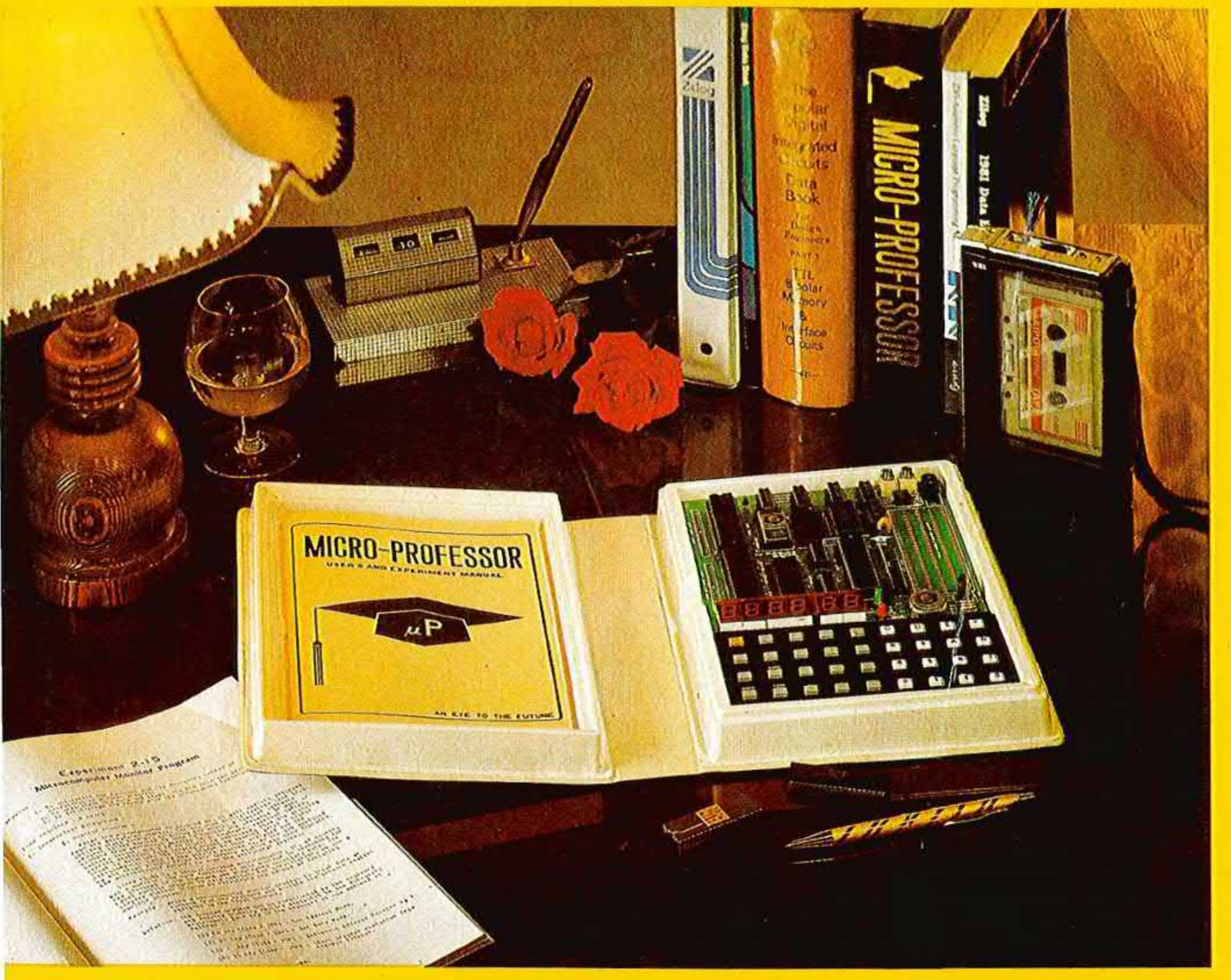
MPF-I

EXPERIMENT MANUAL
(SOFTWARE/HARDWARE)



MPF-I
EXPERIMENT MANUAL
(SOFTWARE/HARDWARE)

COPYRIGHT

DISCLAIMER

Multitech
INDUSTRIAL CORP.

OFFICE/ 9FL. 266 SUNG CHIANG ROAD, TAIPEI 104 TAIWAN R.O.C.
TEL:(02)551-1101 TELEX:"19162 MULTIIC" FAX:(02) 542-2805
FACTORY/ 1 INDUSTRIAL E. RD., III HSINCHU SCIENCE - BASED
INDUSTRIAL PARK, HSINCHU, TAIWAN, R.O.C.

PREFACE

The first 50 years of the 20th century witnessed the invention of the internal combustion engine, which greatly extended the physical strength of the human body.

In the second half of the century, the birth of the microprocessor further extended our mental capabilities. Applications of this amazing product in various industries have introduced so much impact on our lives, hence, it is called the second Industrial Revolution.

Microcomputers represent a total change in designing systems. Both industrial and academic institutions are active in the development and search for new applications for microcomputers.

This book is designed to be used in conjunction with the "multitech" MPF-1 Microcomputer as part of a one-year laboratory class on microcomputers. With the aid of this book, students will be able to learn the fundamentals of microcomputers, from basic CPU instructions to practical applications.

The first part of this book is an introduction to the basic concepts of microcomputer programming. It lays the foundation for later studies, the second part of this book is the source list of monitor program, the third part begins with a series of experiments using microcomputer instructions, such as, data transfers, arithmetic and logic operations, jump and subroutine and memory address allocation in simple programs. Experiments involving more complicated arithmetic operations, such as, binary to decimal conversion, decimal to binary conversion, multiplication, division and square root are presented.

There are two experiments in this book which are designed to familiarize the student with the fundamentals of input/output programming. These programs are centered around the keyboard and display. These experiments establish the foundation for later experiments involving a simple monitor program, which leads to more complicated MPF-1 programs.

MPF- I EXPERIMENT MANUAL

TABLE OF CONTENTS

Preparations

Introduction To Designing Microcomputer Programs

A computer program is an organized series of instructions. The central processing unit will perform a series of logical actions to obtain the desired result.

Before a proram is executed by CPU it must be stored in memory in binary form. This type of program is called a "machine language program". This is the only type of language the computer understands. The machine language program is usually represented by Hexadecimal digits. For example, the 8-bit instruction 1010 1111B(B represents binary) in the Z80 CPU it can be replaced by OAFH (H indicates Hexidecimal). Interpreting a machine language program is extremely difficult and time consuming for the User. The microprocessor manufacturer divides the CPU instructions into several categories according to their functions. The CPU instructions and registers are usually represented by symbols called "mnemonics". For example, the Z80 CPU instruction 70H can be represented by the mnemonic code LD A,L (Load Data into register A from register L). A program written in mnemonic codes is called an "assembly language program." Before an assembly language program can be executed by the CPU, it must be translated into machine language by a special software program called an "Assembler".

Normally a program is written in assembly language. The main advantage of assembly language program over machine language programming is that assembly language programming is much faster to code, the mnemonics makes it much easier for the User to remember the instruction set, and normally the assembler will contain a self-diagnostic package for debugging programs. The main disadvantage of assembly language programs is that it requires an assembler and microcomputer development system. These two items are very costly. With the MPF-I microcomputer the User has to translate assembly programs into machine level programs by hand before executing programs

A. Problem Analysis

The software program of a simple problem may be easily designed with a well-defined flowchart. It may also be obtained by revising some existing programs or combining some simple routines. The design of more complicated programs, such as monitor programs, system control programs or a special purpose program, are usually started after some detailed analysis of the problem has been made. Problem analysis and solution requires a good understanding of the following:

See page (III-3)

(1) Characteristic and requirements of the problem
(2) Conditions which are known
(3) Input information format and how it is converted
(4) Output data format and how it is converted
(5) Type of data and how precise it is
(6) Execution speed required
(7) CPU instructions and performance
(8) Memory size
(9) The possibility that the problem can be solved
(10) Methods to solve the problem
(11) Evaluation of the program
(12) How the resultant program will be disposed

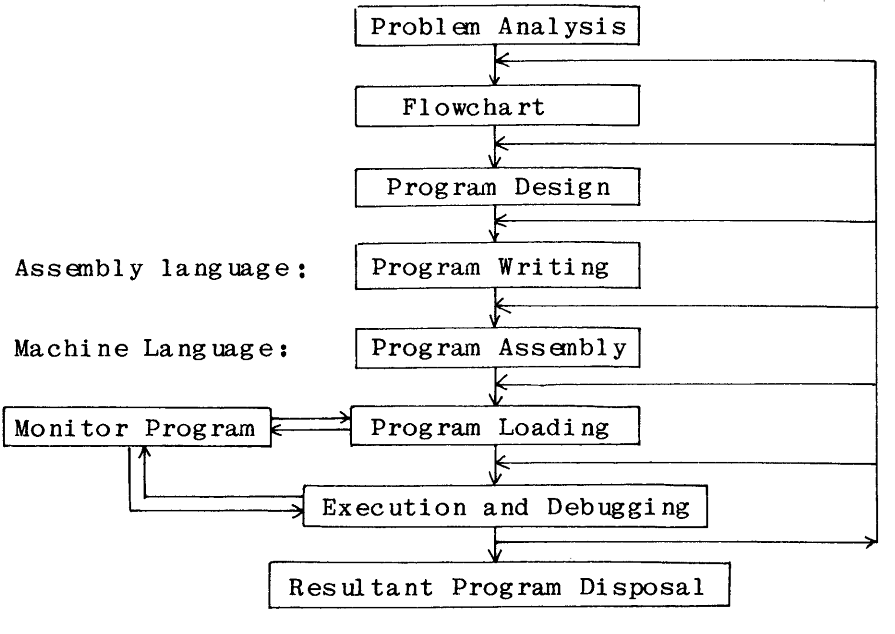


Figure 2-A-1

B. Flowchart

A flowchart can be used to indicate the behavior of algorithms by suitable graphs. Once the complete flowchart has been completed, a full picture of the programmer's thought processes in reaching a solution to the problem may be followed. Flowcharts are especially important in program-debugging. It is an important part of the finished program. It may help other people to understand the exact algorithm used by the programmer.

Two levels of flowcharts are often desirable:

System flowchart -- showing the general flow of the program

Detailed flowchart -- providing details that are of interest mainly to the programmer.

Usually, a complicated program is introduced using a system flowchart outlining the program, and then a detailed flowchart is presented.
The advantage of a flowchart is that it emphasizes the sequential nature of steps by using arrows pointing from each step to its successor. Various symbols are used to indicate the operation that is to be performed at each step. Figure 3-A-2 gives some standard symbols used in flowcharts:

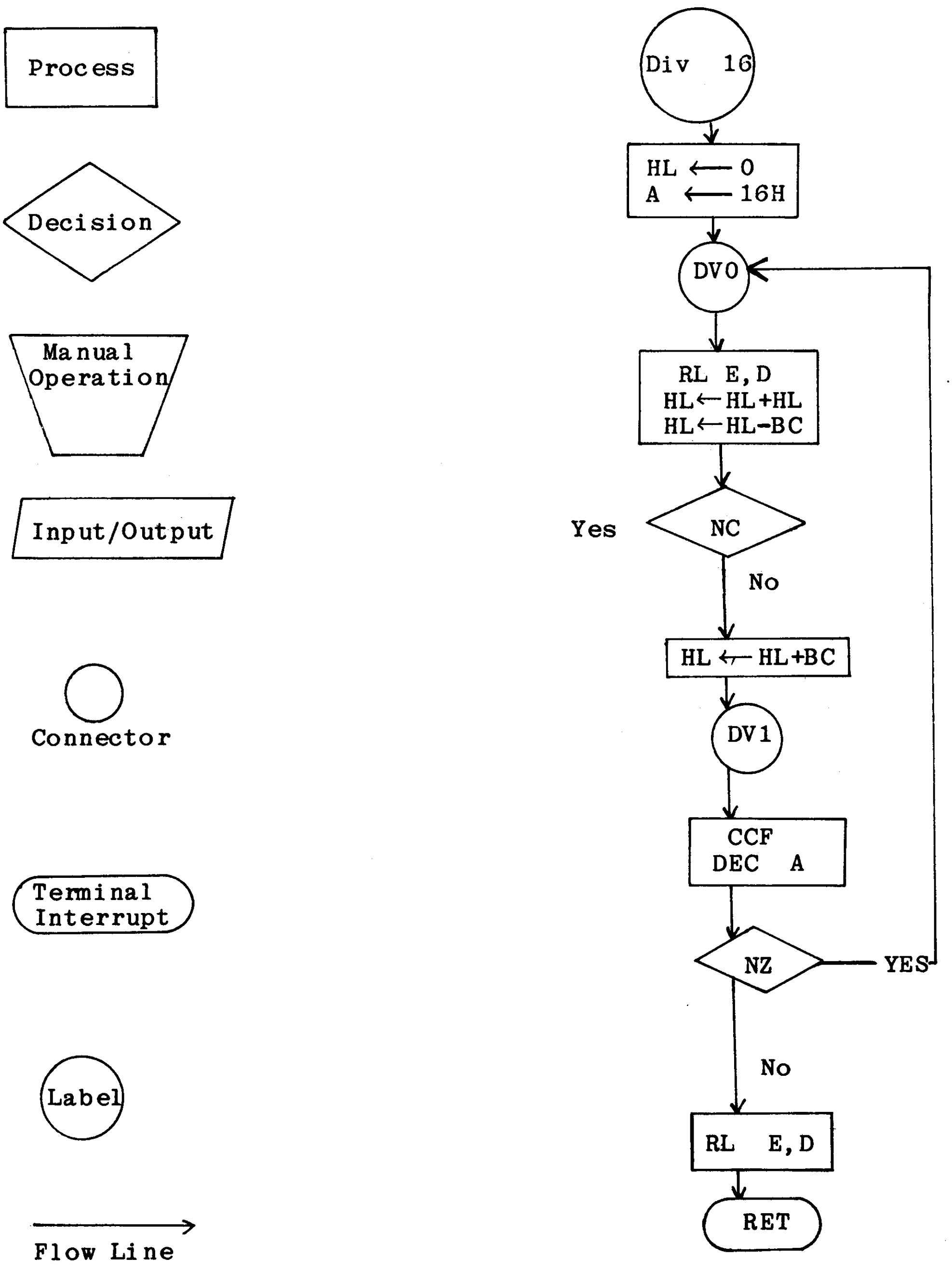

Process
Decision
Manual Operation
Input/Output
Connector
Terminal Interrupt
Label
Flow Line
Div 16
HL ⟵ 0
A ⟵ 16H
DV0
RL E,D
HL←HL+HL
HL←HL-BC
NC
Yes
No
HL ← HL+BC
DV1
CCF
DEC A
NZ
YES
No
RL E,D
RET

Figure 2-A-2

C Program Design

There are many types of programs. Programs for mathematical equations, conversion of input and output signals, coding and decoding of the program data, peripheral device drives, etc. are example of simple programs. Assembler, monitor and system control programs or special purpose applications are examples of more complicated programs. The following items are usually considered in program design:

(1) Acquisition of input signals or data
(2) Generation or conversion of output signals and data
(3) Logical analysis and calculations in the main program
(4) Relation between the main program and subroutines
(5) Use of internal registers
(6) Memory allocation of the main program
(7) Memory allocation of subroutines
(8) Memory allocation of data tables and indexed addressing method
(9) System initialization and constants in the program
(10) Definition of the variables in the program
(11) Consideration of timing sequences and program execution speed
(12) Limitations of memory size
(13) Length and precision of data
(14) Availability of documents and references
(15) Other special items

D. Program Writing

In this book, the programs are written mainly in assembly language. Here only the format of the assembly language program is given.

A statement in the program is composed of four parts : Label, Opcode, Operand and Comment. An example is shown below

```
LABEL   OPCODE & OPERAND                 COMMENT
-----   --------------------             ----------

DTB4    LD       B,16
DB3     SRL      H
        RR       L
        RR       D
        RR       E                       ; ROTATE HL DE RIGHT
        LD       A,H
        CALL     DB1                     ; CORRECT H
        LD       H,A
        LD       A,L
        CALL     DB4                     ; BINARY CORRECT L
        LD       L,A
        DJNZ     DB3
        RET

  BINARY CORRECT ROUTINE
DB4     BIT      7,A
        JR       Z,DB1                   ; IF BIT 7 OF A = 1, SUB FROM 30H
        SUB      30H
DB1     BIT      3,A
        JR       Z,DB2                   ; IF BIT 3 OF A = 1, SUB FROM 03H
        SUB      3
DB2     RET
```

Sometimes, a program statement without a comment is not easy to understand. The comments in the statements are very important especially for a complicated program. Statements with a label and comment field are more convenient for calling and debugging.

E. Program Assembly

Using the resident assembler in a microcomputer system is an effective way to assemble the source program. However, a beginner or a proram designer not familiar with the microcomputer development system must assemble his/her program by hand. The usual procedure for hand assembly is:

(1) Translate each instruction (mnemonic) into the machine code by looking it up in the conversion table. The comment field of each statement is ignored.

(2) After deciding the starting address of the program. Assign an appropriate address to the first byte of each instruction. The exact number of bytes needed must be reserved including space for instructions such as JR, DJNZ, and destination addresses of instructions JP, CALL, etc.

(3) Calculate the relative displacement and put it in the assembled program. A simple formula for calculating the relative displacement is:

displacement = (destination address) - (next instruction address)

If the calculated result is positive, then it is the desired value. If the calculated result is negative, then subtract the result from 100H (i.e. take its 2's complement) and the final result is taken as the operand of this instruction. For instance, in the program listed above, the instruction DJNZ DB3 at address 0014H is first translated into 10xx and then the xx value is calculated.

```
xx  = 0002H (destination address) - 1016H (next instruction's address)
    = -14H (negative value)
xx  = 100H - 14H = 0ECH
```

Therefore, the instruction DJNZ DB3 must be translated into 10EC. In addition, the instruction JR Z, DB 1 at address 0019H is first translated into 28xx, and then the xx value is calculated.

```
xx  = 001DH (destination address) - 001BH (next instruction's address)
    = 2 H
```

The instruction JR Z, DB 1 must be translated into 2802.

The translated machine language is given below:

```
        Machine
Address Language Label  Opcode & Operand Comment
------- -------- -----  ---------------- -------
                  ; ** 4 DIGIT BCD TO BINARY CONVERTION ROUTINE **
                  ; EXTRY : BCD DATA IN HL
                  ; EXIT : BINARY DATA IN DE
                  ; REGISTER CHANGED : AF BC DE HL
```

```
0000    0610    DTB4    LD      B,16     ; B = BIT COUNT
0002    CB3C    DB3     SRL     H
0004    CB1D            RR      L
0006    CB1A            RR      D
0008    CB1B            RR      E        ; ROTATE HL DE RIGHT
000A    7C              LD      A,H
000B    CD1D00          CALL    DB1
000E    67              LD      H,A      ; CORRECT H
000F    7D              LD      A,L
0010    CD1700          CALL    DB4
0013    6F              LD      L,A      ; BINARY CORRECT L
0014    10EC            DJNZ    DB3
0016    C9              RET
                ;
                ; BINARY CORRECT ROUNTINE
0017    CB7F    DB4     BIT     7,A
0019    2802            JR      Z,DB1    ; IF BIT 7 OF A = 1, SUB FROM 30H
001B    D630            SUB     30H
001D    CB5F    DB1     BIT     3,A
001F    2802            JR      Z,DB2    ; IF BIT 3 OF A = 1, SUB FROM 03H
0021    D603            SUB     3
0023    C9      DB2     RET
```

F. Program Loading

The monitor program can be used to assist the user in loading the program into the reserved memory address in MPF-I. The program can be inputted from the keyboard or read from a magnetic tape. After the prgram is loaded into MPF-I RAM, an error-checking process is required to eliminate any errors. Redundant instructions or data may be replaced by an "NOP" instruction. Missed instructions or data are inserted into the desired addresses by using the Block Data Transfer method or simply by reloading the program. While revising the program, it is very important to check whether Jump instructions (JP, JR, DJNZ, CALL, etc.) are affected by the the change in memory addresses. If this happens, then make the necessary correction(s) immediately.

G. Program Execution and Debugging

Before executing a program, it is necessary to set the initialization parameters and set the program counter at the starting address of the program. Pressing the GO key will start the program execution. After the program execution is completed, check the result. If there is any error, the program must be checked step by step with the aid of the monitor program. After the program is revised, execute it again and check the result agian.

Experiment 1

Data-Transfer Experiment

Purposes:

1. To familiarize the user with the function of data-transfer instruction
2. To practise setting the initial value of data
3. To practise assembling, loading and executing a program

Time required: 4 hours

I. Theorectical Background:

1. Most of the data-transfer operation is accomplished by using LD (load) instructions. Data can be transferred in units of 8 bits or 16 bits. Also, instructions such as EX, EXX, PUSH and POP can be used to transfer 16-bit data. Instructions such as LDI and LDIR can be used to transfer blocks of data by moving a series of bytes.

2. A LD instruction must include two operands. The first operand represents the location where data will be stored (register or memory section). This is called its "destination". The second operand represents the original location of the data to be transferred. This is called the "source". For instance, LD A,B indicates that data in register B will be transferred to register A. Register A is the "destination" and Register B is the "source".

3. The direction of data transfer may be:

```
     (1) register <- register   e.g.  LD   A,B      ; LD HL,BC
     (2) register <- memory     e.g.  LD   A,(HL)   ; POP AF
     (3) register <- immediate data
                                e.g.  LD   A,25H    ; LD HL,125AH
     (4) memory <- register     e.g.  LD   (HL),A   ; PUSH BC
     (5) memory <- memory       e.g.  LDD           ; LDIR
     (6) memory <- immediate data
                                e.g.  LD   (HL),5BH
```

II. Experiment 1-1

Write an assembly language program to set the contents of the registers as follows : A=0, B=1, C=2, D=3, E=4, H=5, L=6 (use 8-bit LD instructions to transfer one byte of data each time).

Step 1 Write the assembly language program in the following blank form. The last instruction is RST 38H which returns control of the MPF-I to the monitor program after executing the whole program.

Step 2 Using the table of 8-bit LD instructions, translate the program into machine language with the starting address at 1800H. Assign the proper address to each instruction.

Step 3 Prepare the MPF-I microcomputer. Key in the program from the keyboard. Check the program stored in memory. Set the PC (program counter) to the starting address 1800H and execute the program.

Step 4 Press the REG key and check if the content of each register is correct. If there is any error then return to step 1 and recheck.

Memory Address	Machine Language	Assembly Language
1800H	3E00	LD A,0
.	.	.
.	.	.
.	.	.
	FF	RST 38H

III. Experiment 1-2

Write an assembly language program to set the contents of registers as follows: B=12, C=34, D=56, E=78, H=9, L=A (use 16-bit LD instruction to transfer two bytes of data each time).

Step 1 Same as in Experiment 2-1-1 (Write an assembly language program).

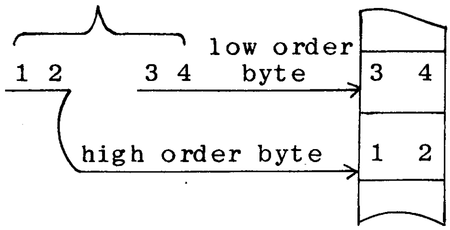
Step 2 Using the 16-bit LD instruction table, translate the program into machine language with starting address at 1820H. Assign the proper address to each instruction.

Step 3 Load the program (same as Experiment 2-1-1). Set the PC to 1820H and execute the program.

Step 4 Check contents of each register same as Experiment 2-1-1.

Note A 16-bit piece of data is composed of two bytes of data. The high-order byte is in the higher memory address and the low-order byte is in the lower meory address. For instance, the 16-bit data 1234H is stored in addresses 1820H - 1821H in the following way:

16-bit data • memory contents memory address



1820H (lower address)

1821H (higher address)

Address	Machine Language	Assembly Language
1820H	013412	LD BC,1234H
1823H		
		RST 38H

Example :

Write a program to clear the contents of memory addresses 1850H - 186FH.

Explanation:

(1) If we use an 8-bit LD instruction to transfer the data to each destination, then 32 (20H) executions of data-transfer is required. It is more convenient to use the loop method in the program.

(2) Use register B as a loop counter. Set register B equal to 20H before the loop program is executed. Use HL as the memory address pointer and set HL to the starting address 1850H. HL is incremented by one and B is decremented by one for each loop. If B=0, then all loops have been executed; otherwise, run the loop again.

(3) The program is given below:

Address	Machine Language	Label	Opcode & Operand		Comment
1800			LD	B,20H	; Set loop counter equal to 32
			LD	HL,1850H	; Set HL equal to the starting address
					; of memory to be cleared
			XOR	A	; Set A=0
		LOOP	LD	(HL),A	; Load 0 into the memory address
					; pointed to by HL
			INC	HL	; Increment HL by 1
			DEC	B	; Decrement B by 1
			JR	NZ,LOOP	; If B not = 0, return to LOOP
	FF		RST	38H	; Return to the monitor program

IV. Experiment 1-3

Translate the program in Example 1-1 into machine language and load it into MPF-I RAM. Then, execute the program and check if the contents of 1850H - 186FH have been cleared. If not, correct the program and execute it again.

V. Experiment 1-4

Write an assembly language program to set the contents of memory address 1840H - 184FH as follows: 0, 1, 2, 3, ...F.

(HINT: Change the loop counter and the value of the starting address. register A is incremented by '1' in the next loop)

ADDRESS	MACHINE LANGUAGE	LABEL	OPCODE & OPERAND

Experiment 2
Basic Applications of Arithmetic and Logic Operation Instructions

Purposes:

1. To familiarize the user with the arithmetic and logic operation instructions
2. To understand the memory addressing mode
3. To understand the meaning of the register status flag
4. To practise arranging data for CPU registers and memory sections

Time Required: 4 hours

I. Theoretical Background:

1. 8-bit arithmetic and logic operation instructions:

The 8-bit arithmetic and logic operations in the Z80 CPU are performed in register A (accumulator). Registers A, B, C, D, E, H, and L can be used as operands in conjunction with register A in the LD instructions. If data are transferred between memory and register A, the memory address can be pointed to by HL, IX or IY registers. The meaning of the following instructions are given in the right-side comment field:

(1)	ADD A	; Data in register A is added to itself, i.e. the data is doubled shifted left one bit.
(2)	ADC B	; Register B and the carry flag are added to register A.
(3)	SUB C	; Data in register C is subtracted from register A.
(4)	SBC (HL)	; Subtract the data in the memory address pointed to by HL and the contents of the carry flag from register A.
(5)	AND D	; Logical "AND" of register D and register A.
(6)	OR OFH	; Logical "OR" of data OFH and register A .
(7)	XOR A	; Exclusive "OR" register A and itself. (Since register A is equal to register A, the result is zero).
(8)	INC H	; Increment the contents of register H by 1.
(9)	INC (IX)	; Increment the contents of the memory address pointed to by register IX by 1.
(10)	DEC C	; Decrement the contents of register C by 1.

(11) DEC (IY+3)

; The sum of the contents of register IY and 3 is used as the memory address pointer. Decrement the contents of memory address IY +3.

2. Data Addressing Mode

In the above assembly language instructions, the addressing modes used can be summarized below. Other addressing modes can be found in the Z80 CPU technical manual.

(1) Register Addressing

Example: In the instruction ADC A,B , ADC is the opcode which represents what kind of operation will be performed. The character A in the right means that the data will be added to A. The character B at the far right means that the data to be added to A is taken from register B.

(2) Register Indirect Addressing

A 16-bit register is used to store the memory address.
Example: In the instruction SBC A,(HL) , (HL) does not mean that HL will be subtracted from register A. Instead, the CPU takes the 16-bit data contained in HL as the memory address and then accesses the 8-bit data stored in this memory address. The 8-bit data pointed to by HL is finally subtracted from register A. IX and IY are called index registers. When a memory address is pointed to by IX or IY, an 8-bit byte which is less than +127 but larger than -128 can be added to this register.

For instance, the following two instructions can be used to add the data stored in the memory address pointed to by IX to the 8-bit data stored in the memory address pointed to by IX+2. The result is stored in register A.

```
LD      A,(IX)
ADD     A,(IX+2)
```

(3) Immediate Addressing

Example : OR OFH. On the right-hand side of the opcode OR, a hexadecimal number, OFH, is given. It means that the number OFH is logically ORed with the contents of register A. Therefore, the data is part of the instruction which is stored in memory. The CPU fetches the data by using the program counter (PC) as a reference address. The following instructions are examples of immediate addressing.

```
LD      B,8
ADD     A,44H
SUB     A,0A4H
```

3. Status Flags

After a logical or arithmetic operation is finished, the result will be stored in register A and some of the status flags (Carry, Overflow, Change Sign, Zero Result, Parity) will also be affected. These status flags will be stored in the flip flops in the Z-80 CPU. These flip flops form a register called the Flag Register. The data in this register can be moved to memory, like data in other registers, by specific instructions (PUSH instruction). Some of the status flags are given below.

(1) Carry Flag

This flag is the carry from highest order bit of the Accumulator. The carry flag will be set in either a signed or unsigned addition where the result is larger than an 8-bit munber. This flag is also set if a borrow is generated during a subtraction instruction. The carry flag can be used as a condition for jump, call, or return instructions. The carry flag also serves as an important linkage in multi-byte arithmetic operations. Three 8-bit data can be connected as a 24-bit data by using carry flag and four 8-bit data can be connected as a 32-bit data.

(2) Overflow/Parity Flag

When signed two's complement arithmetic operations are performed, this flag represents overflow. The Z-80 overflow flag indicates that the signed two's complement number in the accumulator has exceeded the maximum possible (+127) or is less than minimum possible (-128).

When an arithmetic operation is performed in the Z80-CPU, the number in register A can be assumed to be unsigned data (0 - 255) or signed data (-128 ~ +127). Thus, either the carry flag or the overflow flag can be affected by the arithmetic operation. The programmer decides which interpretation is desired. The following arithmetic operations are described on the right-hand side.

```
      10101100       <- unsigned number 172 or signed number -84
    +) 11101000      <- unsigned number 232 or signed number -24
  ----------------
  1 <- 10010100      <- unsigned number 148 with carry or signed
                        number -108 but no overflow
```

```
     01001010        <- signed or unsigned number 74
   +) 01000010        <- signed or unsigned number 66
  --------------
 0 <- 10001100        <- unsigned number 140 but no carry, or
                         signed number -116 but overflow has
                         occurred and the result becomes negative
      change sign
```

For logical operations in the Z80-CPU, this flag is set if the parity of the 8-bit result in the accumulator is even. This flag is very useful in checking for parity errors occurring during data transmission. Since carry and overflow will never occur in logical operations, the parity and overflow status can be stored in the same flip flop. This flip flop is called the P/V flag. By testing this flip flop the programmer can check overflow after arithmetic operations and check parity after logical operations.

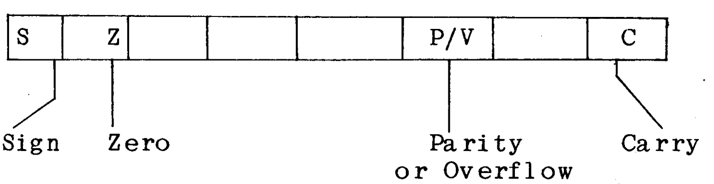
(3) Zero Flag

If register A is zero after a logical or arithmetic operation, this status will be registered in a flip flop called zero flag. The zero flag can be used as a condition for branch instructions. It is very useful in program looping.

(4) Sign Flag

If the leftmost bit (bit 7) of register A is 1 after a logical or arithmetic operation, the number in register A is interpreted as a negative number. The sign flag is then set to 1. This flag will be ignored if the programmer has assigned the data as unsigned numbers.

(5) The other flags designed for BCD arithmetic are not important for the programmer. The bit positions of the flags discussed above are shown below:



In microcomputers, it is usual to represent the contents of the flag register by two hexadecimal digits. The reader reader has to express this two-digit data with an 8-bit binary number. By referring to the bit positions in the flag register, the reader can obtain the status of the flag. For instance, if the flag register is 3CH, then the sign is positive, the value is non-zero, the parity is even or there is overflow has occurred but there is no carry. To know which flags will be affected by an instruction, the reader has to refer to the assembly language manual. Not all instructions will affect the status flags.

II. Example of Experiments

1. The following program can be used to add the contents of register D and register E together. The result will be stored in the pair register HL. Load the program into MPF-I and then execute it. Record the result.

```
ORG     1800H    ; Starting Address <- 1800H
LD      A,E      ; A <- E
ADD     A,D      ; A <- A + D
LD      L,A      ; L <- A
LD      A,0      ; A <- 0
ADC     A,0      ; A <- A + 0 + Carry
LD      H,A      ; H <- A
RST     38H      ; Return to Monitor
```

Preset Value		Result of Program Execution				
Register		Register	Flag			
D	E	HL	Sign	Zero	P/V	Carry
5AH	A6H					
46H	77H					

2. The following program can be used to add the 16-bit data in memory addresses 1A00H - 1A01H to the 16-bit value in the register pair DE. The result will be stored in the register pair HL. Load the program into MPF-I and execute it. Discuss the result obtained.

   preset values of memory: (1A01H) = __________,(1A00H) = __________
   preset value of register DE pair = __________, __________

```
ORG     1800H        ; Starting address <- 1800H
LD      A,(1A00H)    ; A <- (1A00H)
ADD     A,E          ; A <- A + E
LD      L,A          ; L <- A
LD      A,(1A01H)    ; A <- (1A01H)
ADC     A,D          ; A <- A + D + Carry
LD      H,A          ; H <- A
RST     38H          ; Return to monitor.
```

Result:

result	HL	= ________	,	________
	Carry	= ________	,	________
	Zero	= ________	,	________
	Overflow	= ________	,	________
	Sign	= ________	,	________

3. Revise the above program for a subtraction operation.

4. The following program can be used to add the 32-bit data in memory addresses 1A00H - 1A03H to the 32-bit data in memory addresses 1A04H - 1A07H. The result will be stored in memory addresses 1A08H - 1A0BH. The higher-order byte is stored in a higher address (This is conventional in microcomputer programming)

preset memory contents: (1A03H - 1A00H) = ________________

(1A07H - 1A04H) = ________________

```
        ORG     1800H
        LD      B,4
        LD      IX,1A00H
        AND     A
LOOP    LD      A,(IX)
        ADC     A,(IX+4)
        LD      (IX+8),A
        INC     IX
        DEC     B
        JP      NZ,LOOP
        RST     38H
```

Result of program testing:

results of program execution: (1A0BH - 1A08H) = ________________

Flag Register = ________________

5. If the instruction ADC A,(IX+4) is replaced by SBC A, (IX+4), then the above program can be used for a subtraction operation. If the instruction DAA is inserted immediately after the ADC or SBC instruction, then the program becomes a program for decimal addition or subtraction. Load the revised program to MPF-I and test it.

Experiment 3

Binary Addition and Subtraction

Purposes:

1. To understand how an addition or subtraction operation is performed on a microcomputer.
2. To familiarize the reader with software programming techniques.

Time Required: 4 hours

I. Theoretical Background:

1. In this experiment, we only discuss unsigned binary integer addition and subtraction. For a N-bit binary number, its range is < 0,2 -1 >. For instance, if N = 8, the range is < 0,255 >; if N = 16, the range is < 0,65535 >. If the range of the numbers are expressed by hexadecimal digits, the ranges are < 0,FFH > and < 0,FFFFH >, respectively. If the sum of an addition operation is larger than the maximum value that can be represented by N bits, then carry is generated and the carry flag is set. In the subtraction operation, if the subtrahend is more than the minuend, a borrow is generated and the carry flag is set in the high order byte. The set carry bit indicates an incorrect result.

Example 3-1:

Single byte addition and subtraction.

Addition: 7FH + ADH = 12CH

```
     01111111 -> 7FH
 +)  10101101 -> ADH
---------------------
    100101100 -> 12CH
    ↑
  Carry
```

```
Subtraction: 7FH - ADH          Subtraction: ADH - 7FH = 2EH

          01111111                          10101101
       -) 10101101                       -) 01111111
      -------------                     -------------
         111010010                         000101110
         ↑                                 ↑
       Borrow                            Borrow
   The answer is incorrect           The answer is correct
   (   CY = 1 )                      (   CY = 0 )
```

Example 3-2

Three-byte addition and subtraction

Addition: 6A7CBDH + 4B65ACH = B5E269H

```
          6A            7C            BD
          4B            65            AC

      + [0]<-┐      + [1]<-┐      + [0]   <- Carry
      ------  |     ------  |     ------
     [0] B5   |    [0] E2   |    [1] 69
      ↑       └─────┘       └─────┘
    Carry         Carry         Carry
```

Subtraction: 854372H - 69ACBFH =

```
          85            43            72
        - 69          - AC          - BF
       ------        ------        ------
        0 1B          1 97          1 B3

      - [1]<-┐      - [1]<-┐      - [0]   <- Borrow
      ------  |     ------  |     ------
        0 1A  |    [1] 96   |    [1] B3
        ↑     └─────┘       └─────┘
      Borrow        Borrow        Borrow
```

The borrow of the highest-order byte is 0, thus the answer is correct. In multi-byte subtraction, the correctness of the result depends upon the borrow of the highest-order byte. If the borrow is 1, then the result is incorrect.

