

GETTING STARTED WITH AN 1185 AND A PET

1. Connecting up the interface.

1.1 Interface mounted in a box.

Refer to Fig. 1 which shows the connections required between the interface and the drive(s). This circuit shows connections to three drives - if only one or two drives are used, connect the unused FAULT inputs on the interface to Ov. When using Emergency Stop or Stall Detector facilities, refer to manual.

The Bus connection is made using the 25-way D-connector. Page 22 of the manual shows the connections required on the Commodore PET PCB connector. Use an adaptor block or IEEE to IEC lead if the controller has an IEEE connector.

1.2 Interface card mounted in a rack.

Refer to Fig. 2 and connect up accordingly. If only one or two drives are used, connect the unused FAULT inputs on the interface to Ov. Refer to manual if using Emergency Stop or Stall Detector Facilities.

The Bus connection appears on the 26-way strip cable header and Page 22 of the manual shows the connections required. The strip cable may be used as a transition to a 25-way D-connector if required.

1.3 General checks.

Ensure that the 24v supply connections are correct before powering up the unit, and check that the actual supply voltage is between 20 and 28 volts.

2. Setting the interface bit switches.

2.1 Primary address.

Each interface in a system is identified by its primary address which is set up on the bit switches D to H on the smaller PCB. The significance of each switch is shown below :-

D	-	16
E	-	8
F	-	4
G	-	2
H	-	1

The numbers are additive, and a switch is selected in the 'OFF' position. So to set up primary address 5, switches F and H should be off and D, E and G should be on.

2.2 Data coding.

The interface may be set up to accept data either in BCD or hexadecimal.

