TR-81-004

具有多用者編輯能力之智慧型前端機

贊助單位：本研究部份由資訊工業策進會資助，合約III-RD-69-002

執行單位：中央研究院資訊科學研究所

計劃主持人：柯志昇

協同主持人：洪永常

研究生：劉龍駿、鄭聖慶

執行期限：69.年 9 月 1 日～70.年 8 月31日

日　　期：中華民國七十年十月

具有多用者編輯能力之智慧型前端機

摘要：

這篇報告旨在描述一個以微處理機爲底具有多用者編輯能力的前端機之設計與建立。

在這個前端機內，軟體程式包含一個即時事件驅動多工控制核心程式，一個具有再出入性質的編輯器程式，以及一個負責與主機通信的通信程式。同時，我們設計一個檔案傳輸的通信協定，提供可靠而有效的主機與前端機間之通信。

Table of Contents

Design and Implementation of a Front End Processor with Multiuser Editing Capability

Abstract

This report describes the design and implementation of a microcomputer-based front end processor (FEP) which allows multiusers to edit files concurrently.

To provide such an environment with FEP, a real time event-driven multitasking executive has been designed. The application task (editor task) is coded in a reentrant form so that it can be shared by users. A file transfer protocol has also been designed to facilitate reliable and efficient communication between the host and the FEP.

1. Introduction

Fig.1.1 shows a conventional computer system. The users edit their files directly under the control of the host computer (The host computer can be a mini- or main-frame computer). Most of the time the editing task in the system is suspended and waiting for the user's input, since the user's input speed is much slower than the processing speed of the computer. If many users invoke the editing utility at the same time, there will be more tasks in waiting state. From the operating system's viewpoint, the more the tasks are in waiting state, the worse is the system performance, since the waiting task may occupy memory space, require swapping in-out between memory and disk to allow other task's execution, and spend the scheduling time of the operating system.

Moreover, the input of users from terminals to the host computer always induce some overhead for the system to process these inputs. This overhead can not be avoided.

Fig.1.2 shows a general software development procedure; the program is edited and then tested. If the testing result is correct then the procedure ends. If not, some modification has to be made to the program; that is, the program needs to be edited again. We can find that the editing work introduces a lot of terminal I/O. If we can move editing tasks from the host computer to a low cost front end processor (see Fig.1.3) then the overall system performance will be greatly improved. It is the purpose of this report to present a front end processor which allows multiusers to edit programs simultaneously.

In recent years, because of the hardware technique improvement and its cost reduction, the application capability of microcomputer has become more powerful, and the application area also becomes more widely than before. A new area is to use microcomputers for multiuser applications. In this report, we will present a software design model for multiuser editing application.

In order to save the program space, it is required that one editor program be shared by users. This could be accomplished by coding it in a reen-

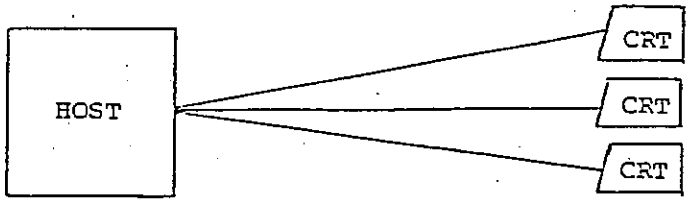


Fig.1.1

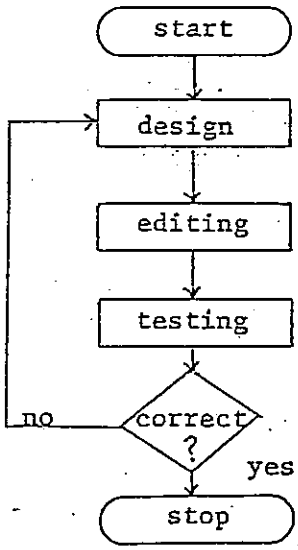


Fig.1.2

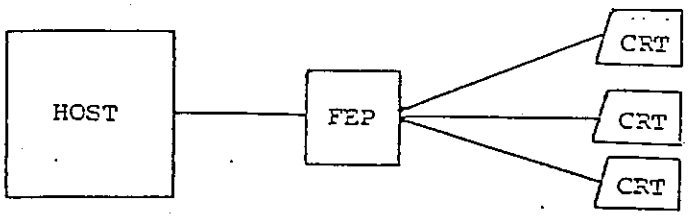


Fig.1.3

trant from. In this report, we will also discuss several techniques of writing reentrant codes in a microcomputer.

Since the file to be edited has to be moved from the host to the FEP and back to the host again after editing is complete, this report finally addresses file transfer protocols between the host and the FEP.

2. System Overview

The hardware architecture is illustrated in Fig.2.1. There are only two boards included in this system, one is the MCB (micro-computer board) that has been modified to allow 16K bytes RAM and 16K bytes ROM, the other is the SIB (serial interface board) which provides four channels of terminal interfaces.

The software architecture of the system is illustrated in Fig.2.2. There is a control task in host computer. It handles all the file-transfer operations, including open, close, retrieval, and storage. It also communicates with the FEP via a physical link according to the file transfer protocol.

On the FEP side, the software is separated into three modules, and each module can be designed and implemented independently. This will reduce the system design cycle. The first module is an executive, its function is more or less the same as the kernel of an operating system.

The second module is a communication task, all the communication operations are processed here, such as message packing/unpacking, the execution of the file transfer commands, etc. It also performs some concurrency control over the editing tasks, for there may be more than one user requesting to retrieve or store files at the same time.

The third module is an editing task, it accepts the user's commands and processes these commands. It also issues a request to communication task to transfer files between the host and the FEP. Since the editing task is shared by several users, it is coded in a reentrant form.

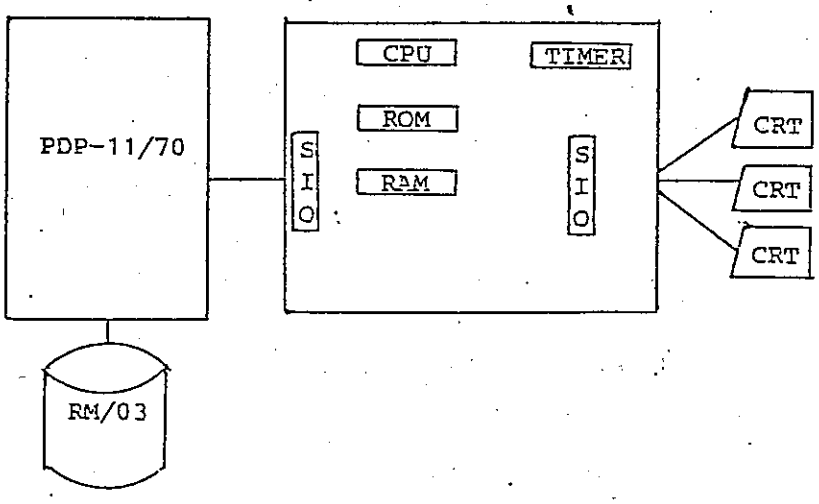

PDP-11/70
RM/03
SIO
CPU
ROM
RAM
TIMER
SIO
CRT
CRT
CRT

Fig.2.1

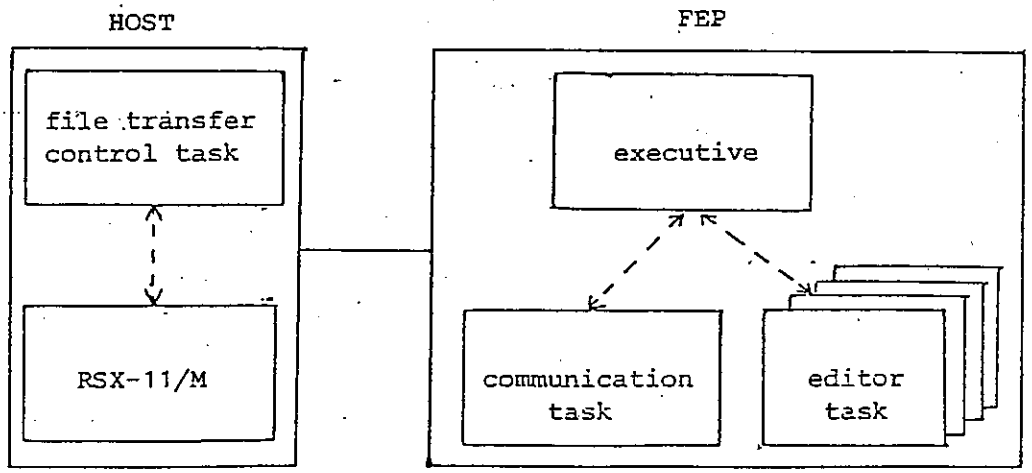

HOST
file transfer control task
RSX-11/M
FEP
executive
communication task
editor task

Fig.2.2

3. Hardware Construction

As a front end processor, a system with a minimum hardware cost seems to fit a least requirement. In the edit FEP, a minimum hardware configuration is considered, only two boards required. In this section we will describe their module functions.

MCB (the Zilog Z80-MCB) is a single board microcomputer designed to be adaptable to a wide range of OEM applications. The block diagram on the Fig.3.1 identifies the major components on the board, the heart of which is the Z80 microprocessor. Associated logic includes 4K bytes of dynamic RAM, provision for up to 4K bytes EPROM, both parallel and serial I/O ports, I/O ports decoders and a crystal controlled clock. The parallel port is implemented with the Z80-PIO with area reserved for user-applied driver and/or receiver logic. The Z80-CTC is used as a baud rate generator for the serial interface implemented with an 8251 USART.

SIB (the Z80-SIB) uses four 8251 USART devices to implement the serial communication channels. Two Z80-CTC devices are used to accommodate Z80 interrupt capability for receive and transmit operations for each bi-directional serial channel. A third Z80-CTC device is provided to accommodate programmable baud rates for each serial port from 50 to 9600 baud derived from an on-board crystal, the system clock, or an external clock.

The schematic diagrams of the two boards are supplied in appendix.

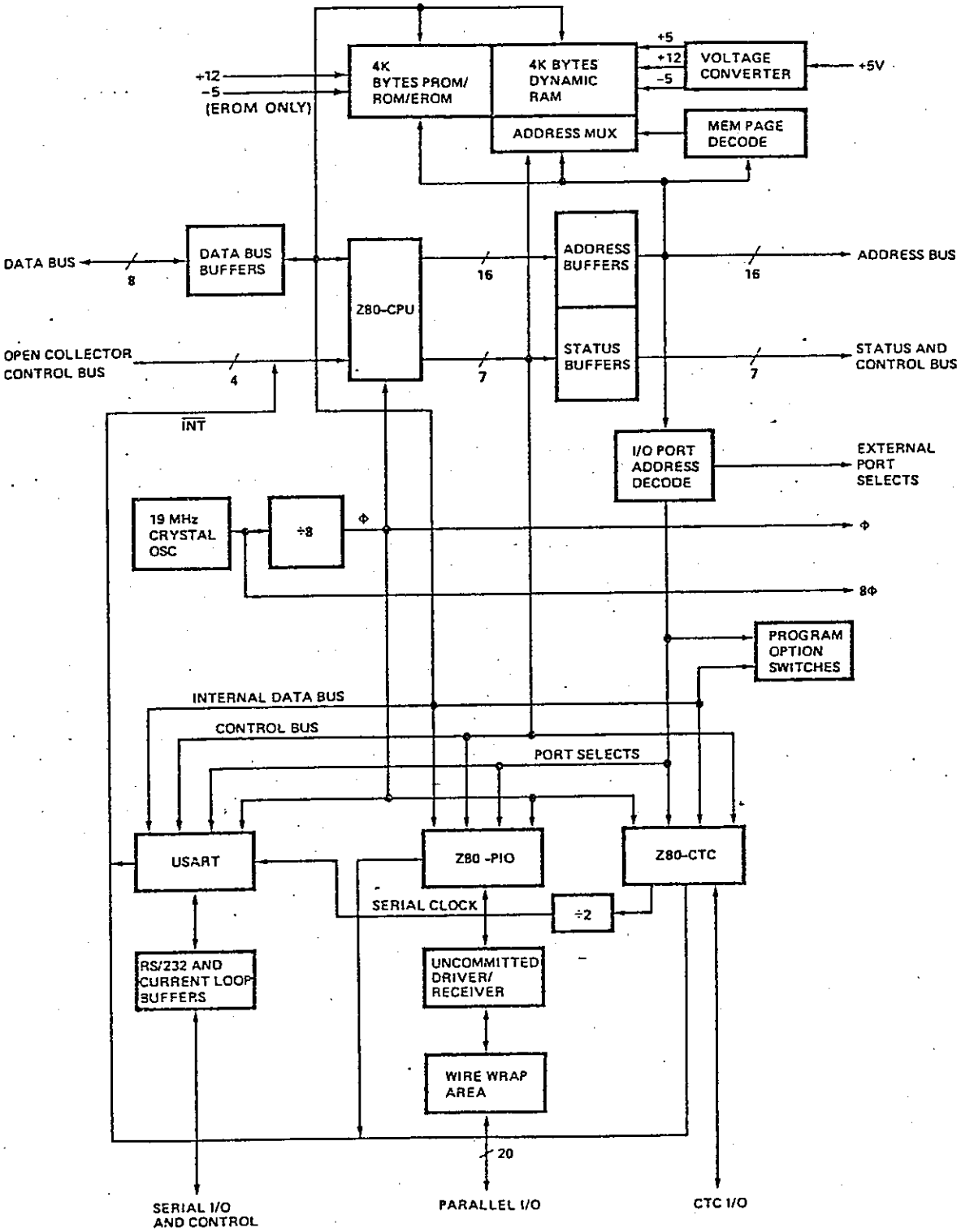
3.1 Hardware Modifications

There are several modifications over the board from its original deliveries to make our own system configurations. The primary modifications are through the 'jumpers' to select memory mapping and I/O addressing.

(1)MCB:

.RAM memory jumpers

We have changed the 4K RAM capacity to 16K RAM capacity



Z80-MCB BLOCK DIAGRAM

Fig.3.1

J1-10/J1-12,J1-9/J1-16,J1-8/J1-11

.RAM page decoding

The RAM is placed on the hex addresses 4000 - 7FFF.

J3-3/J3-9,J3-11/J3-12

.ROM page decoding

The ROM is placed on the hex addresses 0000 - 3FFF.

J3-3/J3-15,J3-13/J3-16

.serial interface interrupt jumper

The serial interface in MCB is used to communicate with host, it will generate interrupt when any receive or transmit operation is occured.

J1-13/J1-1

.serial interface jumpers

The serial interface mode is selected as follows: RS-232-C, MCB='MODEM', always 'clear to send', ignore 'request to send', signal swings from +12V to negative supply.

J4-1/J4-2,J4-9/J4-10,J4-5/J4-6,J4-14/J4-15,
J4-12/J4-11,J4-4/J4-7/J4-16,J5-1/J5-8

(2)SIB:

.port address range selection

The port address range is selected between hex 80 and 9F

J4-5/J4-16,J4-1/J4-7,J4-4/J4-6

.port of device selection

device	J1 jumper	port address
CTC0	1 - 12	80H - 83H
CTC1	2 - 11	84H - 87H
CTC2	3 - 10	88H - 8BH
USART 0	4 - 14	8CH - 8DH
USART 1		8EH - 8FH
USART 2	5 - 13	90H - 91H
USART 3		92H - 93H

.baud rate generation

The on board PHI/2 is used as clock source

J3-6/J3-11/J3-12/J3-13

.USART clock input

CTC CLK0 drives the T×C0,R×C0,T×C1,R×C	J2-2/J2-5/J2-6/J2-13/J2-14
CTC CLK1 drives the T×C ,R×C2	J2-3/J2-15/J2-16
CTC CLK2 drives the T×C3,R×C3	J2-4/J2-1/J2-12

4. A Real Time Event-driven Multitasking Executive

In order to accomplish the multiuser facility, a multitasking architecture is required to reduce the software design effort. Different modules can be separated into independent tasks, with a minimum coupling during system design stage. Moreover, we can assign different tasks to different users, in which different tasks may share the same program. In this system, we have designed a simple but useful executive (REMX), to provide a multitasking environment.

A task in the system is an independently executable program. Associated with each task is a task control block (TCB), which is used to maintain control information about the task, such as the program entry point, stack pointer, event control word, and some pointer fields about messages. Each task can be in one of the three states, namely RUN, READY, and BLOCK (WAIT). The state of a task is stored at the task status word (TSW) in TCB.

The communication between tasks is through the "message exchange channel". There are two types of channels in our system, one is the software channel for inter-task communication and the other is the I/O channel for task-device communication. A task may send a message to a channel or receive a message from a channel.

The synchronization between tasks and I/O devices is through an "event flag" mechanism. A task may send a message to a channel and declare an event flag for synchronization. By associating task operations with event flags, several operations can proceed concurrently and may be synchronized by the mechanism of event flags. Upon the occurrence of an event the processing of the current task may be continued or discontinued, depending on the dynamic requirements of the system.

There are five modules contained in the REMX :

(1) the task scheduler,
(2) the memory manager,
(3) the inter-task communication manager,
(4) the input/output communication manager,
(5) the real time manager.

The task scheduler manages the overall system, schedules tasks, and keeps track of the task status. The memory manager handles the memory allocation and deallocation. The two communication modules control the flow of messages among tasks. The real time manager maintains the real time clock and allows the creation of events based on timers.

