INTERNATIONAL ATOMIC ENERGY AGENCY
UNITED NATIONS EDUCATIONAL, SCIENTIFIC AND CULTURAL ORGANIZATION
INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS
I.C.T.P., P.O. BOX 586, 34100 TRIESTE, ITALY, CABLE: CENTRATOM TRIESTE

UNITED NATIONS INDUSTRIAL DEVELOPMENT ORGANIZATION
INTERNATIONAL CENTRE FOR SCIENCE AND HIGH TECHNOLOGY
c/o INTERNATIONAL CENTRE FOR THEORETICAL PHYSICS 34100 TRIESTE (ITALY) VIA GRIGNANO, 9 (ADRIATICO PALACE) P.O. BOX 586 TELEPHONE 040-224572 TELEFAX 040-224575 TELEX 460449 APH I

SMR/474 - 19

COLLEGE ON
"THE DESIGN OF REAL-TIME CONTROL SYSTEMS"
1 - 26 October

CASE STUDY

C.S. ANG
Computer Laboratory
University of Cambridge
New Museums Site
Pembroke Street
Cambridge
UK

Case Study: A Real-time Traffic Monitoring System for a High Speed LAN

C S Ang

October 1990

Abstract

This is a case study based on a small self-contained system for the purpose of network traffic monitoring. The aim is to illustrate the design and development of a microprocessor-based system suitable for one type of real-time application, viz. the monitoring of data packets on a high speed local area network. Both the hardware and software aspects of the system are discussed.

1 Introduction

This is a real-time application in which a front-end processor, consisting of some dedicated electronics and a microprocessor system, is used in conjunction with a general purpose computer. It is a typical example where commercially available products are either inadequate in performance or too expensive in cost for the required application and consequently the front-end processor has to be designed and implemented.

The basic function of the system is to monitor the performance of an *asynchronous transfer mode* (ATM) network operating at 75 – 100 Mbits/s for a multimedia environment including real-time video applications such as videophony, video conferencing and broadcast. Network statistics are collected at three levels. The integrity of the network is first checked by a packet or cell synchronization circuit. Basic network data such as number of packets, cyclic redundancy check (CRC) errors, packet repetitions form the second group of network statistics. At a third level, more specific information on the packets are collected. These includes cell full/empty condition, source and destination addresses, and types of payload.

A packet interval timer, with one-microsecond resolution, records the occurrence of a particular type of packets, e.g. packets in a video stream from a particular station to another. This allows transmission jitter, which is a critical parameter in the design of real-time application, to be determined. A real-time calendar clock provides time and date information for the various events.

The system consists of a *Sync & Capture Module* (SCM), a *Microprocessor Front-end Unit* (MFU) and a microcomputer as shown in Fig. 1. One end of the sync & capture module is connected to a target network, in this case a *Cambridge Fast Ring* (CFR) operating as an ATM network, via a station *repeater* chip. The other end of the SCM consists of input/output (I/O) ports for function control and a group of output signals corresponding to various events. The I/O ports allow internal registers in the SCM to be accessed for functions such as the setting of packet *mask* and *template*, and the control of a *Content Addressable Memory* (CAM). The I/O ports and the events outputs are connected to the microprocessor front-end unit.

The microprocessor front-end unit controls the operation of the sync & capture unit, records events, and passes partially processed data to the microcomputer. It handles the tasks of continuously monitoring twelve event channels each operating at a minimum time interval of 3 μs, and measuring packet inter-arrival periods at one microsecond resolution. These are stringent real-time requirements that cannot be easily met with on small standard microcomputers.

The microcomputer provides user interface, overall control of the system function, data processing and storage. With the SCM and MFU taking care of the real-time monitoring and control requirements, the response time or the microcomputer can now be in the order of seconds instead of microseconds. Thus although it is basically a real-time system, the timing requirement is such that computers like the IBM PC or ROSY Junior can be used.

In the following sections, a brief description of the functional requirements and hardware description of the traffic monitor is first given. This is followed with a discussion on the design and development of the MFU and the necessary software.

2 The Functional Requirements

In order to discuss the design and implementation of the hardware and software of the system in the subsequent sections, a brief understanding of the functional requirements is necessary. Basically a traffic monitor in a computer network collects network traffic statistics for the purpose of status reporting, performance analysis, network control, etc. The amount of data captured obviously depends on a particular application. In some cases, traffic information may be derived from the *medium access control* (MAC) unit of a standard station in the network or from a dedicated *monitor* station. Traffic monitoring in such cases merely involves the programming of the MAC unit of the *station* chip and no extra hardware is required.

In other cases, due to speed and other design constraints, existing stations are incapable of providing the required network traffic information, and consequently a dedicated monitoring system consisting of both hardware and software must be implemented. This is the case under consideration here.

2.1 Cambridge Fast Ring (CFR)

This is a slotted ring local area network designed to operate at 75 – 100 Mbits/s with a slightly different packet format and transmission protocol form the *Cambridge Ring*, its predecessor. The main features of the ring are as follows:

- 10 – 100 Mbits/s
- Source-release pass-on-free slotted ring
- Link: twisted pair or optical fibre
- 32-byte data field
- 1983 (design), 1984 (VLSI implementation), 1986 (operational)

This LAN lends itself well to operation in ATM mode. At 75 Mbits/s and using a round-robin empty slot protocol, a system with 20 stations has a best case point-to-point transmission speed of about 20 Mbits/s and the worst case of 3 Mbits/s. The corresponding delays of the cells through the ring are 20 μs and 100 μs.

The cell or packet format of the CFR is shown in Fig. 2. It consists of a 4-bit header, two 16-bit destination and source address fields, a 256-bit data field and a 12-bit CRC field which also acts as a response flag. The flags in the header are *start-of-slot, full/empty, monitor-passed* and *spare*.

A typical node structure is shown in Fig. 3. An *ECL Repeater Chip* performs serial to 8-bit parallel conversion and the function of line driver and repeater. Network control is performed in a *CMOS Station Chip* which takes care of most of the data link functions. The functional block diagram of the repeater and station chips are shown in Fig. 4 and 5 respectively.

There are many ways of configuring a CFR network. An distributed system may consist of a number of interconnected CFRs and bridges to Ethernets and ISDN etc. An example is the *MediaNet* (MNet) currently being implemented at Cambridge for multimedia experiments with special emphasis on real-time video services as shown in Fig. 6.

2.2 Network Parameters

In theory network performance and the corresponding control strategies may be obtained by logging and analyzing all the traffic in a network. In practice however a number of assumptions has to be made to reduce the amount of data to be monitored or analyzed. Basically important network parameters to be determined are as follows:

- Throughput or bandwidth, both point-to-point and network overall. Fairness in sharing the total bandwidth.
- Cell delays as a function of network loading. This is an important parameter in real-time services.
- Error rates under various network conditions. These have different implications on real-time traffic and computer data. For example, a certain amount of errors may be tolerated in real-time video and voice applications if suitable coding techniques are used. Errors in computer data normally have to be corrected by higher layer protocol either with forward error correction schemes or retransmission.

These parameters are strongly influenced by the type of network, protocol, and application. In this case study we are interested in the

performance of CFRs in a real-time environment. Some of the more specific parameters to be monitored and analyzed are as follows:

1. Total cell rate or bit rate.
2. Number of full and empty cells.
3. Number of good CRC cells.
4. Number of inverted CRC cells.
5. Number of bad CRC cells.
6. Total traffic from a particular source or group of sources.
7. Total traffic to a particular destination or group of destinations.
8. Traffic of a particular bit pattern.
9. Number of cell repetitions.
10. Inter-arrival time of a particular type of cells.
11. Error rates.
12. Receiver contention condition.

2.3 Design Considerations

Some of the parameters mentioned above are available in the network controller chip and in fact some statistics are collected a a CFR *monitor* station in all networks. Many parameters however are not accessible normally and have to be monitored with a dedicated *traffic monitor*.

In principle the line bit rate of 75 – 100 Mbits/s is not particularly high and standard logic timing analyzers are capable of monitoring the data stream. However, they lack the flexibility in evaluating the host of required network parameters and are generally considered too expensive when several units have to be deployed in a network simultaneously. A dedicated system was thus designed and implemented.

Essentially the entire task is split into three levels according to processing speed: (1) a dedicated random logic module performs tasks in the *microsecond* range, (2) a microprocessor takes care of real-time tasks in the *millisecond* range, and (3) a microcomputer deals with tasks in larger time frame. These are itemized as follows:

- A high speed unit referred to as the SCM earlier performs the following tasks:
  - Serial–parallel conversion to reduce the speed to one eighth the serial line rate.
  - Cell and gap synchronization.
  - CRC generation and comparison for each cell.
  - Cell *mask* and *template* read/write and control.
  - Cell repetition event detection.
  - Matched events detection.
- A microprocessor-based system, the MFU, to relieve the last stage microcomputer of some real-time tasks:
  - Count 10 event channels, with 3 μs inter-event arrival time.
  - Provide a channel for time measurement with 1 μs resolution.
  - Program a set of 128 templates and masks, each of which is 304 bits long.
  - Program a content addressable memory for the purpose of high speed cell repetition detection.
  - Interface to a microcomputer for control, data presentation and storage.
- A final stage consists of a microcomputer. As pointed out previously, the response time in this stage is from seconds onwards. Connection to the MFU is through a standard serial link.