2. Order of data stored in memory:

The conventional way of storing multi-byte data in memory is: the lowest order byte is stored in the lowest address and the highest order byte is stored in the highest address. The address of the multi-byte data is usually expressed by its lowest address. For beginning atstance, the number 7325H is stored beginning at memory address A in the following way:

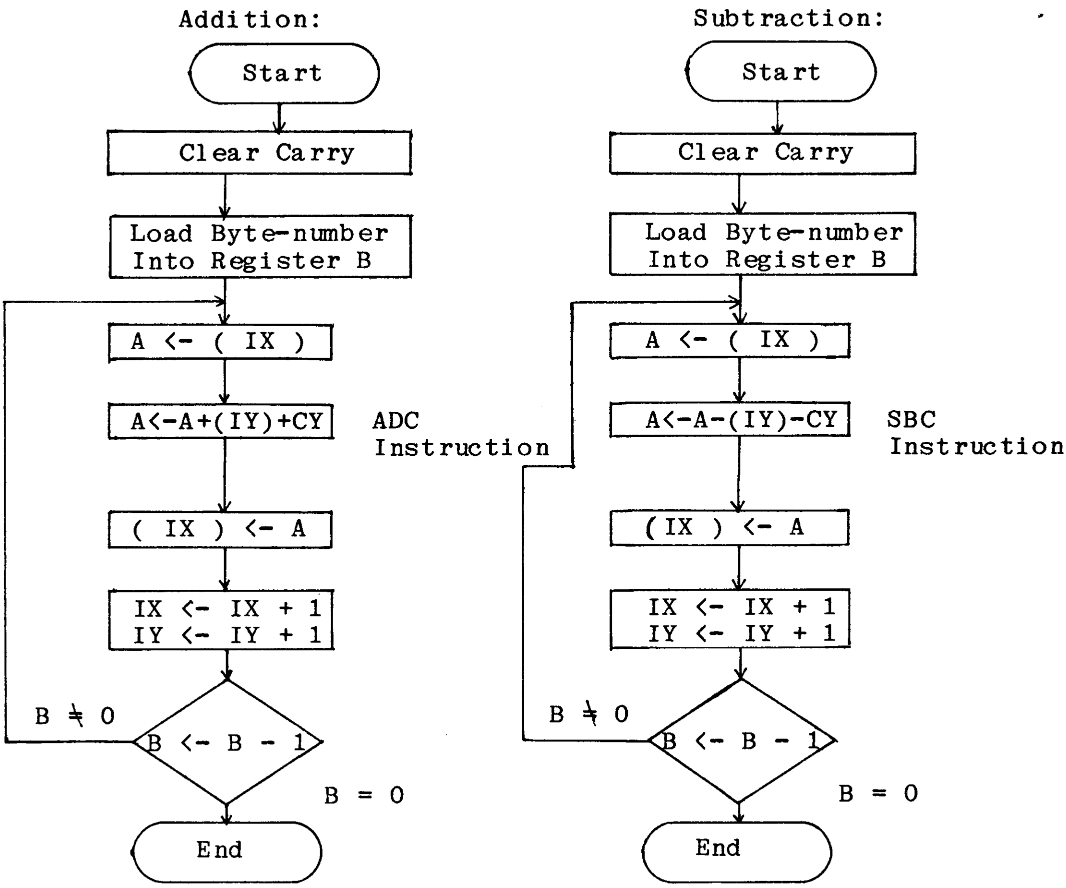
address A	25	<- low-order byte
A + 1	73	<- high-order byte

If the starting address of 4 three-byte numbers stored in memory is A, the data and their addresses can be shown as follows :

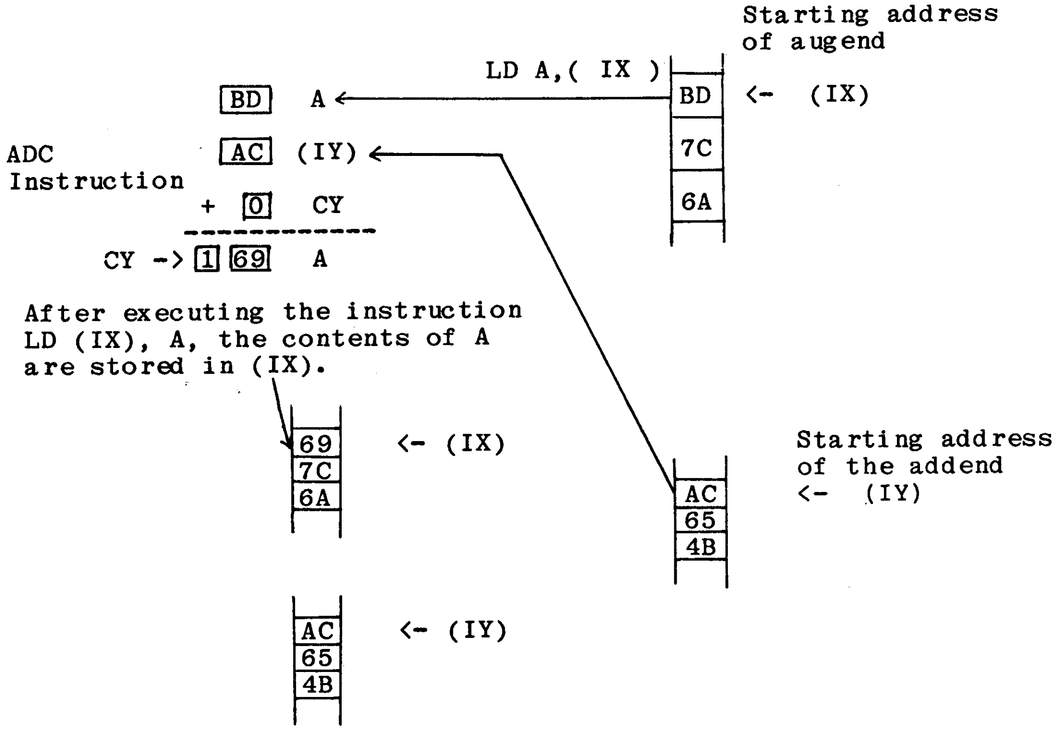
Address	Data	
A	56	987C56H
	7C	
	98	
A + 3	43	AD6943H
	69	
	AD	
A + 6	BC	2501BCH
	01	
	25	
A + 9	78	439578H
	95	
	43	
A + 12	21	....
	96	

3. Design of Addition/Subtraction Programs:

The data used in addition/subtraction operation are stored in memory according to the conventional method given above. The starting address of the augend/minuend is stored in index register IX. The starting address of addend/subtrahend is stored in index register IY. The byte-number of the data is stored in register B. First, clear CY and load the augend/minuend into the accumulator. Then, use the indexed addressing mode instruction ADC (SBC) to proceed with the addition/subtraction operation. The result is stored in the original address of the augend/minuend. Increment the index registers and compare register B with zero. Repeat the load augend, add, store increment cycle until the B register equals zero. Finally, test the carry flag to check if the result is correct. The only difference between the addition program and subtraction program is that the instruction ADC is used for addition operation and the instruction SBC is used for subtraction operation. The flowcharts and programs are given below for comparison:



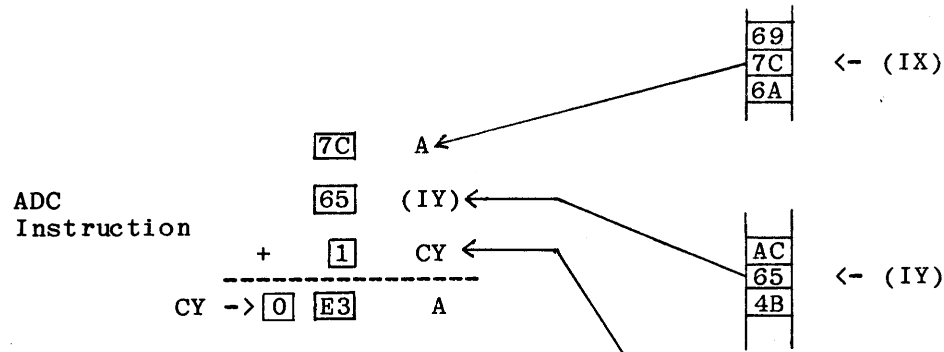
The following block diagram is given to demonstrate data transfer in an addition operation.



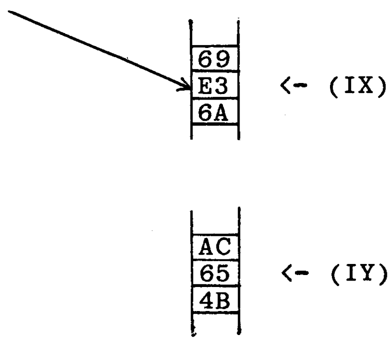
Instruction INC IX increases the value of IX by one. In the comment field the incrementation of IX can be shown as IX   IX + 1
INC IY leads to IY <- IY + 1

In each of frames showing the results of an instruction step the current value pointed to by the index registers are indicated by

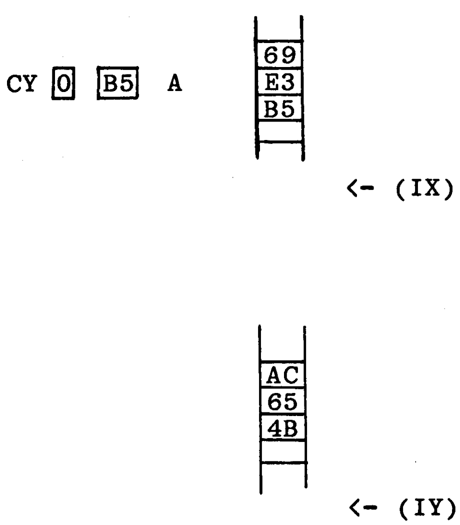
ADC
Instruction



After the instruction LD (IX), A
is executed, the memory becomes



When B = 0, the program execution is finished
and the memory becomes



The addition program is given below. By replacing the instruction ADC A, (IY) by SBC A, (IY), the addition program becomes a subtraction program.

```
 1. *** MPF-I EXAMPLE PROGRAM ***
 2. 3-BYTE ADDITION ( UNSIGNED INTEGER )
 3. ENTRY ; AUGEND ADDRESS IN IX,
 4.         ADDEND ADDRESS IN IY.
 5. EXIT  : SUM IN AUGEND ADDRESS
 6.
 7. ADD3  : XOR A ; CLEAR CARRY FLAG
 8.         LD B, 3 ; BYTE NUMBER IN B
 9. ADDLP : LD A, (IX)
10.         ADC A, (IY)
11.         LD (IX), A
12.         INC IX
13.         INC IY
14.         DJNZ ADDLP
15.         RET
```

4. Programming Technique:

From the above examples (3-1 and 3-2), we can see that the multibyte addition/subtraction operation can be accomplished by repeating the single-byte addition/subtraction operation, that is, by the loop operation of single-byte addition/subtraction. In the above program, register B is used as a loop counter. If the byte-number is 4, then 4 is loaded into B initially. Register B is decremented by 1 after each loop operation. The loop ends when B = 0. The instruction DJNZ is used for conditional jump. When B = 0, the program no longer executes the jump operation. Since ADC and SBC instructions are used in the programs, the CY is included in each addition/subtraction operation. Therefore, before the first byte addition/subtraction operation, the carry flag must be cleared (instruction XOR A). The index registers IX and IY are used as address pointers. By incrementing IX and IY, the CPU can access multibyte values stored in memory.

II. Student Exercises:

1. Load the above addition program into MPF-I and store it on magnetic tape.

2. Replace the last instruction RET in the program by RST 38H. Load the following data into memory. The starting addresses of augend and addend are assigned as 1900H and 1A00H, respectively. Execute the program and record the result in the following table.

Augend	Addend	Answer	Check
793865H	ABCEDFH	CY =	
009543H	AB1236H	CY =	
954717H	003390H	CY =	

3. Replace the ADC instruction by the SBC instruction. Assign the starting addresses of minuend and subtrahend as 1900H and 1A00H, respectively. Execute the program and record the results obtained.

Minuend	Subtrahend	Answer	Check
683147H	336700H		
5935ABH	5877FFH		
049677H	F65B79H		

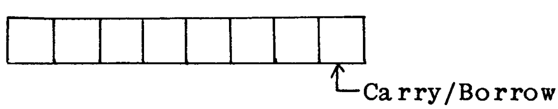
4. Express the data in the above two tables as five-byte data. Change the byte-counter to the proper value and execute the addition/subtraction program.

5. Write a program to add the 7-byte data in memory addresses 1A00H - 1A06H to the 7-byte data in memory addresses 1900H - 1906H and then subtract the 7-byte data in memory addresses 1940H - 1946H from the sum. The final result must be stored in memory with the starting address 1900H.

Experiment 3-1:

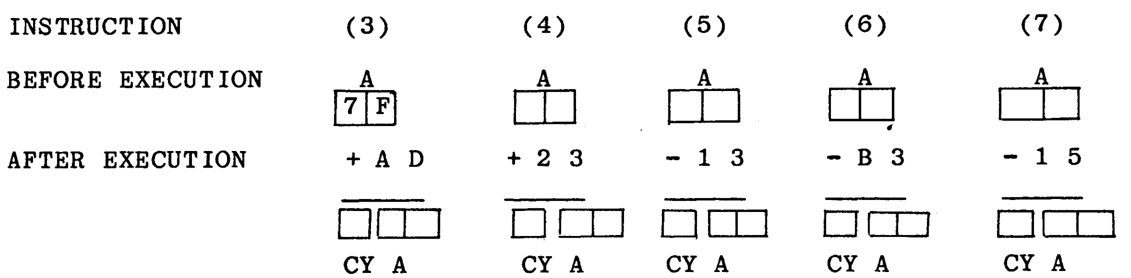
The carry/borrow flag is used to indicate whether a carry/borrow is generated during an arithmetic or logical operation. If a carry/borrow is generated, then the flag is set to 1. Otherwise, the flag is zero. The carry flag is represented by bit 0 of the flag register.

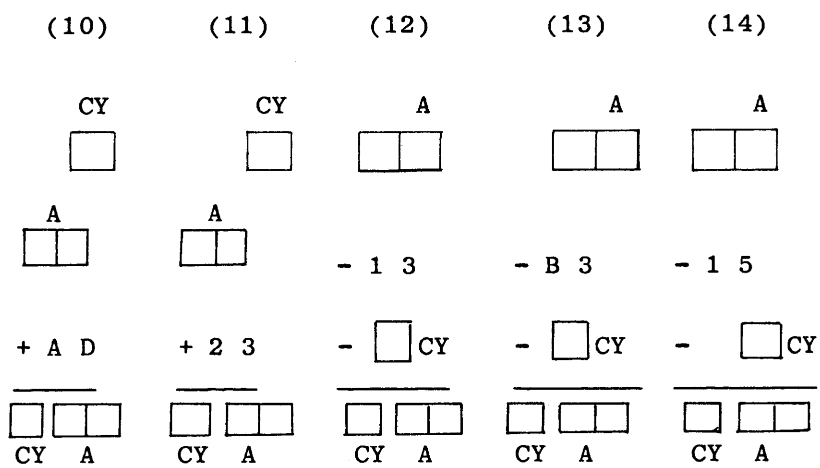
REG.F



In other words, the contents of the F register will be an even number if a carry/borrow is generated during the arithmetic or logical operation. If register F is an odd number, then no carry/borrow has been generated. Load the following program into MPF-I. Execute every instruction by using the Single Instruction method. Observe the variations of register F and record the results in the table.

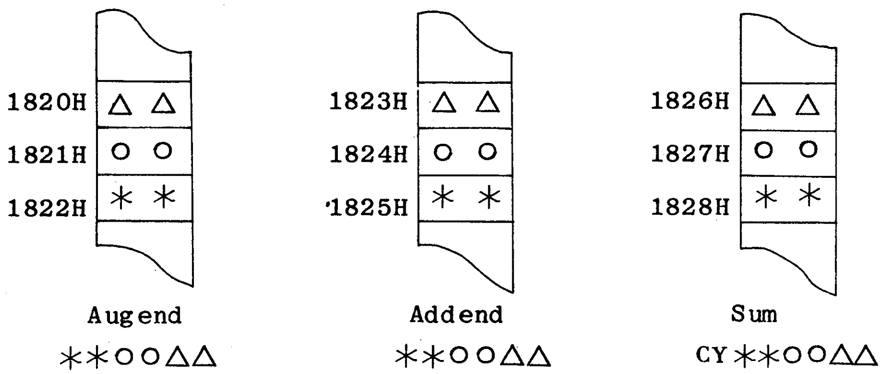
Address	Machine Language		Assembly Language	
·1800H	AF	1	XOR A	A,CY <-- 0
1801H	3E7F	2	LD A, 7FH	A <-- 7FH
1803H	C6 AD	3	ADD A, ADH	CY,A <-- A + ADH
1805H	C6 23	4	ADD A, 23H	CY,A <-- A + 23H
1807H	D6 13	5	SUB A, 13H	CY,A <-- A - 13H
1809H	D6 B3	6	SUB A, B3H	CY,A <-- A - B3H
180BH	D6 15	7	SUB A, 15H	CY,A <-- A - 15H
180DH	AF	8	XOR A	A,CY <-- 0
180EH	3E7F	9	LD A, 7FH	A <-- 7FH
1810H	CEAD	10	ADC A, ADH	CY,A <-- A + ADH + CY
1812H	CE23	11	ADC A, 23H	CY,A <-- A + 23H + CY
1814H	DE13	12	SBC A, 13H	CY,A <-- A - 13H - CY
1816H	DEB3	13	SBC A, B3H	CY,A <-- A - B3H - CY
1818H	DE15	14	SBC A, 15H	CY,A <-- A - 15H - CY
181AH	FF	15	RST, 38H	





Experiment · 3-2:

Referring to the operation for of 3-byte addition in example 3-3-2, write a basic addition program using only three kinds of instructions: XOR A, LD A,(nn) and ADD A,(nn). Assume that the memory addresses of the addend, augend and sum are assigned as follows:



Explanation: In the above example, we see the following rules of addition:

(1) The addition operation moves from the low-order byte to the high-order byte, the carry generated in the low-order byte addition is added to the next higher order byte.

(2) The addition operation is executed with the aid of the accumulator. Its result is also stored in the accumulator. Thus to add two bytes together, one byte must be loaded into the accumulator first (using the LD A,(nn) instruction). The other byte is then added to the accumulator (using the ADD A,(nn) instruction or the ADC A,(nn) instruction). The final result is stored in an assigned memory address (using the LD(nn),A instruction).

Experiment 4
Branch Instructions and Program Loops

Purposes:
1. To familiarize the reader with the applications of conditional and unconditional branch instructions.
2. To familiarize the reader with the technique of designing program loops.
3. To practice using status flags in decision-making.

Time Required: 4 hours

I. Theoretical Background:

1. Program Counter:

The program counter (PC) is an important 16-bit register in the CPU. When the voltage level of the RESET pin (pin 26) of the CPU drops to 0 and then rises to 1 (by pressing the RS key), the PC will be cleared to 0000H. The program execution is then started from address 0000H according to the clock pulses supplied by the system hardware. Once the CPU has fetched one byte of each instruction from memory, the PC will be incremented by one automatically. (The internal control circuit in the CPU determines how many bytes are contained in the instruction after the CPU has fetched the first byte of the instruction. The instruction will be executed only when the PC has been incremented by the number of bytes in the instruction). Usually, the program is fetched from the memory instruction by the instruction for execution, starting from the low memory address.

2. Branch Instructions:

At any address, the PC can be changed to another address if the programmer doesn't want the program executio to continue sequentially (For instance, when there is no memory beyond that address or the program is not stored in that area). The program then jumps to another address and continue its execution. For example, the following assembly language means that the PC will be changed to 1828H after this instruction has been executed, and the program execution continues from address 1828H.

```
LD PC, 1828H  (This instruction is illegal in Z80 assembly
               language)
```

Actually, in assembly language, JP (Jump) is used to indicate the change in sequence of program execution. The instruction has the same meaning as:

```
LD  PC, 1828H
JP  1828H
```

3. Conditional Branch Instructions:

A conditional branch instruction performs the jump operation if some specified conditions are met. These conditions are all dependent on the data in the flag register. This function makes the microcomputer capable of responding to various external conditions. It is also an indispensable tool for designing program loops. The actions of the following instructions are described in the comments to the right of the instruction:

```
CP   10H           ; Compare the accumulator with 10H and
                     set the proper flag.

JP   Z,   1828H    ; If the zero flag is set, i.e. A = 10H,
                     then jump to address 1828H and continue
                     the program execution.

JP   C,   245AH    ; If the carry flag is set, i.e. A < 10H,
                     then jump to 245AH to execute other
                     program.

ADD  A,B           ; Otherwise, i.e. A > 10, continue the
                     program execution.
```

The condition of a conditional branch instruction is written after JP:

```
(1) JP  C, XXXX    ; If there is a carry, or carry flag = 1,
                     then jump to XXXX.

(2) JP  NC, XXXX   ; If there is no carry, or carry flag = 0
                     then jump to XXXX.

(3) JP  Z, XXXX    ; If zero flag = 1, or the result of previous
                     operation is zero, then jump to XXXX.

(4) JP  NZ, XXXX   ; If zero flag = 0, then jump to XXXX.

(5) Jp  PE, XXXX   ; If parity flag = 1 (even), or there was
                     an overflow in the previous arithmetic
                     operation, then jump to XXXX.

(6) JP  PO, XXXX   ; If P/V flag = 0 (odd parity or no overflow)
                     then jump to XXXX.
```

(7) JP P, XXXX ; If sign flag = 0 (the sign of result of previous operation is positive) then jump to XXXX.

(8) JP M, XXXX ; If sign flag = 1 (negative) then jump to XXXX.

4. Jump Relative:

To reduce the memory space occupied by the program and also reduce the cost of the microcomputer system, the Z80 microcomputer can use relative addresses to specify the displacement of a program jump. Since most displacements in a jump are within the rage between +127 and -128, a one byte number can be used to indicate this displacement. One byte of memory is saved for each jump operation compared with the two-byte absolute address in JP instructions. The operations of the following instructions are described in the commands to the right of the instruction.

JR 10H ; Jump forward 10H (16) locations from the present program counter (the address of the next instruction). Actually, the address of the next instruction to be executed is obtained by adding 10H to the present PC.

JR C,FOH ; If carrry flag = 1, then jump backward 10H (16) locations from the present program counter. Since the leftmost bit of FOH is 1, it is recognized as a negative number (its 2's complement is 10H).

JR NC,7FH ; If carry flag = 0, than jump forward 127 locations (maximum value)

JR Z,80H ; If zero flag = 1, i.e. the result of the previous operation is zero, then jump backward 128 locations. 80H (-128) is the minimum negative number that can be used in a relative address.

From the above examples, we can see that a positive relative address means jumping forward. The largest displacement then is 7FH (+127). A negative relative address means jumping backward. Its largest displacement is 80H (-128). The displacement is always measured from the address of the next instruction's op code. Relative jumps can be unconditional or conditional. The conditional jump depends on the status of the carry or zero flag. In the Z80 system, the data in the sign or P/V flag cannot be used as the condition of a relative jump.

5. Program Loop:

One of the important advantages of a computer is that it can repeat the steps in a repetitive task as many times as is necessary to complete the task. This is accomplished by using a program loop. Looping is a very powerful tool in program design. A basic program loop must contain the following:

(1) A loop's counter preset with the number of loops to be executed. Usually, a CPU register is used as a loop counter. Of course, memory can also be used as a counter.

(2) The loop counter is decremented by 1 after one cycle of the loop has been executed. After each cycle the value of the loop counter must be checked. If the counter is not 0, then the loop repeats until the loop counter equals to 0.

The following program can be used to add the 8-bit data in memory addresses 1900H - 190FH and store the result in the DE register pair. This is a typical application of a program loop.

```
    LD C,10H       ; Use register C as the loop counter. Since
                     sixteen bytes data are to be added together,
                     10H is preset in C.

    XOR A          ; Clear the accumulator

    LD HL,1900H    ; Use the HL register pair as the address pointer.
                     The contents of the memory pointed to by HL
                     will be added to register A.
                     The first address is 1900H.

    LD D,A         ; Register D is used to store the carry
                     generated during the addition operation.
                     Clear Register D.

XX  ADD A,(HL)     ; Add the contents of the memory address pointed
                     to by HL to Register A. This instruction will
                     be repeated 16 times. XX is assigned
                     as the label of this instruction's address.

    INC HL         ; Increment HL by 1. The new HL points to the
                     next byte in data memory to be added to
                     Register A.
    JR NC,YY       ; If no carry is generated, jump to address
                     YY to continue program execution.

    INC D          ; If a carry is generated, add this carry to
                     Register D.

YY  DEC C          ; Decrement register C by 1.

    JR NZ,XX       ; If the result is not zero (zero flag = 0),
                     the program loop has not finished. Jump to
                     XX to repeat the loop.
```

```
LD E,A              ; If zero flag = 1, then all data have been
                      added together. Load A into E, the answer
                      will be stored in the DE register pair.

END
```

There are various methods of designing a program loop. Try to design the program loops described in the following illustrations.

II. Example Experiments:

1. A program loop with a loop number of less than 256 : If the loop number is less than 256, register B is recommended as the loop counter. At the end of the loop, the DJNZ instruction can be used to decrement register B. If the result is not zero, jump to the assigned location using the relative jump method to continue the program execution. Try to analyze the following program and verify its function by loading it into the MPF-I and executing it.

```
             ORG     1800H
             LD      HL,1900H
             LD      B,20H
┌──>LOOP     LD      (HL),A
│            INC     HL
└────────────DJNZ    LOOP
             RST     38H
```

Experimental result:

(1) Preset register A to 0 and then execute the above program
Results:
Contents of memory addresses 1900H - 191FH:
Contents of memory address 1920H:

(2) Preset register A to 55H and execute the above program.
Results:

(3) Preset register A to 64H and replace the second instruction LD B,20H by the instruction LD B,0 . Execute the program again.
Results:
Contents of memory addresses 1900H - 19FFH:

Discussion:

2. Nested loops:

In a more complicated program, a loop can be totally nested or embedded inside another loop. The following program can be used to divide the 256 bytes of data stored in memory into 16 groups. The starting address of the memory is 1900H. Put the contents of each group of data in the form of a hexadecimal number:

0.....(1st set), 1.....(2nd set), 2.....(3rd set),....,F.....
(16th set).

```
                     LD      HL,19FFH
                     LD      C,0FH
┌──────────> LOOP2   LD      B,10H
│  ┌───────> LOOP1   LD      (HL),C
│  │                 DEC     HL
│  │ Small loop     ↗DJNZ    LOOP1
│  └──────────────   DEC     C
│ Large loop        ↗JP      NZ,LOOP2
└─────────────────   RST     38H
```

(1) Translate the above program into machine language and then load it into the MPF-I. Execute the program.
Result :

(2) Revise the above program such that the 16 bytes of the first group are all "F", and the 16 bytes of the last group are all "0".

3. A program loop with loop number larger than 256: If the loop number is larger than 256, a 16-bit register can be used as the loop counter. But, in the Z80 system, incrementing or decrementing a 16-bit register can not affect the status flag. Thus, some auxiliary instruction is used to determine whether the loop counter is zero. The following program is supposed to be able to set all data in RAM 1880H - 19FFH to AAH. Try to find the errors in this program and correct them. Load the correct program into the MPF-I and record the result of the program execution.

```
         ORG     1800H
         LD      BC,0180H
         LD      HL,1880H
LOOP     LD      (HL),0AAH
         INC     HL
         DEC     BC
         JR      NZ,LOOP
         HALT
```

4. A program loop without a down counter : A program loop need not use a down counter. The function of the down counter can be replaced by using an up counter or using the method of address comparision or data comparison. Study the method used in the following program loops. Load the programs into MPF-I and execute them.

(1) Move the data string in the memory (RAM) section with starting address 1B00H to the memory (RAM) section with starting address 1A00H. The movement will be terminated when data 0FFH is found.

```
        ORG     1800H
        LD      HL,1B00H
        LD      DE,1A00H
LOOP    LD      A,(HL)
        LD      (DE),A
        CP      0FFH
        JR      Z,EXIT
        INC     HL
        INC     DE
        JR      LOOP
EXIT    RST     38H
```

(2) Replace all the data stored in the memory section starting from the address pointed to by HL to the address pointed to by DE with their corresponding 2's complement. In testing the program, the values of HL and DE must be preset first. The value of HL must be larger than that of DE.

```
        ORG     1800H
LOOP    LD      A,(HL)
        NEG
        LD      (HL),A
        INC     HL
        AND     A
        SBC     HL,DE
        ADD     HL,DE
        JR      NZ,LOOP
```

Experiment 5
Stack and Subroutines

Purposes:

1. To understand the meaning and applications of stack
2. To understand the designing techniques and applications of subroutines.

Time Required: 4 hours

I. Theoretical Background

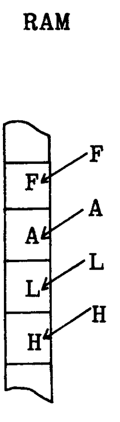
1. Stack: In program design, a stack is recognized as a memory section which has only one port for input and output. Data are written in or retreived from stack via this port. The first item of data placed in stack is said to be at the bottom of stack. The data most recently placed in stack is said to be at the top of stack. Thus, a stack is also called a last-in first-out memory. A stack can be constructed by hardware shift registers or general RAMs. In the Z80 microcomputer system, the programmer can assign a region of RAM as the stack. To define a stack at the top of RAM, the highest address of RAM is incremented by 1 and then loaded into the stack pointer (SP) in the CPU. The following program and diagrams illustrate the operation of stack.

Instruction Number	Instruction	Comment
(1)	LD SP,1FAFH	; Stack pointer is set to 1FAFH, i.e. the RAM section with address less than or equal to 1FAEH is assigned as stack.
(2)	DEC SP	; Decrement SP by 1. Stack pointer is at 1FAEH, i.e. at the bottom of stack.
(3)	LD (SP),H	; Load the contents of register H into memory (RAM) address 1FAEH.
(4)	DEC SP	; Decrement SP by 1 again.
(5)	LD (SP), L	; Place the contents of L at the top of stack (i.e. above H).
(6)	DEC SP	
(7)	LD (SP),A	; Place the contents of A at the top of stack (i.e. above L).
(8)	DEC SP	

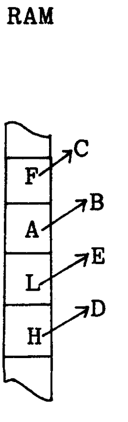
```
(9)   LD (SP), F       ; Place the contents of F at the top of
                         stack (i.e. above A).
      .
      .
      .
      .
(10)  LD C, (SP)       ; Pop one byte of data from the top
                         of stack and move it to register C.
(11)  INC SP           ; Increment SP by 1.  SP is moved towards
                         the top of the stack.
(12)  LD B,(SP)        ; Pop data from the top of stack.
(13)  INC SP           ; Increment SP by 1 again.
(14)  LD E,(SP)        ; Pop data from the top of stack and
                         move it to register E.
(15)  INC SP
(16)  LD D,(SP)        ; Pop data from the top of stack and
                         move it to register D. This data is the
                         first one that is stored in stack.
(17)  INC SP           ;  SP is at the initial value.
```

SP assigned by
the 9th instruction ->

SP assigned by
the 3rd instruction ->
Initial value of SP
1FAFH ->



SP assigned by
the 10th instruction ->
SP assigned by
the 12th instruction ->
SP assigned by
the 14th instruction ->
SP assigned by
the 16th instruction ->
SP assigned by
the 17th instruction ->



Push data onto the stack

Pop data from the stack

From the above illustrations of stack operation, we can see that data can be stored in RAM by using SP as the pointer. SP is decremented by 1 whenever one-byte of data is stored in and the stack area becomes larger. The SP will be incremented by 1 whenever one-byte data is retrieved from the stack area and the stack area becomes smaller. The process of decrementing SP (pushing data onto stack) or incrementing SP (popping data out of stack) can be accomplished automatically by special hardware design. A stack can also be used to store a 16-bit address (or data). In the Z80/8085 system, there are instructions to push a 16-bit register pair onto stack and pop a 16-bit data out of stack. During each operation, SP is decremented or incremented by 2. The following program is equivalent in function to that of the program given above.

```
LD SP, 1FAFH     ; Same as 1st instruction.
PUSH    HL       ; Same as no. (2)(3)(4)(5) instructions.
PUSH    AF       ; Same as no. (6)(7)(8)(9) instructions.
POP     BC       ; Same as no. (10)(11)(12)(13) instructions.
POP     DE       ; Same as no. (14)(15)(16)(17) instructions.
```

Instructions PUSH and POP can be used to temporarily store data in registers and also used to transfer register data. An example is given below.

```
PUSH    BC
POP     IX       ; Move the 16-bit data in BC to IX
PUSH    HL
AND     A
SBC     HL,DE    ; Compare HL with DE to generate status
                   flags. The value of HL is kept
                   unchanged.
```

It is very important to keep the number of PUSH instructions equal to the number of POP instructions in the stack operation.

2. Subroutine:

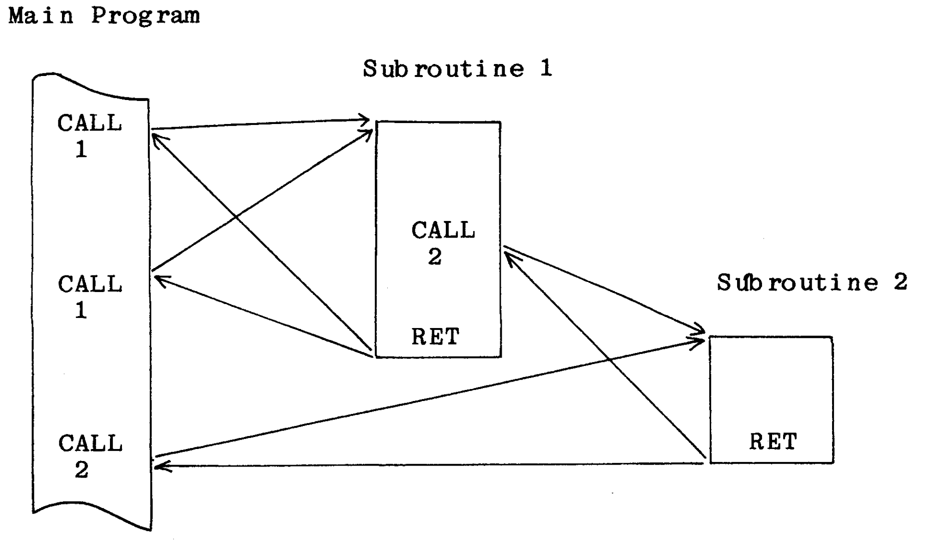
Programs for arithmetic (addition, subtraction, multiplication or division), keyboard and display control, etc are often used as part of a large program in practical applications. If the programmer rewrites these small programs everytime he needs them, the whole program would be very tedious to write. To save memory space for the program and reduce errors, subroutines are often used in a large program. Instructions CALL and RET are used to manipulate the subroutines. The subroutines can be executed unconditionally or according to the conditions of flags. The instruction CALL in the main program is used to call the subroutine. Its function consists of two operations which are illustrated below.

```
┌──→CALL    1A38H    ; Call the subroutine stored in address 1A38H.
│
Equivalent to
│
│   ┌PUSH   PC       ; Push the current program counter onto
└───┤                  stack.
    └JP     1A38H    ; Jump to address 1A38H and continue the
                       program execution.
```

RET instruction doesn't need an operand (1 byte instruction), it is the same as 'POP PC' instruction.

```
┌──RET                  ; Return to original program and continue
│                         to execute.
│
Equivalent to           ; Retrive 16-bit data in stack and load into
│                         PC, then execute program according PC
│                       ; contents.
└──POP PC
```

Calling a subroutine is an important step in a program. Subroutines in a program can be in a nested form that is a subroutine can be another subroutine. The relationship is shown below:



Usually, subroutines are written by a specialist. The user only has to understand its calling procedure . If the subroutine is written by the user himself, the following items must be considered in designing a subroutine:

(1) An easily-remembered name must be chosen for the subroutine.
(2) How to get the data required in the subroutine before executing the subroutine.
(3) How to express the result after executing the subroutine.
(4) Which register will be changed after executing the subroutine.
(5) How much memory will be occupied by the subroutine and how much time is needed for the CPU to execute the subroutine.

The following items must also be considered when a subroutine is called by the main program:

(1) Registers that should not be changed by the execution of the subroutine must be pushed onto stack before calling the subroutine.

(2) How the results obtained from the subroutine execution will be transmitted by the main routine (the calling routine).

The following listing is a sample subroutine named MADD. It can be used for multi-byte BCD addition.

```
                                  MADD    LISTING                  PAGE  1
LOC     OBJ CODE          STMT SOURCE STATEMENT                    ASM 3.0
                             1 ; *** MULTIBYTE BCD ADDITION ROUTINE ***
                             2 ; ENTRY: HL POINTS TO LOW ORDER BYTE OF AUGEND
                             3 ;           DE POINTS TO LOW ORDER BYTE OF ADDEND
                             4 ;           B = BYTE NUMBER, 1 BYTE = 2 BCD DIGIT
                             5 ; EXIT : IX POINTS TO LOW ORDER BYTE OF RESULT
                             6 ; REG. CHANGE : AF,B,HL,DE, IX
                             7 ; MEMORY USED : 15 BYTES
                             8
0000    AF                   9 MADD    XOR A    ; CLEAR CARRY FLAG
0001    1A                  10 MADD1   LD A,(DE)
0002    86                  11         ADD A,(HL)
0003    27                  12         DAA
0004    DD7700              13         LD (IX),A
0007    13                  14         INC DE
0008    23                  15         INC HL
0009    DD23                16         INC IX
000B    10F4                17         DJNZ MADD1
000D    C9                  18         RET

   0 ASSEMBLY ERRORS
```

Two 4-byte BCD data are stored in the memory with starting addresses at 1A00H and 1A40H, respectively. To add these BCD data together and store the result in RAM address 1A08H, subroutine MADD is called by the following procedure:

```
LD      B,  4          ; Set Byte Number = 4 .
LD      HL, 1A00H      ; HL points to the address of augend.
LD      DE, 1A40H      ; DE points to the address of addend.
LD      IX, 1A08H      ; IX points to the address of sum.
CALL    MADD
```

II. Example Experiment:

(1) Using the instructions for stack operation, write a routine to move the data in HL, DE and BC to HL', BC' and DE', respectively. Load the program into MPF-I and execute it.

(2) In the following program, a small loop is embedded in a large loop. The function of this program is to shift all the 8-bit the data in bytes in the address 1A11H - 1A20H left four bits. Use register B as the loop counter for both small and large loops. Load the program into MPF-I and execute it. Discuss the reason why register B can be used as the counter for both loops.

```
1800                        ORG 1800H
1800   0621                 LD B,21H
1802   21001A               LD HL,1A00H
1805   C5        LOOP1      PUSH BC
1806   7E                   LD A,(HL)
1807   0604                 LD B,4
1809   87        LOOP2      ADD A,A
180A   10FD                 DJNZ LOOP2
180C   77                   LD (HL),A
180D   23                   INC HL
180E   C1                   POP BC
180F   10F4                 DJNZ LOOP1
1811   76                   HALT
```

(3) By calling the subroutine given in part 1 of this experiment (multi-byte BCD addition routine), write a program to add two 8-byte data stored in memory 1AØØH and 1AØ8H. The result must be stored in the 8-byte memory starting at 1AØØH.

(4) Revise the above program for BCD subtraction or multi-byte binary addition/subtraction. Test the program and record the method of revision used.

(5) Write a subroutine to change the 16-bit data in HL to its 2's complement. Write a main program to change the data in IX and IY to their 2's complements. Load the program into MPF-I and test it.

(6) By using the above routine for complementing the HL register pair, write a program to subtract DE from the data in IY and store the result in IY.

Experiment 6
Rotate Shift Instructions and multiplication Routines

Purposes:

1. To understand the use of Rotate and Shift instructions
2. To understand the designing techniques and uses of a binary multiplication subroutine.