There are several system primitives that users can issue them to active system functions. Fig.4.1 illustrates these system primitives.

4.1 The data structures of REMX

The whole system, under the monitoring of the REMX, has three basic data structures to achieve multitasking, channel communication, and message exchange facilities. Each of them, task, channel and message, is associated with a control block to maintain its relevant information, namely the TCB, the CCB, and the MCB. In this section the data structures will be presented.

(1) TCB (Task Control Block) :

word offset	TCB
0	TCBLINK
1	ISP
2	IPC
3	TSKNO | PRIO
4	reserved
5	STATUS
6	CCBLINK
7	MCBLINK
8	ECW
9	EBW
10	reserved

.TWAIT: wait a specified time slice

.MARKT: mark a time event flag

.WAITE: wait an event flag

.CMRT : cancel previously marked timer

.SEND : send a message from a channel

.SENDW: send and wait until the other party received

.RECV : receive a message from a channel

.RECVW: wait until a message arrived

.SIGNL: signal an event flag

.IO : issue an I/O request

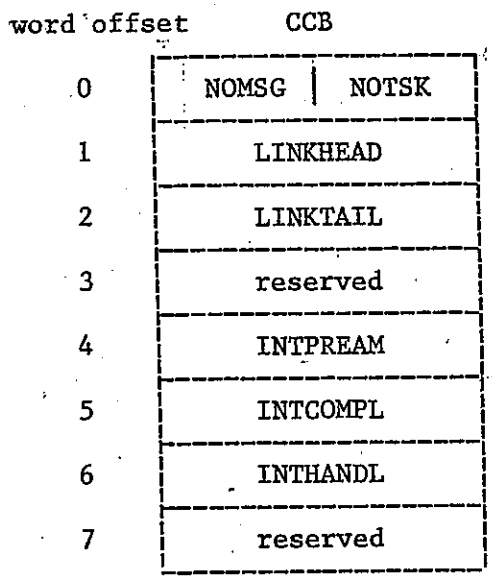
.IOW : issue an I/O request and wait until I/O complete

.CNRIO: cancel the previously issued I/O request

Fig.4.1

TCBLINK: link pointer to other TCB with same priority
ISP : initial stack pointer
IPC : initial program entry point
TSKNO : task number
PRIO : priority level number
STATUS : task status word
CCBLINK: link to CCB while blocked for channel
MCBLINK: link to MCB while receiving a message
ECW : event control word, 1 bit per event
EBW : event block word, 1 bit per event

(2)CCB (Channel Control Block):



NOMSG : number of messages available on the channel
NOTSK : number of tasks waiting for message
LINKHEAD : message queue head pointer
LINKTAIL : message queue tail pointer
INTPREAM : interrupt preamble routine address
INTCOMPL : interrupt completion routine address
INTHANDL : interrupt handler routine address

(3)MCB (Message Control Block):

word offset	MCB
0	TYPE | ECB
1	MCBLINK
2	TCBLINK
3	MSGPTR
4	MSGLTH
5	reserved

TYPE : user supplied message type
ECB : event control byte
MCBLINK: link pointer to next MCB in CCB message queue
TCBLINK: link pointer to the sending TCB
MSGPTR : message buffer pointer
MSGLTH : message length

4.2 The Control Flow and Algorithm of the REMX

The software of REMX is structured and shown as Fig.4.2

(a) INIT : system initialization routine
(b) TSKMGR : task manager
(c) COMMGR : inter-task communication manager
(d) TIMMGR : timer manager
(e) IOMGR : input/output device handling manager

4.2.1 INIT

The INIT module initilizes the whole system including system parameters, TCB, CCB and IOCCB.

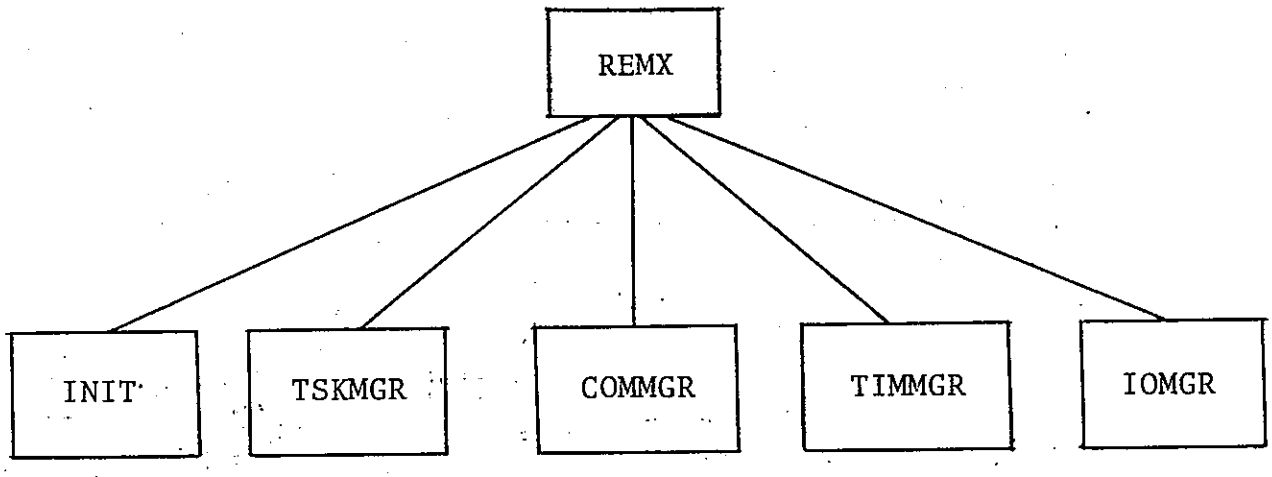


Fig. 4.2

(1) System parameters and interrupt mode initialization

.load system stack pointer

.clear all system parameters to zero

.set all list pointer to -1(null)

.set interrupt mode = 2 (indirect mode)

.set up interrupt vector table

(2) TCB initialization

.get TCB address in TCBD (initial task descriptor)

.if TCB address ≠ -1 (null)

then call BLDTCB to build up this TCB

call SCHTSK to insert it into ready task queue list

(3) CCB initialization

.get CCB address in CCBD (initial channel descriptor)

.if channel no. ≠ -1 (null)

then insert CCB pointer into CCBLST

clear all parameters in CCB

set all pointers in CCB to -1 (null)

(4) IOCCB initialization

.get CCB address from IOCCBD (initial I/O channel descriptor)

.if channel ≠ -1 (null)

then insert CCB pointer into IOLST

.clear all parameters in CCB

.set all pointers to -1 (null)

.get three I/O process routine address and save in IOCCB

IO_PRM (preamble routine)

IO_COM (completion routine)

IO_HDL (handler routine)

.set IO_HDL in interrupt vector table

(5) user's initialization

.call user's start routine

(6) system start

.transfer to dispatcher

4.2.2 TSKMGR

The task manager performs the task scheduling, task switching, and the system states entering or exiting.

(1) TSKMG: preempt the current task and reschedule again

```
.save all registers in user's stack area
.change into system state
.if active task is a null task (all tasks are blocked)
    then transfer to dispatcher
.save stack pointer in active TCB
.reload system stack pointer
.check task status word
.if TSW is blocked
    then transfer to dispatcher
.if TSW is ready
    then call ENQUE to insert the task into ready queue
    if priority   active task priority
      then ACTPRI = priority
.transfer to dispatcher
```

(2) DISPCH (task dispatcher):

```
.for I = MAXPRI (max priority) to 0 (lowest priority)
  do
     if the ready task queue is not empty (not -1 null)
        then
            ACTPRI = I (active task priority)
            SYSPRT = 0 (preemption flog = 0)
            call DEQUE to deque this TCB from ready queue
            ACTTNO = task number (active task number)
            ACTTCB = pointer of TCB (active TCB pointer)
            reload user's stack pointer
            restore all user's registers
            transfer control to user (dispatch the CPU to user)
      end if.
```

.set ACTTNO = -1 (null task)

ACTPRI = 0 (lowest priority)

.reload system stack pointer

(3) ENQUE (enque a node into a list)

entry condition : HL : queue tail pointer

DE : enqued node pointer

.set enqued node's next pointer to null (-1)

.(HL)<-DE save enqued node pointer in queue tail

.save enqued node pointer in the last tail node's next pointer

.if the queue is empty before insertion

then set the queue head = the enqued node

(4) DEQUE: deque a node from a list

entry condition: HL : queue head

exit condition : DE : the dequed node

.DE<-(HL), get the head pointer to DE

.(HL)<-next node pointer of queue head

.if the queue is empty after deque

then set the queue tail to -1 (null)

(5) ENTSYS: entering system state

functions : all system routines must call this routine to set into system state and a return address on stack, after returning from system routine, it will transfer to RETSYS

.set SYSTAT = 1 (system state flag)

.push RETSYS onto stack

(6) RETSYS: returning from system state

functions : after system routine's process there may be some preemption condition occurs, so must be checked here

.set SYSTAT = 0 (clear system state flag)

.if SYSPRM = 1 (any preemption occurs)

then transfer to TSKMG (task manager)

else return to user's routine

(7) INTSYS: entering interrupted system state

.increase interrupt level count

.push DETSYS onto stack

(8) DETSYS: returning from interrupted system state

.decrease interrupt level count

.if interrupt level count ≠ 0

then return

else check system state and preemption flag

if in user's state and preemption occurs

then transfer to TSKMG

4.2.3 COMMGR

The COMMGR module manages the inter-task communication, task synchronization, message sending/receiving, and event flag mechanism.

(1) SEND: send a message to a channel

.call ENTSYS to enter system state

.save event number in MCB_ECB

.get CCB address about the channel number

.increasing message number (NOMSG=NOMSG+1)

.decreasing task number (NOTSK=NOTSK-1)

.if there is a task waiting for message (NOTSK > 0)

then if not a send wait request

then call SETECW to set event control word

call DEQUE to deque the MCB from the list

call SCHTSK to insert it in the ready queue

else call ENQUE to enque the MCB into message list

save TCB in MCB's sending TCB

if not a send wait request

then call SETECW to set event control word

else set block status

transfer to TSKMG

.return

(2) RECV: receive a message from a channel

.call ENTSYS to enter system state

.call GETCCB to get the CCB address

.NOMSG=NOMSG-1

.NOTSK=NOTSK+1

.if a message is available (HOMSG > 0)

then call DEQUE to get an MCB from message queue

if the message is sent by SENDW

then call SCHTSK to schedule the sending task

else return

else if this is a receive wait request (RECVW)

then call ENQUE to insert the TCB into CCB

call BLKCHL to wait until message is available

return the MCB pointer to user

else NOMSG=NOMSG+1

NOTSK=NOTSK-1

return

(3) EVENTF: check event flag condition while an event is occured

.call GETECW to get the event mask

.if any block condition is met for this event

then clear EBW in TCB

else set zero flag

.return

(4) SIGNAL: signal a significant event to a task

.get TCB number of that task

.calculate index to TCBLST

.get TCB address

.call EVENTF to check event state

.if that task is blocked for this event

then call SCHTSK to reschedule that task

else return

(5) WAITE: wait for some events and proceed to execute after any event occurs

.get ECW from TCB

.check event mask with ECW

.if no event has been occured

then call BLKCHL to wait until any event occurs

else calculate the event number that has been occurred

return event number in A to user

4.2.4 TIMMGR

The TIMMGR module maintains a system clock, synchronizes the task with time-based event, and handles the timer interrupt.

(1) TIMGER: declare an event flag under a time basis or wait until a time slice has elapsed.

.calculate index to TIMLST according to the task no.

.get previous time slice

.if previous slice = 0 (no timer request before)

then TIMCNT=TIMCNT+1 (TIMLST count)

.if a time wait request (TWAIT)

then call BLKCHL to wait a specified time delay

else call SETECW to set associated event flag

(2) CMRKT: cancel a previously requested timer event

.calculate index to TIMLST according to the task number

.get previous time slice

.if previous time slice ≠ 0 (mark time before)

then TIMCNT=TIMCNT-1

.if the associated event flag number is not zero

then call GETECW to get event mask

clear the event flag

else return

(3) TIMINT: system clock interrupt handle routine

```
.save all registers
.update the system time for
    second, minute, hour, date, and month
.if a second has been elapsed
    then do I=1 to TIMCNT (scan TIMLST)
            get time slice
            if time slice ≠ 0
               then decrease time slice
                    if time slice count down to zero
                       then TIMCNT=TIMCNT-1
                            call EVENTF to check event flag
                            call SCHTSK to reschedule the task
                    end do.
.restore all registers
.return from interrupt
```

4.2.5 IOMGR

The IOMGR module accepts the I/O requests from user's, enables the I/O channel, handles the I/O interrupt.

(1) IORQS: request an I/O message through an I/O channel

```
.call GETCCB to get IOCCB address
.save ECB in IOMCB
.RQSTNO=RQSTNO+1
.if the I/O channel is not active yet
    then calculate interrupt vector offset in INTVTB
         call user's preamble routine
.call ENQUE to link the new message to message queue
.save TCB address in IOMCB
.clear MSGCNT in IOMCB to zero
.if the ECB < 0 (IOW)
    then call BLKCHL to wait until I/O completion
    else call SETECW to set the associated event flag
.return
```

(2) CNRIO: cancel I/O request on an I/O channel

.call GETCCB to get IOCCB address

.if RQSTNO = 0 (no I/O request has been issued)

then return

else RQSTNO=RQSTNO-1

call DEQUE to remove the current IOMCB

mark an I/O aborted flag on ECB

if no any more I/O request

then call user's I/O completion routine

.return

(3) INTSV: interrupt save routine

.save all registers

.get IOCCB address

.if RQSTNO ≠ 0 (some I/O request has been issued)

then get IOMCB address from IOCCB

return MCB address to user interrupt handle routine

else return from interrupt

(4) INTEX: interrupt exit routine

.if I/O completion flag is not set

then return from interrupt

.RQSTNO=RQSTNO-1

.if RQSTNO = 0

then call user's I/O completion routine

.call DEQUE to deque the current IOMCB

.check the ECB in the IOMCB

.if ECB < 0 (block request)

then call SCHTSK to reschedule this task

else call EVENTF to check event flag condition

.set preemption flag

.transfer to TSKMG

5. File Transfer Protocol and Communication Task

In this system, the file to be edited is transferred between the host and the FEP. In the following we will define some basic mechanisms about data transfer as well as the set of protocol commands and responses.

Handling files over a communication system induces three problems:

(1) File identification
(2) Data representation within files
(3) Mechanisms for transferring the data

Since file naming schemes are different from one computer manufacturer to another, it is difficult to define a standard convention for different computers. In our system, the file identification scheme of the host computer is used. There are several entries in identifying a file, namely the directory name, the file name, and the file version number.

In general, a source file has a sequential file structure, its record length is variable and is stored line by line. A file may be very large, if we transfer one line at a time, the efficiency will be worse. A page transfer mechanism is introduced in this system. The maximum page length is 2K bytes. Each time a user wants to edit a file, a page is transferred from the host to the FEP. After the user has completed his editing on this page, the page is transferred back to the host, and the user may request the next page to edit or may close the file.

File transfer primitives are exchanged in an alternate command-response mode, they are used to initiate, control and terminate a file transfer. They are :

(1) F-OPEN <user ID> <file identification> <mark>
    H-OPEN <user ID> <response code> <status>
(2) F-LOADPAGE <user ID> <mark>
    H-LOADPAGE <user ID> <response code> <page length>
(3) F-STORPAGE <user ID> <page length> <mark length>
    H-STORPAGE <user ID> <response code> <status>
(4) F-CLOSE <user ID> <code>
    H-CLOSE <user ID> <status>

In front of each command there is a letter representing where this command is sent, an "F" for the FEP and an "H" for the host. The user ID represents the user number. The mark is set when any error occured and a retransmission is made.

When a user wants to edit a file, an F-OPEN command is sent to the host. The host will respond with the file status. Then the user requests to load a page by using F-LOADPAGE command. The host reads a page from the disk, and sends it to the FEP. After editing, the user issues an F-STORPAGE command and sends the whole page to the host. The load and store page are proceeded until the file is completely edited. Finally, a close file command (F-CLOSE) is then sent to the host.

The commands and responses between the host and the FEP shown in Fig.5.1. The dynamics of this file transfer protocol is also illustrated in Fig.5.2 using the Petri Net.

5.1 The Data Structure of the Communication Task

The communication task has to receive the command from the edit task and then communicates with the host via a physical link. It coorperates with two channels, one is a software channel that performs the inter-task communication with eidt task, the other is a hardware channel that allows messages to be transmitted and received through a serial interface between the host and the FEP.

