5 PROGRAMMING THE DCE

5.1 STANDARD 8080 INSTRUCTION SET

93 Instructions which can be grouped as:

* Data transfer group: 13 instructions to move data between registers or between registers and memory.

* Arithmetic group: 20 instructions to add, subtract, increment or decrement data in register or memory.

* Logical group: 19 instructions to and, or, exclusive or, compare, rotate or complement data in registers or memory.

* Branch group: 29 instructions to conditionally, or unconditionally jump, call subroutine, or return from subroutines.

* Stack, I/O, machine control group: 12 instructions for input/ output, stack handling, and system control.

For a résumé of the 8080 instruction set see the table of Section 5. 5.

5.2 GIC AND TICC MACRO INSTRUCTIONS

A macro instruction is an indication to the assembler that a symbol appearing in the label field of a statement actually stands for a group of instructions. The following set of GIC and TICC commands are macro instructions, and their definition punched on a paper-tape must preceed the user source program during pass 1 of the MAC-80 assembler (it need not be used during passes 2, 3,or 4). Tapes with the GIC and TICC macro instructions are available to users of the DCE. For those users who do not have access to an assembler and wish to program directly with the HEX codes, a GIC and TICC macro expansion table is provided in Sections 5. 3 and 5. 4 on pages 5-3 and 5-4. The DCE-DM Development System Resident Assembler (UAE) automatically recognizes all the GIC and TICC macros and resolves the register address differences between the different DCE microcomputer modules.

5.2.1 GIC Macro Instructions

GICC	amode, bmode	Configure GIC to one of 63 possible mode combinations.
BSET	nbit	Set specified bit in GIC PORT 2
BCLR	nbit	Clear specified bit in GIC PORT 2
LDGI	nport	Load contents of specified GIC port to accumulator.
STGI	nport	Store contents of accumulator in specified GIC port.
LDGIS	0	Load contents of GIC PORT 0 to accumulator synchronized with handshake signals.
STGIS	0	Store contents of accumulator in GIC PORT 0 synchronized with handshake signals.

5.2.2 TICC Macro Instructions

STXMT		Store contents of accumulator in asynchronous transmitter buffer.
LDRCV		Load contents of asynchronous receiver buffer to accumulator.
STOUT		Store contents of accumulator in TICC output port.
LDIN		Load data present at TICC input port to accumulator.
STTIM	ntimer	Store contents of accumulator in specified timer and start timer.
STIMR		Store contents of accumulator in TICC interrupt mask register.
LDSTA		Load TICC status register to accumulator.
STTCM		Store contents of accumulator in TICC command register.
STCRR		Store contents of accumulator in communications rate register.
LDIRP		Load TICC interrupt pending register to acc.

5.3 GIC MACRO EXPANSIONS *

```
GICC    MACRO   AM, BM
        DB      3EH, AMx8+BM+80H, 32H, 03H, 1CH
        ENDM    MVI A,
                STA 1C03 H.
BSET    MACRO   NB
        DB      3EH, NBx2+1, 32H, 03H, 1CH
        ENDM    MVI A, NBx2+1
                STA 1C03H
BCLR    MACRO   NB
        DB      3EH, NBx2, 32H, 03H, 1CH
        ENDM    MVI A, NBx2
                STA 1C03 H
LDGI    MACRO   NP
        DB      3AH, NP, 1CH          LDA  1C(NP)H
        ENDM

STGI    MACRO   NP
        DB      32H, NP, 1CH          STA  1C(NP)H
        ENDM

LDGIS   MACRO
        DB      3AH, 0, 5CH          LDA  5C00H
        ENDM

STGIS   MACRO
        DB      32H, 0, 5CH        STA  5C00H
        ENDM
```

Note : The above macro expansions are applicable for DCE-1 and DCE-2. The GIC register addresses are different for DCE-X.

5.4 TICC MACRO EXPANSIONS*

```
STXMT   MACRO
        DB        32H, 6, 98H
        ENDM

LDRCV   MACRO
        DB        3AH, 0, 98H
        ENDM

STOUT   MACRO
        DB        32H, 7, 98H
        ENDM

LDIN    MACRO
        DB        3AH, 1, 98H
        ENDM

STTIM   MACRO     NT
        DB        32H, NT+8, 98H
        ENDM

STIMR   MACRO
        DB        32H, 8, 98H
        ENDM

LDSTA   MACRO
        DB        3AH, 3, 98H
        ENDM

STTCM   MACRO
        DB        32H, 4, 98H
        ENDM

STCRR   MACRO
        DB        32H, 5, 98H
        ENDM

LDIRP   MACRO
        DB        3AH, 2, 98H
        ENDM
```

Note: The above macro expansions are applicable for DCE-1 and DCE-2.

The TICC register addresses are different for DCE-X.

