The design and developement of a microprocessor based control system for an electric rail transport system.

by

T M HUMAN

September 1988

A thesis submitted in partial fulfillment of the requirements for the Master's Diploma in Technology in the School of Electrical Engineering at the Cape Technikon.

Abstract

Radioactivity and radioactive radiation are two scientific phenomena which man has always approached with great caution, if not fear. Radioactive radiation cannot be sensed by any of the human senses and experience has proved just how hazardous it can be to the human body. This caution is therefore by no means unfounded and through the years a set of standards has been derived as to what can be regarded as a safe dose to the human body.

At the National Accelerator Centre radioisotopes are being produced by a chemical recovery process from targets which have been irradiated by a high energy proton beam. Targets are prepared by compressing salts, containing the elements to become radioisotopes, into tablets. The high energy protons collide with particles in the targets which give off radioactive radiation. The targets in their turn become radioactive and the isotopes that are produced from these targets are radioactive. The level of radioactive radiation is extremely hazardous and it is therefore impossible for any human being to come into close contact with any of the targets or isotopes. It is for these reasons that an electrical rail transport system was installed at the National Accelerator Centre to transport highly radioactive sources.

The transport system links the two irradiation vaults to the two rows of hot cells, where the chemical recovery takes place, and to a well shielded storage area for storage of isotopes and radioactive waste. A transport system, performing tasks of this nature, must be, above all, extremely reliable. Secondly, commands entered by an operator to control the system, must be simple and straight forward. This thesis describes the control of the transport system at the National Accelerator Centre, including all of its features, advantages and disadvantages.

SCOPE

To describe the design and development of a control system for an electric rail transport system to be used by the isotope-production facility at the National Accelerator Centre.

CONTENTS

LIST OF FIGURES

1. INTRODUCTION

At the National Accelerator Centre a variable-energy separated-sector cyclotron (SSC) and two solid pole injector cyclotrons (SPC1 and SPC2) are being constructed. The energy values for SPC1 and SPC2 are 8 MeV and for SSC 200 MeV. The two injectors produce light-ion and heavy-ion beams respectively. SPC1 was commissioned during 1984 and SSC during 1986. SPC2 is still under construction.

The accelerators were designed to accelerate positively charged ions. The ions are produced by introducing a gas, for example hydrogen, into an ion source where the electrons are stripped off the hydrogen atoms. The ions are then released in the centre of the injector, from where they are accelerated up to the extraction radius. Acceleration is accomplished by applying an RF-voltage across the two dees (so named because of their shape). The whole process takes place in a magnetic field which causes the ions to spiral out to the extraction radius. At extraction the beam is peeled off by magnetic and high-voltage components. From here it travels along the transfer beamline to the SSC. In this beamline quadrupole magnets are used to focus the beam and various diagnostic equipment produce information on the properties of the beam. On entering SSC, the beam is bent by means of magnetic components which steers it into its first orbit. The beam is then accelerated to the desired energy in a similar way as in SPC1. Once again magnetic and high voltage components are used to extract the beam from SSC into the high energy beamline. This beamline also uses quadrupole magnets for focusing and diagnostic components although they are slightly different to the ones in the transfer beamline because of the higher energy of the beam. By using switcher magnets the beam can be directed to several beamlines which serve different facilities. These include isotope production, radiotherapy and nuclear physics research. Another subject worth mentioning is that the whole process takes place in vacuum. This is to limit the beam losses due to charge exchange of ions with residual gas molecules.

At nuclear physics research, the beam can be directed to one of seven beamlines, serving three vaults, and, as the name implies, are being used for pure research. At radiotherapy the beam is used to bombard targets which radiate neutrons. The neutron beam is then directed onto cancerous tumors.

In order to produce radio-isotopes, targets are being irradiated by the beam in one of the two vaults at isotope-production. Vault 1, or the horizontal vault as it is often referred to, has three beamlines. Vault 2, or the vertical vault, has only one beamline. Figure 1.1 shows the rotary target magazine that is used in vault 1. The irradiated targets are transported from the vaults to the area where the chemical recovery process takes

place. This is done in what is called hot cells. Figures 1.2 and 1.3 show the front and rear views of a row of hot cells. Each hot cell consists of a containment box with lead shields of 100mm in front and 50mm on the remaining three sides as well as on top. After the recovery process, the radioactive waste and the radioisotopes are transported from the hot cells to a well shielded storage area. Targets for irradiation are prepared in a laboratory in the vicinity of the hot cells. These targets are also transported to the vaults and placed in special magazines at the end of the beamlines.

An electric rail transport system was installed to transport targets and sources with high radioactive radiation and to transport new targets into dangerous areas like the vaults. The rail system comprises a continuous aluminium track, with power and control rails and a gear rack, between reception hot cells A and vault 1. Switchers are provided to switch a transporter to side tracks serving the parking loop, vault 2 and reception hot cells B. The parking loop forms a closed loop with the switcher as the only means of input or output. It can hold up to nine transporters and the unused transporters are parked here. A schematic layout of the transport system is shown in fig 1.4.

Five transporters, which are in effect motorized trolleys, are used in the transport system. Each transporter has its own geared electric motor operating from a 32V D.C. supply. Power is transferred via copper power brushes in contact with power rails built into the aluminium track. The 32V electric motor has an integral gearbox which turns the drive shaft at approximately 100 rpm, resulting in a track speed of 0,4 metres/second. In addition to the two brushes that transfer the D.C. power, a third brush is fitted which provides a control signal. Reversal of the transporter is achieved by reversing the polarity of the power rails in the track. Irrespective of the polarity of the rails a positive potential is always present on the third or control brush of the transporter. This positive control voltage is fed into a third rail in the track and is used for controlling the movement of the transporter at switchers and stations. Whilst moving on the track, this voltage is also used for indicating its presence at these positions on a mimic panel. Where the transporter is required to stop on a switcher or in a station in a particular position, the polarity of that section of the rails is switched so that both are at a positive potential. This effectively short circuits the drive motor of the transporter, stopping it almost immediately. The drive through the track is achieved by means of a cog on the drive shaft of the gearbox engaging with a nylon gear rack in the track. This, together with the above-mentioned electrical stopping system, ensures a rapid and repeatable stop. For the purpose of being able to remotely and electronically identify each transporter in order to "call" the required transporter from the parking loop, each unit is fitted with two magnets, the positions of which provide

identification to reed contacts on a sensing plate from one to five. One of the transporters is shown in figure 1.5 and figure 1.6 show the transporter on the track.

Figures 1.7 and 1.8 show how a transporter is shunted from the main track into the parking loop by means of a switcher. This is typical of how all the shunting is done in the transport system.

Because of the obvious dangers of radioactive radiation, the transport system must be remotely controlled from a safe area. The appropriate place would be the isotope production control room which is in the white area (the red area is where the radiation levels can be extremely high and the blue area is a buffer area between the red and white areas). The single most important feature of the control system should be its extreme reliability. Some of the radioisotopes to be produced at the NAC have half lives of several weeks. This means that if the system malfunctions while a transporter is carrying such an isotope, it would be almost impossible to go anywhere near the transporter to retrieve it, for a long period.

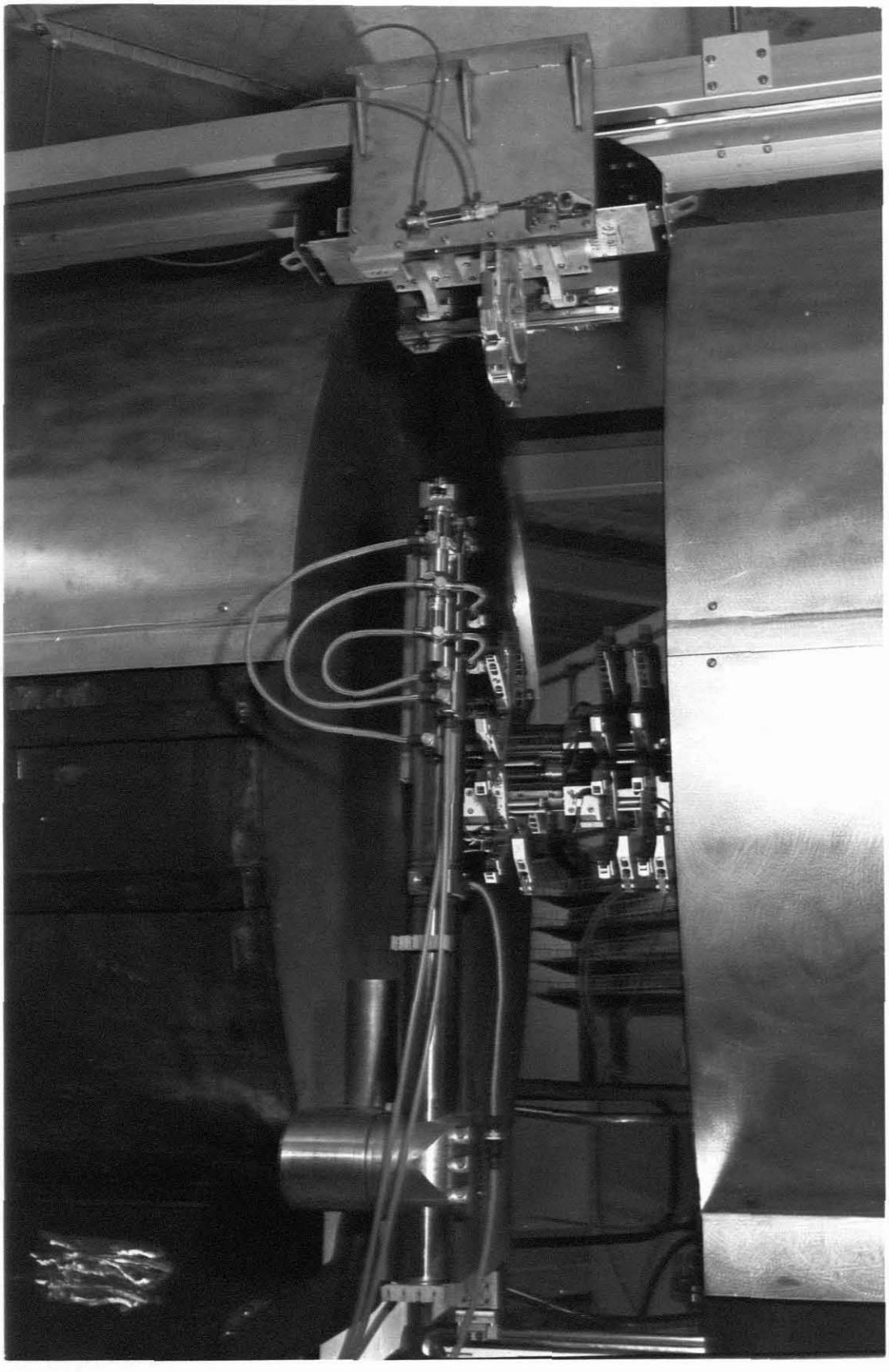


Fig. 1.1

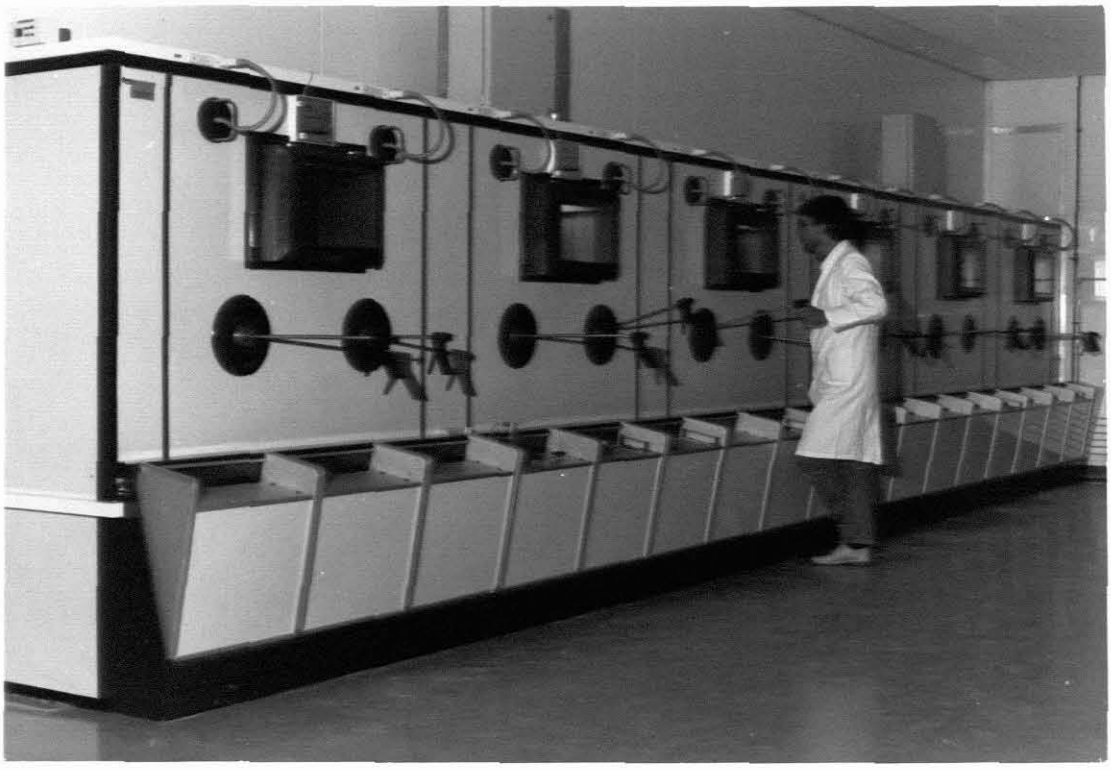


Fig. 1.2

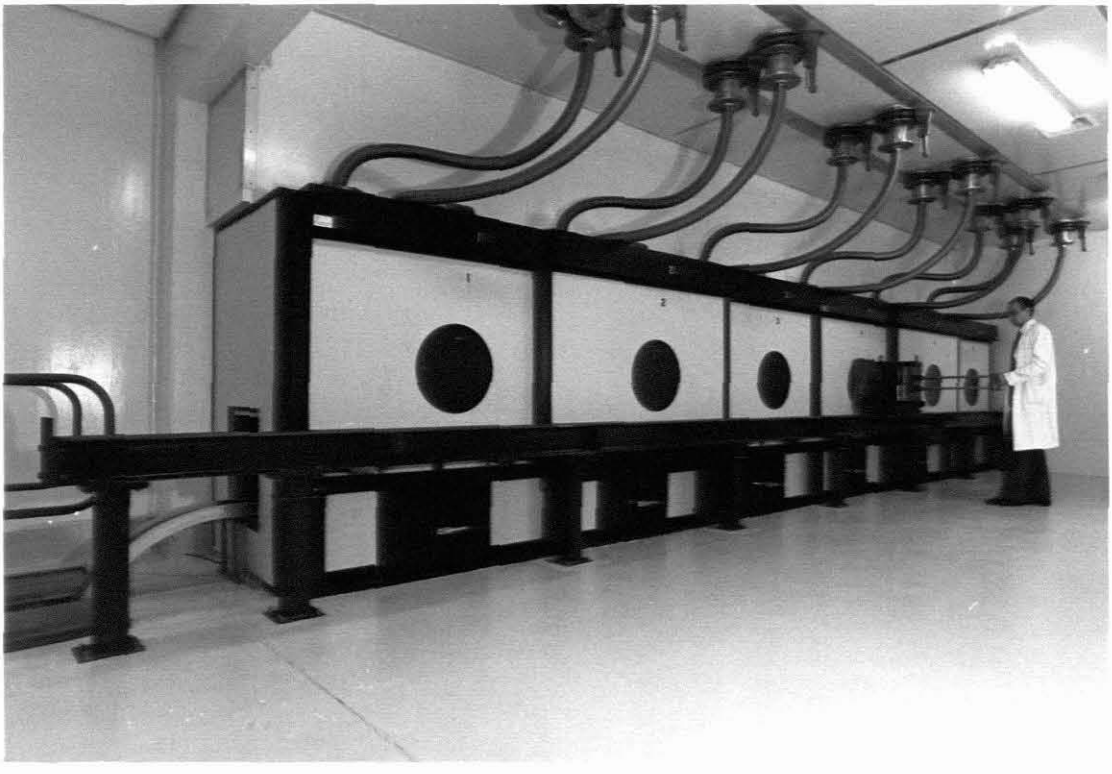


Fig. 1.3

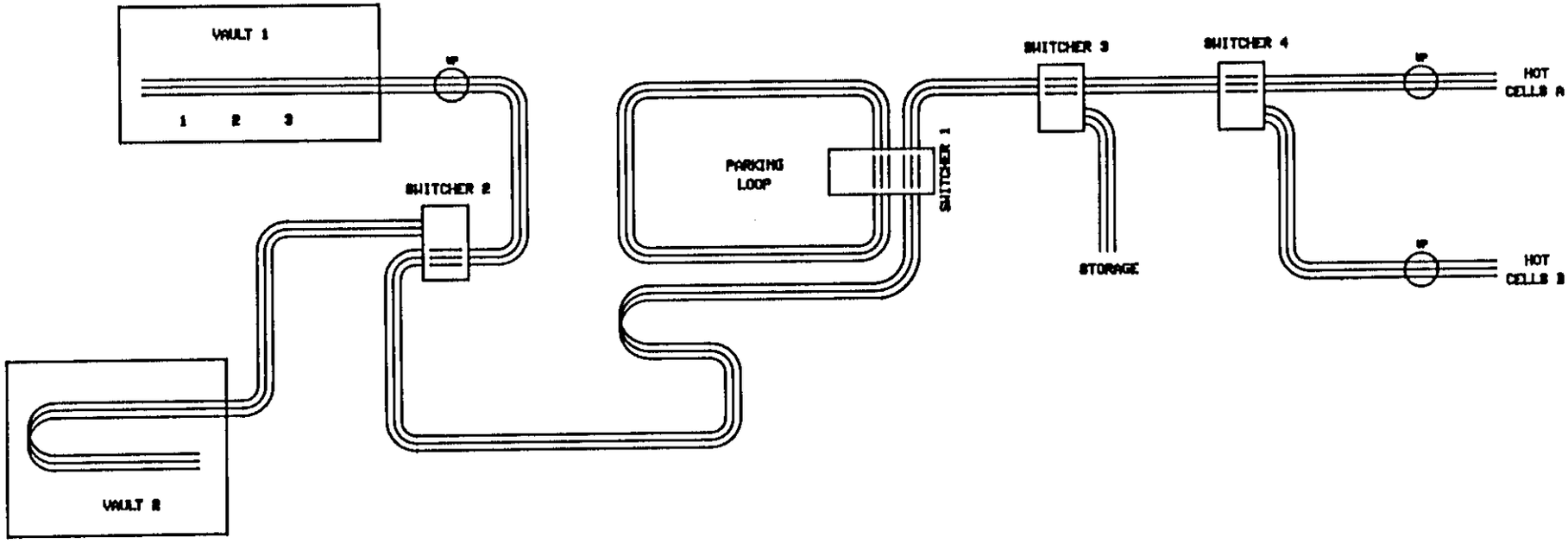

VAULT 1
1
2
3
SWITCHER 2
VAULT 2
PARKING
LOOP
SWITCHER 1
SWITCHER 3
SWITCHER 4
STORAGE
HOT
CELLS A
HOT
CELLS B

Fig. 1.4

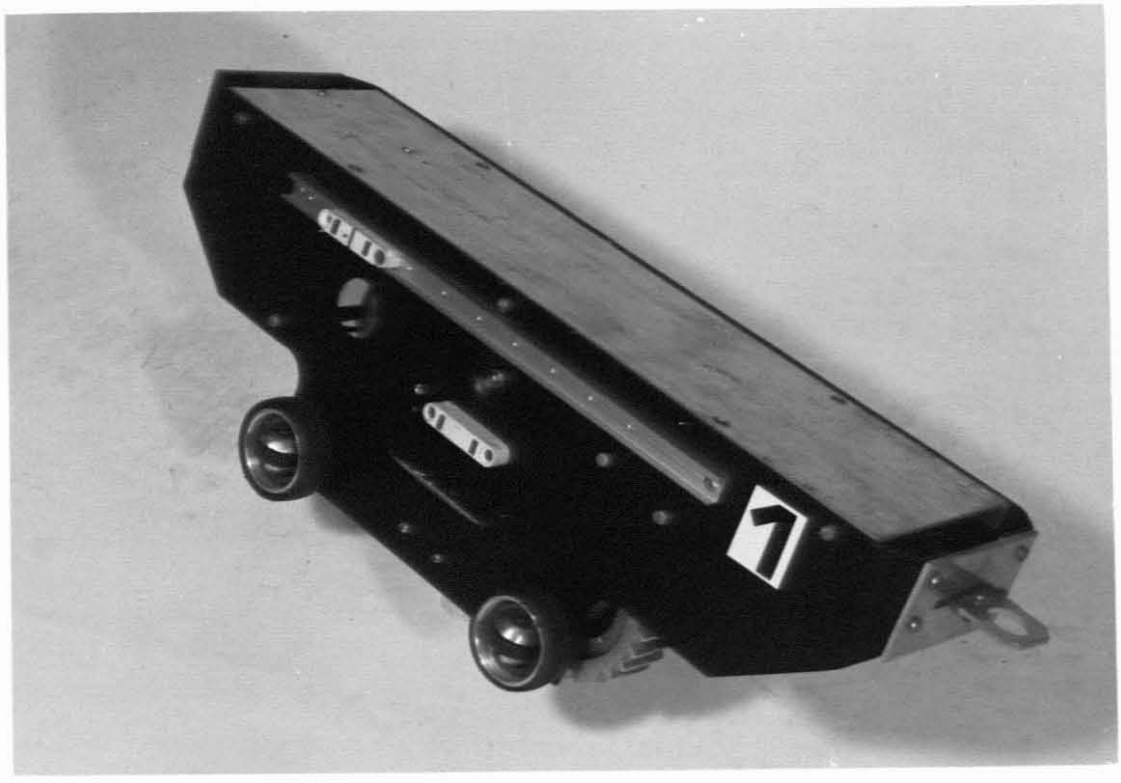


Fig. 1.5

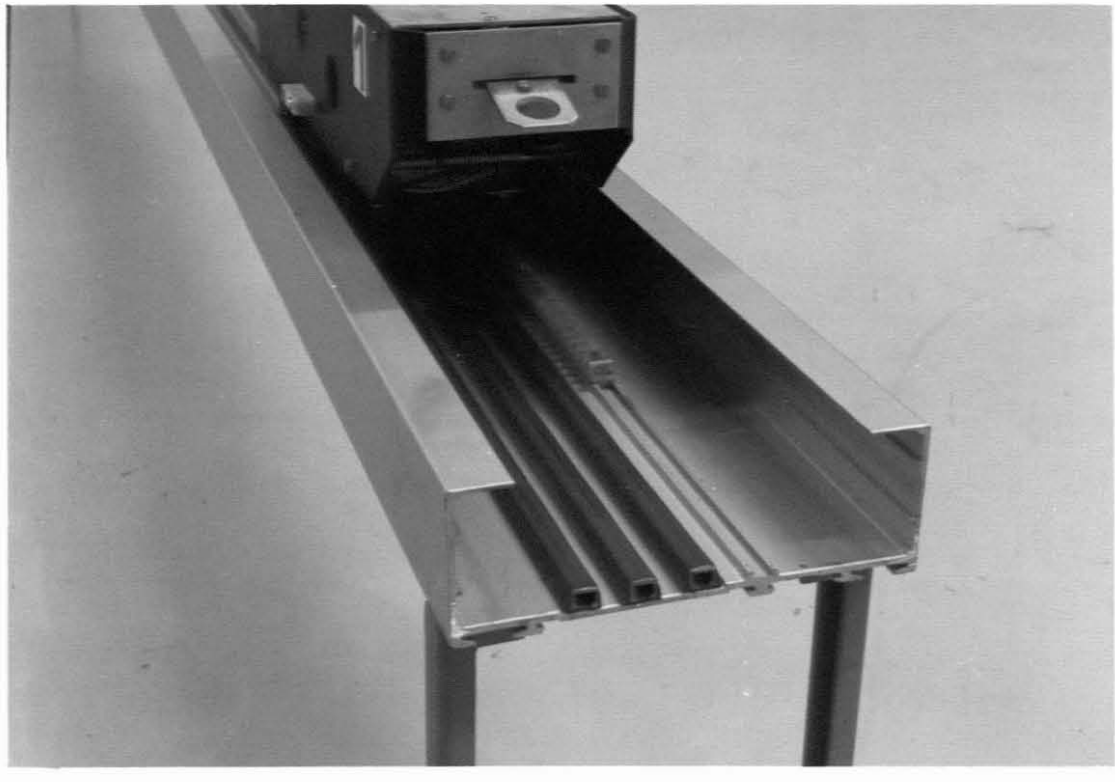


Fig. 1.6

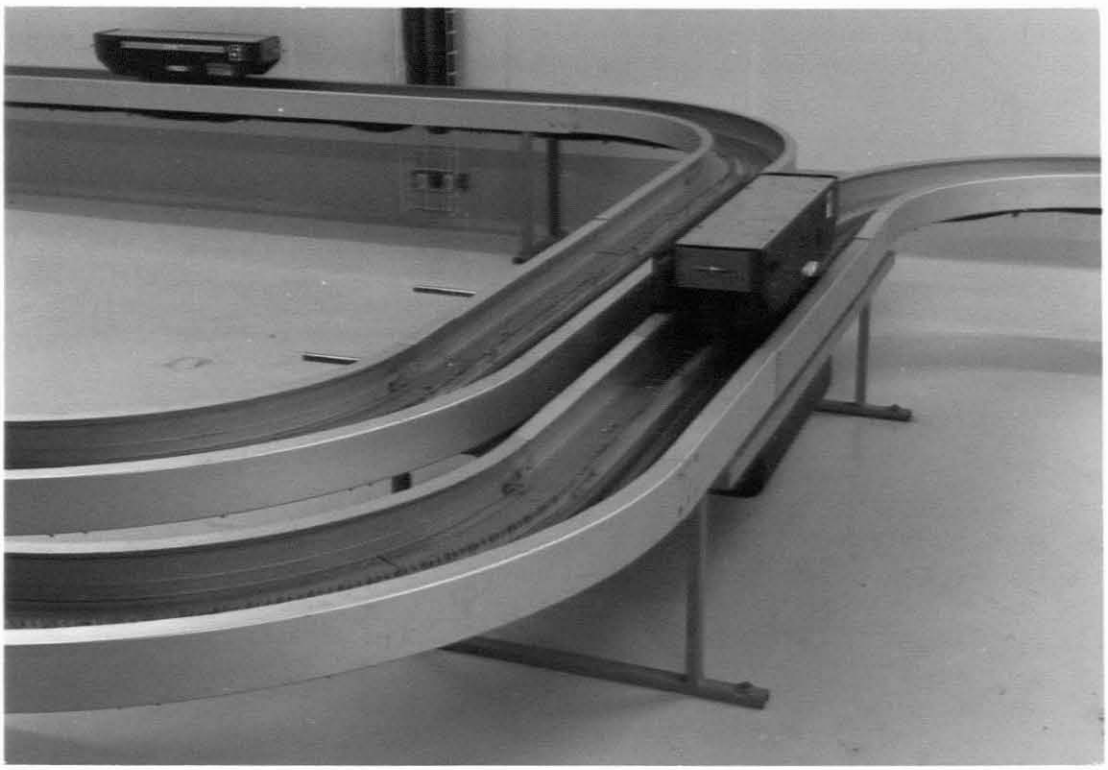


Fig. 1.7

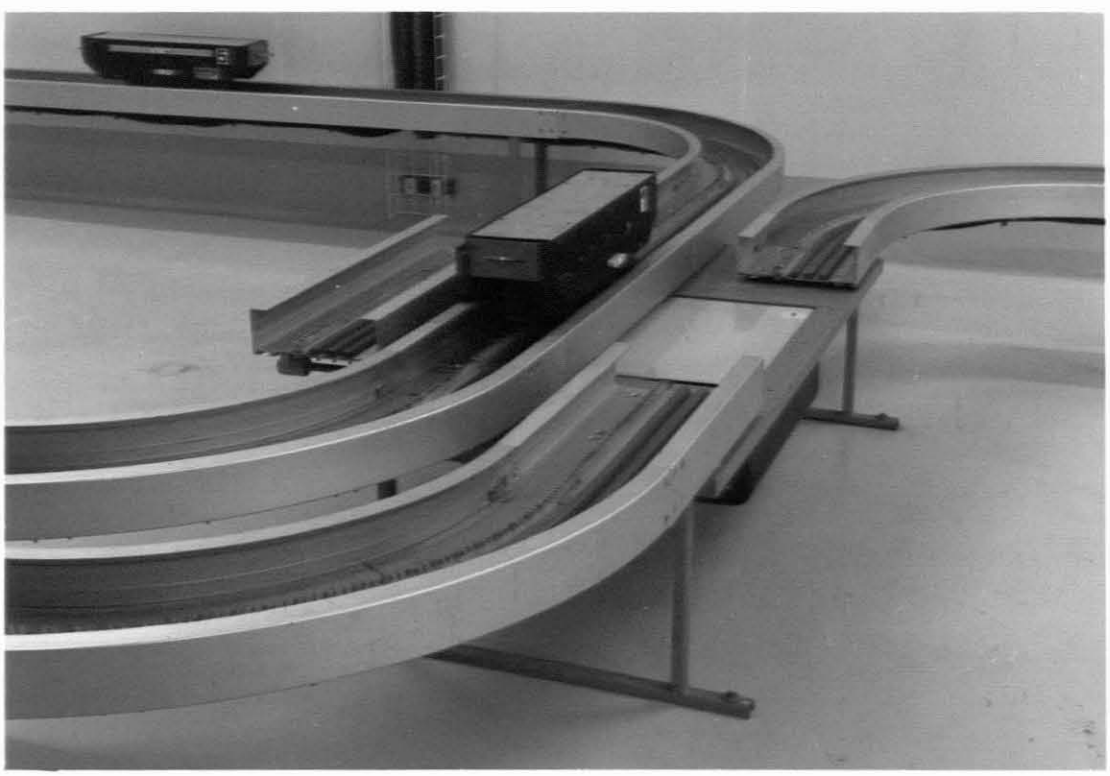


Fig. 1.8

DESIGN CONSIDERATIONS

The controller for the target transport system that was installed originally by a local firm proved to be very unreliable. It was then decided to design a complete new control system based on a microprocessor and using input and output modules that are widely used throughout the NAC. Because of the hazardous nature of radioactive radiation such a system should be extremely reliable. The advantage of using an in-house microprocessor and support modules is that maintenance and breakdown repairing becomes a simple task because this is achieved by module replacement. On the other hand it becomes a very flexible system because software can be changed fairly easily.

The Z-80 microprocessor was chosen to do the control. It is housed in a small module that 'lives' on the SABUS backplane. The other modules that were chosen included an eighty bit opto-isolated input module, three thirty-two bit relay output modules, a GPIB card for communication with another microprocessor and a bubble memory for software storage. These are all housed in a nineteen inch cardframe and connected to the SABUS backplane.

Instead of having to enter commands from a keyboard where typing errors are likely to occur, a local controller was opted for. This instrument was designed so that commands could be entered by push-buttons from the front panel. The logic layout of the push-buttons and the fact that they are well described on the front panel makes it simple and straight forward to operate. This also eliminates the need to remember complicated command strings if a keyboard was used.

A mimic panel was designed to indicate the positions of all the transporters and the status of the switchers. It also indicates the progress of a transporter moving along the tracks. A three-dimensional drawing of the transport system was made and transferred onto the front panel. Together with the information from the local controller lamps, the status of the system is completely displayed.

Instead of having just a fault lamp on the local controller indicating power supply malfunction, illegal command entries, etcetera, it was decided to have an alpha-numeric display. A two line by forty character dotmatrix display was opted for. This display would then exactly state any problem to the operator. An interface card to drive the display was designed to be driven directly by the microprocessor while living on the SABUS.

In order to switch the currents on the tracks for the transporters and to power the switchers, a hardware controller was designed consisting of relays. This controller is situated in the vicinity of the tracks to reduce cable length and to minimize voltage drops in the cables. A second cabinet was

designed for this area to concatenate all the various sensing outputs from the transport system and control signals from SABUS.

This ensures a neat, orderly and logic cabling system to link the transport system to the input and output modules on the SABUS.

The software for the system was designed and written after a thorough study was made of the system's capabilities and the operator's requirements. Turbo Pascal was used as the software language.

The inputs to the 80-bit opto-isolator card amount to sixty-nine. All these inputs are contacts from relays, reed switches, pushbutton switches, etcetera. Switch debouncing was considered to provide predictable results but was decided against for the following reasons. Most of the sensing units in the system consist of mercury wetted switches which do not bounce. When the microprocessor is polling the local controller pushbuttons for a command entry, the operator reaction time is normally slower than the 20ms settling time for switch bounce. Where switch bounce can be critical, the software can provide delays to accommodate for settling times and lastly the debounce circuitry for such an amount of contacts can be costly.

3. HARDWARE DESCRIPTION

A block diagram of the complete control system is shown in figure 3.1. The SABUS cardframe, the mimic panel and the local controller, shown in figure 3.2, are situated in the control room. PE1, PE2 and the two power supplies are situated in the vicinity of the parking loop. Figure 3.3 shows these where they are mounted on the wall above the parking loop. The hardware controller is built into PE1. Power and signal cables from the transport system are terminated in PE1 and PE2. PE1 and PE2 on their turn, are connected to the control system by six twelve pair cables.

3.1 THE Z-80 MICROPROCESSOR, SABUS AND POWER SUPPLIES

The SABUS is widely used in microprocessor applications at the NAC. The Z-80 microprocessor cards, the 32bit relay output cards, the 80-bit opto-isolator input cards and the 128kbyte bubble memory cards were all developed at the NAC to be used on SABUS. It was therefore the logical choice to use these in this particular application. Fig 3.1.1 shows the SABUS cardframe containing all the modules.

The two power supplies came with the transport system and it was decided that it were adequate to be used. The 32V supply is used to power the tracks to drive the transporters. The 24V supply is used to drive the switchers and relays and is also used for information feedback to the 80-bit opto-isolator cards.