Time Required: 4 - 8 hours

I. Theoretical Background:

1. The 9-bit data formed by the carry flag and 8-bit data in a register or memory can be shifted one bit left or right by ROTATE or SHIFT instructions. The ROTATE and SHIFT instructions are mainly used for multiplication and division. We multiply a number by rotating and shifting left the bits that constitute a number, while a division operation is done by rotating or shifting right the bits that constitute a number. There are many ways to rotate or shift the bits of a number. So, there are 13 different types of ROTATE and SHIFT instructions. Please refer to the MPF-I User's Manual, Appendix C. The mnemonic codes of these instructions are described below.

   (1) If the leftmost character of an instruction is "R", it is a "ROTATE" instruction. Such instructions can be used to rotate the 9-bit data (formed by 8-bit data and carry flag) left or right one bit, e.g. RLCA, RL, RRA, etc.
   If the leftmost character is "S", then it is a "SHIFT" instruction. All the 9-bits of the data are shifted left or right by one bit. The bit shifted out from one side will not be moved in from other side. Examples of such instructions are SAL and SRL.

   (2) If the second character from the left is "R", it means "shift right" or "rotate right". Instructions RR, SRL, RRCA, etc. are examples.
   If the second character in the left is "L", it means "shift left" or "rotate left". Instructions RL, SLA, RLCA, etc. are the examples.

   (3) The meaning of the third character is more complicated, but it can be summarized as follows:

      (a) In ROTATE instructions:
      The third character "C" represents the circular rotation of 8-bit data, carry flag is not included. The third character (or the fourth character) "A" means that this instruction is operated with the accumulator.
      Instructions RLA, RRA, RLCA and RRCA are examples.
      The third character "D" indicates the shift operation on decimal or hexadecimal numbers, for example, RLD and RRD. These instructions are designed to rotate the memory pointed to by HL left or right one digit (4 bits)

The digit entering from the left or right direction comes from bit 0 - bit 3 of the accumulator. The digit moving out from the other side is sent to bit 0 - bit 3 of the accumulator.

(b) In SHIFT instructions:
The third character "A" indicates "Arithmetic Shift". Binary data shifted left means multiplying it by 2 . Binary data shifted right means dividing it by 2 . Two of these instructions are SLA and SRA. Because bit 7 is assigned as "sign bit" and the sign of the data is not changed by these operations, the leftmost bit (bit 7) must be kept unchanged.
The third character "L" means "logical shift". Instruction SRL is an example. In these operations, a "0" is always moved to bit 7 from the left direction.

2. Binary Multiplication:

The operation of unsigned binary multiplication can be accomplished by shifting the binary number left or by a program loop of addition. An example of binary multiplication by hand-calculation is illustrated below.

```
     0101 <-- <- Multiplicand ->  --> 0101
   X 1011 <-- <-  Multiplier  ->  --> X 1011
   --------                          --------
     0101 <--                   -->0101
    0101  <--                   -->0000         partial
   0000   <--                   -->0101         product
+) 0101   <--                   -->0101
-----------                      -----------
   0110111     <---  Answer  --->   0110111
```

In the above calculation, one bit of the multiplier is checked. If that bit is 1, the multiplicand is copied as the partial product. If that bit is 0, 0000 is given instead. The position of the partial product is arranged such that the least significant bit of the multiplicand is aligned with the bit of the multiplier being checked. In this example, multiplicand and multiplier are both 4-bit data. Thus, it is necessary to repeat the operations of checking, shifting and addition four times. Similarly, the operations must be repeated 8 times for 8-bit data multiplication and 16 times for 16-bit data multiplication. In the left-hand side calculation given above, the bit-checking process starts from the least significant bit of the multiplier. In the right-hand side calculation, the bit-checking process starts from the most significant bit. But the results of the two calculations are identical. The program of binary multiplication for microcomputers can be designed by a method similar to the above calculation.

Example: Multiply the 8-bit data in register E by the 8-bit data in register A. The product is stored in the HL register pair.

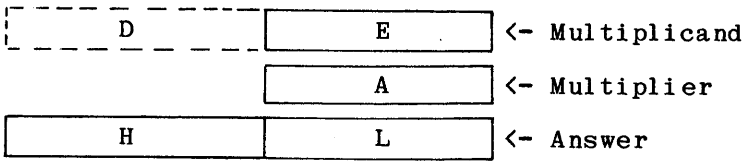
Answer: Specific registers have been assigned to store multiplicand, multiplier and product according to the characteristics of the Z80 instruction set. Using the calculation algorithm given in the right-hand side of the above example, the program is designed as follows.

1. In the above hand calculation, the bit-checking process starts from the least significant bit. A program loop can be employed in the example. The multiplier is 8-bits long, thus the loop number is equal to 8 . In every loop execution, the bit being checked (in register A) can be shifted into the carry flag by the RLCA instruction. Then, according to the condition of the carry flag, we can decide what will (or will not) be done next.

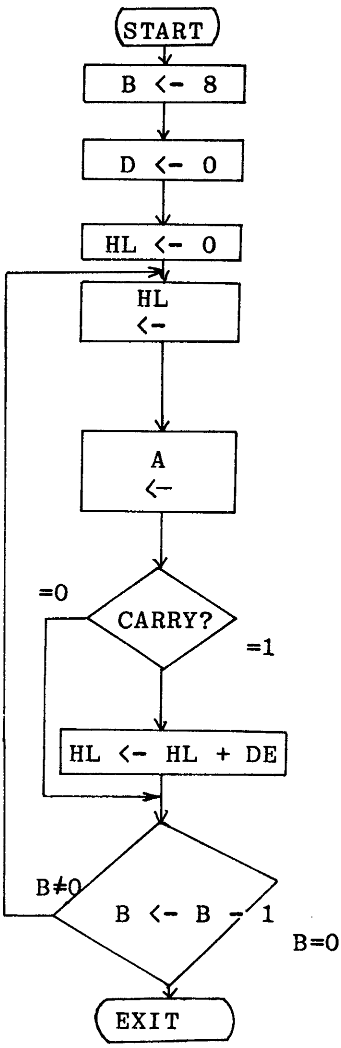
2. If the first bit checked (the leftmost bit) is 1, the partial result is actually obtained by shifting the multiplicand left (n-1) bits, where n is the number of bits in the multiplier. The other partial results are obtained by shifting the partial products left (n-2) bits, (n-3) bits,......., etc.
In this example, no other registers are required to store the partial results. Each partial result can be added directly to the HL register pair.

3. From the above description, we can see that the partial products must be shifted left (n-1) bits, (n-2) bits, (n-3) bits,...,etc. Since the bit-checking is also moving left in the process, we can generate a new intermediate result by immediately adding each partial product to the previous intermediate result. This method is more efficent and is used in the following program flowchart.

4. Register Assignments:



5. Program Flowchart :



<- Set B as the loop counter. For an 8-bit multiplier, B is set to 8

<- 16-bit addition will be performed. First clear D.

<- Set the initial value of answer to 0 .

<- Shift the intermediate result left one bit. The first shift process is invalid. Thus the first partial product will be shifted left n-1 bits after the loop is executed once.

<- The leftmost bit of the multiplier is moved to the carry flag for testing.

} If the leftmost bit of the multiplier is 1, the multiplicand is added to the intermediate result. Otherwise, the addition is ignored.

<- Check if the program loop has been completed. If it is, stop execution. Otherwise, repeat the loop operation.

```
                         MP8   LISTING
LOC   OBJCODE  STMT SOURCE STATEMENT                      ASM 3.0
                 1;***MULTIPLY***
                 2;ENTRY:
                 3          ;MULTIPLER IN E
                 4          ;MULTIPLICAND IN A
                 5;EXIT:
                 6          ;PRODUCT IN HL
                 7;REG. CHANGE    : B,D,HL
                 8;MEMORY BYTE    : 14
                 9;EXICUTION TIME:<395 CLOCK / <197.5 µS.
                10;
                11MP8:
0000  0608      12MULTI LD B,8       ;SET BYTE COUNTER=8
0002  1600      13      LD D,0
0004  62        14      LD H,D
0005  6A        15      LD L,D       ;CLEAR D,HL REGISTER
0006  29        16 LOOP ADD HL,HL    ;SHIFT HL LEFT
0007  07        17      RLCA         ;ROTATE BIT 7 OF "A" INTO
                                     ;CARRY FLAG
0008  3001      18      JR NC,NADD   ;TEST CARRY FLAG
000A  19        19      ADD HL,DE    ;ADD DE TO HL
000B  10F9      20 NADD DJNZ LOOP    ; END?
000D  C9        21      RET
```

II. Example Experiments:

1. The following program can be used to shift the 32-bit data stored in the HL and DE register pairs, which are adjacent, right one bit (or divide the data by 2). Load the program into MPF-I and test it. Next, revise the program such that it can be used to shift the 32-bit data left one bit (or multiply it by 2).

```
ORG     1800H
SRA     H
RR      L
RR      D
RR      E
RST     38H
```

2. Write a program to shift the 32-bit data, stored in RAM addresses 1A00H - 1A03H, left five bits (or multiply it by 20H). Load the program into MPF-I and test it. The starting address of the program is assigned as 1810H.

3. Using the RLD instruction, write a program to shift the BCD data, stored in RAM addresses 1A00H - 1A03H, left four bits. The starting address is assigned as 1830H. Load the program into MPF-I and test it.

4. The following program can be used to multiply the 16 bit data stored in the DE register pair by the contents of register A. Load the program into MPF-I and test it. Compare this program with the program given in Theoretical Background. Discuss the advantages and disadvantages of this program.

```
MPY8    LD      BC,800H
        LD      H,C
        LD      L,C
M1      ADD     HL,HL
        RLA
        JR      NC,M2
        ADD     HL,DE
        ADC     A,C
M2      DJNE    M1
        RST     38H
```

5. Write a program to multiply the 32-bit data stored in RAM addresses 1A00H - 1A03H by the 32-bit data stored in RAM addresses 1A04H - 1A07H. The product must be stored in RAM addresses 1A08H - 1A0FH.

Experiment 7
Binary Division Routine

Purposes:

1. To understand how to write a binary division subroutine for a microcomputer.
2. To familiarize the reader with the technique of software programming.

Time Required: 4 - 8 hours

I. Theoretical Background:

1. Binary division by hand-calculation:

   The following example will be used to illustrate the detailed procedure of binary division.
   Divide 11101101 by 00010100

   (1) Write the dividend on the right-hand side, divisor on the left-hand side, and put the quotient above the divisor.

```
                 [        ]   <- Quotient
                 [11101101]   <- Dividend (237)
       [00010100]   <- Divisor (20)
```

   (2) Shift the dividend and the quotient left one bit.

```
                 [0       ]   <- Quotient (Answer)
                 [11101101]   <- Dividend
          [00010100]          <- Divisor
```

   To compare the dividend and the divisor, place seven zeros after the divisor in the columns beneath the dividend. It can then be seen that the dividend is smaller than the divisor. Therefore put "0" in the position of quotient.

   (3) Continue to test if the dividend is less than the divisor with each shift. If the dividend is still less than the divisor, then put a "0" in the quotient. Otherwise, put a "1" in the quotient and the divisor is subtracted from the dividend. In this example, the dividend and the quotient must be shifted left five bits before a "1" can be put in the quotient. Thus four "0"s and one "1" are put in the quotient in the following way.

00001 <- Quotient

11101101 <- Dividend

00010100 <- Divisor

(when the dividend is larger than the divisor "1" is put in the quotient.

(4) Subtract the divisor from the dividend.
The difference becomes the dividend.

00001 <- Quotient (Answer)

01001101 <- Dividend after subtraction

00010100 <- Divisor

(5) The dividend and the quotient are shifted left two bits, then a "1" is put in the quotient.

0000101 <- Quotient (Answer)

01001101 <- Dividend

00010100 <- Divisor

(6) Subtract the divisor from the dividend.
The difference becomes the dividend.

0000101 <- Quotient (Answer)

00100101 <- Dividend after subtraction

00010100 <- Divisor

(7) Both dividend and quotient are shifted one bit again. Since the dividend is not less than the divisor, put "1" in the quotient.

00001011 <- Quotient (Answer)

00100101 <- Dividend

00010100 <- Divisor

(8) Subtract the divisor from the dividend, the remainder is placed in the position of the dividend.

00001011 <- Quotient (11)

00010001 <- Remainder (17)

00010100 <- Divisor

(9) If the remainder is not zero, the division process can be continued, but the result will contain fractions.

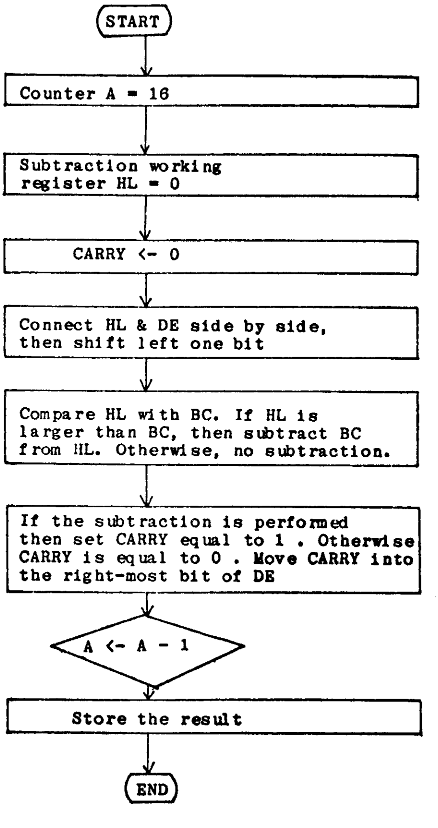
2. Division Program Design:

For the above algorithm, three memory locations are required to store the dividend, divisor and quotient.

Example : Write a program to divide the 16-bit data in the DE register pair by the 16-bit data in the BC register pair. The result (quotient) must be stored in the HL register pair and the remainder in the DE register pair.

Solution: The register assignment has been given in the problem description. The HL register pair can be used as the working register for 16-bit arithmetic subtraction. Shift the 16-bit data in DE left one bit to the HL register pair. Compare HL with BC. If HL is not less than BC, then subtract BC from HL and the carry flag is set to 1 automatically. Otherwise, no subtraction operation is performed and the carry flag will be 0 . Since the right-most bit of DE is now empty, the carry flag is then moved to this position.

The flowchart and the assembly language program are given below.



```
                 1 ; *** MPF-I EXAMPLE PROGRAM 008 ***
                 2 ;16 BIT DIVISION ROUTINE
                 3 ;ENTRY:DIVIDEND IN 'DE'
                 4 ;      :DIVISOR IN 'BC'
                 5 ;EXIT :RESULT IN 'HL'
                 6 ;      :REMAINDER IN 'DE'
                 7 ;REG. CHANG  :AF,DE,HL
                 8
0000   AF        9 DIV16  XOR A          ;CLEAR CARRY FLAG
0001   67       10        LD H,A
0002   6F       11        LD L,A         ;HL=0
0003   3E10     12        LD A,16        ;A = 16,LOOP COUNTER
                13
                14 DV0  ;HL&DE 4 BYTE ROTATE LEFT 1 BIT
0005   CB13     15        RL E           ;SHIFT LEFT,STORE PARTIAL RESULT
                16                                   IN BIT 0
0007   CB12     17        RL D
0009   ED6A     18        ADC HL,HL      ;ROTATE HL LEFT
                19
                20      ; IF HL GREAT THAN BC, SUBTRACT FROM BC
000B   ED42     21        SBC HL, BC     ;HL = HL - BC
000D   3001     22        JR NC,DV1
000F   09       23        ADD HL,BC      ;IF NEGATIVE,RESTORE HL
                24
0010   3F       25 DV1    CCF            ;PARTIAL RESULT IN CARRY FLAG
0011   3D       26        DEC A
0012   20F1     27        JR NZ,DV0
                28
0014   EB       29        EX DE HL
0015   ED6A     30        ADC HL,HL      ;STORE LAST BIT OF RESULT
0017   C9       31        RET
```

(1) Statement 10 and 11 of the program can be replaced by instruction LD HL,0 . But this instruction occupies 3 bytes memory and takes 10 clock cycles to execute. Instead, in this example, LD H,A and LD L,A are used (A is cleared to zero by statement 9). They occupy 2 bytes of memory and can be executed in 8 clock cycles.

(2) Addition and subtraction instructions can be used for "shift left" or "rotation" operations. In this example, instructions ADC HL,HL is identical with rotating the 16-bit data in HL pair left one bit (The bit moved to the carry flag comes from the leftmost bit of register D). The functions of the following instructions are described on the right-hand side.

```
ADD     A,A     ; Shift register A left one bit;
                  or multiply A by 2.

ADC     A,A     ; Rotate A left one bit
```

```
ADD     HL,HL   ; Shift HL left one bit; or double it.
ADC     HL,HL   ; Rotate HL left one bit.
ADD     IX,IX   ; Shift IX left one bit; or double it.
ADD     IY,IY   ; Shift IY left one bit; or double it.
```

II. Illustrations of Experiments :

1. Load the above program into MPF-I and then store it on audio tape.

2. Replace the last instruction (RET) in the above division subroutine by RST 38H and execute it. Record the obtained results in the following table.

Dividend	Divisor	Answer	Remainder	Check
8686H	0020H			
FFFFH	0003H			
5A48H	0142H			
0H	0142H			
1234H	0H			

3. Modify the above program such that the division process can be continued until a 16-bit fractional quotient is obtained.

4. Using the above program as a subroutine, write a main program to divide the data in RAM addresses 1A00H - 1A01H by the data in RAM addresses 1A04H - 1A05H. The result (quotient) must be stored in addresses 1A00H - 1A01H.

5. Write a program to divide the 4-byte data stored in addresses 1A00H - 1A03H by the 4-byte data stored in the memory address pointed to by the HL register pair. The result (quotient) must be in addresses 1A00H - 1A03H. The remainder must be stored in addresses 1A04H - 1A07H.

Experiment 8
Binary-to-BCD Conversion Program

Purposes:

1. To understand the programming techniques of binary-to-BCD conversion and its applications.
2. To understand the relation between subroutines and the main program.
3. To familiarize the reader with the technique of program writing.

Time Required: 4 hours

I. Theoretical Background:

1. Methods of binary-to-BCD conversion:

There are several methods for binary-to-BCD conversion. The method given below will be very neat because it uses the DAA instruction. Two memory sections are assigned to store binary and BCD data, respectively. The memory addresses for BCD data are initially cleared to zero. The following process of shifting and checking data is repeated until all binary data bits are shifted left completely: shift the binary data left one bit, and its leftmost bit is automatically transferred to CARRY. The BCD data is then doubled and its rightmost bit-position is filled with the CARRY of binary data.
The flowchart will be:

(1) Preparation:
Store the binary data in RAM with a starting address of 1A00H. Assign register D as the byte counter for the binary data, and register E as byte counter for the BCD data. (Since the bit number of the BCD data may be larger than that of the binary data, the value of E is usually not less than that of D).

(2) Clear the RAM section (starting address at 1A08H) for the BCD data.

(3) Shift the binary data (stored in RAM with starting address at 1A00H) left one bit. The leftmost bit is automatically transferred to CARRY Flag.

(4) Add CARRY to the BCD data (starting address at 1A08H) and then double the BCD data.

(5) Check if all the bits of binary data have been shifted out of the original memory section. If not, repeat step (3). If yes, it is end of the program.

The actual assembly language program is listed below.

```
                                EX001  LISTING                    PAGE    1
LOC   OBJ CODE  STMT SOURCE STATEMENT                              ASM 3.0

                   1 ;*** MPF-I EXAMPLE PROGRAM 001***
                   2 ;MULTIBYTE BINARY TO BCD CONVERTION
                   3 ;ENTRY:BINARY DATA STORED IN ADDR. 1A00H
                   4 ;EXIT :BCD DATA STORED IN ADDR. 1A08H
                   5 ;REGISTER USE
                   6 ; D  CONTAINS BYTE NUMBER OF BINARY DATA
                   7 ; E  CONTAINS BYTE NUMBER OF BCD DATA
                   8 ; A  BCD DATA WORKING REGISTER
                   9 ; B  LOOP COUNTER
                  10 ; C  BINARY BIT NUMBER
                  11
1800              12         ORG 1800H
                  13 BINBCD:
                  14 ;CLEAR BCD DATA BUFFER
1800    AF        15 CLEAR   XOR A       ;A=0
1801    43        16         LD B,E      ; B=BCD BYTE NUMBER
1802    21081A    1A         LD HL,1A08H
1805    77        18 CLR     LD (HL),A  ;CLEAR MEMORY
1806    23        19         INC HL     ;NEXT ADDRESS
1807    10FC      20         DJNZ CLR
                  21
                  22 ;CALCULATE BIT NUMBER
1809    7A        23         LD A,D     ;A=BYTE NUMBER
180A    87        24         ADD A,A
180B    87        25         ADD A,A
180C    87        26         ADD A,A     ;A=A*8
180D    4F        27         LD  C,A     ;C=BIT NUMBER
                  28
                  29 LOOP:
                  30 ;SHIFT BINARY DATA LEFT
180E    2E00      31         LD L,0      ;HL=1A00=BINARY STARTING ADDRESS
1810    42        32         LD B,D
1811    CB16      33 SHLB    RL (HL)
1813    23        34         INC HL
1814    18FB      35         DJNZ SHLB
                  36
                  37 ;ADD CARRY & DOUBLE BCD DATA
1816    2E08      38         LD L 8      ;HL=1A08=BCD STARTING ADDRESS
1818    43        39         LD B,E
1819    7E        40 BCDADJ LD A,(HL)
181A    8F        41         ADC A,A
181B    27        42         DAA
181C    77        43         LD (HL),A
181D    23        44         INC HL
181E    10F9      45         DJNZ BCDADJ
                  46
1820    0D        47         DEC C
1821    20EB      48         JR NZ,LOOP
1823    FF        49         RST 38H

   0 ASSEMBLY ERRORS
```

2. Assembly Language Programming Technique.

(a) Multiply (or divide) a piece of binary data by a fixed number:

Of course, the standard multiplication (or division) subroutine can be used to multiply (or divide) a binary number by a constant. However, a simple multiplication (or division) can be easily accomplished by shifting, additions or subtraction operations. For instance, in the above program, if the byte number of the binary data is known, then the bit number of the data can be easily obtained by multiplying the byte number by 8. In statements 22 - 27, instruction ADD A,A is used three times for multiplying the data in register D by 8 and then storing the result in register C. If the multiplier is not an exponential of 2, then addition or subtraction instructions must also be used.

Example: Multiply the data in D register by 6 and then store the result in register A. The program can be designed as follows.

```
LD      A,D     ; A = D
ADD     A,A     ; A = 2 * D
ADD     A,D     ; A = 3 * D
ADD     A,A     ; A = 6 * D
```

(b) Addressong method for memory on the same page:

A memory address can be pointed to indirectly by a register pair (16 bits). To change a memory address pointed to by a required pair within the same page (each page contains 256 bytes), only a change in the low-order byte of the register pair is required. For instance, in the program listed above, the binary and BCD data are stored on the same page of memory (page 1AH). Since statement 1A assigns the contents of register H as 1AH, only a change in the contents of register L is required to change the pointed address in statements 31 and 38.

II. Example Experiments:

1. Load the binary-to-BCD conversion program listed in part I into MPF-I and then store it on audio tape for future applications.

2. Test the above program:

First, store the byte numbers of binary and BCD data in registers D and E, respectively. Next, load the binary data into RAM, with a starting address at 1A00H. Record the obtained result and check if is correct.

Binary	Hexadecimal	BCD	registers D & E
1000000000	0200H		D = 2, E = 2
	FFFFH		D = 2, E = 3
	18000H		D = 3, E = 4
	5A48347FH		D = 4, E = 6
	2^{32}		D = 8, E = 0AH
	2^{63}		D = 8, E = 0AH
	$2^{64} - 1$		D = 8, E = 0AH

3. Change the above program to a subroutine format (Replace the last instruction RST 38H by RET). Using this subroutine, write a program to convert the contents of the DE register pair into a BCD number and then store the converted BCD data in the HL register pair. The contents of the DE register pair will not be changed after the program execution. Test the program and write down the complete program in the blanks below.

4. Write a program to multiply the binary data in register E (<20H) by 7 and store the result in register A.

Experiment 9

BCD-to-Binary Conversion Program

Purpuses:

1. To understand the methods of BCD-to-Binary conversion.
2. To familiarize the reader with programming technique.

Time Required: 4 - 8 hours

I. Theoretical Background:

1. Methods of BCD-to-Binary conversion:

There are also several method for BCD-to-Binary conversion. In this experiment, the simple yet efficient method of shifting and checking is used. The RAMs used for storing the binary and BCD data are adjacent (in a row with the low-order digit on the right side). The BCD data is stored on the left-hand side and the converted binary data is stored on the right-hand side. The conversion procedure is given as follows.

(1) Assign the bit number of the binary number as N for N program loops.

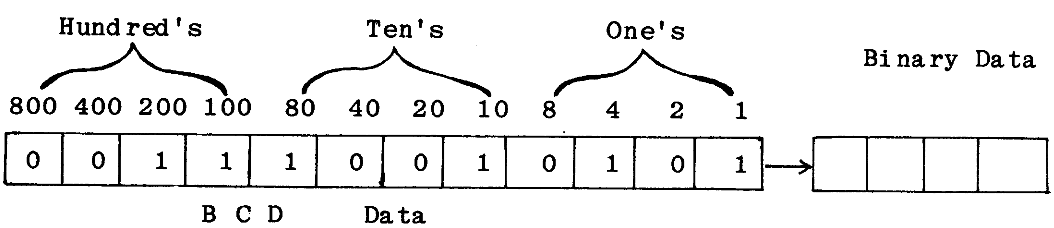
(2) Shift the connected data right one bit.

(3) Check the left-most bit of each digit (4 bits). If the checked bit is 1, then subtract 3 from the corresponding digit.

(4) Repeat step (2) & (3) N times. The conversion process is then completed.

2. Principle of the checking process :

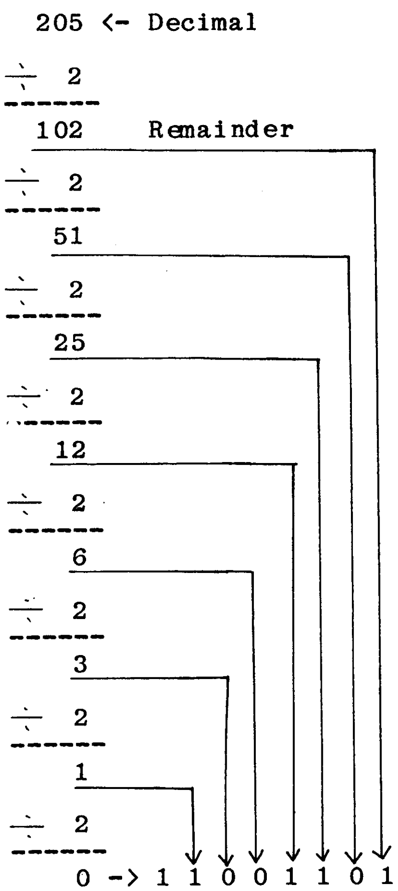
The real purpose of steps (2) & (3) of the above method is to divide the BCD number by 2 and put the remainder in the memory.The principle is illustrated in the following figure.



(1) Each BCD digit contains 4 bits. Shifting the 4 bits of a digit right one bit will divide this digit by 2. For instance, the leftmost digit of the ten's four bits represents 80 if it is "1". If this bit is shifted right, then it represents 40, that is, half of its original value.

(2) If a "1" is shifted from high a order digit to a lower order digit, the value is reduced to 5 (or 50, 500, ---, etc). However, the resulting BCD code will interprete this bit as 8 (or 80, 800, ---, etc). Thus 3 (or 30, 300, ---, etc) must be subtracted from the resulting BCD number.

(3) The conversion method can be illustrated by the following hand-calculation.



2	205	--- 1	BIT 0
2	102	--- 0	BIT 1
2	51	--- 1	BIT 2
2	25	--- 1	BIT 3
2	12	--- 0	BIT 4
2	6	--- 0	BIT 5
2	3	--- 1	BIT 6
	1		BIT 7

1100 1101
C D

3. BCD-to-Binary conversion program:

Once the conversion method is decided, it is very easy to design the program. The following program can be used to convert 5-byte (or 10-digit) BCD data stored in RAM into 4-byte binary data. Since the largest value of 4-byte binary data is 4,294,967,295, the BCD number to be converted can not exceed this value. In RAM, the memory of addresses 1A00H - 1A03H are reserved for storing the binary data (lowest-order byte in 1A00H). The memory of addresses 1A04H - 1A08H are assigned to store the BCD data. Sample programs for BCD-to-Binary conversion and Binary-to BCD conversion are listed below for reference.

```
                    EX007  LISTING
LOC   OBJ CODE STMT SOURCE STATEMENT

                  1 ;*** MPF-I EXAMPLE PROGRAM 007 ***
                  2
                  3 ; 10 DIGIT BCD TO BINARY CONVERSION
                  4 ; ENTRY: BCD DATA IN RAM 1A04H TO 1A08H
                  5 ;       : MAX. BCD DATA IS (4294967295)
                  6 ; EXIT : BINARY DATA IN RAM 1A00H TO 1A03H
                  7 ; REG. CHG : AF,HL,BC
1800              8        ORG 1800H
1800  0E20        9        LD C,32        ;PRESET CONV. LOOP = 32
                 18 DBLP:
                 11 ; DECIMAL DIVID BY 2
1802  0605       12        LD B,5         ;BCD BYTE COUNT = 5
1804  AF         13        XOR A          ;CLEAR CARRY FLAG
1805  21081A     14        LD HL,1A08H    ;HL POINT TO LEFT BYTEL
1808  7E         15 COR0   LD A,(HL)      ;TRANSFER DATA TO A REG.
1809  1F         16        RRA            ;ROTATE RIGHT
180A  F5         1A        PUSH AF        ;SAVE CARRY FLAG
                 18 ;* BCD DIVID CORRECTION
180B  CB7F       19        BIT 7,A        ;TEST BIT 7
180D  2802       20        JR Z,COR1      ;NO CORRECT IF BIT 7 = 0
180F  D630       21        SUB 30H        ;SUBTRACT FROM 30H IF BIT 7 = 1
1811  CB5F       22 COR1   BIT 3,A        ;TEST BIT 3
1813  2802       23        JR Z,COR2
1815  D603       24        SUB 3
                 25
181A  77         26 COR2   LD (HL),A      ;STORE TO MEMORY
1818  2B         27        DEC HL         ;NEXT BYTE
1819  F1         28        POP AF         ;RESTORE CARRY FLAG
181A  10EC       29        DJNZ COR0      ;DONE LOOP
                 30
                 31 ;ROTATE BINARY RIGHT
181C  0604       32        LD B,4         ;BINARY BYTE = 4
181E  CB1E       33 SHR4   RR (HL)
1820  2B         34        DEC HL
1821  10FB       35        DJNZ SHR4
                 36
1823  0D         37        DEC C
1824  20DC       38        JR NZ,DBLP
1826  C9         39        RET
```

```
                       EX007  LISTING
LOC   OBJ CODE STMT SOURCE STATEMENT
                 40 *E
                 41 ;4 BYTE BINARY TO BCD CONVERSION
                 42 ;  ENTRY:BINARY DATA STORE IN ADDR.  1A00H TO 1A03H
                 43 ;  EXIT :BCD DATA STORE IN ADDR. 1A04H TO 1A08H
                 44 ;  REG. CHANG : AF,BC,HL
                 45
                 46 BINBCD:
                 47 ;CLEAR BCD DATA BUFFER
1827  21041A     48        LD HL 1A04H
182A  0605       49        LD B,5
182C  3600       50 CLEAR LD (HL),0
182E  23         51        INC HL
182F  10FB       52        DJNZ CLEAR
                 53
1831  0E20       54        LD C,32
                 55 LOOP
                 56 ;SHIFT BINARY DATA LEFT
1833  68         57        LD L,B         ; HL=1A00=BINARY STARTING ADDRESS
1834  0604       58        LD B,4
1836  AF         59        XOR A
1837  CB16       60 SHLB   RL (HL)
1839  23         61 INC HL
183A  10FB       62        DJNZ SHLB
                 63
                 64 ;ADD CARRY & DOUBLE BCD DATA
183C  0605       65        LD B,5
183E  7E         66 BCDADJ LD A,(HL)
183F  8F         67        ADC A,A
1840  27         68        DAA
1841  77         69        LD (HL),A
1842  23         70        INC HL
1843  10F9       71        DJNZ BCDADJ
                 72
1845  OD         73        DEC C
1846  20EB       74        JR NZ, LOOP
1848  C9         75        RET

   0  ASSEMBLY ERRORS
```

II. Example Experiments:

1. Load the two subroutines for BCD-to-Binary anb Binary-to-BCD conversion into MPF-I and then store them on audio tape for future application.

2. Replace the last instruction RET of the above subroutines by RST 38H so that control of the microcomputer MPF-I will be returned to monitor after program execution. Load an arbitrary 5-byte BCD number in RAM address 1A04H - 1A08H. Convert this BCD data into binary data by using the above program. Check if the result is correct.

3. By a method similar to that described in part I (Theoretical Background), write a program to convert the 4-digit BCD data into binary data : The processing must be held within CPU registers and the result will be stored in the DE register pair.

	Assigned Decimal Number	Converted Binary Number	Re-converted Decimal Number
1			
2			
3			
4			
5			

4. Using the binary multiplication routine and the routines for conversion between binary and BCD data, write a program for decimal multiplication. The decimal multiplier and multiplicand must be stored in the HL and DE register pairs, respectively. The result must be stored in RAM addresses 1A04H - 1A08H. The data in HL and DE must be unchanged after program execution.

Experiment 10
Square-Root Program

Purposes:

1. To understand how the microcomputer calculates the square root of a binary number.
2. To practice microcomputer programming.

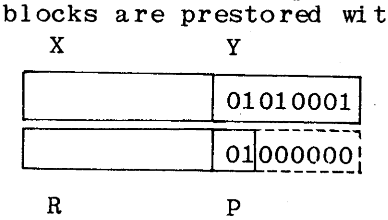
Time Required: 4 - 8 hours

I. Theoretical Background:

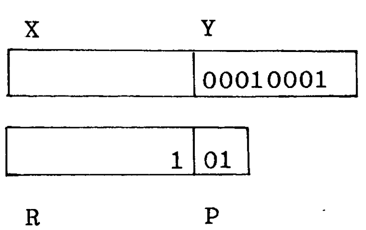
1. Calculating square roots of binary numbers by hand:

There are several methods for calculating the square root of a binary number. The following method for hand-calculation can be easily converted into a microcomputer program. This method is illustrated by calculating the square root of 01010001 (or 81):

(1) Each of the following blocks represents the position for storing data. The original binary number is stored in Y block, the number 01 is permanently stored in P block. X and R blocks are prestored with 0.



(2) Subtract the number formed by the R & P blocks from the number formed by the X and Y blocks. If the result is non-negative, then put 1 at the rightmost position in the R block and shift the original data in the R block left one bit. If the result is negative, then restore the original data in the X & Y blocks and shift the data in R left one bit. In this example, the result of subtraction is positive. Thus, the following result is obtained.



(3) Shift the data in the X & Y blocks left two bits.

X	Y
	00 01000100
1	01
R	P

(4) Since the number in the X and Y blocks after the shift process is still less than that in the R and P blocks, thus the data in the R block must be shifted left one bit and a "0" is put in the rightmost position. The data in the X and Y blocks remains unchanged.

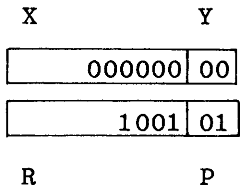
(5) Shift the data in the X and Y blocks left two bits.

X	Y
0001	00010000
10	01
R	P

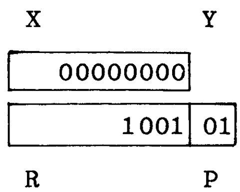
(6) The new data in the X and Y blocks is still less than the R and P block. Thus, shift the data in the R block left one bit again. An "0" is put in the rightomst position of the R block. The data in the X and Y blocks is also shifted left two bits.

X	Y
000100	01
100	01
R	P

(7) The number in the X and Y blocks is not less than that in the R and P blocks. Subtract the number in the R and P blocks from the number in the X and Y blocks. Shift the data in R left one bit and put a "1" in the left-most bit-position.



(8) Shift data in the X and Y blocks left two bits. Since the the orginal data in the Y blocks has been shifted out completly, the final result is given in the R block.



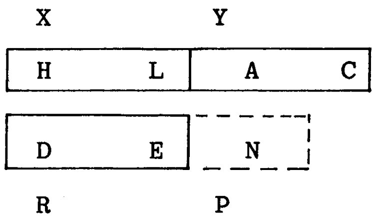
(9) If the original data in the Y block is not the square root of some integral binary number, then the above method may be continued to find the fractional part of the square root.

2. Square root routine

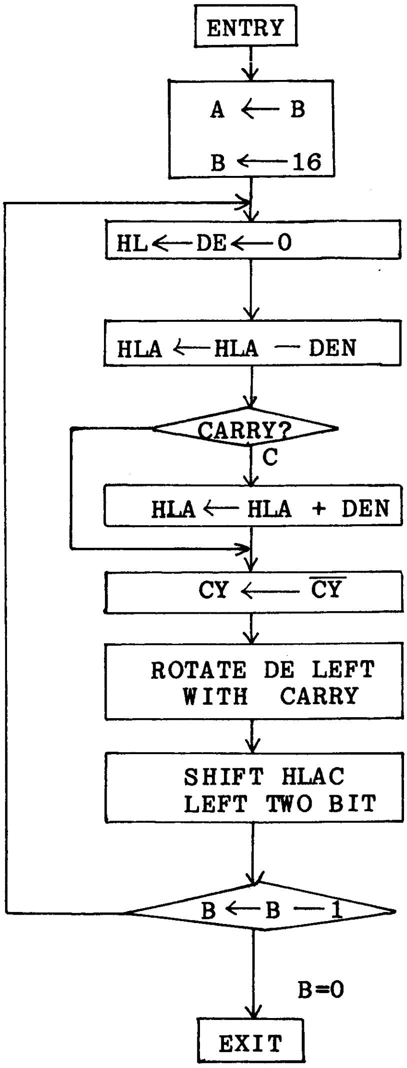
The square root routine can be designed by the method described above. A subroutine for calculating the square root of a 16-bit piece of data is illustrated below.

Example: Find the square root of a 16-bit piece of data stored in the BC register pair. The calculation must be continued till the fractional part of the solution contains 8 bits. The integral part of the solution will be stored in register D, while the fractional part will be stored in register E.

Solution: The CPU registers are assigned as follows:



The original data is stored in registers A and C (Y block). The HL register pair is used as the working area of subtraction operation. The answer will be stored in the DE register pair (R block). The data in the P block is a fixed number, its left-most two bits are 01, i.e. the data in the P block may be written as 01000000B (40H). The program and its flowchart are given below.


