td>
<td>11 011 011</td>
<td>333</td>
<td>&lt;</td>
<td>10 111 011</td>
<td>273</td>
</tr>
<tr>
<td>]</td>
<td>11 011 100</td>
<td>334</td>
<td>&gt;</td>
<td>10 111 100</td>
<td>274</td>
</tr>
<tr>
<td>^</td>
<td>11 011 101</td>
<td>335</td>
<td>?</td>
<td>10 111 101</td>
<td>275</td>
</tr>
<tr>
<td>_</td>
<td>11 011 110</td>
<td>336</td>
<td></td>
<td>10 111 110</td>
<td>276</td>
</tr>
<tr>
<td>\</td>
<td>11 011 111</td>
<td>337</td>
<td></td>
<td>10 111 111</td>
<td>277</td>
</tr>
<tr>
<td>SPACE</td>
<td>11 100 000</td>
<td>240</td>
<td>!</td>
<td>11 000 000</td>
<td>300</td>
</tr>
<tr>
<td>CTRL D</td>
<td>10 000 100</td>
<td>204</td>
<td>CTRL N</td>
<td>10 001 110</td>
<td>216</td>
</tr>
<tr>
<td>CTRL I</td>
<td>10 001 001</td>
<td>211</td>
<td>CTRL S</td>
<td>10 010 011</td>
<td>223</td>
</tr>
<tr>
<td>LINE FEED</td>
<td>10 001 010</td>
<td>212</td>
<td>CTRL T</td>
<td>10 010 100</td>
<td>224</td>
</tr>
<tr>
<td>CTRL L</td>
<td>10 001 100</td>
<td>214</td>
<td>CTRL U</td>
<td>10 010 101</td>
<td>225</td>
</tr>
<tr>
<td>CAR-RET</td>
<td>10 001 101</td>
<td>215</td>
<td>RUE OUT</td>
<td>11 111 111</td>
<td>377</td>
</tr>
</tbody>
</table>

74 CHARACTER ASCII SUBSET

THERE ARE FOUR SEPARATE I/O DRIVER ROUTINES REQUIRED BY THE MONITOR PROGRAM AS PRESENTED. THESE ROUTINES SHOULD BE PREPARED AS SUBROUTINES WHICH WILL BE CALLED BY THE OPERATING PROGRAM. TWO OF THE ROUTINES ARE USED TO COMMUNICATE BETWEEN COMPUTER AND OPERATOR FOR ENTERING COMMANDS AND DATA AND DISPLAYING THE COMMANDS AS ENTERED AND ALSO THE RESULTANT OUTPUT AS REQUESTED BY THE COMMAND. THE OTHER TWO ROUTINES WILL CONTROL THE STORAGE AND RETRIEVAL OF DATA ON THE SYSTEM 'BULK' STORAGE DEVICE. THE REQUIREMENTS FOR THESE I/O ROUTINES, AS FAR AS THIS MONITOR PROGRAM IS CONCERNED, ARE PRESENTED BELOW.

OPERATOR INPUT

THE OPERATOR INPUT ROUTINE WHEN CALLED MUST INPUT A SINGLE CHARACTER FROM A DEVICE, SUCH AS A KEYBOARD, AND RETURN TO THE OPERATING PROGRAM WITH THE ASCII CODE FOR THE INPUTTED CHARACTER IN THE ACCUMULATOR REGISTER OF THE CPU. THIS ROUTINE, CREATED BY THE USER, IS FREE TO USE CPU REGISTERS "A" THRU "E" FOR ITS PROCESSING. IF REGISTERS "H" AND "L" MUST BE USED (TO POINT TO A CONVERSION TABLE, FOR EXAMPLE) THEIR CONTENTS MUST BE SAVED AND THEN RESTORED TO THEIR ORIGINAL VALUE PRIOR TO RETURNING TO THE CALLING PROGRAM. THE OPERATOR INPUT ROUTINE IS REFERRED TO IN THE MONITOR PROGRAM BY THE LABEL "RCV." THERE ARE TWO POINTS IN THIS MONITOR PROGRAM WHERE "CAL RCV" IS USED TO SIGNIFY A CALL TO THE "OPERATOR INPUT" SUBROUTINE. ONE IS AT THE INSTRUCTION LABELED "IN2" IN THE "INPUT" ROUTINE (TO BE PRESENTED LATER). THE OTHER LOCATION WHICH CALLS THIS ROUTINE IS THE LOCATION LABELED "LPIN" IN THE "INSPCL" SUBROUTINE.

AN ADDITIONAL FUNCTION WHICH THE USER SHOULD PROVIDE IN THE "OPERATOR INPUT" SUBROUTINE IS THE CAPABILITY TO "ECHO" THE CHARACTER RECEIVED FROM THE INPUT DEVICE TO THE DISPLAY DEVICE. THAT IS, WHEN A CHARACTER IS ENTERED ON THE KEYBOARD IT IS GENERALLY DESIRED TO HAVE THAT CHARACTER DISPLAYED FOR THE OPERATOR TO VERIFY THE ENTRY. FOR EXAMPLE, IF THE OPERATOR INPUT IS COMING FROM AN ELECTRONIC KEYBOARD WHICH IS COMPLETELY SEPARATE FROM THE DISPLAY DEVICE, IT WOULD BE REQUIRED TO HAVE THE "RCV" ROUTINE OUTPUT THE CHARACTER CODE TO THE DISPLAY DEVICE AS EACH CHARACTER IS RECEIVED. OR, ONE MIGHT HAVE A SYSTEM IN WHICH THE INPUT DEVICE IS COORDINATED WITH THE DISPLAY DEVICES, SUCH AS A TELETYPewriter OR TELEVISION-TYPE-WRITER, WHICH MAY BE COUPLED WITH A HARDWARE INTERFACE TO AUTOMATICALLY ECHO THE KEYBOARD INPUT TO THE DISPLAY DEVICE. IN THIS CASE, THE "RCV" SUBROUTINE WOULD HAVE TO ENABLE THE INTERFACE TO ECHO THE CHARACTERS WHEN ENTERED.
DISPLAY OUTPUT

THE DISPLAY OUTPUT ROUTINE IS DISTINCT FROM THE "ECHO" ROUTINE DESCRIBED IN THE OPERATOR INPUT ROUTINE ABOVE (ALTHOUGH, IN MANY CASES, THE "ECHO" FUNCTION OF THE "RCV" SUBROUTINE MAY SIMPLY BE OBTAINED BY CALLING THIS DISPLAY OUTPUT ROUTINE AS IT IS DEFINED HERE!) THE DISPLAY OUTPUT ROUTINE WHEN CALLED BY THE MONITOR PROGRAM MUST OUTPUT THE ASCII ENCODED CHARACTER CONTAINED IN THE ACCUMULATOR AT THE TIME THE ROUTINE IS CALLED TO THE DISPLAY DEVICE. THE ROUTINE IS FREE TO USE CPU REGISTERS "B" THRU "E" FOR PROCESSING. THE CALLING ROUTINE EXPECTS THE ACCUMULATOR AND REGISTERS "H" AND "L" TO CONTAIN THE ORIGINAL INFORMATION WHEN THE SUBROUTINE IS EXITED. THE DISPLAY OUTPUT SUBROUTINE IS REFERENCED IN THE MONITOR PROGRAM BY A "CAL PRINT" INSTRUCTION. THERE ARE FIVE ROUTINES WHICH USE THE "CAL PRINT" COMMAND. THE "ERROR" ROUTINE USES THE "PRINT" SUBROUTINE TO OUTPUT ERROR MESSAGES TO THE OPERATOR. THE DISPLAY OUTPUT ROUTINE IS ALSO CALLED BY THE SUBROUTINES LABELED "MSG" (TO PRINT VARIOUS MESSAGES), "OCTOUT" (FOR PRINTING 3 DIGIT OCTAL NUMBERS), "COLON" (TO PRINT A :) AND "SPAC" (TO PRINT A SPACE).

BULK STORAGE INPUT

THE BULK STORAGE INPUT ROUTINE WHEN CALLED MUST INPUT DATA FROM THE BULK STORAGE DEVICE. THE FORMAT FOR READING THE DATA AND DETERMINING WHERE THE DATA IS TO BE STORED IS ENTIRELY LEFT UP TO THE USER PROVIDED BULK INPUT ROUTINE. THE ONLY FUNCTION OF THE MONITOR PROGRAM FOR THIS COMMAND IS TO ALLOW THE INITIATION OF A BULK INPUT VIA THE KEYBOARD AND TO RETURN TO THE MONITOR PROGRAM UPON COMPLETION OF THE INPUT SEQUENCE. THEREFORE, THE BULK STORAGE INPUT ROUTINE IS FREE TO USE ALL THE CPU REGISTERS WHILE PERFORMING ITS DATA INPUT. THE BULK STORAGE INPUT ROUTINE IS REFERENCED BY THE INSTRUCTION "CAL READ" WHICH IS LOCATED IN THE BULK READ ROUTINE OF THE MONITOR PROGRAM.

BULK STORAGE OUTPUT

THE BULK STORAGE OUTPUT ROUTINE WHEN CALLED MUST OUTPUT THE DATA INDICATED TO THE BULK STORAGE DEVICE. THE DATA TO BE STORED IS DELINEATED BY REGISTERS "L" AND "H" FOR THE LOW AND PAGE ADDRESS, RESPECTIVELY, FOR THE START ADDRESS AND REGISTERS "E" AND "D" FOR THE LOW AND PAGE ADDRESS, RESPECTIVELY, FOR THE ENDING ADDRESS OF THE BLOCK OF DATA TO BE OUTPUT. AS WITH THE BULK INPUT ROUTINE, THE ACTUAL FORMAT AND PROCEDURE FOR OUTPUTTING THE DATA IS ENTIRELY CONTROLLED BY THIS ROUTINE. THE MONITOR PROGRAM SIMPLY SETS UP THE REGISTERS DESIGNATING THE LIMITS OF THE BLOCK TO BE OUTPUT. THIS BULK STORAGE OUTPUT ROUTINE IS CALLED BY THE BULK WRITE ROUTINE BY THE INSTRUCTION "CAL PUNCH."

I/O INTEGRITY CONSIDERATIONS

THE OPTION OF PERFORMING ERROR CHECKS ON THE TRANSMISSION OF DATA TO AND FROM THE PERIPHERAL DEVICES IS LEFT TO THE USER. THIS IS DONE BECAUSE THERE ARE A VARIETY OF ERROR CHECKING TECHNIQUES POSSIBLE, DEPENDING ON THE TYPE OF DEVICE BEING USED IN THE SYSTEM. FOR EXAMPLE, A USER WITH A PAPER TAPE READER SYSTEM MAY ELECT TO PROVIDE FOR PARITY...
CHECKING TECHNIQUES. SUCH TECHNIQUES MAY BE IMPLEMENTED USING "EVEN" OR "ODD" PARITY CONVENTIONS DEPENDING ON THE TYPE OF DEVICE, OR EVEN THE USER'S PREFERENCE. ANOTHER TYPE OF I/O DEVICE, SUCH AS A COMMERCIAL MAGNETIC TAPE, OR DISC UNIT, MAY HAVE AUTOMATIC "BLOCK" ERROR CHECKING CAPABILITIES, IN WHICH CASE THE USER WOULD WANT TO HAVE THE APPROPRIATE I/O ROUTINE TEST FOR ERROR CONDITIONS AND TAKE APPROPRIATE ACTION. THE USER MAY ELECT, IF ERROR CHECKING CAPABILITIES ARE IMPLEMENTED, TO ADD ADDITIONAL ROUTINES THAT PRESENT ERROR MESSAGES TO THE OPERATOR, OR THAT DIRECT THE OPERATION OF "ERROR CORRECTING" TECHNIQUES. IN ANY EVENT, SUCH TECHNIQUES ARE OUTSIDE THE SCOPE IF THIS PARTICULAR PUBLICATION AND WILL BE LEFT TO THE USER TO IMPLEMENT AS DESIRED.

MEMORY UTILIZATION OF THE MONITOR PROGRAM

THE MONITOR PROGRAM PRESENTED IN THIS MANUAL MAKES OPTIMUM USE OF THE MEMORY BY UTILIZING EFFECTIVE PROGRAMMING TECHNIQUES WHICH TAKE ADVANTAGE OF THE '8008' INSTRUCTION SET. THE ACTUAL AMOUNT OF MEMORY USED BY THE MONITOR WILL VARY DEPENDING ON THE NUMBER OF COMMANDS ONE INCLUDES IN ONE'S VERSION AND ON THE AMOUNT OF PROGRAMMING REQUIRED TO CONTROL THE PERIPHERAL DEVICES. THE MEMORY USAGE FOR THE VERSION PRESENTED IN THIS MANUAL IS AS FOLLOWS.

THE OPERATING PORTION OF THE PROGRAM RESIDES IN PAGES 14 THROUGH PART OF PAGE 17. THE USER PROVIDED I/O ROUTINES MAY BE PLACED ON THE REMAINDER OF PAGE 17, OR, IF MORE ROOM IS REQUIRED, THE USER MAY PUT THE I/O ROUTINES WHEREVER THEY WILL BE MOST CONVENIENT (FOR EXAMPLE, THE BULK STORAGE I/O ROUTINES MAY ALREADY RESIDE IN MEMORY ON A "PROM"). PORTIONS OF PAGE 00 ARE USED AS A "SCRATCH PAD" AREA FOR THE STORAGE OF POINTERS, COUNTERS AND TEMPORARY DATA BY THE MONITOR PROGRAM. THERE IS ALSO A SECTION ON PAGE 00 WHICH CONTAINS "CANNED" MESSAGES AND THE LAST 40 OCTAL LOCATIONS ARE USED AS THE INPUT BUFFER FOR STORING THE COMMANDS AND DATA ENTERED ON THE KEYBOARD INPUT DEVICE. ONE OF THE RESTART LOCATIONS (LOCATION 070) IS USED BY THE BREAKPOINT ROUTINE TO ALLOW A SINGLE RESTART INSTRUCTION TO BE USED TO SET A BREAKPOINT IN A PROGRAM BEING DEBUGGED. THE LOOK-UP TABLE FOR THE COMMAND ROUTINE HAS BEEN SET UP ON PAGE 00 TO ALLOW ROOM FOR EXPANSION, AS WILL BE EXPLAINED LATER.

THE LOCATION OF THE OPERATING PORTION OF THE MONITOR PROGRAM FOR A SPECIFIC USER'S SYSTEM SHOULD BE IN THE UPPER PORTION OF THE AVAILABLE MEMORY. THIS ARRANGEMENT HAS BEEN FOUND TO BE MOST ADVANTAGEOUS FOR A MONITOR PROGRAM, AS IT LEAVES THE LOWER PORTION OF THE MEMORY OPEN TO BE USED FOR PROGRAM DEVELOPMENT. THE MEMORY MAP FOR THIS MONITOR PROGRAM AS ORIGINATED IN THIS MANUAL IS PRESENTED ON THE FOLLOWING PAGE. THE EXACT LOCATIONS USED FOR THE TEMPORARY STORAGE ON PAGE 00 WILL BE DETAILED IN THE ASSEMBLED LISTING.

MONITOR COMMANDS

THE MONITOR PROGRAM IS ESSENTIALLY A COLLECTION OF FUNCTIONS WHICH ENABLE THE OPERATOR OR PROGRAMMER TO CONTROL THE OVER-ALL OPERATION OF THE COMPUTER. THESE FUNCTIONS ARE INITIATED BY THE OPERATOR ENTERING "COMMANDS" ON THE "OPERATOR INPUT DEVICE." EACH COMMAND DIRECTS THE MONITOR PROGRAM TO THE APPROPRIATE ROUTINE TO PERFORM THE FUNCTION INDICATED. THE FORMAT FOR ENTERING EACH COMMAND MAY VARY FROM ONE TO AN-
OTHER, DEPENDING ON WHETHER THE COMMAND REQUIRES MEMORY ADDRESSES OR DATA TO BE SPECIFIED. THE FOLLOWING IS A SUMMARY OF THE VARIOUS COMMANDS PRESENTED IN THIS MONITOR PROGRAM ALONG WITH A BRIEF DESCRIPTION OF THE OPERATION EACH PERFORMS.


"MEMORY DUMP" (D) - OUTPUTS THE CONTENTS OF THE MEMORY LOCATIONS SPECIFIED TO THE DISPLAY DEVICE.
"MEMORY FILL" (F) - Fills the memory locations specified with the data indicated in the command.

"GO TO" (G) - Starts execution of a program by jumping to the address specified in the command. Two types of GO TO commands are possible. A type "1" GO TO command will set the contents of CPU registers A, B and C with pre-determined values before jumping to the program. A type "2" GO TO command will set up registers D, E, H and L.

"MEMORY MODIFY" (M) - Displays the contents of the memory location specified. The operator may then change the contents by entering the desired value, after which the next location will be displayed, or continue on to display the next location without changing the previous one, or return to the command mode.

"BULK READ" (R) - Calls the user provided bulk storage input routine to read data in from the bulk storage device.

"SEARCH" (S) - Searches the memory locations specified for the 8 bit data pattern entered in the command and prints the memory addresses of each location that matches.

"TRANSFER" (T) - Transfers the contents of the section of memory specified to the section of memory indicated by the third address specified in the command.

"BULK WRITE" (W) - Calls the user provided bulk storage output routine to write a specified block of memory out to the bulk storage device.

"EXAMINE REG's" (X) - Displays the contents of the specified "virtual" CPU register or flag status. The "virtual" CPU registers and flag status is their actual contents at the time a "breakpoint" is encountered, or at the time a "GO TO" is issued. The value of the "virtual" CPU registers (but not the flag status) may be altered by this command.

Each of the commands are entered by the operator entering the letter illustrated in the parenthesis followed by whatever data is required to define the action to be taken. Most of the commands require the specification if either command type, memory address (or addresses), or data, or a combination of these to define the exact operation of the command. The format for entering each command is summarized on the following page.
COMMAND

BREAKPOINT (TYPE 1)  B1 HXX LLL
BREAKPOINT (TYPE 2)  B2 HXX LLL
MEMORY DUMP        D HXX LLL, MMM NNN
MEMORY FILL        F HXX LLL, MMM NNN, DDD
GO TO (TYPE 1)      G1 HXX LLL
GO TO (TYPE 2)      G2 HXX LLL
MEMORY MODIFY      M HXX LLL
BULK READ          R
SEARCH             S HXX LLL, MMM NNN, DDD
TRANSFER          T HXX LLL, MMM NNN, YYY ZZZ
BULK WRITE        W HXX LLL, MMM NNN
EXAMINE REGISTER  XP

WHERE "HXX LLL", "MMM NNN", AND "YYY ZZZ" INDICATE MEMORY ADDRESS'S AFFECTED BY THE COMMANDS. "DDD" IS THE DATA VALUE USED IN THE COMMAND AND "P" IS THE REGISTER DESIGNATION IN THE EXAMINE REGISTER COMMAND. "P" IS REPLACED BY THE LETTERS "A" THRU "F", "H" OR "L" TO INDICATE THE "VIRTUAL" CPU REGISTER TO BE EXAMINED OR THE LETTER "F" TO INDICATE THE FLAG STATUS IS TO BE DISPLAYED.