5.5 REPERTOIRE OF DCE INSTRUCTIONS

HEX CODES — ASSEMBLER FORM — FUNCTION — FLAGS

A580 Assembler

(hl)

MOVE GROUP

reg A	B	C	D	E	H	L	M	ASSEMBLER FORM	FUNCTION		S	Z	AC	P	CY
7F	78	79	7A	7B	7C	7D	7E	MOV A, reg	(A) ← (reg)	Ld [illegible], reg	-	-	-	-	-
47	40	41	42	43	44	45	46	MOV B, reg	(B) ← (reg)		-	-	-	-	-
4F	48	49	4A	4B	4C	4D	4E	MOV C, reg	(C) ← (reg)		-	-	-	-	-
57	50	51	52	53	54	55	56	MOV D, reg	(D) ← (reg)		-	-	-	-	-
5F	58	59	5A	5B	5C	5D	5E	MOV E, reg	(E) ← (reg)		-	-	-	-	-
67	60	61	62	63	64	65	66	MOV H, reg	(H) ← (reg)		-	-	-	-	-
6F	68	69	6A	6B	6C	6D	6E	MOV L, reg	(L) ← (reg)	l, reg	-	-	-	-	-
77	70	71	72	73	74	75	-	MOV M, reg	(M) ← (reg)	→ ld (hl), reg.	-	-	-	-	-

ACCUMULATOR GROUP

								ASSEMBLER FORM	FUNCTION	S	Z	AC	P	CY
87	80	81	82	83	84	85	86	ADD reg	(A) ← (A) + (reg)	+	+	+	+	+
8F	88	89	8A	8B	8C	8D	8E	ADC reg adc	(A) ← (A) + (reg) + (CY)	+	+	+	+	+
97	90	91	92	93	94	95	96	SUB reg	(A) ← (A) - (reg)	+	+	+	+	+
9F	98	99	9A	9B	9C	9D	9E	SBB reg sbc	(A) ← (A) - (reg) - (CY)	+	+	+	+	+
A7	A0	A1	A2	A3	A4	A5	A6	ANA reg And.	(A) ← (A) ∩ (reg)	+	+	0	+	0
AF	A8	A9	AA	AB	AC	AD	AE	XRA reg xor.	(A) ← (A) ⊻ (reg)	+	+	0	+	0
B7	B0	B1	B2	B3	B4	B5	B6	ORA reg or.	(A) ← (A) ∪ (reg)	+	+	0	+	0
BF	B8	B9	BA	BB	BC	BD	BE	CMP reg cp	(A) - (reg)	+	+	+	+	+

INCREMENT/DECREMENT REGISTER

								ASSEMBLER FORM	FUNCTION		S	Z	AC	P	CY
3C	04	0C	14	1C	24	2C	34	INR reg	(reg) ← (reg) + 1	INC reg	+	+	+	+	-
3D	05	0D	15	1D	25	2D	35	DCR reg	(reg) ← (reg) - 1	DEC reg	+	+	+	+	-

REGISTER PAIR GROUP

bc de hl sp af.

rp = bc, de, hl, sp

	rp B	D	H	SP	PSW	ASSEMBLER FORM	FUNCTION		S	Z	AC	P	CY
	03	13	23	33	-	INX rp	(rp) ← (rp) + 1	INC rp	-	-	-	-	-
	0B	1B	2B	3B	-	DCX rp	(rp) ← (rp) - 1	DEC rp	-	-	-	-	-
	0A	1A	-	-	-	LDAX rp	(A) ← ((rp))	Ld a, (rp)	-	-	-	-	-
	02	12	-	-	-	STAX rp	((rp)) ← (A)	ld (rp), a	-	-	-	-	-
add hl, bc –	09	19	29	39	-	DAD rp	(H,L) ← (H,L)+(rp)		-	-	-	-	+
	C5	D5	E5	-	F5	PUSH rp	((SP)-1) ← (rh), ((SP)-2) ← (rl), (SP) ← (SP)-2		-	-	-	-	-
	C1	D1	E1	-	F1	POP rp	(rl) ← ((SP)), (rh) ← ((SP)+1), (SP) ← (SP)+2		-	-	-	-	-

IMMEDIATE GROUP

	ASSEMBLER FORM	FUNCTION		S	Z	AC	P	CY
3E dd	MVI A, data	(A) ← data	ld reg, data.	-	-	-	-	-
06 dd	MVI B, data	(B) ← data		-	-	-	-	-
0E dd	MVI C, data	(C) ← data		-	-	-	-	-
16 dd	MVI D, data	(D) ← data		-	-	-	-	-
1E dd	MVI E, data	(E) ← data		-	-	-	-	-
26 dd	MVI H, data	(H) ← data		-	-	-	-	-
2E dd	MVI L, data	(L) ← data		-	-	-	-	-
36 dd	MVI M, data	(M) ← data		-	-	-	-	-
C6 dd	ADI data	(A) ← (A) + data	ADD a, data or data	+	+	+	+	+
CE dd	ACI data	(A) ← (A) + data + (CY)		+	+	+	+	+
D6 dd	SUI data	(A) ← (A) - data		+	+	+	+	+
DE dd	SBI data	(A) ← (A) - data - (CY)		+	+	+	+	+
E6 dd	ANI data	(A) ← (A) ∩ data		+	+	0	+	0
EE dd	XRI data	(A) ← (A) ⊻ data		+	+	0	+	0
F6 dd	ORI data	(A) ← (A) ∪ data		+	+	0	+	0
FE dd	CPI data	(A) - data		+	+	+	+	+
01 al ah	LXI B, addr	(B) ← ah, (C) ← al		-	-	-	-	-
11 al ah	LXI D, addr	(D) ← ah, (E) ← al		-	-	-	-	-
21 al ah	LXI H, addr	(H) ← ah, (L) ← al		-	-	-	-	-
31 al ah	LXI SP, addr	(SP_h) ← ah, (SP_1) ← al		-	-	-	-	-

