A GRAPHICS-ORIENTED OPERATING SYSTEM

A GRAPHICS-ORIENTED OPERATING SYSTEM FOR A SMALL COMPUTER

Kenneth Craig Campbell, B. Eng.

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Master of Engineering.

Department of Electrical Engineering
McGill University
Montreal, Quebec
August, 1974

ABSTRACT

This thesis describes an implementation of Bell Telephone Laboratories' BELLGRAPH operating system on McGill's McGraph facilities. The special considerations and problems encountered in modifying an advanced system designed to drive a core-refreshed display unit such that a disc-refreshed display unit could be used are described. The resulting changes in operating concepts are outlined, and guidelines are presented for possible future development.

RESUME

Cette thèse présente une réalization sur le système McGraph à l'Université McGill du système d'opération BELL GRAPH qu'ont développé les laboratoires du Bell Téléphone. Les considérations spéciales et les problèmes rencontrés en transformant un système advancé conçu pour un système graphique régénération par la mémoire centrale pour un système régénération par disque sont présentés. Des changements résultant dans les idées d'opération sont esquissés, et des lignes de guide sont présentées, peut-être pour un dévelopment futur.

ACKNOWLEDGEMENTS

I wish to express my sincere thanks to the many people who contributed to the success of this thesis.

First, the patience and guidance of Dr. A. Malowany, my thesis advisor, was much appreciated.

Thanks are due to Juhan Leemet, who not only wrote the PDP-8 software and software and maintained the graphics hardware, but also supplied me with many useful ideas.

I am greatly indebted to Bell Telephone Laboratories for generously making the BELLGRAPH software available.

I would also like to thank Mrs. Diane Morley who typed this thesis.

And finally, I'd like to thank a Friend who kept up my spirits in the darkest days of thesis-writing.

This research was supported by the National Research Council.

TABLE OF CONTENTS

SECTION A INTRODUCTION AND BACKGROUND

CHAPTER I INTRODUCTION

Computer graphics has proven to be one of the most intriguing research topics in computer science in recent years. Since its dramatic popularization by Sutherland [S. 3] in the early 1960's many successful applications of computer graphics have been reported. These include forays into such varied fields as computer-aided design [F. 3, G. 2], weather forecasting [G. 1], and computer-aided instruction [M. 3]. Over the course of the years, indeed, many graphics-oriented languages have been devised-GRIN2 [C. 1], Sketchpad [S. 3], Grapple [W. 2], PICADE [T. 3], and PL/OT71 [U. 1] to name but a few — along with numerous packages of graphics subroutines written in existing algorithmic languages such as FORTRAN [D. 2, O. 1, T. 1]. Graphics peripherals are becoming significantly cheaper and more readily available. Indeed, the day is within sight when every medium-size computer installation will have one or more graphics terminals as part of its normal equipment. Existing operating systems must change in order to meet these popular demands. This thesis documents the implementation of a graphics-oriented operating system within the McGraph [M. 1, M. 2] environment, and makes suggestions concerning its further development.

1.1 Interactive Graphics

Why have so many computer installations invested heavily in graphics peripherals, especially interactive graphics terminals? No doubt each installation has its own reasons for doing so, but two ideas underlie all of them. Every exponent of computer graphics tacitly believes in the power of symbolic thought, the ability to, as Herzog [(H. 1)] puts it, "convey ideas in terms of forms". Similarly, every disciple of interactive computing believes that a user can do more useful work if he is allowed to interact with the computer, and influence its course of action. These are the twin bases upon which interactive graphics rests. Together, they may be said to symbolize an attempt to upgrade the man/machine interface, and to make the computer more responsive to the user's demands.

On the other hand, interactive graphics terminals have consistently proven to be a difficult tool to harness successfully. While it has always been easy to visualize applications of graphics terminals, Penney points out that it has been uniformly "difficult to generate useful software of fairly general applicability". [(P. 1)] Foley extends this by mentioning that it is difficult to write high-quality graphics programs at all, and by indicating that "... the clarity and vividness of computer graphic communication is not an automatic consequence of the mere use of drawings" [(F. 2)]. It has long been recognized that interactive programs on a time-sharing system must be carefully written, with human factors taken into account, if they are to be truly effective [(S. 1, W. 1)]. The same is now being said of interactive graphics programs; [(F. 2)] although, of course, all the difficulties have been magnified. Visual processes are inherently complex, and an interactive graphics terminal may well be one of the most complicated peripherals to be interfaced to any computer. If, as Penney contends, graphics support software is generally inadequate (P. 1), the conclusion is inescapable: effective graphics programming can be a real chore. The result is that until now "the most successful applications of computer graphics have been the simple ones". (P. 1)

Efforts are continuing to provide graphics terminal users with an adequate level of software support. The operating system described in this thesis was brought into existence at BTL in 1967, and reported on in 1968 (C. 1). Since then it has been continually updated (M. 4), and now transplanted into a new environment. Although many similar packages have been described in the literature (V. 1, N. 1, M. 4), no method of generating general-purpose software has been agreed upon. As of now it is still a matter for research.

1.2 Experiences with Graphics Systems

It might be instructive to superficially examine a few existing systems to gain a feel for what is required in a graphics-oriented operating system. First, one must be clear on what a "graphics-oriented" system really is. To many, it is simply a system capable of driving some form of graphics terminal. If this definition is accepted as forming a lower bound, many commercially-available systems immediately become graphics systems. XEROX (D. 2), Tektronix (MAN-11) and other terminal manufacturers offer software support for their equipment, in the form of FORTRAN subroutine packages. These permit the user to undertake a certain amount of interactive graphics programming with (apparently) minimum development and training costs. And in many applications, such an approach is entirely valid, as witness the Tektronix sales literature (MAN-11). In some cases, a more advanced system can be provided (at added development cost) by implementing a high-order graphics language on an existing commercial operating system (0.1, P. 2). This has some advantages, such as programmer convenience, but does not address itself to many of the fundamental problems computer graphics.

Consider for a moment the other extreme-the military environment. Highly speicalized software must be developed in a customized environment. In tactical or strategic operations high data rates are the norm. These lead naturally to computers with multiple bus structures, multiple general-purpose registers, and so on. The displays must present accurate, real-time information under extremely adverse conditions. Few, if any, commercial systems could meet military performance standards. On the other hand, high order languages are nice, but quite dispensable in military systems, where efficient systems are everything.

Commercial graphics systems operate in a far more favourable environment, but many problems are similar. Useful graphics programs, for instance, are typically large (P. 1). Data rates are frequently high, partially because graphics displays require a great deal of refresh activity, and partially because the display is often used to control a concurrently-running process. Few commercial systems, traditionally oriented towards data-processing or computational tasks can cope with these requirements at least not in real time.

The result is that a number of graphics oriented systems — in the sense that this term will be used in the balance of the thesis — have been developed to meet these requirements. Many of these, especially the nonmilitary ones, provide the capability of programming in a high-order graphics language. Representative of these systems is GRAPPLE, developed by Bell-Northern Research, and BELLGRAPH, developed by Bell Telephone Labs. A few such systems have even been offered by systems houses such as ADAGE (Y. 1). These systems support true graphics capabilities, rather than treating a graphics terminal much like a fast teletype.

The remainder of the thesis will be spent discussing one of these systems, the McGRAPH system. This is actually an implementation on the McGraph facilities, of the BELLGRAPH system. This system supports programs written in the high-order graphics language GRIN2. Further, it provides rapid memory management, allowing large applications programs to be run interactively in real time. As such, it is a rather interesting system.

1.3 Graphics Peripheral Devices

A staggering number of graphics peripherals have been placed on the market within recent years, giving a system designer unprecedented flexibility in hardware selection. A short review of the more prominent types of peripherals is included here to give the reader an insight into the various tasks a graphics-oriented operating system must fulfil.

The slow, hardcopy device such as microfilm printers and flat-bed plotters are usually run in an off-line mode. The operating system is really only responsible for formatting magnetic tape used to drive these devices, a relatively simple task.