THE MEMORY ADDRESS AND DATA INFORMATION SHOWN ABOVE USES GROUPS OF THREE OCTAL DIGITS TO SPECIFY THE COMMAND'S OPERATION. EACH GROUP HAS A POSSIBLE RANGE OF VALUES FROM 000 TO 377. MEMORY ADDRESSES ARE SPECIFIED BY TWO GROUPS, THE FIRST GROUP BEING THE HIGH, OR PAGE, ADDRESS, WHILE THE SECOND GROUP DEFINES THE LOW PORTION OF THE ADDRESS. THE DATA VALUE IS SPECIFIED BY A SINGLE THREE DIGIT GROUPING. THIS NOTATION WAS CHOSEN BECAUSE IT IS A GENERALLY ACCEPTED FORMAT FOR REPRESENTING 8-BIT BINARY INFORMATION WHICH SHOULD BE FAMILIAR TO MOST MICROCOMPUTER USERS. IT SHOULD BE NOTED THAT WHEN ENTERING A COMMAND, LEADING ZEROS MAY BE DELETED; HOWEVER, EACH GROUP MUST BE REPRESENTED BY AT LEAST ONE DIGIT. THAT IS, IF THE MEMORY LOCATION 000 000 IS TO BE MODIFIED, THE COMMAND MAY BE ENTERED USING ONE OF THE FOLLOWING FORMS:

M 000 000
OR
M 0 0

THE MONITOR PROGRAM

GENERAL UTILITY SUBROUTINES

THERE ARE A GROUP OF SUBROUTINES USED BY THE MAJOR ROUTINES OF THE MONITOR PROGRAM WHICH PERFORM MANY OF THE COMMON TASKS REQUIRED BY THESE ROUTINES. SUCH SMALL SEQUENCES OF INSTRUCTIONS ARE REFERRED TO AS "UTILITY" SUBROUTINES BECAUSE OF THEIR BROAD, GENERAL USAGE THROUGHOUT THIS PROGRAM. THESE SUBROUTINES ARE PRESENTED IN THIS SECTION TO POINT OUT IMPORTANT FACTORS RELATING TO THEIR OPERATION SO THAT THE READER MAY HAVE A GOOD UNDERSTANDING OF THE SUBROUTINES WHICH FORM THE BASE OF THE MONITOR PROGRAM. ALTHOUGH THESE SUBROUTINES WERE WRITTEN FOR THE MONITOR PROGRAM, THE READER MAY FIND MANY OF THEM USEFUL IN APPLYING THEM TO OTHER PROGRAMS ONE MAY DEVELOP.
THE FIRST GROUP OF "UTILITY" SUBROUTINES PERFORM THE TYPE OF OPERATIONS FOUND IN ALMOST ANY PROGRAM. THESE OPERATIONS INCLUDE INCREMENTING THE MEMORY POINTER IN REGISTER PAIR "H" AND "L," INCREMENTING A DOUBLE PRECISION VALUE STORED IN MEMORY AND SWITCHING THE CONTENTS OF REGISTERS "H" AND "L" WITH THE CONTENTS OF REGISTERS "D" AND "E," RESPECTIVELY. THESE SUBROUTINES ARE QUITE BASIC BUT ARE NEVER-THE-LESS IMPORTANT FOR MAINTAINING EFFICIENT USE OF MEMORY. AN ADDITIONAL SUBROUTINE IS INCLUDED HERE LABELED "SETUP" WHICH SETS THE MEMORY POINTER REGISTERS "H" AND "L" TO THE CONTENTS OF MEMORY LOCATIONS 167 AND 166 ON PAGE 00, RESPECTIVELY. THIS SUBROUTINE IS USED TO SET THE MEMORY POINTER TO THE MEMORY LOCATION CURRENTLY BEING OPERATED ON BY THE COMMAND.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>INMEM, INL</td>
<td>/INCR LO ADDR</td>
</tr>
<tr>
<td>RFZ</td>
<td>/IF NON ZERO, RET</td>
</tr>
<tr>
<td>INH</td>
<td>/ELSE, INCR PG ADDR</td>
</tr>
<tr>
<td>RET</td>
<td>/RET TO CALLING PGM</td>
</tr>
<tr>
<td>INC, ADI 001</td>
<td>/INCR CONTENTS OF MEM LOC</td>
</tr>
<tr>
<td>LMA</td>
<td>/RESTORE MEM CONTENTS</td>
</tr>
<tr>
<td>RFC</td>
<td>/IF NO CARRY, RET</td>
</tr>
<tr>
<td>INL</td>
<td>/ELSE, FETCH NXT LOC</td>
</tr>
<tr>
<td>LAM</td>
<td>/INCR MEM CONTENTS</td>
</tr>
<tr>
<td>ADI 001</td>
<td>/RESTORE MEM CONTENTS</td>
</tr>
<tr>
<td>LMA</td>
<td>/RET TO CALLING PGM</td>
</tr>
<tr>
<td>SWITCH, LCH</td>
<td>/SWITCH THE PNTR IN</td>
</tr>
<tr>
<td>LHD</td>
<td>/REG'S H AND L WITH</td>
</tr>
<tr>
<td>LDC</td>
<td>/THE PNTR IN REG'S D AND E</td>
</tr>
<tr>
<td>LCL</td>
<td>LLE</td>
</tr>
<tr>
<td>LEC</td>
<td>/RET TO CALLING PGM</td>
</tr>
<tr>
<td>SETUP, LHI 000</td>
<td>/SET PNTR TO 00 166</td>
</tr>
<tr>
<td>LLI 166</td>
<td>/FETCH LO ADDR</td>
</tr>
<tr>
<td>INL</td>
<td>LHM</td>
</tr>
<tr>
<td>LLC</td>
<td>/FETCH PG ADDR</td>
</tr>
<tr>
<td>RET</td>
<td>/SET PNTR TO MEM LOC</td>
</tr>
<tr>
<td></td>
<td>/RET TO CALLING PGM</td>
</tr>
</tbody>
</table>

THE NEXT GROUP OF SUBROUTINES PRESENTED BELOW ARE USED TO OUTPUT VARIOUS MESSAGES TO THE DISPLAY OUTPUT DEVICE. THREE OF THESE MESSAGE PRINTOUT ROUTINES OUTPUT A FIXED MESSAGE TO THE PRINTER. THE ROUTINE LABELED "SPAC" OUTPUTS A SPACE CHARACTER (ASCII CODE '240') AND THE ROUTINE "COLON" OUTPUTS A COLON (ASCII CODE '272') BY LOADING THE RESPECTIVE CODES IN THE ACCUMULATOR AND CALLING THE DISPLAY OUTPUT ROUTINE. "HDLN" SETS UP A POINT TO THE "CANNED" MESSAGE "CARRIAGE-RETURN/LINE-FEED" AND THEN FALLS THROUGH TO THE SUBROUTINE "MSG" TO PRINT THE "CRLF" COMBINATION. THE "MSG" SUBROUTINE OUTPUTS A STRING OF CHARACTERS STORED IN MEMORY TO THE DISPLAY DEVICE UNTIL A "ZERO" BYTE IS ENCOUNTERED. THE PROGRAM CALLING "MSG" SIMPLY SETS REGISTERS "H" AND "L" TO THE START ADDRESS OF THE MESSAGE TO BE PRINTED AND CALLS "MSG." THIS SUBROUTINE MAY BE OF USE TO THE READER IN DEVELOPING PROGRAMS WHICH REQUIRE
THE PRINTOUT OF "CANNED MESSAGES." THE SUBROUTINE Labeled "PRT166" OutputS THE MEMORY ADDRESS CONTAINED IN LOCATIONS 166 AND 167 ON PAGE 00. LOCATION 167, WHICH CONTAINS THE HIGH PORTION OF THE ADDRESS, IS PRINTED FIRST FOLLOWED BY A SPACE AND THEN THE LOW PORTION, CONTAINED IN LOCATION 166. THIS IS USED BY Several ROUTINES, SUCH AS THE "MODIFY," "DUMP" AND "SEARCH" ROUTINES, TO PRINT THE AFFECTIVE MEMORY ADDRESSES. THIS ROUTINE CALLS THE SUBROUTINE "OCTOUT" TO PRINT EACH THREE DIGIT OCTAL NUMBER. "OCTOUT" SEPARATES EACH DIGIT FROM THE 8-BIT BYTE, FORMS THE ASCII CODE FOR THE DIGIT AND CALLS THE DISPLAY OUTPUT ROUTINE TO PRINT IT. THE FINAL SUBROUTINE, Labeled "MEMPR," PRINTS THE CONTENTS OF THE MEMORY LOCATION INDICATED BY THE POINTER AT LOCATION 166 AND 167 ON PAGE 00. THIS ROUTINE USES THE SUBROUTINE "SETUP" TO SET THE MEMORY POINTER AND THEN CALLS "OCTOUT" TO PRINT THE MEMORY CONTENTS.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPAC, LAI 240</td>
<td>SET ASCII CODE FOR SPACE</td>
</tr>
<tr>
<td>JMP PRINT</td>
<td>PRINT SPACE AND RET</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>COLON, LAI 272</td>
<td>SET ASCII CODE FOR :</td>
</tr>
<tr>
<td>JMP PRINT</td>
<td>PRINT COLON AND RET</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>HDLN, LLI 134</td>
<td>SET PNTR TO C/R, L/F MSG</td>
</tr>
<tr>
<td>LHI 000</td>
<td>FALL THRU TO PRINT MSG</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>MSG, LAM</td>
<td>FETCH CHAR TO PRINT</td>
</tr>
<tr>
<td>NDA</td>
<td>END OF MSG CHAR?</td>
</tr>
<tr>
<td>RTZ</td>
<td>YES, RET TO CALLING PGM</td>
</tr>
<tr>
<td>CAL PRINT</td>
<td>NO, PRINT CHAR</td>
</tr>
<tr>
<td>CAL INMEM</td>
<td>INCR MSG PNTR</td>
</tr>
<tr>
<td>JMP MSG</td>
<td>CONTINUE PRINT OUT</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>PRT166, LLI 167</td>
<td>SET PNTR TO PG ADDR</td>
</tr>
<tr>
<td>LHI 000</td>
<td>OF LO ADDR STORED</td>
</tr>
<tr>
<td>LAM</td>
<td>FETCH PG ADDR</td>
</tr>
<tr>
<td>NDI 077</td>
<td></td>
</tr>
<tr>
<td>CAL OCTOUT</td>
<td>PRINT PAGE ADDR</td>
</tr>
<tr>
<td>CAL SPAC</td>
<td>PRINT A SPACE</td>
</tr>
<tr>
<td>LLI 166</td>
<td>SET PNTR TO LO ADDR</td>
</tr>
<tr>
<td>LAM</td>
<td>FETCH LO ADDR</td>
</tr>
<tr>
<td>CAL OCTOUT</td>
<td>PRINT LO ADDR</td>
</tr>
<tr>
<td>/</td>
<td>FALL THRU TO PRINT SPACE</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>OCTOUT, LLA</td>
<td>SAVE OCTAL NUMBER TO PRINT</td>
</tr>
<tr>
<td>RLC</td>
<td>POSITION HUNDRED'S DIGIT</td>
</tr>
<tr>
<td>RLC</td>
<td></td>
</tr>
<tr>
<td>NDI 003</td>
<td>MASK OFF OTHER BITS</td>
</tr>
<tr>
<td>ORI 260</td>
<td>FORM ASCII CODE</td>
</tr>
<tr>
<td>CAL PRINT</td>
<td>PRINT DIGIT</td>
</tr>
<tr>
<td>LAL</td>
<td>FETCH OCTAL NUMBER</td>
</tr>
<tr>
<td>RRC</td>
<td>POSITION TEN'S DIGIT</td>
</tr>
<tr>
<td>RRC</td>
<td></td>
</tr>
<tr>
<td>RRC</td>
<td></td>
</tr>
<tr>
<td>NDI 007</td>
<td>MASK OFF OTHER DIGITS</td>
</tr>
<tr>
<td>ORI 260</td>
<td>FORM ASCII CODE</td>
</tr>
<tr>
<td>CAL PRINT</td>
<td>PRINT DIGIT</td>
</tr>
<tr>
<td>LAL</td>
<td>FETCH OCTAL NUMBER</td>
</tr>
<tr>
<td>NDI 007</td>
<td>MASK OFF OTHER DIGITS</td>
</tr>
<tr>
<td>ORI 260</td>
<td>FORM ASCII CODE</td>
</tr>
<tr>
<td>JMP PRINT</td>
<td>PRINT DIGIT AND RET</td>
</tr>
</tbody>
</table>
THE READER SHOULD NOW UNDERSTAND THAT THE MONITOR PROGRAM IS CONTROLLED BY THE OPERATOR ENTERING COMMANDS ON THE OPERATOR INPUT DEVICE. ONCE THE COMMAND IS ENTERED AND RECOGNIZED, THE COMPUTER JUMPS TO THE MAJOR ROUTINE TO PERFORM THE DESIGNATED FUNCTION. WHEN THE MAJOR ROUTINE IS ENTERED, IT MAY BE NECESSARY TO RETRIEVE MORE INFORMATION FROM THE INPUT BUFFER IN ORDER TO PROCESS THE COMMAND. THE ADDITIONAL DATA IS ALMOST ALWAYS IN THE FORM OF OCTAL DIGITS WHICH SPECIFY MEMORY ADDRESSES OR DATA. THIS INFORMATION IS STORED IN THE INPUT BUFFER AS A STRING OF ASCII CHARACTERS AND MUST BE TRANSLATED INTO ITS EQUIVALENT BINARY VALUE(S) BEFORE THE MAJOR ROUTINE CAN USE IT. SINCE THIS FUNCTION IS A COMMON PROCESS THE FOLLOWING ASCII TO OCTAL AND OCTAL TO BINARY CONVERSION SUBROUTINES ARE USED TO PERFORM THE TRANSLATION. THE SUBROUTINE "OCTNM" READS IN A MEMORY ADDRESS, CONVERTS IT TO THE BINARY VALUE AND STORES IT IN LOCATIONS 166 AND 167 ON PAGE 00. IF A SECOND ADDRESS FOLLOWS THE FIRST IN THE INPUT BUFFER, THE SECOND ADDRESS WILL BE CONVERTED TO BINARY AND STORED IN LOCATIONS 170 AND 171 ON PAGE 00. IF THERE IS NO SECOND ADDRESS, THE FIRST ADDRESS WILL BE STORED AGAIN IN LOCATIONS 170 AND 171. THE TWO ADDRESSES STORED ARE THEN CHECKED AGAINST EACH OTHER TO DETERMINE THAT THE FIRST IS LESS THAN OR EQUAL TO THE SECOND. IF NOT, AN ERROR MESSAGE IS PRINTED AND CONTROL RETURNS TO THE COMMAND MODE. ALSO, AS THE CONVERSION IS BEING PERFORMED, THE INPUT IS CHECKED FOR POSSIBLE ERRORS, SUCH AS INVALID OCTAL NUMBERS (i.e. 8, 9) OR INVALID ENTRIES (i.e. ONLY ONE THREE DIGIT GROUP DEFINING AN ADDRESS). IF SUCH ERRORS ARE FOUND, AN ERROR MESSAGE IS PRINTED AND CONTROL RETURNS TO THE COMMAND MODE. THE ACTUAL ASCII TO OCTAL ("CDNMM") AND OCTAL TO BINARY ("OCT") ROUTINES ARE IN THE FORM OF SUBROUTINES TO ALLOW THEM TO BE CALLED SEPARATELY WHEN REQUIRED.

```
MENOMNOC            COMMENTS
-------------------  -------------------
/                   /                    
MEMPR, CAL SETUP   /SET PNTR TO MEM LOC
LAM                 /FETCH CURRENT MEM CONTENTS
JMP OCTOUT          /PRINT CONTENTS AND RIT
/                   /                    
```

```
MENOMNOC            COMMENTS
-------------------  -------------------
OCTNM, LEL          /SAVE INP BFR PNTR
CAL OCTPR           /CONVERT 1ST OCTAL PAIR
LLI 166             /SET PNTR TO LO ADDR STRAGE
LMB                 /SAVE LO HALF OF LO ADDR
INL                 /SAVE PG HALF OF LO ADDR
LMC                 /SAVE PG HALF OF LO ADDR
LLE                 /RESTORE INP BFR PNTR
LAM                 /FETCH NXT CHAR
CPI 254             /NO, ONLY ONE ENTRY
JFZ SGL             /YES, INCR INP BFR PNTR
INL                 /SAVE INP BFR PNTR
LAL                 /SAVE INP BFR PNTR
CAL OCTPR           /CONVERT 2ND OCTAL PAIR
SGL, LLI 170        /SET PNTR TO HI ADDR STRAGE
LMB                 /SAVE LO HALF OF HI ADDR
INL                 /SAVE PG HALF OF HI ADDR
LMC                 /SAVE PG HALF OF HI ADDR
LAC                 /SAVE PG HALF OF HI ADDR
```
LLI 167  /IS HI ADDR < LO ADDR?
CPM
JTC ERR  /YES, PRINT ERROR
RFZ  /IF PG HALF NOT =, RET
INL  /ELSE, CHECK LO HALF
LAM
LLI 166  /IS HI ADDR < LO ADDR?
CPM
JTC ERR  /YES, PRINT ERROR MSG
RET  /NO, RET TO CALLING PGM
/
OCTPR, CAL DCDNM  /DECODE 1ST OCTAL NUMBER
LCB  /SAVE OCTAL NUMBER
INE  /INCR INP BFR PNTR
/  /FALL THRU TO DECODE 2ND NMBR
/
DCDNM, LLI 150  /SET PNTR TO DIGIT STRAGE TBL
LMH  /CLEAR TBL BY STORING 000.
INL
LMH
INL
LMH
LLE
LOOP, CAL FNUM  /RESET INP BFR PNTR
JTS CKLNH  /CHECK FOR VALID NUMBER
LAM  /IF NOT, CHECK CHAR CNT = 0
LDL  /FETCH CHAR
NDI 007  /SAVE INP BFR PNTR
LLI 150  /MASK OFF 260
LBM  /STORE OCTAL NUMBER IN
LMA  /TABLE AT LOC 150 PG 00
LMA  /AND SHIFT OTHER NUMBERS
INL  /UP THRU THE TABLE
LAM
LMB
INL
LMA
LLD
INL
JMP LOOP  /RESTORE AND INCR INP BFR PNTR
/
CKLNH, LAL  /FETCH NXT NUMBER
CPE
JTZ ERR  /YES, PRINT ERROR MSG
LEL  /NO, SAVE INP BFR PNTR
CAL OCT  /FETCH FINAL OCTAL NMBR
JFS ERR  /IF INVALID, PRINT ERR MSG
RET  /ELSE, RET TO CALLING PGM
/
FNUM, LAM  /IS CHAR A VALID NUMBER?
CPI 260
RTS  /NO, RET WITH S FLAG SET
SUI 270  /CHECK UPPER LIMIT BY
ADI 200  /SETTING S FLAG TO PROPER
RET  /STATE AND RETURN
THE NEXT SUBROUTINE TO BE PRESENTED IS LABELED "CKEND." THIS SUBROUTINE IS UTILIZED BY A NUMBER OF MAJOR ROUTINES WHICH OPERATE ON A GROUP OF MEMORY LOCATIONS SUCH AS THE "DUMP," "FILL" AND "SEARCH" ROUTINES. THE BASIC FUNCTION OF THIS ROUTINE IS TO COMARE THE VALUES OF THE POINTERS STORED IN THE DATA AREA ON PAGE 00 AT LOCATIONS 166 THRU 171 WHICH WERE INITIALLY SET UP BY INPUTTING THE COMMAND. AS EACH LOCATION IS OPERATED ON, THE TWO POINTERS ARE CHECKED TO DETERMINE IF THEY ARE EQUAL, INDICATING THE OPERATION IS COMPLETE. IF THEY ARE NOT EQUAL, THE POINTER AT LOCATION 166 AND 167 IS INCREMENTED AND THE PROCESSING IS CONTINUED. WHEN THEY BECOME EQUAL, THE PROGRAM RETURNS TO THE COMMAND MODE.