3 The Hardware

The hardware is conveniently divided into three distinct sections as shown in Fig. 1. As the SCM is rather specific to this particular application, only functional description instead of design details, of the various subunits is given. However, to illustrate the general principles of designing and implementing a small microprocessor system, sufficient design, testing and debugging information for the MFU are described in a subsequent section below. Following the SCM the microcomputer is also briefly mentioned.

3.1 Sync & Capture Module

A block diagram of the SCM is shown in Fig. 7. A repeater chip operating in *bypass* mode provides a means of tapping into the network. Its main output to the rest of the SCM are 8 data lines and a *byte clock*.

A synchronization unit extracts timing signals at cell or packet level by means of a modulo-38 counter and some random logic. These timing signals are used for timing CRC generator and checker, clocking comparator in pattern matching circuits, and providing start-of-cell and start-of-gap indications.

The next functional block is a CRC generator and checker. It uses a 4-bit wide generator to reconstruct the 12-bit CRC in every packet of $304 - 12$ bits. This is then compared with the incoming CRC. There are three possibilities. A match shows that the packet has been put on the network by a transmitter but has not been received. A match of the CRC with the last 4 bits flipped indicates that the packet has been received and is on its return path to the transmitter. A mismatch indicates a corrupted packet or out-of-sync network.

A packet matching unit provides a set of 128 templates and masks and the corresponding comparator circuitries. A template contains a bit pattern to be matched whereas a mask provides the *don't care* condition to be imposed on individual bits of the packet. It is possible to selectively match a section or the entire packet. In this implementation the following matching are performed: header (4 bits), header & destination (4 + 16 bits), header, destination & source (4 + 32 bits), header, destination, source & data (4 + 32 + 256 bits). The switching of template/mask is done by the MFU. In a typical application, template/mask patterns are down-loaded from the MFU and then switched in one at a time for an appropriate period of time. This results in sampling the network for a number of traffic patterns. A signal for measuring the inter- arrival time of a particular packet also originates from this unit.

The final unit in the SCM is a *repeats* checker. Repetition in transmission in data link layer is an important network parameter. In network such as the CFR, automatic retransmission is done by hardware, i.e. in the network controller chip, without the intervention of higher layer protocol. This particular service is found to be undesirable in real-time application where a retransmission usually has the some effect as lost packet due to long delay. Moreover, retransmission generates more traffic and may result in overall performance degradation.

The detection of packet retransmission is conceptually easy to do but rather difficult to implement in practice due to the large amount of data and the high speed of on-the-fly processing required. For example if a packet is to be matched against the previous 100 packets, this amounts to comparing a 304-bit pattern with $100x304$ bits in about 3 μs, and loading the incoming packet into the existing buffer. A *content addressable memory* which is designed specifically for such an application is used. However, instead of comparing the entire packet, only the CRC of each packet is used. The CAM is 256 words deep and therefore a repeat of cell within the last 256 cells can be detected.

3.2 Microcomputer

A microcomputer serves as an user interface in two stages. First it acts as a simple development tool by providing cross assembly facility and acting as a host machine for testing and debugging the MFU and SCM. When the entire system is in operation, it serves as a data storage as well as an user interface. The connection to the MFU is a RS232C serial link which is available as a standard feature in most microcomputers. In this case study, it is an Olivetti M380/C; but most other microcomputer including the ROSY Junior may be used.

4 The Design and Development of the MFU

The choice of the front-end processor depends essentially on the speed and storage requirements of the tasks in hand. In this application an 8-bit microprocessor (MC6809) is used. It is a small single board unit consisting of 12 VLSI chips including microprocessor, memories, peripherals and a number of small supporting i.c.

The essential requirements are:

- 10 event counters. These counters must count at a speed up to 330 KHz and accumulate total counts for a relatively long period of time. The maximum number of cumulative events over a period of one year is about $3x10^{14}$. It is implemented with four *Programmable Timer Modules* (MC6840 PTM) each of which has three separate counters of 16 bits each. Each of these internal counters is extended with a 48-bit external one using a non-volatile battery backed-up CMOS RAM.

Since the microprocessor must service all the event counters simultaneously, one must ensure that there is enough time to update them if they require attention at the same time. This is made possible partly by the auto re-initialization feature of the PTM.

- An interval timer to measure the inter-event arrival time and classify them by time. This is also achieved with a PTM channel operating with an internal clock of 1 μs. Thus packet arrival jitter can be measured with 1 μs resolution, and a maximum period of 65 ms.

- A parallel I/O port for controlling and monitoring various functions in the SCM. This is done by a standard peripheral chip, the *Programmable Interface Adapter* (MC6821 PIA).

- A real-time clock serves the purpose of time stamping the various events. Without it the microcomputer has to perform a large amount of tight timing chores. A MC146818A *Real-time Clock* is used with an external standby battery power supply. In the event of mains failure, this clock will be kept running.

- A non-volatile RAM for critical data. This is implemented with a MK48Z02 2K RAM which has a built-in battery.

- A ROM and RAM for system and user programs. These are provided by a 16K EPROM chip and a 32K RAM chip.

- A serial interface to the microcomputer. An *Asynchronous Communications Interface Adapter* (MC6850 ACIA) and its associated baud-rate generator and line drivers are used.

- A set of switches and indicator for the purpose of system debugging, optional function and parameter selection, and status indication. This is done with a PIA.

A block diagram of the MFU is shown in Fig. 8.

4.1 MFU Configuration and Circuit Description

The next step in the design is the configuration of the MFU. A simple procedure is to use a table as shown in Table 1 to determine the various address line connections. The resulting memory map is as follows:

Device		Address
Name	Type	Range
RAM1	62256	$0000 -- $7FFF
RAM2	48Z02	$8000 -- $87FF
PIA1	6821	$9010 -- $9013
PIA2	6821	$9020 -- $9023
PTM1	6840	$A010 -- $A017
PTM2	6840	$A020 -- $A027
PTM3	6840	$A040 -- $A047
PTM4	6840	$A080 -- $A087
RTC	146818	$B000 -- $B040
ACIA	6850	$B010 -- $B011
EPROM	27128	$C000 -- $FFFF

Based on the configuration table, the actual address decoding is done as follows:

1. Address line A15 is connected directly to the chip select $\overline{CS}$ of RAM1 thereby defining an address space from $0000 to $7FFF.
2. A15 and A14 are decoded into four 16-K blocks using a 2-to-4 decoder. The active low output of the highest block ($C000 -- $FFFF) is connected to the $\overline{CE}$ of the EPROM.
3. The block $8000 -- $BFFF is further decoded with A12 and A13 into four groups of 4 K each at $8XXX, $9XXX, $AXXX and $BXXX. These four groups are assigned to RAM2, PIAs, PTMs, and ACIA & RTC respectively.

Since it is a small system of 12 chips in total, bus lines are directly connected from the MPU to memories and peripherals without using external buffers or bus drivers. Thus data bus is straightforwardly connected together while $R/\overline{W}$ from the MPU is connected to all other chips except the EPROM.

The other control, status and timing lines from the MPU are connected as follows:

- The E clock from the MPU is directly connected to the family peripheral devices (ACIA, PIAs and PTMs). The RTC requires special treatment. A small decoding circuit using E, A2, $R/\overline{W}$ and $\overline{\$BXXX}$ as inputs, produces the required address and data strobes (AS and DS) for the RTC.
- BS, BA and Q outputs are not used and thus left open.
- $\overline{BREQ}, MRDY$ and $\overline{HALT}$ features are also not used. They are tied to 5V through resistors.
- Interrupt inputs $\overline{NMI}$, $\overline{IRQ}$ and $\overline{FIRQ}$ are pulled up to 5V and selectively connected to the peripheral devices by jumpers.
- A 4-MHz crystal is connected to the oscillator inputs ($XTAL$ and $EXTAL$) to generate a 1-MHz system clock.

Circuit diagrams of the complete MFU are shown in Fig. 9 – 14. A small watch-dog circuit (Fig. 9) consisting of two retriggerable monostable multivibrators and a $\overline{WD}$ signal from $CA2$ of PIA2 is used to revive the system if the MPU hangs for some reasons. It works by attempting to reset the MPU if $\overline{WD}$ is not received within a certain time interval, thereby providing a certain degree of protection against program running wild or getting into erroneous tight loops.

In the ACIA circuit, a baud rate generator (4060) and a crystal provide commonly used baud rates from 300 to 9600 while a single power supply line-driver is used for some of the RS232C signals.

The PIA2 has an input port consisting of 8 on/off switches and a output port of 8 LEDs for status reporting and function selection.