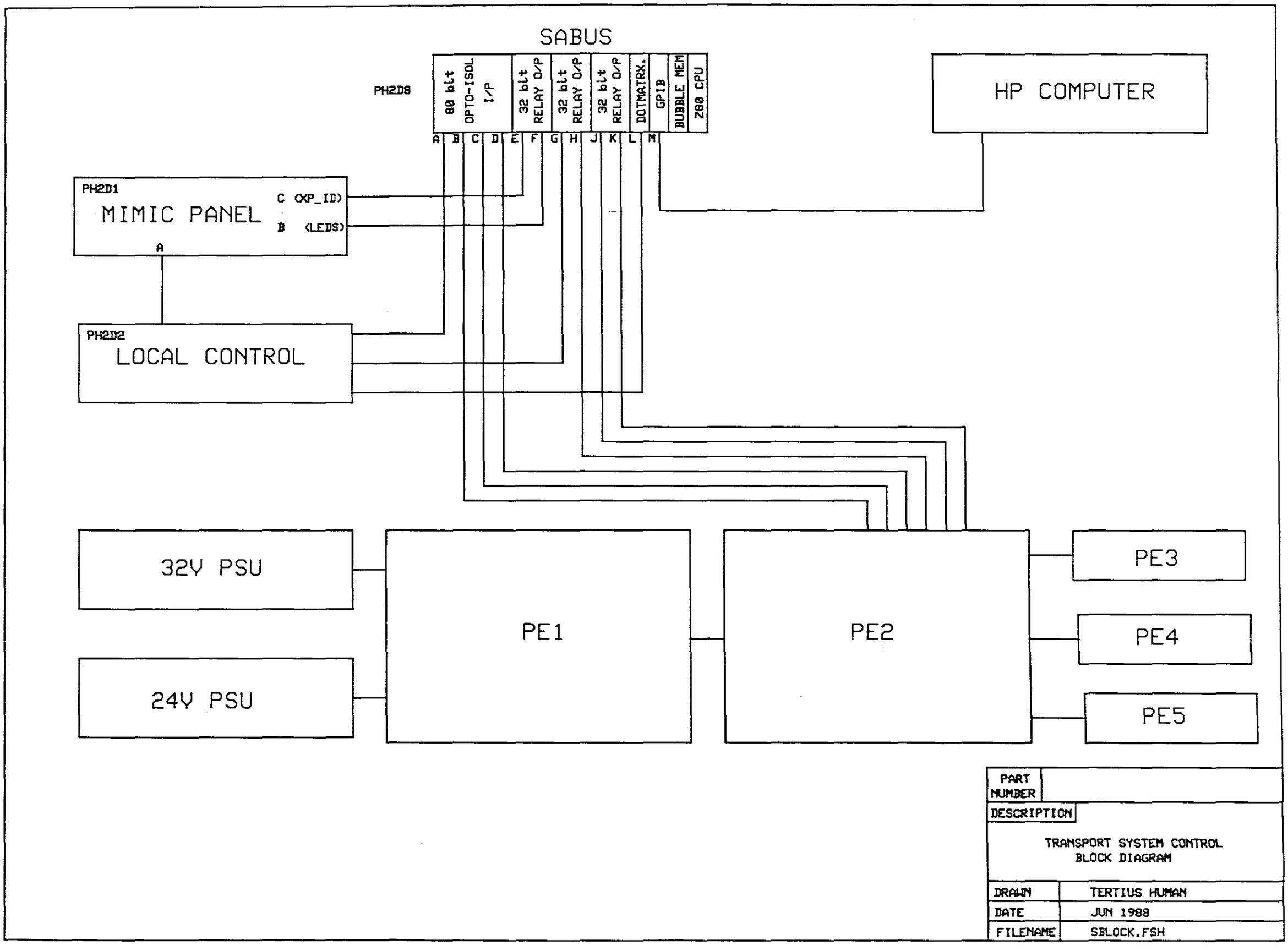

SABUS
PH2D8
80 bit OPTO-ISOL I/P
32 bit RELAY O/P
32 bit RELAY O/P
32 bit RELAY O/P
DOTMATRX.
GPIB
BUBBLE MEM
Z80 CPU
A
B
C
D
E
F
G
H
J
K
L
M
HP COMPUTER
PH2D1
MIMIC PANEL
C (XP_ID)
B (LEDS)
A
PH2D2
LOCAL CONTROL
32V PSU
24V PSU
PE1
PE2
PE3
PE4
PE5
PART NUMBER
DESCRIPTION
TRANSPORT SYSTEM CONTROL
BLOCK DIAGRAM
DRAWN
TERTIUS HUMAN
DATE
JUN 1988
FILENAME
SBLOCK.FSH

Fig. 3.1

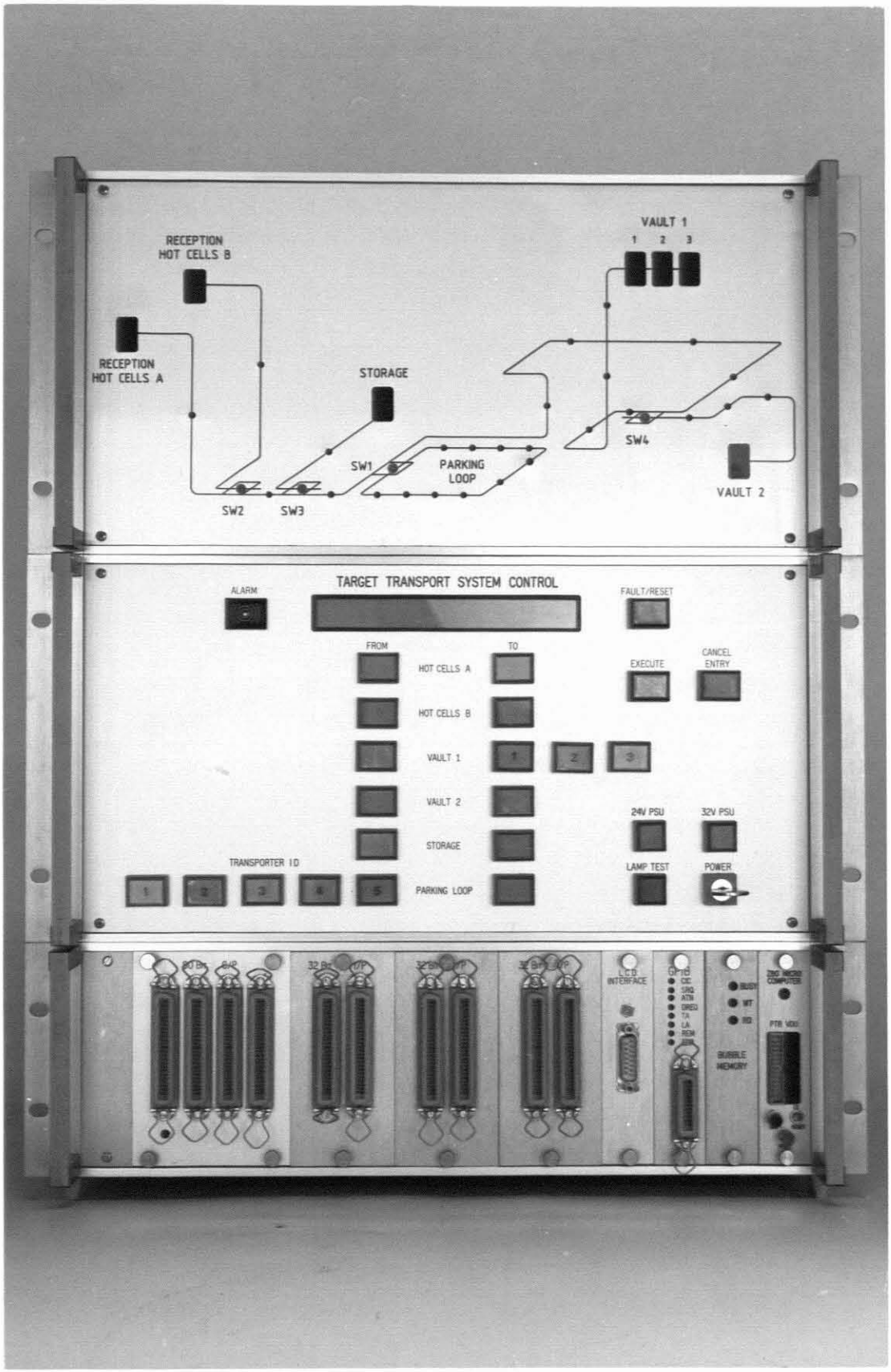

RECEPTION
HOT CELLS B
RECEPTION
HOT CELLS A
STORAGE
SW1
SW2
SW3
SW4
PARKING
LOOP
VAULT 1
1 2 3
VAULT 2
TARGET TRANSPORT SYSTEM CONTROL
ALARM
FAULT/RESET
FROM
TO
HOT CELLS A
HOT CELLS B
VAULT 1
VAULT 2
STORAGE
PARKING LOOP
EXECUTE
CANCEL
ENTRY
24V PSU
32V PSU
LAMP TEST
POWER
TRANSPORTER ID
LCD
INTERFACE
BUSY
WT
RD
BUBBLE
MEMORY
Z80 MICRO
COMPUTER

Fig. 3.2

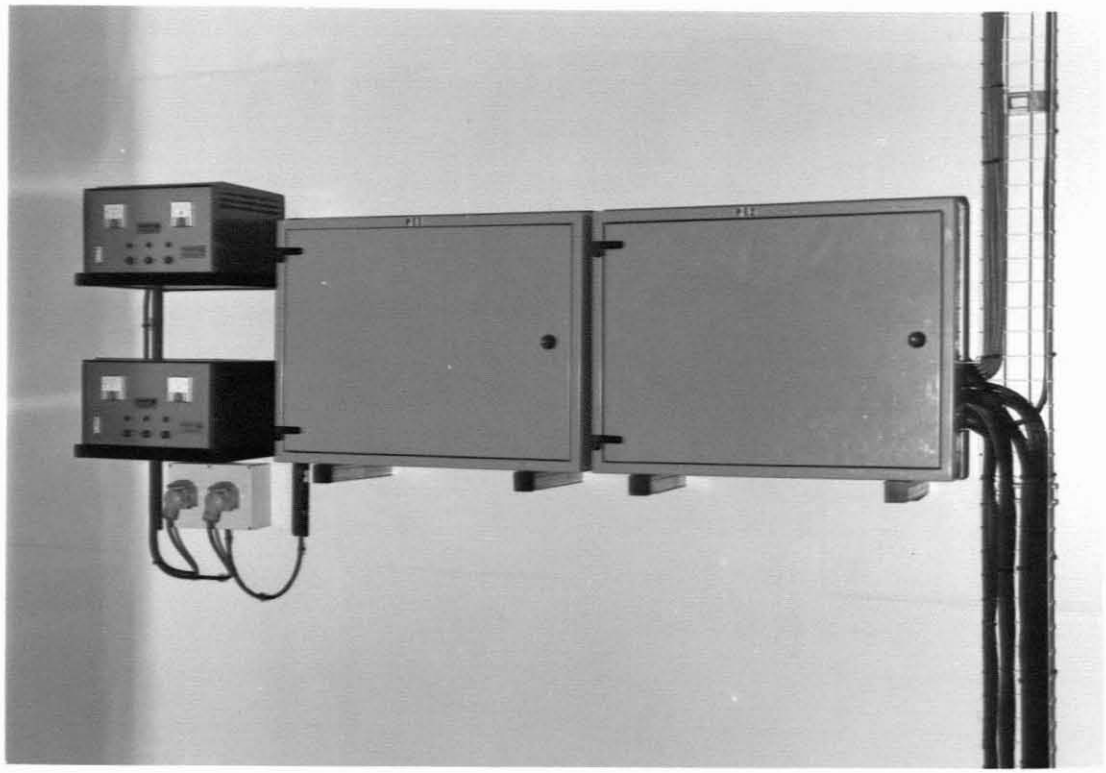


Fig. 3.3

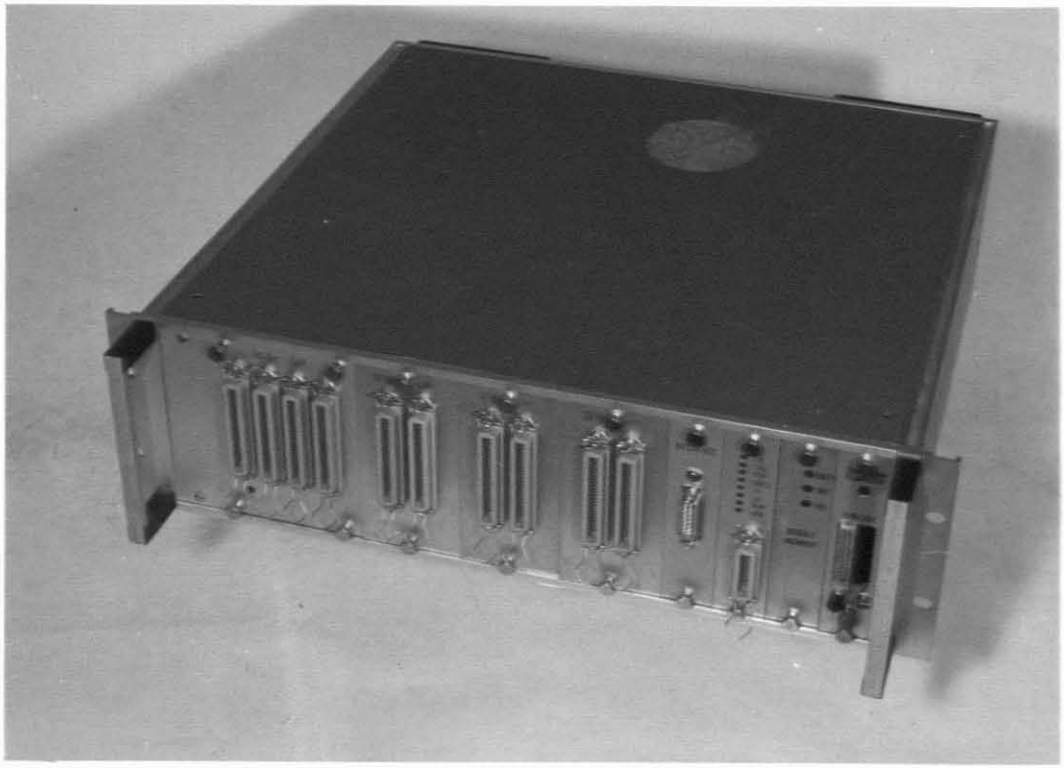


Fig. 3.1.1

3.2 DOTMATRIX DISPLAY INTERFACE

When it was decided to display micro-processor messages and system information to the operator, an alpha-numeric display was purchased. The chosen display is a two line by forty character LCD dotmatrix display. To be able to drive it directly from SABUS, an interface card was designed. The pin assignments and functions for the display are given below.

Signal name	Pin no.	Function
DB0 - DB7	7 - 14	Tristate bidirectional data bus.
E	9	Enable for data read/write.
R/W	5	Select read or write.
RS	4	Register select.
Vee	3	LCD drive power source.
Vcc	2	+5V power.
Vss	1	0V

Two types of registers are contained in the LCD internal controller namely an instruction register (IR) and a data register (DR). The IR stores instruction codes such as clear display, cursor shift as well as address information for the display data RAM. The DR is used for storing temporarily data to be written into the display data RAM. A character generator ROM can generate 160 different 5*7 dot character patterns including the ASCII character set. Data bit 7 on the data bus is used as a busy flag in a cpu read operation and when set no instruction will be accepted.

The interface was designed to fill a slot in SABUS. This way the cpu can communicate directly with the display. Figure 3.2.1 shows the circuit diagram of the interface. IC1 is an eight bit bus transceiver that is directly connected to the data bus. IC2 acts as a digital comparator. The one side is connected to the address bus and the other side to an octal switch block. When the code that is set up on the switch block matches the code on the address bus, IC1 is enabled for data transfer. The card's bus address is in actual fact set up on the switch block.

Address line A0 is used for register select and A1 for read write operation. The SABUS read and write lines are used for enable. A 20k variable resistor is used to set the LCD drive voltage which influences the viewing angle. The pcb was designed on the eurocard standard and using the SABUS pin-out on the card edge connector.

The display also has a back light source using organic film as a substrate. The colour emission is blue green and the driving voltage can be selected in a range of 600 – 1000Hz at 150V AC maximum. The power supply to drive the back light is discussed under the local controller.

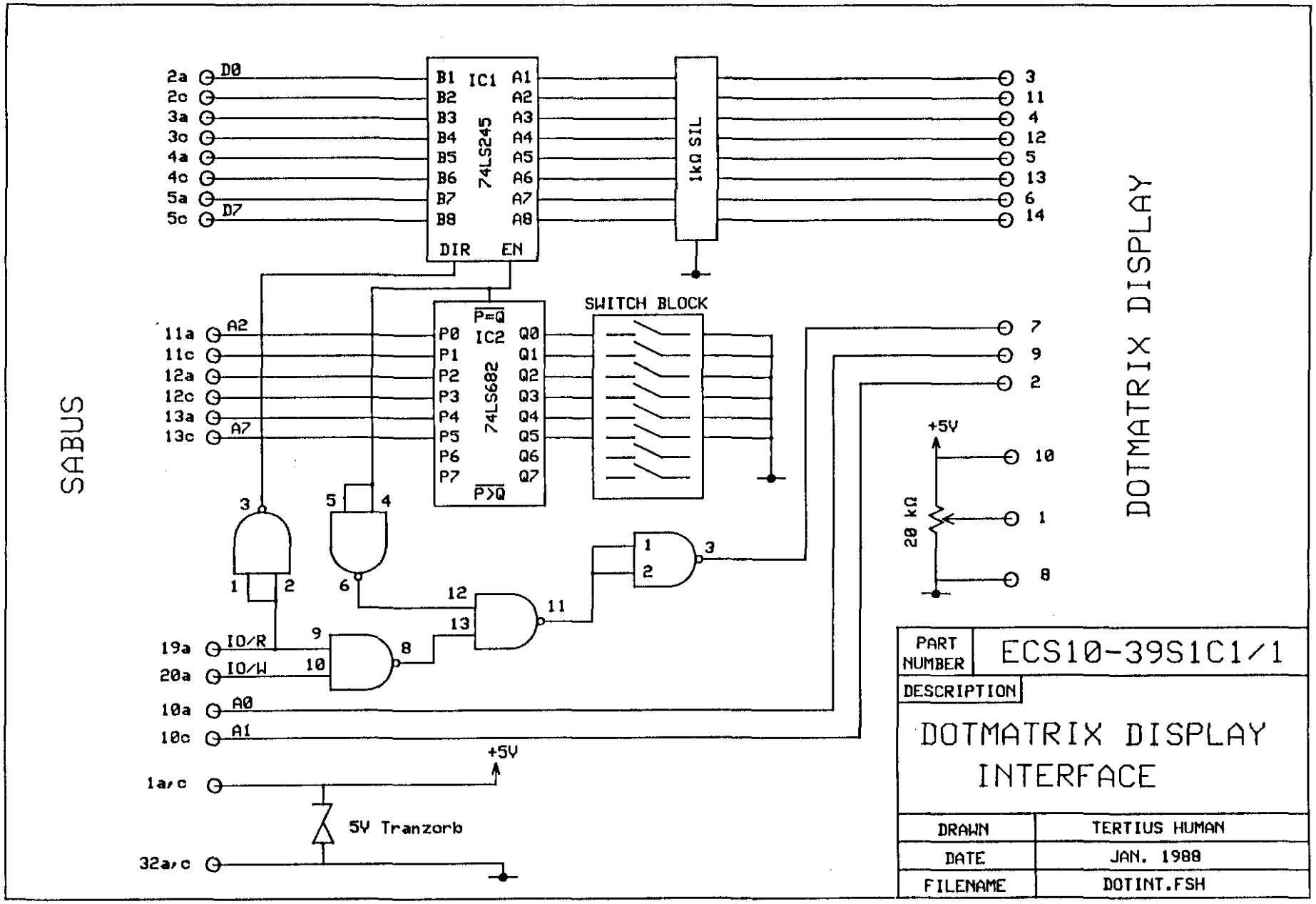

SABUS
2a D0
2c
3a
3c
4a
4c
5a
5c D7
B1 IC1 A1
B2 A2
B3 A3
B4 A4
B5 A5
B6 A6
B7 A7
B8 A8
74LS245
DIR EN
1kΩ SIL
3
11
4
12
5
13
6
14
DOTMATRIX DISPLAY
SWITCH BLOCK
11a A2
11c
12a
12c
13a
13c A7
P=Q
P0 IC2 Q0
P1 Q1
P2 Q2
P3 Q3
P4 Q4
P5 Q5
P6 Q6
P7 Q7
74LS682
P>Q
7
9
2
+5V
10
1
8
20 kΩ
19a IO/R
20a IO/W
10a A0
10c A1
1a,c
32a,c
5V Tranzorb
PART NUMBER
ECS10-39S1C1/1
DESCRIPTION
DOTMATRIX DISPLAY INTERFACE
DRAWN
TERTIUS HUMAN
DATE
JAN. 1988
FILENAME
DOTINT.FSH

Fig. 3.2.1

3.3 THE MIMIC PANEL

The function of the mimic panel is to indicate the status of each switcher, the presence of transporters in the parking loop, the progress of a transporter on the tracks and the presence and identification number of a transporter in the stations. Figures 3.3.1 and 3.3.2 show the front and rear views of the mimic panel cardframe.

The switcher status are indicated by means of bi-coloured LED's. Green indicates the switcher is in the 0 degree position and red the 180 degree position. The transporter progress and parking loop parking positions are indicated by red LED's. At every station position on the mimic panel, a seven-segment display indicates transporter presence and I.D. number. If no transporter is present, the display is blanked and if an unknown transporter is parked in it, a seven is displayed. Figure 3.3.3 shows the block diagram of the mimic panel control and figure 3.3.4 shows the inside of the instrument. The two major components are the transporter I.D. printed circuit board and the LED driver printed circuit board.

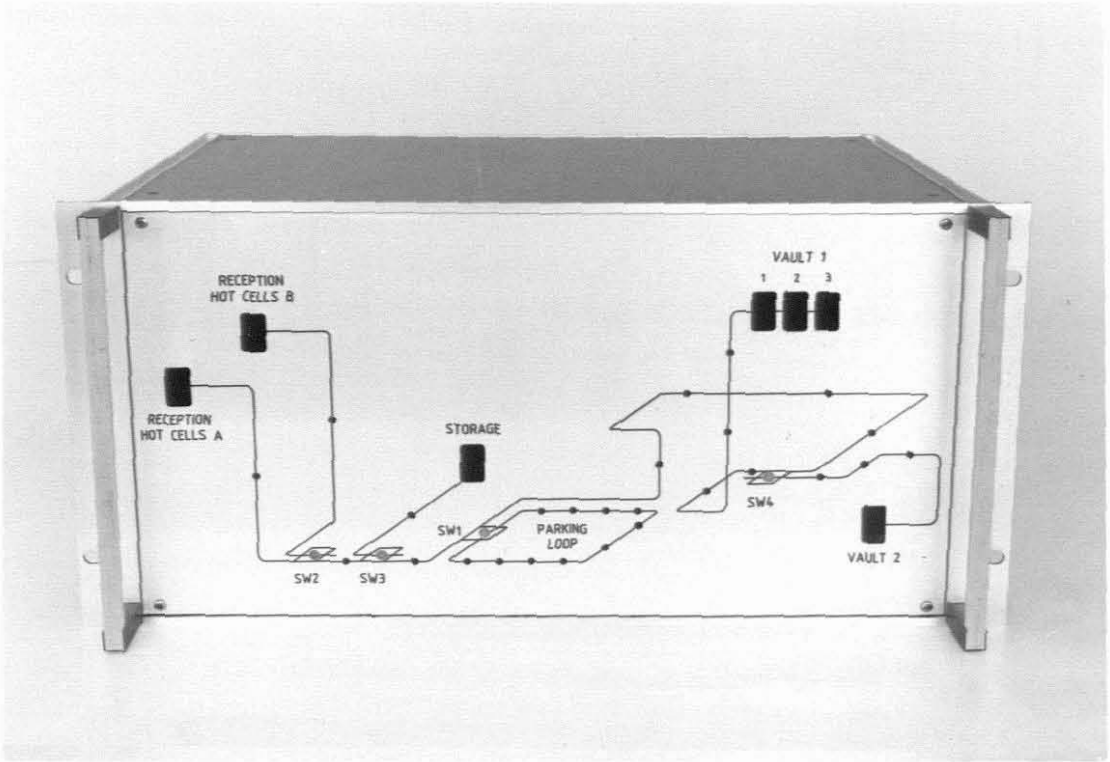


Fig. 3.3.1

Fig. 3.3.2

PH2D1

C

A

B

6

16

LT
COM
V+

1

16

ADAPTER
BOARD

TRANSPORTER ID
DISPLAY

LED DRIVER

VAULT1-1

VAULT1-2

VAULT1-3

VAULT 2

STORAGE

HOT CELLS A

HOT CELLS B

HA HB H SH S Va Vb Vc Vd V2a V2b V2c V1a V1b V1c V1d

TRACK POSITIONS

1 2 3 4

SWITCHES

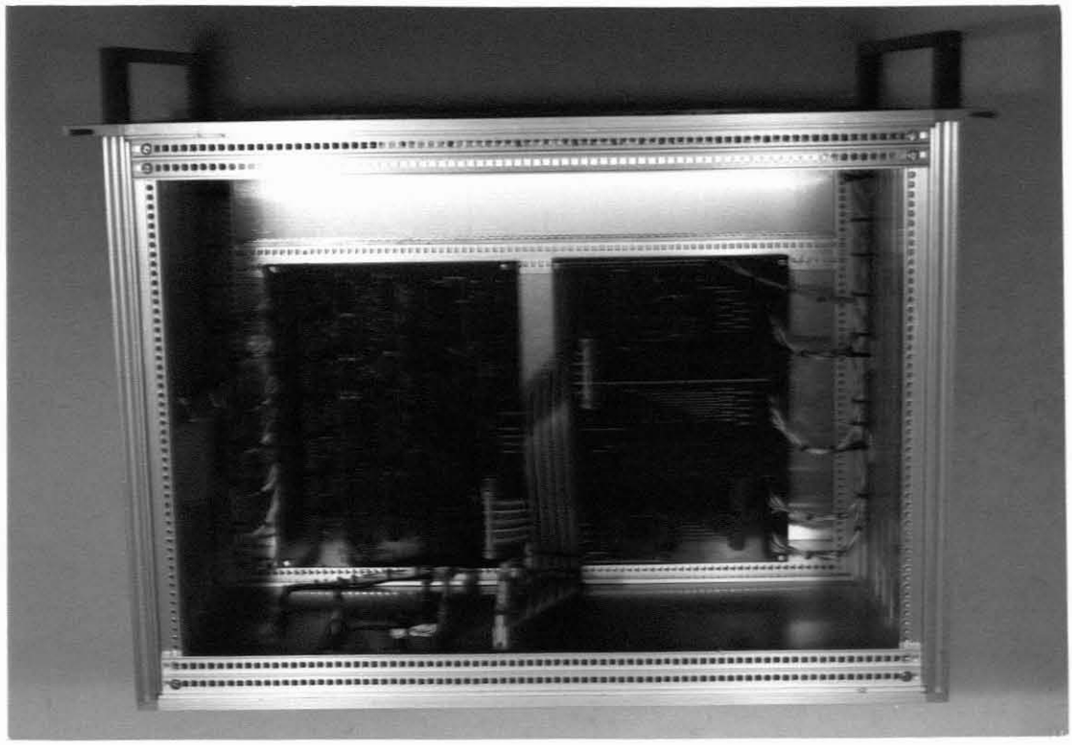
1 2 3 4 5 6 7 8 9 10

PARKING LOOP POSITIONS

FRONT PANEL LEDS

PART NUMBER	
DESCRIPTION	MIMIC PANEL BLOCK DIAGRAM
DRAWN	TERTIUS HUMAN
DATE	JUN 1988
FILENAME	MBLOCK.FSH

Fig. 3.3.3



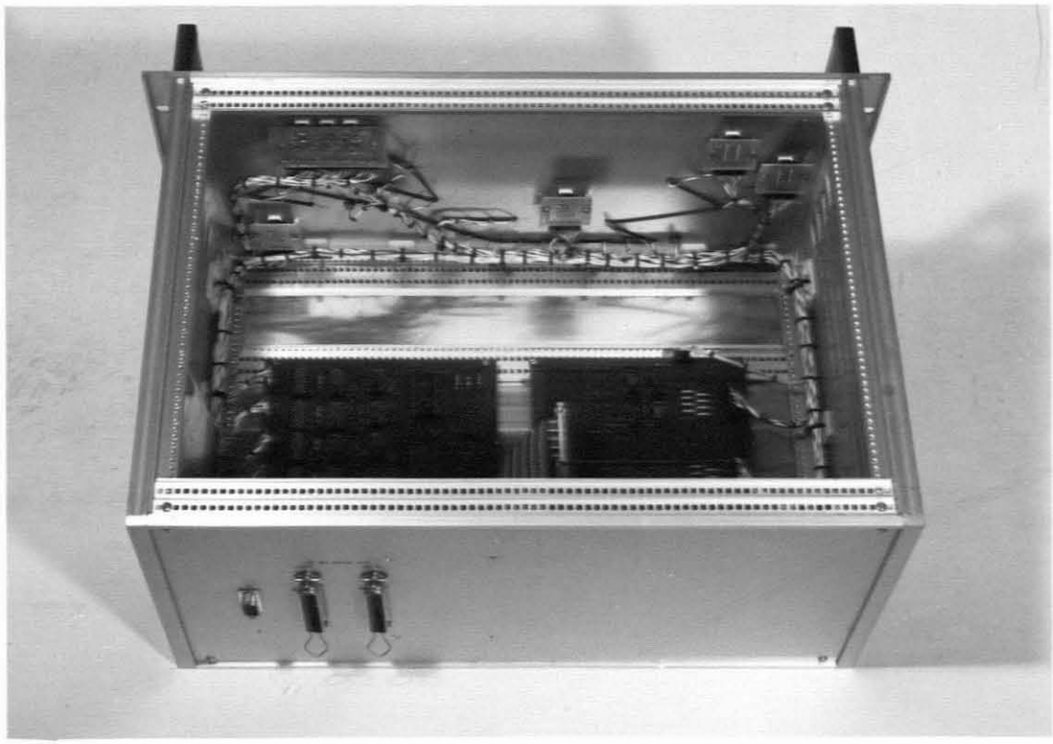
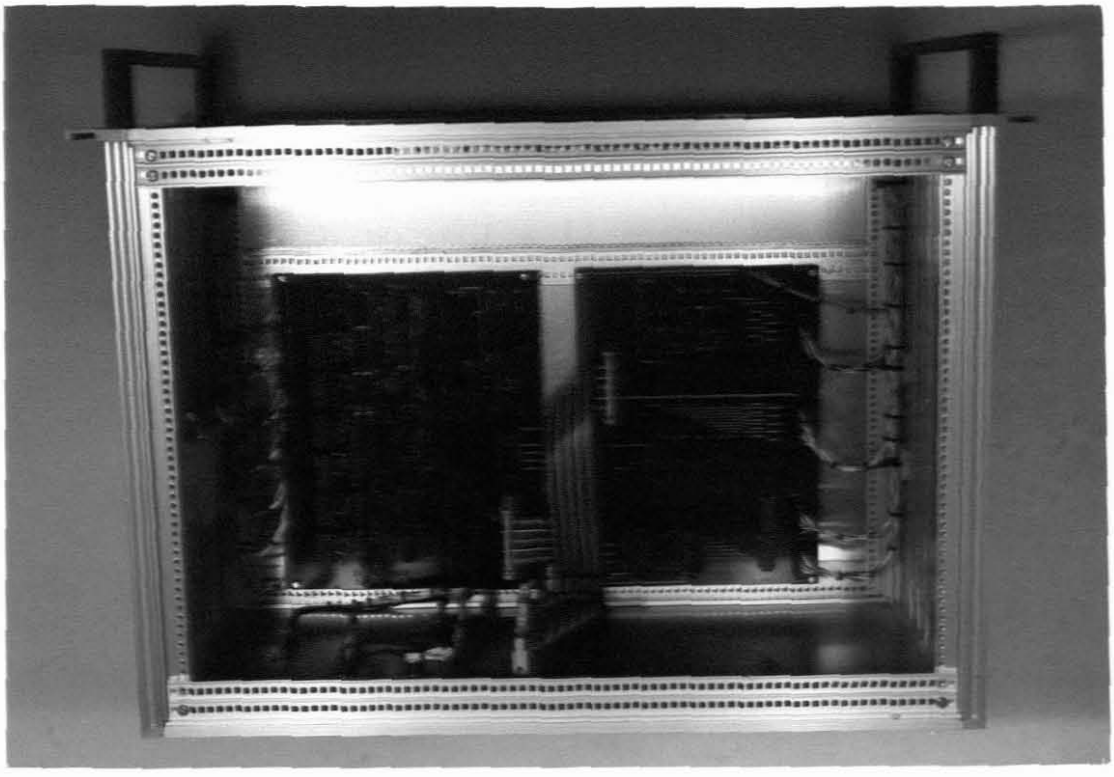


Fig. 3.3.4



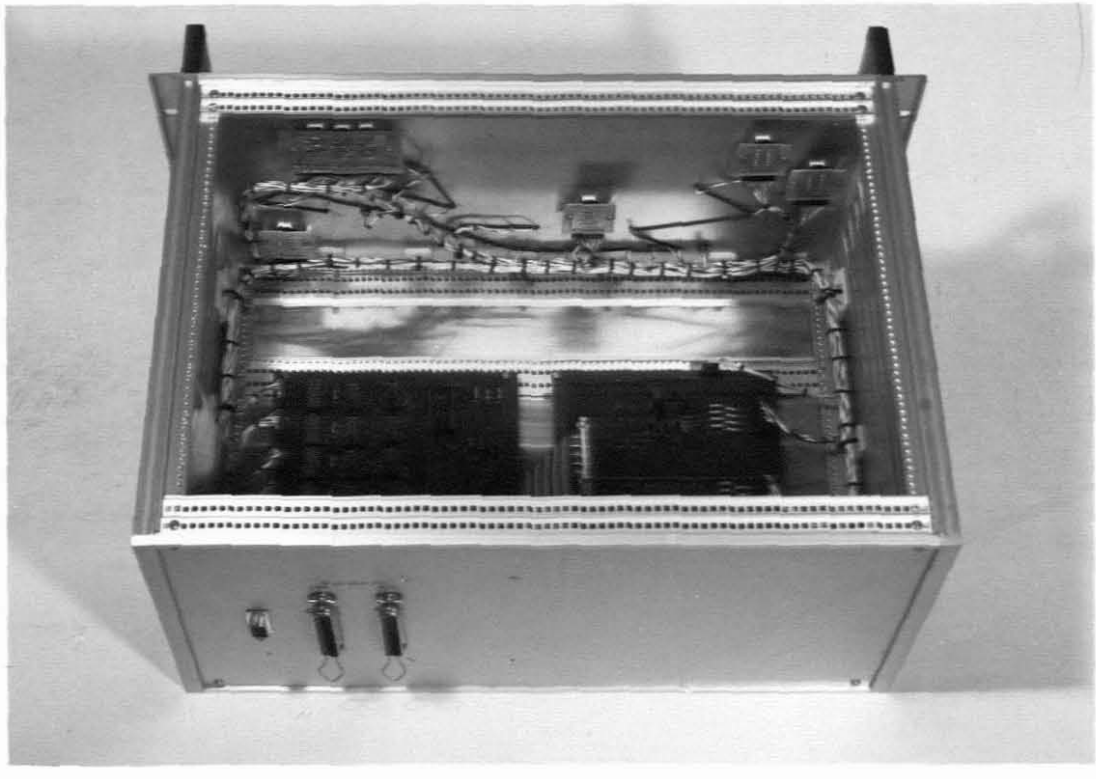


Fig. 3.3.4

3.3.1 LED DRIVER

This pcb has three functions namely parking loop positions, transporter progress and switcher status display. The circuit diagram is shown in fig. 3.3.5. The bi-coloured LED's that indicate switcher status are switched by reed relays. The transporter progress LED's are driven by a 4 to 16 line decoder/driver and the four data lines are latched by IC4. The enable line on the decoder/driver is used to switch off any LE- if no transporter is on the tracks while the enable line on the latch is used as a strobe. One of the 32-bit o/p cards on the SABUS drive all these signals as well as the parking loop position LED's. All the LED's on the mimic panel are ORed by diodes to serve as a lamp test which are driven from the local controller.

