ZENTEC

PROGRAMMER'S LANGUAGE MANUAL

9002

MICROCOMPUTER TERMINAL SYSTEM

Zentec Corporation

Zentec Part No. 88-403-01
REV B

PROGRAMMER'S LANGUAGE MANUAL

9002

MICROCOMPUTER TERMINAL SYSTEM

Zentec Corporation
2390 Walsh Avenue
Santa Clara, California 95050

Issued: April 1975

Table of Contents

List of Illustrations

1.0 INTRODUCTION

The intent of this document is to give the Assembly Language programmer the information necessary to program the Zentec 9002 Display System.

The Zentec 9002 Display System uses the Intel 8008 microcomputer. The information presented in this document may not be meaningful unless the reader is familiar with the Intel 8008. The instruction set of the Intel 8008, as described in the MCS-8 Users Manual is OCTAL oriented. The Zentec 9002 is oriented to hexadecimal representation. Appendix 7-A contains a cross-reference from the 9002 Assembler Language to the INTEL instruction set. Machine codes are included, in both binary (OCTAL form) and in hexadecimal form.

The purpose for hexadecimal code is to relate programmers to other hexadecimal systems as all IBM systems and Interdata machines. No attempt has been made to relate to OCTAL representation except that which appears in Appendix 7-A.

The Intel machine uses RAM which is data destructive when power is removed. In order to protect the code that is necessary for system operation, Read Only Memory (ROM) and Programmable Read Only Memory (PROM) are used. In Section 2.0 the memory organization is described. It is described in terms of decimal addresses but in fact we use hexadecimal addresses internally. The system is essentially arranged in 4 sections of 4,096 decimal bytes each. The first section is devoted to ROM code. The second section is devoted to the refresh RAM that is displayed on the screen. The last two sections are optional RAM space for use by customers for their own application code or for use with the disk option or as telecommunication buffers. Figure 2—1 graphically describes the memory organization.

Hardware and software registers are described in Section 3.0. They are described in some detail and should be carefully studied by the application programmer so that he is aware of the precise meanings of the registers and their locations.

Section 4.0 describes the system list structures. List structures are a convenient mechanism to reduce the amount of code and the amount of space necessary to not only perform functions but to easily change functions. This technique is used throughout Zentec 9002 system code. Appendix 4-0-A graphically describes a list path.

Section 5 describes all the Basic Systems Sub-routines. These routines have been built to support the system package already developed and have been designed to be used by the application programmer in developing his own modules. Each routine acts as a closed sub-routine and is described separately. The term "registers affected" describes all the registers that are modified by the specific function. Any registers not modified can be used to store data during the function operation without expectation of that data being destroyed. The term "example" indicates the Zentec assembler mnemonic to perform that function. The term "results" describes the values that can be expected in the registers at the completion of that function operation. Where relevant, software registers that have been modified, with identifiable results, are depicted.

Section 6.0 describes Device Oriented Routines. It is our intent to describe code sequences, in Zentec assembler format, for all supported devices. This will include key-board handling, disk operation, telecommunications and printer use. The routines described will not necessarily be the ones used by the system but will be tested and can be used directly by the application programmer.

Section 7.0 System Support Routines will describe in detail each of the Zentec packages designed to support the application programmer.

2.0 SYSTEM AND SOFTWARE ORGANIZATION

2.1 Memory Organization

Three forms of memory exist in the 9002: ROM, PROM and RAM. The first 4096 bytes are committed to ROM or PROM. The basic ROM, a set of sub-routines designed for both system and user use is located from decimal address 2048 to 4095. The Tele-Communications package (GT1) is also a ROM, and is located at decimal address 0 to 2047. Locations 0 through 511 decimal hold the 9002 executive, the branch list, and five offset lists.

The space from 4096 thru 16383 is system structured as RAM, although parts of that space can be used as additional ROM or PROM area for customer optioned routines. Hardware and programming registers and system RAM work space is located at 4096 to 4143. The display control line, always the bottom line on the screen, is located in the next eighty bytes from 4144 to 4223. The first page of the display, 1920 bytes, or 24 lines of 80 characters each, is located at 4224 to 6143. The optional second page of 1920 bytes is located at 6144 to 8063. The next 48 bytes, from 8064 to 8111 are reserved for additional hardware and software registers to support optional devices. The space from 8112 thru 8191 is available for user use and is a part of the second page option. The space from 8192 through 16383 is available as additional user RAM space, however the Super Text option will reside from 14,336 through 16,383.

2.2 The Basic ROM

The Basic ROM is a set of forty odd sub-routines designed to reduce programming effort in handling data on the screen and in performing terminal oriented functions. The ROM takes up precisely 2048 bytes. The routines were intended for use by the system, but are in fact easily used by application programmers. Section 5.0 describes these routines and how they are entered.

2.3 List Structure

The 9002 system list structures were designed to reduce programming effort by providing an easily changed, easily added to mechanism to perform desired system functions. A new routine can be inserted at any available location. To insert the routine in the list structure requires two bytes to specify the entry point of the new routine, as well as a single byte offset for each offset list the function is to be related. The current list structures are keyboard RS-232 or merely byte oriented. The advantage to the list approach is the ease of maintenance and ease of entry of new routines within the system. See Section 4.0 for more detailed list information.

2.4 Device Operation

The 9002 system does not use the Intel 8008 family input/output instructions. Devices are instead tied to RAM registers which are interrogated by the system program, and where appropriate by the user program, as necessary. As an example, all keyboard entered characters appear at a single byte location in the memory (X'1002'). It is the programmer's responsibility to see that his programs are written to access the keyboard input byte often enough to guarantee no loss of keyboard information.

As mentioned previously, the hardware takes a passive role in system operation in that the program has almost complete control of system operation, exclusive of RAM and screen refresh. This philosophy gives the programmer significant freedom in performing operations, but added responsibility in accomplishing tasks.

0-2047 Hex (0000-01FF)	System Executive and Lists GT1 - Tele-Communications
2048-4095 Hex (0800-0FFF)	Basic ROM
4096-6143 Hex (1000-17FF)	Hardware and Software Registers Display Page One
6144-8191 Hex (1800-1FFF)	Display Page Two Optional Device Registers
8192-10239 Hex (2000-27FF)	Additional RAM
10240-12287 Hex (2800-2FFF)	Additional RAM
12288-14335 Hex (3000-37FF)	Additional RAM
14336-16383 Hex (3800-3FFF)	Additional RAM or Optional ROM

Figure 2—1
SYSTEM MEMORY ORGANIZATION

3.0 HARDWARE AND SOFTWARE REGISTERS

All system registers are located in RAM. The reason: To allow programs, both system and application, to be able to control system supported devices such as the display, the keyboard, the RS-232 interface, and certain optional interfaces.

3.1 HARDWARE REGISTERS

3.1.1 Cursor – The cursor register is a two byte register. The first byte establishes the row or line member, and the second byte refers to the column. In the standard machine, the row number can range from X'00' to X'18'. Row number X'00' is reserved for the control line. With the two-page option, the row value range is X'00' to X'4F', or eighty bytes. The cursor register is located at X'1000' and X'1001'. Values in these registers are generated by programming.

3.1.2 Keyboard Input – The keyboard input register is located at X'1002'. It is theoretically possible to get 255 different codes through this register. One code, X'FF', is used as the normal quiesced state for this register and cannot be used as an input code. The standard keyboard, using upper and lower case with control characters generates 128 codes. By depressing the control key and any alpha key (and a few special characters), an additional 32 characters can be generated. The eleven key numeric pad generates an additional eleven codes. Thus the standard keyboard provides 171 out of the possible 255 acceptable codes. The keyboard input register has no overrun protection. That is, if multiple characters are keyed and the program in control does not use them, the succeeding keystroked characters overlay one another. No indication of overrun exists. After each character is accepted by the/a program, an X'FF' value should be loaded into the register by the program. Characters can be inserted through the keyboard at a maximum rate of sixty characters per second.

3.1.3 Function Register – The function register currently has but one use: an error beep. When the high-order or left-most bit of this register is 'changed', an error tone is produced for approximately two seconds. The bit transition, from off to on or on to off triggers the error tone. All other bits in the function register are reserved. The function register is located at X'1003'.

3.1.4 Prior Condition Register – The prior condition register is used to establish the initial screen polarity and blinking characteristics for the line scans. Special control characters placed within the refresh RAM area will vary the screen polarity, tone and blinking characteristics, but it is necessary to establish 'initial' conditions or the first refresh RAM location of each line of the screen would be committed to establishing 'current' screen characteristics. The high-order 3 bits, bits numbered 7, 6 and 5 are reserved and must be zero. Bit 4 is used to establish an underscore; Bit 3 is used to make data appear as blanks; Bit 2 reverses screen polarity from dark background to light background; Bit 0 provides a half-tone value for dark background polarity; and Bit 1 provides blinking characters. The prior condition register is located at X'1004'.

3.1.5 Page Register – The page register is used to adjust the screen window of 24 lines to any contiguous set of 24 lines within the range of 48 lines available when the double-page option is in the system. The lowest value acceptable in this register is X'00', but under system operating conditions should not be less than X'01'. The high value should be held at X'19'. Any value above X'19' will give unexpected, unacceptable results. The page register is changed from value X'01' for page one to X'19' for page two. It is varied by the value at location X'1010' to perform scrolling.

3.1.6 RS-232 — The RS-232 interface is made up of six byte registers: a one byte input buffer; a one byte output buffer; a one byte input flag register; a one byte output flag register; a one byte input sense register; and a one byte output sense register. The RS-232 interface operates one byte at a time, and ranges in speed from 110 baud to 9600 baud. One RS-232 is standard, and a second dual RS-232 is optional. The standard RS-232 is located at X'100C' thru X'100F' and X'102E' and X'102F'.

3.2 SOFTWARE REGISTERS

3.2.1 Branch Area — The branch area is used by the system executive to cause branches into the various function routines in the system. It is located at X'1008' thru X'100A'. Location X'1008' must always contain a X'44'.

3.2.2 TAS — The third address (TAS) is a two byte field used as a temporary work area for 16 bit values. Two of the basic routines use TAS. TAS overlays the value portion of the Branch Area. It is located at X'1009' and X'100A'.

3.2.3 Scroll Value — The scroll value is the amount added to or subtracted from the value in the Page Register to vary the starting line of data on the screen. This value is X'02' unless modified by the user. The scroll value is located at X'1010'.

3.2.4 Input Buffer Pointer — The system executive supports a five position buffer to smooth keyboard input. The input buffer pointer holds the next available buffer address. The input buffer pointer is located at X'1011'.

3.2.5 Protected Cursor Flag — The protected cursor flag is used to allow the cursor to be positioned under a protected character. The flag is tested as zero or non-zero. The protected cursor flag is located at X'1012'.

3.2.6 Line Space Count — The line space count is used to speed character inserts. This value defines the number of available spaces from the last character, on the line holding the cursor, to the end of the line. This value is used exclusively in 'insert' sub-mode. The line space count is located at X'1014'.

3.2.7 Local Mode Save — The local mode save location holds the code representing the mode and sub-mode that was in control prior to entering 'Control' mode. It is used to reestablish the proper mode and sub-mode at return from 'Control' mode. The local mode save is located at X'1015'.

3.2.8 Current Mode — The current mode code is maintained for program control and list control purposes. It is located at X'1016'.

3.2.9 Character Previous Hold — The character previous hold maintains the value of the last keyboard character acted upon. It is used for special double-character sequences. It is located at X'1017'.

3.2.10 Character Current Hold — The character current hold maintains the value of the keyboard character currently being processed. It is located at X'1018'.

3.2.11 Keyboard Input Buffer — The keyboard input buffer is a five byte buffer used by the system to support keyboard input at its maximum rate, while allowing functions of various speeds to be performed. The keyboard buffer is located at X'1019' thru X'101E'.

3.2.12 FAS – The first address (FAS) is a two byte field used to hold the 16 bit binary value of the current cursor address. All cursor manipulation programs operate with FAS, and the value then converted into Row and Column values which are then inserted into the cursor hardware registers. FAS is located at X'1020' and X'1021'.

3.2.13 SAS – The second address (SAS) is a two byte field used as a temporary work area for 16 bit values. A number of the basic routines use SAS. SAS is located at X'1022' and X'1023'.

3.2.14 Open work areas – Locations X'1024' thru X'102D' are used by various basic routines as unnamed temporary work spaces. All unspecified locations in the area from X'1000' thru X'102F' are reserved for future hardware options and for programming enhancements.