4.2 Testing and Debugging

Once the circuit design is completed, the next step is circuit board layout and fabrication. Unfortunately the development process does not end there. In most cases a certain degree of hardware testing and debugging must be done. To carry out these jobs, it would be advantages if sophisticated tools such as development system, in-circuit emulator, and logic analyzer are available. However, it is possible to test and debug with the basic electronics laboratory equipment such as multimeter, oscilloscope and function generator alone, if a systematic approach is adopted. An outline of the testing procedure is shown below.

1. Printed circuit board (PCB) inspection for track continuity, and possible bridging. This is a step that is often overlooked. However,

it is a vital step because easily locatable faults if left undetected, usually cause much more debugging efforts at a later stage.

2. Power up the bare PCB and check voltages.

3. Force a NOP ($12) on the data bus by pulling up D1 and D4 to 5V via resistors and grounding all other data lines. It causes the continuous execution of NOP for all memory locations. This in turn results in A0 toggling at half the clock rate, A1 toggling at half the rate of A0 and so forth. The address bus can thus be checked easily with an oscilloscope. In this test, data bus and control bus are also partially verified.

4. If a logic analyzer is not available, implement a tight loop program in the EPROM such as a branch-to-itself loop (LOOP BRA LOOP). This program consists of two bytes ($20 $FE) and takes three MPU cycles to execute. A two-byte reset vector is also needed in the EPROM. The execution of this very short program can be followed cycle by cycle on an oscilloscope and thereby confirming the proper operation, at least partially, of the data and control bus.

5. Test routines for a PIA with input switches and output indicators can be written into the EPROM and executed. Commonly used routines include incrementing the binary value of the output port at a slow rate for visual inspection, reading status of switches and output it to the output port. This stage of testing serves to verify the operation of the PIA and to provide user functional selection. Normally on power up the system is programmed to check the status of the input switches and jump to appropriate test routines or main program.

6. Small test routines for other components in the system are then implemented. This includes testing the ACIA, PTMs, RTC and RAMs.

7. A monitor program is then implemented. At this stage most of the hardware testing are done and the task usually moves on to user or target software debugging. One type of hardware bug which cannot be detected by standard testing are those caused by intermittent faults, glitches or external interference. These are

captured by means of logic analyzer or in-circuit emulator running in some surveillance or baby-sitting modes.

5 The Software

The software of the system consists of programs in the MFU and microcomputer. The MFU program is written in assembly language while that in the microcomputer is in high level languages (QuickC and QuickBASIC). The following section describes the development of the program in the MFU and gives an outline of the functions of the user program in the microcomputer.

5.1 Assembly Language Program in the MFU

Except for very small tasks, the development of assembly language programs normally requires some form of specialized tools. As in the case of hardware development, if sophisticated tools such as simulator and development system are available, they are of great help in software development. In situations where such tools are not available, software development usually involves the use of a cross-assembler or compiler on a computer, and down-loading and EPROM programming facility.

In this case study, an example of software development based on relatively simple tools of a cross-assembler on an IBMPC or compatible, an EPROM programmer and an MC6809 in-circuit emulator is described.

For most assembly language program development, a *monitor* program in the target machine, at least in the development stage, helps a great deal in program modification, testing and debugging. This monitor may be written specifically for the task. However since most of the functions in this monitor are standard functions, one can implement an existing and fully tested version. This is the case here. An M6809 monitor program by Motorola, the ASSIST09, is used. This monitor and the application program are described below.

5.1.1 The ASSIST09

This is a small M6809 monitor program that is usually co-resident with user program in an application. The entire 2K program normally resides on the upper section of the memory space; but it can also be put at other locations since it is position independent. A complete source listing

together with documentation can be found in the *Motorola MC6809 - MC6809E Microprocessor Programming Manual, Appendix B.*

The main features are:

- Position independent codes.
- A set of 15 commands for program development including breakpoint and trace.
- Sophisticated monitor calls for address independent user program services.
- Multiple means of installing user modifications and extensions.
- Hooks for user command tables, I/O handlers, and defaults specifications.
- Easily adapted to run under control of a real-time operating system.

A RAM work area is located relative to the start of ASSIST09 by an offset of -$1900. The default console I/O handlers access an ACIA located at $E008. For trace commands, a PTM with default address $E000 is used to force an $\overline{NMI}$ so that single instructions may be executed. These default addresses may be easily changed using one of several methods.

A vector table is used to address certain service routines and defaults values. This allows user to change the information and the operation of the monitor by using a vector swap service. User routine may also reside at an extension ROM located 2K below the ASSIST09. The vector table initialization subroutine looks for a flag at this address, and if found calls the location following the flag as a subroutine. Any or all defaults may be altered at this stage.

The default system interrupts are:

RESET Build vector table, setup monitor defaults and invoke monitor startup routine.

SWI Request a ASSIST09 service.

FIRQ RTI.

SWI2 Reserved.

SWI3 Reserved.

IRQ Reserved.

NMI Force a breakpoint.

A complete command list with brief descriptions of each command is given in Table 2. A number of services are also provided. These are invoked by using the SWI instruction followed by one byte function code. Table 3 lists all the services that may be used.

Another service provided by the ASSIST09 is the vector swap. The service allows user modifications of the vector table to be easily installed. A list of all the entries is shown in Table 3.

There is a provision for user to add or modify commands to the existing set. The command handler scans two lists, a primary table and a secondary table. The primary table defaults to the ASSIST09 commands and the secondary table to a null list. Each entry in the table consists of a byte for the size of the entry, a command name string, and an offset to the entry point of the command.

5.1.2 The Traffic Monitor Program

The MFU assembly program runs under the control of ASSIST09. It basically consists of a set of functions which can be invoked from the microcomputer like other commands. In normal operation the NMI routine of the ASSIST09 is modified for real-time tasks such as updating internal clock, event counters etc. The program thus begins by initializing the secondary command table vectors and swap the NMI vector.

The NMI routine updates the PTM extended counters on each PTM re-initialization which causes an interrupt. Depending on the function selected, it also switches in the next template/mask in the multiple template/mask mode. This routine basically takes care of the real-time function of the system.

Some of the command functions implemented are:

- INITPIA — Initialize PIAs.
- PIAT1 — Test PIA.
- INITPTM — Initialize PTMs.
- RPTM — Reset PTMs.

- OFFPTM — Disable PTMs.
- PTMT1 — Test PTM by printing all counters.
- CFRTS — CFR traffic simulator mode.
- WT — Write Template.
- RT — Read Template.
- WM — Write Mask.
- RM — Read Mask.
- Q — Request for one PTM data block.
- INITCAM — Initialize CAM chip.
- PEEK — Write to MFU memory or I/O.
- POKE — Read from MFU memory or I/O.
- MTM — Multiple Template/Mask mode.
- STM — Single Template/Mask mode.

5.2 The High Level Language Program

The program in the microcomputer (a) provides an user interface to the system, (b) logs, analyzes, and stores collected data and (c) provides data to other stations or users connected to the network. Two modes of operations are available.

The first is an terminal emulator or program development mode. In this mode an user interfaces directly with the ASSIST09 monitor. All the standard machine language debugging features are available. These includes down-loading of object code, saving of target code in disk, execution or tracing of codes etc. One can also shell into DOS for other tasks that require other facility such as editing and assembling. In a typical developing phase, an Editor is used for source creation or modification. Then a cross-assembler is used to generate the object code, which is loaded into the MFU memory. Debugging is then done under the ASSIST09 environment. The final program may be either in RAM or in EPROM. Since there ample RAM space (32K) in the MFU,

the RAM version is used. This saves the step of EPROM programming but calls for down-loading of the code on each power-up.

The second mode provide window user interface to the traffic monitor. It allows (a) various MFU parameters to be set and (b) traffic information to be displayed in different formats: tables, graphs etc., and (c) data logging and storage parameters to be set or modified.