ENTRY
A ← B
B ← 16
HL ← DE ← 0
HLA ← HLA — DEN
CARRY?
C
HLA ← HLA + DEN
CY ← $\overline{CY}$
ROTATE DE LEFT WITH CARRY
SHIFT HLAC LEFT TWO BIT
B ← B — 1
B=0
EXIT

```
                ; *** MPF-I EXAMPLE PROGRAM 009 ***
                ; 16 BIT SQUARE ROOT ROUTINE
                ;ENTRY: BINARY DATA IN 'BC'
                ;EXIT : RESULT IN 'D'(INTEGER)
                ;                'E' (FRACTION)
                ;REG. CHANG.AF,BC,DE,HL
0000  78        SQRT16 LD A,B        ;A&C = ENTRY DATA
0001  0610             LD B,16       ;LOOP COUNTER
0003  210000           LD HL,0       ;HL:WORKING AREA
0006  54               LD D,H
0007  5C               LD E,H        ;DE=0,RESULT PRESET TO 0
0008  D640      SQ0    SUB 40H       ;A=A-40H,40H IS A FIXED DATA
000A  ED52             SBC HL,DE     ;HL=HL-DE
000C  3004             JR NC,SQ1     ;IS HL > DE ?
000E  C640             ADD A,40H
0010  ED5A             ADC HL,DE     ;IF NOT, RESTORE A&HL
0012  3F        SQ1    CCF           ;PARTIAL RESULT IN CARRY FLAG
0013  CB13             RL E          ;STORE PARTIAL RESULT
0015  CB12             RL D          ;   & SHIFT 'DE' (RESULT) LEFT
                   ;'HL.A C' 4 BYTE SHIFT LEFT TWICE
0017  CB21             SLA C
0019  17               RLA
001A  ED6A             ADC HL,HL
001C  CB21             SLA C
001E  17               RLA
001F  ED6A             ADC HL,HL

0021  10E5             DJNZ SQ0      ;DONE LOOP
0023  C9               RET

   0 ASSEMBLY ERRORS
```

II. Example Experiments:

1. Load the above program onto MPF-I and then store it in audio tape for future applications.

2. Replace the last instruction (RET) by RST 38H. Prestore a 16-bit data in the BC register pair and then execute the square root program. Write down the result obtained.

Data Prestored in BC	Result of Program Execution	Check
0051H		
0000H		
FFFFH		
4000H		

3. Revise the above program such that it can be used for calculating the square root of a 32-bit piece of data. Store the original data in the BC and IX registers. The answer will be stoed in the De register pair. Only the integial part of the square root is required.

4. Using the square root routine and binary multiplication routine, write a program for finding the absolute value of the vector formed by two mutual perpendicular vectors. The length of each vector component can be represented by an 8-bit binary number. These two numbers are stored in the H and L registers, respectively. The result of the program execution will be stored in register D.

   $(D) = \sqrt{(H)^2 + (L)^2}$.

Experiment 11
Introduction to MPF-1 Display

Purposes:

1. To understand how to use subroutines of the monitor program.
2. To understand how a character is displayed by a seven segment display.
3. To understand the application of conversion tables of characters.
4. To understand the structure and characteristics of a matrix-form keyboard.

Time Required: 8 hours

I. Theoretical Background:

1. Structure of seven-segment display

The seven-segment display is one of the least expensive displays for alphanumeric characters. The display is very suitable for applications in microcomputer systems. Illumination of each individual segment can be accomplished by using LEDs, fluorescent devices or small incandescent lamps. The hardware connection is shown in Fig. 11-1. Each digit consists of seven independently controlled segments which are designated as a, b, c, d, e, f, and g. All the cathodes (or anodes) of the same segment in all digits are connected together by a common wire. The control lines for the seven segments are designated as Sa, Sb,...., Sg, respectivly. A common line (e.g. D0, D1, D2,....) connected to each segment of a digit is used for digit selection. A segment is illuminated only when both the control signal and the digit-selection signal are applied simultaneously. The structure of this kind of display is simple but it requires a fast scanning circuit to display each digit.

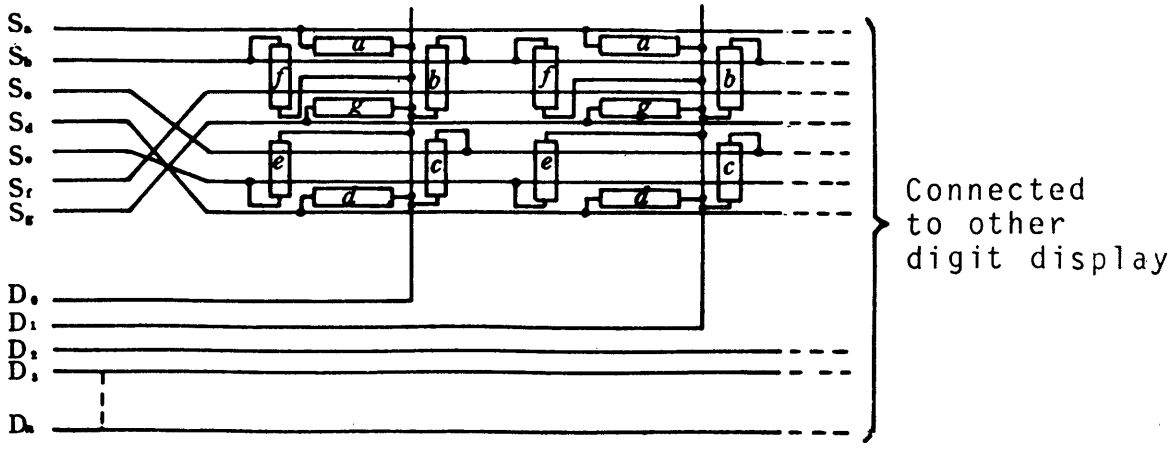


Fig. 11-1 7 Segment Display Connecting Circuit

2. Scanning method of the seven-segment display

The principle of scanning the seven-segment display is as follows: output a digit-selection signal and activate the segment-control line of the corresponding word format. For instance, if the digit-selection line chosen is D0 and only the segment-control line Sa, Sb, and Sc are activated, then a digit "7" will be displayed at the position indicated by the D0 line. The scanning method is : Apply a signal voltage to the digit-selection lines D0, D1,...., Dn in sequence. When a digit-selection line is activated, voltage signals are applied to the segment-control lines Sa, Sb,... Sg of the corresponding word format. After digits have been scanned once, the scanning is repeated from the beginning. Each digit must be scanned at least 40 times per second. Due to persistence of vision of human eyes, all digits in the display appear to be lit simultaneously. The scanning speed can not be too fast, since the residual light of the neighboring digit may cause confusion.

3. Scanning period and keybounce:

The keypad is usually depressed by hand. In general, the microcomputer's reaction is much faster than a human's response. To key in data or a command from the keyboard, the microcomputer must scan the keyboard repeatedly until a key is found depressed. A key bounces for a short time when being depressed or released. Fig. 11-2 is a time response diagram of typical key-depressing or key-releasing operation. Thus, a key-depression might be identified as two or more key-depressions if the key-board scanning rate is too fast. To avoid this problem the period of scanning must be longer than the bouncing time (usually bouncing time is no longer than 10m sec). The period of scanning is between 10m sec and 50m sec. In the figure below an upward arrow indicates when the key is examined. At Tn+2, microcomputer program found that the key was depressed and identified the keycode. At Tn+3, the key was also found depressed. Since the key was found depressed in a previous scan, the microcomputer program would determine that this was not a new key-depression (i.e. the key had not been released during this time interval). Only if the key is found depressed at Tn+4 or Tn+5 is a key-depression found at Tn+6 really a new key-depression. A program for getting data from a keyboard designed by this rule will be error-free, no matter how long the duration of key-depression is and whatever is found at Tn+1 and Tn+4 (0 or 1).

The hardware connection is shown in Fig 11-3

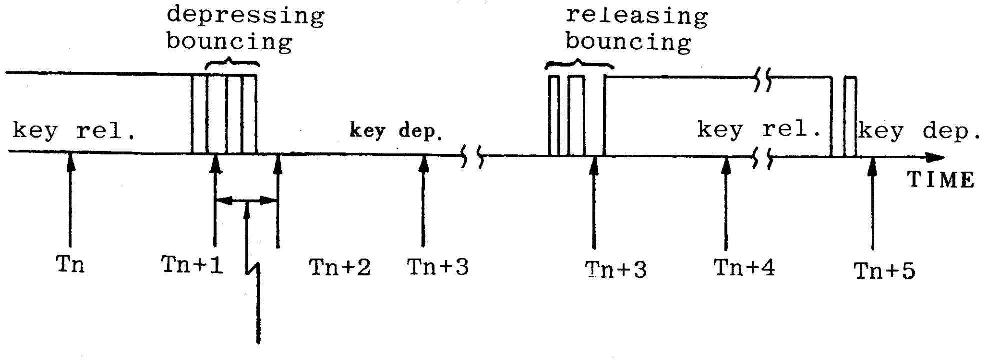


Fig 11-2 The Time Response of Keyboard Scanning

4-1 Construction of MPF-I display:

The display of the MPF-I is composed of 6 LEDs. 14 output lines are used to control the display. The addresses of the 14 output ports are given in Fig 11-3. The 8 output lines with addresses PB0 - PB7 are used to control the seven segments and a decimal point in the display. The 6 output lines with addresses PC0 - PC5 are connected to the 74LS492 to select the digit to be displayed. All the segments are controlled by logic "1" signals. If a segment is at logic "1", then it is lit. If a segment is at logic "0", then it is extinguished. Before MPF-I executes the user's program, the output ports PB0 - PB7, PC0 - PC5 are set at logic "0". If the output port PC0 is set at logic "1", then digit 1 is selected. If the output ports PB0 - PB7 are at logic "1", then only digit 1 of the display is illuminated.

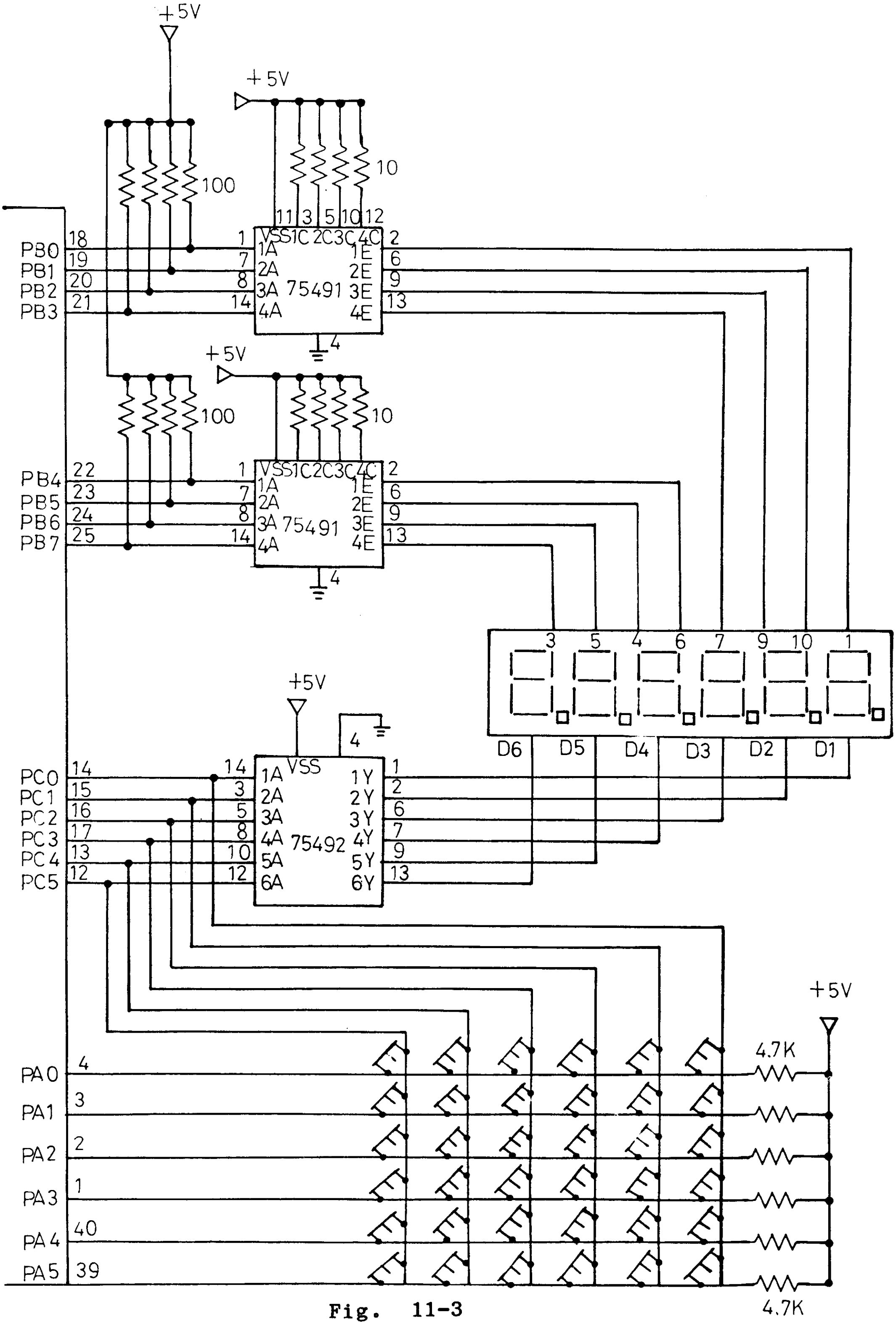


Fig. 11-3

4-2 The structure of matrix-from keyboard:

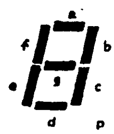
A matrix-form keyboard is an important yet inexpensive input device for the microcomputer. The structure of the keyboard is a number of wires in a matrix form. At each node of the matrix a keypad is positioned. The 6 vertical lines and 6 horizontal lines (6x6) in Fig. 11-3 provide 36 contact points for keyboards. When a key is depressed, it makes a contact between one row and one column of the matrix. The 6 horizontal lines are connected to the microcomputer input port. The input port addresses are PAØ - PA5. When no key is depressed, the 6 input addresses are connected to +5V power supply via 6 resistors. Thus, logic "1" will be read in by the microcomputer input ports. The 6 column lines are connected to the output ports PCØ - PC5 which are also connected to the display circuit.

4-3 Keyboard scanning program

The microcomputer may select the rightmost column line from the output line PC0. The voltage of the 6 row lines are then examined in sequence. At the beginning of keyboard scanning, a counter is set to zero, port "C" will output "11000001", the value of PC5 - PC6 are "000001". PC6 and PC7 must always be high during keyboard scanning because the output line of PC6 is connected to BREAK and PC7 is connected to the speaker. The voltage of the 6 row lines are then examined in sequence. If a key is depressed (a zero voltage at the corresponding row), it can be identified from the port address. If no key in the first column is depressed then the microcomputer port C will output "11000010" to select the second column for examination. In general the keyboard scanning proceeeds in sequence, from upper side to lower side, from right side to left side of the key matrix, to examine if any key is depressed. Each key in the keyboard is encoded. Once a key being examined is found to be not pressed, the counter's value is increased until a key is found depressed. Then the counter's value is the position code of that key.

4-4 Conversion table

A subroutine SCAN in MPF-I monitor program with starting address 05F can be used to control simultaneously 6 byte of data stored in RAM. The addresses of the display buffer are 1FB6 - 1FBB. Fig. 11-4 is a conversion table.



7	6	5	4	3	2	1	0
d	p	c	b	a	f	g	e

DISPLAY FORMAT:

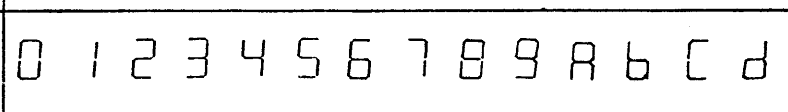
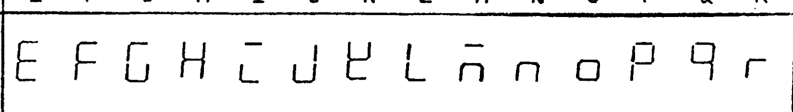
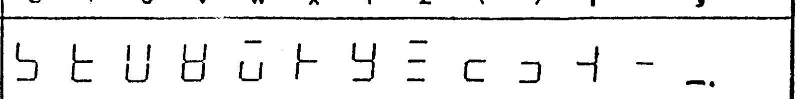
CODE	BD	30	9B	BA	36	AE	AF	38	BF	BE	3F	A7	8D	B3
DATA	0	1	2	3	4	5	6	7	8	9	A	B	C	D
DISP														
CODE	8F	0F	AD	37	89	B1	97	85	2B	23	A3	1F	3E	03
DATA	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
DISP														
CODE	A6	87	B5	B7	A9	07	B6	8A	83	A2	32	02	C0	00
DATA	S	T	U	V	W	X	Y	Z	(	)	+	–	,	
DISP														

Fig. 11-4 Conversion Table

1. Position-Code (CALL SCANI) :

1E SBR	18 CBR	12 '0'	0C '1'	06 '2'	00 '3'
1F '-'	19 PC	13 '4'	0D '5'	07 '6'	01 '7'
20 DATA	1A REG	14 '8'	0E '9'	08 'A'	02 'B'
21 '+'	1B ADDR	15 'C'	0F 'D'	09 'E'	03 'F'
22 INS	1C DEL	16 GO	10 STEP	0A	04
23 MOVE	1D RELA	17 TPWR	11 TPRD	0B	05

2. Internal-Code (CALL SCAN) :

15 SBR	1A CBR	00 '0'	01 '1'	02 '2'	03 '3'
11 '-'	18 PC	04 '4'	05 '5'	06 '6'	07 '7'
14 DATA	1B REG	08 '8'	09 '9'	0A 'A'	0B 'B'
10 '+'	19 ADDR	0C 'C'	0D 'D'	0E 'E'	0F 'F'
16 INS	17 DEL	12 GO	13 STEP	22	20
1C MOVE	1D RELA	1E TPWR	1F TPRD	23	21

Fig. 11-5

Fig. 11-5 are table of Position-code and internal-code. If a scanned key is depressed, then we can pick up a Position-code (by CALL SCAN1). Adding this data to the starting addess of the KEYTAB (The address of KEYTAB in monitor program is 077B) then key position code is converted to key internal code. For example when "F KEY" is depressed we pick up a position code 03, then by the conversion gives the internal code with OF.

4-5 Keyboard and display program

The microcomputer usually executes some part of the user's program before it scans the keyboard to fetch new data or command. Since the keyboard scanning need not be very fast, the microcomputer has can assign time-slot to scan the display. In MPF-I, a combined program is written for both keyboard and display scanning. The interval between two consecutive scans is 10 msec, ie. 100 times per second.

5. Useful Subroutines of the Monitor Program

5.1 Summary

ADDRESS	MNEMONIC	FUNCTION
0624	SCAN1	Scan Keyboard and the display one cycle.
05FE	SCAN	Scan keyboard and the display until a new key-in.
0689	HEX7	Convert a hexadecimal digit into the 7-segment dispaly format.
0678	HEX7SG	Convert two hexadecimal digits into 7-segment display format.
0665	ADDRDP	Convert two bytes data stoed in DE to 7-segment display format. The output is stored in the address field of display buffer 1FB8 - 1FBB.
0671	DATADP	Convert the data stoed in A to 7-segment display format. The output is stored in the data field of display buffer 1FB6 - 1FB7.

5.2 SCAN1

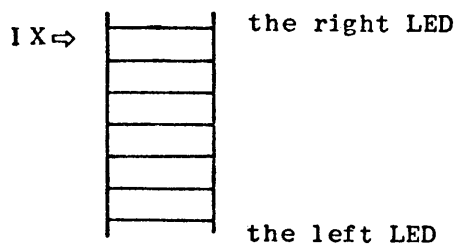
[Address]: 0624

Function]: Scan the keyboard and the display 1 cycle from right to left. Execution time is about 10ms (9.97ms exactly).

[Input]: IX points to the display buffer which contains six bytes.

[Output]: (1) If no key-in, then carry flag = 1.
(2) If key-in, carry flag = 0 and the position-code of the key is stored in register A.

[Supplement]: (1) 6 bytes are required for 6 LED's.
(2) IX points to the rightmost LED, IX+5 points to the left most LED.



(3) See Fig. 11-4 for the relation between each bit and the seven segments.

[Register]: Destroy the contents of AF, B , HL , AF', BC', DE'.

5.3 SCAN

[Address]: 05FE

[Function]: Similar to SCAN1 except:

(1) SCAN1 scans one cycle, but SCAN will scan till a new key-in.
(2) SCAN1 returns the position while SCAN returns the internal code of the key pressed.

[Input]: IX points to the display buffer.

[Output]: Register A contains the internal code of the key pressed.

[Register]: Destroy AF,B, HL, AF', BC', DE'.

5.4 HEX7

[Address]: 0689

Function]: Decode a hexadecimal number to its 7-segment display format.

[Input]: The least significant 4 bits of A register contain the hexdecimal number. (0-F).

[Output]: The result is also stored in A register.

[Register]: Destroy AF only.

5.5 HEX7SG

[Address]: 0678

[Function]: Convert two hexadicimal number into 7-segment display format.

[Input]: The first number is stored in the right 4 bits of A. The second number is stored in the left 4 bits of A.

[Output]: The first display pattern is stored in (HL), the second is in (HL+1), HL is increased by 2.

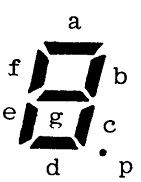
[Register]: Destroy AF, HL.

II. Program Examples

EXAMPLE 1: Display HELPUS , HALT when Step is pressed.

```
                       DISPLAY 'HEL US' UNTIL KEY-STEP PUSHED:
1800                           ORG     1800H
1800  DD212018                 LD      IX,HELP
1804  CDFE05           DISP    CALL    SCAN
1807  FE13                     CP      13H            ;KEY-STEP
1809  20F9                     JR      NZ,DISP
180B  76                       HALT
                       ;
1820                           ORG     1820H
1820  AE               HELP    DEFB    OAEH           ; 'S'
1821  B5                       DEFB    OB5H           ; 'U'
1822  1F                       DEFB    01FH           ; 'P'
1823  85                       DEFB    085H           ; 'L'
1824  8F                       DEFB    08FH           ; 'E'
1825  37                       DEFB    037H           ; 'H'
                        ;
                       SCAN    EQU     05FEH
                               END
```

Details of the display buffer are given below:



Position	Display Format	Segment of Illumination	d p c b a f g e	Data	Addr
Right	S	a,c,d,f,g,	1 0 1 0 1 1 1 0	AE	1820
↑	U	b,c,d,e,f,	1 0 1 1 0 1 0 1	B5	1821
	P	a,b,e,f,g,	0 0 0 1 1 1 1 1	1F	1822
	L	d,e,f,	1 0 0 0 0 1 0 1	85	1823
↓	E	a,d,e,f,g,	1 0 0 0 1 1 1 1	8F	1824
Left	H	b,c,e,f,g,	0 0 1 1 0 1 1 1	37	1825

EXAMPLE 2: Flash 'HELP US'

Use routine SCAN1 to display 'HELPUS' and blank alternately. Each pattern is display 500ms by looping SCAN1 50 times.

```
                      ;FLASH  'HELP US'
1800                          ORG     1800H
1800 212618                   LD      HL,BLANK
1803 E5                       PUSH    HL
1804 DD212018                 LD      IX,HELP
1808 DDE3             LOOP    EX      (SP),IX
180A 0632                     LD      B, 50
180C CD2406        HELFSEG    CAL     SCAN1
180F 10FB                     DJNZ    HELFSEG
1811 18F5                     JR      LOOP
                    ;
1820                          ORG     1820H
1820 AE            HELP       DEFB    0AEH              ; 'S'
1821 B5                       DEFB    0B5H              ; 'U'
1822 1F                       DEFB    01FH              ; 'P'
1823 85                       DEFB    085H              ; 'L'
1824 8F                       DEFB    08FH              ; 'E'
1825 37                       DEFB    037H              ; 'H'
1826 00            BLANK      DEFB    0
1827 00                       DEFB    0
1828 00                       DEFB    0
1829 00                       DEFB    0
182A 00                       DEFB    0
182B 00                       DEFB    0
                    ;
                   SCAN1      EQU     0624H
                              END
```

The content of 180B determines the flash frequency. You may change it to any value.

EXAMPLE 3: Display the key code of the key pressed.

```
                        ;DISPLAY INTERNAL CODE
1800                             ORG      1800H
1800   DD210019                  LD       IX,OUTBF
1804   CDFEO5           LOOP     CALL     SCAN
1807   210019                    LD       HL,OUTBF
180A   CD7806                    CALL     HEX7SG
180D   18F5                      JR       LOOP
                        ;
1900                             ORG      1900H
1900   00               OUTBF    DEFB     0
1901   00                        DEFB     0
1902   00                        DEFB     0
1903   00                        DEFB     0
1904   00                        DEFB     0
1905   00                        DEFB     0
                        ;
                        SCAN     EQU      05FEH
                        HEX7SG   QU       0678H
                                 END
```

When a key is pressed, the internal code of it is displayed on the data filed. The user may compare it with Fig. 2-11-5.

If you want to display the position code of the keys, you may change the program as follow:

```
                        ;DISPLAY POSITION CODE
1800                             ORG      1800H
1800   DD210019                  LD       IX,OUTBF
1800   CD2406           LOOP     CALL     SCAN1
1807   38FB                      JR       C,LOOP
1809   210019                    LD       HL,OUTBF
180C   CD7806                    CALL     HEX7SG
180F   18F3                      JR       LOOP
```

EXAMPLE 4: Convert 3 continuous bytes into 7-segment display format. Store the results in 1903 - 1908 then display them.

```
                 ;DISPLAY 3 BYTES IN RAM TO 6
                  HEXA-DIGITS
1800                           ORG      1800H
1800  110019                   LD       DE,BYTEO
1803  210319                   LD       HL,OUTBF
1806  0603                     LD       B,3
1808  1A          LOOP         LD       A,(DE)
1809  CD7806                   CALL     HEX7SG
180C  13                       INC      DE
180D  10F9                     DJNZ     LOOP
                 ;CONVERSION COMPLETE, BREAK FOR CHECK
180F  DD210319                 LD       IX,OUTBF
1813  CDFE05                   CALL     SCAN
1816  76                       HALT
                 ;
1900                           ORG      1900H
1900  10         BYTEO         DEFB     10H
1901  32                       DEFB     32H
1902  54                       DEFB     54H
1903             OUTBF         DEFS     6
                 ;
                 HEX7SG        EQU      0678H
                 SCAN          EQU      05FEH
                               END
```

The three bytes binary data are stored in 1900 - 1902. The user can set break point at 180F to check if the conversion is correct before displaying the result.

III. Illustrations of Experiments

(1)

(a) Load program 1 into MPF-I and store it on audio tape for future application.

(b) Test this program and record the display response.

(c) Change the content of 1808 to 1A, then display will HALT with STEP-KEY replaced by CBR-KEY, why?

(d) Change the contents of 1820 - 1822 with 3F, BD, 85 respectively what will the display show?

(e) Write a program to display "SYS-SP", HALT when PC-KEY is depressed.

(2)

(a) Load program 2 into MPF-I and store it on audio tape for future application.

(b) Test this program and record the display response.

(c) Change the content of 180B to 01, what will the display show?

(d) Change the content of 180B to 05 and to see what display will show?

(e) Write a program to change the flash frequency. The word format "HELP US" is required to be lit at a rate of 2 secc per cycle.

(3)

(a) Change the contents of 1900 - 1905 to FF, what will the display show?

(b) What is the meaning of position code and internal code in monitor program?

(4)

(a) Change the contents of 1900 - 1902 so that the display will show "333446".

Experiment 12
Fire-Loop Game

Purpose:

1. To understand how to use a subroutine contained in the monitor program.
2. To familiarize the reader with programming techniques.

Time Required: 4 hours

I. Theoretical Background:

1. Monitor Program:

   After the microcomputer is powered on, it will execute programs from the designated address. Besides some initialization task (e.g. setting 8255 or selecting I/O mode), a special software program called monitor is used to monitor the presence of data or commands from peripheral devices (e.g. a keyboard, an external switch, a button, a sensor, etc.) If no signal is monitored, then the scanning process continues (using the looping method to search) until a signal input is detected. The input signal is then analyzed and the microcomputer jumps to the service routine to perform the job assigned by the input signal. After this service routine has been executed, the microcomputer returns to scan the peripheral devices.

   Since MPF-I is a general-purpose microcomputer, it has a monitor. The main function of this monitor is to respond to key closures on the keyboard and displaying necessary data. Tracing the monitor program will improve your programming skill.

2. Fig. 12-1 is the flowchart of the Fire Loop.

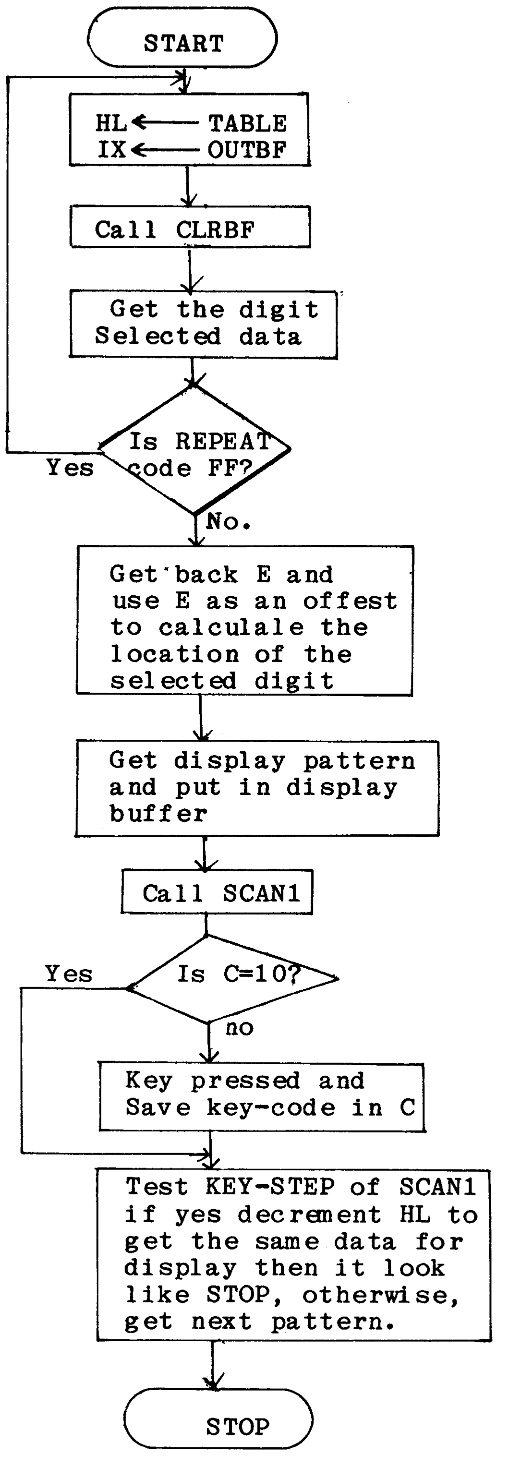


Fig. 12-1 The Flowchart of Fire Loop

```
LOC     OBJ CODE M STMT SOURCE STATEMENT

                        ;
                        ; Segment Illuminates one by one until key-step is pushed
                        ; Any other key will resume looping again.
                        ;
1800                             ORG     1800H
1800    214018          INI      LD      HL,TABLE
1803    DD210019                 LD      IX,OUTBF
1807    CD3018          LOOP     CALL    CLRBF    ;Clear display buffer
180A    5E                       LD      E, (HL)  ;Get the DIGIT-select data
180B    1C                       INC     E        ;Test REPEAT code: FF
180C    28F2                     JR      Z,INI    ;If yes, go to INI
180E    1D                       DEC     E        ;Otherwise, get back E
180F    1600                     LD      D, 0     ;Use E as an offset to
1811    DD19                     ADD     IX,DE    ;Calculate the location of
                                                  ;the selected digit.
1813    23                       INC     HL
1814    7E                       LD      A, (HL)  ;Get display PATTERN
1815    DD7700                   LD      (IX),A   ;Put in display buffer
1818    DD210019                 LD      IX,OUTBF
181C    0603                     LD      B,SPEED
                        ;
                        ; The following 4 instruction display the pattern
                        ; for B times (can be adjusted in the above SPEED)
                        ;
181E    CD2406          LIGHT    CALL    SCAN1
1821    3801                     JR      C,NSCAN
1823    4F                       LD      C,A      ;Key pressed, save key-code in C
                                                  ;Note that, reg C will not be
                                                  ;Changed until next key input
1824    10F8            NSCAN    DJNZ    LIGHT
                        ;
1826    79                       LD      A,C
1827    FE10                     CP      10H      ;Test KEY-STEP of SCAN1
1829    2802                     JR      Z,STOP   ;If yes, decrement HL to get
                                                  ;the same data for display
                                                  ;Then it locks like STOP.
182B    23                       INC     HL       ;Otherwise, get next pattern
182C    23                       INC     HL
182D    2B              STOP     DEC     HL
182E    18D7                     JR      LOOP
                        ;
                        ;
                        CLRBF:
1830    0606                     LD      B,6
1832    DD360000        CLR      LD      (IX),0
1836    DD23                     INC     IX
1838    10F8                     DJNZ    CLR
183A    11FAFF                   LD      DE,-6    ;Get original IX
183D    DD19                     ADD     IX,DE
183F    C9                       RET
```

```
                ;
                ; The 1st byte indicates which DIGIT is to be seleced
                ; The 2nd byte indicates what PATTERN to be displayed
                ;
1840    05      TABLE   DEFB    5
1841    08              DEFB    SEG_A
1842    04              DEFB    4
1843    08              DEFB    SEG_A
```

```
LOC     OBJ CODE M STMT SOURCE STATEMENT

1844    03              DEFB    3
1845    08              DEFB    SEG_A
1846    02              DEFB    2
1847    08              DEFB    SEG_A
1848    01              DEFB    1
1849    08              DEFB    SEG_A
184A    00              DEFB    0
184B    08              DEFB    SEG_A
184C    00              DEFB    0
184D    10              DEFB    SEG_B
184E    00              DEFB    0
184F    20              DEFB    SEG_C
1850    00              DEFB    0
1851    80              DEFB    SEG_D
1852    01              DEFB    1
1853    80              DEFB    SEG_D
1854    02              DEFB    2
1855    80              DEFB    SEG_D
1856    03              DEFB    3
1857    80              DEFB    SEG_D
1858    04              DEFB    4
1859    80              DEFB    SEG_D
185A    05              DEFB    5
185B    80              DEFB    SEG_D
185C    05              DEFB    5
185D    01              DEFB    SEG_E
185E    05              DEFB    5
185F    04              DEFB    SEG_F
1860    FF              DEFB    0FFH
                ;
1900                    ORG     1900H
1900            OUTBF   DEFS    6
                ;
                SPEED   EQU     3
                SEG_A   EQU     08H
                SEG_B   EQU     10H
                SEG_C   EQU     20H
```

```
SEG_D   EQU   80H
SEG_E   EQU   01H
SEG_F   EQU   04H
SCAN1   EQU   0624H
        END
```

3. Further Expriments

(a) Load the above program into MPF-I and then store it on audio tape for future applications.
Test this program and record the display response.

(b) Write a program to make the Fire-Loop illuminate counterclockwise.

(c) Change the contents of 1828. Then KEY-STEP will not respond as before. Why?

(d) Change the contents of 181D and the display will change. Why?

(e) Write a program that will cause the segments to move in a pattern of your choice.

(f) Write a delay program to display "HELP US" for 20 secs, then the display will play the "Fire Loop Game".

Experiment 13

Stop-Watch

Purpose:

1. To illustrate how to use monitor subroutines.

2. To practise programming skills.

Time Required: 2 hours

I. Theoretical Background:

1. The object specification of this experiment is to design a 1/100 second-based stop watch. Actually, this is only roughly accurate. The accuracy varies with the system clock and the number of instructions used in the keyboard/display scan subroutine.

2. The demonstration program calls two monitor subroutines SCAN1 and HEX7SG which are located at 0624H and 0678H respectively.

   The flowchart is given below.