Consider next the modest, CRT-based alphanumeric terminals. Normally these allow only a limited degree of interaction with the user. Alphanumeric characters can be displayed, sometimes only in fixed positions on the screen, though frequently in programmable ones and occassionally in various fonts. These terminals are usually based on direct-view, storage CRT's (TEKTRONIX) or upon cheap, video-scan TV monitor scopes (DATAMEDIA 1500). Some of the more advanced of these terminals (TEK - 4002) are equipped with vector-generators, enabling limited, relatively low-cost graphics.

High-performance, "intelligent" terminals permit more flexible graphic operation. These generally allow extensive operator interaction through the use of such devices as the light pen, joy stick, mouse, tracking ball, RAND tablet, and so on. To permit real-time manipulations and selections these units are usually based on refreshed CRT's. The refresh can come from mass storage (McGraph) or from core, on the DMA (GRAPHIC2). These terminals generally involve caligraphic display processors: the display is "painted" on the screen by the motion of the beam. Raster-scan units are very rare, due to the storage and elaborate controllers required.

Several types of "flat-screen" displays are being developed to eliminate some of the bulk of the CRT displays. The most promising of these, at the moment, are the plasma displays. Indeed, there is at least one commercial plasma display unit, the DIGIVUE. These have many potentially attractive characteristics (size, rear projection of film, selective erasure, and direct electrical read-out of display points), but are relatively slow, with cumbersome electronics [M. 3]. At best they require a good deal more development before they can seriously challenge CRT displays.

Interaction devices are also multiplying. Besides the usual devices already mentioned (light pen, etc.), work is proceeding on touch-sensitive membranes (LEK 114 CRT), and current-sensing panels [G. 4]. All this should cause the system programmer to seriously think about future developments before committing himself to a software program. Any operating system must be flexible enough to support many of these devices, and, hopefully, some not yet developed.

1.4 Development of Interactive Graphics at McGill

Interest in interactive computer graphics began at McGill in the fall of 1968 when it was decided to develop a graphics facility within the Department of Electrical Engineering. At that time the Department possessed a PDP-8 interfaced to an FDP-16 Data Disc and an IBM 360/75. A survey of commercially available systems (IBM, ADAGE, DEC) showed units with interactive graphics. capabilities to be prohibitively expensive, despite the fact that they were supported by varying amounts of software. In an attempt to reduce the costs of computer graphics it was decided to tailor a system to complement the existing facilities. A special purpose disc-refreshed unit was designed, constructed and debugged within the department (F. 1) and placed in service by the spring of 1971.

During the course of this project a PDP-15 was acquired and interfaced to the PDP-8. The initial applications programs for the graphic system (T. 1, N. 4) were written in FORTRAN on the PDP-15, with only a primitive set of handlers residing in the PDP-8. This proved unacceptable, and existing graphics systems were re-examined for ideas on improved software support

Late in 1971 Bell Telephone Labs released a copy of their Bellgraph Operating System (C. 1), supporting the GRIN2 language, to McGill. In 1972 work was begun to modify the system to function on the McGraph facilities. Finally, in early 1974 the first application program in GRIN2 was written at McGill, and run under the guidance of the McGraph operating system.

CHAPTER II INTRODUCTION TO GRIN2

2.1 The Graphic Data Structure

Display units, and associated graphical devices, are found to differ drastically at the hardware level from manufacturer to manufacturer, and to use different command formats to generate pictures. Accordingly, when one operating system must support several different types of display units, it is usually found necessary to represent graphical information in a device independent form. Immediately before the information is to be transmitted to a specific device, a set of programs is invoked to translate the data from standard form into the command set used by the device. The standard method of describing graphical information in the McGraph system is known as the Graphic Data Structure. This structure is as described by Christensen [C.1, MAN-1] in his outline of the GRIN2 language, and will be reviewed here for the sake of completeness.

The graphic data structure consists of the four types of blocks shown in Fig. 2-1: node blocks, branch blocks, leaf blocks and nondisplay data blocks. Each of these blocks consists of a contiguous set of memory locations with a distinctive header word. Node and branch blocks are of fixed length; the others are of indefinite length.

In McGraph, a picture can be represented by a directed graph with no closed loops. In this formalism each node represents a particular sub-picture, and each branch represents a specific instance of the node it points at. Some terminal nodes, called leaves, have special significance; only these blocks can contain display instructions. Nodes and branches, on the other hand, serve to impart structure to

Fig.2.1

Graphic Data Structure Blocks

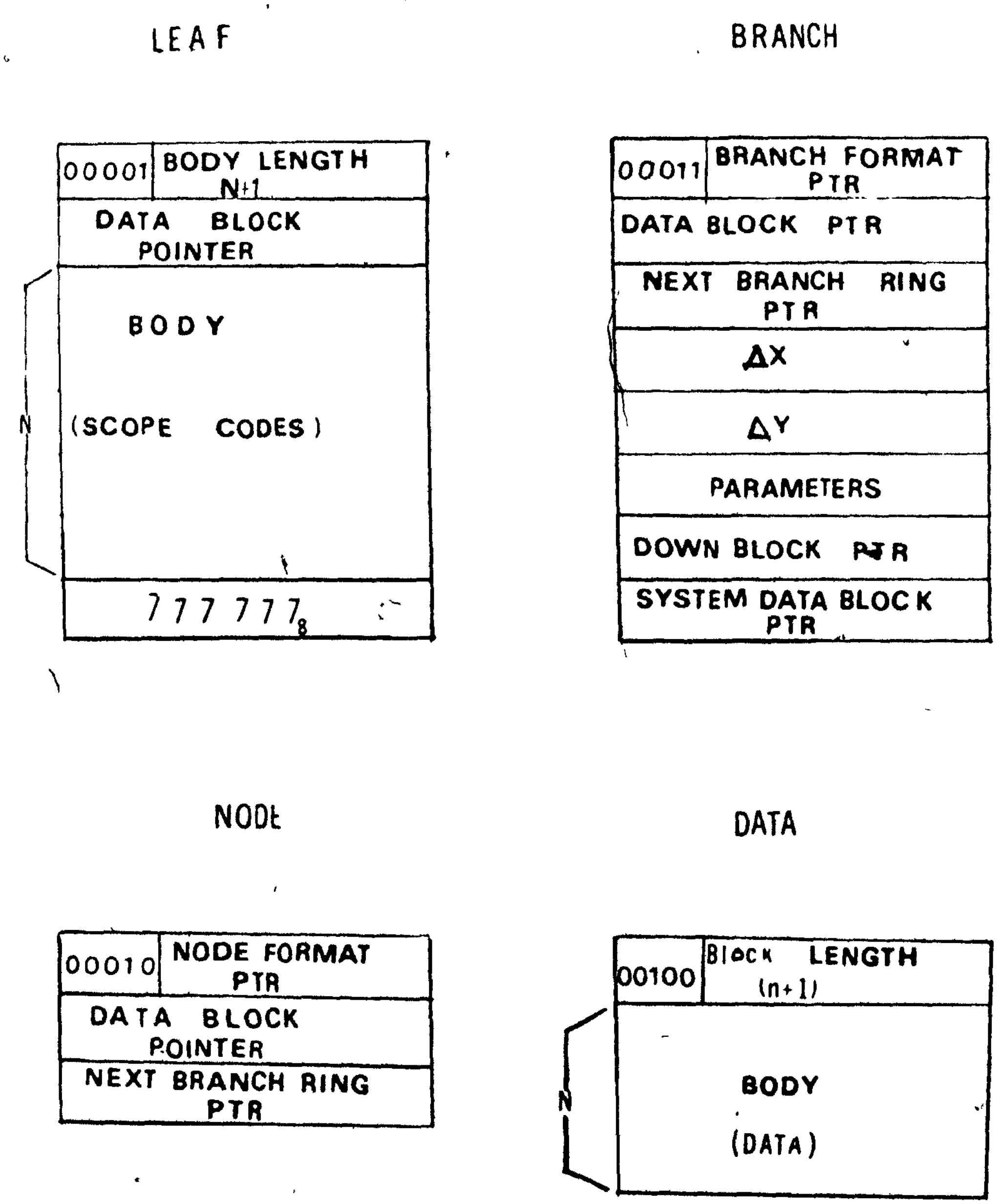
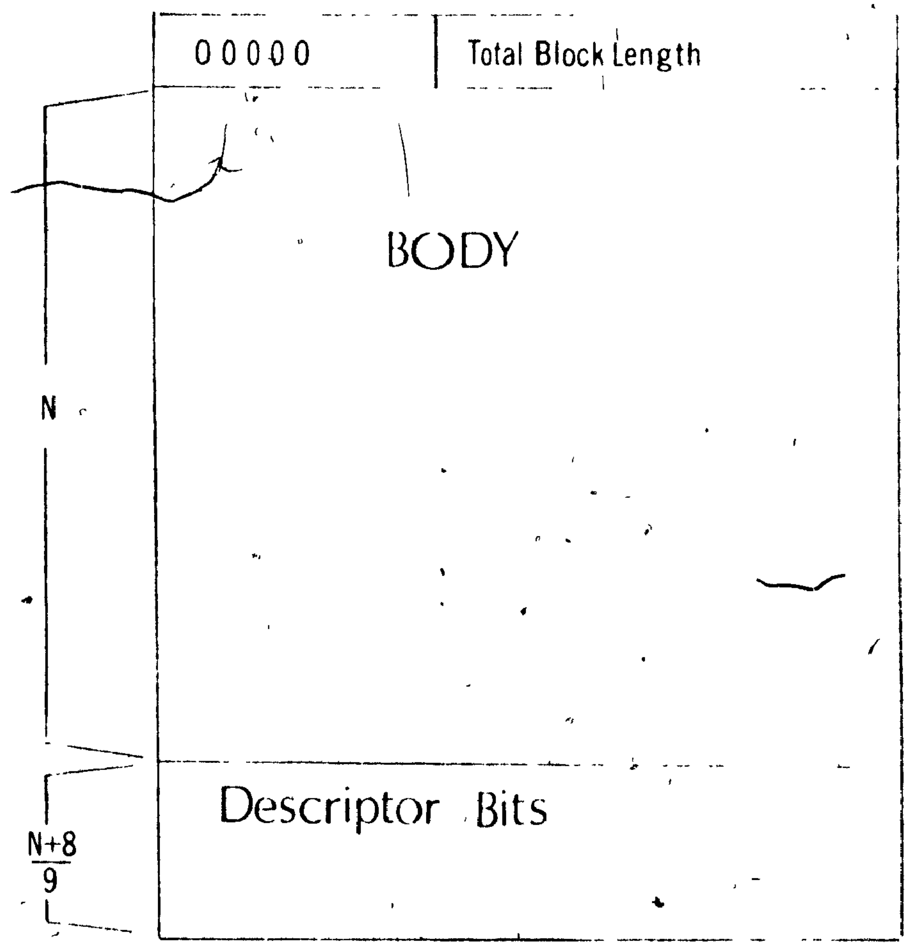