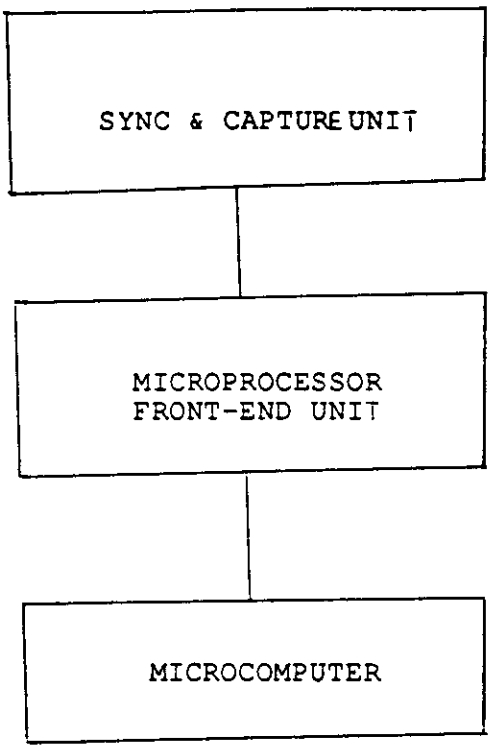


Fig. 1 System layout of Traffic Monitor

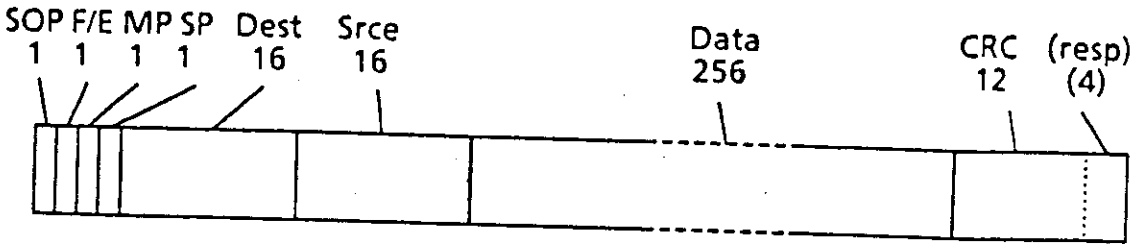


Fig. 2 Packet format of CFR

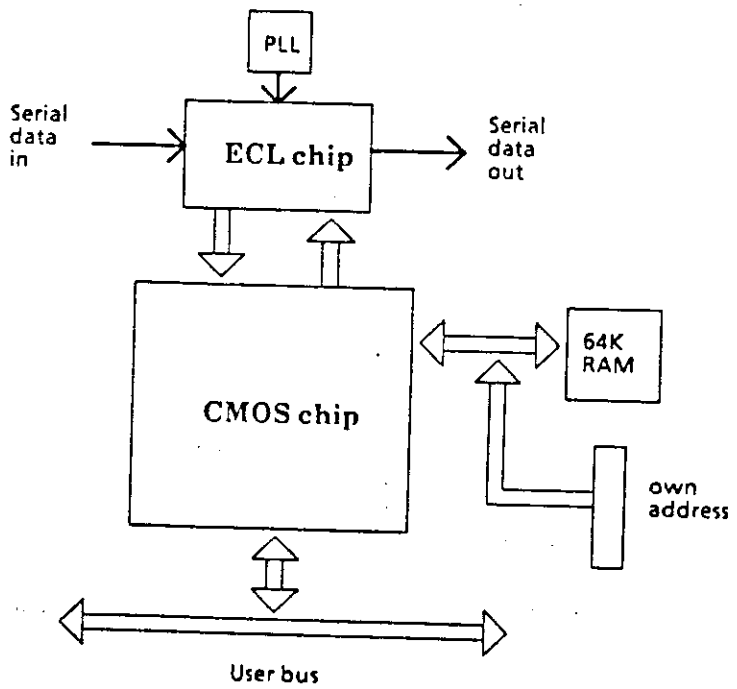


Fig. 3 Node structure of CFR

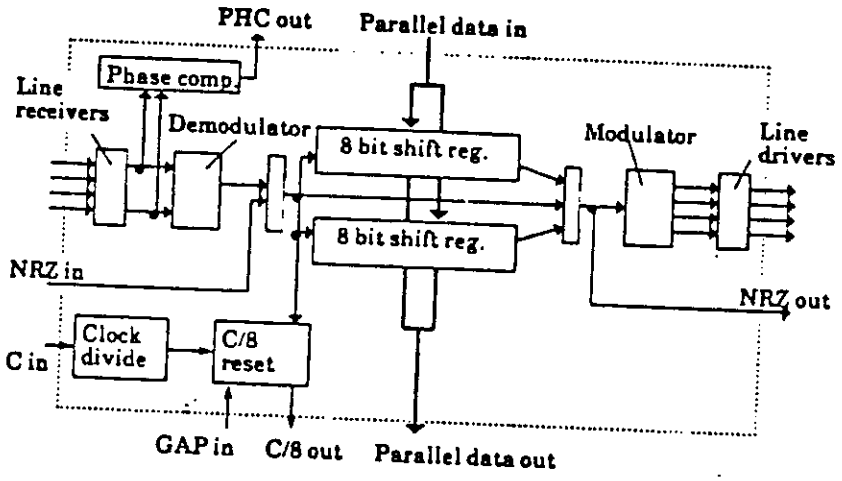


