Chapter 25

The DEC 338 display computer

Introduction

The C(display; 'DEC 338) is a C('DEC PDP-8) with a P.display which can connect to T(#1:8; CRT; display; area: 9.375 × 9.375 in.[2]). The PMS structure is shown in Fig. 1, Chap. 5, describing the PDP-8. The Pc ISP is given in Appendix 1 of Chap. 5.

The C('338), although designed to stand alone, is generally used as a satellite to a larger C, via an L(Dataphone). The rationale for using a C as a T is based on the bandwidth and storage requirements needed to maintain graphical picture displays. A human being manipulating pictures (rotation, scale change, and conversion of internal linked data structure to a picture structure) requires short response time; this requirement places high processing demands on larger C's. Thus this C(display) is a preprocessor for larger, more general C's.

The actual T(CRT) is a 16-inch CRT with a $9\frac{3}{8}$-inch square viewing area covered by 1,024 × 1,024 (XY) points. The diameter of the points is ~0.015 inch. The spot is magnetically deflected and focused. All eight T(CRT)'s can be driven together or used independently. A photomultiplier connected through a fiber-optic bundle link is used as a light pen (a photosensitive sensor) to detect spots on the T. The light pen allows the P.display to detect whether a user has "pointed to" a displayed spot.

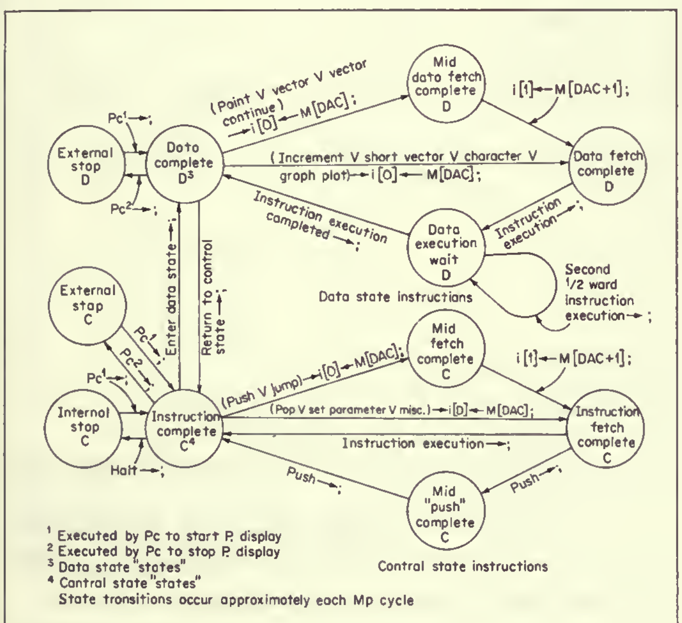

Fig. 1. DEC 338 instruction-interpretation state diagram.

Pc and P.display access the same Mp; the total data rate available from Mp is one 12-bit word/1.5 microseconds. The instruction times of P.display are a function of the point plotting times of the T(CRT):0.3 microsecond to the next incremental unintensified point (approximately 0.010 inch away); 1.2 microseconds to an incremental intensified point; and 35 microseconds to a point plotted at a random position.

The state (registers) of C.display is given in the ISP description of Appendix 1 of this chapter. There are four parts of the state: the control registers for Program Flow State, the Picture State (or position of beam), Console and Light-pen State, and Mp State. The instruction interpreter is fairly simple and is best described by the state diagram (Fig. 1). The instructions are given in Tables 1 and 2. The remainder of the chapter discusses the P.display instructions and the Pc instructions for communicating with P.display.

Principle of operation

The actual picture is held stationary by repeatedly displaying (intensifying) a particular point, line, etc. The number of times a figure has to be displayed so that it appears stationary and does not flicker depends on the CRT phosphor, the figure, and environmental parameters. The generally accepted range is a plotting rate of 20 ~ 50 plots/second; thus a complete picture has to be drawn in 50 ~ 20 milliseconds. If we assume a 30-Hz plot rate, about 28,000 points can be plotted in vector mode (or 280 ~ 1120 inches, depending on the spacing). About 1,000 characters can be displayed in 30 milliseconds using character mode.