Fig. 2.2



00 RELOCATABLE

01 BLOCK PTR

10 ABSOLUTE

11 2nd WORD, TWO WORD ADDR

the display.

An arbitrary number of branches can enter a node and an arbitrary number leave it. Rather than require the node block to contain pointers to each of the branches, a ring structure is used. The node block contains a pointer to the first of the "out-branches", which points to the next, and so on down the line. Each branch is associated with two nodes, and so must have two rings passing through it. One ring is specified as above. Another word in the branch block points to the node heading the "out-branch" ring.

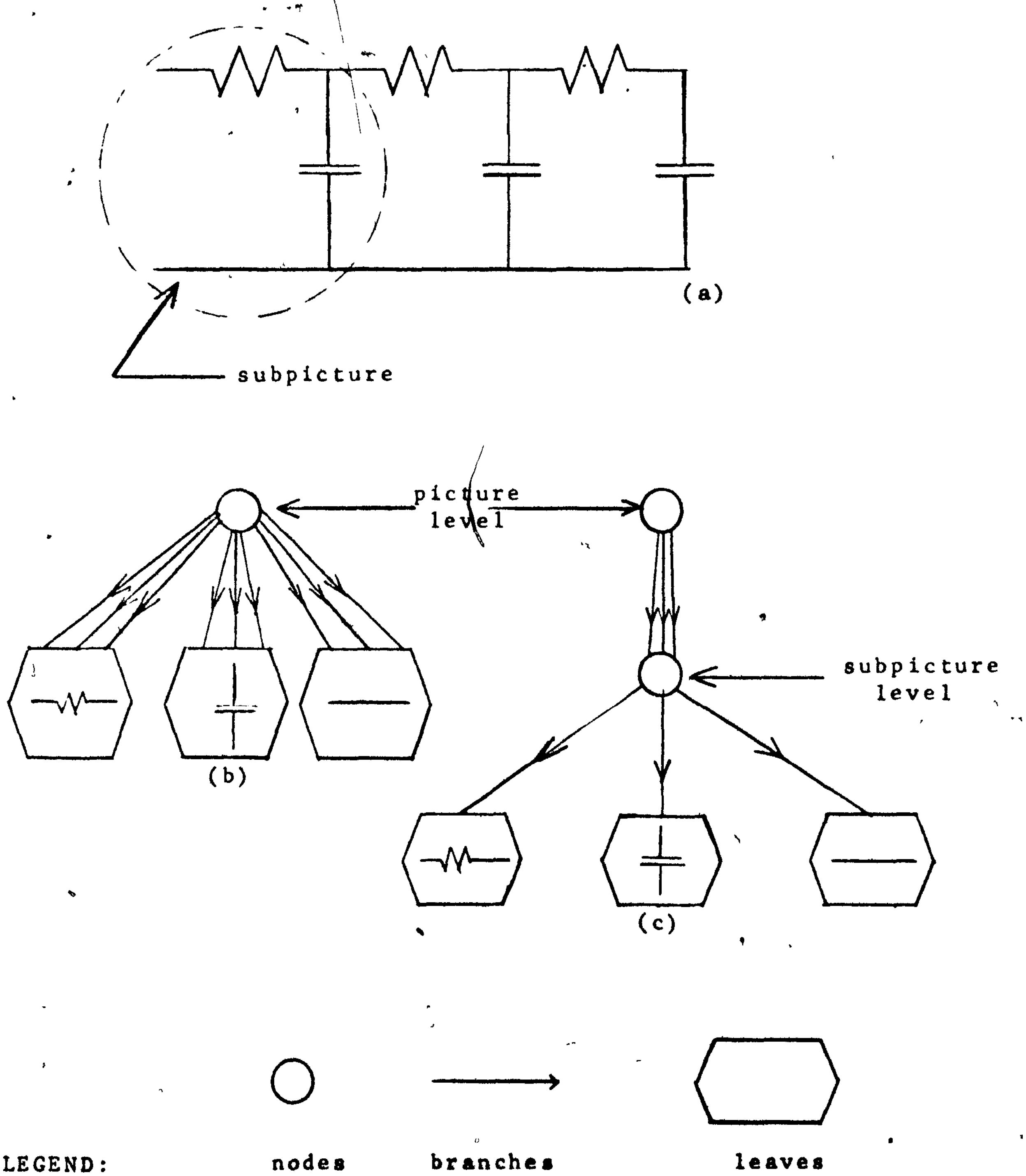
Leaves contain the information defining the visible portion of a picture. When a leaf block is initially generated it has zero body length, and data is "grown" into it at execution time. Text and line-drawing information can be stored (in device-dependent form) in a leaf block.

Non-display data blocks allow the programmer to associate non-display information with the data structure. These blocks are defined and allocated under program control and may be of an arbitrary size. The blocks maybe attached to nodes, leaves or branches in the display, as can be seen from Fig. 2-1.

Consider the picture shown in Fig. 2-3 (a). A group of lines can represent a resistor only if the user grants it that interpretation. To an electrical engineer, there are clearly three graphic elements (leaves) in the picture: the capacitor, resistor and short circuit symbols. Upon reflection it can be seen that Fig. 2-3 (b) and Fig. 2-3 (c) represent two equally valid structures for the picture. Undoubtedly, in a circuit analysis application non-display data blocks would be associated with the structure, denoting element types and values. Indeed, the structure chosen for the representation of the circuit may well depend upon the method of analysis being applied.