Fig. 4 Repeater chip of CFR

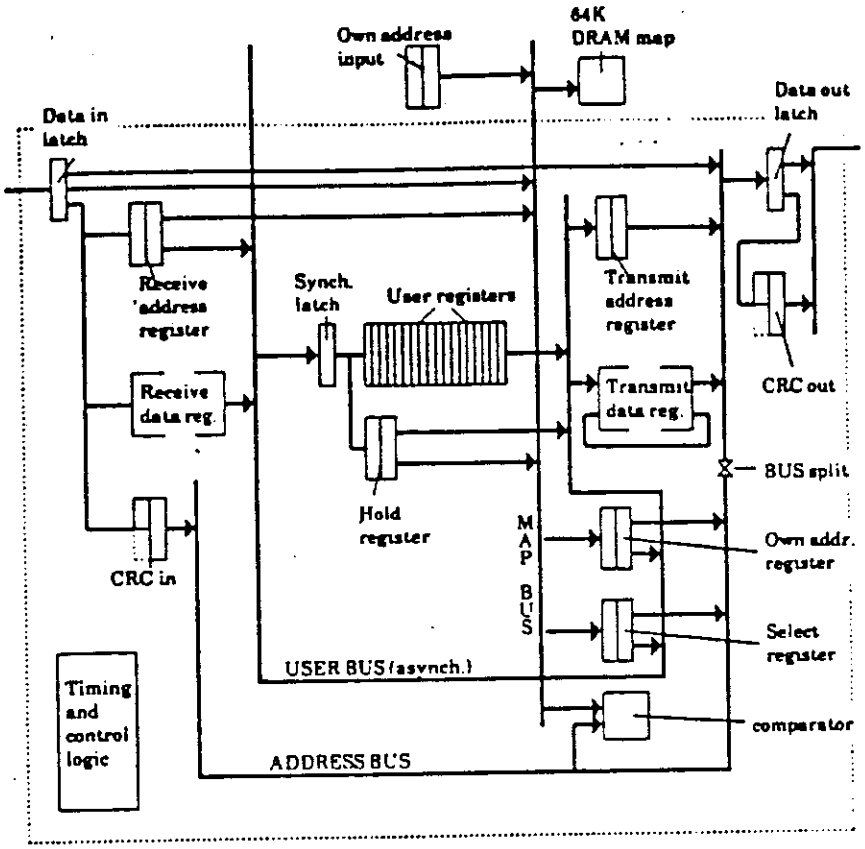


Fig. 5 Station controller chip of CFR

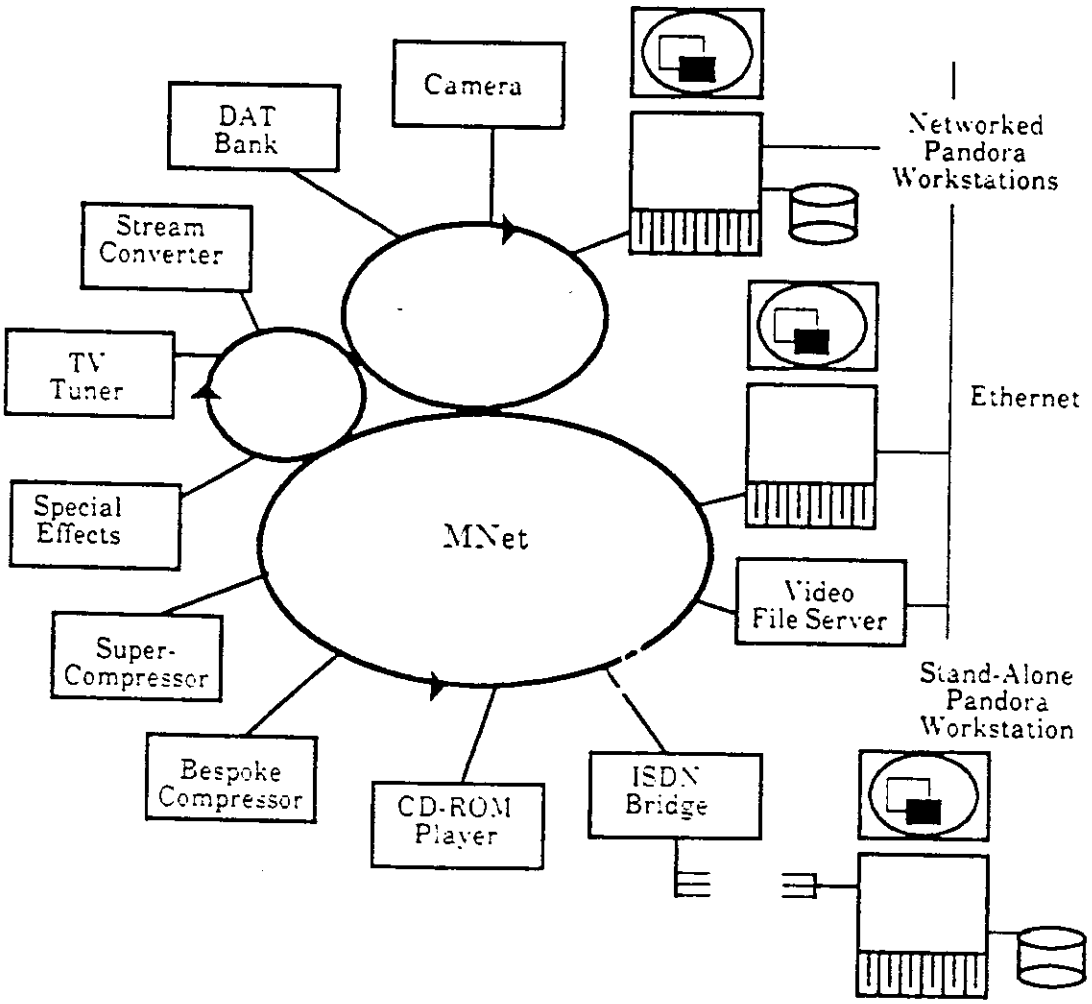


Fig. 6 An example of MediaNet configuration

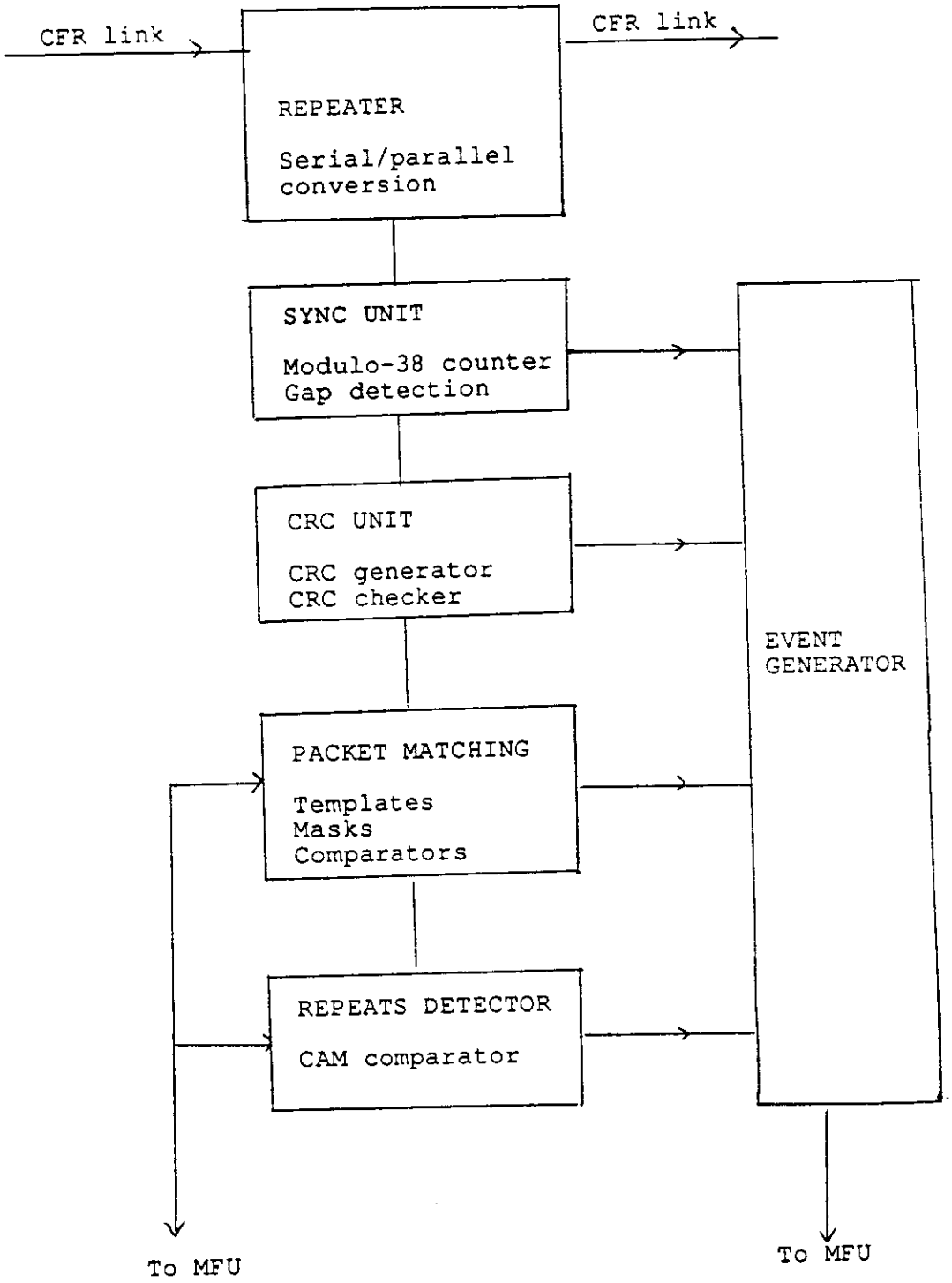


Fig. 7 Block diagram of SCM