4.0 SYSTEM LIST STRUCTURES

The Intel 8008 family use a two byte address for all calls and branches. The two byte address is, of course, the starting point of a programmed routine. The 9002 system strings all function or routine addresses together in one contiguous list, two bytes per entry. This list is the branch list and is the backbone of the 9002 list structure. Whenever a new routine is created, its initial entry point address is attached to the branch list. To acquire any specific address, one would need only the offset in bytes from the start of the branch list and the address of the start of the branch list. The value generated by summing these two values is the address of the pointer to the routine desired. To get to any desired routine then one merely needs a list of offsets because the system branch list address is known. In the 9002 standard system, five offset lists exist; one for each mode and sub-mode REP and INS FORM, REP and INS TEXT and one for the Control Mode. The 9002 offset lists are each 33 bytes in length. The first 32 offsets (one byte each) are directly related to the value of the 32 function codes generated from the keyboard. The function code keys generate hexadecimal numbers from X'00' to X'1F'. As an example, the 'HOME' key generates a code of X'0B'. Therefore it would be appropriate to place the 'HOME' routine branch table offset into the twelfth or '0B'th entry of the desired offset list. When the 'HOME' key is depressed, and the desired offset list is entered, the 'HOME' routine address offset is found. Summing that offset value with the start of the branch table generates the address of the two byte field whose value is the address of the start of the 'HOME' routine. By picking up that address and inserting it properly in a branch instruction, that branch can be executed to enter (and perform) the 'HOME' routine function. As stated, the first 32 bytes of each offset list are associated with the codes generated by pressing the 32 keyboard function keys. The 33rd entry is the offset to the address in the branch table of all other keyboard keys. The branch list, and each of the offset lists must in and of themselves be contiguous but they may be separated from each other. See Figure 4–1 for graphical description.

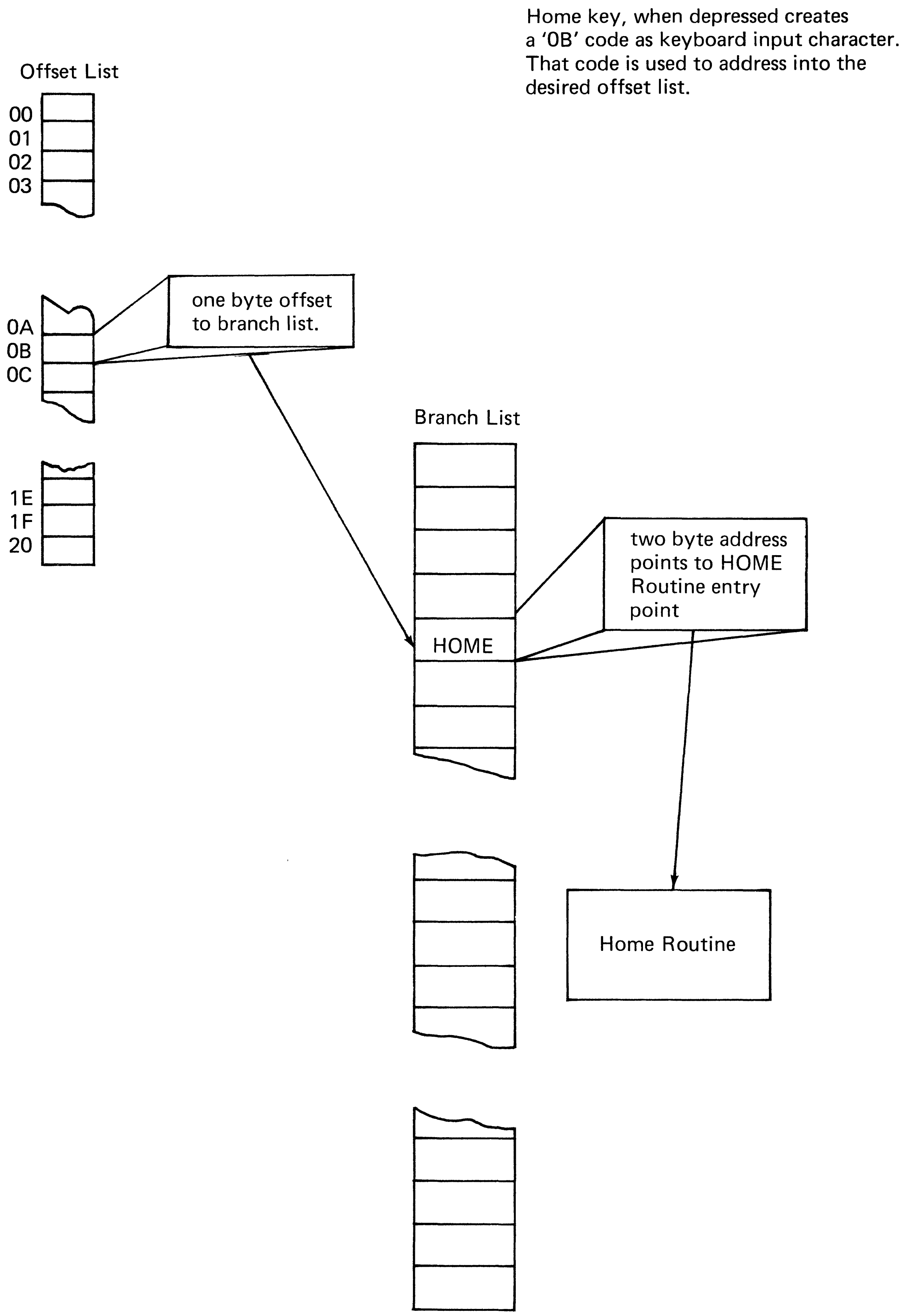

Home key, when depressed creates a '0B' code as keyboard input character. That code is used to address into the desired offset list.
Offset List
00
01
02
03
0A
0B
0C
1E
1F
20
one byte offset to branch list.
Branch List
HOME
two byte address points to HOME Routine entry point
Home Routine

Figure 4—1
BRANCH & LIST GEOGRAPHICAL DESCRIPTION

5.0 SYSTEM BASIC SUB-ROUTINES

5.1 CURSOR MOVE ROUTINES

CRIGHT Moves Cursor One Position to Right.

Moves cursor (on screen) one position to right. If cursor is at last column to right, cursor is moved to first position to left of the next line. If cursor is on last available line at last column, cursor is moved to first position of first line. Cursor will automatically bypass any 'protected' byte locations, unless location X'1012' is non-zero.

Registers affected – All.

Example – Call CRIGHT.

Results – All registers indeterminate. Both actual, terminal cursor registers and binary cursor location (FAS) are properly up dated. Where double-page option exists, page adjustment to cursor position is also up dated.

5.1-1

CLEFT Moves Cursor One Position to Left.

Moves cursor (on screen) one position to left. If cursor is at first column to left, cursor is moved to last column on right of next preceding line. If cursor is on first line, first column to left (true HOME) it will not be moved. Unless location X'1012' is non-zero, cursor will not stop at a 'protected' byte location, but will continue going left. If True Home is found to be protected, cursor will search right and stop at first available not protected byte location.

Registers affected – All.

Example – Call CLEFT.

Results – All registers indeterminate. Both actual, terminal cursor registers and binary cursor location (FAS) are up dated. Where double-page option exists, page adjust is also made.

5.1-2

CUP Moves Cursor One Line Up.

Moves cursor (on screen) one line up. If cursor is already at top-most line, it will not be moved. Cursor will not stop at a protécted byte unless location X'1012' is non-zero. Attempt to move cursor to top-line, if it is protected, and location X'1012' is zero, will result in cursor moving to right on top line until a non-protected byte location is found.

Registers affected — All

Example — Call CUP

Results — All registers indeterminate. Hardware cursor registers (X'1000-1001') and software cursor register (FAS) are up dated. Where double-page option exists, adjustment is made automatically.

5.1-3

CDOWN Moves Cursor One Line Down

Moves cursor (on screen) one line down. If cursor is at bottom-most line, it is moved to top-most line. Cursor will not stop at a protected byte, unless location X'1012' is non-zero.

Registers affected — All.

Example — Call CDOWN.

Results — All registers indeterminate. Hardware cursor registers (X'1000-1001') and software cursor register (FAS) are up dated. Where double-page option exists, page adjustment is made automatically.

5.1-4

RETURN Moves Cursor to first position next line.

Moves cursor from current position to the left-most position of next line. If current cursor position is on last displayable line, cursor is moved to true HOME. Cursor will not stop at protected byte but will scan right to first non-protected location.

Registers affected — All.

Example — Call RETURN.

Result — All registers indeterminate. Hardware and software cursor registers up dated. Where necessary, page is adjusted.

5.1-5

HOME Cursor is moved to HOME location.

Cursor is moved from current location to left-most position, top-line of current screen image. If that position is protected, cursor will scan right until first unprotected byte is found.

Registers affected — All.

Example — Call HOME.

Result — All registers indeterminate. Hardware and software cursor registers up dated.

5.1-6

TAB Move Cursor to Next Tab Stop.

Cursor is moved from current location to next tab stop to right. If there are no more stops on the line, or if no stops exist, the cursor is moved to the left-most position. of the next line. In this case it always acts precisely like RETURN.

Registers affected — All.

Example — Call TAB.

Result — All registers indeterminate. Hardware and software cursor registers are up dated.

5.1-7

BTAB Move Cursor to Next Previous Tab Stop.

Cursor is moved from current location to next previous tab stop. If the next previous tab stop on this line is a protected byte, the cursor will scan right to the first available non-protected byte.* If there is no next previous tab stop on this line, or no tab stops at all, the cursor will move to the left-most position of the current line. If already at the left-most position, the cursor will move to the last position of the next preceding line and scan left. In no case will the cursor move past true HOME in its backward scan.

Registers affected — All.

Example — Call BTAB.

Result — All registers indeterminate. Hardware and software cursor registers are updated.

* The cursor cannot ever move left past a protected byte at the tab stop location, by using this function.

5.1-8

ATAB Auto Tab Forward.

Moves the cursor to the right to the first unprotected byte beyond the next set of protected byte (s). If the right scan searches beyond the last displayable character, the cursor is set at true HOME.

Registers affected – All.

Example – Call ATAB.

Result – All registers indeterminate. Hardware and software cursor registers are updated.

5.1-9

ABTAB Auto Tab Backward.

Moves the cursor to the left past the next preceding set of protected byte (s), and past all the unprotected bytes until it reaches the left-most byte of that unprotected set. If at initial cursor scan, the next preceding byte is not protected, the scan ends at the left-most byte of the current unprotected set.

Registers affected – All.

Example – Call ABTAB.

Result – All registers indeterminate. Hardware and software cursor registers are updated.

5.2 VIEWABLE SCREEN FUNCTIONS

CLEAR Clears all viewable memory.

Clears all viewable memory to blanks. No exceptions.

Registers affected – All.

Example – Call CLEAR.

Result – All registers indeterminate. Cursor is repositioned to true Home location. Where necessary first page is set. All cursor registers are up dated.

5.2-1

NEWFRM Clears viewable memory of all unprotected data.

All viewable memory with exception of Control line and protected bytes is cleared to blanks.

Registers affected – All

Example – Call NEWFRM.

Result – All registers indeterminate. Cursor is repositioned to true Home. Where necessary, first page is set. All cursor registers are up dated.

5.2-2

EOS Erase to End of Screen.

Blanks screen from current cursor location to last screen displayable position. Protected bytes are not blanked unless the value at X'1016' is X'80' or greater.

Registers affected – All.

Example – Call EOS

Result – All registers indeterminate. Cursor remains at same position.

5.2-3

EOL Erase to End of Line.

Blanks screen from current cursor location to end of line. Protected bytes are not blanked unless the value at X'1016' is X'80' or greater.

Registers affected – All.

Example – Call EOL.

Result – All registers indeterminate. Cursor remains at same location.

5.2-4

BLANK Screen Blanking, Protected and Unprotected.

That portion of the screen ranged from the value (binary address) in FAS up to but not including the value in SAS is blanked. Both protected and unprotected areas are blanked. Indeterminate results can be expected if the value in FAS is not less than that in SAS.

Registers affected — All.

Example — After establishing FAS and SAS,
Call BLANK.

Result — All bytes in the area specified are changed blanks (X'20'). Register values at completion are indeterminate.