DIRECT ADDRESS GROUP

	ASSEMBLER FORM	FUNCTION		S	Z	AC	P	CY
3A al ah	LDA addr	(A) ← (addr)	ld a, (address)	-	-	-	-	-
32 al ah	STA addr	(addr) ← (A)	ld (address, a)	-	-	-	-	-
2A al ah	LHLD addr	(L) ← (addr), (H) ← (addr + 1)		-	-	-	-	-
22 al ah	SHLD addr	(addr) ← (L), (addr + 1) ← (H)		-	-	-	-	-

A580

	HEX CODES		ASSEMBLER FORM	FUNCTION		FLAGS S Z AC P CY
	JUMP GROUP					
	C3 al ah		JMP addr	(PC) ← addr	JP addres	- - - - -
	C2 al ah		JNZ addr	If Z=0, (PC) ← addr	z, addes	- - - - -
	CA al ah		JZ addr	If Z=1, (PC) ← addr	—	- - - - -
	D2 al ah		JNC addr	If CY=0, (PC) ← addr	— nc, --	- - - - -
	DA al ah		JC addr	If CY=1, (PC) ← addr	— c, ..	- - - - -
	E2 al ah		JPO addr	If P=0, (PC) ← addr	— p, ..	- - - - -
	EA al ah		JPE addr	If P=1, (PC) ← addr	pe, ...	- - - - -
	F2 al ah		JP addr	If S=0, (PC) ← addr	p, ...	- - - - -
	FA al ah		JM addr	If S=1, (PC) ← addr		- - - - -
	E9		PCHL ↳ JP (hl)	(PC_h) ← (H), (PC_l) ← (L)	mp, ... m,	- - - - -
	CALL GROUP					
	CD al ah	CALL ...	CALL addr	(TOS) ← (PC), (PC) ← addr		- - - - -
	C4 al ah		CNZ addr	If Z=0, (TOS) ← (PC), (PC) ← addr		- - - - -
	CC al ah		CZ addr	If Z=1, (TOS) ← (PC), (PC) ← addr		- - - - -
	D4 al ah	zie JP	CNC addr	If CY=0, (TOS) ← (PC), (PC) ← addr		- - - - -
	DC al ah		CC addr	If CY=1, (TOS) ← (PC), (PC) ← addr		- - - - -
	E4 al ah		CPO addr	If P=0, (TOS) ← (PC), (PC) ← addr		- - - - -
	EC al ah		CPE addr	If P=1, (TOS) ← (PC), (PC) ← addr		- - - - -
	F4 al ah		CP addr	If S=0, (TOS) ← (PC), (PC) ← addr		- - - - -
	FC al ah		CM addr	If S=1, (TOS) ← (PC), (PC) ← addr		- - - - -
	RETURN GROUP					
	C9		RET	(PC) ← (TOS)		- - - - -
	C0	ret	RNZ	If Z=0, (PC) ← (TOS)		- - - - -
	C8		RZ	If Z=1, (PC) ← (TOS)		- - - - -
	D0	zie JP	RNC	If CY=0, (PC) ← (TOS)		- - - - -
	D8		RC	If CY=1, (PC) ← (TOS)		- - - - -
	E0		RPO	If P=0, (PC) ← (TOS)		- - - - -
	E8		RPE	If P=1, (PC) ← (TOS)		- - - - -
	F0		RP	If S=0, (PC) ← (TOS)		- - - - -
	F8		RM	If S=1, (PC) ← (TOS)		- - - - -
	RESTART GROUP					
	C7		RST 0	(TOS) ← (PC), (PC) ← 0_{16}		- - - - -
	CF		RST 1	(TOS) ← (PC), (PC) ← 8_{16}		- - - - -
	D7		RST 2	(TOS) ← (PC), (PC) ← 10_{16}		- - - - -
	DF		RST 3	(TOS) ← (PC), (PC) ← 18_{16}		- - - - -
	E7		RST 4	(TOS) ← (PC), (PC) ← 20_{16}		- - - - -
	EF		RST 5	(TOS) ← (PC), (PC) ← 28_{16}		- - - - -
	F7		RST 6	(TOS) ← (PC), (PC) ← 30_{16}		- - - - -
	FF		RST 7	(TOS) ← (PC), (PC) ← 38_{16}		- - - - -
	ROTATE/CONTROL/ SPECIAL GROUP					
rlca	07		RLC	(A_{n+1}) ← (A_n), (A_0) ← (A_7), (CY) ← (A_7)		- - - - +
	0F	rrca	RRC	(A_n) ← (A_{n+1}), (A_7) ← (A_0), (CY) ← (A_0)		- - - - +
rla	17		RAL	(A_{n+1}) ← (A_n), (A_0) ← (CY), (CY) ← (A_7)		- - - - +
rra	1F		RAR	(A_n) ← (A_{n+1}), (A_7) ← (CY), (CY) ← (A_0)		- - - - +
	00		NOP	No operation		- - - - -
halt.	76		HLT	Processor stopped until interrupt or reset.		- - - - -
	F3		DI	Interrupts disabled		- - - - -
	FB		EI	Interrupts enabled after next instruction		- - - - -
	E3	~~ex~~ hl, (sp)	XTHL	(L) ↔ ((SP)), (H) ↔ ((SP)+1)		- - - - -
	F9	ex hl, (sp)	SPHL	(SP) ← (H) (L)		- - - - -
	EB	ex hl, de	XCHG	(H) ↔ (D), (L) ↔ (E)		- - - - -
daa.	27		DAA	Decimal adjust accumulator		+ + + + +
cpl	2F		CMA	(A) ← $(\overline{A})$		- - - - -
scf	37		STC	(CY) ← 1		- - - - 1
ccf	3F		CMC	(CY) ← $(\overline{CY})$		- - - - +