(1) Message format from edit task :

There are four types of messages from the edit task. The detail data structure and function of every entry is described in the following paragraphs:

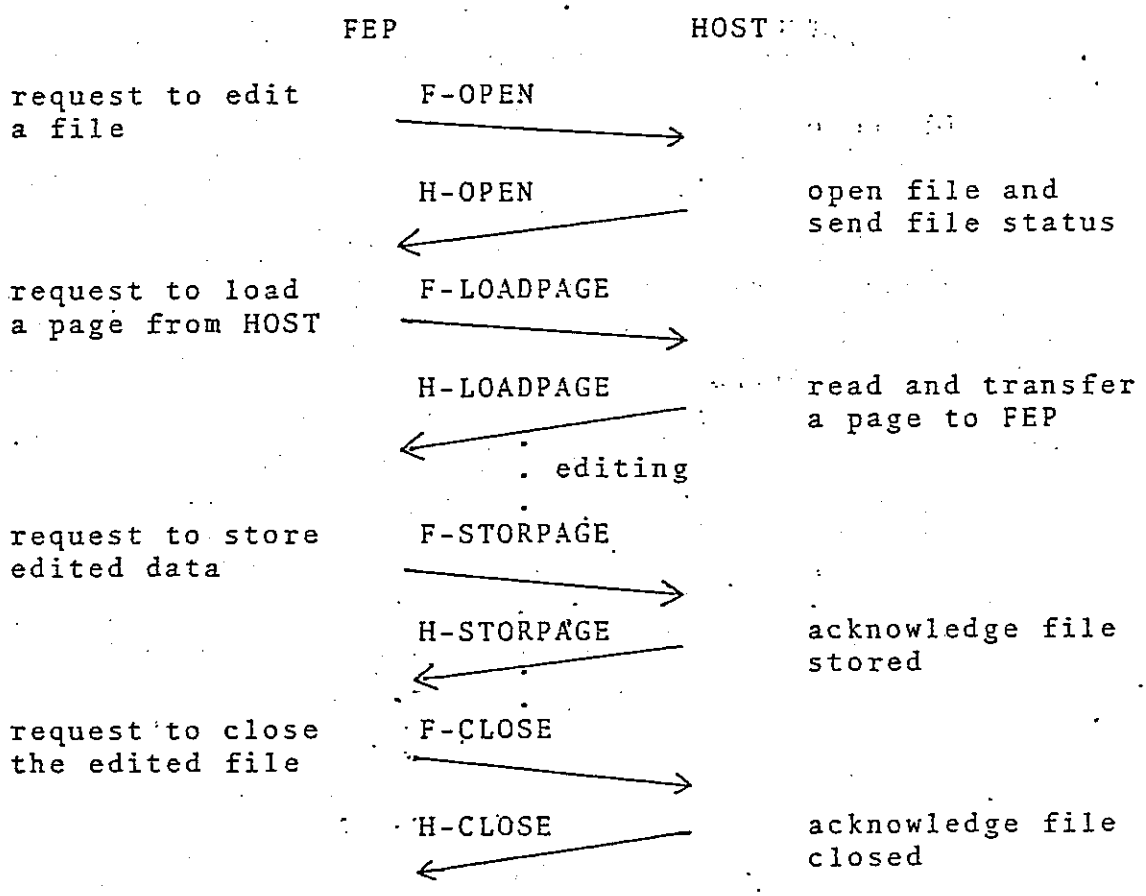

FEP
HOST
request to edit a file
F-OPEN
H-OPEN
open file and send file status
request to load a page from HOST
F-LOADPAGE
H-LOADPAGE
read and transfer a page to FEP
editing
request to store edited data
F-STORPAGE
H-STORPAGE
acknowledge file stored
request to close the edited file
F-CLOSE
H-CLOSE
acknowledge file closed

Fig.5.1

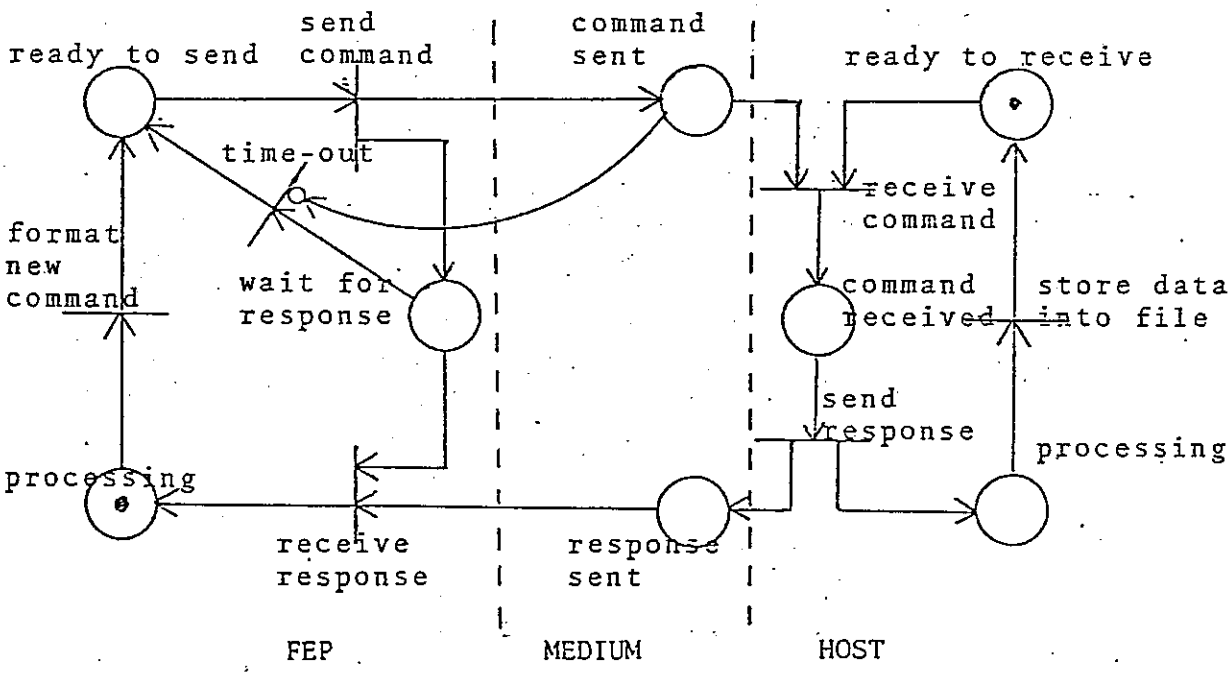
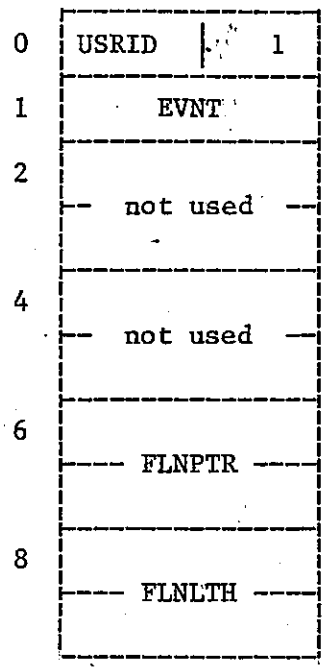

ready to send
send command
command sent
ready to receive
time-out
receive command
format new command
wait for response
command received
store data into file
send response
processing
processing
receive response
response sent
FEP
MEDIUM
HOST

Fig.5.2

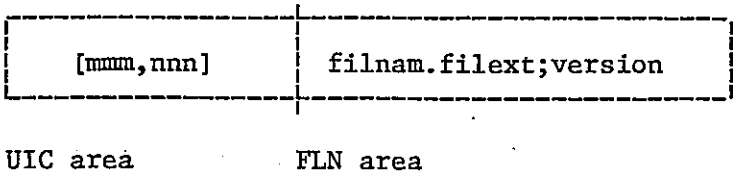
(i) OPEN file message:

MCB:

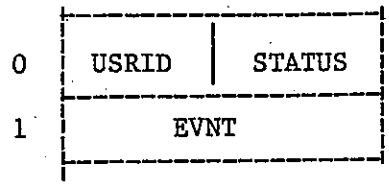


USRID : user ID number (currently 1-4)
EVNT : the associated event flag number
FLNPTR: pointer to the file name
FLNLTH: length in byte of the file name

file name format : two subfields



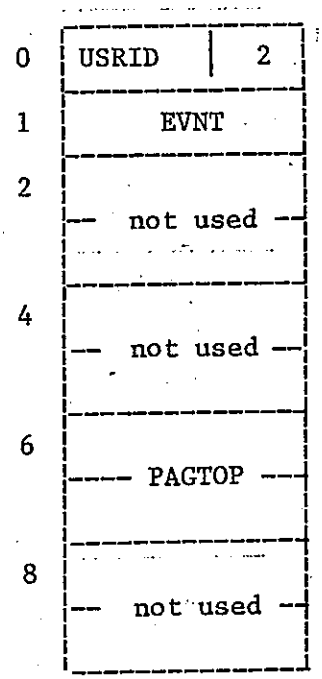
MCB: (response)



STATUS : 0: the file to be edited is existent
1: the file is not found, a new file is created
2: no such directory or protection failure
9: system error

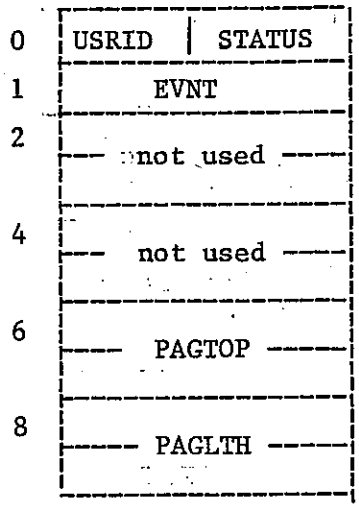
(ii) LOAD Page message:

MCB:



PAGTOP : the starting address of the page that data to be loaded

MCB: (response)

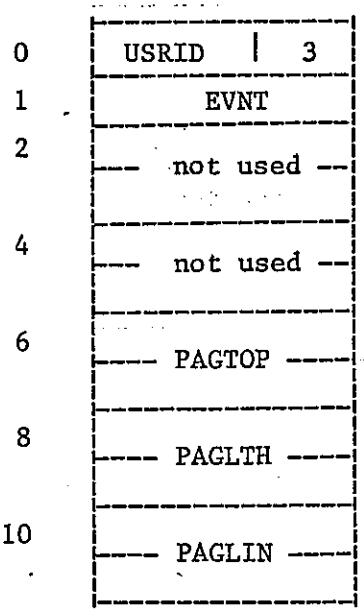


STATUS : 0: a new page is loaded
1: the last page is encountered
9: system error

PAGLTH : the length in byte of the loaded page

(iii) STORE PAGE message :

MCB:

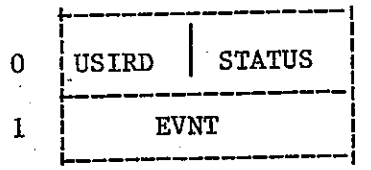


PAGTOP : starting address of the page to be stored

PAGLTH : length in byte of this page

PAGLIN : number of lines contained in this page

MCB: (response)



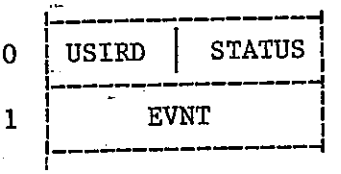
STATUS : 0: no error during the store

1: some error occurred

9: system error

(iv) CLOSE FILE message :

MCB:



STATUS : 0: file is closed successfully

9: error occurs

(2) format of commands to the host :

Since the messages sent to the host are through a RS232 interface, all the messages are in a character-oriented string format. The data transferred are in ASCII code, no binary data will be transferred. There are also four types of commands :

(i) OPEN FILE command :

offset (in character)	
1	USRID
2	1
3-6	FLNLTH
7-15	UIC
16-n	FLN
n+1	<CR>

(n<32)

response : 1 USRID
2 STATUS
3 <CR>

(ii) LOAD PAGE command :

command : 1 USRID
2 '2'
3 <CR>

response : 1 USRID
2 STATUS
3-6 FLNLTH (in bytes)
7 <CR>

After the FEP has received the status and page length without any error, a 'start to send' command will be sent to the host, then the host will send the whole page to the FEP.

(iii) STORE PAGE command :

command :	1	USRID
	2	'3'
	3-6	PAGLIN
	7	<CR>

The data of this page will be stored to the host in a hand-shaking mode, a line is transferred and wait for ACK, then next.

(iv) CLOSE FILE command :

command :	1	USRID
	2	'4'
	3	<CR>
response:	1	USRID
	2	STATUS
	3	<CR>

5.2 The Control Flow of the Communication Task

The software structure of the communication task is described in Fig.5.3. Only three levels of modules is designed in this task. The first level, The main program, receives commands from edit task, determine what type of command it is and then transfer to the next elvel to process these commands. The second level, the command process routine, sets up format of each command, talks with the host according to the file transfer protocol. The third level, the miscellaneous routines, handles the link level protocol, error recovery, binary and ASCII conversion, and message head packing/unpacking.

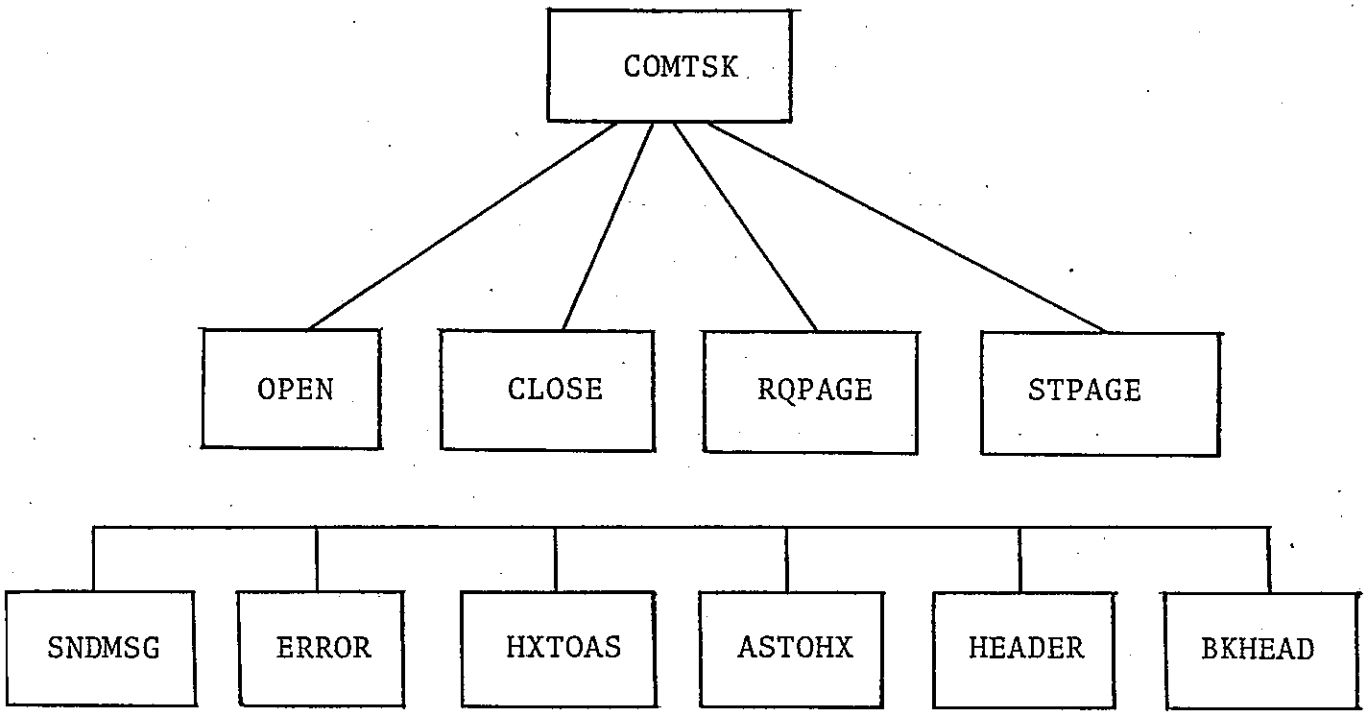

COMTSK
OPEN
CLOSE
RQPAGE
STPAGE
SNDMSG
ERROR
HXTOAS
ASTOHX
HEADER
BKHEAD

Fig.5.3

(1) COMTSK: main program of communication task

```
.receive and wait(MCB)      ;wait a message from edit task
.FUNC := MCB(1) and OF      ;get function code
.do case (FUNC)             ;check function type
    /1/ OPEN(MCB)          ;1- open file command
    /2/ RQPAGE(MCB)        ;2- request to load a page command
    /3/ STPAGE(MCB)        ;3- store a page command
    /4/ CLOSE(MCB)         ;4- close a file command
    else ERROR             ;others, error
.end
end COMTSK.
```

(2) OPEN(MCB): open file command process

```
.call HEADER to pack the message header
.MSGLTH := MCB(LTH)       ;get the message length
.call HXTOAS(MSGLTH)      ;convert the length to ASCII
.move file name to MSGCOM to set up message
.call SNDMSG(MSGCOM)      ;send it to the host
.call BKHEAD(MSGCOM)      ;unpacking the message head
.send(MCB)                ;send it back to edit task
end OPEN.
```

(3) CLOSE(MCB): close file command process routine

```
.call HEADER(MCB)         ;packing the message head
.call SNDMSG(MSGCOM)      ;send it to the host
.call BKHEAD(MSGCOM)      ;unpacking the message head
.send(MCB)                ;send it back to edit task
end close
```

(4) RQPAGE(MCB): load a page command process routine

```
.call HEADER(MCB)          ;packing the message head
.call SNDMSG(MSGCOM)       ;send the message
.LTHPTR := RCVMSG+2        ;get the page length pointer
.call ASTOHX(LTHPTR,PAGLTH)  ;convert the length to binary
.RCVMCB(PTR) := MCB(PTR)     ;page start address pointer
.RCVMCB(LTH) := PAGLTH       ;page length
.input(RCVMCB,EVN1)        ;request to load page and with event 1
.COMMCB(PTR) := ACKMSG     ;set up an ACK message to host
.COMMCB(LTH) := ACKLTH     ;
.output(COMMCB)            ;set an ACK to host
.markt(10,EVN2)            ;initiate a timer with event 2
.waite(EVN1+EVN2)          ;wait until any event occurred
.if event =EVN1
    then call BKHEAD(MSGCOM)
         MCB(LTH) := PAGLTH
         send(MCB)         ;send it back to edit task
    else call ERROR        ;time out
end RQPAGE.
```

(5) STPAGE(MCB): store page command process routine

```
.call HEADER(MCB)          ;packing the message head
.NOLINE := MCB(LTH+2)      ;get no. of line in the page
.call HXTOAS(NOLINE)       ;convert line no. to ASCII
.call SNDMSG(MSGCOM)       ;send it to host
.input (RCBMCB)            ;receive the response from host
.do I := 1 to NOLINE
    MSGPTR := next line in text
    MSGLTH : = length of this line
    COMMCB(PTR) := MSGPTR
    COMMCB(LTH) := MSGLTH
    call SNDMSG(MSGCOM)
  end
```

```
    .COMMCB(PTR) := ACKMSG      ;set up an ACK message
    .COMMCB(LTH) := ACKLTH      ;
    .call SNDMSG(MSGCOM)        ;send the ACK to host
    .call BKHEAD(MSGCOM)        ;unpacking the message head
    .send (MCB)                 ;send it back to edit task
  end STPAGE.
```

(6) SNDMSG: send a message to host and check with a timer

```
    .do I := 1 to RTXCNT        ;retransmit if errors
      output and wait(COMMCB)
      input(RCVMCB,EVN1)        ;wait for ACK with event 1
      markt(5,EVN2)             ;initiate timer with event 2
      wait(EVN1+EVN2)           ;check for any event
      if event = EVN1
        then return
      end
  end SNDMSG.
```

6. RSX-11/M Compatible Editor

In order to make the users have illusion that they are interacting with the host computer directly, the editor utility of the FEP should be compatible with the editor of the host computer. In other words, the FEP is transparent to the users.