5.2-5

CMESSA Control Label Insertion.

The control label at the extreme lower right of the visible screen is loaded by this routine. Any new label can be loaded to that area by specifying the address of the right-most byte of an eight position label, in the register pair H and L. Then call this routine.

Registers affected — All.

```
Example — LBI RH, (label address +7)
          LBI RL, (label address +7)

          Call CMESSA.
```

Result — Register values at completion are indeterminate. The new label appears properly.

5.2-7

DELBYT Delete Byte.

Deletes byte at cursor location, and moves all data to the right, to the end of the line or to the next protected byte to the left one byte. The last byte on the line or the last unprotected byte in the current field is then blanked.

Registers affected — All.

Example — Call DELBYT.

Result — All registers indeterminate. Cursor is not moved.

DELFLD Delete Field.

From current cursor location, a scan is made to the left until either the start of line is reached or a protected byte is found. Then the scan is made to the right until either the end of line is reached or a protected byte is found. Within the established range, all unprotected bytes are blanked. The cursor is then repositioned at the left-most unprotected byte of the range.

Registers affected — All.

Example — Call DELFLD.

Result — All registers indeterminate. Hardware and software cursor registers are up dated.

5.2-9

INSERT Insert Byte.

From the current cursor location, all data to the right, to the end of the line or to the next set of protected byte (s), is moved right one position. If the last position on the line was affected, that last position is changed to a blank. The keyed character is then inserted at the current cursor location. The cursor is advanced automatically.

Registers affected — All.

Example — Call INSERT.

Result — All registers indeterminate. Cursor position advanced one position to right.

5.2-10

DLINE Delete Line.

Cursor is moved to extreme left position of current line. A search is made of successive lines to find an all blank line. When found, all lines from the one following the cursor line, up to and including the all blank line are moved up one line. Thus all following moved lines overlay their next preceding line. If no all blank line is found, the last line is blanked at completion of the move. If the cursor is on the last line, it is blanked.

Registers affected — All.

Example — Call DLINE.

Result — All registers indeterminate. Hardware and software cursor registers are updated.

5.2-11

ILINE Insert Blank Line.

Cursor is moved to extreme left position of current line. A search is made of successive lines to find an all blank line. When found, all preceding lines, up to and including the cursor line are moved down one line. The cursor line is then blanked. If no all blank lines are found, the last available line is blanked, and operation proceeds normally. If the cursor is on the last available line, the move is not performed, the last line is blanked, the routine is exited normally.

Registers affected — All.

Example — Call ILINE.

Result — All registers indeterminate. Hardware and software cursor registers are up dated.

5.2-12

5.3 MICROCOMPUTER ROUTINES

LDFAS Load First Address

The two byte value at address X'1020' is the absolute binary address of the current cursor location. It is known as FAS, or First Address. As the binary counterpart of the current cursor location is used so often, FAS has special load and store routines. Calling LDFAS will load the H and L registers with the value in FAS. That value will also be loaded in registers D and E.

Registers affected — D, E, H and L.

Example — Call LDFAS.

Result — The value in FAS (assume X'189C') will be loaded into registers D and E and also registers H and L. Register D will contain X'18', E will contain X'9C', H will contain X'18', and L will contain X'9C'.

Other — A portion of the LDFAS routine can be used to load registers H and L from almost any addressable memory pair (note exception), by loading the address of the value desired into registers H and L, and then performing a call to location LDFAS +4.

Example —
```
LBI RH, XX
LBI RL, YY

Call LDFAS +4
```

Where XX is the high order byte of the desired location and YY is the low order byte of the desired location.

Result — Same as in basic LDFAS routine, register pair D and E, H and L are loaded with the value located at XX YY.

Exception — Load results will be indeterminate, and almost certainly wrong, if the memory pair addressed crosses a hexadecimal century boundary, i.e. if the memory address to be loaded ends in X'FF'.

5.3-1

STFAS Store First Address.

The two byte value in register pair H and L is stored at FAS (absolute binary address X'1020').

Registers affected — D, E, H and L.

Example — Call STFAS.

Result — At completion, FAS contains value that was in register pair H and L. Register pair D and E also contains value originally in H and L. Register pair H and L contains the address of FAS +1.

Other — A portion of the STFAS routine can be used to store the value in register pair D and E into FAS.

Example — Call STFAS +2.

Result — Same as basic STFAS result.

5.3-2

LDSAS Load Second Address

The two byte space at address X'1022' is used as temporary storage by a significant number of the Basic Call routines. Calling LDSAS will load the register pair H and L with the value in SAS.

Registers affected — D, E, H and L.

Example — Call LDSAS.

Result — The value in SAS will be loaded in register pairs D and E, and H and L.

5.3-3

STSAS Store Second Address.

The two byte value in register pair H and L is stored at SAS (absolute binary address X'1022').

Registers affected — D, E, H and L.

Example — Call STSAS.

Result — At completion, SAS and the register pair D and E will contain the value initially held in register pair H and L. H and L will contain the address of SAS+1.

5.3-4

LDTAS Load Third Address.

The two byte space at address X'1009' is used at temporary storage by several Basic Call routines. Calling LDTAS will load the register pair H and L with the value in TAS.

Registers affected — D, E, H and L.

Example — Call LDTAS.

Result — The value in TAS will be loaded into register pairs D and E, and H and L.

STTAS Store Third Address.

The two byte value in register pair D and E is stored at TAS (absolute binary address X'1009').

Registers affected — D, E, H and L.

Example — Call STTAS.

Result — At completion, TAS and the register pair D and E will contain the value initially held in register pair D and E. Register pair H and L will contain the address of TAS+1.

5.3-6

LDCURS Load Cursor.

The two byte terminal cursor value (discontinuous binary — column and row) located at X'1000' is loaded into the register pair D and E.

Registers affected — D, E, H and L.

Example — Call LDCURS.

Result — At completion, the register pair D and E contain the value from memory address X'1000'. Register pair H and L contain the value X'1000'.

5.3-7

BUMPHL Increase value in registers H and L by one.

The two byte binary value in register pair H and L is increased by one.

Registers affected – H and L.

Example – Call BUMPHL.

Result – 16 bit binary value in register pair H and L increased by one.

5.3-8

DECHL Decrease value in registers H and L by one.

The two byte binary value in register pair H and L is decreased by one.

Registers affected – H and L.

Example – Call DECHL.

Result – 16 bit binary value in register pair H and L decreased by one.

5.3-9

SUBREG Subtract paired register value from another paired register value.

Subtracts 16 bit value in register pair B and C from 16 bit value in register pair D and E and stores 16 bit result in register pair B and C.

Registers affected – A, B, C, D, and E.

Example – After establishing registers B, C, D and E,

Call SUBREG.

Result – Registers D and E will remain as they were just prior to entry to this routine. Registers D and C will hold the new result value.

5.3-10

CONV Converts Binary Cursor Value in FAS to Hardware Cursor Value.

Takes the 16 bit binary current cursor value from FAS, and converts it to row and column discontinuous binary value of terminal, and stores the value in the terminal Hardware register for showing the cursor on the screen.

Registers affected – All.

Example – Call CONV.

Result – Registers A, B, C are indeterminate. Paired registers D and E will contain the new Hardware cursor value, and paired registers H and L will contain the value X'1001'.

5.3-11

RECON Generates Binary Cursor value in FAS from value in Hardware Cursor Register.

Takes the Row/Column current Hardware cursor value and converts it to a 16 bit binary value and stores that value at FAS.

Registers affected – All.

Example – Call RECON.

Result – Registers A, B and C are indeterminate. Register pair D and E contain the new 16 bit binary value representing the current cursor location, and register pair H and L contain the address of FAS+1.

5.3-12

COMPER 16 bit Comparator Routine with High, Low or Equal Results.

The value in register pair B and C is compared against the value in register pair D and E. At completion, Register B holds the High, Low or Equal result.

Registers affected – A, B, C, D and E.

Example – After loading the values to be compared in register pairs B, C and D, E then,

Call COMPER.

Result – Register A is indeterminate. Registers C, D, E, H and L are unchanged. Register B contains:

X'02' if value in register pair D and E is numerically greater than value in B and C.

X'01' if value in register pair D and E is less than value in B and C.

X'00' if values are equal.

5.3-13

ADD2 Adds single byte value to double byte value.

Performs addition of value in register C to value in register pair D and E. Results are placed in register pair D and E.

Registers affected – A, D and E.

Example – After loading 16 bit value in registers D and E, and loading add value in register C, then,

Call ADD2.

Result – Register A is indeterminate. Registers B, C, H, and L are not changed. Register pair D and E contain the new value.

5.3-14

SUBT2 Subtracts single byte value from double byte value.

Performs subtraction of value in register C from value in register pair D and E. Results are placed in register pair D and E.

Registers affected – A, D and E.

Example – After loading 16 bit value in registers D and E, and loading subtract value in register C, then,

Call SUBT2.

Result – Register A is indeterminate. Registers B, C, H, and L are not changed. Register pair D and E contain the new value.

5.3-15

RMOVE Moves Data in RAM, low-order to high-order addresses.

Moves from minimum of one byte to maximum of 256 bytes from any addressable area in memory to any RAM location(s) in memory. The move is byte by byte, moving the leftmost byte of the 'from' block to the leftmost byte location of the 'to' block first, then incrementing addresses and moving each additional byte until all bytes of move have been made. Register pair D and E must be loaded with the starting location of the 'to' block. Register pair H and L must be loaded with the starting location of the 'from' block. Register C is loaded with the value X'01' to X'FF' to move from 1 to 255 bytes. Loading register C with X'00' will cause 256 bytes to be moved.

Registers affected — All.

Example — After loading register pair H and L with the 'from' location, and after loading register pair D and E with the 'to' location, and after loading register C with move count then,

Call RMOVE.

Result — Register A and B are indeterminate. Register C is X'00'. Register pair D and E point to the last byte plus one of the 'to' area. Register pair H and L point to the last byte plus one of the 'from' area.

5.3-16

LMOVE Moves Data in RAM, high-order to low-order addresses.

Moves from minimum of one byte to maximum of 256 bytes from any addressable area in memory to any RAM location(s) in memory. The move is byte by byte, moving the rightmost byte of the 'from' block to the rightmost byte location of the 'to' block first, then decrementing addresses and moving each additional byte until all bytes of move have been made. Register pair D and E must be loaded with the starting location of the 'to' block. Register pair H and L must be loaded with the starting location of the 'from' block. Register C is loaded with the value X'01' to X'00' will cause 256 bytes to be moved.

Registers affected — All.

Example — After loading register pair H and L with the 'from' location, and after loading register pair D and E with the 'to' location, and after loading register C with move count, then,

Call LMOVE.

Result — Registers A and B are indeterminate. Register C is X'00'. Register pair D and E point to the last byte moved minus one, of the 'to' area. Register pair H and L point to the last byte minus one of the 'from' area.

5.3-17

SMOVE Special Move for data going to Control Line.

The Control Line (bottom line of screen — always) has a special function associated with the high-order bit of each byte on that line (Addresses X'1030' - X'107F'). The Special Move inserts data on that line without affecting the high-order bits. In all other respects this move is treated as an 'LMOVE' function. Thus, data moved to the control line must be addressed from the right side rather than the left, etc. See LMOVE for additional information.

Registers affected – All.

Example – After loading register pair D and E with the 'to' location, and loading register pair H and L with the 'from' location, and loading register C with move count,

Call SMOVE.

Result – See LMOVE for results.

5.3-18

5.4 TWO-PAGE OPTION ROUTINES

UPAGE Show Page Two on Screen.

Hardware page register (X'1005') is set to X'19'.

Registers affected – H, L.

Example – Call UPAGE.

Result – Register pair H and L contain X'1005'.

Note – Two page option required.

DPAGE Show Page One on Screen.

Hardware page register (X'1005') is set to X'01'.

Registers affected – H, L.

Example – Call DPAGE.

Result – Register pair H and L contain X'1005'.

Note – Two page option required.

5.4-2

DSCROL Scroll page data downward.

Screen view 'window' of data is moved upward, but page data appears to move downward. Hardware page register value is decreased by value in location X'1010'. Page register value may not be less than X'01'.

Registers affected – A, B, C, H and L.

Example – Call DSCROL.

Result – Registers A, B and C are indeterminate. Register pair H and L contain X'1005'.

Note – Two page option required.

USCROL Scroll page data upward.