MACRO's.

HEX CODES	ASSEMBLER FORM	FUNCTION	FLAGS				
			S	Z	AC	P	CY
TICC GROUP							
32 06 98	STXMT	(Transmit buffer) ← (A)	-	-	-	-	-
3A 00 98	LDRCV	(A) ← (Receive buffer)	-	-	-	-	-
32 07 98	STOUT	(output port) ← (A)	-	-	-	-	-
3A 01 98	LDIN	(A) ← (input port)	-	-	-	-	-
32 09 98	STTIM 1	(Timer 1) ← (A)	-	-	-	-	-
32 0A 98	STTIM 2	(Timer 2) ← (A)	-	-	-	-	-
32 0B 98	STTIM 3	(Timer 3) ← (A)	-	-	-	-	-
32 0C 98	STTIM 4	(Timer 4) ← (A)	-	-	-	-	-
32 0D 98	STTIM 5	(Timer 5) ← (A)	-	-	-	-	-
3A 03 98	LDSTA	(A) ← (TICC status)	-	-	-	-	-
32 04 98	STTCM	(TICC Command) ← (A)	-	-	-	-	-
32 05 98	STCRR	(Rate register) ← (A)	-	-	-	-	-
3A 02 98	LDIPR	(A) ← (Interrupt pending)	-	-	-	-	-
32 08 98	STIMR	(Intr. Mask reg.) ← (A)	-	-	-	-	-
GIC GROUP							
3E cd 32 03 1C	GICC am,bm	(GICC Cmd reg) ← cd;cd=80+8 * am+bm	-	-	-	-	-
3E cc 32 03 1C	BCLR n	(P2Bn) ← 0;cc=2*n	-	-	-	-	-
3E cs 32 03 1C	BSET n	(P2Bn) ← 1;cs=2*n+1	-	-	-	-	-
3A 0m 1C	LDGI m	(A) ← (Port m)	-	-	-	-	-
32 0m 1C	STGI m	(Port m) ← (A)	-	-	-	-	-
3A 0m 5C	LDGIS m	(A) ← (Port m)	-	-	-	-	-
32 0m 5C	STGIS m	(Port m) ← (A)	-	-	-	-	-

5.6 INITIALIZATION OF THE DCE

po IT!

The DCE contains an automatic power-on reset feature. When power is applied to the DCE, the following conditions are established:

- 8080 CPU Program Counter is set to zero.
- 8080 CPU Interrupts are disabled.
- All GIC ports are in input mode (high impedance state).
- All TICC registers contain random values.
- All RAM memory locations contain random values.
- 8080 CPU Registers, Accumulator, Flags, and Stack Pointer contain random values.

After Reset, the program will start executing, starting at memory address 0000 (F000 for DCE-LSA systems). The first few program steps must load the control registers of the GIC and TICC with appropriate values in order to set up the desired DCE configuration.

The TICC device does not have a hardware Reset signal. It must be Reset by software, via Bit 0 of its Command Register. After Power-on reset, this reset bit in the TICC Command Register may not necessarily be zero. It must be cleared, and then set to ensure correct TICC Reset.

