

**Best
Available
Copy**

AD-778 426

GRAPHIC DISPLAY PROCESSOR PROGRAMMERS
GUIDE

Brian Rosen

Carnegie-Mellon University

Prepared for:

Advanced Research Projects Agency
Air Force Office of Scientific Research

20 January 1974

DISTRIBUTED BY:

NTIS

National Technical Information Service
U. S. DEPARTMENT OF COMMERCE
5285 Port Royal Road, Springfield Va. 22151

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR - TR - 74 - 0698	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER AD 778426
4. TITLE (and Subtitle) GRAPHIC DISPLAY PROCESSOR PROGRAMMERS GUIDE		5. TYPE OF REPORT & PERIOD COVERED Interim
7. AUTHOR(s) Brian Rosen		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Carnegie-Mellon University Department of Computer Science Pittsburgh, Pennsylvania 15213		8. CONTRACT OR GRANT NUMBER(s) F44620-73-C-0074
11. CONTROLLING OFFICE NAME AND ADDRESS Defense Advanced Research Projects Agency 1400 Wilson Blvd Arlington, Virginia 22209		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 61101D A02466
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Air Force Office of Scientific Research (NM) 1400 Wilson Blvd Arlington, Virginia 22209		12. REPORT DATE 20 January 1974
		13. NUMBER OF PAGES 33
		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) <p style="text-align: center;">Distributed by NATIONAL TECHNICAL INFORMATION SERVICE U. S. Department of Commerce Springfield VA 22151</p>		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) GDP2 is a Graphic Display Processor which interprets a Display List stored in Memory. It works in conjunction with a PDP 11 computer to produce pictures, characters etc. The system also includes: <ol style="list-style-type: none"> 1. A Keyboard to communicate with the system 2. An asynchronous line interface (ASLI) which is connected to another computer (our PDP10). 		

DD FORM 1 JAN 73 1473

EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (Continued)

3. A Clock which interrupts the PDP11 60 times per second
4. A Read Only Memory (ROM) which contains a program to restart the Graphics System.
5. (optionally) A Spark Pen, which can be used for graphic input and the pointing function (cursor positioning)
6. (coming) A Remote Restart System which allows restarting the ROM program from the terminal.
7. (coming) A Matrix Multiplier and Clipping Divider for rotation, translation and windowing features for more complex graphics.

Most of the Hardware is physically separated from the Display Tube, Keyboard and Spark Pen. The parts of the system the user has to manipulate or see is in the Terminal Room. The noisy stuff is in the Computer Room. This documentation purports to describe the hardware of the GDP itself.

GRAPHIC DISPLAY PROCESSOR
PROGRAMMERS GUIDE

DEPARTMENT OF COMPUTER SCIENCE

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

Brian Rosen
January 20, 1974
Carnegie-Mellon University

DDC
RECEIVED
MAY 10 1974
C

This work was supported by the Advanced Research Projects Agency of the office of the Secretary of Defense ~~(S)~~ and is monitored by the Air Force Office of Scientific Research.

F44620-73-C-0074

GDP2 is a GRAPHIC DISPLAY PROCESSOR which interprets a DISPLAY LIST stored in MEMORY. It works in conjunction with a PDP11 computer to produce pictures, characters etc. The system also includes:

- 1) A KEYBOARD to communicate with the system
- 2) An ASYNCHRONOUS LINE INTERFACE (ASLI) which is connected to another computer (our PDP10).
- 3) A CLOCK which interrupts the PDP11 60 times per second
- 4) A READ ONLY MEMORY (ROM) which contains a program to restart the Graphics System.
- 5) (optionally) A SPARK PEN, which can be used for graphic input and the pointing function (cursor positioning)
- 6) (coming) A REMOTE RESTART SYSTEM which allows restarting the ROM program from the terminal.
- 7) (coming) A MATRIX MULTIPLIER and CLIPPING DIVIDER for rotation, translation and windowing features for more complex graphics.

Most of the Hardware is physically separated from the Display Tube, Keyboard and Spark Pen. The parts of the system the user has to manipulate or see is in the Terminal Room. The noisy stuff is in the Computer Room.

The Graphics System is an INTELLIGENT TERMINAL. Each system has its own PDP11 which is dedicated to it. The PDP11 can be programmed to assist the GDP to create whatever picture the user wishes. For instance, in the absence of hardware Matrix Multiplier, software on the 11 can do the matrix multiply for the display.

This documentation purports to describe the hardware of the GDP itself. Documentation on the PDP11, Keyboard, Clock, ROM, Spark Pen etc. systems are elsewhere, as is the documentation on the standard PDP11 software system implemented for the Graphics Project.

GDP2 has the following major subsystems:

- 1) Instruction Processing
Fetching and executing of display list instructions
- 2) Character Processing
Translation of character strings to vector lists
suitable for drawing
- 3) Line Drawing
Draws vectors on the screen
- 4) Control Word Processing
Mode and state alteration commands
- 5) UNIBUS/Memory Interfacing

Each of these subsystems is described in detail later, but the basic data flow is as follows.

- 1) The PDP11 starts the Display via the UNIBUS by loading the GDP's **Program Counter (GPC)**
- 2) The GDP fetches an **INSTRUCTION** from the memory location pointed to by GPC.
- 3) The instruction is decoded and executed. There are four different instructions. Each of the instructions has a two bit **OPCODE**, and a 14 bit **OPERAND**. The operand is used as a memory address. If it was an **XQT (Execute)** instruction, a **Display File** is executed which causes characters and/or vectors to be drawn. An **INTERRUPT** instruction causes the PDP11 to be interrupted. **JUMP** and **JUMP TO SUBROUTINE** instructions alter the order of instruction execution (which is normally sequential). After execution of an instruction, a new one is fetched, and the process is repeated.
- 4) If **Character Mode** is enabled, an **Execute** instruction causes the Character processing logic to interpret a **Character String**. The memory operand of the **Execute** instruction is used as a pointer to a list of characters. Two characters pointed to by the **CHARACTER POINTER (CPTR)** register are fetched into the **CHARACTER BUFFER REGISTER (CBUF)**.
- 5) Starting with the low byte of **CBUF**, the characters are indexed into a table of addresses pointed to by a **DISPATCH TABLE BASE ADDRESS REGISTER (DTBAR)**. The table is located in memory.
- 6) The address accessed via the table is interpreted as either a list of vectors to be displayed (the representation of the character) or as an **Interrupt Service Routine** address for the PDP11. In the case of a vector list, the **Line Drawer** is activated to display the character.

