ANHANG D-1

FESTLEGUNG DES ZEICHENVORRATS

Mit Hilfe eines auf einer Diskette gespeicherten Programms kann sich der Benutzer selbst einen geeigneten Zeichenvorrat zusammenstellen, in welchen er auch auf dem Bildschirm entworfene Sonderzeichen aufnimmt. Der so entstandene neue Zeichenvorrat wird ebenfalls auf einer Diskette gespeichert.

1. Laden Sie die Diskette mit dem FONTPREP-Programm in das Diskette-Laufwerk Nr. 0.

2. Laden Sie eine DOS/08 Diskette in das Diskette-Laufwerk Nr. 1.

3. Drücken Sie gleichzeitig die Tasten PROG, CTRL und SHIFT, damit DOS/08 in den Speicher kommt.

4. Auf dem Bildschirm erscheint DOS VERSION 2.0

5. Tippen Sie FONTPREP und drücken Sie die Taste RETURN.

6. Mittlerweile gelangt Programm FONTPREP in den Speicher, und auf dem Bildschirm erscheint der Dialog-Fragebogen.

7. Tippen Sie den Namen der Datei und setzen Sie ein X bei der gewünschten Funktion ein. Drücken Sie die Taste ENTER, damit Ihre Anforderung bearbeitet wird.

8. Wird ein CREATE (Neu-Eröffnung einer Datei) gewünscht, dann müssen Sie die geplante Größe des Zeichenvorrats spezifizieren (X bei 128 oder 256).

9. Sobald das Programm die Datei auf der Diskette Nr. 1 eröffnet hat, verschwindet der Fragebogen vom Bildschirm.

10. Geben Sie denselben Dateinamen ein und wählen Sie die EXAMINE-Funktion aus.

11. In dem nun erscheinenden Fragebogen können Sie die Punktmatrix für das erste Zeichen festlegen, indem Sie an den gewünschten Stellen ein X setzen. Mit TAB können Sie die Schreibmarke auf die nächste Zeile rücken.

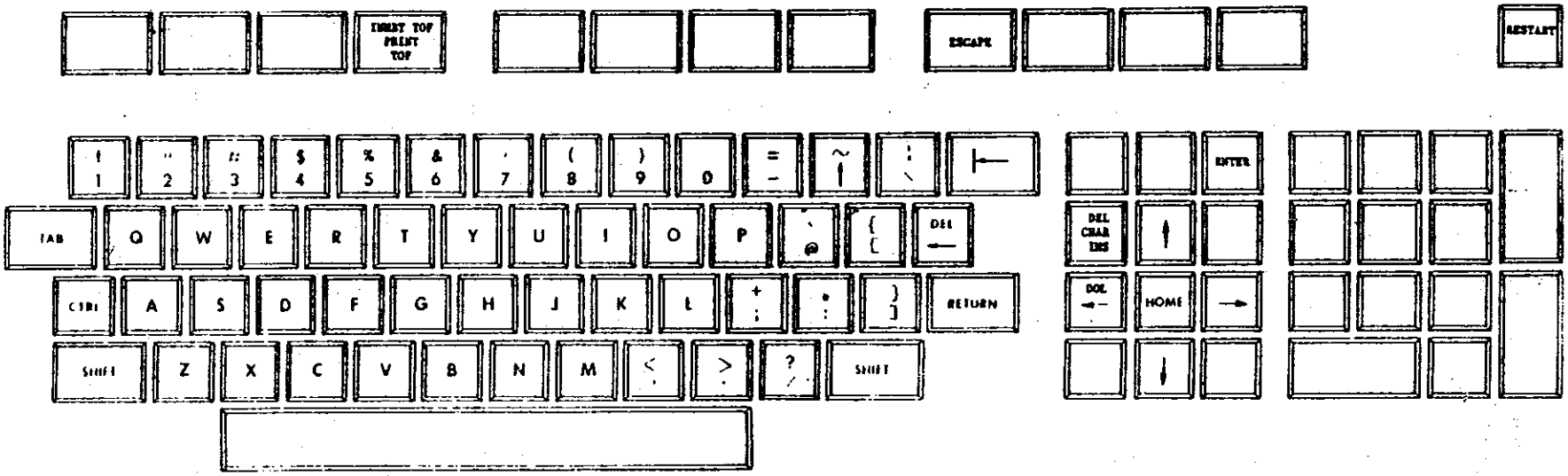
12. Haben Sie das Zeichen festgelegt, dann drücken Sie ENTER, so daß es auf die Diskette gelangt. Nun können Sie das nächste Zeichen „schreiben".

13. Diese Prozedur wird wiederholt, bis der ganze Zeichenvorrat festgelegt ist.

    Anmerkung:

    Bei der Festlegung einer Abtastzeile kann nur entweder ein gerades oder ungerades Bit ausgewählt werden. Sollte der Benutzer irrtümlich sowohl gerade als auch ungerade ankreuzen (x), dann verhindert das Programm diese Doppel-Spezifikation, indem es die Zeile nicht darstellt. Der Hex-Code einer jeden Abtastzeile wird automatisch erzeugt und auf dem Bildschirm dargestellt (der Benutzer braucht sich nicht darum zu kümmern).

14. Die Diskette mit der Beschreibung des Zeichenvorrats wird bei AEG-TELEFUNKEN/ULM weiterverarbeitet (der Zeichenvorrat wird dort in ein PROM eingebrannt).