The following steps must be included in the initialization sequence of the users program (unless program is entered via DCE Utility Program):

1. Clear TICC Interrupt Mask Register to zero.
2. Clear TICC Command Register to zero.
3. Delay 1 second (nominal) while the Reset line settles for Real-World interface cards connected to the DCE-BUS. This may be omitted if no Real-World cards are present.

4. Initialize the TICC by loading the TICC Command Register with a value having Bit 0 equal to 1 and Bits 4 to 7 equal to zero. Select Bits 1,2,3 for conditions desired, (see Section 3. 4. 2).

5. Load the TICC Communications Rate Register to set the desired Baud rate and the number of stop bits (see Section 3. 4. 3).

6. Load the TICC Interrupt Mask Register to enable desired interrupts (see Figure 6-2). Logical one in a bit position enables the corresponding interrupt.

7. Configure the GIC for the desired I/O mode (see Section 2. 7. 2).

8. Initialize the Stack Pointer. Load a value one greater than the highest RAM location (1200H for DCE-1 and 1800H for DCE-2). The first byte pushed on to the Stack will then be saved in the highest RAM location, since the Stack Pointer is decremented before data or an address is pushed on to the Stack.

9. Enable 8080 CPU interrupts, if desired.

5.6.1 A Typical DCE Start-Up Sequence

The following is a typical initialization procedure starting at address 0000.

```
          JMP     INIT
          .
          .
          .
INIT:     XRA     A
          STIMR              zero interrupt mask register.
          STTCM              zero TICC command register.
                             delay about 1 second
          LXI     H,0FFFFH
DELAY:    LXI     D,0FFFFH   D,E = -1
          DAD     D          HL=HL+DE
          JC      DELAY
                       0000 1101
          MVI     A,0DH      reset TICC, select IN7 and interrupt
                             ack. enable
          STTCM
          MVI     A,1        select 110 Baud, 2 stop bits
          STCRR
          MVI     A,0C0H     enable IN7 and timer 4 interrupts
          STIMR
          GICC    1,0        configure GIC ports
          LXI     SP,1800H   initialize Stack Pointer (DCE-2)
          EI                 enable CPU interrupts
          .
          .
          .
```

After execution of this program segment, the TICC will be configured for operation with interrupts. IN7 is selected instead of Timer 5 as the interrupt 7 source. The serial I/O is programmed for 110 Baud with 2 stop bits. The interrupt mask register is programmed to pass interrupt no. 6 and 7 (see Figure 6-2). GIC Ports 0,1,2 (B0-B3) are configured as output, Port 2 (B4-B7) as input.

The first instruction at address 0000 in the previous example program is a Jump. It is necessary because address space from 0000 to 0008 (in PROM) is the interrupt 0 vector area. If none of the interrupts are used, this Jump instruction is not necessary, and the program could start directly at address 0000.

If Timer 1 is to be used, the program segment starting at address 0000 must first determine whether a power-on reset or a time-out has occurred. It can then jump to the initialization routine or Timer 1 service routine accordingly. One possible technique for realizing such a scheme is explained at the end of Section 3.2.2.

Note: All DCE Utility programs perform the DCE start-up procedure during initialization. If, subsequently, the user program performs a software reset of TICC via the reset bit (bit 0) of the TICC Command Register, then all other TICC registers must be re-initialized to desired values.

5.7 PROGRAMMING DCE FOR REAL-TIME MESSAGE TRANSFER

5.7.1 INTRODUCTION

One of the most important features of the DCE is the interrupt controlled serial communications I/O circuits. These circuits allow the DCE to input and output variable length messages on a character-by-character basis simultaneously, while executing programs that realize the work for which it is directed. In other words, the DCE can communicate in real time with other computers, remote printers, data acquisition devices and with other DCE cards.

In a typical application the DCE must control an isolated process, collect data, and transmit messages to a central computer system for further data processing. In turn, it must receive the control parameters from the central system to adjust its control function accordingly. In such an application both the serial input and output must simultaneously co-exist. Also, the length of the input and output messages can be variable and under the control of both the programmer and the input source. Furthermore, the DCE must continue its control function to which it is dedicated.

5.7.2 ARCHITECTURE

To realize simultaneous message transfer and control program execution, the TICC must be programmed to interrupt the CPU each time a character has been sent and each time a character has been received. This can be realized by setting up the TICC mask register to pass interrupts 4 and 5 (load logical ONE into bit positions 4 and 5) and starting the interrupt service routines in the corresponding PROM vector area (section 3.2.1 on page 3-10).

The routines needed to realize this application are: Message Initialization routine, Input Driver Routine and Output Driver Routine. The Message Initialization Routine is contained within the mainline control program. The Input and Output Driver Routines are interrupt driven.

5.7.2.1 Message Initialization Routine

The purpose of the message initialization routine is to initialize the control parameters needed by the Input and Output Driver Routines each time the main-line program desires to transmit a message or receive a message on the serial ports of the TICC. In our example the control parameters are: message length, first-character address of outgoing message, first-character storage address of incoming message.