FIG. 2.3

A SIMPLE CIRCUIT - WITH TWO POSSIBLE GRAPHIC STRUCTURES



2.2 The GRIN2 Language

The GRIN2 (Graphics Interaction) language is a high-level general-purpose graphical programming language which permits the generation and manipulation of the graphical data structure, and provides statements for controlling real-time man-machine interaction (C.1). GRIN2 statements can be classified into five main categories:

(1) Real-time man-machine interaction
(2) Structure generation
(3) Display data generation
(4) Structure editing
(5) Display control

The GRIN2 language provides statements which query the operator seated at the interactive console. The operator can answer through keyboard or lightpen action, causing control to be passed to other statements, which might use his action to define the course of future processing. One of these fundamental questions provided in GRIN2 is posed by the WHICH statement, which gives the operator the privilege of choosing one of the objects on the screen with the light pen. The resulting action depends upon which object is chosen, but in general a selective transfer of control is involved.

Another basic real-time question is posed by the WHERE statement. A tracking cross appears on the screen and may be positioned using a light pen. When the centre of the cross is correctly positioned, the operator can generate a distinctive interrupt to signal his satisfaction, and processing continues.

A characteristic of the GRIN2 real-time statements is that they force the application programmer to obey Wasserman's First Principle (W.1) for "idiot-proofing" interactive programs. A definite, anticipatable response results for every conceivable user input, simply because the application programmer must, under GRIN2 syntax, specify what action to take for any input. Typing a message when a light-pen hit is anticipated will not, then, bomb the system.

A graphic data structure can be built using the three commands LEAF, BRANCH and NODE, as described in the Bellgraph Programmer's Manual.

Display data in leaves are the scope commands which produce the picture. Statements such as TEXT, VECTOR and PARAM store a series of scope codes in the last leaf specified. Structure editting is possible through such statement as DETACH which detaches branches from the structure, or ATTACH which can attach new branches to it.

The structure to be displayed is attached to a privileged node — the display node-through the DSPNOD command. Branches can be added to the display node without disturbing the existing structure by using the ADDDSP statement, or removed by using the SUBDSP statement. An entirely different node becomes the display node when the NEWDSP command is used. All branches previously attached to the display node are detached.

2.3 Subroutine Blocks

One other sort of block also appears in the McGraph system: the subroutine block, shown in Fig. 2-2. Subroutine blocks contain the PDP-15 object code derived from the assembly of GRIN2 program blocks, and consist of three parts: a header word, the body, and a descriptor section. The header word declares the number of octal words in the block. The body contains the object code proper. And the descriptor section provides relocation information about each word in the block. That is, because a subroutine block must occasionally be relocated in core, each word must be tagged as being absolute, relocatable, a block-pointer, or part of a two-word address. (These notions will be fully explained in Chapter 6). When a block is moved, then, all code within it, and all references to it, may be updated, if necessary.

2.4 A Short GRIN2 Program

Consider the following trivial problem. A user wishes to display a triangle and a square on the screen, then have the privilege of using the light pen to select one of them. The chosen one must disappear while the other remains visible. Subsequently, selecting the visible object with the light pen must cause it to disappear and the invisible one to reappear.

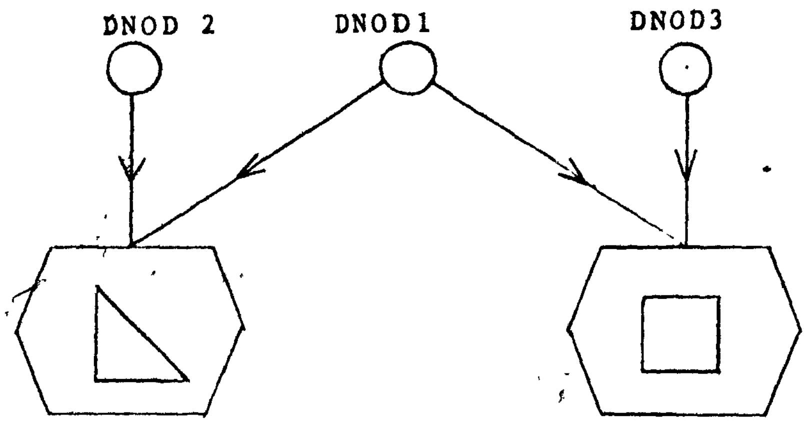
This problem can be represented graphically as in Fig. 2-4. Three display trees are defined, corresponding to the three desired displays. As shown in Fig. 2.4 (a), the node DNOD1 is the root of the tree which consists of branches to the leaf which defines the triangle and the leaf which defines the square. Similarly DNOD2 and DNOD3 are the roots of the other two trees. If DNOD1 were attached to the display node, the triangle and square would be visible, and so on.

A GRIN2 program solving this user's problem appears in Fig. 2.5. The graphic data structure is first defined (by the sequence of NODE, BRANCH and LEAF statements) and the display data generated (by VECTOR and INVECT statements). The node DNOD1 is then attached to the display node, and the display turned on and the light pen enabled by the WHICH statement. Each branch to the square and triangle in the graphic data structure is defined as a light button. That is, selecting either leaf causes a logical transfer to take place. The display node is appropriately redefined, a new picture generated and the light pen re-enabled.

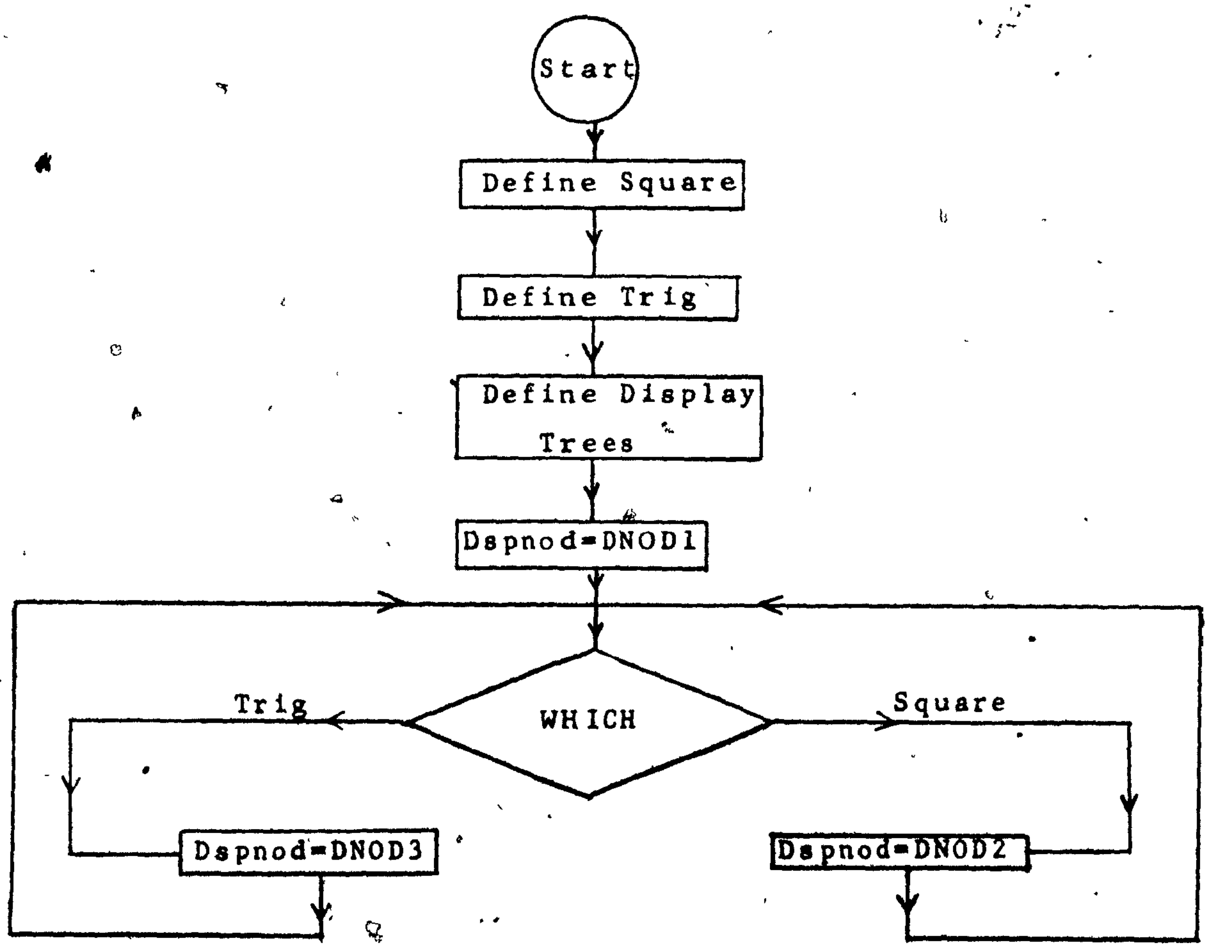
Pictures of this process are shown in Chapter 6, where this program will be studied in far greater detail.