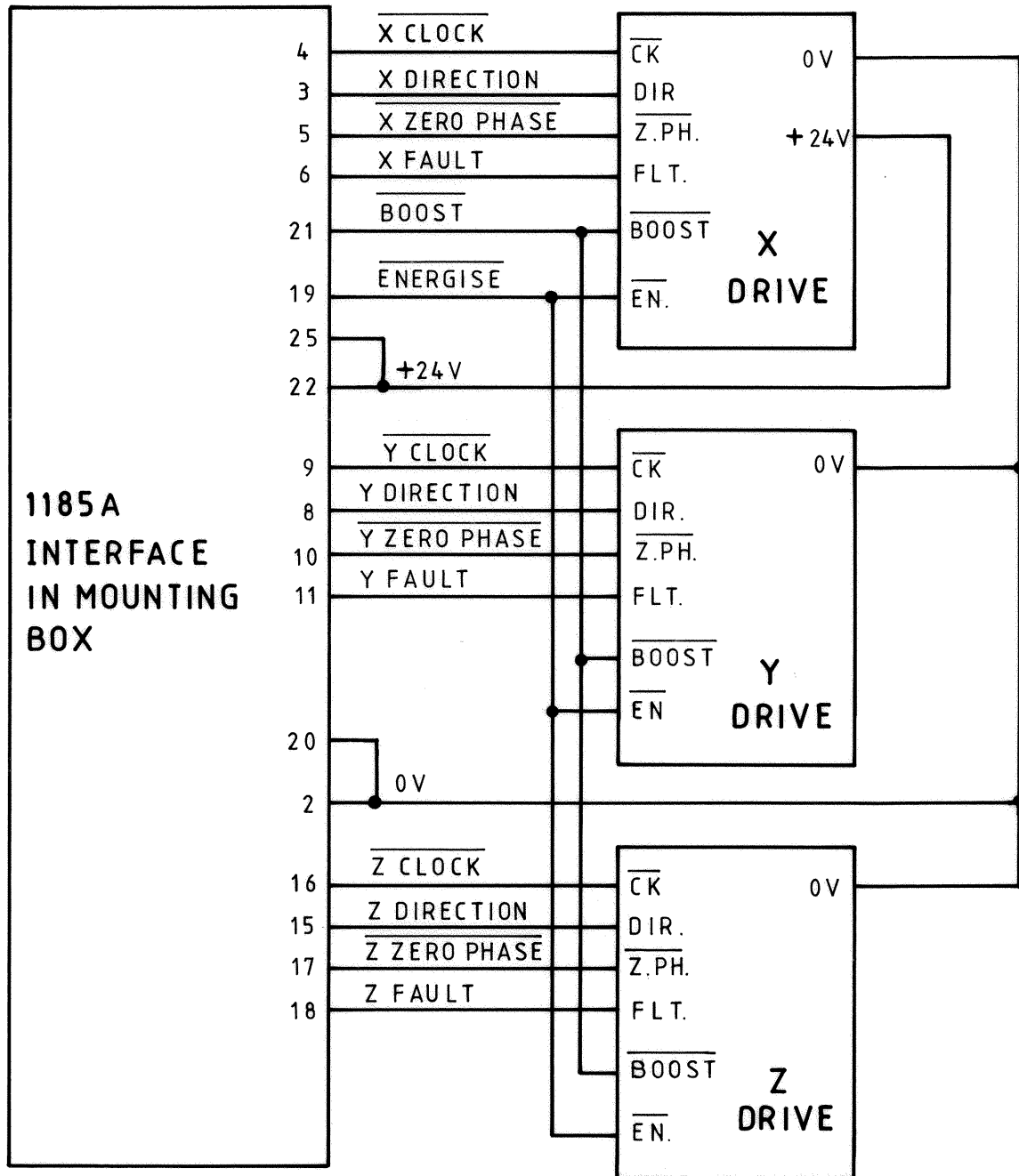


FIG. 1 DRIVE CONNECTIONS - INTERFACE IN MOUNTING BOX

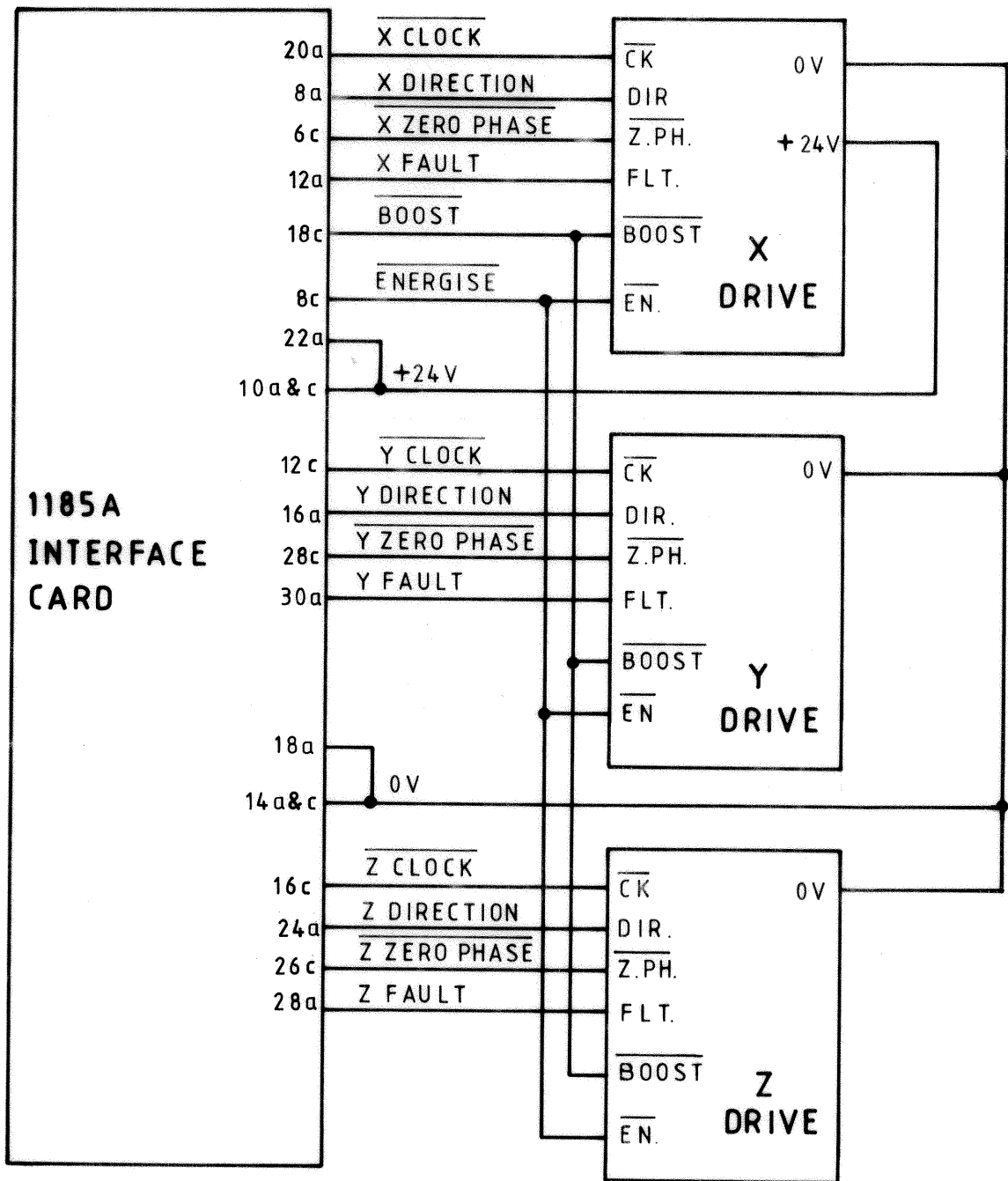


FIG. 2 DRIVE CONNECTIONS - INTERFACE CARD IN A RACK

BCD is used with languages such as BASIC but hex may be required with special-purpose controllers. The code is determined by bit switch C as follows :-