1 Major Subsystems

- 7) When the Line Drawer finishes the character description list, or the PDP11 restarts the GDP from the service routine, another character is displayed. If necessary, a new fetch is made before looking up the byte in the Character Dispatch Table.
- 8) Vector lists from character processing or display files causes the Line Drawing operation to occur. Vectors are fetched from memory pointed to by the **VECTOR POINTER REGISTER (VPTR)** into the **VECTOR BUFFER REGISTER (VBUF)**, and drawn on the screen. The drawing operation affects the X and Y position registers XR and YR.
- 9) Completion of a vector list causes either a new character operation or a new instruction operation, depending on which process (character or instruction) initiated the line drawer process.
- 10) Intermixed in Instruction, Character, and Vector lists are **CONTROL WORDS** which cause Mode and State changes. Among other things, there are Control Words which affect the Intensity, Scale, and Format of vectors, turn on and off Character Mode, cause interrupts to the PDP11, effect **SETPOINTS** (loading of X and/or Y registers), and terminate lists.

Most of the registers in GWII have already been mentioned, but here they are all together:

GPC	Graphics Program Counter Points to next instruction to be executed. Incremented by 2 after each fetch	16 BIT
CPTR	Character Pointer Points to next character to be displayed. Incremented by 2 on each character fetch (two characters per fetch).	16 BIT
CBUF	Character Buffer Holds two characters to be "Displayed".	16 BIT
DTBAR	Dispatch Table Base Address Register Pointer to an in-core table of addresses, one address for every possible character code.	16 BIT
VPTR	Vector Pointer Pointer to a list of vectors to be drawn	16 BIT
VBUF	Vector Buffer Holds a vector to be drawn on the screen	16 BIT
XR	X Register Holds current X position of the beam. The register is sign extended to 16 bits so that negative numbers look like regular two's complement integers to the PDP-11.	12 BIT
YR	Y Register Holds current Y position of the beam. Sign extended like XR.	12 BIT
STATE	State Register Contains the SCALE, INTENSITY LEVEL, CMODE, UNBLANK and FORMAT sub-registers.	16 BIT
GIS	Graphics Interrupt Status Used as the second word of an interrupt "Vector" for the PDP-11 during an interrupt. GIS holds the Program Status for the interrupt service routine.	16 BIT
GCSR	Graphics Control and Status Register Holds the GO, CLEAR, CLOCK DIVIDE, WRAP and INTERRUPT ENABLE bits. These will be explained later.	16 BIT

The Instruction Processing Portion of GDP comprises a small, stored program digital processor, although the instruction set is very limited. As we have noted before, there are only four instructions. The format of each instruction is very similar. You can see a diagram of it in Appendix B. Basically, the end bits of the instruction word, bits 15 and 0, are used as the opcode bits. Two bits gives us our four possible instructions. Since the choice of which bits were opcode bits seems a little odd, I might explain that bit zero is never used in an address, since the GDP always accesses full words, and that reserving bit 15 for use as the opcode only restricts Graphics to accessing a 16k word memory. This is no problem since the memory accessible by the GDP is only 8k words to begin with. The address used as an operand to the instruction always refers to a location in the 8K Double Port Memory. This means that bit 14 of the address is ignored at present. (There is a possibility of future expansion to a 16K Double Port Memory). When the operand of the instruction is used, the opcode bits are masked out, so that we always obtain even addresses in the range 0-37776.

The fetch of an instruction is made via GPC into the VPTR register which is used as an instruction buffer. After the fetch, GPC is incremented by two. Thus, prior to actual operation of the instruction, the GPC is pointing to the next sequential instruction.

The **JMP** (jump) instruction simply causes a transfer of the contents of the VPTR register to the GPC. $GPC \leftarrow VPTR$. Another instruction is immediately fetched, on the new location. The **JMP** is an unconditional jump.

The **JMS** (Jump to Subroutine) instruction works very much like a PDP-8 **JMS** instruction. The operand of the instruction is interpreted as a pointer to a graphic subroutine. The first word of the subroutine (thus the word pointed to by the operand) is assumed to be unused. The GDP deposits the present value of the GPC register (which points to the next instruction in the main stream of instructions) into this first location. Then the operand is transferred to GPC, after being incremented by two.

$M(VPTR) \leftarrow GPC$
 $GPC \leftarrow VPTR + 2$

The GDP then proceeds to execute the first instruction in the subroutine (which, of course, is the second word of that subroutine, the first word now containing the **RETURN ADDRESS**). The GDP proceeds to process more instructions in the subroutine until the subroutine ends. The end of the subroutine is indicated by a **JMP** instruction, whose operand is the address of the first word of the subroutine.

Now, when the GDP deposited that return address, it did a sneaky thing,

3 All You Need To Know About Instruction Processing

it blanked out the opcode bits (15 and 0) of the return address word. This is because the opcode of the JMP instruction is 00. The "instruction" deposited in the first location of the subroutine is actually a JMP instruction back to the caller of the subroutine. This avoids the necessity of an indirect addressing bit. Note that subroutines can be nestable, but cannot be recursive or reentrant without some help from the PDP-11.

The third instruction is INTR, which causes an interrupt to the PDP11. An interrupt is generated at Bus Request Level 4. The contents of VPTR (which is the operand of the INTR instruction) is a pointer to a routine, not an interrupt vector. This is contrary to most PDP11 peripheral device interfaces. If you want to know how this is done, read on. If you don't care, skip the next paragraph.

When the PDP11 honors the bus request, it needs an interrupt vector. This vector is passed to the PDP11 by the device requesting an interrupt. Usually, the address is below location 400. Then the computer uses the interrupt vector it obtained to get a new Program Counter and Program Status word.

$$\begin{aligned} PC &\leftarrow M(\text{VECTOR}) \\ PS &\leftarrow M(\text{VECTOR}+2) \end{aligned}$$

In this case however, we already know the desired new PC. Therefore, we pass the address of VPTR to the PDP11, not its contents. This address is in the peripheral bank, normally 165104. When the PDP11 does the access on the UNIBUS to get a new PC, it goes to address 165104 (VPTR) and puts the contents of VPTR into the PDP11 PC. Now, we need a PS. The PDP11 will access address 165106, (address of VPTR+2). It just so happens that address 165106 is an unused register (called GIS for Graphics Interrupt Status). Before starting up the GDP, the programmer should load GIS with a suitable interrupt status word, similar to one used in a normal interrupt vector. This procedure is known as an INTERRUPT @VPTR.