Screen view 'window' of data is moved downward, but page data appears to move upward. Hardware page register value is increased by value in location X'1010'. Page register value may not exceed X'19'.

Registers affected – A, B, C, H and L.

Example – Call USCROL.

Result – Registers A, B and C are indeterminate. Register pair H and L contain X'1005'.

Note – Two page option required.

5.4-4

6.0 DEVICE ORIENTED ROUTINE

The routines in this section are tested and can be used either directly or as examples by the interested programmer. Routines similar to these are used in the ZENTEC 9002 where necessary.

6.1 ZENTEC 9002 KEYBOARD

Instructions in this example routine are written in ZENTEC assembler format. A translator table between the ZENTEC assembler statements and INTEL's MCS-8 statements will be found in Appendix 7A.

The following is a 'CALLed' sub-routine.

* Keyboard Character Pick Routine

```
PICK     LBI    RH, 10      Load Registers H and L with Keyboard Input Register Address
         LBI    RL, 02
PICKA    LB     RA          Load Accumulator from Memory
         CI     RA, FF      Is NULL character present?
         BE     PICKA       Yes, then branch and try again
         STBI   M, FF       Store the NULL character
         RET                Return with Data Character in Register A
```

This routine will hang the CPU until a character appears from the keyboard.

7.0 System Support Routines

7.1 ZENTEC Assembler

Table 7—1
Operation Codes

OPERATION CODES:

ZENTEC ASSEMBLY STATEMENT		MCS-8 EQUIVALENT	MCS-8 BINARY CODES OCTAL FORM		HEXADECIMAL
Load Register — Data moves from 2nd Register into 1st.					
LR	RA,RA	LAA	11 000 000	=	C0
	RA,RB	LAB	11 000 001	=	C1
	RA,RC	LAC	11 000 010	=	C2
	RA,RD	LAD	11 000 011	=	C3
	RA,RE	LAE	11 000 100	=	C4
	RA,RH	LAH	11 000 101	=	C5
	RA,RL	LAL	11 000 110	=	C6
LR	RB,RA	LBA	11 001 000	=	C8
	RB,RB	LBB	11 001 001	=	C9
	RB,RC	LBC	11 001 010	=	CA
	RB,RD	LBD	11 001 011	=	CB
	RB,RE	LBE	11 001 100	=	CC
	RB,RH	LBH	11 001 101	=	CD
	RB,RL	LBL	11 001 110	=	CE
LR	RC,RA	LCA	11 010 000	=	D0
	RC,RB	LCB	11 010 001	=	D1
	RC,RC	LCC	11 010 010	=	D2
	RC,RD	LCD	11 010 011	=	D3
	RC,RE	LCE	11 010 100	=	D4
	RC,RH	LCH	11 010 101	=	D5
	RC,RL	LCL	11 010 110	=	D6
LR	RD,RA	LDA	11 011 000	=	D8
	RD,RB	LDB	11 011 001	=	D9
	RD,RC	LDC	11 011 010	=	DA
	RD,RD	LDD	11 011 011	=	DB
	RD,RE	LDE	11 011 100	=	DC
	RD,RH	LDH	11 011 101	=	DD
	RD,RL	LDL	11 011 110	=	DE
LR	RE,RA	LEA	11 100 000	=	E0
	RE,RB	LEB	11 100 001	=	E1
	RE,RC	LEC	11 100 010	=	E2
	RE,RD	LED	11 100 011	=	E3
	RE,RE	LEE	11 100 100	=	E4
	RE,RH	LEH	11 100 101	=	E5
	RE,RL	LEL	11 100 110	=	E6
LR	RH,RA	LHA	11 101 000	=	E8
	RH,RB	LHB	11 101 001	=	E9
	RH,RC	LHC	11 101 010	=	EA
	RH,RD	LHD	11 101 011	=	EB
	RH,RE	LHE	11 101 100	=	EC
	RH,RH	LHH	11 101 101	=	ED
	RH,RL	LHL	11 101 110	=	EE

ZENTEC ASSEMBLY STATEMENT		MCS-8 EQUIVALENT	MCS-8 BINARY CODES OCTAL FORM		HEXADECIMAL
LR	RL,RA	LLA	11 110 000	=	F0
	RL,RB	LLB	11 110 001	=	F1
	RL,RC	LLC	11 110 010	=	F2
	RL,RD	LLD	11 110 011	=	F3
	RL,RE	LLE	11 110 100	=	F4
	RL,RH	LLH	11 110 101	=	F5
	RL,RL	LLL	11 110 110	=	F6

Load Byte – Data moves from Memory to Register

LB	RA	LAM	11 000 111	=	C7
	RB	LBM	11 001 111	=	CF
	RC	LCM	11 010 111	=	D7
	RD	LDM	11 011 111	=	DF
	RE	LEM	11 100 111	=	E7
	RH	LHM	11 101 111	=	EF
	RL	LLM	11 110 111	=	F7

Store Byte – Dtat moves from Register to Memory

STB	RA	LMA	11 111 000	=	F8
	RB	LMB	11 111 001	=	F9
	RC	LMC	11 111 010	=	FA
	RD	LMD	11 111 011	=	FB
	RE	LME	11 111 100	=	FC
	EH	LMH	11 111 101	=	FD
	EL	LML	11 111 110	=	FE

Load Byte Immediate – Immediate to Register

LBI	RA,nn	LAI	00 000 110	=	06
	RB,nn	LBI	00 001 110	=	0E
	RC,nn	LCI	00 010 110	=	16
	RD,nn	LDI	00 011 110	=	1E
	RE,nn	LEI	00 100 110	=	26
	RH,nn	LHI	00 101 110	=	2E
	RL,nn	LLI	00 110 110	=	36

Store Byte Immediate – Immediate to Memory

STBI	M,nn	LMI	00 111 110	=	3E

Bump Register – Increment register by one

BUMP	RB	INB	00 001 000	=	08
	RC	INC	00 010 000	=	10
	RD	IND	00 011 000	=	18
	RE	INE	00 100 000	=	20
	RH	INH	00 101 000	=	28
	RL	INL	00 110 000	=	30

Dec Register – Decrement register by one

DEC	RB	DCB	00 001 001	=	09
	RC	DCC	00 010 001	=	11
	RD	DCD	00 011 001	=	19
	RE	DCE	00 100 001	=	21
	RH	DCH	00 101 001	=	29
	RL	DCL	00 110 001	=	31

ZENTEC ASSEMBLY STATEMENT		MCS-8 EQUIVALENT	MCS-8 BINARY CODES OCTAL FORM	HEXADECIMAL
Add Register — Add value in second register to register A				
AR	RA,Rn	ADA to ADL	10 000 000 to 110 =	80 to 86
Add Register Carry — Add value in second register and carry to register A				
ARC	RA,Rn	ACA to ACL	10 001 000 to 110 =	88 to 8E
Subtract Register — Subtract value in second register from register A				
SR	RA,Rn	SUA to SUL	10 010 000 to 110 =	90 to 96
Subtract Register Carry — Subtract value in second register and carry from register A				
SRC	RA,Rn	SBA to SBL	10 011 000 to 110 =	98 to 9E
AND Register — AND value in second register into register A				
ANDR	RA,Rn	NDA to NDL	10 100 000 to 110 =	A0 to A6
Exclusive OR Register — Exclusive OR value in second register into register A				
XR	RA,Rn	XRA to XRL	10 101 000 to 110 =	A8 to AE
OR Register — OR value in second register into register A				
OR	RA,Rn	ORA to ORL	10 110 000 to 110 =	B0 to B6
Compare Register — Compare value in second register with register A				
CR	RA,Rn	CPA to CPL	10 111 000 to 110 =	B8 to BE
Add — Add value in memory to Register A				
A	RA -	ADM	10 000 111 =	87
Add Carry — Add value in memory and carry to Register A				
AC	RA -	ACM	10 001 111 =	8F
Subtract — Subtract value in memory from Register A				
S	RA -	SUM	10 010 111 =	97
Subtract Carry — Subtract value in memory and carry from Register A				
SC	RA -	SBM	10 011 111 =	9F
AND — AND value in memory into Register A				
AND	RA -	NDM	10 100 111 =	A7
OR — OR value in memory into Register A				
O	RA -	ORM	10 110 111 =	B7
Compare — Compare value in memory with Register A				
C	RA -	CPM	10 111 111 =	BF
Exclusive OR — Exclusive OR value in memory into Register A				
X	RA -	XRM	10 101 111 =	AF

ZENTEC ASSEMBLY STATEMENT		MCS-8 EQUIVALENT	MCS-8 BINARY CODES OCTAL FORM		HEXADECIMAL
Add Immediate – Add immediate to Register A					
AI	RA,nn -	ADI	00 000 100	=	04
Add Immediate Carry – Add immediate and carry to Register A					
ACI	RA,nn -	ACI	00 001 100	=	0C
Subtract Immediate – Subtract immediate from Register A					
SI	RA,nn -	SUI	00 010 100	=	14
Subtract Immediate Carry – Subtract immediate and carry from Register A					
SCI	RA,nn -	SBI	00 011 100	=	1C
AND Immediate – AND immediate into Register A					
ANDI	RA,nn -	NDI	00 100 100	=	24
Exclusive OR Immediate – Exclusive OR immediate into Register A					
XI	RA,nn -	XRI	00 101 100	=	2C
OR Immediate – OR immediate into Register A					
OI	RA,nn -	ORI	00 110 100	=	34
Compare Immediate – Compare immediate with Register A					
CI	RA,nn -	CPI	00 111 100	=	3C
Rotate Left – Rotate Register A to left – Bit 7 goes to Bit 0					
ROL	RA -	RLC	00 000 010	=	02
Rotate Right – Rotate Register A to Right – Bit 0 goes to Bit 7					
ROR	RA -	RRC	00 001 010	=	0A
Rotate Left Carry – Rotate Register A to left – Bit 7 goes to Carry, Carry to Bit 0					
RLC	RA -	RAL	00 010 010	=	12
Rotate Right Carry – Rotate Register A to Right – Bit 0 goes to Carry, Carry to Bit 7					
RRC	RA -	RAR	00 011 010	=	1A

ZENTEC ASSEMBLY STATEMENT		MCS-8 EQUIVALENT	MCS-8 BINARY CODES OCTAL FORM		HEXADECIMAL
Branch – Absolute Branch – System uses X'44' only					
B	label	JMP	01 XXX 100	=	44,4C,54,5C, 64,6C,74,7C
Branch True Carry – Branch if Carry Bit on					
BTC	label	JTC	01 100 000	=	60
Branch True Zero – Branch if Accum Zero or Compare equal					
BTZ	label	JTZ	01 101 000	=	68
Branch True Sign – Branch if Accum Sign is negative					
BTS	label	JTS	01 110 000	=	70 (Sign Negative)
Branch True Parity – Branch if Accum Parity is even					
BTP	label	JTP	01 111 000	=	78 (Parity Even)
Branch False Carry – Branch if Carry bit off					
BFC	label	JFC	01 000 000	=	40
Branch False Zero – Branch if Accum Not Zero or Compare Not Equal					
BFZ	label	JFZ	01 001 000	=	48
Branch False Sign – Branch if Accum Sign is positive					
BFS	label	JFS	01 010 000	=	50 (Sign Positive)
Branch False Parity – Branch if Accum Parity is odd					
BFP	label	JFP	01 011 000	=	58 (Parity Odd)
Branch on Zero – Branch if Accum Zero					
BZ	label	JTZ	01 101 000	=	68
Branch on Plus – Branch if Accum Plus or Zero					
BP	label	JFS	01 010 000	=	50
Branch on Minus – Branch if Accum Minus					
BM	label	JTS	01 110 000	=	70
Branch on Equal – Branch if Accum Equal to Comparand					
BE	label	JTZ	01 010 000	=	68