3.3.2 TRANSPORTER ID DISPLAY

In the transporter system there are seven stations where a transporter can be parked. They are hot cells A, hot cells B, storage, vault 2, and three positions in vault 1. Five transporters are used in the system which means that the numbers 1 to 5 are displayed in the stations. If a station is empty the display is blanked.

Figure 3.3.6 shows the transporter I- display circuit diagram. Display data is first set up on the three lines a, b, and c connected to IC's 1,2,3 and 4, which are latches. Next the station code is set up on lines a, b, c and d on IC28, a BCD to decimal decoder, followed by a strobe pulse to IC27. The decoded line from IC28 is latched through IC27 which then acts as a strobe on the data latches IC1 to IC4. At this stage one of IC20 to IC26 is addressed on its data inputs. These IC's are BCD to 7-segment decoder/drivers which are connected via CN1 to CN7 to the 7-segment displays. The OR-gates connected to IC20 to IC26 blanking inputs ensure that the displays are blanked if the received codes are 000. A lamp test line is also provided and is connected to the lamp test button on the local controller.

3.3.3 MISCELLANEOUS

In order to keep wiring of the pcb's and back panel connectors simple, an adapter pcb was designed. Flat cable are used from the two main pcb's to the adapter board from where it breaks out to the back panel connectors. A pcb was also designed to carry each of the 7-segment displays.

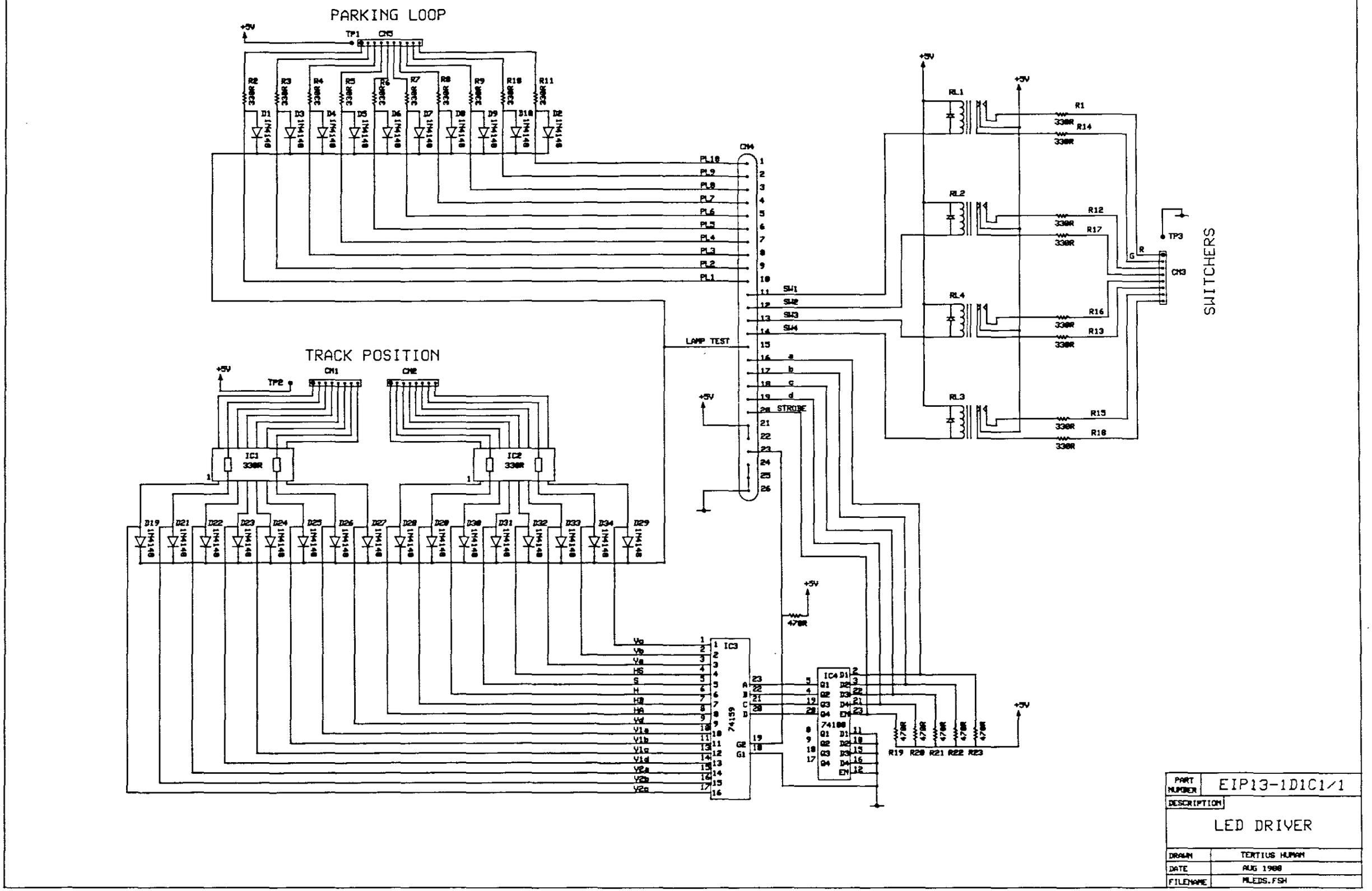
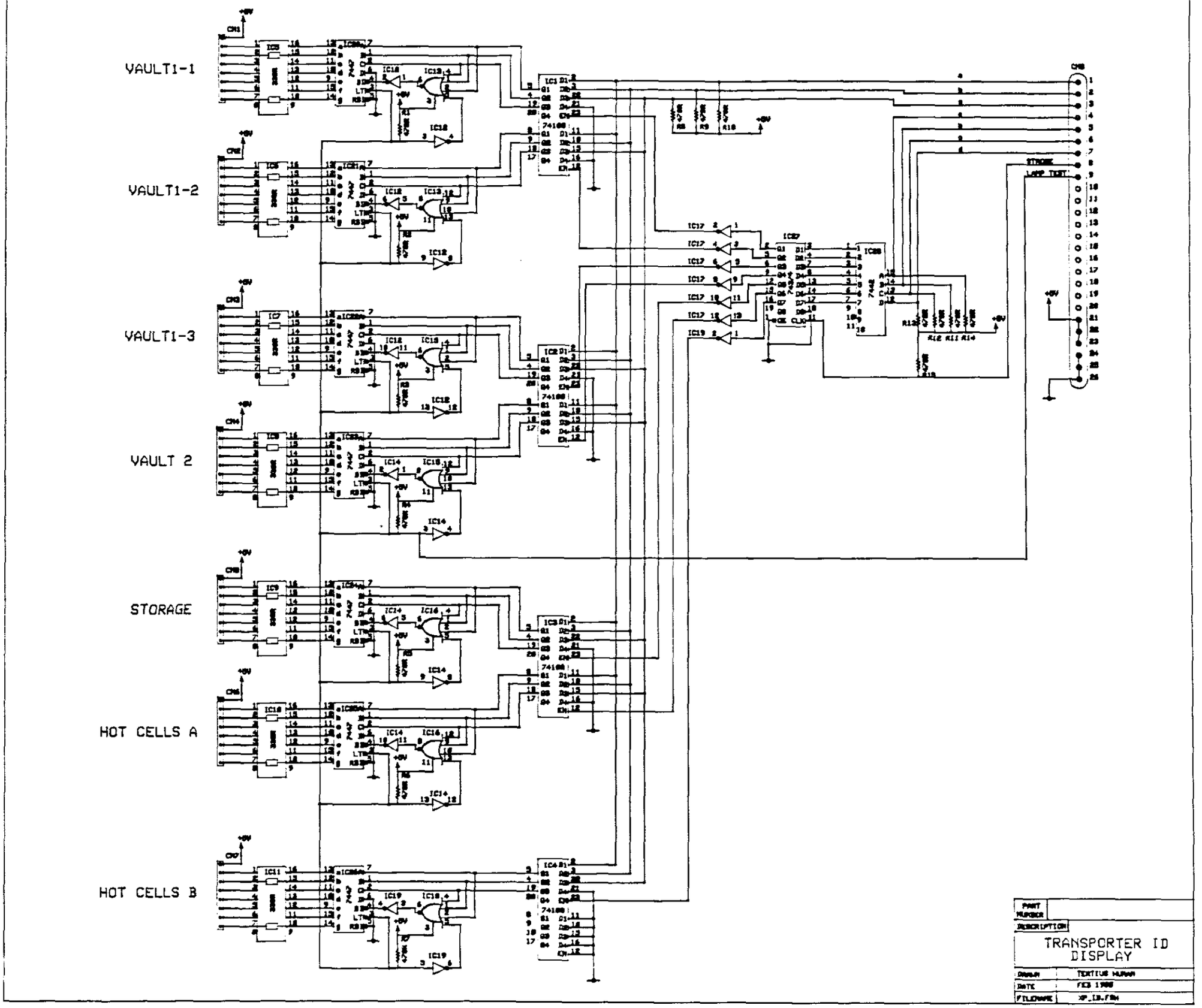

PARKING LOOP
TRACK POSITION
SWITCHERS
LAMP TEST
STROBE
74159
74180
IC1
IC2
IC3
IC4
CN1
CN2
CN3
CN4
CN5
TP1
TP2
TP3
RL1
RL2
RL3
RL4
PART NUMBER
EIP13-1D1C1/1
DESCRIPTION
LED DRIVER
DRAWN
TERTIUS HUMAN
DATE
AUG 1988
FILENAME
MLEDS.FSH

Fig. 3.3.5


VAULT1-1
VAULT1-2
VAULT1-3
VAULT 2
STORAGE
HOT CELLS A
HOT CELLS B
STROBE
LAMP TEST
TRANSPORTER ID DISPLAY

3.4 THE LOCAL CONTROLLER

All the commands to control the transport system are entered on the local controller by means of pushbuttons. It also displays information regarding the system and a piezo-electric buzzer acts as an audible alarm. Although the dotmatrix display is housed on the front panel it is driven directly from SABUS as described previously except for a back light power supply. Figures 3.4.1 and 3.4.2 show the front and back rear of the local controller.

A block diagram of the local controller is shown in figure 3.4.3. The major components are the local pcb, the front panel and the power supplies. Figure 3.4.4 shows the inside of the instrument where these components can be seen.

3.4.1 LOCAL CONTROL PCB

The function of this pcb is to drive all the indication lamps which are integral parts of the pushbutton switches. The circuit diagram is shown in figure 3.4.5. The indication lamps are devided into three groups namely FROM lamps, TO lamps and miscelaneous lamps. The FROM and TO lamps are driven in the following manner. A BCD-code is set up on IC1 which consists of two four bit latches. Once the code is set up, a strobe pulse is applied to the enable input. This causes the data on the inputs to be latched to the outputs which drive IC2 and IC3. These are BCD to decimal decoder/drivers with open collector outputs capable of driving the 24V lamps directly. The lamps are connected via CN1 and CN3 and are once again ORed by diodes to serve as a lamp test.

The fault lamp signal is ANDed with an astable multivibrator to enable it to flash. The two power supply status lamps are driven by retriggerable one-shots. The execute lamp, cancel entry lamp, the piezo-electric alarm are directly driven by a 32bit o/p card on SABUS.

The back light power supply for the LCD dotmatrix display is shown in figure 3.4.6. The Schmitt trigger and RC circuit generates a pulse train at approximately 1kHz. The pulse, amplified the two transistors, which are driven 180 degrees out of phase, is fed into a step-up transformer. The output of the transformer, approximately 160V, is then connected to the LCD back light source.

3.4.2 FRONT PANEL SWITCHES

Figure 3.4.7 shows the layout design and figure 3.4.8 show the circuit diagram for the front panel. The parking loop is the only place from where a specific transporter can be called. For

this reason the 'FROM PARKING LOOP' switches are in a row and from the circuit diagram it can be seen that a BCD code is

generated when a pushbutton is pushed. The FROM and TO switches also generate a BCD code. All these signals are connected to the 80-bit opto-isolated I/P card on SABUS.

An adapter board was designed to concatenate all the signals and OR them by means of diodes. The three ORed signals then act as strobe lines. Figure 3.4.9 shows the circuit diagram for the adapter board.

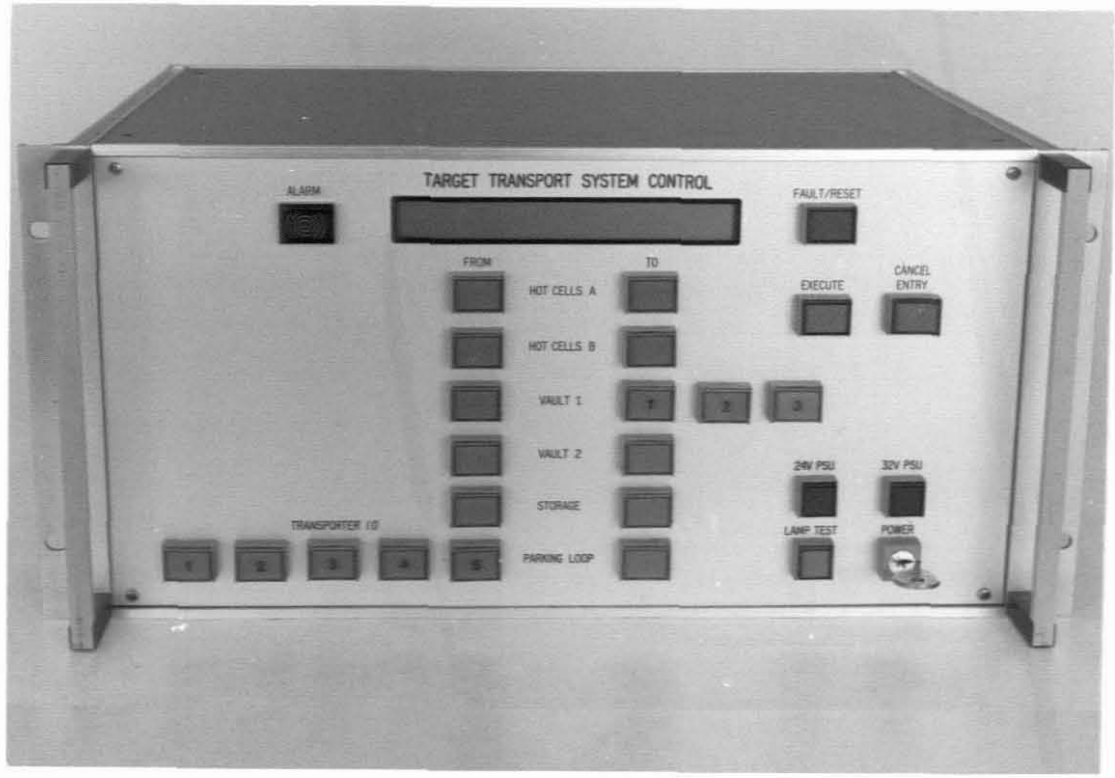


Fig. 3.4.1

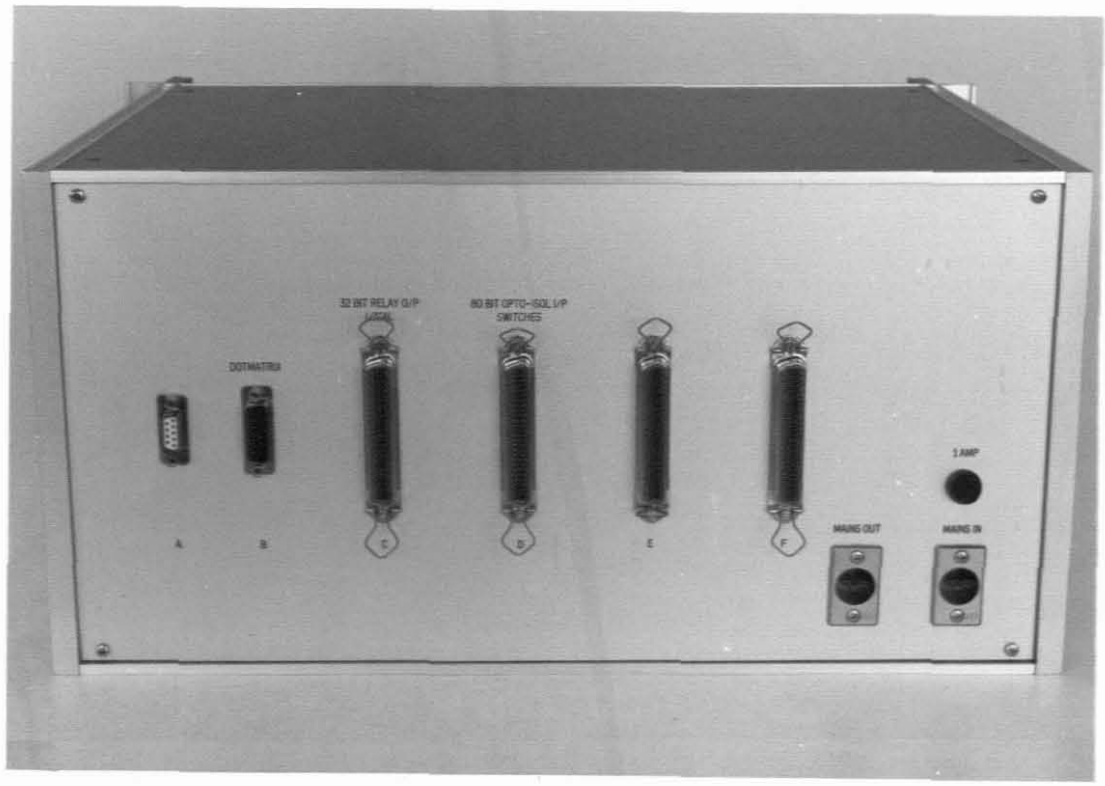
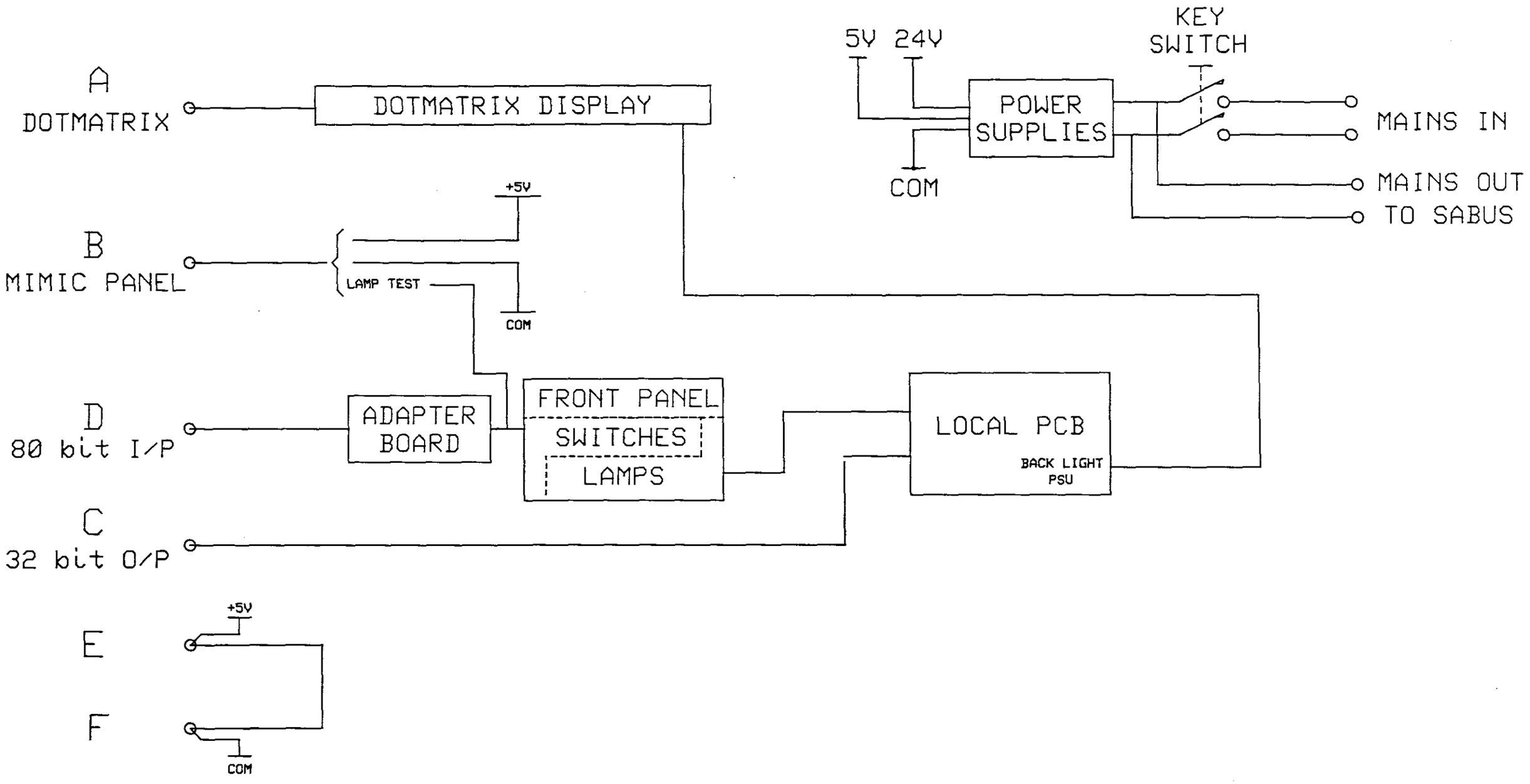


Fig. 3.4.2



PART NUMBER	
DESCRIPTION	LOCAL CONTROLLER BLOCK DIAGRAM
DRAWN	TERTIUS HUMAN
DATE	FEB 1988
FILENAME	LBLOCK.FSH

Fig. 3.4.3

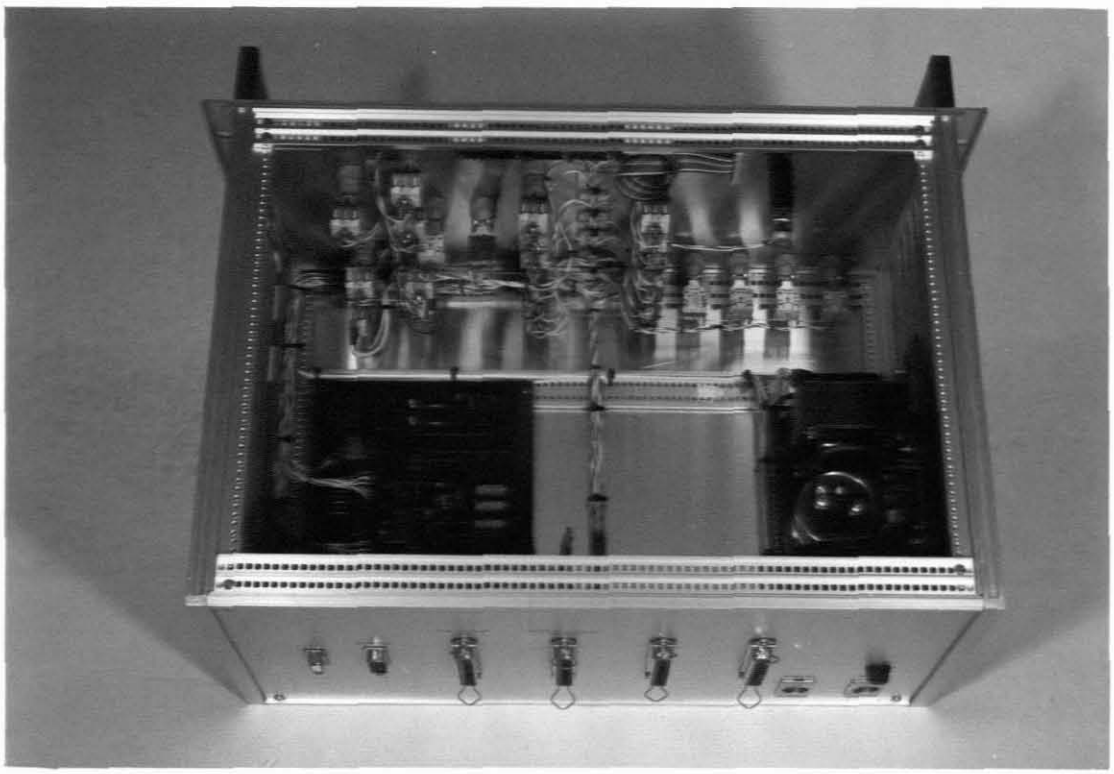


Fig. 3.4.4

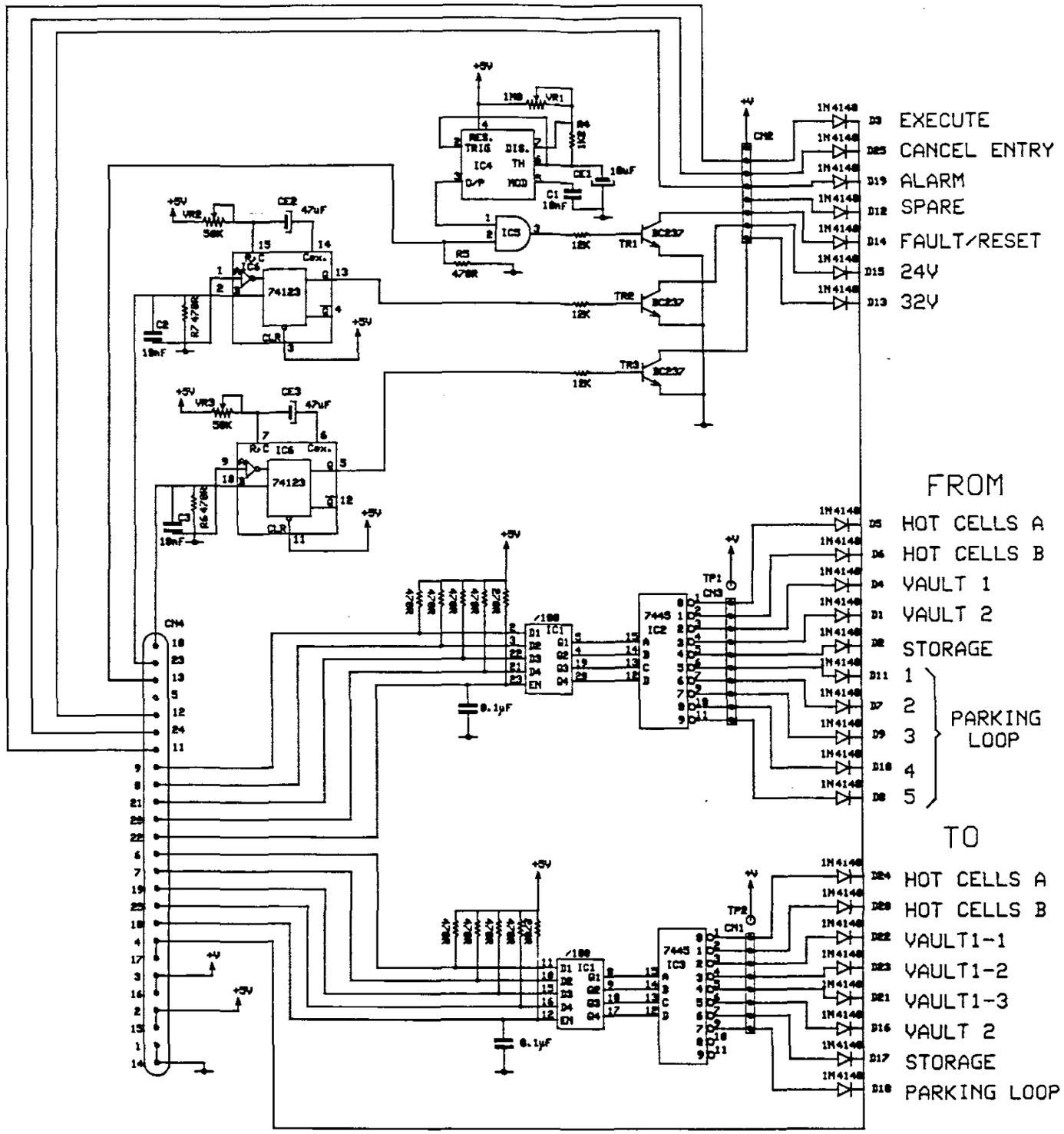
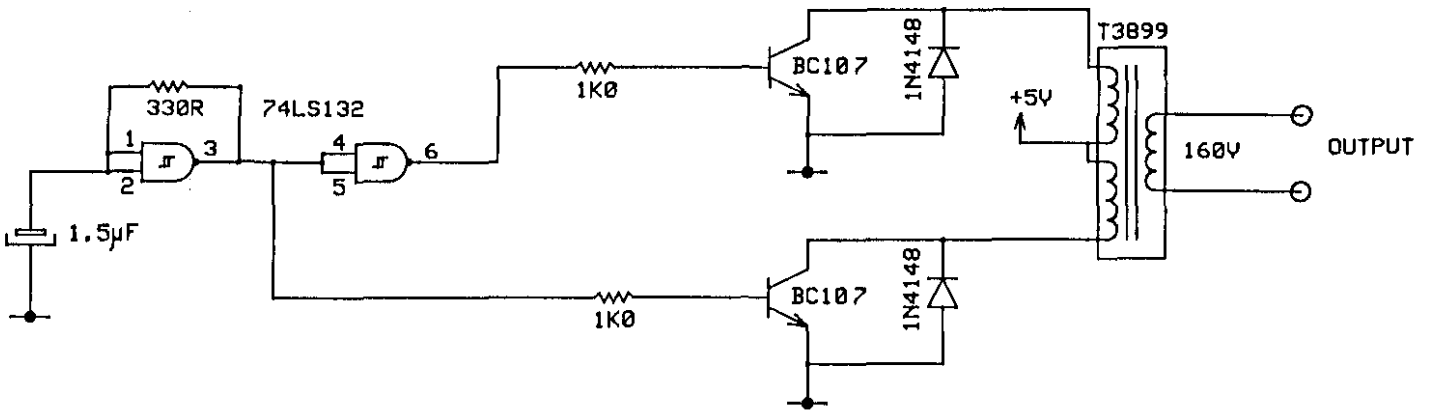

EXECUTE
CANCEL ENTRY
ALARM
SPARE
FAULT/RESET
24V
32V
FROM
HOT CELLS A
HOT CELLS B
VAULT 1
VAULT 2
STORAGE
1
2
3
4
5
PARKING LOOP
TO
HOT CELLS A
HOT CELLS B
VAULT1-1
VAULT1-2
VAULT1-3
VAULT 2
STORAGE
PARKING LOOP
PART NUMBER
EIP13-2D1C1/1
DESCRIPTION
LOCAL CONTROLLER
DRAWN
TERTIUS HUMAN
DATE
JUL 1988
FILENAME
LOCON.FSH