When the PDP11 completes the interrupt service routine, before it executes an RTI instruction, it must continue the GDP. This is done by writing a one into the GO bit of the GCSR. Graphics then continues on, processing the next instruction. It should be noted that the INTERRUPT ENABLE bit of the GCSR must be set before any interrupts can be generated.

The final instruction is XQT, the execute command. It causes the GDP to draw something on the screen. The operand of the XQT instruction is assumed to be a pointer to a list of vectors to be drawn, or a list of characters to be interpreted. The CMODE bit of the STATE register determines whether the character processing logic or the vector process will be activated. If CMODE is a one, character processing is enabled, if CMODE is zero, then vectors are assumed.

When the vector/character process finishes, the GDP is free to fetch and process the next instruction. As we will see later, it is possible to deactivate the character process and activate the vector process while processing an XQT instruction, (and vice versa) by manipulating the CMODE bit with Control Words. Thus the state of the CMODE bit before an XQT instruction determines initial assumptions about the vector/character determination, but the decision can be overriden from within the data.

An example of intruction processing will now be given. The notation used is that of a PDP-11 Assembler format, ie:

```

LABEL:  INSTRUCTION      OPERAND          ;COMMENT

MLOOP:  XQT      BOX          ;DRAW A BOX
        JMP      TG           ;GO TO TAG ROUTINE
        .
        .
        .
TG:     JMS      REPOS        ;POSITION BEAM FOR TAG
        JMS      LABEL       ;DRAW THE LABEL
        INTR     CLKTIC      ;HAVE PDP11 WAIT FOR REFRESH
        JMP      MLOOP       ;THEN REFRESH THE PICTURE AGAIN
        .
        .
        .
REPOS:  0          ;SPACE FOR RETURN ADDRESS
        XQT      SXY         ;POSITION BEAM UNDER THE BOX
        JMP      REPOS       ;SUBROUTINE ROUTINE
        .
        .
        .
LABEL:  0          ;RETURN ADDRRESS SLOT
        XQT      FRSTWD      ;DRAW FIRST HALF OF LABEL
        XQT      RESTLN     ;DRAW THE REST OF THE LINE
        JMP      LABEL       ;RETURN TO CALLER
        .
        .
        .
BOX:    <vector sequence to draw a box on the screen>
SXY:    <vector sequence to reposition beam for tag line,
        and change to CHARACTER MODE>
FRSTWD: <Character list to draw part of the tag line>
RESTLN: <Character list to draw the rest of the line>
    
```

4 How To Use Character Processing

Character processing logic in the GDP translates eight bit character codes into vector lists or PDP-11 service routine calls. Character lists themselves are stored in memory in contiguous bytes, just as the PDP-11 instructions use them. The sequence of displaying a Character list is as follows:

The GDP places the operand of a XQT instruction in the CPTR register. Then a full word fetch is made into the CBUF register .

$$CBUF \leftarrow M(CPTR)$$

$$CPTR \leftarrow CPTR + 2$$

Starting with the low byte in CBUF, the GDP constructs an address by taking the high 7 bits of the Dispatch Table Base Address Register, DTBAR as bits 15-9 and the character code as bits 8-1, always forcing bit 0 to be a zero. Then a fetch is made into the VPTR register.

$$VPTR \leftarrow M(DTBAR \langle 15:9 \rangle + CBUF \langle 7:0 \rangle * 2)$$

This has the result of looking up the character code in a memory table.

The contents of the word accessed by the table is interpreted as an address. The lowest bit in the address is used as a flag. If the flag was a zero, the address is used as a pointer to a list of vectors to be drawn. The vector processing logic is called upon to draw the list of vectors. When it finishes the list of vectors, the vector processing logic returns control to character processing logic.

If the flag was a one, the address is used as a pointer to a PDP11 service routine. This is another case of INTERRUPT @VPTR. The contents of the table value is used as direct pointer to the service routine. Here again, the flag bit (bit 0) is blanked when sending out the contents of VPTR, so that we do not get an illegal (odd) address on the interrupt procedure.

When the vector process completes the first character (or the GO bit is set by the PDP11), a new character is processed. This time the high byte of CBUF is used to obtain a table entry.

$$VPTR \leftarrow M(DTBAR \langle 15:9 \rangle + CBUF \langle 15:8 \rangle * 2)$$

The same procedure is used on succeeding characters. Since full word fetches are done into CBUF, a character string may not start on an odd byte boundary.

Characters are not restricted in size, length, drawing time etc. Characters are defined to be vector lists or PDP11 service routines, pointed to by eight bit codes. That is all the restriction the hardware enforces. In particular, no assumptions are made about vector format, intensity or scale initial conditions. Furthermore, a vector list that comprises a character must include any beam repositioning required. Generally, assumptions are made about any one character set (by the SOFTWARE, not the hardware) as to whether the

beam is on or off initially, which vector format is set up, whether repositioning is done after or before drawing a character etc.

Providing you have room, two or more complete character sets can be loaded into memory, and switching can be done between character sets by changing the contents of DTBAR (to point to the alternate character set's dispatch table). This switching of DTBAR must be done by the PDP11 (perhaps by a routine called as a result of an interrupt character, or an INTR instruction). With 256 possible character codes, sometimes two 128 character sets can be accommodated in one dispatch table.

5 Care and Feeding of Vectors

Vector processing logic is what graphics is all about. A great deal of effort was made in the design of the line drawer to obtain the maximum drawing power for the GDP. Properly set up, the line drawer is capable of drawing over 50,000 vectors in 1/60 of a second, without ever stopping the beam.

In the GDP, vectors are always relative, with the next vector starting where the previous one stopped. Each vector is comprised of two components, Delta X and Delta Y. Vectors are further affected by the current state of the INTENSITY LEVEL, UNBLANK, SCALE and FORMAT registers (which are all part of the STATE register). Vectors are drawn in a Cartesian plane with 12 bits of positioning per axis, of which only the middle 10 bit portions are normally visible (but see WRAP and SCALE discussions later on). The representation of DX and DY is strictly Two's Complement. The screen coordinates are expressed as integers, with 0,0 in the center of the screen. The limits of the PHYSICAL screen are + and - 511 (base 10) in both X and Y axes. The limits of the 12 bit VIRTUAL screen are + and - 2047. The present screen is adjusted to have 100 points per inch on a 10 inch square (the tube however, is specified at 68 lines per inch resolution). Because different kinds of pictures require various lengths of vectors, and because packing density affects the number of vectors that can be stored in the GDP's Double Port Memory, three formats of vectors are available.