3. The counting procedure is halted by depressing a key. This is done by checking the result of SCAN1 routine.

4. Flow Chart of STOP WATCH

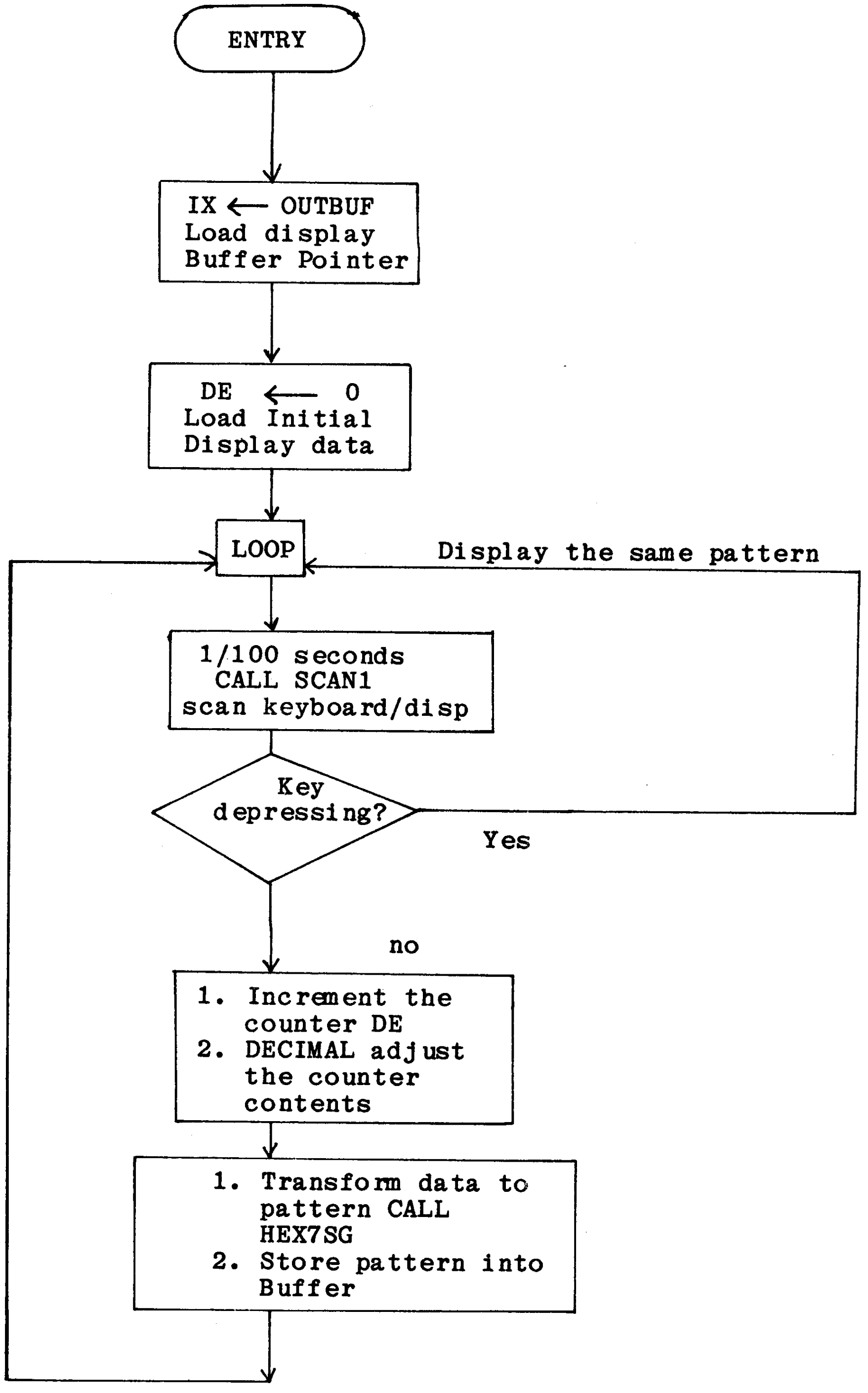


Fig 13-1 Flowchart of stop watch

5. Stop Watch program

```
LOC   OBJ CODE M STMT SOURCE STATEMENT

1800             1          ORG   1800H
1800  DD210019   2          LD    IX,OUTBF ;initial display pointer
1804  110000     3          LD    DE,0     ;initial SEC & 1/100 SEC in DE
1807  CD2406     4LOOP      CALL  SCAN1    ;display for 0.01 second
180A  30FB       5          JR    NC,LOOP  ;if any key pressed, then NC
                 6                         ;so looping the same pattern
180C  7B         7          LD    A,E      ;otherwise increment 1/100
                                            SEC by 1
180D  C601       8          ADD   A,1
180F  27         9          DAA
1810  5F        10          LD    E,A
1811  7A        11          LD    A,D      ;if carry, increment SEC again
1812  CE00      12          ADC   A,0
1814  27        13          DAA
1815  57        14          LD    D,A
1816  7B        15          LD    A,E      ;convert 1/100 SEC to display
                                            format
1817  210019    16          LD    HL,OUTBF ;and put them into display
                                            buffer
181A  CD7806    17          CALL  HEX7SG
181D  3602      18          LD    (HL),2    ;put into display of '-'
181F  23        19          INC   HL
1820  7A        20          LD    A,D       ;convert SEC to display format
1821  CD7806    21          CALL  HEX7SG    ;and put them into display
                                             buffer
1824  3600      22          LD    (HL),0    ;put BLANK into MSD
1826  18DF      23          JR    LOOP
                24
1900            25          ORG   1900H
1900            26 OUTBF    DEFS  6
                27 HEX7SG   EQU   0678H
                28 SCAN1    EQU   0624H
                29          END
```

II. Illustration of the Experiments

(1) Load the program and GO!

(2) Depress any key other than RS and MONI, rscord the results.

(3) Note that the program will loop continuously. The user can interrupt the execution only by RS or MONI.

(4) Users are encouraged to modify the program:

a. Build a 1/10 second based stop watch.

b. Display all zeros at the beginning, start the stop watch by depressing an arbitrary key or the user defined key.

c. Build a stop key.

(5) Check the timing on the display with your watch for one minute. Perhaps, there is an error. Try to find the reasons for the error and note them.

Experiment 14

Clock 1 (How to design a clock)

Purposes:

1. To practise calculating the clock cycle of a program.
2. To construct a software driven digital clock.

Time Required: 4 hours.

I. Theoretical Backqround:

1. This is an example of how to use the software delay to build a digital clock.
2. All the timing is based on the system clock, which is 1.79 MHZ. So that 1 cycle is about 0.56 micro-seconds.
3. The total number of cycles in ONE LOOP has been carefuly calculated.
4. The cycle count calculation is given as follow:

   SCAN1 : 17812

   LOOP1 : (17 + 17812 + 13) * 100 - 5 = 1784195

   TMUPDT : 258

   BFUPDT : 914

   LOOP2 : (4 + 13) * 256 - 5 = 4347

   The total number of counts is 1789755

   and

   0.56 usec x 1789755 ≑ 1 sec

5. Flowchart of clock

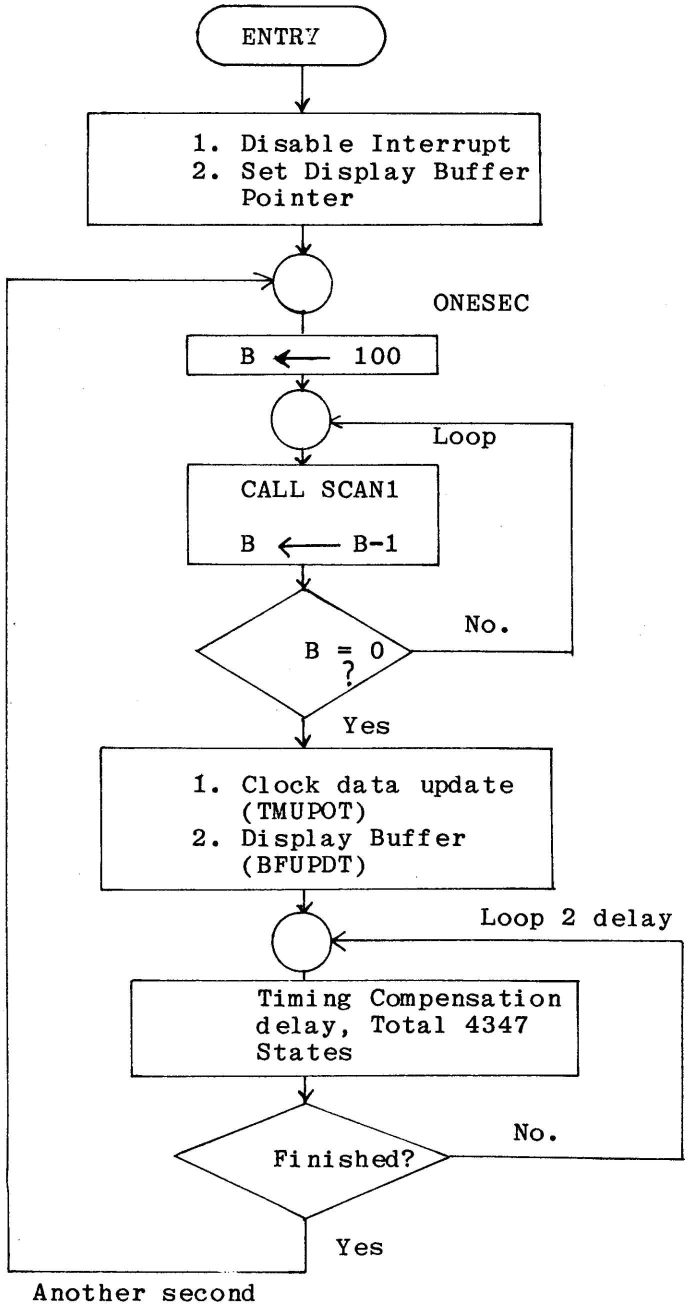


Fig 14-1 Flowchart of clock

Time Update Flowchart

Display buffer Update Flowchart

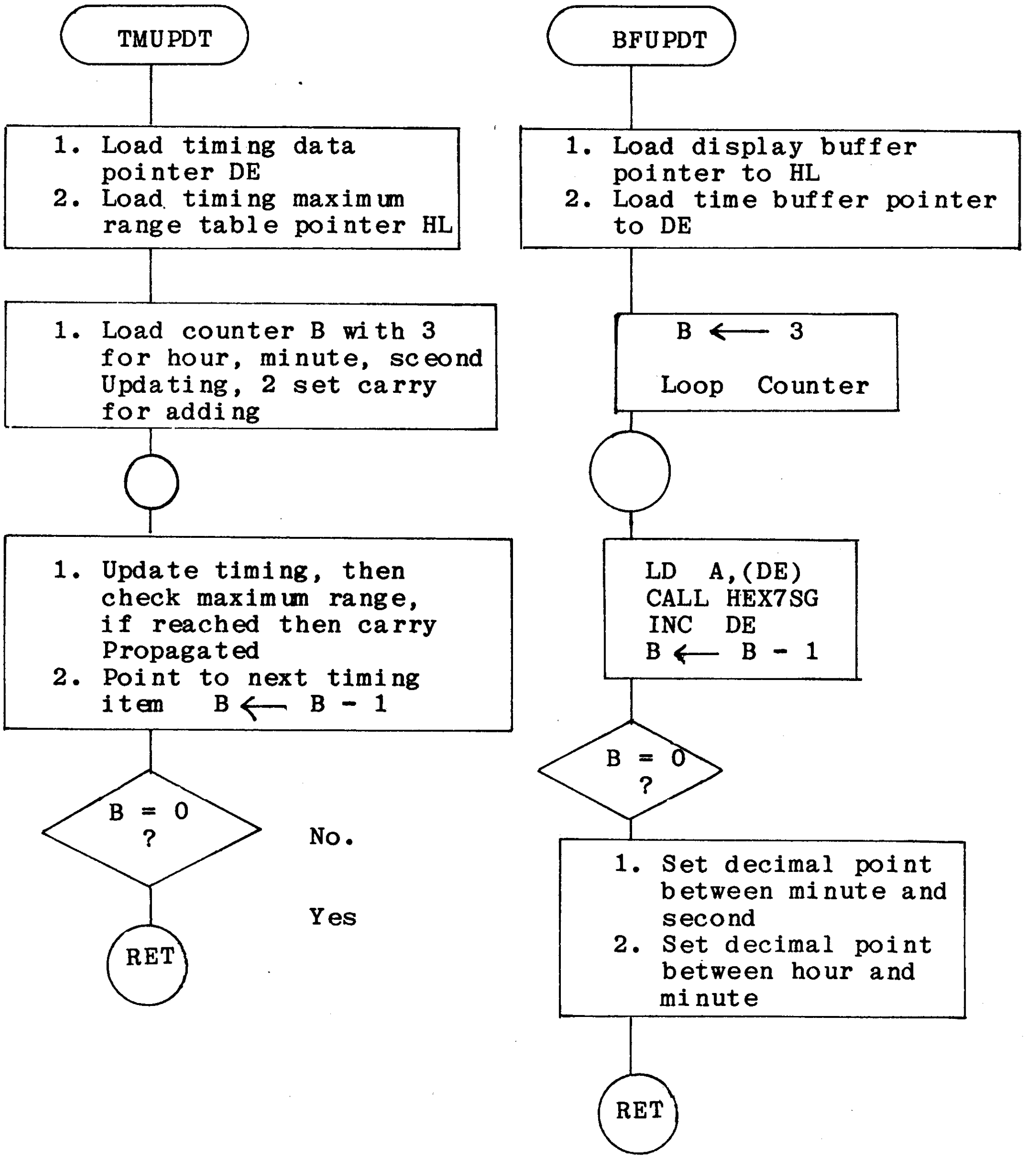


Fig 14-2 Flowchart of Update

6. Program of software designed clock

```
LOC   OBJ CODE M STMT SOURCE STATEMENT


1800              1        ORG     1800H
1800  F3          2        DI      ;Disable interrupt, which affects
                                    timing
1801  DD21031A    3        LD      IX,OUTBF
                  4 ;
                  5 ;ONESEC loop takes 1 second to execute, it
                     consists of 3
                  6 ;subroutines & 1 additional delay process
                  7 ;
                  8 ;ONESEC:
1805  0664        9        LD      B,100        ;7
1807  CD2406     10 LOOP1  CALL    SCAN1
180A  10FB       11        DJNZ    LOOP1        ;(17+17812+13)*100-5=1784195
180C  CD1718     12        CALL    TMUPDT       ;17+258=275
180F  CD2F18     13        CALL    BFUPDT       ;17+914=931
1812  00'        14 LOOP2  NOP
1813  10FD       15        DJNZ    LOOP2        ;(4+13)*256-5=4347
1815  18EE       16        JR      ONESEC       ;12
                 17 ;
                 18 ;Time-buffer is updated here.
                 19 ;Note that this routine takes the same time in any
                 20 ;condition, 275 cycles.
                 21 ;
                 22 TMUPDT:
1817  214718     23        LD      HL,MAXTAB
181A  11001A     24        LD      DE,SEC
181D  0603       25        LD      B, 3
181F  37         26        SCF                  ;Set carry flag: force add 1
1820  1A         27 TMINC  LD      A,(DE)
1821  CE00       28        ADC     A,0
1823  27         29        DAA
1824  12         30        LD      (DE),A
1825  96         31        SUB     (HL)         ;Compare wth data in MAX_TAB
                 32                             ;if the result is less than
                                                 that.
                 33                             ;the following loop will be
                                                 null.
                 34                             ;delay, because of no carry
                                                 propagation
                 35                             ;
1826  3801       36        JR      C,COMPL
1828  12         37        LD      (DE),A
1829  3F         38 COMPL  CCF                  ;complement carry flag
182A  23         39        INC     HL
182B  13         40        INC     DE
182C  10F2       41        DJNZ    TMINC
182E  C9         42        RET
```

```
LOC   OBJ CODE M STMT SOURCE STATEMENT

                  44 ;Display_buffer is updated here.
                  45 ;It takes 914 cycles.
                  46 ;
                  47 BFUPDT:
182F  21031A      48          LD      HL,OUTBF
1832  11001A      49          LD      DE,SEC
1835  0603        50          LD      B,3
1837  1A          51 PUTBF    LD      A,(DE)
1838  CD6D06      52          CALL    HEX7SG
183B  13          53          INC     DE
183C  10F9        54          DJNZ    PUTBF
183E  2B          55          DEC     HL
183F  2B          56          DEC     HL
1840  CBF6        57          SET     6,(HL)    ;Set decimal point of HOUR
1842  2B          58          DEC     HL
1843  2B          59          DEC     HL
1844  CBF6        60          DEC     HL        ;Set decimal point of MTNUTE
1846  C9          61          RET                ;B=0 when return
                  62 ;
                  63 MAXTAB:
1847  60          64          DEFB    60H
1848  60          65          DEFB    60H
1849  12          66          DEFB    12H
                  67 ;
1A00             69          ORG     1A00H
                  70 TMBF:
1A00              71 SEC      DEFS    1
1A01              72 MIN      DEFS    1
1A02              73 HOUR     DEFS    1
                  74 ;
1A03              75 OUTBF    DEFS    6
                  76 ;
                  77 SCAN1    EQU     624H
                  78 HEX7SG   EQU     66DH
                  79          END
```

II. Illustrations of the Experiments

1. Load the program
2. Load the time data into TMBF (1A00 - 1A02)
3. Observe the results.
4. What will happen if preceed as follows?

Key	Display
ADDR 1 8 4 7	1 8 4 7 6 0
DATA 2 4	

5. This program can be modified to be a second counter with arbitrary base. Design a 20 seconds, 20 minutes and 1 hour counter.
6. Trace the program, and draw your own flowchart. Are there any differences between your flow chart and the demonstrated flowchart above?

   Explain why differences occurred.

Experiment 15
Clock 2 (with CTC interrupt mode 2)

Experiment Purpose:

1. To practise using Interrupt Mode 2 through the CTC.
2. To practise programming.

Time Required: 8 hours

I. Theoretical Background:

I-1. Introduction to the Z80 CPU interrupt:

1. Z80 CPU Interrupt Request Lines :

The Z80 CPU can suspend the current program execution by external interrupt request. The CPU then starts executing the interrupt service routine. Once the service routine is completed, the CPU returns to the main program from which it was interrupted.

The Z80 CPU has two interrupt inputs : a non-maskable interrupt and a software maskable interrupt. The non-maskable interrupt (NMI) line can not be disabled by the programmer and will be activated whenever an external device inputs an interrupt request to it. The maskable interrupt (INT) line can be disabled by resetting an internal Interrupt Enable Flip Flop (IFF). The enable flip flop can be set or disabled by the programmer using Enable Interrupt (EI) and Disable Interrupt (DI) instructions.

2. NMI Request :

The NMI signal is sampled by the CPU at the rising edge of the last clock at the end of any instruction. The NMI request line will be at logic "0" if there is a non-maskable interrupt request. The CPU automatically saves the program counter (PC) in the stack area and jumps to location 0066H (a fixed memory address assigned by the Z80 CPU). THE CPU will not respond to any further NMI request. The CPU then executes the service routine until a RETN instruction appears and then it fetches the PC of main program from the stack to continue the execution of main program. At this time, the CPU can accept another NMI request.

In MPF-I, memory addresses 0000H through 07FFH are for the monitor program. Once a non-maskable interrupt is accepted, the CPU automatically jumps to location 0066H . The non-maskable interrupt request line has a higher priority than any other interrupt.
It is very useful in event of a power failure, which obviously takes precedence over all other activities. For instance, if the voltage level of the power supply battery of the micro computer

drops to a certain level, then a voltage comparator circuit will activate a non-maskable interrupt request signal. The CPU then suspends its current program execution and starts battery-recharging. The recharging process is controlled by a software program. The starting address of this control sequence must be at 0066H.

3. INT Interrupt Request :

The interrupt request at the INT line can be masked. For instance, after the battery-recharging process has been started, the CPU can return to its main program execution. When the battery is charged to certain level, another voltage comparator circuit will generate an INT interrupt request signal. If the CPU is not executing a very important program, then it may acknowledge the interrupt request and jump to a service routine designed to stop the recharging process. Usually, stopping the recharging process is not an emergency task, thus the CPU may continue to execute an important program and ignore this kind of interrupt request. For instance, when the CPU is reading data from a tape, and interrupt will cause the data in the tape to be missed. Thus, if a DI instruction is included at the beginning of the "Read Tape" routine, then the INT interrupt request will be masked. An EI (Enable Interrupt) is usually included at the end of the "Read Tape" routine in order to enable the INT interrupt request line.

The Z80 CPU can be programmed to respond to the maskable interrupt in three possible modes by the IM (Interrupt Mode) instruction. With IMO mode, whenever the CPU receives an instruction (usually, it is a "RESTART" operation) in the data bus from a peripheral device, then the CPU will jump to one of the 8 fixed of memory addresses (0000-0038H) and execute the program. When the IM1 mode has been selected by programmer, the CPU will respond to an interrupt by executing a restart instruction to location 0038H. In MPF-I mode 0 can not be used because the addresses specified for the restart instructions are already reserved for the monitor program.

The last mode is the IM2 mode which is the most powerful interrupt response mode. With this mode, the programmer maintains a table of 16-bit starting addresses for interrupt service routines. The low-order 8-bits of the pointer must be supplied by the interrupting device. The high-order 8 bits of the pointer is formed from the contents of the internal I register (Interrupt Vector Register). When an interrupt is accepted, the 16-bit pointer must be formed to obtain the starting address of the desired interrupt service routine from the table.

If the Z80 input/output interface devices (PIO, CTC, SIO) are used in the microcomputer system, then the IM2 mode will give rise to the most useful interrupt request response.

II. Example Experiments:

1. Testing the NMI interrupt response :

   (1) An interrupt request may be generated by touching a copper wire connected to the NMI input line of the CPU to the ground. After touching the NMI input line of the CPU, then the CPU will execut the program with starting address at 0066H.

2. Testing the INT interrupt response :

After a reset, the Z80 CPU will be automatically in the IMO interrupt response mode and will disable the interrupt enable flip flop. Thus, before the CPU responds to the interrupt request, the following program must be executed.

```
ED5E  IM   2       ; Select interrupt mode 2.
3E 18 LD   A, 18H
ED 47 LD   I,A     ; Assign 18H as the high-order byte of
                     the interrupt vector address.
FB    EI           ; Enable the interrupt request line INT.
```

In case the Z80 peripheral devices are not used in the system, the interrupt request signal is sent directly to the CPU. When the CPU acknowledges an interrupt request, the 8-bit data must be read in as the low-order byte of the vector address. If there is no electronic circuit for supplying this 8-bit vector address, then the data bus will be pulled up to "high" voltage state (logic "1") and read as FFH. That is, the CPU will form 18FFH as the 16-bit vector address. This 16-bit vector address is used as a pointer to obtain the starting address of the desired interrupt service routine from the table.

Suppose the starting address of the interrupt service routine is arranged at 1920H, then the number 1920H must be stored in memory addresses 18FFH and 1900H. Load the following program into MPF-I for later testing.

```
                                          FF
LOC OBJ CODE M STMT SOURCE STATEMENT

1800              1          ORG    1800H
1800 3E18         2          LD     A,18H
1802 ED47         3          LD     I,A       ; Define high-order vector address.
1804 212019       4          LD     HL,1920H
1807 22FF18       5          LD     (18FFH),HL ;Store interrupt vector.
180A ED5E         6          IM     2
180C FB           7          EI
180D F7           8          RST    30H       ; Return to monitor program.
1920              9          ORG    1920H     ; Interrupt service routine.
1920 211234      10          LD     HL,3412H
1923 224019      11          LD     (1940H),HL  ;Store 3412H to RAM (1940H).
1926 FB          12          EI               ; Enable another interrupt.
1927 ED4D        13          RETI             ; Return from interrupt.
```

(1) Execute the above program by depressing the control key on the keyboard. After the program is executed, the monitor will resume control of the microcomputer. The interrupt request line INT is also enabled. Then, key in some arbitrary numbers into RAM addresses 1940H and 1941H, and depress the INTR key in the keyboard. That is, an interrupt request signal is input to the CPU INT line. Depress the AD key in teh keyboard to reset the display buffer in the monitor program. Check if the interrupt service routine with starting address 1920H is executed so that the designated numbers have been stored in RAM addresses 1940H - 1941H. Repeat the testing several times (change the contents of RAM before each test).

Results of test:

(2) Instruction RETI is used as the end of an interrupt service It is a routine to signal the I/O device that the interrupt routine has been completed. It facilitates the nesting of routine allowing higher priority devices to suspend service of lower priority service routines. The standard Z80 I/O devices are not used in this experiment, thus, RETI is not a necessary instruction. Replace instruction RETI in the above program by RST 30H and then repeat the test in (1). Record the result shown in the display after the interrupt request signal is input to the CPU. Discuss the results of the test.

(3) instruction EI (Enable Interrupt) must be included in every interrupt service routine, otherwise the INT line will be disabled after the CPU acknowledges an INT interrupt request. Instruction EI must be used to enable the maskable interrupt. The function of the EI instruction can not be replaced by that of the RETI instruction.

Replace instruction Repeat the test for interrupt request and to show that only the first interrupt is acknowledged and all other interrupts are ignored.

Results of test:

(4) Write a program that will cause the PAO line of the Z80 PIO to output "1" after the CPU receives an INT interrupt request and clear this line to "0" after 3 seconds has elapsed.

(I-2) Introduction to the Z80-CTC :

1.0 INTRODUCTION

The Z80-Counter Timer Circuit (CTC) is a programmable component with four independent channels that provide counting and timing functions for microcomputer systems based on the Z80-CPU. The CPU can configure the CTC channels to operate under various modes and conditions as required to interface with a wide range of devices. In most applications, little or no external logic is required. The Z80-CTC utilizes N-channel silicon gate depletion load technology and is packaged in a 28-pin DIP. The Z80-CTC requires only a single 5 volt supply and a one-phase 5 volt clock. Major features of the Z80-CTC include :

* All inputs and outputs are fully TTL compatible.
* Each channel may be selected to operate in either Counter Mode or Timer Mode.
* Used in either mode, a CPU-readable Down Counter indicates number of counts-to-go until zero.
* A Time Constant Register can automatically reload the Down Counter at Count Zero in both Counter and Timer Modes.
* A selectable positive or negative trigger initiates time operation in Timer Mode. The same input is monitored for event counts in Counter Mode.
* Three channels have Zero Count/Timeout outputs capable of driving Darlington transistors.
* Interrupts may be programmed to occur on the zero count condition in any channel.
* Daisy chain priority interrupt logic included to provide for automatic interrupt vectoring without external logic.

2.0 CTC ARCHITECTURE

2.1 OVERVIEW

A block diagram of the Z80-CTC is shown in figure 2.0-1. The internal structure of the Z80-CTC consists of a Z80-CPU bus interface, Internal Control Logic and four sets of Counter/Timer Channels. Timer channels are identified by sequential numbers from 0 to 3. The CTC has the capability of generating a unique interrupt vector for each separate channel (for automatic vectoring to an interrupt service routine). The 4 channels can be connected into four contiguous slots in the standard Z80 priority chain with channel number 0 having the highest priority. The CPU bus interface logic allows the CTC device to interface directly to the CPU with no other external logic. However, port address decoders and/or line buffers may be required for large systems.

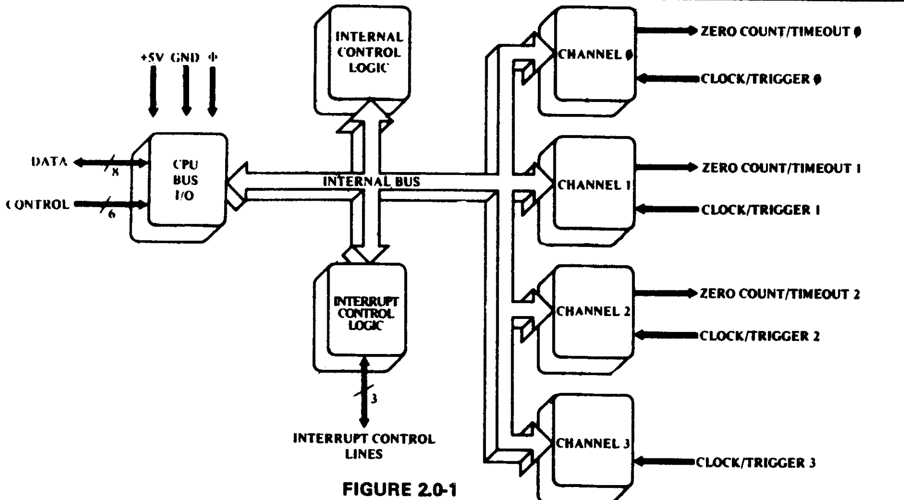


FIGURE 2.0-1
CTC BLOCK DIAGRAM

2.2 STRUCTURE OF CHANNEL LOGIC

The structure of one of the four sets of Counter/Timer Channel Logic is shown in figure 2.0-2. This logic is composed of 2 registers, 2 counters, and control logic. The registers are an 8-bit Time Constant Register and an 8-bit Channel Control Register. The counters are an 8-bit CPU-readable Down Counter and an 8-bit prescaler.

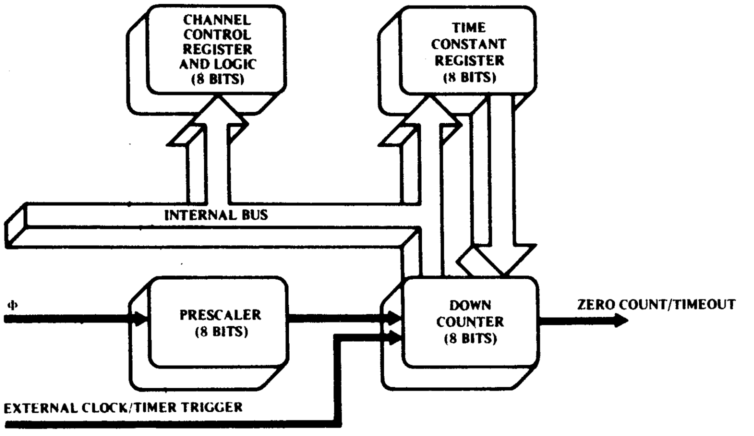


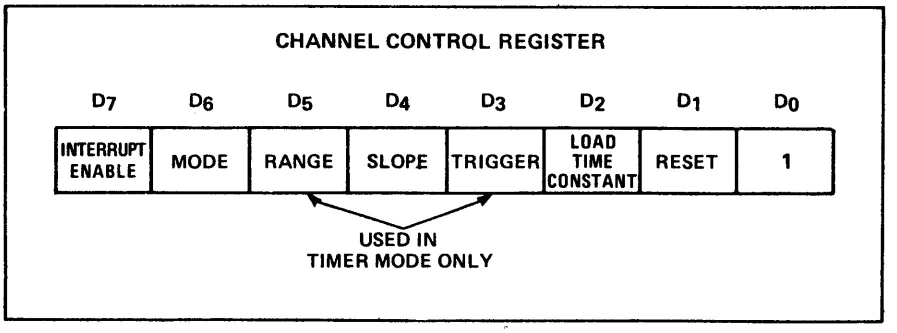
FIGURE 2.0-2
CHANNEL BLOCK DIAGRAM

2.2.1 THE CHANNEL CONTROL REGISTER AND LOGIC

The Channel Control Register (8-bits) and Logic is written to by the CPU to select the modes and parameters of the channel. Within the entire CTC device there are four such registers, corresponding to the four Counter/Timer Channels. Which of the four is being written into depends on the encoding of two channel select input pins : CSO and CS1 (usually attached to A0 and A1 of the CPU address bus. This is illstrated in the truth table below.

	CSI	CSO
Ch0	0	0
Ch1	0	1
Ch2	1	0
Ch3	1	1

In the control word written to program each Channel Control Register, bit 0 is always set and the other 7 bits are programmed to select alternative channel's operating modes and parameters, as shown in the diagram below, (For a more complete discussion see section 4.0 "CTC Operating Modes" and section 5.0 "CTC Programming").



2.2.2 THE PRESCALER

Used in the Timer Mode only, the Prescaler is an 8-bit device which can be programmed by the CPU via the Channel Control Register to divide its input, the System Clock (Φ), by 16 or 256. The output of the Pre-scaler is then fed as an input to clock the Down Counter, which initially, and every time it clocks down to zero, is reloaded automatically with the contents of the Time Constant Register. In effect this again divides the System Clock by an additional factor of the time constant. Every time the Down Counter counts down to zero, its output, Zero Count/Timeout (ZC/TO), is pulsed high.

2.2.3 THE TIME CONSTANT REGISTER

The Time Constant Register is an 8-bit register, used in both Counter Mode and Timer Mode, programmed by the CPU just after the Channel Control Word. It has an integer time constant value of 1 through 256. This register loads the programmed value into the Down Counter when the CTC is first initialized and reloads the same value into the Down Counter automatically whenever it counts down thereafter to zero. If a new time constant is loaded into the Time Constant Register while a channel is counting or timing, the present down count will be completed before the new time constant is loaded into the Down Counter. (For details of how a time constant is written into a CTC channel, see section 5.0: "CTC Programming.")

2.2.4 THE DOWN COUNTER

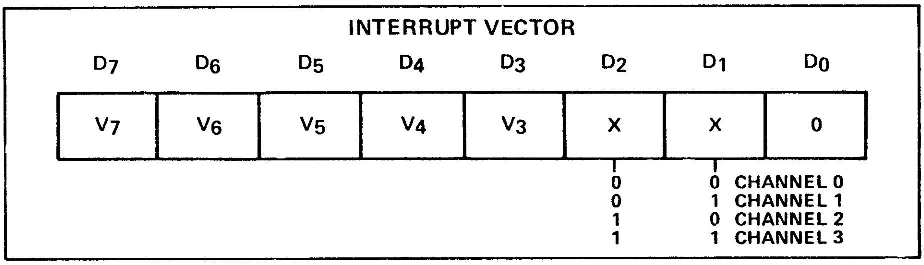
The Down Counter is an 8-bit register, used in both Counter Mode and Timer Mode loaded initially, and later when it counts down to zero, by the Time Constant Register. The Down Counter is decremented by each external clock edge in the Counter Mode or in the Time Mode, by the clock output of the Prescaler. At any time, by performing a simple I/O Read at the port address assigned to the selected CTC channel, the CPU can access the contents of this register and obtain the number of counts-to-zero. Any CTC channel may be programmed to generate an interrupt request sequence each time the zero count is reached.

In channels 0, 1 and 2, when the zero count condition is reached, a pulse appears on the correspronding ZC/TO pin. Due to package pin limitations, however, channel 3 does not have this pin and so may be used only in applications where this output pulse is not required.

2.3 INTERRUPT CONTROL LOGIC

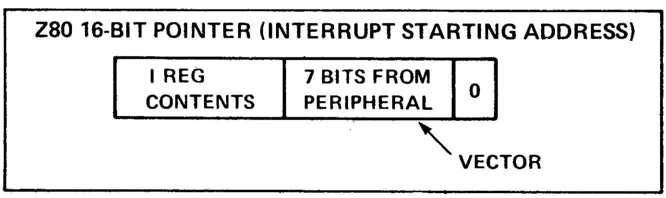
The Interrupt Control Logic insures that the CTC acts in accordance with Z80 system interrupt protocol for nested priority interrupting and return from interrupt. The priority of any system device is determined by physical location in a daisy chain configuration. Two signal lines (IEI and IEO) are provided in CTC devices to form this system daisy chain. The device closest to the CPU has the highest priority: within the CTC interrupt priority is predetermined by channel number with channel 0 having highest priority down to channel 3 which has the lowest priority. The purpose of a CTC-generated interrupt, as with any other peripheral device is to force the CPU to execute an interrupt service routine. According to Z80 system interrupt protocol, lower priority devices or channels may not interrupt higher priority devices or channels that have already interrupted and have not had their interrupt service routines completed. However, high priority devices or channels may interrupt the servicing of lower priority devices or channels.

A CTC channel may be programmed to request an interrupt every time its Down Counter reaches a count of zero. (To utilize this feature requires that the CPU be programmed for interrupt mode 2.) Sometime after the interrupt request, the CPU will send out an interrupt acknowledge, and the CTC's Interrupt Control Logic will determine the highest-priority channel which is requesting an interrupt within the CTC device. Thus if the CTC's IEI input is active, indicating that it has priority within the system daisy chain, it will place an 8-bit Interrupt Vector on the system data bus. The high-order 5 bits of this vector will have been written to the CTC earlier as part of the CTC initial programming process the next two bits will be provided by the CTC's Interrupt Control Logic as a binary code corresponding to the highest-priority channel requesting an interrupt; finally the low-order bit of the vector will always be zero according to a convention described below.



This interrupt vector is used to form a pointer to a location in memory where the address of an interrupt service routine is stored in a table. The vector represents the least singificant 8 bits, while the CPU reads the contents of the I register to provide the most significant 8-bits of the 16-bit pointer. The address in memory

pointed to will contain the low-order byte, and the next highest address will contain the high-order byte of an address which in turn contains the first opcode of the interrupt service routine. Thus in mode 2, a single 8-bit vector stored in an interrupting CTC can result in an indirect call to any memory location.



There is a Z80 system convention that all addresses in the interrupt service routine table should have their low-order byte in the next highest location in memory, and their high-order byte in the next highest location in memory. Which will alway be odd so that the least significant bit of any interrupt vector will always be even. Hence the least significant bit of any interrupt vector will always be zero.

The RETI instruction is used at the end of any interrupt service routine to initialize the daisy chain enable line IEO for proper control of nested priority interrupt handling. The CTC monitors the system data and decodes this instruction when it occurs. Thus the CTC channel control logic will know when the CPU completed servicing an interrupt, without any further communication with the CPU being necessary.

3.0 CTC PIN DESCRIPTION

A diagram of the Z80-CTC pin configuration is shown in figure 3.0.1. This section describes the function of each pin.

D7-D0

Z80-CPU Data Bus (bi-directional, tri-state)

This bus is used to transfer all data and command words between the Z80-CPU and the Z80-CTC. There are 8 bits on this bus, of which D0 is the least significant.

CS1-CS0

Channel Select (input, active high)

These pins form a 2-bit binary address code for selecting one of the four independent CTC channels for an I/O Write or Read. (See truth table below.)

	CS1	CS0
Ch 0	0	0
Ch 1	0	1
Ch 2	1	0
Ch 3	1	1

$\overline{CE}$

Chip Enable (input, active low)

A low level on this pin enables the CTC to accept control words, Interrupt Vectors, or a time constant, date words from the Z80 Data Bus during and I/O Write cycle, or to transmit the contents of the Down Counter to the CPU during an I/O Read cycle. In most applications this signal is decoded from the 8 least significant bits of the address bus for any of the four I/O port addresses that are mapped to the four Counter/Timer Channels.

Clock (Φ)

System Clock (input)

This single-phase clock is used by the CTC to synchronize certain signals internally.

$\overline{M1}$

Machine Cycle One Signal from CPU (input, active low)

When M1 is active and the RD signal is active, the CPU is fetching an instruction from memory. When M1 is active and the IORO signal is active, the CPU is acknowledging an interrupt or alerting the CTC to place an interrupt Vector on the Z80 Data Bus if it has daisy chain priority and one of its channels has requested an interrupt.

$\overline{IORQ}$

Input/Output Request from CPU (input, active low)

The IORQ signal is used in conjunction with the CE and RD signals to transfer data and Channel Control Words between the Z80-CPU and the CTC, During a CTC Write Cycle, IORQ and CE must be true and RD FALSE. The CTC does not receive a specific write signal. Instead it generates its own internally from the inverse of a valid RD signal. In a CTC Read Cycle, IORQ, CE, and RD must be active to place the contents of the Down Counter on the Z80 Data Bus. If IORQ and M1 are both true, the CPU is acknowledging an interrupt request. and the highest-priority interrupting channel will place its Interrupt Vector on the Z80 Data Bus.

3.0 CTC PIN DESCRIPTION (CONT'T)

$\overline{RD}$

Read Cycle Status from the CPU (input, active low).

The RD signal is used in conjunction with the IORQ and CE signals to transfer data and Channel Control Words between the Z80-CPU and the CTC. During a CTC Write Cycle, IORQ and CE must be true and RD false. The CTC does not receive a specific write signal, instead generating its own internally from the inverse of a valid RD signal. In a CTC Read Cycle, IORQ, CE and RD must be active to place the contents of the Down Counter on the Z80 Data Bus.

IEI

Interrupt Enable In (input, active high)

This signal is used to help form a system-wide interrupt daisy chain which establishes priorities when more than one peripheral device in the system has interrupting capability. A high level on this pin indicates that no other interrupting devices of higher priority in the daisy chain are being sericed by the Z80-CPU.

IEO
Interrupt Enable Out (output, active high)

The IEO signal, in conjunction with IEI, is used to form a system-wide interrupt priority daisy chain. IEO is high only if IEI is high and the CPU is not servicing an interrupt from any CTC channel. Thus this signal blocks lower priority devices from interrupting while a higher priority interrupting device is being serviced by the CPU.