FIG. 2.4

A SIMPLE GRIN2 PROGRAM



(a) Graphic Data Structure



(b) Flowchart

Fig.2.5

Simple GRIN2 Program

```
STMT   SOURCE STATEMENT

   1           ICTL    1,79,16
   2           PRINT   NOGEN
   3           G2ID    501
 179           LEAF    TRIG
 180                   *,*                   000001
 185           CURVE   ((W,0),(-W,W),(0,-W))
 190                   *,*                   000005
 209           LEAF    SQUARE
 210                   *,*                   000013
 214           CURVE   ((W,0),(0,W),(-W,0),(0,-W))
 215                   *,*                   000017
 243           TREE    DNOD2,,(,(200,400),TRIG,,SFIX),(,(800,400),SQUARE,,TFIX)
 244                   *,*                   000026
 252                   *,*                   000032
 266                   *,*                   000043
 280           NODE    DNOD1
 281                   *,*                   000054
 289           NODE    DNOD3
 290                   *,*                   000060
 298           BRANCH  BRCHL,DNOD1,(200,400),TRIG,,SFIX
 299                   *,*                   000064
 316           BRANCH  BRCHR,DNOD3,(800,400),SQUARE,,TFIX
 317                   *,*                   000075
 334           DSPNOD  DNOD2
 335                   *,*                   000106
 343 LOOP      WHICH   ,,P1
 345                   *,*                   000115
 351           GOTO    LOOP
 352                   *,*                   000122
 354 SFIX      NEWDSP  (BRCHR)
 356                   *,*                   000123
 365           GOTO    LOOP
 366                   *,*                   000133
 368 TFIX      NEWDSP  (BRCHL)
 370                   *,*                   000134
 379           GOTO    LOOP
 380                   *,*                   000144
 382 W         EQU     62
 383           GEND    ,
 387           END
```

CHAPTER 3 McGRAPH FACILITIES

It will prove useful at this point to review the Bellgraph and McGraph facilities, and see how the differences between them affected the development of the McGraph operating system.

The McGraph facility, previously described by Malowany [M. 1], is shown in Fig. 3.1. As can be seen, the heart of the system is a PDP-15/20 with 24K words of core, a tape drive and two disks. While the bulk of the system code resides on this machine, all graphic peripherals are interfaced to the neighbouring PDP-8. The two computers are in turn interfaced by means of a high-speed data buffer, generally called "the link". Only a minimal set of handlers for the peripherals reside on the PDP-8.

The Bellgraph facility described by Christensen is shown in Fig. 3.2. It consists of a large central computer with many graphic peripherals, including several GRAPHIC2 terminals [C. 1]. These terminals (Fig. 3.3) are actually intelligent satellites with extensive interactive graphic capabilities, not unlike the McGraph facility. Indeed, it is that section of the Bellgraph operating system which resides in these terminals that McGill fell heir to, and which has been developed into the McGraph operating system. It might be noted that PDP-9 object code is actually a subset of PDP-15 object code, so much of the Bellgraph system would run without modification. Thankfully as well, standard DEC peripherals were used extensively in both systems (paper tape reader/punch, console keyboard, DEC-DISK, etc.). In short, there was a strong enough resemblance between the systems to make it seem worthwhile to adopt the Bellgraph system wholesale, and save on development work.