In this system, the host is a PDP-11/70, with the RSX-11/M operating system. The editor commands are chosen from the command set of the RSX-11/M editor utility. All functions are compatible with the editor of the RSX-11/M system.

6.1 Reentrant Coding in a Microcomputer System

The editor program in the FEP has been coded in a reentrant form and can be shared by more than one user.

A reentrant code means that the program is independent of the impure data. If we want to have a program sharable, we must remove the impure data area from the program body, but the program can also access the impure data through some methods.

Among large computers, mini- or mainframe-computers, there are some hardware mechanisms to facilitate program sharing. For a demanded-page system, the program and the impure data can be put in different pages. For a segmented system, we can use segment registers to point to the program and the impure data area. What the designer has to do is just to declare where the program section is and where the data section is. But there is no such hardware mechanism in microcomputers, so new techniques must be developped.

We may add some registers to microcomputer as base registers for separating data dnd program. But this is not feasible to us, the cost will increase and the design period will take a longer time. Fortunately, almost every microprocessor has some index registers, we can use them to point to the impure data area. What we have to do is to put the impure

data area base into the index register of each user's TCB during the system initialization stage. Since each user is associated with a task, and a task has its own stack and register area(in its TCB), the program may just use registers and stack to access different users' data. This makes our program "pure" and sharable.

6.2 The Data Structure of the Impure Data Area

Since the edit task is reentrant, all the data used in the edit task must be separable between different tasks. We collect all the impure data into an area and this area is pointed by an index register, the edit task can access each user's data by this register. Associated with each user a 74 bytes data area is required to keep the data of editor utility. Its format and usage are descirbed in this section.

Name	Offset	Length	Usage
CTCCTL	0	1	CTC port number to initiate terminal baud
URTCTL	1	1	USART port number to terminal I/O
PAGHED	2	2	edited page head pointer
PAGEND	4	2	edited page last address pointer
LINBUF	6	2	input line buffer pointer
STATU1	8	1	status of edit command process
STATU2	9	1	status of edit command process
MCBARE	10	12	MCB area for communication with COMTSK
OPNFLG	22	1	file opened flag
NEWFLG	23	1	the edited file is a new file flag
EOF	24	1	end of file flag
LTHFLN	25	1	file name length
PAGTOP	26	2	page top pointer
PAGBOT	28	2	page bottom pointer
LNADD	30	2	line pointer in page
STRADD	32	2	string pointer in command line
NXSTAD	34	2	next token address

Name	Offset	Length	Usage
CMNORG	36	2	command number register
CHNGBC	38	2	length to page bottom during change
UIC	40	9	UIC area
FLN	49	15	file name area
BUFF1	64	2	buffer pointer
TABLTH	66	1	<TAB> field count
others	67	7	reserved for future expansion

6.3 The Control Flow of the Edit Task

The software structure of the edit task is shown in Fig.6.1. The main program is EDITCM, it receives an edit command from user, and determines what command it is, then transfers to that command process routine. After the command has been processed the control is then returned to EDITCM to wait for another command until an exit command is input.

(1) GETUIC: get user's identification code
- .clear UIC area to [000,000]
- .call PRTSTR to print the UIC message
- .call RDLIN to read the user's UIC
- .call NXTOKN to check the token length
- .call CHKUIC to check the first UIC
- .call NXTSTR to move to next token
- .call NXTOKN to check the token length
- .call CHKUIC to check the second UIC

(2) GETFLN: get the user's file name
- .clear FLN area to blank
- .call PRTSTR to print file name message
- .call RDLIN to read the user's file name
- .call NXTOKN to check the file name length
- .move the file name to the FLN impure area

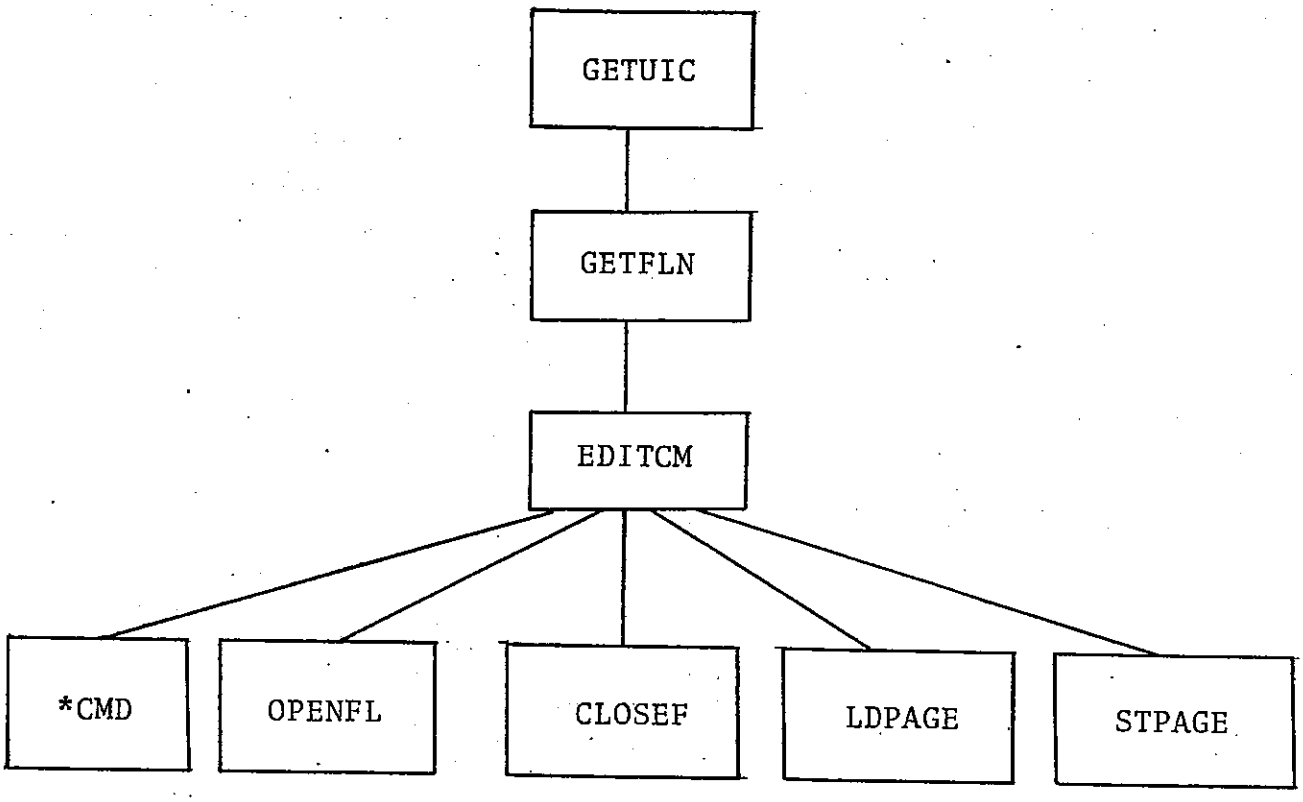


Fig.6.1

(3) OPENFL: send an open file command to COMTSK

.set up the MCB

.call SEND(REMX system call) to send it to COMTSK

.call RECVW(remx system call) to receive the response

.check the status of the file (new or old)

(4) CLOSEF: send a close file command to COMTSK

.set up the MCB

.call SEND to send it to COMTSK

.call RECVW to receive its response

.check error status

(5) LDPAGE: request to load a page from COMTSK

.set up the MCB

.call SEND to send it to COMTSK

.call RECVW to receive tis response

.if no error occurs

then set the end-of-page delimeters

set the page bottom address

if the page is the last page

then set EOF flag

else close the file

(6) STPAGE: store a page to host through COMTSK

.call CRLNNO to calculate line number in the page

.set up the MCB

.call SEND to send it to COMTSK

.call RECVW to receive its response

.check error status

(7) EDITCM: edit command analyzer

.call PRTSTR to print prompt message

.call RDLIN to read a command line

.if just a <CR> key (null line)

then transfer to NCMD with no. 1

.if just a <ESC> key (Pervious line)

then transrer to NCMD with no. -1

.search the command type list

```
.if command type is found
    then calculate index to COMLST
         get command process routine address
         transfer to that command process routine
    else print command error message
(8) ACMD: append a string to the current line
  .if 'AP' is specified
    then set print flag
  .skip to the next token (the appended string)
  .call CHKMSZ to check memory size
  .call NXLINE to move pointer ot next line
  .call DELSTR to delete <CR> in current line
  .call INSERT to insert the string after the current line
  .if print flag = 1
      then call PRTCUR to print this line
(9) BCMD: BOB(begin of block) or BOT(bottom of page)
  .check 'BOB' or 'BOT'
  .if a 'BOB' command
      then transfer to TCMD
  .move current line pointer to page bottom
  .call PRVLIN to move back to last line
  .call PRTCUR to print the last line
(10) CCMD: replace a string by a new string
  .call CHLNST to check current line status
  .call NXNBLK to point to the old string to be replaced
  .call CHKMSZ to check memory size
  .call SRHSTR to search the old string
  .if the string is found
      then call DELSTR to delete the string
           call INSERT to insert the new string
           call CURRLN to point to the line head
           call PRTCUR to print the current line
      else call PRTLIN to print no change message
```

```
(11) DCMD: delete lines
     .if 'DP' is specified
         then set print flag
     .call CHKNEG to check for previous lines deleting
     .call CLCMNO to get no. of lines to be deleted
     .call CHLNST to check current line status
     .if end of page
         then call PRTSTR to print 'EOB' message
     .if CMNNO ≠ 0
         then call DELLN to delete one line
              call DECCMN to decrease CMNNO
(12) ECMD: exit from edit utility
     .call STPAGE to store the current page
     .call CLOSEF to close the file
     .call PRTLIN to print exit message
     .transfer to GETUIC initialization again
(13) ICMD: insert one line or many lines of text
     .check insert one line or many lines
     .if input many lines of text
         then do while input is not a null line (just a  CR )
                 call CHKMSZ to check memory size
                 call NXLINE to move to next line
                 call DELSTR to get length to bottom
                 call INSERT to insert this line
              else end of insert
                   transfer to EDITCM
         else call CHKMSZ to check memory size
              call NXLINE to move to next line
              call DELSTR to get length to bottom
              call INSERT to insert this line
```

(14) LCMD: locate a string from the following lines

.call NXLINE to move to the next line

.call SRHSTR to search the string to be located

.if the string is found

then call CURRLN to point to the line head

call PRTCUR to print the current line

else call PRVLIN to point to the last line in page

call PRTLIN to print not found message

(15) NCMD: move line pointer forward or backward

.if 'NP' is specified

then set print flag

.call CHKNEG to check for backward moving

.call CLCMNO to get no. of lines to be moved

.if CMNNO ≠ 0

then call NXLINE to move to next line

call DECCMN to decrease CMNNO

(16) PCMD: print lines

.call CLCMNO to get no. of lines to be printed

.if CMNNO ≠ 0

then call CHLNST to check line status

call PRTCUR to print current line

call DECCMN to decrease CMNNO

(17) RCMD: replace the current line by a new line

.call CHLNST to check the current line status

.call NXLINE to move to next line

.call DELSTR to delete the current line

.call PRVLIN to move to previous line

.call INSERT to insert the new line

(18) TCMD: top of page('T') or top of file ('TOF')

.if 'TOF' is specified

then call STPAGE to store current page

call CLOSEF to close the file

transfer to OPENFL to open it again

else set current line pointer to page top

7. Conclusion

In this report, we have presented a system that utilizes a microcomputer in communication and multiuser applications. In summary, this system has the following features:

(1) A multitasking scheme is introduced into the microcomputer system through an executive.
(2) A reentrant coding technique allows different users to share the same program.
(3) A file transfer protocol has been designed to facilitate reliable and efficient communication between the host and the FEP.

The potential applications of the FEP seems fruitful and promissing. With the multitasking executive software, a microcomputer-based multiuser application system, such as data entry, query processing,... etc., can be designed and implemented without much difficulty. By adding one more communication task and defining necessary protocols, the FEP can become a communication processor for the distributed processing system.

References:

(1) Marvin V. Zelkowitz and Alan C. Shaw, "Principles of Software Enginering and Desing",

(2) R.C. Holt, "Structured Concurrent Programming with Operating system Applications", Addison Wesley, 1978.

(3) "RMX-80 User's Guide", Intel Corp.

(4) "REX-80 Primer", System and Software Corp.

(5) "RSX-11/M Utility Manual", Digital Equipment Corp.

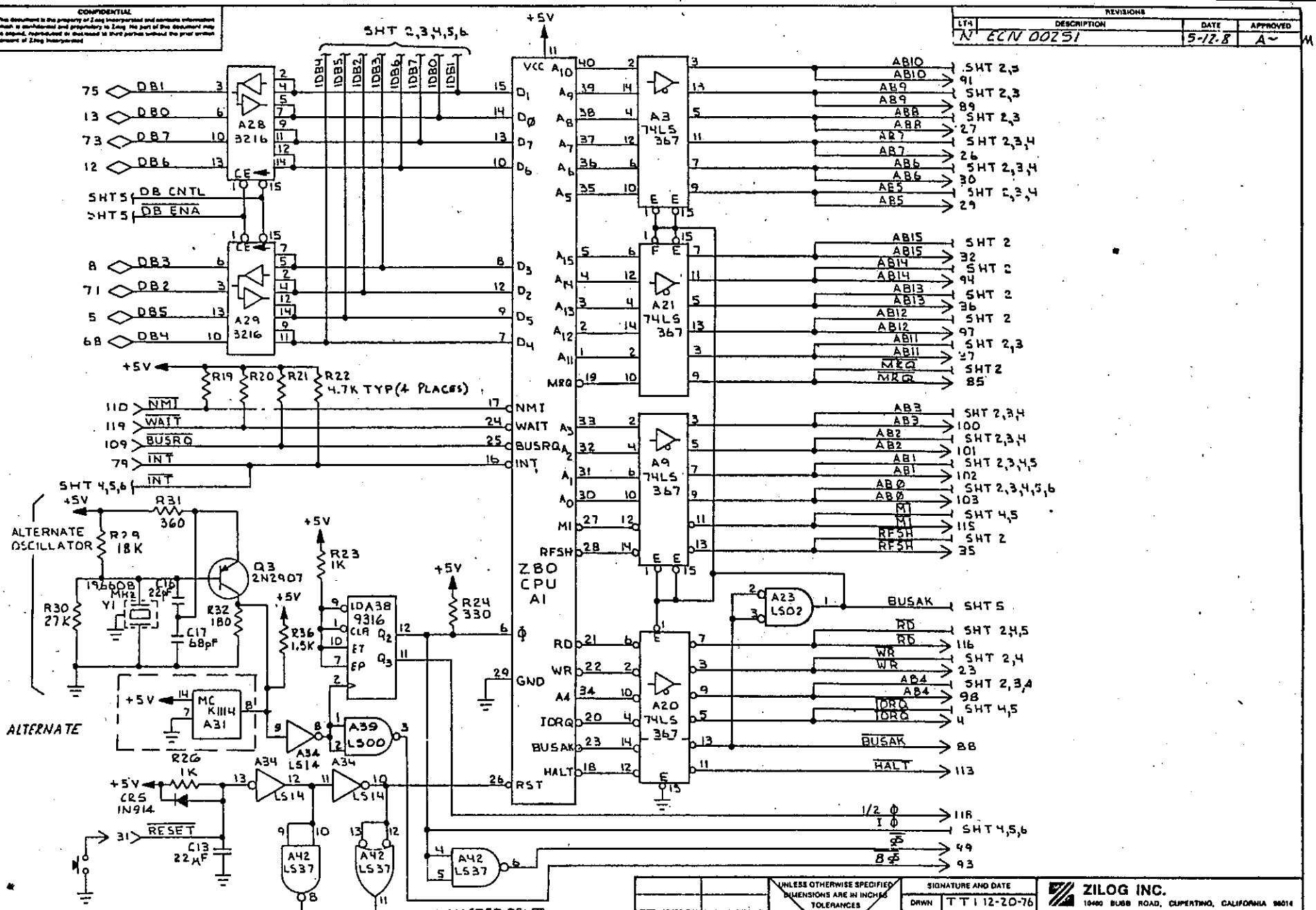
(6) D. W. Davies, "Computer Networks and Their Protocols"

(7) Michel Gien, "A File Transfer Protocol", Computer Network, Fib. 1978.