Mnemonic | Comments
----------|----------
CJKLMNOP  | LHI 000  | /SET PNTR TO HI ADDR
LAM       |          | /FETCH 2ND HALF
CJKLMNOP  | LLI 167  | /SET PNTR TO 2ND HALF LO ADDR
CPM       |          | /2ND HALFS EQUAL?
JFZ CONT  |          | /NO, CONTINUE PROCESS
INL       |          | /FETCH 1ST HALF HI ADDR
LAM       |          | /FETCH 1ST HALF LO ADDR
CJKLMNOP  | LLI 166  | /SET PNTR TO 1ST HALF LO ADDR
CPM       |          | /1ST HALFS EQUAL?
JFZ INCMD |          | /YES, RET TO CMND MODE
LAM       |          | /NO, SET PNTR TO LO ADDR
JMP INCR  |          | /INCR LO ADDR AND RET

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>INSPCL, LLI 340</td>
<td>/SET PNTR TO S.A. OF INP BFR</td>
</tr>
<tr>
<td>LPIN, CAL RCV</td>
<td>/INP CHAR</td>
</tr>
<tr>
<td>LMA</td>
<td>/STORE CHAR IN INP BFR</td>
</tr>
<tr>
<td>CPI 240</td>
<td>/CHAR = SPACE</td>
</tr>
<tr>
<td>RTZ</td>
<td>/YES, RET TO CALLING PGM</td>
</tr>
<tr>
<td>CPI 215</td>
<td>/NO, CHAR = C/R</td>
</tr>
<tr>
<td>JTZ INCMD</td>
<td>/YES, RET TO COMMAND MODE</td>
</tr>
<tr>
<td>INL</td>
<td>/NO, INCR INP BFR PNTR</td>
</tr>
<tr>
<td>JTZ ERR</td>
<td>/INP BFR FULL? YES, ERROR</td>
</tr>
<tr>
<td>JMP LPIN</td>
<td>/NO, INP NXT CHAR</td>
</tr>
</tbody>
</table>

THE SUBROUTINE LABELED "ADRDTA" IS USED BY SEVERAL OF THE ROUTINES WHICH REQUIRE THE SPECIFICATION OF A PAIR OF MEMORY ADDRESSES FOLLOWED BY A DATA BYTE, SUCH AS THE "FILL" AND "SEARCH" ROUTINES. THIS SUBROUTINE CALLS "OCTNM" TO FETCH THE ADDRESSES FROM THE INPUT BUFFER AND STORES THEM IN BINARY FORM IN THE DATA STORAGE AREA ON PAGE 00 AND THEN CALLS "DCDNM" TO FETCH THE DATA BYTE WHICH IS RETURNED IN REGISTER B.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADRDTA, LLI 342</td>
<td>/SET PNTR TO ADDR INP</td>
</tr>
<tr>
<td>CAL OCTNM</td>
<td>/INP START AND END ADDR</td>
</tr>
<tr>
<td>INE</td>
<td>/INCR TO DATA POSITION</td>
</tr>
<tr>
<td>JMP DCDNM</td>
<td>/FETCH DATA FM INP BFR</td>
</tr>
</tbody>
</table>
MAJOR ROUTINES FOR THE MONITOR PROGRAM

"COMMAND" INPUT ROUTINE

This section describes the major operating routines used in the monitor program presented herein. The first such routine in this category is designated the "command input routine." The command input routine is set up with a very general format which may be applied to other programs that require a command "look up" operation. Essentially, the command input routine accepts a command input from the operator input device and directs the computer to the start address of the routine which performs the associated operation. The command input routine is easily expandable to accommodate the addition of other functions the user may desire to include in the monitor program. The basic operating portion of this routine is the same regardless of the number of commands there are in the program. To change the number of commands available, one merely adds the information required to the command "look up table" and increases the command counter to indicate the total number of commands.

The flow chart for the command input routine is illustrated below. As the flow chart indicates, the basic concept of this routine is quite simple and straightforward.

FLOW CHART - COMMAND INPUT ROUTINE

The command input routine starts by displaying a "command mode" symbol on the display device. This symbol (defined as a "->" mark) indicates to the operator that the monitor program is currently in the command mode. The operator input routine (to be described next) is then called to input the command from the operator input device. After the operator enters the command, the command look up table is searched for a match with the first character in the command now stored in the input buffer. This character is assumed to be one of the command identifica-
TION LETTERS, AS DESCRIBED PREVIOUSLY. THE LOOK UP TABLE IS SEARCHED BY
COMPARING THE CHARACTER ENTERED TO EVERY THIRD BYTE OF THE COMMAND "LOOK
UP" TABLE. THE FORMAT FOR THE "LOOK UP" TABLE IS ILLUSTRATED BELOW.

BYTE N XXX = ASCII CODE FOR A COMMAND CHARACTER
BYTE N+1 YYY = LOW ADDR OF ASSOC COMMAND ROUTINE
BYTE N+2 ZZZ = PAGE ADDR OF ASSOC COMMAND ROUTINE
BYTE N+3 MMM = ASCII CODE FOR A COMMAND CHARACTER
BYTE N+4 NNN = LOW ADDR OF ASSOC COMMAND ROUTINE
BYTE N+5 OOO = PAGE ADDR OF ASSOC COMMAND ROUTINE
BYTE N+6 AAA = ASCII CODE FOR A COMMAND CHARACTER

REPEAT SEQUENCE TO END OF COMMAND LOOK UP TABLE

IF A MATCH IS FOUND BETWEEN THE CHARACTER ENTERED AND AN ENTRY IN THE
COMMAND LOOK UP TABLE, THE ADDRESS IN THE SUCCEEDING TWO BYTES OF THE
COMMAND LOOK UP TABLE ARE OBTAINED AND TRANSFERRED TO TWO SPECIAL LOCA-
TIONS ON PAGE 00. THESE LOCATIONS FORM THE SECOND AND THIRD BYTES OF A
"JUMP" INSTRUCTION WHICH IS THEN EXECUTED TO JUMP TO THE COMMAND ROUTINE
AS SPECIFIED IN THE COMMAND JUST RECEIVED. IF, HOWEVER, THERE IS NO
MATCH FOUND IN THE LOOK UP TABLE, THIS IS ASSUMED TO BE AN ERROR CONDI-
TION AND AN ERROR MESSAGE IS OUTPUT TO THE DISPLAY DEVICE. THE PROGRAM
THEN RETURNS TO THE START OF THE COMMAND INPUT ROUTINE TO RECEIVE A NEW
COMMAND ENTRY.

THE LISTING FOR THE COMMAND "LOOK UP" TABLE FOLLOWED BY THE COMMAND
INPUT ROUTINE FOR THIS MONITOR PROGRAM IS PRESENTED BELOW. THE COMMAND
"LOOK UP" TABLE RESIDES ON PAGE 00 STARTING AT LOCATION 210. THIS LOCA-
TION ALLOWS EXPANSION OF THE LOOK UP TABLE BY SIMPLY ADDING THE ASCII
CODE FOR THE IDENTIFYING CHARACTER FOR THE COMMAND TO BE ADDED, FOLLOWED
BY THE LOW AND PAGE PORTION OF THE START ADDRESS OF THE NEW COMMAND, AS
EXPLAINED ABOVE. THEN SIMPLY INCREMENT THE "IMMEDIATE" PORTION OF THE
7'TH INSTRUCTION (LDI 011) IN THE COMMAND INPUT ROUTINE. THE ACTUAL OP-
PERATING PORTION OF THE COMMAND INPUT ROUTINE AND, THUS, THE MONITOR
PROGRAM ITSELF, STARTS AT THE INSTRUCTION LABELED "INCMD."

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>/COMMAND LOOK UP TABLE</td>
<td>/MODIFY</td>
</tr>
<tr>
<td>315</td>
<td>150</td>
</tr>
<tr>
<td>015</td>
<td>304</td>
</tr>
<tr>
<td>275</td>
<td>015</td>
</tr>
<tr>
<td>327</td>
<td>343</td>
</tr>
<tr>
<td>015</td>
<td>322</td>
</tr>
<tr>
<td>/BULK WRITE</td>
<td>015</td>
</tr>
<tr>
<td>371</td>
<td>302</td>
</tr>
<tr>
<td>/BULK READ</td>
<td>377</td>
</tr>
<tr>
<td>015</td>
<td></td>
</tr>
</tbody>
</table>
A FLOW CHART OF THE ENTIRE MONITOR PROGRAM IN THIS MANUAL IS PRESENTED ON THE FOLLOWING PAGE. IT CAN ACTUALLY BE THOUGHT OF AS A MORE DETAILED VERSION OF THE COMMAND INPUT ROUTINE FLOW CHART, SINCE IT DEFINES EACH COMMAND THAT IS SEARCHED FOR IN THE COMMAND INPUT ROUTINE. THE READER MAY DESIRE TO REFER TO THIS FLOW CHART FROM TIME-TO-TIME TO SEE HOW VARIOUS FUNCTIONS OF THE PROGRAM RELATE TO EACH OTHER.
INPUT ROUTINE

THE INPUT ROUTINE IN THIS MONITOR PROGRAM IS USED TO INPUT COMMANDS FROM THE OPERATOR INPUT DEVICE. THE ROUTINE ACCEPTS INPUTS FROM AN EXTERNAL DEVICE BY CALLING THE "RCV" SUBROUTINE AND STORES THE CHARACTERS IN THE INPUT BUFFER RESIDING ON PAGE 00 UNTIL A TERMINATING CHARACTER IS RECEIVED. THE ROUTINE ALLOWS THE CORRECTION OF INDIVIDUAL CHARACTERS ENTERED AND THE CAPABILITY TO ABDOT THE CURRENT INPUT AND RETURN TO THE COMMAND MODE.

THE FLOW CHART FOR THE INPUT ROUTINE IS PRESENTED ON THE FOLLOWING PAGE. THE READER MAY REFER TO THIS DURING THE FOLLOWING DISCUSSION.

THE FIRST OPERATION PERFORMED BY THIS ROUTINE IS TO "CLEAR OUT" THE INPUT BUFFER AREA. THIS IS ACCOMPLISHED BY_FILLING THE INPUT BUFFER AREA WITH THE ASCII CODE FOR A SPACE, '240' OCTAL. THE START ADDRESS OF THE INPUT BUFFER IS THEN SET UP TO BEGIN STORING CHARACTERS AS THEY ARE ENTERED VIA THE "RCV" ROUTINE. AS EACH CHARACTER IS ENTERED, IT IS RETURNED TO THE INPUT ROUTINE IN THE ACCUMULATOR. THE CHARACTER IS THEN TESTED TO DETERMINE IF IT IS ONE OF THE "CONTROL" CHARACTERS.

THE FIRST CONTROL CHARACTER TESTED FOR IS THE "CONTROL/D," ASCII CODE 204 OCTAL. THIS IS GENERALLY ENTERED BY SIMULTANEOUSLY DEPRESSING THE "CONTROL" KEY AND THE "D" ON AN ASCII ENCODED KEYBOARD. RECEIPT OF "CONTROL D" INDICATES THE OPERATOR WISHES TO ABDOT THE CURRENT INPUT AND START A NEW COMMAND INPUT.


THE FINAL CONTROL CHARACTER TESTED FOR BY THE INPUT ROUTINE IS THE ASCII CODE 377 OCTAL, WHICH IS ASSIGNED TO THE "RUBOUT" OR "DELETE" FUNCTION. RECEIPT OF THIS CHARACTER INDICATES TO THE INPUT ROUTINE THAT THE PREVIOUS CHARACTER ENTERED BY THE OPERATOR IS TO BE DELETED FROM THE INPUT BUFFER. THIS IS ACCOMPLISHED BY BACKING UP THE INPUT BUFFER POINTER ONE LOCATION AND INSERTING THE CODE FOR A "SPACE" TO EFFECTIVELY "ERASE" ONE CHARACTER ENTRY FROM THE INPUT BUFFER. AN OPERATOR MAY ERASE MORE THAN ONE CHARACTER BY USING THE "RUBOUT" FUNCTION SEVERAL TIMES IN SUCCESSION.

IF NONE OF THE PREVIOUSLY MENTIONED "CONTROL" CHARACTERS ARE FOUND BY THE INPUT ROUTINE, THE CODE FOR THE CHARACTER ENTERED WILL BE STORED IN THE INPUT BUFFER AND THE INPUT BUFFER POINTER WILL BE ADVANCED. THIS PROCESS WILL CONTINUE AS LONG AS CHARACTERS ARE ENTERED FROM THE OPERATOR INPUT DEVICE. HOWEVER, ONCE THE INPUT BUFFER IS FILLED, NO FURTHER STORAGE WILL TAKE PLACE, PREVENTING THE OPERATOR FROM INADVERTENTLY ENTERING TOO MANY CHARACTERS AND OVERFLOWING ONTO PAGE 01. THE INPUT BUFFER IS CAPABLE OF HOLDING 32 CHARACTERS WHICH IS LONGER THAN ANY OF THE INPUTS REQUIRED BY THIS MONITOR PROGRAM.

THE LISTING FOR THE INPUT ROUTINE Follows the FLOWCHART. THE START OF THIS ROUTINE IS AT THE INSTRUCTION LABELED "CDIN."
START

FILL INPUT BUFFER WITH SPACES

INPUT CHAR

YES

ABORT CURRENT MODE?

RETURN TO COMMAND MODE

NO

TERMINATOR CHAR?

YES

RETURN TO CALLING PGM

NO

RUBOUT?

YES

BACK UP INPUT BUFFER PNTR AND STORE A SPACE

NO

IS INPUT BUFFER FULL?

YES

STORE CHAR IN INPUT BUFFER

NO
CDIN, LLI 340 / SET PNTR TO START OF INP BFR
SPI, LMI 240 / FILL INP BFR WITH SPACES
INL / INCR INP BFR PNTR
JFZ SP1 / DONE? NO, STORE MORE SPACES
LLI 340 / SET INP BFR PNTR
IN2, CAL RCV / INP CHAR FM INP DEVICE
CPI 204 / CHAR = CNT'L D?
JTZ INCMD / YES, RET TO COMMAND MODE
CPI 215 / CHAR = CAR RET?
RTZ / YES, RET TO CALLING PGM
CPI 214 / CHAR = CNT'L L?
RTZ / YES, RET TO CALLING PGM
CPI 377 / CHAR = RUBOUT?
JTZ BDCR / YES, DELETE CHAR FM INP BFR
INL / IS INP BFR FULL?
DCL
JTZ IN2 / YES, DON'T STORE CHAR
LMA / NO, STORE CHARACTER
INL / INCR INP BFR PNTR
JMP IN2 / INP NEXT CHAR
/
BDCR, LAI 340 / SET ACC TO INP BFR 5.A.
CPL / ANY CHARACTERS YET?
JTZ IN2 / NO, CONTINUE INPUT
DCL / YES, BACK UP INP BFR PNTR
LMI 240 / STORE SPACE OVER LAST CHAR
JMP IN2 / CONTINUE INPUT
/

IT SHOULD BE EASY TO SEE THAT THE READER MAY ELECT TO ASSIGN DIFFERENT CHARACTERS TO OPERATE AS "CONTROL" CHARACTERS IN THE INPUT ROUTINE. THIS IS READILY ACCOMPLISHED BY CHANGING THE IMMEDIATE PORTION OF THE "CPI" INSTRUCTIONS IN THE INPUT ROUTINE. FOR EXAMPLE, IF THE USER DESIRES TO HAVE THE CODE FOR "CONTROL 0" (217 OCTAL) SERVE AS THE CONTROL CHARACTER FOR THE "RUBOUT" FUNCTION INSTEAD OF 377 OCTAL, THE USER SIMPLY SUBSTITUTES "217" FOR "377" IN THE "CPI" INSTRUCTION USED TO TEST FOR THE "RUBOUT."

ADDITIONALLY, IF THE USER DESIRES TO ADD OTHER TYPES OF "CONTROL" FUNCTIONS TO THE INPUT ROUTINE, IT COULD BE READILY DONE BY ADDING "CPI" INSTRUCTIONS FOLLOWED BY APPROPRIATE CONDITIONAL "JUMPS" TO USER PROVIDED Routines TO PERFORM THE DESIRED OPERATION.