C = OFF :- BCD
C = ON :- Hexadecimal

2.3 Start/Stop speed.

This is the speed at which the motor will start when accelerating up to higher speeds. Below this speed there will be no acceleration or deceleration. The standard versions of the interface offer a choice of start/stop speeds as shown in the table. The start/stop speed is fixed at 400 steps/sec. on the linear interpolation version (1185A-105).

Switch Position		Speed	Speed
A	B	1185A-101	1185A-102
ON	ON	100 s/s	200 s/s
OFF	ON	200 s/s	400 s/s
ON	OFF	400 s/s	800 s/s
OFF	OFF	800 s/s	1600 s/s

3. Preliminary checks.

The first task is to verify that the interface responds when its primary address is sent. The following and all subsequent examples assume that the interface is set to address 5, and that the system is programmed in BASIC.

Load the following instructions, pressing 'RETURN' at the end of each line :-

```
OPEN#5,5  
PRINT#5,"X100@50$"
```

The X motor should now run for 100 steps at a rate of 50 steps/sec. If Y and Z motors are used, check them in turn using the following instructions :-

```
PRINT#5,"Y100@50$"  
PRINT#5,"Z100@50$"
```

Having verified that the controller and interface are talking to each other, we

can now go on to cover the structure of the program instructions.

4. Indexing instructions - single axis.

The first example illustrates the basic form of an indexing instruction, namely :-

```
PRINT#5, "X100@50$"
```

The first part, PRINT#5, instructs the interface with address 5 to carry out the operation defined between the quotation marks. The order in which this instruction is presented is :-

```
axis
distance (steps)
'@' sign
speed (steps/sec)
'$' sign
```

Therefore the above instruction calls up the X axis to run for 100 steps at 50 steps/second. It is important for the instruction to be presented EXACTLY as shown, complete with comma, quotation marks etc. for the interface to operate correctly.

To change the direction of rotation, a minus character is included after the axis code. Hence to perform the same move in the opposite direction, use the following instruction :-

```
PRINT#5, "X-100@50$"
```

It is not in fact necessary to repeat the index and speed data if it has not changed.

For example :-

```
PRINT#5, "X$"
```

will cause the motor to run the same distance and speed as previously in the positive direction.

Similarly :-

```
PRINT#5, "X-@10$"
```

will cause it to run 100 steps in the negative direction but at 10 steps/sec. Note that if speed and distance data are not sent, the motor will perform the

same operation as the last axis that was moved, for example :-

```
PRINT#5,"X100@50$"  
PRINT#5,"Y25$"  
PRINT#5,"X$"
```

This last X move will be 25 steps and not the 100 steps previously moved by X.

To run at speeds above the stop/start speed, it is necessary to define the required rate of acceleration and deceleration. The interface produces linear acceleration and deceleration ramps which have the same slope, but a choice of 16 different slopes is available. To program the slope into the interface (which will remember it unless overwritten as with other data) use the '↑' code followed by a number between 0 and F from the following table :-

<u>Program Number</u>	<u>Slope (Steps/Sec²)</u>
0	7000
1	10040
2	13030
3	18560
4	24500
5	33100
6	43750
7	61250
8	81666
9	111360
A	153125
B	204166
C	306250
D	408333
E	612500
F	1225000

The acceleration rate is normally included at the end of the instruction, for example :-

```
PRINT#5,"X5000@1500↑6$"
```

will load an acceleration rate of 6 and cause the motor to run 5000 steps at 1500 steps/sec. The acceleration rate may be loaded without requesting any movement, for example :-

```
PRINT#5,"↑6$"
```

will load the acceleration rate 6.

Similarly :-

```
PRINT#5, "X↑3$"
```

will index X the previous amount at the previous rate with a new acceleration rate of 3. An index may be cancelled before it has been completed by sending the cancel code '#'.
Thus :-

```
PRINT#5, "#$"
```

will terminate an index on any axis.