ZENTEC ASSEMBLY STATEMENT		MCS-8 EQUIVALENT	MCS-8 BINARY CODES OCTAL FORM		HEXADECIMAL
Branch on Low – Branch if Accum Lower than Comparand					
BL	label	JTC	01 100 000	=	60
Branch on Not Zero – Branch if Accum Not Zero					
BNZ	label	JFZ	01 001 000	=	48
Branch on Not Plus – Branch if Accum Minus					
BNP	label	JTS	01 110 000	=	70
Branch on Not Minus – Branch if Accum Plus or Zero					
BNM	label	JFS	01 010 000	=	50
Branch on Not Equal – Branch if Accum Not Equal to Comparand					
BNE	label	JFZ	01 001 000	=	48
Branch on High – Branch if Accum Equal to or Higher than Comparand					
BH	label	JFC	01 000 000	=	40
Call – Absolute Call – System uses X'46' Only					
Call	label	CAL	01 XXX 110	=	46,4E,56,5E, 66,6E,76,7E
Call True Carry – Call if Carry Bit on					
CTC	label	CTC	01 100 010	=	62
Call True Zero – Call if Accum Zero or Compare equal					
CTZ	label	CTZ	01 101 010	=	6A
Call True Sign – Call if Accum Sign is negative					
CTS	label	CTS	01 110 010	=	72 (Sign Negative)
Call True Parity – Call if Accum Parity is even					
CTP	label	CTP	01 111 010	=	7A (Parity Even)
Call False Carry – Call if Carry bit off					
CFC	label	CFC	01 000 010	=	42
Call False Zero – Call if Accum Not Zero or Compare Not Equal					
CFZ	label	CFZ	01 010 010	=	4A
Call False Sign – Call if Accum Sign is positive					
CFS	label	CFS	01 010 010	=	52
Call False Parity – Call if Accum Parity is odd					
CFP	label	CFP	01 011 010	=	5A

ZENTEC ASSEMBLY STATEMENT		MCS-8 EQUIVALENT	MCS-8 BINARY CODES OCTAL FORM		HEXADECIMAL
Call on Zero – Call if Accum Zero					
CZ	label	CTZ	01 101 010	=	6A
Call on Plus – Call if Accum Plus or Zero					
CP	label	CFS	01 010 010	=	52
Call on Minus – Call if Accum Minus					
CM	label	CTS	01 110 010	=	72
Call on Equal – Call if Accum Equal to Comparand					
CE	label	CTZ	01 101 010	=	6A
Call on Low – Call if Accum Lower than Comparand					
CL	label	CTC	01 100 010	=	62
Call Not Zero – Call if Accum Not Zero					
CNZ	label	CFZ	01 001 010	=	4A
Call Not Plus – Call if Accum Minus					
CNP	label	CTS	01 110 010	=	72
Call Not Minus – Call If Accum Plus or Zero					
CNM	label	CFS	01 010 010	=	52
Call Not Equal – Call if Accum Not Equal to Comparand					
CNE	label	CFZ	01 001 010	=	4A
Call on High – Call if Accum Equal to or Higher than Comparand					
CH	label	CFC	01 000 010	=	42
Return – Absolute Return – System Uses X'07' Only					
RET	label	RET	00 XXX 111	=	07,0F,17,1F 27,2F,37,3F
Return True Carry – Return if Carry Bit on					
RTC	label	RTC	00 100 011	=	23
Return True Zero – Return if Accum Zero or Compare equal					
RTZ	label	RTZ	00 101 011	=	2B
Return True Sign – Return if Accum Sign is negative					
RTS	label	RTS	00 110 011	=	33
Return True Parity – Return if Accum Parity is even					
RTP	label	RTP	00 111 011	=	3B
Return False Carry – Return if Carry bit off					
RFC	label	RFC	00 000 011	=	03

ZENTEC ASSEMBLY STATEMENT		MCS-8 EQUIVALENT	MCS-8 BINARY CODES OCTAL FORM		HEXADECIMAL
Return False Zero — Return if Accum Not Zero or Compare Not Equal					
RFZ	label	RFZ	00 001 011	=	0B
Return False Sign — Return if Accum Sign is positive					
RFS	label	RFS	00 010 011	=	13
Return False Parity — Return if Accum Parity is odd					
RFP	label	RFP	00 011 011	=	1B
Return on Zero — Return if Accum Zero					
RZ	label	RTZ	00 101 011	=	2B
Return on Plus — Return if Accum Plus or Zero					
RP	label	RFS	00 010 011	=	13
Return on Minus — Return if Accum Minus					
RM	label	RTS	00 110 011	=	33
Return on Equal — Return if Accum Equal to Comparand					
RE	label	RTZ	00 101 011	=	2B
Return on Low — Return if Accum Lower than Comparand					
RL	label	RTC	00 100 011	=	23
Return Not Zero — Return if Accum Not Zero					
RNZ	label	RFZ	00 001 011	=	0B
Return Not Plus — Return if Accum Minus					
RNP	label	RTS	00 110 011	=	33
Return Not Minus — Return if Accum Plus or Zero					
RNM	label	RFS	00 010 011	=	13
Return Not Equal — Return if Accum Not Equal to Comparand					
RNE	label	RFZ	00 110 011	=	0B
Return on High — Return if Accum Equal to or Higher than Comparand					
RH	label	RFC	00 000 011	=	03

7.2 ZIM (ZENTEC INTERROGATION MODULE) PROGRAM (9002A)

The ZIM provides means for visual access to the contents of ROM, PROM, and RAM memories in the system. The contents of each location in the memory is displayed on the screen in hexadecimal-coded form and various sections of the memory can be moved on or off the screen with the cursor controls. In addition, contents of any memory location in the RAM segment can be altered from the keyboard when operating under the control of ZIM program. Consequently, ZIM program is useful for programming, program debugging, as well as maintenance purposes.

Installation of the ZIM program requires that the page two video display option is present in the system.

The ZIM program is entered from the CONTROL MODE by depressing the ZIM key. The 25th line will display CONTROL. A segment of the memory content will be displayed in hexadecimal in rows across the screen with their address appearing in the left hand column.

```
0F80    C2 0A 0A 0A 0A 24 0F B1 2E 10 36 24 F8 07 46 00
0F90    08 2E 10 36 01 C4 97 40 9B 0F 19 E0 46 0C 08 16
0FA0    50 46 99 09 46 7A 09 46 59 08 09 0B EB F4 16 50
0FB0    C7 3C 20 48 9F 0F 46 34 08 11 48 B0 0F 07 CA 2E
0FC0    10 36 14 F9 44 C5 0D 00 46 8D 0A 1E 10 26 00 2E
0FD0    0F 36 DE 16 22 46 27 09 46 D3 0B 44 16 00 01 00
0FE0    FF 00 00 01 00 00 44 00 00 00 00 00 00 00 02 19
                                                                   CONTROL
```

* The cursor move keys will allow indexing through the memory. The cursor appears as a reverse video character.

* Indexing through memory can also be achieved by keying in the four digit memory address number and then depressing the lower case "l" key. This will cause the cursor to jump to the specified memory address.

* The SPACE BAR can be used to enter hexadecimal code into the RAM memory locations. This is achieved by depressing the appropriate alpha-numeric key (two key sequence) and then depressing the SPACE BAR. The hexadecimal characters will be inserted at the cursor location.

* Depressing the RESET key or branching using the "g" key to location X`0008` will return the 9002 to the normal operating program.

* The 9002 program can be made to BRANCH to a specific MEMORY location by keying in the four digit memory address number and then depressing the "g" key. This will cause the 9002 to BRANCH to the specific memory location and execute the program residing at that location. Depressing the RESET key or branching using the "g" key to location X`0008` will return the 9002 to the normal operating program.

NOTE: The most significant number is entered first when keying in the hexadecimal memory address.

A P P E N D I X   A

KEYBOARD LAYOUT

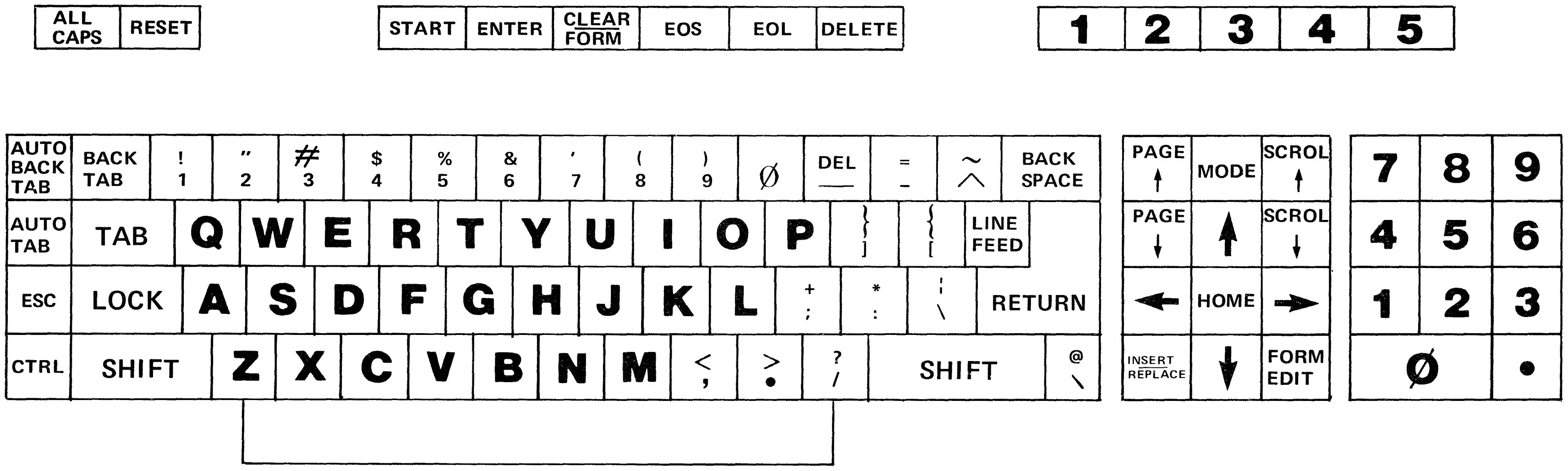
KEYBOARD CODE ASSIGNMENT

CHARACTER GENERATOR CODE ASSIGNMENT

Bit No. 765 / 4321	0 000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8	9	A	B
0 0000	FORM	CLEAR	SP	0	@	P	`	*p*	CTRL @	CTRL P		NP Ø
1 0001	ENTER	1	!	1	A	Q	*a*	*q*	CTRL A	CTRL Q		NP 1
2 0010	DELETE	2	"	2	B	R	*b*	*r*	CTRL B	CTRL R		NP 2
3 0011	5L	3	#	3	C	S	*c*	*s*	CTRL C	CTRL S		NP 3
4 0100	4	RETURN	S	4	D	T	*d*	*t*	CTRL D	CTRL T		NP 4
5 0101	LINE FEED	MODE	%	5	E	U	*e*	*u*	CTRL E	CTRL U		NP 5
6 0110	START	↑ SCROL	&	6	F	V	*f*	*v*	CTRL F	CTRL V		NP 6
7 0111	ERASE END DISP	↓ SCROL	'	7	G	W	*g*	*w*	CTRL G	CTRL W		NP 7
8 1000	← BACK SPACE	⟶	(	8	H	X	*h*	*x*	CTRL H	CTRL X		NP 8
9 1001	TAB	BACK TAB	)	9	I	Y	*i*	*y*	CTRL I	CTRL Y		NP 9
A 1010	↓	↑	*	:	J	Z	*j*	*z*	CTRL J	CTRL Z		
B 1011	HOME	ESC	+	;	K	[	*k*	{	*CTRL K*	*CTRL* [		
C 1100	↑ PAGE	↓ PAGE	,	<	L	\	*l*	¦	CTRL L	CTRL \		
D 1101	AUTO TAB	AUTO BACK TAB	-	=	M	]	m	}	CTRL M	CTRL]		
E 1110	5U	INS REP	.		N	^	*n*	~	CTRL N	CTRL ^	NP •	
F 1111	ERASE END LINE	FORM EDIT	/	?	O	—	*o*	DEL	CTRL O	CTRL –		

* SOFTWARE TREATS NUMERIC PAD KEYS (COLUMN A,B)
AS COLUMN 2 & 3 KEYS
SOFTWARE TREATS CTRL KEYS (COLUMN 8 & 9)

HARDWARE KEYBOARD CODE



TERMINAL KEYBOARD

Table 4-3, Character Generator Chart

4321 \ 765	000	001	010	011	100	101	110	111
0000	□	▬		⊙	@	P	'	p
0001	⊥	⌈	!	1	A	Q	a	q
0010	T	⌋	"	2	B	R	b	r
0011	≥	≤	#	3	C	S	c	s
0100	ω	¨	$	4	D	T	d	t
0101	▢	▢	%	5	E	U	e	u
0110	ϵ	—	&	6	F	V	f	v
0111	ρ	α	'	7	G	W	g	w
1000	⬅	➡	(	8	H	X	h	x
1001	△	▽	)	9	I	Y	i	y
1010	⬇	⬆	*	:	J	Z	j	z
1011	Σ	^	+	;	K	[	k	{
1100	≀	÷	,	<	L	\	l	¦
1101	≠	X	–	=	M	]	m	}
1110	∫	∩	•	>	N	^	n	~
1111	⊂	⊃	/	?	O	—	o	■