Short vectors allow high density storage, as two vectors can be accommodated in one 16 bit word. However, this restricts the length of the vectors to 4 bits, including sign, limiting vector length (unscaled) to less than 1/10 inch. Short vectors are very useful in character descriptions and speech waveforms, and in other types of curve approximations.

At the other end of the stick, Long vectors allow full screen deflections at the cost of two 16 bit words per vector. When you need to be able to get across the entire screen, or cannot predict in advance the lengths of your vectors, long vectors is your kind of format. In between, Medium vectors provide an intermediate length/packing density factor. They are packed one per word, with an eight bit DX/DY. The maximum (unscaled) length is 1.25 inches, sufficient for a great variety of uses.

Thus we have Short, Medium, and Long formats of vectors. The current format in use is encoded in the FORMAT register. Code 0 is short, 1 is medium and 2 is long. Code 3 is undefined (if you try it, you get medium, but it may not always be so). The format can be changed via Control Words in the middle of the data. Diagrams of the formats are in the Appendix. Note that Short vectors (two per word) always are processed low byte first. Long vectors are limited to 13 bits (including sign), but the should be sign extended to 16 bits. Generally, sign extension does not need to be explicitly done, the normal representation of numbers in the PDP11 is exactly like that

of the GDP, so that the only care needed is to assure that the maximum length of either component does not exceed that permitted by the desired format (4 bits on Short vectors, 8 bits on Medium, and 13 on Long, all including sign).

The vector processing logic has two stages. Vectors are obtained from memory by fetching on the VPTR register, into the Vector Buffer, VBUF.

VBUF←M(VPTR)

VPTR←VPTR+2

In the case of Long vectors, another fetch must be made to get the other component.

VBUF←M(VPTR)

VPTR←VPTR+2

For your information, there is another register, which you can't get at, which receives the contents of VBUF before the second fetch is made. When both vector components are available, the line drawer extracts the DX and DY from the VBUF according to the current format, and draws the vector. Delta Y always has a lower UNIBUS address than Delta X. This occurs because DX in medium and short vectors is in the rightmost byte, which has a higher UNIBUS address.

We now turn our attention to the Intensity of the vectors. The brightness of the vectors is controlled by two registers, INTENSITY LEVEL, and UNBLANK. UNBLANK provides gross control of visible vs invisible lines. It is a single bit. If UNBLANK is on (a 1) vectors are drawn visibly, if it is off vectors are invisible. The UNBLANK bit is modified by Control Words, which we will examine in the next section. When vectors are visible, the brightness is further controlled by the current INTENSITY LEVEL. This is a 4 bit (16 value) register. All sixteen grey levels should be distinguishable from each other. Like UNBLANK, INTENSITY LEVEL can be changed via Control Words.

In addition, there is a bit in the GCSR called WRAP. This is used to control wrap-around. If the WRAP bit is set, vectors which are drawn off one edge of the 10 bit physical area will appear on the opposite edge (wrapped-around), and continue drawing, visibly. If WRAP is off, (the default case on existing software) vectors drawn off the edge will disappear. So long as the virtual position does not exceed the 12 bit boundary, vectors drawn off screen will be calculated correctly, and accurate positioning maintained. Note that drawing offscreen does not abort drawing, it merely turns off the intensity. This feature can be used in a clever way. If your entire picture is made up of only relative components (see SETPOINTS, they aren't relative), you can cause Graphics to display any arbitrary 10 bit window of the virtual 12 bit picture. The window need not be the center section. To do this, you initialize the beam position before drawing such that the relative movements put the desired window in the visible section of the screen. Please do not confuse this scissoring of the vectors with true windowing and remember

5 Care and Feeding of Vectors

that it takes just as much time to draw an offscreen vector as one which is entirely on screen.

The current position of the beam is always kept in the XR and YR registers. These registers are accessible by the UNIBUS, and can also be changed via Control Words. The registers have only 12 bits of significance, but are sign extended to appear like 16 bit registers.

The lengths of vectors can be modified by the current SCALE value. This register, 4 bits long, is encoded according to the table in the Appendix. Lengths of vector components are multiplied times the scale factor. Quasi-logarithmic scaling is provided to increase or decrease picture segments in approximately 207 increments. Scaling can magnify a picture up to 3 3/4 times normal size, and as small as 1/4 of normal size. In case you were wondering, SCALE can be changed by Control Words. A scale value of 10 (octal) gives you a normal size picture. Scaling actually modifies the DX and DY of vectors (it does not alter memory however), thus a two times normal size scaling factor applied to a 2,2 vector will result in a 4,4 vector from the current X and Y beam positions. If, for example, the beam was at 3,1 before drawing the vector, the next vector will start at 7,5.

There is one more bit which affects vectors, the CLKDIV bit. This is part of GCSR. The basic clock frequency of the line drawer can be divided by two with the CLKDIV bit. Slower clock speed (CLKDIV=1) gets you brighter, cleaner lines. Fast clock gives you dimmer, grainier lines, but gives you twice as many flicker free inches of vectors. Use slow mode if you can get away with it, high speed if your picture begins to flicker.

When processing vectors, the GDP will do a full vector look-ahead if possible. This means that a new vector can be fetched into VEIUF as soon as the line drawer begins to draw the previous one. In fact, it is possible to start drawing a very long vector from vector processing in character mode, discover that the vector list is finished, get back to character processing, find out that you have finished the character list, go back to instruction processing, fetch a new instruction, get back into character processing, pick up a character, look it up in the table, reactivate vector processing and fetch the first vector of the first character all before the previous vector is finished drawing. Two things are noteworthy. Firstly, the line drawer will not stop drawing when it finishes a vector if another one is waiting in VBUF. Secondly, if you get an interrupt from the GDP, you cannot assume that the vector drawer is finished. The DONE bit of the GCSR will not come up until the vector is finished, so that you must examine it if you need to load or read XR or YR. The intensity modification registers can be changed however, because their old values are saved by the line drawer when it starts drawing a vector. SETPOINTS (see Control Words) will not be processed until the line is finished drawing. This is another method of assuring that the XR and YR registers are valid from within an interrupt service routine.