Fig. 3.4.5



PART NUMBER	
DESCRIPTION	LCD BACK LIGHT PSU
DRAWN	TERTIUS HUMAN
DATE	DEC 1987
FILENAME	BLI.FSH

Fig. 3.4.6

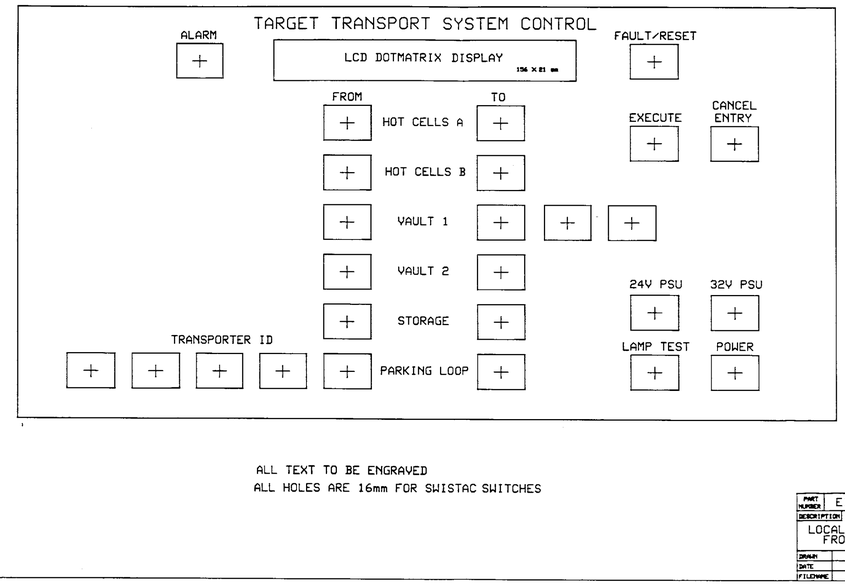
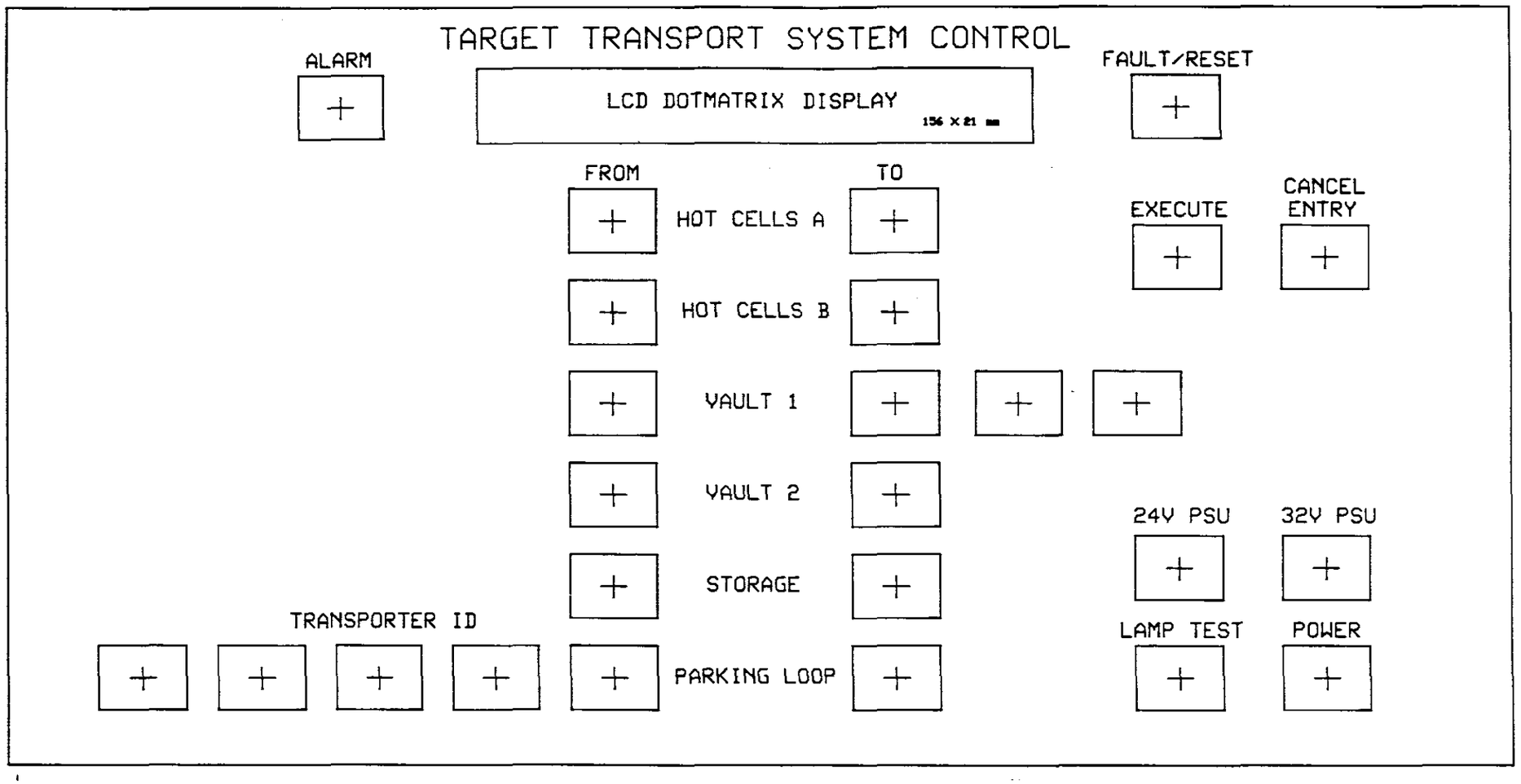

TARGET TRANSPORT SYSTEM CONTROL
ALARM
LCD DOTMATRIX DISPLAY
156 X 81 mm
FAULT/RESET
FROM
TO
HOT CELLS A
HOT CELLS B
VAULT 1
VAULT 2
STORAGE
PARKING LOOP
EXECUTE
CANCEL ENTRY
24V PSU
32V PSU
LAMP TEST
POWER
TRANSPORTER ID
ALL TEXT TO BE ENGRAVED
ALL HOLES ARE 16mm FOR SWISTAC SWITCHES
PART NUMBER
EIP13-2F1L1/1
DESCRIPTION
LOCAL CONTROLLER FRONT PANEL
DRAWN
TERTIUS NUPWA
DATE
JUL 1988
FILENAME
LFP.FIG

Fig. 3.4.7



ALL TEXT TO BE ENGRAVED
ALL HOLES ARE 16mm FOR SWISTAC SWITCHES

PART NUMBER	EIP13-2F1L1/1
DESCRIPTION	LOCAL CONTROLLER FRONT PANEL
DRAWN	TERTIUS HUMAN
DATE	JUL 1988
FILENAME	LFP.FSH

Fig. 3.4.7

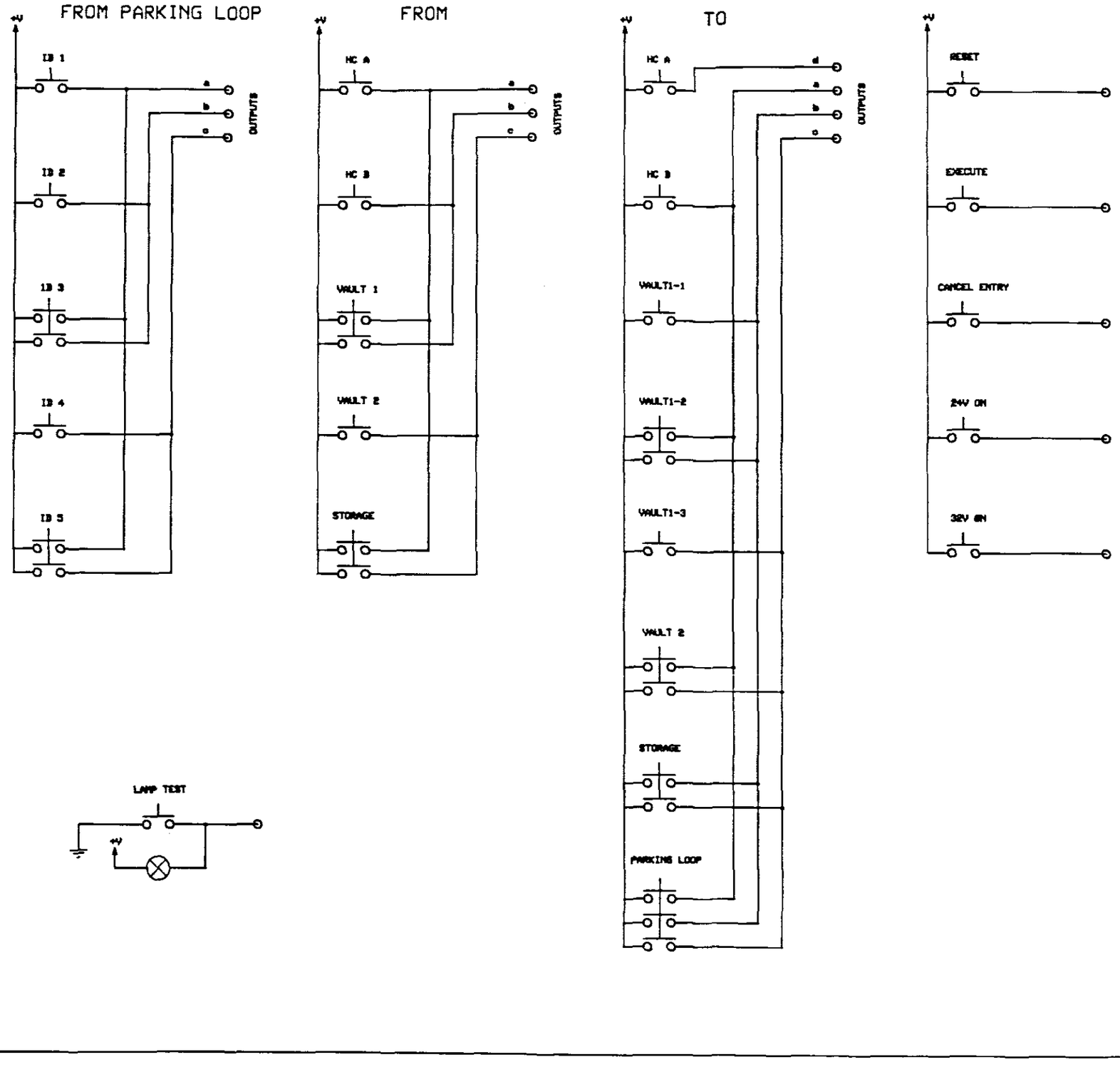

FROM PARKING LOOP
FROM
TO
+V
ID 1
ID 2
ID 3
ID 4
ID 5
a
b
c
OUTPUTS
HC A
HC B
VAULT 1
VAULT 2
STORAGE
VAULT1-1
VAULT1-2
VAULT1-3
PARKING LOOP
d
RESET
EXECUTE
CANCEL ENTRY
24V ON
32V ON
LAMP TEST
PART NUMBER
DESCRIPTION
LOCAL CONTROLLER
FRONT PANEL SWITCHES
DRAWN
TERTIUS HUMAN
DATE
JAN 1988
FILENAME
LOCSWI.FSH

Fig. 3.4.8

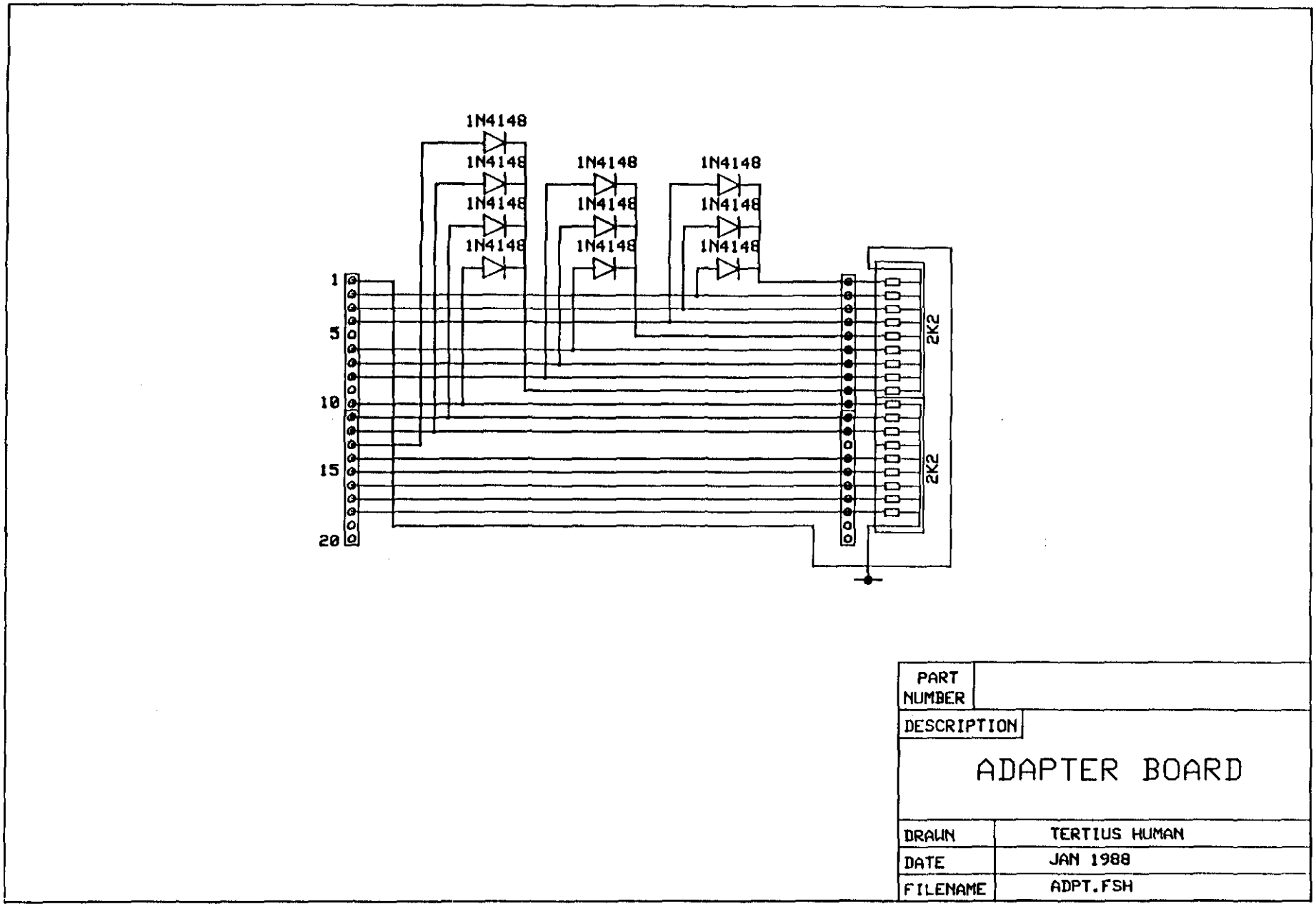

1N4148
1N4148
1N4148
1N4148
1N4148
1N4148
1N4148
1N4148
1N4148
1
5
10
15
20
2K2
2K2
PART NUMBER
DESCRIPTION
ADAPTER BOARD
DRAWN
TERTIUS HUMAN
DATE
JAN 1988
FILENAME
ADPT.FSH

Fig. 3.4.9

3.4.3 POWER SUPPLIES

Two power supplies are used for the local controller and mimic panel. A 5V supply drives the TTL circuitry and LED's and a 24V drives the lamps and the 80-bit I/P card.

3.5 THE HARDWARE CONTROLLER

As mentioned previously the transporters pick up the drive power from two copper rails in the tracks by means of brushes. The tracks are divided into different sections and figure 3.5.1 shows the track layout indicating the switchers and track sections. The sections are HA, HB, H, HS, V, V1, V2 and PL1 to PL10 in the parking loop. The function of the hardware controller is to switch the drive power to the track sections, the tracks on the switchers, the switcher motors and the parking loop track sections by means of relays. These relays are driven by the 24V power supply and controlled by the 32bit O/P cards on SABUS. When power is switched off on a section, both rails are at a positive potential. This serves as a brake for the transporters because it effectively short circuits the transporter motor. A third rail in the track is used for position sensing and the positive potential picked up by the power brushes is fed into this rail.

There are ten parking positions in the parking loop resulting into ten sections of track. Figure 3.5.2 shows the circuit diagram for the parking loop track power.

In order to drive a transporter in both directions, the power is applied in either direction to the track rails. This means that for every section of track two relays are used. Figure 3.5.3 shows the circuit diagram for the track power where the pairs of relays are clearly indicated. The method for 'power off - both rails at positive potential' can also be seen.

Figure 3.5.4 shows the circuit diagram for the switchers where the same configuration is used for the track power. An additional relay powers the switcher motor to drive it to the 0 degree or the 180 degree position.

The hardware controller is built into a cabinet marked PE1 which is situated near the parking loop. From here the cables carry the power to the transport system. The wiring diagram and photograph are shown in section 3.6.

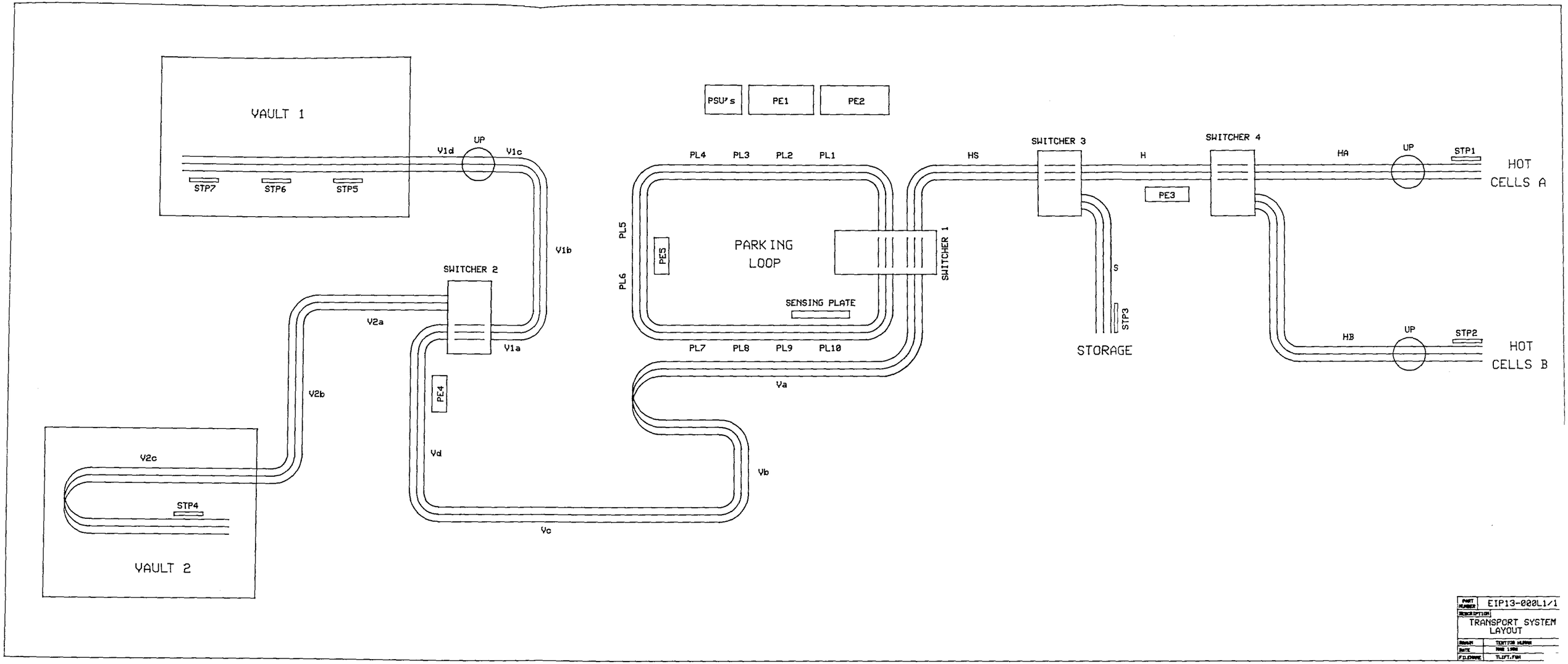
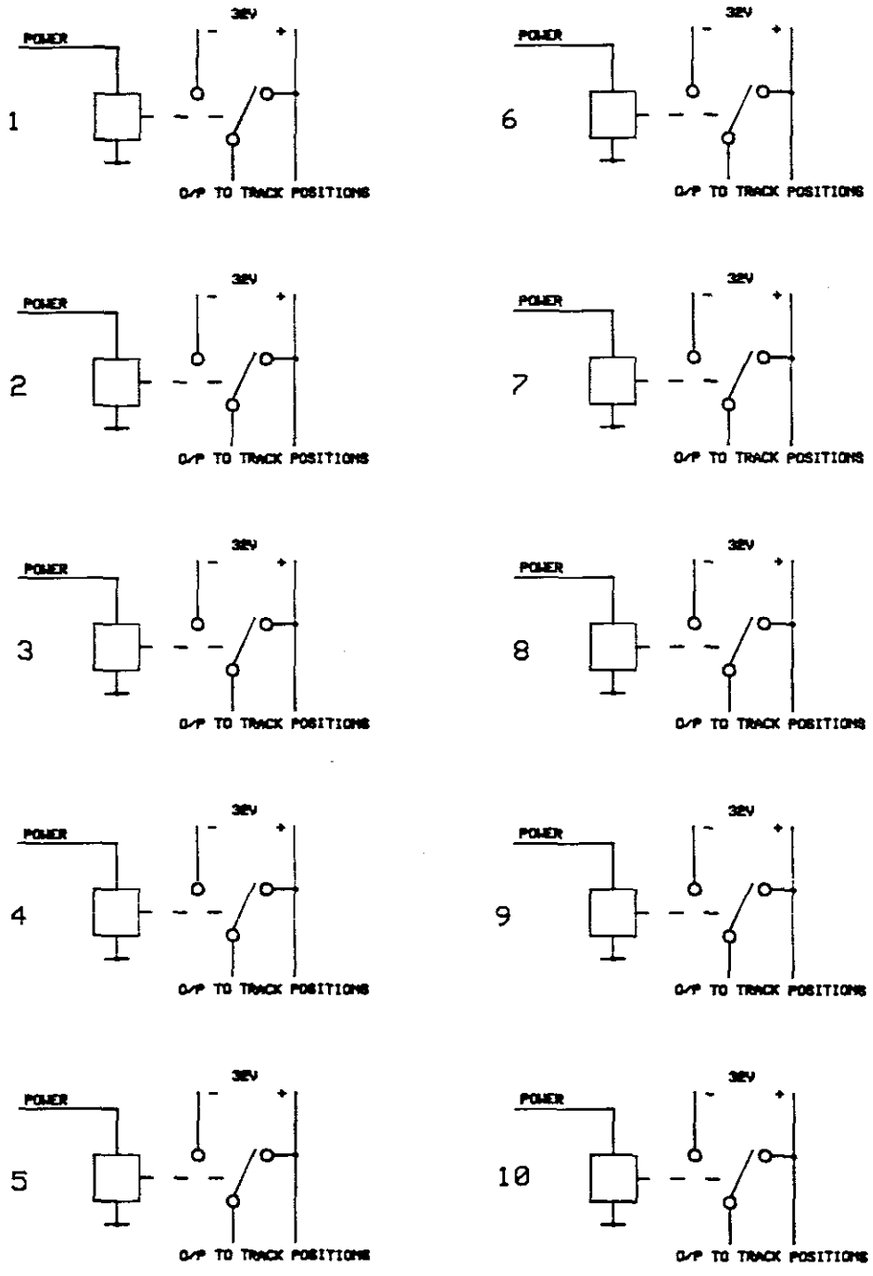
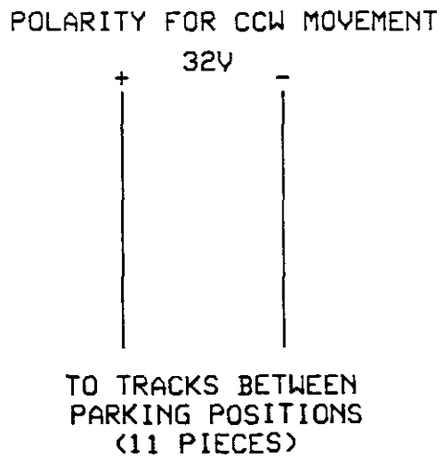

VAULT 1
STP7
STP6
STP5
V1d
UP
V1c
V1b
PSU's
PE1
PE2
PL4
PL3
PL2
PL1
HS
SWITCHER 3
H
SWITCHER 4
HA
UP
STP1
HOT CELLS A
PE3
PL5
PE5
PARKING LOOP
SWITCHER 1
S
STP3
STORAGE
SWITCHER 2
V2a
V1a
PL6
SENSING PLATE
PL7
PL8
PL9
PL10
HB
UP
STP2
HOT CELLS B
Va
PE4
V2b
Vd
Vb
V2c
STP4
Vc
VAULT 2
PART NUMBER
EIP13-000L1/1
DESCRIPTION
TRANSPORT SYSTEM LAYOUT
DRAWN
TEXTOR HUMAN
DATE
MAR 1990
FILENAME
TLIFT.FOH

Fig. 3.5.1





ALL INPUTS FROM
SABUS 32 BIT RELAY O/P

PART NUMBER	
DESCRIPTION	HARDWARE CONTROL PARKING LOOP
DRAWN	TERTIUS HUMAN
DATE	DEC 1987
FILENAME	HCPL.FSH

Fig. 3.5.2

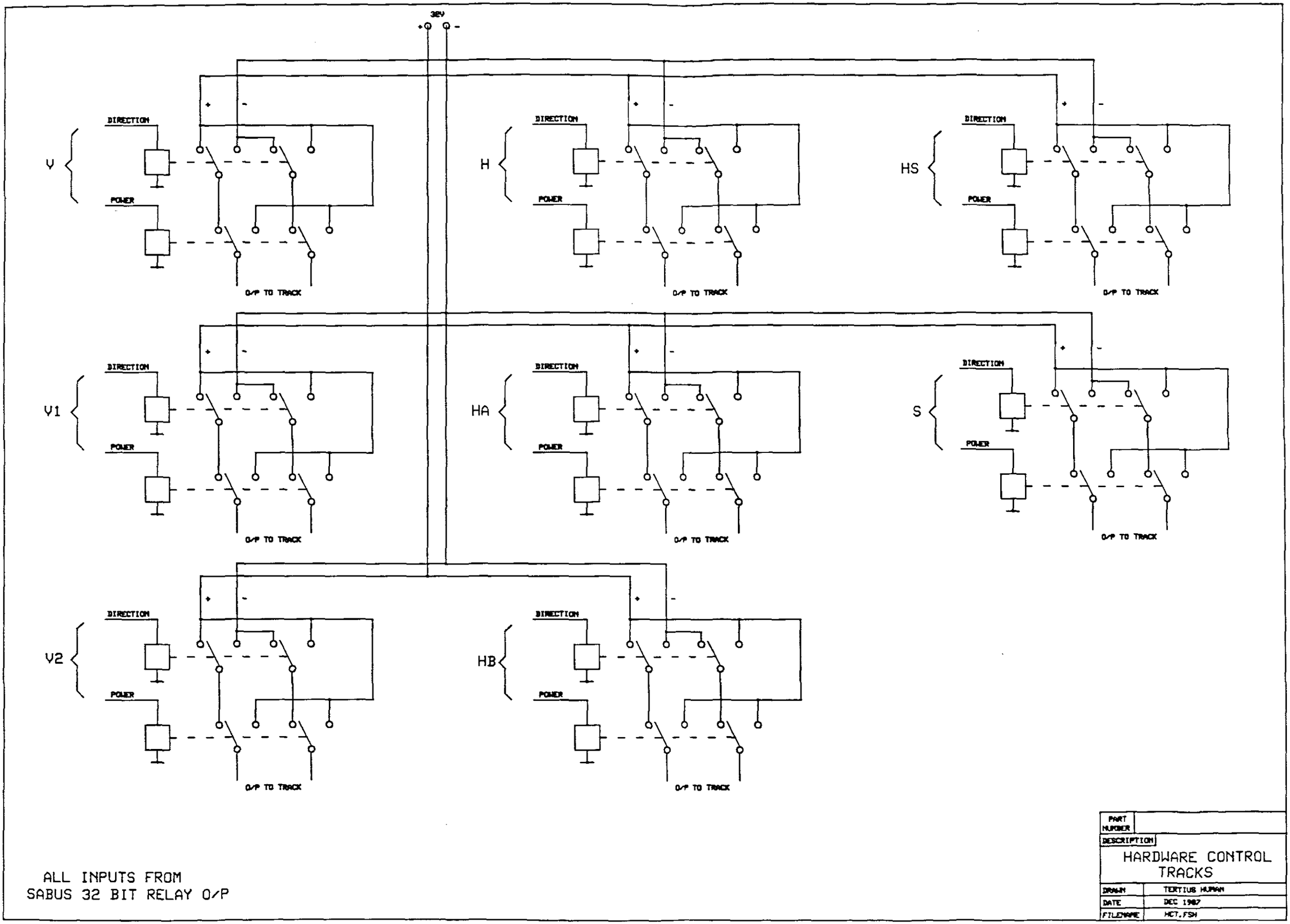
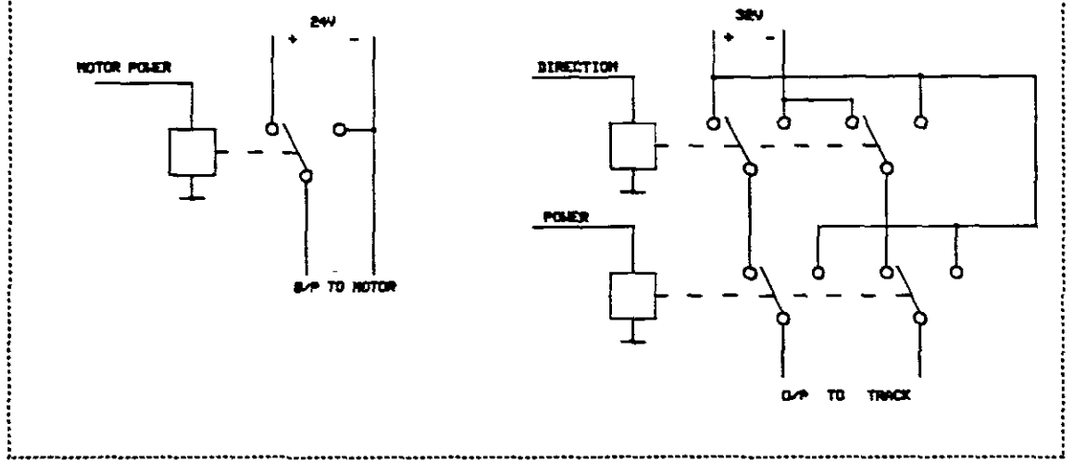
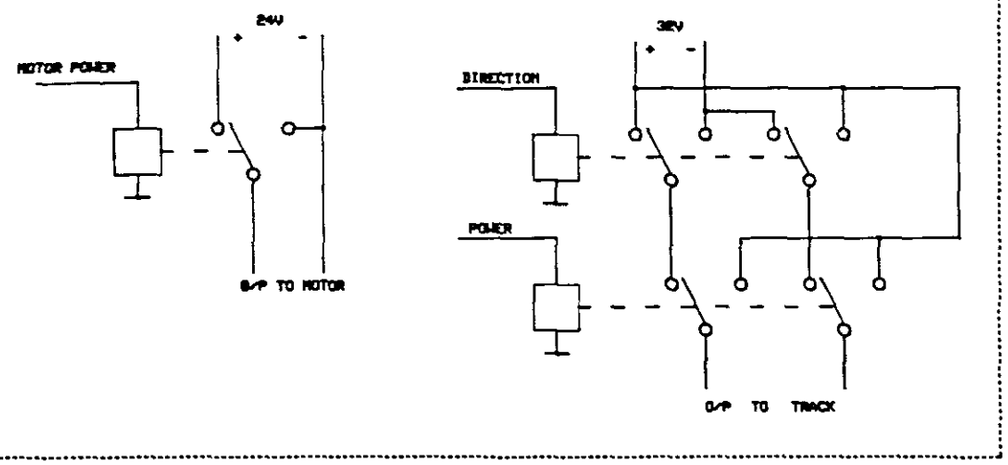

32V
+
-
V
DIRECTION
POWER
O/P TO TRACK
H
HS
V1
HA
S
V2
HB
ALL INPUTS FROM
SABUS 32 BIT RELAY O/P
PART NUMBER
DESCRIPTION
HARDWARE CONTROL TRACKS
DRAWN
TERTIUS HUMAN
DATE
DEC 1987
FILENAME
HCT.FSH

Fig. 3.5.3

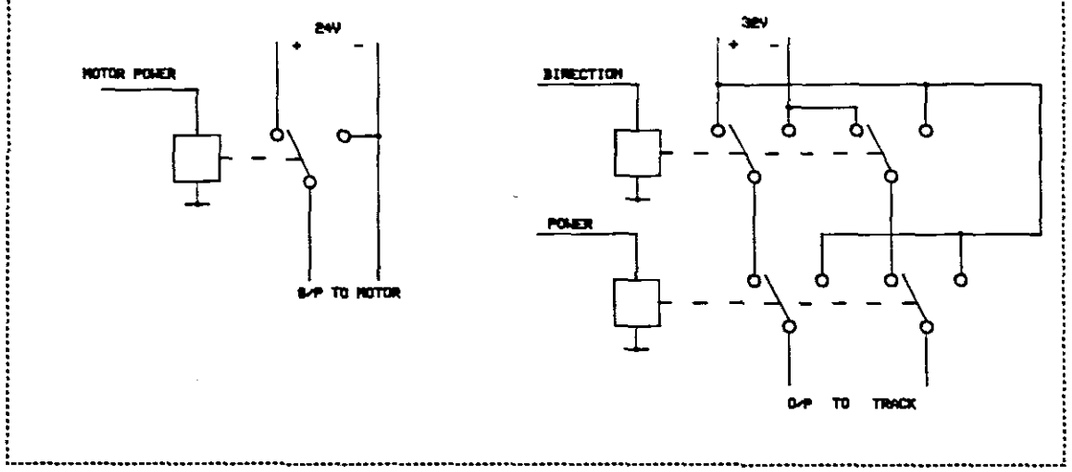
SWITCHER 2



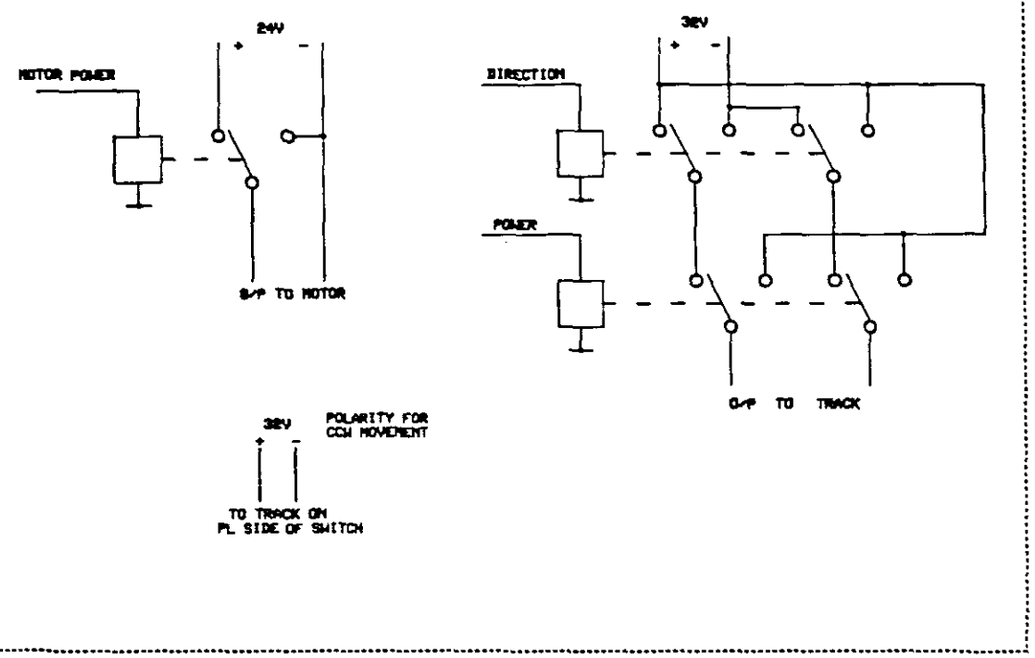
SWITCHER 3



SWITCHER 4



SWITCHER 1



PART NUMBER	
DESCRIPTION	HARDWARE CONTROL SWITCHERS
DRAWN	TERTIUS HUMAN
DATE	DEC 1987
FILENAME	HCS.FSN

Fig. 3.5.4

3.6 CABLING AND BREAK-OUT CABINETS

In order to keep wiring and cabling efficient, orderly and neat, two break-out cabinets were designed. All the inputs and outputs to and from the transport system are terminated in these cabinets. The signals that belong together are then concatenated and cabled out to the SABUS I/O cards.

