A PROM BURNER MODULE EXTENSION CARD FOR AN IBM PC

A Thesis Presented to

The Faculty of the College of Engineering and Technology

Ohio University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Chiung-Hsing Chen

March, 1989

ACKNOWLEDGEMENTS

I would like to express my sincere appreciation and gratitude to my advisor Dr. Robert A. Curtis for his guidance, friendship and support throughout the project. I also like to make a special thanks to the committee members, Dr. Janusz Starzyk, Dr. Henry Lozykowski and Dr. John Gilliam for their invaluable time and assistance.

Finally, I wish to thanks to my parents and friends, especially Wu Pi Fang and Chien Cheng Yu, for their helps and encouragements during my difficult moments.

CONTENTS

CHAPTER 1

INTRODUCTION

More and more electronic devices are being "programmed", whether they contain a microprocessor or not. In many instances, like household appliances or games, those programs are not entered by hand or from a magnetic medium such as disk or tape, but are contained in integrated circuits (ICs) called ROM's (Read Only Memory).

1.1 What is a Read-Only Memory (ROM)

The ROM (Read-Only Memory) is a type of memory device that is permanently programmed. Unlike RAM (Random Access Memory) which is a Read/Write memory device, when system power goes down, ROM still retains its contents. The contents in ROMs can be read many times, but can not be changed. That is why it is called "read-only".

In general, there are two ways that ROM's are programmed. The first of these is called **mask programming** and is actually a part of the IC manufacturing process. The mask programming is useful when large quantities of idential ROM's are needed. ROM which programmed by this process is called Mask ROM.

Another type of ROM is known as PROM (Programmable Read-Only Memory) which is a kind of the user-programmable version of ROMs. An instrument, PROM **programmer**, is used to program this type of memory devices.

There are two types of PROMs are available: One is one-time programmable and the other is various erasable or reprogrammable versions. Bipolar and fusible link PROMs are programmed once. The fusible links are broken by application of current pulses through the output terminals. Once the links are blown, the fixed pattern is permanent and can not be altered.

This type of memory device is supplied with all its bits at either a high or low logic level. In programming it, the user changes those logic states to meet his or her requirements. Typically, that programming is accomplished by " blowing " (burning_out) internal titanium_tungsten (Ti-W) fuse links, each one representing a bit. That is done by applying a specific excess voltage to the power input of the PROM IC after selecting those bits that are to be a logic-high, and those that are to be a logic-low.

A great deal of care must be taken when " burning " a PROM. Even if only one bit is programmed incorrectly, the entire PROM is ruined. If the voltage is too low, the fuse links will not be burned out; if it is too high, or applied for too long a time, undesirable side effects may result: For example, the extra one or two bits may be burned by exceedingly high current into the selected bit address.

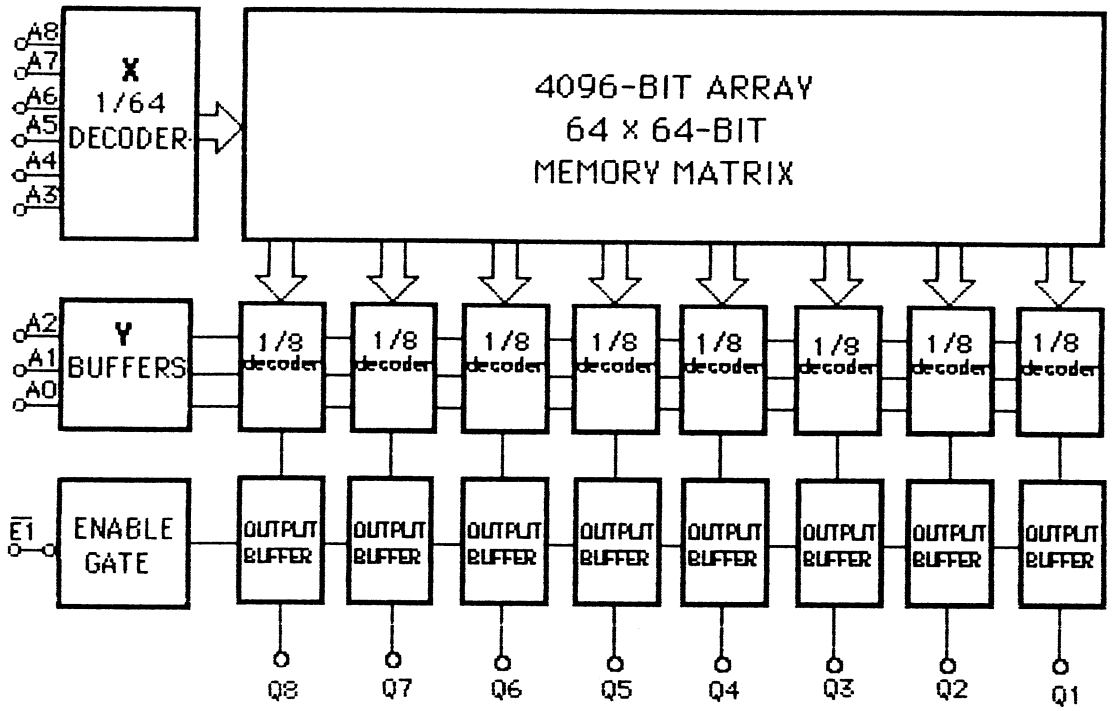
As one alternative to fusible-link programming, EPROM (Erasable Programmable ROM) used charge-storage programming. Three kinds of erasable PROMs are currently being offered: UVPROM (Ultra-Violet PROM), EAPROM (Electrically Alterable ROM) and EEPROM (Electrically Erasable PROM). These erasable PROMs can be erased by using different technologies, like ultraviolet light or electricty. In this work the emphasis is on bipolar and fusible link PROMs.

1.2 Discussion of a Programmable Read-Only Memory

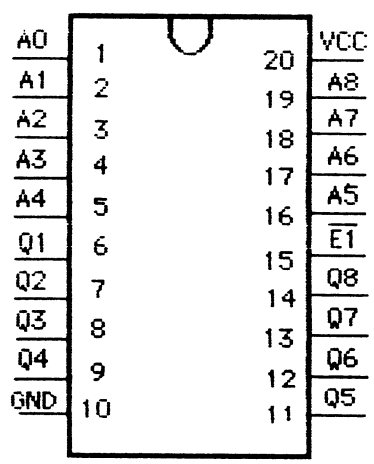
A practical 512 x 8 bipolar PROM's block diagram is shown in Figure 1-1a. The DM54S472/DM74S472 is a TTL chip in a 20 pin package (Figure 1-1b) and requires only a single DC power supply voltage of 5 volts. It has tri-state outputs. If the enable input is high, all 8 outputs are "OFF", ie., in a high impedance state.

The nine address lines (A0 - A8), are sent through input buffers. Six address lines (A3 - A8) are fed to the " X " decoder, while three (A0 - A2) are fed to the " Y " decoder. One of the 64 " X " decoder lines is presented to the memory array. This memory array is organized as a matrix of 64 x 64 for a total of 4096 cells. The X decoder input causes an output of eight 8-bit words. One of the these eight words is selected by the Y decoder gating. The chip select logic enables the eight output buffers by making E-1' low. When the chip is selected the output buffers change from the high impedance to the active state. The stored word is then available at the output pins [1].

Two parameters are important in using a PROM: address access time and chip select enable time. The access time is the time required to receive an address, decode it, drive a set of junctions in the PROM array and transmit the result through buffers to the PROM output. The maximum address access time is 60 nanoseconds for the 54S472/74S472. The output enable is controlled by the chip select inputs. The time required to sense a chip select signal and enable the output buffers to transmit the result to the PROM output is called enable access time. The maximum enable access time is 30 nanoseconds for the 54S472/74S472.



(a)



(b)

Figure 1-1 Example of organization of a PROM(DM74S472)
(a) Block diagram. [1]
(b) Pin connections

1.3 Purpose of this Thesis

The access time of a PROM is much shorter than an EPROM's which is why the PROM programmer is desirable even though EPROM's are popular now and can be reprogrammed many times. The disadvantage of using a PROM is that it can be programmed only once.

This thesis reports the design and construction of a PROM programmer controlled by IBM PC-XT or PC-AT or compatible system. The circuit design presented is intended to program **NATIONAL SEMICONDUCTOR**'s bipolar TTL DM54/74S and DM77/87S-series of programmable memories and their equivalents.

There were several reasons for selecting these.

(1) They are readily available.

(2) They are relatively inexpensive - a few dollars for a 32-word type.

(3) They require only a single five-volt DC power supply when these PROMs are being used.

(4) When they are being programmed, only a single 10.5V ±0.5V programming voltage is needed.

(5) They are available with either open-collector or three-state output configurations, and have a fast access time. On the average, a bipolar PROM has only an access time of 35 nanoseconds, while EPROM access times are of the order of hundreds of nanoseconds.

1.4 Design Features of this PROM Programmer

This programmer is not designed as a stand-alone unit, but is a plug-in card connected to an IBM PC-XT, PC-AT or compatible system's I/O slot. Thus it can make use of all utilities of a personal computer, including the monitor and disk driver. With the monitor, users can have full screen operation. Using the disk driver they can easily access the custom operating program when it is needed. The most important thing is that the card does not need its own central processor unit (CPU) and power supply system.

This programmer allows the user to inventory blank PROMs and program them when needed. Desirable features of this unit are:

(1) It is easy to use.

(2) It can program up to a 2048 x 8 memory size. Table 1-1 shows all the PROM types that can be programmed.

(3) It can display PROM contents in ASCII characters representing numbers in binary or hexadecimal format.

(4) It can check for blank PROMs.

(5) By using computer buffer memory , it can manipulate data before loading it into the PROM (For example: Kill, Insert, or Modify).

(6) Powerful programming software (using Assembly and Turbo C languages) makes it more efficient to write a program and burn it into a PROM IC.

Table 1-1 Chips compatible with the programmer

TYPE	SIZE
74S188/54S188	32 * 8
74S288/54S288	32 * 8
74S287/54S287	256 * 8
74S387/54S387	256 * 8
74S472/54S472	512 * 8
74S473/54S473	512 * 8
74S474/54S474	512 * 8
74S475/54S475	512 * 8
74S570/54S570	512 * 4
74S571/54S571	512 * 4
74S572/54S572	1024 * 4
74S573/54S573	1024 * 4
77S180/87S180	1024 * 8
77S181/87S181	1024 * 8
77S184/87S184	2048 * 4
77S185/87S185	2048 * 4
77S190/87S190	2048 * 8
77S191/87S191	2048 * 8

(7) It can easily download a custom operating program to the host computer's buffer memory.

(8) It can easily upload a hexadecimal file into a diskette.

1.5 Summary

This work describes the construction of a PROM programmer. In the next chapter the programming steps and system block diagram are discussed. Detailed hardware design is described in Chapter 3. Related software design is dealt with in Chapter 4, and Chapter 5 is a discussion of the overall limitations and capabilities of the final unit.

CHAPTER 2

Overall Design of the PROM Programmer

To design the National Semiconductors' bipolar PROM programmer, the programming steps and programming waveforms must be considered. Before the design can be realized, the circuitry and operation of a PROM needs to be understood.

In this chapter, the circuitry and operation of a PROM is discussed at the beginning. Programming steps with the programming waveforms and the system block diagram are discussed in the following sections. Detailed hardware and related software descriptions are given in later chapters.

2.1 Introduction to a PROM

Figure 2-1 is the general configuration of a 256 x 8-bit bipolar PROM: that is, there are eight address lines, and therefore , 2^8 =256 states can be represented. This PROM can also be described as containing 256 words with 8 bits per word. There are as many words as there are address states.

Two groups of input lines called " X " and " Y " are used to drive word lines and column lines respectively. The X-decode addresses (A3 - A7) have Schottky clamping diodes as protection and PNP transistor buffers for reducing input loading. The Y-decode address buffers (A0 - A2) also have Schottky diode clamps and PNP transistors driving a Schottky diode matrix. This diode matrix selects one of eight columns on each of the eight output bits. The selected column line drives the sense amplifier input to a high

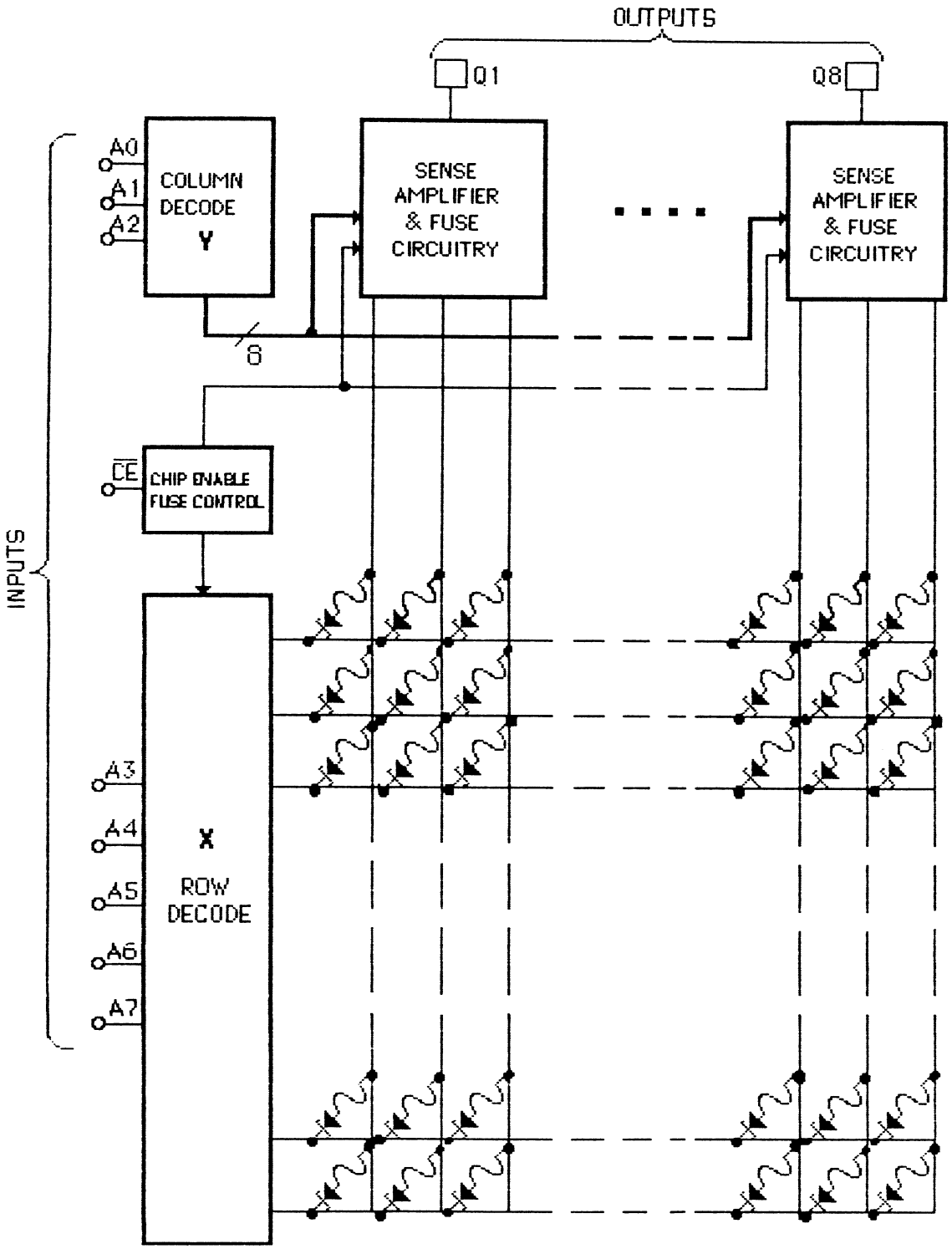


Figure 2-1 PROM circuitry block diagram [2]

level in the case of a blown fuse, or current is shunted through an unblown fuse through the selected word driver to ground resulting in a low input to the sense amplifier.

A few kinds of material are used as fuses in bipolar PROM IC's. Platinum silicide is used in Advanced Micro Devices' bipolar generic PROMs; nickle chromium fuses are used in a Harris bipolar PROM, and titanium-tungsten (Ti-W) fuses are implemented in National Semiconductors' bipolar TTL PROMs.

The sense amplifier is a fast level shifter-inverter with temperature and voltage threshold sensitivities compensated by the driving circuitry. Each of the eight sense amplifiers in this circuit provides active drive to an output buffer. Each output buffer also contains a disable input, which is driven from the chip-enable buffer. The chip-enable buffer input is a Schottky diode clamped transistor buffered design with an active pull up and pull down to drive the output [2].

2.2 Programming Steps

Programming involves selecting the word of memory to be programmed by setting the address lines, grounding the bit that is to be changed, and holding the bits that are not to be programmed at the supply voltage (five volts).

The timing diagram of Figure 2-2 shows the relationships between the data applied, the address signals, and the programming pulses to the PROM IC device. Programming parameters are also shown in Figure 2-2.

Below are the detailed programming steps:

< a > Select the desired word by applying a high or low level to the appropriate address inputs. Disable the chip by applying a high level to one or both enable inputs.

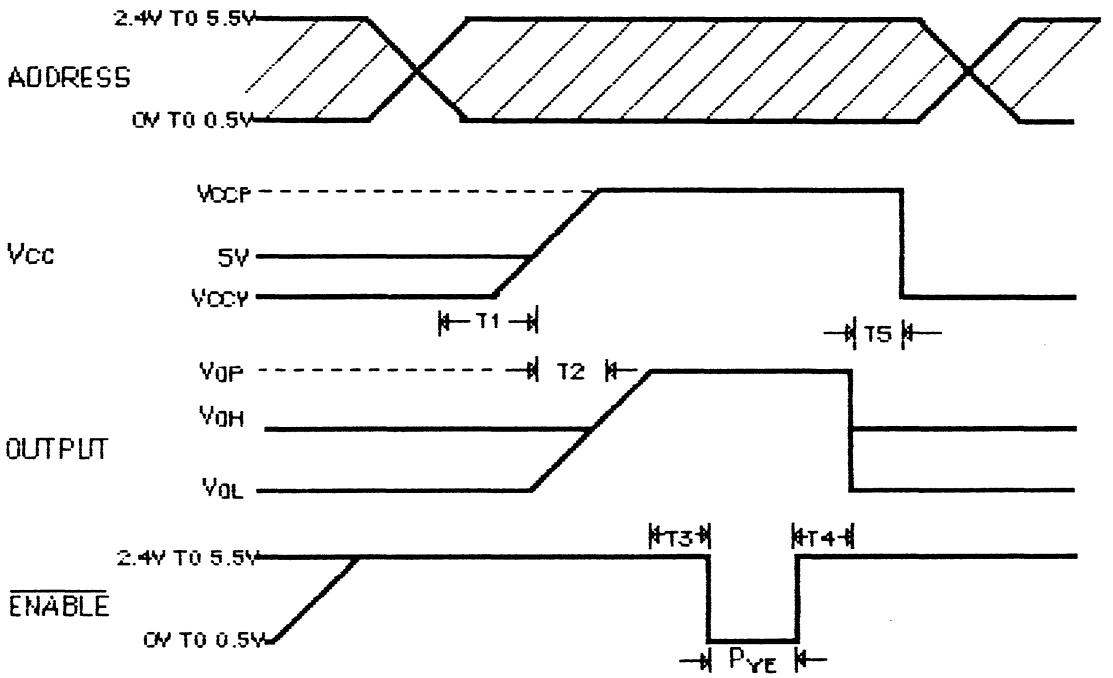
< b > Increase Vcc to 10.5V ±0.5V with the rate of increase being between 1.0 and 10.0V/μs. Since Vcc supplies the current to blow the fuse as well as the Icc of the device at the programming voltage, it must be capable of supplying 750 mA at 11.0V.

< c > Select the output where a high level is desired by raising that output voltage to 10.5V ±0.5V. Limit the rate of increase to a value between 1.0 and 10.0V/μs. This voltage change may occur simultaneously with the increase in Vcc but must not precede it. It is critical that only one output at a time be programmed since the internal circuits can only supply programming current to one bit at a time. Outputs not being programmed must be left open or tied to a high impedance source of at least 20 KΩ.

< d > Enable the device by taking both chip enables to a low level. This is done with a pulse of 10 μs. Normal input levels are used and rise and fall times are not critical.

< e > Verify that the bit has been programmed by first removing the programming voltage from the output and then reducing Vcc to 4.0V ±0.2V. Verification at a Vcc level of 4.0V will guarantee proper output states over the range of power supply voltage and temperature for the programmed part. The chip must be enabled to sense the state

	PARAMETERS	CONDITIONS	MIN	RECOMMANDED VALUE	MAX	UNITS
V_{CCP}	Required Vcc for Programming		10.0	10.5	11.0	V
I_{CCP}	Icc during Programming	VCC= 11V			750	mA
V_{OP}	Required Output voltage for Programming		10.0	10.5	11.0	V
I_{OP}	Output Current while Programming	VOUT= 11V			20	mA
t_{RR}	Rate of Voltage Change of Vcc or Output		1.0		10.0	V/µs
P_{WE}	Programminf Pulse Width (enabled)		9	10	11	µs
V_{CCV}	Required Vcc for Verification		3.8	4.0	4.2	V
M_{DC}	Maximum Duty Cycle for Vcc at Vcop			25	25	%



T1 = 100 ns min
T2 = 5µs min (T2 may be >=0 if Vccp rises at the same rate or faster than Vop)
T3 = 100 ns min
T4 = 100 ns min
T5 = 100 ns min
$\overline{\text{ENABLE}}$ is made low for 5 additional pulses after verification of Voh indicates a bit has been programmed

Figure 2-2 Programming waveform and parameters [1]

of the outputs. During verification, the loading of the output must be within specified I_{OL} and I_{OH} limits.

< f > Steps b, c and d must be repeated 10 times or until verification that the bit has been programmed.

< g > Following verification, apply five additional programming pulses to the bit being programmed. The programming procedure is now complete for the selected bit.

< h > Repeat steps a through g for each bit to be programmed to a high level. The duty cycle of Vcc at the programming voltage must be limited to a maximum of 25% [1].

In addition to the above, the following conditions must be observed. First, programming should be attempted only at temperatures between 15° and 30° C. Second, addresses and chip enable pins must be driven from normal TTL logic levels during both programming and verification. As mentioned above, programming will occur at a selected address when Vcc is held at 10.5 ±0.5V and the chip is subsequently enabled.

2.3 Classification of PROMs

Several different PROMs can be programmed by this system. In order to cover all these PROMs, to use the same 24-pin IC socket for each, and to simplify the hardware design work, the different output configurations need to be classified. Table 2-1 gives the pin configuration for each device. See Table 2-1. All PROMs are alligned with pin 1 in the socket's pin 1 position. When each PROM is classified by its output configuration there are four different types. Type_1 (74S188/74S288) has outputs on pin1 - pin7.

Table 2-1 PROM IC pin configurations

PIN#	TYPE_1 74S188 74S288	TYPE_2 74S472 74S473
01	Q1	A0
02	Q2	A1
03	Q3	A2
04	Q4	A3
05	Q5	A4
06	Q6	Q1
07	Q7	Q2
08	GND	Q3
09		Q4
10		GND
11		
12		

PIN#	TYPE_1 74S188 74S288	TYPE_2 74S472 74S473
24	VCC	VCC
23	$\overline{E1}$	A8
22	A4	A7
21	A3	A6
20	A2	A5
19	A1	$\overline{E1}$
18	A0	Q8
17	Q8	Q7
16		Q6
15		Q5
14		
13		

PIN#	TYPE_3 74S287 74S387	74S570 74S571	74S572 74S573	77S184 77S185
01	A6	A6	A6	A6
02	A5	A5	A5	A5
03	A4	A4	A4	A4
04	A3	A3	A3	A3
05	A0	A0	A0	A0
06	A1	A1	A1	A1
07	A2	A2	A2	A2
08	GND	GND	E1	A10
09			GND	GND
10				
11				
12				

PIN#	TYPE_3 74S287 74S387	74S570 74S571	74S572 74S573	77S184 77S185
24	VCC	VCC	VCC	VCC
23	A7	A7	A7	A7
22	$\overline{E2}$	A8	A8	A8
21	$\overline{E1}$	$\overline{E1}$	A9	A9
20	Q1	Q1	Q1	Q1
19	Q2	Q2	Q2	Q2
18	Q3	Q3	Q3	Q3
17	Q4	Q4	Q4	Q4
16			$\overline{E2}$	$\overline{E}$
15				
14				
13				

PIN#	TYPE_4 77S180 77S181	77S190 77S191	74S474 74S475
01	A7	A7	A7
02	A6	A6	A6
03	A5	A5	A5
04	A4	A4	A4
05	A3	A3	A3
06	A2	A2	A2
07	A1	A1	A1
08	A0	A0	A0
09	Q1	Q1	Q1
10	Q2	Q2	Q2
11	Q3	Q3	Q3
12	GND	GND	GND

PIN#	TYPE_4 77S180 77S181	77S190 77S191	74S474 74S475
24	VCC	VCC	VCC
23	A8	A8	A8
22	A9	A9	NC
21	$\overline{E1}$	A10	$\overline{E1}$
20	$\overline{E2}$	$\overline{E1}$	$\overline{E2}$
19	E3	E2	E3
18	E4	E3	E4
17	Q8	Q8	Q8
16	Q7	Q7	Q7
15	Q6	Q6	Q6
14	Q5	Q5	Q5
13	Q4	Q4	Q4

Type_2 (74S472/74S473) has them on pin6 - pin9 and pin15 - pin18. Type_3 (74S287/74S387, 74S570/74S571, 74S572/74S573, and 77S184/77S185) uses pin17 - pin20 for all outputs. Finally, type_4 (77S180/77S181, 77S190/77S191, and 74S474/74S475) has output signals on pin9- pin11 and pin13 - pin17.

2.4 System Block Diagram

Figure 2-3 is the system block diagram of the PROM programmer. The system fits into an IBM PC-XT or PC-AT or compatible system's I/O slot. It contains an address decoder, data buffer, PROM type selector, output registers for PROM address and chip enable code words, output registers for data to the PROM, input registers to read PROM data, an output voltage switch for programming the PROM data, and a high current voltage source for the PROM's Vcc input pin.

2.5 Function of each Block

(1) The IBM PC-XT's I/O slot provides the desired address and data for the system bus and supplies all the desired power levels to the programmer.

(2) The address decoder gives all the desired chip enable signals.

(3) The data buffer provides bidirectional data bus buffering.

(4) The PROM type selector selects the specified PROM type for the programming operation.

(5) Output registers for PROM address and chip enable are used to capture the address sent by the IBM PC-XT's CPU.

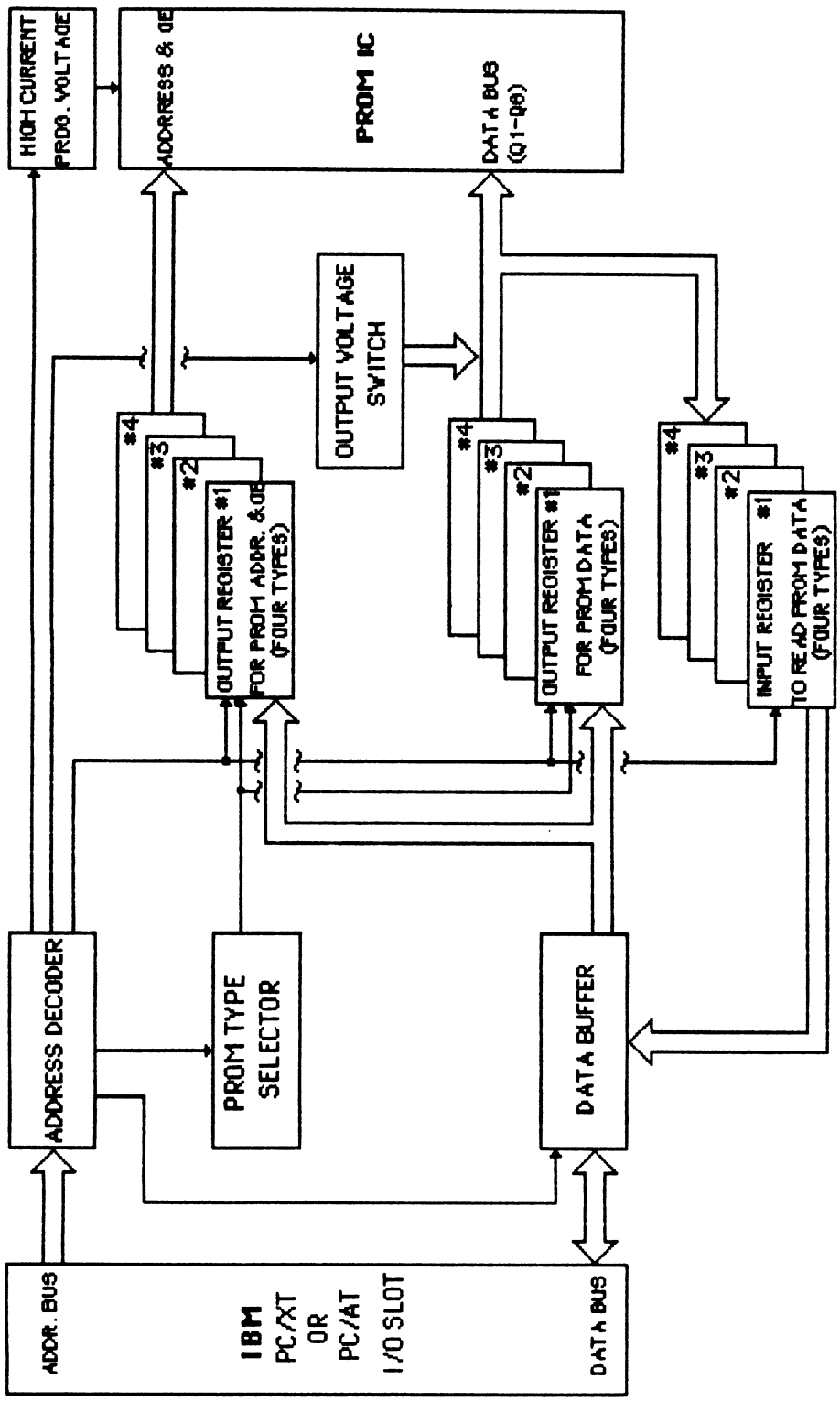


Figure 2-3 System block diagram

(6) The high current program voltage is used to supply high voltage and current for PROM's Vcc input when it is being programmed.

(7) Two sets of output/input registers for PROM data are used. One captures the data sent by the CPU and the other senses information from the programmed PROM.

(8) The output voltage switch is used to switch the programming voltage for a PROM's output terminal from +5V up to +10.5V.

2.6 Programming a PROM

In this design the IBM PC-XT or PC-AT's I/O slot not only provides all the desired addresses and data but also supplies all the needed voltages to the system. First, the *PROM type selector* is used to select the specified device type to be programmed. Four different PROM types can be accommodated.

Second, the specified address needs to be issued. Here the *PROM address and chip enable output registers* are used to transfer the desired addresses from the IBM PC-XT's data bus to the particular PROM's address bus.

Third, the PROM's Vcc input pin must be increased by 10.5 ±0.5 volts from +5V. The *high current program voltage* is designed to supply the desired voltage at up to 500 mA of current. All PROMs have the Vcc input pin in the same location (pin24).

Fourth, the data is transmitted from the PC-XT's data bus to the PROM's data bus by using the proper *output registers for PROM data*.

Finally, the programming voltage for the PROM output (Qi) is applied to the specified data i bit by the *output voltage switch* and is switched from +5 volts to 10.5 volts. Note only one bit is programmed. The specified programmed bit Qi is chosen by software through a programmable peripheral device Intel 8255A.

In order to blow out the PROM's fuses correctly, a 10 μs pulse to the chip enables must be held when the PROM is being programmed. This is done with the same output register which supplies the PROM addresses. After programming, verification follows.

During the read operation, the PROM must be in the **READ** mode - like a ROM. First, the programming voltage for the PROM output (Qi) is removed by block of *output voltage switch* Second, the Vcc input voltage is reduced from 10.5V ±0.5V to +5V (TTL level) using block of *high current program voltage* Third, the PROM IC's content is then read to the PC-XT's buffer by using the proper *input register to read PROM data* This is accomplished by an 8088 IN instruction.

2.7 Summary.

In this chapter some of the important characteristics of PROMs were discussed, and how PROMs are programmed was explained. The system block diagram for the hardware was also presented. In the next chapter the detailed hardware is discussed for each block that is mentioned here.

CHAPTER 3

Hardware Design

In this chapter the hardware for the PROM programmer controlled by an IBM PC-XT or PC-AT or compatible system is presented, showing how the details of programming discussed in Chapter 2 are implemented. The following sections discuss each block mentioned in section 2.5. The software for the system is discussed in Chapter 4.

3.1 The IBM PC-XT's I/O Slot

In this work, the system power supply and CPU (Central Processor Unit) are already available. All the needed voltages and signals (For example, address bus, data bus and control signals) are supported by the IBM PC-XT's I/O slot (Appendix D). The I/O slot is an extension of the 8088 microprocessor bus. The connectors are capable of supporting 62 signals to a card, 31 on each side. Figure 3-1 is a drawing defining the signal and slot labeling convention and also indicating those signals that are used. There are four power levels available on the signal tabs in each card slot. Table 3-1 summarizes those levels.

The PROMs require a power supply voltage of +5V when they are operating in the **read mode**. They require an additional +10.5V programming pulse when in the **programming mode**. The voltages needed in the full design are +5 and ±12 volts.

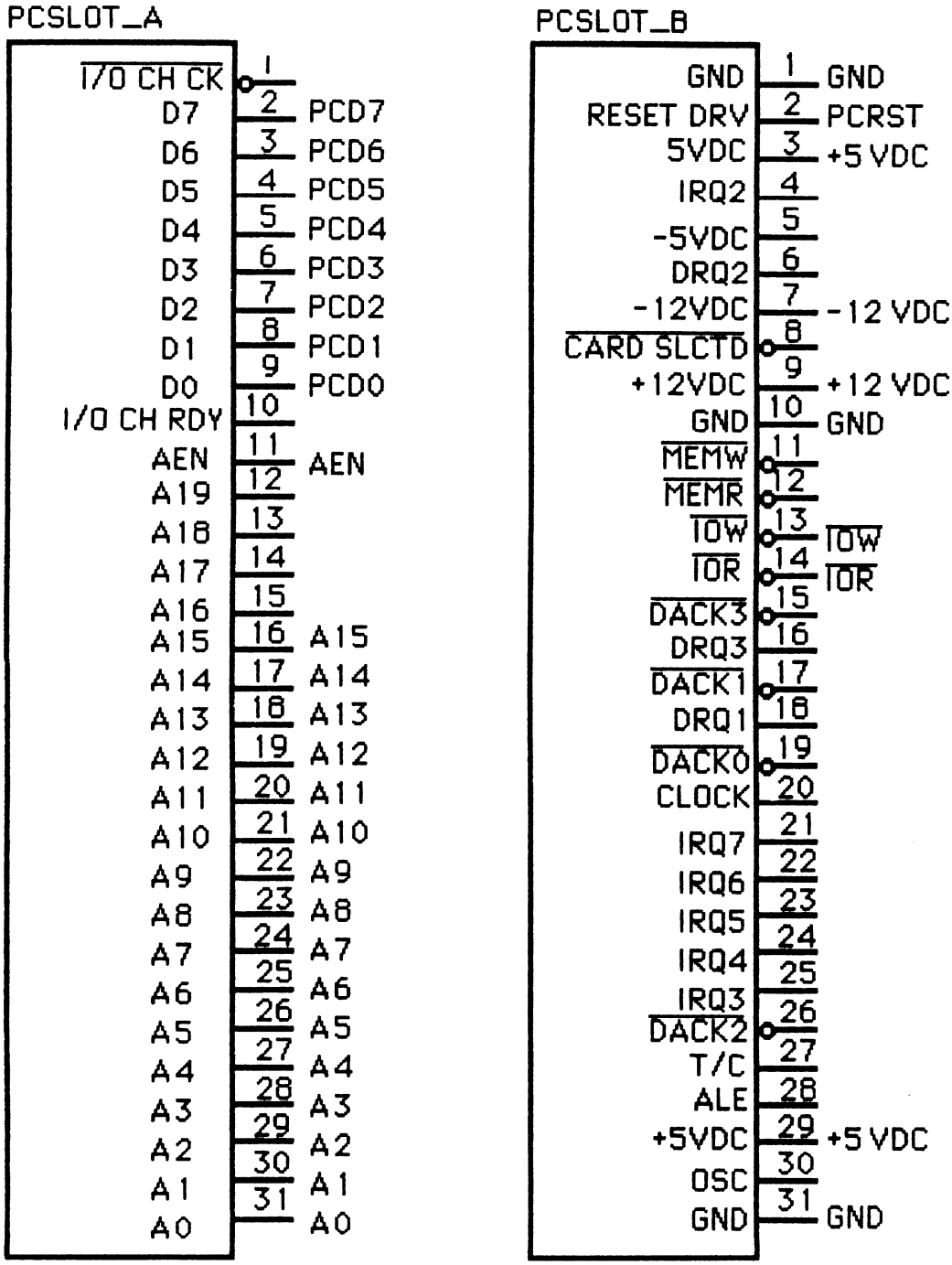


Figure 3-1 The IBM PC-XT's I/O slot
(showing signals used by the card)

Table 3-1 IBM PC/XT Power Supply

DC Power	Max(Vdc)	Min(Vdc)	Current (Amps)
+ 5 VDC	5.25	4.8	7.0
- 5 VDC	5.50	4.6	0.3
+12 VDC	12.60	11.52	2.0
-12 VDC	13.20	10.92	0.25

3.2 Address Decoding and Bus Buffer Circuit

The most common method of interfacing a microcomputer system, such as the PC (Personal Computer), is through the use of programmable input and output registers. Since the system bus is available in the card slot, it is not a difficult task to attach several devices to provide Input/Output functions in the design. Figure 3-2 is the circuit used to perform all I/O of the address and data. The address latch and data latch are individually discussed in the following sections.

The programmer card in the IBM PC-XT is controlled and sensed using the input and output registers. These registers are addressed using the I/O port address space of the 8088 microprocessor. With output registers, the microprocessor can write data into the register, treating the register as an I/O (Input/Output) port or a memory location. The output of those registers are wired to an interface device. Table 3-2 defines the address port's functions on the programmer card in this system.

Figure 3-3 is the system decoder and bus buffer used to decode all the desired I/O port addresses. The decode circuitry is designed so that the

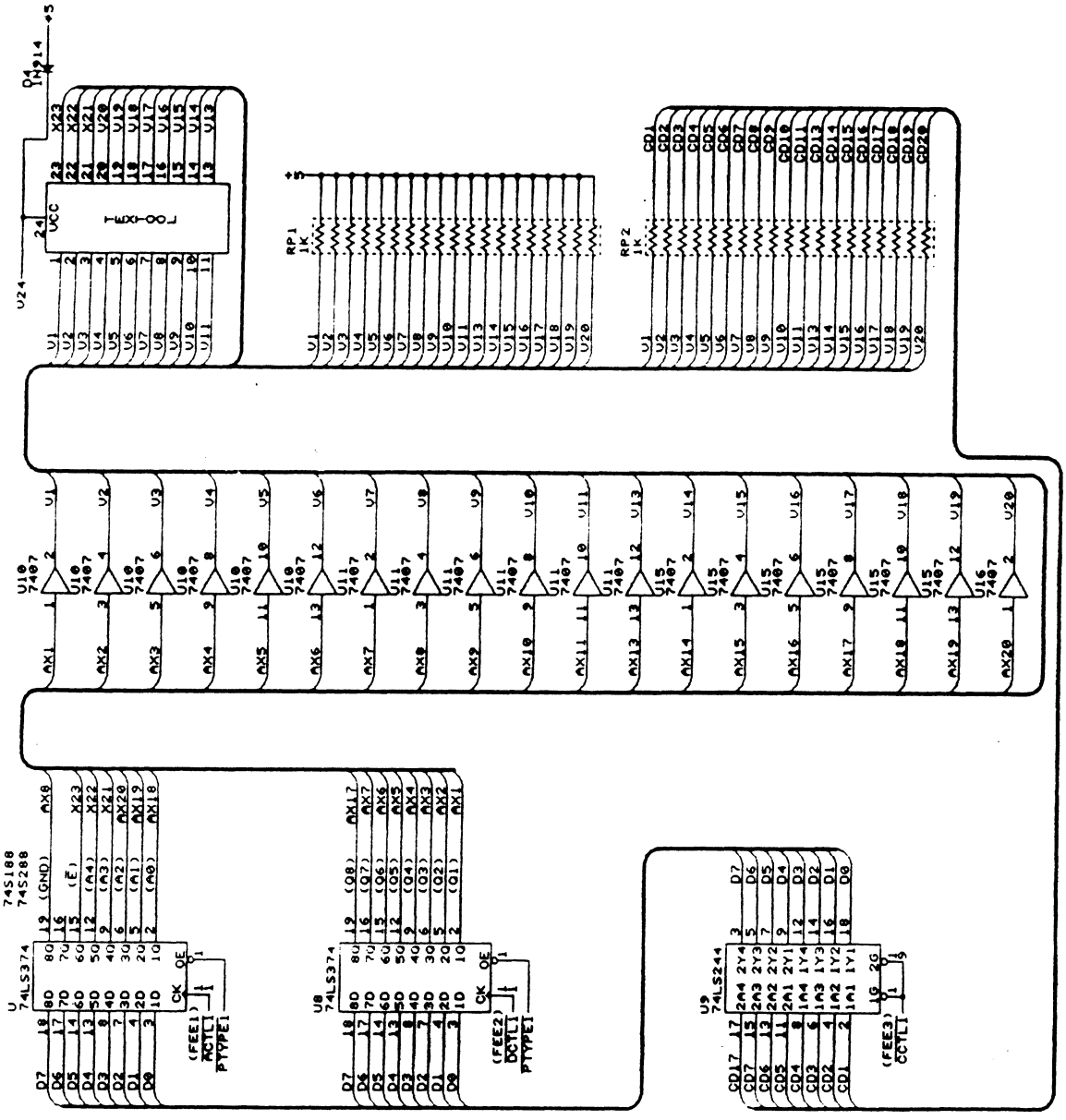


Figure 3-2 The PROM fits in the textool socket

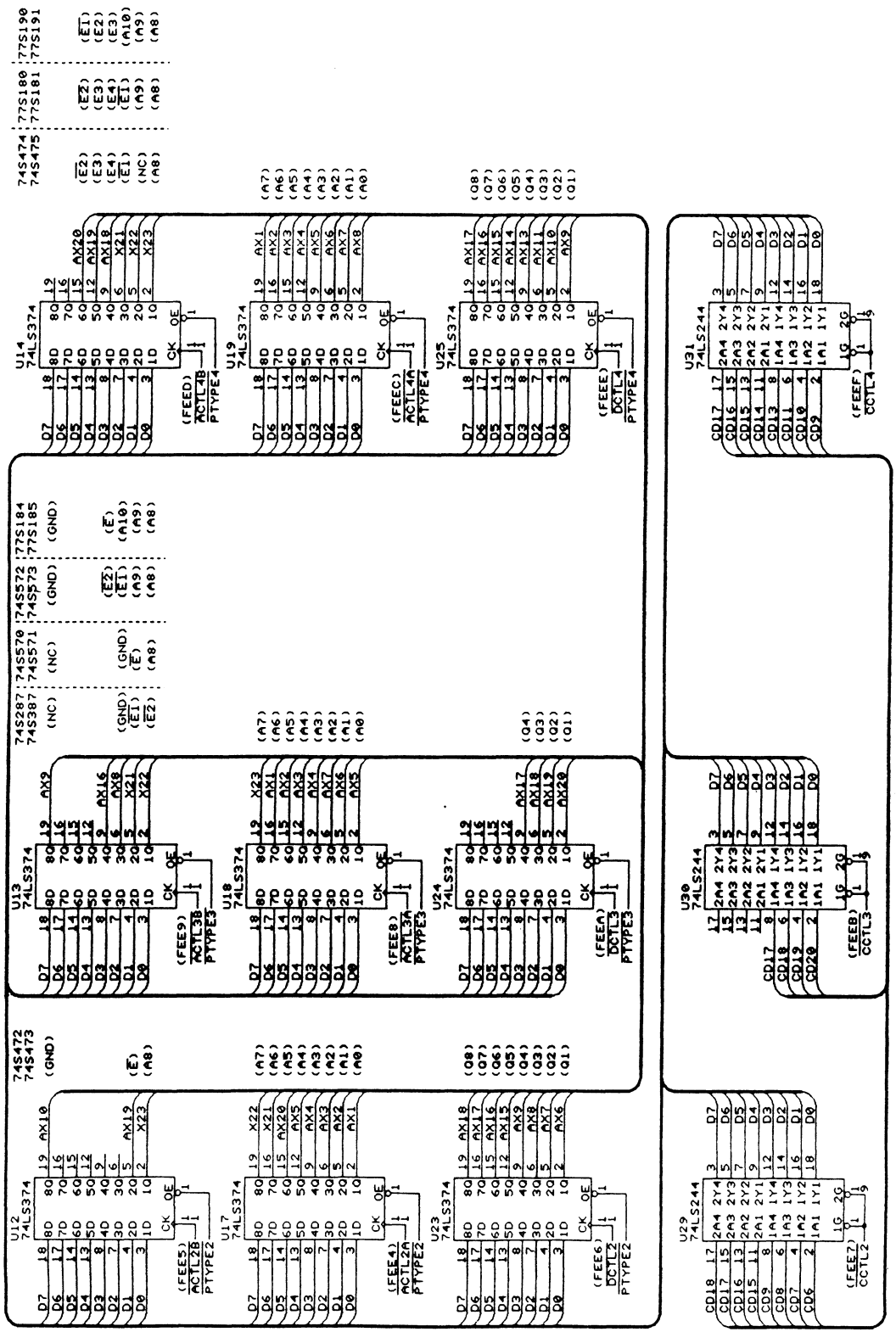


Figure 3-2 (continued)

Table 3-2 I/O port address usage

HEX address used	**Function**
FEE0H	for U35 clock input;to control PROM Vcc
Port addresses for type_1:	
FEE1H	for U7 clock input;to supply word address for PROM
FEE2H	for U8 clock input;to supply data for PROM
FEE3H	for U9 output enables;to read data from PROM
Port addresses for type_2:	
FEE4H	for U17 clock input;to supply word address I for PROM
FEE5H	for U12 clock input;to supply word address II for PROM
FEE6H	for U23 clock input;to supply data for PROM
FEE7H	for U29 output enables;to read data from PROM
Port addresses for type_3:	
FEE8H	for U18 clock input;to supply word address I for PROM
FEE9H	for U13 clock input;to supply word address II for PROM
FEEAH	for U24 clock input;to supply data for PROM
FEEBH	for U30 output enables;to read data from PROM
Port addresses for type_4:	
FEECH	for U19 clock input;to supply word address I for PROM
FEEDH	for U14 clock input;to supply word address II for PROM
FEEEH	for U25 clock input;to supply data for PROM
FEEFH	for U31 output enables;to read data from PROM
FEF0H-FEF3H	for U6 (8255A) chip select;to switch program voltage
FEF4H-FEF7H	not used
FEF8H-FEFBH	for U39 clock input;to select a specified PROM type
FEFCH-FEFFH	not used

location of a block of port addresses can be set selectively in the I/O port address space by simply setting a new value in the DIP switch. In this design, an SN74LS688 (U26) identity comparator and an eight bit DIP switch (U33) are used. On one side of the compare circuit, the address bus bits A5 through A12 are attached. On the other side , the DIP switches are connected. When the value set in the DIP switches is the same as the value set on the address bus, the compare equal output signal (P=Q)′ is activated, and can be used as the group select control signal. This technique allows the I/O ports to be moved so that an I/O-port address overlap can easily be avoided.

As an example of how this circuit is used, let us say we wanted to limit the I/O port address range from FFC0hex through FFDFhex. First, it would be necessary to set the values of the switches. The DIP switch corresponding to the address line A5 would be set to its active low state (ON). The remaining DIP switch pinouts are pulled high by the pull-up resistors. The DIP switches corresponding to those inputs would be set high - these are inactive (OFF). The set condition of the SN74LS688 device is NANDed together with the address lines A13, A14, and A15 by means of the enable input G′, thus requiring all three of the signals to have a HIGH condition before a compare equal output (P=Q)′ is activated.

In this system, the port address range is chosen from FEE0hex through FEFFhex, thus the DIP switch corresponding to the address line A8 must be set to its active low state. Table 3-3 gives some of the possible port address blocks.