When the light pen is used, a display program is required to "track" the pen. The pen's position is determined by displaying known points. The pen, of course, detects the points when it is present at the displayed points position; therefore the program knows the location of the pen.

The parameters of interest for a display vary, depending on the application. However, the general parameters are:

Table 1 DEC 338 control-mode instruction set

Instruction Op Code *Bits*	0:2	3	4	5	6	7	8	9	10	11
Parameters	0	set† Scale	Scale ⟨0:1⟩		set lt pen	lt pen	set Intensity	Intensity ⟨0:2⟩		
Mode	1	stop	clear flags	set mode	Data_Mode	⟨0:2⟩		clear sector	clear X, Y	enter Data_State
Jump‡	2	set Scale	Scale ⟨0:1⟩		set lt pen	lt pen	push	Memory field ⟨0:2⟩		
Pop	3	set Scale	Scale ⟨0:1⟩		set lt pen	lt pen	inh§ Data_Mode	inh Scale, lt pen	inh intensity	enter Data_State
Conditional skip	4	reverse¶ test	clear bits after test	complement after test	Push_Buttons ⟨0:5⟩/PB ⟨0:5⟩					
Conditional skip	5	reverse¶ test	clear bits after test	complement after test	Push_Buttons ⟨6:11⟩/PB ⟨6:11⟩					
Arithmetic compare PB	6	0	0	0	Push_Buttons ⟨0:5⟩					
Arithmetic compare PB	6	0	0	1	Push_Buttons ⟨6:11⟩					
Skip on flags	6	0	1	0	skip	skip if not in sector	skip if PB ⟨0:5⟩ = 0	skip if PB ⟨5:11⟩ = 0		
Count	6	0	1	1	count scale	0 → +1 1 → −1	count Intensity	0 → +1 0 → −1		
Set slaves	6	1	Group number ⟨0:1⟩		set unit 0	lt pen	Intensity	set unit 1	lt pen	Intensity
Spare	7									

† Set; allow instruction bits to specify new value.

‡ A two-word instruction, second word contains low-order 12 bits for DAC (jump address).

¶ Skip can be for true or false.

§ Inhibit restoration of bits.

1 Picture
   a Display area
   b Phosphor type (intensity and color as function of time)
   c Spot size
   d Resolution
   e Linearity
   f Short-term and long-term stability

2 Figure plotting (generation) characteristics
   a Data types: points, lines (vectors), graphs, characters (from a fixed set), characters (from a defined set), curved-line segments, etc.
   b Plotting time

3 Transformation and internal representations
   a Space to encode (specify) a figure
   b Scale change, rotation, coordinate-system transformation abilities
   c Ability to communicate between a displayed data structure and an internal representation of a picture

4 Light-pen or graphic input capability

Instructions and their interpretation in P(display)

Two instruction-set types are interpreted in the P.display: Data State, in which instructions specify display information; and Control State, in which instructions specify program control information (e.g., jumps, modes, etc.). A state diagram for the interpretation process is given in Fig. 1.

Data-state instructions

There are seven instructions (which DEC calls modes) that can be executed while P.display is in data state. The instructions (modes) are really substates of data state. The instructions (actually more like data) are interpreted for the mode. When all the data-mode instructions have been interpreted, an escape instruction returns the P.display to control state. A control instruction is issued to select a mode and simultaneously place the display in data state.

Increment mode. This mode is used to draw curves and alphanumeric characters and other small symbols. Two instructions are stored per word. An instruction will cause the beam position to be moved one, two, or three times, in 0.010-inch increments, in one of eight directions. Direction 0 is to the right, direction 1 is up and to the right, etc.