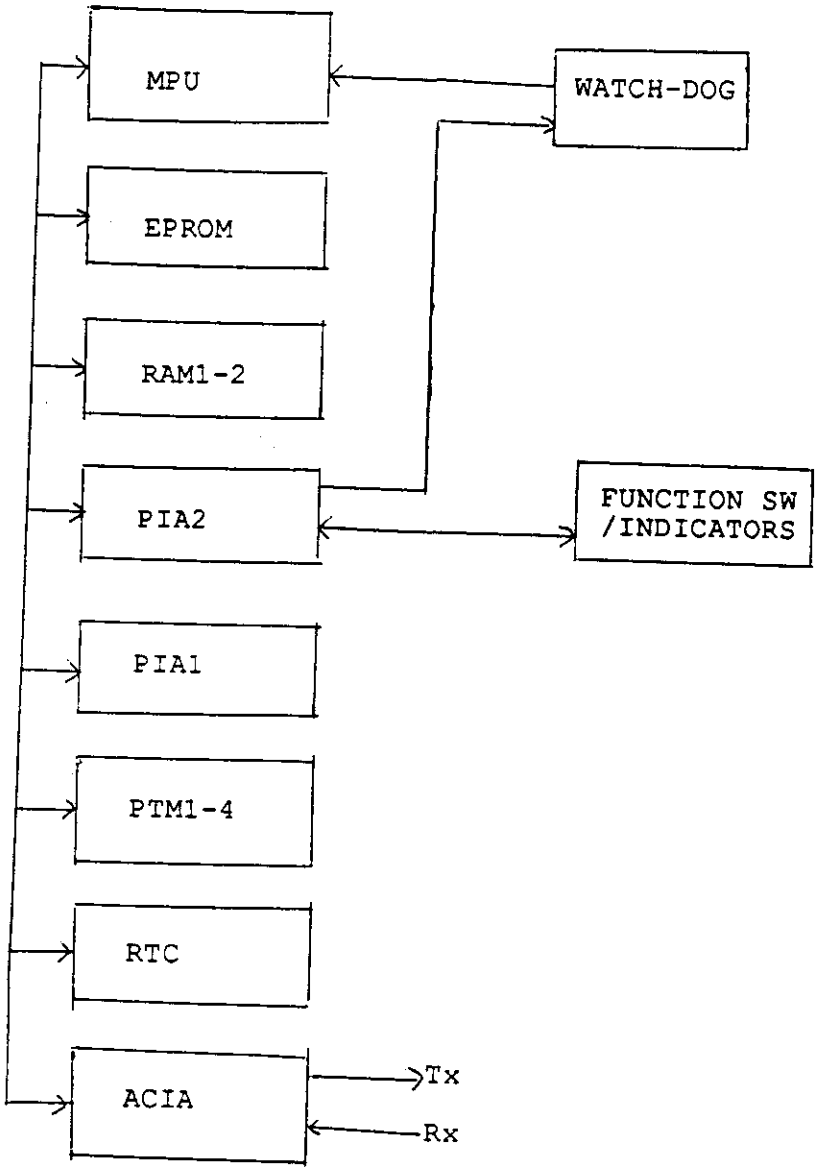


Fig. 8 Block diagram of MFU

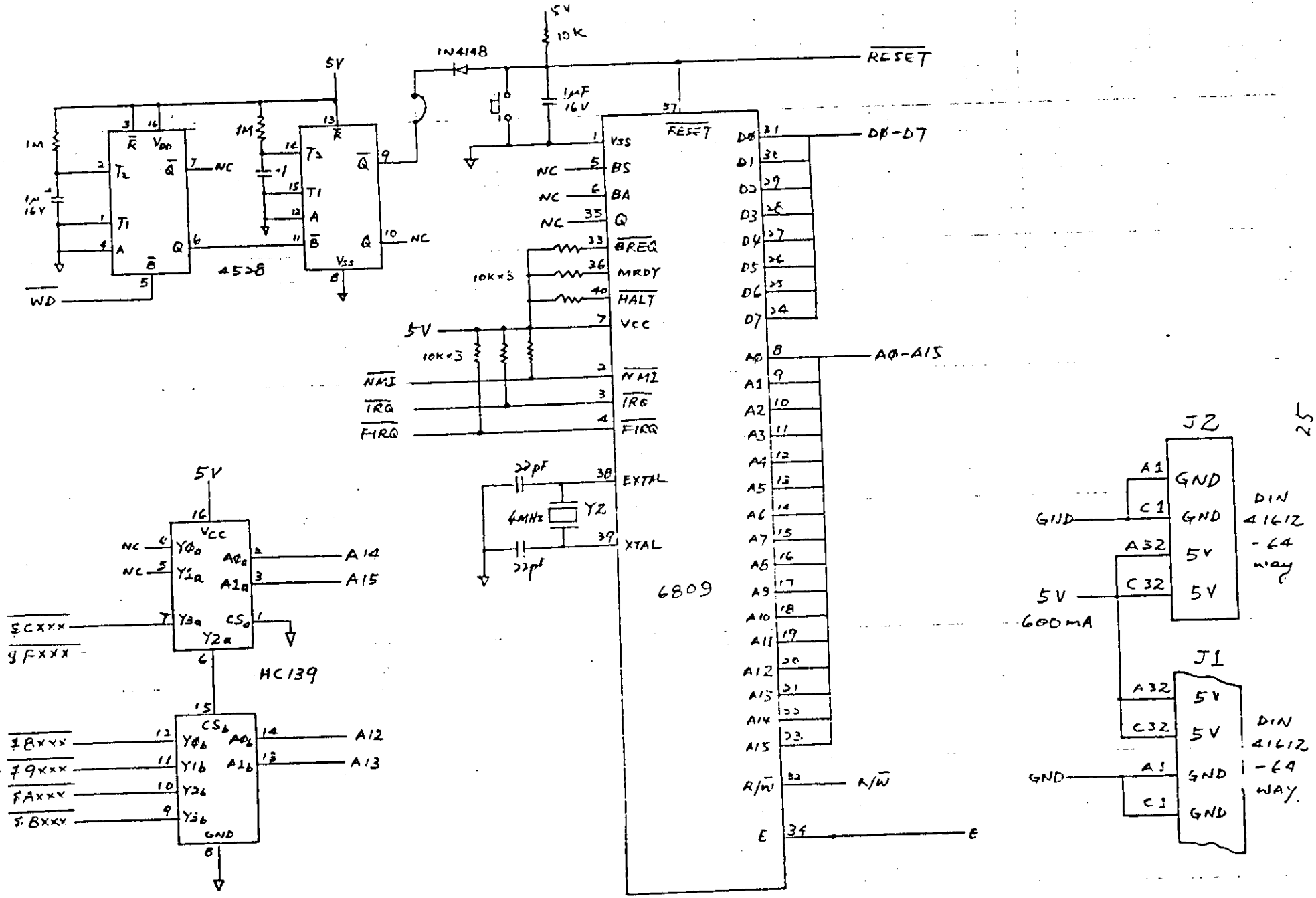


Fig. 9 Circuit diagram of MFU (1)

CAMBRIDGE FAST RING (CFR)

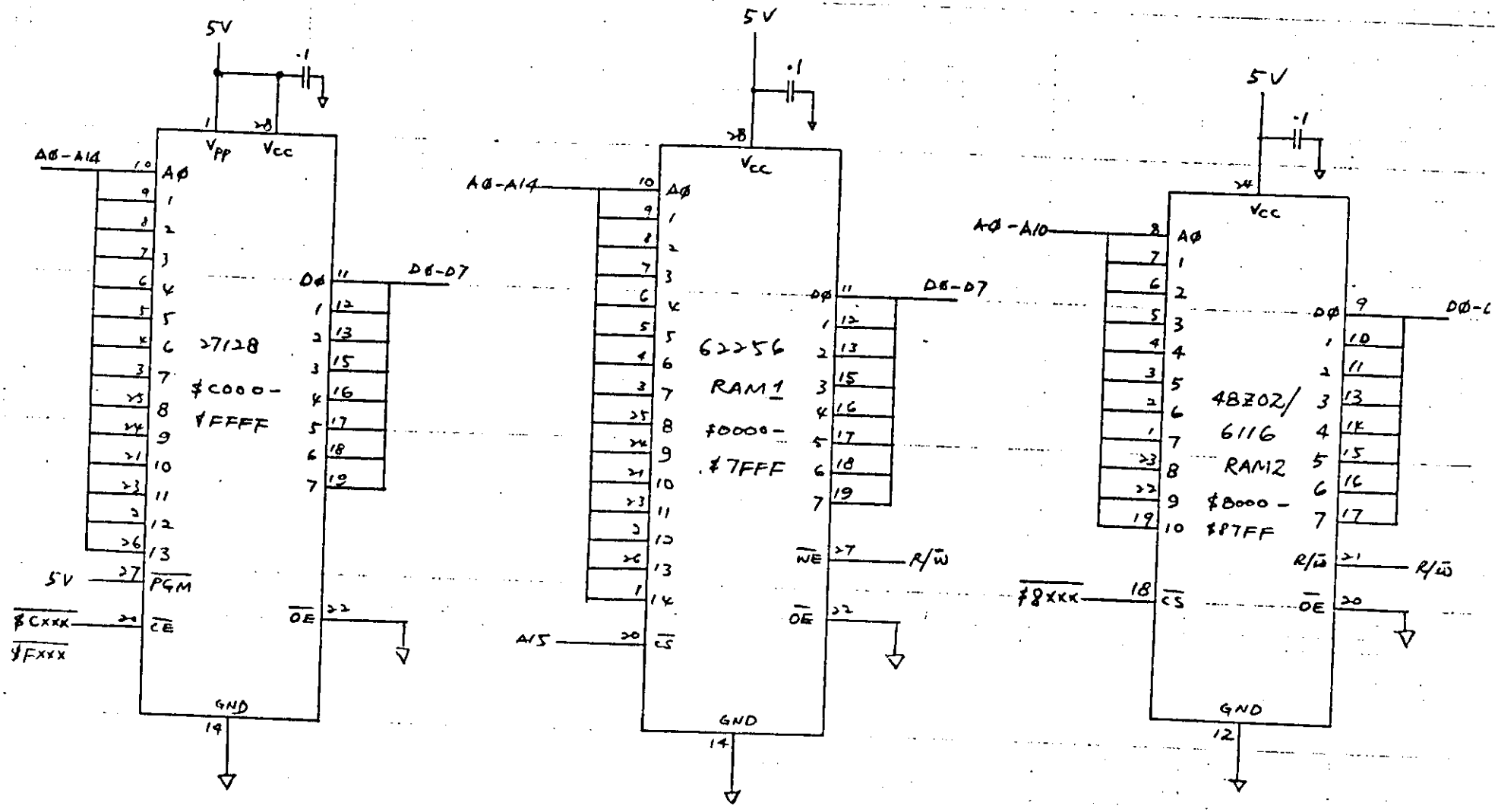


Fig. 10 Circuit diagram of MFU (2)