$\overline{\text{INT}}$

Interrupt Request (output, open drain, active low)

This signal goes true when any CTC channel which has been programmed to enable interrupts has a zerocount condition in its Down Counter.

$\overline{\text{RESET}}$

Reset (input, acive low)

This signal stops all channels from counting and resets channel interrupt enable bits in all control resisters thereby disabling CTC-generated interrupts. The ZC/TO and INT outputs go to their inactive states. IEO reflect IEI, and the CTC's data bus output drivers go-to the high impedance state.

CLK/TRG3-CLK/TRGO

External Clock/Timer Trigger (input, user-selectable active high or low)

There are four CLK/TRG pins, corresponding to the four independent CTC channels. In the Counter Mode every active edge on this pin decrements the Down Counter. In the Timer Mode, an active edge on this pin initiates the timing function. The user may select the active edge to be either rising or falling.

ZC/TO2-AC/TOO
Zero Count/Timeout (output, active high)

There are three ZC/TO pins, corresponding to CTC channels 2 through 0. (Due to package pin limitations channel 3 has no ZC/TO pin.) In either Counter Mode or Timer Mode, when the Down Counter decrements to zero an active high going pulse appears at this pin.

3.0 CTC PIN DESCRIPTION

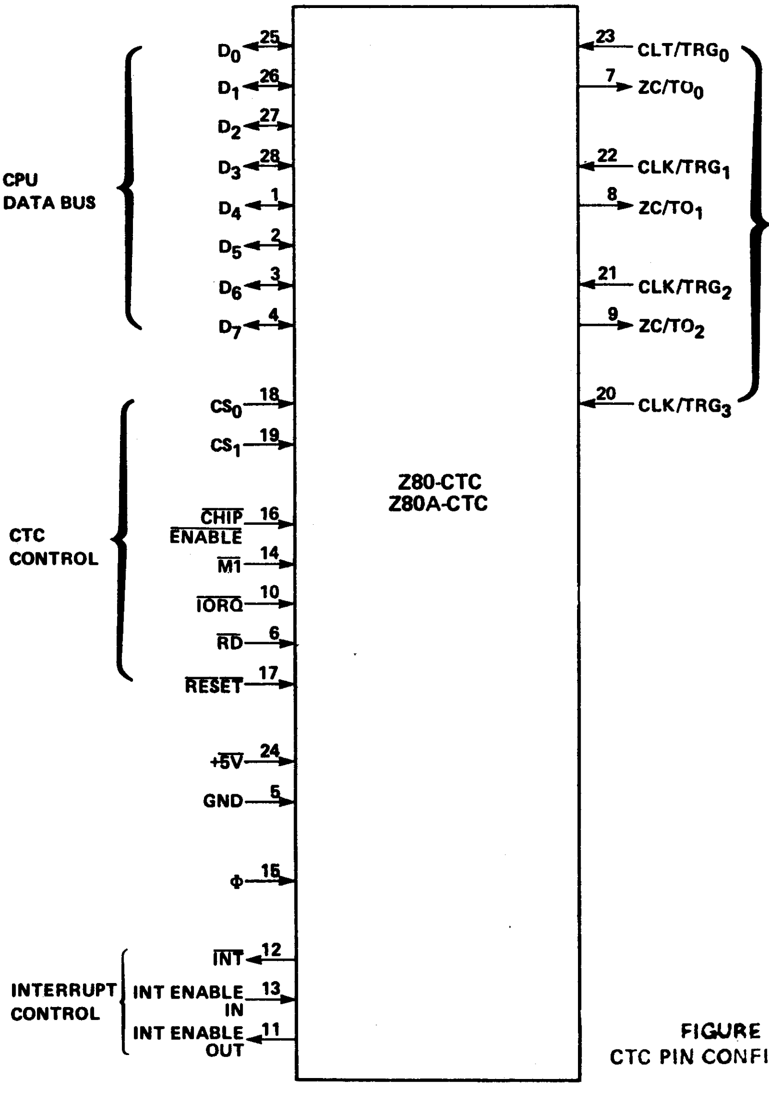


FIGURE 3.0-1
CTC PIN CONFIGURATION

4.0 CTC OPERATING MODES

At power-on, the Z80-CTC state is undefined. Asserting RESET puts the CTC in a known state. Before any channel can begin counting or timing, a Channel Control Word and a time constant data word must be written on the appropriate registers of that channel.

Further, if any channel has been programmed to enable int interrupts, an Interrupt Vector word must be written to the CTC's Interrupt Control Logic (For further details, refer to section 5.0 "CTC Programming"). When the CPU has written all of these words to the CTC all active channels will be programmed for immediate operation in either the Counter Mode or the Time Mode.

4.1 CTC COUNTER MODE

In this mode the CTC counts edges of the CLK/TRG input. The Counter Mode is programmed for a channel when its Channel Control Word is written with bit 6 set. The Channel's External Clock CLK/TRG) input is monitored for a series of triggering edges; after each edge, in synchronization with the next rising edge of the System Clock, the Down Counter (which was initialized with the time constant data word at the start of any sequence of down-counting) is decremented. Although there is no set-up time requirement between the triggering edge of the External Clock and the rising edge of the Clock, the Down Counter will not be decremented until the following pulse. (See the parameter ts (CK) in section 8.3: "A.C. Characteristics"). A channels's External Clock input is pre-programmed by bit 4 of the Channel Control Word to trigger the decrementing sequence with either a high or a low going edge.

In any of Channels 0, 1, or 2, when the Down Counter is successively decremented from the original time constant until it finally reaches zero, the Zero Count (ZC/TO) output pin for that channel will be pulsed active (high). However, due to package pin limitations channel 3 does not have this pin and so may only be used in applications where this output pulse is not required. Further, if the channel has been so pre-programmed by bit 7 of the Channel Control Word, an interrupt request sequence will be generated.

As the above sequence is proceeding, the zero count condition also results in the automatic reload of the Down Counter with the original time constant data word in the Time constant Register. There is no interruption in the sequence of continued down-counting. If the Time Constant Register is written on with a new time constant data word while the Down Counter is decrementing, the present count will be completed before the new time constant will be loaded into the Down Counter.

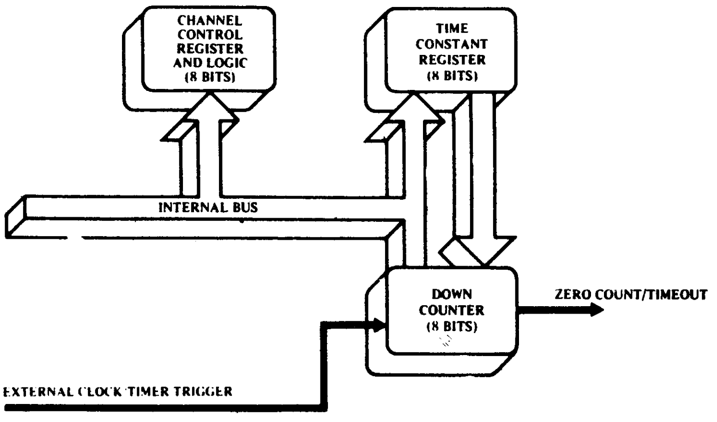


FIGURE 4.1-0
CHANNEL—COUNTER MODE

4.2 CTC TIMER MODE

In this mode the CTC generates timing intervals that are an integer value of the system clock period. The Time Mode is programmed for a channel when its Channel Control Word is written with bit 6 reset. The channel then may be used to measure intervals of time based on the System Clock Period. The System Clock is fed through two successive counters, the Prescaler and the Down Counter. Depending on the pre-programmed bit 5 in the Channel Control Word the Prescaler divides the System Clock by a factor of either 16 or 256. The output of the Prescaler is then used as a clock to decrement the Down Counter, which may be pre-programmed with any time constant integer between 1 and 256. As in the Counter Mode, the time constant is automatically reloaded into the Down Counter at each zero-count condition, and counting continues. Also at zero-count, the channel's Time Out (ZC/TO) output (which is the output of the Down Counter) is pulsed, resulting in a uniform pulse train of the precise period given by the product:

$$t_c * P * TC$$

where t is the System Clock period, P is the Prescaler factor of 16 or 256 and TC is the pre-programmed time constant.

Bit 3 of the Channel Control Word is pre-programmed to select whether timing will be automatically initiated, or whether it will be initiated with a triggering edge at the channel's Timer Trigger (CLK/TRC) input. If bit 3 is reset the timer automatically begins operation at the start of the CPU cycle following the I/O Write machine cycle that loads the time constant data word into the channel. If bit 3 is set the timer begins operation on the second succeeding rising edge after the Time Trigger edge following the loading of the time constant data word. If no time constant data word is to follow then the timer begins operation on the second succeeding rising edge of after the Time Trigger edge following the control word write cycle. Bit 4 of the Channel Control Word is pre-programmed to select whether the Timer Trigger will be sensitive to a rising or falling edge. Although there is no set-up requirement between the active edge of the Timer Trigger and the next rising edge. If the Timer Trigger edge occurs closer than a specified minimum set-up time to the rising edge, the Down Counter will not begin decrementing until the following rising edge.

If bit 7 in the Channel Control Word is set, the zero-count condition in the Down Counter, besides causing a pulse at the channel's Time Out pin, will be used to initiate an interrupt request sequence.

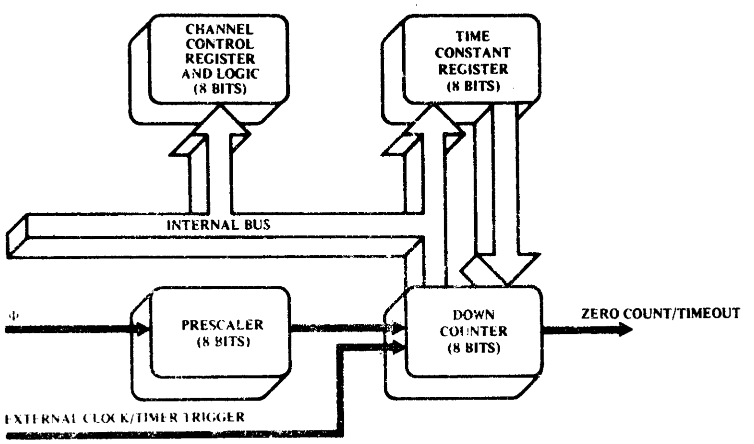


FIGURE 4.2-0
CHANNEL—TIMER MODE

5.0 CTC PROGRAMMING

Before a Z80-CTC channel can begin counting or timing operations, a Channel Control Word and a Time Constant data word must be written on it by the CPU. These words will be stored in the Channel Control Register and the Time Constant Register of that channel. In addition, if any of the four channels have been programmed with bit 7 of their Channel Control Words to enable interrupts, an Interrupt Vector must be written an the appropriate register in the CTC. Due to automatic features in the Interrupt Control Logic, one pre-programmed Interrupt Vector suffices for all four channels.

5.1 LOADING THE CHANNEL CONTROL REGISTER

To load a Channel Control Word, the CPU performs a normal I/O write sequence to the port address corresponding to the desired CTC channel. Two CTC input pins, namely CS0 and CS1, are used to form a 2-bit binary address to select one of four channels within the device (For a truth table, see section 2.2.1: "The Channel Control Register and Logic"). In many system architectures, these two input pins are connected to Address Bus lines A0 and A1 respectively, so that the four channels on a CTC device will occupy contiguous I/O port addresses. A word written on a CTC channel will be interpreted as a Channel Word and loaded into the Channel Control Control Register, its bit 0 is a logic 1. The other seven bits of this word select operating modes and conditions as indicated in the diagram below. Following the diagram the meaning of each bit will be discussed in detail.

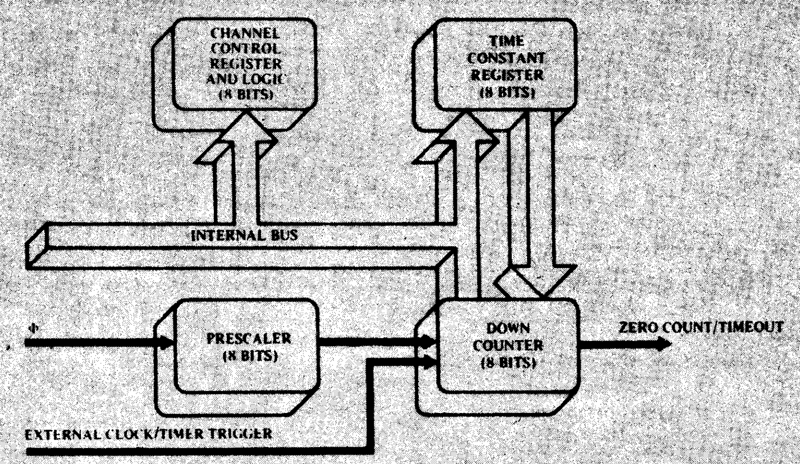
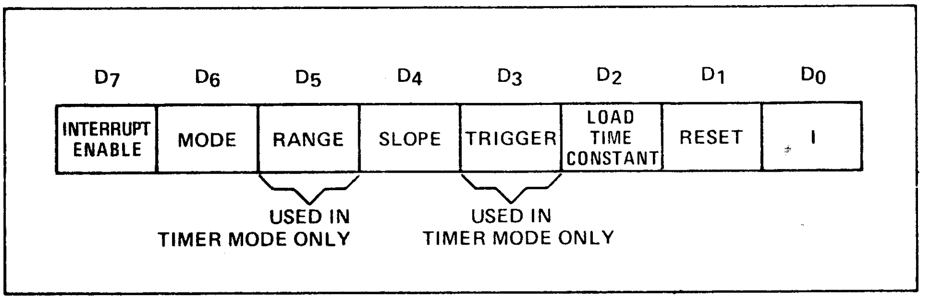


FIGURE 5.1-0
CHANNEL BLOCK DIAGRAM

5.1 LOADING THE CHANNEL CONTROL REGISTER (CONT'D)



Bit 7=1

The channel is enabled to generate an interrupt request sequence every time the Down Counter reaches a zero-count sondition. To set this bit to 1 in any of the four Channel Control Registers necessitates that an Interrupt Vector also be witten on the CTC before operation begins. Channel interrupts may be programmed in either Counter Mode or Timer Mode. If an updated Channel Control Word is written on a channel already in operation, with bit 7 set, the interrupt enable selection will not be retroactive to a preceding zero-count condition.

Bit 7=0

Channel interrupt disabled

Bit 6=1

Counter Mode selected. The Down Counter is decremented by each triggering edge of the External Clock (CLK/TRG) input. The Prescaler is not used.

Bit 6=0

Timer Mode selected. The Prescaler is clocked by the System Clock(Φ), and the outpt of the Prescaler in turn clocks the Down Counter. The output of the Down Counter (the channel's ZC/TO output) is a uniform pulse train of period given by the product:

$$t_c * P * TC$$

where t is the period of the System Clock, P is the Prescaler factor of 16 or 256, and TC is the time constant data word.

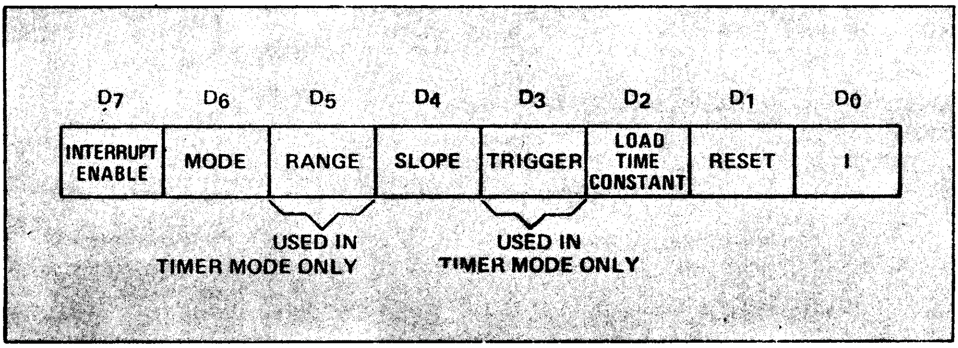
Bit 5=1

(Defined for Timer Mode only) Prescaler factor is 256.

Bit 5=0

(Defined for Timer Mode only) Prescaler factor is 16.

5.1 LOADING THE CHANNEL CONTROL REGISTER (CONT'D)



Bit 4=1

TIMER MODE -- positive edge trigger starts timer operation.
COUNTER MODE -- positive edge decrements the down counter.

Bit 4=0

TIMER MODE -- negative edge trigger starts timer operation.
COUNTER MODE -- negative edge decrements the down counter.

Bit 3=1

Timer Mode Only -- External trigger is valid for starting timer operation after rising edge T of the machine cycle following the one that loads the time constant. The Prescaler is decremented 2 clock cycles later if the setup time is met, otherwise 3 clock cycles later.

Bit 3=0

Timer Mode Only -- Timer begins operation on the rising edge T of the machine cycle following the one that loads the time constant.

Bit 2=1

The time constant data word for the Time Constant Register will be the next word written on this channel. If an updated Channel Control Word and time constant data word are written on a channel while it is already in operation, the Down Counter will continue decrementing to zero before the new time constant is loaded into it.

Bit 2=0

No time constant data word for the Time Constant Register should be expected to follow. To program bit 2 to this state implies that this Channel Control Word is intended to update the status of a channel already in operation, since a channel will not operate without a correctly programmed data word in the Time Constant Register, and a set bit 2 in this Channel Control Word provides the only way of writing to the Time Constant Register.

Bit 1=1

Reset channel. Channel stops counting or timing. This is not a stored condition. Upon writing into this bit a reset pulse discontinues current channel operation, however, none of the bits in the channel control register are changed. In both bit 2=1 and bit 1=1 the channel will resume operation upon loading a time constant.

Bit 1=0

Channel continues current operation.

5.2 LOADING THE TIME CONSTANT REGISTER

A channel may not begin operation in either Timer Mode or Counter Mode unless a time constant data word is written into the Time Constant Register by the CPU. This data word will be expected on the next I/O write to this channel following the I/O Write of the Channel Control Word, provided that bit 2 of the Channel Control Word is set. The time constant data word may be any integer value in the range 1-256. If all eight bits in his word are zero, it is interpreted as 256. If a time constant data word is loaded into a channel already in operation the Down Counter will continue decrementing to zero before the new time constant is loaded from the Time Constant Register in to the Down Counter.

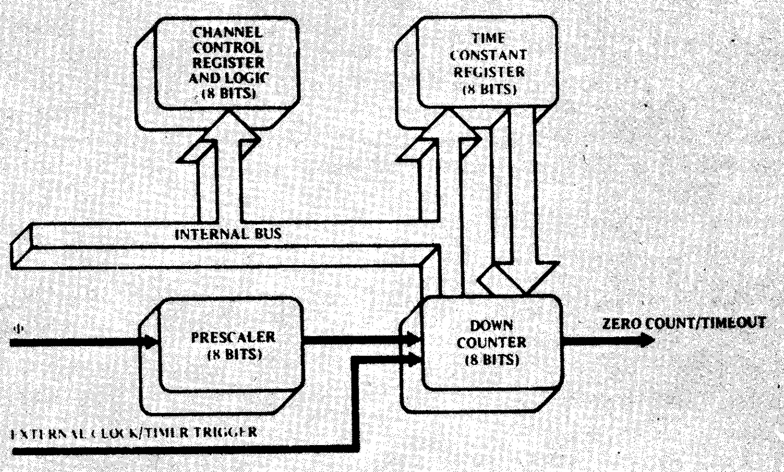
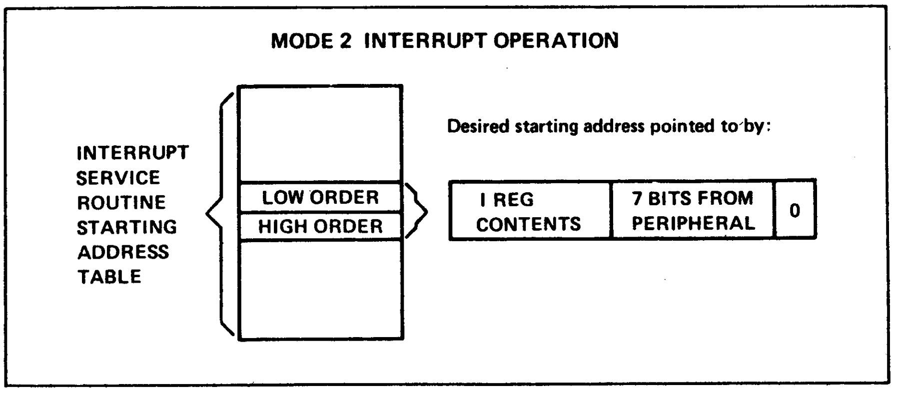


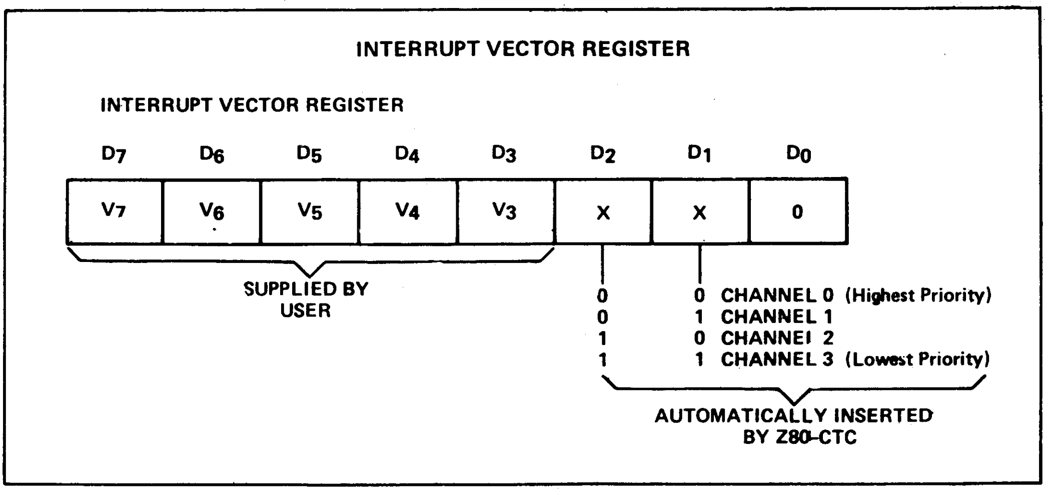
FIGURE 5.2-0
CHANNEL BLOCK DIAGRAM

5.3 LOADING THE INTERRUPT VECTOR REGISTER

The Z80-CTC has been designed to operate with the Z80-CPU programmed for mode 2 interupt response. Under the requirements of this mode, when a CTC channel requests an interrupt and is acknowledged, a 16-bit pointer must be formed to obtain a corresponding interrupt service routine starting address from a table in memory. The upper 8 bits of this pointer are provided by the CPU's I register and the lower 8 bits of the pointer are provided by the CTC in the form of an Interrupt Vector unique to the particular channel that requsted the interrupt.



The high order 5 bits of this Interrupt Vector must be written to the CTC in advance as part of the initial programming sequence. To do so, the CPU must write to the I/O port address corresponding to the CTC channel 0, just as it would if a Channel Control Word were being written to that channel, except that bit 0 of the word being written must contain a 0 (As explained above in section 5.1, if bit 0 of a word written to a channel were set to 1, the word would be interpreted as a Channel Control Word, so a 0 in bit 0 signals the CTC to load the incoming word into the Interrupt Vector Register). Bits 1 and 2, however, are not used when loading this vector. At the time when the interrupting channel must place the Interrupt Vector on the Z80 Data Bus, the Interrupt Control Logic of the CTC automatically supplies a binary code in bit 1 and 2 identifying which of the four CTC channels is to be serviced.



6.0 To see the program as follows you will have the whole ieda about CTC programming.

```
START    LD     A,18H
         LD     I,A
         LD     A,10110101B
         OUT    (CTC0),A
         LD     A,020H
         OUT    (CTC0),A
         LD     A,0A8H
         OUT    (CTC0),A
         IM     2
         EI
```

In MPF-I the four port addresses of the CTC are 40, 41, 42, 43 respectively. Since the contents of the Channel Control Register is "10110101B", So CH0 is programmed to be in the timer mode. Since bit 5 in the Channel Control word is set the Prescaler divides the System Clock by a factor of 256. Since the content of Time Constant Register is 020H, CTC channel 0 will request a interrupt every time its Down Counter reaches a count of zero. In other words, CTC will generate an interrupt when the count of the System Clock reaches 8192 (i.e. 256 x 32). Since RESET forces the program counter to zero and initializes the CPU. The CPU initialization includes:

1. Disabling the interrupt enable flip-flop
2. Setting Register I=00H

II. The flowchart of the clock are given below.

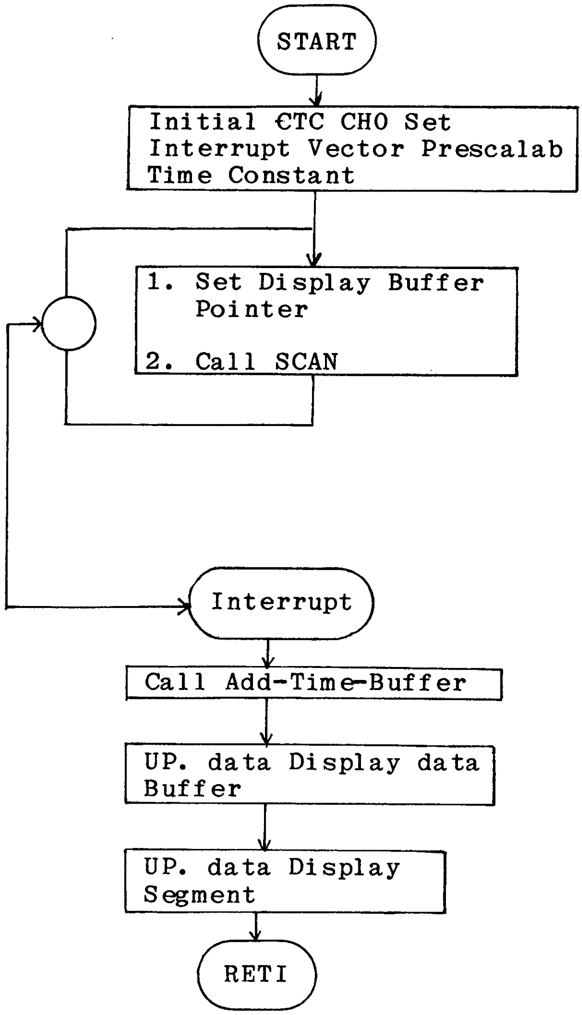


Fig 15-3 The flowchart of clock

```
LOC    OBJ CODE M STMT SOURCE STATEMENT

                        ;
1800                              ORG       1800H
                        ;
                        CTC0      EQU       40H
                        SCAN      EQU       05FEH
                        START:
1800   3E18                       LD        A,18H    ;Loading the interrupt register
1802   ED47                       LD        I,A
1804   3EB5                       LD        A,10110101B      ;Loading the channel
                                                              control
1806   D340                       OUT       (CTC0),A
1808   3E20                       LD        A,020H   ;Loading the constant register
180A   D340                       OUT       (CTC0),A
180C   3EA8                       LD        A,0A8H   ;Loading the interrupt vector
                                                      register
180E   D340                       OUT       (CTC0),A
1810   ED5E                       IM        2        ;Set interrupt mode 2
1812   FB                         EI
                        MAIN:
1813   DD21041A                   LD        IX,DISP_BUFFER
1817   CDFE05                     CALL      SCAN
181A   18F7                       JR        MAIN
                        ; ********************************************
                        ADD_TIME_BUFFER:
181C   11001A                     LD        DE,TIME_BUFFER
181F   1A                         LD        A,(DE)
1820   3C                         INC       A
1821   12                         LD        (DE),A
1822   FEDA                       CP        0DAH     ;Increment SEC only if the
1824   0604                       LD        B,4      ;number of interrupt reaches
1826   C0                         RET       NZ       ;218 (ie 0DAH).
1827   AF                         XOR       A
1828   05                         DEC       B
1829   12                         LD        (DE),A
182A   13                         INC       DE
182B   215318                     LD        HL,MAX_TIME_TABLE
                        ATB1:
182E   1A                         LD        A,(DE)
182F   C601                       ADD       A,1
1831   27                         DAA
1832   12                         LD        (DE),A
1833   96                         SUB       (HL)     ;Comparea with data in
                                                      MAX_TIME_TABLE
1834   D8                         RET       C
1835   12                         LD        (DE),A
1836   23                         INC       HL       ;If the result is less that, t
1837   13                         INC       DE
1838   10F4                       DJNZ      ATB1     ;following loop will be null.
183A   C9                         RET
                        SET_DISP_BUFFER:
183B   21041A                     LD        HL,DISP_BUFFER   ;Convert data in
                                                              display buffer
183E   11011A                     LD        DE,SECOND        ;to display format.
1841   0603                       LD        B,3
```

```
                        51   SDB1
1843    1A              52           LD      A,(DE)    ;
1844    CD7806          53           CALL    HEX7SG
1847    13              54           INC     DE
1848    10F9            55           DJNZ    SDB1
184A    2B              56           DEC     HL
184B    2B              57           DEC     HL
184C    CBF6            58           SET     6,(HL)    ;Set decimal point for hour
```

```
LOC    OBJ CODE M STMT SOURCE STATEMENT

184E   2B             59           DEC     HL
184F   2B             60           DEC     HL
1850   CBF6           61           SET     6,(HL)
1852   C9             62           RET
                      63   ; ********************************************
                      64   MAX_TIME_TABLE:
1853   60             65           DEFB    60H      ;The maximal value of the
                                                     time constant
1854   60             66           DEFB    60H      ;e.g. the maximum of second
                                                     is 60,
1855   12             67           DEFB    12H      ;the maximum of hour is 12.
                                                     (The use may change
18A8                  68           ORG     18A8H    ;12 to 24 as he wished)
18A8   AA18           69           DEFW    INTERRUPT
                      70   INTERRUPT:               ;Entry point of interrpt
                                                     service
18AA   F5             71           PUSH    AF       ;routine.
18AB   C5             72           PUSH    BC
18AC   D5             73           PUSH    DE
18AD   E5             74           PUSH    HL
18AE   CD1C18         75           CALL    ADD_TIME_BUFFER
18B1   78             76           LD      A,B
18B2   FE04           77           CP      4
18B4   C43B18         78           CALL    NZ,SET_DISP_BUFFER
18B7   E1             79           POP     HL
18B8   D1             80           POP     DE
18B9   C1             81           POP     BC
18BA   F1             82           POP     AF
18BB   FB             83           EI
18BC   ED4D           84           RETI
                      85   HEX7SG  EQU     678H
1A00                  86           ORG     1A00H
                      87   TIME_BUFFER:
1A00   00             88           DEFB    0
                      89   SECOND
1A01                  90           DEFS    1        ;Locations for pressetting
                                                     values.
                      91   MINUTE
1A02                  92           DEFS    1
                      93   HOUR
1A03                  94           DEFS    1
                      95   DISP_BUFFER:
1A04                  96           DEFS    6
```

III. Illustration of Experiments 3-15

1. The timer mode is used in this experiment. This program carefully calculates the total number of counts of System Clock. The frequency of the System Clock is 1.7898 MHZ. In this experiment we use 1785856 (256 x 32 x 218) count for each SECOND, so the count error per second is 1789772-1785856=3916.
The SECOND error is $(1 \div 1798772) \times 3916 = 2.2$ msec
Hence there is 1 second error for every 455 seconds.

2. This program uses the Z80 CPU interrupt mode 2. The contents of the Interrupt Register are 018H and the content of the Interrupt vector Register are 0A8H, then the contents of the Interrupt Service Routine's starting address is stored in addresses 18A8 - 18A9.
We can see that the Interrupt Service Routine's starting address in this expriment is 18AAH.

3. The statements 6-15 set the CTC with control words. Now in this expimernt we use the CTC Timer Mode and the prescaler is set to 256. The contents of the Time Constant Register are 20H. The interrupt service routine's starting address is 18AAH. Statements 21-32 check whether the count of interrupts reaches 218 or not. Statements 33-45 compares data with MAX-TIME-TABLE. Statements 46-61 convert data in the display buffer to a 7-segment diaplay format. Statements 64-67 set the decimal points for both hour and minute.

4. Load the program into MPF-I and record it on audio tape for future use.

5. Convert the contents of 1823 to 6D. What will the display show ?

6. If we want to use CH2 of the CTC what shall do ?

7. If we change the contents of the MAX-TIME-TABLE, what will the display show?

8. Typically, there are 1789772 T-cycles in one second. This program approximates one second with 1785856 T-cycles, it is pretty rough. So if a user needs more precise timing, Software compensation is needed.

Experiment 16

Telephone Tone

Purposes:

1. To simulate a telephone ring.
2. To familiarize the reader with the application of 'tone' subroutine.

Time requirod: 4 hours.

I. Theoretical Background:

1. The telephone ring can be simulated as a repeating 1 second tone with 2 seconds silence.
2. This tone is a frequency shift keying signal modulated by two 20HZ square waves (half-period of 25 m sec).
   The low & high states of this 20HZ signal correspond to 320HZ and 480HZ, so that it takes 8 & 12 cycles respectively.
3. In the following program, register C controls the frequency of the sound and register pair HL controls the length of the sound.

   a. Low frequency: C = 211, HL = 8, so the period is

      (44 + 13 x 211) x 2 x 0.56 = 3121 micro-sec.

      frequency : f = 1/3121 = 320Hz

      length of the sound: 3121 micro-sec x 8 = 25m sec.

   b. High frequency: C = 140, HL = 12, so the period is

      (44 + 13 x 140) x 2 x 0.56 = 2087 micro-sec

      frequency: 1/2087 = 480HZ.

      length of the sound: 2087 micro-sec x 2 = 25m sec.

4. Output Circuit of tone

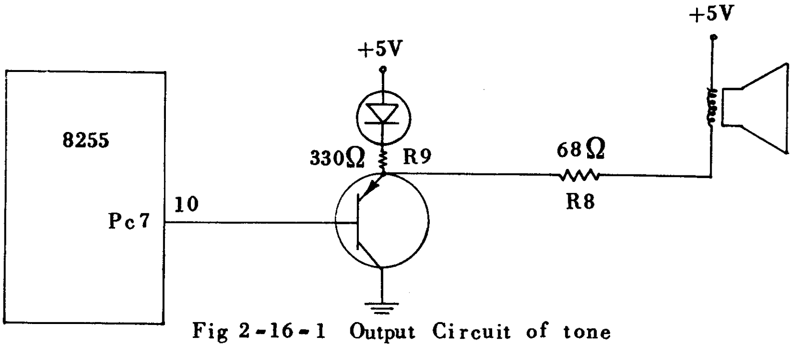


Fig 2-16-1 Output Circuit of tone

The output of the tone is sent via PC7 of 8255, 2N9015, R8, to the speaker. When the voltage of PC7 is low, the transistor will conduct; the volotage of PC7 is high, the transistor will nonconduct. By means of the transistor conducts and nonconducts, the speaker will make sound.

5. Flowchart of Telephone Tone

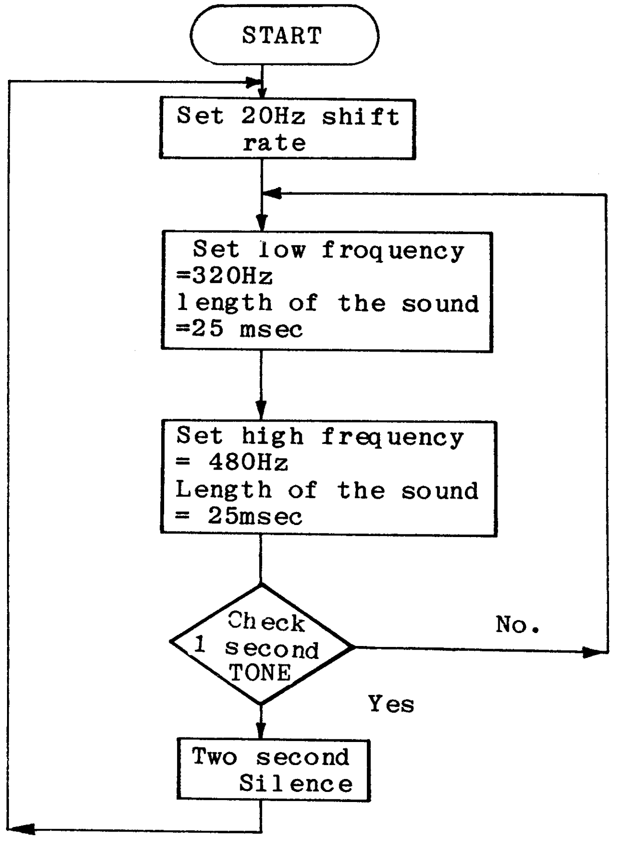


Fig 16-2 Flowchart of a telephone tone simalation

6. Telephone Tone Program

```
LOC   OBJ CODE M STMT SOURCE STATEMENT

1800                              ORG     1800H
1800  3E14              RINGBK    LD      A,20        ;20HZ FREQ SHIFT RATE
                                                      ;SO THAT 1 SEC HAS 20 LOOPS
1802  08                RING      EX      AF,AF'      ;SAVE TO A'
1803  0ED3                        LD      C,211
1805  210800                      LD      HL,8
1808  CDE405                      CALL    TONE        ;320HZ, 25 MSEC
180B  0E8C                        LD      C,140
180D  210C00                      LD      HL,12
1810  CDE405                      CALL    TONE        ;480HZ, 25 MSEC
1813  08                          EX      AF,AF'      ;RETRIEVE FROM A'
1814  3D                          DEC     A           ;DECREMENT 1 COUNT
1815  20EB                        JR      NZ,RING
                        ;
1817  0150C3                      LD      BC,50000
181A  CD1F18                      CALL    DELAY       ;SILENT, 2 SEC
181D  18E1                        JR      RINGBK
                        ;DELAY SUBROUTINE: (BC) * 40 MICRO-SEC
                        ;BASED ON THE 1. 79 MHZ SYSTEM CLOCK
181F  E3                DELAY     EX      (SP),HL     ;19 STATES
1820  E3                          EX      (SP),HL     ;19
1821  EDA1                        CPI                 ;16
1823  E0                          RET     PO          ;5
1824  18F9                        JR      DELAY       ;12
                        ;
                        ;
                        TONE      EQU     05E4H
                                  END
```

II. Example and Practice Experiments

1. Load the above program into MPF-I and then store it on audio tape.

2. Execute the program and listen to it. Does it like the telephone ring? If it doesn't try to modify the frequency of tone to closer simulate the sound.

3. Try to simulate the telephone busy tone

   Hint: The busy tone can be simulated as follows: a repeating 0.5 seccond 400HZ tone with 0.5 seconds of silence.