INSERT TOP
PRINT TOP
ESCAPE
RESTART
TAB
Q W E R T Y U I O P
DEL
CTRL
A S D F G H J K L
RETURN
SHIFT
Z X C V B N M
SHIFT
ENTER
DEL CHAR INS
DOL
HOME

ANHANG D-2

CODIERUNG DES HERVORHEBUNGS-PROM

Es handelt sich hier um ein 512 X 4 PROM. Die ersten 256 Adressen (Hex 000 bis 0FF) dienen im Schnell-Lösch-Modus zur Auswahl der Zeichen für LEOL und FLEOL. Die zweiten 256 Adressen dienen im Hervorhebungs-Modus der Auswahl der unterdrückbaren Zeichen sowie der Zeichen für die START-und STOP-Begrenzer für die Attribute 1 bis 3. Der Hex-Code des ausgewählten Zeichens wird als niederigerwertige 8 Bits der PROM-Adresse verwendet.

Modus 1

Funktion	PROM-Adresse	PROM-Code (Hex)
LEOL	0XX	A
FLEOL	0XX	C

Modus 2 (DISPLY2 Bit 0 = 0)

Funktion	PROM-Adresse	PROM-Code (Hex)
START Attribut 1	1XX	9
STOP Attribut 1	1XX	1
START Attribut 1	1XX	A
STOP Attribut 2	1XX	2
START Attribut 3	1XX	C
STOP Attribut 3	1XX	4
UNTERDRÜCKUNG	1XX	8
Andere Adressen	—	0

Modus 2 (DISPL2 Bit 0 = 1)

Funktion	PROM-Adresse	PROM-Code (Hex)
START Attribut 1	1XX	9
START Attribut 2	1XX	A
START Attribut 3	1XX	C
STOP alle Attribute	1XX	8
UNTERDRÜCKUNG	1XX	1
Andere Adressen	—	0

ASSEMBLY LANGUAGE ASM 80

This section describes the CPU and the Instruction Repertoire. The basic devices directly connected to the CPU I/O bus: Fixed Data Switches, Keyboard and Asynchronous I/O Adapter are described in later sections.

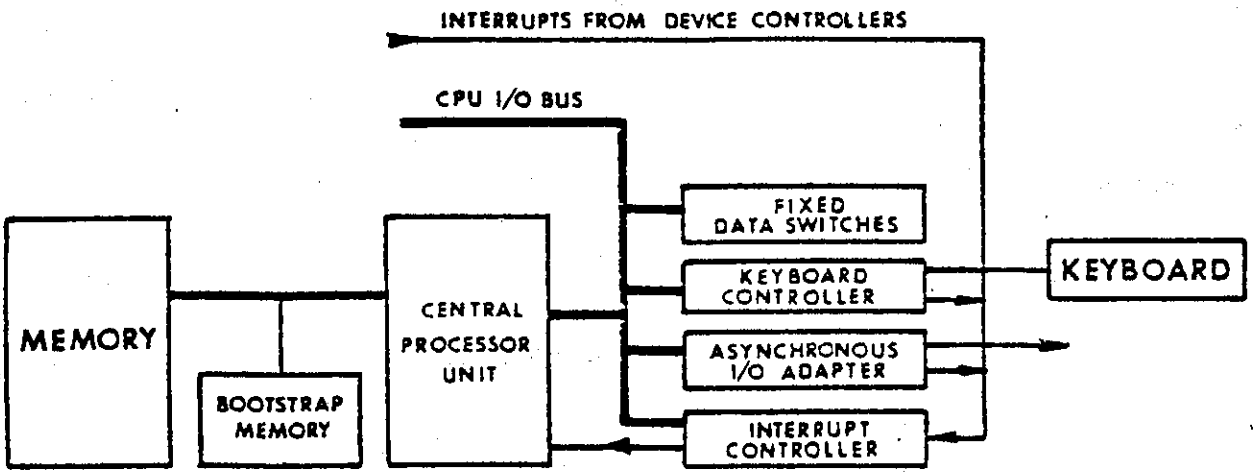


Figure E-1-1. CPU and I/O Bus

CENTRAL PROCESSOR UNIT

The CPU consists of an Arithmetic/Logic Unit, five condition flags, seven general purpose 8-bit registers, and a pushdown stack pointer and a program counter, each 16 bits long. The CPU is capable of directly addressing main memory and up to 512 bytes of read only bootstrap memory.

Access to the CPU internal environment can be provided via an optional systems test console described in Appendix A-1.

ARITHMETIC/LOGIC UNIT

The Arithmetic/Logic Unit is an 8-bit parallel binary computation device that performs addition, subtraction and logical operations.

All individual register arithmetic and logical operations are carried out between the A Register (Accumulator) and any one of the seven general purpose registers or between the A Register and memory. Register pair addition operations are carried out between the H and L registers and any one of the four register pairs.

Five condition flags: Auxiliary Carry, Carry, Sign, Zero and Parity are set as the result of each arithmetic and logical operation as defined below:

Auxiliary Carry (AC) Flag - Set when an arithmetic operation results in an overflow (carry) from bit 3 to bit 4.

Carry (C) Flag - Set when an arithmetic operation results in an overflow (carry) or an underflow (borrow) from bit 7.

Sign (S) Flag - Set when the most significant bit (bit 7) in the result of an arithmetic or logical operation contains a "one".

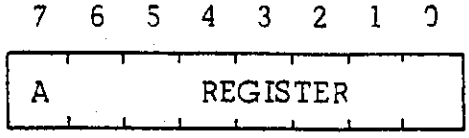
Zero (Z) Flag - Set when all bits in the result of an arithmetic or logical operation are "zero".

Parity (P) Flag - Set when the number of "one" bits in the result of an arithmetic or logical operation is even.