THE McGRAPH FACILITY

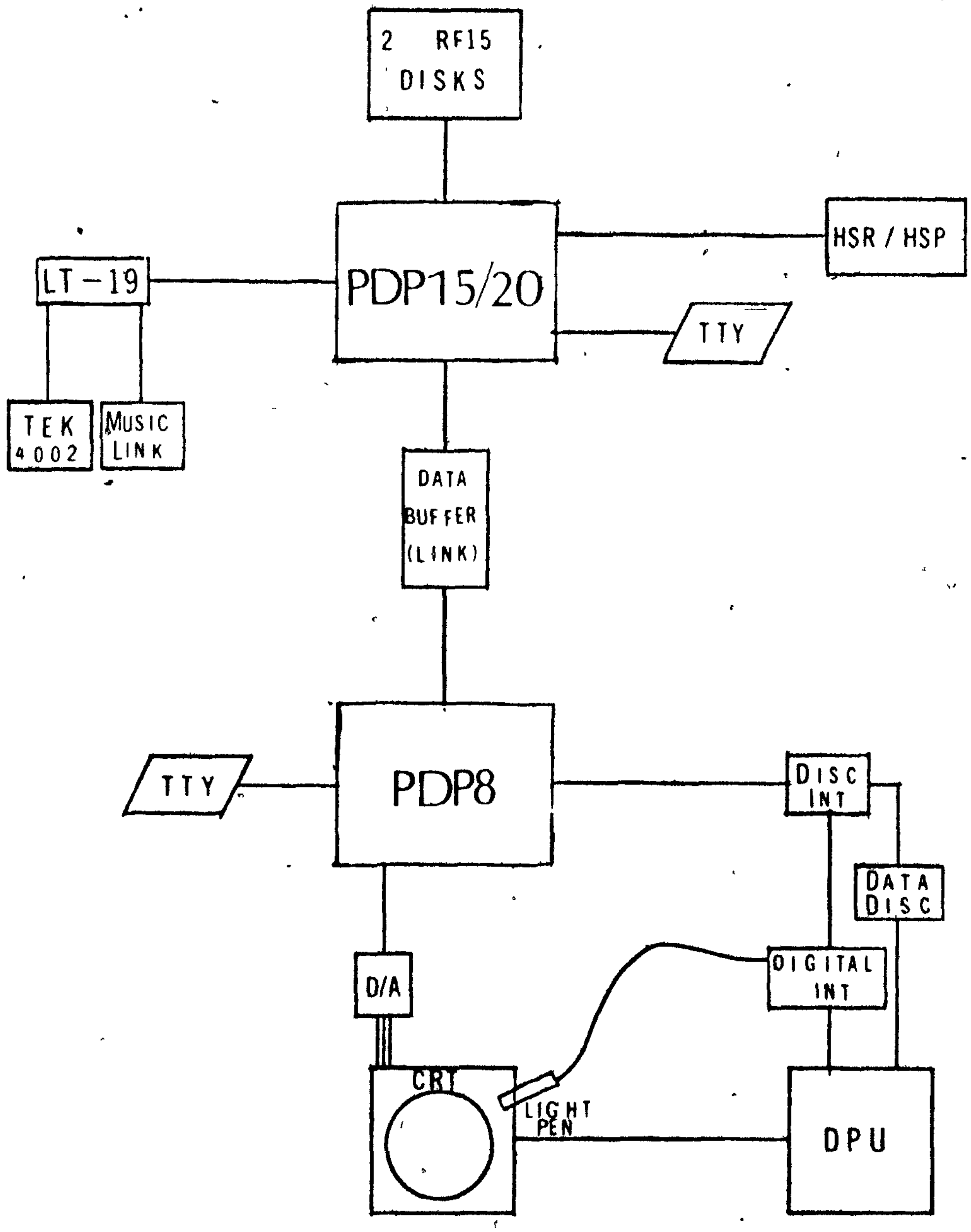


FIG. 3.1

Fig.3.2

Bellgraph Facility, Holmdel, 1970

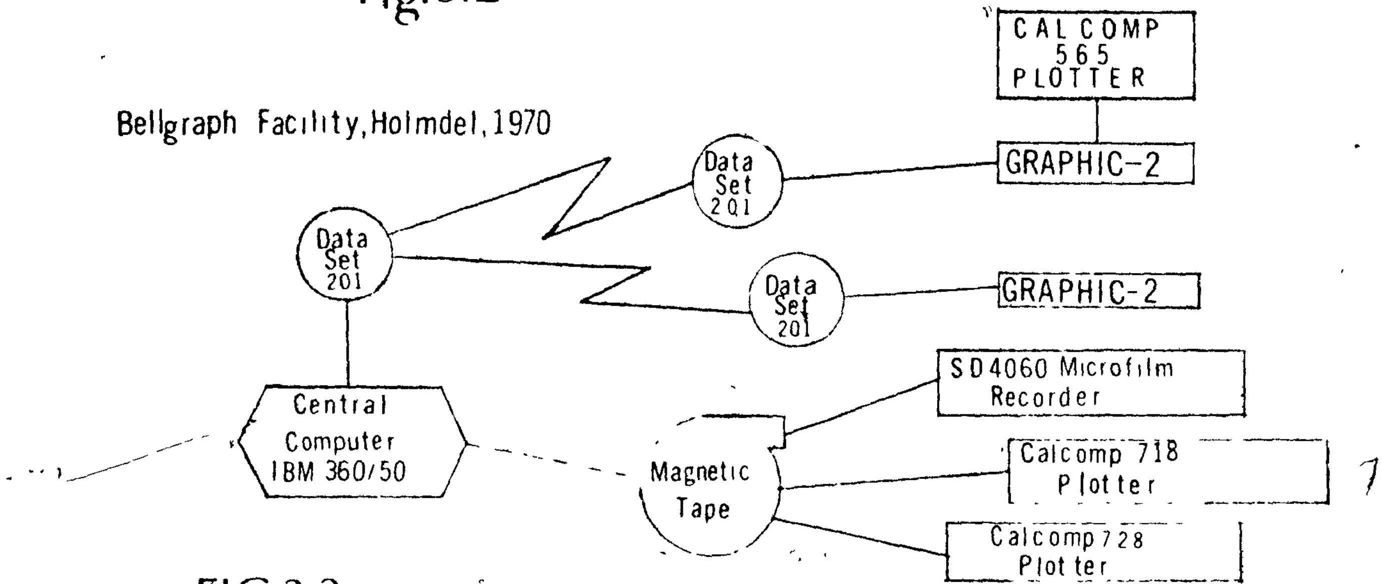
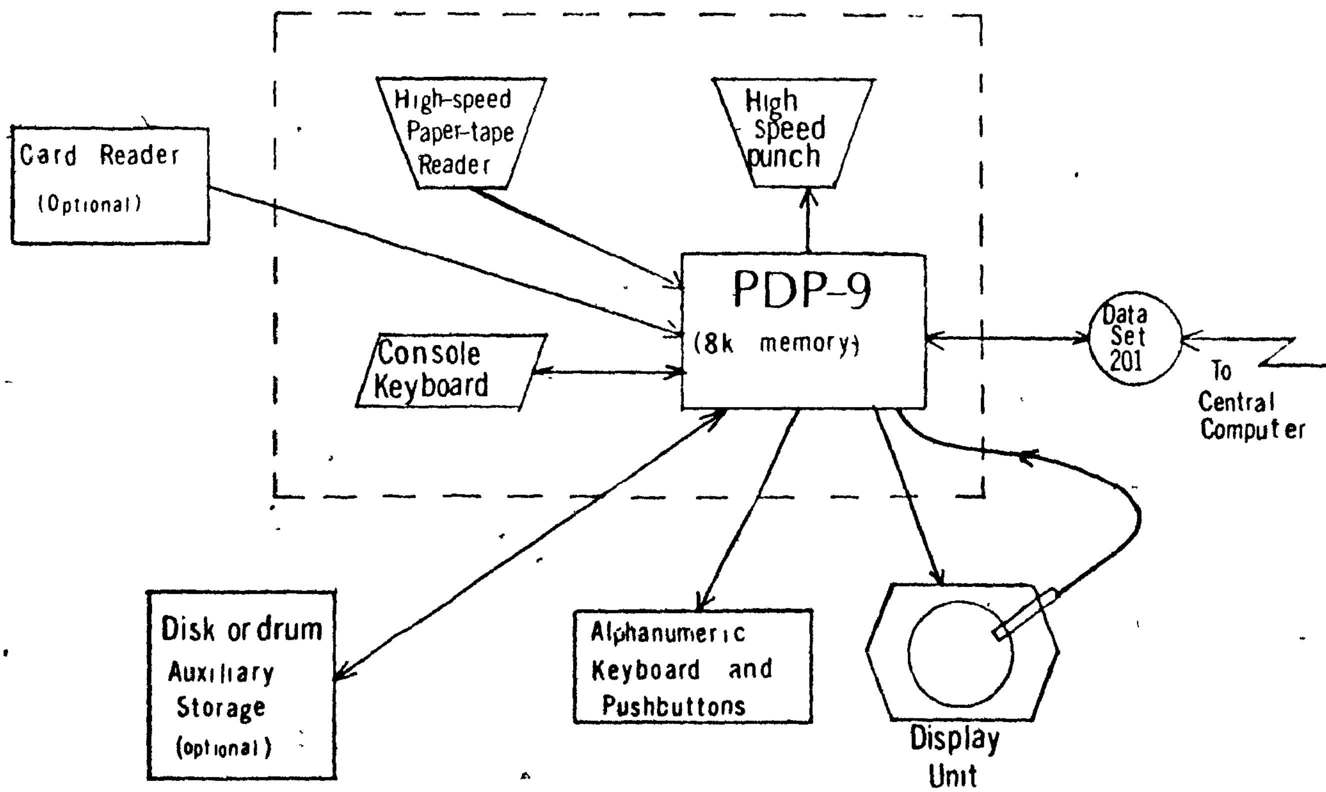


FIG.3.3

Graphic-2 Organization



The many and fundamental differences between the systems precluded all possibility of a quick and easy implementation, however. Some differences were easily overcome, once recognized; others proved to be recurring nightmares. For instance, neither Automatic Priority Interrupt nor a real-time clock exist on the PDP-15 at McGill, while they are standard features on the GRAPHIC2 units. Upon investigation, it was found that the real-time clock was no longer necessary with the current refresh philosophy, while API could by easily circumvented through the use of a skip chain- of course, with an attendant loss of speed in interrupt servicing. Further, when the system was first loaded a subtle difference in addressing schemes was noted. In standard indirect-addressing a sixteen-bit address is developed and referenced [MAN-5]. In addition to this, internal indirect addressing [MAN-1] is available on the GRAPHIC2 unit but not on the PDP-15 at McGill. In this scheme, a thirteen-bit address, capable of referencing words within the current bank, is developed. This method was, unfortunately, used extensively in Bellgraph since it is somewhat faster than ordinary indirect addressing, and since the GRAPHIC2 units had only 8K of core. Once recognized, this problem was solved by an appropriate software patch [M. 4].

More seriously, the display units in the two systems are very different. Not only do they have different control and scope codes, but the GRAPHIC2 unit is refreshed through the use of a DMA channel, while the McGraph unit is disc-refreshed. This has far-reaching consequences. The point may be made that any general operating system must be able to support new devices with a minimum of pain. Bellgraph is a general purpose operating system; that part of it which resides in the GRAPHIC2 units is manifestly not. It was designed with a given, high-performance piece of hardware in mind, and optimized accordingly. It was not meant to be transplanted and to a degree refused to be.

Many functions associated with GRAPHIC2 hardware are performed very differently by McGraph hardware and software. Files cannot be blinked in McGraph, for instance. Clipping and windowing must be performed by software rather than hardware. Physical pushbuttons are simply unavailable, and there is no keyboard directly associated with the McGraph display unit. Finally, and possibly most important, upon a light-pen interrupt the information available from the McGraph unit is entirely different from that available from the GRAPHIC2 unit. The GRAPHIC2 returns the beam co-ordinates at interrupt time; the McGraph unit returns a filename ID, associated with a block of data written on the DATA DISC.