THE "MODIFY" ROUTINE

THE "MODIFY" ROUTINE IS USED TO DISPLAY AND, IF DESIRED, MODIFY THE CONTENTS OF MEMORY LOCATIONS FOR THE PURPOSE OF LOADING PROGRAMS USING THE KEYBOARD AS THE ENTRY DEVICE, OR CHANGING THE INSTRUCTIONS IN A PROGRAM OR EXAMINING AND REVISING DATA STORED IN MEMORY. THIS ROUTINE DISPLAYS ONE LOCATION AT A TIME, ALLOWING THE OPERATOR TO ENTER CHANGES OR CONTINUE TO DISPLAY THE NEXT LOCATION OR TERMINATE THE OPERATION. THE "MODIFY" ROUTINE PERFORMS IN THE FOLLOWING MANNER.

FIRST, THE ADDRESS ENTERED IN THE COMMAND IS CONVERTED AND STORED IN THE DATA AREA AT LOCATION 166 AND 167 ON PAGE 00. THE "MODIFY" ROU-
TIME THEN PRINTS THE CONTENTS OF THE DESIGNATED MEMORY LOCATION AND CALLS THE "INSPCL" SUBROUTINE TO ALLOW THE OPERATOR TO ENTER THE MODIFICATION. IF A "MOD" IS ENTERED, THE "DCDMN" SUBROUTINE IS CALLED TO DECODE THE NUMBER FROM THE INPUT BUFFER WHICH IS THEN STORED AS THE NEW CONTENTS OF THE SPECIFIED MEMORY LOCATION. WHEN THIS IS COMPLETE, OR IF NO MODIFICATION WAS ENTERED, THE ADDRESS STORED FOR THIS COMMAND WILL BE INCREMENTED AND THIS NEW ADDRESS WILL BE PRINTED ON A NEW LINE ON THE DISPLAY DEVICE. THE PROGRAM THEN LOOPS BACK TO PRINT AND MODIFY THE CONTENTS OF THIS LOCATION. THE LOOP IS TERMINATED BY THE OPERATOR ENTERING A CARRIAGE RETURN OR AN INVALID OCTAL NUMBER FOR THE MODIFICATION.

THE LISTING FOR THIS "MODIFY" ROUTINE IS PRESENTED BELOW AND THE FLOW CHART OF ITS OPERATION FOLLOWS ON THE NEXT PAGE.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODIFY, LLI 342</td>
<td>/SET INP BFR PNTR</td>
</tr>
<tr>
<td>CAL OCTNM</td>
<td>/FETCH ADDR TO MODIFY</td>
</tr>
<tr>
<td>CAL SPAC</td>
<td>/PRINT SPACE</td>
</tr>
<tr>
<td>MODI, CAL MEMPR</td>
<td>/PRINT CONTENTS OF MEM LOC</td>
</tr>
<tr>
<td>CAL COLON</td>
<td>/PRINT COLON</td>
</tr>
<tr>
<td>CAL INSPCL</td>
<td>/INP MODIFICATION</td>
</tr>
<tr>
<td>LAI 340</td>
<td>/WAS MOD ENTERED?</td>
</tr>
<tr>
<td>CPL</td>
<td>/NO, SET UP NXT LOC</td>
</tr>
<tr>
<td>JTZ NXLOC</td>
<td>/YES, SAVE INP PNTR</td>
</tr>
<tr>
<td>LEA</td>
<td>/CONVERT TO OCTAL NUMBER</td>
</tr>
<tr>
<td>CAL DCDNM</td>
<td>/SAVE OCTAL NUMBER</td>
</tr>
<tr>
<td>LAB</td>
<td>/SET PNTR TO MEM ADDR STRAGE</td>
</tr>
<tr>
<td>LLI 166</td>
<td>/SET PNTR TO MEM ADDR STRAGE</td>
</tr>
<tr>
<td>LEM</td>
<td>/FETCH MEM PNTR</td>
</tr>
<tr>
<td>INL</td>
<td>/SET PNTR TO MEM LOC</td>
</tr>
<tr>
<td>LDM</td>
<td>/LOAD MEM WITH NEW VALUE</td>
</tr>
<tr>
<td>CAL SWITCH</td>
<td>/SET PNTR TO PG 00</td>
</tr>
<tr>
<td>LMA</td>
<td>/SET PNTR TO MEM ADDR STRAGE</td>
</tr>
<tr>
<td>NXLOC, LHI 000</td>
<td>/FETCH LO HALF</td>
</tr>
<tr>
<td>LLI 166</td>
<td>/INCR MEM ADDR</td>
</tr>
<tr>
<td>LAM</td>
<td>/PRINT NEXT ADDR TO MODIFY</td>
</tr>
<tr>
<td>CAL MCONT</td>
<td>/PRINT C/R, L/F</td>
</tr>
<tr>
<td>JMP MODI</td>
<td>/ PRINT ADDR TO MODIFY AND RET</td>
</tr>
</tbody>
</table>

THE "DUMP" ROUTINE

THE MEMORY "DUMP" ROUTINE ENABLES THE OPERATOR TO EXAMINE A LARGE BLOCK OF MEMORY LOCATIONS WITH A SINGLE COMMAND ENTRY, AS OPPOSED TO HAVING TO ENTER A CHARACTER IN BETWEEN THE COMPUTER DISPLAYING EACH LOCATION, AS REQUIRED BY THE "MODIFY" ROUTINE. THIS ROUTINE WILL DISPLAY AS MANY LOCATIONS AS DEFINED BY THE START AND END ADDRESSES SPECIFIED IN THE COMMAND.
START

COMMAND VALID?

PRINT MEMORY CONTENTS

INPUT CHANGE FM KYBD

FINSISHED?

PRINT ADDR OF NXT MEM LOC

WAS CHANGE ENTERED?

NO

CHANGE VALID?

NO

RETURN TO COMMAND MODE

YES

PRINT ERROR MSG

YES

STORE NEW MEM CONTENTS

- 25 -
AFTER CONVERTING AND STORING THE ADDRESSES SPECIFIED IN THE COMMAND BY CALLING THE "OCTNM" SUBROUTINE, THE "DUMP" ROUTINE PRINTS THE ADDRESS OF THE FIRST LOCATION TO BE DISPLAYED. A COUNTER IS THEN SET UP WHICH INDICATES THE NUMBER OF LOCATIONS TO BE PRINTED ON THE CURRENT LINE. THIS COUNTER IS SET FOR 20 OCTAL LOCATIONS PER LINE IN THIS PROGRAM AND IS TEMPORARILY STORED ON PAGE 00. THE CONTENTS OF THE MEMORY LOCATIONS ARE THEN PRINTED UNTIL EITHER THE LOCATION PER LINE COUNTER REACHES ZERO OR THE LAST LOCATION SPECIFIED HAS BEEN PRINTED. WHEN THE L/L COUNTER REACHES ZERO, THE L/L COUNTER IS SET TO 20 AGAIN AND A NEW LINE IS STARTED WITH THE ADDRESS OF THE NEXT LOCATION PRINTED FIRST FOLLOWED BY THE CONTENTS OF THE NEXT 20 OCTAL LOCATIONS. THIS ROUTINE RETURNS TO THE COMMAND MODE WHEN THE LAST LOCATION SPECIFIED IN THE COMMAND HAS BEEN PRINTED.

THE DETAILED LISTING FOR THE "DUMP" ROUTINE IS GIVEN BELOW WITH THE FLOW CHART PRESENTED ON THE FOLLOWING PAGE.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDUMP, LLI 342</td>
<td>/SET PNTR TO INP BFR</td>
</tr>
<tr>
<td>CAL OCTNM</td>
<td>/FETCH MEM DUMP LIMITS</td>
</tr>
<tr>
<td>CAL HDLN</td>
<td>/PRINT C/R, L/F</td>
</tr>
<tr>
<td>MDMP1, CAL MCONT</td>
<td>/PRINT ADDR OF 1ST LOC</td>
</tr>
<tr>
<td>CAL SPAC</td>
<td>/PRINT SPACE</td>
</tr>
<tr>
<td>MDMP2, LLI 164</td>
<td>/SET PNTR TO TEMP STRAGE</td>
</tr>
<tr>
<td>LMI 020</td>
<td>/SAVE LOC PER LINE CNTR</td>
</tr>
<tr>
<td>OUTAGN, CAL MIMPRT</td>
<td>/PRINT MEM CONTENTS</td>
</tr>
<tr>
<td>CAL CKEND</td>
<td>/CHECK FOR LAST LOC PRTD</td>
</tr>
<tr>
<td>CAL SPAC</td>
<td>/PRINT SPACE</td>
</tr>
<tr>
<td>LLI 164</td>
<td>/SET PNTR TO L/L CNTR</td>
</tr>
<tr>
<td>LBM</td>
<td>/FETCH CNTR</td>
</tr>
<tr>
<td>DCB</td>
<td>/DECR CNTR</td>
</tr>
<tr>
<td>LMB</td>
<td>/SAVE CNTR. CNTR = 0?</td>
</tr>
<tr>
<td>JTZ MDMP1</td>
<td>/YES, START NEW LINE</td>
</tr>
<tr>
<td>JMP OUTAGN</td>
<td>/NO, PRINT MORE CONTENTS</td>
</tr>
</tbody>
</table>

THE "BULK WRITE" ROUTINE

THE "BULK WRITE" ROUTINE PRESENTED IN THIS MONITOR PROGRAM SIMPLY PROVIDES A SET UP FUNCTION FOR THE USER PROVIDED BULK WRITE OUTPUT ROUTINE. THE PURPOSE OF THIS FUNCTION IS TO PROVIDE A MEANS OF STORING THE CONTENTS OF MEMORY (PROGRAMS OR BLOCKS OF DATA) ON A BULK STORAGE DEVICE VIA A COMMAND FROM THE MONITOR PROGRAM. THE USER'S BULK WRITE ROUTINE IS CALLED BY THIS ROUTINE WITH THE START AND END ADDRESSES OF THE MEMORY LOCATIONS, AS SPECIFIED IN THE COMMAND, STORED IN REGISTERS H AND L FOR THE START LOCATION AND REGISTERS D AND E FOR THE ENDING LOCATION. THIS IS DONE TO MAKE THE INFORMATION READILY AVAILABLE TO THE USER'S BULK WRITE ROUTINE. THE ADDRESSES ARE ALSO CONTAINED IN THE DATA AREA ON PAGE 00, LOCATIONS 166 THRU 171. THE SHORT LISTING FOR THIS ROUTINE IS GIVEN NEXT FOLLOWED BY SOME SUGGESTIONS FOR THE USER'S BULK WRITE OUTPUT ROUTINE.
<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>WRITE, LLI 342</td>
<td>/SET PNTR TO INP BFR</td>
</tr>
<tr>
<td>CAL OCTNM</td>
<td>/FETCH START AND END ADDR</td>
</tr>
<tr>
<td>LLI 166</td>
<td>/SET REG'S H AND L WITH</td>
</tr>
<tr>
<td>LCM</td>
<td>/THE START ADDR AND</td>
</tr>
<tr>
<td>INL</td>
<td>/REG’S D AND E WITH</td>
</tr>
<tr>
<td>LBM</td>
<td>/THE END ADDR OF THE</td>
</tr>
<tr>
<td>INL</td>
<td>/BLOCK OF MEm TO BE</td>
</tr>
<tr>
<td>LEM</td>
<td>/WRITTEN TO THE BULK</td>
</tr>
<tr>
<td>INL</td>
<td>/STORAGE DEVICE.</td>
</tr>
<tr>
<td>LDM</td>
<td></td>
</tr>
<tr>
<td>LHB</td>
<td></td>
</tr>
<tr>
<td>LLC</td>
<td></td>
</tr>
<tr>
<td>CAL PUNCH</td>
<td>/GO TO USER BULK WRITE RTN</td>
</tr>
<tr>
<td>JMP INCMD</td>
<td>/RET TO COMMAND MODE</td>
</tr>
</tbody>
</table>

**NOTES AND SUGGESTIONS FOR THE USER PROVIDED BULK STORAGE Routines**

When creating a bulk storage output routine, one should keep several factors in mind. First, the device being used to store the data will have to be considered when defining the format for storing the data. For example, if a paper tape system is used, the output routine should precede the data with a sequence of "leader/trailer" code, to give the reader a place to start when reading the tape back, followed by addressing information and then the data from the specified memory locations. The sequence can be terminated by either leader/trailer or an "end-of-data" code and then leader/trailer. The leader/trailer code should be a code which is unique to the other data codes transmitted and should provide enough leader and trailer to allow ease of handling. The addressing information can be both the start and end addresses or only the start address with the "end-of-data" code or trailer signaling the end of the data on the tape. A similar format may be used for a magnetic tape system.

Another factor to consider is whether additional information is needed to effectively use the storage device. For example, a disc unit may require the specification of track and/or sector number to store the data. Or, there may be several devices on the system which can be used for storing the data. This information can easily be defined at the time the command is entered, since the command is still available in the input buffer area when the bulk storage routines are called. Suppose there are two tape units associated with the computer system. One will be referred to as unit "A" and the other as unit "B." One could select either tape unit "A" or "B" at the time the read or write command is entered by including a letter at the end of the command which designates the tape unit to be used. The format for the command might look like the following:

```
W HHH LLL,XXX YYY,A OR R,B
```

For these commands, the bulk write routine would write to tape unit "A" and the bulk read would call upon tape unit "B" to receive the data. The user provided bulk storage routines would simply have to look in the input buffer area for the unit designation to determine which is to be used.
Another possibility would be to include a "displacement" address in the bulk read command. That is, when the address information is read in from the storage device, the "displacement" address would be "added" to the address received. This new address would be used as the pointer indicating where to store the data as it is received. Thus, data that was written to the bulk storage from page 01 could be read back and stored in page 03, for example, by specifying a "displacement" address of 002 000.

Above all, the important factor in writing the bulk storage routines is that the data written by the bulk write routine must be in a format that can be read in by the routine called by the bulk read routine discussed next.

The "Bulk Read" Routine

The "bulk read" routine presented here simply calls the user provided bulk storage read routine to read in the data available at the system bulk storage device. The only real function it performs is that of providing a means of accessing the bulk input device by a command from the keyboard and allowing a return to the monitor when the operation is complete.

<table>
<thead>
<tr>
<th>MNE MONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>RDBULK CAL READ</td>
<td>GO TO USER BULK READ RTN</td>
</tr>
<tr>
<td>JMP INCMD</td>
<td>RET TO COMMAND MODE</td>
</tr>
</tbody>
</table>

The routines presented to this point require only 1/2 K of memory for the operating portion, not including the user's I/O routines and omitting the "ADRSTA" subroutine which has not been called as yet. The user with a limited amount of memory may desire to end the monitor program here, since the routines included are sufficient to be used as a small system monitor. For those with an abundance of memory, the following routines will be found to be very helpful in program development and general system operation.

The "Breakpoint" Routine

One of the most difficult tasks in operating a computer system is that of debugging programs. Finding out exactly what is happening to this register or that memory location when a new program is being tried out can be very time consuming if one does not have the proper tools to aid in the process. One "tool" that can be very effective is a "breakpoint" program. A "breakpoint" can be set at a particular point in a program which, when encountered, will stop execution of the program, return to the monitor and save the contents of the CPU registers and flag status at the time the breakpoint was reached. The programmer may then examine the CPU register's contents and the CPU flag status and also the contents of memory locations, which will contain their values at the time the breakpoint was encountered. The breakpoint routine presented here performs this function.
THIS BREAKPOINT ROUTINE IS WRITTEN TO STORE THE CPU REGISTERS IN TWO SEPARATE GROUPS. THE REASON BEING THAT THE 8008 INSTRUCTION SET DOES NOT PROVIDE FOR THE STORAGE OF REGISTERS IN MEMORY UNLESS REGISTERS H AND L HAVE BEEN SET TO POINT TO THE STORAGE LOCATION. THEREFORE, AT LEAST TWO REGISTER VALUES MUST BE SACRIFICED WHEN A BREAKPOINT IS ENCOUNTERED. THIS ROUTINE ALLOWS THE SPECIFICATION OF TWO TYPES OF BREAKPOINTS. A "TYPE 1" BREAKPOINT WILL SAVE THE VALUES OF REGISTERS A, B AND C AND A "TYPE 2" BREAKPOINT WILL SAVE THE VALUES OF REGISTERS D, E, H AND L.

AS NOTED IN THE FLOW CHART ON THE FOLLOWING PAGE, THE BREAKPOINT ROUTINE IS ACTUALLY MADE UP OF TWO SEPARATE ROUTINES. THE FIRST ROUTINE SETS UP THE BREAKPOINT BY STORING A "RESTART 7" INSTRUCTION AT THE LOCATION SPECIFIED IN THE COMMAND AND SAVING THE CONTENTS OF THAT LOCATION SO THAT IT WILL BE RESTORED BACK TO ITS ORIGINAL VALUE AFTER THE BREAKPOINT IS PERFORMED. THE "TYPE" OF BREAKPOINT (1 OR 2) IS THEN DETERMINED FROM THE COMMAND AND THE START ADDRESS FOR THAT TYPE (THE ADDRESS OF "BRK1" FOR A TYPE "1" BREAKPOINT, "BRK2" FOR A TYPE "2" BREAKPOINT) IS STORED AS THE SECOND AND THIRD BYTES OF A JUMP INSTRUCTION AT THE "RESTART 7" LOCATION, PAGE 00 LOCATION 070. IT IS IMPORTANT TO NOTE THAT SHOULD THE BREAKPOINT ROUTINE BE ORIGINATED IN A DIFFERENT LOCATION THAN THE ASSEMBLED VERSION PRESENTED IN THIS MANUAL, THE FOUR INSTRUCTIONS WHICH HAVE THE COMMENTS STARTING WITH FOUR ASTERISK'S (****) MUST HAVE THE IMMEDIATE PORTION OF THE INSTRUCTION CHANGED TO INDICATE THE NEW LOW ADDRESS AND PAGE ADDRESS OF THE INSTRUCTIONS LABELED "BRK1" AND "BRK2." THIS FIRST ROUTINE IS LABELED "BREAK."