Figures 3.6.1 and 3.6.2 show the inside of PE1 and PE2 respectively and figures 3.6.3 and 3.6.4 show the wiring diagrams. As mentioned previously the hardware controller forms an integral part of PE1. The position sensing signals come from every part of the transport system and they were also terminated in small break-out boxes. These break-out boxes, PE3, PE4 and PE5, are fixed on the tracks. The different signal wires are fed into the boxes from where it runs in a cable to PE2. Figure 3.6.5 shows PE5 fixed to the parking loop track. Figure 3.6.6 shows the inside where the incoming signal wires can be seen on the bottom left and the outgoing cable on the bottom right. Figure 3.6.7 shows the wiring diagrams for PE3, PE4 and PE5.

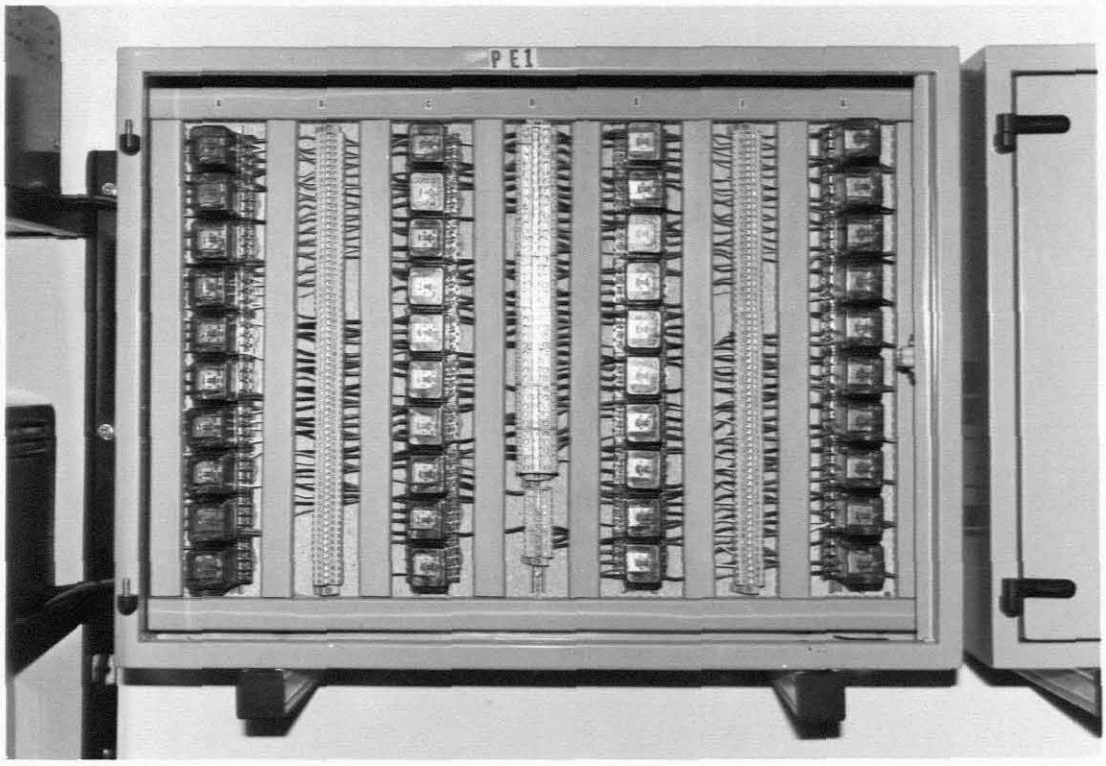


Fig. 3.6.1

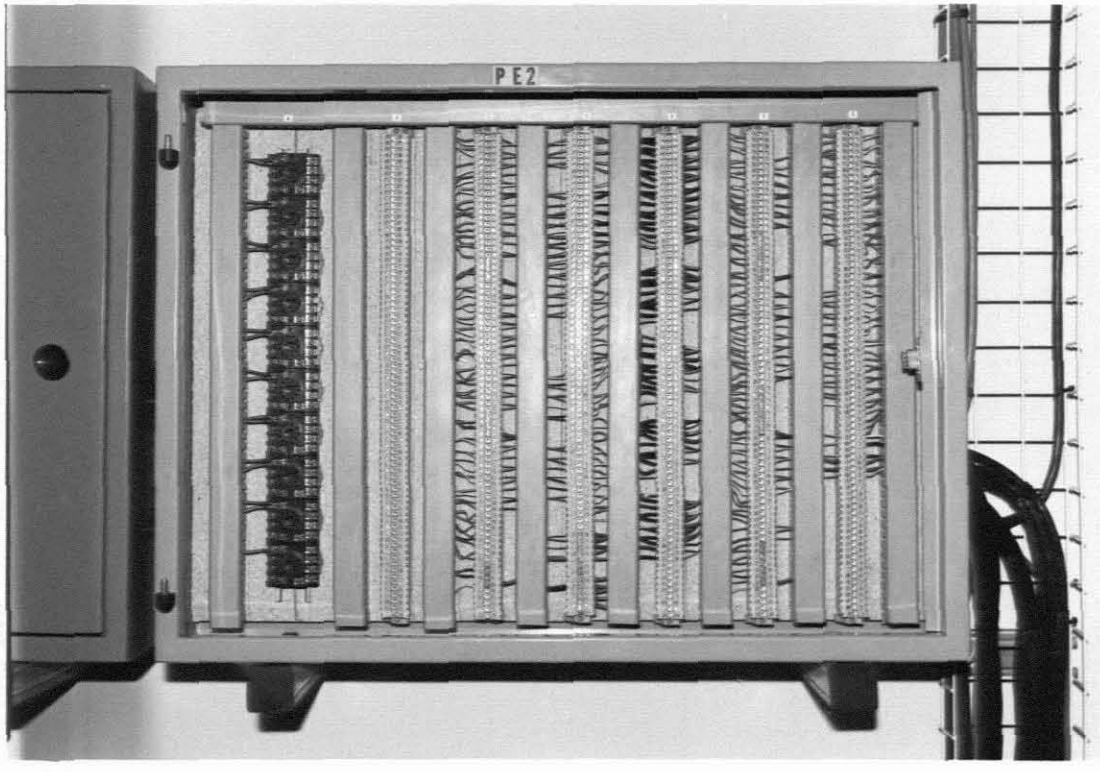


Fig. 3.6.2.

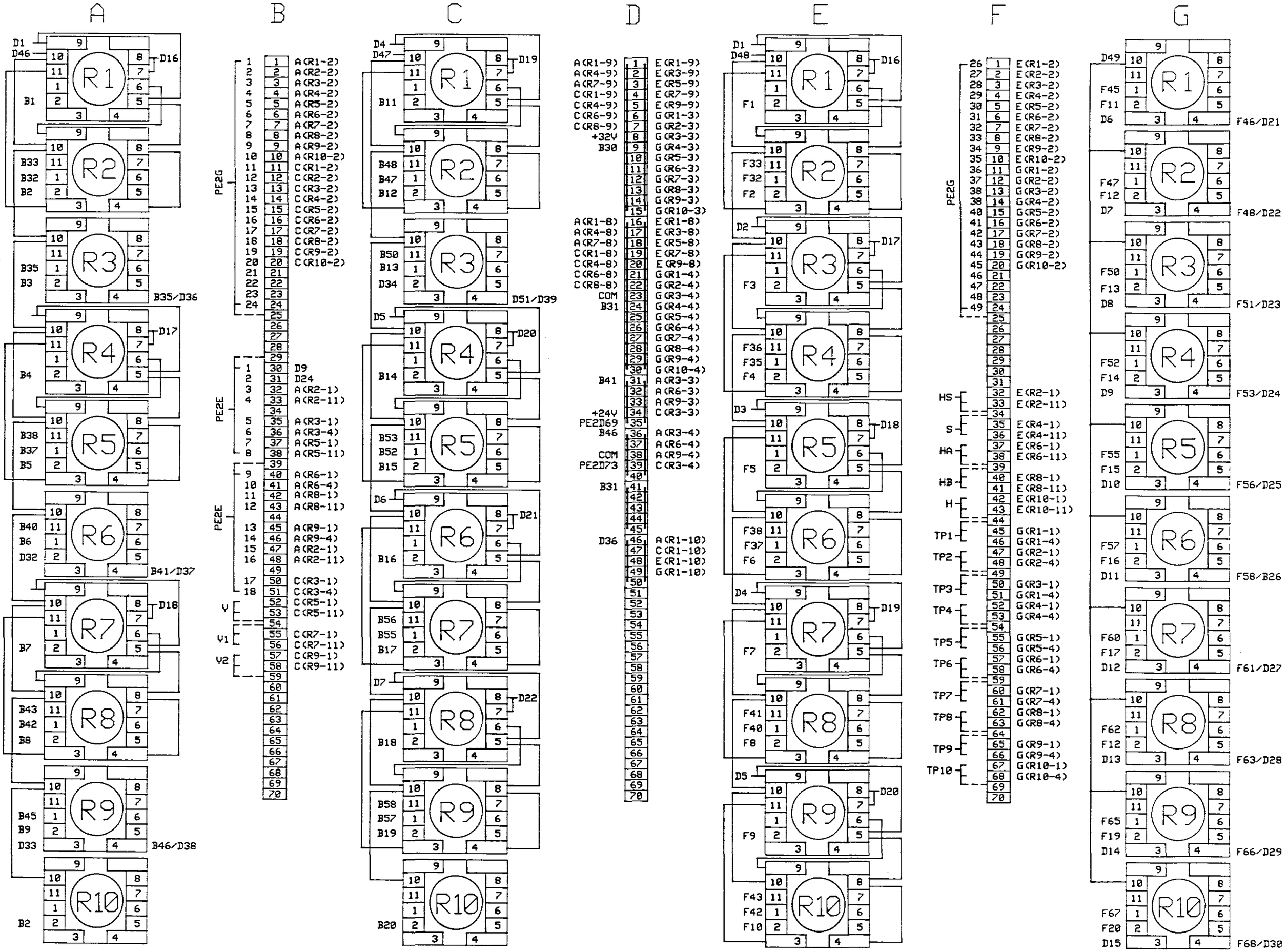

A
B
C
D
E
F
G
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
D1
D46
D16
B1
B33
B32
B2
B35
B3
B35/D36
D17
B4
B38
B37
B5
B40
B6
D32
B41/D37
D18
B7
B43
B42
B8
B45
B9
D33
B46/D38
PE2G
PE2E
A(R1-2)
A(R2-2)
A(R3-2)
A(R4-2)
A(R5-2)
A(R6-2)
A(R7-2)
A(R8-2)
A(R9-2)
A(R10-2)
C(R1-2)
C(R2-2)
C(R3-2)
C(R4-2)
C(R5-2)
C(R6-2)
C(R7-2)
C(R8-2)
C(R9-2)
C(R10-2)
D9
D24
A(R2-1)
A(R2-11)
A(R3-1)
A(R3-4)
A(R5-1)
A(R5-11)
A(R6-1)
A(R6-4)
A(R8-1)
A(R8-11)
A(R9-1)
A(R9-4)
A(R2-1)
A(R2-11)
C(R3-1)
C(R3-4)
C(R5-1)
C(R5-11)
C(R7-1)
C(R7-11)
C(R9-1)
C(R9-11)
V
V1
V2
D4
D47
D19
B11
B48
B47
B12
B50
B13
D34
D51/D39
D5
D20
B14
B53
B52
B15
D6
D21
B16
B56
B55
B17
D7
D22
B18
B58
B57
B19
B20
A(R1-9)
A(R4-9)
A(R7-9)
C(R1-9)
C(R4-9)
C(R6-9)
C(R8-9)
+32V
B30
A(R1-8)
A(R4-8)
A(R7-8)
C(R1-8)
C(R4-8)
C(R6-8)
C(R8-8)
COM
B31
B41
+24V
PE2D69
B46
COM
PE2D73
B31
D36
E(R1-9)
E(R3-9)
E(R5-9)
E(R7-9)
E(R9-9)
G(R1-3)
G(R2-3)
G(R3-3)
G(R4-3)
G(R5-3)
G(R6-3)
G(R7-3)
G(R8-3)
G(R9-3)
G(R10-3)
E(R1-8)
E(R3-8)
E(R5-8)
E(R7-8)
E(R9-8)
G(R1-4)
G(R2-4)
G(R3-4)
G(R4-4)
G(R5-4)
G(R6-4)
G(R7-4)
G(R8-4)
G(R9-4)
G(R10-4)
A(R3-3)
A(R6-3)
A(R9-3)
C(R3-3)
A(R3-4)
A(R6-4)
A(R9-4)
C(R3-4)
A(R1-10)
C(R1-10)
E(R1-10)
G(R1-10)
D1
D48
D16
F1
F33
F32
F2
D2
D17
F3
F36
F35
F4
D3
D18
F5
F38
F37
F6
D4
D19
F7
F41
F40
F8
D5
D20
F9
F43
F42
F10
PE2G
E(R1-2)
E(R2-2)
E(R3-2)
E(R4-2)
E(R5-2)
E(R6-2)
E(R7-2)
E(R8-2)
E(R9-2)
E(R10-2)
G(R1-2)
G(R2-2)
G(R3-2)
G(R4-2)
G(R5-2)
G(R6-2)
G(R7-2)
G(R8-2)
G(R9-2)
G(R10-2)
HS
S
HA
HB
H
TP1
TP2
TP3
TP4
TP5
TP6
TP7
TP8
TP9
TP10
E(R2-1)
E(R2-11)
E(R4-1)
E(R4-11)
E(R6-1)
E(R6-11)
E(R8-1)
E(R8-11)
E(R10-1)
E(R10-11)
G(R1-1)
G(R1-4)
G(R2-1)
G(R2-4)
G(R3-1)
G(R1-4)
G(R4-1)
G(R4-4)
G(R5-1)
G(R5-4)
G(R6-1)
G(R6-4)
G(R7-1)
G(R7-4)
G(R8-1)
G(R8-4)
G(R9-1)
G(R9-4)
G(R10-1)
G(R10-4)
D49
F45
F11
D6
F46/D21
F47
F12
D7
F48/D22
F50
F13
D8
F51/D23
F52
F14
D9
F53/D24
F55
F15
D10
F56/D25
F57
F16
D11
F58/B26
F60
F17
D12
F61/D27
F62
F12
D13
F63/D28
F65
F19
D14
F66/D29
F67
F20
D15
F68/D30

Fig 363

PF1

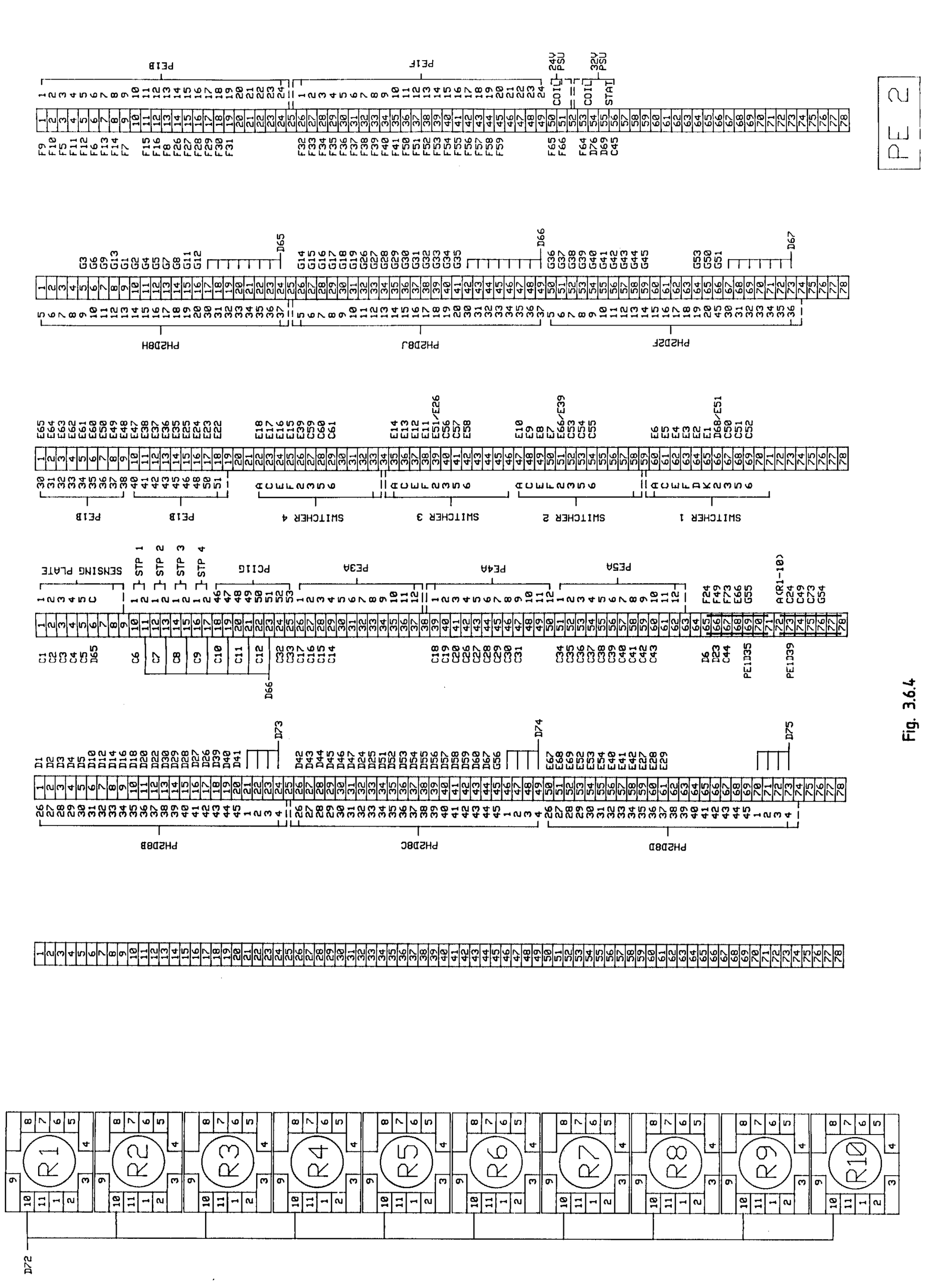

G
F
E
D
C
B
A
PE 2
PE1B
PE1F
24V PSU
32V PSU
COIL
STAT
F9 F10 F5 F11 F12 F6 F13 F14 F7 F15 F16 F8 F26 F27 F28 F29 F30 F31
F32 F33 F34 F35 F36 F37 F38 F39 F40 F41 F50 F51 F52 F53 F54 F55 F56 F57 F58 F59
F65 F66 F64 D76 D69 C45
D65 D66 D67
G3 G6 G9 G13 G1 G2 G4 G5 G7 G8 G11 G12
G14 G15 G16 G17 G18 G19 G26 G27 G28 G29 G30 G31 G32 G33 G34 G35
G36 G37 G38 G39 G40 G41 G42 G43 G44 G45
G53 G50 G51
PH2D8H
PH2D8J
PH2D2F
E65 E64 E63 E62 E61 E60 E50 E49 E48 E47 E38 E37 E36 E35 E25 E24 E23 E22
E18 E17 E16 E15 E39 C59 C60 C61
E14 E13 E12 E11 E51/E26 C56 C57 E58
E10 E9 E8 E7 E66/E39 C53 C54 C55
E6 E5 E4 E3 E2 E1 D68/E51 C50 C51 C52
PE1B
PE1B
SWITCHER 4
SWITCHER 3
SWITCHER 2
SWITCHER 1
SENSING PLATE
STP 1
STP 2
STP 3
STP 4
PC11G
PE3A
PE4A
PE5A
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
D72
C1 C2 C3 C4 C5 D65 C6 C7 C8 C9 C10 C11 C12 D66 C32 C33 C17 C16 C15 C14
C18 C19 C20 C26 C27 C28 C29 C30 C31
C34 C35 C36 C37 C38 C39 C40 C41 C42 C43
D6 D23 C44
PE1D35
PE1D39
F24 F49 F73 E66 G55
C24 C49 C73 G54
D1 D2 D3 D4 D5 D10 D12 D14 D16 D18 D20 D22 D30 D29 D28 D27 D26 D39 D40 D41
D42 D43 D44 D45 D46 D47 D24 D25 D51 D52 D53 D54 D55 D56 D57 D58 D59 D60 D67 G56
E67 E68 E69 E52 E53 E54 E40 E41 E42 E27 E28 E29
D73 D74 D75
PH2D8B
PH2D8C
PH2D8D

Fig. 3.6.4

Fig. 3.6.5

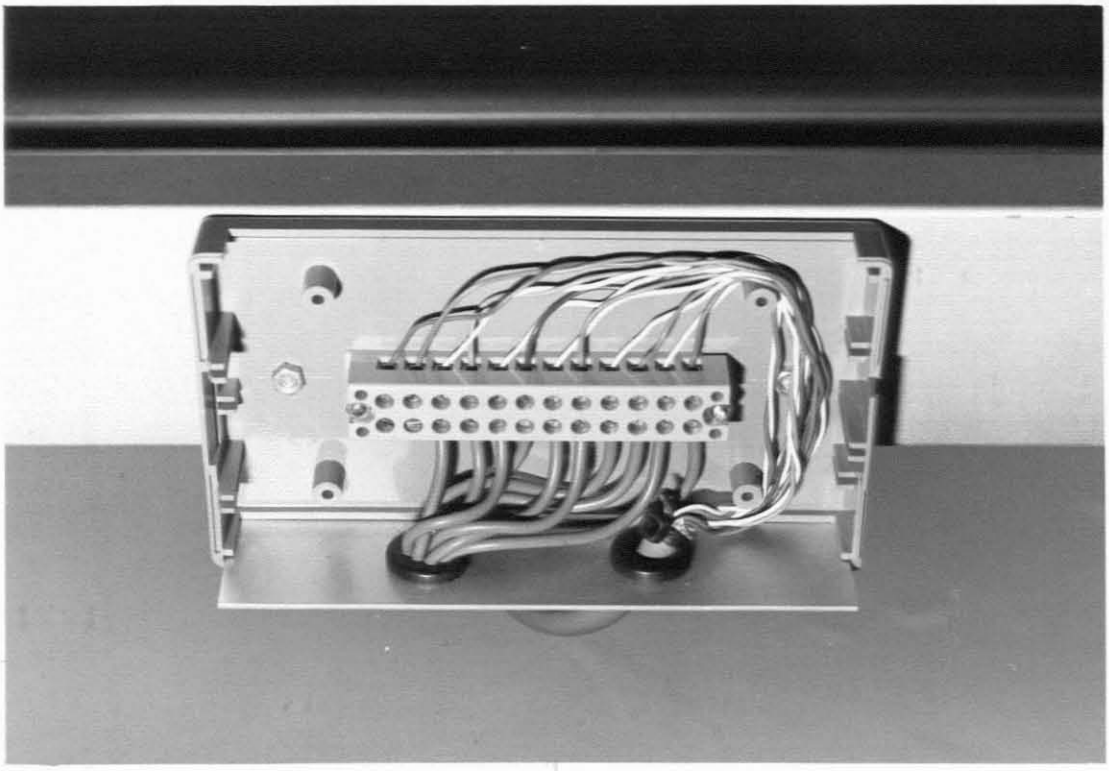
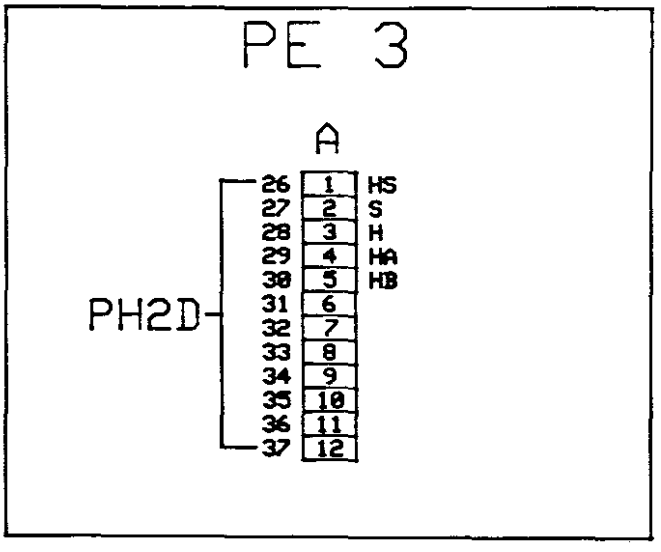
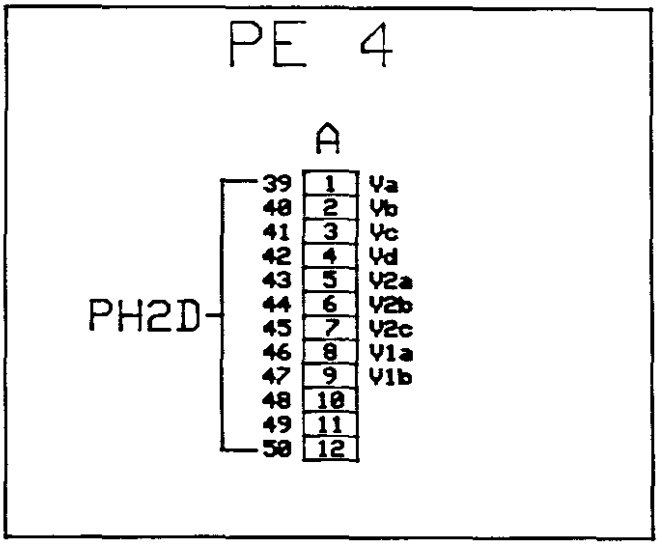


Fig. 3.6.6


PE 3
A
PH2D
26 1 HS
27 2 S
28 3 H
29 4 HA
30 5 HB
31 6
32 7
33 8
34 9
35 10
36 11
37 12


PE 4
A
PH2D
39 1 Va
40 2 Vb
41 3 Vc
42 4 Vd
43 5 V2a
44 6 V2b
45 7 V2c
46 8 V1a
47 9 V1b
48 10
49 11
50 12

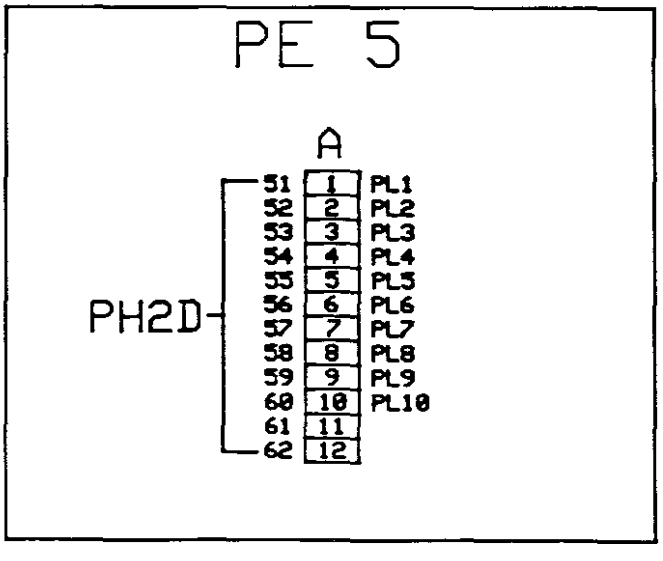

PE 5
A
PH2D
51 1 PL1
52 2 PL2
53 3 PL3
54 4 PL4
55 5 PL5
56 6 PL6
57 7 PL7
58 8 PL8
59 9 PL9
60 10 PL10
61 11
62 12

Fig. 3.6.7

4. ASSEMBLY

4.1 CABINETS

For the mimic panel and local controller two 4UT3 Elma cardframes were used. The amount of components used in each of these instruments does not justify such a big cabinet but the size was determined by the front panel components and layout thereof. On the local controller front panel and and on both the rear panels the appropriate information concerning switches and connecters were engraved. For the mimic display front panel a three dimensional drawing of the transport system was designed. The drawing was transferred onto a photosensitive material and fixed to the front plate.

For the break-out cabinets PE1 and PE2 two switchboard type, wall mounted cabinets were used. These were mounted on standoff brackets about 200mm away from the wall to allow for cable entry through the back. For PE3, PE4 and PE5 small vero boxes were used and these were mounted directly onto the tracks.

4.2 PRINTED CIRCUIT BOARDS

All the printed circuit boards were designed with the aid of a CAD system and no serious layout problems were encountered.

4.3 CABLING AND WIRING OF THE TRANSPORT SYSTEM

All the cabling and wiring from the transport system to PE1 and PE2 were done by using standard inhouse cables. Once these were terminated, the six cables connecting PE2 to SABUS were laid over a route of approximately 100m.

5. SOFTWARE DESCRIPTION

5.1 SPECIFICATIONS

The function of the Z-80 microprocessor is to read the 80-bit opto-isolator input card and the GPIB interface card and evaluate this data. Once a decision is made on what action is to be taken, new data is written to the appropriate output module. These can be the 32-bit relay output cards, the dotmatrix display interface, the bubble memory or the GPIB card.
The various inputs and outputs, from and to the transport system, are assigned to the I/O modules. Each module has a unique address on the SABUS which makes every bit unique. The bit assignments, the module addresses and the function of each bit are shown in section 5.2. In section 5.3 the data evaluation and the actions to be taken is discussed in the program design.

5.2 BIT ASSIGNMENTS

The 80-bit opto-isolator input card consists out of four banks of opto-isolators. Each bank of twenty bits has its' own connector. This results in ten groups of one byte each. The addresses assigned to these are from $60 to $69 (the $ sign indicates hexadecimal notation). Tables one and two show the inputs assigned to the bits and also some connector information.

The three 32-bit relay output cards consist out of two banks of read relays. Each bank of sixteen relays has its' own connector resulting in four groups of one byte each. The assigned addresses are from $30 to $3B and tables three, four and five show the outputs assigned to the bits and connector information.