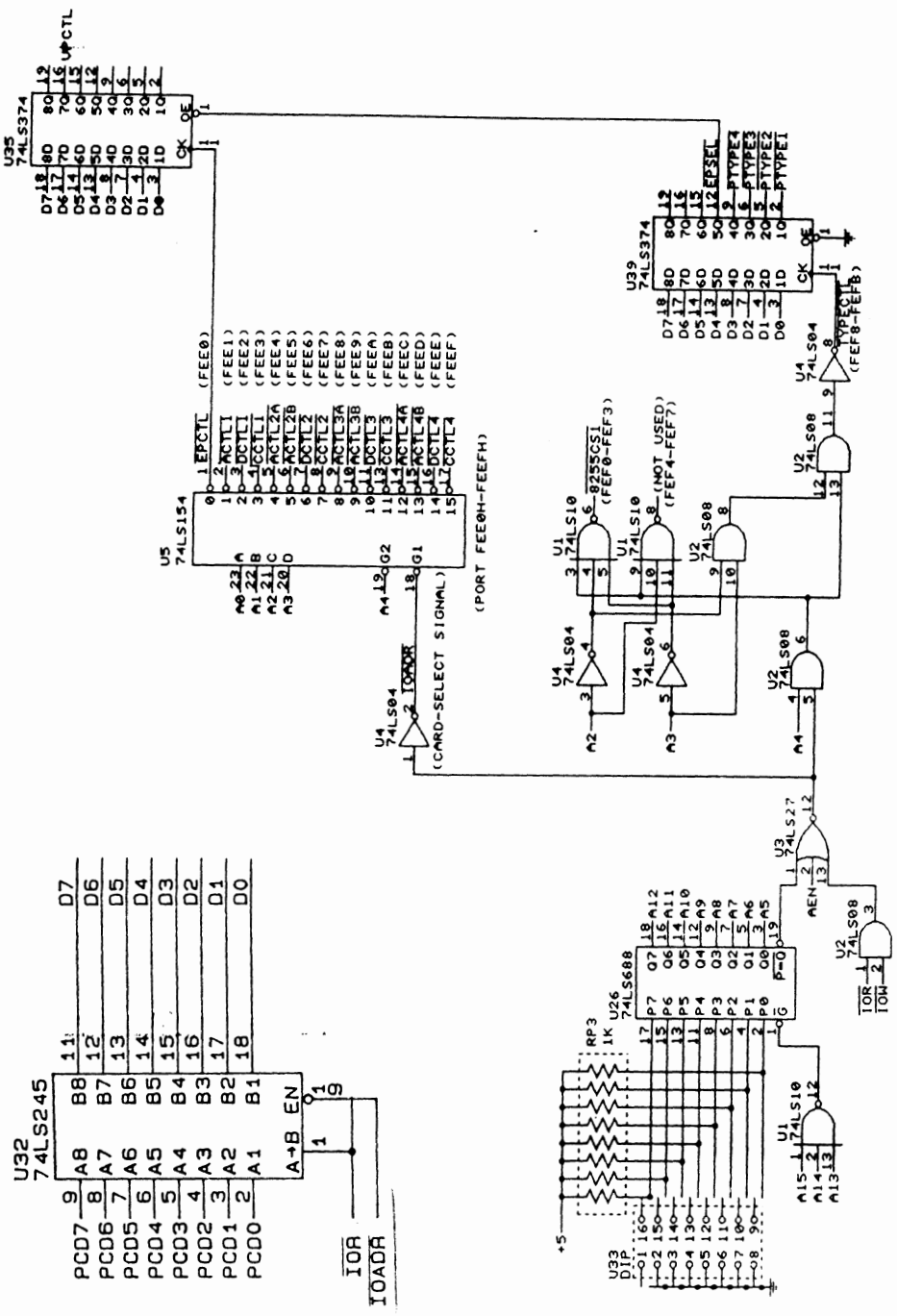


Figure 3-3 System address decoder and data bus buffer

Table 3-3 I/O address maps used

I/O ADDRESS MAP	SWITCH (U33 DIP SW)							
	1	2	3	4	5	6	7	8
EFE0H-EFFFH	ON	OFF	OFF	OFF	OFF	OFF	OFF	OFF
F7E0H-F7FFH	OFF	ON	OFF	OFF	OFF	OFF	OFF	OFF
FBE0H-FBFFH	OFF	OFF	ON	OFF	OFF	OFF	OFF	OFF
FDE0H-FDFFH	OFF	OFF	OFF	ON	OFF	OFF	OFF	OFF
FEE0H-FEFFH	OFF	OFF	OFF	OFF	ON	OFF	OFF	OFF
FF60H-FF7FH	OFF	OFF	OFF	OFF	OFF	ON	OFF	OFF
FFA0H-FFBFH	OFF	OFF	OFF	OFF	OFF	OFF	ON	OFF
FFC0H-FFDFH	OFF	OFF	OFF	OFF	OFF	OFF	OFF	ON

The output of the SN74LS688 (U26) is NORed together with AEN and the combined signals (IOR' AND IOW'), thus requiring all three of the signals to have a LOW condition before the active high signal IOADR is generated. The bus signal AEN (Address Enable) is used to disable I/O port address decodes during any DMA (Direct Memory Access) cycle in the host computer. The signal IOR' and IOW' must be ANDed together to generate the signal IOADR when either IOR' or IOW' is low.

The signal IOADR is used to enable the further decoding of the I/O addresses from A0, A1, A2, and A3. This is accomplished by using the SN74LS154 4-to-16 decoder circuit and a combinational logic circuit. In this design, several port addresses are decoded: some produce a group of "control signals" to enable the four types of PROMs (FEE0hex through FEEFhex); two groups of ports for the 8255As; one group (FEF0hex - FEF3hex) is used for the *output voltage switchs* to switch the programming voltage to each PROM's data bit; another group (FEF4hex - FEF7hex) is for future development of an EPROM programmer; and one group of addresses (FEF8hex - FEFBhex) is for the signal TYPECTL' to enable the "PROM type

selector (U39) which is used to activate a specified PROM for programming. Figure 3-4 is the I/O map for the system.

Figure 3-5 shows the pin connections of an SN74LS154 4-to-16 line decoder (U5) and its truth table. Four address lines A0 through A3 are used to address the 16 ports from FEE0hex through FEEFhex which control all Input/Output registers. Address line A4 and the signal IOADR' need to be low at the same time to enable the decoder. Table 3-4 gives the function table for the U5 outputs.

Figure 3-6 is the combinational logic circuitry used to choose three groups of ports and their function table. To program an Intel 8255A (Appendix D), four address ports are needed, one for the control word, and three for port A, port B, and port C. When the signal IOADR has a logic 1, address line A4 is logic 1, A3 and A2 are both logic 0, port addresses between FEF0H through FEF3H will be generated (see Figure 3-6b). The type-select signal TYPECTL' is activated when IOADR', address lines A3 and A4 are all logic 1 and A2 has a logic 0.

The IOADR' signal is also used to enable the SN74LS245 (U32) octal bus transceiver buffer in Figure 3-7. It is intended to activate the data bus when this programmer is being used. The PCD0 through PCD7 signals are from the IBM PC-XT's data bus and D0 through D7 forms the system data bus on the card. The IOR' signal which is connected to the DIR signal input of the SN74LS245 is used to set the direction of the enabled U32 bus transceiver. When IOR' is HIGH, the I/O slot will transfer data to the programmer's data bus. If the IOR' is in a low condition, the I/O slot will receive data from the card.

Table 3-4 Function table

$\overline{\text{EPCTL}}$ (FEE0hex) : control signal for PROM Vcc input

Type_1 control signals:

$\overline{\text{ACTL1}}$ (FEE1hex) :control signal for PROM address and enables
$\overline{\text{DCTL1}}$ (FEE2hex) :control signal for PROM data
$\overline{\text{CCTL1}}$ (FEE3hex) :control signal to read PROM data

Type_2 control signals:

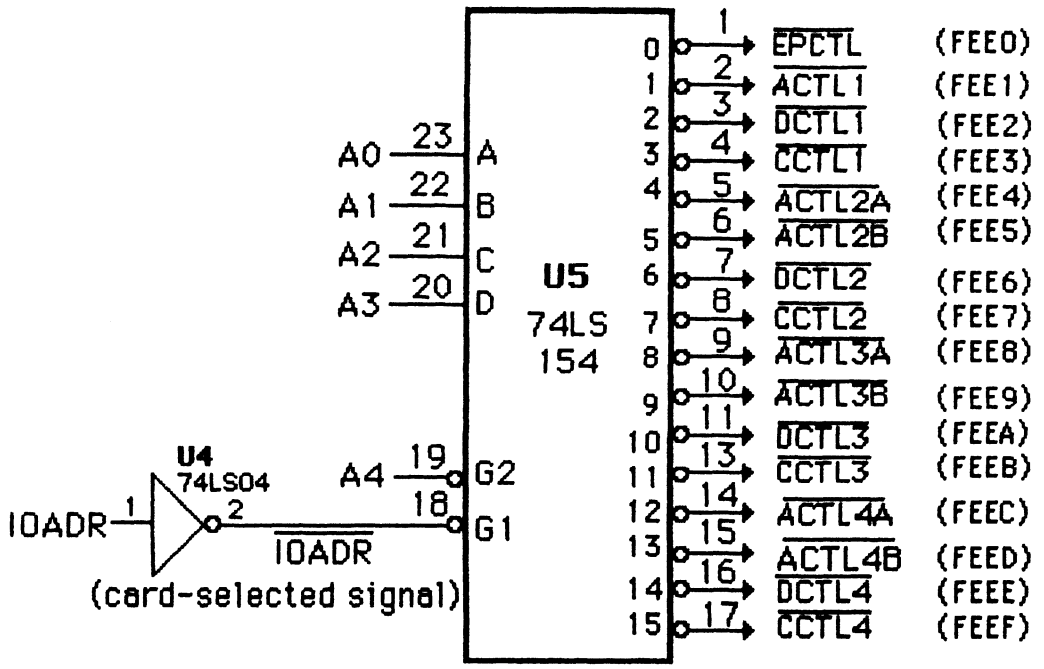
$\overline{\text{ACTL2A}}$ (FEE4hex) :control signal for PROM address and enables (I)
$\overline{\text{ACTL2B}}$ (FEE5hex) :control signal for PROM address and enables (II)
$\overline{\text{DCTL2}}$ (FEE6hex) :control signal for PROM data
$\overline{\text{CCTL2}}$ (FEE7hex) :control signal to read PROM data

Type_3 control signals:

$\overline{\text{ACTL3A}}$ (FEE8hex) :control signal for PROM address and enables (I)
$\overline{\text{ACTL3B}}$ (FEE9hex) :control signal for PROM address and enables (II)
$\overline{\text{DCTL3}}$ (FEEAhex) :control signal for PROM data
$\overline{\text{CCTL3}}$ (FEEBhex) :control signal to read PROM data

Type_4 control signals:

$\overline{\text{ACLT4A}}$ (FEEChex) :control signal for PROM address and enables (I)
$\overline{\text{ACTL4B}}$ (FEEDhex) :control signal for PROM address and enables (II)
$\overline{\text{DCTL4}}$ (FEEEHEX) :control signal for PROM data
$\overline{\text{CCTL4}}$ (FEEFHEX) :control signal to read PROM data



(A)

IOADR (FEE0H-FEFFH)	A3	A2	A1	A0	OUTPUT 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	PORT ADDRESS	FUNCTION
1	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	FEE0H	$\overline{EPCTL}$
1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	FEE1H	$\overline{ACTL1}$
1	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	FEE2H	$\overline{DCTL1}$
1	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	FEE3H	$\overline{CCTL1}$
1	0	1	0	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	FEE4H	$\overline{ACTL2A}$
1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	FEE5H	$\overline{ACTL2B}$
1	0	1	1	0	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	FEE6H	$\overline{DCTL2}$
1	0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	FEE7H	$\overline{CCTL2}$
1	1	0	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	FEE8H	$\overline{ACTL3A}$
1	1	0	0	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	FEE9H	$\overline{ACTL3B}$
1	1	0	1	0	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	FEEAH	$\overline{DCTL3}$
1	1	0	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	FEEBH	$\overline{CCTL3}$
1	1	1	0	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	FEECH	$\overline{ACTL4A}$
1	1	1	0	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	FEEDH	$\overline{ACTL4B}$
1	1	1	1	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FEEEH	$\overline{DCTL4}$
1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	FEEFH	$\overline{CCTL4}$

(B)

Figure 3-5 Decoding of 16 port-address space for IOADR= FEE0H-FEFFH
(A) block diagram
(B) truth table for each output

Table 3-4 Function table

$\overline{\text{EPCTL}}$ (FEE0hex) : control signal for PROM Vcc input

Type_1 control signals:

$\overline{\text{ACTL1}}$ (FEE1hex) :control signal for PROM address and enables
$\overline{\text{DCTL1}}$ (FEE2hex) :control signal for PROM data
$\overline{\text{CCTL1}}$ (FEE3hex) :control signal to read PROM data

Type_2 control signals:

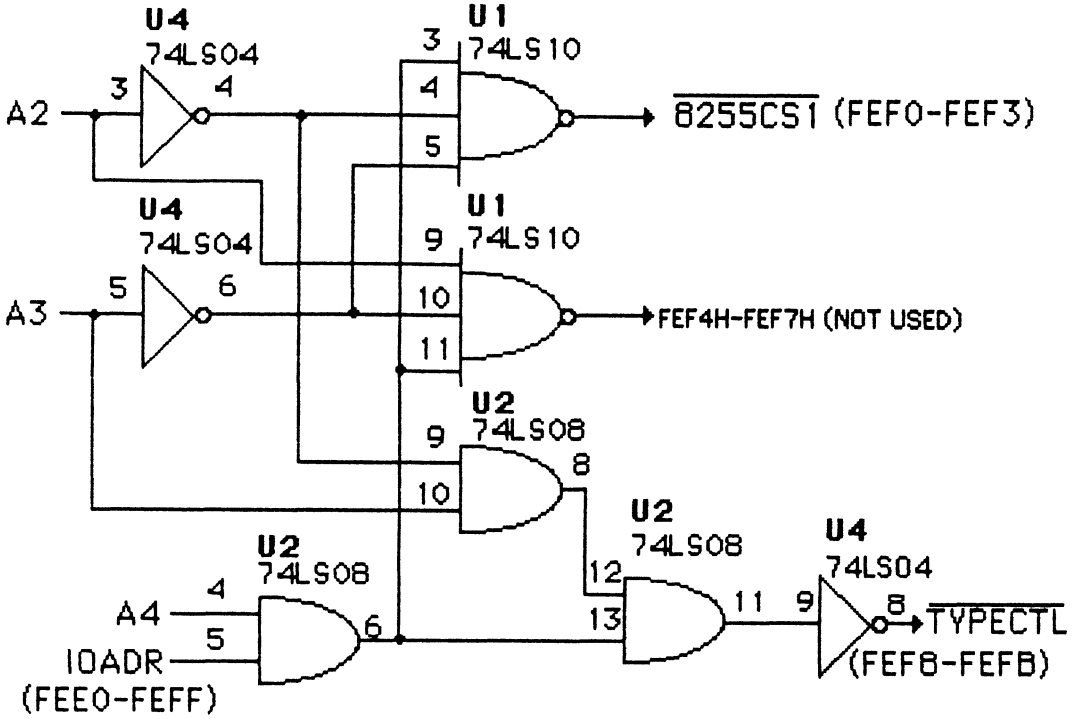
$\overline{\text{ACTL2A}}$ (FEE4hex) :control signal for PROM address and enables (I)
$\overline{\text{ACTL2B}}$ (FEE5hex) :control signal for PROM address and enables (II)
$\overline{\text{DCTL2}}$ (FEE6hex) :control signal for PROM data
$\overline{\text{CCTL2}}$ (FEE7hex) :control signal to read PROM data

Type_3 control signals:

$\overline{\text{ACTL3A}}$ (FEE8hex) :control signal for PROM address and enables (I)
$\overline{\text{ACTL3B}}$ (FEE9hex) :control signal for PROM address and enables (II)
$\overline{\text{DCTL3}}$ (FEEAhex) :control signal for PROM data
$\overline{\text{CCTL3}}$ (FEEBhex) :control signal to read PROM data

Type_4 control signals:

$\overline{\text{ACLT4A}}$ (FEEChex) :control signal for PROM address and enables (I)
$\overline{\text{ACTL4B}}$ (FEEDhex) :control signal for PROM address and enables (II)
$\overline{\text{DCTL4}}$ (FEEEHEX) :control signal for PROM data
$\overline{\text{CCTL4}}$ (FEEFHEX) :control signal to read PROM data



(A)

IOADR (FEE0-FEFF)	A4	A3	A2	A1	A0	PORT ADDRESSES	$\overline{8255CS1}$	FUNCTION
1	1	0	0	0	0	FEF0H	0	DATA BUS=>PORT A
1	1	0	0	0	1	FEF1H	0	DATA BUS=>PORT B
1	1	0	0	1	0	FEF2H	0	DATA BUS=>PORT C
1	1	0	0	1	1	FEF3H	0	DATA BUS=>CONTROL

IOADR (FEE0-FEFF)	A4	A3	A2	A1	A0	PORT ADDRESSES	FUNCTION
1	1	0	1	X	X	FEF4H-FEF7H	NOT USED

IOADR (FEE0-FEFF)	A4	A3	A2	A1	A0	PORT ADDRESSES	$\overline{TYPECTL}$	FUNCTION
1	1	1	0	X	X	FEF8H-FEFBH	0	DATA BUS LATCHED

(B)

Figure 3-6 Decoding of three groups of ports
for IOADR= FEE0H-FEFFH x: don't care
(A) logic circuit
(B) function table

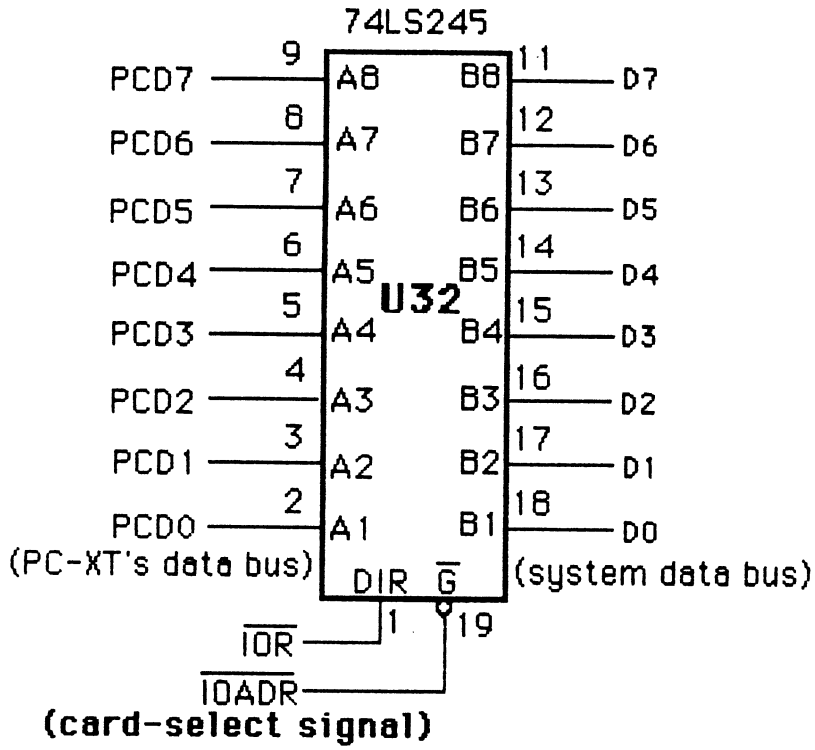


Figure 3-7 Connection of data bus buffer

3.3 PROM Type Selecting

Before operation begins, the programmer needs to know which type of PROM will be programmed. An SN74LS374 (U39) octal D-type latch is used to latch four individual enable signals, one for each PROM type. These four PROM type enable signals are shown in Figure 3-8. For example, the signal PTYPE1' is used to configure the system for type_1. The output signal EPSEL' is connected to "U35" in Figure 3-3. It is designed to enable PROM Vcc input voltage, once a specified PROM type is activated. During the programming period, the PROM enable signals must be held so the SN74LS374 is always enabled (OE' is grounded).

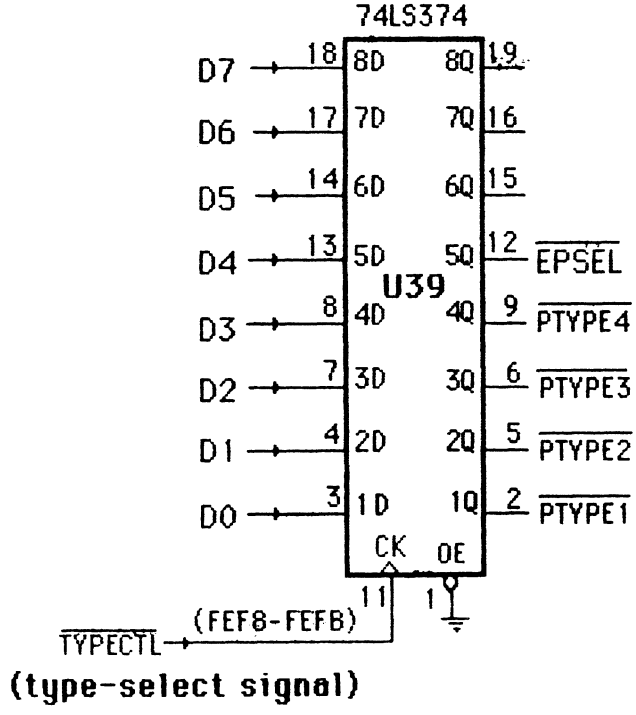


Figure 3-8 PROM type selector

The inputs to each bit of the latch (U39) are from the buffered data bus and are latched when the clock input signal CK is enabled by port addresses FEF8hex - FEFBhex. These port addresses are decoded by the combinational logic circuit in Figure 3-6. U39 will latch the desired data bits on the trailing edge of the type-select signal TYPECTL' at address FEF8H - FEFBH.

3.4 Output Registers for PROM address and Chip Enables

According to programming step a in section 2.2, the selected word needs to be addressed and the PROM output must be disabled. Figure 3-9 is a circuit used to latch PROM address bits A0 through A4, to activate the chip enable input E' for type_1, and to provide proper grounding for the PROM. The ground (GND) of type_1 is at pin 8 which is not the same as the 24-pin socket's ground pin (pin 12). It is therefore necessary to have a low condition at pin 8 (AX8) to properly connect type_1. After the specified PROM is selected, the CPU issues the address of the word to be programmed. The octal D-type latch (U7), the SN74LS374, is used to capture the data sent by the OUT instructions from the CPU. The output control input of U7 is connected to the type_1 enable signal PTYPE1'. The clock signal CK is ACTL1' which is fed from U5 in Figure 3-5, so that the port address FEE1hex is used to activate U7.

Type_2 PROMs have 9 address lines (A0 - A8), so that the address inputs to these devices must be treated as two separate output ports latched by ACTL2A' and ACTL2B'. One output port ACTL2A' is for address inputs A0-A7. The other output port ACTL2B' is for address input A8 and enable input E'. To realize these output ports, two SN74LS374's (U12 and

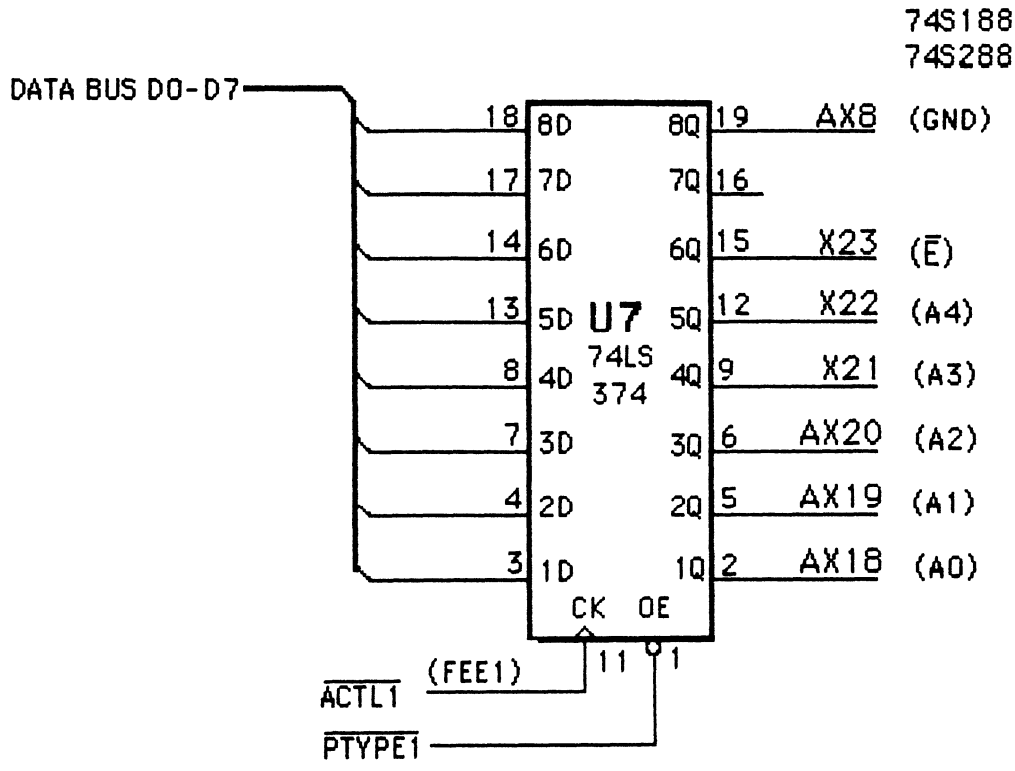


Figure 3-9 Address and chip enable latch for type_1

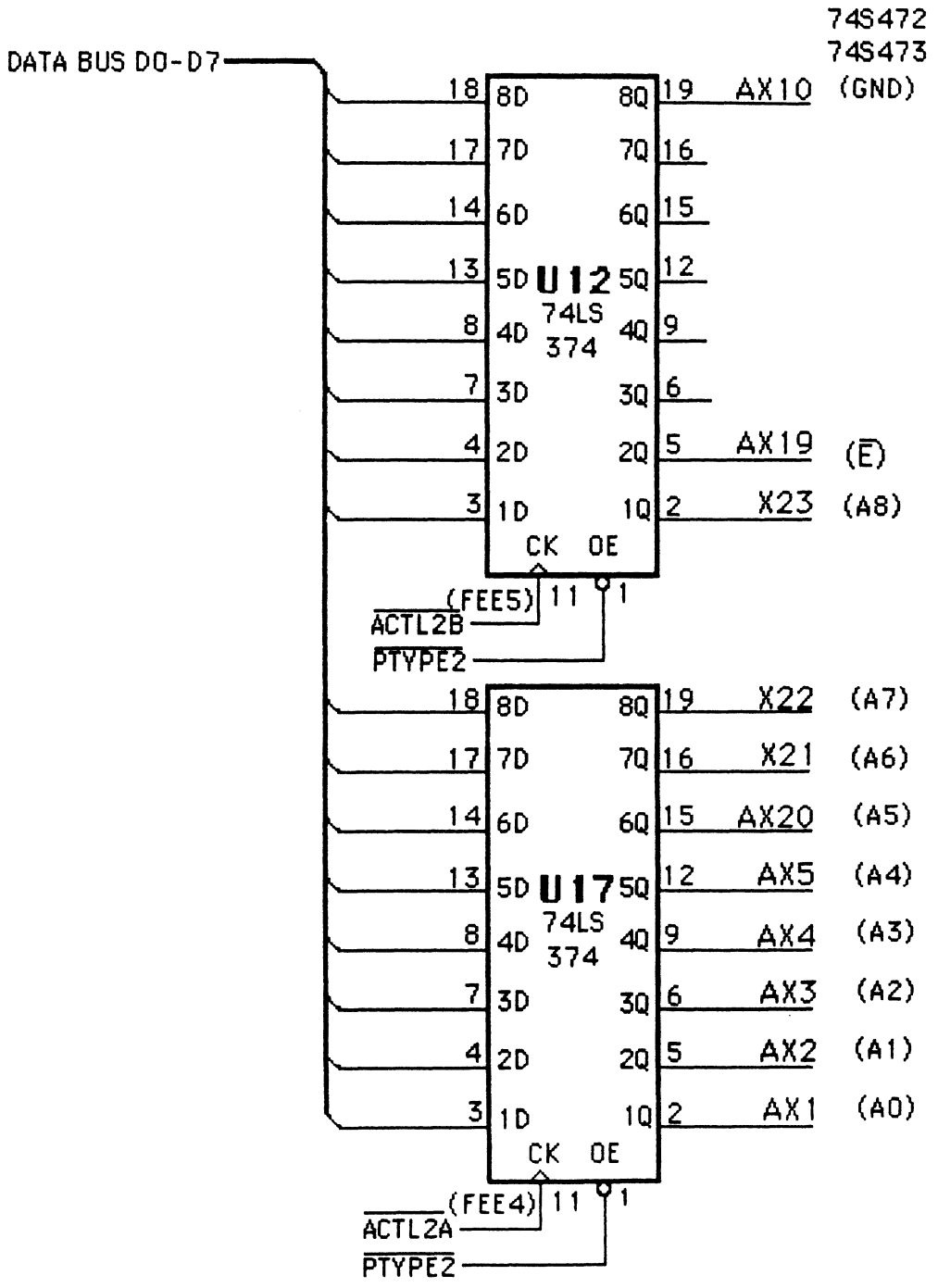


Figure 3-10 Address and Chip enable latches for type_2

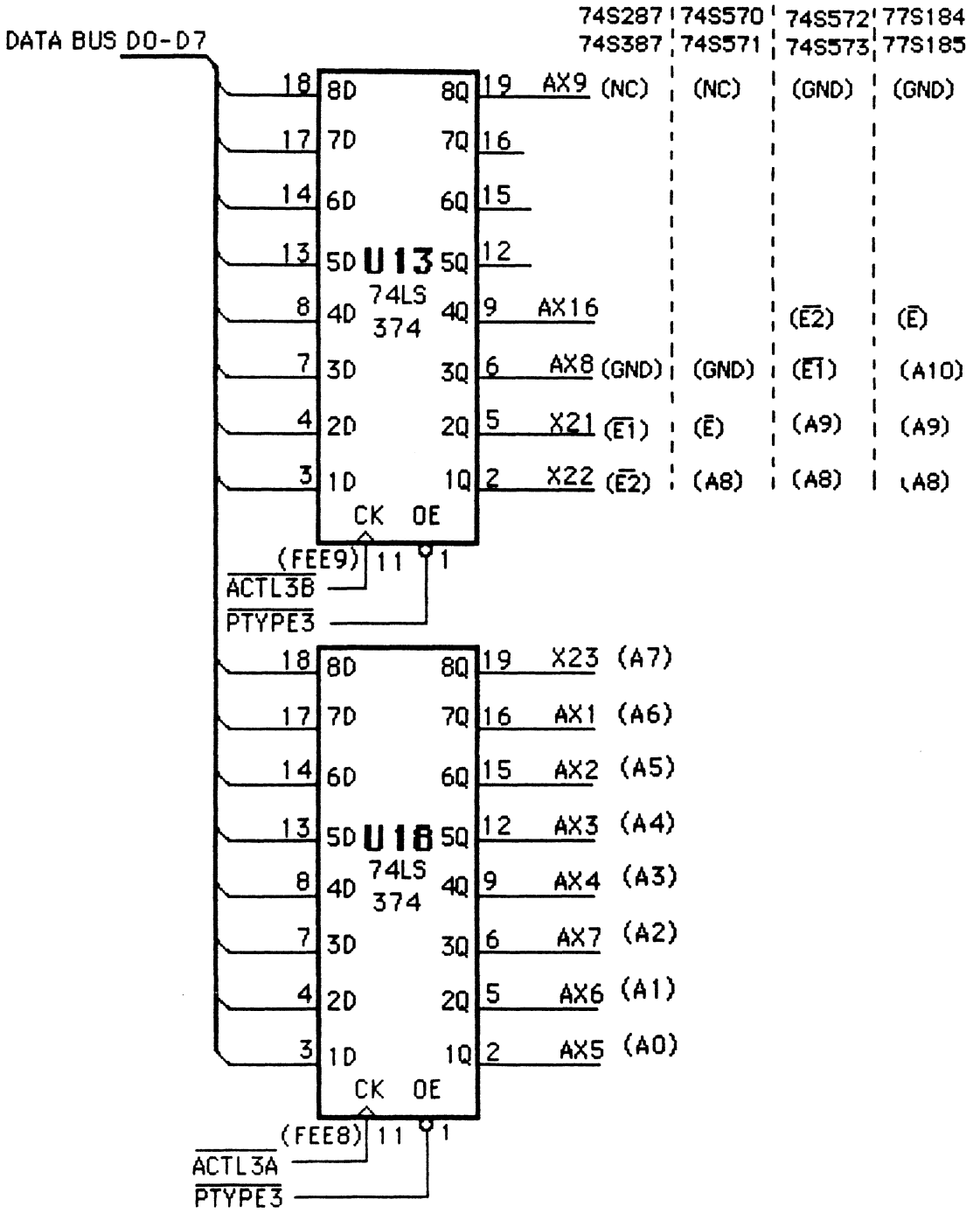


Figure 3-11 Address and chip enable latches for type_3

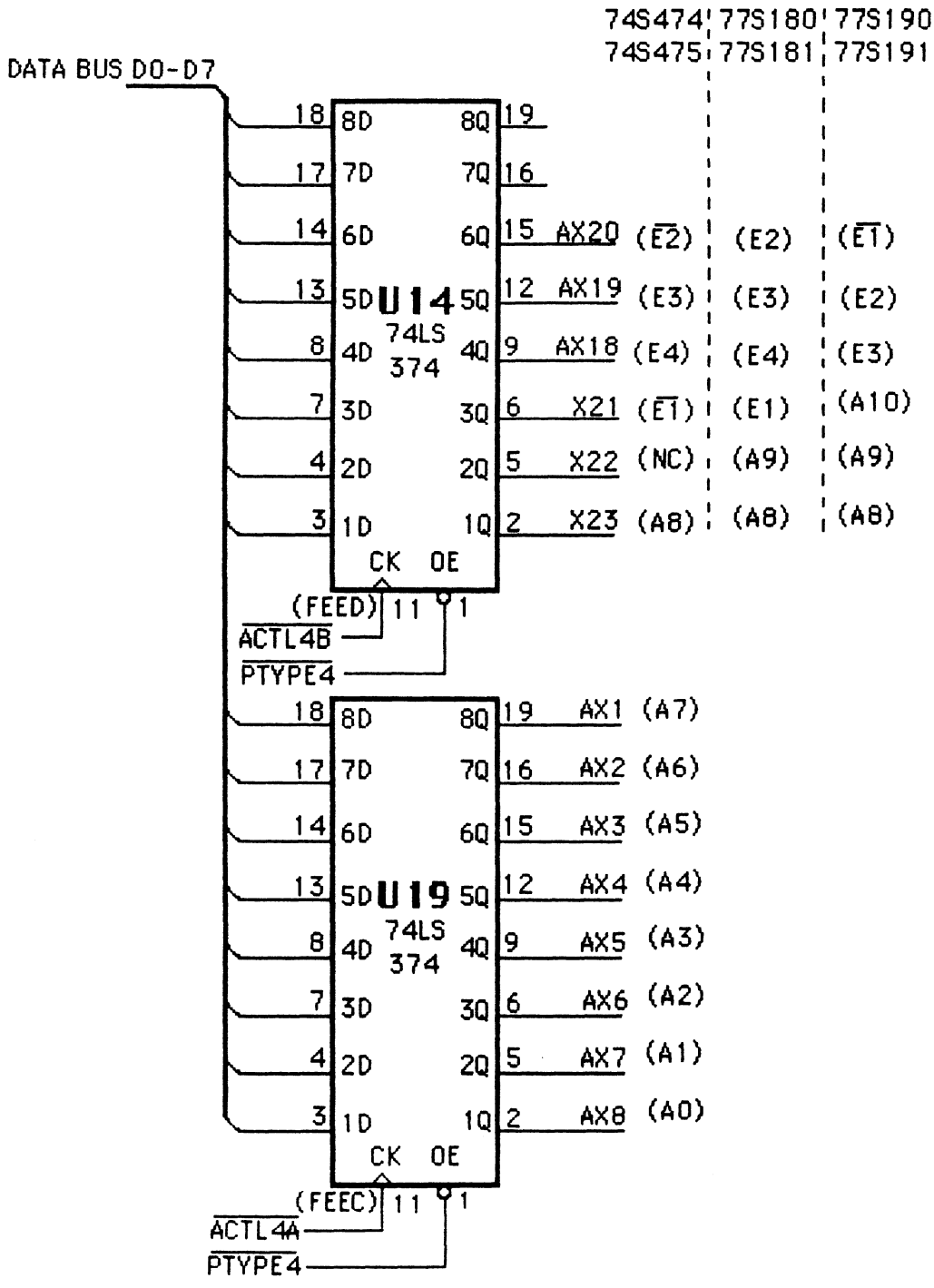


Figure 3-12 Address and chip enable latches for type_4

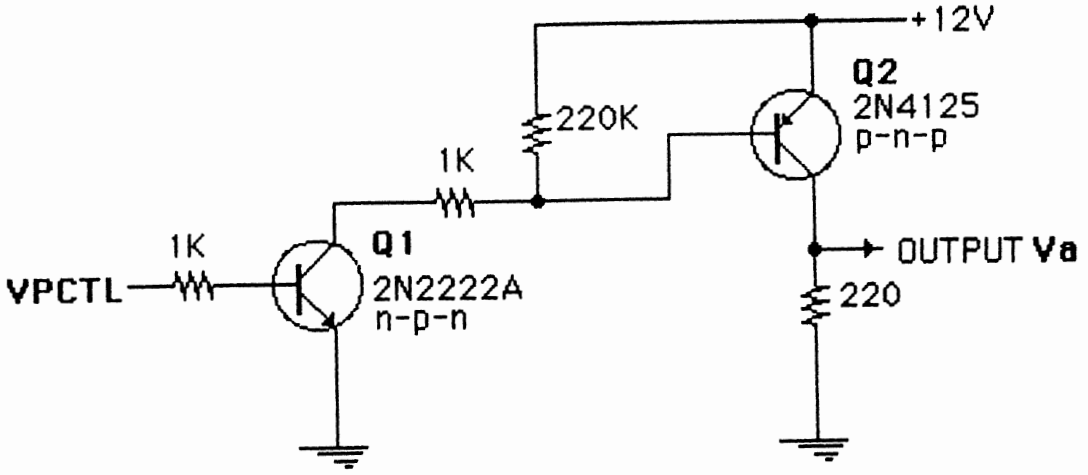
U17) are used. The hardware required for this is shown in Figure 3-10. The ground (GND) of type_2 is at the pin 10 position and is supplied by U12. In the same way, PROM type_3 and type_4 which have more than 8 address lines are addressed by the circuits shown in Figure 3-11 and Figure 3-12 respectively. Note that type_4 PROMs have the ground (GND) at the pin 12 position which is already connected to GND. The figures show what outputs must be present for each PROM.

3.5 High Current Driver for the PROM Vcc Input

In Figure 3-2 the PROM socket pin 24, which is Vcc for all PROM types, is connected to the signal "V24" and also to the +5-volt power level through D4 (1N914). While the PROM is being programmed, the V24 input signal supplies the desired 10.5V ±0.5V to pin 24, but otherwise it needs to be kept at the normal TTL voltage level in order to avoid heat damage. To achieve this requirement, Q1 and Q2, a n-p-n transistor and a p-n-p transistor respectively, are used as a switch for V24.

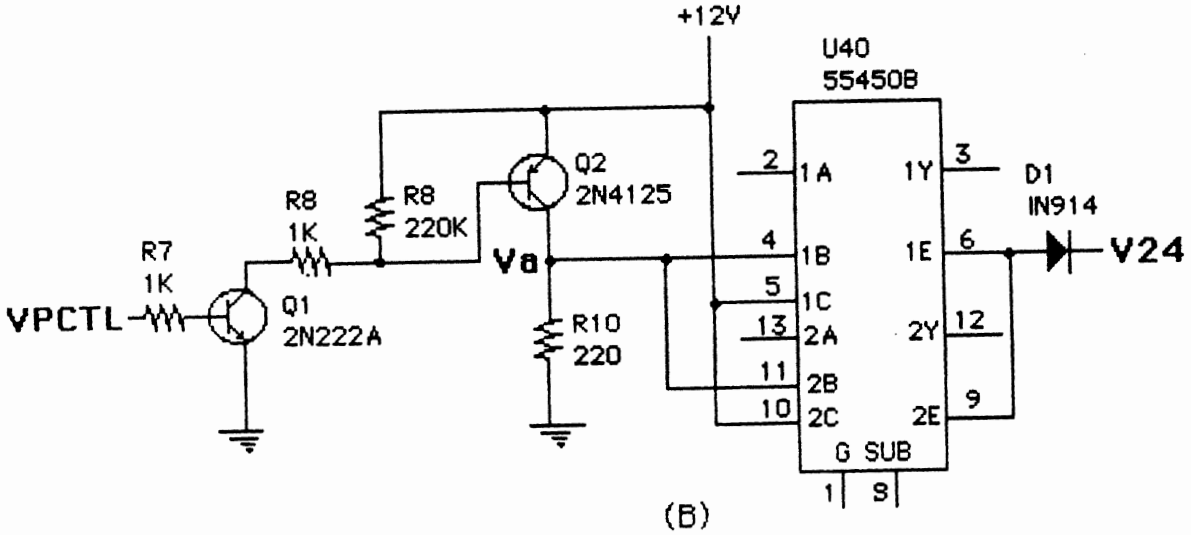
Figure 3-13a is the circuit for voltage switching for V24. The control for this VPCTL is latched into U35, a 74LS374 at port address FEE0hex. When the VPCTL input signal from the U35 output pin is in a high condition, Q1 and Q2 are ON and the output Va is equal to 12 volts to supply the desired PROM Vcc voltage. If the VPCTL input signal is in a low condition, Q1 and Q2 are OFF, the output Va is equal to 0 volts and V24 is blocked from pin 24. In this situation, the PROM is supplied by the +5-volt power level through the 1N914 diode.

Note that in Figure 3-13b the output Va of Q2 is connected to the SN554505B (Appendix E) to increase current. This dual peripheral driver has



VPCTL=" 1 " Q1, Q2 ON
VPCTL=" 0 " Q1, Q2 OFF

(A)



(B)

Figure 3-13 Programming voltage with high current driver for PROM Vcc input pin

two standard Series 54/74 TTL gates and two uncommitted high-current, high-voltage n-p-n transistors. Here the two n-p-n transistors are used as emitter followers to obtain higher current for programming the PROM.

3.6 Output/Input Registers for PROM Data

An input register can be thought of as a memory location or an I/O port. To program a PROM, the CPU needs to issue specified data to the PROM's data output pins. This is accomplished by an output register. When checking to insure that the PROM device has been programmed correctly, reading the data out of the PROM is needed. Sensing information from an interface is normally accomplished by using an input register. In this design a port register that acts as either input or output register is used. The direction can be changed under program control.

Figure 3-14a illustrates the port design for PROM type_1. It has programmable input or output bits. This design not only latches its data from the CPU, but also is able to read data from the PROM. The portion which outputs from the CPU is similar to the latched-output register discussed in section 3.4. It is implemented with U8, an SN74LS374, which is used to latch data from the CPU for the PROM. The clock input CK of U8 is enabled by DCTL1' at port address FEE2hex. The output control OE is connected to the type_1 enable signal PTYPE1'. Each output bit of the output register (U8) is used to drive an open-collector device: in this design, an SN7407. The output of each open-collector device is pulled high through a 1k resistor to +5 volts. These are now tied to the inputs of the input register (U9) and also connected to the corresponding PROM output terminals

(Q1 - Q8) in the socket. To use this design for PROM data, simply write the desired data to the output register (U8) using an OUT instruction.

Figure 3-15 shows the output circuit of one of the SN7407 buffers. When the data input is HIGH, with this noninverting buffer, the output will be equal to "Vi" (i=1 - 11, and 13 - 20) and is applied to the PROM output (Qi) for programming. The programming voltage "Vi" is switched by the DG201s which is discussed in the next section. If the data input signal is changed from HIGH to LOW, then the SN7407 buffer output will be "ON" and the PROM output will be grounded. During this time, the DG201 keeps Vi OFF.

When the programming command is executed, the signal CCTL1' is HIGH and the data read register (U9), an SN74LS244 bus transceiver, is inactive and not connected to the buffered data bus (D0 - D7). However, the input register is activated for a read or verification operation. To read data from the PROM, first write all ones (0FFhex) to U8. This will turn off the open-collector drivers and allow the PROM outputs to control the inputs to U9. CCTL1' (at address FEE3hex) activates U9 and stays LOW to keep it active (Figure 3-13a). This will cause the output lines of the 74LS244 to be connected to the PC-XT's data bus and data can be transferred to the CPU.

The output and input registers will never both be active at the same time. These registers are active when an OUT or IN instruction is executed and the proper I/O port address is decoded. All of their control signals for PROM types 1 through 4 are generated from the 74LS154 (U5).

In Figure 3-14a, the input lines of U9 are connected to bits CD1 through CD7 and CD17 of the PROM socket which are from the PROM's output terminals Q0 through Q7 for type_1. The corresponding output lines are then

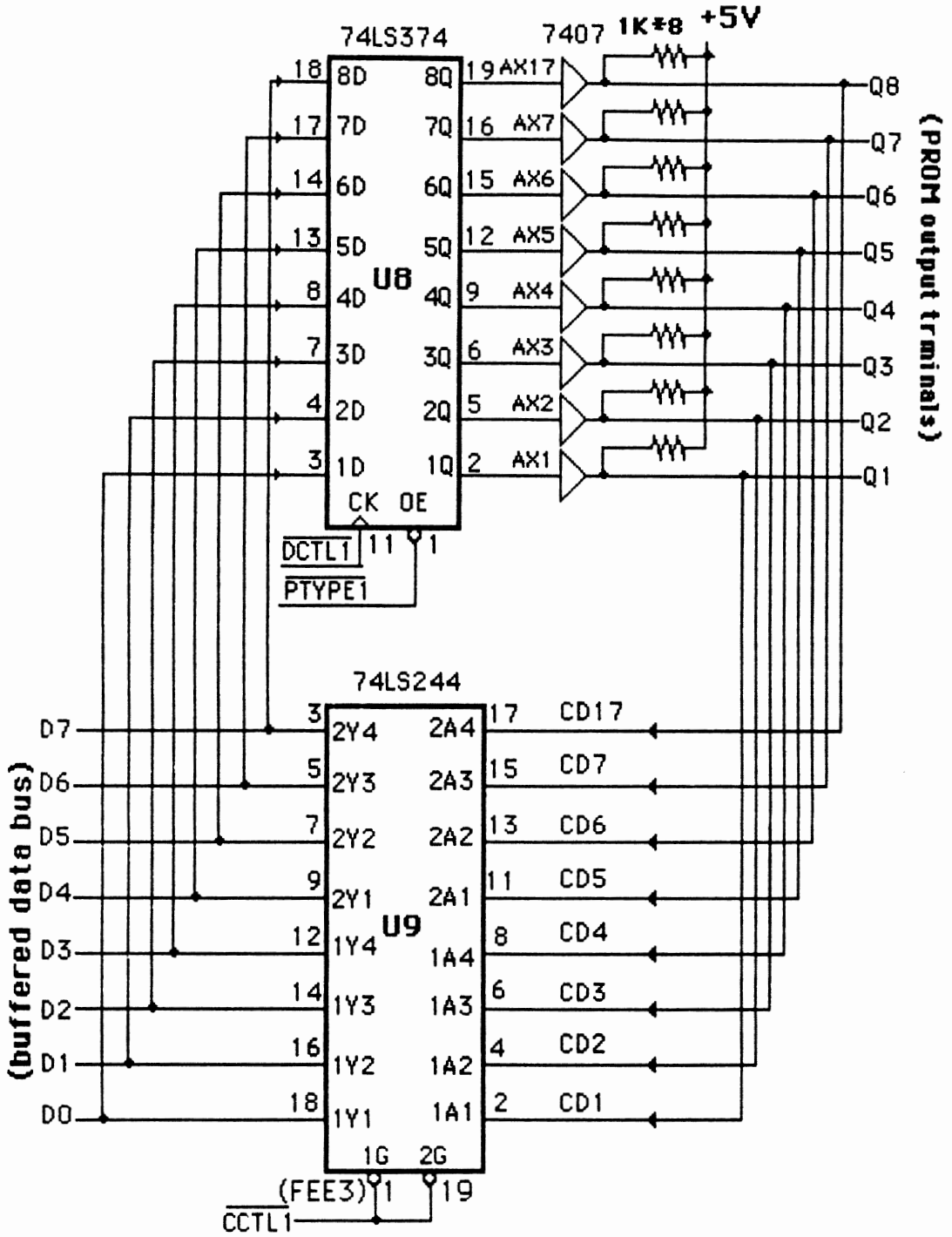


Figure 3-14a A combinational DI/DO register for type_1

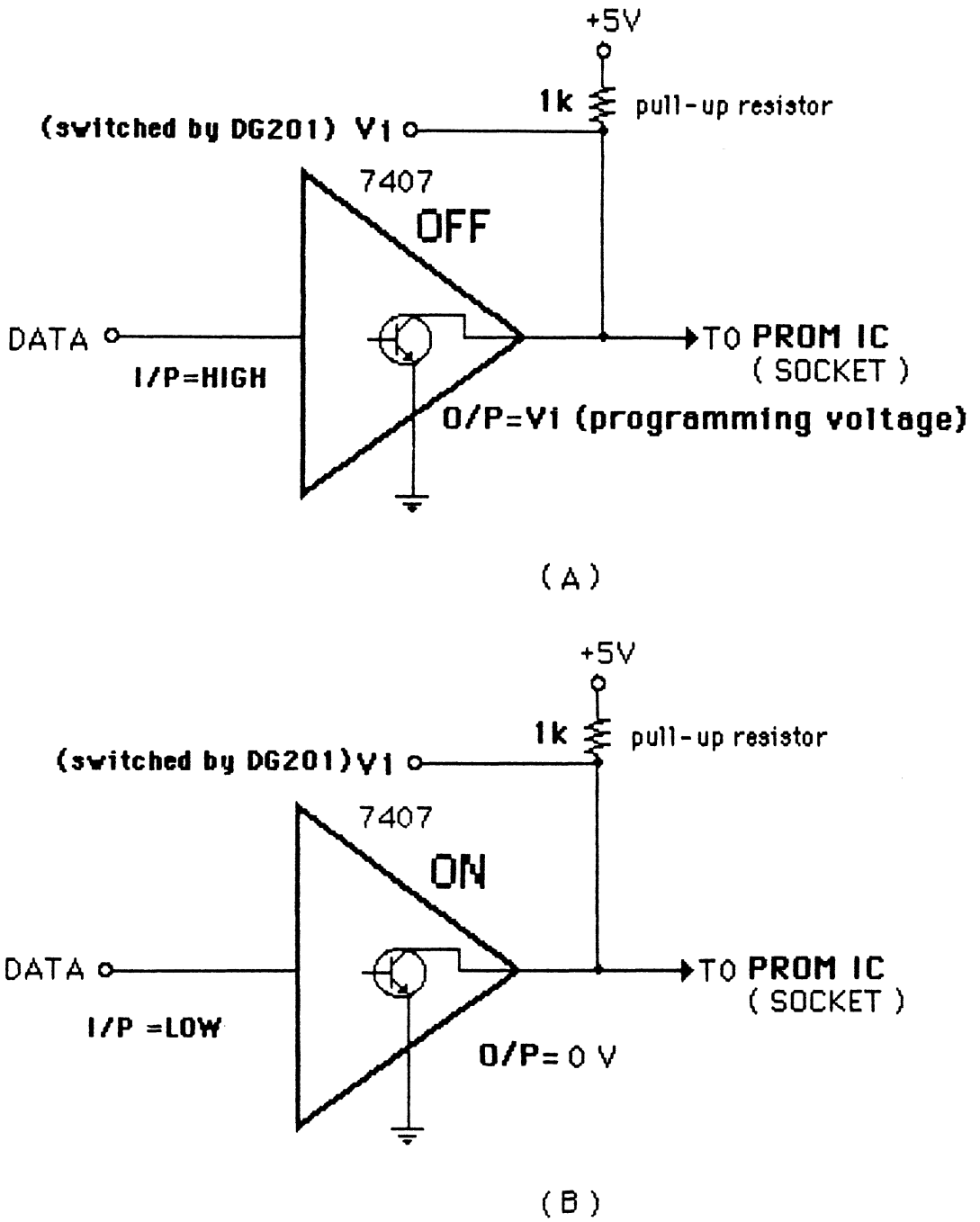


Figure 3-15 Operation for buffer/driver SN7407
(A) Input signal on logic 1
(B) Input signal on logic 0