THE LISTINGS FOR THE BREAKPOINT ROUTINES ARE PRESENTED NEXT.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>BREAK, CAL ANLYZ</td>
<td>/SET UP ADDRESS OF BP</td>
</tr>
<tr>
<td>LLE</td>
<td></td>
</tr>
<tr>
<td>LHD</td>
<td></td>
</tr>
<tr>
<td>JTZ B1</td>
<td>/DETERMINE IF B1 OR B2</td>
</tr>
<tr>
<td>CPI 262</td>
<td></td>
</tr>
<tr>
<td>JTZ ERR</td>
<td>/ERROR IF NEITHER</td>
</tr>
</tbody>
</table>

- 30 -
THE "BREAKPOINT" ROUTINE FLOW CHART
<table>
<thead>
<tr>
<th><strong>Mnemonic</strong></th>
<th><strong>Comments</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>B2, CAL SETBK</td>
<td>/SET UP BP RST COMMAND</td>
</tr>
<tr>
<td>LMI 176</td>
<td>/**** STORE BP2 FND LA</td>
</tr>
<tr>
<td>INL</td>
<td></td>
</tr>
<tr>
<td>LMI 016</td>
<td>/**** STORE BP2 FND PG</td>
</tr>
<tr>
<td>JMP FINEK</td>
<td>/TO REST OF BP SET UP RTN</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>B1, CAL SETBK</td>
<td>/SET UP BP RST COMMAND</td>
</tr>
<tr>
<td>LMI 112</td>
<td>/**** STORE BP1 FND LA</td>
</tr>
<tr>
<td>INL</td>
<td></td>
</tr>
<tr>
<td>LMI 016</td>
<td>/**** STORE BP1 FND PG</td>
</tr>
<tr>
<td>FINBK, INL</td>
<td></td>
</tr>
<tr>
<td>LME</td>
<td>/STORE BP ORIG LOW ADDR</td>
</tr>
<tr>
<td>INL</td>
<td></td>
</tr>
<tr>
<td>LMD</td>
<td>/STORE BP ORIG PG ADDR</td>
</tr>
<tr>
<td>INL</td>
<td></td>
</tr>
<tr>
<td>LMA</td>
<td>/STORE ORIG BP INSTRUCTION</td>
</tr>
<tr>
<td>JMP INCMD</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>ANLYZ, LLI 343</td>
<td>/SET PNTR TO BUFF SA</td>
</tr>
<tr>
<td>CAL OCTNM</td>
<td>/FETCH ADDR INTO 166, 167</td>
</tr>
<tr>
<td>LLI 341</td>
<td>/RESTORE BUFF SA</td>
</tr>
<tr>
<td>LAM</td>
<td>/GET BP 1 OR 2 COMMAND</td>
</tr>
<tr>
<td>LLI 166</td>
<td>/GET BP LOW ADDR</td>
</tr>
<tr>
<td>LEM</td>
<td>/INTO &quot;E&quot;</td>
</tr>
<tr>
<td>INL</td>
<td></td>
</tr>
<tr>
<td>LDM</td>
<td>/AND BP PAGE</td>
</tr>
<tr>
<td>LLI 156</td>
<td>/PNTR TO JUMP COMMAND</td>
</tr>
<tr>
<td>LME</td>
<td></td>
</tr>
<tr>
<td>INL</td>
<td>/SET UP JUMP ADDRESS</td>
</tr>
<tr>
<td>LMD</td>
<td></td>
</tr>
<tr>
<td>CPI 261</td>
<td>/DETERMINE IF CMND 1 OR 2</td>
</tr>
<tr>
<td>RET</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>SETBK, LAM</td>
<td>/SAVE ORIG CONTENTS OF BP</td>
</tr>
<tr>
<td>LMI 075</td>
<td>/INSERT BP RESTART INSTR</td>
</tr>
<tr>
<td>LEL</td>
<td>/CHANGE PointERS</td>
</tr>
<tr>
<td>LDH</td>
<td></td>
</tr>
<tr>
<td>LHI 000</td>
<td>/SET PAGE 00</td>
</tr>
<tr>
<td>LLI 070</td>
<td>/SET PNTR TO RST 1 LOC</td>
</tr>
<tr>
<td>LMI 104</td>
<td>/STORE JUMP INSTRUCTION</td>
</tr>
<tr>
<td>INL</td>
<td></td>
</tr>
<tr>
<td>RET</td>
<td></td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>BRK1, LHI 000</td>
<td>/SET PAGE 00</td>
</tr>
<tr>
<td>LLI 200</td>
<td>/CPU REGISTER STORAGE LOCS</td>
</tr>
<tr>
<td>LMA</td>
<td>/SAVE ACCUMULATOR</td>
</tr>
<tr>
<td>LLI 201</td>
<td></td>
</tr>
<tr>
<td>LMB</td>
<td>/AND CPU REGS B &amp; C</td>
</tr>
<tr>
<td>LLI 202</td>
<td></td>
</tr>
<tr>
<td>BRKCOM, LMC</td>
<td>/WITHOUT DISTURRING FLAGS</td>
</tr>
<tr>
<td>FLAGT, LAH</td>
<td>/SET UP TEMP REGS</td>
</tr>
<tr>
<td>LBA</td>
<td></td>
</tr>
<tr>
<td>LCA</td>
<td></td>
</tr>
<tr>
<td>JFC NOC</td>
<td>/*TEST FOR CARRY FLAG</td>
</tr>
<tr>
<td>LAI 001</td>
<td>/*SET 1 IN &quot;A&quot; IF CARRY TRUE</td>
</tr>
<tr>
<td>NOC, JFZ NOZ</td>
<td>/*TEST FOR ZERO FLAG</td>
</tr>
<tr>
<td>LBI 010</td>
<td>/*SET 1 IN &quot;B&quot; IF ZERO TRUE</td>
</tr>
<tr>
<td>MNEMONIC</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>NOZ, JFP NOP</td>
<td>TEST FOR PARITY FLAG</td>
</tr>
<tr>
<td>LCI 100</td>
<td>SET 1 IN &quot;C&quot; IF PARITY &quot;1&quot;</td>
</tr>
<tr>
<td>NOP, JFS NOS</td>
<td>TEST FOR SIGN FLAG</td>
</tr>
<tr>
<td>ADI 200</td>
<td>SET MSB IF SIGN TRUE</td>
</tr>
<tr>
<td>NOS, ADB</td>
<td></td>
</tr>
<tr>
<td>ADC</td>
<td>FORM FLAG STATUS BYTE</td>
</tr>
<tr>
<td>LLI 207</td>
<td>STORE FLAG STATUS</td>
</tr>
<tr>
<td>LLI 073</td>
<td>PNTNR TO ORIG BP LOW ADDR</td>
</tr>
<tr>
<td>LEM</td>
<td>GET ORIG LOC OF BP</td>
</tr>
<tr>
<td>INL</td>
<td>AND ORIG PG OF BP</td>
</tr>
<tr>
<td>LDM</td>
<td>AND ORIG BP INSTRUCTION</td>
</tr>
<tr>
<td>INL</td>
<td>SET UP ORIGINAL</td>
</tr>
<tr>
<td>LAM</td>
<td>BREAK POINT POINTERS</td>
</tr>
<tr>
<td>LLE</td>
<td>RESTORE ORIG BKPNT INSTR</td>
</tr>
<tr>
<td>LHD</td>
<td>BACK TO MONITOR</td>
</tr>
<tr>
<td>/</td>
<td>SAVE ORIG VALUE OF H &amp; L</td>
</tr>
<tr>
<td>BRK2, LBH</td>
<td>SET PNTNR TO PAGE 00</td>
</tr>
<tr>
<td>LCL</td>
<td>CPU REGISTER STORAGE LOCS</td>
</tr>
<tr>
<td>LLI 000</td>
<td>SAVE REGS D AND E</td>
</tr>
<tr>
<td>LLI 203</td>
<td>AS WELL AS ORIG H AND L</td>
</tr>
<tr>
<td>LMD</td>
<td>WITHOUT DISTURRING FLAGS</td>
</tr>
<tr>
<td>LLI 204</td>
<td></td>
</tr>
<tr>
<td>LME</td>
<td></td>
</tr>
<tr>
<td>LLI 205</td>
<td></td>
</tr>
<tr>
<td>LMB</td>
<td></td>
</tr>
<tr>
<td>LLI 206</td>
<td></td>
</tr>
<tr>
<td>JMP BRKCOM</td>
<td>TO REST OF BREAKPT RTN</td>
</tr>
</tbody>
</table>

**THE "GO TO" ROUTINE**

The "GO TO" ROUTINE PROVIDES A MEANS OF INITIATING EXECUTION OF A PROGRAM IN MEMORY BY DIRECTING THE MONITOR TO JUMP TO A SPECIFIED ADDRESS. AFTER FETCHING THE ADDRESS FROM THE COMMAND, THE "GO TO" ROUTINE DETERMINES WHICH "TYPE" OF GO TO IS REQUESTED. THAT IS, THE "GO TO" FUNCTION ALLOWS THE SETTING OF A GROUP OF CPU REGISTERS BEFORE JUMPING TO THE PROGRAM. THE TWO GROUPS ARE THE SAME AS THOSE FOR THE BREAKPOINT ROUTINE. A TYPE "1" "GO TO" WILL SET THE VALUES OF REGISTERS A, B AND C FROM THE "VIRTUAL" CPU REGISTER TABLE WHILE A TYPE "2" "GO TO" WILL SET THE VALUES OF REGISTERS D, E, H AND L. THE VALUES IN THE "VIRTUAL" CPU REGISTER TABLE ARE SET UP BY EITHER THE "BREAKPOINT" ROUTINE OR BY THE "EXAMINE REGISTER" ROUTINE TO BE PRESENTED NEXT. THE "GO TO" ROUTINE STARTS AT THE LOCATION LABELED "GOTO." THE LISTING AND FLOW CHART ARE PRESENTED ON THE NEXT PAGE. THE READER WILL NOTE THAT THE "ANLYZ" SUBROUTINE OF THE BREAKPOINT ROUTINE IS ALSO USED BY "GO TO" TO FETCH THE START OF EXECUTION ADDRESS AND FORM THE JUMP INSTRUCTION WHICH IS THE FINAL STEP IN THE "GO TO" ROUTINE.
**MNEMONIC**

GOTO, CAL ANLYZ
JZ G01
CP1 262
JFZ ERR
GO2, LLI 203
LDM
INL
LEM
GOCOM, INL
LBM
INL
LCM
LLC
LHB
JMP 155 000
GO1, LLI 200
LAM
JMP GOCOM

**COMMENTS**

/SET UP ADDR OF GOTO
/TO SET UP CPU REGS A, B, C
/ERROR IF NOT G1 OR G2
/SET UP CPU REGS D, E, H & L
/SET UP CPU REGS A, B, C

---

**THE "GO TO" ROUTINE FLOW CHART**
THE "EXAMINE REGISTER" ROUTINE


THIS ROUTINE STARTS BY FETCHING THE REGISTER DESIGNATION FROM THE INPUT BUFFER AND DETERMINING WHICH IS SPECIFIED. IF A "VIRTUAL" CPU REGISTER IS SPECIFIED, A POINTER IS FORMED TO INDICATE WHICH LOCATION IN THE TABLE IS TO BE DISPLAYED. THE CURRENT VALUE IS PRINTED, FOLLOWED BY A COLON, AND THEN THE "INSPCL" SUBROUTINE IS CALLED TO INPUT ANY CHANGES THE OPERATOR MAY DESIRE TO MAKE TO THE VALUE STORED FOR THAT REGISTER. IF NO MODIFICATION IS ENTERED, THE ROUTINE SIMPLY RETURNS TO THE COMMAND MODE AND THE ORIGINAL CONTENT IS MAINTAINED. IF A MODIFICATION IS ENTERED, THE "DCDMN" SUBROUTINE CONVERTS THE INPUT TO BINARY FORM AND THE NEW VALUE IS STORED IN THE TABLE. IF THE FLAG STATUS IS REQUESTED, THE VALUE CURRENTLY STORED AT LOCATION 207 ON PAGE 00 WILL BE PRINTED AND THE ROUTINE AUTOMATICALLY RETURNS TO THE COMMAND MODE. IF THE REGISTER DESIGNATION IS NOT VALID, THE ILLEGAL ENTRY ERROR MESSAGE IS DISPLAYED.

THE DETAILED LISTING FOR THE "EXAMINE REGISTER" ROUTINE IS PRESENTED BELOW AND THE FLOW CHART IS ON THE FOLLOWING PAGE.

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>XREG, LLI 341</td>
<td>/SET INP BFR PNTR</td>
</tr>
<tr>
<td>LAM</td>
<td>/FETCH REG LETTER</td>
</tr>
<tr>
<td>RGAGN, CPI 301</td>
<td>/IS REG VALID?</td>
</tr>
<tr>
<td>JTC ERR</td>
<td>/NO, PRINT ERROR</td>
</tr>
<tr>
<td>CPI 306</td>
<td>/YES, IS REG A THRU E?</td>
</tr>
<tr>
<td>JFC PHL</td>
<td>/NO, TRY H, L OR F</td>
</tr>
<tr>
<td>SUI 161</td>
<td>/SET UP REG TBL PNTR</td>
</tr>
<tr>
<td>XCOM, LLI 164</td>
<td>/SAVE TBL PNTR IN TEMP STRAGE</td>
</tr>
<tr>
<td>LMA</td>
<td>/SET PNTR TO REG TBL LOC</td>
</tr>
<tr>
<td>LLA</td>
<td>/PRINT SPACE</td>
</tr>
<tr>
<td>CAL SPAC</td>
<td>/FETCH CURRENT REG VALUE</td>
</tr>
<tr>
<td>LAM</td>
<td>/PRINT CURRENT REG VALUE</td>
</tr>
<tr>
<td>CAL OQTOUT</td>
<td>/PRINT COLON</td>
</tr>
<tr>
<td>CAL COLON</td>
<td>/INP MODIFICATION</td>
</tr>
<tr>
<td>CAL INSPCL</td>
<td>/SET INP BFR PNTR</td>
</tr>
<tr>
<td>LEI 340</td>
<td>/WAS MOD ENTERED?</td>
</tr>
<tr>
<td>LAL</td>
<td>/NO, RET TO COMMAND MODE</td>
</tr>
<tr>
<td>CPE</td>
<td>/YES, DECODE OCTAL NUMBER</td>
</tr>
<tr>
<td>JTZ INCMD</td>
<td>/YES, DECODE OCTAL NUMBER</td>
</tr>
<tr>
<td>CAL DCDNM</td>
<td>/SET PNTR TO TEMP STRAGE</td>
</tr>
<tr>
<td>LLI 164</td>
<td>/FETCH REG TBL PNTR</td>
</tr>
<tr>
<td>LLM</td>
<td>/STORE NEW REG VALUE</td>
</tr>
<tr>
<td>LMB</td>
<td>/RET TO COMMAND MODE</td>
</tr>
<tr>
<td>JMP INCMD</td>
<td>/RET TO COMMAND MODE</td>
</tr>
</tbody>
</table>
THE "EXAMINE REGISTER" ROUTINE FLOW CHART
<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>FHL, CPI 310</td>
<td>/IS REG = H?</td>
</tr>
<tr>
<td>JFZ LORF</td>
<td>/NO, TRY L OR F</td>
</tr>
<tr>
<td>LAI 205</td>
<td>/YES, SET REG TBL PNTR</td>
</tr>
<tr>
<td>JMP XCOM</td>
<td>/INP MOD TO REG VALUE</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>LORF, CPI 314</td>
<td>/IS REG = L?</td>
</tr>
<tr>
<td>JFZ F</td>
<td>/NO, TRY F</td>
</tr>
<tr>
<td>LAI 206</td>
<td>/YES, SET REG TBL PNTR</td>
</tr>
<tr>
<td>JMP XCOM</td>
<td>/INP MOD TO REG VALUE</td>
</tr>
<tr>
<td>/</td>
<td></td>
</tr>
<tr>
<td>L, CPI 306</td>
<td>/IS REG = L, FOR FLAGS?</td>
</tr>
<tr>
<td>JFZ ERR</td>
<td>/NO, PRINT ERROR</td>
</tr>
<tr>
<td>CAL SPAC</td>
<td>/PRINT SPACE</td>
</tr>
<tr>
<td>LLI 207</td>
<td>/SET REG TBL PNTR</td>
</tr>
<tr>
<td>LAM</td>
<td>/FETCH FLAG WORD</td>
</tr>
<tr>
<td>CAL OCTOUT</td>
<td>/PRINT FLAG WORD</td>
</tr>
<tr>
<td>JMP INCMD</td>
<td>/RET TO COMMAND MODE</td>
</tr>
</tbody>
</table>

The three routines just presented are all inter-related in one way or another. The "Examine Register" routine sets up the values to be loaded in the CPU registers at the time the "Go To" operation is performed. The "Go To" routine may start the execution of a program which will eventually reach a "Breakpoint" which returns to the "Breakpoint" routine to store the CPU register values and the flag status, which, in turn may be examined by the "Examine Register" routine. This coordination between these routines makes the inclusion of these routines, as a group, a convenient point to complete one's monitor program. The operating portion of the monitor program presented to this point occupies slightly more than 3/4 k bytes of memory. So, if one feels that the routines presented thus far will be sufficient for one's monitor program, the program can be ended here and used to give the operator the necessary basics for a good "Operating System" and "Program Debugging" monitor program. The following routines are presented to give the reader an idea for other types of "Convenience" routines that may be added.

The "Fill" Routine

The memory "Fill" routine is used to fill a block of memory with a specific 8 bit data value. This routine is useful in "Zeroing" a block of memory before executing a program to determine whether that program is writing into the section of memory "Zeroed" out or not. As the reader will see from the listing, this program makes very effective use of subroutines to perform its function. The "ADRDTA" subroutine fetches the pertinent information from the input buffer. The "SETUP" subroutine sets the memory pointer to the memory location to receive the data bytes, and the "CKEND" subroutine determines when the final location has been loaded.