(8) T. Agerwala, "Putting Petri Net to Work", Computer, Dec. 1979.

APPENDIX

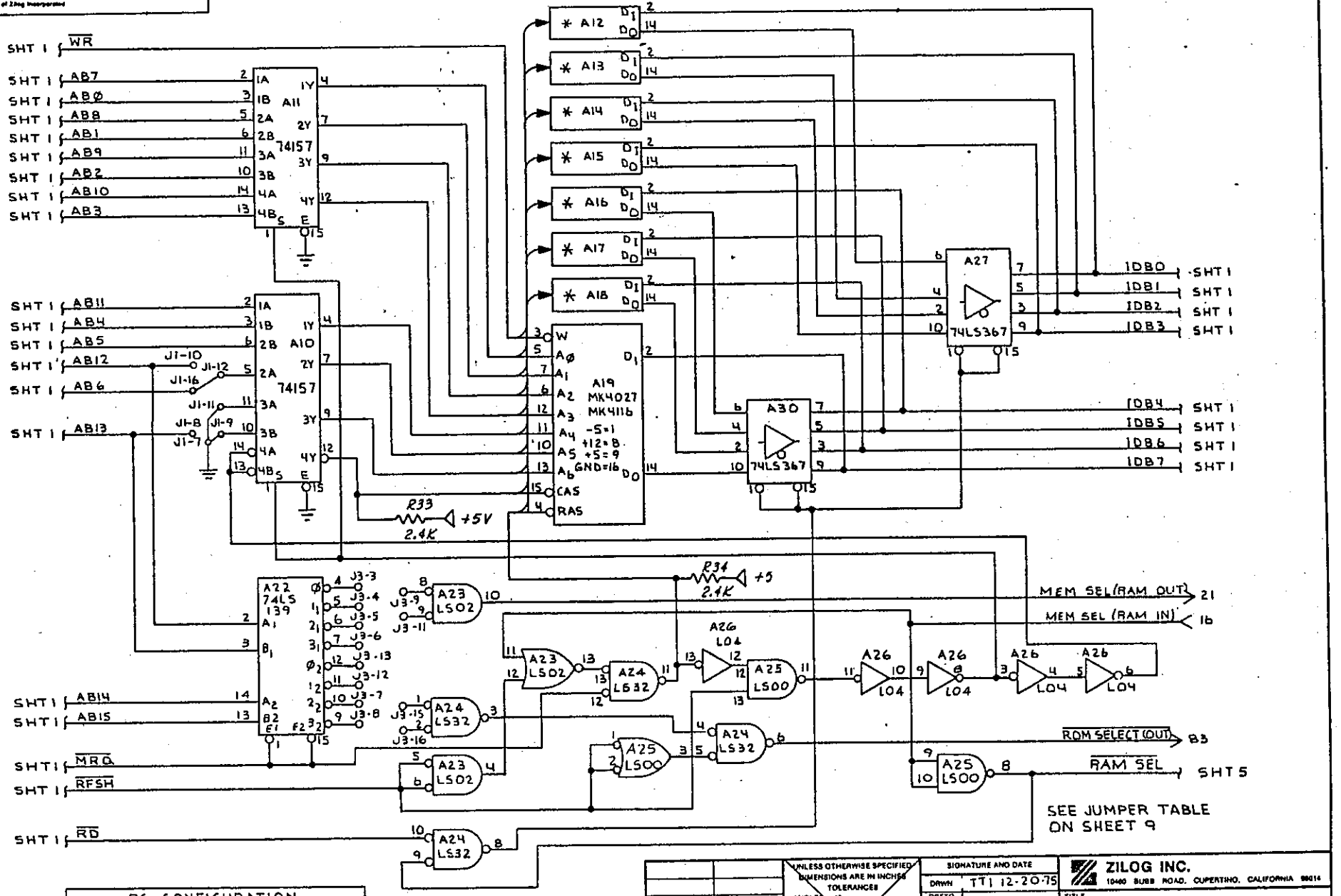
REVISIONS			
LTR	DESCRIPTION	DATE	APPROVED
N	ECN 00251	5-12-8	A~ M



UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ANGLES ± 1° FRACTIONS ± 1/64 3 PLACE DEC ± .010 2 PLC DEC ± .02	SIGNATURE AND DATE		ZILOG INC. 10460 BUBB ROAD, CUPERTINO, CALIFORNIA 95014
	DRWN	TT 12-20-76	TITLE LOGIC DIAGRAM MICRO-COMPUTER BOARD
	DRFTG CHK		
	APVD		SIZE C DRAWING NO DZ-0013-01 ISSUE N
NEXT ASSY / USED ON / APPLICATION / MATL / FINISH			SCALE / SHEET 1 OF 9

A-1

REVISIONS			
LTR	DESCRIPTION	DATE	APPROVED



PC CONFIGURATION	
ROM (0000H)	J3-3/J3-16; J3-15/J3-13
RAM (1000H)	J3-13/J3-11; J3-4/J3-9

* A12-A19 ARE RAMS. SEE P/L

NEXT ASSY	USED ON		
APPLICATION			

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ANGLES ± 1° FRACTIONS ± 1/64 3 PLACE DEC ± .010 2 PLC DEC ± .03

MATL

FINISH

SIGNATURE AND DATE
DRWN TTI 12-20-75
DRFTG CHK
APVD

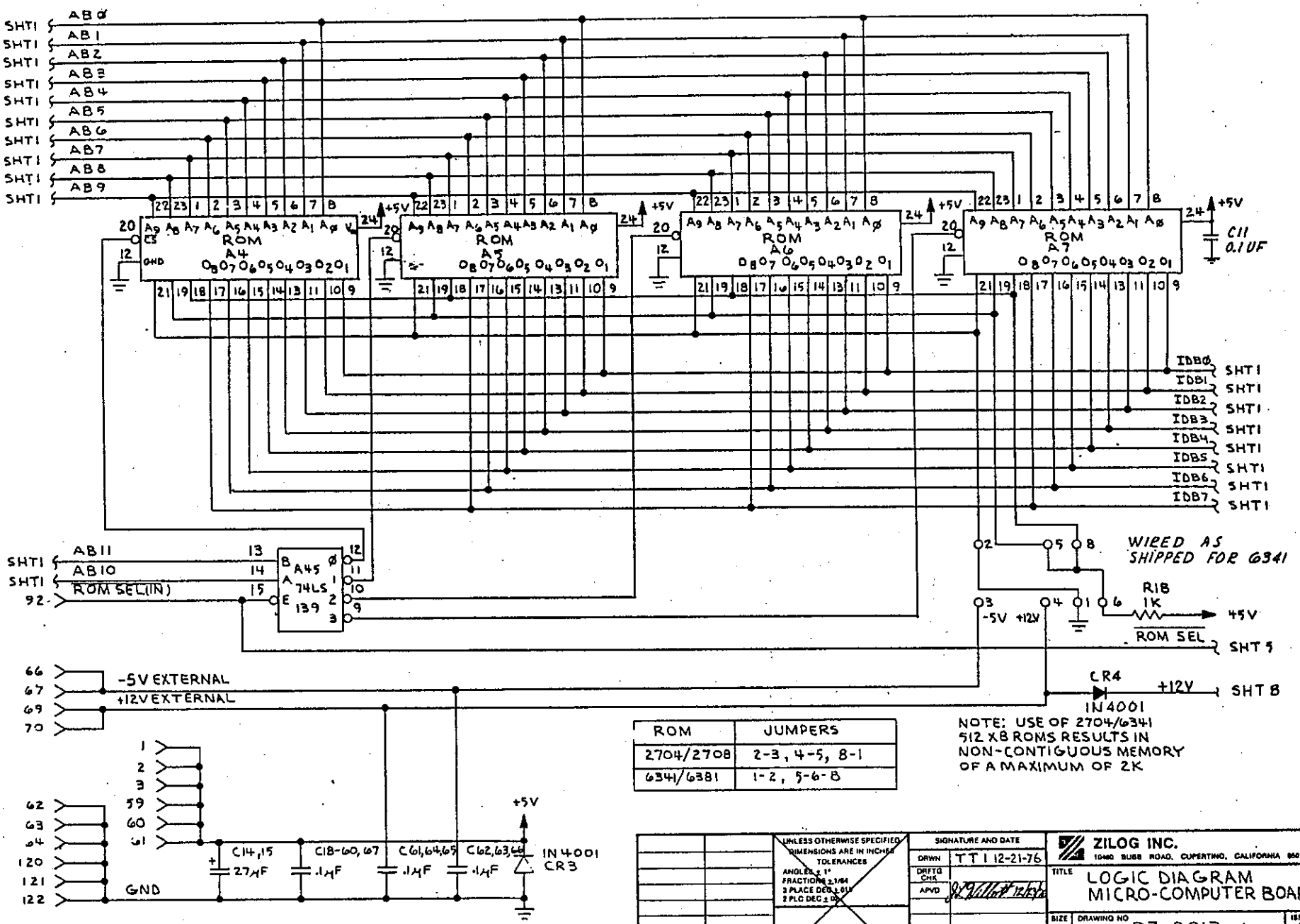
ZILOG INC.
10460 BUBB ROAD, CUPERTINO, CALIFORNIA 95014

TITLE LOGIC DIAGRAM MICRO-COMPUTER BOARD

SIZE C | DRAWING NO DZ-0013-01 | REVISE A

SCALE | SHEET 2 OF 9

REVISIONS			
LTR	DESCRIPTION	DATE	APPROVED



ROM	JUMPERS
2704/2708	2-3, 4-5, 8-1
6341/6381	1-2, 5-6-8

NOTE: USE OF 2704/6341 512 X8 ROMS RESULTS IN NON-CONTIGUOUS MEMORY OF A MAXIMUM OF 2K

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES
ANGLES ± 1°
FRACTIONS ± 1/64
3 PLACE DEC ± .01
2 PLC DEC ± .0

SIGNATURE AND DATE	
DRWN	TT 1 12-21-76
DRFTG CHK	
APVD	

NEXT ASSY | USED ON
APPLICATION
MATL
FINISH

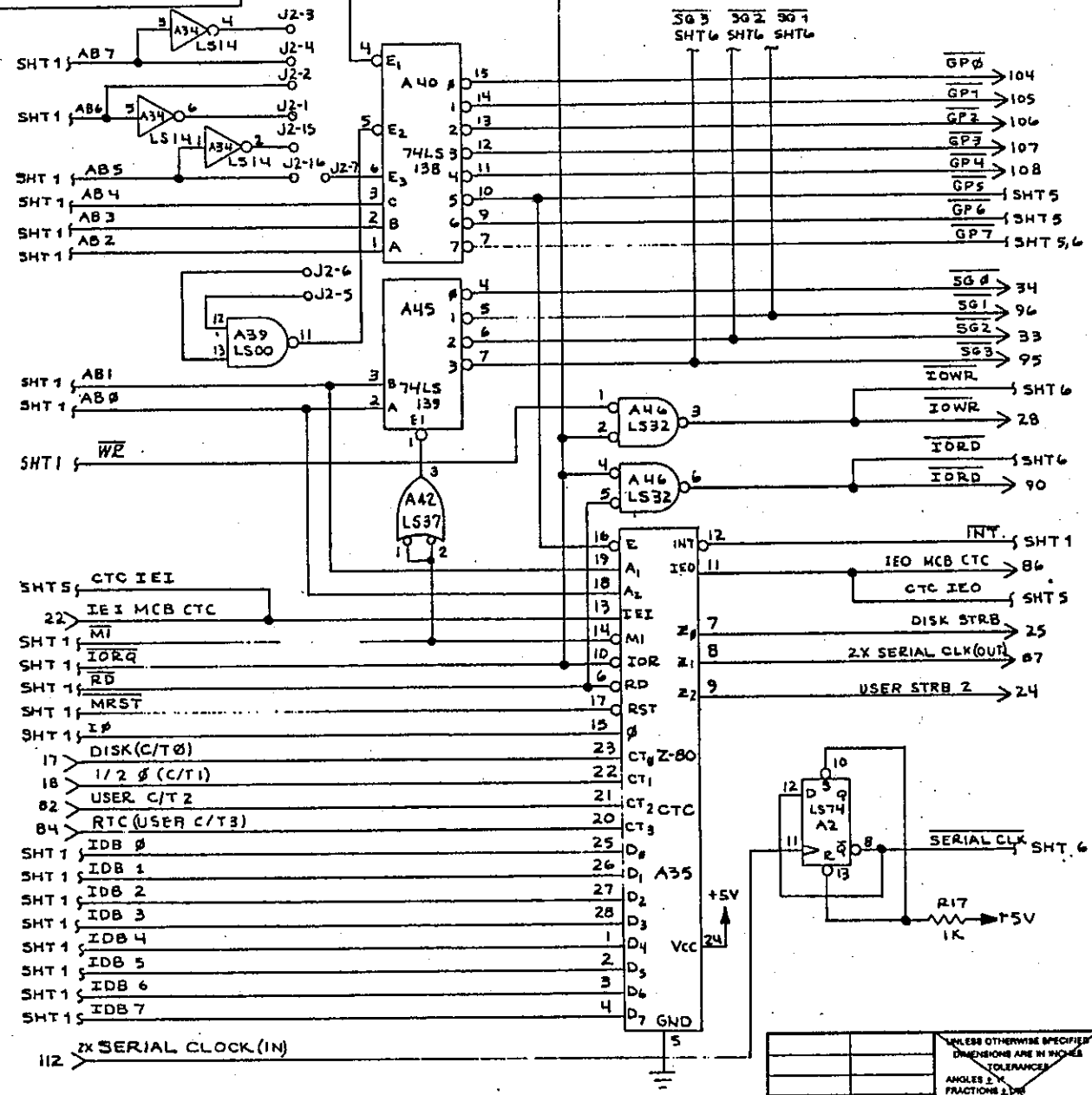
ZILOG INC.
10460 BUBB ROAD, CUPERTINO, CALIFORNIA 95014

TITLE LOGIC DIAGRAM MICRO-COMPUTER BOARD

SIZE C | DRAWING NO DZ-0013-01 | REV N

SCALE | SHEET 3 OF 9

REVISIONS			
LTR	DESCRIPTION	DATE	APPROVED



JUMPER 18 TO 118 FOR HIGH SPEED SERIAL.
JUMPER 87 TO 112 FOR ON BOARD SERIAL TIMING GENERATION.