The gravest problems can be traced to differences in hardware organization, which forced changes in software philosophy. The GRAPHIC2 terminal is entirely interrupt-driven. In McGraph, on the other hand, all graphic peripherals are interfaced to the PDP-8 rather than the PDP-15. This, coupled with the difficulties previously mentioned, forced a complete restructuring of the real-time philosophy. This will be pursued in more detail in Chapter 4.

SECTION B THE McGRAPH OPERATING SYSTEM

CHAPTER 4 THE McGRAPH EXECUTIVE PROGRAM

The McGraph operating system has been developed from the Bellgraph operating system, and so possesses the same basic structure. Of course the demands of a new environment had to be met, but this required more of a change in system content than of philosophy. The structure of the McGraph operating system is shown in Fig. 4-1. The executive program can be considered to be composed of a set of core-resident routines, and certain members of a disk-resident library. Associated packages include a library editor, the GRIN2 language assembly system, and a set of on-line debug routines. The executive program will be considered in this chapter, and the associated packages in Chapter 7.

The executive program may be broken down as shown in Fig. 4-2. In this division the real-time subsystem handles all I/O activity; the memory-management subsystem controls picture generation; and the device translation subsystem generates device-dependent code from GRIN2 object code. Each of these systems will be described in turn.

The McGraph Operating System

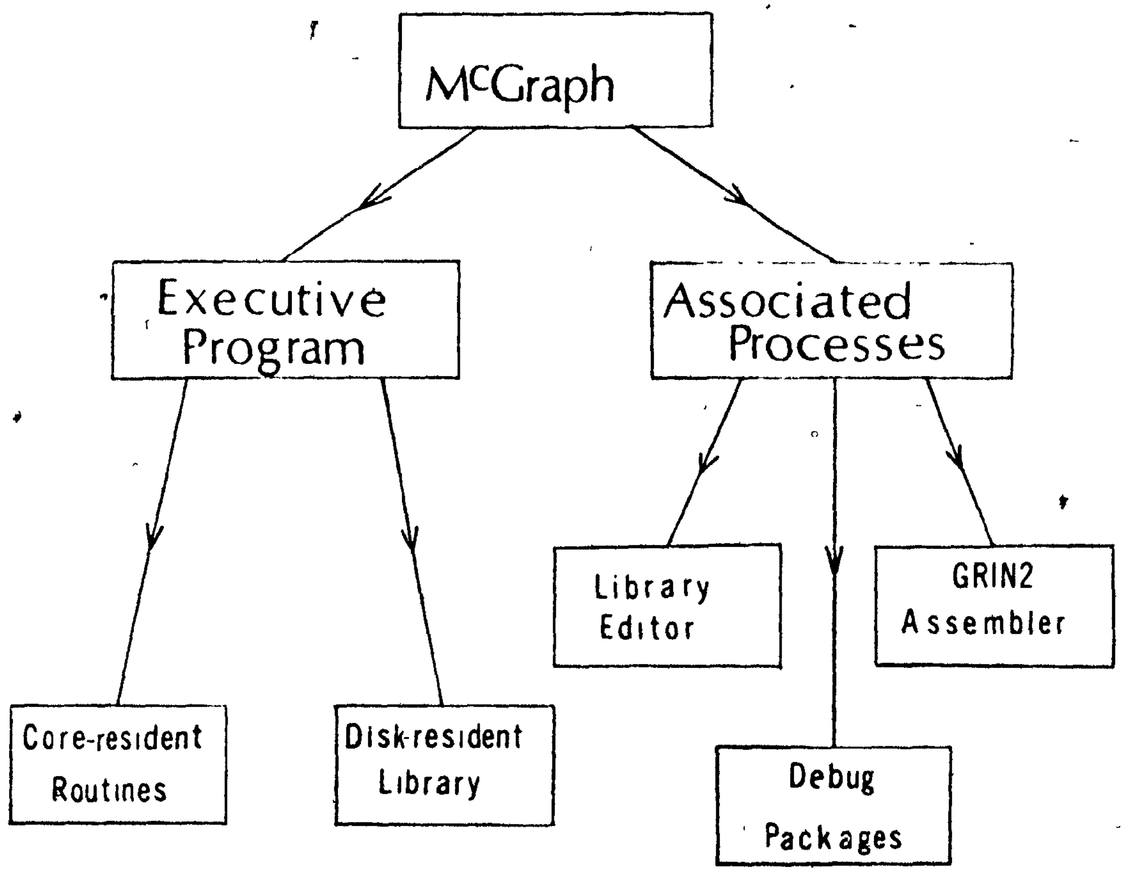
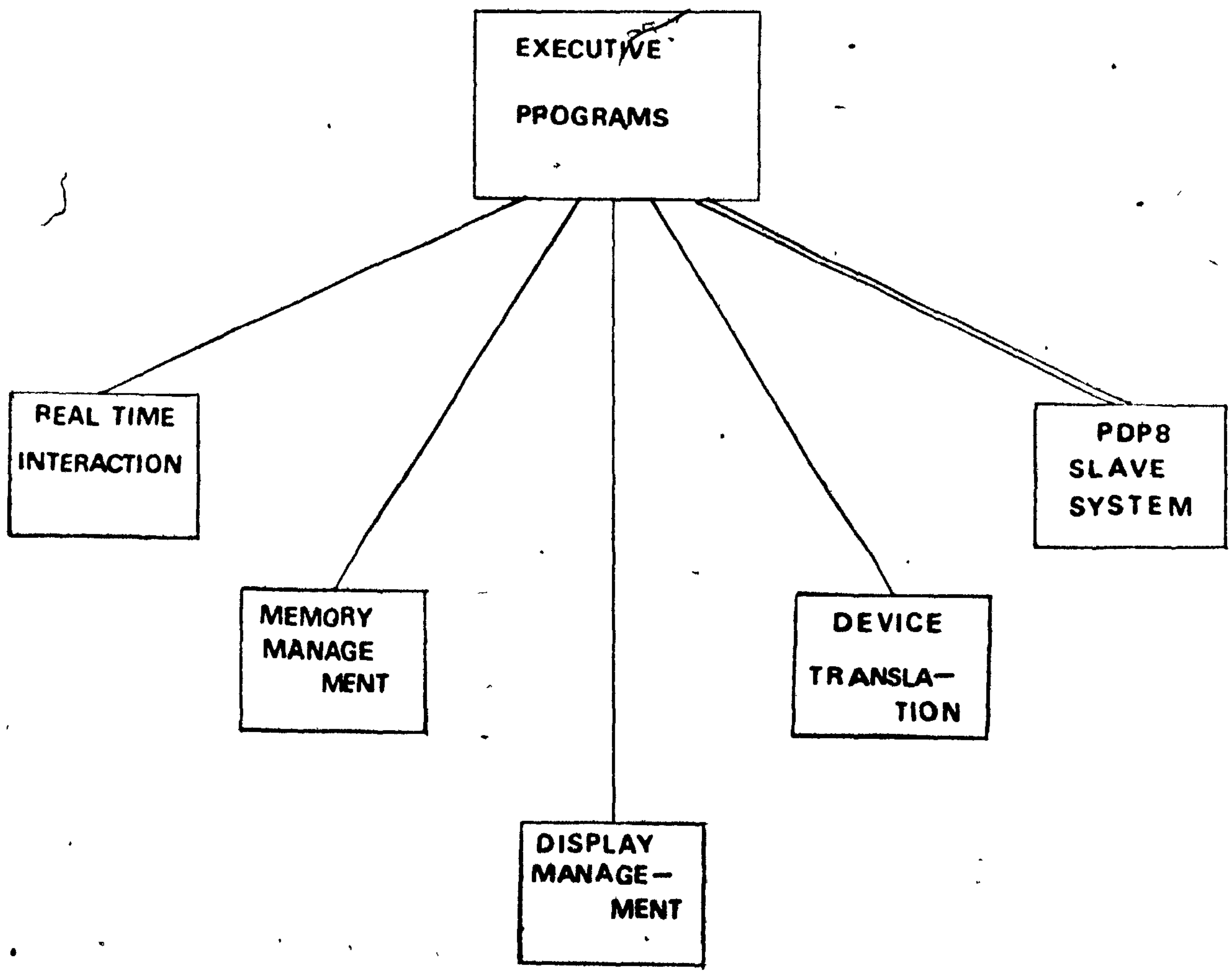


FIG.4.1

FIG.4.2

The McGraph Executive



4. 1 The Real-Time Subsystem

The heart of any executive program is a set of I/O interrupt handlers. Because McGraph is a graphics-oriented operating system, service routines must exist not only for the standard devices but also for the graphics peripherals: notably the light-pen, joystick and display unit. The true extent of the difference between the McGraph facility and the GRAPHIC-2 can only be appreciated after having examined the changes made within the real-time subsystem.