GENERAL PURPOSE REGISTERS

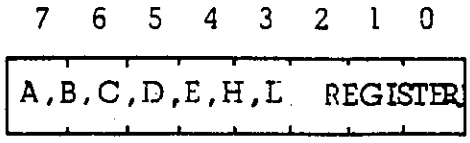
Seven general purpose registers are used for temporary data storage internal to the CPU:

Accumulator



The A Register (Accumulator) receives the result of individual register arithmetic, logical and rotate operations. The A Register is also used as the Input/Output Register for data and control information exchanged between the CPU and the I/O Devices.

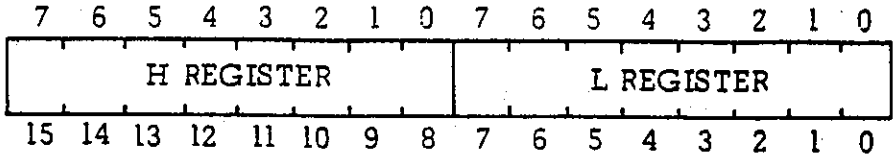
Individual Registers



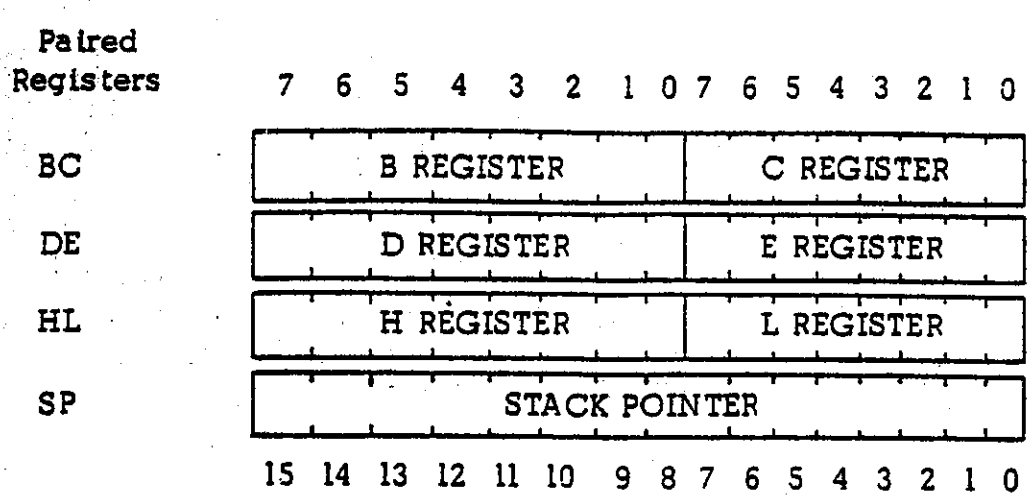
The A, B, C, D, E, H and L Registers can be used in conjunction with the A Register for individual register arithmetic and logical operations. All registers are independent and can be incremented, decremented or loaded from another register or from memory.

Memory Addressing Registers

M = (H□L)



The H and L Registers, besides being used individually, are also used to provide memory addressing capability. The L Register contains the eight lower order address bits and H Register the eight higher order address bits of the memory location referenced. The contents of memory pointed to by the H and L registers is denoted by the letter M.



The invidividual registers can be concatenated in pairs to form a 16-bit register pair. The pair can be used to address memory or can be added to the HL pair. The standard names for the pairs are shown above.

THE STACK

A stack is an area of memory allocated for subroutine or interrupt linkage or for temporary storage. Various data bytes may be "pushed" onto the stack in sequential order and later "popped" or retrieved from the stack in reverse order. To keep track of the last byte pushed to the stack, a stack pointer is provided. The stack pointer (SP) is a 16-bit register which always stores the address of the last byte in the stack. As illustrated in Figure E-1-2,a stack starts at its initial location and expands linearly toward lower addresses as items are pushed to the stack. It is the programmer's responsibility to initiate the stack pointer register and reserve enough room for stacking purposes so that pushing data to the stack never destroys other data stored in memory. Any portion of the memory can be allocated for stack purposes.