An additional task of the Message Initialization Routine is to call the Output Driver Routine once each time a message unit is to be transmitted. This action causes the Driver to transmit the first character. The Driver will be activated by the TICC transmit-buffer-empty interrupt to transmit the rest of the characters.

5.7.2.2 Output Driver Routine

The task of the Output Driver Routine is to respond to the transmit-buffer-empty interrupt generated by the TICC, to send out the next character in the message string, to check if there are more characters to be transmitted, and to turn off the output message active flag that was set by the Message Initialization Routine.

5.7.2.3 Input Driver Routine

The task of the Input Driver Routine is to respond to the receiver-buffer-loaded interrupt generated by the TICC, to store the received character into the next sequential message storage location, to realize that the end of message has occurred,and to turn off the input message active flag that was set by the Message Initialization Routine.

5. 7. 3 DESIGN

5. 7. 3. 1 Message Initialization Routine

The Message Initialization Routine is initiated by the main-line program under execution in the DCE at a moment when it wishes to send a message or receive a message on the serial ports of the DCE. Illustrated in Fig.5 -1, page 5-15 is a flow diagram of the Message Initialization Routine.

When the main program is ready to send or receive a message string via the TICC serial port it must first check if the appropriate routine is free, or if it is still engaged in transmitting or receiving a previous message. The main program does this by examining the appropriate message active flag. If it is not set, the respective driver routines are free. In this case it places the address of the first string character and the message length into a predetermined location in the RAM memory. It then sets the appropriate message active flag and initiates execution of the first character.

5. 7. 3. 2 Output Driver Routine

The Output Driver Routine is initiated by recognition of the interrupt by the TICC after it has completed the output of a character. Illustrated in Fig.5 -2 , page 5-16 is a flow diagram of the Output Driver Routine.

The output routine sends the character from the address specified in the length register to the TICC transmitter register, it then checks if there are more characters to be sent and returns control to the interrupted program. While the TICC is shifting out the character, the DCE continues with the main program.

When the transmitter register is empty the TICC interrupts the CPU and returns control to the output driver routine for transmitting the next character. When all the characters have been transmitted the output routine resets the OUT MSG flag as an indication to the main program that it is possible to accept the next desired message. Note that when the TICC interrupts the CPU after the last character, the output routine, receiving the command, sees the OUT MSG flag reset and returns control to the main program immediately.