Pg 2/6
ACS/TKW
8/89

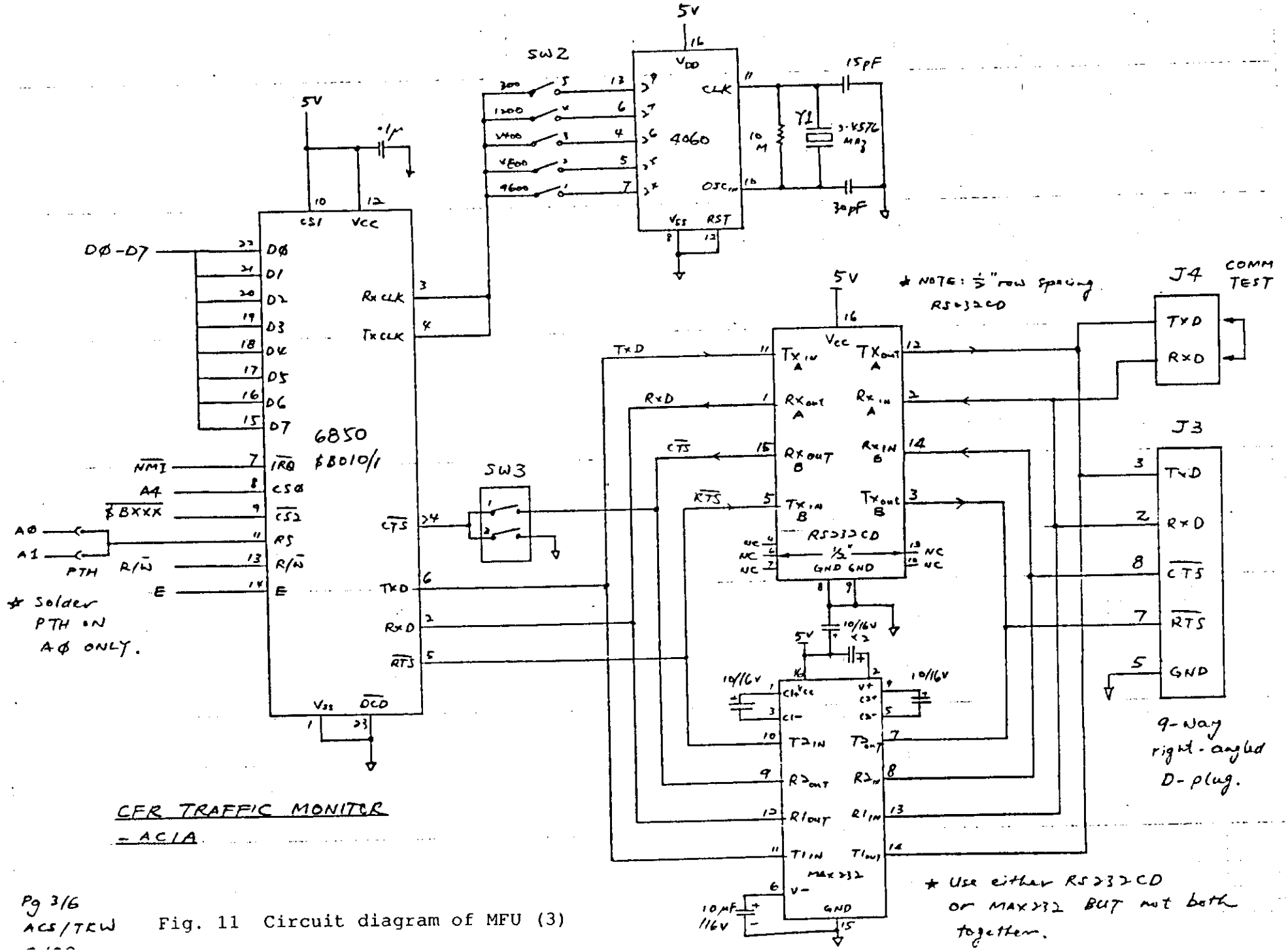
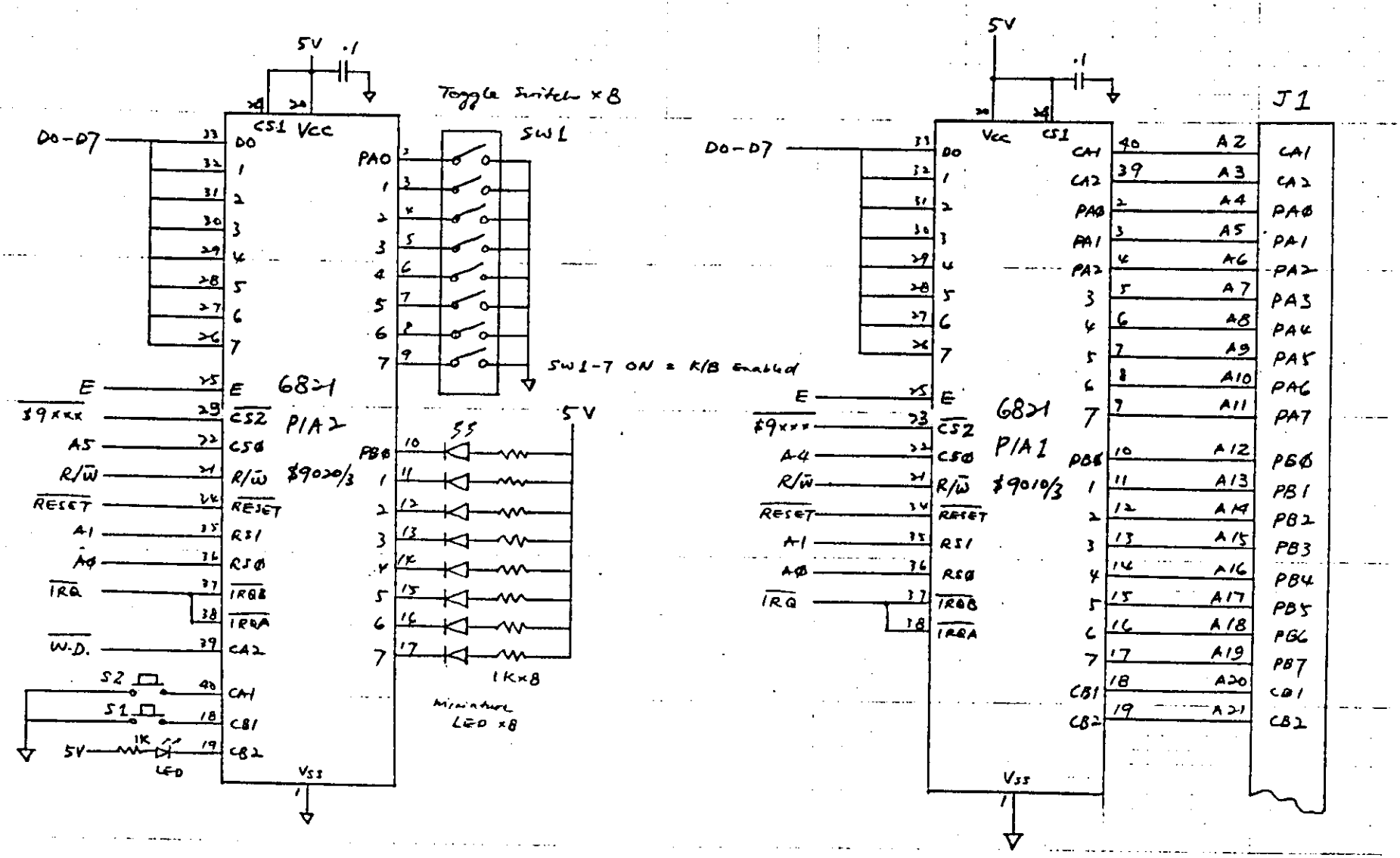
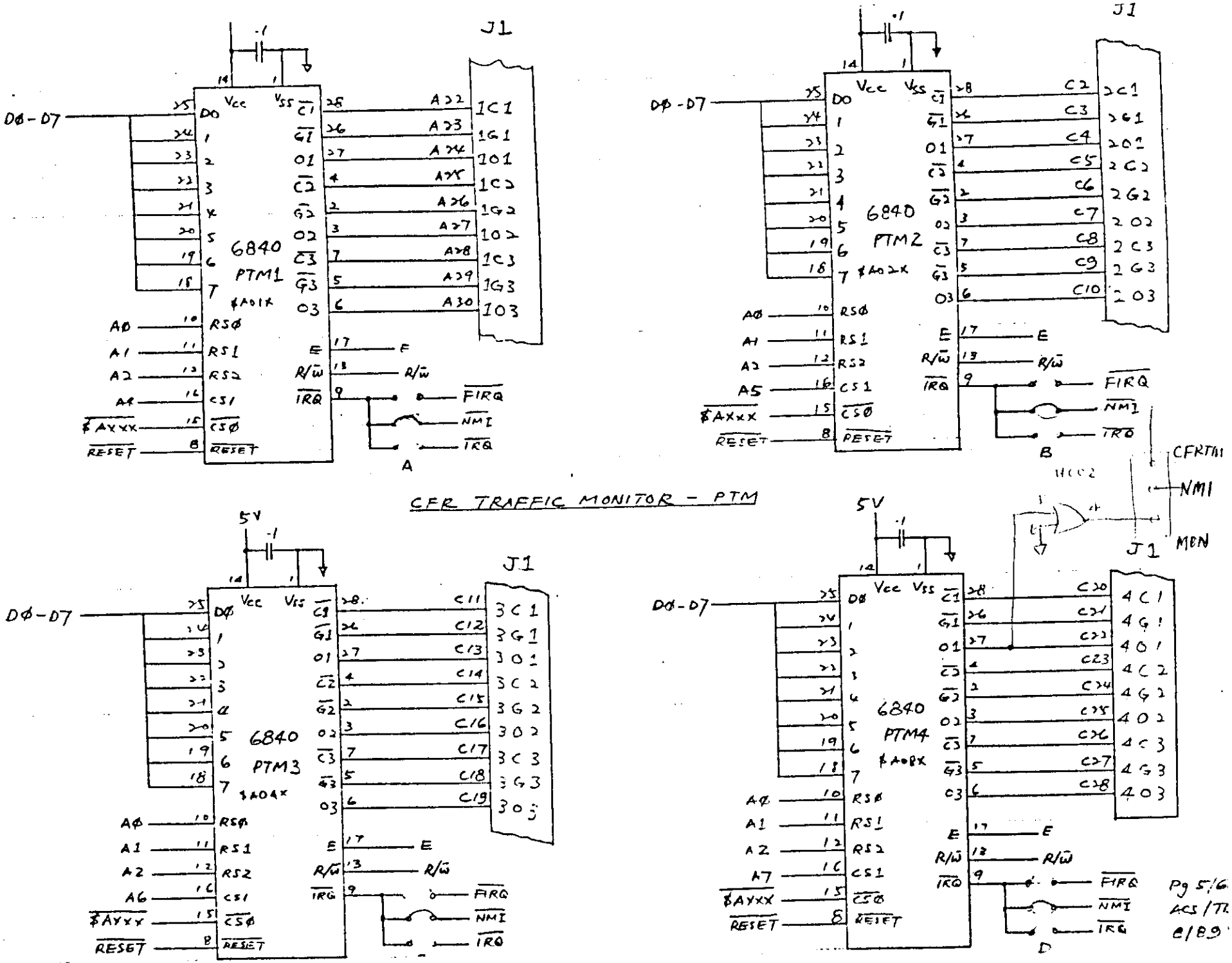