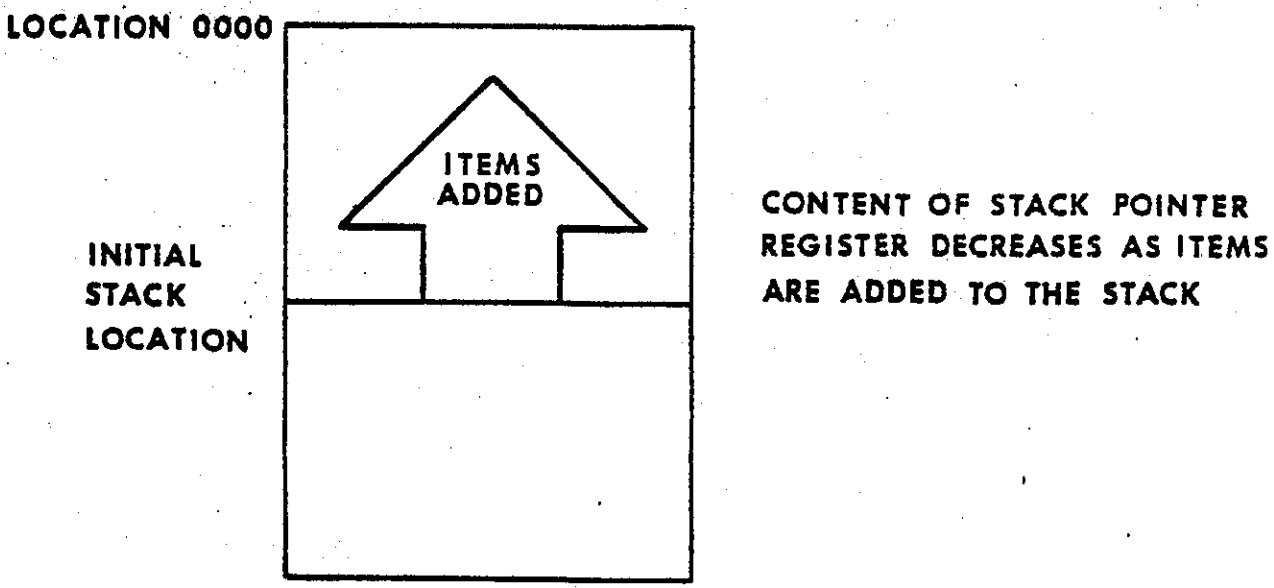


Figure E-1-2. The Stack

BOOTSTRAP MEMORY

The bootstrap memory can be factory programmed to load a program from any source such as tape, communication line or disk. When power is turned on, or when an SBT instruction is executed, or the PROG key is depressed in conjunction with the SHIFT and CTRL keys, the CPU will execute the program starting at location 0 of the bootstrap memory.

The bootstrap memory overlaps the main memory from locations 0 to 01FF until an EBT instruction is executed.

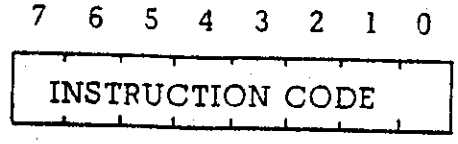
INSTRUCTION REPERTOIRE

The T 52 instructions have been grouped by function and are described in this section. Machine instructions are explained in detail. A general description of I/O instructions is provided, since they may vary depending on the device selected; the details are contained in the appropriate section for each device.

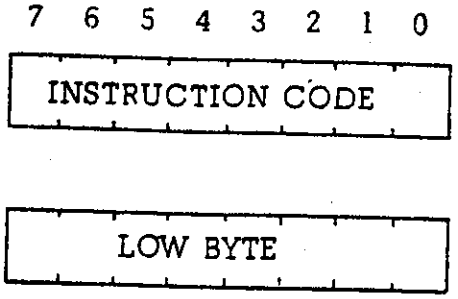
A complete T 52 instruction set including the hexadecimal codes and timing is also contained in Appendix E-2 and Appendix E-3.

INSTRUCTION/PRESENTATION FORMAT

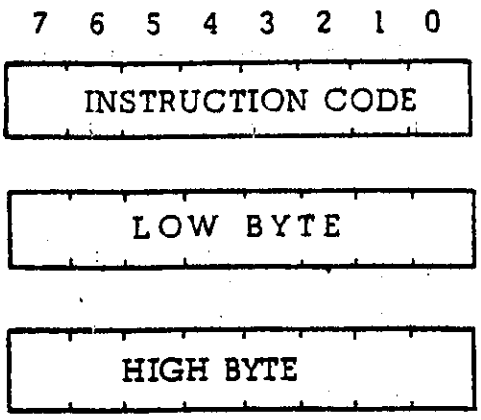
T 52 instructions are either one, two or three bytes long. Multiple byte instructions are stored in successive memory locations. Byte 1 always contains the instruction code. For two byte instructions, Byte 2 contains the data source or I/O extension of the operation. For three byte instructions, Byte 2 contains information such as the least significant address byte, and Byte 3 the most significant address byte.



ONE BYTE INSTRUCTIONS



TWO BYTE INSTRUCTIONS



THREE BYTE INSTRUCTIONS

(CALL, JUMP, ETC.)

Instructions are presented in the following format:

INSTRUCTION

MNEMONIC

SYMBOLIC REPRESENTATION

FLAGS:

FLAGS AFFECTED

NO. OF BYTES

- = UNAFFECTED
X = AFFECTED
0 = CLEARED TO ZERO
1 = SET TO ONE

VERBAL DESCRIPTION

The symbols and abbreviations used are explained in Table E-1-1.

All addresses and codes are expressed in hexadecimal notation.

A	A Register(Accumulator)
AC	Auxiliary Carry
□	Concatenation
dst	One of the destinations: A A Register (Accumulator) B B Register C C Register D D Register E E Register H H Register L L Register M Memory Addressed by H□L
f	One of the condition flags: C Carry - Overflow, Underflow S Sign - Bit 7 = 1 Z Zero - Result is Zero P Parity - Even Parity
n	The number 0 - 7
P	Program Counter
rp	One of the register pairs: BC B□C DE D□E HL H□L SP Stack Pointer
src	One of the sources: A A Register (Accumulator) B B Register C C Register D D Register E E Register H H Register L L Register M Memory Addressed by H□L I Memory Addressed by P + 1
ST	Stack
[ ]	Arithmetic Grouping
$[\,]_n$	Bit n of the value
$[\,]_{n-m}$	Bits n through m of the value (n > m)
+	Arithmetic sum
-	Arithmetic difference
&	Logical AND
!	Logical OR
*	Logical Exclusive OR
→	Is transferred to
↔	Is exchanged with
(X)	Content of Memory location addressed by X

Table E-1-1. Symbols and Abbreviations

ACCUMULATOR INSTRUCTIONS

LOAD ACCUMULATOR DIRECT

@I.A

OPERATION: $((P + 2) \square (P + 1)) \rightarrow A$

BYTES: 3

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of the memory location whose address immediately follows the instruction is moved to the accumulator.

STORE ACCUMULATOR DIRECT

A.@I

OPERATION: $A \rightarrow ((P + 2) \square (P + 1))$

BYTES: 3

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of the accumulator is moved to the memory location whose address immediately follows the instruction.

TEST ACCUMULATOR

TST

OPERATION: $[A] \; ! \; [A] \rightarrow A$

BYTES: 1

FLAGS:

AC	C	S	Z	P
0	0	X	X	X

Set the sign, zero and parity flags according to the contents of the accumulator. Clear the carry and auxiliary carry flags.