6 Making CONTROL WORDS Work for You

Throughout this discussion we have made reference to entities called CONTROL WORDS. As we have intimated, they can alter the mode and state of the GDP from within graphics data. Control Words are unique bit patterns that are recognized in any context. They can appear in Instruction, Character and Vector lists. With a few exceptions, they cause identical action in any context. A diagram of the CONTROL WORD is in the Appendix, as is a table of the available options.

A word whose upper byte is 1 0 0 0 0 0 0 0 is always a CONTROL WORD when it is encountered in an instruction, character or vector fetch. These are known as Full Word Control Words. In the case of a Control word found in Long Vector data, the Control Word will be recognized in either the DY or the DX. If it is found as the DX, the DY is discarded. The lower eight bits of the Control Word are used to determine what action is requested. The eight bit field is divided into two smaller fields, a four bit OPCODE and a four bit OPERAND. The OPCODE determines the class of Control Word desired. We will consider each class separately.

Opcode 0 is the TERMINATE class of Control Words. They are used to delimit the end of instruction, character and vector lists. The opcode is ignored by the hardware. It can be used by the software if desired, to flag certain kinds of data. The TERMINATE Control Word always causes an end of the current process and a reactivation of the next higher process. There are four cases to be considered. A TERMINATE in an Instruction list causes Graphics to halt. No further processing of data will be done. Note that no indication is given to the PDP11 that the GDP has halted, except that the DONE bit will be set (providing that the last vector is finished being drawn). A TERMINATE in a character list will cause a new Instruction to be fetched and processed. A TERMINATE in a Vector list will cause a new Character to be processed if CMODE is on (that is, if the Vector process was activated from the character process). If CMODE is off, a new Instruction is fetched. This last case is a result of an XQT Instruction with CMODE off. In general, objects of XQT instructions, (that is, that which is pointed to by the operand of the XQT) and character description lists are terminated by TERMINATE Control Words.

The second class of Control Words is INTERRUPT. These cause interrupts to be made to the PDP11. In this case, a service routine address is not known by the GDP, and a normal vectored interrupt is generated (at BR4, vector 104). Again, the operand of the Interrupt Control Word is not used by hardware but is often utilized by the service routine to determine the action required. As with INTERRUPT @VPTR, the GDP is put into a pause state by an Interrupt Control Word, and must be continued by turning on the GO bit in the GCSR.

Opcode three of Control Words is the LOAD CMODE. It is used to alter

the state of the CMODE bit. The low bit of the operand is loaded into CMODE. The remaining three bits are ignored. There are two special cases of LOAD CMODE. If CMODE was a 1, and it is loaded with a 0 while character processing is activated, the following additional register transfer is done:

VPTR←CPTF

Then, Character processing is disabled and vector processing is activated. Similarly, if CMODE is off, and Vector processing is activated, if a LOAD CMODE Control Word is processed with an operand of 1, the transfer:

CPTR←VPTR

is made, and Character processing is initiated. This is used to switch from Character processing to Vector processing and vice versa. A LOAD CMODE Control Word, when encountered in an Instruction fetch, just changes the state of the CMODE bit. Thus it changes the assumptions made about the object of the next XQT instruction. LOAD CMODE with an operand of 0 when CMODE is already a zero is a No-op, as is LOAD CMODE 1 with CMODE already set.

The fourth opcode is the LOAD FORMAT class. The lower two operand bits are placed in the FORMAT register. The upper two operand bits are ignored. A change in FORMAT only affects vectors fetched after the LOAD FORMAT Control Word, it can never affect vectors already fetched.

The next four classes of opcodes affect the INTENSITY LEVEL and SCALE registers. For each register there are two types of Control Words, ABSOLUTE and RELATIVE. The absolute types load the operand of the opcode into the desired register. The relative varieties add the operand to the current value of the register. This add is a two's complement addition, with a four bit (including sign) number. Thus, you can alter the contents of SCALE or INTENSITY LEVEL + and - 7. Overflow from the addition is ignored, the value "wraps around". Thus LOAD INTENSITY RELATIVE 4, when the INTENSITY LEVEL is at 5 will result in a final value of 9, but if the initial value was 14, the result will be 0. We have LOAD INTENSITY ABSOLUTE, where the operand is moved into INTENSITY LEVEL, and LOAD INTENSITY RELATIVE, where we add the operand to the current INTENSITY LEVEL. Similarly, LOAD SCALE ABSOLUTE is used to force the SCALE to a particular value, and LOAD SCALE RELATIVE is used to make pictures bigger or smaller, without having to know what the previous SCALE value was. The relative forms of these Control Words are especially useful in subroutines, where the initial conditions are not known, and it is desired to not alter the state permanently. By including LOAD SCALE RELATIVE 1 before the first vector, and LOAD SCALE RELATIVE -1 after the last vector, a "pure" subroutine can be created which makes part of the picture 20% bigger.

The next two classes of opcodes are similar in that the operands are encoded to provide up to 16 different Control Word functions in one class. They are the SPECIAL CLASS 1, and SPECIAL CLASS 2 groups. We will

consider class 1 first. In these Control Words, the upper operand bit is ignored, giving us 8 available Control Words. The first one is another TERMINATE. The usefulness of this control word will be seen in the section about HALFWORD Control Words. The rest of the SPCL1 Control Words affect the UNBLANK bit of the STATE register, and thus affect visible vs invisible lines. ION and IOFF directly turn the UNBLANK bit on and off. The ICOM Control Word complements the current value of UNBLANK. The IOF1 (Intensity Off for 1 vector) Control Word blanks the next vector only. This is used to reposition the beam, prior to drawing a new figure, or piece of a figure. Since that is the usual use for invisible vectors, IOF1 will save you 1 word each time it is used over the IOFF-vector-ION combination that would otherwise be necessary. Sometimes however, the format you are in will not allow you to reposition the beam with only 1 vector. Therefore the IOF2 and IOF3 Control Words are available. These will usually suffice to move the beam to the desired point.

The last variety of SPCL1 Control Words is the IALT. This one alternately blanks and unblanks the beam. The first vector drawn after a IALT will be invisible, the next visible, the third invisible etc. The alternation will continue until the next Control Word is encountered. In fact, any of the IOFF1, IOFF2, IOFF3 or IALT Control Words will be canceled by any Control Word, including TERMINATE. This means that the effect of an IOFF 1, 2, 3, or IALT cannot carry over between characters or XQT lists.