Table 2 DEC 338 data-mode instruction set

				Instruction bits:											
Mode	*Function*	*Time (μs)*	*Word*	0	1	2	3	4	5	6	7	8	9	10	11
0	point	6 ~ 35	1 of 2	int[a]	inh[b]	Y coordinate									
			2 of 2	esc[c]	inh	X coordinate									
1	increment	1.5 + 2 × (.9 ~ 3.6)	1	int	move count[d]		move direction[e]			same as bits 0 ~ 5					
2	vector	1 ~ 150	1 of 2	int	±	Delta Y									
			2 of 2	esc	±	Delta X									
3	vector	1 ~ 1,200	1 of 2	int	±	Delta Y									
	continue		2 of 2	esc	±	Delta X									
4	short vector	1.8 ~ 24	1	int	±	Delta Y				±	esc		Delta X		
5	6-bit character	3.75+	1	character 1						character 2					
5	7-bit character	4.5+	1	blank					character						
6	graph plot	6 ~ 35	1	esc	X/Y[f]	Y or X coordinate									
7	spare														

[a] Intensify; turn on beam.

[b] Inhibit; do not set value into Y or X coordinate.

[c] Escape; enter control state.

[d] 0 → move 1 and escape; 1, 2, 3, → move 1, 2, 3.

[e] 8 directions.

[f] 0 → set Y and increment X; 1 → set X and increment Y.

Vector mode. The vector mode is used to draw straight-line segments. This two-word instruction causes the beam position to be moved along a line represented by an 11-bit delta y and an 11-bit delta x.

Vector continue mode. This mode is used to draw a straight line to the edge of the screen. It is similar to vector mode but causes the line to be extended until an "edge" is encountered.

Short vector mode. The short vector mode is used to draw figures composed of short line segments. A one-word instruction specifies a 5-bit delta y and a 5-bit delta x quantity. It is transformed within the display to the same format as vector mode and operates in the same manner.

The preceding modes move the beam by counting the X and Y position registers. The counting is done at 1.2 microseconds per step on an intensified move and at 0.30 microsecond per step on a nonintensified move.

Point mode. Point mode is used for random point plotting. A two-word instruction specifies new Y and/or X coordinates to be placed into the Y and X position registers.

Graph-plot mode. This is used to draw curves of mathematical functions. A one-word instruction has data for the Y or X position register; at the same time, X or Y, respectively, is incremented by a count of one, two, four, or eight, depending on the scale factor.

Point and graph-plot modes operate at a rate depending upon the position of the new point with respect to the previous point. If a point is only one-eighth of the screen away, the delay for beam-settling time is 6 microseconds; otherwise the settling time is 35 microseconds.

Character generation option instructions. The alphanumeric characters or special symbols which make up a character set are stored in Mp in increment mode or short vector mode. These characters can be arbitrarily defined. A 6-bit (or 7-bit) character code in the instruction is used to locate a word in a table in Mp called the dispatch table. The base address of the table is specified by the Starting Address Register/SAR⟨0:5⟩.

SAR may be loaded by instructions from the Pc. The SAR represents the most significant 6 bits of a 15-bit memory address. The character code represents the least significant 6 (or 7) bits. A seventh SAR bit, corresponding to the octal position 100, is used with 6-bit characters as a case bit (i.e., uppercase or lowercase characters) and may be set or cleared with a control character.

A word in the dispatch table has the following format:

Bit 0: If bit 0 is a 1, bits 1 to 11 are used to perform a control function as specified by particular control instructions. If bit 0 is a 0, bits 2 to 11 are combined with SAR to specify the address at which the character definition program starts. (The address bit 2 is common to both the SAR and bit 2 of the dispatch word and so may be specified in either place or in both places.)

Bit 1: Determines the mode in which the character is to be displayed. If bit 1 is a 0, the increment mode is used to plot the character used; if bit 1 is a 1, the short vector mode is used to plot the character.

Control-state instructions

There are six control-state instructions.

Parameter. Parameter is used to set values in scale, light-pen, and intensity registers.

Mode. Mode is used to set up the data-state mode (or data-mode instruction). Mode also is used to stop the display.

Conditional skip. The skip instruction tests the state of the P.display and the pushbuttons.