COMPLEMENT ACCUMULATOR

CMA

OPERATION: $[A] * OFF \rightarrow A$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of the accumulator is complemented, i.e. each individual bit is inverted.

LOAD ACCUMULATOR INDIRECT @rp.A

OPERATION: (rp) → A

BYTES: 1 FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of the memory location whose address is contained in the register pair is moved to the accumulator. @HL.A is not implemented, use M.A; also, @SP.A is not implemented.

STORE ACCUMULATOR INDIRECT A.@rp

OPERATION: A → (rp)

BYTES: 1 FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of the accumulator is moved to the memory location whose address is contained in the register pair. A.@HL is not implemented, use A.M; also, A.@SP is not implemented.

DECIMAL ADJUST ACCUMULATOR DAA

OPERATION: First: IF $[A]_{3-0} > 9$ or auxiliary carry = 1

A +06 → A

Second: IF $[A]_{7-4} > 9$ or carry = 1

A + 060 → A

BYTES: 1 FLAGS:

AC	C	S	Z	P
X	X	X	X	X

The eight-bit binary number in the accumulator is adjusted to form two four-bit binary coded decimal digits by:

1. Adding 6 to the least significant four bits of the accumulator if these bits are greater than 9 or if the auxiliary carry is set.

2. Adding 6 to the most significant four bits of the accumulator if these bits are now greater than 9 or if the carry is now set.

ROTATE LEFT CIRCULAR — RLC

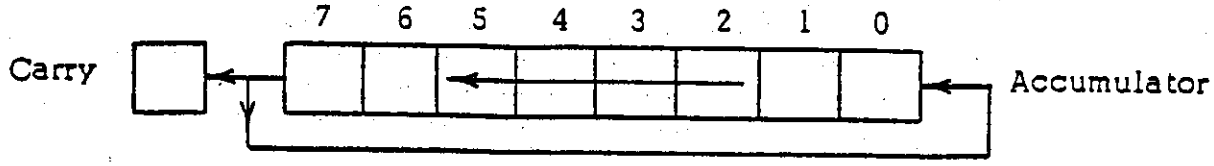
OPERATION: $[A]_n \rightarrow [A]_{n+1}, [A]_7 \rightarrow [A]_0, [A]_7 \rightarrow carr$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	X	-	-	-

Rotate the contents of the accumulator one bit to the left. Place $[A]_7$ into the vacated $[A]_0$ position and into the carry flag (C) as illustrated below:



ROTATE RIGHT CIRCULAR — RRC

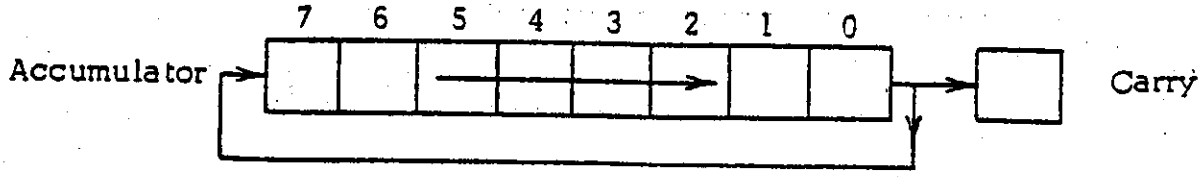
OPERATION: $[A]_n \rightarrow [A]_{n-1}, [A]_0 \rightarrow [A]_7, [A]_0 \rightarrow carry$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	X	-	-	-

Rotate the contents of the accumulator one bit to the right. Place $[A]_0$ into the vacated $[A]_7$ position and into the carry flag (C) as illustrated below:



ROTATE LEFT THROUGH CARRY

RAL

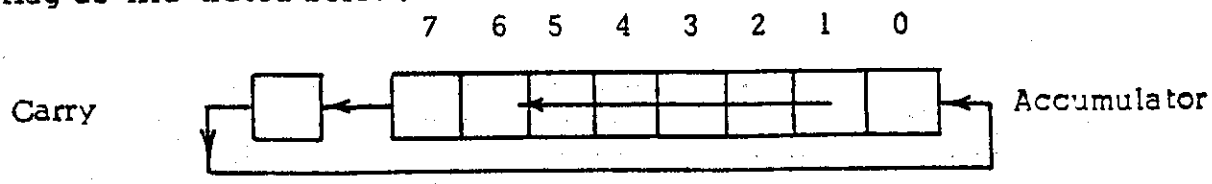
OPERATION: $[A]_{n} \leftarrow [A]_{n+1}, [carry] \leftarrow [A]_{0}, [A]_{7} \rightarrow carry$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	X	-	-	-

Rotate the contents of the accumulator one bit to the left. Place the content of the carry flag (C) into the $[A]_{0}$ position; place $[A]_{7}$ into the vacated carry flag as illustrated below:



ROTATE RIGHT THROUGH CARRY

RAR

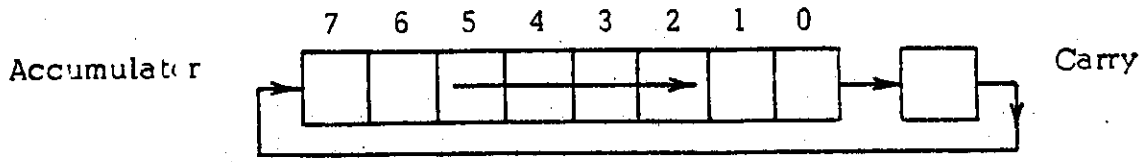
OPERATION $[A]_{n} \leftarrow [A]_{n-1}, [carry] \leftarrow [A]_{7}, [A]_{0} \rightarrow carry$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	X	-	-	-

Rotate the contents of the accumulator one bit to the right. Place the content of the carry flag into the $[A]_{7}$ position; place $[A]_{0}$ into the vacated carry flag (c) as illustrated below:



INDIVIDUAL REGISTER INSTRUCTIONS

LOAD

src.dst

OPERATION: [src]→dst

BYTES: 1

2 if I.dst

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Load the contents of the source (src) into the destination (dst). The flags are not affected by a load operation. A.A, B.B, C.C, D.D, E.E, H.H, L.L and M.M are not implemented, use NOP.