The SPCL2 Control Words are the SET class. They cause loading of the XR, YR or STATE registers. The data to load the registers immediately follows the Control Word in memory. There are four currently implemented Set Control Words. SET X loads the next word into the XR register. SET Y is similar except that the YR register is affected. SET XY loads both XR and YR. In this case the X value is first, followed by the Y (sorry about that). A SETXY is sometimes referred to as a SETPOINT. The SETPOINT is an absolute placement of the beam. The current SCALE value does not affect the data. values between +2047 and -2047 are legal. The last flavor of SET commands is the SET STATE Control Word. This loads the STATE register with the next word. Since SCALE, INTENSITY LEVEL, CMODE, UNBLANK and FORMAT are all part of STATE, a completely new environment can be established in two words with the SET STATE Control Word. The operands between 4 and 7 of SET class are No-ops. The upper bit of the operand is currently ignored.

All other classes of Full Word Control Words are currently No-ops. We should note that Full Word Control Words always start on even byte boundaries, so that a null vector or character must sometimes be used to fill up a wasted high byte.

The problem of bit wastage in Short vectors was deemed important in the design, as was the realization that using 16 bits to turn off the beam is

poor efficiency. For this reason, HALF WORD Control Words have been implemented. These Control Words are only available in Short vector lists. They are recognized by a 1 0 0 0 1 in bits 7:3 of any byte in a Short vector file. The remaining three bits are used as an operand. The operand bits are interpreted exactly as the operand of SPCL1 Control Words. Thus we can alter the UNBLANK bit, and TERMINATE a short vector list, at the cost of only 8 bits.

Before we leave Control Words, we might explain how the choice of bit patterns was made and how it affects the data. In instructions, the operand of XQT or JMS instructions is defined to be restricted to the range 400-37776 octal. This eliminates confusion between an instruction with the high opcode bit=1 and an operand below 400, and a Control Word. In character processing, the character code 200 (octal) may not appear in the high byte of a word. This also eliminates confusion between Control Words and character data. In vector lists, we note that Long vectors may not have values lower than -10000 (octal). Thus the Control Word pattern (which is a very large negative number) can not occur. The lengths of the components of medium vectors are defined to be in range +177 to -177, so that having a DX of 400 (which is -200 octal) is illegal. Note that there is no positive complement for this number in an 8 bit space. In short vectors, the restriction is that DX and DY must be between -7 and +7. The octal 200 in the upper byte which signifies a Full Word Control Word is a vector -10,0, which is illegal. Likewise, the Half Word Control Words appear as vectors with DX's of -10, and negative DY's, and therefore illegal. In all cases of possible confusion of a Control Word with another data item, the existence of the Control Word is assumed.

The Double Port Memory is a great boon to Graphics because it allows the GDP to get almost the minimum 650 ns. cycle time of the memory. The only problem is that if the PDP11 requests a word in the Double Port Memory, the GDP must wait until the processor is finished before it can get its cycle. To avoid this problem, the programmer must attempt to minimize the number of cycles the PDP11 requests on the memory. Some of the Graphics Terminals will have more than 8K of memory. If this is available, the more stuff that is kept in the PDP11 stand alone memory, the less accesses it will make on the Double Port. In any case, attempt to minimize the number of cycles the PDP11 does in the Two Port Memory.

There are a large number of No-ops in the Control Word system. These are mainly reserved for future expansion. Opcode 17 will always be kept as a No-op however, so that programs can use them as desired. Similarly, the "ignored" bits in the LOAD FORMAT, LOAD CMODE, SPCL1, and SPCL2 may be used some day, so don't use them for anything else.

We have so far ignored the very basic instructions to programmers of how to initialize and start up the GDP. There really are very few things to do. First of all, a suitable Dispatch Table should be set up, and the Base Address stored in the DTBAR. The initial STATE should be established, unless one of the first "instructions" includes a SET STATE Control Word. The INTERRUPT ENABLE bit in the CSR should be set, and a suitable status loaded into the GIS register. Then, move the address of the first instruction into the GPC, and GDP will take over from there. Don't forget to set up the PDP11's stack register if you expect interrupts. The usual method for entering a list is an INTR to a service routine which causes a wait for a refresh clock tick. It is preferable to run the GDP at a 60 hz refresh rate. Slower refresh cycles will begin to how flicker eventually. If the refresh rate is not a integral multiple of the power line frequency, hum can sometimes be noticed. When your display list is too big, switch to high clock speed (CLKDIV=0), and run as fast as you can.

Although it was designed mainly for maintainance reasons, there are two additional features of the GDP. If the CPTR register is loaded from the UNIBUS, the GDP will begin character processing (CMODE is forced to be a 1). This can be used to draw a single character string. Similarly, the VPTR register can be loaded via the UNIBUS, and a vector list will be displayed. TERMINATE Control Words in the character string or vector list in these cases will cause GDP to halt much like the effect of an Instruction list TERMINATE. This does not imply that the end of character description lists should not be TERMINATEs when starting via a UNIBUS load of CPTR. The TERMINATE here still causes a new character fetch.

The CLEAR bit of the GCSR is used to "Crash" the GDP. It is similar to a RESET instruction, but only affects the GDP. Extensive use of this to stop the

GDP is not recommended, because very strange things can happen in the process of stopping the GDP. This feature is designed for diagnostic purposes and halting a runaway display list. Writing a one into the CLEAR bit will turn off the INTERRUPT ENABLE bit. DO NOT READ OR WRITE ANY GRAPHICS REGISTERS WHILE GRAPHICS IS RUNNING except to write the CLEAR bit. You will be DOOMED. Wait for an interrupt before addressing any of the registers.

Approximate timing information about the various processes in the GDP will be given next. These are approximate times only, and assume no memory conflicts by the PDP11. An instruction takes about 700 ns to fetch and execute, except for JMS which takes another 650 ns for another memory cycle for the return address deposit. Character overhead is about 1 microsecond, including the character fetch and the table access. Control Words take 700 ns each, except for SET X, SET Y, and SET STATE, which need 1.4 usec and SET XY which takes 2.1 usec. Vector drawing times are estimated by determining the next power of two higher than the larger of (DX,DY), and multiplying that by 30 ns. (60 ns for slow clock speed). If the number is less than 650 ns, then the access time of the memory overrides the draw time (this is not strictly true for short vectors, where there are two vectors accessed per 650 ns, nor for long vectors, where 1.4 usec is necessary to fetch the vector components).

APPENDIX A

A list of Registers and their UNIBUS address assignments.