Experiment 17

Microcomputer Organ

Purposes:

1. To enable the part of the Microprofessor to simulate an electronic organ.

2. To familiarize the reader with the application of the keyboard -scaning routine.

Time Rquired: 4 hours

I. Theoretical Background:

1. This experiment converts the MPF-I into a simple electronic organ.

2. When a key is pressed, the speaker will generate a tone corresponding this key. This tone will not terminate until the key is released.

3. Acceptable keyboard: key 0 - key F.

   If other keys are entered, the response is unpredictable.

4. Key Mapping To Tones

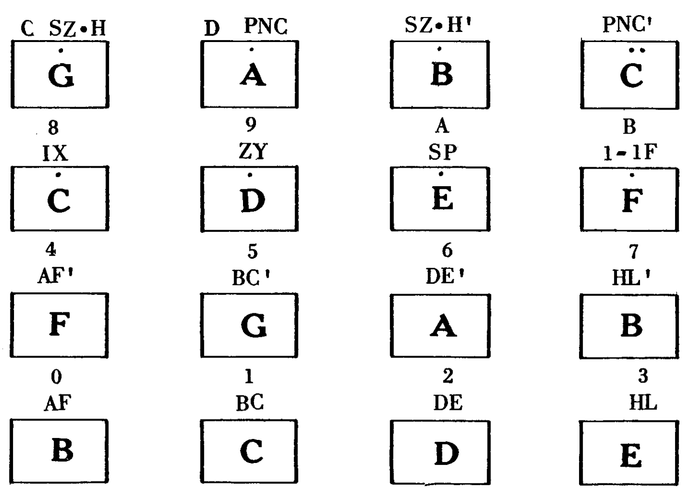


Fig 17·1 Key Mapping To Tone

5. An octave ranges from a C to a B. The cotave is divided into 5 full-tone and 2 half-tones, which equals to 12 half-tones, as follows:

   C #C D #D E F #F G #G A #A B

The next octave is just twice the frequency of the current one, There is a lograrithmic relationship between each half-tone. The frequency of each half-tone can be calculated by multiplying the last one by 2 ** (1/12), which is approximately 1.059.

For example, if the frequency of E is 503HZ, then the frequency of F is equal to

503HZ x 1.059 = 532HZ.

6. Flow chart of microcomputer organ program

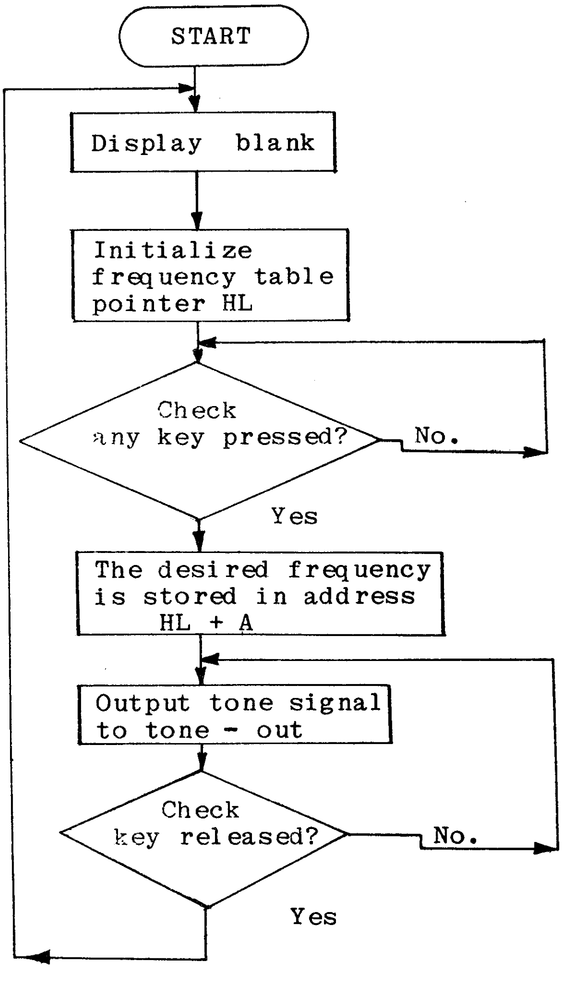


Fig 17-2 Flowchart of orgran

```
LOC    OBJ CODE M STMT SOURCE STATEMENT

1800                                  ORG     1800H
                        START:
1800   DD21A507                       LD      IX,BLANK
1804   CDFE05                         CALL    SCAN      ;Display blank, return when
                                                         any
                                                        ;Key is pressed.  A register
                                                        ;Contains the key-code.
1807   212318                         LD      HL,FREQTAB ;Base address of freque
                                                             -ncy table.

                        ;After routine SCAN, A contains the code of the key
                         pressed.
                        ;Use this code as table offset.  The desired f
                        ;frequency is stored in address HL+A.

180A   85                             ADD     A,L       ;Add A to HL.
180B   6F                             LD      L,A
180C   3EC0                           LD      A,11000000B

                        HALF_PERIOD:
180E   D302                           OUT     (DIGIT),A ;Output tone signal to
                                                          TONE-OUT.
                                                        ;Activate all 6 columns of
                                                        ;the Keyboard matrix.
1810   46                             LD      B, (HL)   ;Get the frequency from
                                                         FREQTAB.
                                                        ;HL has been calculated in
                                                        ;previous instructions.
1811   00               DELAY:        NOP
1812   00                             NOP
1813   00                             NOP
1814   10FB                           DJNZ    DELAY     ;Loop B times.
1816   EE80                           XOR     80H       ;Complement bit 7 of A.
                                                        ;This bit will be output to
                                                         TONE.
1818   4F                             LD      C,A       ;Store A in C
1819   DB00                           IN      A,(KIN)   ;Check if this key is released.
                                                        ;All 6 columns have been
                                                         activated.
                                                        ;If any key is pressed, the
                                                        ;corres-ponding matrix row
                                                        ;input must be at low.
181B   F6C0                           OR      11000000B ;Mask out bit 6 (tape input)
                                                        ;and bit 7 (User's K) of
                                                         register  A.
181D   3C                             INC     A         ;If A is 11111111, increase
                                                        ;A by one will make A zero
                                                        ;Zero flag is changed here.
181E   79                             LD      A,C       ;Restore A from register C.
181F   28DF                           JR      Z,START   ;If all keys are released,
                                                         re-start.
                                                        ;Otherwise, continue this
                                                         frequency.
1821   18EB                           JR      HALF_PERIOD

                        FREQTAB:
```

```
1823   B2          47          DEFB    0B2H     ;Key 0
1824   A8          48          DEFB    0A8H     ;Key 1
1825   96          49          DEFB    096H     ;Key 2
1826   85          50          DEFB    085H     ;Key 3
1827   7E          51          DEFB    07EH     ;Key 4
1828   70          52          DEFB    070H     ;Key 5
1829   64          53          DEFB    064H     ;Key 6
182A   59          54          DEFB    059H     ;Key 7
182B   54          55          DEFB    054H     ;Key 8
182C   4A          56          DEFB    04AH     ;Key 9
182D   42          57          DEFB    042H     ;Key A
182E   3E          58          DEFB    03EH     ;key B


 LOC   OBJ CODE M STMT SOURCE STATEMENT

182F   37          59          DEFB    037H     ;Key C
1830   31          60          DEFB    031H     ;Key D
1831   2C          61          DEFB    02CH     ;Key E
1832   29          62          DEFB    029H     ;Key F
                   63
                   64  BLANK   EQU     07A5H
                   65  SCAN    EQU     05FEH
                   66  DIGIT   EQU     2
                   67  KIN     EQU     0
                   68          END
```

II. Example and Practice Experiments

1. Load the above program into MPF-I and then store it on audio tape.

2. Execute the program. When a key is pressed, the speaker will generate a tone corresponding to this key. Acceptable keys are key 0 to key F.

   Are these tones accurate?

3. Try to play a song using organ.

4. Extend this program so that more keys of the key board can be used as input keys of the organ.

Experiment 18

Music Box

Purposes:

1. To construct a music box.

2. To familiarize the reader with programming techniques.

Time Required: 4 hours.

I. Theoretical Background:

1. This experiment generates a song using programming techniques.

2. There are two tables (frequency-table & song-table) in this program, which is described below:

   a. Frequency-table

      Every element of this table has 2 bytes, the 1st byte is the frequency parameter and the 2nd byte is the number of half-periods in a unit-time duration.

      One octave ranges from C to B. It is divided into 5 full-tones and 2 half-tones, which equals 12 half-tones, as follows:

      C #C D #D E F #F G #G A #A B

      The next octave is just twice the frequency of the current one, and there is a logarithmic relationship between each half-tone. So that the frequency of each half-tone can be calculated by multiplying the last tone by 2 ** 1/12, which is approximately 1.059.

   b. Song-Table:

      Each element of this table has 2 bytes:

      The 1st byte contains the code of the NOTE or REST or command of REPEAT or STOP. These codes are:

      ```
      bit 7 ---- STOP
      bit 6 ---- REPEAT
      bit 5 ---- REST
      bit 4-0 ---- NOTE CODE
      ```

      The 2nd byte contains the counts of the unit-time, i.e. the NOTE length.

3. A flowchart of music box simulation is given below:

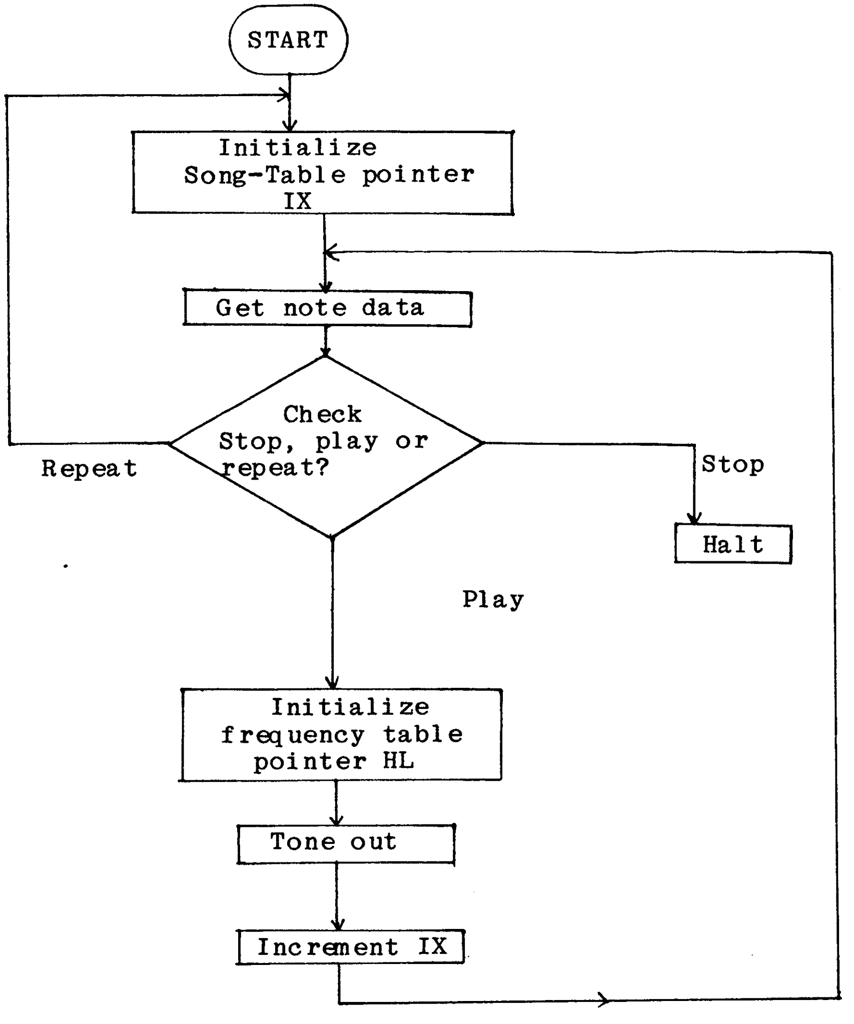


Fig 18-1 Flowchart of music box simulation

```
                                                          PAGE    1
LOC    OBJ CODE M STMT SOURCE STATEMENT                    ASM 5.8

1800                                  ORG     1800H
                        ;
1800   DD218018         START   LD      IX,SONG    ;Initial SONG-TABLE pointer
1804   DD7E00           FETCH   LD      A,(IX)     ;Get note data
1807   87                       ADD     A,A        ;Each note data have 2 bytes
1808   3830                     JR      C,STOP     ;STOP?
180A   FA0018                   JP      M,START    ;REPEAT?
180D   0E00                     LD      C,0        ;Reset TONE-BIT (BIT-7 of C)
180F   CB77                     BIT     6,A        ;REST?
1811   2002                     JR      NZ,PLAY
1813   CBF9                     SET     7,C        ;Set ONTE-BIT
1815   E63F             PLAY    AND     3FΗ        ;Mask out note data
1817   213B18                   LD      HL,FRQTAB
181A   85                       ADD     A,L
181B   6F                       LD      L,A        ;Locate pointer in FRQTAB
181C   5E                       LD      E,(HL)     ;Counts of loop per HALF-PERIOD
                                                    delay

181D   23                       INC     HL
181E   56                       LD      D,(HL)     ;Counts of HALF-PERIODS per UNIT-
                                                    TIME
181F   DD23                     INC     IX
1821   DD6600                   LD      H,(IX)     ;Counts of UNIT-TIME for this note
1824   3EFF                     LD      A,0FFH
                        ;
                        ;The following loop runs for one NOTE or REST:
                        ;
                        TONE:
1826   6A                       LD      L,D
1827   D302             UNIT    OUT     (02H),A    ;Bit 7 is NOTE-OUT
1829   43                       LD      B,E
182A   00               DELAY   NOP                ;delay loop B*25-5 states
182B   00                       NOP
182C   00                       NOP
182D   10FB                     DJNZ    DELAY
182F   A9                       XOR     C          ;If C=80H then TONE-OUT.
                                                   ;If C=00H then REST.
1830   2D                       DEC     L
1831   20F4                     JR      NZ,UNIT
1833   25                       DEC     H
1834   20F0                     JR      NZ,TONE
                        ;
                        ;The current note has ended, increment pointer next.
                        ;
1836   DD23                     INC     IX
1838   18CA                     JR      FETCH
                        ;
183A   76               STOP    HALT
                        ;
```

```
                         49  FRQTAB:
                         50
                         51  ;
                         52  ;1st byte: counts of delay loop per HALF-PERIOD.
                         53  ;2nd byte: counts of HALF-PERIOD per UNIT-TIME.
                         54  ;
                         55  ;OCTAVE 3.
183B    E118             56          DEFW    18E1H    ;CODE 00 , G
183D    D41A             57          DEFW    1AD4H    ;CODE 01 , #G
183F    C81B             58          DEFW    1BC8H    ;CODE 02 , A


                                  PO                              PAGE   2
 LOC    OBJ CODE M STMT SOURCE STATEMENT                          ASM 5.8

1841    BD1D             59          DEFW    1DBDH    ;CODE 03 , #A
1843    B21E             60          DEFW    1EB2H    ;CODE 04 , B
                         61  ;OCTAVE 4
1845    A820             62          DEFW    20A8H    ;CODE 05 , C
1847    9F22             63          DEFW    229FH    ;CODE 06 , #C
1849    9624             64          DEFW    2496H    ;CODE 07 , D
184B    8D26             65          DEFW    268DH    ;CODE 08 , #D
184D    8529             66          DEFW    2985H    ;CODE 09 , E
184F    7E2B             67          DEFW    2B7EH    ;CODE 0A , F
1851    772E             68          DEFW    2E77H    ;CODE 0B ,#F
1853    7031             69          DEFW    3170H    ;CODE 0C ,G
1855    6A33             70          DEFW    336AH    ;CODE 0D ,#G
1857    6437             71          DEFW    3764H    ;CODE 0E , A
1859    5E3A             72          DEFW    3A5EH    ;CODE 0F , #A
185B    593D             73          DEFW    3D59H    ;CODE 10 , B
                         74  ;OCTAVE 5
185D    5441             75          DEFW    4154H    ;CODE 11 , C
185F    4F45             76          DEFW    454FH    ;CODE 12 , #C
1861    4A49             77          DEFW    494AH    ;CODE 13 , D
1863    464D             78          DEFW    4D46H    ;CODE 14 , #D
1865    4252             79          DEFW    5242H    ;CODE 15 , E
1867    3E57             80          DEFW    573EH    ;CODE 16 , F
1869    3B5C             81          DEFW    5C3BH    ;CODE 17 , #F
186B    3762             82          DEFW    6237H    ;CODE 18 , G
186D    3467             83          DEFW    6734H   ;CODE 19 , #G
186F    316E             84          DEFW    6E31H   ;CODE 20 , A
1871    2E74             85          DEFW    742EH   ;CODE 21 , #A
1873    2C7B             86          DEFW    7B2CH   ;CODE 1C , B
                         87  ;OCTAVE 6
1875    2982             88          DEFW    8229H   ;CODE 1D , C
1877    278A             89          DEFW    8A27H   ;CODE 1E , #C
1879    2592             90          DEFW    9225H   ;CODE 1F , D
```

```
                    91   ;
                    92   ;
                    93   ;1st byte, bit 7,6,5 & 4-0 STOP, REPEAT, REST & NOTE
                    94   ;       Code of STOP:      80H
                    95   ;       Code of REPEAT:    40H
                    96   ;       Code of REST:      20H
                    97   ;2nd byte, NOTE LENGTH: counts of UNTI-TIME (N*0.077 sec)
                    98   ;
                    99   ;JINGLE BELL: (Truncated)
1880               100   SONG    ORG      1880H
1880    09         101           DEFB     9
1881    04         102           DEFB     4
1882    09         103           DEFB     9
1883    04         104           DEFB     4
1884    09         105           DEFB     9
1885    06         106           DEFB     6
1886    20         107           DEFB     20H       ;REST
1887    02         108           DEFB     2
1888    09         109           DEFB     9
1889    04         110           DEFB     4
188A    09         111           DEFB     9
188B    04         112           DEFB     4
188C    09         113           DEFB     9
188D    06         114           DEFB     6
188E    20         115           DEFB     20H       ;REST
188F    02         116           DEFB     2


                                        PO
 LOC    OBJ CODE  M STMT SOURCE STATEMENT

1890    09         117           DEFB     9
1891    04         118           DEFB     4
1892    0C         119           DEFB     0CH
1893    04         120           DEFB     4
1894    05         121           DEFB     5
1895    04         122           DEFB     4
1896    07         123           DEFB     7
1897    04         124           DEFB     4
1898    09         125           DEFB     9
1899    08         126           DEFB     8
189A    20         127           DEFB     20H       ;REST
189B    08         128           DEFB     8
189C    80         129           DEFB     80H       ;STOP
```

```
130  ;
131  ;The following data are codes of the song 'GREEN SLEEVES'.
132  ;The user can put them at the SONG-table, i.e. from 1880H.
133  ;It will play until 'RS' key is pressed.
134  ;
135  ;
136  ;1880   07 08 0A 10 0C 08 0E 10   10 04 0E 04   0C 10 09 08
137  ;1890   05 10 07 04 09 04 0A 10   07 08 07 10   06 04 07 04
138  ;18A0   09 10 06 08 02 10 07 08   0A 10 0C 08   0E 10 10 04
139  ;18B0   0E 04 0C 10 09 08 05 10   07 04 09 04   0A 08 09 08
140  ;
141  ;18C0   07 08 06 08 04 08 06 08   07 10 20 08   11 10 11 08
142  ;18D0   11 10 10 04 0E 04 0C 10   09 08 05 10   07 04 09 04
143  ;18E0   0A 10 07 08 07 10 06 04   07 04 09 10   06 08 02 10
144  ;18F0   20 08 11 10 11 08 11 10   10 04 0E 04   0C 10 09 08
145  ;
146  ;1900   05 10 07 04 09 04 0A 08   09 08 07 08   06 08 04 08
147  ;1910   06 08 07 18 20 10 40
148  ;
149  ;
150  ;The ending address is 1916H.
151  ;
152
153
154
```

II. Example and practice Experiments:

1. Load the above program into MPF-I and them store it on audio tape.
2. Execute the program beginning at line 100 and listen to it. Does it sound like the song "JINGLE BELLS"?
3. On the last page of the above program line 30 there are codes for the song "GREEN SLEEVES". Put them on the SONG-table. The program will play until "RS" key is pressed.
4. Try to translate your favorite song into code and load it into MPF-I.

Micro-Professor Application Note

—DOC. NO. MPF-I-01-210A —

MPF-I AS A FREQUENCY COUNTER

An Application Example of Z8Ø-CTC.

MULTITECH INDUSTRIAL CORPORATION

OFFICE : 977, MIN SHEN E. ROAD, TAIPEI, 105, TAIWAN, R.O.C.
TEL. (02)769-1225 (10 LINES) TELEX: 23756 MULTIIC
FACTORY: 5, TECHNOLOGY ROAD III
HSINCHU SCIENCE-BASED INDUSTRIAL PARK,
HSINCHU, TAIWAN, 300, R.O.C.
TEL: (035)775102 (3 LINES)

US$1.00

Purpose: Use CTC to design a frequency counter

Required Equipment: MPF-1 (included CTC)

Expriment Explanation:

1. CTC has four channels CHØ - CH3 is mapping to 4ØH,41H, 42H,43H. In this program, we use CHØ & CH1. The function is shown below.

   CH1: used for timer interrupt, triggered by the internal clock of MPF-1 (1789772 Hz), we set CHØ to
   Mode: timer
   Range: 256
   Time constant: 233 (ØE9H)
   So after interrupt 3Ø (Ø1EH) times. It will be approximately 1 sec, 256*233*3Ø=ØFFH*ØE9H*Ø1EH= 178944Ø. It has error (1789772-178944Ø)/1789772= Ø.ØØ185%

   CHØ: used for counter interrupt, triggered by user signal
   Mode: counter
   And set 'down counter'=1ØØ. Each time interrupts happen, we can add frequency counter by 1. We can get signal frequency when one sec is up.

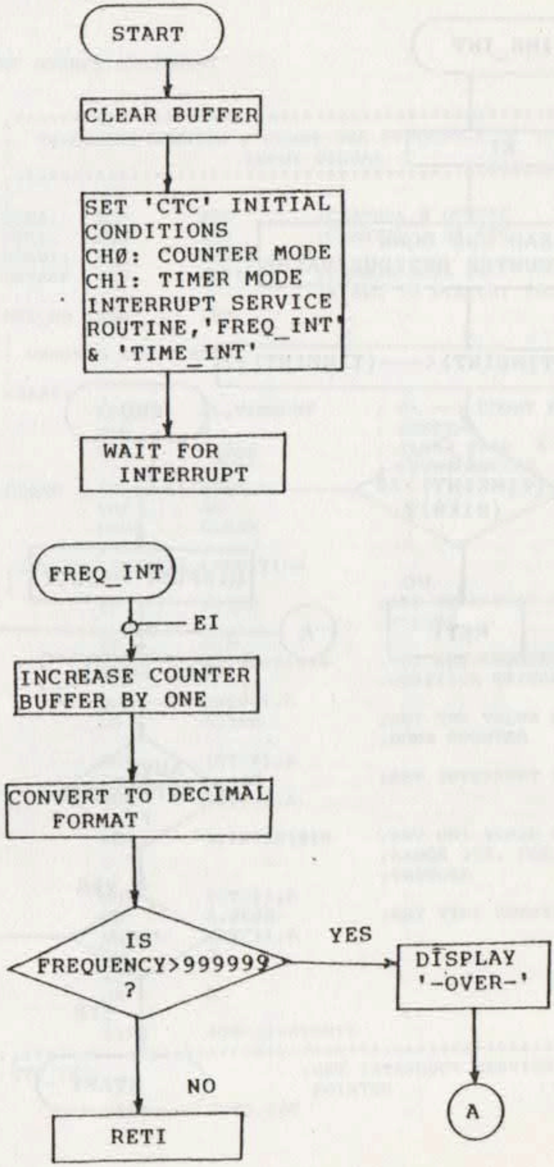
2. LED display is in decimal format (six digit). So the largest value is 999999. When frequency is over this it will get some warning message, eg., display 'over' message,and tone 2K

3. Special care should be exerted when using the CTC. To avoid burning out the CTC, you should first refer to the Z8Ø Handbook, Z8Ø-CTC Technical Manual, section 8.1 D.C. Characteristics. User signal used to trigger the CTC should comply with the following characteristics:

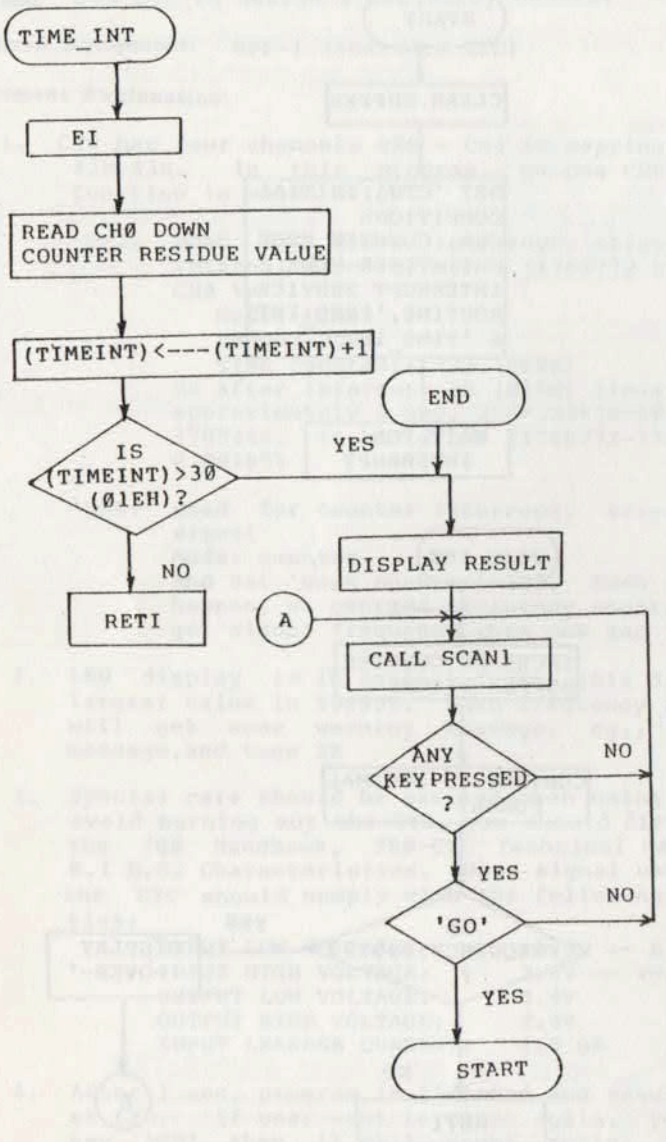
INPUT LOW VOLTAGE:	-Ø.3V -- Ø.8V
INPUT HIGH VOLTAGE:	2.ØV -- Vcc.
OUTPUT LOW VOLTAGE:	Ø.4V
OUTPUT HIGH VOLTAGE:	2.4V
INPUT LEAKAGE CURRENT:	1.Ø uA

4. After 1 sec, program is finished and result will display at LED. If user want to count again, you can pressed key 'GO' then it will count again. Othersize, it will continue to display data.

Flowchart:



START
CLEAR BUFFER
SET 'CTC' INITIAL CONDITIONS
CHØ: COUNTER MODE
CH1: TIMER MODE
INTERRUPT SERVICE ROUTINE,'FREQ_INT' & 'TIME_INT'
WAIT FOR INTERRUPT
FREQ_INT
EI
INCREASE COUNTER BUFFER BY ONE
CONVERT TO DECIMAL FORMAT
IS FREQUENCY>999999 ?
YES
DISPLAY '-OVER-'
A
NO
RETI


TIME_INT
EI
READ CHØ DOWN COUNTER RESIDUE VALUE
(TIMEINT) <---(TIMEINT)+1
IS (TIMEINT)>3Ø (Ø1EH)?
YES
NO
RETI
END
DISPLAY RESULT
A
CALL SCAN1
ANY KEYPRESSED ?
NO
YES
'GO'
NO
YES
START

```
LOC  OBJ CODE M STMT SOURCE STATEMENT

                  1  ;**********************************************
                  2  ;  FREQUENCY COUNTER : COUNT THE FREQUENCY OF THE*
                  3  ;                     INPUT SIGNAL            *
                  4  ;**********************************************
                  5  ;
                  6  CTCØ:   EQU     4ØH          ;CHANNEL Ø OF CTC
                  7  CTC1:   EQU     41H          ;CHANNEL 1 OF CTC
                  8  SCAN1:  EQU     Ø624H
                  9  DATADP  EQU     Ø671H        ;CHANGE DATA IN
                                                 ;A_REG TO DISPLAY FORMAT
                 1Ø  DEY_GO  EQU     16H
                 11  ;
                 12  ; PROGRAM BEGIN HERE!
                 13  ;
                 14  START:
18ØØ   21A5ØØ    15          LD      HL,TIMECNT    ; HL --> COUNT BUFFER
18Ø3   AF        16          XOR     A             ; RESET A
18Ø4   Ø6ØA      17          LD      B,ØAH         ; CLEAR FREQ  & TIMER
                                                   ; COUNT BUFFER
18Ø6   77        18  CLEAR   LD      (HL),A
18Ø7   23        19          INC     HL
18Ø8   1ØFC      2Ø          DJNZ    CLEAR
                 21          ;
                 22  ;SET CTC INITIAL CONDITION
                 23  ;
18ØA   3E2Ø      24          LD      A,2ØH         ;SET INTERRUPT REGISTER
                                                   ;VALUE
18ØC   ED47      25          LD      I,A
18ØE   3ED5      26          LD      A,11Ø1Ø1Ø1B   ;SET CHØ COUNTER MODE,
                                                   ;POSITIVE TRIGGER
181Ø   D34Ø      27          OUT     (CTCØ),A
1812   3E64      28          LD      A,1ØØ         ;SET THE VALUE OF
                                                   ;DOWN COUNTER
1814   D34Ø      29          OUT     (CTCØ),A
1816   3E4Ø      3Ø          LD      A,4ØH         ;SET INTERRUPT VECTOR
1818   D34Ø      31          OUT     (CTCØ),A
                 32          ;
181A   3EB5      33          LD      A,1Ø11Ø1Ø1B   ;SET CH1 TIMER CODE,
                                                   ;RANGE 256, POSITIVE
                                                   ;TRIGGER
181C   D341      34          OUT     (CTC1),A
181E   3EE9      35          LD      A,ØE9H        ;SET TIME CONSTANT
182Ø   D341      36          OUT     (CTC1),A
1822   ED5E      37          IM      2
1824   FB        38          EI
1825   18FE      39          JR      $
                 4Ø          ;
1827             41          DEFS    4ØH-($-START)
                 42  ;++++++++++++++++++++++++++++++++++++++++++++++++++
                 43  CTC_INT:               ;SET INTERRUPT SERVICE ROUTINE
                                              POINTER
184Ø   44ØØ      44          DEFW    FREQ_INT
```

```
1842  5D00              DEFW    TIME_INT
                        ;
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                FREQ_INT:
                ;FREQUENCY COUNTER INTERRUPT SERVICE ROUTINE
                ;INCREASE COUNTER AND CONVERT TO DECIMAL FORMAT
                ;
1844  FB                EI                      ;WHEN CPU ENTER THIS INTERRUPT
                                                ;SERVICE
                                                ;ROUTINE, IT WILL DISABLE
                                                ;ANOTHER INTERRUPT
                                                ;SO  "EI"  CAN  LET  COUNT_INT
                                                ;HAPPEN
1845  21A700            LD      HL,FREQCNT+1
1848  0602              LD      B,2              ;SET FREQUENCY BUFFER
                                                 ;COUNTERR
                ADDONE:
184A  FB                EI
184B  7E                LD      A,(HL)
184C  C601              ADD     A,1             ;INCREASE COUNTE BY ONE
184E  27                DAA                     ;CHANGE TO DECIMAL FORMAT
184F  77                LD      (HL),A          ;RESTORE VALUE
1850  3009              JR      NC,NOTOVER      ;NOT OVER 99
1852  23                INC     HL              ;OVER,MUST INCREASE  HIGH
                                                ;ORDER BUE ONE
1853  10F5              DJNZ    ADDONE
1855  DD21AF00          LD      IX,OVER         ;FREQUENCY LARGER THEN
                                                ;999999 DISPLAY '-OVER-'
1859  183B              JR      DISPLAY
                NOTOVER:
185B  ED4D              RETI
                        ;
                ;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++
                TIME_INT:
                ;CH1 (TIMER MODE)  INTERRUPT SERVICE ROUTINE
                ;CH1: TRIGGERED BY 'MPF-1' CLOCK(1789772HZ)
                ;EVERY 256*233 HZ, THIS SERVICE ROUTINE WILL DO AGAIN
                ;UNTIL 30 TIMES (--1 SEC),PROGRAM HALT AND SIGNAL
                ;FREQUENCY GET
185D  FB                EI                      ;LET ANOTHER  INTERRUPT
                                                ;CAN HAPPEN ANYTIME
185E  DB40              IN      A,(CTC0)        ;GET CH0  GOWN  COUNTER
                                                ;RESIDUE VALUE
1860  D664              SUB     100
1862  ED44              NEG
1864  32A600            LD      (FREQCNT),A     ;SAVE THIS TO BUFFER
                        ;
1867  21A500            LD      HL,TIMECNT      ;INCREASE TIME COUNTER BY
                                                ;ONE
186A  7E                LD      A,(HL)
186B  3C                INC     A
186C  77                LD      (HL),A          ;RESTORE VALUE
186D  FE1E              CP      01EH            ;CHECK ONE SEC ?
186F  3002              JR      NC,END          ;YES
1871  ED4D              RETI
```

```
                    ;*************************************************************
                    END:
1873   F3                   DI
1874   Ø6Ø2                 LD      B,2           ;CHANGE LOWER BYTE
                                                  ;OF FREQUENCY COUNTER
                                                  ;TO DECIMAL FORMAT
1876   AF                   XOR     A
1877   21A6ØØ               LD      HL,FREQCNT
187A   ED67         CHANGE: RRD
187C   CEØØ                 ADC     A,Ø
187E   27                   DAA
187F   1ØF9                 DJNZ    CHANGE
1881   ED67                 RRD
                            ;
1883   Ø6Ø3                 LD      B,3
1885   11A6ØØ               LD      DE,FREQCNT    ;CHANGE FREQ TO DISPLAY
                                                  ;PATTERN
1888   21A9ØØ               LD      HL,OUTBUF
                    CONVERT:
188B   1A                   LD      A,(DE)
188C   13                   INC     DE
188D   CD74Ø6               CALL    DATADP+3
189Ø   1ØF9                 DJNZ    CONVERT
                            ;
1892   DD21A9ØØ             LD      IX,OUTBUF     ;DISPLAY DATA TO LED
1896   CD24Ø6       DISPLAY:CALL    SCAN1
1899   38FB                 JR      C,DISPLAY     ;NO KEY PRESSED,SCAN
                                                  ;AGAIN
189B   FE16                 CP      KEY_GO        ;PRESSED 'GO' ?
189D   2ØF7                 JR      NZ,DISPLAY    ;NO
189F   21ØØØØ               LD      HL,START      ;RETURN TO PROGRAM
                                                  ;STRTING ADDR. &
                                                  ;COUNT FREQUENCY AGAIN
18A2   E3                   EX      (SP),HL
18A3   ED4D                 RETI
                            ;
                            ;
18A5                TIMECNT DEFS    1             ;TIMER COUNT BUFFER
18A6                FREQCNT DEFS    3             ;FREQUENCY COUNT BUFFER
18A9                OUTBUF  DEFS    6             ;DISPLAY BUFFER
                    OVER:
18AF   Ø2                   DEFB    Ø2H     ;'-'
18BØ   Ø3                   DEFB    Ø3H     ;'R'
18B1   8F                   DEFB    Ø8FH    ;'E'
18B2   B7                   DEFB    ØB7H    ;'V'
18B3   A3                   DEFB    ØA3H    ;'O'
18B4   Ø2                   DEFB    Ø2H     ;'-'
```

Micro-Professor Application Note

—DOC. NO. MPF-I-02-210A —

MPF-I AS A TRAFFIC LIGHT CONTROLLER

An Application Example of Z8Ø-PIO.

MULTITECH INDUSTRIAL CORPORATION

OFFICE : 977, MIN SHEN E. ROAD, TAIPEI, 105, TAIWAN, R.O.C.
TEL: (02) 769-1225 (10 LINES) TELEX: 23756 MULTIIC
FACTORY: 5, TECHNOLOGY ROAD III
HSINCHU SCIENCE-BASED INDUSTRIAL PARK.
HSINCHU, TAIWAN, 300, R.O.C.
TEL: (035) 775102 (3 LINES)

US$1.00

Purpose: Use PIO for traffic light control

Required Equipment: A PIO chip, a 75492, three LED lamps (one in green, one in red, and one in yellow), three resistors, and some wire.

You are required to use the necessary devices to make the hardware connections in accordance with the diagram shown below:

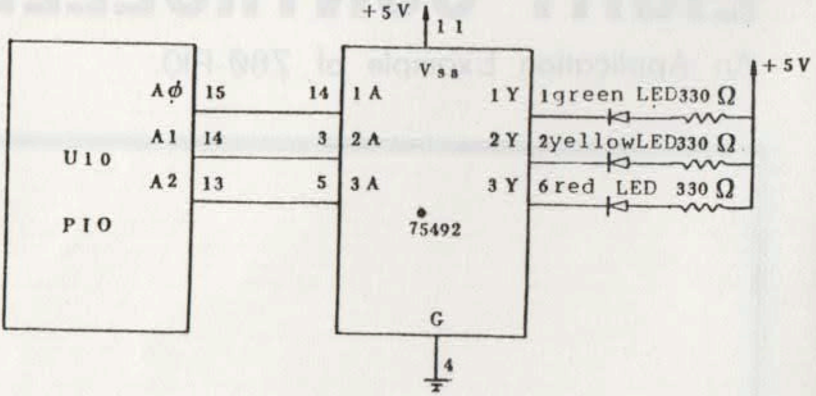


fig A

Expriment Explanation:

1. The PIO is a 40-pin large-scale integrated circuit (LSI) especially designed to provide TTL compatible interface between peripheral devices and the Z80 CPU. The CPU can configure the Z80-PIO to interface with a wide range of peripheral devices with no other external logic required. Typical peripheral devices that are fully compatible with the Z80-CPU include most keyboard, paper tape readers and punches, printers, and PROM programmers, etc. It is programmable. The PIO has two I/O ports--port A and port B. Each port is connected to eight pins. The addresses of the PIO are from 80 to 83 (in hexadecimal). In this experiment, port A will be used. For detailed description of the PIO and its operation, refer to "Z80 Microprocessor Programming and Interfacing, Book 2" by Nichols, Rony, published by Blacksburg; or Z80 Handbook.