The program listing and flow chart for the "Fill" routine is presented on the next page.
**Mnemonic**

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>FILL, CAL ADRDTA</td>
<td>/INP ADDR AND DATA FM BFR</td>
</tr>
<tr>
<td>FL1, CAL SETUP</td>
<td>/SET UP MEM PNTR</td>
</tr>
<tr>
<td>LMB</td>
<td>/FILL MEM LOC WITH DATA</td>
</tr>
<tr>
<td>CAL CKEND</td>
<td>/DONE? YES, RET TO CMND MODE</td>
</tr>
<tr>
<td>JMP FL1</td>
<td>/NO, CONTINUE WITH FILL</td>
</tr>
</tbody>
</table>

**THE MEMORY "FILL" ROUTINE FLOW CHART**
THE "SEARCH" ROUTINE


<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>SEARCH, CAL ADRDTA</td>
<td>INP ADDR AND DATA FM BFR</td>
</tr>
<tr>
<td>LLI 165</td>
<td>SET PNTR TO SAVE DATA</td>
</tr>
<tr>
<td>LMB</td>
<td>SAVE SEARCH DATA IN MEM</td>
</tr>
<tr>
<td>SH1, LLI 165</td>
<td>SET PNTR TO SRCH DATA</td>
</tr>
<tr>
<td>LAM</td>
<td>FETCH SEARCH DATA</td>
</tr>
<tr>
<td>CAL SETUP</td>
<td>FETCH CONTENTS OF MEM</td>
</tr>
<tr>
<td>CPM</td>
<td>DATA EQUAL SRCH DATA?</td>
</tr>
<tr>
<td>CTZ MCONT</td>
<td>YES, PRINT ADDR</td>
</tr>
<tr>
<td>CAL CKEND</td>
<td>DONE? YES, RET TO CMND MODE</td>
</tr>
<tr>
<td>JMP SH1</td>
<td>NO, CONTINUE SEARCH</td>
</tr>
</tbody>
</table>

THE "TRANSFER" ROUTINE

THE "TRANSFER" ROUTINE ALLOWS THE OPERATOR TO TRANSFER A BLOCK OF MEMORY FROM ONE SECTION OF MEMORY TO ANOTHER, BY SIMPLY SPECIFYING THE START AND END ADDRESS OF THE BLOCK TO BE MOVED, FOLLOWED BY THE START ADDRESS OF THE SECTION TO RECEIVE THE MEMORY CONTENTS IN THE COMMAND. THE "TRANSFER" ROUTINE THEN SETS UP A "FROM" POINTER AND A "TO" POINTER WHICH ARE USED TO TRANSFER THE DATA "FROM" THE ORIGINAL LOCATION "TO" THE NEW LOCATION. THIS ROUTINE USES A SUBROUTINE CALLED "SWAP" NOT ONLY DURING THE ACTUAL TRANSFER OF THE DATA BUT ALSO TO TEMPORARILY SAVE THE ADDRESSES AS THEY ARE READ IN FROM THE INPUT BUFFER. THIS COMMAND CAN BE USEFUL IN SAVING A BLOCK OF DATA IN ONE SECTION OF MEMORY BEFORE USING THE ORIGINAL DATA AREA AGAIN. AFTER THE SECOND USAGE, THE TWO BLOCKS WILL BE AVAILABLE FOR EXAMINATION AND/OR COMPARISON. ANOTHER POSSIBLE APPLICATION IS TO RE-ORIGIN A PROGRAM FROM ONE AREA OF MEMORY TO ANOTHER. OF COURSE, THE JUMP AND CALL INSTRUCTIONS WOULD HAVE TO BE CHANGED TO INDICATE THE NEW ADDRESSES, BUT THIS CAN BE ASSISTED BY USING THE "SEARCH" ROUTINE TO LOCATE THE JUMP AND CALL INSTRUCTIONS WITHIN THE PROGRAM. THIS METHOD OF MOVING PROGRAMS CAN BE EFFECTIVE FOR PROGRAMS WHICH ARE NOT TOO LONG, AS COMPARED TO RE-ASSEMBLING THE PROGRAM.

THE FLOW CHART AND LISTING FOR THE "TRANSFER" ROUTINE ARE PRESENTED FOLLOWING THE "SEARCH" ROUTINE FLOW CHART.
START

COMMAND VALID?

YES

FETCH START ADDR AND DATA

SEARCH MEM FOR DATA PATTERN SPECIFIED

MATCH?

YES

PRINT MEM ADDR

NO

INCR MEM PNTR

FINISHED?

NO

RETURN TO COMMAND MODE

NO

PRINT ERROR MSG

YES

THE "SEARCH" ROUTINE FLOW CHART

- 40 -
THE MEMORY "TRANSFER" ROUTINE FLOW CHART

<table>
<thead>
<tr>
<th>MNEMONIC</th>
<th>COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRNSFR, LLI 342</td>
<td>SET PNTR TO ADDR INP</td>
</tr>
<tr>
<td>CAL OCTNM</td>
<td>FETCH 'FROM' ADDR</td>
</tr>
<tr>
<td>LLI 166</td>
<td>SET PNTR TO ADDR INP</td>
</tr>
<tr>
<td>LBE</td>
<td>SAVE INP BFR PNT</td>
</tr>
<tr>
<td>LEI 172</td>
<td>SAVE 'FROM' IN TEMP STRGE</td>
</tr>
<tr>
<td>LDH</td>
<td></td>
</tr>
<tr>
<td>SVSA, CAL SWAP</td>
<td>MOVE ADDR TO TEMP STRGE</td>
</tr>
<tr>
<td>LAI 172</td>
<td>IS XFR COMPLETE?</td>
</tr>
<tr>
<td>CPL</td>
<td></td>
</tr>
<tr>
<td>JFZ SVSA</td>
<td>NO, CONTINUE MOVE</td>
</tr>
<tr>
<td>MNEMONIC</td>
<td>COMMENTS</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------</td>
</tr>
<tr>
<td>INB</td>
<td>/RESTORE INP BFR PNTR</td>
</tr>
<tr>
<td>LLB</td>
<td>/INP 'TO' ADDR</td>
</tr>
<tr>
<td>CAL 0CTNM</td>
<td>/SET PNTR TO SAVE 'TO' ADDR</td>
</tr>
<tr>
<td>LLI 176</td>
<td>/SAVE LO ADDR</td>
</tr>
<tr>
<td>LMB</td>
<td>/SAVE PG ADDR</td>
</tr>
<tr>
<td>LMC</td>
<td>/SET PNTR TO TEMP STAGE</td>
</tr>
<tr>
<td>LLI 172</td>
<td>/SET TO MOVE 'FROM' BACK</td>
</tr>
<tr>
<td>LEI 166</td>
<td>/XFR 'FROM'</td>
</tr>
<tr>
<td>LDH</td>
<td>/XFR COMPLETE?</td>
</tr>
<tr>
<td>TFI, CAL SWAP</td>
<td>/NO, CONTINUE</td>
</tr>
<tr>
<td>LAI 176</td>
<td>/FETCH 'TO' PNTR</td>
</tr>
<tr>
<td>CPL</td>
<td>/FETCH 'FROM' PNTR</td>
</tr>
<tr>
<td>JFZ TFI</td>
<td>/SET 'FROM' PNTR</td>
</tr>
<tr>
<td>LEM</td>
<td>/SWAP MEM CONTENTS</td>
</tr>
<tr>
<td>INL</td>
<td>/DONE? YES, RET TO CMND MODE</td>
</tr>
<tr>
<td>LDM</td>
<td>/CONTINUE XFR</td>
</tr>
<tr>
<td>TFE, CAL SETUP</td>
<td>/FETCH BYTE TO XFR</td>
</tr>
<tr>
<td>CAL SWAP</td>
<td>/INC 'FROM' PNTR</td>
</tr>
<tr>
<td>CAL CKEND</td>
<td>/CHANGE PNTRS</td>
</tr>
<tr>
<td>JMP TFE2</td>
<td>/STORE BYTE IN NEW LOC</td>
</tr>
<tr>
<td>/SWAP, LAM</td>
<td>/CHANGE PNTRS AND RET</td>
</tr>
</tbody>
</table>

PUTTING IT ALL TOGETHER - THE ASSEMBLED MONITOR PROGRAM

AND AFTER ALL IS SAID AND DONE, HERE IT IS! THE MONITOR PROGRAM PRESENTED IN ITS FINAL ASSEMBLED FORM. THE ROUTINES DISCUSSED ARE NOW LISTED WITH THEIR ADDRESSES AND MACHINE CODE TO PROVIDE THE READER WITH A MONITOR PROGRAM THAT SIMPLY REQUIRES THE ADDITION OF THE I/O DRIVERS (DETAILED PREVIOUSLY) TO TURN ONE'S COMPUTER SYSTEM INTO A HIGHLY FUNCTIONAL "OPERATING SYSTEM!"

THE FIRST PART OF THE LISTING SHOWS THE LOCATIONS ON PAGE 00 WHICH ARE USED BY THE MONITOR FOR STORING POINTERS, COUNTERS, TEMPORARY DATA, THE COMMAND LOOK UP TABLE AND THE INPUT BUFFER. THE READER WILL NOTE THAT SEVEN OF THE EIGHT RESTART LOCATIONS ARE AVAILABLE FOR THE USER'S PROGRAMS.

THE OPERATING PORTION OF THE MONITOR PROGRAM HAS BEEN ORIGINATED ON PAGES 14 THROUGH THE FIRST HALF OF PAGE 17, WITH THE EXPECTED STARTING LOCATIONS OF THE USER PROVIDED I/O DRIVERS ON THE SECOND HALF OF PAGE 17. THE READER MAY DESIRE TO RE-ORIGIN THE OPERATING PORTION TO THE UPPER SECTION OF THE MEMORY AVAILABLE IN ONE'S SYSTEM.

THE START OF EXECUTION ADDRESS FOR THE MONITOR PROGRAM, AS LISTED, IS AT PAGE 14 LOCATION 000.
000 000 ORG 000 070
000 070 /
000 070 104 000 000 JMP 000 000 /JUMP INSTRUCTION FOR BRKPT
000 073 000 000 /BRKPT LOCATION - LOW ADDR
000 074 000 000 /BRKPT LOCATION - PG ADDR
000 075 000 000 /ORIG. BRKPT INSTRUCTION
000 076 /
000 076 /LOC. 076 THRU 127 AVAILABLE FOR USER
000 076 /
000 076 /MONITOR MESSAGE TABLE
000 076 /
000 076 ORG 000 130
000 130 215 215 /CAR. RET.
000 131 212 212 /LINE FEED
000 132 276 276 />
000 133 000 207 000
000 134 215 000 215 /CAR. RET.
000 135 212 212 /LINE FEED
000 136 000 212 000
000 137 /
000 137 /LOC. 137 THRU 147 AVAILABLE FOR USER
000 137 /
000 137 ORG 000 150
000 150 /
000 150 000 000 /DIGIT STORAGE
000 151 000 000 /FOR OCTAL NUMBER
000 152 000 000 /SUBROUTINE
000 153 000 000 /AVAILABLE
000 154 000 000 /AVAILABLE
000 155 /
000 155 /COMMAND AND GO TO JUMP INSTRUCTION
000 155 /
000 155 104 000 000 JMP 000 000 /CMND RTN FILL IN ADDR
000 160 /
000 160 000 000 /AVAILABLE
000 161 000 000 /AVAILABLE
000 162 000 000 /AVAILABLE
000 163 000 000 /AVAILABLE
000 164 000 000 /TEMP STORAGE
000 165 000 000 /TEMP STORAGE
000 166 000 000 /LOW ADDRESS - LOW PORTION
000 167 000 000 /LOW ADDRESS - PAGE PORTION
000 170 000 000 /HIGH ADDRESS - LOW PORTION
000 171 000 000 /HIGH ADDRESS - PAGE PORTION
000 172 000 000 /TEMP STORAGE
000 173 000 000 /TEMP STORAGE
000 174 000 000 /TEMP STORAGE
000 175 000 000 /TEMP STORAGE
000 176 000 000 /TEMP STORAGE
000 177 000 000 /TEMP STORAGE
000 200 /
000 200 000 000 /VIRTUAL CPU REG "A"
000 201 000 000 /VIRTUAL CPU REG "B"
000 202 000 000 /VIRTUAL CPU REG "C"
000 203 000 000 /VIRTUAL CPU REG "D"
000 204 000 000 /VIRTUAL CPU REG "E"
000 205 000 000 /VIRTUAL CPU REG "H"
000 206 000 000 /VIRTUAL CPU REG "L"
000 207 000 000 /FLAG STATUS BYTE

- 43 -
000 210  /COMMAND LOOK UP TABLE
  000 210
  000 210 315
  000 211 150
  000 212 015
  000 213 304
  000 214 275
  000 215 015
  000 216 327
  000 217 343
  000 220 015
  000 221 322
  000 222 371
  000 223 015
  000 224 302
  000 225 377
  000 226 015
  000 227 307
  000 230 220
  000 231 016
  000 232 330
  000 233 257
  000 234 016
  000 235 306
  000 236 005
  000 237 017
  000 240 323
  000 241 022
  000 242 017
  000 243 324
  000 244 061
  000 245 017
  000 246
  000 246  /LOG. 246 THRU 337 AVAILABLE FOR
  000 246  /USER OR TO EXPAND COMMAND TABLE
  000 246
  000 246  /LOG. 340 THRU 377 - INPUT BUFFER
  000 246
  000 246  /PAGES 01 THRU 13 AVAILABLE
  000 246  /FOR USER'S PROGRAMS
  000 246
  000 246
  000 246  /COMMAND INPUT ROUTINE
  000 246
  000 246  ORG 014 000
  014 000 056 000  INCMD, LHI 000
  014 002 066 130  LLI 130
  014 004 106 155 014  CAL MSG
  014 007 106 066 014  CAL CDIN
  014 012 066 340  LLI 340
  014 014 307  LAM
  014 015 036 012  LD! 012
  014 017 066 210  LLI 210
  014 021 277  CKCMD, CPM
  014 022 150 047 014  JTZ FOUND
  014 025 060  INL
  014 026 060  INL
  014 027 060  INL
  014 030 031  DCD
  014 031 110 021 014  JFZ CKCMD

    /SET PNTR TO HEADING MSG
    /PRINT C/R, L/F, >
    /INPUT COMMAND FM KYBD
    /FETCH COMMAND CHAR
    /SET CMND NMBR CNTR
    /SET CMND TABLE PNTR
    /IS CMND CHAR FOUND IN TBL?
    /YES, PROCESS COMMAND
    /NO, ADVANCE CMND TBL PNTR
    /IS LAST CMND CHECKED?
    /NO, CHECK NEXT
014 034  106 151 014 ERR, CAL HDLN
014 037  006 311  LAI 311
014 041  106 300 017 CAL PRINT
014 044  104 000 014 JMP INCMD
014 047  
014 047  060  FOUND, INL
014 050  337  LDM
014 051  060  INL
014 052  327  LCM
014 053  066 156  LLI 156
014 055  056 000  LHI 000
014 057  373  LMD
014 060  060  INL
014 061  372  LMC
014 062  364  LLE
014 063  104 155 000  JMP 155 000
014 066  
014 066  066 340  CDIN, LLI 340
014 070  076 240  SPI, LMI 240
014 072  060  INL
014 073  110 070 014 JFZ SPI
014 076  066 340  LLI 340
014 100  106 200 017  IN2, CAL RCV
014 103  074 024  CPI 204
014 105  150 000 014  JTZ INCMD
014 110  074 215  CPI 215
014 112  053  RTZ
014 113  074 214  CPI 214
014 115  053  RTZ
014 116  074 377  CPI 377
014 120  150 135 014  JTZ BDCR
014 123  060  INL
014 124  061  DCL
014 125  150 100 014  JTZ IN2
014 130  370  LMA
014 131  060  INL
014 132  104 100 014  JMP IN2
014 135  
014 135  006 340  BDCR, LAI 340
014 137  276  CPL
014 140  150 100 014  JTZ IN2
014 143  061  DCL
014 144  076 240  LMI 240
014 146  104 100 014  JMP IN2
014 151  
014 151  066 134  HDLN, LLI 134
014 153  056 000  LHI 000
014 155  
014 155  307  MSG, LAM
014 156  240  NDA
014 157  053  RTZ
014 160  106 300 017 CAL PRINT
014 163  106 171 014  CAL INMEM
014 166  104 155 014  JMP MSG
014 171  
014 171  060  INMEM, INL
014 172  013  RFZ
014 173  050  INH
014 174  007  RET
014 175  /
014 175 346  / OCTNM, LEL  / SAVE INP BFR PNTR
014 176 106 250 014 371  / CONVERT 1ST OCTAL PAIR  / CONVERT 1ST OCTAL PAIR
014 201 066 166  LLI 166  / SET PNTR TO LO ADDR STRAGE  / CONVERT 1ST OCTAL PAIR
014 203 371  LMB  / SAVE LO HALF OF LO ADDR
014 204 060  INL  / SAVE PG HALF OF LO ADDR  / RESTORE INP BFR PNTR
014 205 372  LMC  / FETCH NXT CHAR  / RESTORE INP BFR PNTR
014 206 364  LLE  / CHAR = COMMA?  / RESTORE INP BFR PNTR
014 207 307  LAM  / NO, ONLY ONE ENTRY  / RESTORE INP BFR PNTR
014 210 074 254  CP1 254  / CONVERT 2ND OCTAL PAIR  / RESTORE INP BFR PNTR
014 212 110 222 014 610  JFF SGL  / SET PNTR TO HI ADDR STRAGE  / CONVERT 2ND OCTAL PAIR
014 215 060  INL  / SAVE LO HALF OF HI ADDR
014 216 346  LEL  / SAVE PG HALF OF HI ADDR
014 217 106 250 014 371  / IS HI ADDR < LO ADDR?
014 222 066 170  SGL, LLI 170  / RESTORE INP BFR PNTR
014 224 371  LMB  / YES, PRINT ERROR
014 225 060  INL  / IF PG HALF NOT =, RET
014 226 372  LMC  / ELSE, CHECK LO HALF
014 227 302  LAC  / IS HI ADDR < LO ADDR?
014 230 066 167  LLI 167  / YES, PRINT ERROR MSG
014 232 277  CPM  / NO, RET TO CALLING PGM
014 233 140 034 014 371  JTC ERR  / DECODE 1ST OCTAL NUMBER
014 236 013  RFZ  / SAVE OCTAL NUMBER
014 237 060  INL  / INCR INP BFR PNTR
014 240 307  LAM  / FALL THRU TO DECODE 2ND NMBR
014 241 066 166  LLI 166  / SET PNTR TO DIGIT STRAGE TBL
014 243 277  CPM  / CLEAR TBL BY STORING 000.
014 244 140 034 014 371  JTC ERR  / RESET INP BFR PNTR
014 247 007  RET  / CHECK FOR VALID NUMBER
014 250  / IF NOT, CHECK CHAR CNT = 0
014 253 321  LCB  / FETCH CHAR
014 254 040  INE  / SAVE INP BFR PNTR
014 255  / MASK OFF 260
014 255  / STORE OCTAL NUMBER IN
014 255 066 150  DCDNM, LLI 150  / TABLE AT LOC 150 PG 00
014 257 375  LMI  / AND SHIFT OTHER NUMBERS
014 260 060  INL  / UP THRU THE TABLE
014 261 375  LMI  / RESTORE AND INCR INP BFR PNTR
014 262 060  INL  / FETCH NXT NUMBER
014 263 375  LMI  / LLE
014 264 364  / LOOP, CAL FNUM
014 265 106 332 014 371  / JTS CKLNH
014 270 160 315 014 371  / LAM
014 273 307  / SAVE INP BFR PNTR
014 274 336  / MASK OFF 260
014 275 044 007  NDI 007  / STORE OCTAL NUMBER IN
014 277 066 150  LLI 150  / TABLE AT LOC 150 PG 00
014 301 317  LBM  / AND SHIFT OTHER NUMBERS
014 302 370  LMA  / UP THRU THE TABLE
014 303 060  INL  / RESTORE AND INCR INP BFR PNTR
014 304 307  LAM  / FETCH NXT NUMBER
014 305 371  LMB  / LLA
014 306 060  INL  / LLA
014 307 370  LMA  / LLA
014 310 363  LLD  / LLA
014 311 060  INL  / LLA
014 312 104 265 014 371  JMP LOOP
014 315  /
014 315  306  CKLH., LAL
014 316  274  CPE
014 317  150 034 014  JULZ ERR
014 322  346  LEL
014 323  106 355 014  CAL OCT
014 326  120 034 014  JFS ERR
014 331   007  RET
014 332   /
014 333   307  FNUN, LAM
014 333  074 260  CPI 260
014 335   063  RTS
014 336   024 270  SUI 270
014 340   004 200  ADI 200
014 342   007  RET
014 343   /
014 343  004 001  INCR, ADI 001
014 344  370  LMA
014 346   003  RFC
014 347   060  INL
014 350   307  LAM
014 351  004 001  ADI 001
014 353   370  LMA
014 354   007  RET
014 355   /
014 355  066 152  OCT, LLI 152
014 357   307  LAM
014 360   074 004  CPI 004
014 362   023  RFS
014 363   044 003  NDI 003
014 365   012  RRC
014 366   012  RRC
014 367   310  LBA
014 370   061  DCL
014 371   307  LAM
014 372   002  RLC
014 373   002  RLC
014 374   002  RLC
014 375  201  ADB
014 376   061  DCL
014 377   207  LMC
015 000   310  LBA
015 001  006 200  LAI 200
015 003  240  NDA
015 004   007  RET
015 005   /
015 005  325  SWITCH, LCH
015 006   353  LHD
015 007   332  LDC
015 010   326  LCL
015 011   364  LLE
015 012   342  LEC
015 013   007  RET
015 014   /
015 014  360  OCTOUT, LLA
015 015   002  RLC
015 016   002  RLC
015 017   044 003  NDI 003
015 021  064 260  ORI 260
015 023 106 300 017  CAL PRINT