A P P E N D I X   B

INTEL 8008

OPERATION AND INSTRUCTION SET

COURTESY - INTEL CORPORATION

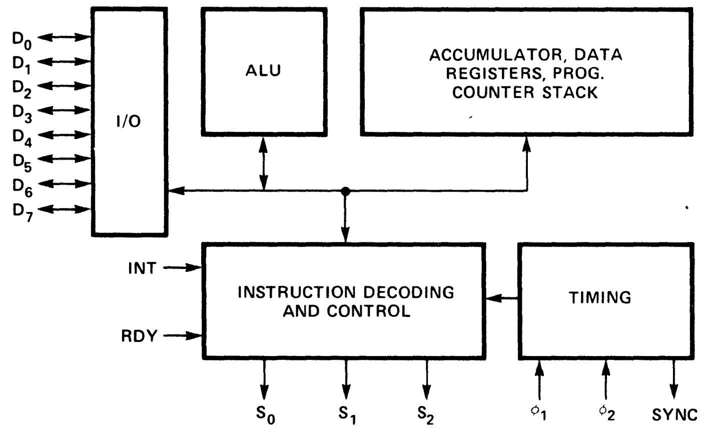
8008
8 Bit Parallel Central Processor Unit

The 8008 is a complete computer system central processor unit which may be interfaced with memories having capacities up to 16K bytes. The processor communicates over an 8-bit data and address bus and uses two leads for internal control and four leads for external control. The CPU contains an 8-bit parallel arithmetic unit, a dynamic RAM (seven 8-bit data registers and an 8x14 stack), and complete instruction decoding and control logic.

Features

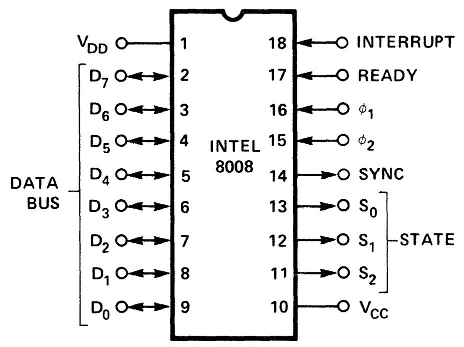
- **8-Bit Parallel CPU on a Single Chip**
- **48 Instructions, Data Oriented**
- **Complete Instruction Decoding and Control Included**
- **Instruction Cycle Time — 12.5 μs with 8008-1 or 20 μs with 8008**
- **TTL Compatible (Inputs, Outputs and Clocks)**
- **Can be used with any type or speed semiconductor memory in any combination**
- **Directly addresses 16K x 8 bits of memory (RAM, ROM, or S.R.)**
- **Memory capacity can be indefinitely expanded through bank switching using I/O instructions**
- **Address stack contains eight 14-bit registers (including program counter) which permit nesting of subroutines up to seven levels**
- **Contains seven 8-bit registers**
- **Interrupt Capability**
- **Packaged in 18-Pin DIP**

BLOCK DIAGRAM



D0
D1
D2
D3
D4
D5
D6
D7
I/O
ALU
ACCUMULATOR, DATA REGISTERS, PROG. COUNTER STACK
INT
RDY
INSTRUCTION DECODING AND CONTROL
TIMING
S0
S1
S2
φ1
φ2
SYNC

PIN CONFIGURATION



I. INTRODUCTION

The 8008 is a single chip MOS 8-bit parallel central processor unit for the MCS-8 micro computer system. A micro computer system is formed when the 8008 is interfaced with any type or speed standard semiconductor memory up to 16K 8-bit words. Examples are INTEL's 1101, 1103, 2102 (RAMs), 1302, 1602A, 1702A (ROMs), 1404, 2405 (Shift Registers).

The processor communicates over an 8-bit data and address bus (D_0 through D_7) and uses two input leads (READY and INTERRUPT) and four output leads (S_0, S_1, S_2 and Sync) for control. Time multiplexing of the data bus allows control information, 14 bit addresses, and data to be transmitted between the CPU and external memory.

This CPU contains six 8-bit data registers, an 8-bit accumulator, two 8-bit temporary registers, four flag bits, and an 8-bit parallel binary arithmetic unit which implements addition, subtraction, and logical operations. A memory stack containing a 14-bit program counter and seven 14-bit words is used internally to store program and subroutine addresses. The 14-bit address permits the direct addressing of 16K words of memory (any mix of RAM, ROM or S.R.).

The control portion of the chip contains logic to implement a variety of register transfer, arithmetic control, and logical instructions. Most instructions are coded in one byte (8 bits); data immediate instructions use two bytes; jump instructions utilize three bytes. Operating with a 500kHz clock, the 8008 CPU executes non-memory referencing instructions in 20 microseconds. A selected device, the 8008-1, executes non-memory referencing instructions in 12.5 microseconds when operating from an 800kHz clock.

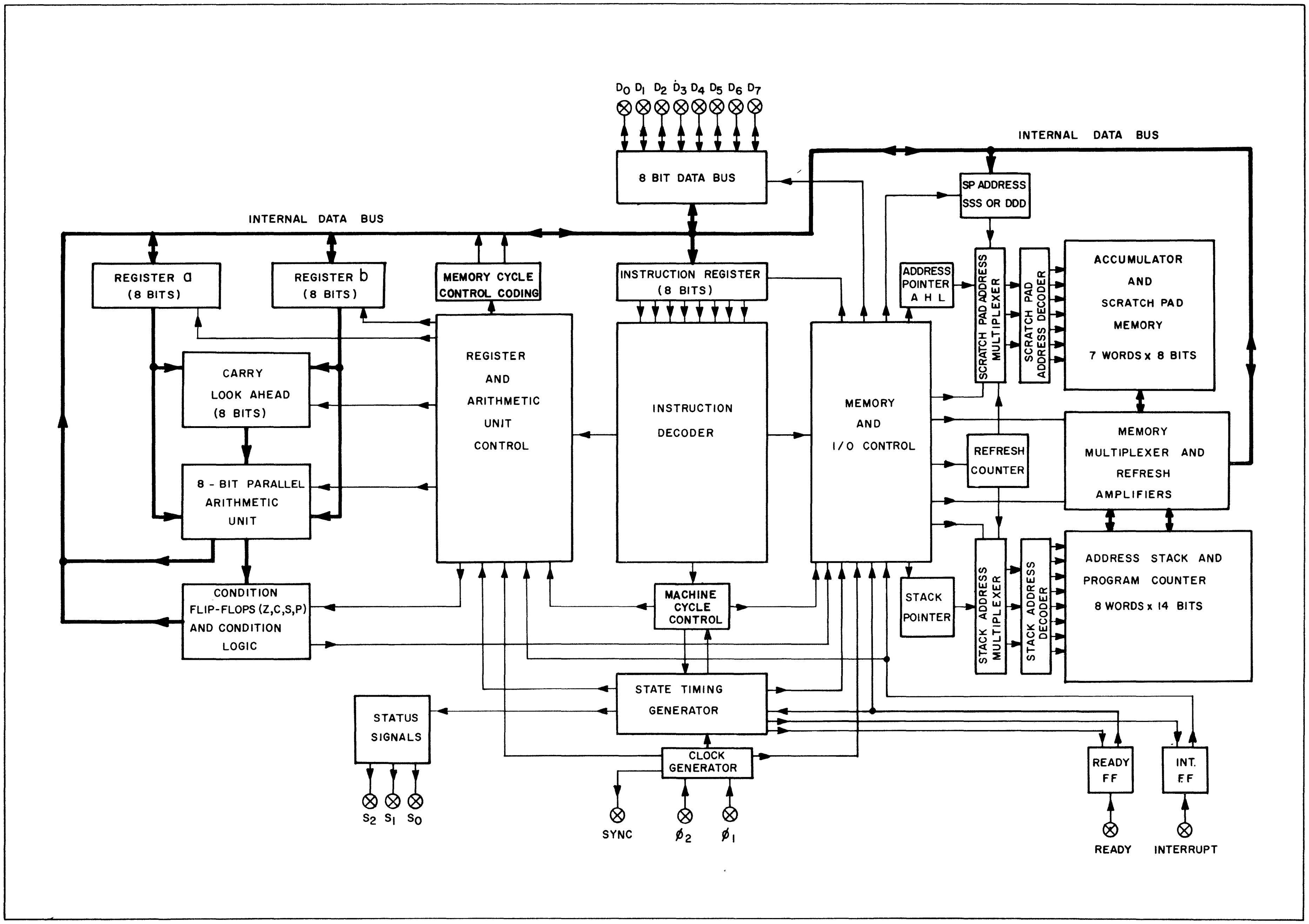
All inputs (including clocks) are TTL compatible and all outputs are low-power TTL compatible.

The instruction set of the 8008 consists of 48 instructions including data manipulation, binary arithmetic, and jump to subroutine.

The normal program flow of the 8008 may be interrupted through the use of the "INTERRUPT" control line. This allows the servicing of slow I/O peripheral devices while also executing the main program.

The "READY" command line synchronizes the 8008 to the memory cycle allowing any type or speed of semiconductor memory to be used.

STATE and SYNC outputs indicate the state of the processor at any time in the instruction cycle.



D_0 D_1 D_2 D_3 D_4 D_5 D_6 D_7
INTERNAL DATA BUS
8 BIT DATA BUS
INTERNAL DATA BUS
SP ADDRESS SSS OR DDD
REGISTER a (8 BITS)
REGISTER b (8 BITS)
MEMORY CYCLE CONTROL CODING
INSTRUCTION REGISTER (8 BITS)
ADDRESS POINTER A H L
SCRATCH PAD ADDRESS MULTIPLEXER
SCRATCH PAD ADDRESS DECODER
ACCUMULATOR AND SCRATCH PAD MEMORY 7 WORDS x 8 BITS
CARRY LOOK AHEAD (8 BITS)
REGISTER AND ARITHMETIC UNIT CONTROL
INSTRUCTION DECODER
MEMORY AND I/O CONTROL
REFRESH COUNTER
MEMORY MULTIPLEXER AND REFRESH AMPLIFIERS
8 - BIT PARALLEL ARITHMETIC UNIT
CONDITION FLIP-FLOPS (Z,C,S,P) AND CONDITION LOGIC
MACHINE CYCLE CONTROL
STACK POINTER
STACK ADDRESS MULTIPLEXER
STACK ADDRESS DECODER
ADDRESS STACK AND PROGRAM COUNTER 8 WORDS x 14 BITS
STATE TIMING GENERATOR
STATUS SIGNALS
CLOCK GENERATOR
READY FF
INT. FF
S_2 S_1 S_0
SYNC
ϕ_2
ϕ_1
READY
INTERRUPT

Figure 3. 8008 Block Diagram

III. BASIC FUNCTIONAL BLOCKS

The four basic functional blocks of this Intel processor are the instruction register, memory, arithmetic-logic unit, and I/O buffers. They communicate with each other over the internal 8-bit data bus.

A. Instruction Register and Control

The instruction register is the heart of all processor control. Instructions are fetched from memory, stored in the instruction register, and decoded for control of both the memories and the ALU. Since instruction executions do not all require the same number of states, the instruction decoder also controls the state transitions.

B. Memory

Two separate dynamic memories are used in the 8008, the pushdown address stack and a scratch pad. These internal memories are automatically refreshed by each WAIT, T3, and STOPPED state. In the worst case the memories are completely refreshed every eighty clock periods.

1. Address Stack

   The address stack contains eight 14-bit registers providing storage for eight lower and six higher order address bits in each register. One register is used as the program counter (storing the effective address) and the other seven permit address storage for nesting of subroutines up to seven levels. The stack automatically stores the content of the program counter upon the execution of a CALL instruction and automatically restores the program counter upon the execution of a RETURN. The CALLs may be nested and the registers of the stack are used as last in/first out pushdown stack. A three-bit address pointer is used to designate the present location of the program counter. When the capacity of the stack is exceeded the address pointer recycles and the content of the lowest level register is destroyed. The program counter is incremented immediately after the lower order address bits are sent out. The higher order address bits are sent out at T2 and then incremented if a carry resulted from T1. The 14-bit program counter provides direct addressing of 16K bytes of memory. Through the use of an I/O instruction for bank switching, memory may be indefinitely expanded.