2. Each of the two ports of the PIO has four modes of operation; namely, byte output, byte input, byte bidirectional bus, and bit control mode. The mode of operation must be established by writing a control word to the PIO in the following format:

D7	D6	mode of operation
ϕ	ϕ	Byte output
ϕ	1	Byte input
1	ϕ	Byte bidirectional
1	1	Bit control

fig B

We can change the contents of bit D7 and D6 to form a control word in order to change the mode of operation of port A.

3. In this experiment, the mode of operation of port A is byte output. Thus, the contents of bit D7 and D6 should be zero, and the contents of bit D3 through bit DØ should be one. The contents of bit D5 and D4 make no difference to the control word.

D7	D6	D5	D4	D3	D2	D1	Dϕ
M1	Mϕ	×	×	1	1	1	1

control word

4. Of the four addresses of PIO, two addresses are assigned to port A--8ØH is used as the data port of port A, and 82H is used as the control port of port A. Since we use port A in its byte output mode, the control word is set ØØØØ1111(binary) (or ØFH). The value of the control word should be sent to the control port of Port A to set Port A to its byte output mode.

5. We use the bit Ø (AØ) of Port A to control the green light, A1 to control the yellow light, and A2 to control the red light. To illuminate the red light, the value Ø1 should be sent to the data port of PIO (whose address is 8ØH). By sending Ø1H to the data port of PIO, the eight bits on the Port A will become

A7	A6	A5	A4	A3	A2	A1	AØ
Ø	Ø	Ø	Ø	Ø	Ø	Ø	1

The 75492 will convert the input from AØ to low, so the output at pin 1Y of 75492 is low. This will cause the electrical current to flow from the resistor to the green LED lamp.

To illuminate the yellow LED, the byte (Ø2H) should be sent to the data port of the PIO. This byte will cause the A1 high and 2Y low. To illuminate the red lamp, the byte (Ø4H) is sent to the data port of the PIO.

6. For how long will a lamp be illuminated? This is controlled by time delay subroutines--DELAY, DELAY1, and DELAY2.

Since the MPF-I operates at 1.79MHz, a T state is Ø.56 micro-seconds. Therefore, the time delay achieved by the DELAY subroutine is

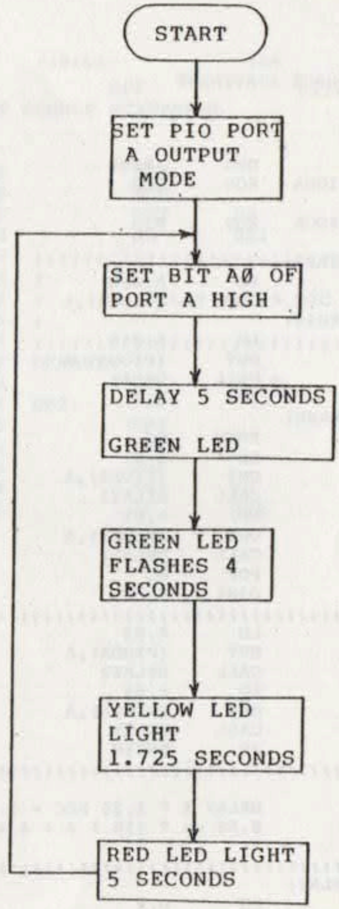
Ø.56 micro-seconds x {7+4[1Ø+(16+4+4+1Ø)x65536+4+12]-5+1Ø}=4.9912867 sec

And the time delay for DELAY1 is

Ø.56 x [1Ø+(16+4+4+11+12)x19ØØØ]=Ø.5ØØØ856 sec

The time delay for DELAY2 is

Ø.56 x [1Ø+(16+4+4+11+12)x65536]=1.7249131 sec


START
SET PIO PORT A OUTPUT MODE
SET BIT AØ OF PORT A HIGH
DELAY 5 SECONDS
GREEN LED
GREEN LED FLASHES 4 SECONDS
YELLOW LED LIGHT 1.725 SECONDS
RED LED LIGHT 5 SECONDS

```
                              MPF               821015               PAGE   1
LOC    OBJ CODE M STMT SOURCE STATEMENT                              ASM 5.8

1800                                 ORG     1800H
                         PIODA       EQU     80H              ;DATA PORT OF PIO
                                                               CHANNEL A
                         PIOCA       EQU     82H              ;CONTROL PORT OF PIO
                                                               CHANNEL A
                         START:
1800   3E0F                          LD      A,0FH
1802   D382                          OUT     (PIOCA),A        ;PIO PORT A OUTPUT MODE
                         BEGIN:
1804   3E01                          LD      A,01H
1806   D380                          OUT     (PIODA),A        ;GREEN LED LIGHT
1808   CD3018                        CALL    DELAY            ;DELAY 5 SEC
180B   0604                          LD      B,4
                         FLASH:
180D   C5                            PUSH    BC
180E   3E00                          LD      A,0
1810   D380                          OUT     (PIODA),A        ;FLASH 4 SEC
1812   CD4018                        CALL    DELAY1
1815   3E01                          LD      A,01
1817   D380                          OUT     (PIODA),A
1819   CD4018                        CALL    DELAY1
181C   C1                            POP     BC
181D   10EE                          DJNZ    FLASH
                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
181F   3E02                          LD      A,02
1821   D380                          OUT     (PIODA),A        ;YELLOW LED LIGHT
1823   CD4A18                        CALL    DELAY2           ;1.725 SEC
1826   3E04                          LD      A,04
1828   D380                          OUT     (PIODA),A        ;RED LED LIGHT
182A   CD3018                        CALL    DELAY            ;5 SEC
182D   C30418                        JP      BEGIN .
                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         ;
                         ;           DELAY 4 * 1.25 SEC = 5 SEC  SUBROUTINE
                         ;           0.56 us * (16 + 4 + 4 + 10 ) * 65536 =1.25 SEC
                         ;
                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         DELAY:
1830   1604                          LD      D,4              ; 7T
                         DELX:
1832   010000                        LD      BC,0             ; 10T
                         DE0:
1835   EDA1                          CPI                      ; 16T
1837   00                            NOP                      ; 4T
1838   00                            NOP                      ; 4T
1839   EA3518                        JP      PE,DE0           ; 10T
183C   15                            DEC     D
183D   20F3                          JR      NZ,DELX
183F   C9                            RET
                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         ;
                         ;           DELAY 0.5 SEC SUBROUTINE
                         ;           0.56 us * (16 + 4 + 4 + 11 + 12 ) * 19000 =0.5 SEC
                         ;
                         ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                         DELAY1:
1840   01384A                        LD      BC,4A38H
                         DE1:
1843   EDA1                          CPI
```

```
LOC    OBJ CODE M STMT SOURCE STATEMENT

1845   ØØ             59            NOP
1846   ØØ             6Ø            NOP
1847   EØ             61            RET    PO
1848   18F9           62            JR     DE1
                      63   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                      64   ;
                      65   ;          DELAY 1.725 SEC SUBROUTINE
                      66   ;
                      67   ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
                      68   DELAY2:
184A   Ø1ØØØØ         69            LD     BC,Ø
                      7Ø   DE2:
184D   EDA1           71            CPI
184F   ØØ             72            NOP
185Ø   ØØ             73            NOP
1851   EØ             74            RET    PO
1852   18F9           75            JR     DE2
```

Micro-Professor Application Note

— DOC. NO. MPF-I-03-210A —

MPF-I AS A GAME SOUND GENERATOR

Flying Saucer & Laser Gun.

MULTITECH INDUSTRIAL CORPORATION

OFFICE : 977, MIN SHEN E. ROAD, TAIPEI, 105, TAIWAN, R.O.C.
TEL: (02)769-1225 (10 LINES) TELEX: 23756 MULTIIC

FACTORY: 5. TECHNOLOGY ROAD III
HSINCHU SCIENCE-BASED INDUSTRIAL PARK.
HSINCHU, TAIWAN, 300, R.O.C.
TEL: (035)775102 (3 LINES)

US$0.50

Purpose: Simulating flying saucer sound and other computer-controlled sound.

Required Equipment: MPF-I

Expriment Explanation:

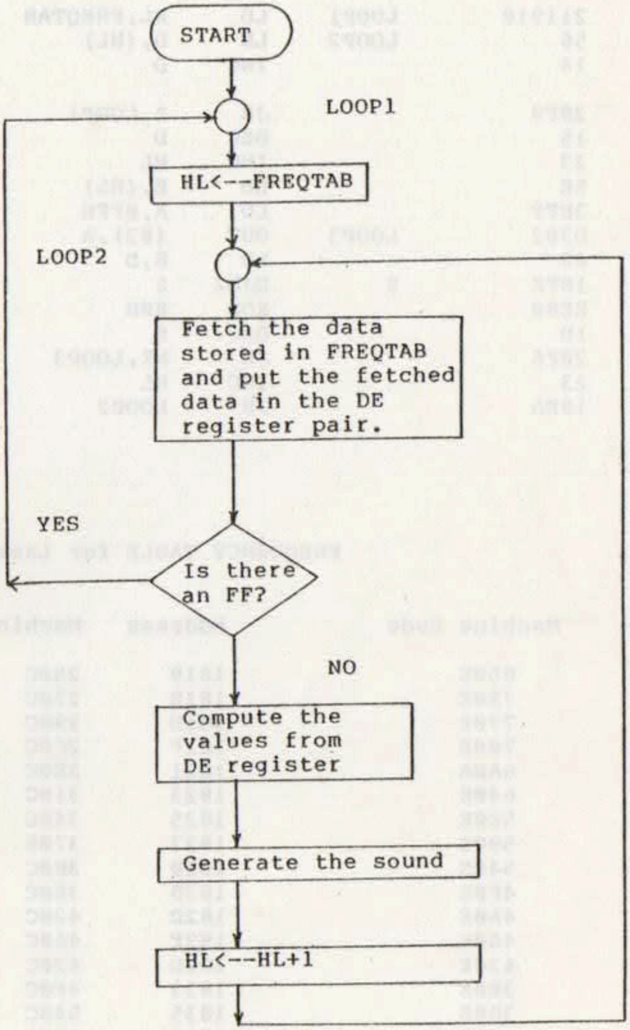
1. The simulated flying saucer sound is created by rapidly raising the frequency from 800Hz to 2400Hz.

2. The frequency table is stored in the memory locations pointed to by the addresses following 1819H. The first byte is the parameter of frequency, and the second byte is used to store the length of a sound. The machine code "FF" at the end of the frequency table is designed to begin another sound cycle.

3. The frequency table can be changed so that various computer synthesized sounds can be generated.

4. The frequency table for the sound of a laser gun is provided. The sound of the laser gun is generated by rapidly reducing the frequency from 2400Hz to 800Hz.

Flowchart


START
LOOP1
HL<--FREQTAB
LOOP2
Fetch the data stored in FREQTAB and put the fetched data in the DE register pair.
YES
Is there an FF?
NO
Compute the values from DE register
Generate the sound
HL<--HL+1

Address	Machine Code	Label	OP Code	Operand	Comment
18Ø0	211918	LOOP1	LD	HL,FREQTAB	
18Ø3	56	LOOP2	LD	D,(HL)	
18Ø4	14		INC	D	Check FEPEAT code --FF
18Ø5	28F9		JR	Z,LOOP1	
18Ø7	15		DEC	D	
18Ø8	23		INC	HL	
18Ø9	5E		LD	E,(HL)	
18ØA	3EFF		LD	A,ØFFH	
18ØC	D3Ø2	LOOP3	OUT	(Ø2),A	
18ØE	42		LD	B,D	
18ØF	1ØFE	$	DJNZ	$	LOOP B times
1811	EE8Ø		XOR	8ØH	
1813	1D		DEC	E	
1814	2ØF6		JR	NZ,LOOP3	LOOP E times
1816	23		INC	HL	
1817	18EA		JR	LOOP2	

FREQUENCY TABLE for Laser Gun

Address	Machine Code	Address	Machine Clde
1819	85ØE	1819	25ØC
181B	7EØE	181B	27ØC
181D	77ØE	181D	29ØC
181F	7ØØE	181F	2CØC
1821	6AØA	1821	2EØC
1823	64ØE	1823	31ØC
1825	5EØE	1825	34ØC
1827	59ØE	1827	37ØE
1829	54ØE	1829	3BØC
182B	4FØE	182B	3EØC
182D	4AØE	182D	42ØC
182F	46ØE	182F	46ØC
1831	42ØE	182D	42ØC
1833	3EØE	1833	4FØC
1835	3BØE	1835	54ØC
1837	37ØE	1837	59ØC
1839	34ØE	1839	5EØC
183B	31ØE	183B	64ØC
183D	2EØE	183D	6AØC
183F	2CØE	183F	7ØØC
1841	29ØE	1841	77ØC
1843	27ØE	1843	7EØC
1845	25ØE	1845	85ØC
1847	FF	1847	FF
	END		END

Micro-Professor Application Note

—DOC. NO. MPF-I-04-210A —

PLAYING POKER GAME WITH MPF-I

An Application Example of Z8Ø-PIO.

MULTITECH INDUSTRIAL CORPORATION
OFFICE : 977, MIN SHEN E. ROAD, TAIPEI, 105, TAIWAN, R.O.C.
TEL: (02)769-1225 (10 LINES) TELEX: 23756 MULTIIC
FACTORY: 5, TECHNOLOGY ROAD III ,
HSINCHU SCIENCE-BASED INDUSTRIAL PARK,
HSINCHU, TAIWAN, 300, R.O.C.
TEL: (035)775102 (3 LINES)

US$1.00

Purpose: Simulating poker with PIO in order to familiarize with its functions.

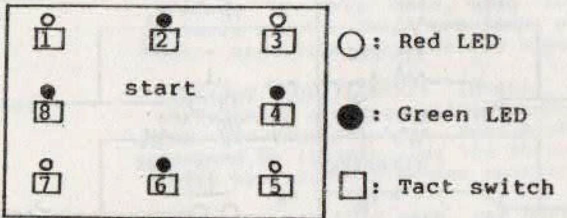
Required Equipment: The MPF-I, a PIO, tact switch x 9
10K ohm 1/4W x 16 2SA 1015 x 8
330 ohm 1/4W x 8 Green LED x 4
PCB 100mm x 70mm Red LED x 4

Expriment Explanation:

1. This experiment is similar to poker game. The only difference is that the conventional poker game uses five cards as a suit. However, this experiment deals only three-card suit to players.

2. You are required to make your own keyboard and external circuitry. The keyboard is shown below

Keyboard



The needed parts for making a keyboard are:

1. 10K ohm 1/4W x 16
2. 330 ohm 1/4W x 8
3. Red LED (30) x 4
4. Green LED (30) x 4
5. 2SA1015 x 8
6. Tact switch x 9

3. Your keyboard should be connected to the CTC PIO I/O BUS on the MPF-I board. You can find a 40-pin male connector which is to the left of the empty socket reserved for the PIO. The pin functions of the connector are shown in the schematic in the MPF-I User's Manual on Sheet 4 of 4, P2 PIN FUNCTION. You are required to make the connections in accordance with the schematics provided in this experiment, which is shown below:

4. After you finished your own keyboard and the external circuitry, you are required to key in the program provided in this experiment. After you have keyed in the program, you are ready to play the game. But before you proceed to play the poker, you must know how the MPF-I respond to the key press and how to calculate your score.

5. To play the game, you have to press the START key located in the center of your keyboard. After you have pressed the START key, the MPF-I will deal three cards (though you can not see these cards in reality.) The three cards will come in

   a. THREE OF A KIND: That means the three cards dealed are of the same number. Once you get this suit, you score 20 points, and only one LED lamp is illuminated on your keyboard.

   b. FLUSH: That means the three cards are of the same group-either all in red or all in green, In this case, you score three points, and the three LED lamps of the same color are illuminated at the same time.

   c. STRAIGHT (OR TIERCE): In this case, the three cards are sequentially numbered, and you get five points. Your keyboard will respond by illuminating the three LED lamps which are adjacent to one another.

   d. ONE PAIR: In this case, you have a pair of cards whose number is identical, and you score five points minus. Your keyboard will respond by illuminaing two LED lamps.

   e. OTHERS: In this case, your score is deducted by two.

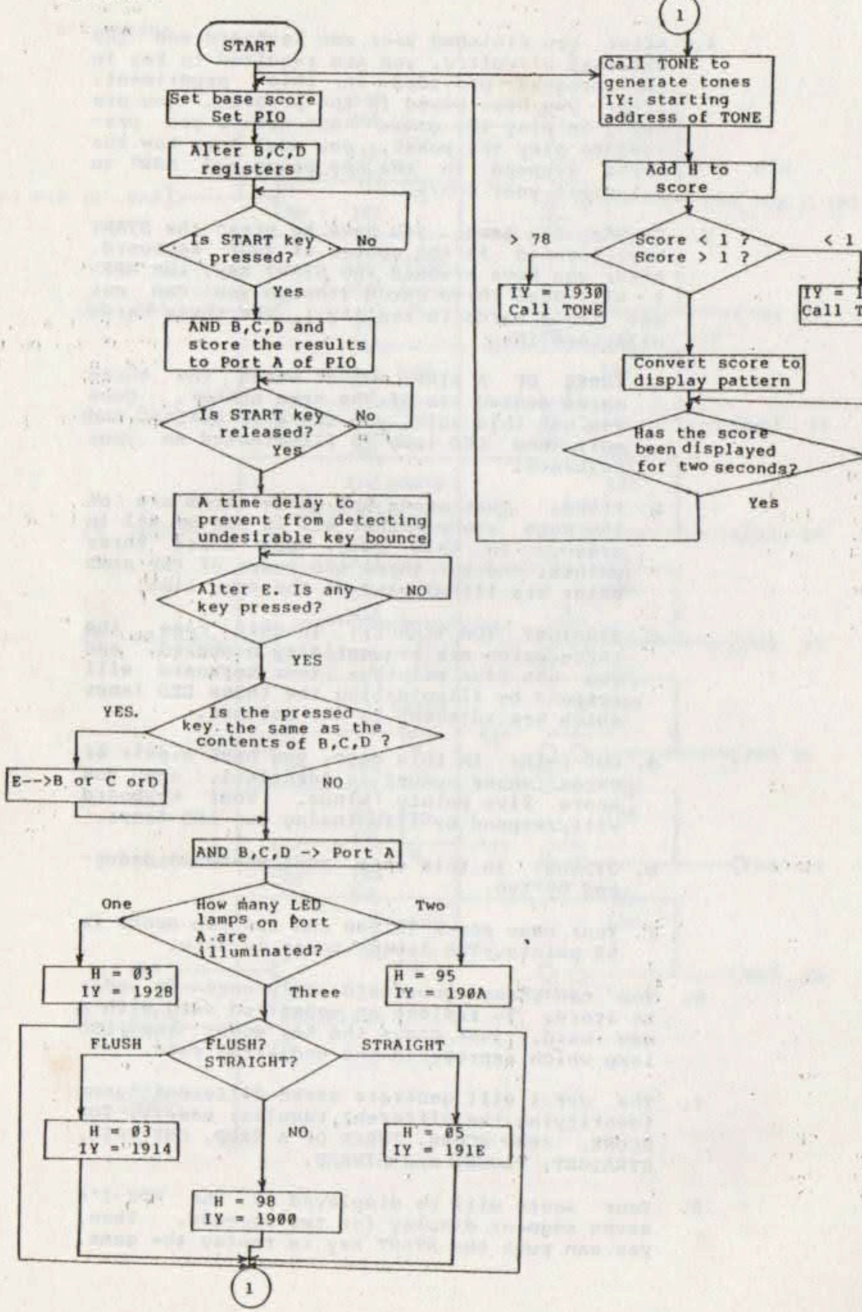
   f. Your base score is ten and the top score is 50 points. The lowest score is zero.

6. You can change your card--only once--in order to score. To replace an undesired card with a new card, just press the key under the LED lamp which represents the undesired card.

7. The MPF-I will generate seven different tones identifying the different results; namely, TOP SCORE, ZERO SCORE, THREE OF A KIND, ONE PAIR, STRAIGHT, FLUSH, and OTHERS.

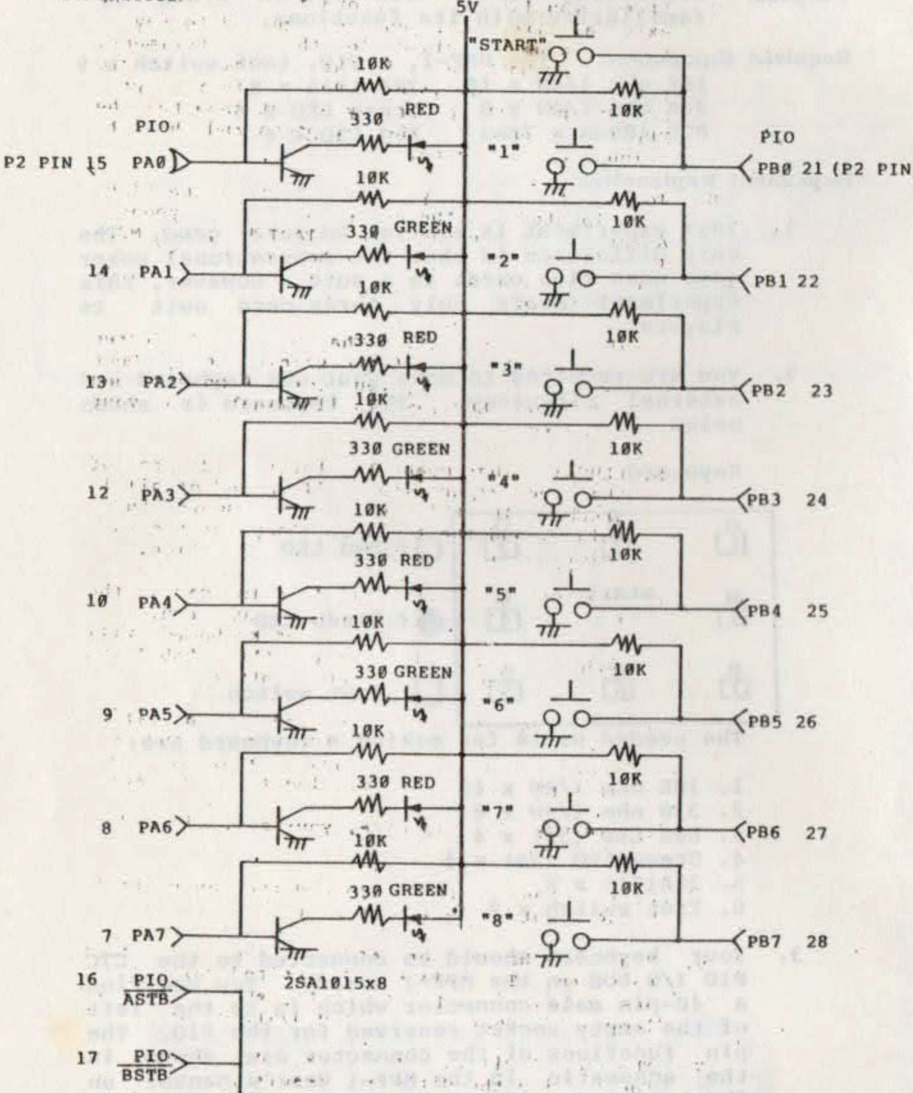
8. Your score will be displayed on the MPF-I's seven segment display for two seconds. Then, you can push the START key to replay the game.

Flowchart



START
Set base score
Set PIO
Alter B,C,D registers
Is START key pressed?
No
Yes
AND B,C,D and store the results to Port A of PIO
Is START key released?
No
Yes
A time delay to prevent from detecting undesirable key bounce
Alter E. Is any key pressed?
NO
YES
Is the pressed key the same as the contents of B,C,D ?
YES.
E-->B or C orD
NO
AND B,C,D -> Port A
How many LED lamps on Port A are illuminated?
One
Two
Three
H = Ø3
IY = 1928
H = 95
IY = 19ØA
FLUSH?
STRAIGHT?
FLUSH
STRAIGHT
NO
H = Ø3
IY = 1914
H = Ø5
IY = 191E
H = 98
IY = 19ØØ
1
1
Call TONE to generate tones
IY: starting address of TONE
Add H to score
Score < 1 ?
Score > 1 ?
> 78
< 1
IY = 193Ø
Call TONE
IY = 195Ø
Call TONE
Convert score to display pattern
Has the score been displayed for two seconds?
No
Yes

schematics



5V
"START"
10K
330 RED
PIO
P2 PIN 15 PA0
"1"
PIO
PB0 21 (P2 PIN)
330 GREEN
14 PA1
"2"
PB1 22
330 RED
13 PA2
"3"
PB2 23
330 GREEN
12 PA3
"4"
PB3 24
330 RED
10 PA4
"5"
PB4 25
330 GREEN
9 PA5
"6"
PB5 26
330 RED
8 PA6
"7"
PB6 27
330 GREEN
7 PA7
"8"
PB7 28
16 PIO ASTB
17 PIO BSTB
2SA1015x8

```
LOC   OBJ CODE M STMT SOURCE STATEMENT

1800               1         ORG    1800H
                   2  PIODA  EQU    80H
                   3  PIODB  EQU    81H
                   4  PIOCA  EQU    82H
                   5  SCAN1  EQU    0624H
                   6  HEX7SG EQU    0678H
1800  210000       7         LD     HL,0
1803  220419       8         LD     (SCOBF+2),HL
1806  220619       9         LD     (SCOBF+4),HL
1809  3E0A        10  M1:    LD     A,10          ;Set base score
180B  320019      11         LD     (SCORE),A     ;to 10.
180E  DD210219    12  M2:    LD     IX,SCOBF
1812  3E8F        13         LD     A,8FH         ;Set PIO's Port A to
1814  D382        14         OUT    (PIOCA),A     ;byte output and port
                  15                              ;B to byte input mode.
1816  3EFF        16         LD     A,0FFH        ;Port A outputs
                  17                              ;11111111.
1818  D380        18         OUT    (PIODA),A     ;to keep the LED off.
181A  16FE        19  M3     LD     D,0FEH
181C  4A          20         LD     C,D
181D  42          21         LD     B,D
181E  DB81        22  M4     IN     A,(PIODB)     ;Check if a key
1820  FEFE        23         CP     0FEH          ;is pressed? If the
1822  180C        24         JR     M5            ;START key was not
1824  CB00        25         RLC    B             ;pressed, then
                  26                              ;alter the
1826  38F6        27         JR     C,M4          ;value of registers
                  28                              ;B,C,D.
1828  CB01        29         RLC    C             ;If the START key was
182A  38DD        30         JR     C,M1          ;pressed, then execute
182C  CB02        31         RLC    D             ;M5.
182E  18EE        32         JR     M4
                  33         ;*********************************************
1830  78          34  M5     LD     A,B           ;AND B,C,D
1831  A1          35         AND    C
1832  A2          36         AND    D
1833  D380        37         OUT    (PIODA),A     ;Store the results of
                  38                              ;ANDed B,C,D to Port A
                  39                              ;to illuminate 1 to 3
                  40                              ;LED lamps.
1835  1EFE        41         LD     E,0FEH
                  42  M6:
                  43  ; The next three instructions are to check
                  44  ; whether the pressed key is released?
1837  DB81        45         IN     A,(PIODB)
1839  FEFF        46         CP     0FFH
183B  20FA        47         JR     NZ,M6
183D  26FF        48         LD     H,0FFH        ;A time delay
183F  2EFF        49  M7     LD     L,0FFH        ;to prevent from
1841  2D          50  M8     DEC    L             ;detecting undesirable
1842  20FD        51         JR     NZ,M8         ;key bounce.
1844  25          52         DEC    H
1845  20F8        53         JR     NZ,M7
1847  DB81        54         IN     A,(PIODB)     ;Detect if a key
1849  CB03        55  M9     RLC    E             ;is pressed?  If not,
184B  FEFF        56         CP     0FFH          ;alter the value
184D  28FA        57         JR     Z,M9          ;of E.
184F  B8          58         CP     B             ;If the key pressed
```

```
LOC   OBJ CODE M STMT SOURCE STATEMENT

1850  2003        59          JR    NZ,M1Ø  ;has the same
1852  43          6Ø          LD    B,E     ;internal code as
                  61                        ;the content of the
                  62                        ;B register,
                  63                        ;load E to B.
1853  18ØA        64          JR    M12
1855  B9          65   M1Ø    CP    C       ;If the internal code of
1856  2003        66          JR    NZ,M11  ;the key pressed is the same
1858  4B          67          LD    C,E     ;as the content of C,
                  68                        ;load E to C.
1859  1804        69          JR    M12
185B  BA          7Ø   M11    CP    D       ;If the internal code of
185C  2006        71          JR    NZ,M13  ;the key pressed is the same
185E  53          72          LD    D,E     ;as the content of C,
                  73                        ;load E to D.
185F  78          74   M12    LD    A,B     ;Output the ANDed value
186Ø  A1          75          AND   C       ;of B,C,D to Port A
1861  A2          76          AND   D       ;to illuminate 1 to 3 LED.
1862  D38Ø        77          OUT   (PIODA),A
1864  DB8Ø        78   M13    IN    A,(PIODA);Check the illuminated
1866  2609        79          LD    H,Ø9    ;LED lamps connected to
1868  2EØØ        8Ø          LD    L,Ø     ;Port A
186A  25          81   M14    DEC   H
186B  28Ø7        82          JR    Z,M15
186D  CBØ7        83          RLC   A
186F  38F9        84          JR    C,M14
1871  2C          85          INC   L
1872  18F6        86          JR    M14
1874  7D          87   M15    LD    A,L     ;If L = 1, only one
1875  FEØ1        88          CP    Ø1      ;LED lamp was illuminated.
1877  2ØØ8        89          JR    NZ,M16
1879  FD212819    9Ø          LD    IY,TAB5 ;Load the starting address
                  91                        ;of TONE to IY.
187D  2614        92          LD    H,2Ø    ;Put the score to H.
187F  1845        93          JR    M23
1881  FEØ2        94   M16    CP    Ø2      ;If L = 2, two LED
1883  2ØØ8        95          JR    NZ,M17  ;lamps were illuminated.
1885  FD21ØF19    96          LD    IY,TAB2 ;Load the stating address
                  97                        ;of TAB2 to IY.
1889  265F        98          LD    H,95    ;Load score to H.
188B  1839        99          JR    M23
188D  DB8Ø        1ØØ  M17    IN    A,(PIODA);Check to see how many
188F  2EØ8        1Ø1         LD    L,Ø8    ;LED lamps on Port A
1891  2D          1Ø2  M18    DEC   L       ;are illuminated?
1892  281Ø        1Ø3         JR    Z,M19   ;If three adjacent LED
1894  CBØ7        1Ø4         RLC   A       ;lamps are illuminated,
1896  38F9        1Ø5         JR    C,M18   ;load 5 to H (which is
1898  CBØ7        1Ø6         RLC   A       ;the register used to put
189A  38F5        1Ø7         JR    C,M18   ;score.)
189C  FD211F19    1Ø8         LD    IY,TAB4 ;Load the starting address
                  1Ø9                       ;of TAB4 to IY.
18AØ  26Ø5        11Ø         LD    H,Ø5
18A2  1822        111         JR    M23
18A4  DB8Ø        112  M19    IN    A,(PIODA);Check the illuminated
18A6  2EØ8        113         LD    L,Ø8    ;LED lamps on Port A.
18A8  1EØ3        114  M2Ø    LD    E,Ø3    ;If the three lighted
18AA  2D          115         DEC   L       ;LED lamps are of the
18AB  2813        116         JR    Z,M22   ;same color,
```

```
18AD  CB07       117  M21    RLC   A
18AF  38F7       118         JR    C,M20
18B1  CB07       119         RLC   A
18B3  30F3       120         JR    NC,M20
18B5  1D         121         DEC   E
18B6  20F5       122         JR    NZ,M21
18B8  2603       123         LD    H,03       ;then load 3 to H,
18BA  FD211819   124         LD    IY,TAB3    ;and load the stating address
                 125                          ;of TAB3 to IY.
18BE  1806       126         JR    M23
18C0  2662       127  M22    LD    H,98       ;If none of the above happens,
                 128                          ;load 98H (-2 in decimal)
                 129                          ;to H.
18C2  FD210819   130         LD    IY,TAB1    ;and load TAB1 to IY.
18C6  CD5D19     131  M23    CALL  TONE       ;Generate a tone.
18C9  3A0019     132         LD    A,(SCORE)  ;Add accumulated score
18CC  84         133         ADD   A,H        ;together.
18CD  27         134         DAA
18CE  FE01       135         CP    01         ;If the total score < 1,
18D0  F2DD18     136         JP    P,M24
18D3  FD214C19   137         LD    IY,TAB7    ;load TAB7 to IY.
18D7  CD5D19     138         CALL  TONE       ;After generating a tone,
18DA  C30918     139         JP    M1         ;jump to M1 to replay
                 140                          ;the game.
18DD  FE32       141  M24    CP    50         ;If the total score > 50,
18DF  FAEC18     142         JP    M,M25
18E2  FD213D19   143         LD    IY,TAB6    ;load TAB6 to IY.
18E6  CD5D19     144         CALL  TONE       ;After generating a tone,
18E9  C30918     145         JP    M1         ;jump to M1 to replay.
18EC  210219     146  M25    LD    HL,SCOBF
18EF  320019     147         LD    (SCORE),A;Load score to SCORE.
18F2  CD7806     148         CALL  HEX7SG     ;Convert score to
                 149                          ;display pattern.
18F5  26BB       150         LD    H,0BBH
18F7  CD2406     151  M26    CALL  SCAN1
18FA  25         152         DEC   H          ;Press any key of the MPF-I,
18FB  20FA       153         JR    NZ,M26
18FD  C31A18     154         JP    M3         ;then jump to M3.
1900             155  SCORE  DEFS  2
1902             156  SCOBF  DEFS  6
1908  22BB       157  TAB1:  DEFW  0BB22H
190A  66DD       158         DEFW  0DD66H
190C  99FF       159         DEFW  0FF99H
190E  00         160         DEFB  0
190F  6655       161  TAB2:  DEFW  5566H
1911  99AA       162         DEFW  0AA99H
1913  CCDD       163         DEFW  0DDCCH
1915  99FF       164         DEFW  0FF99H
1917  00         165         DEFB  0
1918  11AA       166  TAB3:  DEFW  0AA11H
191A  2288       167         DEFW  8822H
191C  3366       168         DEFW  6633H
191E  00         169         DEFB  0
191F  11AA       170  TAB4:  DEFW  0AA11H
1921  3388       171         DEFW  8833H
1923  11AA       172         DEFW  0AA11H
1925  3388       173         DEFW  8833H
1927  00         174         DEFB  0
```

```
LOC   OBJ CODE M STMT SOURCE STATEMENT

1928  77FF        175  TAB5:  DEFW   ØFF77H
192A  66EE        176         DEFW   ØEE66H
192C  55DD        177         DEFW   ØDD55H
192E  44CC        178         DEFW   ØCC44H
193Ø  33BB        179         DEFW   ØBB33H
1932  22AA        18Ø         DEFW   ØAA22H
1934  1199        181         DEFW   9911H
1936  1188        182         DEFW   8811H
1938  8888        183         DEFW   8888H
193A  9977        184         DEFW   7799H
193C  ØØ          185         DEFB   Ø
193D  11FF        186  TAB6:  DEFW   ØFF11H
193F  22EE        187         DEFW   ØEE22H
1941  33DD        188         DEFW   ØDD33H
1943  44CC        189         DEFW   ØCC44H
1945  55BB        19Ø         DEFW   ØBB55H
1947  66AA        191         DEFW   ØAA66H
1949  7799        192         DEFW   9977H
194B  ØØ          193         DEFB   Ø
194C  FF11        194  TAB7:  DEFW   11FFH
194E  EE22        195         DEFW   22EEH
195Ø  DD33        196         DEFW   33DDH
1952  CC44        197         DEFW   44CCH
1954  BB55        198         DEFW   55BBH
1956  AA66        199         DEFW   66AAH
1958  9977        2ØØ         DEFW   7799H
195A  FFAA        2Ø1         DEFW   ØAAFFH
195C  ØØ          2Ø2         DEFB   Ø
195D  D9          2Ø3  TONE   EXX
195E  FD7EØØ      2Ø4  M9Ø    LD     A,(IY)
1961  5F          2Ø5         LD     E,A
1962  FEØØ        2Ø6         CP     ØØH
1964  2818        2Ø7         JR     Z,M2ØØ
1966  FD23        2Ø8         INC    IY
1968  FD4EØØ      2Ø9         LD     C,(IY)
196B  3EFF        21Ø         LD     A,ØFFH
196D  D3Ø2        211  M1ØØ   OUT    (Ø2),A
196F  47          212         LD     B,A
197Ø  ØØ          213  M15Ø   NOP
1971  ØØ          214         NOP
1972  ØØ          215         NOP
1973  1ØFB        216         DJNZ   M15Ø
1975  EE8Ø        217         XOR    8ØH
1977  ØD          218         DEC    C
1978  2ØF3        219         JR     NZ,M1ØØ
197A  FD23        22Ø         INC    IY
197C  18EØ        221         JR     M9Ø
197E  D9          222  M2ØØ   EXX
197F  C9          223         RET
```

SYMBOL	VAL	M	DEFN	REFS				
HEX7SG	0678		6	148				
M1	1809		10	30	139	145		
M10	1855		65	59				
M100	196D		211	219				
M11	185B		70	66				
M12	185F		74	64	69			
M13	1864		78	71				
M14	186A		81	84	86			
M15	1874		87	82				
M150	1970		213	216				
M16	1881		94	89				
M17	188D		100	95				
M18	1891		102	105	107			
M19	18A4		112	103				
M2	180E		12					
M20	18A8		114	118	120			
M200	197E		222	207				
M21	18AD		117	122				
M22	18C0		127	116				
M23	18C6		131	93	99	111	126	
M24	18DD		141	136				
M25	18EC		146	142				
M26	18F7		151	153				
M3	181A		19	154				
M4	181E		22	27	32			
M5	1830		34	24				
M6	1837		42	47				
M7	183F		49	53				
M8	1841		50	51				
M9	1849		55	57				
M90	195E		204	221				
PIOCA	0082		4	14				
PIODA	0080		2	18	37	77	78	100	112
PIODB	0081		3	22	45	54		
SCAN1	0624		5	151				
SCOBF	1902		156	8	9	12	146	
SCORE	1900		155	11	132	147		
TAB1	1908		157	130				
TAB2	190F		161	96				
TAB3	1918		166	124				
TAB4	191F		170	108				
TAB5	1928		175	90				
TAB6	193D		186	143				
TAB7	194C		194	137				
TONE	195D		203	131	138	144		

PART NO.: 49.00710.001