INCREMENT

INdst

OPERATION: [dst]+ 1→dst

BYTES: 1

FLAGS:

AC	C	S	Z	P
X	-	X	X	X

Increment the contents of the destination (dst) by one.

DECREMENT

DCdst

OPERATION: [dst]- 1→dst

BYTES: 1

FLAGS:

AC	C	S	Z	P
X	-	X	X	X

Decrement the contents of the destination (dst) by one.

COMPARE

CPsrc

OPERATION: [A]-[src]

BYTES: 1

2 if CPI

FLAGS:

AC	C	S	Z	P
X	X	X	X	X

Compare the contents of the source (src) with the contents of the accumulator. The contents of the accumulator remain unchanged. The instruction assumes unsigned arithmetic; the zero flag (Z) is set to a "1" if src = accumulator. The carry flag (C) is set to a "1" if src > accumulator; cleared to "0" if src ≤ accumulator.

ADD ADsrc

OPERATION: [A]+[src] → A

BYTES: 1 FLAGS:

2 if ADI

AC	C	S	Z	P
X	X	X	X	X

Add the contents of the source (src) to the contents of the accumulator and retain the sum in the accumulator.

SUBTRACT SUsrc

OPERATION: [A] - [src] → A

BYTES: 1 FLAGS:

2 if SUI

AC	C	S	Z	P
X	X	X	X	X

Subtract the contents of the source (src) from the contents of the accumulator and retain the difference in the accumulator. The carry flag (C) is set to a "1" if an underflow condition occurs. SUA is not implemented, use XRA.

ADD WITH CARRY ACsrc

OPERATION: [A]+[src]+[carry] → A

BYTES: 1 FLAGS:

2 if ACI

AC	C	S	Z	P
X	X	X	X	X

Add the contents of the source (src) and the carry flag to the contents of the accumulator and retain the sum in the accumulator.

SUBTRACT WITH BORROW SBsrc

OPERATION [A]-[src]-[carry] → A

BYTES: 1 FLAGS:

2 if SBI

AC	C	S	Z	P
X	X	X	X	X

Subtract the contents of the source (src) and the carry flag from the contents of the accumulator and retain the difference in the accumulator The carry flag (C) is set to a "1" if an underflow condition occurs.

AND

NDsrc

OPERATION: [A]&[src] → A

BYTES: 1

2 if NDI

FLAGS:

AC	C	S	Z	P
*	0	X	X	X

Compute the logical product of the contents of the source (src) and the contents of the accumulator and place the result in the accumulator. NDA is not implemented, use TST. *Flag "AC" is undefined.

EXAMPLE:

$[A]_n$	1	1	0	0
$[src]_n$	1	0	0	1
$[A]_n$	1	0	0	0

EXCLUSIVE OR

XRsrc

OPERATION: [A] * [src] → A

BYTES: 1

2 if XRI

FLAGS:

AC	C	S	Z	P
0	0	X	X	X

Compute the logical difference of the contents of the source (src) and the contents of the accumulator and place the result in the accumulator.

EXAMPLE:

$[A]_n$	1	1	0	0
$[src]_n$	1	0	0	1
$[A]_n$	0	1	0	1

INCLUSIVE OR

ORsrc

OPERATION: [A] ! [src] → A

BYTES: 1

2 if ORI

FLAGS:

AC	C	S	Z	P
0	0	X	X	X

Compute the logical sum of the contents of the source (src) and the contents of the accumulator and place the result in the accumulator. ORA is not implemented, use TST.

EXAMPLE:

$[A]_n$	1	1	0	0
$[src]_n$	1	0	0	1
$[A]_n$	1	1	0	1

REGISTER PAIR INSTRUCTIONS

LOAD REGISTER PAIR IMMEDIATE — I.rp

OPERATION: (P + 2) □ (P + 1) → rp

BYTES: 3

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Load the register pair with the contents of the two bytes in memory immediately following the instruction.

INCREMENT REGISTER PAIR — INrp

OPERATION: [rp] + 1 → rp

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Increment the contents of the register pair by one.

DECREMENT REGISTER PAIR — DCrp

OPERATION: [rp] - 1 → rp

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Decrement the contents of the register pair by one.

ADD REGISTER PAIRS — ADrp

OPERATION: [H □ L] + [rp] → H □ L

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	X	-	-	-

The contents of the register pair is added to the contents of the HL register pair and the sum is retained in the HL register pair. The carry flag (C) is set to a "1" if the double precision add overflows; otherwise, it is cleared to "0".

LOAD H AND L DIRECT

@I.HL

OPERATION:

$$\left((P + 2) \square (P + 1)\right) \rightarrow L$$

$$\left(\left[(P + 2) \square (P + 1)\right] + 1\right) \rightarrow H$$

BYTES: 3

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of the memory location whose address immediately follows the instruction is moved to register L. The contents of the succeeding location is moved to register H.

STORE H AND L DIRECT

HL.@I

OPERATION:

$$L \rightarrow \left((P + 2) \square (P + 1) \right)$$

$$H \rightarrow \left(\left[(P + 2) \square (P + 1) \right] + 1 \right)$$

BYTES: 3

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of register L is moved to the memory location whose address immediately follows the instruction. The contents of the register H is moved to the succeeding location.