/IS CHAR CNT = 0?
/YES, PRINT ERROR MSG
/NO, SAVE INP BFR PNTR
/FETCH FINAL OCTAL NUMBER
/IF INVALID, PRINT ERR MSG
/ELSE, RET TO CALLING PGM

/IS CHAR A VALID NUMBER?
/NO, RET WITH S FLAG SET
/CHECK UPPER LIMIT BY
/SETTING S FLAG TO PROPER STATE AND RETURN

/INCR CONTENTS OF MEM LOC
/RESTORE MEM CONTENTS
/IF NO CARRY, RET
/ELSE, FETCH NXT LOC

/INCR MEM CONTENTS
/RESTORE MEM CONTENTS
/RET TO CALLING PGM

/SET PNTR TO 3RD DIGIT
/IS 3RD DIGIT > 3?
/YES, RET WITH S FLAG RESET
/CLEAR CARRY
/POSITION DIGIT

/SAVE IN REG B
/DECN PNTR
/FETCH NEXT DIGIT
/POSITION DIGIT

/ADD TO REG B
/DECN PNTR

/SAVE FINAL NUMBER
/SET S FLAG TO INDICATE THAT THE NUMBER IS VALID
/RET TO CALLING PGM

/SWITCH THE PNTR IN
/REG'S H AND L WITH
/THE PNTR IN REG'S D AND E

/RET TO CALLING PGM

/SAVE OCTAL NUMBER TO PRINT
/POSITION HUNDRED'S DIGIT

/MASK OFF OTHER BITS
/FORM ASCII CODE
/PRINT DIGIT
015 026 306 LAL
015 027 012 RRC
015 030 012 RRC
015 031 012 RRC
015 032 044 007 NDI 007
015 034 064 260 ORI 260
015 036 106 300 017 106 CAL PRINT
015 041 306 LAL
015 042 044 007 NDI 007
015 044 064 260 ORI 260
015 046 104 300 017 106 JMP PRINT
015 051 
015 051 006 272 COLON, LAI 272
015 053 104 300 017 106 JMP PRINT
015 056 
015 056 066 167 PRT166, LLI 167
015 060 056 000 LHI 000
015 062 307 LAM
015 063 044 077 NDI 077
015 065 106 014 015 106 CAL OCTOUT
015 070 106 101 015 106 CAL SPAC
015 073 066 166 LLI 166
015 075 307 LAM
015 076 106 014 015 106 CAL OCTOUT
015 101 
015 101 006 240 106 240 SPAC, LAI 240
015 103 104 300 017 106 JMP PRINT
015 106 
015 106 056 000 SETUP, LHI 000
015 110 066 166 LLI 166
015 112 327 LCM
015 113 060 INL
015 114 357 LHM
015 115 362 LLC
015 116 007 RET
015 117 
015 117 056 000 CKEND, LHI 000
015 121 066 171 LLI 171
015 123 307 LAM
015 124 066 167 LLI 167
015 126 277 CPM
015 127 110 142 015 127 JFZ CONT
015 132 060 INL
015 133 307 LAM
015 134 066 166 LLI 166
015 136 277 CPM
015 137 150 000 014 137 JTZ INCMD
015 142 066 166 CONT, LLI 166
015 144 307 LAM
015 145 104 343 014 137 JMP INCR
015 150 
015 150 066 342 014 137 MODIFY, LLI 342
015 152 106 175 014 155 CAL OCTNM
015 155 106 101 015 155 CAL SPAC
015 160 106 266 015 160 MODI, CAL MEMPR
015 163 106 051 015 163 CAL COLON
015 166 106 241 015 166 CAL INSPCL
015 171 006 340 LAI 340
015 173 276 CPL
NO, SET UP NXT LOC
YES, SAVE INP PNTR
CONVERT TO OCTAL NUMBER
SAVE OCTAL NUMBER
SET PNTR TO MEM ADDR STRAGE
FETCH MEM PNTR

SET PNTR TO MEM LOC
LOAD MEM WITH NEW VALUE
SET PNTR TO PG 00
SET PNTR TO MEM ADDR STRAGE
FETCH LO HALF
INCR MEM ADDR
PRINT NXT ADDR TO MODIFY

PRINT C/R, L/F
PRINT ADDR TO MODIFY AND RET

SET PNTR TO S.A. OF INP BFR
INP CHAR
STORE CHAR IN INP BFR
CHAR = SPACE?
YES, RET TO CALLING PGM
NO, CHAR = C/R?
YES, RET TO COMMAND MODE
NO, INCR INP BFR PNTR
INP BFR FULL? YES, ERROR
NO, INP NXT CHAR

SET PNTR TO MEM LOC
FETCH CURRENT MEM CONTENTS
PRINT CONTENTS AND RET

SET PNTR TO INP BFR
FETCH MEM DUMP LIMITS
PRINT C/R, L/F
PRINT ADDR OF 1ST LOC
PRINT SPACE
SET PNTR TO TEMP STRAGE
SAVE LOC PER LINE CNTR
PRINT MEM CONTENTS
CHECK FOR LAST LOC PRTD
PRINT SPACE
SET PNTR TO L/L CNTR
FETCH CNTR
DECR CNTR
SAVE CNTR. CNTR = 0?
YES, START NEW LINE
NO, PRINT MORE CONTENTS

SET PNTR TO INP BFR
FETCH START AND END ADDR
SET REG'S H AND L WITH
THE START ADDR AND
REG'S D AND E WITH
THE END ADDR OF THE
BLOCK OF MEM TO BE
016 112 /          BRK1, LHI 000
016 114 066 200 LLI 200
016 116 370 LMA
016 117 066 201 LLI 201
016 121 371 LMB
016 122 066 202 LLI 202
016 124 372 BRKCOM, LMC
016 125 305 FLAGT, LAH
016 126 310 LBA
016 127 320 LGA
016 130 100 135 016 JFC N0C
016 133 006 001 LAI 001
016 135 110 142 016 N0C, JFZ NOZ
016 140 016 010 LBI 010
016 142 130 147 016 NOZ, JFP NOP
016 145 026 100 LCI 100
016 147 120 154 016 NOP, JFS N0S
016 152 004 200 ADI 200
016 154 201 NOS, ADB
016 155 202 ADC
016 156 066 207 LLI 207
016 160 370 LMA
016 161 066 073 LLI 073
016 163 347 LEM
016 164 060 INL
016 165 337 LDM
016 166 060 INL
016 167 307 LAM
016 170 364 LLE
016 171 353 LMD
016 172 370 LMA
016 173 104 000 014 JMP INCMD
016 176 /          JMP INCMD
016 176 315 BRK2, LBH
016 177 326 LCL
016 200 056 000 LHI 000
016 202 066 203 LLI 203
016 204 373 LMD
016 205 066 204 LLI 204
016 207 374 LME
016 210 066 205 LLI 205
016 212 371 LMB
016 213 066 206 LLI 206
016 215 104 124 016 JMP BRKCOM
016 220 /          JMP BRKCOM
016 220 106 050 016 GOTO, CAL ANLYZ
016 223 150 251 016 JTZ G01
016 226 074 262 CPI 262
016 230 110 034 014 JFZ ERR
016 233 /          JFZ ERR
016 233 066 203 G02, LLI 203
016 235 337 LDM
016 236 060 INL
016 237 347 LEM
016 240 060 G0COM, INL
016 241 317 LBM
016 242 060 INL
016 243 327 LCM
SET UP CPU REGS A, B, C

SET IMP BFR PNTR
FETCH REG LETTER
IS REG VALID?
NO, PRINT ERROR
YES, IS REG A THRU E?
NO, TRY H, L OR F
SET UP REG TBL PNTR
SAVE TBL PNTR IN TEMP STRAGE

SET PNTR TO REG TBL LOC
PRINT SPACE
FETCH CURRENT REG VALUE
PRINT CURRENT REG VALUE
PRINT COLON
INP MODIFICATION
SET INP BFR PNTR

WAS MOD ENTERED?
NO, RET TO COMMAND MODE
YES, DECODE OCTAL NUMBER
SET PNTR TO TEMP STRAGE
FETCH REG TBL PNTR
STORE NEW REG VALUE
RET TO COMMAND MODE

IS REG = H?
NO, TRY L OR F
YES, SET REG TBL PNTR
INP MOD TO REG VALUE

IS REG = L?
NO, TRY F
YES, SET REG TBL PNTR
INP MOD TO REG VALUE

IS REG = F, FOR FLAGS?
NO, PRINT ERROR
PRINT SPACE
SET REG TBL PNTR
FETCH FLAG WORD
PRINT FLAG WORD
RET TO COMMAND MODE

INP ADDR AND DATA FM BFR
SET UP MEM PNTR
FILL MEM LOC WITH DATA
DONE? YES, RET TO CMWD MODE
NO, CONTINUE WITH FILL
017 022 106 050 01/76 SEARCH, CAL ADRDTA
017 025 066 165 LLI 165
017 027 371 LMB
017 030 066 165 SHI, LLI 165
017 032 307 sham
017 033 106 106 01/76 CAL SETUP
017 036 277 CPX
017 037 152 233 01/76 UTZ MCNT
017 042 106 117 01/76 CAL CKEND
017 045 104 030 01/76 JMP SHI
017 050 106 342 01/76 ADRDTA, LLI 342
017 052 106 175 01/4 CAL OCTNM
017 055 040 INE
017 056 104 255 01/4 JMP DCNM
017 061 106 342 TRNSFR, LLI 342
017 063 106 175 01/4 CAL OCTNM
017 066 066 166 LLI 166
017 070 314 LBE
017 071 046 172 LEI 172
017 073 335 LDH
017 074 106 154 01/76 SUS, CAL SWAP
017 077 066 172 LAI 172
017 101 276 CPL
017 102 110 074 01/76 JFZ SUS
017 105 010 INB
017 106 361 LLB
017 107 106 175 01/4 CAL OCTNM
017 112 066 176 LLI 176
017 114 371 LMB
017 115 060 INL
017 116 372 LMC
017 117 066 172 LLI 172
017 121 046 166 LEI 166
017 123 335 LDH
017 124 106 154 01/76 TFI, CAL SWAP
017 127 006 176 LAI 176
017 131 276 CPL
017 132 110 124 01/10 JFZ TFI
017 135 347 LEM
017 136 060 INL
017 137 337 LDM
017 140 106 106 01/5 TFE, CAL SETUP
017 143 106 154 01/76 CAL SWAP
017 146 106 117 01/5 CAL CKEND
017 151 104 140 01/76 JMP TFE2
017 154 307 307
017 155 106 171 01/4 CAL INMEM
017 160 106 005 01/5 CAL SWITCH
017 163 370 LMA
017 164 106 171 01/4 CAL INMEM
017 167 104 005 01/5 JMP SWITCH
017 172
OPERATING THE MONITOR PROGRAM

As a review of the Monitor Program functions and, also, to serve as an operator's guide, the operation of each of the Monitor commands will now be described.

THE "MODIFY" COMMAND

The "modify" command is initiated by typing in the "M" command followed by the address to be modified, in the following format:

M HHH LLL (CTRL/L)

Where "HHH" is the page address and "LLL" is the low address (in octal) of the RAM memory address where one desires to begin examining and/or modifying the contents of memory locations. The operator should note that a space should be inserted between the "M" and the page address as well as between the page address and the low address when entering the command string.

When the operator depresses the "CTRL/L" combination to execute the "M" command, the following will occur. The output device will display the following information:

HHH LLL XXX:

The "XXX" is the current contents of the memory location specified. The program will then wait for the operator to select either a "modify" option or take the option of not modifying the current location being displayed but continue to display the next location, or terminate the "M" sequence. To elect to modify the contents of the memory location being displayed, the operator simply types in the desired octal contents immediately following the ": " sign and then depresses the "space" bar. The number entered will become the new value for the memory location and the program will proceed to display the address and contents of the next sequential memory location.

If the operator does not wish to modify the contents of a location, but does desire to examine the contents of the next memory location, then it is only necessary to depress the "space" bar. The program will proceed to display the memory address and contents of the next memory location.
IF THE OPERATOR DESIRES TO TERMINATE THE "MODIFY" PROCESS, THEN THE "CARRIAGE RETURN" IS ENTERED AND THE PROGRAM WILL RETURN TO THE MONITOR COMMAND MODE AND DISPLAY THE ">" MONITOR "READY" CHARACTER.

IT IS IMPORTANT TO NOTE THAT WHEN ELECTING TO MODIFY A MEMORY LOCATION, THE "SPACE" CHARACTER MUST BE ENTERED AFTER ENTERING THE OCTAL NUMBER THAT IS TO BE THE NEW VALUE IN THE MEMORY LOCATION! THIS WILL CAUSE THE NEW VALUE TO BE PLACED IN THE MEMORY LOCATION AND AUTOMATICALLY CAUSE THE NEXT LOCATION IN MEMORY TO BE DISPLAYED. HITTING THE "C/R" IMMEDIATELY AFTER ENTERING A NEW VALUE FOR A MEMORY LOCATION WILL CAUSE THE PROGRAM TO RETURN TO THE MONITOR AND WILL NOT RESULT IN THE VALUE BEING PLACED IN MEMORY! THIS FORMAT ALLOWS THE OPERATOR TO ELECT NOT TO CHANGE A MEMORY LOCATION EVEN AFTER HAVING TYPED IN A VALUE. IF, HOWEVER, THE RULE IS NOT REMEMBERED, THE OPERATOR MIGHT INADVERTENTLY FAIL TO INSERT THE DESIRED CHANGES.

CORRECTING ERRORS WHEN IN THE MONITOR COMMAND MODE

IF THE OPERATOR MAKES A TYPING MISTAKE WHILE ENTERING A COMMAND SEQUENCE TO THE MONITOR, THE CURRENT COMMAND CAN BE ERASED BY ENTERING THE CHARACTER "CONTROL/D". THIS WILL CAUSE THE PROGRAM TO GO BACK TO THE INITIAL "READY" CONDITION ("->" DISPLAYED) TO WAIT A NEW ENTRY. IF ONLY ONE OR TWO CHARACTERS ARE ENTERED IN ERROR, THE "RUBOUT" CHARACTER MAY BE ENTERED TO DELETE ONE CHARACTER TO THE LEFT FOR EACH RUBOUT ENTERED.

SHOULD THE OPERATOR INADVERTENTLY ENTER AN INVALID COMMAND OR COMMAND SEQUENCE, THE PROGRAM WILL CAUSE THE LETTER "1" (ILLEGAL COMMAND) TO BE PRINTED.

THE MEMORY "DUMP" COMMAND

THE MONITOR MEMORY "DUMP" COMMAND IS INITIATED BY TYPING IN THE "D" COMMAND IN THE FOLLOWING FORMAT:

D HHH LLL, MMM NNN (CTRL/L)

WHERE "HHH" AND "LLL" SIGNIFIES THE STARTING ADDRESS (OCTAL) AND "MMM" AND "NNN" INDICATE THE ENDING ADDRESS OF THE BLOCK OF MEMORY THAT ONE DESIRES TO HAVE DISPLAYED. WHEN THE "CTRL/L" (OR "C/R" MAY BE USED) IS ENTERED, THE PROGRAM WILL PROCEED TO DISPLAY THE CONTENTS OF THE MEMORY LOCATIONS SPECIFIED. THE OUTPUT FORMAT WILL BE THE FOLLOWING:

HHH LLL XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX
HHH+020 XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX
HHH+040 XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX XXX

EACH LINE PRINTED STARTS WITH THE ADDRESS OF THE FIRST LOCATION DISPLAYED FOLLOWED BY THE CONTENTS OF THE NEXT 20 (OCTAL) LOCATIONS IN MEMORY. THE PROCESS CONTINUES UNTIL THE LAST LOCATION SPECIFIED IN THE COMMAND HAS BEEN PRINTED.