Miscellaneous. These instructions include both tests and additional parameter control.

Display jump and push-jump subroutine instructions. The display jump instruction has 15 address bits, so that a jump may be executed to any location in the display file within the 32-kw memory.

The display subroutine instructions are push-jump (an extension of the jump instruction) and pop, the return from subroutine. The push-jump works as follows: The current state of the display (Light Pen Enable, Data Mode, Scale, and Intensity) is stored, along with the return address, in two successive locations in the first 4,096 words of memory. The locations are determined by the pushdown pointer, PDP. This pointer is initially set by a Pc instruction. The normal jump is then executed.

To return from a subroutine, the pop instruction is executed. It has no address bits. Its function is to return the display to a previous state by sending the last words on the push-down stack back to the display.

The stack approach to subroutining as implemented on the 338 has certain advantages over the jump to subroutine instruction normally used in Pc's:

1 Memory space is conserved since return address locations are not required in each subroutine in memory.

2 A subroutine can be called any number of times before return to the main routine.

3 Since the state of the display is saved on the stack and subsequently restored, subroutines are truly transparent; that is, after the return they leave the state of the display program the same as before the subroutine call.

4 The subroutines can either retain the same state or change the state of the display by using one or more of the "inhibit restore" bits available in the pop instruction. The programmer can elect independently to inhibit restoration of mode, light pen, and scale, or intensity information.

Instructions in Pc for communicating with P(display)

Instructions in Pc communicate with P.display. The physical connection is by the S('I/O Bus). The in-out transfer instructions in Pc are used to initialize and read the state of P.display.

P.display state initialization from Pc instructions

Set Push Down Pointer from AC
Set Display Address Counter from AC
Set Push Button contents from AC
Set miscellaneous flag and status bits from AC
Set character generator SAR address

P.display status to Pc instructions

Read Push Down Pointer into AC
Read X register into AC
Read Y register into AC
Read Display Address Counter into AC
Read Status words 1, 2, 3, 4, 5 into AC (60 miscellaneous bits of flags, modes, etc.)

Picture debugging modes. These modes aid programmed and picture debugging. A bit can be set to override the nonintensify bit in data-mode instructions. When this bit is a 1, all points and vectors are plotted, whether they are to be intensified or not. The search enable instruction forces the display to run until a particular instruction type is found. The instruction type is specified by the search enable instruction.

APPENDIX 1 DEC 338 DISPLAY PROCESSOR ISP DESCRIPTION

Appendix 1

DEC 338 Display Processor ISP Description (partially complete)

P.display State

Program Flow State

DAC<0:14>	*Display Address Counter; holds memory address of display instruction*
PDP<0:11>	*Push Down Pointer to stack holding subroutine return addresses*
Internal_Stop	*denotes halt by a P.display instruction*
External_Stop	*denotes a request by Pc for P.display to halt*

Data_State and Control_State are two mutually exclusive states. Data_State instructions are interpreted by P.display as points, lines, and characters to be displayed on T. There are 7 modes for specifying the data types. The Data_Mode register holds the data type being interpreted. Control_State instructions include jump to subroutines using the stack, controlling P.display state registers and switching to a specific data mode.

Data_State	
Control_State := ¬ Data_State	
Data_Mode/DM<0:2>	*specifies interpretation of Data_State instructions*
SAR<0:5>	*Starting Address Register: base register of a dispatch table for calling character display subroutines*

Picture State

X<0:12> Y<0:12>	*beam position; only integers in range* $0 \le X \mid Y \le 2^{10+dimension-1}$ *are plotted*
Vertical_edge_flag/Vef	*denotes if beam is within a displayable area*
Horizontal_edge_flag/Hef	*set when beam moves outside the display area*
Edge_Interrupt/EI	
CHSZ	*Character Size, 0 indicates 6 bit character set 1 indicates 7 bit character set*
Scale<0:1> Scale'<0:2> := (Scale<0:1> + 1)	*used to set increment size for Data_Mode instructions, increments are* $\times\ 2^{scale}$
Intensity<0:2>	*brightness of displayed points*
X_dimension<0:1> Y_dimension<0:1>	*maximum dimension of plotting area, 9.375, 18.75, 37.5, 75.0 in*
Beam	*on, to display a point or line; automatically turned off at instruction completion*