EXCHANGE HL AND DE

HL \ DE

OPERATION:

$H \leftrightarrow D$

$L \leftrightarrow E$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of registers H and L are exchanged with the contents of registers D and E.

MOVE HL TO SP

HL.SP

OPERATION:

$$[H \square L] \rightarrow SP$$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of register pair HL is moved to the stack pointer register.

STACK INSTRUCTIONS

PUSH REGISTER PAIR

rp.ST

$[rp]_{15-8} \rightarrow (SP-1)$

$[rp]_{7-0} \rightarrow (SP-2)$

$[SP]-2 \rightarrow SP$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The two registers of the register pair are saved on the stack. SP.ST is not implemented.

PUSH ACCUMULATOR AND FLAGS

A.ST

OPERATION:

$A \rightarrow (SP-1)$

carry $\rightarrow [(SP-2)]_0$

$1 \rightarrow [(SP-2)]_1$

parity $\rightarrow [(SP-2)]_2$

$0 \rightarrow [(SP-2)]_3$

auxiliary carry $\rightarrow [(SP-2$

$0 \rightarrow [(SP-2)]_5$

zero $\rightarrow [(SP-2)]_6$

sign $\rightarrow [(SP-2)]_7$

$[SP]-2 \rightarrow SP$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of the accumulator and the state of the condition flags are saved on the stack.

POP REGISTER PAIR ST.rp

OPERATION: $[(SP + 1) \square (SP)] \rightarrow rp$

$[SP] + 2 \rightarrow SP$

BYTES: 1 FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The two registers of the register pair are restored from the stack. ST.SP is not implemented.

POP ACCUMULATOR AND FLAGS ST.A

OPERATION: $[(SP)]_0 \rightarrow$ carry

$[(SP)]_2 \rightarrow$ partty

$(SP)]_4 \rightarrow$ auxiliary carry

$[(SP)]_6 \rightarrow$ zero

$[(SP)]_7 \rightarrow$ sign

$(SP + 1) \rightarrow A$

$[SP] + 2 \rightarrow SP$

BYTES: 1 FLAGS:

AC	C	S	Z	P
X	X	X	X	X

The contents of the accumulator and the state of the condition flags are restored from the stack.

EXCHANGE HL AND STACK TOP HL \ ST

OPERATION: $H \leftrightarrow (SP + 1)$

$L \leftrightarrow (SP)$

BYTES: 1 FLAGS:

AC	C	S	Z	P
-	-	-	-	-

The contents of registers H and L are exchanged with the two bytes on the top of the stack.

PROGRAM CONTROL INSTRUCTIONS

JUMP JMP

OPERATION: (P + 2) □ (P + 1) → P

BYTES: 3

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Jump to the address specified by the contents of the two memory locations immediately following the instruction.

JUMP IF CONDITION TRUE JTf

OPERATION: IF [f] = 1, (P + 2) □ (P + 1) → P

Otherwise, [P] + 3 → P

BYTES: 3

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

If the content of the designated flag (f) is one, jump to the address specified by the contents of the two memory locations immediately following the instruction. Otherwise, execute the next sequentially available instruction.

JUMP IF CONDITION FALSE JFf

OPERATION: IF [f] = 0, (P + 2) □ (P + 1) → P

Otherwise, [P] + 3 → P

BYTES: 3

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

If the content of the designated flag (f) is zero, jump to the address specified by the contents of the two memory locations immediately following the instruction. Otherwise, execute the next sequentially available instruction.

JUMP INDIRECT

J@HL

OPERATION: H □ L → P

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

BYTES: 1

Jump to the address specified by the contents of registers H and L.

CALL

CAL

OPERATION:

$[P + 3]_{15-8} \rightarrow (SP - 1)$

$[P + 3]_{7-0} \rightarrow (SP - 2)$

$[SP] - 2 \rightarrow SP$

$(P + 2) \square (P + 1) \rightarrow P$

BYTES: 3

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Transfer the address of the next sequentially available instruction to the pushdown stack and jump to the address specified by the contents of the two memory locations immediately following the instruction.

CALL IF CONDITION TRUE

CTf

OPERATION: IF $[f] = 1$,

$[P + 3]_{15-8} \rightarrow (SP - 1)$

$[P + 3]_{7-0} \rightarrow (SP - 2)$

$[SP] - 2 \rightarrow SP$

$(P + 2) \square (P + 1) \rightarrow P$

Otherwise, $[P] + 3 \rightarrow P$

BYTES: 3

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

If the content of the designated flag (f) is one, transfer the address of the next sequentially available instruction to the pushdown stack and jump to the address specified by the contents of the two memory locations immediately following the instruction. Otherwise, execute the next sequentially available instruction.

CALL IF CONDITION FALSE — CFf

OPERATION: IF [f] = 0,

$[P + 3]_{15-8} \rightarrow (SP - 1)$

$[P + 3]_{7-0} \rightarrow (SP - 2)$

$[SP] - 2 \rightarrow SP$

$(P + 2) \square (P + 1) \rightarrow P$

Otherwise, $[P] + 3 \rightarrow P$

BYTES: FLAGS:

AC	C	S	Z	P
-	-	-	-	-

If the content of the designated flag (f) is zero, transfer the address of the next sequentially available instruction to the pushdown stack and jump to the address specified by the contents of the two memory locations immediately following the instruction. Otherwise, execute the next sequentially available instruction.

RESTART — RSTn

OPERATION:

$[P + 1]_{15-8} \rightarrow (SP - 1)$

$[P + 1]_{7-0} \rightarrow (SP - 2)$

$[SP] - 2 \rightarrow SP$

$8\ [n] \rightarrow P$

BYTES: 1 FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Transfer the address of the next sequentially available instruction to the pushdown stack and jump to the address specified by the number n of the designated restart instruction multiplied by eight.