PORT GROUP DECODE		
Y	Z	JUMPER PINS OF J2
Ø	1	5-15, 1-7, 3-6
2	3	5-16, 1-7, 3-6
4	5	5-15, 2-7, 3-6
6	7	5-16, 2-7, 3-6
8	9	5-15, 1-7, 4-6
A	B	5-16, 1-7, 4-6
C	D	5-15, 2-7, 4-6
E	F	5-16, 2-7, 4-6

I/O PORT DECODE					
	SGØ	SG1	SG2	SG3	
GP Ø	Y0	Y1	Y2	Y3	
GP1	Y4	Y5	Y6	Y7	
GP2	Y8	Y9	YA	YB	
GP3	Yc	YD	YE	YF	
GP4	Z0	Z1	Z2	Z3	
GP5	Z4	Z5	Z6	Z7	CTC
GP6	Z8	Z9	ZA	ZB	PIO
GP7	ZC	ZD	ZE	ZF	SERIAL

NOTE: MDC USES GP4 AND GP3/SG3 (CF, DØ, D1, D2, D3)

NEXT ASSY	USED ON		
APPLICATION			

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ANGLES ± ° FRACTIONS ± 1/16 3 PLACE DEC ± .010 2 PLC DEC ± .02

MAT'L:

FINISH

SIGNATURE AND DATE	
DRAWN	TTI 12-17-76
DRFTG CHK	
APVD	

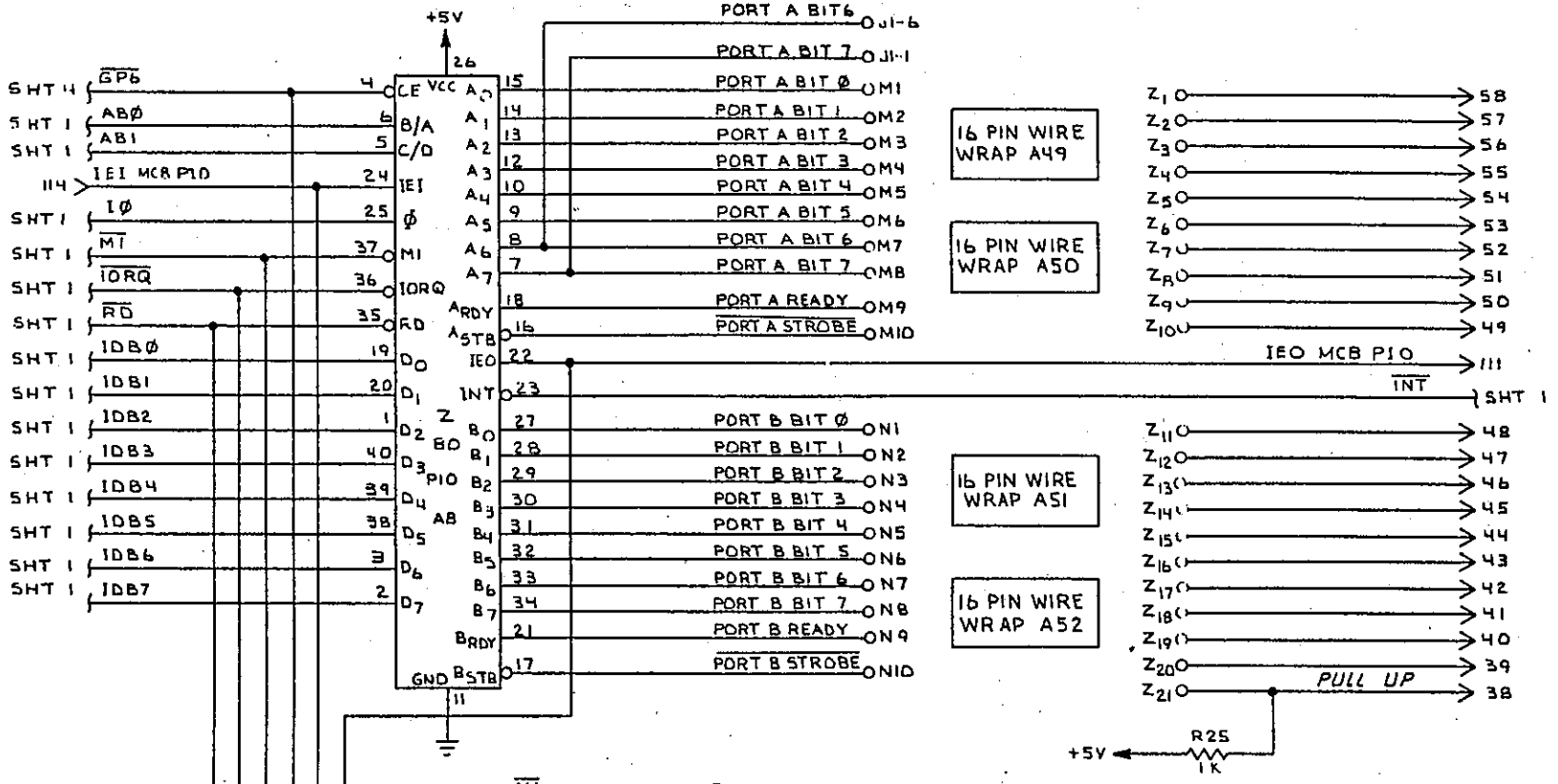
ZILOG INC.
10460 BUBB ROAD, CUPERTINO, CALIFORNIA 95014

TITLE: LOGIC DIAGRAM MICRO-COMPUTER BOARD

SIZE C | DRAWING NO. D2-0013-01 | REV N

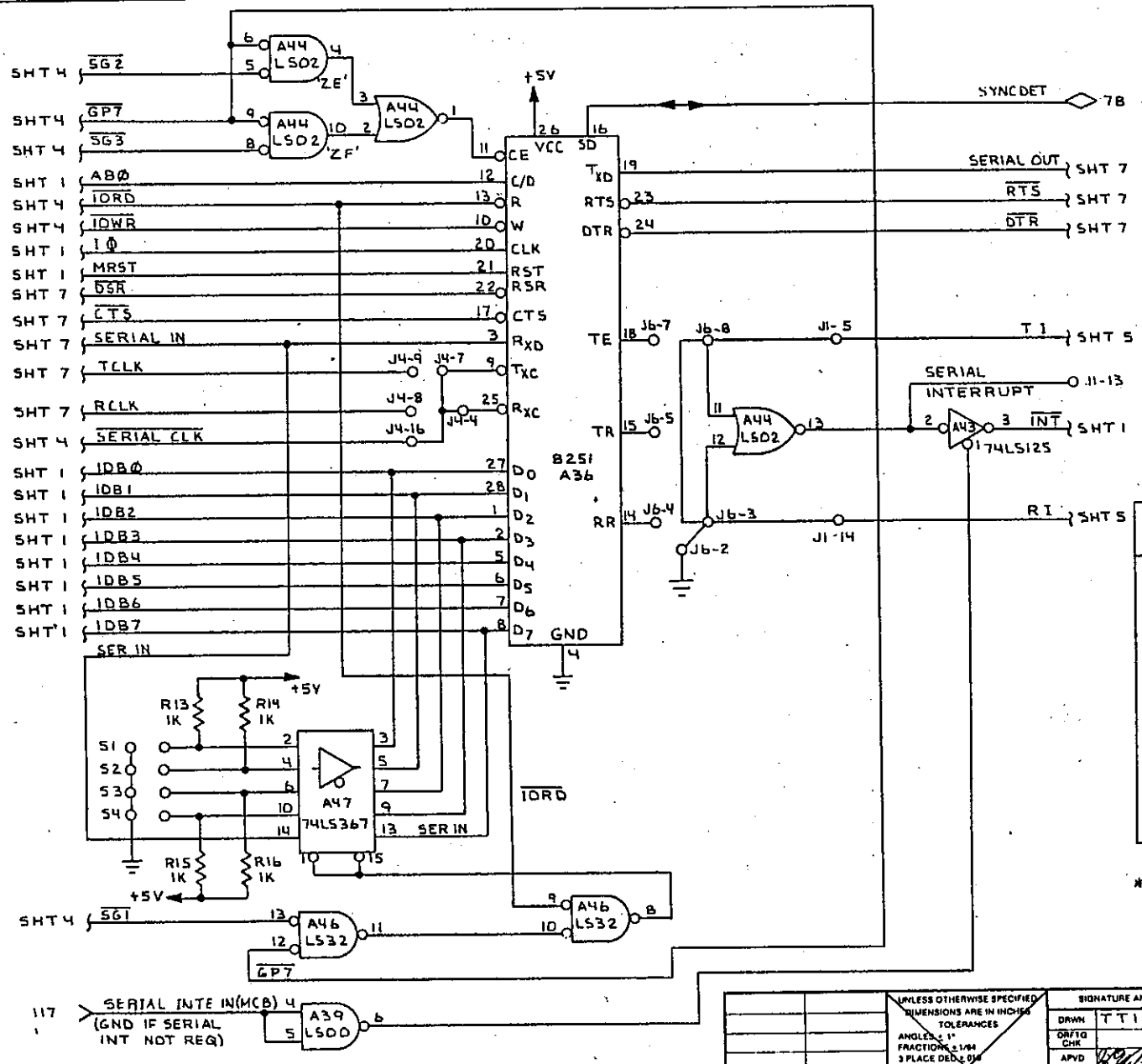
SCALE | SHEET 4 OF 9

REVISIONS			
LTR	DESCRIPTION	DATE	APPROVED



		UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ANGLES ± 1° FRACTIONS ± 1/64 3 PLACE DEC ± .010 2 PLC DEC ± .02	SIGNATURE AND DATE	ZILOG INC. 10460 BUBB ROAD, CUPERTINO, CALIFORNIA 95014
			DRWN TTI 12-19-76	TITLE LOGIC DIAGRAM MICRO-COMPUTER BOARD
			DRFTG CHK	
			APVD	
NEXT ASSY	USED ON	MATL	FINISH	SIZE C DRAWING NO DZ-0013-01 REV N
APPLICATION				SCALE SHEET 5 OF 9

REVISIONS			
LTR	DESCRIPTION	DATE	APPROVED



JUMPER J6-5 TO J6-8 OR J6-7 TO J6-8 AND J6-4 TO J6-3 FOR SERIAL INTERRUPTS. IF NO INTERRUPT IS REQUIRED, JUMPER J6-2 TO J6-3 AND J6-8 AND TIE 117 TO GND (120).

FOR EXTERNAL TRANSMIT/RECEIVE CLOCKS JUMPER J4-9 TO J4-7 AND J4-4 TO J4-8. FOR INTERNAL CLOCK JUMPER J4-16 TO J4-4 AND J4-7.

PROGRAM 8251 FOR ÷16 FOR BAUD RATES SHOWN BELOW.

$\Phi = 2.4576\,MHz \quad f_s = \Phi \div 2N$

BAUD RATE	f_s 16X RATE	JUMPERS S1	S2	S3	S4	N
50	800Hz	ON	ON	ON	ON	96X16
75	12KHz	OFF	ON	ON	ON	64X16
110	1.745KHz	ON	OFF	ON	ON	44X16
134.5	2.133KHz	OFF	OFF	ON	ON	36X16
150	2.4KHz	ON	ON	OFF	ON	32X16
200	3.2KHz	OFF	ON	OFF	ON	24X16
300	4.8KHz	ON	OFF	OFF	ON	16X16
600	9.6KHz	OFF	OFF	OFF	ON	8X16
1200	19.2KHz	ON	ON	ON	OFF	4X16
2400	38.4KHz	OFF	ON	ON	OFF	2X16
4800	76.8 KHz	ON	OFF	ON	OFF	1X16
9600	153.6KHz	OFF	OFF	ON	OFF	4**
19200	307.2KHz	ON	ON	OFF	OFF	2**
38400	614.4KHz	OFF	ON	OFF	OFF	1**
AUTO*	AUTO	OFF	OFF	OFF	OFF	AUTO*

** JUMPER 1B TO 11B FOR HIGH SPEED SERIAL.
* SPECIAL SOFTWARE SUPPORT REQUIRED.

NEXT ASSY	USED ON	APPLICATION	

UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES
ANGLES ± 1°
FRACTIONS ± 1/64
3 PLACE DEC ± .010
2 PLC DEC ± .02

SIGNATURE AND DATE	
DRWN	TT 12-20-76
CHK'D CHK	
APVD	
MATL	
FINISH	

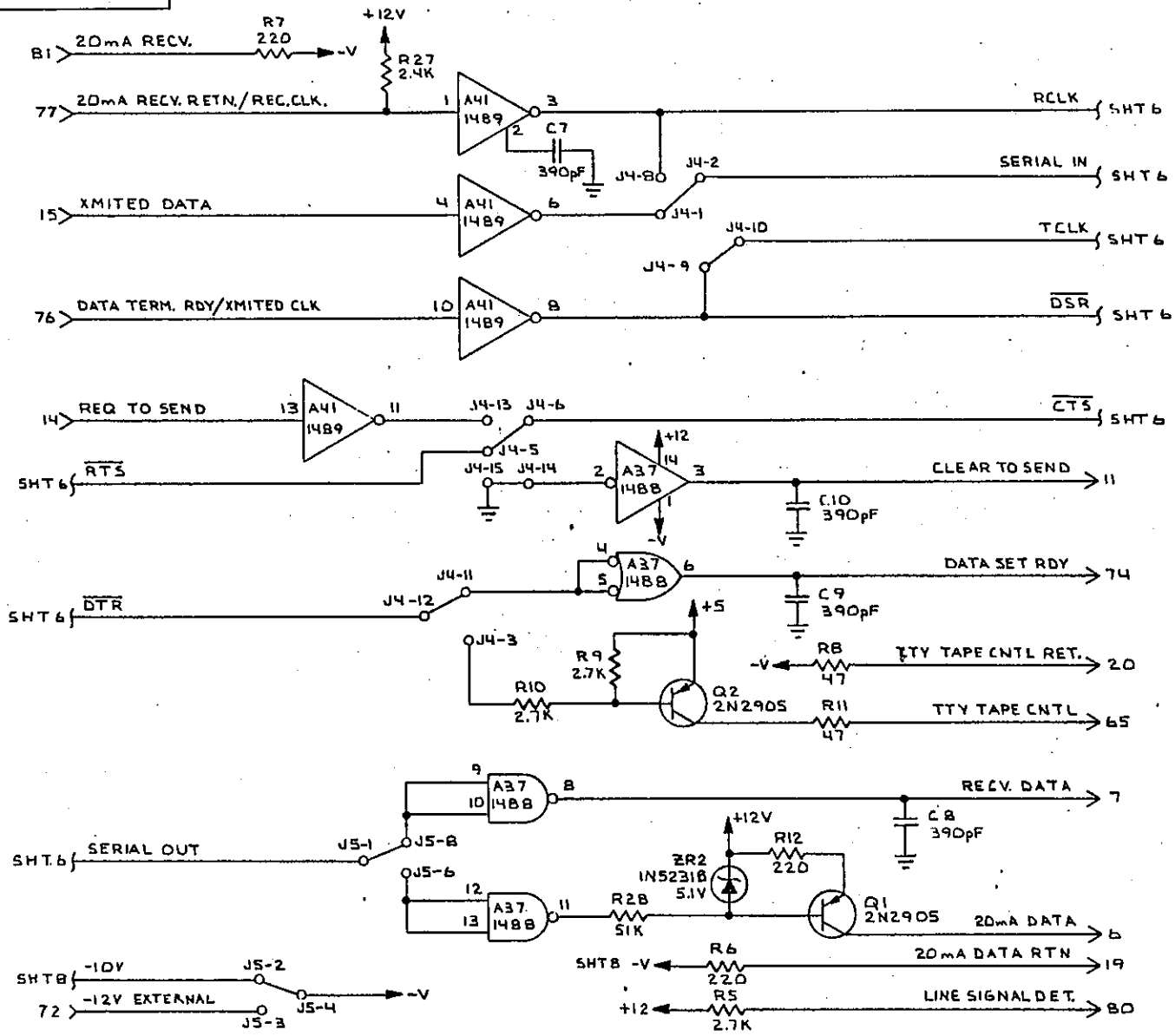
ZILOG INC.
10460 BUBB ROAD, CUPERTINO, CALIFORNIA 95014

TITLE
LOGIC DIAGRAM
MICRO-COMPUTER BOARD

SIZE	DRAWING NO	ISSUE
C	DZ-0013-01	N

SCALE | SHEET 6 OF 9

REVISIONS			
LTR	DESCRIPTION	DATE	APPROVED



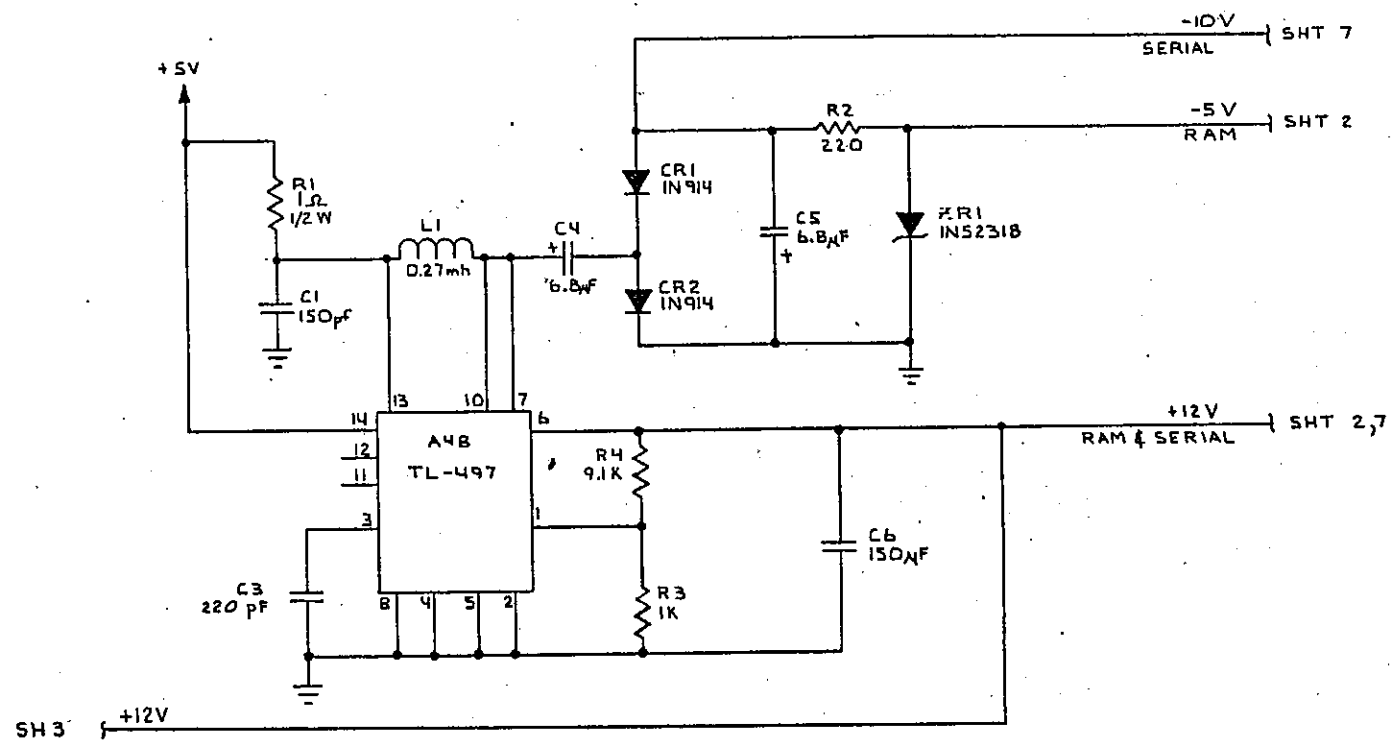
JUMPERS SHOWN ARE FOR RS-232C, ALWAYS CLEAR TO SEND AND REQUEST TO SEND IGNORED.