5. Run instructions - single axis.

The motor may be run continuously by sending the run code 'R' after the axis code. It will run at the previously-loaded speed and acceleration rate unless new data is included in the instruction. Therefore :-

```
PRINT#5, "XR$"
```

will start the X axis running at the previous rate.

Similarly :-

```
PRINT#5, "X-R@5000↑4$"
```

will start the X axis running at 5000 steps/sec. and an acceleration rate of 4 in the negative direction. To stop the motor, use the cancel code as when terminating an index.

6. Multiple-axis instructions.

Where more than one motor is used on a single interface, it is possible to instruct them to perform the same operation simultaneously, i.e. to run at the same speed and distance, although the directions may be different. For example :-

```
PRINT#5, "XY100@50$"
```

will cause both X and Y motors to run 100 steps positively at 50 steps/sec.

Similarly :-

```
PRINT#5, "XZ-@5$"
```

will index X positively and Z negatively the previous increment at 5 steps/sec. The same rules apply to run instructions, for instance :-

```
PRINT#5, "X-YR@25$"
```

will start X running negatively and Y positively at 25 steps/sec. As another example :-

```
PRINT#5, "XYZR↑7$"
```

starts all 3 axes running positively at the previously-loaded speed and a new acceleration rate of 7.

To terminate motion, use the cancel code as before.

7. Program example.

The following simple program will illustrate simple indexing operations. By entering line numbers the program will not run until 'RUN' is keyed in.

Load the following, pressing 'return' at the end of each line :-

```
10 OPEN5,5
20 PRINT#5, "X4000@1600↑1$"
30 GOSUB1000
40 PRINT#5, "X-@400$"
50 GOSUB1000
60 PRINT#5, "X2000@2000↑6$"
70 GOSUB1000
80 PRINT#5, "X- $"
90 GOSUB1000
100 CLOSE5
110 END
1000 OPEN6,5,0
1010 GET#6, A$, X$, X$
1020 R=ASC(A$)
1030 IF(RANDB)=8 THEN 1010
1040 CLOSE6
1050 RETURN
```

To run the program, type 'RUN' and press 'return'.

This will perform the following operations :-

1. Run X for 4000 steps at 1600 steps/sec, acceleration rate of 1.
2. Return in the opposite direction at 400 steps/sec.
3. Run X for 2000 steps at 2000 steps/sec, faster acceleration rate of 6.
4. Return at same speed.

The subroutine starting at 1000 allows the controller to establish when the operation is completed and it can pass on to the next instruction. For an explanation of this subroutine see Section 9.

8. Additional operations.

8.1 Stop on zero phase.

The asterisk '*' code causes the motor to stop on the next zero phase position or, if it was previously stationary, to advance to the next zero phase position.

```
PRINT#5, "XZ*#"
```

will cause both X and Z axes to move to the next zero phase position.

A minus sign may be inserted after the axis code to reverse the direction of approach to zero phase.

8.2 Energise.

One energise output is provided which may be taken to any or all drives served by the interface. The drives will automatically be energised when power is applied. The 'P' code is used to energise, and the cancel function '#' followed by 'P' to de-energise. Therefore to de-energise the drives send

```
PRINT#5, "#P"
```

The 'energise' instruction will normally be combined with a request to move, thus :-

```
PRINT#5, "XP5#"
```

will energise all the connected drives and index the X axis 5 steps at the previous rate. Drives may be energised without motion by sending

```
PRINT#5, "P"
```

8.3 Boost.

As with 'energise', one boost output is provided which may be connected to any or all drives. Boost is applied by sending the '>' code.

For example :-

```
PRINT#5,"X>$"
```

will apply boost and index X the previous amount. Boost is cancelled using the cancel code, i.e.

```
PRINT#5,"#>"
```

9. Getting status information back from the interface.

The control computer may wish to get certain information back from the interface such as whether the motor is running or not. In addition, the interface may wish to tell the controller about a problem that has arisen such as a drive fault. Such data is referred to as STATUS information and falls into two categories :

1. FAULT STATUS

If a fault situation arises, the interface alerts the controller by sending back a 'Service Request'. This does not tell the controller what is wrong, just that there is a problem that it thinks the controller should know about. The controller may then ask the interface for its 'fault status' which includes the following possibilities :

Data fault (the interface has received a character which it can't recognise)

Zero phase fault (it has tried to get to zero phase but can't find it)

Emergency stop

Drive Fault

Stall Fault (when a stall detector is used).

2. AXIS STATUS.

The following information comes under this heading :-

Instruction completed
Acceleration complete
Motion (i.e. motor running)
X, Y or Z on zero phase

A Service Request is also generated when motion is complete.