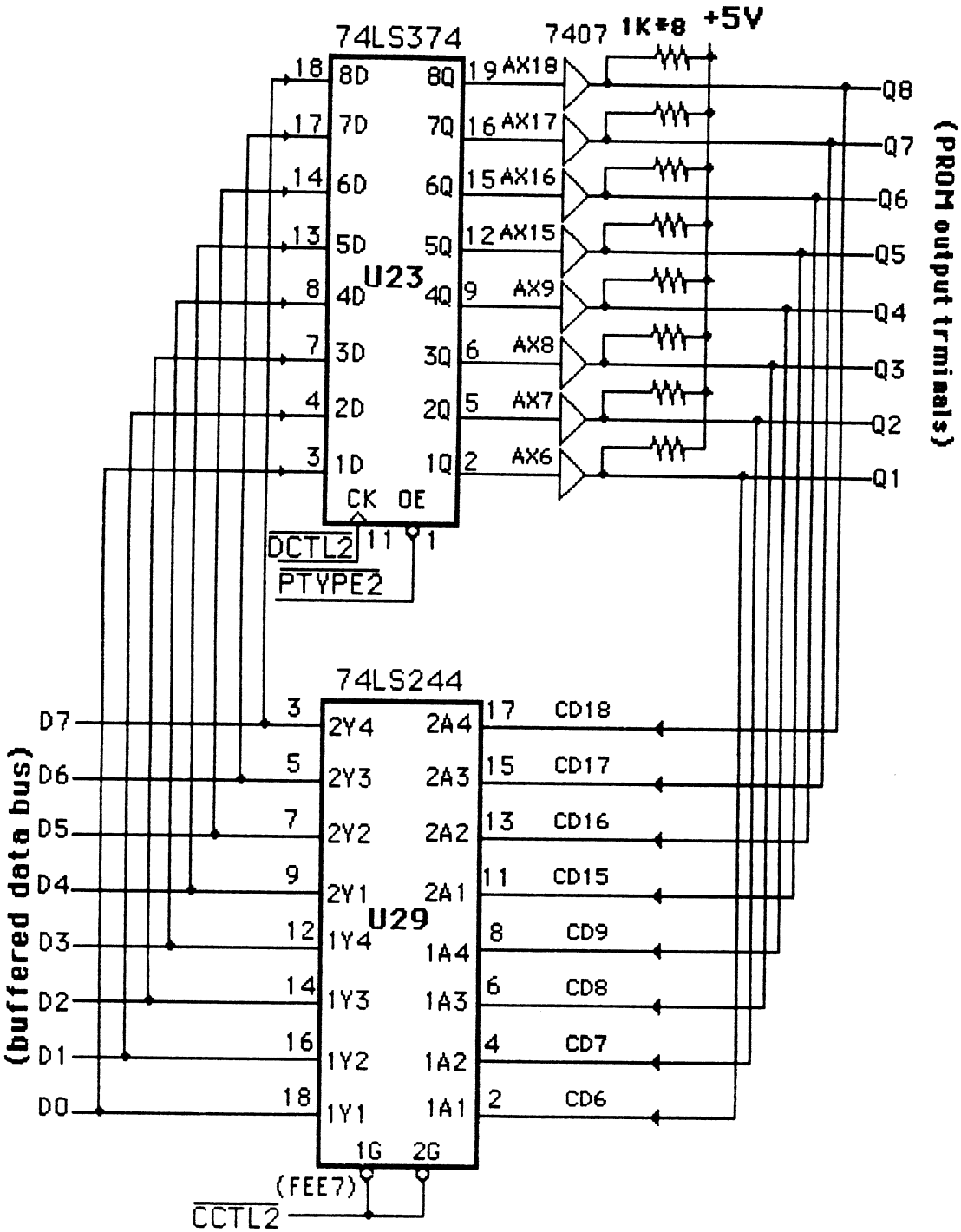


Figure 3-14b A combinational DI/DO register for type_2

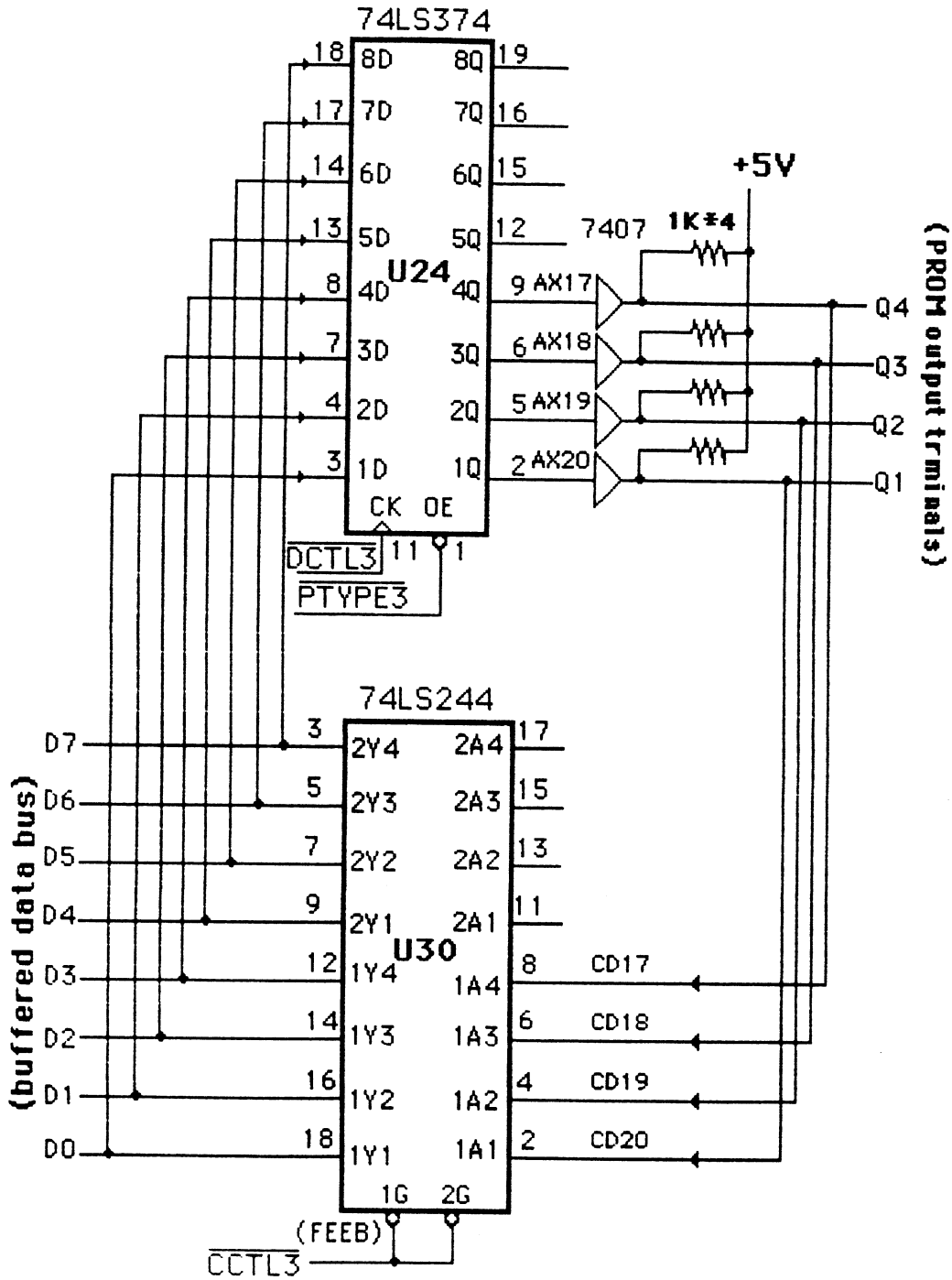


Figure 3-14c A combinational DI/DO register for type_3

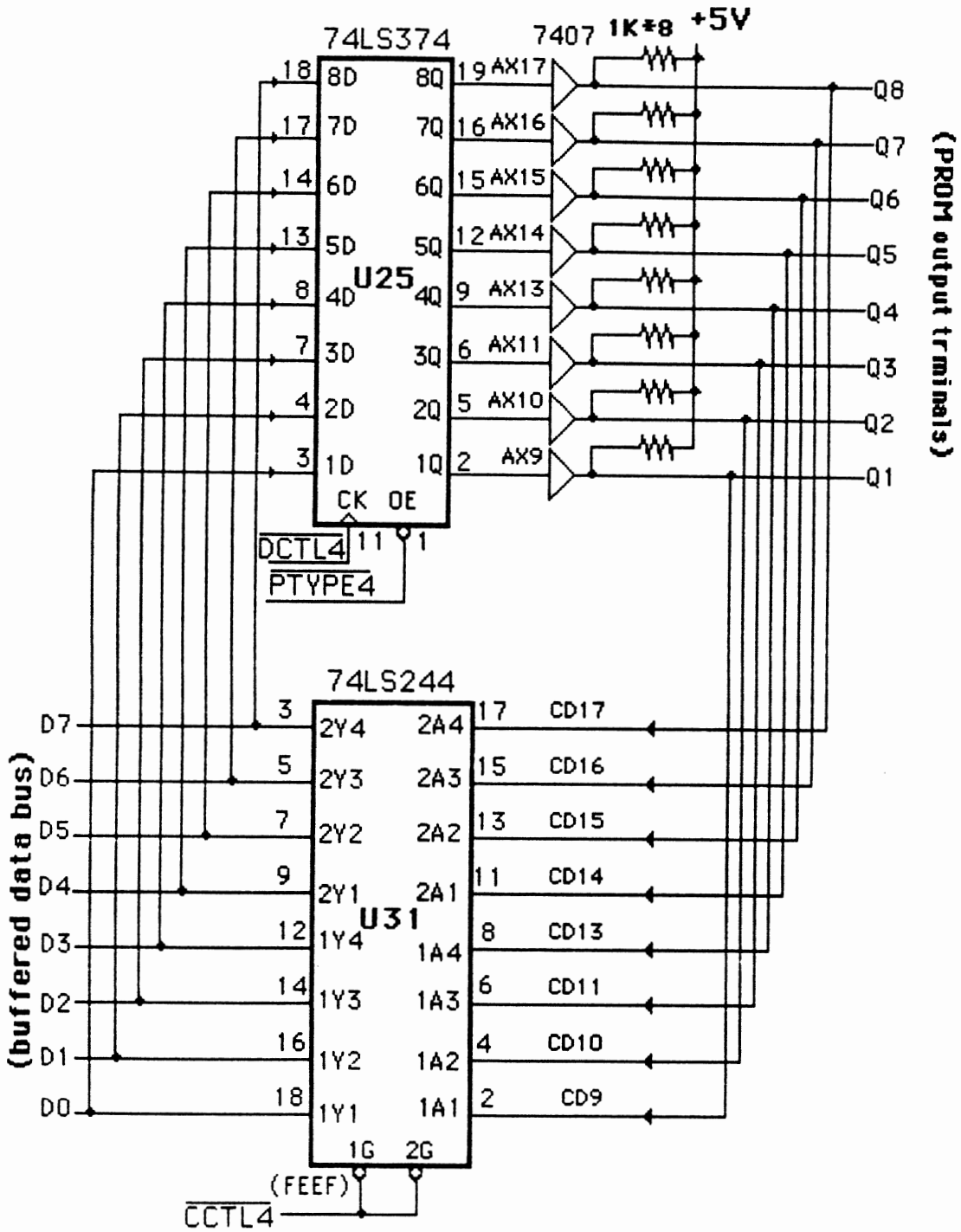


Figure 3-14d Acombinational DI/DO register for type_4

connected to the bits D0 through D7 of the card's data bus and transferred to the PC-XT's I/O slot through data buffer U32. In the same way, the inputs of U29 are connected to bits CD6 - CD9 and CD15 - CD18 for type_2 In Figure 3-14b. In Figure 3-14c, the inputs of U30 are bits CD17 through CD20 for type_3. In Figure 3-14d, the input signals of the U31 are bits CD9 - CD11 and CD13 - CD17 for type_4.

3.7 PROM Output Voltage Switching at Qi

When the PROM is being programmed, each output Qi (i=1 to 8), needs to be supplied by +10.5 ±0.5 volts. Figure 3-16 is the circuit schematic designed to achieve this. Because there are 19 different positions (pin1-pin11 and pin13-pin20) in the IC socket for prom output terminal pins, five DG201s (Appendix F) are needed to switch all the possible input signals.

The DG201 is a quad single pole, single throw analog switch which employs a parallel combination of a PMOS and an NMOS field effect transistor. Figure 3-17 is the pin configuration and its function diagram. These DG201s are used to switch the programming voltages to the correct pins. In the ON condition each switch will conduct current in either direction, and in the OFF condition, each switch will block voltages up to 30 volts peak-to-peak. The result is a savings in wiring complexity and board space.

In Figure 3-17a, the DG201 (U20) is used to switch the programming voltages V1 through V4 which correspond to the PROM IC's pin1 - pin4 as seen in Figure 3-2. See Figure 3-16. U21 is used to switch the programming voltage to V5 - V8, U22 is used to switch V9 - V11 and V13,

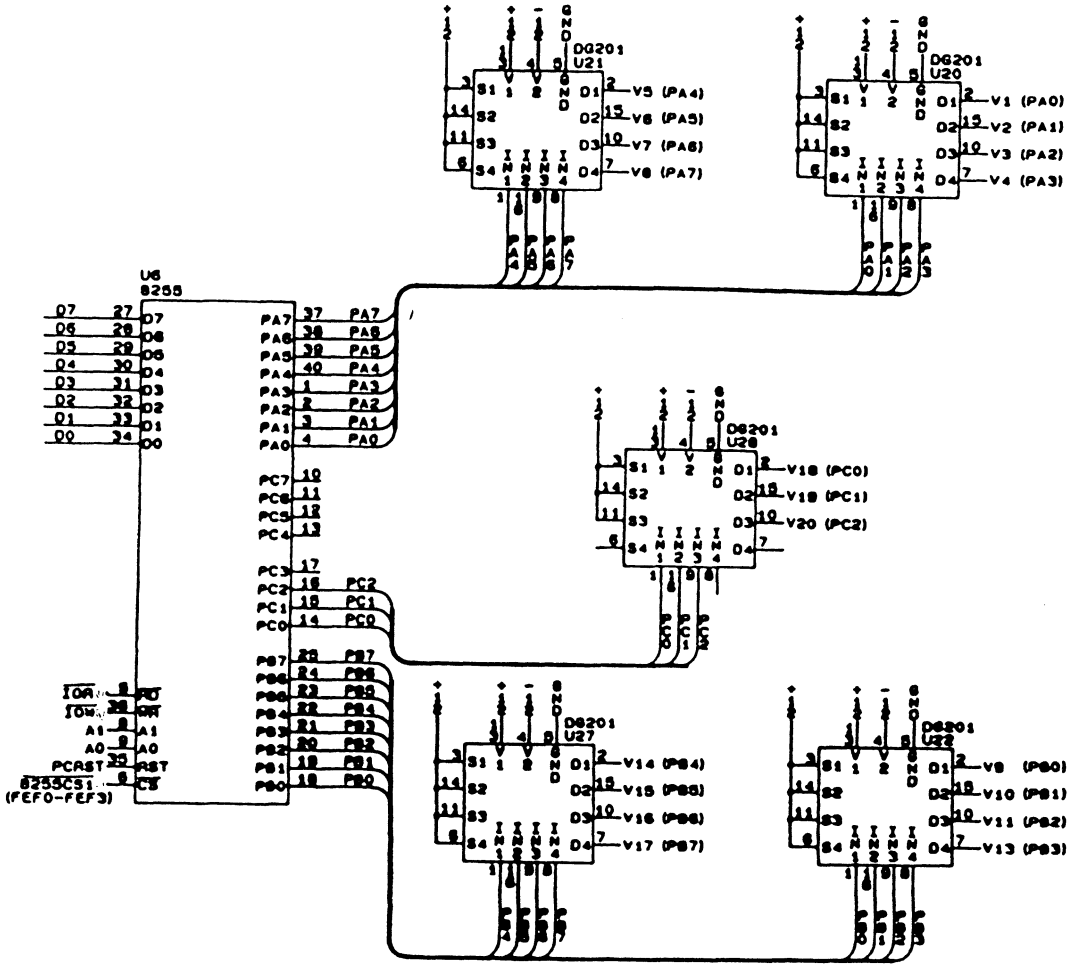


Figure 3-16 PROM output (Qi) voltage switch

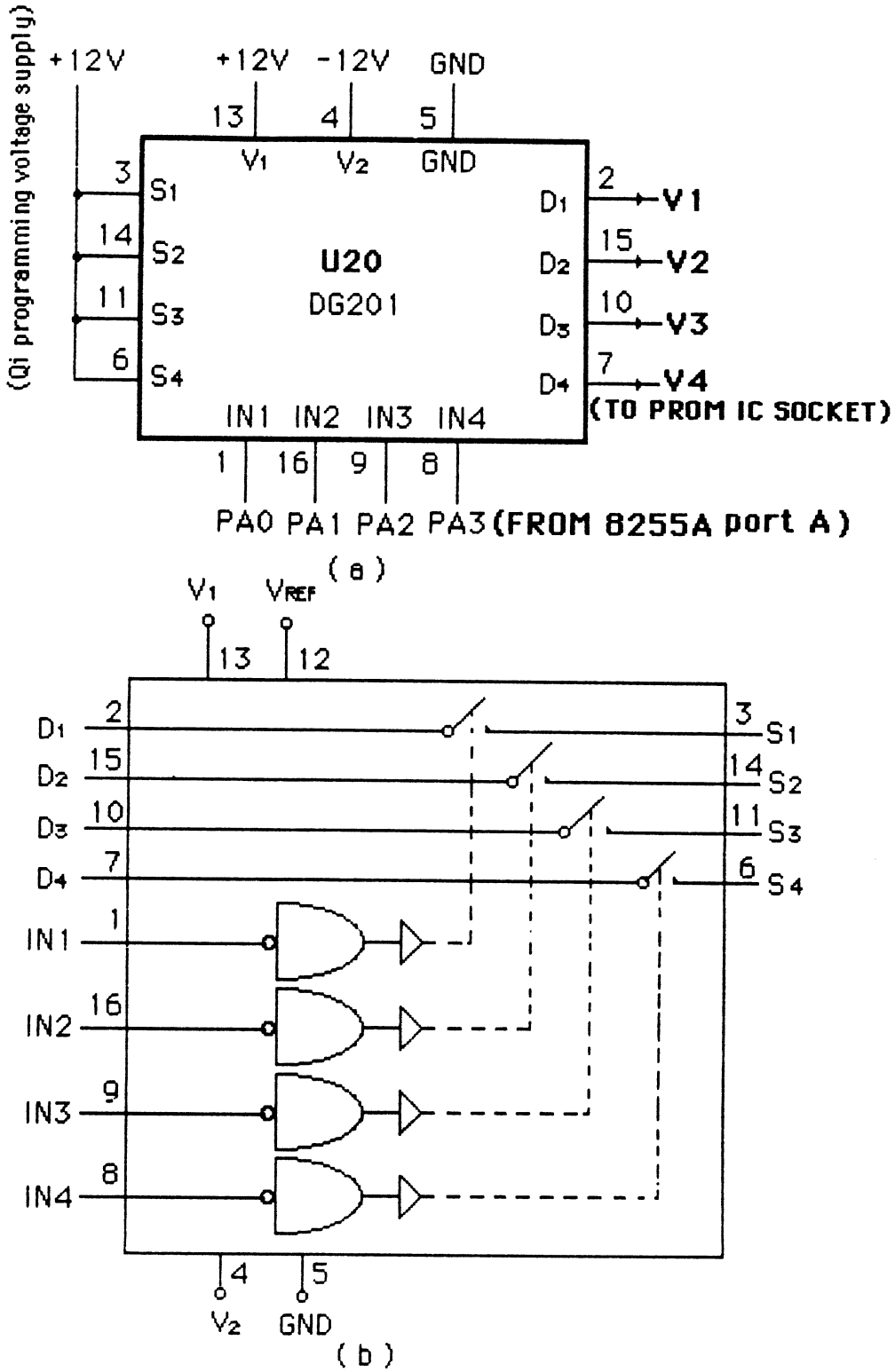


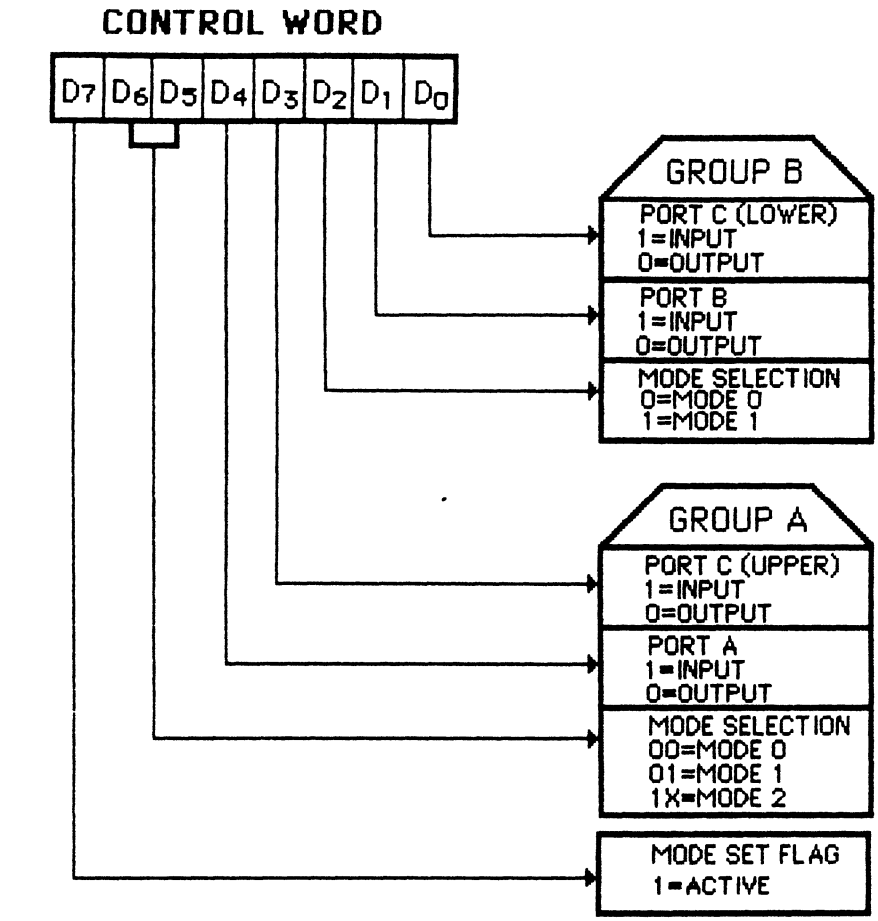
Figure 3-17 Circuit schematic of a DG201 with its functional diagram

U27 is used to switch V14 through V17, and U28 is intended to switch V18 - V20.

Suppose it is desired to program Qn. With a logic " 0 " at the proper DG201 input, the switch is ON and the +12V programming voltage is applied to the PROM's Qn output to program that bit. With a logic " 1 "at the input, the switch is OFF and no programming voltage goes to the PROM. Because the PROM must be programmed bit by bit, each time only one switch will be ON.

The DG201 inputs are controlled by an Intel 8255A programmable peripheral interface (U6). The Intel 8255A (Appendix G) is a general purpose programmable I/O device. Its function is to interface peripheral equipment to the microcomputer system bus. It has 24 I/O pins which may be individually programmed in two groups of 12 and used in three major modes of operation. In this system, the 8255A needs to generate 19 controlled output signals used to switch the desired programming voltages for the specified PROM outputs. Thus, all 24 I/O pins are set as outputs (Mode 0). The CPU "outputs" a control word to the 8255A that initializes the functional configuration of the 8255A. The functional configuration of the 8255A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures. Figure 3-18 shows the general control word format. In this design, 80hex (ie.,10000000 B) gives the proper results.

Figure 3-19 gives the pin connections of the Intel 8255A. The PA0-PA7, PB0-PB7, and PC0-PC2 output pins of the 8255A are all connected to DG201s to switch the specified programming voltage ON or OFF. PA0-PA3



CONTROL WORD #0 (**MODE 0**)

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

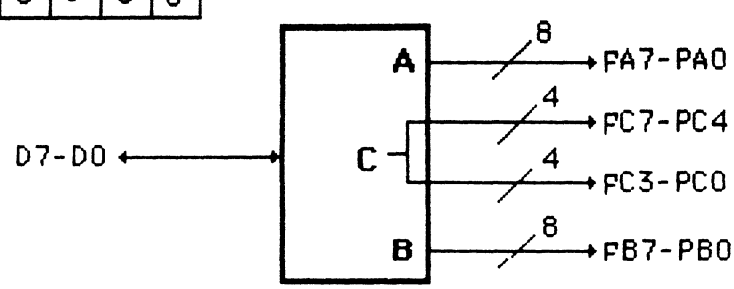


Figure 3-18 Mode 0 configuration and control word definition

are used to switch U20, PA4-PA7 are connected to U21, PB0-PB3 are used to switch U22, PB4-PB7 are used to switch U27, and PC0-PC2 are used to switch U28.

The pin35 RESET input of the 8255A is connected to the PC-XT's I/O slot RESET DRV pinout. When this input goes "high" all ports will be reset and all the DG201's switches are off.

The chip select (CS') of the 8255A is connected to the 8255CS1' signal generated in Figure 3-4. A low on this input pin will enable communication between the 8255A and the CPU. The 8255A is enabled at address ports FEF0H through FEF3H: FEF0hex for port A, FEF1hex for port B, FEF2hex for port C, and FEF3hex for the control word.

The port addresses for the 8255A are:

HEX address used	Function
FEF0H	for port A
FEF1H	for port B
FEF2H	for port C
FEF3H	control word

Pin27-pin34 of the 8255A are connected to the system data bus D0-D7. Data is transmitted or received by executing input or output instructions from the CPU.

The RD' input is connected to PC-XT's I/O slot IOR' line. A "low" on this input pin will enable the 8255A to send status information to the CPU on the data bus. The WR' input is connected to the PC-XT's IOW' line. A "low" on this input pin will allow the CPU to write data and control words

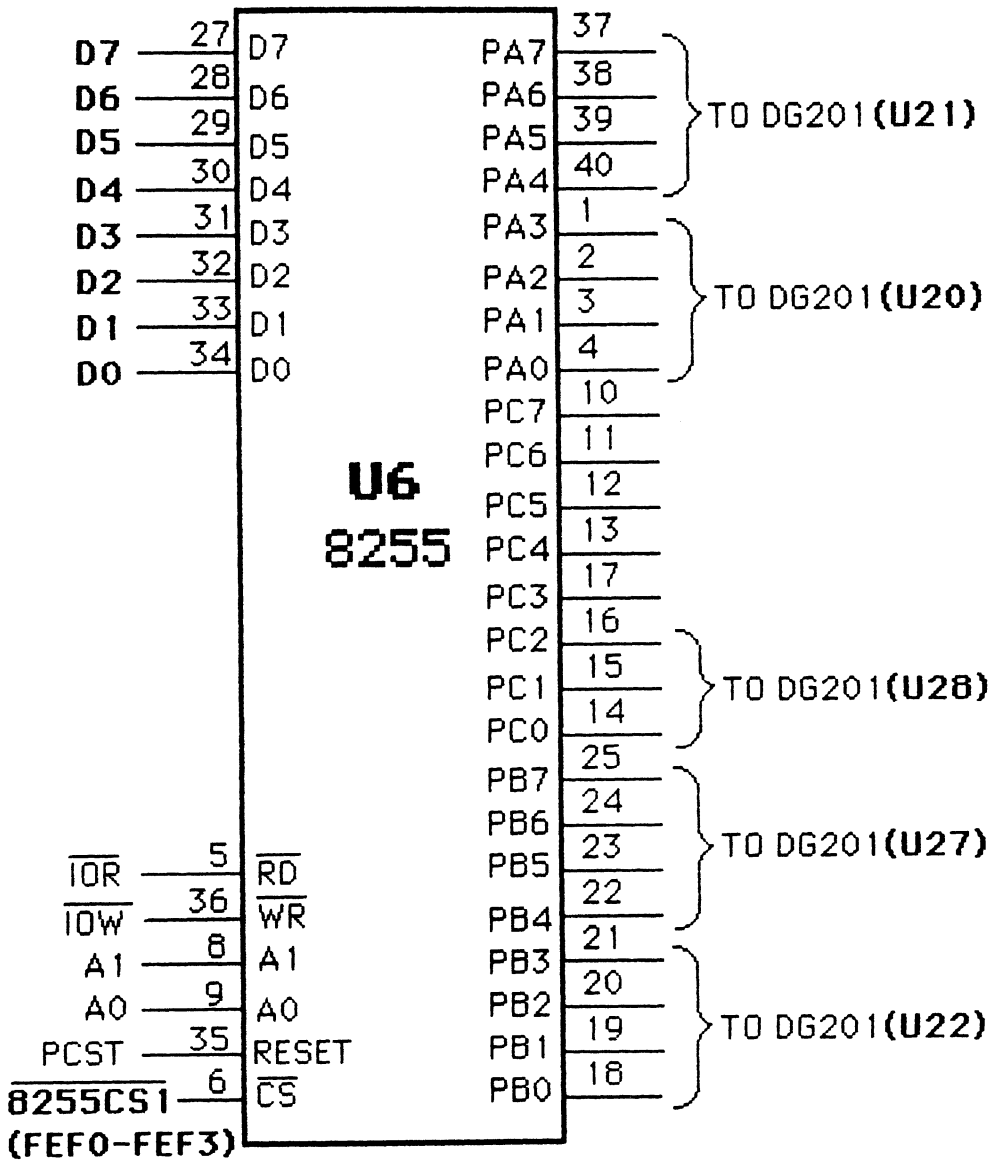


Figure 3-19 Using a programmable peripheral interface to control the Qi voltages on the PROM

into the 8255A. The RD' and WR' signal will never be "low" at the same time.

The A0 and A1 input pins of U6 are connected to the least significant bits of the system address bus (external A0 and A1). These two input signals are used to control the selection of one of the three ports or the control word register. 00hex will select port A, 01hex will select port B, 02hex will select port C and 03hex will allow writing of a control word.

3.8 Summary

In this chapter the hardware necessary to construct the IBM PC-XT controlled PROM programmer has been described. However, in the discussion of the design of this system, many interfacing techniques that are applicable to general microprocessor-controlled systems are also shown.

The complete hardware circuit schematic is shown in Appendix H and its parts listing is shown in Appendix I. In Chapter 4 the software to control the system is discussed in detail.

CHAPTER 4

Software Design

In this chapter the software required to control the system hardware is presented. The software is written in Intel 8086/8088 assembly language and turbo C language. To achieve the best performance, the functions that relate to the programmer card are written in assembly language. These are the BLANK CHECK, PROGRAMMING, VERIFY, and READ functions; the functions that relate to the IBM PC-XT's buffer memory are written in turbo C.

In the following sections, familiarity with C language and assembly language are assumed. The discussions emphasize those functions written in assembly. Key subroutines are discussed with their flowcharts.

4.1 Getting Started

Before the system software is described a few preliminary details need to be discussed. One of these is the software layout of the system; another is hardware initialization.

The I/O map is a listing of all of the address space for the input and output ports and of the corresponding function of each block. The mapping of the addresses for the system hardware presented in Chapter 3 is given in Table 3-2. Table 4-1 shows the label used in the software for each port.

Table 4-1 System control words

```
----- control words for programming the 8255A -----

        u68255a     equ     0fef0H
        u68255b     equ     0fef1H
        u68255c     equ     0fef2H
        u68255ctl   equ     0fef3H
        u68255cmd   equ     080H    ;command word for 8255A

----- control word for setting PROM Vcc voltage -----

        epctl       equ     0fee0H

----- control word for PROM type selecting -----

        typectl     equ     0fef8H

----- control words for type_1 enable signals-----

        actl1       equ     0fee1H
        dctl1       equ     0fee2H
        cctl1       equ     0fee3H

----- control words for type_2 enable signals-----

        actl2a      equ     0fee4H
        actl2b      equ     0fee5H
        dctl2       equ     0fee6H
        cctl2       equ     0fee7H

----- control words for type_3 enable signals -----

        actl3a      equ     0fee8H
        actl3b      equ     0fee9H
        dctl3       equ     0feeaH
        cctl3       equ     0feebH

----- control words for type_4 enable signals -----

        actl4a      equ     0feecH
        actl4b      equ     0feedH
        dctl4       equ     0feeeH
        cctl4       equ     0feefH
```

The input and output devices must be initialized when the power is first applied to the programmer. Figure 4-1 is the program segment for the programmer initialization. This is the major hardware initialization. More software initialization steps are needed as well. For instance, variables must be set to certain values. These types of initializations will be discussed as the need arises.

```
_init   proc    near
        push    ax              ; push AX onto stack
        push    dx              ; push DX onto stack

        mov     al,u68255cmd    ; set 8255A to mode 0
        mov     dx,u68255ctl    ; set port a, port b, and port c
        out     dx,al           ; all in output mode
        mov     al,0ffh
        mov     dx,u68255a
        out     dx,al           ; set 8255A port a = 0ffh
        mov     dx,u68255b
        out     dx,al           ; set 8255A port b = 0ffh
        mov     dx,u68255c
        out     dx,al           ; set 8255A port c = 0ffh

        mov     al,0
        mov     dx,epctl
        out     dx,al           ; U35 set and PROM Vcc input = 5V

        mov     al,0fh
        mov     dx,typectl
        out     dx,al           ; U39 set and disable all PROM inputs

        pop     dx              ; restore DX
        pop     ax              ; restore AX
        ret                     ; return to calling program (terminated)
```

Figure 4-1 Program for system hardware initialization

4.2 Main Program

In this work many function commands are used, so the main program needs to include all the function subroutines. The main flowchart of the software is shown in Figure 4-2. It displays the direction of program execution and shows the subroutines which are used for each function command.

This main program directs the system execution based on the command keyword that is input, so that the first event to occur in the main program is to call the subroutine *read keyboard* for scanning the input keyword. The system will not return from this subroutine until this command has been executed or a unvalid command keyword has been input to the system.

Since this program interacts with the keyboard, the system must wait for a particular key to be pressed before executing certain routines in the program. The main idea is this: The system inputs a command keyword. Then, based on what the function of that particular key is, the system executes the specified subroutine. A complete list of the system commands and their functions is given in Table 4-2.

4.3 Subroutine for the BLANK CHECK Function

Before programming a PROM the " BLANK CHECK " procedure is usually executed. This function is useful to check if the PROM is blank. The flowchart for the blank check subroutine is given in Figure 4-3. There are four types of PROM's intended to work in this system, so the first event to

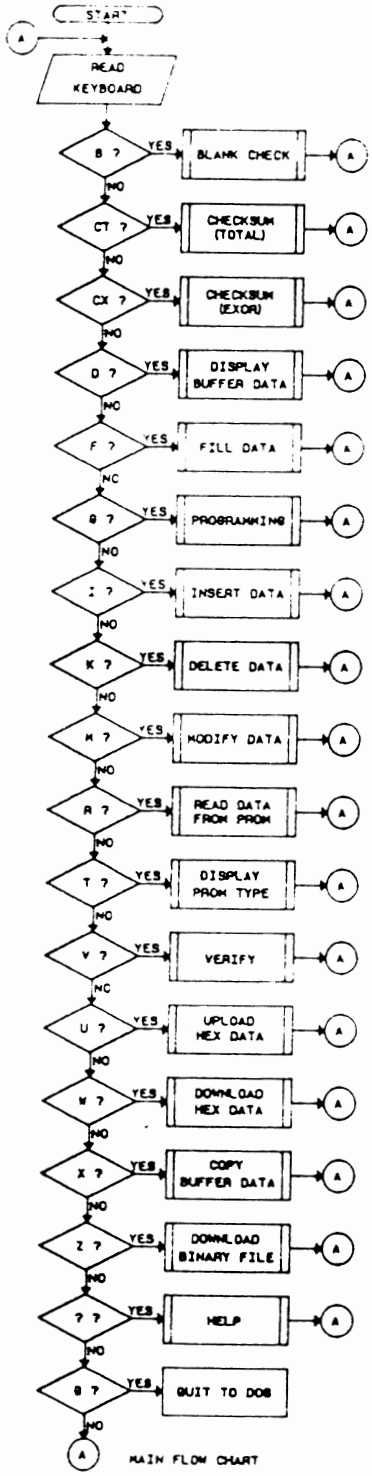


Figure 4-2 Flowchart for the main program

Table 4-1 Command summaries

COMMAND	DESCRIPTION
B T	BLANK CHECK
CT a1 a2 [a3]	CHECKSUM (TOTAL)
CX a1 a2 [a3]	CHECKSUM (EXOR)
D a1 a2	DISPLAY BUFFER MEMORY
F a1 a2 d	FILL DATA
G T[a1 a2 a3]	PROM PROGRAMMING
I a1	INSERT DATA
K a1 a2	DELETE DATA
M a1	MODIFY DATA
R T[a1 a2 a3]	READ DATA FROM PROM
T	DISPLAY PROM TYPES
V T[a1 a2 a3]	VERIFY
U	UPLOAD HEX FILE
W	DOWNLOAD HEX FILE
X a1 a2 a3	COPY BUFFER MEMORY
Z	DOWNLOAD BINARY FILE
?	HELP
Q	QUIT TO DOS

Note:

a1: start-address (hexadecimal)
a2: end-address (hexadecimal)
a3: destination-address (hexadecimal)
d: data (hexadecimal)
T: part number of a PROM

occur in each function subroutine is a *PROM type check*. This is done to test if the input PROM type is correct or not.

If the input type is not matched with the actual type, the subroutine will display " PROM Type Error " and return to the main program. This is accomplished by step 2. To configure for the specified PROM type and access the data, the hardware initialization must be made (step 3). The program segment for this hardware initialization is:

hardware initialization for type_1 :

```
mov     al,0
mov     dx,epctl
out     dx,al           ; U35 set and PROM Vcc input = 5V

mov     al,0ffh
mov     dx,dctl1        ; set all ones to turn off 7407
out     dx,al           ; U8 set and sets to read data from PROM

mov     al,0eh
mov     dx,typectl
out     dx,al           ; U39 set and enables PROM type_1
```

hardware initialization for type_2:

```
mov     al,0
mov     dx,epctl
out     dx,al           ; U35 set and PROM Vcc input = 5V

mov     al,0ffh
mov     dx,dctl2        ; set all ones to turn off 7407
out     dx,al           ; U23 set and sets to read data from PROM

mov     al,0dh
mov     dx,typectl
out     dx,al           ; U39 sets and enables PROM type_2
```

hardware initialization for Type_3:

```
mov     al,0
mov     dx,epctl
out     dx,al           ; U35 sets and PROM Vcc input = 5V

mov     al,0ffh
mov     dx,dctl3        ; set all ones to turn off 7407
out     dx,al           ; U24 sets and sets to read data from PROM

mov     al,0bh
mov     dx,typectl
out     dx,al           ; U39 sets and enables PROM type_3
```

hardware initialization for type_4:

```
mov     al,0
mov     dx,epctl
out     dx,al           ; U35 sets and PROM Vcc input = 5V

mov     al,0ffh
mov     dx,dctl4        ; set all ones to turn off 7407
out     dx,al           ; U25 sets and sets to read data from PROM

mov     al,07h
mov     dx,typectl
out     dx,al           ; U39 sets and enables PROM type_4
```

The system then reads data bytes from the PROM memory and writes that data into the IBM PC-XT's buffer (step 6 in the flowchart). The addresses are incremented and the data is checked at each address. This is done with a loop of steps 6 through 10. If all the data is equal to zero, the system then displays "Blank Check OK" on the monitor and returns to the main program. If any data not equal to zero is checked, the system will display "Blank Check Fail" and jump to the main calling program (step 11).

The program segment for the blank check function for type_1 is given in Figure 4-4. The beginning of this program segment is in step 3 in the flowchart (hardware initialization). The PROM memory size is set in CX

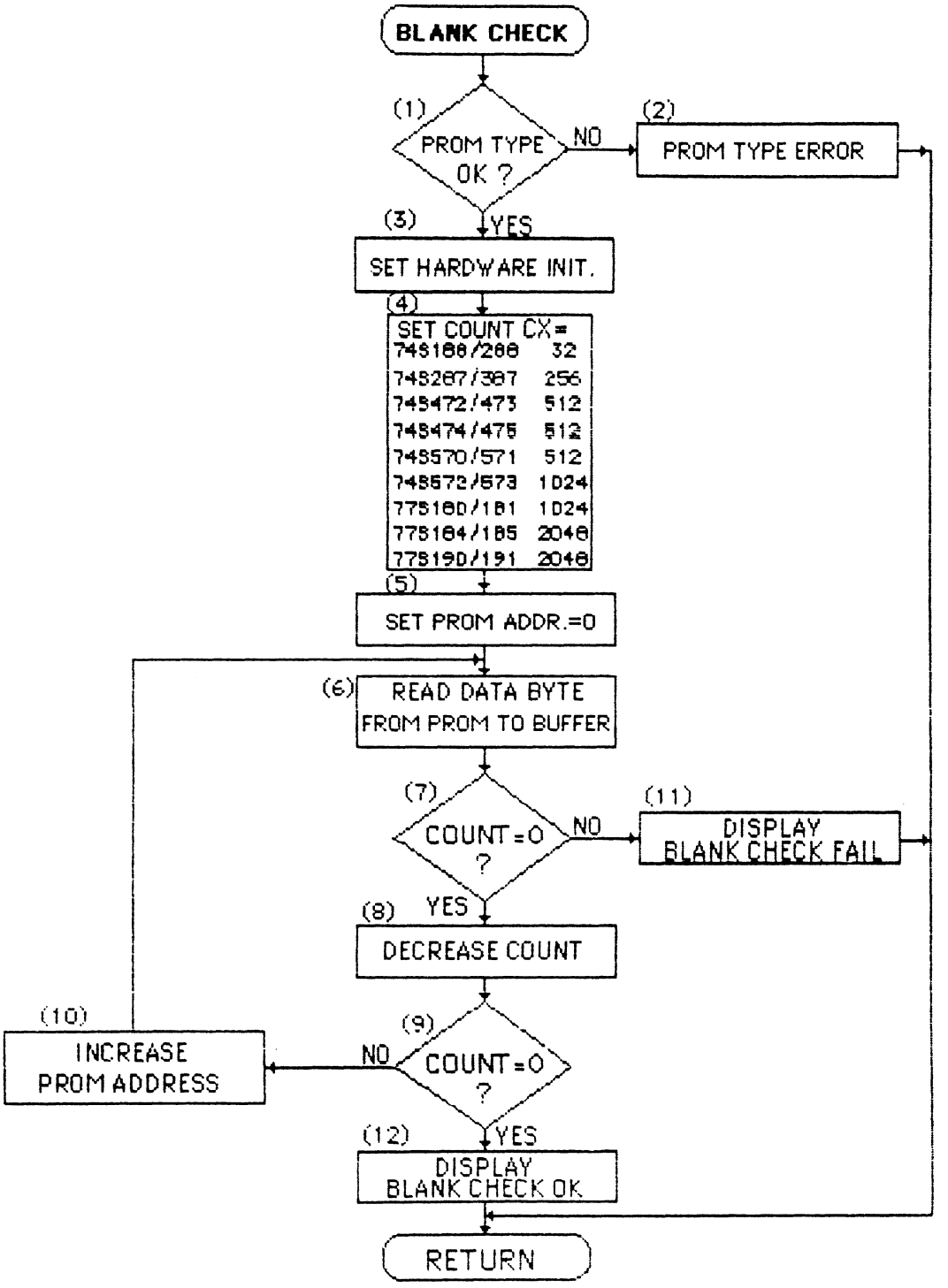


Figure 4-3 Flowchart for the BLANK CHECK function

```
_blank1         proc    near
        ;------ set hardware initialaization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V

                mov     al,0ffh
                mov     dx,dctl1        ; set all ones to turn off 7407's
                out     dx,al           ; U8 set and sets up to read data from PROM

                mov     al,0eh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_1

                mov     cx,32           ; set byte count (initize NO. of loops)
                mov     bl,0            ; set PROM address = 0, OE (output enable) low
loopb1 :
                mov     al,bl
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address (A0 - A4) to PROM
                inc     bl              ; increase PROM's address
                mov     dx,cctl1
                in      al,dx           ; U9 set and read data from PROM
                cmp     al,0            ; compare PROM data with 0
                jnz     non_blank1      ; jump if not equal to zero
                loop    loopb1          ; decrease CX, loop if nonzero

                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,1            ; blank check success (default, 1 : success)
                ret                     ; return to calling routine to display
                                        ; "blank check OK" (terminated)
non_blank1 :
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,0            ; blank check fail (default, 0: fail)
                ret                     ; return to calling routine (C language part)
                                        ; to display result
```

Figure 4-4 Program for BLANK CHECK subroutine (Type_1)

(the count register) to check all addresses in the device. For example, type_1 has a memory size equal to 32 bytes. The loop of steps 6 through 10 is the lable *looph* in this program. After one loop CX is decreased by one automatically.

4.4 Subroutine for the PROGRAMMING Function

Programming of a PROM is undertaken when the need exists to store information (a fixed program) that will be used over and over without changes. In programming a PROM, the desired program is first stored in the IBM PC-XT's buffer memory and then transferred in the programming operation to the PROM.

The software required to program the PROM is discussed here. The system sets the PROM into the program mode. The desired address word is issued from the IBM PC-XT's data bus to the appropriate PROM's address inputs. The PROM Vcc input is set to +10.5V ±0.5V. Data from the PC-XT's data bus is applied to the PROM's output lines. The program voltage is then applied to the specific PROM output pin to be programmed. The system pulses this output pin on the PROM and repeats the programming pulse at each successive bit to be altered. The address is then changed and the whole process repeated.

After programming each bit, the system does a compare operation. The compare operation matches the data in the PROM just programmed against the original data stored in the IBM PC-XT's buffer memory. The flowchart for this subroutine is given in Figure 4-5.

There we see that the first two events to occur (steps 1 and 2 in the flowchart) are the same as in the Blank Check function. In this function there are three optional parameters a1, a2, and a3, so the system needs to check if the parameters exist or not (step 3).

These optional parameters are defined as:

a1: is a hexadecimal address in the PC-XT's buffer memory specifying the start of the data range to be programmed.

a2: is a hexadecimal address in the PC-XT's buffer memory indicating the last location of the data range to be programmed.

a3: is the beginning hexadecimal address in PROM memory where the specified data will be programmed.

If these parameters do not exist, the system then executes steps 4 and 5. Otherwise the system goes through to check the relationship between a1, a2, and a3 (step 6). This is done to insure that the user inputs the correct memory address range. For the IBM PC-XT's buffer memory, the address range is from 0000H-7FFFH. The address range of the PROM depends on each PROM's memory size.

If any out of range parameter is input, the system then jumps to the main program. For example, the maximum value of parameter a3 is equal to the particular PROM's memory size. It is impossible to program data outside of the PROM's memory range. If there are no parameters, or if the input parameters are in the correct range, the hardware is initialized (step 12). The program segment for hardware initialization is the same as in BLANK CHECK.

As mentioned in Chapter 2, National Semiconductor's PROMs are manufactured with all 0's and are programmed bit by bit. The system reads a data byte from the PC-XT's buffer (step 13) and rotates it one bit to the right (step 15). If the data bit stored in the PC-XT's buffer is a " 0 ", then the system jumps to step 27 to program the next bit (step 16). Recall the programming steps f and g in Chapter 2. The 10 initial tries and 5 additional pulses are set up in step 17.

The system sets up the PROM in the programming mode (step 19). We recall that this is accomplished by forcing the Vcc input pin of the PROM to +10.5 volts and applying +10.5V to the PROM output (Qi) terminal. After one bit is programmed, the system checks whether that bit has been changed in the PROM. This is shown in step 24. If the bit has not been changed, the system continues back to execute a loop of steps 18 through 24 and 32 - 33. This loop is repeated until the data bit compares o.k.. If it does not, the loop continues for a maximium of 10 tries. If the data still is not o.k., this programming attempt failed. If the bit has been changed and does compare o.k., the system still executes a loop of steps 18 through 26. This is done to have 5 additional pulses of programming for that bit.

After each byte is completed, the system goes to the next step (step 28). If there are more data bytes, then the system executes a loop of steps 13 through 31 to program the next byte in the PROM. The addresses are incremented until the data is programmed for the whole specified memory range.