- 55 -
THE "WRITE" COMMAND

THE "WRITE" COMMAND IS INITIATED BY THE OPERATOR ENTERING THE "W" COMMAND IN THE FOLLOWING FORMAT:

W HHH LLL, MMM NNN (CTRL/L)

WHERE "HHH" AND "LLL" INDICATE THE START ADDRESS AND "MMM" AND "NNN" INDICATE THE ENDING ADDRESS OF THE BLOCK TO BE WRITTEN TO THE BULK STORAGE DEVICE. NATURALLY, THE OPERATOR MUST MAKE WHATEVER PREPARATIONS ARE NECESSARY FOR THE BULK STORAGE DEVICE TO RECEIVE THE DATA BEFORE THE COMMAND IS ISSUED (BY ENTERING THE "CTRL/L" (OR "C/R")). AT THE CONCLUSION OF THE DATA TRANSFER, IT IS ASSUMED THAT THE BULK STORAGE OUTPUT ROUTINE WILL RETURN TO THE MONITOR COMMAND MODE.

THE "READ" COMMAND

THE "READ" COMMAND IS INITIATED BY THE OPERATOR ENTERING THE "R" COMMAND IN THE FOLLOWING FORMAT:

R (CTRL/L)

THE ISSUANCE OF THIS COMMAND CALLS THE BULK STORAGE INPUT ROUTINE TO BEGIN READING IN THE DATA FROM THE BULK STORAGE DEVICE. ADDRESSING INFORMATION IS ASSUMED TO BE EITHER SET UP BY THE BULK STORAGE INPUT ROUTINE OR RECEIVED FROM THE DATA AS IT IS READ IN. THE OPERATOR MUST SET UP THE BULK STORAGE DEVICE PRIOR TO ENTERING THIS COMMAND OR AS IS REQUIRED BY THE BULK INPUT ROUTINE.

THE "BREAKPOINT" COMMAND

THE MONITOR "BREAKPOINT" COMMANDS ARE INITIATED BY TYPING IN ONE OF THE FOLLOWING COMMANDS:

B1 HHH LLL (CTRL/L)
OR
B2 HHH LLL (CTRL/L)

WHERE "HHH LLL" DESIGNATES THE MEMORY ADDRESS AT WHICH THE BREAKPOINT IS TO BE INSERTED.

NOTICE

IN CASES WHERE A BREAKPOINT IS TO BE INSERTED IN A MULTI-BYTE INSTRUCTION, SUCH AS "IMMEDIATE," "JUMP" OR "CALL" INSTRUCTIONS, THE ADDRESS INDICATED MUST BE THE ADDRESS OF THE FIRST BYTE IN THE INSTRUCTION!

PROGRAM BEING TESTED TO ASCERTAIN WHETHER PROGRAM OPERATION IS ACTUALLY
REACHING A CERTAIN POINT, OR TO VALIDATE THE STATUS OF THE SELECTED CPU
REGISTERS AT GIVEN POINTS WITHIN A PROGRAM UNDER DEVELOPMENT. WHEN THE
PROGRAM BEING TESTED REACHES THE ADDRESS AT WHICH A BREAKPOINT HAS BEEN
INSERTED, CONTROL WILL REVERT TO THE MONITOR AND THE ORIGINAL IN-
STRUCTION IN THE PROGRAM WILL BE RESTORED AT THE BREAKPOINT ADDRESS.

CAUTION

WHEN UTILIZING THE BREAKPOINT FACILITY THERE ARE SEVERAL CON-
SIDERATIONS THAT THE OPERATOR MUST KEEP IN MIND:

1. THE PROGRAM BEING TESTED MAY NEVER REACH THE SELECTED
BREAKPOINT ADDRESS IN WHICH CASE THE OPERATOR MAY HAVE TO MAN-
UALLY STOP THE PROGRAM AND RESTART THE MONITOR PROGRAM. IF
THIS OCCURS, THE OPERATOR SHOULD USE THE "MODIFY" FUNCTION TO
REMOVE THE "BREAKPOINT" INSTRUCTION FROM THE LOCATION THAT IT
WAS INSERTED (WHICH WILL APPEAR AS AN "075" CODE) AND RESTORE
THE ORIGINAL INSTRUCTION CODE TO THE PROGRAM UNDER TEST. THE
OPERATOR WOULD MOST LIKELY THEN CONTINUE TO "DEBUG" THE PRO-
GRAM BY SELECTING A BREAKPOINT AT SOME OTHER LOCATION.

2. ONLY ONE BREAKPOINT SHOULD BE ESTABLISHED AT ONE TIME.
ATTEMPTING TO ESTABLISH MORE THAN ONE BREAKPOINT WILL RESULT
IN THE FIRST BREAKPOINT ENCONTRRED BEING RESTORED WITH THE IN-
STRUCTION CODE CONTAINED IN THE ORIGINAL PROGRAM AT THE LAST
POINT AT WHICH A BREAKPOINT WAS ESTABLISHED. THIS MIGHT NOT BE
APPROPRIATE.

3. A TYPE "1" BREAKPOINT SHOULD NOT BE CHANGED TO A TYPE "2"
BREAKPOINT (OR VICE-VERSA) UNTIL THE BREAKPOINT HAS ACTUALLY
BEEN ENCOUNTERED. ATTEMPTING TO DO SO WILL RESULT IN AN "075"
CODE BEING INCORRECTLY RESTORED TO THE ORIGINAL BREAKPOINT.

IT SHOULD BE APPARENT THAT IF ONE DESIRES TO EXAMINE ALL THE CPU
REGISTERS AT A GIVEN POINT IN A PROGRAM'S OPERATION, ONE WILL NEED TO
OPERATE THE PROGRAM TWICE - ONCE WITH A "B1 BREAKPOINT ESTABLISHED, AND
ONCE WITH A "B2" BREAKPOINT ESTABLISHED AT THE SAME ADDRESS.

SINCE THE "VIRTUAL" CPU REGISTERS ARE ONLY UPDATED WHEN A BREAK-
POINT IS REACHED (OR WHEN THE OPERATOR SPECIFICALLY SETS THEM UP) IT IS
POSSIBLE TO REVIEW THE STATUS OF THE TWO GROUPS OF CPU REGISTERS AT SEV-
ERAL DIFFERENT POINTS IN A PROGRAM. FOR INSTANCE, ONE COULD SET UP A
"B1" TYPE BREAKPOINT AT LOCATION "A," HAVE THE BREAKPOINT ENCOUNTERED
AND THE ASSOCIATED "A," "B" AND "C" CPU REGISTERS SAVED IN THE VIRTUAL
LOCATIONS, THEN INSERT A TYPE "B2" BREAKPOINT AT LOCATION "B," HAVE IT
ENCOUNTERED, AND THEN REVIEW THE STATUS OF THE CPU REGISTERS USING THE
"X" TYPE COMMANDS. ONE COULD CONTINUE, SAY, TO INSERT AND ENCOUNTER
MORE TYPE "B2" BREAKPOINTS WHILE STILL SAVING THE ORIGINAL "A," "B" AND
"C" VALUES FOR REVIEW. (PARTICULARLY VALUABLE FOR THOSE THAT HAVE SHORT
MEmORIES WHEN WORKING ON DEBUGGING A COMPLEX PROGRAM.)

THE "GO TO" COMMAND

THE "GO TO" COMMANDS ARE INITIATED BY TYPING IN ONE OF THE FOL-
LOWING COMMANDS:
WHERE "HHH LLL" REPRESENTS THE MEMORY ADDRESS AT WHICH PROGRAM OPERATION IS TO COMMENCE WITH THE "A", "B" AND "C" REGISTERS FOR "G1" OR THE "D", "E", "H" AND "L" REGISTERS FOR "G2" INITIALIZED TO THE VALUES RESIDING IN THE VIRTUAL CPU REGISTER STORAGE LOCATIONS. IN MANY CASES, WHERE THE OPERATOR DOES NOT CARE WHAT THE STATUS OF THE CPU REGISTERS ARE WHEN PROGRAM OPERATION BEGINS, THE SELECTION OF THE "G1" OR "G2" TYPE "GO TO" COMMAND IS PURELY ARBITRARY. HOWEVER, WHEN DESIRED, THE OPERATOR MAY SET UP EITHER GROUP OF CPU REGISTERS TO CONTAIN SPECIFIC VALUES (USING THE "X" COMMAND) PRIOR TO EXECUTING THE "GO TO" COMMAND. THOSE VALUES WILL BE PLACED IN THE CPU REGISTERS WHEN THE "GO TO" COMMAND IS EXECUTED AND THE PROGRAM WILL THEN JUMP TO COMMENCE PROGRAMMED OPERATION AT THE ADDRESS SPECIFIED IN THE "GO TO" COMMAND. (NOTE THAT SINCE A BREAKPOINT IS ENCOUNTERED AFTER A "GO TO" COMMAND HAS BEEN EXECUTED, SETTING UP THE DESIRED VALUES IN CPU REGISTERS FOR A "GO TO" COMMAND WILL NOT EFFECT THE BREAKPOINT PROCESS OF "SAVING" THE CONTENTS OF A GROUP OF CPU REGISTERS WHEN A BREAKPOINT IS ENCOUNTERED.)

THE "EXAMINE REGISTER" COMMAND

THE "EXAMINE REGISTER" COMMANDS ARE INITIATED BY TYPING IN ONE OF THE FOLLOWING COMMANDS:

XA (CTRL/L)
XB (CTRL/L)
XC (CTRL/L)
XD (CTRL/L)
XE (CTRL/L)
XH (CTRL/L)
XL (CTRL/L)
XF (CTRL/L)

WHERE THE LETTER FOLLOWING THE "X" INDICATES THE "VIRTUAL" CPU REGISTER TO BE DISPLAYED. THE "CTRL/L" MUST BE USED IN THIS COMMAND AS THE TERMINATING CHARACTER TO MAINTAIN THE DISPLAY DEVICE AT THE POSITION FOLLOWING THE "XR" COMMAND. THE CONTENTS OF THE SPECIFIED REGISTER WILL BE DISPLAYED IN THE FOLLOWING FORMAT:

XR XXX:


IF IT IS DESIRED TO MODIFY THE CONTENTS OF A VIRTUAL REGISTER, THE OPERATOR TYPES IN THE DESIRED OCTAL VALUE AND DEPRESSES THE SPACE BAR.

IF THE OPERATOR SHOULD TYPE IN A NEW OCTAL VALUE AND THEN DECIDE THAT IT IS NOT DESIRABLE TO CHANGE THE ORIGINAL VALUE, THE OPERATOR MAY STRIKE THE "C/R" KEY TO RETURN TO THE COMMAND MODE, IN WHICH CASE THE ORIGINAL VALUE WILL REMAIN UNCHANGED.

THE "XF" COMMAND CAUSES THE STATUS OF THE CPU FLAGS (AS THEY WERE
WHICH THE LAST BREAKPOINT WAS ENCOUNTERED) TO BE DISPLAYED ACCORDING TO THE FOLLOWING ARRANGEMENT.

B7 B6 B5 B4 B3 B2 B1 B0

THE FOUR FLAGS CONNECTED WITH THE CPU HAVE BEEN ASSIGNED TO THE FOLLOWING POSITIONS IN THE EIGHT BIT GROUP.

B7 = SIGN FLAG
B6 = PARITY FLAG
B3 = ZERO FLAG
B0 = CARRY FLAG


THE "FILL" COMMAND

THE "FILL" COMMAND IS INITIATED BY TYPING IN THE "F" COMMAND IN THE FOLLOWING FORMAT:

F HHH LLL, MMM NNN, DDD (CTRL/L)

WHERE "HHH LLL" IS THE START ADDRESS AND "MMM NNN" IS THE END ADDRESS OF THE SECTION OF MEMORY THAT IS TO BE FILLED WITH THE DATA BYTE "DDD." WHEN THE CTRL/L (OR C/R) IS ENTERED, THE PROGRAM WILL PROCEED TO LOAD THE MEMORY LOCATIONS SPECIFIED WITH THE 8 BIT DATA BYTE ENTERED IN THE COMMAND. AT THE CONCLUSION, THE PROGRAM RETURNS TO THE MONITOR COMMAND MODE.

THE "SEARCH" COMMAND

THE SEARCH COMMAND IS INITIATED BY TYPING IN THE "S" COMMAND IN THE FOLLOWING FORMAT:

S HHH LLL, MMM NNN, DDD (CTRL/L)

WHERE "HHH LLL" SIGNIFIES THE START ADDRESS AND "MMM NNN" INDICATE THE ENDING ADDRESS OF THE BLOCK OF MEMORY TO BE SEARCHED FOR THE DATA PATTERN "DDD." WHEN THE OPERATOR ENTERS THE CTRL/L (OR C/R), THE PROGRAM BEGINS SEARCHING THE DESIGNATED MEMORY LOCATIONS FOR THE DATA PATTERN SPECIFIED IN THE COMMAND AND EACH TIME A MATCH IS FOUND, THE ASSOCIATED MEMORY ADDRESS IS OUTPUT TO THE DISPLAY DEVICE, PRECEDED BY A C/R, L/F COMBINATION TO START EACH ADDRESS OUTPUT ON A NEW LINE. THE PROGRAM RETURNS TO THE COMMAND MODE WHEN THE ENTIRE BLOCK HAS BEEN SEARCHED.
THE "TRANSFER" COMMAND

THE "TRANSFER" COMMAND IS INITIATED BY TYPING IN THE "T" COMMAND IN THE FOLLOWING FORMAT:

T HHH LLL, MMM NNN, YYY ZZZ (CTRL/L)

WHERE "HHH LLL" SPECIFIES THE START ADDRESS AND "MMM NNN" THE END ADDRESS OF THE BLOCK OF MEMORY THAT IS TO BE TRANSFERRED TO THE SECTION OF MEMORY WHICH STARTS AT LOCATION "YYY ZZZ." WHEN THE CTRL/L (OR C/R) IS ENTERED, THE PROGRAM BEGINS THE TRANSFER BY FETCHING THE CONTENTS OF THE MEMORY LOCATION "HHH LLL" AND STORES THAT VALUE IN THE LOCATION "YYY ZZZ." THE CONTENTS OF "HHH LLL+1" IS THEN TRANSFERRED TO "YYY ZZZ+1" AND SO ON, UNTIL THE CONTENTS OF THE LAST LOCATION "MMM NNN" HAS BEEN TRANSFERRED. THE PROGRAM THEN RETURNS TO THE COMMAND MODE.

PUTTING THE MONITOR PROGRAM ON "PROMS"

ONCE THE MONITOR PROGRAM PRESENTED IN THIS MANUAL HAS BEEN "CUSTOMIZED" TO THE READER'S PARTICULAR SYSTEM, BY MODIFYING OR EXPANDING THE PROGRAM TO MEET THE REQUIREMENTS OF ONE'S SYSTEM, IT CAN BE EASILY ADAPTED FOR PERMANENT STORAGE ON "PROMS" TO ALLOW THE COMPUTER TO BE "ON-LINE" ONCE THE POWER IS TURNED ON BY SIMPLY JUMPING TO THE START ADDRESS OF THE MONITOR PROGRAM. THIS IS MADE POSSIBLE BY HAVING ALL TEMPORARY DATA STORED IN THE FIRST 256 LOCATIONS OF RAM MEMORY. IF ONE IS TO PUT THE MONITOR PROGRAM ON "PROMS" THERE ARE SEVERAL FACTS THAT MUST BE BROUGHT OUT. FIRST, THE PROGRAM SHOULD BE LOCATED IN THE UPPER-MOST SECTION OF MEMORY THAT THE SYSTEM IS CAPABLE OF ADDRESSING. NEXT, THE COMMAND LOOK UP TABLE AND CANNED MESSAGES SHOULD BE MOVED TO BE INCLUDED IN THE PROM SECTION OF THE PROGRAM. THIS REQUIRES THAT THE POINTERS TO THESE TWO AREAS IN THE "COMMAND INPUT" ROUTINE AND THE "HDLA" SUBROUTINE, BE CHANGED TO INDICATE THE NEW START ADDRESSES. ALSO, IN THE COMMAND INPUT ROUTINE, WHEN THE START ADDRESS OF THE COMMAND TO BE EXECUTED IS STORED AT LOCATIONS 156 AND 157, THE PROGRAM SHOULD ALSO STORE THE "104" PORTION OF THE JUMP INSTRUCTION AT LOCATION 155, TO SET UP THE JUMP INSTRUCTION PROPERLY WHEN THE FIRST COMMAND IS ENTERED. AND FINALLY, BEFORE PUTTING THE PROGRAM ON "PROMS," MAKE SURE THAT EACH FUNCTION IS CHECKED OUT THOROUGHLY, THEREBY, DECREASING THE LIKELYHOOD THAT THE PROMS WILL HAVE TO BE RE-PROGRAMMED TO CORRECT SOMETHING THAT WAS OVERLOOKED ON THE INITIAL PROGRAMMING.

HAVING THIS TYPE OF PROGRAM ON PROM HAS SEVERAL IMPORTANT ADVANTAGES. AS MENTIONED ABOVE, IT ALLOWS IMMEDIATE "ON-LINE" CAPABILITY. IT ALSO PREVENTS A PROGRAM BEING DELETED FROM "WIPE IT OUT." SHOULD THE NEW PROGRAM HAVE A NEVER-ENDING LOOP IN IT WHICH TRIES TO STORE SOME DATA IN EVERY MEMORY LOCATION THE COMPUTER CAN ACCESS. FINALLY, THE SUBROUTINES OF THE MONITOR PROGRAM WILL ALWAYS BE AVAILABLE FOR OTHER PROGRAMS TO CALL AS THEY REQUIRE.

THE MONITOR PROGRAM IS AN EXTREMELY USEFUL TOOL, AS ANYONE WILL TEST TO THAT HAS WORKED ON A COMPUTER WITH AND WITHOUT A MONITOR. IT IS HOPEFUL THAT THIS MONITOR PROGRAM WILL GET THE READER OFF ON THE RIGHT FOOT TOWARDS TRANSFORMING ONE'S COMPUTER SYSTEM FROM A BOX THAT MERELY BLINKS ITS LIGHTS TO A FULLY FUNCTIONAL OPERATING SYSTEM THAT WILL PERFORM MANY OF THE TASKS EXPECTED OF IT.