80-bit OPTO-ISOLATOR I/P CARD PORT $60 - $64

CONNECTOR	PH2D8A (Local control)		PH2D8B	
PIN No.	BIT	FUNCTION	BIT	FUNCTION
26	60.0	FROM PL STROBE	63.0	SEP 1
27	60.1	a	63.1	SEP 2
		FROM		
28	60.2	b	63.2	SEP 3
		PARKING LOOP		
29	60.3	c	63.3	SEP 4
30	60.4	FROM STROBE	63.4	SEP 5
31	60.5	a	63.5	STP 1
		FROM		
32	60.6	b	63.6	STP 2
		CODE		
33	60.7	c	63.7	STP 3
34	61.0	TO STROBE	64.0	STP 4
35	61.1	a	64.1	STP 5
		TO		
36	61.2	b	64.2	STP 6
		CODE		
37	61.3	c	64.3	STP 7
38	61.4		64.4	H2
39	61.5	RESET	64.5	H1
40	61.6	EXECUTE	64.6	H
41	61.7	CANCEL ENTRY	64.7	S
42	62.0	24V ON	62.4	HS
43	62.1	32V ON	62.5	Va
44	62.2		62.6	Vb
45	62.3		62.7	Vc

PINS 21-25,46-50 NOT CONNECTED
PINS 1-20 = 0V (COM)

table 1

80-bit OPTO-ISOLATOR I/P CARD PORT $65 - $69

CONNECTOR	PH2D8C		PH2D8D		
PIN No.	BIT	FUNCTION	BIT	FUNCTION	
26	65.0	Vd	68.0	STP	
27	65.1	V2a	68.1	0 DEG.	SW1
28	65.2	V2b	68.2	180 DEG.	
29	65.3	V2c	68.3	STP	
30	65.4	V1a	68.4	0 DEG.	SW2
31	65.5	V1b	68.5	180 DEG.	
32	65.6	V1c	68.6	STP	
33	65.7	V1d	68.7	0 DEG.	SW3
34	66.0	PL1	69.0	180 DEG.	
35	66.1	PL2	69.1	STP	
36	66.2	PL3	69.2	0 DEG.	SW4
37	66.3	PL4	69.3	180 DEG.	
38	66.4	PL5	69.4		
39	66.5	PL6	69.5		
40	66.6	PL7	69.6		
41	66.7	PL8	69.7		
42	67.0	PL9	67.4		
43	67.1	PL10	67.5		
44	67.2	24V PSU	67.6		
45	67.3	32V PSU	67.7		

PINS 21-25,46-50 NOT CONNECTED
PINS 1-20 = 0V (COM)

table 2

32-bit RELAY O/P CARD PORT $30 - $33

CONNECTOR	PH2D8E (MLEDS)			PH2D8F (MLEDS)	
PIN No.	BIT		FUNCTION	BIT	FUNCTION
20	30.0	a		32.0	PL 10
19	30.1	b	TRACK POSITION	32.1	PL 9
18	30.2	c	CODE	32.2	PL 8
17	30.3	d		32.3	PL 7
16	30.4	T P strobe		32.4	PL 6
15	30.5	T P enable		32.5	PL 5
14	30.6			32.6	PL 4
13	30.7			32.7	PL 3
12	31.0	a	XP_ID	33.0	PL 2
11	31.1	b	CODE	33.1	PL 1
10	31.2	c		33.2	SW 1
9	31.3	a		33.3	SW 2
8	31.4	b	STATION	33.4	SW 3
7	31.5	c	CODE	33.5	SW 4
6	31.6	d		33.6	
5	31.7	STATION STROBE		33.7	

PINS 1-4,26-29 NOT CONNECTED

PINS 30-45 = 0V

table 3

32-bit RELAY O/P CARD PORT $34 - $37

CONNECTOR	P2D8G (Localcontrol)			PH2D8H(Switchers)	
PIN No.	BIT	FUNCTION		BIT	FUNCTION
20	34.0	a		36.0	T POWER SW 1
19	34.1	b	FROM	36.1	DIRECTION SW 1
18	34.2	c	CODE	36.2	T POWER SW 2
17	34.3	d		36.3	DIRECTION SW 2
16	34.4	a		36.4	T POWER SW 3
15	34.5	b	TO	36.5	DIRECTION SW 3
14	34.6	c	CODE	36.6	T POWER SW 4
13	34.7	d		36.7	DIRECTION SW 4
12	35.0	TO STROBE		37.0	M POWER SW 1
11	35.1	EXECUTE		37.1	M POWER SW 2
10	35.2	CANCEL ENTRY		37.2	M POWER SW 3
9	35.3	ALARM		37.3	M POWER SW 4
8	35.4	FROM STROBE		37.4	
7	35.5	FAULT/RESET		37.5	
6	35.6	24V ON		37.6	
5	35.7	32V ON		37.7	

PINS 1-4,26-29 NOT CONNECTED

P2D8G PINS 30-33,37-45 = +5V
PINS 34-36 = 0V

P2D8H PINS 30-45 = +24V

table 4

32-bit RELAY O/P CARD PORT $38 - $3B

CONNECTOR	PH2D8J (Tracks)		PH2D8K (Parking loop)	
PIN No.	BIT	FUNCTION	BIT	FUNCTION
20	38.0	POWER H	3A.0	24V PSU ON
19	38.1	DIRECTION	3A.1	32V PSU ON
18	38.2	POWER H2	3A.2	
17	38.3	DIRECTION	3A.3	
16	38.4	POWER H1	3A.4	
15	38.5	DIRECTION	3A.5	
14	38.6	POWER S	3A.6	TP 10
13	38.7	DIRECTION	3A.7	TP 9
12	39.0	POWER HS	3B.0	TP 8
11	39.1	DIRECTION	3B.1	TP 7
10	39.2	POWER V2	3B.2	TP 6
9	39.3	DIRECTION	3B.3	TP 5
8	39.4	POWER V1	3B.4	TP 4
7	39.5	DIRECTION	3B.5	TP 3
6	39.6	POWER V	3B.6	TP 2
5	39.7	DIRECTION	3B.7	TP 1

PINS 1-4,26-29 NOT CONNECTED

PH2D8J PINS 30-45 = +24V

PH2D8K PINS 30-44 = +24V,PIN 45 = +24V in Local controller

table 5

5.3 PROGRAM DESIGN

5.3.1 START UP

```
Initialize the dotmatrix display.
     8-bit data length
     Cursor movement to the right.
     Display data RAM address = $80.
     Clear display and return cursor to top left.
     Set the cursor invisible.
Initialize all the variables.
Bleep once.
Display : 'THE TARGET TRANSPORT SYSTEM
           Switch on the Power Supplies.'
Poll the inputs from the local controller.
     24V on : $62.0       32V on : $62.1
When it goes true, switch on the appropriate power supply.
     24V : $40.0          32V : $40.1
Read the power supply status $67.2 and $67.3.
Indicate on the local controller lamps $35.6 and $35.7.
Display  '24V Power Supply on.
          32V Power Supply on. '

Read the position status of the switchers.
0  degrees :   SW1:$68.1   SW2:$68.4   SW3:$68.7   SW4:$69.2
If they are not all at 0 degrees then
call the routine to drive it to 0 deg.
Display : ' Switchers will now be tested
            Observe the Mimic LED's.  '
Call the routines to drive the switchers to 180 degrees.
Indicate on Mimic panel LED,s.
Call the routine to drive the switchers to 0 degrees.
Indicate on Mimic panel LED's.
If a switcher does not respond then display :
     ' Switcher no.x is faulty
       RESET to continue. '
Bleep three times.
When RESET, drive the switcher until  it responds.
If all responded then display :
     ' All four switchers are O K. '

Display : ' Parking Loop positions will now be checked.
            Observe the Mimic Panel. '
Read the parking loop positions.
   $66.0 .1 .2 .3 .4 .5 .6 .7 .8 .9 $67.0 .1
Determine which ones are occupied.
Indicate on mimic panel LED's.
  $32.1 .2 .3 .4 .5 .6 .7 $33.0 .1
Determine if transporters are positioned from position 10
downwards with no gaps.
If not display :
     ' Transporters will now be positioned. '
Call the positioning routine.
```

```
Display : ' Stations will now be checked and updated.
           Observe the Mimic Panel. '
Read from the bubble memory the stations and ID's parked
in that stations.
Check the stations and determine which ones are occupied.
  $63.5 .6 .7 $64.0 .1 .2 .3
Compare the station check to the bubbles file.
If the bubble shows a station occupied but the station check
did not find one there, discard the information.
If the check shows a station occupied but the bubble does not
then display a 0 in that station.
If the check and the bubble matches, then display the bubble's
ID code in that station.
To display data :
                  Set up 7-segment data      $31.0 .1 .2
                  Set up the station code    $31.3 .4 .5 .6
                  Strobe                     $31.7
                  Set the station code to 0000
                  Strobe again.
Bleep once.

Display : ' Stray transporter check. '
Check if a transporter is in any position other than in the
parking loop or in a station by reading all the track positions.
     $64.4 .5 .6 .7
     $62.4 .5 .6 .7
     $65.0 .1 .2 .3 .4 .5 .6 .7
If one is found display :
     ' A stray transporter was found on the tracks.
       RESET to continue. '
Bleep until the RESET button is pressed.
Read the RESET button from the local controller.
If pressed stop the bleeper.
Display : ' Transporter will now be driven to the parking loop.
           Press EXECUTE. '
Read the EXECUTE button from the local controller.
If pressed call the routine to drive the transporter to the
parking loop.
Display : ' Check the number of transporters in the system = 5
          (RESET) '
Wait for RESET.
Display : ' Make sure one is not jammed on the tracks. (RESET) '
Wait for RESET.
Display : '----------------READY----------------
           Enter a FROM and a TO position. '
```

5.3.2 THE MAIN POLLING ROUTINE.

```
The inputs to be polled are :
     The stations           $63.5 --> $64.3
     The parking loop       $66.0 --> $67.1
     The power supplies     $67.2 .3
     The switchers          $68.1 .4 .7 $69.2
     The local controller   $60.0 .4 $61.0 (3 strobe lines)

Read the bytes one by one.
Compare it to its' previous value.
If it is the same read again.
If it changed, call the appropriate service routine.

Stations service routine.
Determine which station.
Determine if a transporter was removed or inserted.
If removed display :
     ' Transporter was removed from station. '
Write a blank to the seven segment display.
Update the bubble memory.
If inserted display :
     'Transporter was inserted into station. '
Write a seven to the seven segment display.
Update the bubble memory.
Resume polling.

Parking loop service routine.
Determine which position.
Determine if transporter was removed or inserted.
Display : 'Transporter was removed/inserted in the parking loop.'
Update the mimic panel LED's.
Call the routine to position the transporters.
Resume polling.

Power supplies polling routine.
Determine which power supply failed  $67.2 $67.3
Update local controller lamps  $35.6 $35.7
Fault lamp on $35.5
Bleep until reset.
Display : 'xxV Power Supply off.
          RESET to continue. '
Check for RESET.
Silence bleeper.
Display : 'First check and then switch on the Power Supply. '
Check $62.0 $62.1
Switch the power supply on  $40.0 $40.1
Check the status $67.2 $67.3
If still of go back to beginning.
If on : fault lamp off
      : Update local controller lamps.
Bleep once.
```

```
Display ready.
Resume polling.

Switchers position polling routine.
Determine which switcher is out of position.
Bleep until RESET.
Fault lamp on.
Mimic LED to red.
Display : ' Switcher no.x is out of position.
            RESET to continue. '
Check for RESET.
Silence bleeper.
Display : 'Check the switcher before you RESET to continue.'
Call routine to drive switcher to 0 degrees.
Fault lamp off.
Display ready
Resume polling.


Local controller polling routine.
Determine which bit changed.
 FROM strobe : $60.4
    Read FROM strobe :  $60.5 .6 .7
    Store the code.
    Indicate on local controller lamps.
    Resume polling.
 FROM Parking Loop strobe : $60.0
    Read FROM P L code  $60.1 .2 .3
    Store the code.
    Indicate on local controller lamps
    Resume polling.
 TO strobe : $61.0
    Read TO code  $61.1 .2 .3
    Store the code.
    Indicate on local controller lamps.
    Resume polling.
 CANCEL ENTRY : $61.7
    Switch off all lamps on local controller.
    Delete all stored values.
    Resume polling.
 EXECUTE : $61.7
    Check the stored FROM value.
    Check the stored FROM Parking Loop value.
    If both are 0 then
      Bleep once.
      Display : 'No FROM entered. '
      Display ready.
      Resume polling.
    Check the stored TO value.
      If value is 0 then
      Bleep once.
      Display : 'No TO entered. '
```

```
      Resume polling.
    If FROM and TO values <> 0 then
      Check if TO = FROM.
      If true then
      Bleep once.
      Display : 'FROM and TO the same position. '
      Display ready.
      Resume polling.
    If FROM <> TO then
      Check if FROM position is occupied by a transporter.
      If unoccupied then
      Bleep once.
      Display : 'The FROM position entered is empty. '
      Display ready.
      Resume polling.
      Check if TO position is unoccupied.
      If occupied then
      Bleep once.
      Display : 'The TO position entered is occupied.
      Display ready.
      Resume polling.
 If all the above conditions are met then TO and FROM will
 have legal values at this stage.
 Display : ' Transporter now on route ----------------> '
 Call the routine to drive the transporters.
```

5.3.3 THE ROUTINE TO DRIVE THE TRANSPORTERS

```
From Hot Cells A.
Set up the track power.
Set up the switcher track power.
Update the bubble memory : Hot cells A is now empty.
Update the Mimic panel.

From Hot Cells B.
Set up the track power.
Set up the switcher track power.
Update the bubble memory : Hot cells B is now empty.
Update the Mimic panel.
Drive switcher 2 to 180 degrees.
Indicate progress on Mimic panel.
Poll switcher 2 stopping plate for transporter arrival.
On arrival switcher 2 track power off.
Drive switcher 2 to 0 degrees.
Determine TO code and set up switcher 2 track power accordingly.

From Vault 1
Vault  1 has three stopping positions.,  so first determine if TO
code is in vault 1.
If so set up track power accordingly.
Else set up track power to drive transporter to Hot cells A.
```

```
Update the bubble memory.
Update the Mimic panel.

From Vault 2.
Set up the track power.
Set up the switcher track power.
Update the bubble memory : Vault 2 is now empty.
Update the Mimic panel.
Drive switcher 4 to 180 degrees.
Indicate progress on Mimic panel.
Poll switcher 4 stopping plate for transporter arrival.
On arrival switcher 4 track power off.
Drive switcher 4 to 0 degrees.
Determine TO code and set up switcher 4 track power accordingly.

From Storage.
Set up the track power.
Set up the switcher track power.
Update the bubble memory : Storage is now empty.
Update the Mimic panel.
Drive switcher 3 to 180 degrees.
Indicate progress on Mimic panel.
Poll switcher 3 stopping plate for transporter arrival.
On arrival switcher 3 track power off.
Drive switcher 3 to 0 degrees.
Determine TO code and set up switcher 3 track power accordingly.

From Parking Loop.
Start the first transporter in the parking loop and read its' ID
 on the sensing plate.
If it is not the wanted ID drive it to the back of the loop.
Rearrange the transporters so that the next one is in the first
 parking position.
Drive the next one past the sensing plates and read its' ID.
If it is still not the wanted ID repeat the above.
If the ID is the same as the very first one the rearrange and
 display : ' The ID entered was not found in the Parking Loop.'
             RESET to continue. '
Check for RESET and go to main polling routine.
If the correct ID is found then :
Drive switcher 1 to 180 degrees.
Set up the switcher track power.
Poll switcher 1 stopping plate for arrival.
On arrival switcher 1 track power off.
Drive switcher 1 to 0 degrees.
Set up the track power.
Set up the switchers track power.

To Hot Cells A.
Set up the track power.
Indicate the transporter progress on the Mimic panel.
Poll hot cells A stopping plate for arrival.
```

```
On arrival switch off all the track power.
Update the Mimic panel.
Update the bubble memory.

To Hot Cells B.
Set up the track power.
Indicate the transporter progress on the Mimic panel.
Poll switcher 2 stopping plate for arrival.
On arrival switcher 2 track power off.
Drive switcher 2 to 180 degrees.
Set up switcher 2 track power.
When transporter has left the switcher then
 drive switcher 2 to 0 degrees.
Indicate the transporter progress on the Mimic panel.
Poll hot cells B stopping plate for arrival.
On arrival switch off all the track power.
Update the Mimic panel.
Update the bubble memory.

To Vault 1-1,2,3.
Set up the track power.
Indicate the transporter progress on the Mimic panel.
Poll the appropriate stopping plate for arrival.
On arrival switch off all the track power.
Update the Mimic panel.
Update the bubble memory.

To Vault 2.
Set up the track power.
Indicate transporter progress on the Mimic panel.
Poll switcher 4 stopping plate for arrival.
On arrival switcher 4 track power off.
Drive switcher 4 to 180 degrees.
Set up switcher 4 track power.
When transporter has left the switcher then
 drive switcher 4 to 0 degrees.
Indicate the transporter progress on the Mimic panel.
Poll Vault 2 stopping plate for arrival.
On arrival switch off all the track power.
Update the Mimic panel.
Update the bubble memory.

To Storage.
Set up the track power.
Indicate the transporter progress on the Mimic panel.
Poll switcher 3 stopping plate for arrival.
On arrival switcher 3 track power off.
Drive switcher 3 to 180 degrees.
Set up switcher 3 track power.
When transporter has left the switcher then
 drive switcher 3 to 0 degrees.
Indicate the transporter progress on the Mimic panel.
```

Poll Storage stopping plate for arrival.
On arrival switch off all the track power.
Update the Mimic panel.
Update the bubble memory.

To Parking Loop.
Set up the track power.
Indicate the transporter progress on the Mimic panel.
Poll switcher 1 stopping plate for arrival.
On arrival switcher 1 track power off.
Drive switcher 1 to 180 degrees.
Set up switcher 1 track power.
When transporter has left the switcher then
drive switcher 1 to 0 degrees.
Set up track power for parking position 10.
On arrival switch off all the track power.
Rearrange the parking loop transporters.
Update the Mimic panel.

At this stage the transporter would have reached its' destination and the program can go back to the main polling routine.

5.4 THE PROGRAM

The program consists of a main program and thirteen include files. This makes it modular, easy to understand and easy to maintain. A printout of the whole program follows on the next page.

```
PROGRAM MAIN;

{ MAIN is the program to control the isotope production target transport
  system.  It was written during 1988 by Tertius Human.                  }

{GLOBAL DECLERATIONS          GLOB_DEC.PAS}

CONST
  dot_matrix = $5Ø;

TYPE
  string8Ø = STRING[8Ø];

VAR
  dollar3Ø,dollar31,dollar32,dollar33,
  dollar34,dollar35,dollar36,dollar37,
  dollar38,dollar39,dollar3A,dollar3B : BYTE;

  test6Ø,test61,test62,test63,test64,
  test65,test66,test67,test68,test69 : BYTE;

  dot_matrix_control,dot_matrix_data : BYTE;
  mess : string8Ø;

  cntr,green_led,n,from_code,to_code : INTEGER;
  switcher_no,motor_power,
  switcher_destination,switcher_pos_status : BYTE;
  switcher_faulty : BOOLEAN;


 stations_poll_1,stations_poll_2,stations_store_1,stations_store_2,
 parking_loop_store_1,parking_loop_poll_1,
 parking_loop_store_2,parking_loop_poll_2 : BYTE;

{PRORAM MISCELLANEOUS         MISCEL.PAS

This program does the following:

a)Sounds the alarm
  Alarm port := $35   Alarm bit := $F7  1111 Ø111

b)Checks the RESET from the Local controller
  Port $61     reset bit := $2Ø   11Ø1 1111

c)Checks the EXECUTE from the Local controller
  Port $61     execute bit := $4Ø   1Ø11 1111
}


CONST
 alarm = $F7;



PROCEDURE bleep;
 BEGIN
  dollar35 := dollar35 AND $F7;
  PORT[$35] := dollar35;
  DELAY(2ØØ);
  dollar35 := dollar35 OR 8;
  PORT[$35] := dollar35;
 END;
```

```
FUNCTION local_reset : BOOLEAN;
 BEGIN
  IF (PORT[$61] AND $2Ø) = Ø THEN
   BEGIN
    local_reset := TRUE;
    dollar35 := dollar35 AND $DF;
    PORT[$35] := dollar35;          {Lamp on}
    DELAY(3ØØ);
    dollar35 := dollar35 OR $2Ø;
    PORT[$35] := dollar35;          {Lamp off}
   END
  ELSE local_reset := FALSE;
 END;


FUNCTION local_execute : BOOLEAN;
 BEGIN
  local_execute := (PORT[$61] AND $4Ø) = Ø;
 END;



PROCEDURE bleep_until_reset;
 BEGIN
  dollar35 := dollar35 AND alarm;
  PORT[$35] := dollar35;
  REPEAT
  UNTIL local_reset;
  dollar35 := dollar35 OR ($FF - alarm);
  PORT[$35] := dollar35;
 END;

{PROGRAM INITIAL           INITIAL.PAS

 This program:

a)Initializes the dotmatrix display.

  function_set : sets 8 bit data lenght
  entry_mode_set : sets cursor movement to right
  DD_ram : sets Display Data RAM address
  clear : clears display , returns cursor to top left
  curs_off : sets cursor invisible

b)Initializes all the variables when power is switched on.

 }


CONST
  function_set = $38;
  entry_mode_set = 6;
  DD_ram = $8Ø;
  clear = 1;
  curs_off = 12;

PROCEDURE dotmatrix_init_write (func : BYTE);
 BEGIN
  PORT[dot_matrix_control] := func;
```

```
  DELAY(1);
END;



PROCEDURE dotmatrix_initialize;
 BEGIN
  dot_matrix_control := dot_matrix;
  dotmatrix_init_write(function_set);
  dotmatrix_init_write(curs_off);
  dotmatrix_init_write(entry_mode_set);
  dotmatrix_init_write(clear);
  dotmatrix_init_write(DD_ram);
 END;



PROCEDURE variables_initialize;
 BEGIN
  dollar30 := $E0;
  dollar31 := $00;
  dollar32 := $FF;
  dollar33 := $FF;
  dollar34 := $FF;
  dollar35 := $EE;
  dollar36 := $FF;
  dollar37 := $FF;
  dollar38 := $FF;
  dollar39 := $FF;
  dollar3A := $FB;
  dollar3B := $FF;
  PORT[$30] := dollar30;
  PORT[$31] := dollar31;
  PORT[$32] := dollar32;
  PORT[$33] := dollar33;
  PORT[$34] := dollar34;
  PORT[$35] := dollar35;
  PORT[$36] := dollar36;
  PORT[$37] := dollar37;
  PORT[$38] := dollar38;
  PORT[$39] := dollar39;
  PORT[$3A] := dollar3A;
  PORT[$3B] := dollar3B;
 END;

{PROGRAM DOTMATRX         DOTMATRX.PAS

This program writes data to the dotmatrix display.
}


FUNCTION ready : BOOLEAN;
{To test the busy flag}
 BEGIN
  ready := (PORT[dot_matrix_control + 2] < $80);
 END;

PROCEDURE clear_rest (ind : INTEGER);
 {Writes blanks , $20 , in the rest of the dotmatrix display}
 VAR
  index : INTEGER;
 BEGIN
```

```
  FOR index := ind TO 79 DO
  BEGIN
   REPEAT
   UNTIL ready;
   IF ready THEN
    BEGIN
     PORT[dot_matrix_data] := $AØ;  {space = $AØ}
    END;
  END;
 END;

PROCEDURE write_dotmatrix (mess : string8Ø);
 {Writes the message , mess , to the dotmatrix}
 VAR
  index : INTEGER;
 BEGIN
  PORT[dot_matrix_control] := clear;
  dot_matrix_data := dot_matrix + 1;
  FOR index := 1 TO Length(mess) DO
  BEGIN
   REPEAT
   UNTIL ready;
   IF ready THEN
    BEGIN
     PORT[dot_matrix_data] :=ORD(mess[index]);
    END;
  END;
  clear_rest(1+Length(mess));
 END;


{PROGRAM  POWER SUPPLIES          POW_SUP.PAS

This program polls the inputs from the Local Controller concerning
the 24V & 32V Power Supplies. When either of these bits change it
switches the appropriate P S on via the relay outputs. It then checks
the status of the power supplies. If both are not on, it will stay in
the loop untill true. When true it will update the Mimic Panel lamps,
display the message and exit.
The following PORT addresses are used:
 Status
    24V on status : $67.2    1111 1Ø11
    32V on status : $67.3    1111 Ø111

 Command from Local Controller
    24V on : $62.Ø           1111 111Ø
    32V on : $62.1           1111 11Ø1

 Output to switch Power Supply on
    24V on : $3A.Ø           1111 111Ø
    32V on : $3A.1           1111 11Ø1

 Output to Local Controller indication lamps
    24V on : $35.6           1Ø11 1111
    32V on : $35.7           Ø111 1111
}


 CONST
 ps24V_on_output = $FE;
 ps32V_on_output = $FD;
 ps24V_on_lamp = $BF;
```

```
ps32V_on_lamp = $7F;

VAR
ps24V_status,ps32V_status,switch_ps_on : INTEGER;
ps24V_on,ps32V_on : BOOLEAN;



FUNCTION ps_on_command : BOOLEAN;
 BEGIN
  IF (PORT[$62] AND 1 = Ø) OR (PORT[$62] AND 2 = Ø) THEN
  BEGIN
   ps_on_command := TRUE;
  WRITELN('PS_ON_COMMAND = TRUE');
  END;
 END;



PROCEDURE ps_on_24V;
 BEGIN
  WRITELN('24V on out');
  ps24V_on := FALSE;
  dollar3A := dollar3A AND ps24V_on_output;
  PORT[$3A] := dollar3A;
  DELAY(1ØØ);
  ps24V_status := PORT[$67] AND 4;
  IF ps24V_status = Ø THEN
  BEGIN
   dollar35 := dollar35 AND ps24V_on_lamp;
   PORT[$35] := dollar35;
   ps24V_on := TRUE;
   WRITELN('24V Power supply on.');
  END;
 END;



PROCEDURE ps_on_32V;
 BEGIN

  ps32V_on := FALSE;
  dollar3A := dollar3A AND ps32V_on_output;
  PORT[$3A] := dollar3A;
  DELAY(1ØØ);
  ps32V_status := PORT[$67] AND 8;
  IF ps32V_status = Ø THEN
  BEGIN
   dollar35 := dollar35 AND ps32V_on_lamp;
   PORT[$35] := dollar35;
   ps32V_on := TRUE;
  END;
 END;



PROCEDURE power_supplies;
 BEGIN
  WHILE NOT(ps24V_on AND ps32V_on) DO
  BEGIN
   REPEAT UNTIL ps_on_command;
   IF ps_on_command THEN
```

```
   BEGIN
    switch_ps_on := PORT[$62] AND 3;
    CASE switch_ps_on OF
     2 : ps_on_24V;
     1 : ps_on_32V;
     Ø : BEGIN
          ps_on_24V;
          ps_on_32V;
         END;
    END;
   END;
 END;
 mess := ' 24V Power Supply on                    32V Power Supply on';
 write_dotmatrix(mess);
END;
```

```
{PROGRAM Switcher Driver          SW_DRV.PAS

This program drives a switcher to Ø deg. or 18Ø deg. If the switcher does
not respond in a certian time, the variable switcher_faulty is returned
true. It also updates the switcher status on the Mimic Panel leds.

Ports used

Mimic panel switcher LED's  = $33

green_led    1Ø  SW1  red      $FB
             11  SW1  green    4
             2Ø  SW2  red      $F7
             21  SW2  green    8
             3Ø  SW3  red      $EF
             31  SW3  green    $1Ø
             4Ø  SW4  red      $DF
             41  SW4  green    $2Ø


switcher_no:Port for motor power   motor_power:bit for motor power

        SW1      $37                      $FE     1111 111Ø
        SW2      $37                      $FD     1111 11Ø1
        SW3      $37                      $FB     1111 1Ø11
        SW4      $37                      $F7     1111 Ø111


switcher_position_status:Port      switcher_destination:bit
     SW1        $68                Ø deg     2    ØØØØ ØØ1Ø
                                   18Ø deg   4    ØØØØ Ø1ØØ
     SW2        $68                Ø deg     $1Ø  ØØØ1 ØØØØ
                                   18Ø deg   $2Ø  ØØ1Ø ØØØØ
     SW3        $68                Ø deg     $8Ø  1ØØØ ØØØØ
                $69                18Ø deg   1    ØØØØ ØØØ1
     SW4        $69                Ø deg     4    ØØØØ Ø1ØØ
                                   18Ø deg   8    ØØØØ 1ØØØ

}
```

```
PROCEDURE switcher_faulty_redrive;FORWARD;


FUNCTION time_out : BOOLEAN;
BEGIN
 cntr := SUCC(cntr);
 DELAY(1);
 IF cntr >= 5000 THEN                          {Approximately 5 seconds}
  time_out := TRUE
 ELSE time_out := FALSE;
END;

PROCEDURE update_mleds_switcher;
 BEGIN
  CASE green_led OF
   10 : dollar33 := dollar33 OR 4;
   40 : dollar33 := dollar33 OR 8;
   30 : dollar33 := dollar33 OR $10;
   20 : dollar33 := dollar33 OR $20;
   11 : dollar33 := dollar33 AND $FB;
   41 : dollar33 := dollar33 AND $F7;
   31 : dollar33 := dollar33 AND $EF;
   21 : dollar33 := dollar33 AND $DF;
  END;
  PORT[$33] := dollar33;
 END;


PROCEDURE drive_switcher;

 VAR
  degrees : BYTE;
  switcher_fin : BOOLEAN;

 BEGIN
  cntr := 0;
  switcher_fin := FALSE;
  time_out := FALSE;

  dollar37 := motor_power AND dollar37;          {Motor Power on}
  PORT[$37] := dollar37;

  REPEAT
   degrees := PORT[switcher_pos_status] AND switcher_destination;
   IF degrees = 0 THEN
   switcher_fin := TRUE;
  UNTIL  switcher_fin OR time_out;               {Check for dest. or time out

  dollar37 := dollar37 OR ($FF - motor_power);
  PORT[$37] := dollar37;                         {Motor Power off}
  update_mleds_switcher;

  IF time_out THEN
  switcher_faulty_redrive;
 END;


PROCEDURE switcher_faulty_redrive;

 VAR
  reset_in : BYTE;
  reset : BOOLEAN;
```

```
BEGIN
 reset := FALSE;
 bleep;
 DELAY(200);
 bleep;
 DELAY(200);
 bleep;
 CASE motor_power OF
  $FE : mess := ' Switcher no.1 is faulty                    RESET to continue
  $FD : mess := ' Switcher no.2 is faulty                    RESET to continue
  $FB : mess := ' Switcher no.3 is faulty                    RESET to continue
  $F7 : mess := ' Switcher no.4 is faulty                    RESET to continue
 END;
 write_dotmatrix(mess);
 REPEAT
 UNTIL local_reset;
 mess := '-->';
 write_dotmatrix(mess);
 drive_switcher;
END;


PROCEDURE switcher1_to_180;
  BEGIN
   WRITELN('SW 1 TO 180 deg.');
   green_led := 10;
   motor_power := $FE;
   switcher_pos_status := $68;
   switcher_destination := 4;
   drive_switcher;
  END;


PROCEDURE switcher2_to_180;
  BEGIN
   WRITELN('SW 2 TO 180 deg.');
   green_led := 20;
   motor_power := $FD;
   switcher_pos_status := $68;
   switcher_destination := $20;
   drive_switcher;
  END;


PROCEDURE switcher3_to_180;
  BEGIN
   WRITELN('SW 3 TO 180 deg.');
   green_led := 30;
   motor_power := $FB;
   switcher_pos_status := $69;
   switcher_destination := 1;
   drive_switcher;
  END;


PROCEDURE switcher4_to_180;
  BEGIN
   WRITELN('SW 4 TO 180 deg.');
   green_led := 40;
   motor_power := $F7;
   switcher_pos_status := $69;
```

```
   switcher_destination := 8;
   drive_switcher;
  END;


PROCEDURE switcher1_to_zero;
  BEGIN
   WRITELN('SW 1 TO Ø deg.');
   green_led := 11;
   motor_power := $FE;
   switcher_pos_status := $68;
   switcher_destination := 2;
   drive_switcher;
  END;


PROCEDURE switcher2_to_zero;
  BEGIN
   WRITELN('SW 2 TO Ø deg.');
   green_led := 21;
   motor_power := $FD;
   switcher_pos_status := $68;
   switcher_destination := $1Ø;
   drive_switcher;
  END;


PROCEDURE switcher3_to_zero;
  BEGIN
   WRITELN('SW 3 TO Ø deg.');
   green_led := 31;
   motor_power := $FB;
   switcher_pos_status := $68;
   switcher_destination := $8Ø;
   drive_switcher;
  END;


PROCEDURE switcher4_to_zero;
  BEGIN
   WRITELN('SW 4 TO Ø deg.');
   green_led := 41;
   motor_power := $F7;
   switcher_pos_status := $69;
   switcher_destination := 4;
   drive_switcher;
  END;

{PROGRAM Switcher status check & test          SW_CHK.PAS
 This program reads the switcher status port and updates the Mimic Panel
 if all the switcers are at zero degrees. If one or more are not at
 zero degrees it will call the routine do drive it to zero degrees.

 The second part drives all the Switchers to 18Ø deg. and back to Ø deg.

    sw no.       3   2    1
    sw_stat_1 : 1ØØ1 ØØ1Ø

    sw no.              4
    sw_stat_2 : ØØØØ Ø1ØØ
}
```

```
VAR
 sw_stat_1,sw_stat_2 : BYTE;


PROCEDURE switcher_status_check;
BEGIN
 mess := '  Switcher status check.';
 write_dotmatrix(mess);
 bleep;
 DELAY(1000);
 sw_stat_1 := PORT[$68] AND $92;
 sw_stat_2 := PORT[$69] AND 4;
 IF sw_stat_1 = 0 THEN
  BEGIN
   dollar33 := dollar33 AND $E3;
   PORT[$33] := dollar33;
  END
 ELSE
  BEGIN
   IF sw_stat_1 AND  2  =  2  THEN
   switcher1_to_zero;
   IF sw_stat_1 AND $10 =  $10 THEN
   switcher2_to_zero;
   IF sw_stat_1 AND $80 =  $80 THEN
   switcher3_to_zero;
   dollar33 := dollar33 AND $E3;  {Change these to become ones Green = 1,Red =
   PORT[$33] := dollar33;
  END;
 IF sw_stat_2 = 0 THEN
  BEGIN
   dollar33 := dollar33 AND $CF;     {Same as above}
   PORT[$33] := dollar33;
  END
 ELSE
  BEGIN
   If sw_stat_2 AND  4  =  4  THEN
   switcher4_to_zero;
  END;
END;


PROCEDURE switcher_check;

 BEGIN
  mess := 'Switchers will now be tested            Observe the Mimic LEDs';
  write_dotmatrix(mess);
  DELAY(4000);
  mess := 'Switcher no 1';
  write_dotmatrix(mess);
  switcher1_to_180;
  DELAY(500);
  switcher1_to_zero;
  DELAY(500);

  mess := 'Switcher no 2';
  write_dotmatrix(mess);
  switcher2_to_180;
  DELAY(500);
  switcher2_to_zero;
```

```
  DELAY(500);

  mess := 'Switcher no 3';
  write_dotmatrix(mess);
  switcher3_to_180;
  DELAY(500);
  switcher3_to_zero;
  DELAY(500);

  mess := 'Switcher no 4';
  write_dotmatrix(mess);
  switcher4_to_180;
  DELAY(500);
  switcher4_to_zero;
  DELAY(500);

  mess := 'All four switchers O K';
  write_dotmatrix(mess);
 END;

{PROGRAM Parking Loop                P_LOOP.PAS

a) Parking_loop_check will check the occupied positions in the parking loop
   and update the Mimic panel LED's accordingly.
b) Parking_loop_positioning will position the transporters in the parking
   loop so that the first one will be in position 10, the next one will be
   in position 9 etc.

}



CONST
 pl_position : ARRAY[1..10] OF
               BYTE = ($66,$66,$66,$66,$66,$66,$66,$66,$67,$67);

 test : ARRAY[1..10] OF
        BYTE = (1,2,4,8,$10,$20,$40,$80,1,2);

 pl_output : ARRAY[1..10] OF
             BYTE = ($7F,$BF,$DF,$EF,$F7,$FB,$FD,$FE,$7F,$BF);


VAR
 out_put,empty,empty_next : BYTE;
 I,J,pointer : INTEGER;



PROCEDURE parking_loop_check;

VAR
 pl_positions1,pl_positions2 : BYTE;

BEGIN

 pl_positions1 := PORT[$66];
 pl_positions2 := PORT[$67] AND 3;

 IF pl_positions1 AND 1 = 0 THEN
  dollar33 := dollar33 AND $FD
  ELSE dollar33 := dollar33 OR 2;
```

```
IF pl_positions1 AND 2 = Ø THEN
 dollar33 := dollar33 AND $FE
 ELSE dollar33 := dollar33 OR 1;

IF pl_positions1 AND 4 = Ø THEN
 dollar32 := dollar32 AND $7F
 ELSE dollar32 := dollar32 OR $8Ø;

IF pl_positions1 AND 8 = Ø THEN
 dollar32 := dollar32 AND $BF
 ELSE dollar32 := dollar32 OR $4Ø;

IF pl_positions1 AND $1Ø = Ø THEN
 dollar32 := dollar32 AND $DF
 ELSE dollar32:= dollar32 OR $2Ø;

IF pl_positions1 AND $2Ø = Ø THEN
 dollar32 := dollar32 AND $EF
 ELSE dollar32 := dollar32 OR $1Ø;

IF pl_positions1 AND $4Ø = Ø THEN
 dollar32 := dollar32 AND $F7
 ELSE dollar32 := dollar32 OR 8;

IF pl_positions1 AND $8Ø = Ø THEN
 dollar32 := dollar32 AND $FB
 ELSE dollar32 := dollar32 OR 4;


IF pl_positions2 AND 1 = Ø THEN
 dollar32 := dollar32 AND $FD
 ELSE dollar32 := dollar32 OR 2;

IF pl_positions2 AND 2 = Ø THEN
 dollar32 := dollar32 AND $FE
 ELSE dollar32 := dollar32 OR 1;

PORT[$33] := dollar33;
PORT[$32] := dollar32;
END;



PROCEDURE move_transporter_pl(pl_pos_destination,pl_pos_origan : INTEGER);

BEGIN
 FOR pointer := pl_pos_destination DOWNTO pl_pos_origan DO
 BEGIN
  IF pointer > 8 THEN
   BEGIN
   dollar3A := dollar3A AND pl_output[pointer];
   END
  ELSE dollar3B := dollar3B AND pl_output[pointer];
 END;

 IF pl_pos_destination > 8 THEN
  PORT[$3A] := dollar3A
 ELSE PORT[$3B] := dollar3B;

 IF pl_pos_origan > 8 THEN
  PORT[$3A] := dollar3A
```

```
 ELSE PORT[$3B] := dollar3B;

END;



PROCEDURE poll_pl_positions(pl_pos,tst,I,J : BYTE);

BEGIN
 WRITELN('DEST = ',I,'  ORG = ',J);
 REPEAT
 parking_loop_check;
 empty := PORT[pl_pos] AND tst;
 UNTIL empty = Ø;
 DELAY(25);
 dollar3A := dollar3A OR $CØ;
 dollar3B := $FF;
 PORT[$3A] := dollar3A;
 PORT[$3B] := dollar3B;
END;



PROCEDURE parking_loop_positioning;

BEGIN
 mess := 'Transporters will now be positioned in  the Parking Loop.';
 write_dotmatrix(mess);
 FOR I := 1Ø DOWNTO 1 DO
 BEGIN
  empty := PORT[pl_position[I]] AND test[I];
  IF empty <> Ø THEN
  BEGIN
   FOR J := (I - 1) DOWNTO 1 DO
   BEGIN
    empty_next := PORT[pl_position[J]] AND test[J];
    IF empty_next = Ø THEN
    BEGIN
     move_transporter_pl(I,J);
     poll_pl_positions(pl_position[I],test[I],I,J);
     parking_loop_check;                          {To update the Mimic panel
     J := Ø;                                      {To end the loop}
    END;
   END;
  END;
 END;
END;

{PROGRAM  STATIONS                    STATIONS.PAS

This program does the following:

a) It updates the memory when a transporter reaches it's destination.
   The data is stored in a file on A:STATION.DAT.

b) When the system is first powered up it checks which transporters is
   parked where by reading the file. It then checks if a transporter is
   actually in that station. If no transporter was found in the station
   or the file shows the station is empty, it writes a Ø to the Mimic
   panel wich blanks the display. If a transporter is parked in a station
   the ID number is written to the Mimic panel display. This is done for
   every station in sequence.
```

c) The following codes represent the actual storage positions in memory.
   The ID number of the transporter is stored in the memory position.