2. Scratch Pad Memory or Index Registers

   The scratch pad contains the accumulator (A register) and six additional 8-bit registers (B, C, D, E, H, L). All arithmetic operations use the accumulator as one of the operands. All registers are independent and may be used for temporary storage. In the case of instructions which require operations with a register in external memory, scratch pad registers H & L provide indirect addressing capability; register L contains the eight lower order bits of address and register H contains the six higher order bits of address (in this case bit 6 and bit 7 are "don't cares").

C. Arithmetic/Logic Unit (ALU)

All arithmetic and logical operations (ADD, ADD with carry, SUBTRACT, SUBTRACT with borrow, AND, EXCLUSIVE OR, OR, COMPARE, INCREMENT, DECREMENT) are carried out in the 8-bit parallel arithmetic unit which includes carry-look-ahead logic. Two temporary resisters, register "a" and register "b", are used to store the accumulator and operand for ALU operations. In addition, they are used for temporary address and data storage during intra-processor transfers. Four control bits, carry flip-flop (c) , zero flip-flop (z) , sign flip-flop (s) , and parity flip-flop (p) , are set as the result of each arithmetic and logical operation. These bits provide conditional branching capability through CALL, JUMP, or RETURN on condition instructions. In addition, the carry bit provides the ability to do multiple precision binary arithmetic.

D. I/O Buffer

This buffer is the only link between the processor and the rest of the system. Each of the eight buffers is bi-directional and is under control of the instruction register and state timing. Each of the buffers is low power TTL compatible on the output and TTL compatible on the input.

IV. BASIC INSTRUCTION SET

The following section presents the basic instruction set of the 8008.

A. Data and Instruction Formats

Data in the 8008 is stored in the form of 8-bit binary integers. All data transfers to the system data bus will be in the same format.

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

DATA WORD

The program instructions may be one, two, or three bytes in length. Multiple byte instructions must be stored in successive words in program memory. The instruction formats then depend on the particular operation executed.

		TYPICAL INSTRUCTIONS
One Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	Register to register, memory reference, I/O arithmetic or logical, rotate or return instructions
Two Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OPERAND	Immediate mode instructions
Three Byte Instructions		
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	OP CODE	
D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0	LOW ADDRESS	JUMP or CALL instructions
X X D_5 D_4 D_3 D_2 D_1 D_0	HIGH ADDRESS*	

*For the third byte of this instruction, D_6 and D_7 are "don't care" bits.

For the MCS-8 a logic "1" is defined as a high level and a logic "0" is defined as a low level.

B. Summary of Processor Instructions

Index Register Instructions

The load instructions do not affect the flag flip-flops. The increment and decrement instructions affect all flip-flops except the carry.

MNEMONIC	MINIMUM STATES REQUIRED	INSTRUCTION CODE D_7 D_6	D_5 D_4 D_3	D_2 D_1 D_0	DESCRIPTION OF OPERATION
(1) Lr_1r_2	(5)	1 1	D D D	S S S	Load index register r_1 with the content of index register r_2.
(2) LrM	(8)	1 1	D D D	1 1 1	Load index register r with the content of memory register M.
LMr	(7)	1 1	1 1 1	S S S	Load memory register M with the content of index register r.
(3) LrI	(8)	0 0 B B	D D D B B B	1 1 0 B B B	Load index register r with data B . . . B.
LMI	(9)	0 0 B B	1 1 1 B B B	1 1 0 B B B	Load memory register M with data B . . . B.
INr	(5)	0 0	D D D	0 0 0	Increment the content of index register r (r ≠ A).
DCr	(5)	0 0	D D D	0 0 1	Decrement the content of index register r (r ≠ A).

Accumulator Group Instructions

The result of the ALU instructions affect all of the flag flip-flops. The rotate instructions affect only the carry flip-flop.

ADr	(5)	1 0	0 0 0	S S S	Add the content of index register r, memory register M, or data B . . . B to the accumulator. An overflow (carry) sets the carry flip-flop.
ADM	(8)	1 0	0 0 0	1 1 1	
ADI	(8)	0 0 B B	0 0 0 B B B	1 0 0 B B B	
ACr	(5)	1 0	0 0 1	S S S	Add the content of index register r, memory register M, or data B . . . B to the accumulator with carry. An overflow (carry) sets the carry flip-flop.
ACM	(8)	1 0	0 0 1	1 1 1	
ACI	(8)	0 0 B B	0 0 1 B B B	1 0 0 B B B	
SUr	(5)	1 0	0 1 0	S S S	Subtract the content of index register r, memory register M, or data B . . . B from the accumulator. An underflow (borrow) sets the carry flip-flop.
SUM	(8)	1 0	0 1 0	1 1 1	
SUI	(8)	0 0 B B	0 1 0 B B B	1 0 0 B B B	
SBr	(5)	1 0	0 1 1	S S S	Subtract the content of index register r, memory register M, or data B . . . B from the accumulator with borrow. An underflow (borrow) sets the carry flip-flop.
SBM	(8)	1 0	0 1 1	1 1 1	
SBI	(8)	0 0 B B	0 1 1 B B B	1 0 0 B B B	

MNEMONIC	MINIMUM STATES REQUIRED	INSTRUCTION CODE $D_7 D_6$	$D_5 D_4 D_3$	$D_2 D_1 D_0$	DESCRIPTION OF OPERATION
NDr	(5)	1 0	1 0 0	S S S	Compute the logical AND of the content of index register r, memory register M, or data B . . . B with the accumulator.
NDM	(8)	1 0	1 0 0	1 1 1	
NDI	(8)	0 0 B B	1 0 0 B B B	1 0 0 B B B	
XRr	(5)	1 0	1 0 1	S S S	Compute the EXCLUSIVE OR of the content of index register r, memory register M, or data B . . . B with the accumulator.
XRM	(8)	1 0	1 0 1	1 1 1	
XRI	(8)	0 0 B B	1 0 1 B B B	1 0 0 B B B	
ORr	(5)	1 0	1 1 0	S S S	Compute the INCLUSIVE OR of the content of index register r, memory register m, or data B . . . B with the accumulator.
ORM	(8)	1 0	1 1 0	1 1 1	
ORI	(8)	0 0 B B	1 1 0 B B B	1 0 0 B B B	
CPr	(5)	1 0	1 1 1	S S S	Compare the content of index register r, memory register M, or data B . . . B with the accumulator. The content of the accumulator is unchanged.
CPM	(8)	1 0	1 1 1	1 1 1	
CPI	(8)	0 0 B B	1 1 1 B B B	1 0 0 B B B	
RLC	(5)	0 0	0 0 0	0 1 0	Rotate the content of the accumulator left.
RRC	(5)	0 0	0 0 1	0 1 0	Rotate the content of the accumulator right.
RAL	(5)	0 0	0 1 0	0 1 0	Rotate the content of the accumulator left through the carry.
RAR	(5)	0 0	0 1 1	0 1 0	Rotate the content of the accumulator right through the carry.

Program Counter and Stack Control Instructions

(4) JMP	(11)	0 1 B_2 B_2 X X	X X X B_2 B_2 B_2 B_3 B_3 B_3	1 0 0 B_2 B_2 B_2 B_3 B_3 B_3	Unconditionally jump to memory address $B_3 \ldots B_3B_2 \ldots B_2$.
(5) JFc	(9 or 11)	0 1 B_2 B_2 X X	0 C_4 C_3 B_2 B_2 B_2 B_3 B_3 B_3	0 0 0 B_2 B_2 B_2 B_3 B_3 B_3	Jump to memory address $B_3 \ldots B_3B_2 \ldots B_2$ if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence.
JTc	(9 or 11)	0 1 B_2 B_2 X X	1 C_4 C_3 B_2 B_2 B_2 B_3 B_3 B_3	0 0 0 B_2 B_2 B_2 B_3 B_3 B_3	Jump to memory address $B_3 \ldots B_3B_2 \ldots B_2$ if the condition flip-flop c is true. Otherwise, execute the next instruction in sequence.
CAL	(11)	0 1 B_2 B_2 X X	X X X B_2 B_2 B_2 B_3 B_3 B_3	1 1 0 B_2 B_2 B_2 B_3 B_3 B_3	Unconditionally call the subroutine at memory address $B_3 \ldots B_3B_2 \ldots B_2$. Save the current address (up one level in the stack).
CFc	(9 or 11)	0 1 B_2 B_2 X X	0 C_4 C_3 B_2 B_2 B_2 B_3 B_3 B_3	0 1 0 B_2 B_2 B_2 B_3 B_3 B_3	Call the subroutine at memory address $B_3 \ldots B_3B_2 \ldots B_2$ if the condition flip-flop c is false, and save the current address (up one level in the stack.) Otherwise, execute the next instruction in sequence.
CTc	(9 or 11)	0 1 B_2 B_2 X X	1 C_4 C_3 B_2 B_2 B_2 B_3 B_3 B_3	0 1 0 B_2 B_2 B_2 B_3 B_3 B_3	Call the subroutine at memory address $B_3 \ldots B_3B_2 \ldots B_2$ if the condition flip-flop c is true, and save the current address (up one level in the stack). Otherwise, execute the next instruction in sequence.
RET	(5)	0 0	X X X	1 1 1	Unconditionally return (down one level in the stack).
RFc	(3 or 5)	0 0	0 C_4 C_3	0 1 1	Return (down one level in the stack) if the condition flip-flop c is false. Otherwise, execute the next instruction in sequence.
RTc	(3 or 5)	0 0	1 C_4 C_3	0 1 1	Return (down one level in the stack) if the condition flip-flop c is true. Otherwise, execute the next instruction in sequence.
RST	(5)	0 0	A A A	1 0 1	Call tha subroutine at memory address AAA000 (up one level in the stack).

Input/Output Instructions

INP	(8)	0 1	0 0 M	M M 1	Read the content of the selected input port (MMM) into the accumulator.
OUT	(6)	0 1	R R M	M M 1	Write the content of the accumulator into the selected output port (RRMMM, RR ≠ 00).

Machine Instruction

HLT	(4)	0 0	0 0 0	0 0 X	Enter the STOPPED state and remain there until interrupted.
HLT	(4)	1 1	1 1 1	1 1 1	Enter the STOPPED state and remain there until interrupted.

NOTES:

(1) SSS = Source Index Register
DDD = Destination Index Register } These registers, r_i, are designated A(accumulator—000), B(001), C(010), D(011), E(100), H(101), L(110).

(2) Memory registers are addressed by the contents of registers H & L.

(3) Additional bytes of instruction are designated by BBBBBBBB.

(4) X = "Don't Care".

(5) Flag flip-flops are defined by C_4C_3: carry (00-overflow or underflow), zero (01-result is zero), sign (10-MSB of result is "1"), parity (11-parity is even).

C. Complete Functional Definition

The following pages present a detailed description of the complete 8008 Instruction Set.

Symbols	Meaning
<B2>	Second byte of the instruction
<B3>	Third byte of the instruction
r	One of the scratch pad register references: A, B, C, D, E, H, L
c	One of the following flag flip-flop references: C, Z, S, P
C_4C_3	Flag flip-flop codes / Condition for True
	00 carry — Overflow, underflow
	01 zero — Result is zero
	10 sign — MSB of result is "1"
	11 parity — Parity of result is even
M	Memory location indicated by the contents of registers H and L
( )	Contents of location or register
$\wedge$	Logical product
$\forall$	Exclusive "or"
V	Inclusive "or"
A_m	Bit m of the A-register
STACK	Instruction counter (P) pushdown register
P	Program Counter
$\leftarrow$	Is transferred to
XXX	A "don't care"
SSS	Source register for data
DDD	Destination register for data

Register # (SSS or DDD)	Register Name
000	A
001	B
010	C
011	D
100	E
101	H
110	L

INDEX REGISTER INSTRUCTIONS

LOAD DATA TO INDEX REGISTERS – One Byte

Data may be loaded into or moved between any of the index registers, or memory registers.