SIGNAL NAMES ARE SUCH THAT MCB WILL REPLACE MODEM IN TERMINAL NETWORK.

SEE JUMP TABLE ON SHEET 9 FOR OTHER CONFIGURATIONS.

NEXT ASSY	USED ON	UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ANGLES ± FRACTIONS ± 3 PLACE DEC ± 2 PLC DEC ± .02	SIGNATURE AND DATE	ZILOG INC. 10460 BUBB ROAD, CUPERTINO, CALIFORNIA 95014
APPLICATION		MAT'L.	DRWN TTI 12-17-76	TITLE LOGIC DIAGRAM MICRO-COMPUTER BOARD
		FINISH	DRFTG CHK	
			APVD	SIZE C DRAWING NO DZ-0013-01 ISSUE A
				SCALE SHEET 7 OF 9

REVISIONS			
LTR	DESCRIPTION	DATE	APPROVED



NOT USED	LAST USED
C11, C12, C25	AS2, C67, CR3, J6, L1, MID, NID, Q3, R32, Z21, ZR2

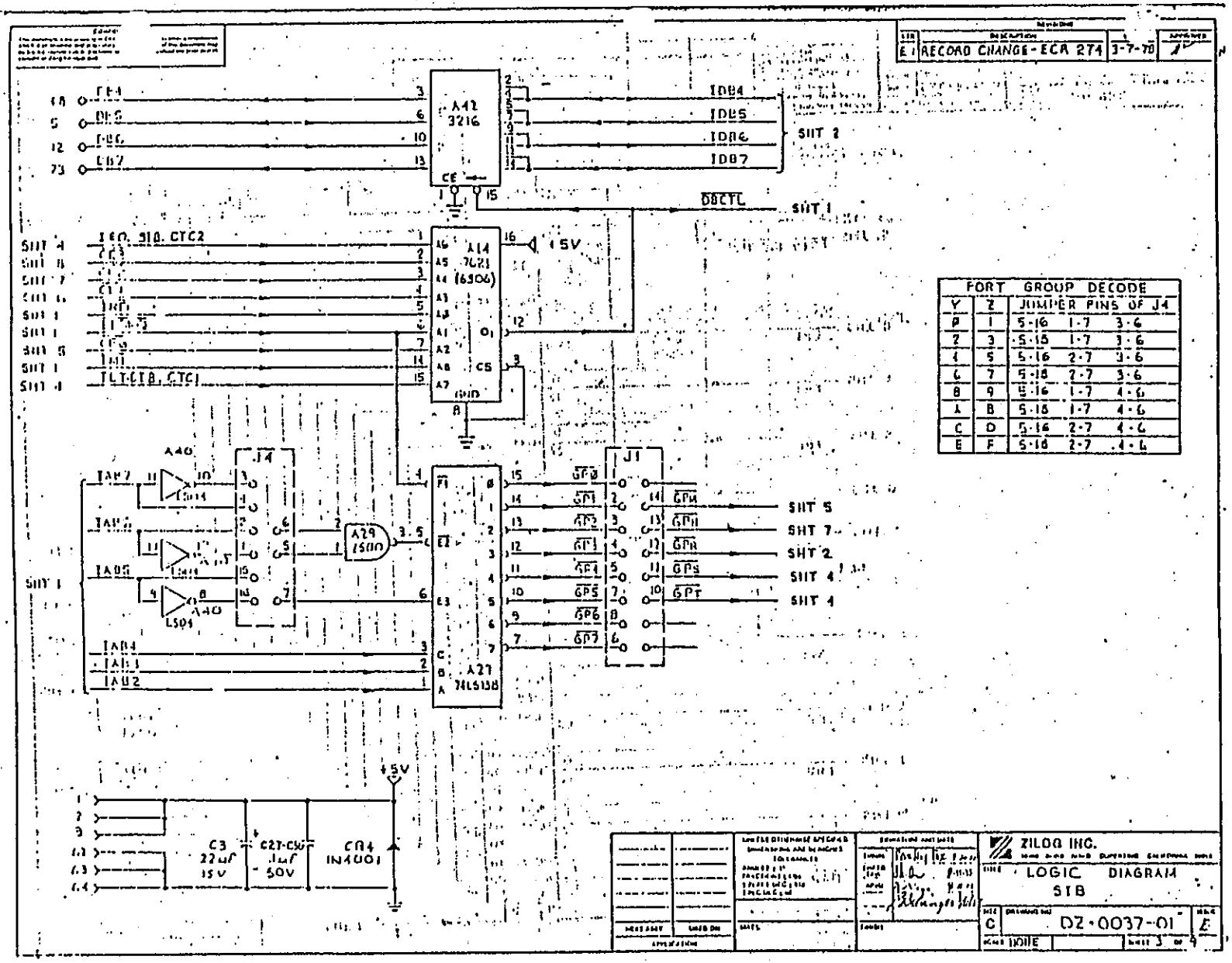
UNLESS OTHERWISE SPECIFIED DIMENSIONS ARE IN INCHES TOLERANCES ANGLES ± 1° FRACTIONS ± 1/64 3 PLACE DEC ± .010 2 PLC DEC ± .03	SIGNATURE AND DATE	
	DRWN	T T I 12-20-76
	DRFTG CHK	
	APVD	
NEXT ASSY / USED ON / APPLICATION	MATL	FINISH

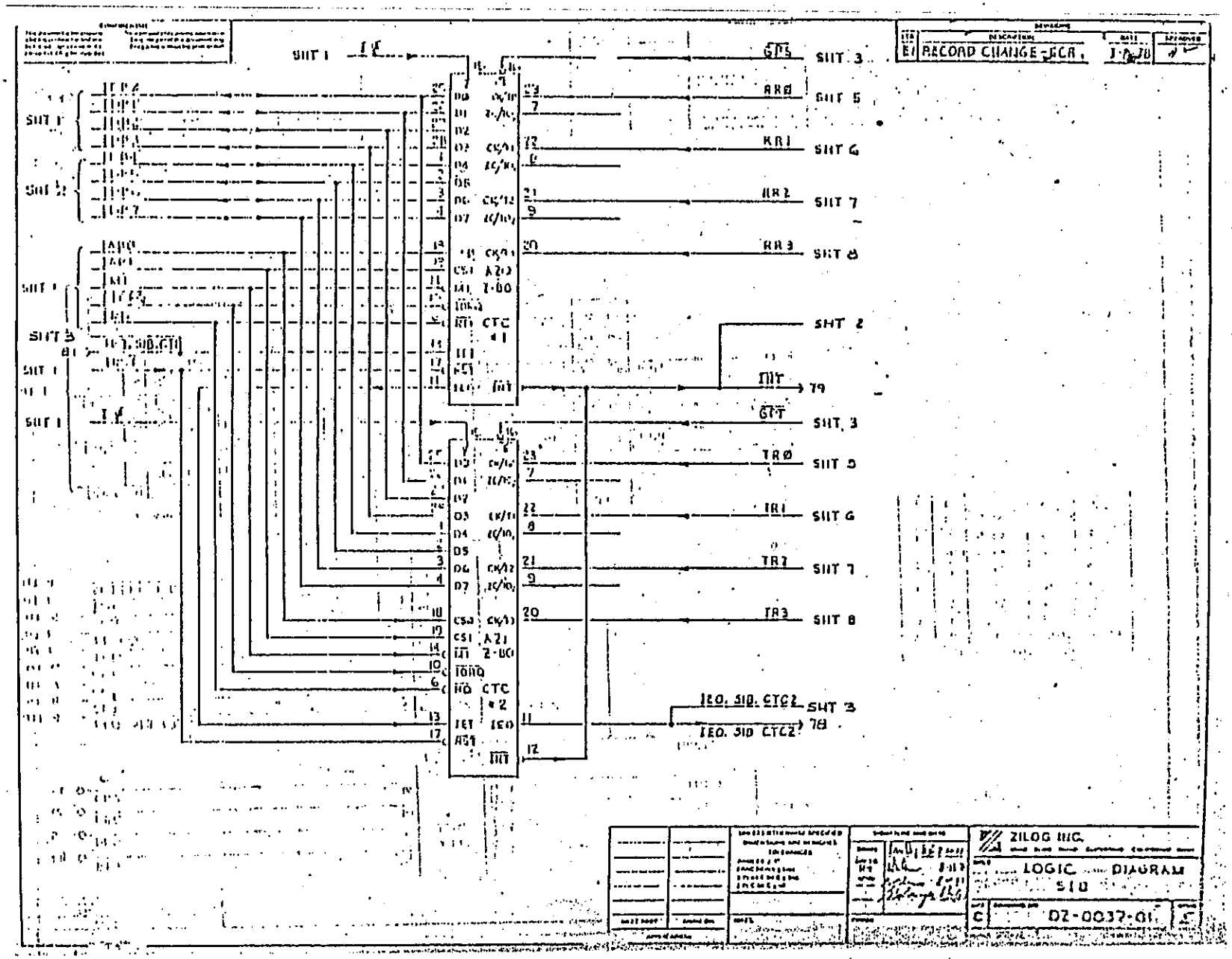
ZILOG INC.
10460 BUBB ROAD, CUPERTINO, CALIFORNIA 95014

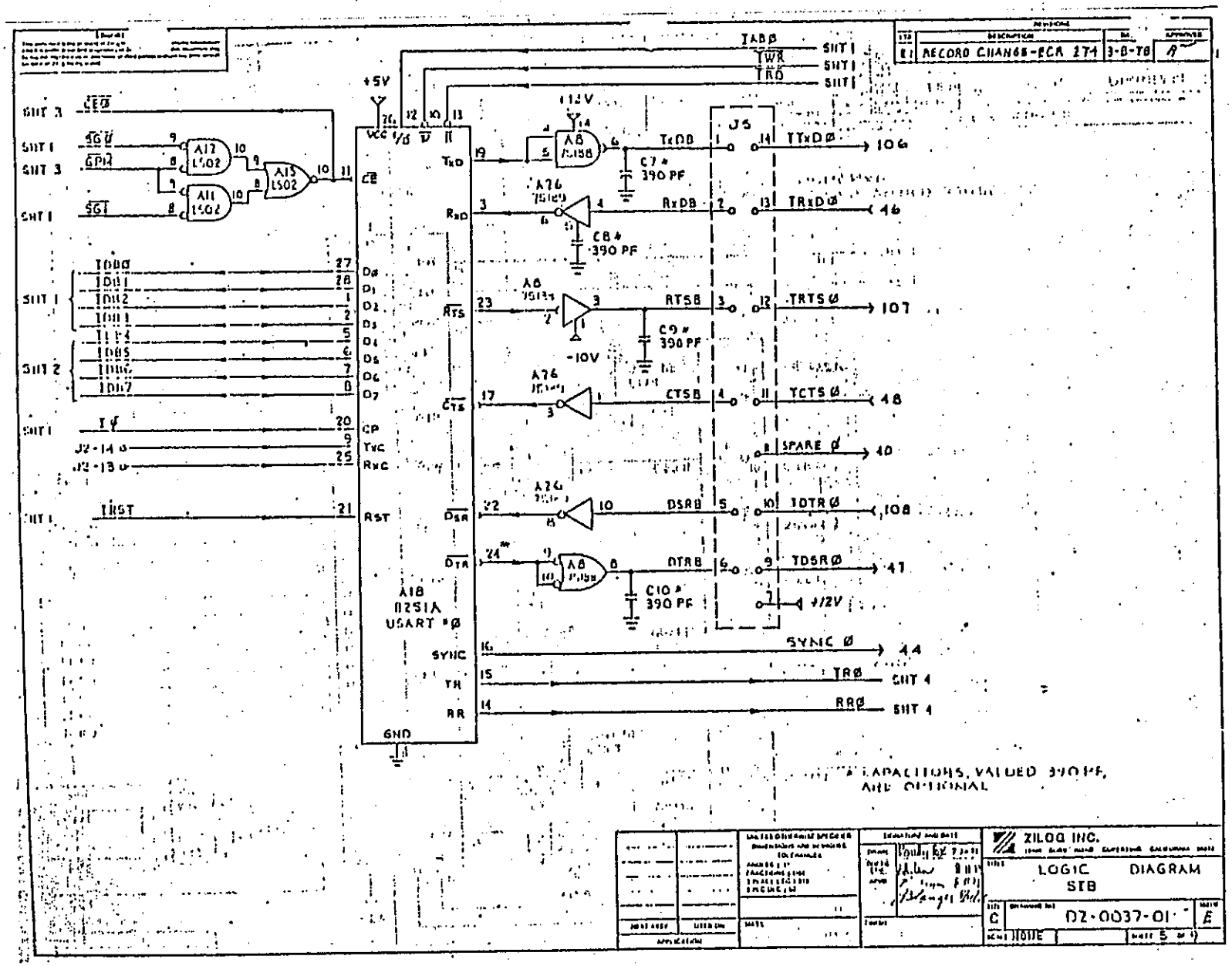
TITLE: LOGIC DIAGRAM MICRO-COMPUTER BOARD

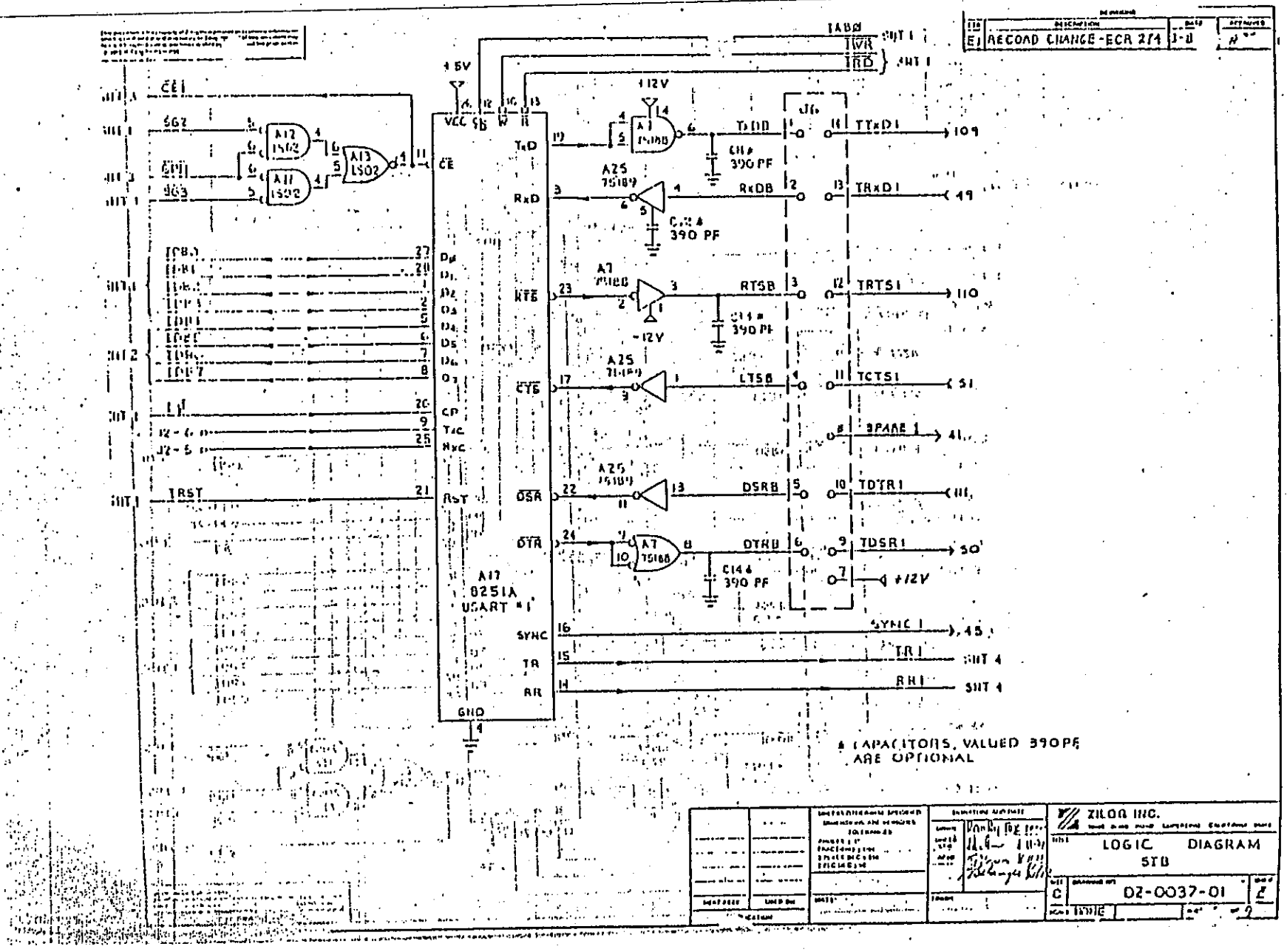
SIZE C | DRAWING NO DZ-0013-01 | ISSUE A