Console and Light Pen State

Push_Buttons/PB<0:11>	*register with lights: can be complemented manually or by processor*
Push_Button_Hit/PBH	*flag is set by manually striking any push button*
Manual_Interrupt/MI	*key which is used to interrupt Pc and becomes one when struck*
Light_Pen_Find/LPF	*stops the display and interrupts Pc whenever the Light Pen has seen a displayed spot and the Light_Pen_Enable is a one*
Light_Pen_Enable/LPE	*a bit to enable the Light_Pen_Find flag to cause an interrupt*

Mp State

M[0:7] [0:4095]<0:11>	*primary memory for P.display and Pc*

Instruction Format

instruction/i<0:11>		*The individual instructions fields are defined below. Each instruction type has its own bit field assignments.*
enter_data_state	:= i<11>	*common bits for several instructions*
pb_sense	:= i<3>	*push button control bits*

APPENDIX 1 DEC 338 DISPLAY PROCESSOR ISP DESCRIPTION (Continued)

```
        pb_clear                      := i<4>
        pb_complement                 := i<5>
        pb_select<0:5>                := i<6:11>
        scale_change/sc               := i<3>                       scale (size) control bits
        scale_value/sv<0:1>           := i<4:5>
        light_pen_change/lpc          := i<6>                       light pen test control bits
        light_pen_bit/lpb             := i<7>

Instruction Interpretation Process
    (¬ Internal_Stop ∨ ¬ External_Stop) →                           fetch
        (instruction[0:1] ← M[DAC:DAC+1]; DAC ← DAC + 1; next
        (Control_State ∧ (instruction<0:1> = 2)) → (DAC ← OAC + 1);  2 w instruction
        (Data_State ∧ ((Data Mode = 0) ∨ (Data_Mode = 2) ∨           2 w data
            (Data Mode = 3))) → (DAC ← DAC + 1);
        next Instruction_execution)                                 execute

Instruction Set and Instruction Execution Process
 The following instruction set definition is not complete.  It does not include the complete character instruction definition or
 the miscellaneous and conditional skip instructions.  Most of the instructions are microcoded.
    Instruction_execution := (
        Control Instructions
        parameter<0:11> := i[0]<0:11>                               set parameter instruction format
            parameter_opcode           := (i<0:2> = 000)
            parameter_intensity_change := parameter<8>              set parameter execution
            parameter_intensity<0:2>   := parameter<9:11>
        parameter_opcode ∧ Control_State → (
            scale_change → (Scale ← scale_value);
            light_pen_change → (Light_Pen_Find ← ¬ light_pen_bit);
            intensity_change → (Intensity ← parameter_intensity));

        mode<0:11> := i<0:11>                                       set mode instruction format
            mode_opcode                := (i<0:2> = 001)
            mode_stop_code             := mode<3>
            mode_clear_push_button_flag:= mode<4>
            mode_data_mode_change      := mode<5>
            mode_set<0:2>              := mode<6:8>
            mode_clear_sector          := mdde<9>
            mode_clear_coordinate      := mode<10>
        mode_opcode ∧ Control_State → (                             set mode execution
            mode_stop_code → (Internal_Stop ← 1);
            mode_clear_push_button_flag → (Push_Button_Hit ← 0);
            mode_data_mode_change → (Oata_Mode ← mode_set);
            mode_clear_sector → (X<0:2> ← 0; Y<0:2> ← 0);
            mode_clear_coordinate → (X<3:12> ← 0; Y<3:12> ← 0);
            ebter_data_state → (Oata_State ← 1));
```