If the data now in the PROM compares at each address location with the original data in the PC-XT's buffer, then the burn-in was successful. If

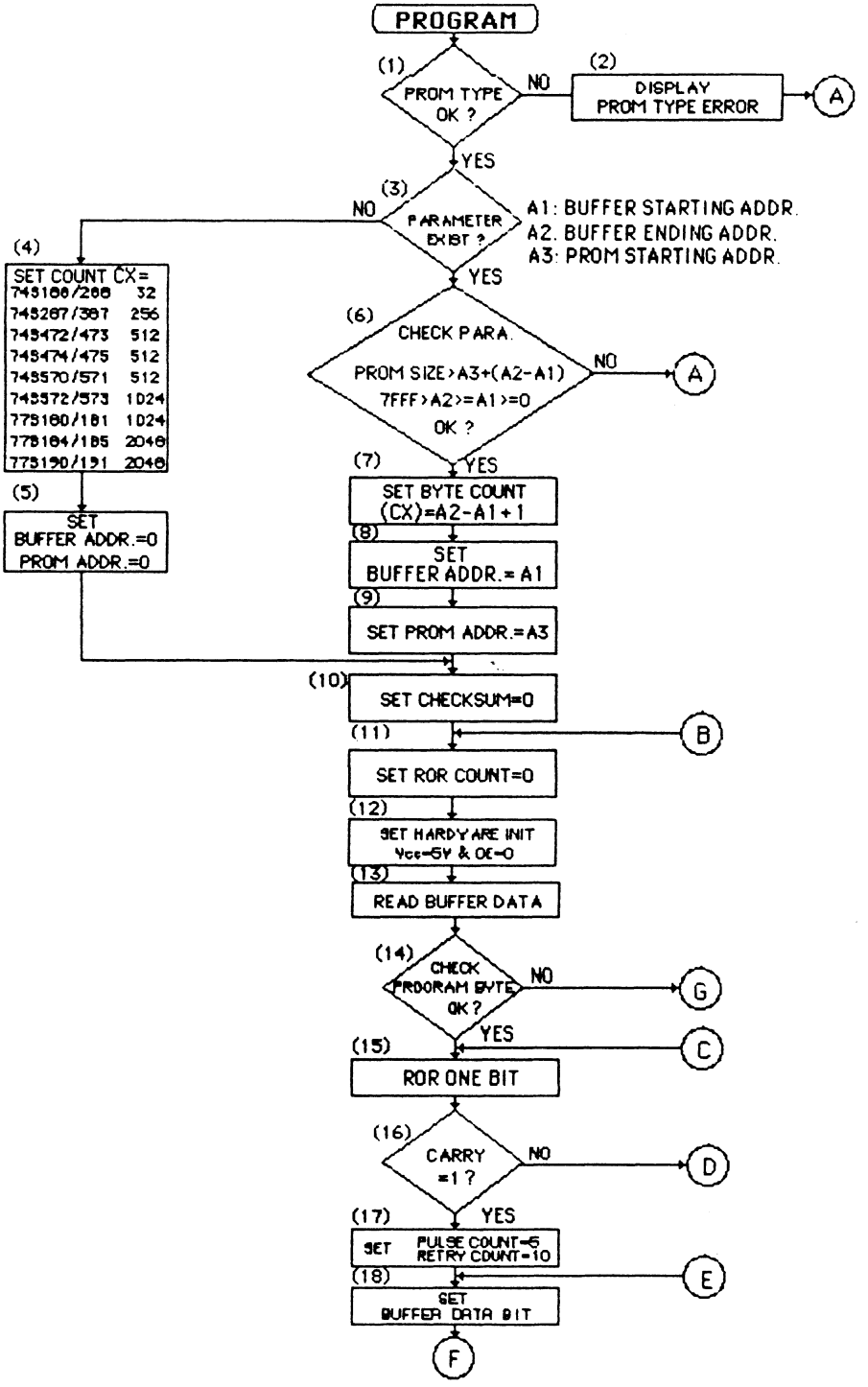


Figure 4-5 Flowchart for the PROGRAMMING function

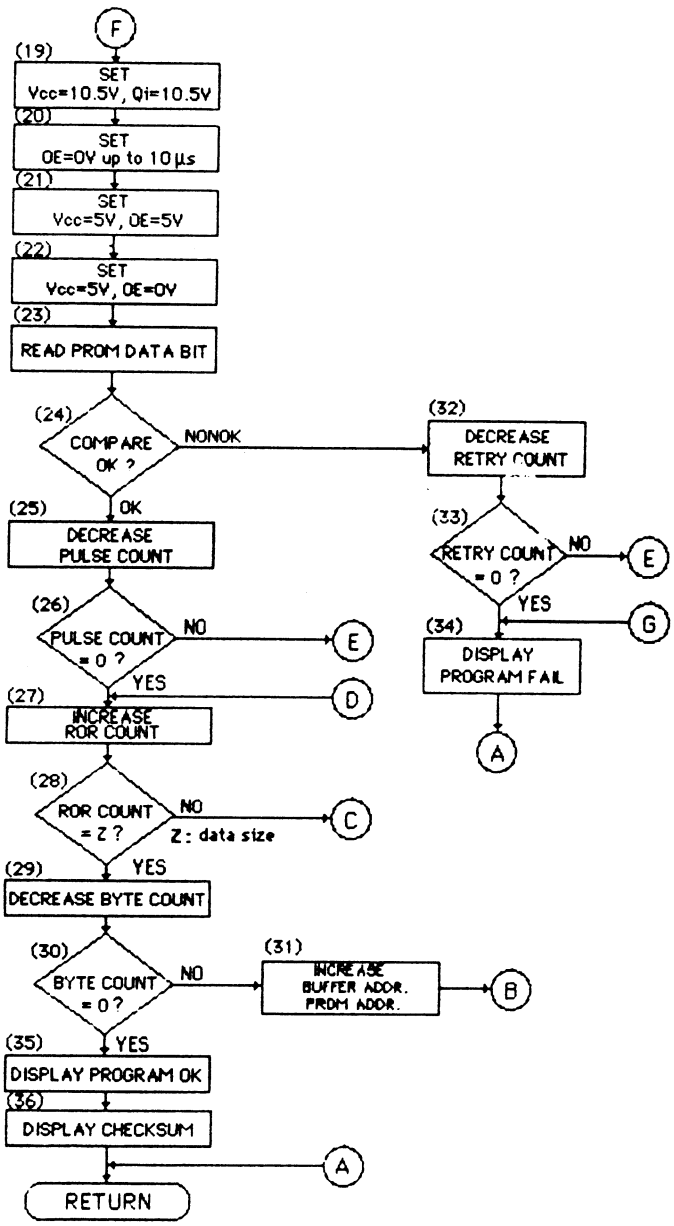


Figure 4-5 (continue)

```
_program1       proc    near
        ;------ set hardware initialization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V

                mov     al,0ffh
                mov     dx,dctl1        ; set all ones to turn off 7407's
                out     dx,al           ; U8 set and sets up to read data from PROM

                mov     al,0eh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_1

                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                push    si              ; push SI onto stack
                mov     bx,[bp+4]       ; get prom starting address & OE low pattern
                mov     _dspadr,bx      ; store current address for display
                mov     cx,[bp+6]       ; get byte count (range to be programmed)
                mov     di,[bp+8]       ; get starting address of buffer memory
                mov     buf_ptr,di      ; store buffer memory pointer
                mov     chksum,0        ; clear checksum
                mov     _verifyok,1     ; assume verify ok ! (default, 1 : success)
proceed1 :
                mov     di,buf_ptr      ; initize DI
                mov     al,_buffer[di]  ; request program byte
                mov     prog_byte,al    ; store program byte
                mov     al,bl
                and     al,11011111b    ; get PROM output enable (OE) low
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address (A0 - A4) to PROM
                mov     dx,cctl1
                in      al,dx           ; U9 set and read data from PROM
                or      al,prog_byte
                cmp     al,prog_byte    ; get exact programming data byte
                jz      pc11            ; jump if the programming data byte ok
                mov     _verifyok,0     ; PROM data error (default, 0: fail)
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
```

Figure 4-6 Program for PROGRAMMING function (type_1)

```
pc11:
                push    bx              ; push BX (general register) onto stack
                push    cx              ; pusg CX (count register) onto stack
                call    _promdsp        ; display current address
                pop     cx              ; restore CX
                pop     bx              ; restore BX
                cmp     prog_byte,0     ; compare programming data byte with 0
                jnz     pc21            ; jump if the data byte not equal to 0
                jmp     nextbyte1       ; jump to program next data byte
pc21:
                mov     si,0            ; initial SI (store being programmed data bit)
rotate1:
                ror     prog_byte,1     ; rotate right one bit to obtain programming data bit
                jc      pbit1           ; jump if carry=1
                jmp     nextbit1        ; jump to program next data bit
pbit1:
                mov     retrycnt,10     ; set retry count=10 (NO. of loops)
                mov     verifybit,0     ; clear program verify flag
                mov     di,si
                add     di,di
                mov     di,ptype1[di]
                mov     al,[di]
                mov     v12_data,al
                mov     ax,[di+1]
                mov     v12_adr,ax      ; request DG201 switch data
retry1:

;---- <a> ----       disable PROM by setting OE high -----
                mov     al,bl
                or      al,00100000b    ; get PROM output enable high
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address to PROM
        ;-----         set program data bit -----
                mov     al ,program_tbl[si]
                mov     dx,dctl1
                out     dx,al           ; U8 set and issues data bit to be programmed
;----<b> ----   set PROM Vcc input +12 V -----
                mov     al,01000000b
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input =12V
;----<c> ----   set exact program data bit (Qi) +12 V -----
                mov     al,v12_data
                mov     dx,v12_adr
                out     dx,al           ; DG201 set to supply program voltage
```

Figure 4-6 (continue)

```
;----<d>----          set PROM OE enable ------
                mov     al,bl
                and     al,11011111b    ; get PROM output enables low
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address (A0 - A4) for PROM
;----<e>----          disable PROM by setting OE high -----
                mov     al,bl
                or      al,00100000b    ; get PROM output enables (OE) high
                mov     dx,actl1
                out     dx,al           ; U7 set
        ;-----          set program bit (Qi) +5 V -----
                mov     al,11111111b
                mov     dx,v12_adr
                out     dx,al           ; disable DG201 to remove +12V from PROM output
        ;-----          set PROM Vcc input +5 V -----
                mov     al,00000000b
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input =5V
        ;-----          set up to read PROM ------
                mov     al,0ffh
                mov     dx,dctl1        ; set all ones to turn off 7407
                out     dx,al           ; U8 set and sets up to read data from PROM
        ;-----          set PROM OE enable ------
                mov     al,bl
                and     al,11011111b    ; get PROM output enables low
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address (A0 - A4) for PROM
        ;-----          read data from PROM-----
                mov     dx,cctl1
                in      al,dx           ; U9 set and read data from PROM
                cmp     verifybit,0     ; compare PROM data with 0
                jz      no_verify1      ; jump if the data bit is equal to 0 (not verified)
                dec     verifycnt       ; decrease verify count (initial value=5)
                jz      nextbit1        ; jump if  the verify count is equal to 0
                                        ; and to program next bit
                jmp     retry1          ; loop for 10 times
no_verify1:                             ; loop for retry count
                test    al,program_tbl[si] ; get program data bit which will be programmed
                jnz     vfbitok1        ; jump if not equal 0 (verify data bit ok)
                dec     retrycnt        ; decrease retry count (initial value=10)
                jz      vfbitfail1      ; jump if retry count is equal to 0
                jmp     retry1          ; jump to reprogram the data bit
vfbitok1:
                mov     verifycnt,5     ; set additional 5 pulse for programming
                mov     verifybit,1     ; bit verify ok (default, 1 : success)
                jmp     retry1          ; jump to reprogram the data bit
```

Figure 4-6 (continue)

```
vfbitfail1 :
                mov     _verifyok,0     ; verify fail (programming fail)
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs

                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
nextbit1 :
                inc     si              ; increase programmed data bit count
                cmp     si,8            ; compare the programming data bit with 8
                jz      nextbyte1       ; jump if the data byte (8 bits) has been programmed
                jmp     rotate1         ; jump to program next data bit
nextbyte1 :
                mov     ah,0            ; set AH=0
                mov     al,prog_byte    ; fetch program byte
                add     chksum,ax       ; add checksum value
                inc     bx              ; incerase PROM address
                mov     _dspadr,bx      ; store display address
                inc     buf_ptr         ; increase buffer memory address
                dec     cx              ; decrease byte count which is to be programmed
                jz      end1            ; jump if byte count is equal to 0
                jmp     proceed1        ; jump to program next byte
end1 :
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value

                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
```

Figure 4-6 (continue)

the data at any address of the PROM does not compare correctly with that of the PC-XT's buffer memory, then the system displays "Program PROM Fail" and jumps to the main calling program.

The program segment required to realize the PROGRAMMING function for type_1 is given in Figure 4- 6. The beginning of this program is at step 12 in the flowchart (hardware initialization). The PROM starting address is set in BX (the base register). When this programming routine is activated, the first four bytes of stack are used by BP (the base pointer) and IP (the instruction pointer), so the parameters (a1, a2, a3) are stored successively from the address [bp+4]. Lables <a> through <e> in the program segment correspond to the programming steps (a) through (e) which are discussed in section 2.2.

4.5 Subroutine for the VERIFY Function

The next function we shall discuss is VERIFY. The verify operation matches the data that is in the PROM against the original data stored in the PC-XT's buffer memory. If any data bit fails to compare, the system will display "Verify PROM Fail" and the checksum value on the screen. This function is especially useful if the user is unsure if the PROM has been programmed correctly.

The flowchart for the verify function is given in Figure 4-7. There steps 1 through 9 in the flowchart are the same as in the PROGRAMMING function's flowchart. In verifying the PROM the hardware initialization is set (step 10). The system reads data from the PROM memory (step 12) and calculates the checksum value (step 13).

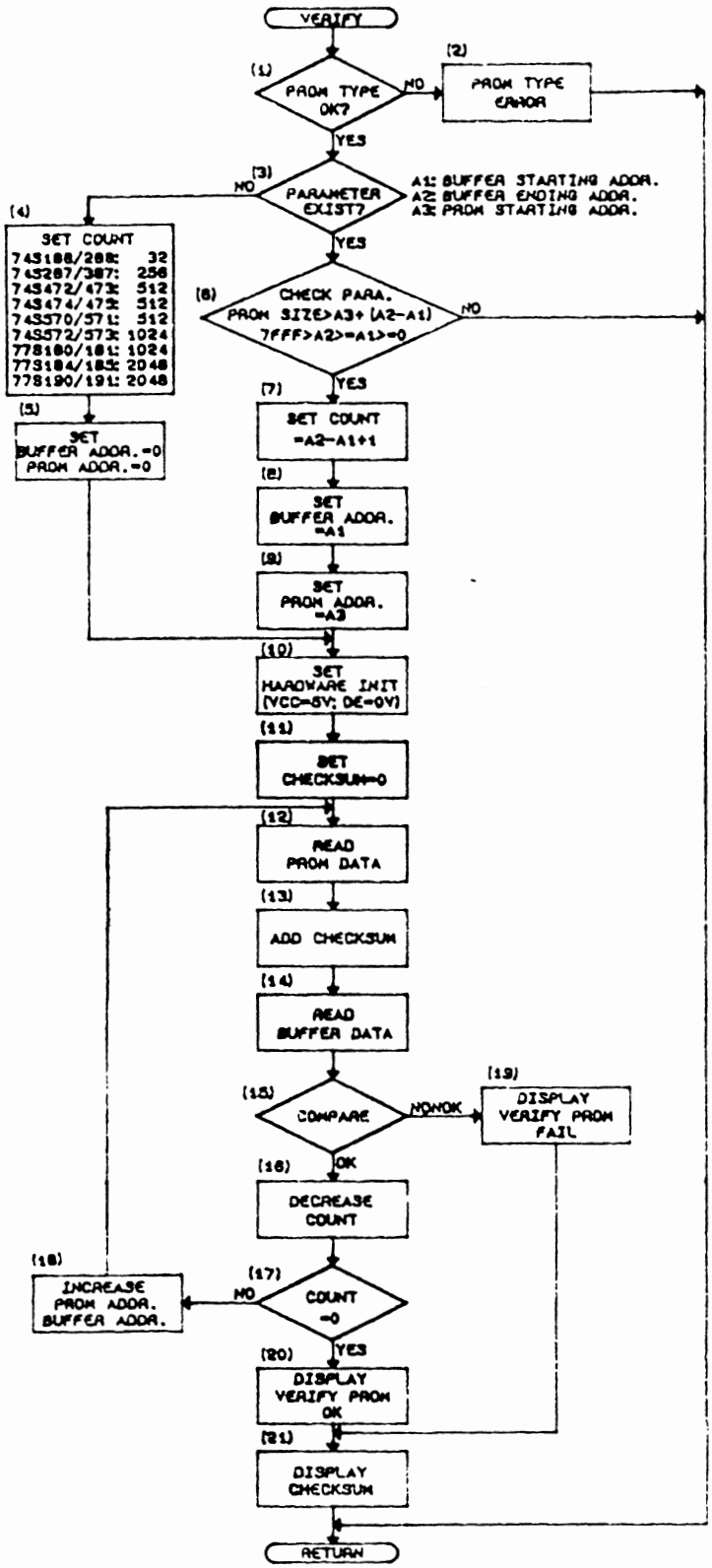


Figure 4-7 Flowchart for the VERIFY function

```
_verify1        proc    near
        ;------ set hardware initialization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V

                mov     al,0ffh
                mov     dx,dctl1        ; set all ones to turn off 7407
                out     dx,al           ; U8 set and sets up to read data from PROM

                mov     al,0eh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_1

                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                mov     bx,[bp+4]       ; get PROM OE low pattern & starting address
                mov     cx,[bp+6]       ; get byte count (range to be verified)
                mov     di,[bp+8]       ; get starting address of buffer memory
                                        ;these variables are transfered from C language part
                mov     ah,0            ; set AH=0
                mov     chksum,0        ; clear checksum
                mov     _verifyok,1     ; assume verify ok ! ( default, 1 : success)
loopv1 :
                mov     al,bl
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address (A0 - A4) to PROM
                inc     bl              ; increase PROM address
                mov     dx,cctl1
                in      al,dx           ; U9 set and read data from PROM
                cmp     _buffer[di],al  ; compare buffer memory's data with PROM's data

                jz      vok1            ; jump if data is the same
                mov     _verifyok,0     ; verify fail if data is not the same
vok1 :
                inc     di              ; increase buffer memory address
                add     chksum,ax       ; add checksum
                loop    loopv1          ; decrease CX, loop if nonzero

                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value

                pop     di              ; restore DI
                pop     bp              ; restore BP
                ret                     ; return to calling routine (C language part)
```

Figure 4-8 Program for VERIFY function (type_1)

In step 15 the system compares the PC-XT's buffer data at the proper location address to the PROM data. The addresses are incremented and the data is compared at each address. This is accomplished by a loop of steps 12 through 18. If the data is different, the system displays "Verify PROM Fail" and the checksum value, and jumps to the main program (steps 19 and 21).

If the data is the same, then the system increments to the next address. This action is repeated until any data fails to match or the end of the memory range is reached. If the data at all addresses are the same, the system then displays "Verify PROM OK " and the checksum, and returns to the main program, as shown in steps 20 and 21.

The program segment required for the VERIFY function for type_1 is given in Figure 4-8. The beginning of this program is at step 10 (set hardware initialization) in the flowchart. The loop of steps 12 through 18 is the lable *loapv1* in this program segment.

4.6 Subroutine for the READ Function

The final function we will discuss is READ, where data read from the PROM is stored into the PC-XT's buffer memory. The read function is useful if the user wishes to know what contents exist in the PROM. The flowchart for this function is given in Figure 4-9. As before, the first several events (steps 1 through 11 in the flowchart) are the same as the VERIFY function's subroutine.

However, in this function the optional parameters a1, a2, and a3 have a totally different meaning than before. These are defined as:

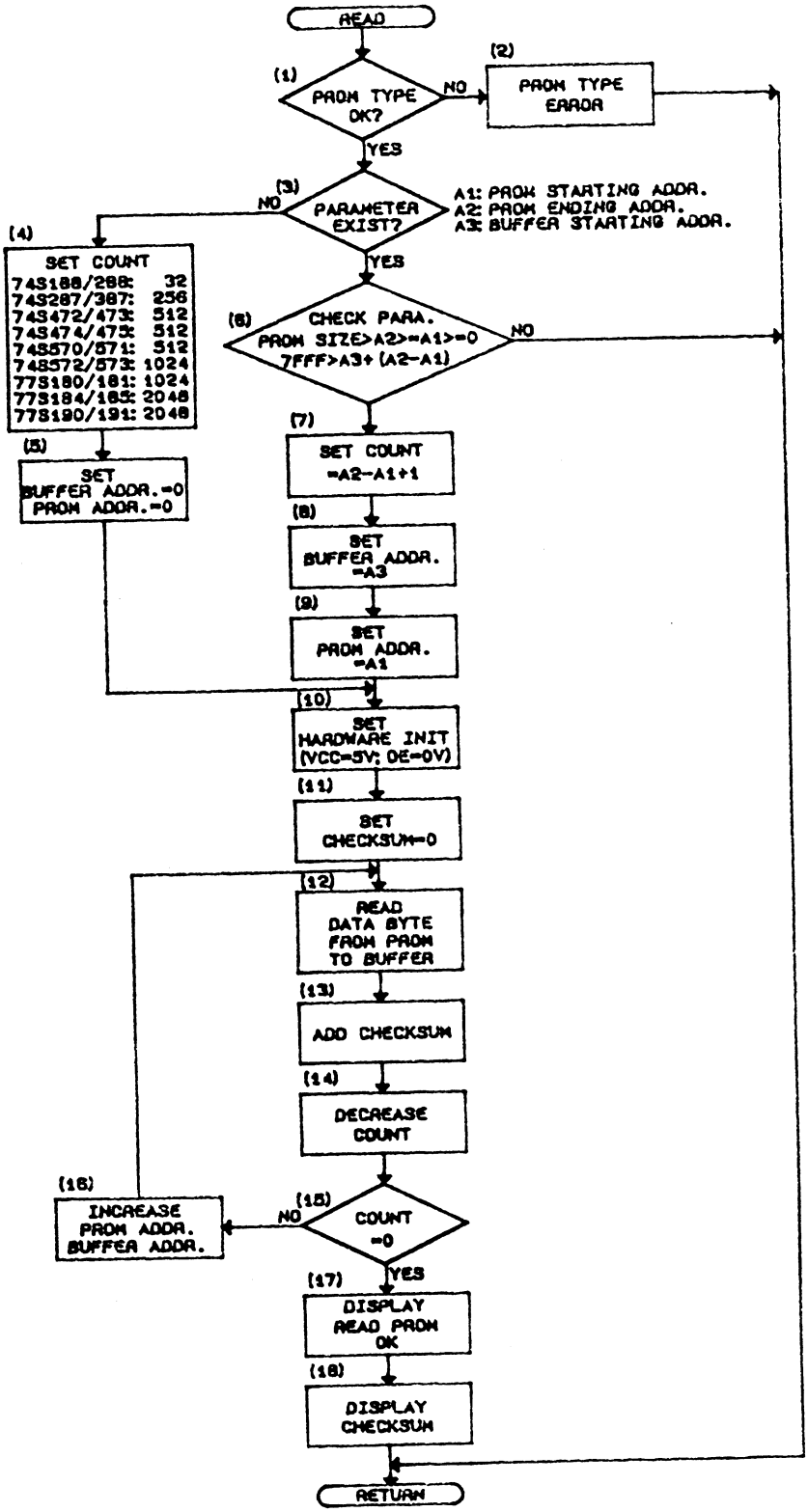


Figure 4-9 Flowchart for the READ function

```
_read1          proc    near
        ;------ set hardware initization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V

                mov     al,0ffh
                mov     dx,dctl1        ; set all ones to turn off 7407
                out     dx,al           ; U8 set and sets up to read data from PROM

                mov     al,0eh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_1

                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                mov     bx,[bp+4]       ; get PROM CE low pattern & starting address
                mov     cx,[bp+6]       ; get byte count (range to be read)
                mov     di,[bp+8]       ; get starting address of buffer memory
                                        ;these variables are transfered from C language part
                mov     ah,0            ; set AH=0
                mov     chksum,0        ; clear checksum
loopr1:
                mov     al,bl
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address (A0 - A4) to PROM
                inc     bl              ; increase PROM address
                mov     dx,cctl1
                in      al,dx           ; U9 set and read data from PROM
                mov     _buffer[di],al  ; store data to buffer memory
                inc     di              ; increase buffer memory address
                add     chksum,ax       ; add checksum
                loop    loopr1          ; decrease CX, loop if nonzero

                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value

                pop     di              ; restore DI
                pop     bp              ; restore BP
                ret                     ; return to calling routine (terminated)
```

Figure 4-10 Program for READ function (type_1)

a1: is a hexadecimal address in PROM memory specifying the start of the data range to be read.

a2: is a hexadecimal address in PROM memory indicating the last location of the data range to be read.

a3: is the beginning hexadecimal address in the IBM PC-XT's buffer memory where the specified data will be written.

The system reads a data byte from the PROM memory and writes that data into the PC-XT's buffer (step 12). The addresses are incremented to read the next data byte (step 16). This action is repeated for all address locations until the end of the PROM's memory or until a2 in the PROM is read. For example, the 74S188/74S288 memory range is 00H - 1FH.

The program segment required to realize the READ function for type_1 is given in Figure 4- 10. The beginning of this program is at step 10 (set hardware initialization) in the flowchart. The loop of steps 12 through 16 is the label *loopr1* in this program.

4.7 Summary

In this chapter the software written in assembly language for the programmer card has been discussed. Some of the subroutines- those written in turbo C language and related to the internal operation of a personal computer are not mentioned. The flowcharts for these commands will be in appendix B. An understanding of the discussions presented in this chapter gives an idea of how the system software works. Finally, an entire program listing is given in Appendix A. The detailed command description (manual) is in Appendix C.

CHAPTER 5

Conclusions

In this thesis a PROM programmer was built for the IBM PC/XT/AT, or compatible system, although the fundamental principles and algorithms apply to many other personal computer systems. The software of the system described in this work was written using assembly and turbo C languages. The C language is widely used for real-time interfacing applications. The system hardware was assembled and built by using wire-wrapping methods. Thirty-five chips are used. All components required are listed in Appendix I.

This design is built on an extension card for the PC-XT's I/O slot, so the IBM PC can be used to control the system and can manipulate the data on the screen before burn-in to the PROM. Data can be electrically programmed, as desired, at any bit location with the programmer and it can easily download or upload a file. The software, however, requires that the DIP switch (U33) be ON at pin 5 before it can be used. If different port addresses were desired, the DIP switch would be set differently. An advantage of using the IBM PC-XT is that the card does not need its own CPU and power supply.

This system can be further developed to make it easier to operate. The programmer card provides on-board and off-board expansion capability. The PROM IC can be plugged into the vacant socket on the card. If that is not convenient, off board expansion is readily accomplished since all address,

data, and control bus lines can be brought out to an expansion IC socket (a 24-pin textool). The disadvantage of using a plug-in card is that a different microcomputer may have a different CPU (For example, Apple II plus has the Motorola MC68000 CPU) and different specifications for its own I/O slot.

A problem in this circuit design is data buffering. To solve this problem, the 74LS245 (U32) bus transciver buffer was added to the system data bus to drive the data between the host computer and the programmer card. To meet the current requirement of the program voltage for PROM outputs, the 55450B (U40) was added to supply higher current to blow out the PROM's fuses.

Another problem was signal coupling through the power supply line. It was solved by using capacitors between the power supply and ground at the card edge connectors and at the extreme ends and tops of the card. It is particularly important to decouple the +5 volts power level since it will most likely draw the most power.

Final tests on this unit have been made. It was tested for two types of PROMs (type_1: 74S188/74S288 and type_3: 74S287) and the results were satisfactory. The functions supported by software were tested, including *read* data from PROM, *display* the content on the monitor and *manipulate data* before programming etc., The detailed command description is given in Appendix C.

There is always room for improvement. One direction to follow in future expansions is to build in the capability to program an EPROM (Erasable PROM). Figure 3-4 given in Chapter 3 contains two detailed four-port addresses which allow for this expansion. The programmer presented

in this work used only one 8255CS1' signal (ports FEF0H through FEF3H) for the PROM output (Qi) programming voltage switching. By using the other signal 8255ACS2' (ports FEF4H through FEF7H) to enable another Intel 8255A under software control the programmer can be expanded as an EPROM/PROM programmer. The only extra hardware needed is a power supply to support the desired programming voltages (up to +25V). The software could be supplemented to allow for EPROM programming.

Another interesting possibility is to match another manufacture's PROMs by means of a few hardware and software changes. To allow for the possibility of programming other PROM types (For example, AMD, Texas Instruments, and Harris's PROMs), Figure 3-8 for the PROM selector (U39) has some output pins not used. By using these blank pins to select another PROM type under software control the system may be expanded.

To summarize: The system presented here is easy to use and offers many useful capabilities:

(1) It can program up to a 2048 x 8 memory size.

(2) It can display PROM contents in ASCII characters representing numbers in binary or hexadecimal format.

(3) It can check for blank PROMs.

(4) By using computer buffer memory , it can manipulate data before loading it into the PROM (For example: Kill, Insert, or Modify).

(5) It can easily download a custom operating program to the host computer's buffer memory.

(6) It can easily upload a hexadecimal file into a diskette.

REFERENCE

1. Memory Databook by National Semiconductor Corp.
2. Bipolar Memory Generic PROM Series Data Book by Advanced Micro Devices Inc.,
3. Generic Programmable Read Only memories Data Book by Harris Inc.,
4. The Semiconductor Memory Data Book for Design Engineer by Texas Instruments Inc.,
5. Electronics with Digital and Analog Integrated Circuits by Richard J. Higgins 1983.
6. TTL Data Book by Texas Instruments Inc.,
7. Interfacing to the IBM Personal Computer by Lewis C. Eggebrecht.
8. Intergrated Circuits Applications Handbook by Arthur H. Seidman.
9. Micro Electronics Digital and Analog Circuits and Systems by Jacob Millman, Ph.D.
10. Microprocessor and Peripheral Handbook Volume II Peripheral by Intel Corporated Inc., 1988
11. Getting Started with 8080, 8085, Z80, and 6800 Microprocessor System by James W. Coffron 1984
12. Practical Interfacing Techniques for Microprocessor Systems by James W. Coffron 1983
13. The C Programming Language by Brain W. Kernighan & Dennis M. Richie.
14. Turbo C User's Guide IBM Version PC/XT/AT and true Compatibles
15. Turbo C Reference Guide IBM Version PC/XT/AT and true Compatibles

16. Surfire Programming C by Warren A. Stewart 1985

17. Assembler for the IBM PC and PC-XT by Peter Abel

18. IBM PC/8088 Assembly Language Programming by Avtar Singh, Aderson Jacobson,Inc., Walter A. Triebel Intel Corporation.