Ø	1	2	3	4	5	6
HotCellsA	HotCellsB	Storage	Vault1_1	Vault1_2	Vault1_3	Vault2

}

```
TYPE
 xp_id_type = Ø..7;      {Ø = station empty , 1 - 5 = transporter ID in station
                         {7 = unknown transporter in station

data_type = RECORD
             xp_id : xp_id_type;
            END;
VAR
 data : data_type;
 data_file : FILE OF data_type;
 mem_pos : INTEGER;
 stations : BYTE;


PROCEDURE save_station;

BEGIN
 ASSIGN(data_file,'A:STATION.DAT');
 RESET(data_file);
 SEEK(data_file,mem_pos);
 {$I-} WRITE(data_file,data) {$I+};
 CLOSE(data_file);
END;


PROCEDURE read_station;

BEGIN
 ASSIGN(data_file,'A:STATION.DAT');
 RESET(data_file);
 SEEK(data_file,mem_pos);
 READ(data_file,data);
 CLOSE(data_file);
END;


PROCEDURE station_strobe;

BEGIN
 DELAY(1);                              {Data set up-time for 74LS374}
 dollar31 := dollar31 OR $8Ø;           {strobe high}
 PORT[$31] := dollar31;
 DELAY(1);                              {clock pulse width for 74LS374}
 dollar31 := dollar31 AND $7F;          {strobe low}
 PORT[$31] := dollar31;
 dollar31 := dollar31 OR $78;           {station code to = 15  X111 1XXX}
```

```
 PORT[$31] := dollar31;
 DELAY(1);                              {Data set-up time}
 dollar31 := dollar31 OR $8Ø;           {strobe high}
 PORT[$31] := dollar31;
 DELAY(1);                              {clock pulse witdh}
 dollar31 := dollar31 AND $7F;          {strobe low}
 PORT[$31] := dollar31;
END;



PROCEDURE HCA_display;
VAR
 I: INTEGER;

BEGIN
FOR I := 1 TO 2 DO
 BEGIN
  stations := PORT[$63] AND $2Ø;
  IF stations <> Ø THEN
   BEGIN
    dollar31 := dollar31 AND $F8;          {ID code = Ø : BLANK}
    WRITELN('Hot cells A is empty');
    IF data.xp_id <> Ø THEN
     BEGIN
      data.xp_id := Ø;
      save_station;
     END;
   END
   ELSE
   BEGIN
    WRITELN('Transporter ',data.xp_id,'  parked in Hot cells A');
    IF data.xp_id = Ø THEN
     BEGIN
      data.xp_id := 7;
      save_station;
      dollar31 := dollar31 OR 7;
     END;
   END;
   PORT[$31] := dollar31;
   dollar31 := (dollar31 OR $28) AND $AF;      {station code = 5  XØ1Ø 1XXX}
   PORT[$31] := dollar31;
   station_strobe;
 END;
END;


PROCEDURE HCB_display;

BEGIN
 stations := PORT[$63] AND $4Ø;
 IF stations <> Ø THEN
  BEGIN
   dollar31 := dollar31 AND $F8;          {ID code = Ø : BLANK}
   WRITELN('Hot cells B is empty');
   IF data.xp_id <> Ø THEN
    BEGIN
     data.xp_id := Ø;
     save_station;
    END;
  END
```

```
 ELSE
  BEGIN
   WRITELN('Transporter ',data.xp_id,'  parked in Hot cells B');
   IF data.xp_id = Ø THEN
     BEGIN
      data.xp_id := 7;
      save_station;
      dollar31 := dollar31 OR 7;
     END;
   END;
 PORT[$31] := dollar31;
 dollar31 := (dollar31 OR $3Ø) AND $B7;      {station code = 6  XØ11 ØXXX}
 PORT[$31] := dollar31;
 station_strobe;
END;



PROCEDURE S_display;

BEGIN
 stations := PORT[$63] AND $8Ø;
 IF stations <> Ø THEN
  BEGIN
   dollar31 := dollar31 AND $F8;          {ID code = Ø : BLANK}
   WRITELN('Storage is empty');
   IF data.xp_id <> Ø THEN
    BEGIN
     data.xp_id := Ø;
     save_station;
    END;
  END
 ELSE
  BEGIN
   WRITELN('Transporter ',data.xp_id,'  parked in Storage');
   IF data.xp_id = Ø THEN
     BEGIN
      data.xp_id := 7;
      save_station;
      dollar31 := dollar31 OR 7;
     END;
   END;
 PORT[$31] := dollar31;
 dollar31 :=  (dollar31 OR $2Ø) AND $A7;     {station code = 4  XØ1Ø ØXXX}
 PORT[$31] := dollar31;
 station_strobe;
END;



PROCEDURE V1_1_display;

BEGIN
 DELAY(3ØØ);
 stations := PORT[$64] AND 2;
 IF stations <> Ø THEN
  BEGIN
   dollar31 := dollar31 AND $F8;          {ID code = Ø : BLANK}
   WRITELN('Vault1 1 is empty');
   IF data.xp_id <> Ø THEN
    BEGIN
     data.xp_id := Ø;
```

```
    save_station;
   END;
  END
 ELSE
  BEGIN
   WRITELN('Transporter ',data.xp_id,'  parked in Vault1 1');
   IF data.xp_id = 0 THEN
     BEGIN
      data.xp_id := 7;
      save_station;
      dollar31 := dollar31 OR 7;
     END;
   END;
 PORT[$31] := dollar31;
 dollar31 :=  dollar31 AND $87;              {station code = 0  X000 0XXX}
 PORT[$31] := dollar31;
 station_strobe;
END;


PROCEDURE V1_2_display;

BEGIN
 DELAY(300);
 stations := PORT[$64] AND 4;
 IF stations <> 0 THEN
  BEGIN
   dollar31 := dollar31 AND $F8;          {ID code = 0 : BLANK}
   WRITELN('Vault1 2 is empty');
   IF data.xp_id <> 0 THEN
    BEGIN
     data.xp_id := 0;
     save_station;
    END;
  END
 ELSE
  BEGIN
   WRITELN('Transporter ',data.xp_id,'  parked in Vault1 2');
   IF data.xp_id = 0 THEN
     BEGIN
      data.xp_id := 7;
      save_station;
      dollar31 := dollar31 OR 7;
     END;
   END;
 PORT[$31] := dollar31;
 dollar31 := (dollar31 OR  8 ) AND $8F;      {station code = 1  X000 1XXX}
 PORT[$31] := dollar31;
 station_strobe;
END;


PROCEDURE V1_3_display;

BEGIN
 DELAY(300);
 stations := PORT[$64] AND 8;
 IF stations <> 0 THEN
  BEGIN
   dollar31 := dollar31 AND $F8;          {ID code = 0 : BLANK}
```

```
   WRITELN('Vault1 3 is empty');
   IF data.xp_id <> Ø THEN
    BEGIN
     data.xp_id := Ø;
     save_station;
    END;
  END
 ELSE
  BEGIN
   WRITELN('Transporter ',data.xp_id,'  parked in Vault1 3');
   IF data.xp_id = Ø THEN
     BEGIN
      data.xp_id := 7;
      save_station;
      dollar31 := dollar31 OR 7;
     END;
   END;
 PORT[$31] := dollar31;
 dollar31 := (dollar31 OR $1Ø) AND $97;      {station code = 2  XØØ1 ØXXX}
 PORT[$31] := dollar31;
 station_strobe;
END;


PROCEDURE V2_display;

BEGIN
 stations := PORT[$64] AND 1;
 IF stations <> Ø THEN
  BEGIN
   dollar31 := dollar31 AND $F8;          {ID code = Ø : BLANK}
   WRITELN('Vault2 is empty');
   IF data.xp_id <> Ø THEN
    BEGIN
     data.xp_id := Ø;
     save_station;
    END;
  END
 ELSE
  BEGIN
   WRITELN('Transporter ',data.xp_id,'  parked in Vault2');
   IF data.xp_id = Ø THEN
     BEGIN
      data.xp_id := 7;
      save_station;
      dollar31 := dollar31 OR 7;
     END;
   END;
 PORT[$31] := dollar31;
 dollar31 :=  (dollar31 OR $18) AND $9F;      {station code = 3  XØØ1 1XXX}
 PORT[$31] := dollar31;
 station_strobe;
END;


PROCEDURE mimic_update;

BEGIN
 CASE data.xp_id OF
    Ø : dollar31 :=  dollar31 AND $F8;      {transporter ID code = Ø : BLANK}
```

```
    1 : dollar31 := (dollar31 OR 1) AND $F9;    { transporter }
    2 : dollar31 := (dollar31 OR 2) AND $FA;    {     ID      }
    3 : dollar31 := (dollar31 OR 3) AND $FB;    {    code     }
    4 : dollar31 := (dollar31 OR 4) AND $FC;    {    from     }
    5 : dollar31 := (dollar31 OR 5) AND $FD;    {   1 to 5    }
    7 : dollar31 :=  dollar31 OR 7;             {Unknown transporter}
   END;
   CASE mem_pos OF
    Ø : HCA_display;
    1 : HCB_display;
    2 : S_display;
    3 : V1_1_display;
    4 : V1_2_display;
    5 : V1_3_display;
    6 : V2_display;
   END;
END;



PROCEDURE stations_update;

BEGIN
 mess := 'Stations will now be updated          Obseve the Mimic Panel';
 write_dotmatrix(mess);
 FOR mem_pos := Ø TO 6 DO
  BEGIN
   read_station;
   mimic_update;
  END;
 bleep;
END;

{PROGRAM STRAY CHECK       STRAYCHK.PAS

 This program checks for stray transporters on the tracks when the system
 is switched on. When it finds one it sets up the codes to drive it to the
 Parking loop.
}

PROCEDURE local_commands_poll;FORWARD;

VAR
 test1 : BYTE;
 stray_routine_serviced,stray_to_station,
 V1_off,HCA_off : BOOLEAN;



PROCEDURE strobe_TP;

BEGIN
 DELAY(1);                                  {set-up time}
 dollar3Ø := dollar3Ø AND $DF;              {enable low}
 PORT[$3Ø] := dollar3Ø;
 DELAY(1);
 dollar3Ø := dollar3Ø OR $1Ø;               {strobe high}
 PORT[$3Ø] := dollar3Ø;
 DELAY(1);                                  {pulse width}
 dollar3Ø := dollar3Ø AND $EF;              {strobe low}
 PORT[$3Ø] := dollar3Ø;
END;
```

```
PROCEDURE transporter_on_track_display;

VAR
 I : INTEGER;

BEGIN
IF (test62 AND test64 AND test65) = $FF THEN
 BEGIN
  dollar30 := dollar30 OR $20;                    {Display enable high}
  PORT[$30] := dollar30;                          { XX1X XXXX }
 END
ELSE
 BEGIN
  {PORT 62 looks at HS,Va,Vb,Vc}
  test1 := $10;                                    {0001 0000}
  FOR I := 1 TO 4 DO
   BEGIN
    IF test62 AND test1 = 0 THEN
    BEGIN
     CASE I OF
      1 : dollar30 := (dollar30 OR 3) AND $F3;    {HS = 3     XXXX 0011}
      2 : dollar30 := (dollar30 OR 2) AND $F2;    {Va = 2     XXXX 0010}
      3 : dollar30 := (dollar30 OR 1) AND $F1;    {Vb = 1     XXXX 0001}
      4 : dollar30 := (dollar30 OR 0) AND $F0;    {Vc = 0     XXXX 0000}
     END;
     PORT[$30] := dollar30;
     strobe_TP;
    END;
    test1 := test1 SHL 1;
   END;

  {PORT 64 looks at HA,HB,H,S}
  test1 := $10;                                    {0001 0000}
  FOR I := 1 TO 4 DO
   BEGIN
    IF test64 AND test1 = 0 THEN
    BEGIN
     CASE I OF
      1 : dollar30 := (dollar30 OR 6) AND $F6;    {HB = 6     XXXX 0110}
      2 : dollar30 := (dollar30 OR 7) AND $F7;    {HA = 7     XXXX 0111}
      3 : dollar30 := (dollar30 OR 5) AND $F5;    {H  = 5     XXXX 0101}
      4 : dollar30 := (dollar30 OR 4) AND $F4;    {S  = 4     XXXX 0100}
     END;
     PORT[$30] := dollar30;
     strobe_TP;
    END;
    test1 := test1 SHL 1;
   END;

  {PORT 65 looks at Vd,V2a,,V2b,V2c,V1a,V1b,V1c,V1d}
  test1 := $01;                                    {0000 0001}
  FOR I := 1 TO 8 DO
   BEGIN
    IF test65 AND test1 = 0 THEN
    BEGIN
     CASE I OF                                     {Bit $20 is display enable}
      1 : dollar30 := (dollar30 OR 08) AND $F8;    {Vd  = 8    XXXX 1000}
      2 : dollar30 := (dollar30 OR $D) AND $FD;    {V2a = 13   XXXX 1101}
      3 : dollar30 := (dollar30 OR $E) AND $FE;    {V2b = 14   XXXX 1110}
```

```
      4 : dollar3Ø := (dollar3Ø OR $F) AND $FF;    {V2c = 15    XXXX 1111}
      5 : dollar3Ø := (dollar3Ø OR Ø9) AND $F9;    {V1a = 9     XXXX 1ØØ1}
      6 : dollar3Ø := (dollar3Ø OR $A) AND $FA;    {V1b = 1Ø    XXXX 1Ø1Ø}
      7 : dollar3Ø := (dollar3Ø OR $B) AND $FB;    {V1c = 11    XXXX 1Ø11}
      8 : dollar3Ø := (dollar3Ø OR $C) AND $FC;    {V1d = 12    XXXX 11ØØ}
     END;
     PORT[$3Ø] := dollar3Ø;
     strobe_TP;
    END;
    test1 := test1 SHL 1;
   END;
 END;
END;


PROCEDURE stray_check;

BEGIN
 mess := 'Stray transporter check';
 write_dotmatrix(mess);
 DELAY(2ØØØ);
 stray_to_station := FALSE;
 test62 := (PORT[$62] AND $FØ) OR $ØF;
 test64 := (PORT[$64] AND $FØ) OR $ØF;
 test65 :=  PORT[$65];
 IF (test62 AND test64 AND test65) <> $FF THEN
  BEGIN
    stray_to_station := TRUE;
    WRITELN('STRAY TRANSPORTER ON TRACKS');
    transporter_on_track_display;
    mess := 'A stray transporter was found           on the tracks. RESET to
    write_dotmatrix(mess);
    bleep_until_reset;
    mess := 'Transporter will now be driven to the   Parking Loop.  Press EXE
    write_dotmatrix(mess);
    stray_routine_serviced := FALSE;
    to_code := 6;
    CASE test62 OF
     $EF : from_code := 1;
     $DF : from_code := 3;
     $BF : from_code := 3;
     $7F : from_code := 3;
    END;
    CASE test64 OF
     $EF : from_code := 2;
     $DF : from_code := 1;
     $BF : from_code := 1;
     $7F : from_code := 5;
    END;
    CASE test65 OF
     $FE : from_code := 3;
     $FD : from_code := 4;
     $FB : from_code := 4;
     $F7 : from_code := 4;
     $EF : from_code := 3;
     $DF : from_code := 3;
     $BF : from_code := 3;
     $7F : from_code := 3;
    END;
    WRITELN('from_code = ',from_code);
    test63 := PORT[$63] AND $2Ø;
```

```
    test64 := (PORT[$64] AND $E) OR $F1;
    IF (from_code = 1) AND (test63 = Ø) THEN HCA_off := TRUE
     ELSE HCA_off := FALSE;
    IF (from_code = 3) AND (test64 <> $FF) THEN V1_off := TRUE
     ELSE V1_off := FALSE;
    REPEAT
    local_commands_poll;
    UNTIL stray_routine_serviced;
   END;
 V1_off := FALSE;
 HCA_off := FALSE;
 mess := 'Check the number of transporters        in the system = 5.  (RESET)'
 write_dotmatrix(mess);
 REPEAT
 UNTIL local_reset;
 DELAY(5ØØ);    {Operator time on switch}
 mess := 'Make sure it is not jammed on           the tracks.         (RESET)'
 write_dotmatrix(mess);
 REPEAT
 UNTIL local_reset;
END;

{MOVE TRANSPORTER                    TR_MOVE.PAS

a) The routine move_transporter_station_to_station sets up the appropriate
   path for power and direction and polls the stopping plates. It also
   calls the routines to update the bubble memory and the mimic panel.
b) The routine parking_loop_ID_search reads the sesnsing plate as a
   transporter passes it. If it is the correct one, it calls the routine
   to park it on the switcher. If it is not the correct one it calls the
   routine to rearange the Parking loop and and sends the next one to the
   sensing plate. If the first one passes the sesnsing plate again it rearang
   the Parking loop and then displays a message "The ID entered was not found
   in the Parking loop." before it exits to the local polling routine.
c) The track_display routine updates the Mimic panel LED's as the transporter
   progresses on the tracks.
}


VAR
 data_new : data_type;

 first_transporter,destination_arrival,
 ID_test                                    : INTEGER;

 ID_not_found                               : BOOLEAN;

 switcher1_track_power,switcher2_track_power,
 switcher3_track_power,switcher4_track_power,
 store62,store64,store65                     : BYTE;

 PROCEDURE display_ready;FORWARD;



PROCEDURE next_transporter_in_parking_loop;

BEGIN
 dollar3A := dollar3A AND $BF;
 PORT[$3A] := dollar3A;                        {Power to PL 1Ø}

 REPEAT
```

```
 test63 := (PORT[$63] AND $1F) OR $E0;        {Read the sensing plate}
UNTIL test63 <> $FF;
DELAY(15);                    {Contact bounce for mercury wetted reed switch}
test63 := (PORT[$63] AND $1F) OR $E0;        {Read the sensing plate again}
parking_loop_check;
CASE test63 OF            {Check which transporter passed the sensing plate}
 $FE : ID_test := 1;
 $FD : ID_test := 2;
 $FB : ID_test := 3;
 $F7 : ID_test := 4;
 $EF : ID_test := 5
 ELSE ID_test := 10;
END;
END;



PROCEDURE rearange_transporters_parking_loop;

 BEGIN
   dollar3B := dollar3B AND $7F;
   PORT[$3B] := dollar3B;                          {Power to PL 1}
   REPEAT
    test66 := PORT[$66] AND 1;          {Test for arrival of transporter}
   UNTIL test66 = 0;
   dollar3A := dollar3A OR $C0;
   dollar3B := $FF;
   PORT[$3A] := dollar3A;          {  Power off for  }
   PORT[$3B] := dollar3B;          {   PL1 to PL10   }
   parking_loop_check;             {To update the Mimic panel}
   parking_loop_positioning;       {Rearange the transporters}
  END;



PROCEDURE transporter_on_switcher_parking_loop;

BEGIN
 dollar36 := dollar36 AND $FE;
 PORT[$36] := dollar36;             { SW 1 track power <-  }
 switcher1_to_180;
 REPEAT
  test68 := PORT[$68] AND 1;        {Read the stopping plate}
 UNTIL test68 = 0;
 dollar36 := dollar36 OR 1;
 PORT[$36] := dollar36;             { SW 1 track power off}
 dollar3A := dollar3A OR $40;
 PORT[$3A] := dollar3A;             { PL 10 power off }
 switcher1_to_zero;
END;



PROCEDURE parking_loop_ID_search;

BEGIN
 ID_not_found := FALSE;
 next_transporter_in_parking_loop;
 first_transporter := ID_test;
 IF ID_test = data_new.xp_id THEN
  BEGIN
   transporter_on_switcher_parking_loop;
```

```
  EXIT;                {to 'FROM parking loop' in station_to_station proc.}
 END
ELSE rearange_transporters_parking_loop;

REPEAT
 next_transporter_in_parking_loop;
 IF ID_test = data_new.xp_id THEN
  BEGIN
   transporter_on_switcher_parking_loop;
   EXIT;               {to 'FROM parking loop' in station_to_station proc.}
  END
 ELSE rearange_transporters_parking_loop;
UNTIL ID_test = first_transporter;
ID_not_found := TRUE;
parking_loop_check;
parking_loop_positioning;
mess := 'The ID entered was not found in the     Parking Loop.    RESET to cor
bleep;
DELAY(300);
bleep;
write_dotmatrix(mess);
REPEAT
UNTIL local_reset;
END;


PROCEDURE track_display;

BEGIN
 test62 := (PORT[$62] AND $F0) OR $0F;
 IF test62 <> store62 THEN
  BEGIN
   transporter_on_track_display;
   store62 := test62;
  END;
 test64 := (PORT[$64] AND $F0) OR $0F;
 IF test64 <> store64 THEN
  BEGIN
   transporter_on_track_display;
   store64 := test64;
  END;
 test65 := PORT[$65];
 IF test65 <> store65 THEN
  BEGIN
   transporter_on_track_display;
   store65 := test65;
  END;
END;


PROCEDURE all_power_off;

BEGIN
 dollar38 := $FF;
 PORT[$38] := dollar38;
 dollar39 := $FF;                    {    Track     }
 PORT[$39] := dollar39;              {    Power     }
 dollar36 := $FF;                    {    off to    }
 PORT[$36] := dollar36;              { all sections }
 track_display;                      {Mimic LED's off}
```

```
END;



PROCEDURE bubble_update;

BEGIN
 read_station;                          {Read  ID of transporter in bubble mem.}
 data_new.xp_id := data.xp_id;          {Save this in RAM}
 data.xp_id := Ø;                       {Save this on bubble}
 save_station;
END;


PROCEDURE Vault1_update(pointer:INTEGER);

BEGIN
 CASE pointer OF
  1 : BEGIN  {Vault1 1}
       mem_pos := 3;
       bubble_update;
       V1_1_display;
      END;
  2 : BEGIN  {Vault1 2}
       mem_pos := 4;
       bubble_update;
       V1_2_display;
      END;
  3 : BEGIN  {Vault1 3}
       mem_pos := 5;
       bubble_update;
       V1_3_display;
      END;
 END;
END;



PROCEDURE move_transporter_station_to_station;

BEGIN
 CASE from_code OF              {The arrows indicate the direction of movement

  1 : BEGIN  {FROM Hot Cells A}
       IF HCA_off THEN
        dollar38 := dollar38 AND $FC        { H->         }
       ELSE
        dollar38 := dollar38 AND $CC;       { HA->   H-> }
       PORT[$38] := dollar38;
       dollar39 := dollar39 AND $3C;
       PORT[$39] := dollar39;               {  V->  HS-> }
       dollar36 := dollar36 AND Ø;
       PORT[$36] := dollar36;               { SW1-> SW2-> SW3-> SW4-> }
       DELAY(5ØØ);
       mem_pos := Ø;
       bubble_update;
       HCA_display;                         {Update the Mimic panel}
      END;

  2 : BEGIN  {FROM Hot Cells B}
       dollar38 := dollar38 AND $FØ;
```

```
    PORT[$38] := dollar38;                  { HB->   H-> }
    dollar39 := dollar39 AND $3C;
    PORT[$39] := $3C;                       {  V->  HS-> }
    dollar36 := dollar36 AND Ø;
    PORT[$36] := dollar36;                  { SW1-> SW2-> SW3-> SW4-> }
    mem_pos := 1;
    bubble_update;
    DELAY(5ØØ);
    HCB_display;                            {Update the Mimic panel}
    switcher4_to_18Ø;
    IF to_code = 1 THEN
     switcher4_track_power := $BF           { <- }
     ELSE switcher4_track_power := $3F;     { -> }
    REPEAT
     track_display;                         {Indicate on Mimic panel}
     test69 := PORT[$69] AND 2;             {Poll SW4 stopping plate}
    UNTIL test69 = Ø;
    dollar36 := dollar36 OR $CØ;            {Track power SW4 off}
    PORT[$36] := dollar36;
    switcher4_to_zero;
    dollar36 := dollar36 AND switcher4_track_power;
    PORT[$36] := dollar36;                  {Track power SW4 on}
   END;

3 : BEGIN  {FROM Vault 1}
     CASE to_code OF
      31 : BEGIN
            dollar39 := dollar39 AND $EF;
            PORT[$39] := dollar39;          { V1<-       }
           END;
      33 : BEGIN
            dollar39 := dollar39 AND $CF;
            PORT[$39] := dollar39;          { V1->       }
           END;
      32 : BEGIN
            test64 := PORT[$64] AND $ØA;
            CASE test64 OF
             8 : BEGIN
                  dollar39 := dollar39 AND $CF;
                  PORT[$39] := dollar39;    { V1->       }
                 END;
             2 : BEGIN
                  dollar39 := dollar39 AND $EF;
                  PORT[$39] := dollar39;    { V1<-       }
                 END;
            END;
           END;
     ELSE
     BEGIN
      test64 := PORT[$64] AND $ØE;
      IF V1_off THEN
       dollar39 := dollar39 AND $BE         { V1<- HS<-     }
      ELSE
       dollar39 := dollar39 AND $AE;        { V<- V1<- HS<-}
      PORT[$39] := dollar39;
      dollar38 := dollar38 AND $FE;
      PORT[$38] := dollar38;                { H<-           }
      DELAY(5ØØ);
      CASE test64 OF
       $C : Vault1_update(1);
       $A : Vault1_update(2);
        6 : Vault1_update(3);
```

```
      END;
      dollar36 := dollar36 AND $AA;
      PORT[$36] := dollar36;                { SW1<- SW2<- SW3<- SW4<- }
     END;
    END;
   END;

4 : BEGIN  {FROM Vault 2}
     dollar39 := dollar39 AND $FB;
     PORT[$39] := dollar39;                {V2<- }
     dollar36 := dollar36 AND $A2;
     PORT[$36] := dollar36;                { SW1<- SW2-> SW3<- SW4<- }
     DELAY(500);
     mem_pos := 6;
     bubble_update;
     V2_display;                           {Update the Mimic panel}
     switcher2_to_180;
     IF (to_code DIV 10 = 3) THEN
      BEGIN
       switcher2_track_power := $F3;       { -> }
       dollar39 := dollar39 AND $CB;
       PORT[$39] := dollar39;              { V1-> V2<- }
      END
     ELSE
      BEGIN
       switcher2_track_power := $FB;       { <- }
       dollar38 := dollar38 AND $FE;
       PORT[$38] := dollar38;              { H<- }
       dollar39 := dollar39 AND $BA;
       PORT[$39] := dollar39;              { V<- V2<- HS<- }
      END;
     REPEAT
      track_display;                       {Indicate on Mimic panel}
      test68 := PORT[$68] AND 8;           {Poll SW2 stopping plate}
     UNTIL test68 = 0;
     dollar36 := dollar36 OR $C;           {Track power SW2 off}
     PORT[$36] := dollar36;
     switcher2_to_zero;
     dollar36 := dollar36 AND switcher2_track_power;
     PORT[$36] := dollar36;                {Track power SW2 on}
    END;

5 : BEGIN  {FROM Storage}
     dollar38 := dollar38 AND $3E;
     PORT[$38] := dollar38;                {  S->   H<- }
     dollar39 := dollar39 AND $3C;
     PORT[$39] := dollar39;                {  V->  HS-> }
     dollar36 := dollar36 AND $80;
     PORT[$36] := dollar36;                { SW1-> SW2-> SW3-> SW4<- }
     DELAY(100);
     mem_pos := 2;
     bubble_update;
     S_display;                            {Update the Mimic panel}
     switcher3_to_180;
     IF (to_code = 1) OR (to_code = 2) THEN
      switcher3_track_power := $EF         { <- }
      ELSE switcher3_track_power := $CF;   { -> }
      REPEAT
      track_display;                       {Indicate on Mimic panel}
      test68 := PORT[$68] AND $40;         {Poll SW3 stopping plate}
     UNTIL test68 = 0;
     dollar36 := dollar36 OR $30;          {Track power SW3 off}
```

```
      PORT[$36] := dollar36;
      switcher3_to_zero;
      dollar36 := dollar36 AND switcher3_track_power;
      PORT[$36] := dollar36;                    {Track power SW3 on}
     END;
END;

IF (from_code DIV 1Ø) = 6 THEN  {FROM Parking Loop}
 BEGIN
  parking_loop_ID_search;
  IF ID_not_found = TRUE THEN
   BEGIN
    display_ready;
    EXIT;                                     {To main polling routine}
   END;
  dollar38 := dollar38 AND $FE;
  PORT[$38] := dollar38;                      { H<-          }
  dollar39 := dollar39 AND $3E;
  PORT[$39] := dollar39;                      { V->   HS<-   }
  dollar36 := dollar36 AND $A3;               { SW2-> SW3<- SW4<- }
  PORT[$36] := dollar36;
  IF (to_code DIV 1Ø = 3) OR (to_code = 4) THEN
   switcher1_track_power := $FC               { SW1-> }
   ELSE switcher1_track_power := $FE;         { SW1<- }
  IF to_code <> 6 THEN
  BEGIN
   dollar36 := dollar36 AND switcher1_track_power;
   PORT[$36] := dollar36;                     {Transporter enroute}
  END;
 END;

CASE to_code OF

 1 : BEGIN   {TO Hot Cells A}
      dollar38 := dollar38 AND $EE;
      PORT[$38] := dollar38;                  { HA<-  H<- }
      dollar38 := $FF;
      REPEAT
       track_display;
       test63 := PORT[$63] AND $2Ø;           {Poll Hot cells A stopping plate}
      UNTIL test63 = Ø;
      PORT[$38] := dollar38;
      all_power_off;
      track_display;
      data.xp_id := data_new.xp_id;
      WRITELN('ID = ',data.xp_id);
      mem_pos := Ø;
      mimic_update;                           {Display on Mimic panel}
      save_station;                           {Save on Bubble memory}
     END;

 2 : BEGIN   {TO Hot Cells B}
      dollar38 := dollar38 AND $FB;
      PORT[$38] := dollar38;                  { HB<-  }
      REPEAT
       track_display;
      test69 := PORT[$69] AND 2;              {Poll SW4 stopping plate}
      UNTIL test69 = Ø;
      dollar36 := dollar36 OR $CØ;            {SW4 track power off}
      PORT[$36] := dollar36;
      switcher4_to_18Ø;
      dollar36 := dollar36 AND $BF;           {SW4 <- }
```

```
     PORT[$36] := dollar36;
     track_display;
     dollar38 := $FF;
     REPEAT
      track_display;
      test63 := PORT[$63] AND $40;          {Poll Hot Cells B stopping plate}
     UNTIL test63 = 0;
     PORT[$38] := dollar38;
     all_power_off;
     track_display;
     switcher4_to_zero;
     data.xp_id := data_new.xp_id;
     mem_pos := 1;
     mimic_update;
     save_station;
    END;

31: BEGIN   {TO Vault1 1}
     IF from_code <> 3 THEN
     BEGIN
      dollar39 := dollar39 AND $CF;         { V1-> }
      PORT[$39] := dollar39;
      REPEAT
       track_display;
       test68 := PORT[$68] AND 8;           {Poll SW2 stopping plate}
      UNTIL test68 = 0;
     END;

     REPEAT
      track_display;
      test64 := PORT[$64] AND 2;           {Poll Vault1 1 stopping plate}
     UNTIL test64 = 0;
     all_power_off;
     track_display;
     data.xp_id := data_new.xp_id;
     mem_pos := 3;
     mimic_update;                         {Display on Mimic panel}
     save_station;                         {Save on Bubble memory}
     vault1_update(2);
     vault1_update(3);
    END;

32: BEGIN   {TO Vault1 2}
     IF from_code <> 3 THEN
     BEGIN
      dollar39 := dollar39 AND $CF;
      PORT[$39] := dollar39;                { V1-> }
      REPEAT
       track_display;
       test68 := PORT[$68] AND 8;           {Poll SW2 stopping plate}
      UNTIL test68 = 0;
     END;
     dollar39 := $FF;

     REPEAT
      track_display;
      test64 := PORT[$64] AND 4;           {Poll Vault1 2 stopping plate}
     UNTIL test64 = 0;
     PORT[$39] := dollar39;
     all_power_off;
     track_display;
     data.xp_id := data_new.xp_id;
```

```
     mem_pos := 4;
     mimic_update;                          {Display on Mimic panel}
     save_station;                          {Save on Bubble memory}
     vault1_update(1);
     vault1_update(3);
    END;

33: BEGIN   {TO Vault1 3}
     IF from_code <> 3 THEN
     BEGIN
      dollar39 := dollar39 AND $CF;
      PORT[$39] := dollar39;                { V1-> }
      REPEAT
       track_display;
       test68 := PORT[$68] AND 8;           {Poll SW2 stopping plate}
      UNTIL test68 = 0;
     END;

     REPEAT
      track_display;
      test64 := PORT[$64] AND 8;            {Poll Vault1 3 stopping plate}
     UNTIL test64 = 0;
     all_power_off;
     track_display;
     data.xp_id := data_new.xp_id;
     mem_pos := 5;
     mimic_update;                          {Display on Mimic panel}
     save_station;                          {Save on Bubble memory}
     vault1_update(1);
     vault1_update(2);
    END;

 4 : BEGIN   {TO Vault 2}
     dollar39 := dollar39 AND $33;
     PORT[$39] := dollar39;                 { V2-> V-> }
     REPEAT
      track_display;
      test68 := PORT[$68] AND 8;            {Poll SW2 stopping plate}
     UNTIL test68 = 0;
     dollar36 := dollar36 OR $C;            {SW2 track power off}
     PORT[$36] := dollar36;
     track_display;
     switcher2_to_180;
     dollar36 := dollar36 AND $FB;          {SW2 <- }
     PORT[$36] := dollar36;
     REPEAT
      track_display;
      test64 := PORT[$64] AND 1;            {Poll Vault 2 stopping plate}
     UNTIL test64 = 0;
     all_power_off;
     track_display;
     switcher2_to_zero;
     data.xp_id := data_new.xp_id;
     mem_pos := 6;
     mimic_update;
     save_station;
    END;

 5 : BEGIN   {TO Storage}
     dollar38 := dollar38 AND $BC;
     PORT[$38] := dollar38;                 {  H->  S<-   }
     dollar39 := dollar39 AND $BE;
```

```
      PORT[$39] := dollar39;                { HS<-  V<-   }
      REPEAT
       track_display;
       test68 := PORT[$68] AND $40;         {Poll SW3 stopping plate}
      UNTIL test68 = 0;
      dollar36 := dollar36 OR $30;          {SW3 track power off}
      PORT[$36] := dollar36;
      switcher3_to_180;
      dollar36 := dollar36 AND $EF;         {SW3 <- }
      PORT[$36] := dollar36;
      track_display;
      dollar38 := $FF;
      REPEAT
       test63 := PORT[$63] AND $80;         {Poll Storage stopping plate}
      UNTIL test63 = 0;
      PORT[$38] := dollar38;
      all_power_off;
      track_display;
      switcher3_to_zero;
      data.xp_id := data_new.xp_id;
      mem_pos := 2;
      mimic_update;
      save_station;
     END;

 6 : BEGIN   {TO Parking Loop}
      dollar38 := dollar38 AND $FC;
      PORT[$38] := dollar38;                { H->   }
      dollar39 := dollar39 AND $BC;
      PORT[$39] := dollar39;                { HS->  V<-   }
      REPEAT
       track_display;
       test68 := PORT[$68] AND 1;           {Poll SW1 stopping plate}
      UNTIL test68 = 0;
      dollar36 := dollar36 OR $3;           {SW1 track power off}
      PORT[$36] := dollar36;
      switcher1_to_180;
      dollar36 := dollar36 AND $FE;         {SW1 -> }
      PORT[$36] := dollar36;
      dollar3B := dollar3B AND $7F;         {PL1 power on}
      PORT[$3B] := dollar3B;
      DELAY(1500);                          {Time for transporter to leave SW1
      track_display;
      switcher1_to_zero;
      REPEAT
       test66 := PORT[$66] AND 1;           {Poll PL1 position sensing}
      UNTIL test66 = 0;
      dollar3B := dollar3B OR $80;          {PL1 power off}
      PORT[$3B] := dollar3B;
      all_power_off;
      track_display;
      parking_loop_positioning;
      parking_loop_check;
     END;

 END;

 IF (from_code DIV 10) = 6 THEN
  parking_loop_positioning;
  parking_loop_check;

END;
```

```
{LOCAL POLLING ROUTINE                LOCPOLL.PAS

 This is the local polling routine which looks at commands entered from
 the Local controller.

 a) From_parking_loop reads PORT[$6Ø] and checks XXXX cbaX to determine
    transporter ID entered. It sets the FROM code to 6 and switches the
    appropriate lamp on.
 b) From_station reads PORT[$6Ø] and checks cbaX XXXX to determine which
    station was entered. It switches the appropriate lamp on and sets the
    FROM code to : 1 : Hot Cells A
                   2 : Hot Cells B
                   3 : Vault 1
                   4 : Vault 2
                   5 : Storage
 c) To_stations_or_pl reads PORT[$61] and checks XXXX cbaX to determine which
    station was entered. It switches the appropriate lamp on and sets the
      TO code to : 1 : Hot Cells A
                   2 : Hot Cells B
                  31 : Vault_1 1
                  32 : Vault_1 2
                  33 : Vault_1 3
                   4 : Vault 2
                   5 : Storage
                   6 : Parking Loop
 d) Cancel_entry reads PORT[$61]  bit ØXXX XXXX. The lamp will be switched
    on for 3ØØms. The TO code and FROM code is then set to Ø and the lamps
    on the front panel will be switced off.
 e) Execute  i)Switch on the lamp for 3ØØms
            ii)Check if the FROM code is Ø ,complain if so and EXIT.
           iii)Check if the TO code is Ø ,complain if so and EXIT.
            iv)Check if the TO code = FROM code ,complain if so and EXIT.
             v)Check if a transporter is actually parked in the FROM station
               that was entered. If not it will complain and EXIT.
            vi)Check if the station is empty to which the TO command wants
               it to go. If not it will complain and EXIT.

    VAR
     from_code,to_code : INTEGER;

}



VAR
 valid_commands,station_empty : BOOLEAN;


PROCEDURE from_strobe;

BEGIN
 DELAY(2Ø);                                        {Contact bounce}
 dollar35 := dollar35 OR $1Ø;                      {Strobe high}
 PORT[$35] := dollar35;
 DELAY(1);                                         {Pulse width}
 dollar35 := dollar35 AND $EF;                     {Strobe low}
 PORT[$35] := dollar35;
END;


PROCEDURE to_strobe;
```

```
BEGIN
 DELAY(2Ø);                                          {Contact bounce}
 dollar35 := dollar35 OR 1;
 PORT[$35] := dollar35;
 DELAY(1);
 dollar35 := dollar35 AND $FE;
 PORT[$35] := dollar35;
END;


PROCEDURE from_parking_loop;

VAR
 from_pl_code : BYTE;

BEGIN
 from_pl_code := PORT[$6Ø] AND $F;
 from_pl_code := from_pl_code SHR 1;
 CASE from_pl_code OF
  6 : BEGIN      {Parking loop 1}
       data_new.xp_id := 1;
       dollar34 := (dollar34 OR 5) AND $F5;
       from_code := 61;
      END;
  5 : BEGIN      {Parking loop 2}
       data_new.xp_id := 2;
       dollar34 := (dollar34 OR 6) AND $F6;
       from_code := 62;
      END;
  4 : BEGIN      {Parking loop 3}
       data_new.xp_id := 3;
       dollar34 := (dollar34 OR 7) AND $F7;
       from_code := 63;
      END;
  3 : BEGIN      {Parking loop 4}
       data_new.xp_id := 4;
       dollar34 := (dollar34 OR 8) AND $F8;
       from_code := 64;
      END;
  2 : BEGIN      {Parking loop 5}
       data_new.xp_id := 5;
       dollar34 := (dollar34 OR 9) AND $F9;
       from_code := 65;
      END;
 END;

 PORT[$34] := dollar34;
 from_strobe;
 WRITELN('from_code = ',from_code,
         '  data_new.xp_id = ',data_new.xp_id);
END;


PROCEDURE from_station;

VAR
 from_station_code : BYTE;

BEGIN
 from_station_code := PORT[$6Ø] AND $EØ;
 from_station_code := from_station_code SHR 5;
```

```
CASE from_station_code OF
 6 : BEGIN      {Hot cells A}
     dollar34 := (dollar34 OR Ø) AND $FØ;
     from_code := 1;
     END;
 5 : BEGIN      {Hot cells B}
     dollar34 := (dollar34 OR 1) AND $F1;
     from_code := 2;
     END;
 4 : BEGIN      {Vault 1}
     dollar34 := (dollar34 OR 2) AND $F2;
     from_code := 3;
     END;
 3 : BEGIN      {Vault 2}
     dollar34 := (dollar34 OR 3) AND $F3;
     from_code := 4;
     END;
 2 : BEGIN      {Storage}
     dollar34 := (dollar34 OR 4) AND $F4;
     from_code := 5;
     END;
END;
PORT[$34] := dollar34;
from_strobe;
WRITELN('from_code = ',from_code);
END;


PROCEDURE to_stations_or_pl;

VAR
 to_station_code : BYTE;

BEGIN
 to_station_code := PORT[$61] AND $ØE;
 to_station_code := to_station_code SHR 1;
 CASE to_station_code OF
  Ø : BEGIN   {Parking loop}
       dollar34 := (dollar34 OR $7Ø) AND $7F;
       to_code := 6;
      END;
  1 : BEGIN   {Storage}
       dollar34 := (dollar34 OR $6Ø) AND $6F;
       to_code := 5;
      END;
  2 : BEGIN   {Vault 2}
       dollar34 := (dollar34 OR $5Ø) AND $5F;
       to_code := 4;
      END;
  3 : BEGIN   {Vault1 3}
       dollar34 := (dollar34 OR $4Ø) AND $4F;
       to_code := 33;
      END;
  4 : BEGIN   {Vault1 2}
       dollar34 := (dollar34 OR $3Ø) AND $3F;
       to_code := 32;
      END;
  5 : BEGIN   {Vault1 1}
       dollar34 := (dollar34 OR $2Ø) AND $2F;
       to_code := 31;
      END;
  6 : BEGIN   {Hot cells B}
```

```
      dollar34 := (dollar34 OR $10) AND $1F;
      to_code := 2;
     END;
  7 : BEGIN   {Hot cells A}
      dollar34 := dollar34 AND $0F;
      to_code := 1;
     END;
 END;
 PORT[$34] := dollar34;
 to_strobe;
 WRITELN('to_code = ',to_code);
END;


PROCEDURE cancel_entry;

BEGIN
 dollar35 := dollar35 AND $FB; {Indicate on Local lamp}
 PORT[$35] := dollar35;
 DELAY(300);
 dollar35 := dollar35 OR $4;
 PORT[$35] := dollar35;

 dollar34 := $FF;            {TO code = 15      To switch off all the lamps  }
 PORT[$34] := dollar34;      {FROM code = 15    on the Local controller      }
 to_strobe;
 from_strobe;
 to_code := 0;
 from_code := 0;
 WRITELN('ALL VALUES CANCELLED');
END;



PROCEDURE invalid_commands;

BEGIN
 valid_commands := FALSE;
 write_dotmatrix(mess);
 bleep;
 DELAY(4000);
 display_ready;
END;


PROCEDURE test_station_status(station_status_check : INTEGER);

BEGIN
 CASE station_status_check OF
   1 : BEGIN
        test63 := PORT[$63] AND $20;
        IF test63 <> 0 THEN station_empty := TRUE ELSE station_empty := FALSE;
       END;
   2 : BEGIN
        test63 := PORT[$63] AND $40;
        IF test63 <> 0 THEN station_empty := TRUE ELSE station_empty := FALSE;
       END;
   3 : BEGIN
        test64 := (PORT[$64] AND $E) OR $F1;
        IF test64 = $FF THEN station_empty := TRUE ELSE station_empty := FALSE
       END;
```

```
   4 : BEGIN
        test64 := PORT[$64] AND 1;
        IF test64 <> Ø THEN station_empty := TRUE ELSE station_empty := FALSE;
       END;
   5 : BEGIN
        test63 := PORT[$63] AND $8Ø;
        IF test63 <> Ø THEN station_empty := TRUE ELSE station_empty := FALSE;
       END;
   6 : BEGIN
        test66 := PORT[$66] AND 1;
        IF test66 <> Ø THEN station_empty := TRUE ELSE station_empty := FALSE;
       END;
 END;
 IF (station_status_check DIV 1Ø) = 3 THEN
  BEGIN
   test64 := (PORT[$64] AND $E) OR $F1;
   IF test64 = $FF THEN station_empty := TRUE ELSE station_empty := FALSE;
  END;
END;



PROCEDURE test_station_status_V1(station_status_check : INTEGER);

BEGIN
 CASE station_status_check OF
  31 : BEGIN
        test64 := PORT[$64] AND 2;
        IF test64 <> Ø THEN station_empty := TRUE  ELSE station_empty := FALSE
       END;
  32 : BEGIN
        test64 := PORT[$64] AND 4;
        IF test64 <> Ø THEN station_empty := TRUE  ELSE station_empty := FALSE
       END;
  33 : BEGIN
        test64 := PORT[$64] AND 8;
        IF test64 <> Ø THEN station_empty := TRUE  ELSE station_empty := FALSE
       END;
 END;
END;



PROCEDURE execute;

BEGIN

 dollar35 := dollar35 AND $FD; {Indicate on Local lamp}
 PORT[$35] := dollar35;
 DELAY(5ØØ);
 dollar35 := dollar35 OR 2;
 PORT[$35] := dollar35;

 IF from_code = Ø THEN
  BEGIN
   mess := 'No FROM entered !';
   invalid_commands;
   EXIT;
  END;

 IF to_code = Ø THEN
  BEGIN
```

```
  mess := 'No TO entered !';
  invalid_commands;
  EXIT;
 END;

IF from_code = to_code THEN
 BEGIN
  mess := 'FROM and TO the same station !';
  invalid_commands;
  EXIT;
 END;

IF from_code <> Ø THEN
 BEGIN
  IF (from_code DIV 1Ø) = 6 THEN
   BEGIN
    test67 := PORT[$67] AND 2;
    IF test67 <> Ø THEN station_empty := TRUE ELSE station_empty := FALSE;
   END;
  test_station_status(from_code);
  IF stray_to_station THEN
   station_empty := FALSE;
  IF station_empty = TRUE THEN
   BEGIN
    mess := 'The FROM station entered is empty !';
    invalid_commands;
    EXIT;
   END;
 END;

IF to_code <> Ø THEN
 BEGIN
  IF ((to_code DIV 1Ø) = 3) AND (from_code = 3) THEN
   test_station_status_V1(to_code)
  ELSE
   test_station_status(to_code);
  IF station_empty = FALSE THEN
   BEGIN
    mess := 'The TO station entered is occupied !';
    invalid_commands;
    EXIT;
   END;
 END;

 {At this stage the from and to values will be legal and the
  appropriate stations occupied and empty }

mess := 'Transporter now on route.                ---------------->';
write_dotmatrix(mess);

move_transporter_station_to_station;

stations_store_1 := PORT[$63] AND $EØ;          {  Store  }
stations_store_2 := PORT[$64] AND $ØF;          {   all   }
parking_loop_store_1 := PORT[$66];              {   the   }
parking_loop_store_2 := PORT[$67] AND 3;        {variables}

display_ready;

stray_routine_serviced := TRUE;

END;
```

```
PROCEDURE local_commands_poll;

BEGIN
 test6Ø := PORT[$6Ø] AND $11;
 test61 := PORT[$61];
 IF (test6Ø AND 1) = Ø THEN
  from_parking_loop;
 IF (test6Ø AND $1Ø) = Ø THEN
  from_station;
 IF (test61 AND 1) = Ø THEN
  to_stations_or_pl;
 IF (test61 AND $8Ø) = Ø THEN
  cancel_entry;
 IF (test61 AND $4Ø) = Ø THEN
  execute;
 IF (test61 AND $2Ø) = Ø THEN      {RESET button}
  BEGIN
   stations_update;
   display_ready;
  END;
END;

{PROGRAM POLLING ROUTINE                    POLLROUT.PAS

This program contains the main polling routine in the procedure
called 'poll'. The following inputs are polled:

a) Stations
   If a transporter is removed or inserted in a station the message
   is displayed, the Mimic panel is updated as well as the information
   on the bubble memory.
b) Parking Loop
   The same as for a) but no bubble updating.
c) Power Supplies
   This looks at the two status bits. A message is also diplayed but
   here the operator is prompted to switch the power supplies on from
   the Local controller panel.
d) Switchers
   This checks the Ø degree status of the switchers. If this status
   is broken, the Mimic panel LED's is updated, a message is displayed
   and the program will try to drive the switcher back to zero degrees.
e)Local Controller comands
   In another file.

}


VAR
 switcher_poll_1,switcher_poll_2,
 p_s_poll : BYTE;

PROCEDURE display_ready;

BEGIN
 mess := '---------------- READY ----------------Enter a FROM and a TO posi
 write_dotmatrix(mess);
END;
```

```
PROCEDURE HCA_change;

BEGIN
 IF stations_poll_1 AND $2Ø = Ø THEN
  BEGIN
   mess := 'Transporter was inserted into          Hot Cells A.';
   data.xp_id := 7;
   dollar31 := dollar31 OR 7;
   HCA_display;
  END
 ELSE
  BEGIN
   mess := 'Transporter was removed from           Hot Cells A.';
   data.xp_id := Ø;
   HCA_display;
  END;
 bleep;
 write_dotmatrix(mess);
 DELAY(2ØØØ);
 mem_pos := Ø;
 save_station;
END;


PROCEDURE HCB_change;

BEGIN
 IF stations_poll_1 AND $4Ø = Ø THEN
  BEGIN
   mess := 'Transporter was inserted into          Hot Cells B.';
   data.xp_id := 7;
   dollar31 := dollar31 OR 7;
   HCB_display;
  END
 ELSE
  BEGIN
   mess := 'Transporter was removed from           Hot Cells B.';
   data.xp_id := Ø;
   HCB_display;
  END;
 bleep;
 write_dotmatrix(mess);
 DELAY(2ØØØ);
 mem_pos := 1;
 save_station;
END;


PROCEDURE S_change;

BEGIN
 IF stations_poll_1 AND $8Ø = Ø THEN
  BEGIN
   mess := 'Transporter was inserted into          the Storage.';
   data.xp_id := 7;
   dollar31 := dollar31 OR 7;
   S_display;
  END
 ELSE
  BEGIN
   mess := 'Transporter was removed from           the Storage.';
   data.xp_id := Ø;
```

```
   S_display;
  END;
 bleep;
 write_dotmatrix(mess);
 DELAY(2000);
 mem_pos := 2;
 save_station;
END;


PROCEDURE V2_change;

BEGIN
 IF stations_poll_2 AND 1 = 0 THEN
  BEGIN
   mess := 'Transporter was inserted into          Vault 2.';
   data.xp_id := 7;
   dollar31 := dollar31 OR 7;
   V2_display;
  END
 ELSE
  BEGIN
   mess := 'Transporter was removed from           Vault 2.';
   data.xp_id := 0;
   V2_display;
  END;
 bleep;
 write_dotmatrix(mess);
 DELAY(2000);
 mem_pos := 6;
 save_station;
END;


PROCEDURE V1_1_change;

BEGIN
 IF stations_poll_2 AND 2 = 0 THEN
  BEGIN
   mess := 'Transporter was inserted into          Vault1  1.';
   data.xp_id := 7;
   dollar31 := dollar31 OR 7;
   V1_1_display;
  END
 ELSE
  BEGIN
   mess := 'Transporter was removed from           Vault1  1.';
   data.xp_id := 0;
   V1_1_display;
  END;
 bleep;
 write_dotmatrix(mess);
 DELAY(2000);
 mem_pos := 3;
 save_station;
END;


PROCEDURE V1_2_change;

BEGIN
 IF stations_poll_2 AND 4 = 0 THEN
```

```
 BEGIN
  mess := 'Transporter was inserted into          Vault1  2.';
  data.xp_id := 7;
  dollar31 := dollar31 OR 7;
  V1_2_display;
 END
ELSE
 BEGIN
  mess := 'Transporter was removed from           Vault1  2.';
  data.xp_id := 0;
  V1_2_display;
 END;
bleep;
write_dotmatrix(mess);
DELAY(2000);
mem_pos := 4;
save_station;
END;


PROCEDURE V1_3_change;

BEGIN
 IF stations_poll_2 AND 8 = 0 THEN
  BEGIN
   mess := 'Transporter was inserted into          Vault1  3.';
   data.xp_id := 7;
   dollar31 := dollar31 OR 7;
   V1_3_display;
  END
 ELSE
  BEGIN
   mess := 'Transporter was removed from           Vault1  3.';
   data.xp_id := 0;
   V1_3_display;
  END;
 bleep;
 write_dotmatrix(mess);
 DELAY(2000);
 mem_pos := 5;
 save_station;
END;


PROCEDURE stations_poll_update;

VAR
 I,change : INTEGER;

BEGIN

 test63 := $20;
 FOR I := 1 TO 3 DO
  BEGIN
   IF (test63 AND stations_poll_1) <> (test63 AND stations_store_1) THEN
   change := I;
   test63 := test63 SHL 1;
  END;

 test64 := 1;
 FOR I := 4 TO 7 DO
  BEGIN
```

```
   IF (test64 AND stations_poll_2) <> (test64 AND stations_store_2) THEN
   change := I;
   test64 := test64 SHL 1;
  END;

  CASE change OF
   1 : HCA_change;
   2 : HCB_change;
   3 : S_change;
   4 : V2_change;
   5 : V1_1_change;
   6 : V1_2_change;
   7 : V1_3_change;
  END;
 stations_store_1 := stations_poll_1;
 stations_store_2 := stations_poll_2;
 display_ready;
END;


PROCEDURE parking_loop_poll_update;

BEGIN
 bleep;
 mess := 'Transporter was removed/inserted        in the Parking loop.';
 write_dotmatrix(mess);
 DELAY(2000);
 parking_loop_check;          {To update the Mimic LED's}
 parking_loop_store_1 := parking_loop_poll_1;
 parking_loop_store_2 := parking_loop_poll_2;
 display_ready;
END;


PROCEDURE power_supplies_poll;

BEGIN
 p_s_poll := PORT[$67] AND $C;
 IF p_s_poll <> 0 THEN
  BEGIN
   CASE p_s_poll OF
    4 : BEGIN
         mess := '24V Power Supply OFF                    RESET to continue';
         dollar35 := (dollar35 OR $40) AND $DF;   {PS lamp off   FAULT lamp or
         dollar3A := dollar3A OR 1;               {PS_on command off}
         ps24V_on := FALSE;
        END;
    8 : BEGIN
         mess := '32V Power Supply OFF                    RESET to continue';
         dollar35 := (dollar35 OR $80) AND $DF;   {PS lamp off   FAULT lamp or
         dollar3A := dollar3A OR 2;               {PS_on command off}
         ps32V_on := FALSE;
        END;
    $C: BEGIN
         mess := 'Both Power Supply OFF                   RESET to continue';
         dollar35 := (dollar35 OR $C0) AND $DF;   {PS lamp off   FAULT lamp or
         dollar3A := dollar3A OR 3;               {PS_on command off}
         ps24V_on := FALSE;
         ps32V_on := FALSE;
        END;
   END;
   write_dotmatrix(mess);
```

```
   PORT[$35] := dollar35;
   PORT[$3A] := dollar3A;
   bleep_until_reset;
   mess := 'Check the Power Supply before          you attempt to switch on';
   write_dotmatrix(mess);
   power_supplies;
   bleep;
   dollar35 := dollar35 OR $20;                  {Fault lamp off}
   PORT[$35] := dollar35;
   display_ready;
  END;
END;

PROCEDURE switcher_poll_update;

VAR
 I,change : INTEGER;

BEGIN
 test68 := 2;
 FOR I := 1 TO 3 DO
  BEGIN
   IF (switcher_poll_1 AND test68) <> 0 THEN
   change := I;
   test68 := test68 SHL 3;
  END;
 test69 := 4;
 IF (switcher_poll_2 AND test69) <> 0 THEN
  change := 4;
 CASE change OF
  1 : BEGIN
       mess := 'Switcher no. 1 out of position          RESET to continue';
       green_led := 10;                        {Mimic panel LED to red}
      END;
  2 : BEGIN
       mess := 'Switcher no. 2 out of position          RESET to continue';
       green_led := 20;                        {Mimic panel LED to red}
      END;
  3 : BEGIN
       mess := 'Switcher no. 3 out of position          RESET to continue';
       green_led := 30;                        {Mimic panel LED to red}
      END;
  4 : BEGIN
       mess := 'Switcher no. 4 out of position          RESET to continue';
       green_led := 40;                        {Mimic panel LED to red}
      END;
 END;
 write_dotmatrix(mess);
 dollar35 := dollar35 AND $DF;                 {Fault lamp on}
 PORT[$35] := dollar35;
 update_mleds_switcher;
 bleep_until_reset;
 mess := 'Check the Switcher before you          RESET to continue';
 write_dotmatrix(mess);
 REPEAT
 UNTIL local_reset;
 CASE change OF
  1 : switcher1_to_zero;
  2 : switcher2_to_zero;
  3 : switcher3_to_zero;
  4 : switcher4_to_zero;
```

```
 END;
 dollar35 := dollar35 OR $2Ø;                    {Fault lamp off}
 PORT[$35] := dollar35;
 display_ready;
END;


PROCEDURE poll;

BEGIN

 stations_poll_1 := PORT[$63] AND $EØ;
 stations_poll_2 := PORT[$64] AND $ØF;
 IF (stations_poll_1 <> stations_store_1) OR
  (stations_poll_2 <> stations_store_2) THEN
  stations_poll_update;

 parking_loop_poll_1 := PORT[$66];
 parking_loop_poll_2 := PORT[$67] AND 3;
 IF (parking_loop_poll_1 <> parking_loop_store_1) OR
    (parking_loop_poll_2 <> parking_loop_store_2) THEN
  parking_loop_poll_update;

 power_supplies_poll;

switcher_poll_1 := PORT[$68] AND $92;
switcher_poll_2 := PORT[$69] AND 4;
IF (switcher_poll_1 <> Ø ) OR (switcher_poll_2 <> Ø) THEN
 switcher_poll_update;

local_commands_poll;

END;

VAR
sw_end : BOOLEAN;
key_in,switch_in : INTEGER;


PROCEDURE start_up;

BEGIN
 dotmatrix_initialize;
 variables_initialize;
 dollar3A := dollar3A OR 4; {Switch on local controller 24V ØØØØ Ø1ØØ}
 PORT[$3A] := dollar3A;
 WRITELN('Local controller 24V psu on.');
 ps24V_status := 1;
 ps24V_status := 1;
 ps24V_on := FALSE;
 ps32V_on := FALSE;
 from_code := Ø;
 to_code := Ø;
 cancel_entry;
 bleep;
 mess := '-----  THE TARGET TRANSPORT SYSTEM  ----Switch on the Power Supplies
 write_dotmatrix(mess);
 power_supplies;
 DELAY(2ØØØ);
 switcher_status_check;
 switcher_check;
```

```
{This part helps to test faulty switchers and should not be included
 in the working version of the program.
 sw_end := FALSE;
 WRITELN('SW1 to Ø : 1Ø     SW1 to 18Ø : 18');
 WRITELN('SW2 to Ø : 2Ø     SW2 to 18Ø : 28');
 WRITELN('SW3 to Ø : 3Ø     SW3 to 18Ø : 38');
 WRITELN('SW4 to Ø : 4Ø     SW4 to 18Ø : 48');
 WRITELN;
 WRITELN('To terminate : 5Ø');
 REPEAT
 READLN(switch_in);
 CASE switch_in OF
  1Ø : switcher1_to_zero;
  18 : switcher1_to_18Ø;
  2Ø : switcher2_to_zero;
  28 : switcher2_to_18Ø;
  3Ø : switcher3_to_zero;
  38 : switcher3_to_18Ø;
  4Ø : switcher4_to_zero;
  48 : switcher4_to_18Ø;
  5Ø : sw_end := TRUE;
 END;
 UNTIL sw_end;
 }

 mess := 'Parking Loop positions check';
 write_dotmatrix(mess);
 DELAY(2ØØØ);
 parking_loop_check;
 parking_loop_positioning;
 stations_update;


 { This part helps to check the tracks for shorts between the sense line
   and the power lines. It should not be included in the working version.
 WRITELN;
 WRITELN;
 WRITELN('Track_display until 1 pressed.');
 REPEAT
  track_display;
  DELAY(1ØØ);
  READLN(key_in);
  DELAY(1ØØ);
 UNTIL key_in = 1;
 }

 stray_check;

 stations_store_1 := PORT[$63] AND $EØ;
 stations_store_2 := PORT[$64] AND $ØF;
 parking_loop_store_1 := PORT[$66];
 parking_loop_store_2 := PORT[$67] AND 3;

 display_ready;
END;


BEGIN
 start_up;
 REPEAT
  poll;
 UNTIL KEYPRESSED;
```