APPENDIX 1 DEC 338 DISPLAY PROCESSOR ISP DESCRIPTION (Continued)

```
PB␣1<0:11> := i<0:11>                                       group 1 push button test and set instruction format for
   PB␣1␣opcode := (PB␣1<0:2> = 100)                         Push Buttons 0 to 5
                                                            group 2 (not defined) is for Push Buttons 6 to 11
   PB␣1␣opcode ∧ Control␣State → (                          PB␣1 instruction execution
      pb␣sense ⊕ (pb␣select<0:5> ≡ (PB<0:5> ∧ pb␣select<0:5>)) → ( skip test
         DAC ← DAC + 2);

      pb␣clear → (PB<0:5> ← PB<0:5> ∧ pb select<0:5>); next
      pb␣complement → (PB<0:5> ← PB<0:5> + pb␣select<0:5>));
jump[0:1]<0:11> := i[0:1]<0:11>                             jump and stack push down (subroutine calling) instruction
   jump␣op            := (i[0]<0:2> = 010)                  format
   jump␣push          := i[0]<8>
   jump␣field<0:2>    := i[0]<9:11>
jump␣op ∧ Control␣State → (
   scale␣change → (Scale ← scale␣value);                    jump and push down execution
   light␣pen␣change → (Light␣Pen␣Find ← light␣pen␣bit);
   DAC ← jump␣field□i[1];
   jump␣push → (
      M[PDP + 1] ← DAC<0:2>□LPF□Scale□Data␣Mode□Intensity:
      M[PDP + 2] ← DAC<3:14>;
      PDP ← PDP + 2);
pop<0:11> := i[0]<0:11>                                     stack pop instruction format; subroutine return
   pop␣op␣code                := (i<0:2> = 011)
   pop␣inhibit␣mode           := pop<8>
   pop␣inhibit␣scale␣pen      := pop<9>
   pop␣inhibit␣intensity      := pop<10>
pop␣op␣code ∧ Control␣State → (                             pop execution
   DAC<3:14> ← M[PDP];
   DAC<0:2> ← M[PDP-1];
   ¬ pop␣inhibit␣intensity → (Intensity ← M[PDP-1]<9:11>);
   ¬ pop␣inhibit␣mode → (Data␣Mode ← M[PDP-1]<6:8>);
   ¬ pop␣inhibit␣scale␣change → (
     Scale ← M[PDP-1]<4:5>
     LPF ← M[PDP-1]<3>);
   PDP ← PDP - 2; next
   scale␣change → (Scale ← scale␣value):
   light␣pen␣change → (LPF ← light␣pen␣bit);
   enter␣data␣mode → (Data␣Mode ← 1));
Data Mode Instructions                                      point data instruction format
point[0:1]<0:11> := i[0:1]<0:11>
   point␣intensity        := point[0]<0>
   point␣inhibit␣y        := point[0]<1>
   point␣y<0:9>           := point[0]<2:11>
   point␣x<0:9>           := point[1]<2:11>
   point␣escape           := point[1]<0>
   point␣inhibit␣x        := point[1]<1>
```