APPENDIX A

```
#include <ctype.h>
#include <string.h>
#include <stdio.h>
#include <fcntl.h>

#define END_BUFFER 32767
#define STR_LENGTH 79
/* ------------------------------------------------------------------------
        public variables for assembly routine use                   */

        int     verifyok;       /* 1 --> OK; 0 --> FAIL */
        int     dspadr;         /* address when in programming */
/* ------------------------------------------------------------------------
        public procedure for assembly routine use                   */

        void    promdsp ();
/* ------------------------------------------------------------------------ */

main ()
{
int     blankchk ( char *string, int str_ptr );
int     cal_num ( char *string, int *str_ptr, unsigned int *a, int num );
unsigned int chksumx ( unsigned int start_adr, unsigned int end_adr, unsigned int store_adr, int f,
                       unsigned char *buf_ptr );
unsigned int chksumt ( unsigned int start_adr, unsigned int end_adr, unsigned int store_adr, int f,
                       unsigned char *buf_ptr );
void    copy ( unsigned int start_adr, unsigned int end_adr, unsigned int des_adr, unsigned char
              *buf_ptr );
void    display ( unsigned int start_adr, unsigned int end_adr, unsigned char *buf_ptr );
void    downloadbin ( FILE *fname, char *buf_ptr );
int     downloadhex ( FILE *fname, char *buf_ptr );
void    fill ( unsigned int start_adr, unsigned int end_adr, unsigned int d, unsigned char *buf_ptr );
void    get_str ( int *str_ptr, char *in_buf );
int     grychk ( unsigned int *a1, unsigned int *a2, unsigned int *a3, unsigned int b1, unsigned int b2,
                 char *string, int k );
void    help ();
void    insert ( unsigned int start_adr, unsigned char *buf_ptr );
void    kill ( unsigned int start_adr, unsigned int end_adr, unsigned char *buf_ptr );
void    modify ( unsigned int start_adr, unsigned char *buf_ptr );
int     programprom ( char *string, int str_ptr, int *chksum );
void    promhelp ();
int     readprom ( char *string, int str_ptr, int *chksum );
void    upload (FILE *fname,unsigned int start_adr,unsigned int end_adr, unsigned char *buf_ptr, int k );
int     verifyprom ( char *string, int str_ptr, int *chksum );
void    extern  init ();        /* hardware initialization routine */
void    extern  cls ();         /* clear screen routine */

        FILE    *hex;
        char extern buffer;         /* buffer memory */
        char c;
        char in_buf[ STR_LENGTH ];          /* keyboard input buffer */
        unsigned char *index;
        unsigned int  a1,a2,a3;
        int i,j,k,sys_flag;
```

```
init ();  /* hardware initialization */
cls ();
printf ( "\n      ** NS-PROM WRITER **\n\n% " );
index = &buffer;          /* buffer memory starting address */
while ( 1 )
{
        get_str ( &i, in_buf );  /* get keyboard input string */
        sys_flag = 0;     /* assume system flag = 0 */
        for ( j = 0; j < i; j ++ )
        {
                c = in_buf[ j ];
                switch ( c )
                {
                        case 'B' :
                                k = j + 1;
                                while ( in_buf[ k ] == ' ' && k < i )
                                        k ++;
                                if ( k == i )
                                {
                                        sys_flag = 1;
                                        break;
                                }
                                j = k + 6;
                                while ( in_buf[ j ] == ' ' && j < i )
                                        j ++;
                                if ( j != i )
                                {
                                        printf ( "\n\n    PROM TYPE ERROR !\n" );
                                        sys_flag = 3;    /* command cancelled */
                                        break;
                                }
                                j = blankchk ( in_buf, k );
                                if ( j == -1 )
                                {
                                        printf ( "\n\n    PROM TYPE ERROR !\n" );
                                        sys_flag = 3;    /* command cancelled */
                                        break;
                                }
                                if ( j == 0 )
                                {
                                        printf ( "\n\n    BLANK CHECK FAIL !\n" );
                                        sys_flag = 3;    /* command terminated */
                                        break;
                                }
                                if ( j == 1 )
                                {
                                        printf ( "\n\n    BLANK CHECK OK !\n" );
                                        sys_flag = 3;    /* command success */
                                        break;
                                }
                        case 'C':
                                j += 1;
                                if ( j < i && in_buf[ j ] == 'T' )
```

```
                        /*checksum command(total)*/
{
        j += 1;
        if ( j >= i || cal_num ( in_buf, &j, &a1 , 4 ) == 0 )
        {
                sys_flag = 2;     /* address error */
                break;
        }
        if ( j >= i || cal_num ( in_buf, &j, &a2, 4 ) == 0 )
        {
                sys_flag = 2;     /* address error */
                break;
        }
        if ( a1 <= a2 )
        {
                k = j;
                while ( in_buf[ k ] == ' ' && k < i )
                        k ++;
                if ( k == i )
                {
                        printf ( "\n\n     THE CHECKSUM
                                IS : " );
                        printf ( "%04X ( TOTAL )",
                        chksumt ( a1 , a2, 0, 0, index ) );
                        sys_flag = 3;
                                /* processing success */
                        break;
                }
                if ( j < i && cal_num ( in_buf, &j, &a3, 4 )
                                                == 1 )
                {
                        if ( a3 == END_BUFFER )
                        {
                        sys_flag = 2; /* address error */
                                break;
                        }
                        k = j;
                        while ( in_buf[ k ] == ' ' && k < i )
                                k ++;
                        if ( k != i )
                        {
                                sys_flag = 1;
                                break;
                        }
                        printf ( "\n\n     THE CHECKSUM
                                                IS : " );
                        printf ( "%04X ( TOTAL )",
                        chksumt ( a1 , a2, a3, 1 , index ) );
                        sys_flag = 3
                                /*processing success */
                }
                else
                        sys_flag = 2; /* address error */
}
```

```
        else
                sys_flag = 2;    /* address error */
        break;
}
if ( j < i && in_buf[ j ] == 'X' )
                        /* checksum command ( exor ) */
{
        j += 1;
        if ( j >= i || cal_num ( in_buf, &j, &a1, 4 ) == 0 )
        {
                sys_flag = 2;    /* address error */
                break;
        }
        if ( j >= i || cal_num ( in_buf, &j, &a2, 4 ) == 0 )
        {
                sys_flag = 2;    /* address error */
                break;
        }
        if ( a1 <= a2 )
        {
                k = j;
                while ( in_buf[ k ] == ' ' && k < i )
                        k ++;
                if ( k == i )
                {
                        printf ( "\n\n    THE CHECKSUM
                                                IS : " );
                        printf ( "%02X ( EXOR )", chksumx
                                ( a1 , a2, 0, 0, index ) );
                        sys_flag = 3;
                                /* processing success */
                        break;
                }
                if ( j < i && cal_num ( in_buf, &j, &a3, 4 )
                                                == 1 )
                {
                        k = j;
                        while ( in_buf[ k ] == ' ' && k < i )
                                k ++;
                        if ( k != i )
                        {
                                sys_flag = 1;
                                break;
                        }
                        printf ( "\n\n    THE CHECKSUM
                                                IS : " );
                        printf ( "%02X ( EXOR )", chksumx
                                ( a1 , a2, a3, 1, index ) );
                        sys_flag = 3;
                                /* processing success */
                }
                else
                        sys_flag = 2; /* address error */
        }
```

```
                else
                        sys_flag = 2;    /* address error */
                break;
        }
        sys_flag = 1;            /* syntax error */
        break;
case 'D' :
        j += 1;
        if ( j >=i || cal_num ( in_buf, &j, &a1 , 4 ) == 0 )
        {
                sys_flag = 2;    /* address error */
                break;
        }
        if ( j >= i || cal_num ( in_buf, &j, &a2, 4 ) == 0 )
        {
                sys_flag = 2;    /* address error */
                break;
        }
        if ( a1 <= a2 )
        {
                k = j;
                while ( in_buf[ k ] == ' ' && k < i )
                        k ++;
                if ( k != i )
                {
                        sys_flag = 1;
                        break;
                }
                display ( a1 , a2, index );
                sys_flag = 3;            /* processing success */
        }
        else
                sys_flag = 2;    /* address error */
        break;
case 'F' :
        j += 1;
        if ( j >= i || cal_num ( in_buf, &j, &a1 , 4 ) == 0 )
        {
                sys_flag = 2;    /* address error */
                break;
        }
        if ( j >= i || cal_num ( in_buf, &j, &a2, 4 ) == 0 )
        {
                sys_flag = 2;    /* address error */
                break;
        }
        if ( a1 <= a2 )
        {
                if ( j < i && cal_num ( in_buf, &j, &a3, 2 ) == 1 )
                {
                        k = j;
                        while ( in_buf[ k ] == ' ' && k < i )
                                k ++;
                        if ( k != i )
```

```
                        {
                                sys_flag = 1;
                                break;
                        }
                        fill ( a1, a2, a3, index );
                        printf ( "\n\n    FILL DATA COMPLETE !" );
                        sys_flag = 3;    /* processing success */
                }
                else
                {
                        printf ( "\n    Fill data Error ! \n% " );
                        sys_flag = 3;    /* command cancelled */
                }
        }
        else
                sys_flag = 2;    /* address error */
        break;
case 'G' :
        k = j + 1;
        while ( in_buf[ k ] == ' ' && k < i )
                k ++;
        if ( k == i )
        {
                sys_flag = 1;
                break;
        }
        j = programprom ( in_buf, k, &i );
        if ( j == -1 )
        {
                printf ( "\n\n    PROM TYPE ERROR !\n" );
                sys_flag = 3;    /* command cancelled */
                break;
        }
        if ( j == 0 )
        {
                sys_flag = 2;    /* command cancelled */
                break;
        }
        if ( j == 1 )
        {
                if ( verifyok )
                {
                        printf ( "\n\n    PROM PROGRAM OK !" );
                        printf ( "\n\n    THE CHECKSUM IS : %02X
                                        ( EXOR )", -i & 0x00ff );
                        printf ( "\n    THE CHECKSUM IS : %04X (
                                                TOTAL )\n", i );
                }
                else
                        printf ( "\n\n    PROM PROGRAM FAIL !" );
                sys_flag = 3;    /* command success */
                break;
        }
case 'I' :
```

```
                j += 1;
                if ( j >= i || cal_num ( in_buf, &j, &a1, 4 ) == 0 )
                {
                        sys_flag = 2;     /* address error */
                        break;
                }
                k = j;
                while ( in_buf[ k ] == ' ' && k < i )
                        k ++;
                if ( k != i )
                {
                        sys_flag = 1;
                        break;
                }
                insert ( a1, index );
                sys_flag = 3;             /* processing success */
                break;
        case 'K' :
                j += 1;
                if ( j >= i || cal_num ( in_buf, &j, &a1, 4 ) == 0 )
                {
                        sys_flag = 2;     /* address error */
                        break;
                }
                if ( j >= i || cal_num ( in_buf, &j, &a2, 4 ) == 0 )
                {
                        sys_flag = 2;     /* address error */
                        break;
                }
                if ( a1 <= a2 )
                {
                        k = j;
                        while ( in_buf[ k ] == ' ' && k < i )
                                k ++;
                        if ( k != i )
                        {
                                sys_flag = 1;
                                break;
                        }
                        kill ( a1, a2, index );
                        printf ( "\n\n    DELETE DATA COMPLETE !" );
                        sys_flag = 3;             /* processing success */
                }
                else
                        sys_flag = 2;     /* address error */
                break;
        case 'M' :
                j += 1;
                if ( j >= i || cal_num ( in_buf, &j, &a1, 4 ) == 0 )
                {
                        sys_flag = 2;     /* address error */
                        break;
                }
                k = j;
```

```
        while ( in_buf[ k ] == ' ' && k < i )
                k ++;
        if ( k != i )
        {
                sys_flag = 1;
                break;
        }
        modify ( a1 , index );
        sys_flag = 3;                /* processing success */
        break;
case 'Q' :
        k = j + 1;
        while ( in_buf[ k ] == ' ' && k < i )
                k ++;
        if ( k != i )
        {
                sys_flag = 1;
                break;
        }
        sys_flag = -1;
        printf ( "\n" );
        break;
case 'R' :
        k = j + 1;
        while ( in_buf[ k ] == ' ' && k < i )
                k ++;
        if ( k == i )
        {
                sys_flag = 1;
                break;
        }
        j = readprom ( in_buf, k, &i );
        if ( j == -1 )
        {
                printf ( "\n\n    PROM TYPE ERROR !\n" );
                sys_flag = 3;    /* command cancelled */
                break;
        }
        if ( j == 0 )
        {
                sys_flag = 2;    /* command cancelled */
                break;
        }
        if ( j == 1 )
        {
                printf ( "\n\n    READ PROM OK !" );
                printf ( "\n\n    THE CHECKSUM IS : %02X ( EXOR
                                )", -i & 0x00ff );
                printf ( "\n    THE CHECKSUM IS : %04X ( TOTAL
                                        )\n", i );
                sys_flag = 3;    /* command success */
                break;
        }
case 'T' :
```

```
        k = j + 1;
        while ( in_buf[ k ] == ' ' && k < i )
                k ++;
        if ( k != i )
        {
                sys_flag = 1;
                break;
        }
        promhelp ();
        sys_flag = 3;                   /* processing success */
        break;
case 'U' :
        k = j + 1;
        while ( in_buf[ k ] == ' ' && k < i )
                k ++;
        if ( k != i )
        {
                sys_flag = 1;
                break;
        }
        printf( "\n    UPLOAD HEX FILE FROM MEMORY, ENTER FILE
                                                NAME : " );
        get_str ( &j, in_buf );
        if ( j == 0 )
        {
                sys_flag = 3;                   /* command cancelled */
                break;
        }
        hex = fopen ( in_buf, "w" );
        if ( hex == 0 )
        {
                printf( "\n\n    CREATE FILE FAILURE !\x07" );
                sys_flag = 3;                   /* command cancelled */
                break;
        }
        printf( "\n\n    DISPLAY WHEN IN PROCESSING (Y/N) : " );
        while ( 1 )
        {
                c = getch () ;
                if ( c == 'n' || c == 'N' )
                {
                        putch ( c );
                        k = 0;                  /* indicate no display */
                        break;
                }
                if ( c == 'y' || c == 'Y' )
                {
                        putch ( c );
                        k = 1;   /* indicate display */
                        break;
                }
        }
        printf ( "\n\n    UPLOAD MEMORY RANGE : " );
        get_str ( &i, in_buf );
```

```
        if ( i == 0 )
        {
                printf ( "\n\n    MEMORY RANGE ERROR !" );
                sys_flag = 3;           /* command cancelled */
                break;
        }
        j = 0;
        if ( j < i && cal_num ( in_buf, &j, &a1, 4 ) == 1 )
                if ( j < i && cal_num ( in_buf, &j, &a2, 4 ) == 1 )
                        if ( a1 <= a2 )
                        {
                                while ( in_buf[ j ] == ' ' && j < i )
                                        j ++;
                                if ( j != i )
                                {
                                        sys_flag = 1;
                                        break;
                                }
                                upload ( hex, a1, a2, index, k );
                                if ( k == 1 )
                                        printf ( ":00000001FF" );
                                fprintf ( hex, ":00000001FF" );
                                fclose ( hex );
                                printf ( "\n\n    UPLOAD
                                        COMPLETE !" );
                                sys_flag = 3;
                                /* processing complete */
                                break;
                        }
        printf ( "\n\n    MEMORY RANGE ERROR !" );
        sys_flag = 3;           /* command cancelled */
        break;
case 'V' :
        k = j + 1;
        while ( in_buf[ k ] == ' ' && k < i )
                k ++;
        if ( k == i )
        {
                sys_flag = 1;
                break;
        }
        j = verifyprom ( in_buf, k, &i );
        if ( j == -1 )
        {
                printf ( "\n\n    PROM TYPE ERROR !\n" );
                sys_flag = 3;   /* command cancelled */
                break;
        }
        if ( j == 0 )
        {
                sys_flag = 2;   /* command cancelled */
                break;
        }
        if ( j == 1 )
```

```
        {
                if ( verifyok )
                        printf ( "\n\n    PROM VERIFY OK !" );
                else
                        printf ( "\n\n    PROM VERIFY FAIL !" );
                printf ( "\n\n    THE CHECKSUM IS : %02X ( EXOR
                                        )", -i & 0x00ff );
                printf ( "\n    THE CHECKSUM IS : %04X ( TOTAL
                                                )\n", i );
                sys_flag = 3;    /* command success */
                break;
        }
case 'W' :
        k = j + 1;
        while ( in_buf[ k ] == ' ' && k < i )
                k ++;
        if ( k != i )
        {
                sys_flag = 1;
                break;
        }
        printf( "\n    DOWNLOAD HEX FILE TO MEMORY, ENTER FILE
                                                NAME : " );
        get_str ( &j, in_buf );
        if ( j == 0 )
        {
                sys_flag = 3;            /* command cancelled */
                break;
        }
        hex = fopen ( in_buf, "r" );
        if ( hex == 0 )
        {
                printf ( "\n\n    OPEN FILE FAILURE !" );
                sys_flag = 3;            /* command cancelled */
                break;
        }
        if ( downloadhex ( hex, index ) )
                printf ( "\n\n    DOWNLOAD COMPLETE !" );
        else
                printf ( "\n\n    DOWNLOAD FILE ERROR !" );
        sys_flag = 3;            /* command cancelled */
        fclose ( hex );
        break;
case 'X' :
        j += 1;
        if ( j >= i || cal_num ( in_buf, &j, &a1, 4 ) == 0 )
        {
                sys_flag = 2;    /* address error */
                break;
        }
        if ( j >= i || cal_num ( in_buf, &j, &a2, 4 ) == 0 )
        {
                sys_flag = 2;    /* address error */
                break;
```

```
                }
                if ( a1 <= a2 )
                {
                        if ( j < i && cal_num ( in_buf, &j, &a3, 4 ) == 1 )
                        {
                                k = j;
                                while ( in_buf[ k ] == ' ' && k < i )
                                        k ++;
                                if ( k != i )
                                {
                                        sys_flag = 1;
                                        break;
                                }
                                if ( ( a2 - a1 ) <= ( END_BUFFER - a3 ) )
                                {
                                        copy ( a1, a2, a3, index );
                                        printf ( "\n\n     COPY DATA
                                                        COMPLETE !" );
                                        sys_flag = 3;
                                                /* processing success */
                                }
                                else
                                {
                                        printf ( "\n\n     DESTINATION
                                                RANGE TOO SMALL !" );
                                        sys_flag = 3;   /* range error */
                                }
                        }
                        else
                        {
                                sys_flag = 2;     /* address error */
                        }
                }
                else
                        sys_flag = 2;     /* address error */
                break;
        case 'Z' :
                k = j + 1;
                while ( in_buf[ k ] == ' ' && k < i )
                        k ++;
                if ( k != i )
                {
                        sys_flag = 1;
                        break;
                }
                printf( "\n     DOWNLOAD BINARY FILE TO MEMORY, ENTER
                                                        FILE NAME : " );
                get_str ( &j, in_buf );
                if ( j == 0 )
                {
                        sys_flag = 3;             /* command cancelled */
                        break;
                }
                hex = fopen ( in_buf, "rb" );  /* read binary file */
```

```
                                if ( hex == 0 )
                                {
                                        printf ( "\n\n    OPEN FILE FAILURE !" );
                                        sys_flag = 3;           /* command cancelled */
                                        break;
                                }
                                downloadbin ( hex, index );
                                printf ( "\n\n    DOWNLOAD COMPLETE !" );
                                sys_flag = 3;           /* command cancelled */
                                fclose ( hex );
                                break;
                        case '?' :
                                k = j + 1;
                                while ( in_buf[ k ] == ' ' && k < i )
                                        k ++;
                                if ( k != i )
                                {
                                        sys_flag = 1;
                                        break;
                                }
                                help ();
                                sys_flag = 3;           /* processing success */
                                break;
                        default :
                                sys_flag = 3;
                                printf ( "\n    Command Error !" );
                                break;
                }
                if ( sys_flag != 0 )
                        break;
        }
        if ( sys_flag == -1 )
                break;
        switch ( sys_flag )
        {
                case ( 0 ) :
                        printf ( "\n% " );
                        break;
                case ( 1 ) :
                        printf ( "\n    Syntax Error ! \n% " );
                        break;
                case ( 2 ) :
                        printf ( "\n    Address Error ! \n% " );
                        break;
                case ( 3 ) :
                        printf ( "\n% " );
                        break;
        }
    }
}
/* ------------------------------------------------------------------------
|       PURPOSE     :     scan input string to operate prom blank check
|       PARAMETERS  :
|                         string    --> input string
```

```
|                          str_ptr   --> input string pointer
|        RETURN VALUE :    0 : blank check fail
|                          1 : blank check success
|                         -1 : prom type error
  ------------------------------------------------------------------------------ */
int         blankchk ( char *string, int str_ptr )
{
        int       extern  blank1 ();
        int       extern  blank2 ();
        int       extern  blank3 ( int count, int pattern );
        int       extern  blank4 ( int count, int pattern );

        if ( strncmp ( "77S180", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 1024, 0xd800 ) );
        if ( strncmp ( "87S180", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 1024, 0xd800 ) );
        if ( strncmp ( "77S181", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 1024, 0xd800 ) );
        if ( strncmp ( "87S181", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 1024, 0xd800 ) );
        if ( strncmp ( "77S184", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 2048, 0x7000 ) );
        if ( strncmp ( "87S184", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 2048, 0x7000 ) );
        if ( strncmp ( "77S185", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 2048, 0x7000 ) );
        if ( strncmp ( "87S185", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 2048, 0x7000 ) );
        if ( strncmp ( "77S190", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 2048, 0xd800 ) );
        if ( strncmp ( "87S190", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 2048, 0xd800 ) );
        if ( strncmp ( "77S191", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 2048, 0xd800 ) );
        if ( strncmp ( "87S191", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 2048, 0xd800 ) );
        if ( strncmp ( "74S188", string + str_ptr, 6 ) == 0 )
                return ( blank1 () );
        if ( strncmp ( "54S188", string + str_ptr, 6 ) == 0 )
                return ( blank1 () );
        if ( strncmp ( "74S288", string + str_ptr, 6 ) == 0 )
                return ( blank1 () );
        if ( strncmp ( "54S288", string + str_ptr, 6 ) == 0 )
                return ( blank1 () );
        if ( strncmp ( "74S287", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 256, 0xf800 ) );
        if ( strncmp ( "54S287", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 256, 0xf800 ) );
        if ( strncmp ( "74S387", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 256, 0xf800 ) );
        if ( strncmp ( "54S387", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 256, 0xf800 ) );
        if ( strncmp ( "74S472", string + str_ptr, 6 ) == 0 )
                return ( blank2 () );
```

```
        if ( strncmp ( "54S472", string + str_ptr, 6 ) == 0 )
                return ( blank2 () );
        if ( strncmp ( "74S473", string + str_ptr, 6 ) == 0 )
                return ( blank2 () );
        if ( strncmp ( "54S473", string + str_ptr, 6 ) == 0 )
                return ( blank2 () );
        if ( strncmp ( "74S474", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 512, 0xda00 ) );
        if ( strncmp ( "54S474", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 512, 0xda00 ) );
        if ( strncmp ( "74S475", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 512, 0xda00 ) );
        if ( strncmp ( "54S475", string + str_ptr, 6 ) == 0 )
                return ( blank4 ( 512, 0xda00 ) );
        if ( strncmp ( "74S570", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 512, 0xf800 ) );
        if ( strncmp ( "54S570", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 512, 0xf800 ) );
        if ( strncmp ( "74S571", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 512, 0xf800 ) );
        if ( strncmp ( "54S571", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 512, 0xf800 ) );
        if ( strncmp ( "74S572", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 1024, 0x7000 ) );
        if ( strncmp ( "54S572", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 1024, 0x7000 ) );
        if ( strncmp ( "74S573", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 1024, 0x7000 ) );
        if ( strncmp ( "54S573", string + str_ptr, 6 ) == 0 )
                return ( blank3 ( 1024, 0x7000 ) );
        return ( -1 );
}
/* ------------------------------------------------------------------------
|       PURPOSE     :     scan input string to extract hexadecimal number
|       PARAMETERS  :
|                         string  --> input string
|                         str_ptr --> input string pointer
|                         a         --> converted hexadecimal number
|                         num       --> sacn character length
|                                       2 : extract 1 byte hexadecimal number
|                                       4 : extract 2 bytes hexadecimal number
|       RETURN VALUE :   0 : fail
|                         1 : success
  ------------------------------------------------------------------------ */
int     cal_num ( char *string, int *str_ptr, unsigned int *a, int num )
{
        int i,hexdigit;
        char c;

        while ( string[ *str_ptr ] == ' ' )
                ( *str_ptr ) ++;
        *a = 0;
        for ( i = 0; i < num; i ++ )
        {
```

```
                c = string[ *str_ptr ];
                if ( isxdigit ( c ) )
                {
                        if ( isalpha ( c ) )
                                hexdigit = c - 'A' + 10;
                        else
                                hexdigit = c - '0';
                        *a = ( *a << 4 ) + hexdigit;
                        ( *str_ptr ) ++;
                }
                else
                        if ( ( c == 32 || c == 0 ) && i > 0 && *a <= END_BUFFER )
                                return ( 1 );
                        else
                                return ( 0 );
        }
        if ( ( string[ *str_ptr ] == 0 || string[ *str_ptr ] == ' ' ) && *a <= END_BUFFER )
                return ( 1 );
        else
                return ( 0 );
}
/* ---------------------------------------------------------------------------------
|       PURPOSE     :     calculate specified memory range checksum value ( total )
|       PARAMETERS  :
|                         start_adr --> starting address where the checksum operation begin
|                         end_adr   --> the last location of the range to be checked
|                         store_adr --> optional parameter that stores the checksum value to a
|                                       specified address
|                         f         --> storing result flag
|                                       0 : not stored
|                                       1 : stored
|                         buf_ptr   --> the starting address of buffer memory
|       RETURN VALUE :   checksum value
  --------------------------------------------------------------------------------- */
unsigned int chksumt ( unsigned int start_adr, unsigned int end_adr, unsigned int store_adr, int f,
                        unsigned char *buf_ptr )
{
        unsigned int i;
        i = 0;
        while ( start_adr <= end_adr )
        {
                i = i + *( buf_ptr + start_adr );
                start_adr ++;
        }
        if ( f )
        {
                *( buf_ptr + store_adr ) = i >> 8;
                *( buf_ptr + store_adr + 1 ) = i;
        }
        return ( i );
}
/* ---------------------------------------------------------------------------------
|       PURPOSE     :     calculate specified memory range checksum value ( exor )
|       PARAMETERS  :
```

```
|                       start_adr --> starting address where the checksum operation begin
|                       end_adr   --> the last location of the range to be checked
|                       store_adr --> optional parameter that stores the checksum value to a
|                                     specified address
|                       f         --> storing result flag
|                                     0 : not stored
|                                     1 : stored
|                       buf_ptr   --> the starting address of buffer memory
|       RETURN VALUE :  checksum value
  ------------------------------------------------------------------------ */
unsigned int chksumx ( unsigned int start_adr, unsigned int end_adr, unsigned int store_adr, int f,
                       unsigned char *buf_ptr )
{
        unsigned int i;

        i = 0;
        while ( start_adr <= end_adr )
        {
                i = i + *( buf_ptr + start_adr );
                start_adr ++;
        }
        i = - i & 0x00ff;
        if ( f )
                *( buf_ptr + store_adr ) = i;
        return ( i );
}
/* ------------------------------------------------------------------------------
|       PURPOSE     :     copy data in buffer memory
|       PARAMETERS  :
|                       start_adr --> starting address where copy operation begin
|                       end_adr   --> the last location of the range to be copy
|                       des_adr   --> the beginning address where the specified data to be copy
|                       buf_ptr   --> the starting address of buffer memory
|       RETURN VALUE :  none
  ------------------------------------------------------------------------------ */
void    copy (unsigned int start_adr,unsigned int end_adr, unsigned int des_adr,unsigned char *buf_ptr )
{
        unsigned int i;

        i = end_adr - start_adr + 1;
        memmove ( buf_ptr + des_adr, buf_ptr + start_adr, i );
}
/* ------------------------------------------------------------------------------
|       PURPOSE     :     display specified memory range data
|       PARAMETERS  :
|                       start_adr --> starting address where display begin
|                       end_adr   --> the last location of the range to be displayed
|                       buf_ptr   --> the starting address of buffer memory
  ------------------------------------------------------------------------------ */
void    display (unsigned int start_adr,unsigned int end_adr,unsigned char *buf_ptr )
{
        char asc[ 17 ];
        int i,j,k;
```

```
printf ( "\n\n    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F\n" );
strcpy ( asc, "                " );
j = start_adr % 16;
k = start_adr - j;
printf ( "%04X ", k );
for ( i = 0; i < j; i ++ )
        printf ( "   " );
if ( start_adr + 15 - j < end_adr )
{
        for ( i = j; i < 16; i ++ )
        {
                printf ( "%02X ", *( buf_ptr + start_adr ) );
                if( isprint ( *( buf_ptr + start_adr ) ) )
                        asc[ i ] = *( buf_ptr + start_adr );
                else
                        asc[ i ] = '.';
                start_adr ++;
        }
        j = 0;
        printf ( " %s\n", asc );
        strcpy ( asc,"                " );
        while ( start_adr + 15 < end_adr )
        {
                printf ( "%04X ", start_adr );
                for ( i = 0; i < 16; i ++ )
                {
                        printf ( "%02X ", *( buf_ptr + start_adr ) );
                        if ( isprint ( *( buf_ptr + start_adr ) ) )
                                asc[ i ] = *( buf_ptr + start_adr );
                        else
                                asc[ i ] = '.';
                        start_adr ++;
                }
        printf ( " %s\n", asc );
        strcpy ( asc, "                " );
        }
        printf ( "%04X ", start_adr );
}
k = end_adr - start_adr + 1 + j;
for ( i = j; i < k; i ++ )
{
        printf ( "%02X ", *( buf_ptr + start_adr ) );
        if ( isprint ( *( buf_ptr + start_adr ) ) )
                asc[ i ] = *( buf_ptr + start_adr );
        else
                asc[ i ] = '.';
        start_adr ++;
}
while ( i < 16 )
{
        printf ( "   " );
        i ++;
}
printf ( " %s\n", asc );
```

```
}
/* ------------------------------------------------------------------------------
|       PURPOSE    :    download specified binary file to buffer memory
|       PARAMETERS :
|                       fname   --> file pointer
|                       buf_ptr --> the starting address of buffer memory
  ------------------------------------------------------------------------------ */
void    downloadbin ( FILE *fname, char *buf_ptr )
{
        int i;
        char c;

        i = 0;
        c = fgetc ( fname );
        while ( feof ( fname ) == 0 && i <= END_BUFFER )
        {
                *( buf_ptr + i ) = c;
                c = fgetc ( fname );
                i ++;
        }
}
/* ------------------------------------------------------------------------------
|       PURPOSE    :    download specified hex file to buffer memory ( INTEL format )
|       PARAMETERS :
|                       fname   --> file handle
|                       buf_ptr --> the starting address of buffer memory
|       RETURN VALUE :  0 : fail
|                       1 : success
  ------------------------------------------------------------------------------ */
int     downloadhex ( FILE *fname, char *buf_ptr )
{
        int i,chksum,count,rectype,buffer;
        unsigned int pointer;

        while ( fgetc ( fname ) == ':' )
        {
                if ( fscanf ( fname, "%2x", &count ) != 1 )
                        return ( 0 );
                if ( count == 0 )
                {
                        if ( fscanf ( fname, "%8x", &buffer ) != 1 )
                                return ( 0 );
                        if ( buffer == 0x01ff )
                                return ( 1 );
                        else
                                return ( 0 );
                }
                if ( fscanf ( fname, "%4x", &pointer ) != 1 )
                        return ( 0 );
                if ( pointer + count - 1 > END_BUFFER )
                        return ( 0 );
                if ( fscanf ( fname, "%2x", &rectype ) != 1 )
                        return ( 0 );
                switch ( rectype )
```

```
                {
                        case ( 2 ) :
                                chksum = rectype + count + ( pointer & 0x00ff ) + ( ( pointer >> 8 ) &
                                                                                        0x00ff );
                                for ( i = 0; i < 3; i ++ )
                                {
                                        if ( fscanf ( fname, "%2x", &buffer ) != 1 )
                                                return ( 0 );
                                        chksum += buffer;
                                }
                                if ( ( chksum & 0x00ff ) != 0 )
                                        return ( 0 );
                                break;
                        case ( 0 ) :
                                chksum = rectype + count + ( pointer & 0x00ff ) + ( ( pointer >> 8 ) &
                                                                                0x00ff );
                                for ( i = 0; i < count; i ++ )
                                {
                                        if ( fscanf ( fname, "%2x", &buffer ) != 1 )
                                                return ( 0 );
                                        chksum += buffer;
                                        *( buf_ptr + pointer ) = buffer;
                                        pointer ++;
                                }
                                if ( fscanf ( fname, "%2x", &buffer ) != 1 )
                                        return ( 0 );
                                chksum += buffer;
                                if ( ( chksum & 0x00ff ) != 0 )
                                        return ( 0 );
                                break;
                        default:
                                return ( 0 );
                }
                if ( fgetc ( fname ) != '\n' )
                        return ( 0 );
        }
        return ( 0 );
}
/* ---------------------------------------------------------------------------
|       PURPOSE     :     fill the memory range with a specified data
|       PARAMETERS  :
|                         start_adr --> starting address where the FILL operation begins
|                         end_adr   --> the last location of the range to be filled
|                         d         --> hexadecimal value to be written into the specified memory range
|                         buf_ptr   --> the starting address of buffer memory
 --------------------------------------------------------------------------- */
void    fill ( unsigned int start_adr, unsigned int end_adr, unsigned int d, unsigned char *buf_ptr )
{
        while ( start_adr <= end_adr )
        {
                *( buf_ptr + start_adr ) = d;
                start_adr ++;
        }
}
```

```
/* ------------------------------------------------------------------------
|       PURPOSE    :    get input string from keyboard
|       PARAMETERS :
|                       str_ptr --> pointer of string length
|                       in_buf  --> starting pointer of input string
 ------------------------------------------------------------------------ */
void    get_str ( int *str_ptr, char *in_buf )
{
        int i;
        char c,in_str[ STR_LENGTH ];

/*        --- read command ---          */
        in_str[ STR_LENGTH - 1 ] = 0;               /* null */
        i = 0;
        while ( i < STR_LENGTH - 1 )
        {
                c = getch ();
                if ( isprint ( c ) )
                {
                        putch ( c );
                        in_str[ i ] = c;
                        i++;
                }
                else
                {
                        if ( c == 8 && i > 0 )              /* backspace */
                        {
                                printf ( "\b \b" );
                                i--;
                        }
                        if ( c == 0 && i == 0 )
                                getch ();           /* discard next character */
                        if ( c == 0 && i > 0 )
                        {
                                if ( getch () == 'K' )  /* left arrow */
                                {
                                        printf ( "\b \b" );
                                        i--;
                                }
                        }
                        if ( c == 13 )              /* enter */
                        {
                                in_str[ i ] = 0;
                                break;
                        }
                        if ( c == 27 )              /* esc */
                        {
                                i = 0;              /* discard this command */
                                break;
                        }
                }
        }
        *str_ptr = i;
        strcpy ( in_buf, strupr ( in_str ) );           /* convert lowercase to uppercase */
```

```
}
/* ------------------------------------------------------------------------------
|       PURPOSE     :     check parameters ( a1 , a2, a3 ) for G, R & V commands
|       PARAMETERS  :  a1       --> address of the first parameter
|                      a2       --> address of the second parameter
|                      a3       --> address of the third parameter
|                      b1       --> the boundary value of *a1 & *a2
|                      b2       --> the boundary value of *a3
|                      string --> the starting address of input string
|                      k        --> the pointer of input string
|       RETURN VALUE :  -1 : no parameters
|                       0 : parameters error
|                       1 : success
  ------------------------------------------------------------------------------ */
int     grvchk ( unsigned int *a1 , unsigned int *a2, unsigned int *a3, unsigned int b1 , unsigned int b2,
                 char *string, int k )
{
        while ( *( string + k ) == ' ' )
                k ++;
        if ( *( string + k ) == 0 )
                return ( -1 );
        if ( cal_num ( string, &k, a1 , 4 ) == 0 )
                return ( 0 );
        if ( *a1 > b1 || cal_num ( string, &k, a2, 4 ) == 0 )
                return ( 0 );
        if ( *a1 > *a2 || *a2 > b1 || cal_num ( string, &k, a3, 4 ) == 0 )
                return ( 0 );
        if ( *a3 > b2 || *a2 - *a1 > b2 - *a3 )
                return ( 0 );
        while ( *( string + k ) == ' ' )
                k ++;
        if ( *( string + k ) == 0 )
                return ( 1 );
        return ( 0 );
}
/* ------------------------------------------------------------------------------
|       PURPOSE     :     display the command message
  ------------------------------------------------------------------------------ */
void      help ()
{
        printf ( "\n      COMMAND          DESCRIPTION" );
        printf ( "\n-------------------------------------------------------" );
        printf ( "\n    B  T            BLANK CHECK" );
        printf ( "\n    CT a1 a2 [a3]      CHECKSUM (TOTAL)" );
        printf ( "\n    CX a1 a2 [a3]      CHECKSUM (EXOR)" );
        printf ( "\n    D  a1 a2          DISPLAY BUFFER MEMORY" );
        printf ( "\n    F  a1 a2 d        FILL DATA" );
        printf ( "\n    G  T[ a1 a2 a3]    PROM PROGRAMMING" );
        printf ( "\n    I  a1             INSERT DATA" );
        printf ( "\n    K  a1 a2          DELETE DATA" );
        printf ( "\n    M  a1             MODIFY DATA" );
        printf ( "\n    R  T[ a1 a2 a3]    READ DATA FROM PROM" );
        printf ( "\n    T               DISPLAY PROM TYPES" );
        printf ( "\n    V  T[ a1 a2 a3]    VERIFY" );
```

```
        printf ( "\n    U           UPLOAD HEX FILE" );
        printf ( "\n    W           DOWNLOAD HEX FILE" );
        printf ( "\n    X a1 a2 a3      COPY BUFFER MEMORY" );
        printf ( "\n    Z           DOWNLOAD BINARY FILE" );
        printf ( "\n    ?           HELP" );
        printf ( "\n    Q           QUIT TO DOS\n" );
}
/* ---------------------------------------------------------------------------
|       PURPOSE     :     insert data into buffer memory
|       PARAMETERS  :
|                         start_adr --> starting address where the insert operation begins
|                         buf_ptr   --> the starting address of buffer memory
 --------------------------------------------------------------------------- */
void    insert ( unsigned int start_adr, unsigned char *buf_ptr )
{
        char c;
        unsigned int k,buffer,hexdigit;

        printf ( "\n" );
        while ( 1 )
        {
                printf ( "\n%04X ", start_adr );
                k = 0;
                buffer = 0;
                while ( k < 3 )
                {
                        c = getch () ;
                        if ( isxdigit ( c ) && k <= 1 )
                        {
                                putch ( c );
                                if ( isalpha ( c ) )
                                {
                                        if ( isupper ( c ) )
                                                hexdigit = c - 'A' + 10;
                                        else
                                                hexdigit = c - 'a' + 10;
                                }
                                else
                                        hexdigit = c - '0';
                                buffer = buffer * 16 * k + hexdigit;
                                k ++;
                        }
                        else
                        {
                                if ( c == 27 )                  /* esc */
                                {
                                        printf ( "\n" );
                                        return;
                                }
                                if ( c == 13 && k == 0 )    /* enter & k == 0 */
                                {
                                        printf ( "\n" );
                                        return;
                                }
```

```
                        if ( c == 13 && k >= 1 )    /* enter & k >= 1 */
                        {
                                memmove ( buf_ptr + start_adr + 1, buf_ptr + start_adr,
                                                        END_BUFFER - start_adr );
                                *( buf_ptr + start_adr ) = buffer;
                                start_adr ++;
                                if ( start_adr > END_BUFFER )
                                        start_adr = 0;
                                break;
                        }
                        if ( c == 8 && k == 1 )     /* backspace & k == 1 */
                        {
                                printf ( "\b \b" );
                                buffer = 0;
                                k = 0;
                        }
                        if ( c == 8 && k == 2 )     /* backspace & k == 2 */
                        {
                                printf ( "\b \b" );
                                buffer = ( buffer - hexdigit ) / 16;
                                k = 1;
                        }
                        if ( c == 0 && k == 0 )     /* special key */
                                c = getch () ;
                        if ( c == 0 && k == 1 )     /* special key */
                        {
                                c = getch () ;
                                if ( c == 'K' )      /* left arrow */
                                {
                                        printf ( "\b \b" );
                                        buffer = 0;
                                        k = 0;
                                }
                        }
                        if ( c == 0 && k == 2 )     /* special key */
                        {
                                c = getch () ;
                                if ( c == 'K' )      /* left arrow */
                                {
                                        printf ( "\b \b" );
                                        buffer = ( buffer - hexdigit ) / 16;
                                        k = 1;
                                }
                        }
                }
        }
    }
}
/* ---------------------------------------------------------------------------
|       PURPOSE     :     the contents of the specified memory range are deleted
|       PARAMETERS  :
|                         start_adr --> starting address where the delete operation begins
|                         end_adr   --> the last location of the range to be deleted
|                         buf_ptr   --> the starting address of buffer memory
```

```
  ------------------------------------------------------------------------------ */
void    kill ( unsigned int start_adr, unsigned int end_adr, unsigned char *buf_ptr )
{
        unsigned int i;

        i = END_BUFFER - end_adr;
        memmove ( buf_ptr + start_adr, buf_ptr + end_adr + 1, i );
}
/* ------------------------------------------------------------------------------
|       PURPOSE     :     the contents of the buffer memory are modified
|       PARAMETERS  :
|                         start_adr --> starting address where the modify operation begins
|                         buf_ptr   --> the starting address of buffer memory
  ------------------------------------------------------------------------------ */
void    modify ( unsigned int start_adr, unsigned char *buf_ptr )
{
        char c;
        unsigned int i,j,k,buffer,hexdigit;

        printf ( "\n\n    00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F\n" );
        j = start_adr % 16;
        k = start_adr - j;
        printf ( "%04X ", k );
        for ( i = k; i < k + 16; i ++ )
                printf ( "%02X ", *( buf_ptr + i ) );
        printf ( "\n    " );

        for ( i = 0; i < j; i ++ )
                printf ( "   " );
        while ( 1 )
        {
                for ( i = j; i < 16; i ++ )
                {
                        k = 0;
                        buffer = 0;
                        while ( k < 3 )
                        {
                                c = getch ();
                                if ( isxdigit ( c ) && k <= 1 )
                                {
                                        putch ( c );
                                        if ( isalpha ( c ) )
                                        {
                                                if ( isupper ( c ) )
                                                        hexdigit = c - 'A' + 10;
                                                else
                                                        hexdigit = c - 'a' + 10;
                                        }
                                        else
                                                hexdigit = c - '0';
                                        buffer = buffer * 16 * k + hexdigit;
                                        k ++;
                                }
                                else
```

```
{
        if ( c == 27 )                          /* esc */
        {
                printf ( "\n" );
                return;
        }
        if ( c == 13 && k == 0 )    /* enter & k == 0 */
        {
                start_adr = start_adr + 16 - i;
                break;
        }
        if ( c == 13 && k == 1 )    /* enter & k == 1 */
        {
                printf ( "\b%02X ", buffer );
                *( buf_ptr + start_adr ) = buffer;
                start_adr = start_adr + 16 - i;
                break;
        }
        if ( c == 13 && k == 2 )    /* enter & k == 2 */
        {
                *( buf_ptr + start_adr ) = buffer;
                start_adr = start_adr + 16 - i;
                break;
        }
        if ( c == 32 && k == 0 )    /* space & k == 0 */
        {
                printf ( "%02X ", *( buf_ptr + start_adr ) );
                start_adr ++;
                break;
        }
        if ( c == 32 && k == 1 )    /* space & k == 1 */
        {
                printf ( "\b%02X ", buffer );
                *( buf_ptr + start_adr ) = buffer;
                start_adr ++;
                break;
        }
        if ( c == 32 && k == 2 )    /* space & k == 2 */
        {
                printf ( " " );
                *( buf_ptr + start_adr ) = buffer;
                start_adr ++;
                break;
        }
        if ( c == 8 && k == 1 )     /* backspace & k == 1 */
        {
                printf ( "\b \b" );
                buffer = 0;
                k = 0;
        }
        if ( c == 8 && k == 2 )     /* backspace & k == 2 */
        {
                printf ( "\b \b" );
                buffer = ( buffer - hexdigit ) / 16;
```

```
                                        k = 1;
                                }
                                if ( c == 0 && k == 0 )     /* special key */
                                        c = getch ();
                                if ( c == 0 && k == 1 )     /* special key */
                                {
                                        c = getch ();
                                        if ( c == 'K' )       /* left arrow */
                                        {
                                                printf ( "\b \b" );
                                                buffer = 0;
                                                k = 0;
                                        }
                                }
                                if ( c == 0 && k == 2 )     /* special key */
                                {
                                        c = getch ();
                                        if ( c == 'K' )       /* left arrow */
                                        {
                                                printf ( "\b \b" );
                                                buffer = ( buffer - hexdigit ) / 16;
                                                k = 1;
                                        }
                                }
                        }
                }
                if ( c == 13 )              /* enter */
                        break;
        }
        j = 0;
        if ( start_adr > END_BUFFER )
                start_adr = 0;
        printf ( "\n%04X ", start_adr );
        for ( i = start_adr; i < start_adr + 16; i ++ )
                printf ( "%02X ", *( buf_ptr + i ) );
        printf ( "\n    " );
    }
}
/* ------------------------------------------------------------------------------
|       PURPOSE     :     display address when in prom programming
|       PARAMETERS  :     dspadr --> public variable set by assembly routine
  ------------------------------------------------------------------------------ */
void    promdsp ()
{
        printf ("\b\b\b\b%04X",dspadr);
}
/* ------------------------------------------------------------------------------
|       PURPOSE     :     scan input string to operate program prom function
|       PARAMETERS  :
|                         string    --> input string
|                         str_ptr   --> input string pointer
|                         chksum    --> checksum value
|       RETURN VALUE :    0 : parameter error
|                         1 : program prom success
```

```
|                        -1 : prom type error
  ------------------------------------------------------------------------------ */
int    programprom ( char *string, int str_ptr, int *chksum )
{
int    extern  program1 ( unsigned int s_start, int count, unsigned d_start );
int    extern  program2 ( unsigned int s_start, int count, unsigned d_start );
int    extern  program3 ( unsigned int s_start, int count, unsigned d_start, int e_pattern, int d_pattern );
int   extern   program4 ( unsigned int s_start, int count, unsigned d_start, int e_pattern, int d_pattern );

        unsigned int a1, a2, a3;
        int i;

        if ( strncmp ( "77S180", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
                if ( i == 1 )
                        *chksum = program4 ( a3, a2 - a1 +1, a1, 0xd800, 0xe400 );
                else
                        *chksum = program4 ( 0, 1024, 0, 0xd800, 0xe400 );
                return ( 1 );
        }
        if ( strncmp ( "87S180", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
                if ( i == 1 )
                        *chksum = program4 ( a3, a2 - a1 +1, a1, 0xd800, 0xe400 );
                else
                        *chksum = program4 ( 0, 1024, 0, 0xd800, 0xe400 );
                return ( 1 );
        }
        if ( strncmp ( "77S181", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
                if ( i == 1 )
                        *chksum = program4 ( a3, a2 - a1 +1, a1, 0xd800, 0xe400 );
                else
                        *chksum = program4 ( 0, 1024, 0, 0xd800, 0xe400 );
                return ( 1 );
        }
        if ( strncmp ( "87S181", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
```

```
        if ( i == 1 )
                *chksum = program4 ( a3, a2 - a1 +1, a1, 0xd800, 0xe400 );
        else
                *chksum = program4 ( 0, 1024, 0, 0xd800, 0xe400 );
        return ( 1 );
}
if ( strncmp ( "77S184", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n       ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0x7000, 0x7800 );
        else
                *chksum = program3 ( 0, 2048, 0, 0x7000, 0x7800 );
        return ( 1 );
}
if ( strncmp ( "87S184", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n       ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0x7000, 0x7800 );
        else
                *chksum = program3 ( 0, 2048, 0, 0x7000, 0x7800 );
        return ( 1 );
}
if ( strncmp ( "77S185", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n       ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0x7000, 0x7800 );
        else
                *chksum = program3 ( 0, 2048, 0, 0x7000, 0x7800 );
        return ( 1 );
}
if ( strncmp ( "87S185", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n       ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0x7000, 0x7800 );
        else
                *chksum = program3 ( 0, 2048, 0, 0x7000, 0x7800 );
        return ( 1 );
}
```

```
if ( strncmp ( "77S190", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n     PROGRAMMING NOW ....\n\n        " );
        if ( i == 1 )
                *chksum = program4 ( a3, a2 - a1 +1, a1, 0xd800, 0xe000 );
        else
                *chksum = program4 ( 0, 2048, 0, 0xd800, 0xe000 );
        return ( 1 );
}
if ( strncmp ( "87S190", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n     PROGRAMMING NOW ....\n\n        " );
        if ( i == 1 )
                *chksum = program4 ( a3, a2 - a1 +1, a1, 0xd800, 0xe000 );
        else
                *chksum = program4 ( 0, 2048, 0, 0xd800, 0xe000 );
        return ( 1 );
}
if ( strncmp ( "77S191", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n     PROGRAMMING NOW ....\n\n        " );
        if ( i == 1 )
                *chksum = program4 ( a3, a2 - a1 +1, a1, 0xd800, 0xe000 );
        else
                *chksum = program4 ( 0, 2048, 0, 0xd800, 0xe000 );
        return ( 1 );
}
if ( strncmp ( "87S191", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n     PROGRAMMING NOW ....\n\n        " );
        if ( i == 1 )
                *chksum = program4 ( a3, a2 - a1 +1, a1, 0xd800, 0xe000 );
        else
                *chksum = program4 ( 0, 2048, 0, 0xd800, 0xe000 );
        return ( 1 );
}
if ( strncmp ( "74S188", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 31, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n     PROGRAMMING NOW ....\n\n        " );
```

```
        if ( i == 1 )
                *chksum = program1 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = program1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "54S188", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 31, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program1 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = program1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "74S288", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 31, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program1 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = program1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "54S288", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 31, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program1 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = program1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "74S287", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 255, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0xf800, 0xfb00 );
        else
                *chksum = program3 ( 0, 256, 0, 0xf800, 0xfb00 );
        return ( 1 );
}
```

```
if ( strncmp ( "54S287", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 255, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n     PROGRAMMING NOW ....\n\n       " );
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0xf800, 0xfb00 );
        else
                *chksum = program3 ( 0, 256, 0, 0xf800, 0xfb00 );
        return ( 1 );
}
if ( strncmp ( "74S387", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 255, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n     PROGRAMMING NOW ....\n\n       " );
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0xf800, 0xfb00 );
        else
                *chksum = program3 ( 0, 256, 0, 0xf800, 0xfb00 );
        return ( 1 );
}
if ( strncmp ( "54S387", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 255, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n     PROGRAMMING NOW ....\n\n       " );
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0xf800, 0xfb00 );
        else
                *chksum = program3 ( 0, 256, 0, 0xf800, 0xfb00 );
        return ( 1 );
}
if ( strncmp ( "74S472", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n     PROGRAMMING NOW ....\n\n       " );
        if ( i == 1 )
                *chksum = program2 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = program2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S472", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n     PROGRAMMING NOW ....\n\n       " );
```

```
        if ( i == 1 )
                *chksum = program2 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = program2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S473", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program2 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = program2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S473", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program2 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = program2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S474", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program4 ( a3, a2 - a1 +1, a1, 0xda00, 0xe600 );
        else
                *chksum = program4 ( 0, 512, 0, 0xda00, 0xe600 );
        return ( 1 );
}
if ( strncmp ( "54S474", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program4 ( a3, a2 - a1 +1, a1, 0xda00, 0xe600 );
        else
                *chksum = program4 ( 0, 512, 0, 0xda00, 0xe600 );
        return ( 1 );
}
```

```
if ( strncmp ( "74S475", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n       ");
        if ( i == 1 )
                *chksum = program4 ( a3, a2 - a1 +1, a1, 0xda00, 0xe600 );
        else
                *chksum = program4 ( 0, 512, 0, 0xda00, 0xe600 );
        return ( 1 );
}
if ( strncmp ( "54S475", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n       ");
        if ( i == 1 )
                *chksum = program4 ( a3, a2 - a1 +1, a1, 0xda00, 0xe600 );
        else
                *chksum = program4 ( 0, 512, 0, 0xda00, 0xe600 );
        return ( 1 );
}
if ( strncmp ( "74S570", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n       ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0xf800, 0xfa00 );
        else
                *chksum = program3 ( 0, 512, 0, 0xf800, 0xfa00 );
        return ( 1 );
}
if ( strncmp ( "54S570", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n       ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0xf800, 0xfa00 );
        else
                *chksum = program3 ( 0, 512, 0, 0xf800, 0xfa00 );
        return ( 1 );
}
if ( strncmp ( "74S571", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n       ");
```

```
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0xf800, 0xfa00 );
        else
                *chksum = program3 ( 0, 512, 0, 0xf800, 0xfa00 );
        return ( 1 );
}
if ( strncmp ( "54S571", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0xf800, 0xfa00 );
        else
                *chksum = program3 ( 0, 512, 0, 0xf800, 0xfa00 );
        return ( 1 );
}
if ( strncmp ( "74S572", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0x7000, 0x7c00 );
        else
                *chksum = program3 ( 0, 1024, 0, 0x7000, 0x7c00 );
        return ( 1 );
}
if ( strncmp ( "54S572", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0x7000, 0x7c00 );
        else
                *chksum = program3 ( 0, 1024, 0, 0x7000, 0x7c00 );
        return ( 1 );
}
if ( strncmp ( "74S573", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        printf ("\n\n    PROGRAMMING NOW ....\n\n      ");
        if ( i == 1 )
                *chksum = program3 ( a3, a2 - a1 +1, a1, 0x7000, 0x7c00 );
        else
                *chksum = program3 ( 0, 1024, 0, 0x7000, 0x7c00 );
        return ( 1 );
}
```

```
        if ( strncmp ( "54S573", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                printf ("\n\n     PROGRAMMING NOW ....\n\n        " );
                if ( i == 1 )
                        *chksum = program3 ( a3, a2 - a1 +1, a1, 0x7000, 0x7c00 );
                else
                        *chksum = program3 ( 0, 1024, 0, 0x7000, 0x7c00 );
                return ( 1 );
        }
        return ( -1 );
}
/* ------------------------------------------------------------------------------
|       PURPOSE    :     display the prom types help message
  ------------------------------------------------------------------------------ */
void        promhelp ()
{
        printf ( "\n      PROM TYPE              SIZE" );
        printf ( "\n-------------------------------------------------" );
        printf ( "\n    74S188/54S188            32 * 8" );
        printf ( "\n    74S288/54S288            32 * 8" );
        printf ( "\n    74S287/54S287           256 * 4" );
        printf ( "\n    74S387/54S387           256 * 4" );
        printf ( "\n    74S472/54S472           512 * 8" );
        printf ( "\n    74S473/54S473           512 * 8" );
        printf ( "\n    74S474/54S474           512 * 8" );
        printf ( "\n    74S475/54S475           512 * 8" );
        printf ( "\n    74S570/54S570           512 * 4" );
        printf ( "\n    74S571/54S571           512 * 4" );
        printf ( "\n    74S572/54S572          1024 * 4" );
        printf ( "\n    74S573/54S573          1024 * 4" );
        printf ( "\n    77S180/87S180          1024 * 8" );
        printf ( "\n    77S181/87S181          1024 * 8" );
        printf ( "\n    77S184/87S184          2048 * 4" );
        printf ( "\n    77S185/87S185          2048 * 4" );
        printf ( "\n    77S190/87S190          2048 * 8" );
        printf ( "\n    77S191/87S191          2048 * 8\n" );
}
/* ------------------------------------------------------------------------------
|       PURPOSE    :     scan input string to operate read prom function
|       PARAMETERS :
|                        string    --> input string
|                        str_ptr   --> input string pointer
|       RETURN VALUE :   0 : parameter error
|                        1 : raed prom success
|                       -1 : prom type error
  ------------------------------------------------------------------------------ */
int     readprom ( char *string, int str_ptr, int *chksum )
{
int     extern  read1 ( unsigned int s_start, int count, unsigned d_start );
int     extern  read2 ( unsigned int s_start, int count, unsigned d_start );
int     extern  read3 ( unsigned int s_start, int count, unsigned d_start );
```

```
int     extern  read4 ( unsigned int s_start, int count, unsigned d_start );

        unsigned int a1, a2, a3;
        int i;

        if ( strncmp ( "77S180", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, 1023, END_BUFFER, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                if ( i == 1 )
                        *chksum = read4 ( a1 | 0xd800, a2 - a1 +1, a3 );
                else
                        *chksum = read4 ( 0xd800, 1024, 0 );
                return ( 1 );
        }
        if ( strncmp ( "87S180", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, 1023, END_BUFFER, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                if ( i == 1 )
                        *chksum = read4 ( a1 | 0xd800, a2 - a1 +1, a3 );
                else
                        *chksum = read4 ( 0xd800, 1024, 0 );
                return ( 1 );
        }
        if ( strncmp ( "77S181", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, 1023, END_BUFFER, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                if ( i == 1 )
                        *chksum = read4 ( a1 | 0xd800, a2 - a1 +1, a3 );
                else
                        *chksum = read4 ( 0xd800, 1024, 0 );
                return ( 1 );
        }
        if ( strncmp ( "87S181", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, 1023, END_BUFFER, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                if ( i == 1 )
                        *chksum = read4 ( a1 | 0xd800, a2 - a1 +1, a3 );
                else
                        *chksum = read4 ( 0xd800, 1024, 0 );
                return ( 1 );
        }
        if ( strncmp ( "77S184", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, 2047, END_BUFFER, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
```

```
        if ( i == 1 )
                *chksum = read3 ( a1 | 0x7000, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0x7000, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "87S184", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 2047, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0x7000, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0x7000, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "77S185", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 2047, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0x7000, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0x7000, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "87S185", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 2047, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0x7000, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0x7000, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "77S190", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 2047, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read4 ( a1 | 0xd800, a2 - a1 +1, a3 );
        else
                *chksum = read4 ( 0xd800, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "87S190", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 2047, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
```

```
                return ( 0 );
        if ( i == 1 )
                *chksum = read4 ( a1 | 0xd800, a2 - a1 +1, a3 );
        else
                *chksum = read4 ( 0xd800, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "77S191", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 2047, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read4 ( a1 | 0xd800, a2 - a1 +1, a3 );
        else
                *chksum = read4 ( 0xd800, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "87S191", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 2047, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read4 ( a1 | 0xd800, a2 - a1 +1, a3 );
        else
                *chksum = read4 ( 0xd800, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "74S188", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 31, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read1 ( a1, a2 - a1 +1, a3 );
        else
                *chksum = read1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "54S188", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 31, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read1 ( a1, a2 - a1 +1, a3 );
        else
                *chksum = read1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "74S288", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 31, END_BUFFER, string, str_ptr +6 );
```

```
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read1 ( a1, a2 - a1 +1, a3 );
        else
                *chksum = read1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "54S288", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 31, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read1 ( a1, a2 - a1 +1, a3 );
        else
                *chksum = read1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "74S287", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 255, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0xf800, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0xf800, 256, 0 );
        return ( 1 );
}
if ( strncmp ( "54S287", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 255, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0xf800, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0xf800, 256, 0 );
        return ( 1 );
}
if ( strncmp ( "74S387", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 255, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0xf800, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0xf800, 256, 0 );
        return ( 1 );
}
if ( strncmp ( "54S387", string + str_ptr, 6 ) == 0 )
{
```

```
        i = grvchk ( &a1, &a2, &a3, 255, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0xf800, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0xf800, 256, 0 );
        return ( 1 );
}
if ( strncmp ( "74S472", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read2 ( a1, a2 - a1 +1, a3 );
        else
                *chksum = read2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S472", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read2 ( a1, a2 - a1 +1, a3 );
        else
                *chksum = read2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S473", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read2 ( a1, a2 - a1 +1, a3 );
        else
                *chksum = read2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S473", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read2 ( a1, a2 - a1 +1, a3 );
        else
                *chksum = read2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S474", string + str_ptr, 6 ) == 0 )
```

```
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read4 ( a1 | 0xda00, a2 - a1 +1, a3 );
        else
                *chksum = read4 ( 0xda00, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S474", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read4 ( a1 | 0xda00, a2 - a1 +1, a3 );
        else
                *chksum = read4 ( 0xda00, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S475", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read4 ( a1 | 0xda00, a2 - a1 +1, a3 );
        else
                *chksum = read4 ( 0xda00, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S475", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read4 ( a1 | 0xda00, a2 - a1 +1, a3 );
        else
                *chksum = read4 ( 0xda00, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S570", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0xf800, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0xf800, 512, 0 );
        return ( 1 );
}
```

```
if ( strncmp ( "54S570", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0xf800, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0xf800, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S571", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0xf800, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0xf800, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S571", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 511, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0xf800, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0xf800, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S572", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 1023, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0x7000, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0x7000, 1024, 0 );
        return ( 1 );
}
if ( strncmp ( "54S572", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, 1023, END_BUFFER, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = read3 ( a1 | 0x7000, a2 - a1 +1, a3 );
        else
                *chksum = read3 ( 0x7000, 1024, 0 );
        return ( 1 );
```

```
        }
        if ( strncmp ( "74S573", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, 1023, END_BUFFER, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                if ( i == 1 )
                        *chksum = read3 ( a1 | 0x7000, a2 - a1 +1, a3 );
                else
                        *chksum = read3 ( 0x7000, 1024, 0 );
                return ( 1 );
        }
        if ( strncmp ( "54S573", string + str_ptr, 6 ) == 0 )
        {
                i = grvchk ( &a1, &a2, &a3, 1023, END_BUFFER, string, str_ptr + 6 );
                if ( i == 0 )
                        return ( 0 );
                if ( i == 1 )
                        *chksum = read3 ( a1 | 0x7000, a2 - a1 +1, a3 );
                else
                        *chksum = read3 ( 0x7000, 1024, 0 );
                return ( 1 );
        }
        return ( -1 );
}
/* ---------------------------------------------------------------------------
|       PURPOSE     :     upload specified buffer memory range to hex file ( INTEL format )
|       PARAMETERS  :
|                         fname     --> file handle
|                         start_adr --> starting address where the upload operation begins
|                         end_adr   --> the last location of the range to be upload
|                         buf_ptr   --> the starting address of buffer memory
|                         k           --> display flag while in processing
|                                 0 : no display
|                                 1 : display
  --------------------------------------------------------------------------- */
void    upload (FILE *fname, unsigned int start_adr, unsigned int end_adr, unsigned char *buf_ptr, int k)
{
        int i,j,chksum;

        if ( k == 1 )
                printf ( "\n\n" );
        while ( start_adr + 15 < end_adr )
        {
                if ( k == 1 )
                printf ( ":10%04X00", start_adr );
                fprintf ( fname, ":10%04X00", start_adr );
                chksum = 16 + ( start_adr & 0x00ff ) + ( ( start_adr >> 8 ) & 0x00ff );
                for ( i = 0; i < 16; i++ )
                {
                        if ( k == 1 )
                                printf( "%02X", *( buf_ptr + start_adr ) );
                        fprintf ( fname, "%02X", *( buf_ptr + start_adr ) );
                        chksum += *( buf_ptr + start_adr );
```

```
                start_adr ++;
            }
            if ( k == 1 )
                printf ( "%02X\n", -chksum & 0x00ff );
            fprintf ( fname, "%02X\n", -chksum & 0x00ff );
        }
        j = end_adr - start_adr + 1;
        if ( k == 1 )
            printf ( ":%02X%04X00", j, start_adr );
        fprintf ( fname, ":%02X%04X00", j, start_adr );
        chksum = j + ( start_adr & 0x00ff ) + ( ( start_adr >> 8 ) & 0x00ff );
        for ( i = 0; i < j; i ++ )
        {
            if ( k == 1 )
                printf ( "%02X", *( buf_ptr + start_adr ) );
            fprintf ( fname, "%02X", *( buf_ptr + start_adr ) );
            chksum += *( buf_ptr + start_adr );
            start_adr ++;
        }
        if ( k == 1 )
            printf ( "%02X\n", -chksum & 0x00ff );
        fprintf ( fname, "%02X\n", -chksum & 0x00ff );
}
/* ------------------------------------------------------------------------------
|       PURPOSE     :     scan input string to operate verification prom function
|       PARAMETERS  :
|                         string    --> input string
|                         str_ptr   --> input string pointer
|                         chksum    --> checksum value
|       RETURN VALUE :    0 : parameter error
|                         1 : verify prom success
|                        -1 : prom type error
  ------------------------------------------------------------------------------ */
int     verifyprom ( char *string, int str_ptr, int *chksum )
{
int     extern  verify1 ( unsigned int s_start, int count, unsigned d_start );
int     extern  verify2 ( unsigned int s_start, int count, unsigned d_start );
int     extern  verify3 ( unsigned int s_start, int count, unsigned d_start );
int     extern  verify4 ( unsigned int s_start, int count, unsigned d_start );

        unsigned int a1, a2, a3;
        int i;

        if ( strncmp ( "77S180", string + str_ptr, 6 ) == 0 )
        {
            i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
            if ( i == 0 )
                return ( 0 );
            if ( i == 1 )
                *chksum = verify4 ( a3 | 0xd800, a2 - a1 +1, a1 );
            else
                *chksum = verify4 ( 0xd800, 1024, 0 );
            return ( 1 );
        }
```

```
if ( strncmp ( "87S180", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify4 ( a3 | 0xd800, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xd800, 1024, 0 );
        return ( 1 );
}
if ( strncmp ( "77S181", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify4 ( a3 | 0xd800, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xd800, 1024, 0 );
        return ( 1 );
}
if ( strncmp ( "87S181", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify4 ( a3 | 0xd800, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xd800, 1024, 0 );
        return ( 1 );
}
if ( strncmp ( "77S184", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0x7000, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0x7000, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "87S184", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0x7000, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0x7000, 2048, 0 );
        return ( 1 );
```

```
}
if ( strncmp ( "77S185", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0x7000, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0x7000, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "87S185", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0x7000, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0x7000, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "77S190", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify4 ( a3 | 0xd800, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xd800, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "87S190", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify4 ( a3 | 0xd800, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xd800, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "77S191", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify4 ( a3 | 0xd800, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xd800, 2048, 0 );
```

```
        return ( 1 );
}
if ( strncmp ( "87S191", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 2047, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify4 ( a3 | 0xd800, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xd800, 2048, 0 );
        return ( 1 );
}
if ( strncmp ( "74S188", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 31, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify1 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = verify1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "54S188", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 31, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify1 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = verify1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "74S288", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 31, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify1 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = verify1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "54S288", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 31, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify1 ( a3, a2 - a1 +1, a1 );
        else
```

```
                *chksum = verify1 ( 0, 32, 0 );
        return ( 1 );
}
if ( strncmp ( "74S287", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 255, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0xf800, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0xf800, 256, 0 );
        return ( 1 );
}
if ( strncmp ( "54S287", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 255, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0xf800, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0xf800, 256, 0 );
        return ( 1 );
}
if ( strncmp ( "74S387", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 255, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0xf800, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0xf800, 256, 0 );
        return ( 1 );
}
if ( strncmp ( "54S387", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 255, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0xf800, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0xf800, 256, 0 );
        return ( 1 );
}
if ( strncmp ( "74S472", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify2 ( a3, a2 - a1 +1, a1 );
```

```
        else
                *chksum = verify2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S472", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify2 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = verify2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S473", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify2 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = verify2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S473", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify2 ( a3, a2 - a1 +1, a1 );
        else
                *chksum = verify2 ( 0, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S474", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify4 ( a3 | 0xda00, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xda00, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S474", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
```

```
                *chksum = verify4 ( a3 | 0xda00, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xda00, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S475", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify4 ( a3 | 0xda00, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xda00, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S475", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify4 ( a3 | 0xda00, a2 - a1 +1, a1 );
        else
                *chksum = verify4 ( 0xda00, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S570", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0xf800, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0xf800, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S570", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0xf800, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0xf800, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S571", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
```

```
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0xf800, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0xf800, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "54S571", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 511, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0xf800, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0xf800, 512, 0 );
        return ( 1 );
}
if ( strncmp ( "74S572", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0x7000, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0x7000, 1024, 0 );
        return ( 1 );
}
if ( strncmp ( "54S572", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0x7000, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0x7000, 1024, 0 );
        return ( 1 );
}
if ( strncmp ( "74S573", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
        if ( i == 0 )
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0x7000, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0x7000, 1024, 0 );
        return ( 1 );
}
if ( strncmp ( "54S573", string + str_ptr, 6 ) == 0 )
{
        i = grvchk ( &a1, &a2, &a3, END_BUFFER, 1023, string, str_ptr + 6 );
        if ( i == 0 )
```

```
                return ( 0 );
        if ( i == 1 )
                *chksum = verify3 ( a3 | 0x7000, a2 - a1 +1, a1 );
        else
                *chksum = verify3 ( 0x7000, 1024, 0 );
        return ( 1 );
    }
    return ( -1 );

}
```