6. TESTING AND MODIFICATIONS

6.1 LOCAL CONTROLLER AND MIMIC PANEL

An instrument, consisting of switches and LED's, was made to simulate the SABUS I/O cards. This was then used to test the local controller and mimic panel once they were assembled. Because of noise problems, a few capacitor filters had to be added, especially on the strobe inputs.

Initially only one 24V power supply was used. The 5V was generated by a voltage regulator and bypass transistor, but after some testing was done, it was found that a separate 5V supply was needed. This was then added in the local controller without any problem.

6.2 SOFTWARE

Because the program was divided into include files, these could be tested individually. A panel consisting of 80 miniature toggle switches and 96 LED's were designed and assembled to assist in the software testing. The 80 switches were connected to the 80-bit opto-isolated input card and represented the inputs from the system. The 96 LED's were connected to the three 32-bit relay output cards and represented the outputs to the system. After weeks of writing source code, compiling and testing, the final program emerged containing thirteen include files and using 30kbytes of RAM space.

6.3 COMPLETE SYSTEM

For testing the complete system a close circuit TV camera was used. By entering commands on the local controller in the isotope-production control room, the movement of transporters and switchers could be followed on a monitor. The biggest problem that occurred was that when power was applied for the first time, all the contacts on the 32-bit relay output cards were closed. This powered all the relays driving the track sections and switcher motors resulting in a total chaos of moving transporters and switchers. This problem was overcome by introducing a small relay timer card between the local controller 5V and 24V power supplies. When mains is switched on, only the 5V power supply is powered. The 24V power supply is only powered via the 32-bit relay card once the relays are in the correct state. The circuit diagram is shown in figure 6.3.1.

A number of wiring and cabling faults were encountered and rectified without any problems.

The stopping of a transporter is achieved by sensing its presence at stations and on switchers by means of reed switches activated by magnets on the transporter chassis. This method of sensing

proved to be unreliable and the reed switches were replaced by limit switches. Limit switch activators were made and fixed onto the transporters' chassis.

7. COMMISSIONING

The system was commissioned in October 1988 after extensive testing. All the possible routes were tested a number of times and power failures were simulated to observe the systems reaction on power restoration.

8. CONCLUSIONS

To control a system of this nature, two methods can be used namely a direct hardware control or a combination of software and hardware control. To control it purely by hardware, a dedicated hardware controller has to be used. On system break down and

malfunction it will require a skilled person to rectify it and will invariably take some time. Using a hardware/software combination, as in this case, has a number of advantages. First of all the Z-80 microprocessor and SABUS I/O cards are standard NAC in-house modules which can be replaced quickly because of their availability. Secondly, should the software crash while running, it can be reloaded from the bubble memory on SABUS, by simply pressing the reset button on the microprocessor. Thirdly, all the commands to the microprocessor are entered on pushbutton switches which are relatively trouble free. Fourthly the hardware part consists of straight forward wiring to relays which also give relatively trouble free operation. Lastly it is a very flexible system because software can be changed to suit new processes and needs as often as required. It is therefore much more advantageous to use a microprocessor based system in such applications. If one compares the previously used hardware system, with its ratsnest of wiring, to this control system, one can clearly see its supremacy.

Entering commands on the local controller pushbuttons instead of a keyboard, narrows down the possibility of operator typing errors. It also eliminates the need to remember complicated command strings.

By using a mimic panel, the entire system status can be seen at a glance. It also indicates transporter progress on the tracks in areas where CCTV cameras cannot be used. The use of CCTV cameras can become quite expensive.

Although a high degree of reliability was designed and built into the control system and thoroughly tested, its real test will only come with time.

9. REFERENCES

1. Borland International,1984,Turbo Pascal reference manual,BI.

2. Borland International,1986,Turbo Pascal Tutor,BI.

3. General Instruments,1985,Catalog of optoelectronic products,GI.

4. National Accelerator Centre,1985,Annual Report,NAC.

5. National Accelerator Centre,1986,Annual Report,NAC.

6. National Accelerator Centre,1987,Annual Report,NAC.

7. National Semiconductors,1984,Logic data book Volume 2,NS.

8. Optrex Corporation,1987,Dot matrix LCD module user's manual,OC.

9. Texas Instruments,1981,The TTL data book for design engineers,TI.