APPENDIX 1 DEC 338 DISPLAY PROCESSOR ISP DESCRIPTION (Continued)

```
(Data␣Mode = 000) ∧ Data␣State → (                      point data execution
  ¬ point␣inhibit␣x → (X ← point␣X);
  ¬ point␣inhibit␣y → (Y ← point␣Y);
    point␣intensify → (Beam ← 1);
    point␣escape → (Data␣State ← 0));
vector[0]<0:11>:= i[0:1]<0:11>                           vector data instruction format
   vector␣intensify      := vector[0]<0>
   vector␣escape         := vector[1]<0>
   vector␣dy<0:10>       := vector[0]<1:11>
   vector␣dx<0:10>       := vector[1]<1:11>
(Data␣Mode = 010) ∧ Data␣State → (                      vector data execution
   Y ← Y + vector␣dy;                                    not correct, since the vector from point Y,X to Y+ vector␣dy,
   X ← X + vector␣dx;                                     X+ vector␣dx is plotted
   vector␣intensify → (Beam ← 1);
   vector␣escape → Data␣State ← 0);
vector continue[0:1]<0:11> := i[0:1]<0:11>               vector continue instruction format same as vector
(Data␣Mode = 011) ∧ Data␣State → (                      vector continue execution
   Y ← Y + sign␣extend(vector␣dy);
   X ← X + sign␣extend(vector␣dx);                       not correct, as vector continues plotting until edge is found
   vector␣intensify → (Beam ← 1);
   vector␣escape → (Data␣State ← 0));
short␣vector<0:11> := i[0]<0:11>                         short vector instruction format
   short␣vector␣intensify    := short␣vector<0>
   short␣vector␣escape       := short␣vector<6>
   short␣vector␣dx           := short␣vector<8:11>
   short␣vector␣dy           := short␣vector<1:5>
(Data␣Mode = 100) ∧ Data␣State → (                      short vector execution
   X ← X + sign␣extend(short␣vector␣dx);
   Y ← Y + sign␣extend(short␣vector␣dy);
   short␣vector␣intensify → (Beam ← 1);
   short␣vector␣escape → (Data␣State ← 0));
increment<0:5>                                           increment instruction format; 2 increment/instruction
   increment␣intensify            := increment<0>
   increment␣direction/id<0:2>    := increment<3:5>      1 of 8 directions
   increment␣count/ic<0:1>        := increment<1:2>
      icle    := (ic = 0)                                count 1 and escape to Control␣State
      ic1     := (ic = 1)                                count 1
      ic2     := (ic = 2)                                count 2
      ic3     := (ic = 3)                                count 3
(Data␣Mode = 001) ∧ Data␣State → (                      increment instruction execution
   increment ← i<0:5>; next plot␣increment␣vector; next
   increment ← i<6:11>; next plot␣increment␣vector)
```

APPENDIX 1 DEC 338 DISPLAY PROCESSOR ISP DESCRIPTION (Continued)

```
plot␣increment␣vector := (
   icle → (move␣1␣position; Control␣State ← 1);             move 1 and escape
   ic1 → (move␣1␣position);                                 move 1
   ic2 → (move␣1␣position; next move␣1␣position)            move 2
   ic3 → (move␣1␣position; next move␣1␣position; next       move 3
         move␣1␣position)
Move␣1␣position := (                                        sub process for moving beam
   (id = 0) → (X ← X + Scale);                              1 of 8 positions
   (id = 1) → (X ← X + Scale; Y ← Y + Scale);
   (id = 2) → (Y ← Y + Scale);
   (id = 3) → (Y ← Y + Scale; X ← X - Scale);
   (id = 4) → (X ← X - Scale);
   (id = 5) → (Y ← Y - Scale; X ← X - Scale);
   (id = 6) → (Y ← Y - Scale);
   (id = 7) → (Y ← Y - Scale; X ← X + Scale);
   increment␣intensify → Beam ← 1)

character<0:11> := i<0:11>                                  character instruction format
   6␣bit[0:1]<0:5> := character<0:11>
   7␣bit<5:11>     := character<5:11>
(Data␣Mode = 101) ∧ Data␣State → (                          character instruction execution;
   (CHSZ = 0) → (
      X,Y ← f(M[SAR□6␣bit[0]],M);                           plot function;
      X,Y ← f(M[SAR□6␣bit[1]],M));
   (CHSZ = 1) → (X,Y ← f(M[SAR□7␣bit],M)));                 see text

graph␣plot<0:11> := i[0]<0:11>                              graph data instruction format
   graph␣plot␣escape<0> := graph␣plot<0>
   graph␣plot␣x␣y<0>    := graph␣plot<1>
   graph␣plot␣data<0:9> := graph␣plot<2:11>
(Data␣mode = 110) ∧ Data␣State → (                          graph data execution
  ¬ graph␣plot␣x␣y → (X ← X + Scale'; Y ← graph␣plot␣data; Beam ← 1);
    graph␣plot␣x␣y → (Y ← Y + Scale'; X ← graph␣plot␣data; Beam ← 1);
    graph␣plot␣escape → (Data␣State ← 0))
                                    )                       end Instruction␣execution
```