Fig. 11 Circuit diagram of MFU (3)

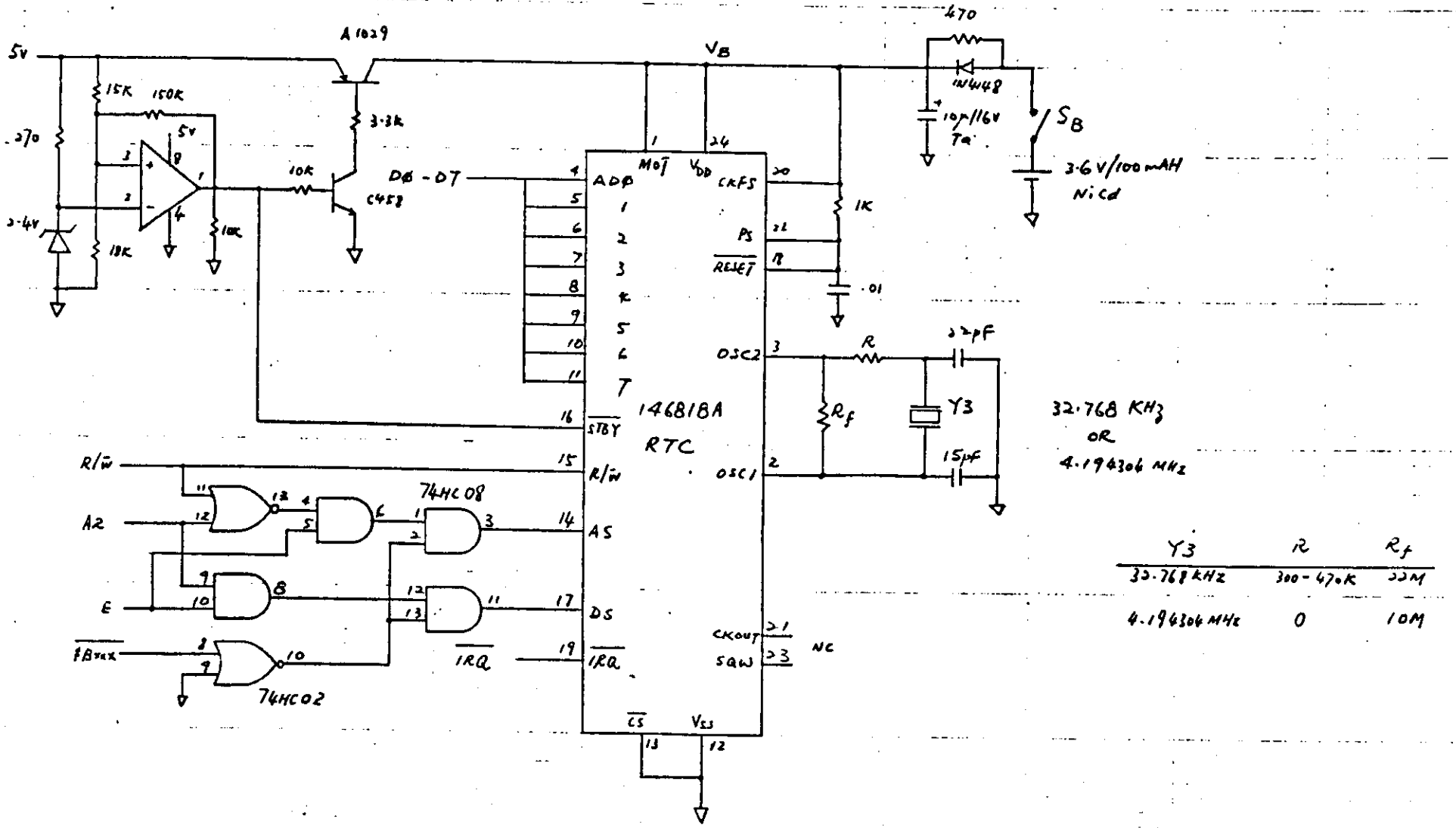


CFR TRAFFIC MONITOR - PIA.

Fig. 12 Circuit diagram of MFU (4)



CFR TRAFFIC MONITOR - PTM



CFR TRAFFIC MONITOR - RTC

Fig. 14 Circuit diagram of MFU (6)

Pg 61C
ACS/TRU
8/89

Device		ADDRESS																Address
Name	Type	15	14	13	12	11	10	09	08	07	06	05	04	03	02	01	00	Range
RAM1	62256	0	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	$0000 - $7FFF
RAM2	48Z02	1	0	0	0		X	X	X	X	X	X	X	X	X	X	X	$8000 - $87FF
PIA1	6821	1	0	0	1								1			X	X	$9010 - $9013
PIA2	6821	1	0	0	1							1				X	X	$9020 - $9023
PTM1	6840	1	0	1	0								1		X	X	X	$A010 - $A017
PTM2	6840	1	0	1	0							1			X	X	X	$A020 - $A027
PTM3	6840	1	0	1	0						1				X	X	X	$A040 - $A047
PTM4	6840	1	0	1	0					1					X	X	X	$A080 - $A087
RTC	146818	1	0	1	1										X			$B000 - $B040
ACIA	6850	1	0	1	1								1				X	$B010 - $B011
EPROM	27128	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	$C000 - $FFFF

Table 1. MFU configuration table

Command Name	Description	Command Entry
Breakpoint	Set, clear, display, or delete breakpoints	B
Call	Call program as subroutine	C
Display	Display memory block in hex and ASCII	D
Encode	Return indexed postbyte value	E
Go	Start or resume program execution	G
Load	Load memory from tape	L
Memory	Examine or alter memory	M
	Memory change or examine last referenced	/
	Memory change or examine	hex/
Null	Set new character and new line padding	N
Offset	Compute branch offsets	O
Punch	Punch memory on tape	P
Registers	Display or alter registers	R
Stlevel	Alter stack trace level value	S
Trace	Trace number of instructions	T
	Trace one instruction	.
Verify	Verify tape to memory load	V
Window	Set a window value	W

Table 2 Command list of ASSIST09

Service	Entry	Code	Description
Obtain input character	INCHP	0	Obtain the input character in register A from the input handler
Output a character	OUTCH	1	Send the character in the register A to the output handler
Send string	PDATA1	2	Send a string of characters to the output handler
Send new line and string	PDATA	3	Send a carriage return, line feed, and string of characters to the output handler
Convert byte to hex	OUT2HS	4	Display the byte pointed to by the X register in hex
Convert word to hex	OUT4HS	5	Display the word pointed to by the X register in hex
Output to next line	PCRLF	6	Send a carriage return and line feed to the output handler
Send space	SPACE	7	Send a blank to the output handler
Fireup ASSIST09	MONITR	8	Enter the ASSIST09 monitor
Vector swap	VCTRSW	9	Examine or exchange a vector table entry
User breakpoint	BRKPT	10	Display registers and enter the command handler
Program break and check	PAUSE	11	Stop processing and check for a freeze or cancel condition

Table 3 List of services of ASSIS09

Entry	Code	Description
.AVTBL	0	Returns address of vector table
.CMDL1	2	Primary command list
.RSVD	4	Reserved MC6809 interrupt vector appendage
.SWI3	6	Software interrupt 3 interrupt vector appendage
.SWI2	8	Software interrupt 2 interrupt vector appendage
.FIRQ	10	Fast interrupt request vector appendage
.IRQ	12	Interrupt request vector appendage
.SWI	14	Software interrupt vector appendage
.NMI	16	Non-maskable interrupt vector appendage
.RESET	18	Reset interrupt vector appendage
.CION	20	Input console intiialization routine
.CIDTA	22	Input data byte from console routine
.CIOFF	24	Input console shutdown routine
.COON	26	Output console initialization routine
.CODTA	28	Output/data byte to console routine
.COOFF	30	Output console shutdown routine
.HSDTA	32	High speed display handler routine
.BSON	34	Punch/load initialization routine
.BSDTA	36	Punch/load handler routine
.BSOFF	38	Punch/load shutdown routine
.PAUSE	40	Processing pause routine
.CMDL2	44	Secondary command list
.ACIA	46	Address of ACIA
.PAD	48	Character and new line pad counts
.ECHO	50	Echo flag
.PTM	52	Programmable timer module address

Table 4 Vector table entries of ASSIST09