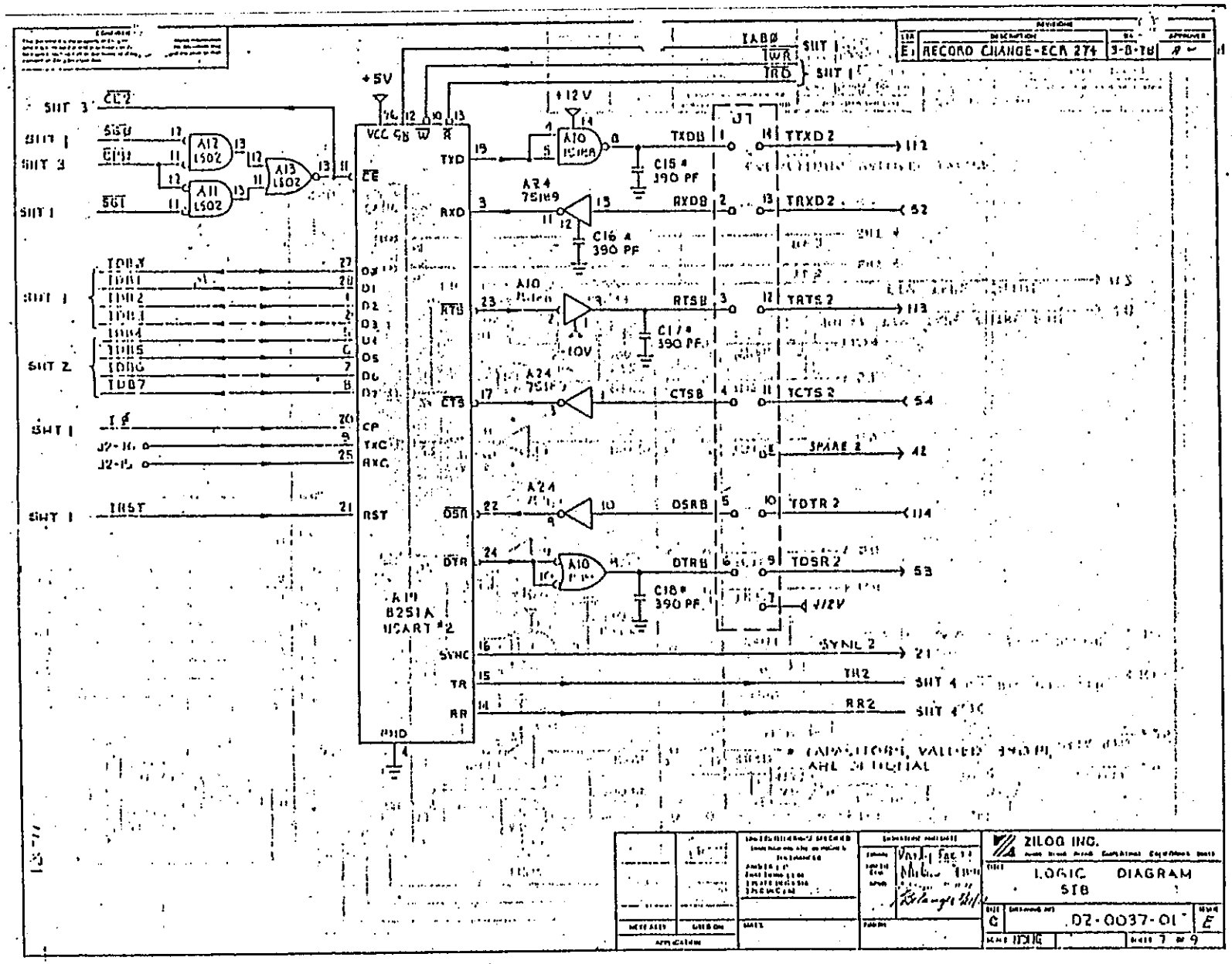
SCALE | SHEET 8 OF 9

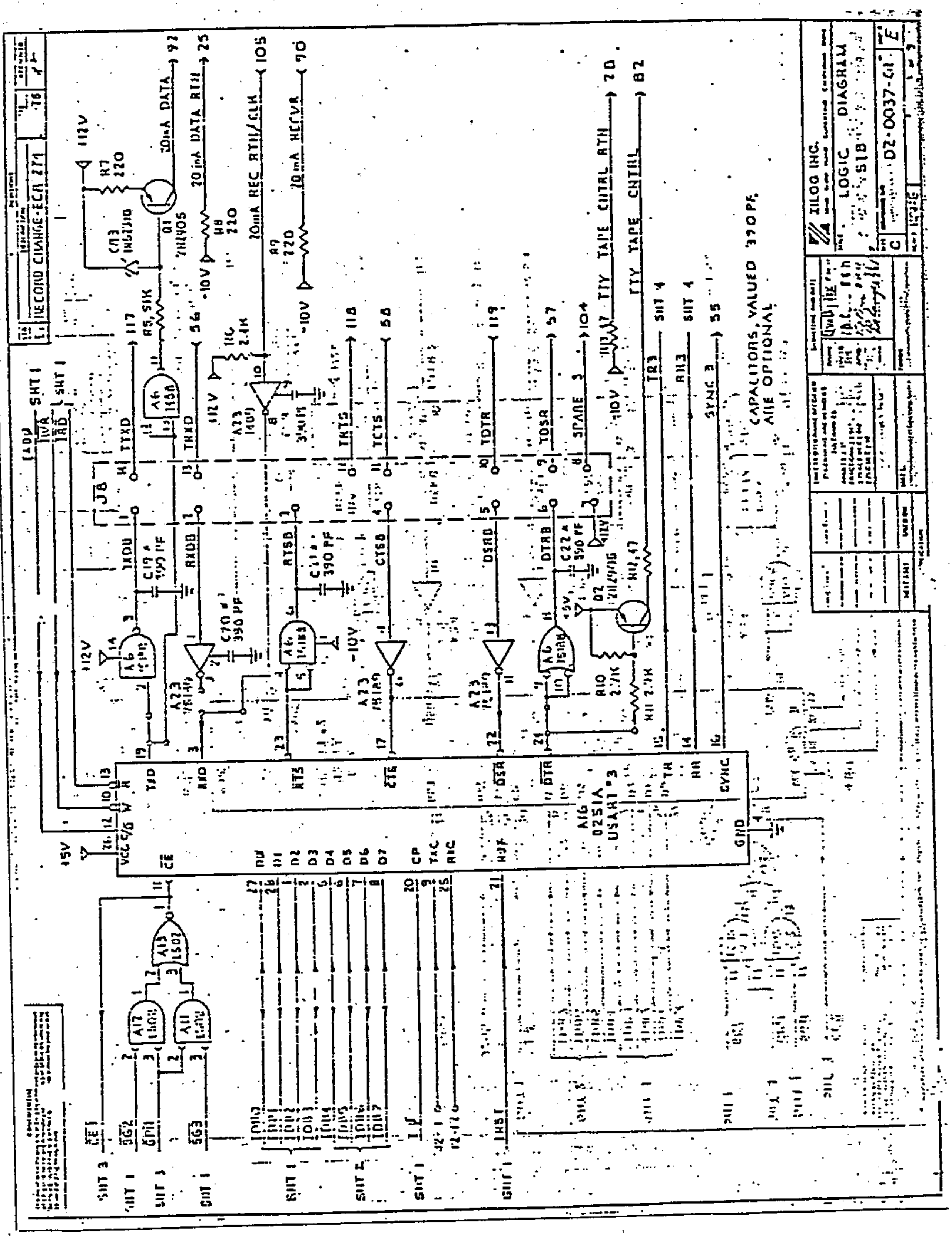

REVISIONS
LTR DESCRIPTION DATE APPROVED
E RECORD CHANGE - ECR 274 3-7-78
A42 3216
CE
DB4
DB5
DB6
DB7
SHT 2
DBCTL
SHT 1
+5V
A14 7621 (6504)
IEO, SIO, CTC2
CTC1
SHT 5
SHT 7
SHT 2
SHT 4
SHT 1
A40
LS04
A29 LS00
J4
J1
GP0
GP1
GP2
GP3
GP4
GP5
GP6
GP7
A27 74LS138
TAB7
TAB6
TAB5
TAB4
TAB3
TAB2
C3 22uF 15V
C27-CW 1uF 50V
CR4 IN4001
FORT GROUP DECODE
Y Z JUMPER PINS OF J4
P 1 5-16 1-7 3-6
2 3 5-15 1-7 3-6
1 5 5-16 2-7 3-6
4 7 5-15 2-7 3-6
8 9 5-16 1-7 4-6
A B 5-15 1-7 4-6
C D 5-16 2-7 4-6
E F 5-15 2-7 4-6
ZILOG INC.
LOGIC DIAGRAM
STB
SIZE C
DZ-0037-01
REV E
SCALE NONE
SHEET 3 OF 4

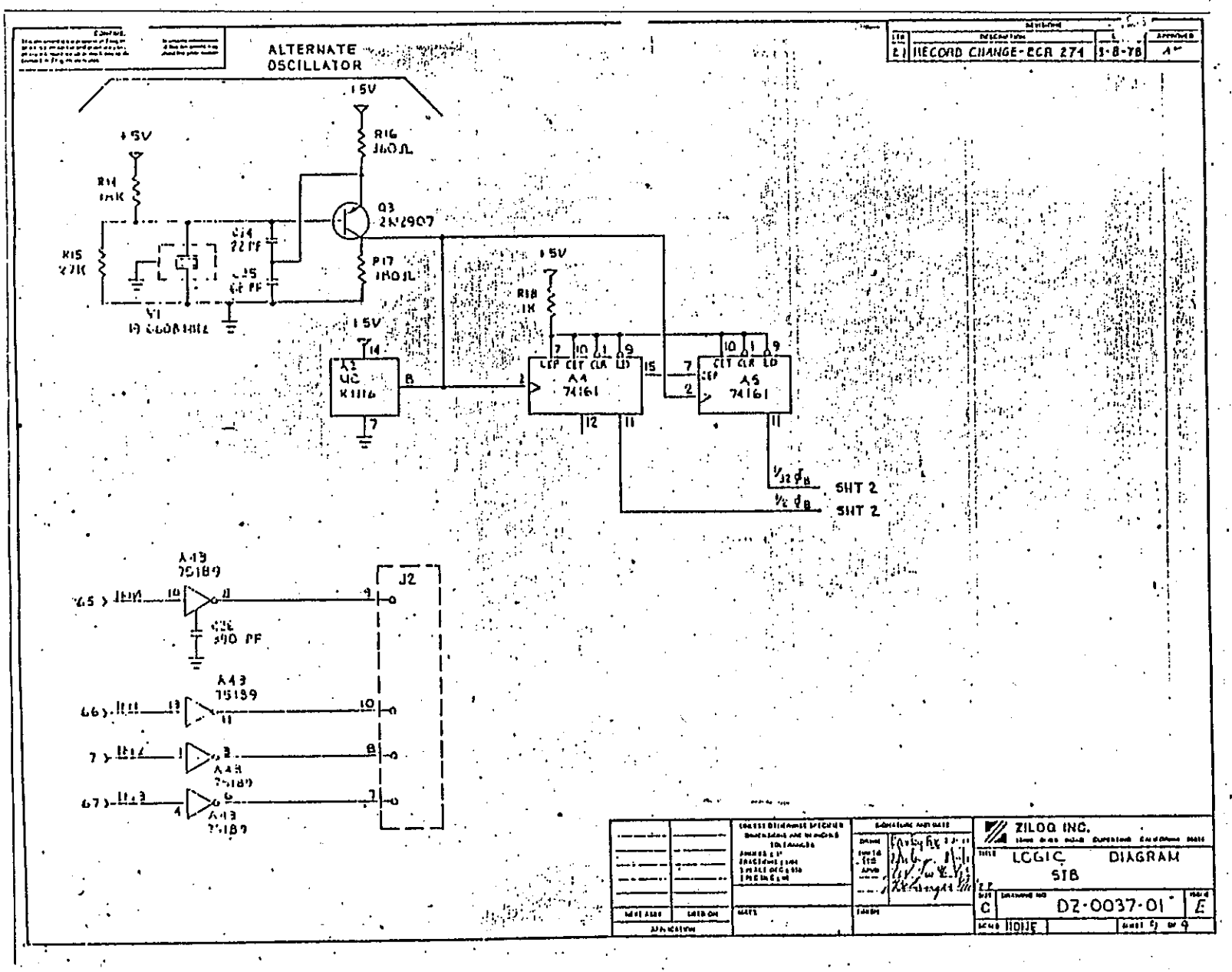

RECORD CHANGE - ECR
ZILOG INC.
LOGIC DIAGRAM
SIO
02-0037-01
CTC #1
CTC #2
SHT 1
SHT 2
SHT 3
SHT 5
SHT 6
SHT 7
SHT 8
SHT 4
SHT 79
SHT 78
GPS
ARØ
AR1
AR2
AR3
TRØ
TR1
TR2
TR3
IEO. SIO. CTC2


RECORD CHANGES-ECR 274
3-8-78
A
TABØ
SHT 1
TWR
SHT 1
TRD
SHT 1
+5V
+12V
SHT 3
CEØ
SHT 1
SGWØ
SHT 3
GPWØ
SHT 1
SGTØ
A17 LS02
A11 LS02
A15 LS02
CE
TxD
RxD
Dø
D1
D2
D3
D4
D5
D6
D7
RTS
CTS
CP
TxC
RxC
RST
DSR
DTR
SYNC
TR
RR
GND
A18
8251A
USART #Ø
SHT 1
IDBØ
IDB1
IDB2
IDB3
IDB4
SHT 2
IDB5
IDB6
IDB7
IØ
J2-14
J2-13
IRST
A8 75188
A26 75189
A6 75189
C7 * 390 PF
C8 * 390 PF
C9 * 390 PF
C10 * 390 PF
-10V
J5
TxDB
RxDB
RTSB
CTSB
DSRB
DTRB
TTxDØ
TRxDØ
TRTSØ
TCTSØ
SPARE Ø
TDTRØ
TDSRØ
+12V
106
46
107
48
40
108
47
SYNC Ø
44
TRØ
SHT 4
RRØ
SHT 4
CAPACITORS, VALUED 390 PF, ARE OPTIONAL
ZILOG INC.
LOGIC DIAGRAM
STB
C
02-0037-01
E
SHEET 5 OF 9


RECORD CHANGE - ECR 274
+5V
+12V
-12V
A17 8251A USART #1
TxD
RxD
RTS
CTS
DSR
DTR
SYNC
TR
RR
GND
TRST
TTxD1
TRxD1
TRTS1
TCTS1
SPARE 1
TDTR1
TDSR1
SYNC 1
TR 1
RR 1
RTSB
LTSB
DSRB
DTRB
C143 390 PF
C144 390 PF
C145 390 PF
C146 390 PF
* CAPACITORS, VALUED 390PF ARE OPTIONAL
ZILOG INC.
LOGIC DIAGRAM
STB
D2-0037-01


+5V
+12V
SHT 3
SHT 1
SHT 2
CLK2
SGD
SGT
A12 LS02
A13 LS02
A11 LS02
TDB0
TDB1
TDB2
TDB3
TDB4
TDB5
TDB6
TDB7
IRST
J2-16
J2-15
A11 8251A USART #2
VCC GND W R
TXD
RXD
CTS
RTS
DSR
DTR
SYNC
TR
RR
RSTD
CP
TXC
RXC
RST
D0 D1 D2 D3 D4 D5 D6 D7
TXDB
RXDB
RTSB
CTSB
DSRB
DTRB
A10
A24 75189
C15 * 390 PF
C16 * 390 PF
C17 * 390 PF
C18 * 390 PF
10V
J1
TTXD 2
TRXD 2
TRTS 2
TCTS 2
SPARE 2
TDTR 2
TDSR 2
SYNC 2
TR2
RR2
SHT 4
112
52
113
54
48
114
53
21
* CAPACITORS VALUED
ARE OPTIONAL
ZILOG INC.
LOGIC DIAGRAM
SIB
02-0037-01
SHEET 7 OF 9


ZILOG INC.
LOGIC DIAGRAM
DZ-0037-01
RECORD CHANGE-ECR 274
CAPACITORS, VALUED 390 PF, ARE OPTIONAL
20mA DATA
20mA DATA RTN
20mA REC RTN/CLK
20mA RECVR
TTY TAPE CNTRL RTN
TTY TAPE CNTRL
8251A USART #3



ZILOG INC.

LOGIC DIAGRAM
SIB

C | D2-0037-01 | E