ADDRESS REGISTER USE

165100	CSR	Control and Status
165102	GPC	Graphics Program Counter. Points to next instruction
165104	VPTR	Vector Pointer. Points to next vector, also holds interrupt PC
165106	GIS	Graphics Interrupt Status. Holds PS for INTR VPTR interrupts
165110	STATE	Holds SCALE, INTENSITY LEVEL, CMODE, UNBLANK, and FORMAT
165112	DTBAR	Dispatch Table Base Address Register for character description lists
165114	CBUF	Character Buffer. Holds two 8 bit characters to be drawn
165116	CPTR	Character Pointer. Points to next character to be interpreted.
165120	VBUF	Vector Buffer. Holds vector to be drawn
165122	XR	X Register. Holds present X coordinate of beam
165124	YR	Y Register. Holds present Y coordinate of beam

Bits in CSR are:

15	8	7	6	3	2	1	0
	IDONE	IE	I	IWRAP	CLKDV	GO	ICLEAR

BIT	NAME	USE
0	CLEAR	Reset graphics to turn on conditions
1	GO	Continue from interrupts
2	CLKDV	If a 1, run line drawer at full speed, if a 0, run at half speed. Half speed is brighter.
3	WRAP	If a 1, don't wrap around at 10 bit boundary
6	IE	Interrupt Enable for all interrupts
7	DONE	Graphics is not doing anything useful

Bits in STATE are:

15	12	11	8	7	4	1	0
	SCALE		INT LVL	CMODE	UNBLANK	FMT	

BIT	NAME	USE
0-1	FMT	Format of vectors 0=short, 1=medium, 2=long, 3=unimplemented
4	UNBLANK	If a 1, vectors drawn are visible, if 0, invisible
7	CMODE	If a 1, Characters are/will be drawn, If a 0, Vectors are/will be drawn.
11-8	INT LVL	Intensity Level, 15 is maximum, 0 is minimum
15-12	SCALE	Scale for vectors

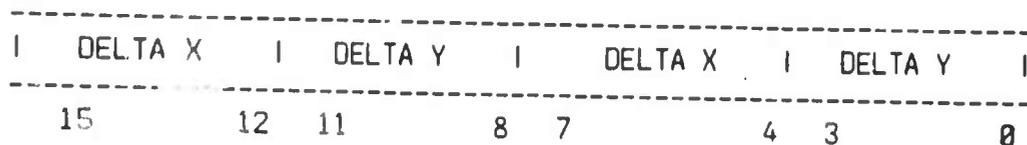
SCALE bits are encoded as follows:

OCT	DEC	
17	15	= 3 1/2 Times Normal Size
16	14	= 3 "
15	13	= 2 1/2 "
14	12	= 2 "
13	11	= 1 3/4 "
12	10	= 1 1/2 "
11	9	= 1 1/4 "
10	8	= NORMAL SIZE (1 X)
7	7	= 7/8 Times Normal Size
6	6	= 3/4 "
5	5	= 5/8 "
4	4	= 1/2 "
3	3	= 7/16 "
2	2	= 3/8 "
1	1	= 5/16 "
0	0	= 1/4 "

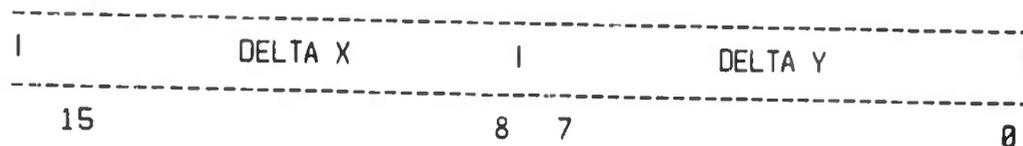
APPENDIX B
 VECTOR, CHARACTER and INSTRUCTION FORMATS

VECTORS

SHORT



MEDIUM

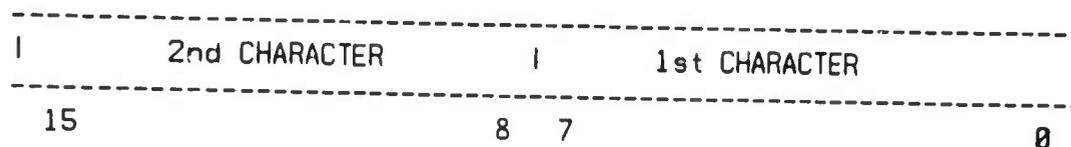


LONG



Delta X and Delta Y are in twos complement format.
 Long Vectors are limited to 14 bits (13 bits of magnitude plus sign),
 but should be sign extended to 16 bits.

CHARACTER FORMAT IS:



DISPATCH TABLE BASE ADDRESS REGISTER FORMAT:



INSTRUCTION FORMAT IS:

IOPC1I		OPERAND ADDRESS	IOPC2I	
15	14		1	0

FETCH: VPTR ← M(GPC)
 GPC ← GPC + 2

INSTRUCTION ARE:

OPC	OPC		
0	1		
0	0	GJMP	PC ← VPTR
0	1	GJMS	M(VPTR) ← PC PC ← VPTR + 2
1	0	GINTR	INTERRUPT @VPTR
1	1	GXQT	EXECUTE

Execute a data file at VPTR.
 Execution is controlled by the CMODE bit.
 when CMODE is a 1, the data file
 is assumed to be a list of characters
 otherwise, it is a list of vectors.
 if CMODE = 1, than an additional CPTR←VPTR
 transfer is done before execution.

APPENDIX C
 CONTROL WORD FORMATS AND ACTIONS

Full Word Control Words are of the following form:



Where the opcodes cause the following actions

OPCODE	NAME	ACTION																								
0	TERM	Terminate, go up 1 level. If processing vectors, then end vector list and fetch next character (if CMODE=1) or next instruction (if CMODE = 0). If Processing Characters, TERM causes new instruction fetch. TERM while processing instructions sets done and stops all action.																								
1	INTR	Interrupt POP11 at vector 104, BR4. Ignores operand field.																								
2	LCMD	Load CMODE CMODE+OPERAND<0>																								
3	LFMT	Load FORMAT FORMAT+OPERAND<1:0>																								
4	LILA	Load Intensity Level Absolute INT LVL←OPERAND<3:0>																								
5	LILR	Load Intensity Level Relative INT LVL←INT LVL+OPERAND<3:0> (operand is taken to be in twos complement representation)																								
6	LSCA	Load Scale Absolute SCALE←OPERAND<3:0>																								
7	LSCR	Load Scale Relative SCALE←SCALE+OPERAND<3:0> (Twos complement representation)																								
10	SPL1	Special Codes 1. Operand is interpreted as: <table border="0" style="margin-left: 20px;"> <tr> <td>0</td> <td>TERM1</td> <td>Terminate same as TERM</td> </tr> <tr> <td>1</td> <td>IDN</td> <td>Intensity On UNBLANK←1</td> </tr> <tr> <td>2</td> <td>IOFF</td> <td>Intensity Off UNBLANK←0</td> </tr> <tr> <td>3</td> <td>ICOM</td> <td>Intensity Complement UNBLANK←NOT UNBLANK</td> </tr> <tr> <td>4</td> <td>IDF1</td> <td>Intensity of for next vector only</td> </tr> <tr> <td>5</td> <td>IOF2</td> <td>Intensity off for next 2 vectors</td> </tr> <tr> <td>6</td> <td>IDF3</td> <td>Intensity off for next 3 vectors</td> </tr> <tr> <td>7</td> <td>IALT</td> <td>Intensity alternate on and off Continues until next control word. Starts as off.</td> </tr> </table>	0	TERM1	Terminate same as TERM	1	IDN	Intensity On UNBLANK←1	2	IOFF	Intensity Off UNBLANK←0	3	ICOM	Intensity Complement UNBLANK←NOT UNBLANK	4	IDF1	Intensity of for next vector only	5	IOF2	Intensity off for next 2 vectors	6	IDF3	Intensity off for next 3 vectors	7	IALT	Intensity alternate on and off Continues until next control word. Starts as off.
0	TERM1	Terminate same as TERM																								
1	IDN	Intensity On UNBLANK←1																								
2	IOFF	Intensity Off UNBLANK←0																								
3	ICOM	Intensity Complement UNBLANK←NOT UNBLANK																								
4	IDF1	Intensity of for next vector only																								
5	IOF2	Intensity off for next 2 vectors																								
6	IDF3	Intensity off for next 3 vectors																								
7	IALT	Intensity alternate on and off Continues until next control word. Starts as off.																								

OPCODE	NAME	ACTION
11	SPL2	Special Codes 2. Operand is interpreted as:
		0 SETX Set X XR+M(POINT); NEXT POINT←POINT+2 POINT is VPTR,CPTR or GPC, whichever is applicable
		1 SETY Set YR Same as above except YR←M(POINT)
		2 STXY Set X and Y XR←M(POINT);POINT←POINT+2 YR←M(POINT);POINT←POINT+2
		3 SETS Set State STATE←M(POINT);POINT←POINT+2
		4-7 ---- Unimplemented. Currently acts as NOPs.
12-16		Unimplemented. Currently act as NOPs.
17	NOP	No Operation

NOTE: High order operand bit of SPCL1 and SPCL2 codes is ignored.

Halfword Control Words are in the following format

```

-----
| 1 0 0 0 1 | <3 bit operand> |
-----

```

The operand bits are interpreted the same as SPCL1 operands.
 These Control Words can only occur in Short Vector (FMT=0) lists.

ASLI	2
Asynchronous Line Interface	2
Base Address	9
CBUF	3, 5, 9
Character Buffer	3, 5, 9
Character Fetch	9
Character Interrupt	9
Character Mode	3, 4, 7, 14
Character Pointer	3, 5, 9
Character Processing	3, 9, 17
Character Sets	10
CLEAR	5, 18
CLKDIV	5, 13
Clock	2
CMODE	5, 7, 14
Computer	2
Continue from Interrupt	7
Control Words	3, 4, 14
CPTR	3, 5, 9
Crashing Graphics	18
Dispatch Table	9
Dispatch Table Base Address	3
Dispatch Table Base Address Register	5
Display List	2
Display Tube	2
Double Port Memory	6, 18
DTBAR	3, 5, 9
DX	11
DY	11
Example of Instructions	8
Execute Instruction	3, 7
Format	15
FORMAT	5, 15
Format	4
Formats	11
Full Word Control Words	14
GCSR	5
GDP	2
GIS	5, 7
GO	5, 7, 14

GPC	3, 5, 6
Graphic Display Processor	2
Graphic Program Counter	3
Graphics Control and Status Register	5
Graphics Interrupt Status	5, 7
Graphics Program Counter	5
Half Word Control Word	16
HALT	16
ICOM	16
Initialization	18
Instruction Buffer	6
Instruction Fetch	6
Instruction Processing	3
Instruction Processing	6
Intensity	4, 12
INTENSITY LEVEL	5, 12, 15
Interrupt	7, 14
Interrupt Dismissal	7
Interrupt Enable	7
Interrupt Instruction	3, 7
Interrupt Service Routine	3
Interrupt Status	7
INTR	3, 7
INTRRUPT @VPTR	9
INTRRUPT VPTR	7
Invisible Vectors	12
IOF1	16
IOF2	16
IOF3	16
IOFF	16
ION	16
JMP	3, 6
JMS	3, 6, 17
Jump Instruction	3, 6
Jump To Subroutine Instruction	6
Jump to Subroutine Instruction	3
Keyboard	2
Line Drawer	3, 11
Line Drawing	3
LOAD CMODE	14
LOAD FORMAT	15
LOAD INTENSITY	15

LOAD SCALE	15
Lookahead	13
Memory	2
No-op	16, 18
Opcode	3, 6, 14
Operand	3, 6, 14
Pause	7
PDP11	2
Physical Screen	11
Read Only Memory	2
Registers	5
Resolution	11
Return Address	6
ROM	2
SCALE	5, 13, 15
Scale	4
SET STATE	16
Setpoint	16
Setpoints	4
SETX	16
SETXY	16
SETY	16
Sign Extension	5, 11
SPCL1	15
SPCL2	15
Starting Graphics	18
STATE	5
State Register	5
Subroutines	6
Terminate	14, 16
Timing	19
UNBLANK	5, 12, 15
VBUF	4, 5
Vector Buffer	4, 5
Vector Fetch	12
Vector Format	11
Vector Intensity	12
Vector Pointer	4, 5

Vector Processing	4, 11, 17
Virtual Screen	11
Visible Vectors	12
VPTR	4, 5, 6
Windowing	12
WRAP	5, 12
Wraparound	12
X Position	13
X Register	5
XQT	3, 7, 14, 17
XR	4, 5, 13
Y Position	13
Y Register	5
YR	4, 5, 13