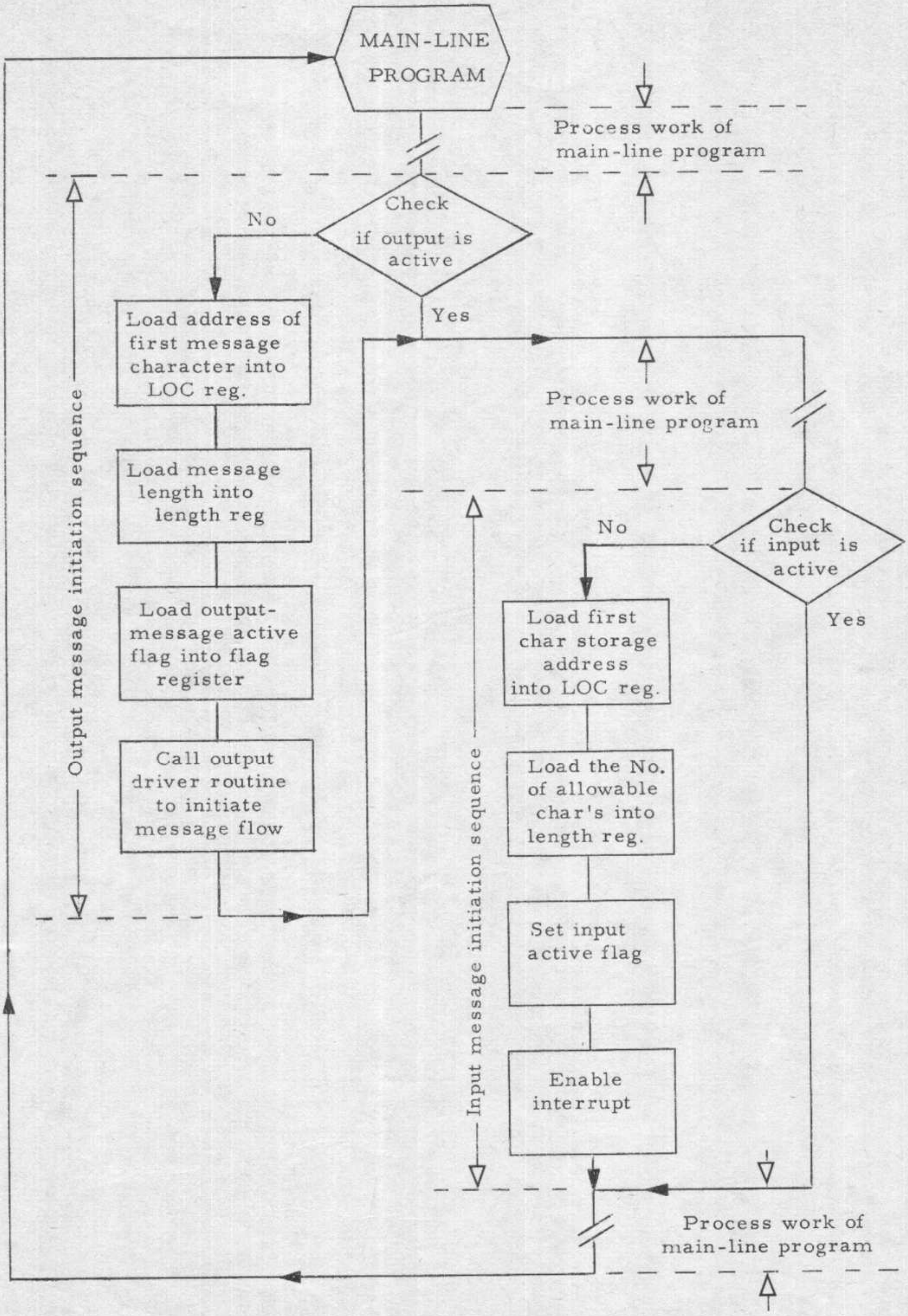
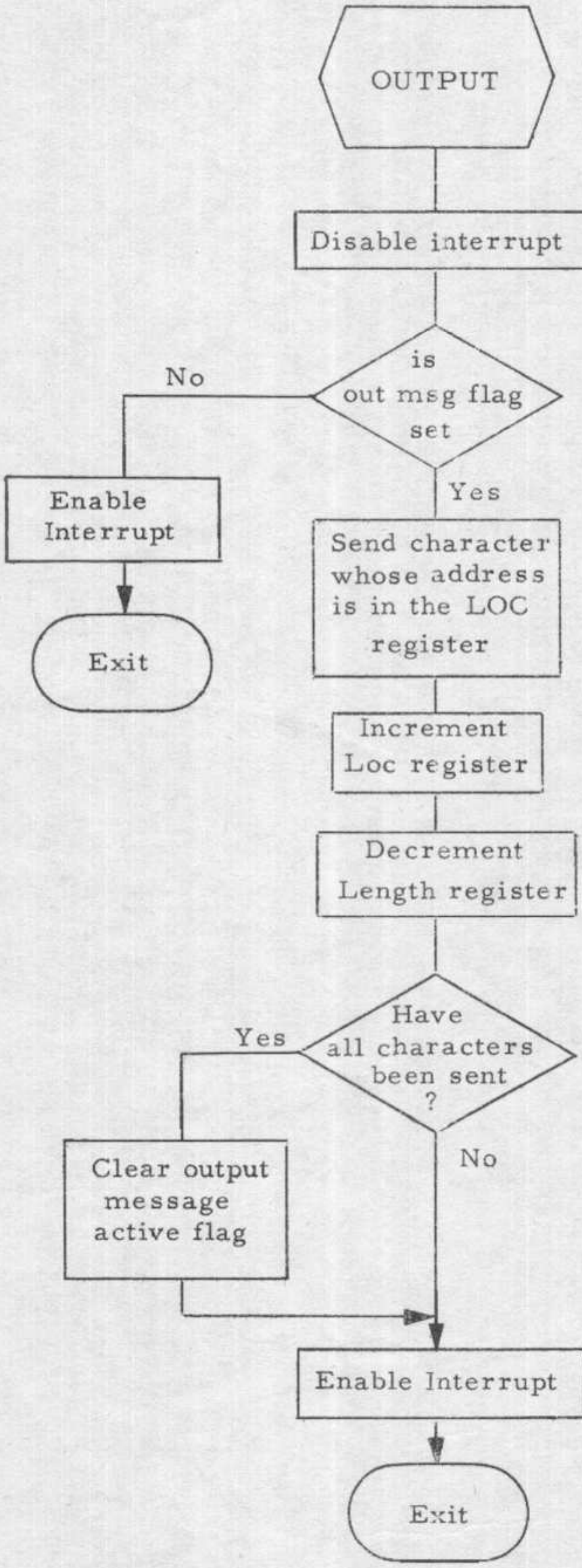

MAIN-LINE PROGRAM
Process work of main-line program
Check if output is active
No
Yes
Load address of first message character into LOC reg.
Load message length into length reg
Load output-message active flag into flag register
Call output driver routine to initiate message flow
Output message initiation sequence
Process work of main-line program
Check if input is active
No
Yes
Load first char storage address into LOC reg.
Load the No. of allowable char's into length reg.
Set input active flag
Enable interrupt
Input message initiation sequence
Process work of main-line program

Figure 5-1 Message Initialization Routine Flow Chart


OUTPUT
Disable interrupt
is out msg flag set
No
Yes
Enable Interrupt
Exit
Send character whose address is in the LOC register
Increment Loc register
Decrement Length register
Have all characters been sent ?
Yes
No
Clear output message active flag
Enable Interrupt
Exit

Location register (= M# BGN)

Figure 5-2 Output Driver Routine Flow Chart

5.7.3.3 Input Driver Routine

The Input Driver Routine is initiated by the recognition of an interrupt by the TICC after it has received a character from the serial input port. Illustrated in Fig. 5-3, page 5-18 is a flow diagram of the Input Driver Routine.