□

```
u68255a         equ     0fef0h
u68255b         equ     0fef1h
u68255c         equ     0fef2h
u68255ctl       equ     0fef3h
u68255cmd       equ     080h
epctl           equ     0fee0h
typectl         equ     0fef8h
actl1           equ     0fee1h
dctl1           equ     ofee2h
cctl1           equ     ofee3h
actl2a          equ     0fee4h
actl2b          equ     0fee5h
dctl2           equ     0fee6h
cctl2           equ     0fee7h
actl3a          equ     0fee8h
actl3b          equ     0fee9h
dctl3           equ     0feeah
cctl3           equ     0feebh
actl4a          equ     0feech
actl4b          equ     0feedh
dctl4           equ     0feeeh
cctl4           equ     0feefh
                public  _buffer
                public  _init
                public  _cls
                public  _blank1,_blank2,_blank3,_blank4
                public  _read1,_read2,_read3,_read4
                public  _verify1,_verify2,_verify3,_verify4
                public  _program1,_program2,_program3,_program4
                extern  _verifyok:word
                extern  _promdsp:near,_dspadr:word
;-------------------------------------------------------------------------------
_TEXT           segment byte public 'CODE'
DGROUP          group   _DTAT,_BSS
                assume cs:_TEXT,ds:DGROUP,ss:DGROUP
_TEXT           ends
;-------------------------------------------------------------------------------
_DATA           segment word public 'DATA'
_DATA           ends
;-------------------------------------------------------------------------------
_BSS            segment word public 'BSS'
_BSS            ends
;-------------------------------------------------------------------------------
_DATA           segment word public 'DATA'
chksum          dw      0
v12_adr         dw      0
v12_data        db      0
prog_byte       db      0
buf_ptr         dw      0
retrycnt        db      0
verifycnt       db      0
verifybit       db      0
e_pattern       db      0
d_pattern       db      0
```

```
temp            db      0
program_tbl     label   byte
                db      00000000b
                db      00000010b
                db      00000100b
                db      00001000b
                db      00010000b
                db      00100000b
                db      01000000b
                db      10000000b
ptype1          dw      ax1
                dw      ax2
                dw      ax3
                dw      ax4
                dw      ax5
                dw      ax6
                dw      ax7
                dw      ax17
ptype2          dw      ax6
                dw      ax7
                dw      ax8
                dw      ax9
                dw      ax15
                dw      ax16
                dw      ax17
                dw      ax18
ptype3          dw      ax20
                dw      ax19
                dw      ax18
                dw      ax17
ptype4          dw      ax9
                dw      ax10
                dw      ax11
                dw      ax13
                dw      ax14
                dw      ax15
                dw      ax16
                dw      ax17
ax1             db      11111110b
                dw      u68255a
ax2             db      11111101b
                dw      u68255a
ax3             db      11111011b
                dw      u68255a
ax4             db      11110111b
                dw      u68255a
ax5             db      11101111b
                dw      u68255a
ax6             db      11011111b
                dw      u68255a
ax7             db      10111111b
                dw      u68255a
ax8             db      01111111b
                dw      u68255a
```

```
ax9             db      11111110b
                dw      u68255b
ax10            db      11111101b
                dw      u68255b
ax11            db      11111011b
                dw      u68255b
ax13            db      11110111b
                dw      u68255b
ax14            db      11101111b
                dw      u68255b
ax15            db      11011111b
                dw      u68255b
ax16            db      10111111b
                dw      u68255b
ax17            db      01111111b
                dw      u68255b
ax18            db      11111110b
                dw      u68255c
ax19            db      11111101b
                dw      u68255c
ax20            db      11111011b
                dw      u68255c
_buffer         db      32768 dup (0)
_DATA           ends
;-------------------------------------------------------------------------------
_TEXT           segment byte public 'CODE'
;-------------------------------------------------------------------------------
_init                   proc    near
                        push    ax                      ; push AX onto stack
                        push    dx                      ; push DX onto stack
                        mov     al,u68255cmd            ; set 8255A to mode 0
                        mov     dx,u68255ctl            ; set port a, port b, and port c
                        out     dx,al                   ; all in output mode
                        mov     al,0ffh
                        mov     dx,u68255a
                        out     dx,al                   ; set 8255A port a = 0ffh
                        mov     dx,u68255b
                        out     dx,al                   ; set 8255A port b = 0ffh
                        mov     dx,u68255c
                        out     dx,al                   ; set 8255A port c = 0ffh
                        mov     al,0
                        mov     dx,epotl
                        out     dx,al                   ; U35 set and PROM Vcc input = 5V
                        mov     al,0fh
                        mov     dx,typectl
                        out     dx,al                   ; U39 set and disable all PROM inputs
                        pop     dx                      ; restore DX
                        pop     ax                      ; restore AX
                        ret                             ; return to calling program (terminated)
_init                   endp
;-------------------------------------------------------------------------------
;               clear screen: set initial cursor position
_cls                    proc    near
                        push    ax              ; push AX onto stack
```

```
                push    bx              ; push BX onto stack
                push    cx              ; push CX onto stack
                push    dx              ; push DX onto stack
                mov     ah,of           ; current video state (AH=15)
                int     10h             ; invoke CRT interface
                mov     cx,0b000h       ; count of character to write
                cmp     al,7            ; 7-->mono display (AL=7, 80x25 B&W card)
                jz      clear           ; jump if the CRT card is monochrome display
                mov     cx,0b800h       ;  count of character to write
clear:
                push    es              ; push ES onto stack
                push    di              ; push DI onto stack
                mov     es,cx           ; request no. of count
                mov     al,25
                mul     ah              ; get no. of character columns on screen
                mov     cx,ax           ; store no. of characters (80x25)
                mov     ax,0720h        ; scroll active page down
                cld                     ; clear direction flag
                rep     stosw           ; request all zeros to set screen
                pop     di              ; restore DI
                pop     es,             ; restore ES
                mov     ah,2            ; set cursor position (AH=2)
                mov     dx,0            ; upper left position (dh,dl)= row,column
                int     10h             ; invoke CRT interface
                pop     dx              ; restore DX
                pop     cx              ; restore CX
                pop     bx              ; restore BX
                pop     ax              ; restore AX
                ret                     ; terminate
_cls            endp
;-------------------------------------------------------------------------------
;
;               BLANK CHECK FOR TYPE_1 :
;               54/74S188       54/74S288

_blank1         proc    near
            ;------ set hardware initialaization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl1        ; set all ones to turn off 7407
                out     dx,al           ; U8 set and sets up to read data from PROM
                mov     al,0eh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_1
                mov     cx,32           ; set byte count (initize NO. of loops)
                mov     bl,0            ; set PROM address=0, OE (output enable) low
loop1:
                mov     al,bl
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address (A0 - A4) to PROM
                inc     bl              ; increase PROM's address
                mov     dx,cctl1
                in      al,dx           ; U9 set and read data from PROM
```

```
                cmp     al,0            ; compare PROM data with 0
                jnz     non_blank1      ; jump if not equal to zero
                loop    loopb1          ; decrease CX, loop if nonzero
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,1            ; blank check success (default, 1 : success)
                ret                     ; return to calling routine (terminated)
        non_blank1 :
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,0            ; blank check fail (default, 0: fail)
                ret                     ; return to calling routine (C language part)
_blank1         endp
;-------------------------------------------------------------------------------
;
;               BLANK CHECK FOR TYPE_2:
;               54/74S472       54/74S473

_blank2         proc    near
            ;------ set hardware initialaization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl2        ; set all ones to turn off 7407
                out     dx,al           ; U23 set and sets up to read data from PROM
                mov     al,0dh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_2
                mov     cx,512          ; set byte count (initize NO. of loops)
                mov     bl,0            ; set PROM address=0, OE (output enable) low
        loopb2:
                mov     al,bl
                mov     dx,actl2a
                out     dx,al           ; U17 set and issues address (A0 - A7) to PROM
                mov     al,bh
                mov     dx,actl2b
                out     dx,al           ; U12 set to activate address line A8 input, OE low
                inc     bx              ; increase PROM's address
                mov     dx,cctl2
                in      al,dx           ; U29 set and read data from PROM
                cmp     al,0            ; compare PROM data with 0
                jnz     non_blank2      ; jump if not equal to zero
                loop    loopb2          ; decrease CX, loop if nonzero
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,1            ; blank check success (default, 1 : success)
                ret                     ; return to calling routine (terminated)
        non_blank2:
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
```

```
                    mov     ax,0            ; blank check fail (default, 0: fail)
                    ret                     ; return to calling routine (C language part)
_blank2             endp
;-------------------------------------------------------------------------------------
;
;                   BLANK CHECK FOR TYPE_3:
;       54/74S287   54/74S387   54/74S570   54/74S571
;       54/74S572   54/74S573   77/87S184   77/87S185

_blank3             proc    near
                ;------ set hardware initialaization ------
                    mov     al,0
                    mov     dx,epctl
                    out     dx,al           ; U35 set and PROM Vcc input = 5V
                    mov     al,0ffh
                    mov     dx,dctl3        ; set all ones to turn off 7407
                    out     dx,al           ; U24 set and sets up to read data from PROM
                    mov     al,0bh
                    mov     dx,typectl
                    out     dx,al           ; U39 set and enables PROM type_3
                    push    bp              ; push BP onto stack
                    mov     bp,sp           ; request stack pointer value
                    mov     cx,[bp+4]       ; set byte count (initize NO. of loops)
                    mov     bx,[bp+6]       ; set PROM address = 0, OE (output enable) low
                    pop     bp              ; restore BP
        loopb3:
                    mov     al,bl
                    mov     dx,actl3a
                    out     dx,al           ; U18 set and issues address (A0 - A7) to PROM
                    mov     al,bh
                    mov     dx,actl3b
                    out     dx,al           ; U13 set to activate another address input, OE low
                    inc     bx              ; increase PROM's address
                    mov     dx,cctl3
                    in      al,dx           ; U30 set and read data from PROM
                    test    al,00001111b    ; get low nibble (low 4 bits)
                    jnz     non_blank3      ; jump if not equal to zero
                    loop    loopb3          ; decrease CX, loop if nonzero
                    mov     al,0ffh
                    mov     dx,typectl
                    out     dx,al           ; U39 set and disables all PROM inputs
                    mov     ax,1            ; blank check success (default, 1 : success)
                    ret                     ; return to calling routine (terminated)
        non_blank3:
                    mov     al,0ffh
                    mov     dx,typectl
                    out     dx,al           ; U39 set and disables all PROM inputs
                    mov     ax,0            ; blank check fail (default, 0: fail)
                    ret                     ; return to calling routine (C language part)
_blank3             endp
;-------------------------------------------------------------------------------------
;
;                   BLANK CHECK FOR TYPE_4:
;       54/74S474   54/74S475   77/87S180
;       77/87S181   77/87S190   77/87S191
```

```
_blank4         proc    near
              ;------ set hardware initialaization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl4        ; set all ones to turn off 7407
                out     dx,al           ; U25 set and sets up to read data from PROM
                mov     al,07h
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_4
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                mov     cx,[bp+4]       ; set byte count (initize NO. of loops)
                mov     bx,[bp+6]       ; set PROM address = 0, OE (output enable) low
                pop     bp              ; restore BP
        loopb4:
                mov     al,bl
                mov     dx,actl4a
                out     dx,al           ; U19 set and issues address (A0 - A7) to PROM
                mov     al,bh
                mov     dx,actl4b
                out     dx,al           ; U14 set to activate another address pin, OE low
                inc     bx              ; increase PROM's address
                mov     dx,cctl4
                in      al,dx           ; U31 set and read data from PROM
                cmp     al,0            ; compare PROM data with 0
                jnz     non_blank4      ; jump if not equal to zero
                loop    loopb4          ; decrease CX, loop if nonzero
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,1            ; blank check success (default, 1 : success)
                ret                     ; return to calling routine (terminated)
        non_blank4:
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,0            ; blank check fail (default, 0: fail)
                ret                     ; return to calling routine (C language part)
_blank4         endp
;-------------------------------------------------------------------------------------------
;               READ FOR TYPE_1 :
;               54/74S188       54/74S288

_read1          proc    near
              ;------ set hardware initization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl1        ; set all ones to turn off 7407
                out     dx,al           ; U8 set and sets up to read data from PROM
```

```
                mov     al,0eh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_1
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                mov     bx,[bp+4]       ; get PROM CE low pattern & starting address
                mov     cx,[bp+6]       ; get byte count (range to be read)
                mov     di,[bp+8]       ; get starting address of buffer memory
                                        ;these variables are transfered from C language part
                mov     ah,0            ; get low byte AL
                mov     chksum,0        ; clear checksum
        loopr1:
                mov     al,bl
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address (A0 - A4) to PROM
                inc     bl              ; increase PROM address
                mov     dx,cctl1
                in      al,dx           ; U9 set and read data from PROM
                mov     _buffer[di],al  ; store data to buffer memory
                inc     di              ; increase buffer memory address
                add     chksum,ax       ; add checksum
                loop    loopr1          ; decrease CX, loop if nonzero
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value
                pop     di              ; restore DI
                pop     bp              ; restore BP
                ret                     ; return to calling routine (terminated)
_read1          endp
;-------------------------------------------------------------------------------------------
;               READ FOR TYPE_2:
;               54/74S472       54/74S473

_read2          proc    near
            ;------ set hardware initization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dotl2        ; set all ones to turn off 7407
                out     dx,al           ; U23 set and sets up to read data from PROM
                mov     al,0dh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_2
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                mov     bx,[bp+4]       ; get PROM OE low pattern & starting address
                mov     cx,[bp+6]       ; get byte count (range to be read)
                mov     di,[bp+8]       ; get starting address of buffer memory
                                        ;these variables are transfered from C language part
                mov     ah,0            ; get low byte AL
```

```
                mov     chksum,0        ; clear checksum
        loopr2:
                mov     al,bl
                mov     dx,actl2a
                out     dx,al           ; U17 set and issues address (A0 - A7) to PROM
                mov     al,bh
                mov     dx,actl2b
                out     dx,al           ; U12 set to activate address line A8 input, OE low
                inc     bx              ; increase PROM address
                mov     dx,cctl2
                in      al,dx           ; U29 set and read data from PROM
                mov     _buffer[di],al  ; store data to buffer memory
                inc     di              ; increase buffer memory address
                add     chksum,ax       ; add checksum
                loop    loopr2          ; decrease CX, loop if nonzero
                mov     al,0ffh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value
                pop     di              ; restore DI
                pop     bp              ; restore BP
                ret                     ; return to calling routine (terminated)
_read2          endp
;-------------------------------------------------------------------------------------------
;               READ FOR TYPE_3:
;       54/74S287       54/74S387       54/74S570       54/74S571
;       54/74S572       54/74S573       77/87S184       77/87S185

_read3          proc    near
            ;------ set hardware initization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl3        ; set all ones to turn off 7407
                out     dx,al           ; U24 set and sets up to read data from PROM
                mov     al,0bh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_3
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                mov     bx,[bp+4]       ; get PROM OE low pattern & starting address
                mov     cx,[bp+6]       ; get byte count (range to be read)
                mov     di,[bp+8]       ; get starting address of buffer memory
                                        ;these variables are transfered from C language part
                mov     ah,0            ; get low byte AL
                mov     chksum,0        ; clear checksum
        loopr3:
                mov     al,bl
                mov     dx,actl3a
                out     dx,al           ; U18 set and issues address (A0 - A7) to PROM
                mov     al,bh
```

```
                        mov     dx,actl3b
                        out     dx,al           ; U13 set to activate another address input, OE low
                        inc     bx              ; increase PROM's address
                        mov     dx,cctl3
                        in      al,dx           ; U30 set and read data from PROM
                        and     al,00001111b    ; get low nibble (low 4 bits)
                        mov     _buffer[di],al  ; store data to buffer memory
                        inc     di              ; increase buffer memory address
                        add     chksum,ax       ; add checksum
                        loop    loopr3          ; decrease CX, loop if nonzero
                        mov     al,0fh
                        mov     dx,typectl
                        out     dx,al           ; U39 set and disables all PROM inputs
                        mov     ax,chksum       ; get checksum value
                        pop     di              ; restore DI
                        pop     bp              ; restore BP
                        ret                     ; return to calling routine (terminated)
_read3                  endp
;-------------------------------------------------------------------------------------------
;
;                       READ FOR TYPE_4:
;       54/74S474       54/74S475       77/87S180
;       77/87S181       77/87S190       77/87S191

_read4                  proc    near
                ;------ set hardware initization ------
                        mov     al,0
                        mov     dx,epctl
                        out     dx,al           ; U35 set and PROM Vcc input = 5V
                        mov     al,0ffh
                        mov     dx,dctl4        ; set all ones to turn off 7407
                        out     dx,al           ; U25 set and sets up to read data from PROM
                        mov     al,07h
                        mov     dx,typectl
                        out     dx,al           ; U39 set and enables PROM type_4
                        push    bp              ; push BP onto stack
                        mov     bp,sp           ; request stack pointer value
                        push    di              ; push DI onto stack
                        mov     bx,[bp+4]       ; get PROM CE low pattern & starting address
                        mov     cx,[bp+6]       ; get byte count (range to be read)
                        mov     di,[bp+8]       ; get starting address of buffer memory
                                                ;these variables are transfered from C language part
                        mov     ah,0            ; get low byte AL
                        mov     chksum,0        ; clear checksum
        loopr4:
                        mov     al,bl
                        mov     dx,actl4a
                        out     dx,al           ; U19 set and issues address (A0 - A7) to PROM
                        mov     dx,actl4b
                        out     dx,al           ; U14 set to activate another address input, OE low
                        inc     bx              ; increase PROM's address
                        mov     dx,cctl4
                        in      al,dx           ; U31 set and read data from PROM
                        mov     _buffer[di],al  ; store data to buffer memory
```

```
                inc     di              ; increase buffer memory address
                add     chksum,ax       ; add checksum
                loop    loopr4          ; decrease CX, loop if nonzero
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value
                pop     di              ; restore DI
                pop     bp              ; restore BP
                ret                     ; return to calling routine (terminated)
_read4          endp
;-------------------------------------------------------------------------------
;               VERIFY FOR TYPE_1 :
;               54/74S188       54/74S288

_verify1        proc    near
            ;------ set hardware initialization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl1        ; set all ones to turn off 7407
                out     dx,al           ; U8 set and sets up to read data from PROM
                mov     al,0eh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_1
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                mov     bx,[bp+4]       ; get PROM OE low pattern & starting address
                mov     cx,[bp+6]       ; get byte count (range to be verified)
                mov     di,[bp+8]       ; get starting address of buffer memory
                                        ;these variables are transfered from C language part
                mov     ah,0            ; get low byte AL
                mov     chksum,0        ; clear checksum
                mov     _verifyok,1     ; assume verify ok ! ( default, 1 : success)
        loopv1 :
                mov     al,bl
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address (A0 - A4) to PROM
                inc     bl              ; increase PROM address
                mov     dx,cctl1
                in      al,dx           ; U9 set and read data from PROM
                cmp     _buffer[di],al  ; compare buffer memory's data with PROM's data
                jz      vok1            ; jump if data is the same
                mov     _verifyok,0     ; verify fail if data is not the same
        vok1 :
                inc     di              ; increase buffer memory address
                add     chksum,ax       ; add checksum
                loop    loopv1          ; decrease CX, loop if nonzero
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
```

```
                mov     ax,chksum       ; get checksum value
                pop     di              ; restore DI
                pop     bp              ; restore BP
                ret                     ; return to calling routine (C language part)
_verify1        endp
;-------------------------------------------------------------------------------
;               VERIFY FOR TYPE_2:
;               54/74S472       54/74S473

_verify2        proc    near
            ;------ set hardware initialization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl2        ; set all ones to turn off 7407
                out     dx,al           ; U23 set and sets up to read data from PROM
                mov     al,0dh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_2
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                mov     bx,[bp+4]       ; get PROM OE low pattern & starting address
                mov     cx,[bp+6]       ; get byte count (range to be verified)
                mov     di,[bp+8]       ; get starting address of buffer memory
                                        ;these variables are transfered from C language part
                mov     ah,0            ; get low byte AL
                mov     chksum,0        ; clear checksum
                mov     _verifyok,1     ; assume verify ok !( default, 1 : success)
        loopv2:
                mov     al,bl
                mov     dx,actl2a
                out     dx,al           ; U17 set and issues address (A0 - A7) to PROM
                mov     al,bh
                mov     dx,actl2b
                out     dx,al           ; U12 set to activate address line A8 input, OE low
                inc     bx              ; increase PROM address
                mov     dx,cctl2
                in      al,dx           ; U29 set and read data from PROM
                cmp     _buffer[di],al  ; compare buffer memory's data with PROM's data
                jz      vok2            ; jump if data is the same
                mov     _verifyok,0     ; verify fail if data is not the same
        vok2:
                inc     di              ; increase buffer memory address
                add     chksum,ax       ; add checksum
                loop    loopv2          ; decrease CX, loop if nonzero
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value
                pop     di              ; restore DI
                pop     bp              ; restore BP
                ret                     ; return to calling routine (C language part)
```

```
_verify2        endp
;-------------------------------------------------------------------------------
;                       VERIFY FOR TYPE_3:
;       54/74S287   54/74S387   54/74S570   54/74S571
;       54/74S572   54/74S573   77/87S184   77/87S185

_verify3        proc    near
            ;------ set hardware initialization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl3        ; set all ones to turn off 7407
                out     dx,al           ; U24 set and sets up to read data from PROM
                mov     al,0bh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_3
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                mov     bx,[bp+4]       ; get PROM OE low pattern & starting address
                mov     cx,[bp+6]       ; get byte count (range to be verified)
                mov     di,[bp+8]       ; get starting address of buffer memory
                                        ;these variables are transfered from C language part
                mov     ah,0            ; get low byte AL
                mov     chksum,0        ; clear checksum
                mov     _verifyok,1     ; assume verify ok !( default, 1 : suocess)
        loopv3:
                mov     al,bl
                mov     dx,actl3a
                out     dx,al           ; U18 set and issues address (A0 - A7) to PROM
                mov     al,bh
                mov     dx,actl3b
                out     dx,al           ; U13 set to activate another address input, OE low
                inc     bx              ; increase PROM's address
                mov     dx,cctl3
                in      al,dx           ; U30 set and read data from PROM
                and     al,00001111b    ; get low nibble (low 4 bits)
                mov     dh,_buffer[di]  ; get buffer memory data
                and     dh,00001111b    ; get low nibble (low 4 bits)
                cmp     dh,al           ; compare buffer memory's data with PROM's data
                jz      vok3            ; jump if data is the same
                mov     _verifyok,0     ; verify fail if data is not the same
        vok3:
                inc     di              ; increase buffer memory address
                add     chksum,ax       ; add checksum
                loop    loopv3          ; decrease CX, loop if nonzero
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value
                pop     di              ; restore DI
                pop     bp              ; restore BP
                ret                     ; return to calling routine (C language part)
```

```
_verify3        endp
;-------------------------------------------------------------------------------
;               VERIFY FOR TYPE_4:
;       54/74S474       54/74S475       77/87S180
;       77/87S181       77/87S190       77/87S191

_verify4        proc    near
            ;------ set hardware initialization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl4        ; set all ones to turn off 7407
                out     dx,al           ; U25 set and sets up to read data from PROM
                mov     al,07h
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_4
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                mov     bx,[bp+4]       ; get PROM OE low pattern & starting address
                mov     cx,[bp+6]       ; get byte count (range to be verified)
                mov     di,[bp+8]       ; get starting address of buffer memory
                                        ;these variables are transfered from C language part
                mov     ah,0            ; get low byte AL
                mov     chksum,0        ; clear checksum
                mov     _verifyok,1     ; assume verify ok ! ( default, 1 : success)
        loopv4:
                mov     al,bl
                mov     dx,actl4a
                out     dx,al           ; U19 set and issues address (A0 - A7) to PROM
                mov     al,bh
                mov     dx,actl4b
                out     dx,al           ; U14 set to activate another address input, OE low
                inc     bx              ; increase PROM's address
                mov     dx,cctl4
                in      al,dx           ; U31 set and read data from PROM
                cmp     _buffer[di],al  ; compare buffer memory's data with PROM's data
                jz      vok4            ; jump if data is the same
                mov     _verifyok,0     ; verify fail if data is not the same
        vok4:
                inc     di              ; increase buffer memory address
                add     chksum,ax       ; add checksum
                loop    loopv4          ; decrease CX, loop if nonzero
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value
                pop     di              ; restore DI
                pop     bp              ; restore BP
                ret                     ; return to calling routine (C language part)
_verify4        endp
```

```
;-------------------------------------------------------------------------------
;                       PROGRAM FOR TYPE_1 :
;                       54/74S188       54/74S288

_program1               proc    near
                ;------ set hardware initialization ------
                        mov     al,0
                        mov     dx,epctl
                        out     dx,al           ; U35 set and PROM Vcc input = 5V
                        mov     al,0ffh
                        mov     dx,dctl1        ; set all ones to turn off 7407
                        out     dx,al           ; U8 set and sets up to read data from PROM
                        mov     al,0eh
                        mov     dx,typectl
                        out     dx,al           ; U39 set and enables PROM type_1
                        push    bp              ; push BP onto stack
                        mov     bp,sp           ; request stack pointer value
                        push    di              ; push DI onto stack
                        push    si              ; push SI onto stack
                        mov     bx,[bp+4]       ; get prom starting address & OE low pattern
                        mov     _dspadr,bx      ; store current address for display
                        mov     cx,[bp+6]       ; get byte count (range to be programmed)
                        mov     di,[bp+8]       ; get starting address of buffer memory
                        mov     buf_ptr,di      ; store buffer memory pointer
                        mov     chksum,0        ; clear checksum
                        mov     _verifyok,1     ; assume verify ok ! (default, 1 : success)
        proceed1 :
                        mov     di,buf_ptr      ; initize DI
                        mov     al,_buffer[di]  ; request program byte
                        mov     prog_byte,al    ; store program byte
                        mov     al,bl
                        and     al,11011111b    ; get PROM output enable (OE) low
                        mov     dx,actl1
                        out     dx,al           ; U7 set and issues address (A0 - A4) to PROM
                        mov     dx,cctl1
                        in      al,dx           ; U9 set and read data from PROM
                        or      al,prog_byte
                        cmp     al,prog_byte    ; get exact programming data byte
                        jz      pc11            ; jump if the programming data byte ok
                        mov     _verifyok,0     ; PROM data error (default, 0: fail)
                        mov     al,0fh
                        mov     dx,typectl
                        out     dx,al           ; U39 set and disables all PROM inputs
                        pop     si              ; restore source index (SI)
                        pop     di              ; restore destination index (DI)
                        pop     bp              ; restore base pointer (BP)
                        ret                     ; return to calling routine
        pc11 :
                        push    bx              ; push BX (general register) onto stack
                        push    cx              ; pusg CX (count register) onto stack
                        call    _promdsp        ; display current address
                        pop     cx              ; restore CX
                        pop     bx              ; restore BX
```

```
                cmp     prog_byte,0     ; compare programming data byte with 0
                jnz     pc21            ; jump if the data byte not equal to 0
                jmp     nextbyte1       ; jump to program next data byte
pc21:
                mov     si,0            ; initial SI (store being programmed data bit)
rotate1:
                ror     prog_byte,1     ; rotate right one bit to obtain programming data bit
                jc      pbit1           ; jump if carry=1
                jmp     nextbit1        ; jump to program next data bit
pbit1:
                mov     retrycnt,10     ; set retry count=10 (NO. of loops)
                mov     verifybit,0     ; clear program verify flag
                mov     di,si
                add     di,di
                mov     di,ptype1[di]
                mov     al,[di]
                mov     v12_data,al
                mov     ax,[di+1]
                mov     v12_adr,ax      ; request exact program voltage input pin
retry1:
;---- <a> ----       disable PROM by setting OE high -----
                mov     al,bl
                or      al,00100000b    ; get PROM output enable high
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address to PROM
        ;-----        set program data bit -----
                mov     al ,program_tbl[si]
                mov     dx,dctl1
                out     dx,al           ; U8 set and issues data bit to be programmed
;----<b> ----        set PROM Vcc input +12 V -----
                mov     al,01000000b
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input =12V
;----<c> ----        set exact program data bit (Qi) +12 V -----
                mov     al,v12_data
                mov     dx,v12_adr
                out     dx,al           ; DG201 set to supply program voltage
;----<d> ----        set PROM OE enable ------
                mov     al,bl
                and     al,11011111b    ; get PROM output enables low
                mov     dx,aotl1
                out     dx,al           ; U7 set and issues address (A0 - A4) for PROM
;----<e> ----        disable PROM by setting OE high -----
                mov     al,bl
                or      al,00100000b    ; get PROM output enables (OE) high
                mov     dx,actl1
                out     dx,al           ; U7 set
        ;-----        set program bit (Qi) +5 V -----
                mov     al,11111111b
                mov     dx,v12_adr
                out     dx,al           ; disable DG201 to remove +12V from PROM output
        ;-----        set PROM Vcc input +5 V -----
```

```
                mov     al,00000000b
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input =5V
        ;-----          set up to read PROM ------
                mov     al,0ffh
                mov     dx,dctl1        ; set all ones to turn off 7407
                out     dx,al           ; U8 set and sets up to read data from PROM
        ;-----          set PROM OE enable ------
                mov     al,bl
                and     al,11011111b    ; get PROM output enables low
                mov     dx,actl1
                out     dx,al           ; U7 set and issues address (A0 - A4) for PROM
        ;-----          read data from PROM-----
                mov     dx,cctl1
                in      al,dx           ; U9 set and read data from PROM
                cmp     verifybit,0     ; compare PROM data with 0
                jz      no_verify1      ; jump if the data bit is equal to 0 (not verified)
                dec     verifycnt       ; decrease verify count (initial value=5)
                jz      nextbit1        ; jump if  the verify count is equal to 0
                                        ; and to program next bit
                jmp     retry1          ; loop for 10 times
no_verify1 :                            ; loop for retry count
                test    al,program_tbl[si] ; get program data bit which will be programmed
                jnz     vfbitok1        ; jump if not equal 0 (verify data bit ok)
                dec     retrycnt        ; decrease retry count (initial value=10)
                jz      vfbitfail1      ; jump if retry count is equal to 0
                jmp     retry1          ; jump to reprogram the data bit
vfbitok1:
                mov     verifycnt,5     ; set additional 5 pulse for programming
                mov     verifybit,1     ; bit verify ok (default, 1 : success)
                jmp     retry1          ; jump to reprogram the data bit
vfbitfail1 :
                mov     _verifyok,0     ; verify fail (programming fail)
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                pop     si              ;restore source index (SI)
                pop     di              ;restore destination index (DI)
                pop     bp              ;restore base pointer (BP)
                ret                     ;return to calling routine
nextbit1 :
                inc     si              ; increase programmed data bit count
                cmp     si,8            ; compare the programming data bit with 8
                jz      nextbyte1       ; jump if the data byte (8 bits) has been programmed
                jmp     rotate1         ; jump to program next data bit
nextbyte1 :
                mov     ah,0            ; get low byte AL
                mov     al,prog_byte    ; store program byte
                add     chksum,ax       ; add checksum value
                inc     bx              ; incerase PROM address
                mov     _dspadr,bx      ; store display address
                inc     buf_ptr         ; increase buffer memory address
                dec     cx              ; decrease byte count which is to be programmed
                jz      end1            ; jump if byte count is equal to 0
```

```
                jmp     proceed1        ; jump to program next byte
        end1:
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value
                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
_program1       endp
;-------------------------------------------------------------------------------
;
;               PROGRAM FOR TYPE_2:
;               54/74S472       54/74S473

_program2       proc    near
            ;------ set hardware initialization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl2        ; set all ones to turn off 7407
                out     dx,al           ; U23 set and sets up to read data from PROM
                mov     al,0dh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_2
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                push    si              ; push SI onto stack
                mov     bx,[bp+4]       ; get prom starting address & OE low pattern
                mov     _dspadr,bx      ; store current address for display
                mov     cx,[bp+6]       ; get byte count (range to be programmed)
                mov     di,[bp+8]       ; get starting address of buffer memory
                mov     buf_ptr,di      ; store buffer memory pointer
                mov     chksum,0        ; clear checksum
                mov     _verifyok,1     ; assume verify ok ! (default, 1 : success)
        proceed2:
                mov     di,buf_ptr      ; initize DI
                mov     al,_buffer[di]  ; request program byte
                mov     prog_byte,al    ; store program byte
                mov     al,bl
                mov     dx,actl2a
                out     dx,al           ; U17 set and issues address (A0 - A7) to PROM
                mov     al,bh
                and     al,11111101b    ; get PROM output enable (OE) low
                mov     dx,actl2b
                out     dx,al           ; U12 set and activate address line A8 input, OE low
                mov     dx,cctl2
                in      al,dx           ; U29 set and read data from PROM
                or      al,prog_byte
                cmp     al,prog_byte    ; get exact programming data byte
                jz      pc12            ; jump if the programming data byte ok
                mov     _verifyok,0     ; PROM data error (default, 0: fail)
```

```
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ;U39 set and disables all PROM inputs
                pop     si              ;restore source index (SI)
                pop     di              ;restore destination index (DI)
                pop     bp              ;restore base pointer (BP)
                ret                     ;return to calling routine
pc12:
                push    bx              ;push BX (general register) onto stack
                push    cx              ;pusg CX (count register) onto stack
                call    _promdsp        ;display current address
                pop     cx              ;restore CX
                pop     bx              ;restore BX
                cmp     prog_byte,0     ;compare programming data byte with 0
                jnz     pc22            ;jump if the data byte not equal to 0
                jmp     nextbyte2       ;jump to program next data byte
pc22:
                mov     si,0            ;initial SI (store being programmed data bit)
rotate2:
                ror     prog_byte,1     ;rotate right one bit to obtain programming data bit
                jc      pbit2           ;jump if carry=1
                jmp     nextbit2        ;jump to program next data bit
pbit2:
                mov     retrycnt,10     ;set retry count=10 (NO. of loops)
                mov     verifybit,0     ;clear program verify flag
                mov     di,si
                add     di,di
                mov     di,ptype2[di]
                mov     al,[di]
                mov     v12_data,al
                mov     ax,[di+1]
                mov     v12_adr,ax      ;request exact program voltage input pin
retry2:
;---- <a> ----       disable PROM by setting OE high -----
                mov     al,bh
                or      al,00000010b    ;get PROM output enable high
                mov     dx,actl2b
                out     dx,al           ;U12 set ,OE high
        ;-----          set program data bit -----
                mov     al ,program_tbl[si]
                mov     dx,dctl2
                out     dx,al           ;U23 set and issues data bit to be programmed
;----<b> ----           set PROM Vcc input +12 V -----
                mov     al,01000000b
                mov     dx,epctl
                out     dx,al           ;U35 set and PROM Vcc input =12V
;----<c> ----           set exact program data bit (Qi) +12 V -----
                mov     al,v12_data
                mov     dx,v12_adr
                out     dx,al           ;DG201 set to supply program voltage
;----<d> ----           set PROM OE enable ------
                mov     al,bh
                and     al,11111101b    ;get PROM output enables low
```

```
                mov     dx,act12b
                out     dx,al           ; U12 set and activate address line A8 input,OE low
;----<e>----             disable PROM by setting OE high -----
                mov     al,bl
                or      al,00000010b    ; get PROM output enables (OE) high
                mov     dx,act12b
                out     dx,al           ; U12 set ,OE high
        ;-----          set program bit (Qi) +5 V -----
                mov     al,11111111b
                mov     dx,v12_adr
                out     dx,al           ; disable DG201 to remove +12V from PROM output
        ;-----          set PROM Vcc input +5 V -----
                mov     al,00000000b
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input =5V
        ;-----          set up to read PROM ------
                mov     al,0ffh
                mov     dx,dct12        ; set all ones to turn off 7407
                out     dx,al           ; U23 set and sets up to read data from PROM
        ;-----          set PROM OE enable ------
                mov     al,bl
                and     al,11111101b    ; get PROM output enables low
                mov     dx,act12b
                out     dx,al           ; U12 set and issues address (A8) for PROM
        ;-----          read data from PROM-----
                mov     dx,cct12
                in      al,dx           ; U29 set and read data from PROM
                cmp     verifybit,0     ; compare PROM data with 0
                jz      no_verify2      ; jump if the data bit is equal to 0 (not verified)
                dec     verifycnt       ; decrease verify count (initial value=5)
                jz      nextbit2        ; jump if  the verify count is equal to 0
                                        ; and to program next bit
                jmp     retry2          ; loop for 10 times
no_verify2:                             ; loop for retry count
                test    al,program_tbl[si] ; get program data bit which will be programmed
                jnz     vfbitok2        ; jump if not equal 0 (verify data bit ok)
                dec     retrycnt        ; decrease retry count (initial value=10)
                jz      vfbitfail2      ; jump if retry count is equal to 0
                jmp     retry2          ; jump to reprogram the data bit
vfbitok2:
                mov     verifycnt,5     ; set additional 5 pulse for programming
                mov     verifybit,1     ; bit verify ok (default, 1 : success)
                jmp     retry2          ; jump to reprogram the data bit
vfbitfail2:
                mov     _verifyok,0     ; verify fail (programming fail)
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
nextbit2:
```

```
                inc     si              ; increase programmed data bit count
                cmp     si,8            ; compare the programming data bit with 8
                jz      nextbyte2       ; jump if the data byte (8 bits) has been programmed
                jmp     rotate2         ; jump to program next data bit
        nextbyte2:
                mov     ah,0            ; get low byte AL
                mov     al,prog_byte    ; store program byte
                add     chksum,ax       ; add checksum value
                inc     bx              ; incerase PROM address
                mov     _dspadr,bx      ; store display address
                inc     buf_ptr         ; increase buffer memory address
                dec     cx              ; decrease byte count which is to be programmed
                jz      end2            ; jump if byte count is equal to 0
                jmp     proceed2        ; jump to program next byte
        end2:
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value
                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
_program2       endp
;-------------------------------------------------------------------------------------------
;               PROGRAM FOR TYPE_3:
;       54/74S287   54/74S387   54/74S570   54/74S571
;       54/74S572   54/74S573   77/87S184   77/87S185

_program3       proc    near
            ;------ set hardware initialization ------
                mov     al,0
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl3        ; set all ones to turn off 7407
                out     dx,al           ; U24 set and sets up to read data from PROM
                mov     al,0bh
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_3
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                push    si              ; push SI onto stack
                mov     bx,[bp+4]       ; get prom starting address & OE low pattern
                mov     _dspadr,bx      ; store current address for display
                mov     cx,[bp+6]       ; get byte count (range to be programmed)
                mov     di,[bp+8]       ; get starting address of buffer memory
                mov     ax,[bp+10]
                mov     e_pattern,ah    ; get PROM OE low pattern (enabled)
                mov     ax,[bp+12]
                mov     d_pattern,ah    ; get PROM OE high pattern (disabled)
                mov     buf_ptr,di      ; store buffer memory pointer
                mov     chksum,0        ; clear checksum
```

```
                mov     _verifyok,1     ; assume verify ok ! (default, 1 : success)
proceed3:
                mov     di,buf_ptr      ; initize DI
                mov     al,_buffer[di]  ; request program byte
                and     al,00001111b    ; get low nibble (low 4 bits)
                mov     prog_byte,al    ; store program byte
                mov     temp,al         ; store program byte
                mov     al,bl
                mov     dx,actl3a
                out     dx,al           ; U18 set and issues address (A0 - A7) to PROM
                mov     al,bh
                or      al,e_pattern
                mov     dx,actl3b
                out     dx,al           ; U13 set to activate address (A8-A10) input,OE low
                mov     dx,cctl3
                in      al,dx           ; U30 set and read data from PROM
                and     al,00001111b    ; get low nibble (low 4 bits)
                or      al,prog_byte    ; get exact programming data byte
                cmp     al,prog_byte
                jz      pc13            ; jump if the programming data byte ok
                mov     _verifyok,0     ; PROM data error (default, 0: fail)
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
pc13:
                push    bx              ; push BX (general register) onto stack
                push    cx              ; pusg CX (count register) onto stack
                call    _promdsp        ; display current address
                pop     cx              ; restore CX
                pop     bx              ; restore BX
                cmp     prog_byte,0     ; compare programming data byte with 0
                jnz     pc23            ; jump if the data byte not equal to 0
                jmp     nextbyte3       ; jump to program next data byte
pc23:
                mov     si,0            ; initial SI (store being programmed data bit)
rotate3:
                ror     prog_byte,1     ; rotate right one bit to obtain programming data bit
                jc      pbit3           ; jump if carry=1
                jmp     nextbit3        ; jump to program next data bit
pbit3:
                mov     retrycnt,10     ; set retry count=10 (NO. of loops)
                mov     verifybit,0     ; clear program verify flag
                mov     di,si
                add     di,di
                mov     di,ptype3[di]
                mov     al,[di]
                mov     v12_data,al
                mov     ax,[di+1]
                mov     v12_adr,ax      ; request exact program voltage (DG201)
```

```
retry3:
;---- <a> ----        disable PROM by setting OE high -----
                mov     al,bh
                or      al,d_pattern      ; get PROM output enable high (disable PROM)
                mov     dx,actl3b
                out     dx,al             ; U13 set, OE high
        ;-----          set program data bit -----
                mov     al ,program_tbl[si]
                mov     dx,dctl3
                out     dx,al             ; U24 set and issues data bit to be programmed
;----<b>----             set PROM Vcc input +12 V -----
                mov     al,01000000b
                mov     dx,epctl
                out     dx,al             ; U35 set and PROM Vcc input =12V
;----<c> ----            set exact program data bit (Qi) +12 V -----
                mov     al,v12_data
                mov     dx,v12_adr
                out     dx,al             ; DG201 set to supply program voltage
;----<d> ----            set PROM OE enable ------
                mov     al,bh
                or      al,e_pattern      ; get PROM output enables low (enable PROM)
                mov     dx,actl3b
                out     dx,al             ; U13 set and issues address to PROM
;----<e> ----            disable PROM by setting OE high -----
                mov     al,bh
                or      al,d_pattern      ; get PROM output enables (OE) high (disable PROM)
                mov     dx,actl3b
                out     dx,al             ; U13 set, OE high
        ;-----          set program bit (Qi) +5 V -----
                mov     al,11111111b
                mov     dx,v12_adr
                out     dx,al             ; disable DG201 to remove +12V from PROM output
        ;-----          set PROM Vcc input +5 V -----
                mov     al,00000000b
                mov     dx,epctl
                out     dx,al             ; U35 set and PROM Vcc input =5V
        ;-----          set up to read PROM ------
                mov     al,0ffh
                mov     dx,dctl3          ; set all ones to turn off 7407
                out     dx,al             ; U24 set and sets up to read data from PROM
        ;-----          set PROM OE enable ------
                mov     al,bh
                or      al,e_pattern      ; get PROM output enables low
                mov     dx,actl3b
                out     dx,al             ; U13 set, OE high
        ;-----          read data from PROM-----
                mov     dx,cctl3
                in      al,dx             ; U30 set and read data from PROM
                cmp     verifybit,0       ; compare PROM data with 0
                jz      no_verify3        ; jump if the data bit is equal to 0 (not verified)
                dec     verifycnt         ; decrease verify count (initial value=5)
                jz      nextbit3          ; jump if  the verify count is equal to 0
```

```
                                        ; and to program next bit
                jmp     retry3          ; loop for 10 times
no_verify3:                             ; loop for retry count
                test    al,program_tbl[si] ; get program data bit which will be programmed
                jnz     vfbitok3        ; jump if not equal 0 (verify data bit ok)
                dec     retrycnt        ; decrease retry count (initial value=10)
                jz      vfbitfail3      ; jump if retry count is equal to 0
                jmp     retry3          ; jump to reprogram the data bit
vfbitok3:
                mov     verifycnt,5     ; set additional 5 pulse for programming
                mov     verifybit,1     ; bit verify ok (default, 1 : success)
                jmp     retry3          ; jump to reprogram the data bit
vfbitfail3:
                mov     _verifyok,0     ; verify fail (programming fail)
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
nextbit3:
                inc     si              ; increase programmed data bit count
                cmp     si,4            ; compare the programming data bit with 8
                jz      nextbyte3       ; jump if the data byte (8 bits) has been programmed
                jmp     rotate3         ; jump to program next data bit
nextbyte3:
                mov     ah,0            ; get low byte AL
                mov     al,temp         ; store program byte
                add     chksum,ax       ; add checksum value
                inc     bx              ; incerase PROM address
                mov     _dspadr,bx      ; store display address
                inc     buf_ptr         ; increase buffer memory address
                dec     cx              ; decrease byte count which is to be programmed
                jz      end3            ; jump if byte count is equal to 0
                jmp     proceed3        ; jump to program next byte
end3:
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value
                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
_program3       endp
;-------------------------------------------------------------------------------
;                       PRGORAM FOR TYPE_4:
;       54/74S474       54/74S475       77/87S180
;       77/87S181       77/87S190       77/87S191

_program4       proc    near
                ;------ set hardware initialization ------
                mov     al,0
```

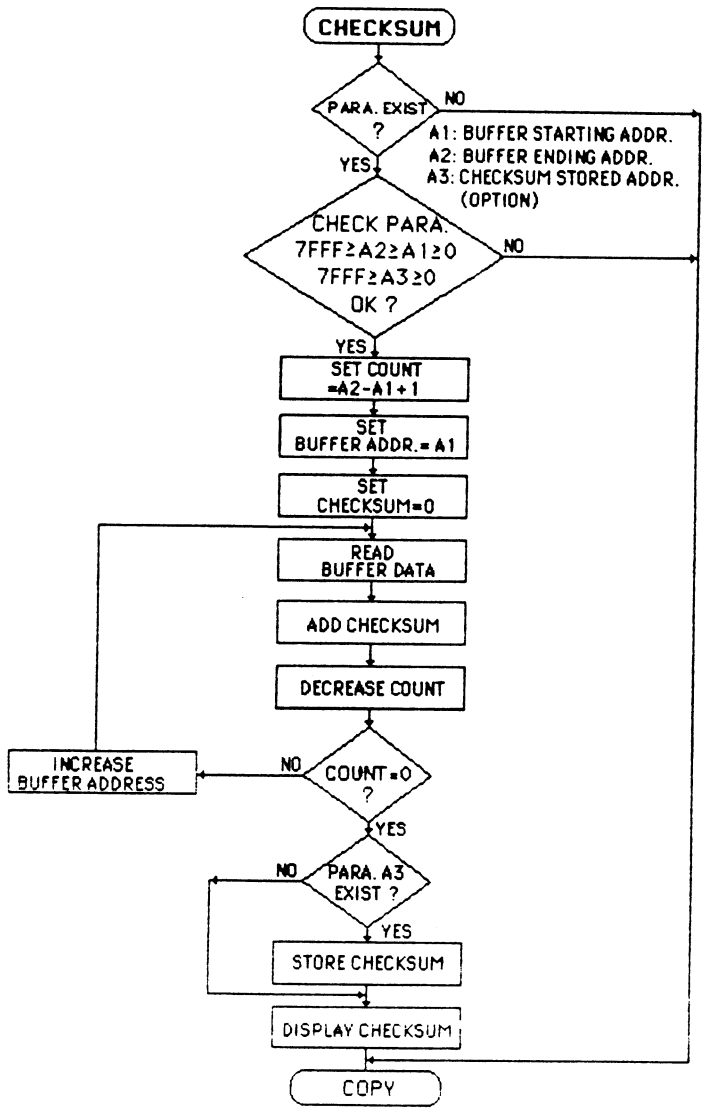
```
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input = 5V
                mov     al,0ffh
                mov     dx,dctl4        ; set all ones to turn off 7407
                out     dx,al           ; U25 set and sets up to read data from PROM
                mov     al,07h
                mov     dx,typectl
                out     dx,al           ; U39 set and enables PROM type_4
                push    bp              ; push BP onto stack
                mov     bp,sp           ; request stack pointer value
                push    di              ; push DI onto stack
                push    si              ; push SI onto stack
                mov     bx,[bp+4]       ; get prom starting address & OE low pattern
                mov     _dspadr,bx      ; store current address for display
                mov     cx,[bp+6]       ; get byte count (range to be programmed)
                mov     di,[bp+8]       ; get starting address of buffer memory
                mov     ax,[bp+10]
                mov     e_pattern,ah    ; store PROM enable pattern
                mov     ax,[bp+12]
                mov     d_pattern,ah    ; store PROM disable pattern
                mov     buf_ptr,di      ; store buffer memory pointer
                mov     chksum,0        ; clear checksum
                mov     _verifyok,1     ; assume verify ok ! (default, 1 : success)
proceed4:
                mov     di,buf_ptr      ; initize DI
                mov     al,_buffer[di]  ; request program byte
                mov     prog_byte,al    ; store program byte
                mov     al,bl
                mov     dx,actl4a
                out     dx,al           ; U19 set and issues address (A0 - A7) to PROM
                mov     al,bh
                or      al,e_pattern
                mov     dx,actl4b
                out     dx,al
                mov     dx,cctl4
                in      al,dx           ; U31 set and read data from PROM
                or      al,prog_byte
                cmp     al,prog_byte    ; get exact programming data byte
                jz      pc14            ; jump if the programming data byte ok
                mov     _verifyok,0     ; PROM data error (default, 0: fail)
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
pc14:
                push    bx              ; push BX (general register) onto stack
                push    cx              ; pusg CX (count register) onto stack
                call    _promdsp        ; display current address
                pop     cx              ; restore CX
                pop     bx              ; restore BX
                cmp     prog_byte,0     ; compare programming data byte with 0
```

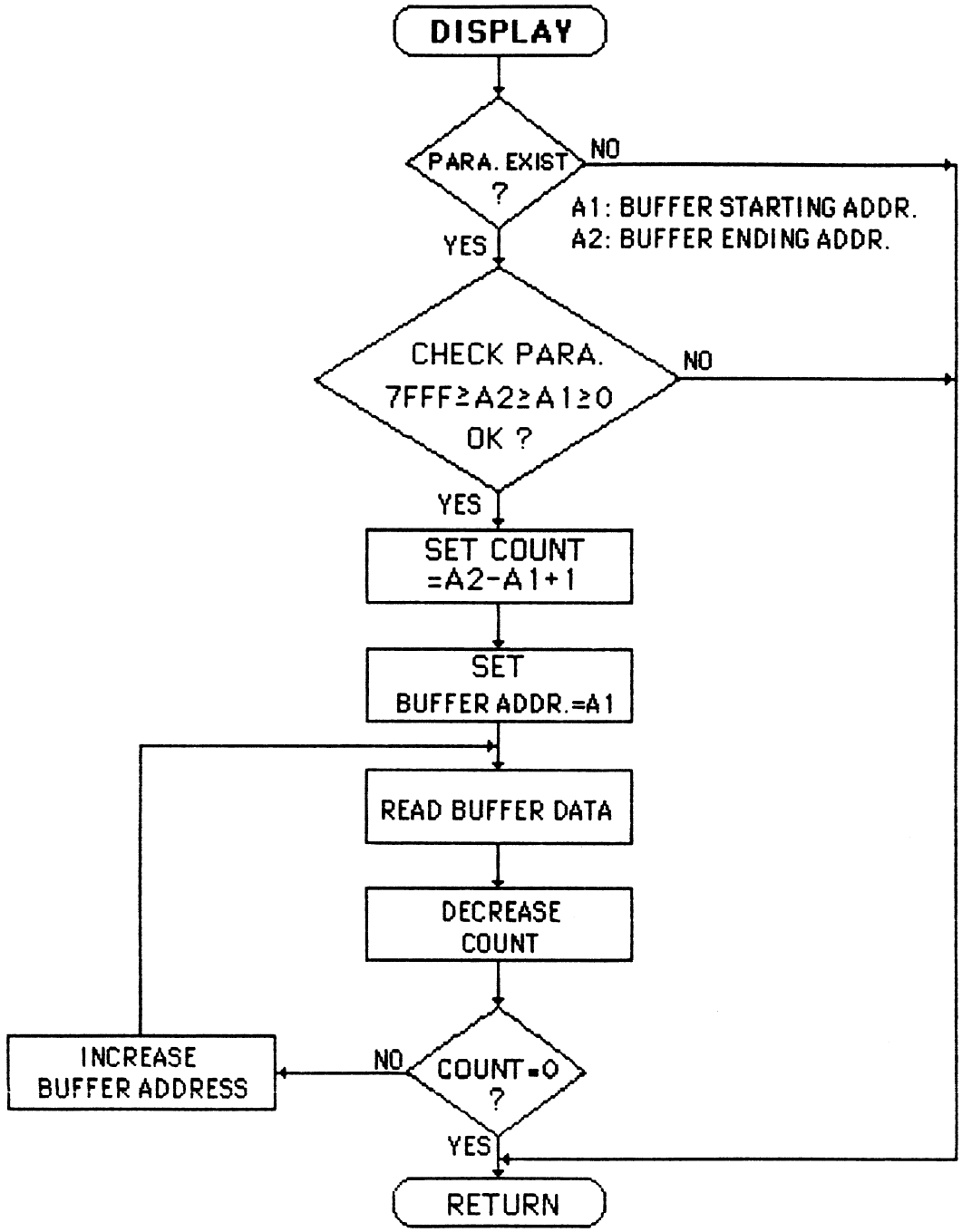
```
                jnz     pc21            ; jump if the data byte not equal to 0
                jmp     nextbyte4       ; jump to program next data byte
pc24:
                mov     si,0            ; initial SI (store being programmed data bit)
rotate4:
                ror     prog_byte,1     ; rotate right one bit to obtain programming data bit
                jc      pbit4           ; jump if carry=1
                jmp     nextbit4        ; jump to program next data bit
pbit4:
                mov     retrycnt,10     ; set retry count=10 (NO. of loops)
                mov     verifybit,0     ; clear program verify flag
                mov     di,si
                add     di,di
                mov     di,ptype4[di]
                mov     al,[di]
                mov     v12_data,al
                mov     ax,[di+1]
                mov     v12_adr,ax      ; request exact program voltage (DG201)
retry4:
; ---- <a> ----       disable PROM by setting OE high -----
                mov     al,bh
                or      al,d_pattern    ; get PROM output enable high
                mov     dx,actl4b
                out     dx,al           ; U14 set , OE high
        ; -----        set program data bit -----
                mov     al ,program_tbl[si]
                mov     dx,dctl4
                out     dx,al           ; U25 set and issues data bit to be programmed
; ----<b> ----        set PROM Vcc input +12 V -----
                mov     al,01000000b
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input =12V
; ----<c> ----        set exact program data bit (Qi) +12 V -----
                mov     al,v12_data
                mov     dx,v12_adr
                out     dx,al           ; DG201 set to supply program voltage
; ----<d> ----        set PROM OE enable ------
                mov     al,bh
                or      al,e_pattern    ; get PROM output enables low
                mov     dx,actl4b
                out     dx,al           ; U14 set and issues address (A0 - A7) for PROM
; ----<e> ----        disable PROM by setting OE high -----
                mov     al,bh
                or      al,d_pattern    ; get PROM output enables (OE) high
                mov     dx,actl4b
                out     dx,al           ; U14 set, OE high
        ; -----        set program bit (Qi) +5 V -----
                mov     al,11111111b
                mov     dx,v12_adr
                out     dx,al           ; disable DG201 to remove +12V from PROM output
        ; -----        set PROM Vcc input +5 V -----
                mov     al,00000000b
```