To get status information back, an instruction of the following type is used :

OPENG,5,0

The 6 is a file number (use a different number from the one used previously). The second digit (5 in this case) is the primary address of the interface. The third digit is 0 for axis status, and 4 for fault status. These last two digits comprise the 'Secondary Command Group' (SCG) code.

This instruction is followed by :

GET#6,A\$,X\$,X\$

In response to this the interface sends back 3 characters - a status character, followed by line feed and carriage return. The one we are interested in is the status character which is loaded into the 'A' string.

This status character comprises 8 bits, and each bit represents one of the axis status or fault conditions (apart from certain bits which are not used). So the complete status character might look like this :

00000100

The bits are numbered from 1 to 8, with bit 1 the least significant bit (i.e. on the right hand side). If the example shown is a fault status character, the '1' in the third position indicates a drive fault.

The significance of each bit in the character is shown below:

(1) Axis status character :

<u>BIT NO.</u>	<u>CONDITION</u>	<u>DECIMAL SIGNIFICANCE</u>
1	(not used)	1
2	Instruction complete	2
3	Acceleration complete	4
4	Motion	8
5	Zero phase X	16
6	Zero phase Y	32
7	Zero phase Z	64
8	(not used)	128

(2) Fault status character :

<u>BIT NO.</u>	<u>CONDITION</u>	<u>DECIMAL SIGNIFICANCE</u>
1	Data or zero phase fault	1
2	Emergency stop	2
3	Drive fault	4
4	Stall fault	8
5	(not used)	16
6	(not used)	32
7	(Service Request)	64
8	(not used)	128

The program must therefore analyse the status character according to the information required. This character is of course a binary number with the digits having the decimal significance 1, 2, 4, 8 etc., as shown in the right-hand column. A logical-AND operation is used to test for the presence of a particular bit by using the decimal number corresponding to that bit position. Let's take as an example the common requirement to test for the absence of motion, indicating that the last move has been completed. The 'motion' status is conveyed by bit 4 of the axis status character, and this bit is equivalent to decimal 8. We can write a subroutine to test for motion completed as follows :

```
1000 OPEN6,5,0
1010 GET#6,A$,X$,X$
1020 R=ASC(A$)
1030 IF(RAND8)=8THEN1010
1040 CLOSE6
1050 RETURN
```

The first two instructions fetch the axis status from the interface as explained previously, and load it into 'A' string. Line 1020 makes the variable R equal to the decimal number equivalent to the 'A' string, i.e. the axis status character. The logical-AND operation is carried out next, and is a means of looking for a specific bit in the status character. If this bit is present, then AND-ing with 8 will give the answer 8. If it is not present the answer will be zero. So whilst motion is present the result of the AND operation is 8 and the program goes back to line 1010. This continues until motion stops, when the result is zero and the program continues after closing file 6.

10. AXIS POSITION REQUEST

If motion is terminated for any reason, such as emergency stop, the controller may want to find out how far the motor moved before it was stopped. It can do this by making an Axis Position Request in a similar way to Status Request. However there may be a certain amount of juggling required before the information that comes back is in a useable form.

The number that the interface sends back is in hexadecimal code, and in the case of an indexing instruction it shows the number of steps remaining. Therefore to find out how much of the index was completed the received number must be subtracted from the total index. If the system was operating in the 'run' mode then the received number must be subtracted from the hexadecimal number FEFF0 to find the distance moved. In either case it will usually be necessary to convert the hexadecimal number to decimal.

To obtain position information, the SCG code must contain the primary address followed by 8. Thus the first instruction will be of the form

```
OPEN10,5,8
```

This is followed by an Input statement :

```
INPUT#10,D$
```

to which the interface responds by returning a 5-character hexadecimal message which is loaded into 'D' string.

The following program is an example of how the hexadecimal number may be converted to decimal, the result being stored in D. Remember that this indicates the number of steps left in the indexing mode, and in the run mode must be subtracted from 1044464 (equivalent to hex FEFF0) to find the distance moved.

```
100 OPEN10,5,8
110 INPUT#10,D$
120 CLOSE10
130 D=0
140 FORX=(LEN(D$)-1)TO0STEP-1
150 S=LEN(D$)-X
160 W$=MID$(D$,S,1)
170 IFW$<"A"GOTO190:IFW$>"F"GOTO190
180 D=D+16↑X*(ASC(W$)-55):GOTO200
190 D=D+16↑X*(VAL(W$))
200 NEXT
```