Lr_1r_2 (one cycle – PCI)	11 DDD SSS	$(r_1) \leftarrow (r_2)$ Load register r_1 with the content of r_2. The content of r_2 remains unchanged. If SSS=DDD, the instruction is a NOP (no operation).
LrM (two cycles – PCI/PCR)	11 DDD 111	$(r) \leftarrow (M)$ Load register r with the content of the memory location addressed by the contents of registers H and L. (DDD≠111 – HALT instr.)
LMr (two cycles – PCI/PCW)	11 111 SSS	$(M) \leftarrow (r)$ Load the memory location addressed by the contents of registers H and L with the content of register r. (SSS≠111 – HALT instr.)

LOAD DATA IMMEDIATE – Two Bytes

A byte of data immediately following the instruction may be loaded into the processor or into the memory

LrI (two cycles – PCI/PCR)	00 DDD 110 $<B_2>$	$(r) \leftarrow <B_2>$ Load byte two of the instruction into register r.
LMI (three cycles – PCI/PCR/PCW)	00 111 110 $<B_2>$	$(M) \leftarrow <B_2>$ Load byte two of the instruction into the memory location addressed by the contents of registers H and L.

INCREMENT INDEX REGISTER – One Byte

INr (one cycle – PCI)	00 DDD 000	$(r) \leftarrow (r)+1$. The content of register r is incremented by one. All of the condition flip-flops except carry are affected by the result. Note that DDD≠000 (HALT instr.) and DDD≠111 (content of memory may not be incremented).

DECREMENT INDEX REGISTER – One Byte

DCr (one cycle – PCI)	00 DDD 001	$(r) \leftarrow (r)-1$. The content of register r is decremented by one. All of the condition flip-flops except carry are affected by the result. Note that DDD≠000 (HALT instr.) and DDD≠111 (content of memory may not be decremented).

ACCUMULATOR GROUP INSTRUCTIONS

Operations are performed and the status flip-flops, C, Z, S, P, are set based on the result of the operation. Logical operations (NDr, XRr, ORr) set the carry flip-flop to zero. Rotate operations affect only the carry flip-flop. Two's complement subtraction is used.

ALU INDEX REGISTER INSTRUCTIONS – One Byte

(one cycle – PCI)

Index Register operations are carried out between the accumulator and the content of one of the index registers (SSS=000 thru SSS=110). The previous content of register SSS is unchanged by the operation.

ADr	10 000 SSS	$(A) \leftarrow (A)+(r)$ Add the content of register r to the content of register A and place the result into register A.
ACr	10 001 SSS	$(A) \leftarrow (A)+(r)+(carry)$ Add the content of register r and the contents of the carry flip-flop to the content of the A register and place the result into Register A.
SUr	10 010 SSS	$(A) \leftarrow (A)-(r)$ Subtract the content of register r from the content of register A and place the result into register A. Two's complement subtraction is used.

ACCUMULATOR GROUP INSTRUCTIONS - Cont'd.

SBr	10	011	SSS	$(A) \leftarrow (A) - (r) - (\text{borrow})$ Subtract the content of register r and the content of the carry flip-flop from the content of register A and place the result into register A.
NDr	10	100	SSS	$(A) \leftarrow (A) \wedge (r)$ Place the logical product of the register A and register r into register A.
XRr	10	101	SSS	$(A) \leftarrow (A) \forall (r)$ Place the "exclusive - or" of the content of register A and register r into register A.
ORr	10	110	SSS	$(A) \leftarrow (A) \vee (r)$ Place the "inclusive - or" of the content of register A and register r into register A.
CPr	10	111	SSS	$(A) - (r)$ Compare the content of register A with the content of register r. The content of register A remains unchanged. The flag flip-flops are set by the result of the subtraction. Equality (A=r) is indicated by the zero flip-flop set to "1". Less than (A<r) is indicated by the carry flip-flop, set to "1".

ALU OPERATIONS WITH MEMORY – One Byte
(two cycles – PCI/PCR)
Arithmetic and logical operations are carried out between the accumulator and the byte of data addressed by the contents of registers H and L.

ADM	10	000	111	$(A) \leftarrow (A) + (M)$ ADD
ACM	10	001	111	$(A) \leftarrow (A) + (M) + (\text{carry})$ ADD with carry
SUM	10	010	111	$(A) \leftarrow (A) - (M)$ SUBTRACT
SBM	10	011	111	$(A) \leftarrow (A) - (M) - (\text{borrow})$ SUBTRACT with borrow
NDM	10	100	111	$(A) \leftarrow (A) \wedge (M)$ Logical AND
XRM	10	101	111	$(A) \leftarrow (A) \forall (M)$ Exclusive OR
ORM	10	110	111	$(A) \leftarrow (A) \vee (M)$ Inclusive OR
CPM	10	111	111	$(A) - (M)$ COMPARE

ALU IMMEDIATE INSTRUCTIONS – Two Bytes
(two cycles –PCI/PCR)
Arithmetic and logical operations are carried out between the accumulator and the byte of data immediately following the instruction.

ADI	00	000 $<B_2>$	100	$(A) \leftarrow (A) + <B_2>$ ADD
ACI	00	001 $<B_2>$	100	$(A) \leftarrow (A) + <B_2> + (\text{carry})$ ADD with carry
SUI	00	010 $<B_2>$	100	$(A) \leftarrow (A) - <B_2>$ SUBTRACT
SBI	00	011 $<B_2>$	100	$(A) \leftarrow (A) - <B_2> - (\text{borrow})$ SUBTRACT with borrow
NDI	00	100 $<B_2>$	100	$(A) \leftarrow (A) \wedge <B_2>$ Logical AND
XRI	00	101 $<B_2>$	100	$(A) \leftarrow (A) \forall <B_2>$ Exclusive OR
ORI	00	110 $<B_2>$	100	$(A) \leftarrow (A) \vee <B_2>$ Inclusive OR
CPI	00	111 $<B_2>$	100	$(A) - <B_2>$ COMPARE

ROTATE INSTRUCTIONS – One Byte

(one cycle – PCI)

The accumulator content (register A) may be rotated either right or left, around the carry bit or through the carry bit. Only the carry flip-flop is affected by these instructions; the other flags are unchanged.

RLC	00 000 010	$A_{m+1} \leftarrow A_m$, $A_0 \leftarrow A_7$, $(carry) \leftarrow A_7$ Rotate the content of register A left one bit. Rotate A_7 into A_0 and into the carry flip-flop.
RRC	00 001 010	$A_m \leftarrow A_{m+1}$, $A_7 \leftarrow A_0$, $(carry) \leftarrow A_0$ Rotate the content of register A right one bit. Rotate A_0 into A_7 and into the carry flip-flop.
RAL	00 010 010	$A_{m+1} \leftarrow A_m$,$A_0 \leftarrow (carry)$,$(carry) \leftarrow A_7$ Rotate the content of Register A left one bit. Rotate the content of the carry flip-flop into A_0. Rotate A_7 into the carry flip-flop.
RAR	00 011 010	$A_m \leftarrow A_{m+1}$,$A_7 \leftarrow (carry)$, $(carry) \leftarrow A_0$ Rotate the content of register A right one bit. Rotate the content of the carry flip-flop into A_7. Rotate A_0 into the carry flip-flop.

PROGRAM COUNTER AND STACK CONTROL INSTRUCTIONS

JUMP INSTRUCTIONS – Three Bytes

(three cycles – PCI/PCR/PCR)

Normal flow of the microprogram may be altered by jumping to an address specified by bytes two and three of an instruction.

JMP (Jump Unconditionally)	01 XXX 100 $<B_2>$ $<B_3>$	$(P) \leftarrow <B_3><B_2>$ Jump unconditionally to the instruction located in memory location addressed by byte two and byte three.
JFc (Jump if Condition False)	01 $0C_4C_3$ 000 $<B_2>$ $<B_3>$	If (c) = 0, $(P) \leftarrow <B_3><B_2>$. Otherwise, (P) = (P)+3. If the content of flip-flop c is zero, then jump to the instruction located in memory location $<B_3><B_2>$ otherwise, execute the next instruction in sequence.
JTc (Jump if Condition True)	01 $1C_4C_3$ 000 $<B_2>$ $<B_3>$	If (c) = 1, $(P) \leftarrow <B_3><B_2>$. Otherwise, (P) = (P)+3. If the content of flip-flop c is one, then jump to the instruction located in memory location $<B_3><B_2>$; otherwise, execute the next instruction in sequence.

CALL INSTRUCTIONS – Three Bytes

(three cycles – PCI/PCR/PCR)

Subroutines may be called and nested up to seven levels.

CAL (Call subroutine Unconditionally)	01 XXX 110 $<B_2>$ $<B_3>$	$(Stack) \leftarrow (P)$, $(P) \leftarrow <B_3><B_2>$. Shift the content of P to the pushdown stack. Jump unconditionally to the instruction located in memory location addressed by byte two and byte three.
CFc (Call subroutine if Condition False)	01 $0C_4C_3$ 010 $<B_2>$ $<B_3>$	If (c) = 0, $(Stack) \leftarrow (P)$, $(P) \leftarrow <B_3><B_2>$. Otherwise, (P) = (P)+3. If the content of flip-flop c is zero, then shift contents of P to the pushdown stack and jump to the instruction located in memory location$<B_3><B_2>$ otherwise, execute the next instruction in sequence.
CTc (Call subroutine if Condition True)	01 $1C_4C_3$ 010 $<B_2>$ $<B_3>$	If (c) = 1, $(Stack) \leftarrow (P)$, $(P) \leftarrow <B_3><B_2>$. Otherwise, (P) = (P)+3. If the content of flip-flop c is one, then shift contents of P to the pushdown stack and jump to the instruction located in memory location$<B_3><B_2>$; otherwise, execute the next instruction in sequence.

In the above JUMP and CALL instructions $<B_2>$ contains the least significant half of the address and $<B_3>$ contains the most significant half of the address. Note that D_6 and D_7 of$<B_3>$are "don't care" bits since the CPU uses fourteen bits of address.

RETURN INSTRUCTIONS – One Byte
(one cycle – PCI)
A return instruction may be used to exit from a subroutine; the stack is popped-up one level at a time.

RET	00	XXX	111	(P)←(Stack). Return to the instruction in the memory location addressed by the last value shifted into the pushdown stack. The stack pops up one level.
RFc (Return Condition False)	00	$0C_4C_3$	011	If (c) = 0, (P)←(Stack); otherwise, (P) = (P)+1. If the content of flip-flop c is zero, then return to the instruction in the memory location addressed by the last value inserted in the pushdown stack. The stack pops up one level. Otherwise, execute the next instruction in sequence.
RTc (Return Condition True)	00	$1C_4C_3$	011	If (c) = 1, (P)←(Stack); otherwise, (P) = (P)+1. If the content of flip-flop c is one, then return to the instruction in the memory location addressed by the last value inserted in the pushdown stack. The stack pops up one level. Otherwise, execute the next instruction in sequence.

RESTART INSTRUCTION – One Byte
(one cycle – PCI)
The restart instruction acts as a one byte call on eight specified locations of page 0, the first 256 instruction words.

RST	00	AAA	101	(Stack)←(P),(P)←(000000 00AAA000) Shift the contents of P to the pushdown stack. The content, AAA, of the instruction register is shifted into bits 3 through 5 of the P-counter. All other bits of the P-counter are set to zero. As a one-word "call", eight eight-byte subroutines may be accessed in the lower 64 words of memory.

INPUT/OUTPUT INSTRUCTIONS

One Byte
(two cycles – PCI/PCC)
Eight input devices may be referenced by the input instruction

INP	01	00M	MM1	(A)←(input data lines). The content of register A is made available to external equipment at state T1 of the PCC cycle. The content of the instruction register is made available to external equipment at state T2 of the PCC cycle. New data for the accumulator is loaded at T3 of the PCC cycle. MMM denotes input device number. The content of the condition flip-flops, S,Z,P,C, is output on D_0, D_1, D_2, D_3 respectively at T4 on the PCC cycle.

Twenty-four output devices may be referenced by the output instruction.

OUT	01	RRM	MM1	(Output data lines)←(A). The content of register A is made available to external equipment at state T1 and the content of the instruction register is made available to external equipment at state T2 of the PCC cycle. RRMMM denotes output device number (RR ≠ 00).

MACHINE INSTRUCTION

HALT INSTRUCTION – One Byte
(one cycle – PCI)

HLT	00 11	000 or 111	00X 111	On receipt of the Halt Instruction, the activity of the processor is immediately suspended in the STOPPED state. The content of all registers and memory is unchanged. The P-counter has been updated and the internal dynamic memories continue to be refreshed.