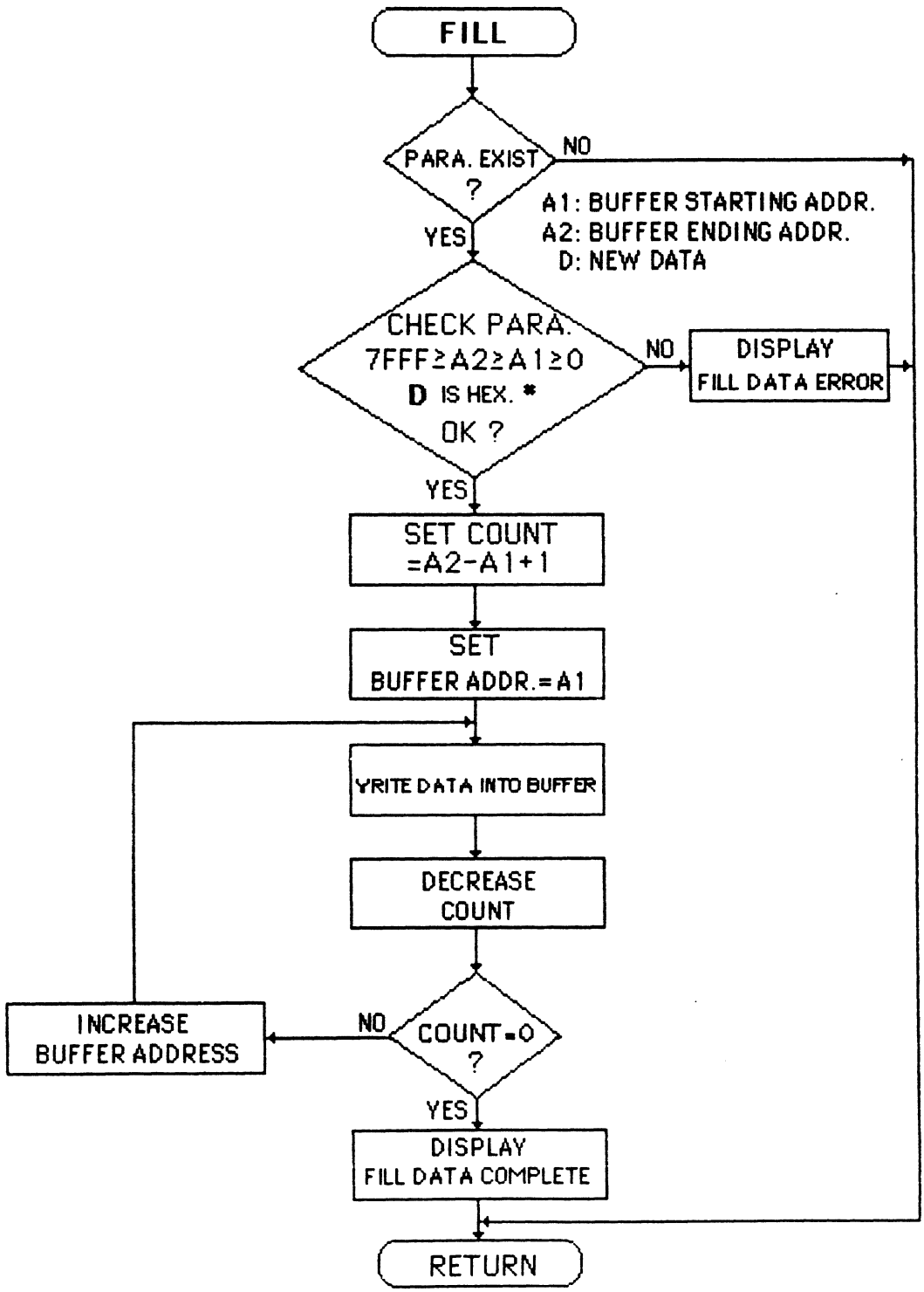
```
                mov     dx,epctl
                out     dx,al           ; U35 set and PROM Vcc input =5V
        ; -----         set up to read PROM ------
                mov     al,0ffh
                mov     dx,dctl4        ; set all ones to turn off 7407
                out     dx,al           ; U25 set and sets up to read data from PROM
        ; -----         set PROM OE enable ------
                mov     al,bh
                or      al,e_pattern    ; get PROM output enables low
                mov     dx,actl4b
                out     dx,al           ; U19 set and issues address (A0 - A7) for PROM
        ; -----         read data from PROM-----
                mov     dx,cctl4
                in      al,dx           ; U31 set and read data from PROM
                cmp     verifybit,0     ; compare PROM data with 0
                jz      no_verify4      ; jump if the data bit is equal to 0 (not verified)
                dec     verifycnt       ; decrease verify count (initial value=5)
                jz      nextbit4        ; jump if  the verify count is equal to 0
                                        ; and to program next bit
                jmp     retry4          ; loop for 10 times
no_verify4:                             ; loop for retry count
                test    al,program_tbl[si] ; get program data bit which will be programmed
                jnz     vfbitok4        ; jump if not equal 0 (verify data bit ok)
                dec     retrycnt        ; decrease retry count (initial value=10)
                jz      vfbitfail4      ; jump if retry count is equal to 0
                jmp     retry4          ; jump to reprogram the data bit
vfbitok4:
                mov     verifycnt,5     ; set additional 5 pulse for programming
                mov     verifybit,1     ; bit verify ok (default, 1 : success)
                jmp     retry4          ; jump to reprogram the data bit
vfbitfail4:
                mov     _verifyok,0     ; verify fail (programming fail)
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
nextbit4:
                inc     si              ; increase programmed data bit count
                cmp     si,8            ; compare the programming data bit with 8
                jz      nextbyte4       ; jump if the data byte (8 bits) has been programmed
                jmp     rotate4         ; jump to program next data bit
nextbyte4:
                mov     ah,0            ; get low byte AL
                mov     al,prog_byte    ; store program byte
                add     chksum,ax       ; add checksum value
                inc     bx              ; incerase PROM address
                mov     _dspadr,bx      ; store display address
                inc     buf_ptr         ; increase buffer memory address
                dec     cx              ; decrease byte count which is to be programmed
                jz      end4            ; jump if byte count is equal to 0
                jmp     proceed4        ; jump to program next byte
```

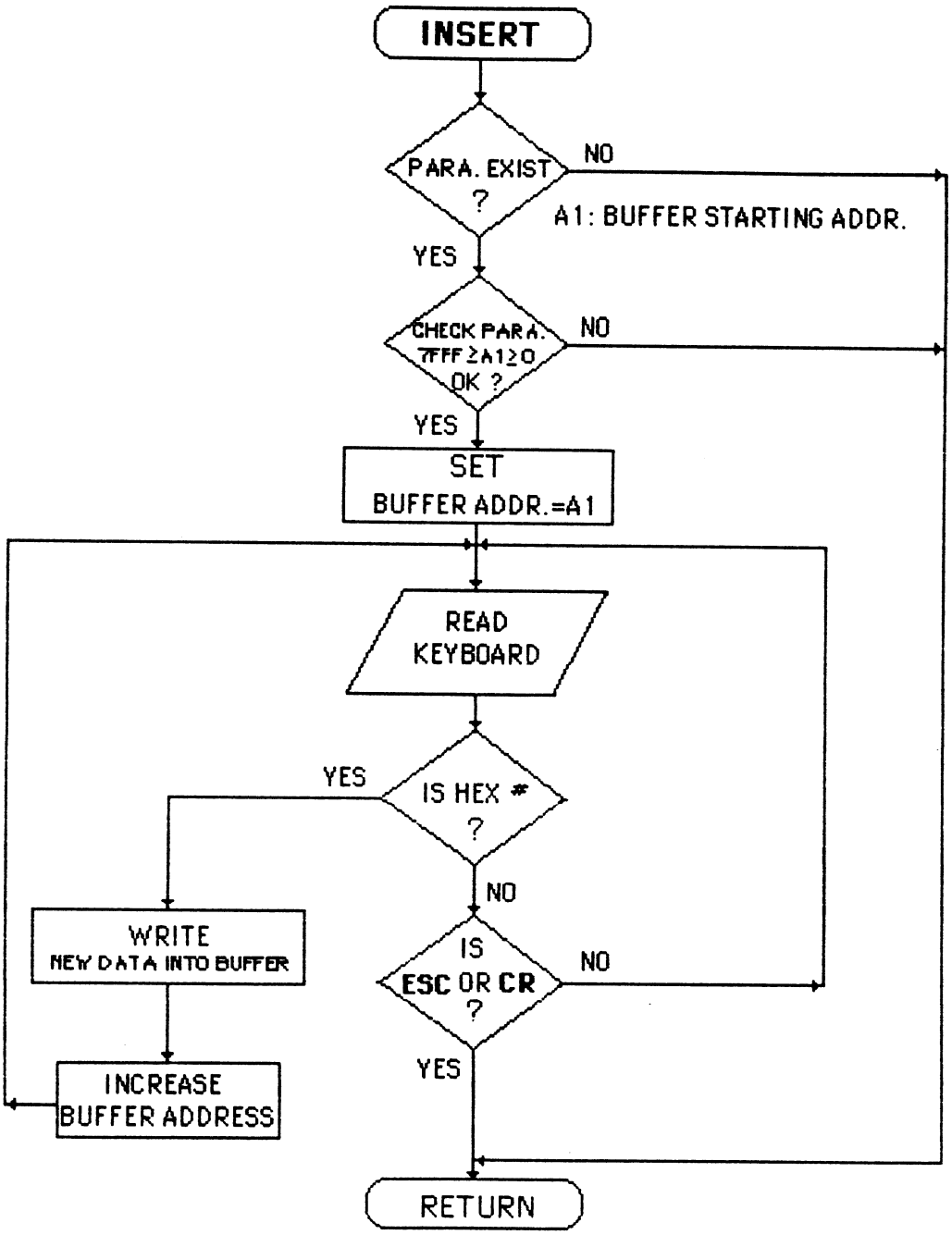
```
end4:
                mov     al,0fh
                mov     dx,typectl
                out     dx,al           ; U39 set and disables all PROM inputs
                mov     ax,chksum       ; get checksum value
                pop     si              ; restore source index (SI)
                pop     di              ; restore destination index (DI)
                pop     bp              ; restore base pointer (BP)
                ret                     ; return to calling routine
_program4       endp
_TEXT           ends
                end
```

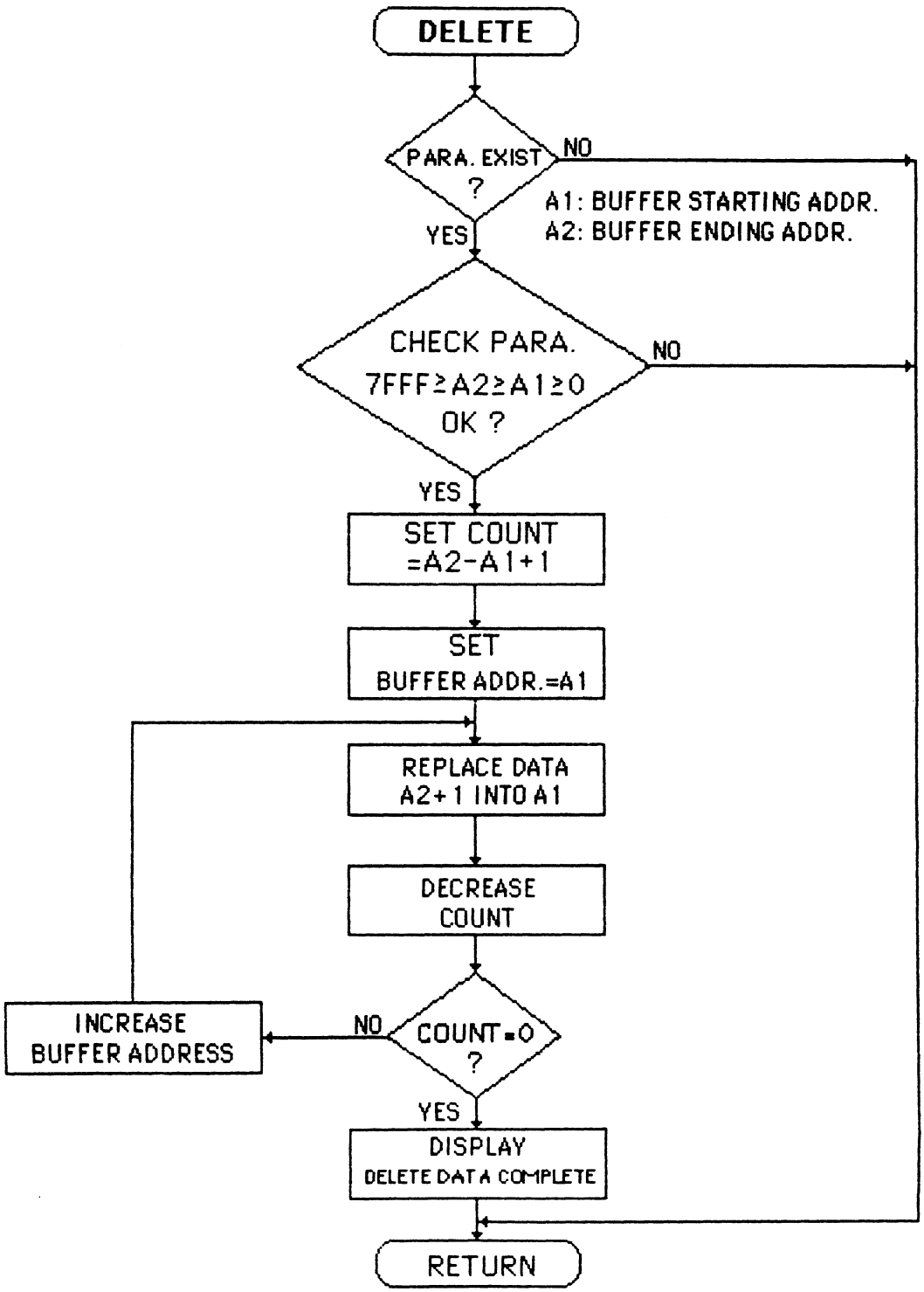
APPENDIX B

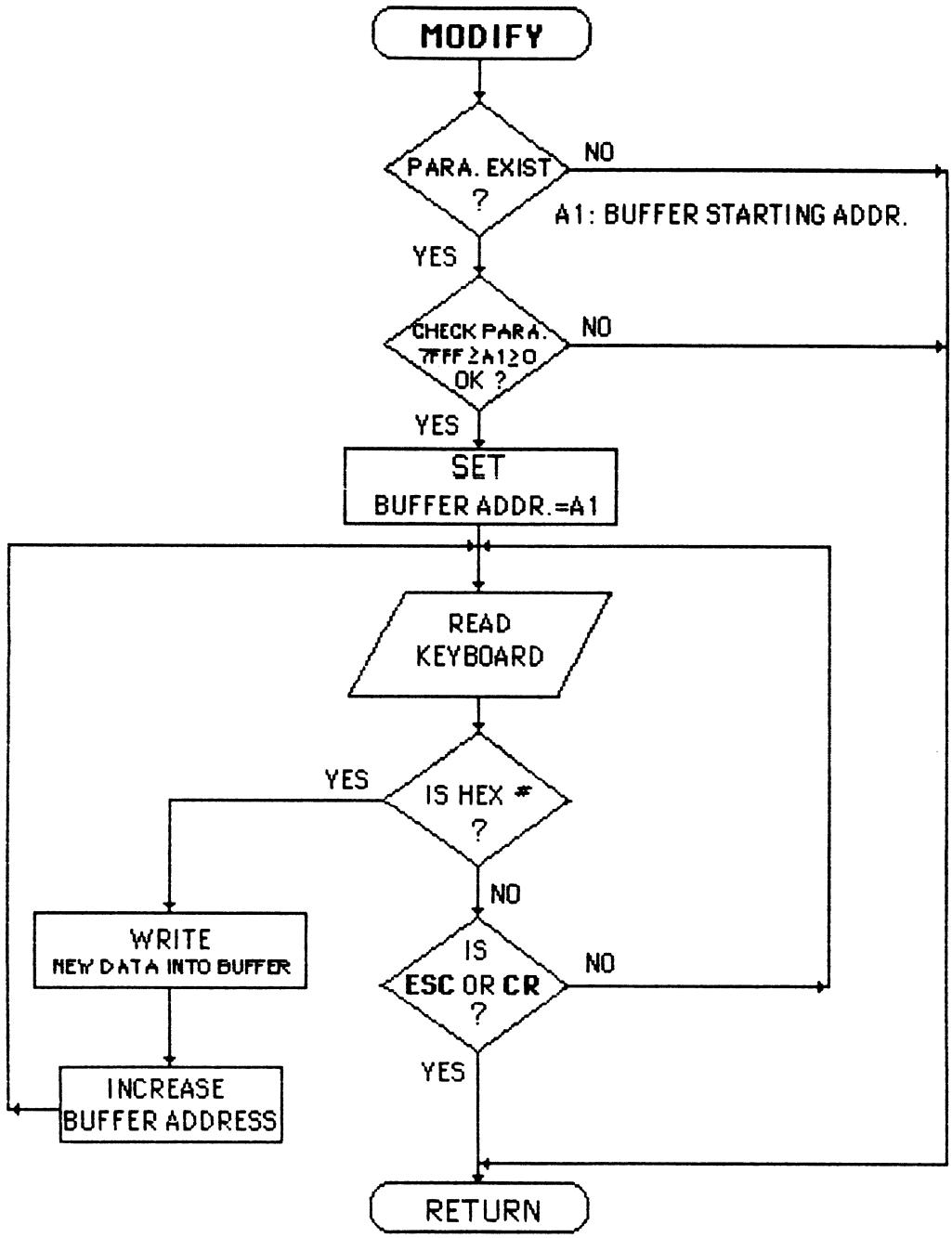

CHECKSUM
PARA. EXIST ?
NO
YES
A1: BUFFER STARTING ADDR.
A2: BUFFER ENDING ADDR.
A3: CHECKSUM STORED ADDR. (OPTION)
CHECK PARA:
7FFF≥A2≥A1≥0
7FFF≥A3≥0
OK ?
NO
YES
SET COUNT =A2-A1+1
SET BUFFER ADDR.=A1
SET CHECKSUM=0
READ BUFFER DATA
ADD CHECKSUM
DECREASE COUNT
COUNT=0 ?
NO
INCREASE BUFFER ADDRESS
YES
PARA. A3 EXIST ?
NO
YES
STORE CHECKSUM
DISPLAY CHECKSUM
COPY

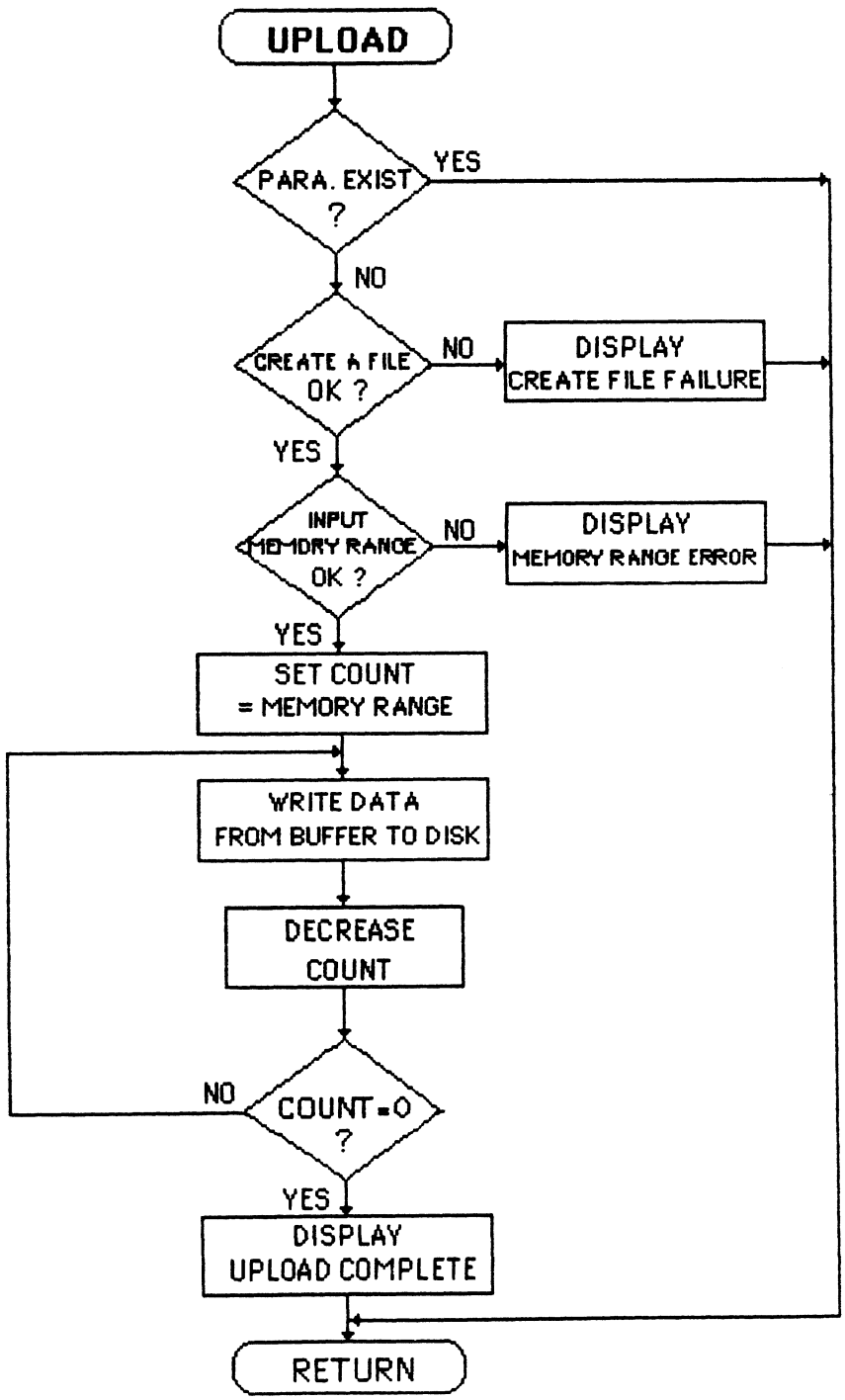

DISPLAY
PARA. EXIST ?
NO
A1: BUFFER STARTING ADDR.
A2: BUFFER ENDING ADDR.
YES
CHECK PARA.
7FFF≥A2≥A1≥0
OK ?
NO
YES
SET COUNT
=A2-A1+1
SET
BUFFER ADDR.=A1
READ BUFFER DATA
DECREASE
COUNT
COUNT=0
?
NO
INCREASE
BUFFER ADDRESS
YES
RETURN

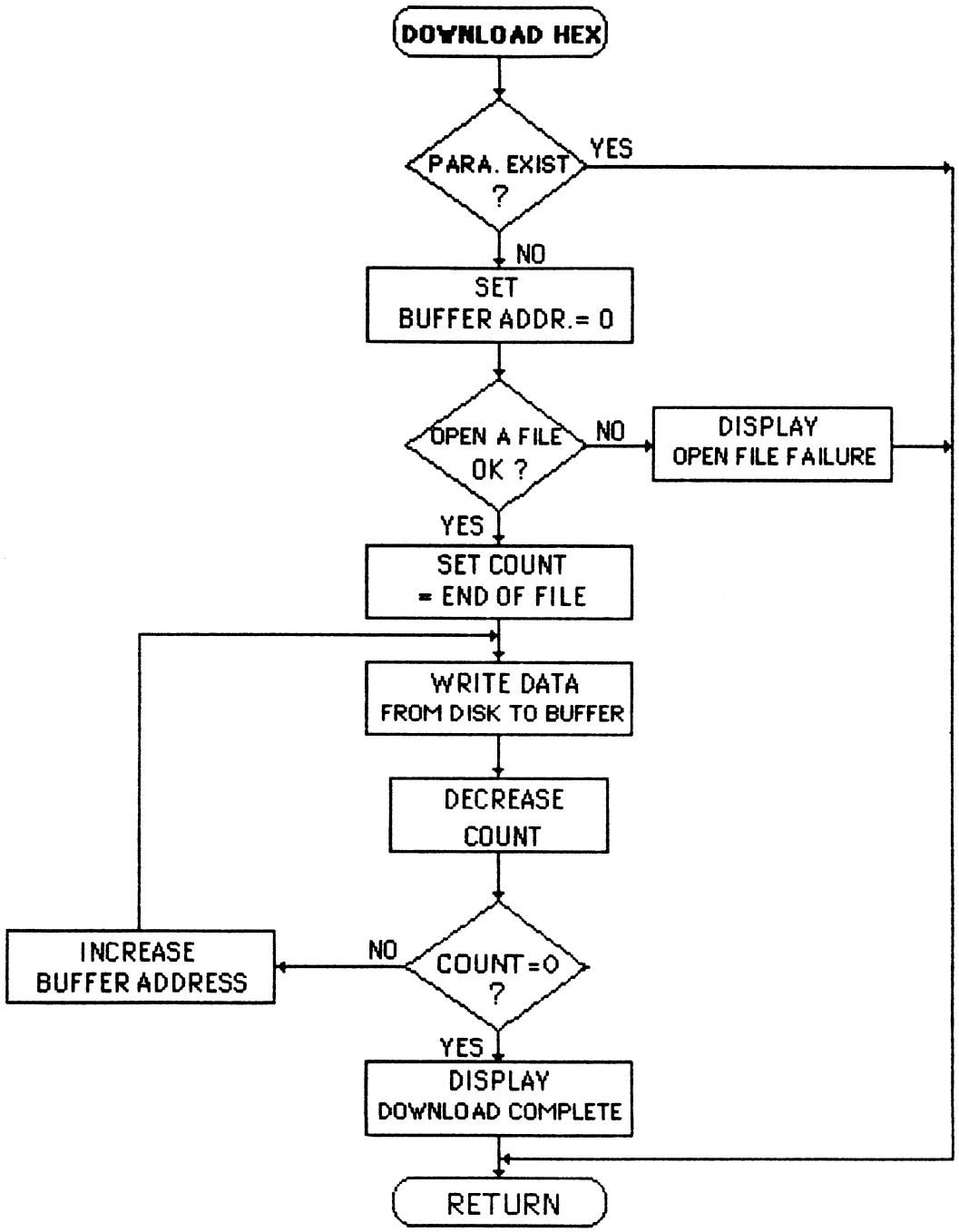

FILL
PARA. EXIST ?
NO
YES
A1: BUFFER STARTING ADDR.
A2: BUFFER ENDING ADDR.
D: NEW DATA
CHECK PARA.
7FFF≥A2≥A1≥0
D IS HEX. *
OK ?
NO
DISPLAY
FILL DATA ERROR
YES
SET COUNT
=A2-A1+1
SET
BUFFER ADDR.=A1
YRITE DATA INTO BUFFER
DECREASE
COUNT
INCREASE
BUFFER ADDRESS
NO
COUNT=0
?
YES
DISPLAY
FILL DATA COMPLETE
RETURN

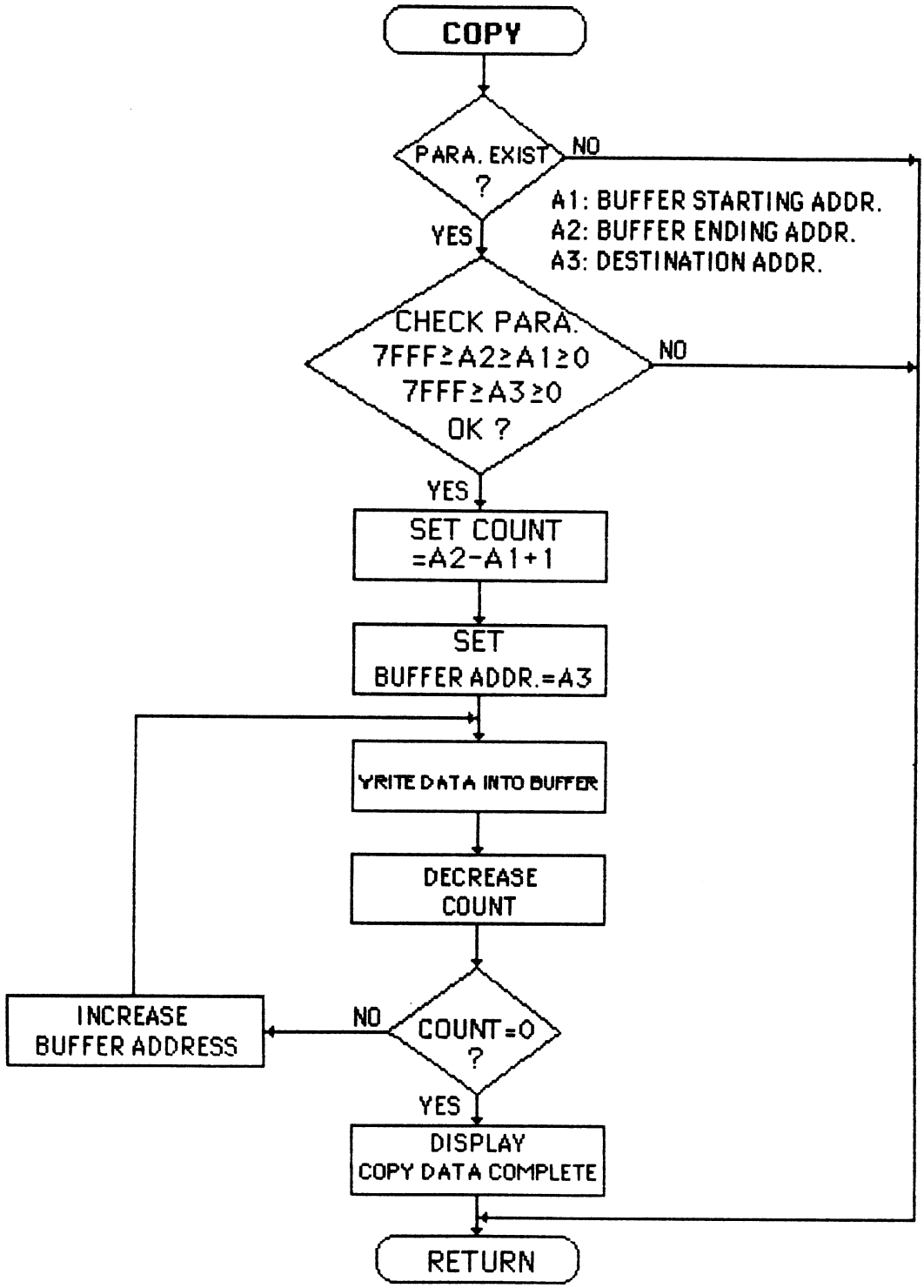

INSERT
PARA. EXIST ?
NO
A1: BUFFER STARTING ADDR.
YES
CHECK PARA.
7FFF≥A1≥0
OK ?
NO
YES
SET
BUFFER ADDR.=A1
READ
KEYBOARD
YES
IS HEX # ?
NO
WRITE
NEW DATA INTO BUFFER
IS
ESC OR CR
?
NO
YES
INCREASE
BUFFER ADDRESS
RETURN

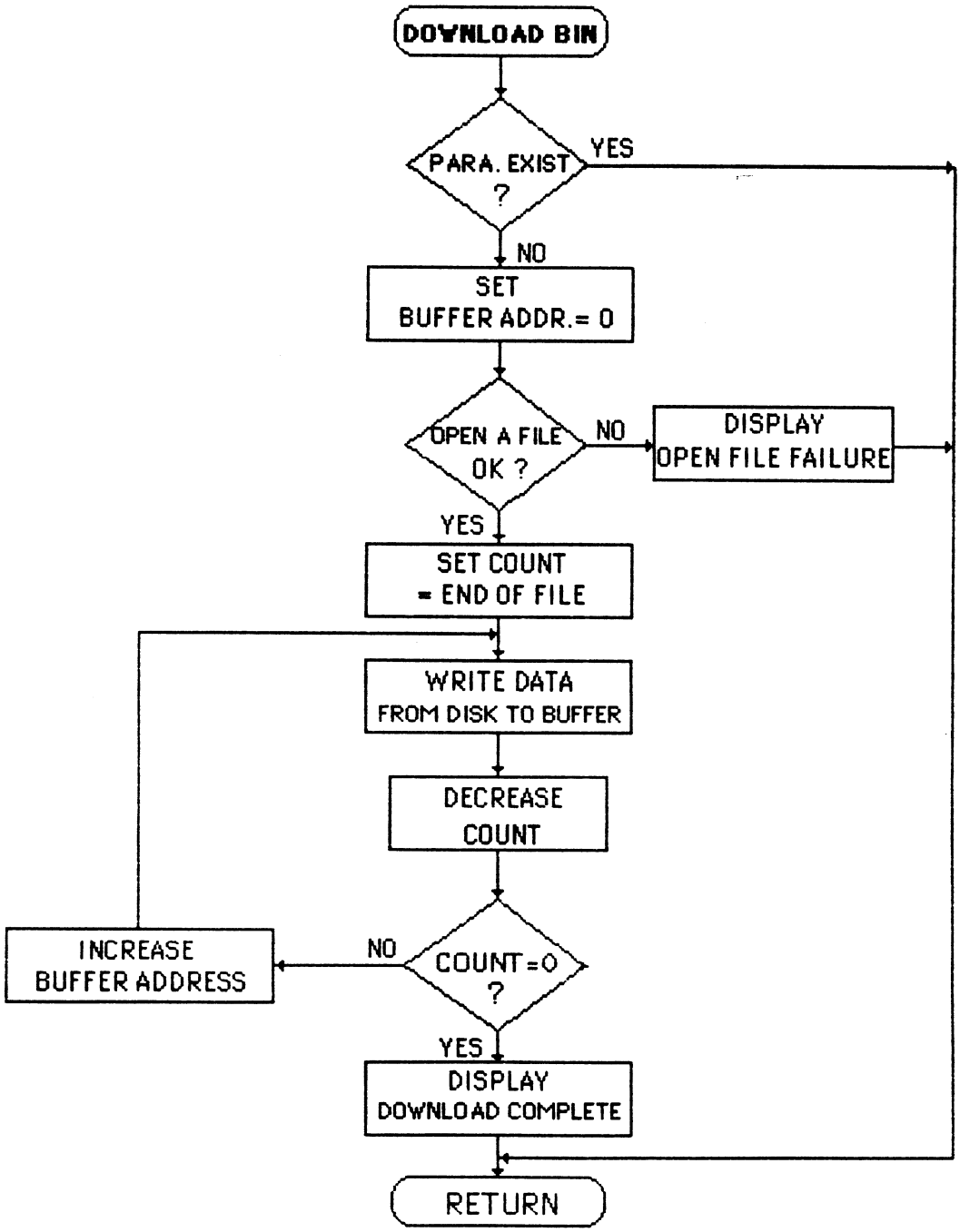

DELETE
PARA. EXIST ?
NO
YES
A1: BUFFER STARTING ADDR.
A2: BUFFER ENDING ADDR.
CHECK PARA.
7FFF≥A2≥A1≥0
OK ?
NO
YES
SET COUNT
=A2-A1+1
SET
BUFFER ADDR.=A1
REPLACE DATA
A2+1 INTO A1
DECREASE
COUNT
COUNT=0
?
NO
INCREASE
BUFFER ADDRESS
YES
DISPLAY
DELETE DATA COMPLETE
RETURN

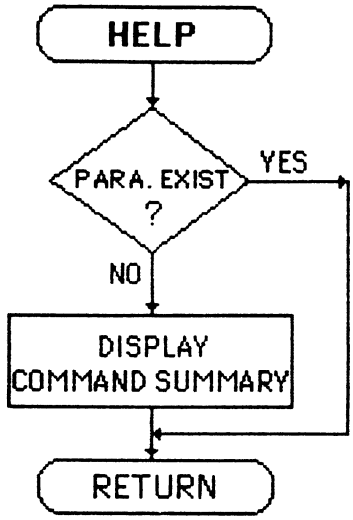

MODIFY
PARA. EXIST ?
NO
A1: BUFFER STARTING ADDR.
YES
CHECK PARA. 7FFF≥A1≥0 OK ?
NO
YES
SET BUFFER ADDR.=A1
READ KEYBOARD
YES
IS HEX # ?
NO
WRITE NEW DATA INTO BUFFER
IS ESC OR CR ?
NO
INCREASE BUFFER ADDRESS
YES
RETURN

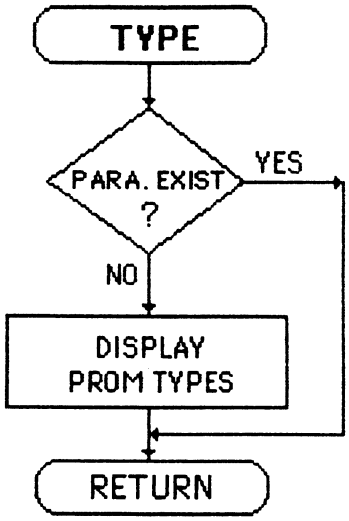

UPLOAD
PARA. EXIST ?
YES
NO
CREATE A FILE OK ?
NO
DISPLAY CREATE FILE FAILURE
YES
INPUT MEMORY RANGE OK ?
NO
DISPLAY MEMORY RANGE ERROR
YES
SET COUNT = MEMORY RANGE
WRITE DATA FROM BUFFER TO DISK
DECREASE COUNT
COUNT=0 ?
NO
YES
DISPLAY UPLOAD COMPLETE
RETURN

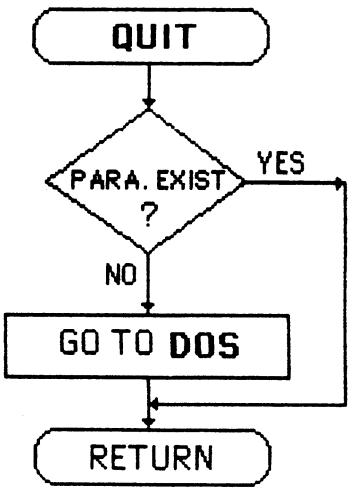

DOWNLOAD HEX
PARA. EXIST ?
YES
NO
SET BUFFER ADDR.= 0
OPEN A FILE OK ?
NO
DISPLAY OPEN FILE FAILURE
YES
SET COUNT = END OF FILE
WRITE DATA FROM DISK TO BUFFER
DECREASE COUNT
INCREASE BUFFER ADDRESS
NO
COUNT=0 ?
YES
DISPLAY DOWNLOAD COMPLETE
RETURN


COPY
PARA. EXIST ?
NO
YES
A1: BUFFER STARTING ADDR.
A2: BUFFER ENDING ADDR.
A3: DESTINATION ADDR.
CHECK PARA.
7FFF≥A2≥A1≥0
7FFF≥A3≥0
OK ?
NO
YES
SET COUNT
=A2-A1+1
SET
BUFFER ADDR.=A3
WRITE DATA INTO BUFFER
DECREASE
COUNT
INCREASE
BUFFER ADDRESS
NO
COUNT=0
?
YES
DISPLAY
COPY DATA COMPLETE
RETURN


DOWNLOAD BIN
PARA. EXIST ?
YES
NO
SET BUFFER ADDR.= 0
OPEN A FILE OK ?
NO
DISPLAY OPEN FILE FAILURE
YES
SET COUNT = END OF FILE
WRITE DATA FROM DISK TO BUFFER
DECREASE COUNT
COUNT=0 ?
NO
INCREASE BUFFER ADDRESS
YES
DISPLAY DOWNLOAD COMPLETE
RETURN


HELP
PARA. EXIST ?
YES
NO
DISPLAY COMMAND SUMMARY
RETURN


TYPE
PARA. EXIST ?
YES
NO
DISPLAY PROM TYPES
RETURN


QUIT
PARA. EXIST ?
YES
NO
GO TO DOS
RETURN

APPENDIX C

HELP Command

?

? is the command keyword for Help.

This command displays a summary of commands and syntax.
Example:

% ?<CR>

COMMAND	DESCRIPTION
B T	BLANK CHECK
CT a1 a2 [a3]	CHECKSUM (TOTAL)
CT a1 a2 [a3]	CHECKSUM (EXOR)
D a1 a2	DISPLAY BUFFER MEMORY
F a1 a2 d	FILL DATA
G T [a1 a2 a3]	PROM PROGRAMMING
I a1	INSERT DATA
K a1 a2	DELETE DATA
M a1	MODIFY DATA
R T [a1 a2 a3]	READ DATA FROM PROM
T	DISPLAY PROM TYPES
Y T [a1 a2 a3]	VERIFY
U	UPLOAD HEX FILE
W	DOWNLOAD HEX FILE
X a1 a2 a3	COPY BUFFER MEMORY
Z	DOWNLOAD BINARY FILE
?	HELP
Q	QUIT TO DOS

%

TYPE Command

T

T is the command keyword for PROM Type.

This command displays all PROM types and size.
Example:
% T<CR>

PROM TYPES	SIZE
74S188/54S288	32X8
74S288/54S288	32X8
74S287/54S287	256X4
74S387/54S387	256X4
74S472/54S472	512X8
74S473/54S473	512X8
74S474/54S474	512X8
74S475/54S475	512X8
74S570/54S570	512X4
74S571/54S571	512X4
74S572/54S572	1024X4
74S573/54S573	1024X4
77S180/87S180	1024X8
77S181/87S181	1024X8
77S184/87S184	2048X8
77S185/87S185	2048X8
77S190/87S190	2048X8
77S191/87S191	2048X8

%

*** MEMORY COMMANDS***

These comamnds allow the user to access and manipulate data in the PC-XT's buffer memory. The address range is from 0000H-7FFFH.

DISPLAY Command

D start-address end-address

D	is the command keyword for Display.
start-address	is a hexadecimal address where the Display operation begins.
end-address	is a hexadecimal address indicating the last memory location of the range to be displayed.

Start- and end-addresses define the display range.

Example: Display the memory contents 0000H to 000FH.

```
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
%
```

FILL Command

F start-address end-address value

F	is the command keyword for Fill.
start-address	is a hexadecimal address where the Fill operation begins.
end-address	is a hexadecimal address indicating the last memory location of the range to be filled.
value	is a hexadecimal value to be written into the specified memory range.

Fill causes a specified value to be written into the defined range (overwritting previous data).

Example: Fill the data range from 0000H to 000FH with a value of 8 and then display the results.

```
% F 0 F 8<CR>
              FILL DATA COMPLETE !
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   08 08 08 08 08 08 08 08 08 08 08 08 08 08 08 08   ................
%
```

INSERT Command

I start-address

I	is the command keyword for Insert.
start-address	is a hexadecimal address where the Insert operation begins.

Enter " I " and the address where data is to be inserted. Input a hexadecimal value and then press <CR> to advance to the next memory address. Specified values are inserted into memory beginning at the start-address. Original data at the loaction where new values are inserted is pushed down in the PC-XT buffer memory. Enter a <CR> or <ESC> without a value to terminate insert mode.

Example: Display the region to be affected, execute the Insert command and then display the results.

```
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00   ................

% I 5<CR>
0005 1<CR>
0006 2<CR>
0007 3<CR>
0004 4<CR>
0009<CR>
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 00 00 00 00 01 02 03 04 00 00 00 00 00 00 00   ................
                      -- -- -- -- --
%
```

KILL Command

K start-address end-address

K	is the command keyword for Kill (delete).
start-address	is a hexadecimal address where the Kill operation begins.
end-address	is a hexadecimal address indicating the last memory location of the range to be deleted.

The contents of the specified range are deleted and data immediately following the end-address is shifted to the start-address (filling in the deleted memory range).

Example: Display the area to be affected, execute the Kill command and then display the results.

```
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   08 08 08 04 05 08 08 09 09 09 00 00 00 00 00 00  ................
% K 4 6<CR>
                DELETE DATAN COMPLETE !
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   08 08 08 04 09 09 09 00 00 00 00 00 00 00 00 00  ................
%
```

COPY Command

X start-address end-address destination-address

X	is the command keyword for Copy.
start-address	is a hexadecimal address where the copy operation begins.
end-address	is a hexadecimal address indicating the last memory location of the range to be copied.
destination-address	is the beginning hexadecimal address in the PC-XT buffer memory where the specified data will be copied.

Data in the specified range is copied to the destination address. Data in the memory range that is copied does not change; while the original data at the destination memory range is overwritten.

Example: Display the area to be affected, perform the Copy command and then display the results.

```
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 01 02 03 04 05 06 07 00 00 00 00 00 00 00 00  ................
% X 1 6 10<CR>
                COPY DATA COMPLETE !
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 01 02 03 04 05 06 07 00 00 00 00 00 00 00 00  ................
0010   00 01 02 03 04 05 06 07 00 00 00 00 00 00 00 00  ................
%
```

MODIFY Command

M start-address

M	is the command keyword for Modify.
start-address	is a hexadecimal address where the Modify operation begins.

When " M " and a start-address is specified, the CRT displays a range (from 00H to 0FH) of memory contents including the start-address. Data can be selectively modified by inputting the start-address and making the necessary changes. To modify data at the present cursor location, enter the desired data and a space bar to move the cursor to the next byte. To move the cursor to the next byte without changing data enter a spqce bar; current data values are echoed. To display the next line simply press a <CR>. Enter <ESC> to exit modification mode and abort any changes made to the current line.

Example: Display buffer memory contents 0000H to 000FH with the Modify command.

```
% M 4<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
                  01 02 03 04 05<CR>
0010   00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00<ESC>
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 00 00 00 01 02 03 04 00 00 00 00 00 00 00 00   ................
%
```

Note: When editing data only enter one or two characters at a time. If the first character is zero, it can be omitted (Example: Entering " A " is the same as entering "0A").

* FUNCTION COMMANDS *

BLANK CHECK Command

B type

B	is the command keyword for Blank Check.
type	is the part number of the device in the socket.

The range checked depends on device type (Example: the range of 74S188 is 0000H to 001FH). Place the device to be checked into the socket and input the Blank Check command. If the memory content of the device is all 00H, it indicates the device is empty.

Example:

```
% B 74S188<CR>
              BLANK CHECK OK !
%
```

CHECKSUM (TOTAL) Command

CT start-address end-address [destination-address]

CT	is the command keyword for Checksum (TOTAL).
start-address	is a hexadecimal address in the PC-XT buffer memory where the Checksum operation begins.
end-address	is a hexadecimal address indicating the last location of the range to be checked.
destination-address	is an optional parameter that stores the checksum value to a specified address in the PC-XT buffer memory.

Checksum is automatically performed on data stored in the PC-XT buffer memory after most operations. It may also be performed at any other time by using the Checksum command. These commands are most useful after download operations, though start- and end-addresses must be specified for checksum commands.

Example: Display the area to be affected, execute the Checksum (TOTAL) command and then display the results.

```
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 01 02 03 04 05 06 07 00 00 00 00 00 00 00 00  ................

% CT 0 F E<CR>

                THE CHECKSUM IS : 001C (TOTAL)

% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 01 02 03 04 05 06 07 00 00 00 00 00 00 00 1C  ................
%
```

DOWNLOAD Command (BINARY file)

Z

Z	is the command keyword for Download (binary file)

Example: To download a binary file 222 from diskette to the PC-XT buffer.

```
% Z<CR>

                DOWNLOAD BINARY FILE TO MEMORY, ENTER FILE NAME : 222
                DOWNLOAD COMPLETE !
%
```

CHECKSUM (EXOR) Command

CX start-address end-address [destination-address]

CX	is the command keyword for Checksum (EXOR).
start-address	is a hexadecimal address in the PC-XT buffer memory where the checksum operation begins.
end-address	is a hexadecimal address indicating the last location of the range to be checked.
destination-address	is an optional parameter that stores the checksum value to a specified address in the PC-XT buffer memory.

Checksum (EXOR) is performed exactly the same as Checksum (TOTAL). The only difference is that a one-byte 2's complement for the sum of the specified data range is provided.

Example: Display the area to be affected, execute the Checksum (EXOR) command and then display the results.

```
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 01 02 03 04 05 06 07 00 00 00 00 00 00 00 00   ................

% CT 0 F B<CR>
                THE CHECKSUM IS : E4 (EXOR)
% D 0 F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   00 01 02 03 04 05 06 07 00 00 00 00 00 00 E4 00   ................
%
```

DOWNLOAD Command (HEX file)

W

W	is the command keyword for Download (HEX file).

Example: Assume there are a hex file " 223 " in the diskette, execute the Download command and then display the results.

```
% W<CR>
                DOWNLOAD HEX FILE TO MEMORY, ENTER FILE NAME:  223

                DOWNLOAD COMPLETE !
% D 0 2F<CR>
       00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
0000   45 83 C4 06 0B C0 75 6D 8B 46 0A 05  06 00 50 57   E.....um.F....PW
0010   B8 FF  7F 50 B8 FF 00 50 8D 46 FE  50  8D 46 FC 50   ...P...P.F.P.F.P
%
```

READ Command

R type [start-address end-address destination-address]

R	is the command keyword for Read.
type	is the part numer of the device in the socket.
start-address	is an optional parameter which is a hexadecimal address in the device's memory specifying the start of the data range to be read.
end-address	is an optional parameter which is a hexadecimal address in the device's memory indicating the last location of the data range to be read.
destination-address	is an optional parameter which is the beginning hexadecimal address in the PC-XT buffer memory where the specified data will be written.

Place the device to be read into the socket and input the Read command. The start-address must be less than the end-address. The start- and end-addresses must be within the device's memory address range. And the destination-address must be within the PC-XT buffer memory address range. Otherwise, an error message displays.

Example:

```
% R 74S188
            READ PROM OK !
            THE CHECKSUM IS : 20 (EXOR)
            THE CHECKSUM IS : 1FE0 (TOTAL)
%
```

VERIFY Command

V type [start-address end-address destination-address]

V	is the command keyword for Verify.
type	is the part numer of the device in the socket.
start-address	is an optional parameter which is a hexadecimal address in the device's memory specifying the start of the data range to be verified.
end-address	is an optional parameter which is a hexadecimal address in the device's memory indicating the last location of the data range to be verified.
destination-address	is an optional parameter which is the beginning hexadecimal address in the PC-XT buffer memory where the specified data will be verified.