RETURN

RET

OPERATION:

$(SP + 1) \square (SP) \rightarrow P$

$[SP] + 2 \rightarrow SP$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Transfer the last address pushed to the stack into the program counter thereby returning to the instruction following the last executed call or restart.

RETURN IF CONDITION TRUE

RTf

OPERATION: If $[f] = 1$,

$(SP + 1) \square (SP) \rightarrow P$

$[SP] + 2 \rightarrow SP$

Otherwise, $[P] + 1 \rightarrow P$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

If the content of the specified flag (f) is one, transfer the last address pushed to the stack into the program counter thereby returning to the instruction following the last executed call or restart. Otherwise, execute the next sequentially available instruction.

RETURN IF CONDITION FALSE

RFf

OPERATION: IF $[f] = 0$,

$(SP + 1) \square (SP) \rightarrow P$

$[SP] + 2 \rightarrow SP$

Otherwise, $[P] + 1 \rightarrow P$

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

If the content of the specified flag (f) is zero, transfer the last address pushed to the stack into the program counter thereby returning to the instruction following the last executed call or restart. Otherwise, execute the next sequentially available instruction.

MISCELLANEOUS INSTRUCTIONS

SET CARRY

STC

OPERATION: 1 → carry

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	1	-	-	-

The carry flag is set to one.

COMPLEMENT CARRY

CMC

OPERATION: carry * 1 → carry

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	X	-	-	-

The carry flag is inverted.

NO OPERATION

NOP

OPERATION: None

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

No operation is performed.

ENABLE INTERRUPTS

EIN

OPERATION: 1 → master interrupt enable

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Allows any one of the eight individual interrupt requests to generate an interrupt if its corresponding bit in the interrupt mask is set. Interrupts are disabled by a DIN instruction or by the hardware when any interrupt is accepted.

DISABLE INTERRUPTS

DIN

OPERATION: 0 → master interrupt enable

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Disallows any of the eight individual interrupt requests, even if its corresponding bit in the interrupt mask is set, from generating an interrupt.

HALT

HLT

OPERATION: P → P

BYTES: 1

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

Program execution is stopped. Execution can be restarted only if an interrupt is sensed.

INPUT AND OUTPUT INSTRUCTIONS

Input/Output instructions have a different specific interpretation depending on the device selected and not all I/O instructions are used for each device. All I/O instructions are two bytes long and do not affect the flags. A detailed I/O description for each device is contained in the appropriate section.

INPUT IPT

OPERATION: (P + 1) → determine input operation

external device → A

BYTES: 2

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

A family of two byte instructions that perform input operations. The first byte is the IPT code. The second byte is listed below:

MNEMONIC	CODE	OPERATION
IFL	00	INPUT FLAG transfers an operational status byte from the selected device to the accumulator.
INP	01	INPUT transfers a data byte from the selected device to the accumulator.
IIN	02	INPUT INTERRUPT STATUS loads the contents of the Interrupt Status Register into the accumulator.
FIX1	03	FIXED DATA SWITCH 1 transfers the contents of fixed data switch 1 to the accumulator.
FIX2	04	FIXED DATA SWITCH 2 transfers the contents of fixed data switch 2 to the accumulator.

OUTPUT

OPT

OPERATION: (P + 1) → determine output operation

A → external device

BYTES: 2

FLAGS:

AC	C	S	Z	P
-	-	-	-	-

A family of two byte instructions that perform output operations. The first byte is the CPT code. The second byte is listed below:

MNEMONIC	CODE	OPERATION
INIT	00	INITIALIZE stops any current device activity and clears and initializes all devices.
SEL	01	SELECT selects the device specified by the device address in the accumulator for I/O operations. Table E-1-2 lists the presently implemented device address assignments.
OUT	02	OUTPUT transfers a data byte from the accumulator to the selected device.
DVCL	03	DEVICE CLEAR stops any current activity and resets the selected device. DVCL should be used prior to issuing other commands when the status of the device is uncertain.
OFL	04	OUTPUT FLAGS outputs an operational status byte from the accumulator to the selected device.
COM1	05	COMMAND 1,2,3 transfers a command byte from the accumulator to the selected device. The command instructions have a different meaning to each device selected.
COM2	06	
COM3	07	

MNEMONIC	CODE	OPERATION
SBT	08	START BOOT executes an RST0 and then executes instructions stored in the bootstrap memory. Instructions executed or data read from locations 0 to 01FF are read from the bootstrap memory while data written to these locations are written to main memory. All memory accesses to address 0200 or higher access main memory only.
EBT	09	END BOOT executes an RST0 and then executes instructions from main memory. All memory accesses are to main memory.
ATCL	0B	ATTENTION CLEAR clears the attention interrupt request from the communications controller without changing any communications activities. The communications controller does not have to be selected.
SMSK	0C	SET INTERRUPT MASK selects the device(s) from which to accept interrupts.
BEEP	0D	BEEP activates a one second audible tone. The keyboard does not have to be selected.
CLICK	0E	CLICK activates an impulse audible sound. The keyboard does not have to be selected.

Standard Device Address assignments are listed below.

ADDRESS	DEVICE
F0	Asynchronous I/O Channel
E1	Keyboard
D2	
C3	Communications Controller (Synchronous)
B4	Printer/Byte String Controller
A5	
96	Tape Controller
87	Disk Controller
78	Multiprocessor Controller
69	Communication Controller (Asynchronous)
5A	Diskette Controller
4B	
3C	
2D	
1E	
0F	

Table E-1-2. Device Address Assignments