The input routine is called automatically by the TICC when it has received a character from the serial input port. It then checks to see if the characte is a special end-of-message character. (The end of message character ma be any character desired by the programmer, although it is not necessary to have it at all). If the character is the special end of message character, the input routine clears the Input -message-active flag and returns control to the main-line program. When the end of message has not been specified, the input routine stores the received character into the memory location indicated by the contents of the ILOC register and increments the ILOC register to set up the address for the next coming character of the message. The input routine then decrements the length register and checks to determine if all allowable characters have been received. In the event that this condition is true, the input routine clears the Input-message-active flag and transfers control to the main-line program. In the event that this is false it simply transfers control back to the main-line program.

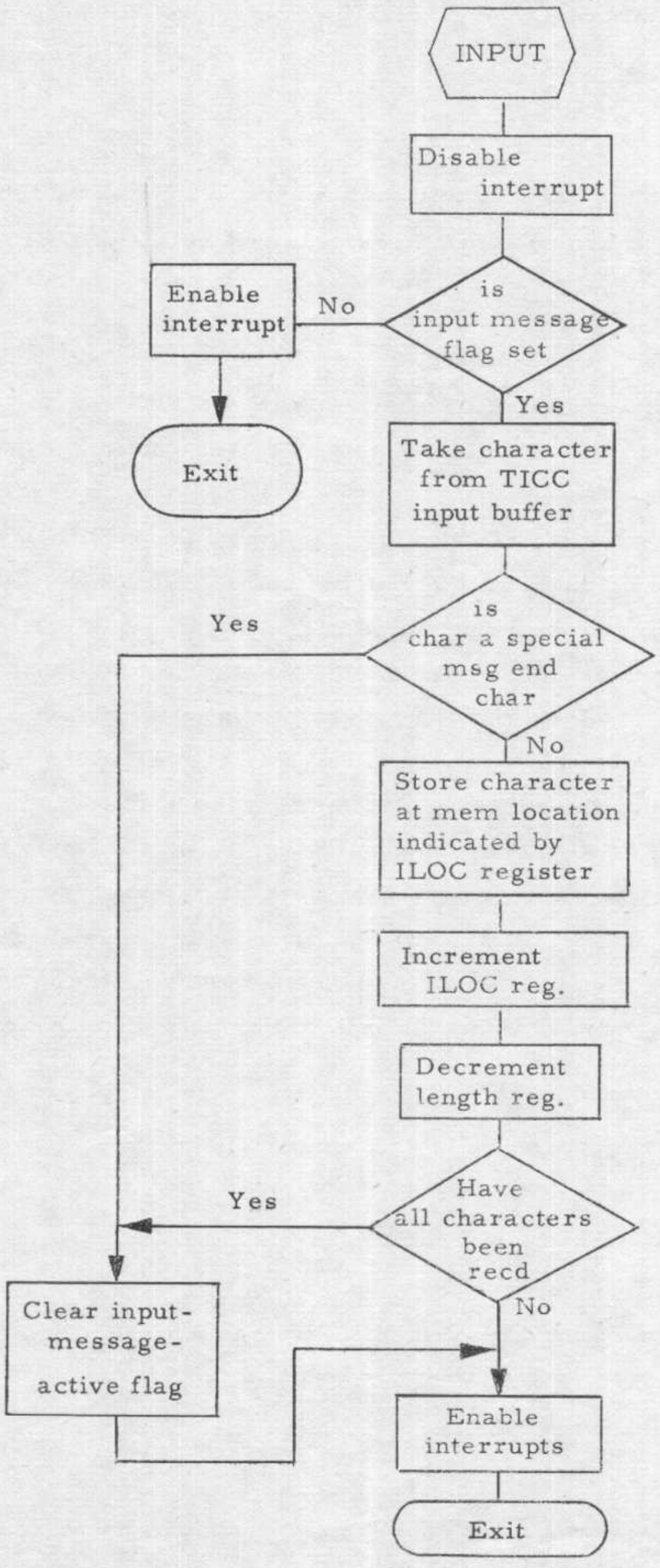

INPUT
Disable interrupt
is input message flag set
No
Enable interrupt
Exit
Yes
Take character from TICC input buffer
is char a special msg end char
Yes
No
Store character at mem location indicated by ILOC register
Increment ILOC reg.
Decrement length reg.
Have all characters been recd
Yes
No
Clear input-message-active flag
Enable interrupts
Exit

Figure 5-3 Input Driver Routine Flow Chart