A device in the socket can be verified by comparing its memory contents with data in the PC-XT buffer memory. Place a device into the socket; program it according to previous instructions, and then input the Verify command.

Example:

```
% V 74S188
            PROM VERIFY OK !

            THE CHECKSUM IS : 20 (EXOR)
            THE CHECKSUM IS : 1FE0 (TOTAL)
%
```

UPLOAD Command (HEX file)

U

U is the command keyword for Upload.

Example:

```
% U<CR>
              UPLOAD HEX FILE FROM MEMORY, ENTER FILE NAME : 224<CR>
              DISPLAY WHEN IN PROCESSORING (Y/N) : Y
              UPLOAD MEMORY RANGE : 0 2F

: 10000000000102030405060708090A0B0C0D0E0F78
: 10001000101112131415161718191A1B1C1D1E1F68
: 10002000000000000000000000000000000000000000
: 00000001FF
              UPLOAD COMPLETE !
%
```

PROGRAM Command

G type [start-address end-address destination-address]

G	is the command keyword for Program.
type	is the part numer of the device in the socket.
start-address	is an optional parameter which is a hexadecimal address in the PC-XT buffer memory specifying the start of the data range to be programmed.
end-address	is an optional parameter which is a hexadecimal address in the PC-XT buffer indicating the last location of the data range to be programmed.
destination-address	is an optional parameter which is the beginning hexadecimal address in the device memory where the specified data will be programmed.

Place the device to be programmed into the socket and input the Program command. The start-address must be less than the end-address. The start- and end-addresses must be within the PC-XT buffer memory address range. And the destination-address must be within the device's internal memory address range. Otherwise, an error message displays.

Example:

```
% G 74S188<CR>
              PROGRAMMING NOW !...
              001F
              PROGRAMMING OK !
              THE CHECKSUM IS : 20 (EXOR)
              THE CHECKSUM IS : 1FE0 (TOTAL)
%
```

APPENDIX D

I/O Channel Signals

Signal	Function	Direction	Pin
GND	Signal ground	—	B1, B31, B10
RESET DRV	Active high signal to reset system on power-up, synchronized to falling edge of clock	Output	B2
+5 V + 5%	Supply voltage	—	B3, B29
IRQ2–IRQ7	Interrupt request lines 2 through 7. Interrupt request is generated by a low-to-high transition that is held high until acknowledged by processor. Prioritized with decreasing priority from IRQ2 to IRQ7	Input	B4 (IRQ2), B21 (IRQ7) through B25 (IRQ3)
−5 V + 10%	Supply voltage	—	B5
DRQ1–DRQ3	Asynchronous DMA request lines prioritized with decreasing priority from DRQ1 to DRQ3. DMA requested by bringing line high and holding it high until acknowledged by DACK	Input	B18 (DRQ1), B6 (DRQ2), B16 (DRQ3)
−12 V + 10%	Supply voltage	—	B7
Reserved	—	—	B8
+12 V + 5%	Supply voltage	—	B9
MEMW	Active low memory write command that indicates to memory that data present on data bus are to store into memory. May originate with processor or DMA controller	Output	B11
MEMR	Active low memory read command that indicates to memory that it should place data on the data bus to be read. May originate with processor or DMA controller	Output	B12
IOW	Active low I/O write command that indicates to an output device that data are present on the data bus to be read. May originate with processor or DMA controller	Output	B13
IOR	Active low I/O read command that indicates to an output device that it should place data on the data bus to be read. May originate with processor or DMA controller	Output	B14
DACK0–DACK3	Active low DMA request acknowledge lines. DACK0 is used for system memory refresh and DACK1–DACK3 acknowledge DMA requests DRQ1–DRQ3, respectively	Output	B19, B17, B26, B15

(Continued)

Signal	Function	Direction	Pin
CLOCK	4.77-MHz system clock that is derived by dividing oscillator output (pin 30) by 3. Clock period is 210 ns with 33% duty cycle	Output	B20
T/C	Active high terminal count line that will present an output pulse when a terminal count is reached on any DMA channel	Output	B27
ALE	Address Latch Enable line that is used by system board to latch valid addresses generated by the processor. When used with AEN signal, it can identify valid processor addresses. These addresses are latched by using the falling edge of ALE. ALE is generated by the 8288 bus controller	Output	B28
OSC	14.31818 MHz clock oscillator signal with 70-ns period and 50% duty cycle	Output	B30
I/O CH CK	Active low signal that indicates a parity error associated with data in memory or I/O devices	Input	A1
D0–D7	Active high data bits 0–7. D0 is lsb		
I/O CH RDY	I/O channel ready line that is normally high and is pulled low by memory or I/O devices to lengthen a memory or I/O cycle. Pulling this line low extends machine cycles by an integral number of 210-ns clock cycles. This feature allows slow devices to interface with the processor. I/O CH RDY should not be held low more than 10 system clock cycles	Input	A10
AEN	Address enable line that, when it is high, essentially isolates the processor and other devices from I/O channel. Thus DMA controller takes over address bus, data bus, IOR, IOW, MEMR, and MEMW lines to effect DMA transfers	Output	A11
A0–A19	Active high address bus lines. A0 is lsb and A19 is msb.	Output	A31–A1

INTERFACE CIRCUITS

SERIES 55450B/75450B DUAL PERIPHERAL DRIVERS

BULLETIN NO. DL-S 7712424, DECEMBER 1976—REVISED AUGUST 1977

PERIPHERAL DRIVERS FOR HIGH-CURRENT SWITCHING AT HIGH SPEEDS

performance

- Characterized for Use to 300 mA
- High-Voltage Outputs
- No Output Latch-Up at 20 V
- High-Speed Switching

ease-of-design

- Circuit Flexibility for Varied Applications and Choice of Logic Function
- TTL- or DTL-Compatible Diode-Clamped Inputs
- Standard Supply Voltages
- Available in Plastic and Ceramic Packages

SUMMARY OF SERIES 55450/75450

DEVICE	LOGIC OF COMPLETE CIRCUIT	PACKAGES
SN55450B	AND†	J
SN55451B	AND	JG
SN55452B	NAND	JG
SN55453B	OR	JG
SN55454B	NOR	JG
SN75450B	AND†	J, N
SN75451B	AND	JG, P
SN75452B	NAND	JG, P
SN75453B	OR	JG, P
SN75454B	NOR	JG, P

†With output transistor base connected externally to output of gate.

description

Series 55450B/75450B dual peripheral drivers are a family of versatile devices designed for use in systems that employ TTL or DTL logic. The 55450B/75450B family is functionally interchangeable with and replaces the 75450 family and the 75450A family devices manufactured previously. The speed of the 55450B/75450B family is equal to that of the 75450 family and a test to ensure freedom from latch-up has been added. Diode-clamped inputs simplify circuit design. Typical applications include high-speed logic buffers, power drivers, relay drivers, lamp drivers, MOS drivers, line drivers, and memory drivers. Series 55450B drivers are characterized for operation over the full military temperature range of −55°C to 125°C; Series 75450B drivers are characterized for operation from 0°C to 70°C.

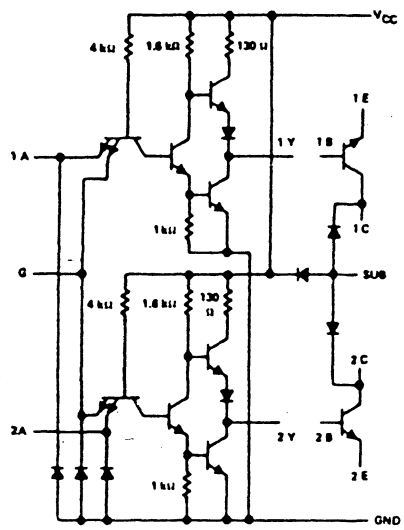
The SN55450B and SN75450B are unique general-purpose devices each featuring two standard Series 54/74 TTL gates and two uncommitted, high-current, high-voltage n-p-n transistors. These devices offer the system designer the flexibility of tailoring the circuit to the application.

The SN55451B/SN75451B, SN55452B/SN75452B, SN55453B/SN75453B, and SN55454B/SN75454B are dual peripheral AND, NAND, OR, and NOR drivers, respectively, (assuming positive logic) with the output of the logic gates internally connected to the bases of the n-p-n output transistors.

CONTENTS

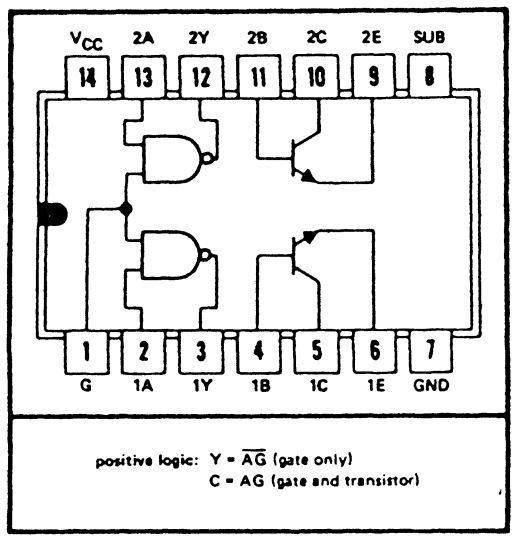
TYPES SN55450B, SN75450B
DUAL PERIPHERAL POSITIVE-AND DRIVERS

schematic



Resistor values shown are nominal.

SN55450B . . . J
SN75450B . . . J OR N
DUAL-IN-LINE PACKAGE (TOP VIEW)



electrical characteristics over recommended operating free-air temperature range (unless otherwise noted)

TTL gates

PARAMETER			TEST CONDITIONS†	SN55450B			SN75450B			UNIT
				MIN	TYP‡	MAX	MIN	TYP‡	MAX	
V_{IH}	High-level input voltage			2			2			V
V_{IL}	Low-level input voltage					0.8			0.8	V
V_{IK}	Input clamp voltage		V_{CC} = MIN, I_I = −12 mA		−1.2	−1.5		−1.2	−1.5	V
V_{OH}	High-level output voltage		V_{CC} = MIN, V_{IL} = 0.8 V, I_{OH} = −400 μA	2.4	3.3		2.4	3.3		V
V_{OL}	Low-level output voltage		V_{CC} = MIN, V_{IH} = 2 V, I_{OL} = 16 mA		0.25	0.5		0.25	0.4	V
I_I	Input current at maximum input voltage	input A	V_{CC} = MAX, V_I = 5.5 V			1			1	mA
		input G				2			2	
I_{IH}	High-level input current	input A	V_{CC} = MAX, V_I = 2.4 V			40			40	μA
		input G				80			80	
I_{IL}	Low-level input current	input A	V_{CC} = MAX, V_I = 0.4 V			−1.6			−1.6	mA
		input G				−3.2			−3.2	
I_{OS}	Short-circuit output current§		V_{CC} = MAX	−18	−35	−55	−18	−35	−55	mA
I_{CCH}	Supply current, outputs high		V_{CC} = MAX, V_I = 0		2.8	4		2.8	4	mA
I_{CCL}	Supply current, outputs low		V_{CC} = MAX, V_I = 5 V		7	11		7	11	mA

† For conditions shown as MIN or MAX, use the appropriate value specified under recommended operating conditions.
‡ All typical values at V_{CC} = 5 V, T_A = 25°C.
§ Not more than one output should be shorted at a time.

SERIES 55450B/75450B
DUAL PERIPHERAL DRIVERS

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

		SN55450B	SN55451B SN55452B SN55453B SN55454B	SN75450B	SN75451B SN75452B SN75453B SN75454B	UNIT
Supply voltage, V_{CC} (see Note 1)		7	7	7	7	V
Input voltage		5.5	5.5	5.5	5.5	V
Interemitter voltage (see Note 2)		5.5	5.5	5.5	5.5	V
V_{CC}-to-substrate voltage		35		35		V
Collector-to-substrate voltage		35		35		V
Collector-base voltage		35		35		V
Collector-emitter voltage (see Note 3)		30		30		V
Emitter-base voltage		5		5		V
Off-state output voltage			30		30	V
Continuous collector or output current (see Note 4)		400	400	400	400	mA
Peak collector or output current ($t_w \leq 10$ ms, duty cycle $\leq 50\%$, see Note 4)		500	500	500	500	mA
Continuous total dissipation at (or below) 25°C free-air temperature (see Note 5)	J package	1375		1025		mW
	JG package		1050		825	
	N package			1150		
	P package				1000	
Operating free-air temperature range		−55 to 125	−55 to 125	0 to 70	0 to 70	°C
Storage temperature range		−65 to 150	−65 to 150	−65 to 150	−65 to 150	°C
Lead temperature 1/16 inch from case for 60 seconds	J or JG package	300	300	300	300	°C
Lead temperature 1/16 inch from case for 10 seconds	N or P package	260	260	260	260	°C

NOTES: 1. Voltage values are with respect to network ground terminal unless otherwise specified.
2. This is the voltage between two emitters of a multiple-emitter transistor.
3. This value applies when the base-emitter resistance (R_{BE}) is equal to or less than 500 Ω.
4. Both halves of these dual circuits may conduct rated current simultaneously; however, power dissipation averaged over a short time interval must fall within the continuous dissipation rating.
5. For operation above 25°C free-air temperature, refer to Dissipation Derating Curves in the Thermal Information Section, which starts on page 4-21. In the J and JG packages, SN55450B through SN55454B chips are alloy-mounted; SN75450B through SN75454B chips are glass-mounted.

recommended operating conditions (see Note 6)

	SERIES 55450B			SERIES 75450B			UNIT
	MIN	NOM	MAX	MIN	NOM	MAX	
Supply voltage, V_{CC}	4.5	5	5.5	4.75	5	5.25	V
Operating free-air temperature, T_A	−55		125	0		70	°C

NOTE 6: For the SN55450B and SN75450B only, the substrate (pin 8) must always be at the most-negative device voltage for proper operation.

APPENDIX F

DG201

MILITARY (A SUFFIX) −55 to +125°C
INDUSTRIAL (B SUFFIX) −20 to +85°C
COMMERCIAL (C SUFFIX) 0 to +70°C

ANALOG SWITCHES

DG201 QUAD SPST CMOS ANALOG TRANSMISSION GATE

MONOLITHIC CMOS SWITCH WITH DRIVER

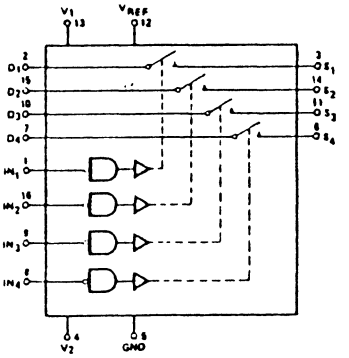
Features

- ±15 V Analog Signal Range
- ±15 V Supplies
- r_{DS} < 250 Ohms Over Full Temperature and Signal Range
- Break-Before-Make Switching Action
- TTL, DTL, and CMOS Direct Control Interface Over Military Temperature Range Without Need For Interface Components
- All Terminals Have Protective Circuitry to Prevent Static Damage to Gates

Description

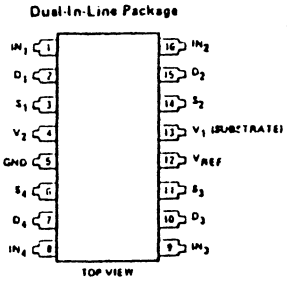
The DG201 is a quad single pole, single throw analog switch which employs a parallel combination of a PMOS and a NMOS field effect transistor. In the ON condition each switch will conduct current in either direction, and in the OFF condition, each switch will block voltages up to 30 V peak-to-peak. The ON-OFF state of each switch is controlled by a driver. With logic "0" at the input the switch will be ON, with logic "1" at the input the switch will be OFF. The logic input will recognize voltages between 0 and 0.8 V as logic "0" voltages, and voltages between 2.4 and 15 V as logic "1" voltages. The input can thus be directly interfaced with TTL, DTL, RTL, CMOS and certain special PMOS circuits. Switch action is break-before-make.

FUNCTIONAL DIAGRAM



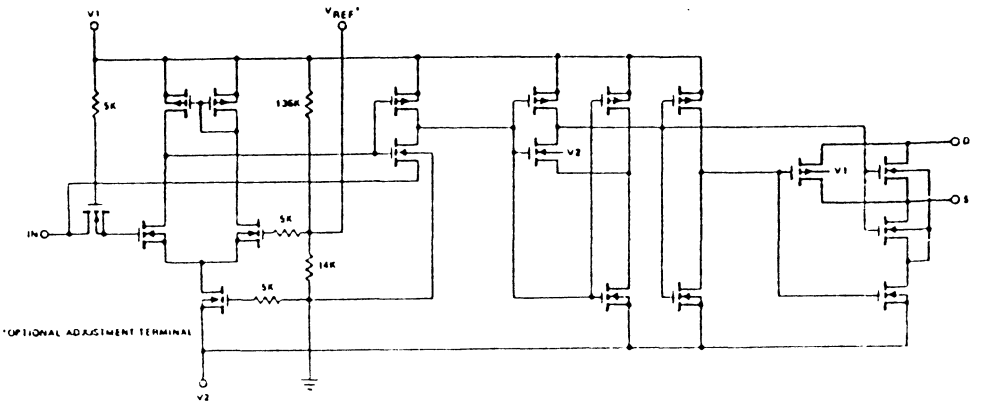
SWITCH OPEN FOR LOGIC "1" INPUT (V_{INH})
(POSITIVE LOGIC)

PIN CONFIGURATION



ORDER NUMBERS: DG201AK OR DG201BK
SEE PACKAGE 10
DG201CJ
SEE PACKAGE 8

SCHEMATIC DIAGRAM (Typical Channel)



ABSOLUTE MAXIMUM RATINGS

V_{IN} and V_{REF} to Ground ..... −0.3 V, V_1
V_S or V_D to V_1 ..... 0, −32 V
V_S or V_D to V_2 ..... 0, 32 V
V_1 to Ground ..... 16 V
V_2 to Ground ..... −16 V
Current, Any Terminal Except S or D ..... 20 mA
Current, S or D ..... 2 mA
Operating Temperature (A & B Suffix).... −55 to 125°C
(C Suffix)..... 0 to 70°C
Storage Temperature (A & B Suffix) ..... −65 to 150°C
(C Suffix)..... −65 to 150°C

Power Dissipation*
16 Pin CERDIP**..... 450 mW
16 Pin Plastic DIP***..... 470 mW
Thermal Resistance
(θ_{JA}, J package Suffix)..... 0.16°C/mW

*Device mounted with all leads soldered or welded to PC board.
**Derate 6 mW/°C above 75°C.
***Derate 6.5 mW/°C above 25°C.

ELECTRICAL CHARACTERISTICS

All DC parameters are 100% tested at 25°C. Lots are sample-tested for AC parameters and high and low temperature limits to assure conformance with specifications.

		CHARACTERISTIC		TYPI 25 C	MAX LIMITS DG201AK −55 C	DG201AK 25 C	DG201AK 125 C	DG201BK/CJ −20 C/ 0 C	DG201BK/CJ 25 C	DG201BK/CJ 85 C/ 70 C	UNIT	TEST CONDITIONS, UNLESS NOTED: V_1 = 15 V, V_2 = −15 V, Gnd = 0, V_{REF} = Open***	
1	SWITCH	$r_{DS(on)}$	Drain Source ON Resistance	130	175	175	250	200	200	250	Ω	V_D = 10 V	V_{IN} = 0.8 V, I_S = −1 mA
2				130	175	175	250	200	200	250		V_D = −10 V	
3		$I_{S(off)}$	Source OFF Leakage Current	0.02		1	600		5	250	nA	V_S = 15 V, V_D = −15 V	V_{IN} = 2.4 V
4				−0.02		−1	−600		−5	−250		V_S = −15 V, V_D = 15 V	
5		$I_{D(off)}$	Drain OFF Leakage Current	0.02		1	500		5	250		V_D = 15 V, V_S = −15 V	
6				−0.02		−1	−500		−5	−250		V_D = −15 V, V_S = 15 V	
7		$I_{D(on)}$*	Drain ON Leakage Current	0.05		1	500		5	250		V_D = V_S = 15 V	V_{IN} = 0.8 V
8				−0.05		−1	−500		−5	−250		V_D = V_S = −15 V	
9	INPUT	I_{INH}	Input Current, Input Voltage High			−1	−10		−1	−10	μA	V_{IN} = 2.4 V	
10						1	10		1	10		V_{IN} = 15 V	
11		$I_{IN(peak)}$	Peak Input Current Required for Transition	−120								See Curve I_{IN} vs V_{IN}	
12		I_{INL}	Input Current, Input Voltage Low			−1	−10		−1	−10		V_{IN} = 0	
13		t_{on}	Turn-ON Time	520		1000					ns	See Switching Time Test Circuit	
14		t_{off}	Turn-OFF Time	330		500							
15	DYNAMIC	$C_{S(off)}$	Source OFF Capacitance	3							pF	V_S = 0, V_{IN} = 5 V	f = 140K Hz to 1 MHz
16		$C_{D(off)}$	Drain OFF Capacitance	3								V_D = 0, V_{IN} = 5 V	
17		$C_{D(on)}$ + $C_{S(on)}$	Channel ON Capacitance	9								V_D = V_S = 0, V_{IN} = 0	
18		Off Isolation**		80							dB	V_{IN} = 5 V, R_L = 1K Ω, C_L = 10 pF, V_S = 1 VRMS, f = 100K Hz	
19	SUPPLY	I_1	Positive Supply Current	2.2		4.0			4.0		mA	One Channel "ON," V_{IN} = 0	
20		I_2	Negative Supply Current	−2.2		−4.0			−4.0				
21		I_1 Standby	Positive Supply Current	1.6		3.0			3.0			All Channels "OFF," V_{IN} = 5 V	
22		I_2 Standby	Negative Supply Current	−1.6		−3.0			−3.0				

†Typical Values are for DESIGN AID ONLY, not guaranteed and not subject to production testing. ICAP

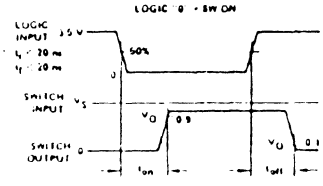
*$I_{D(on)}$ is leakage from driver into "ON" switch.

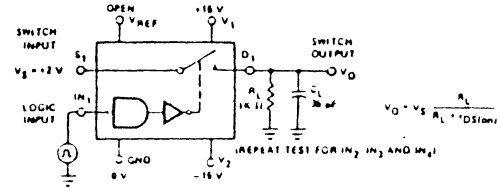
**"OFF" Isolation $\geq 20 \log \frac{|V_A|}{|V_B|}$, A = output terminal of "OFF" switch, B = any other switch terminal

***Functional operation is possible for supply voltages less than 15 V, but the input logic threshold will shift. For V_1 = −V_2 = 10 V, +15 V may be applied to the V_{REF} terminal. The V_{REF} terminal has $R_{IN} \geq 13K$ Ω.

SWITCHING TIME TEST CIRCUIT

Switch output waveform shown for V_S = constant with logic input waveform as shown. Note that V_S may be + or − as per switching time test circuit. V_O is the steady state output with switch on. Feedthrough via gate capacitance may result in spikes at leading and trailing edge of output waveform.





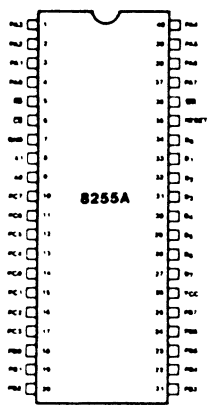
APPENDIX G

8255A/8255A-5
PROGRAMMABLE PERIPHERAL INTERFACE

- MCS-85™ Compatible 8255A-5
- 24 Programmable I/O Pins
- Completely TTL Compatible
- Fully Compatible with Intel® Microprocessor Families
- Improved Timing Characteristics
- Direct Bit Set/Reset Capability Easing Control Application Interface
- 40-Pin Dual In-Line Package
- Reduces System Package Count
- Improved DC Driving Capability

The Intel® 8255A is a general purpose programmable I/O device designed for use with Intel® microprocessors. It has 24 I/O pins which may be individually programmed in 2 groups of 12 and used in 3 major modes of operation. In the first mode (MODE 0), each group of 12 I/O pins may be programmed in sets of 4 to be input or output. In MODE 1, the second mode, each group may be programmed to have 8 lines of input or output. Of the remaining 4 pins, 3 are used for handshaking and interrupt control signals. The third mode of operation (MODE 2) is a bidirectional bus mode which uses 8 lines for a bidirectional bus, and 5 lines, borrowing one from the other group, for handshaking.

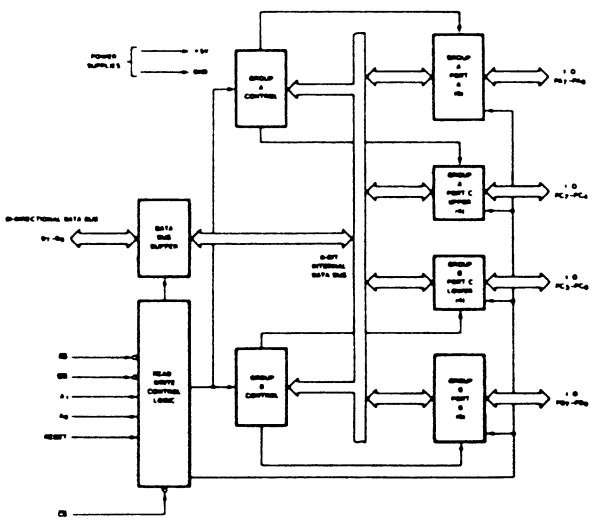
PIN CONFIGURATION



PIN NAMES

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255A BLOCK DIAGRAM



8255A FUNCTIONAL DESCRIPTION

General

The 8255A is a programmable peripheral interface (PPI) device designed for use in Intel® microcomputer systems. Its function is that of a general purpose I/O component to interface peripheral equipment to the microcomputer system bus. The functional configuration of the 8255A is programmed by the system software so that normally no external logic is necessary to interface peripheral devices or structures.

Data Bus Buffer

This 3-state bidirectional 8-bit buffer is used to interface the 8255A to the system data bus. Data is transmitted or received by the buffer upon execution of input or output instructions by the CPU. Control words and status information are also transferred through the data bus buffer.

Read/Write and Control Logic

The function of this block is to manage all of the internal and external transfers of both Data and Control or Status words. It accepts inputs from the CPU Address and Control busses and in turn, issues commands to both of the Control Groups.

($\overline{CS}$)

Chip Select. A "low" on this input pin enables the communiction between the 8255A and the CPU.

($\overline{RD}$)

Read. A "low" on this input pin enables the 8255A to send the data or status information to the CPU on the data bus. In essence, it allows the CPU to "read from" the 8255A.

($\overline{WR}$)

Write. A "low" on this input pin enables the CPU to write data or control words into the 8255A.

(A_0 and A_1)

Port Select 0 and Port Select 1. These input signals, in conjunction with the RD and WR inputs, control the selection of one of the three ports or the control word registers. They are normally connected to the least significant bits of the address bus (A_0 and A_1).

8255A BASIC OPERATION

A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	INPUT OPERATION (READ)
0	0	0	1	0	PORT A – DATA BUS
0	1	0	1	0	PORT B – DATA BUS
1	0	0	1	0	PORT C – DATA BUS
					OUTPUT OPERATION (WRITE)
0	0	1	0	0	DATA BUS – PORT A
0	1	1	0	0	DATA BUS – PORT B
1	0	1	0	0	DATA BUS – PORT C
1	1	1	0	0	DATA BUS – CONTROL
					DISABLE FUNCTION
X	X	X	X	1	DATA BUS – 3-STATE
1	1	0	1	0	ILLEGAL CONDITION
X	X	1	1	0	DATA BUS – 3-STATE

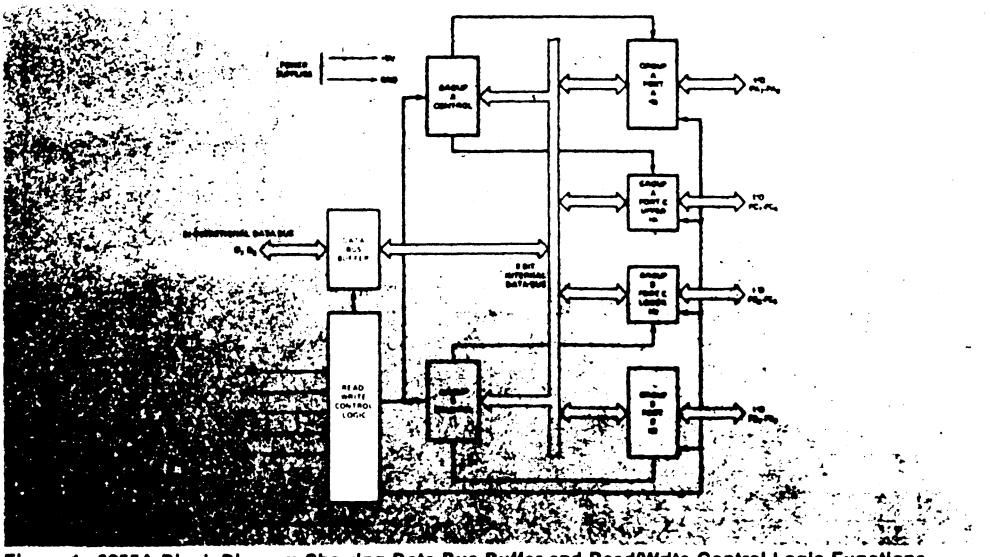


Figure 1. 8255A Block Diagram Showing Data Bus Buffer and Read/Write Control Logic Functions

(RESET)

Reset. A "high on this input clears the control register and all ports (A, C, C) are set to the input mode.

Group A and Group B Controls

The functional configuration of each port is programmed by the systems software. In essence, the CPU "outputs" a control word to the 8255A. The control word contains information such as "mode", "bit set", "bit reset", etc., that initializes the functional configuration of the 8255.

Each of the Control blocks (Group A and Group B) accepts "commands" from the Read/Write Control Logic, receives "control words" from the internal data bus and issues the proper commands to its associated ports.

Control Group A – Port A and Port C upper (C7-C4)
Control Group B – Port B and Port C lower (C3-C0)

The Control Word Register can Only be written into. No Read operation of the Control Word Register is allowed.

Ports A, B, and C

The 8255A contains three 8-bit ports (A, B, and C). All can be configured in a wide variety of functional characteristics by the system software but each has its own special features or "personality" to further enhance the power and flexibility of the 8255A.

Port A. One 8-bit data output latch/buffer and one 8-bit data input latch.

Port B. One 8-bit data input/output latch/buffer and one 8-bit data input buffer.

Port C. One 8-bit data output latch/buffer and one 8-bit data input buffer (no latch for input). This port can be divided into two 4-bit ports under the mode control. Each 4-bit port contains a 4-bit latch and it can be used for the control signal outputs and status signal inputs in conjunction with ports A and B.

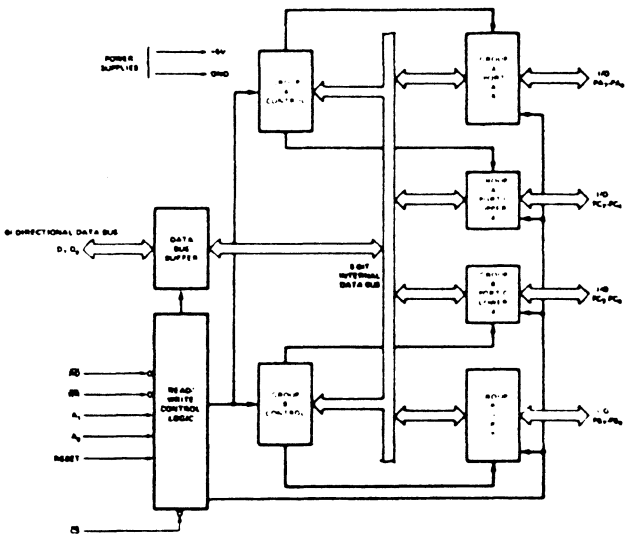
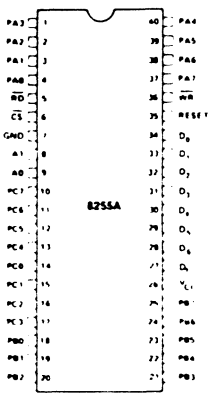


Figure 2. 8225A Block Diagram Showing Group A and Group B Control Functions

PIN CONFIGURATION



PIN NAMES

D_7-D_0	DATA BUS (BI-DIRECTIONAL)
RESET	RESET INPUT
$\overline{CS}$	CHIP SELECT
$\overline{RD}$	READ INPUT
$\overline{WR}$	WRITE INPUT
A0, A1	PORT ADDRESS
PA7-PA0	PORT A (BIT)
PB7-PB0	PORT B (BIT)
PC7-PC0	PORT C (BIT)
V_{CC}	+5 VOLTS
GND	0 VOLTS

8255A OPERATIONAL DESCRIPTION

Mode Selection

There are three basic modes of operation that can be selected by the system software:

Mode 0 – Basic Input/Output
Mode 1 – Strobed Input/Output
Mode 2 – Bi-Directional Bus

When the reset input goes "high" all ports will be set to the input mode (i.e., all 24 lines will be in the high impedance state). After the reset is removed the 8255A can remain in the input mode with no additional initialization required. During the execution of the system program any of the other modes may be selected using a single output instruction. This allows a single 8255A to service a variety of peripheral devices with a simple software maintenance routine.

The modes for Port A and Port B can be separately defined, while Port C is divided into two portions as required by the Port A and Port B definitions. All of the output registers, including the status flip-flops, will be reset whenever the mode is changed. Modes may be combined so that their functional definition can be "tailored" to almost any I/O structure. For instance; Group B can be programmed in Mode 0 to monitor simple switch closings or display computational results, Group A could be programmed in Mode 1 to monitor a keyboard or tape reader on an interrupt-driven basis.

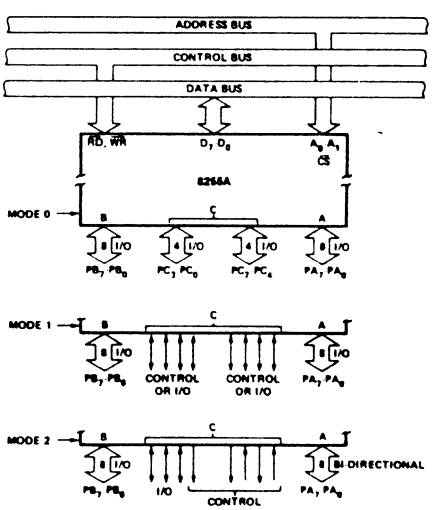


Figure 3. Basic Mode Definitions and Bus Interface

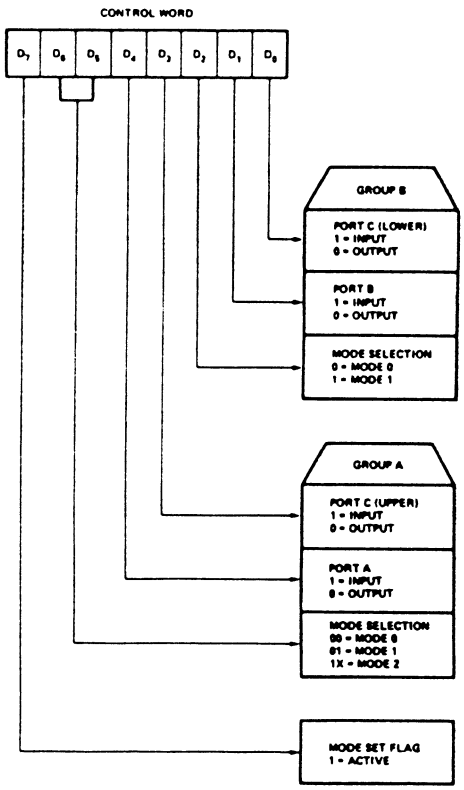


Figure 4. Mode Definition Format

The mode definitions and possible mode combinations may seem confusing at first but after a cursory review of the complete device operation a simple, logical I/O approach will surface. The design of the 8255A has taken into account things such as efficient PC board layout, control signal definition vs PC layout and complete functional flexibility to support almost any peripheral device with no external logic. Such design represents the maximum use of the available pins.

Single Bit Set/Reset Feature

Any of the eight bits of Port C can be Set or Reset using a single OUTput instruction. This feature reduces software requirements in Control-based applications.

Input Control Signal Definition

STB (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F)

A "high" on this output indicates that the data has been loaded into the input latch; in essence, an acknowledgement. IBF is set by STB input being low and is reset by the rising edge of the RD input.

INTR (Interrupt Request)

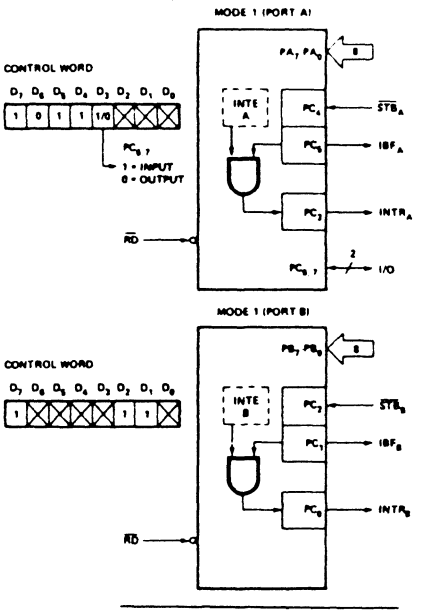
A "high" on this output can be used to interrupt the CPU when an input device is requesting service. INTR is set by the $\overline{STB}$ is a "one", IBF is a "one" and INTE is a "one". It is reset by the falling edge of $\overline{RD}$. This procedure allows an input device to request service from the CPU by simply strobing its data into the port.

INTE A

Controlled by bit set/reset of PC_4.

INTE B

Controlled by bit set/reset of PC_2.



MODE 1 Input

Output Control Signal Definition

$\overline{OBF}$ (Output Buffer Full F/F). The OBF output will go "low" to indicate that the CPU has written data out to the specified port. The OBF F/F will be set by the rising edge of the WR input and reset by ACK input being low.

$\overline{ACK}$ (Acknowledge Input). A "low" on this input informs the 8255A that the data from port A or port B has been accepted. In essence, a response from the peripheral device indicating that it has received the data output by the CPU.

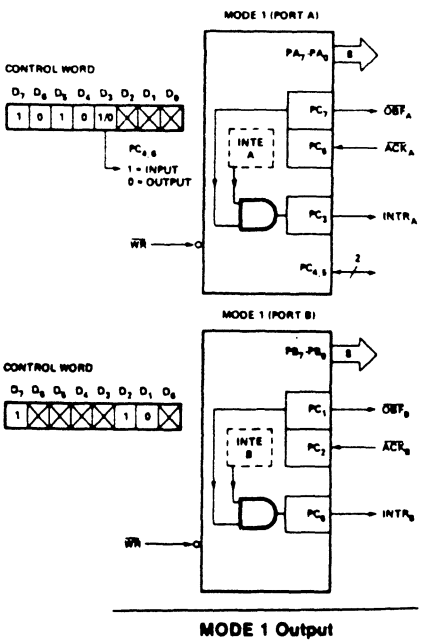
INTR (Interrupt Request). A "high" on this output can be used to interrupt the CPU when an output device has accepted data transmitted by the CPU. INTR is set when ACK is a "one", OBF is a "one" and INTE is a "one". It is reset by the falling edge of WR.

INTE A

Controlled by bit set/reset of PC_6.

INTE B

Controlled by bit set/reset of PC_2.



MODE 1 Output

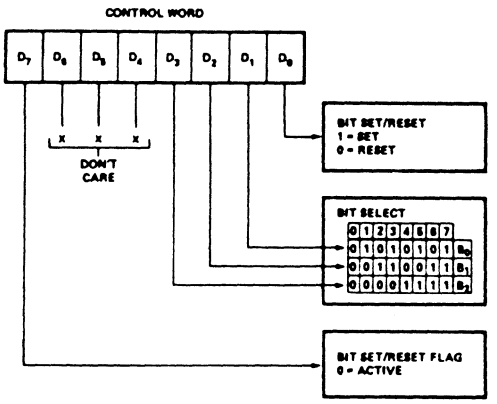


Figure 5. Bit Set/Reset Format

When Port C is being used as status/control for Port A or B, these bits can be set or reset by using the Bit Set/Reset operation just as if they were data output ports.

Interrupt Control Functions

When the 8255A is programmed to operate in mode 1 or mode 2, control signals are provided that can be used as interrupt request inputs to the CPU. The interrupt request signals, generated from port C, can be inhibited or enabled by setting or resetting the associated INTE flip-flop, using the bit set/reset function of port C.

This function allows the Programmer to disallow or allow a specific I/O device to interrupt the CPU without affecting any other device in the interrupt structure.

INTE flip-flop definition:

(BIT-SET) – INTE is SET – Interrupt enable
(BIT-RESET) – INTE is RESET – Interrupt disable

Note: All Mask flip-flops are automatically reset during mode selection and device Reset.

Operating Modes

MODE 0 (Basic Input/Output). This functional configuration provides simple input and output operations for each of the three ports. No "handshaking" is required, data is simply written to or read from a specified port.

Mode 0 Basic Functional Definitions:

- Two 8-bit ports and two 4-bit ports.
- Any port can be input or output.
- Outputs are latched.
- Inputs are not latched.
- 16 different Input/Output configurations are possible in this Mode.

Operating Modes

MODE 1 (Strobed Input/Output). This functional configuration provides a means for transferring I/O data to or from a specified port in conjunction with strobes or "handshaking" signals. In mode 1, port A and Port B use the lines on port C to generate or accept these "handshaking" signals.

Mode 1 Basic Functional Definitions:

- Two Groups (Group A and Group B)
- Each group contains one 8-bit data port and one 4-bit control/data port.
- The 8-bit data port can be either input or output. Both inputs and outputs are latched.
- The 4-bit port is used for control and status of the 8-bit data port.

Operating Modes

MODE 2 (Strobed Bidirectional Bus I/O). This functional configuration provides a means for communicating with a peripheral device or structure on a single 8-bit bus for both transmitting and receiving data (bidirectional bus I/O). "Handshaking" signals are provided to maintain proper bus flow discipline in a similar manner to MODE 1. Interrupt generation and enable/disable functions are also available.

MODE 2 Basic Functional Definitions:

- Used in Group A only.
- One 8-bit, bi-directional bus Port (Port A) and a 5-bit control Port (Port C).
- Both inputs and outputs are latched.
- The 5-bit control port (Port C) is used for control and status for the 8-bit, bi-directional bus port (Port A).

Bidirectional Bus I/O Control Signal Definition

INTR (Interrupt Request). A high on this output can be used to interrupt the CPU for both input or output operations.

Output Operations

$\overline{OBF}$ (Output Buffer Ful). The OBF output will go "low" to indicate that the CPU has written data out to port A.

$\overline{ACK}$ (Acknowledge). A "low" on this input enables the tri-state output buffer of port A to send out the data. Otherwise, the output buffer will be in the high impedance state.

INTE 1 (The INTE Flip-Flop Associated with OBF). Controlled by bit set/reset of PC_6.

Input Operations

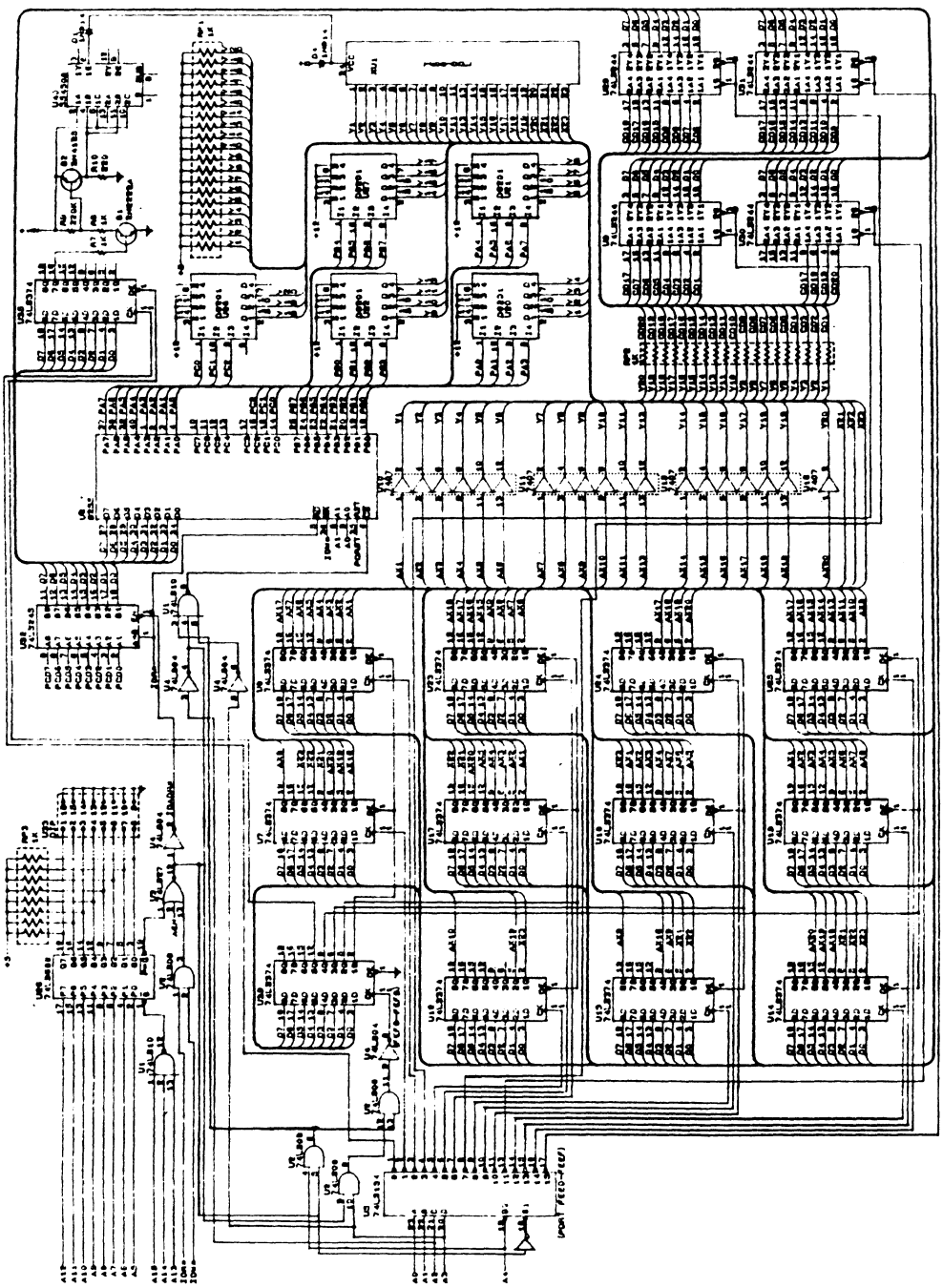
$\overline{STB}$ (Strobe Input)

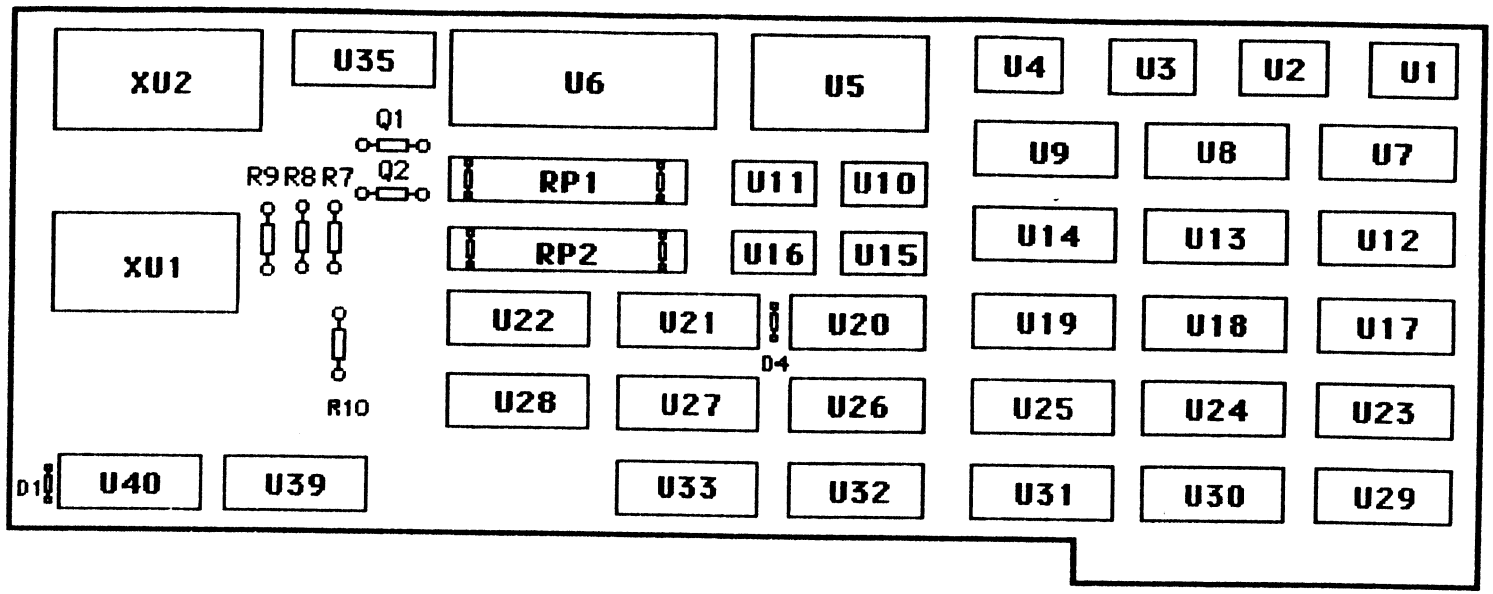
STB (Strobe Input). A "low" on this input loads data into the input latch.

IBF (Input Buffer Full F/F). A "high" on this output indicates that data has been loaded into the input latch.

INTE 2 (The INTE Flip-Flop Associated with IBF). Controlled by bit set/reset of PC_4.

APPENDIX H





U1	74LS10	U11	7407	U21	DG201	U31	74LS374	R9	220K
U2	74LS08	U12	74LS374	U22	DG201	U32	74LS254	R10	220
U3	74LS27	U13	74LS374	U23	74LS374	U33	DIP SWITCH	D1	1N914
U4	74LS04	U14	74LS374	U24	74LS374	U35	74LS374	D4	1N914
U5	74LS154	U15	7407	U25	74LS374	U39	74LS374	Q1	2N2222A
U6	8255A	U16	7407	U26	74LS688	U40	55450B	Q2	2N4125
U7	74LS374	U17	74LS374	U27	DG201	RP1	1K * 19	XU1	textool-24
U8	74LS374	U18	74LS374	U28	DG201	RP2	1K * 19	XU2	textool-28
U9	74LS374	U19	74LS374	U29	74LS374	R7	1K		
U10	7407	U20	DG201	U30	74LS374	R8	1K		