Some changes were relatively straight-forward. The API facility, for instance, would have been aesthetically pleasing, and might have simplified some of the programming. It was not essential. It was also discovered that the real-time clock had been used only to test for central computer time-outs, and to signal the start of a new display trace. Since neither of these operations were meaningful in McGraph, the real-time clock could be safely deleted.

Bellgraph was an interrupt-driven system. All operator interactions, light-pen hits, keyboard strikes, etc. — were detected by means of interrupts, as were alarm conditions such as display edge violation and the end of a leaf. Upon an interrupt, control was handed to the subroutine associated with the interrupt. This association was changeable from outside the handler, allowing programs to specify the reaction to any input.

The McGraph architecture forced a re-evaluation of this philosophy. First, the hardware does not trap edge violations or ends of leaves. If it is considered that these are still worth flagging down, this must be accomplished through software. As such, these operations can no longer be considered to be part of the real-time sub-system, but of the display management system.

Second, all graphics peripherals are currently interfaced to the PDP-8 rather than the PDP-15. There can, then, only be one common interrupt from all the graphical devices. This turned out to have far-reaching consequences.

At the beginning of the project it was decided that it was more important to get some system working than it was to design an efficient one. Efficiency could always be improved at a later date. Accordingly it was decided that the best approach would be the one which required the fewest changes to the Bellgraph coding. A package was developed which simulated the operation of the GRAPHIC-2 unit. Each IOT issued was simulated by this package, and each scope command was translated into appropriate McGraph coding.

After a short period of time it was found that, using this approach, it was impractical to run McGraph in an interrupt-driven form. Fortunately, it was later seen to be unnecessary. Upon close scrutiny of the listings it was seen that all graphical I/O transfers could occur at one place within the program, with no loss in generality. After one full display pass a light-pen hit or teletype interrupt must occur before processing can resume. One does not really have to wait for an interrupt. Rather, the teletype and PDP-8 link flags can be continuously scanned. This is the approach which was ultimately implemented and shown to work.

4.2 Device Translation Subsystem

The Bellgraph device translation subsystem was mostly implemented in the form of GRIN2 language library subroutines. These generated GRAPHIC2 code on-line and placed it in the leaves, as they were grown. It was these routines which converted the device-independent GRIN2 picture description into explicit scope codes. Therefore, by changing this set of subroutines one could theoretically generalize Bellgraph to run with any display unit.

This was not clear until the project was well under way; the hazards of the approach became evident even later. In any case, it was decided (M. 4) that the best approach for a preliminary system would be not to modify these subroutines, but rather to re-interpret the content of the leaves (and system messages) as they were sent to the display unit. The subprogram which translates GRAPHIC2 scope code into McGraph code is known as G2TRAN (Fig. 4.4). The extent of the inefficiency inherent in this approach was not really appreciated until the system was run. But the approach remains valid, in that it definitely reduced debug time. When using routines which had been run successfully on a GRAPHIC2 unit, one knew that any peculiarities in the displays had to be due to the operation of the McGraph system. This greatly simplified the system debug procedures. Now that McGraph has, essentially been debugged G2TRAN is expendable. A method of eliminating it will be presented in Chapter 8.

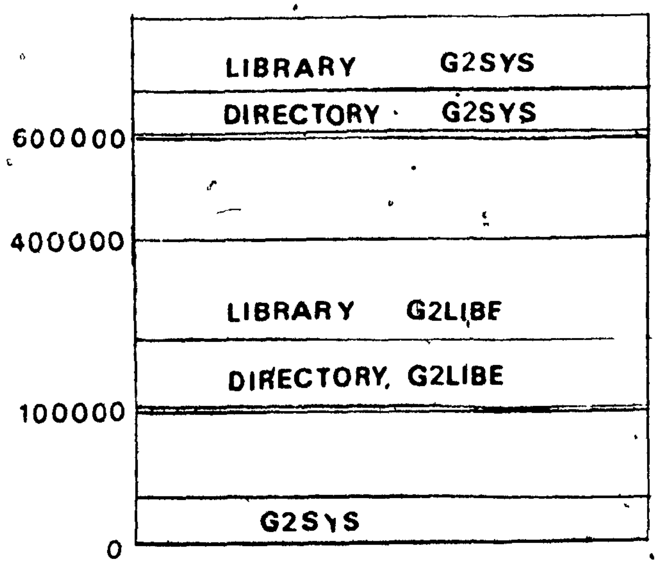
4.3 Memory Management

Although McGraph is constrained to run in 8K of core on the PDP-15, the memory management programs give it a very large virtual memory, using an RF15 disk as secondary storage. To make this possible the user's program and data are broken into blocks of various sizes. Each subroutine block is given a unique 17-bit ID before it is loaded into core, and all interblock communication is in terms of these ID's and an offset within a block. To facilitate interblock addressing, a Block Table is developed in core (Fig. 4-4). Each block loaded into core requires another two-word entry in the block table: the block ID and starting address. Now, of course, a correspondence between block ID and core position is established. When a block is relocated in core, only the core address in it's block table entry need be updated. If a referenced ID is not the block table, the block is not in core, and must be loaded from disk or paper tape. When a block is deleted from core its entry in the block table is purged. The hole created by removing a block is added to the free space (Fig. 4-4) by relocating all blocks above it. There are never any gaps between blocks.

Very fortunately, the memory management section of Bellgraph was easily adaptable to the McGraph environment. In fact, the difference of any importance is that there is no link to a host computer in the McGraph system. After carefully deleting all references to dual-processing and making the disk handler permanently core-resident, the Bellgraph memory management system ran perfectly in its new environment.

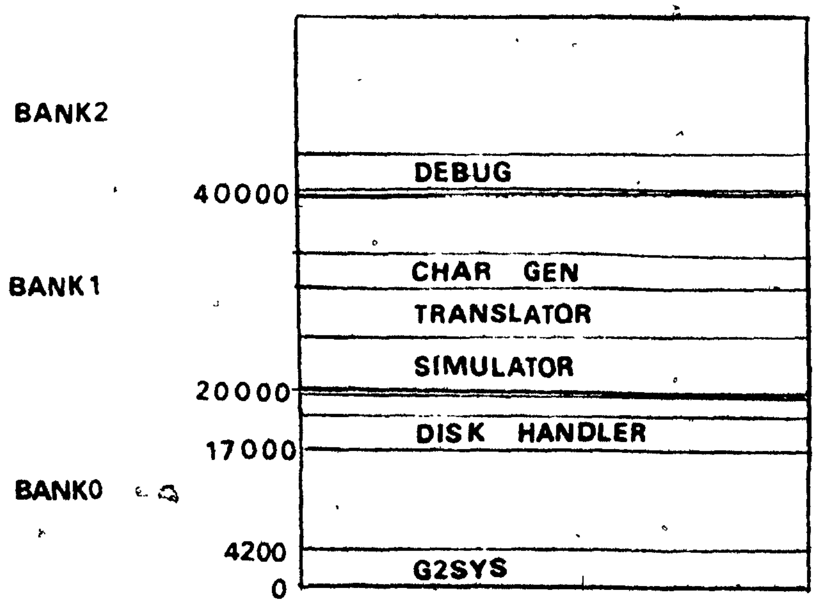
Disk Map

Fig.4.3



Core Map

Fig.4.4



A core map for the McGraph system is shown in Fig. 4.4. Note, the disk handler appears at the top of the lowest 8K memory bank. In Bellgraph it was continually swapped with the character-generator table, but the construction of G2TRAN obviates this action. The disk handler could, then become core-resident.

A disk map is shown in Fig. 4.3. Basically, two storage regions are defined by addresses within the control of the system programmer: the scratch region and the program library. First, the scratch area contains program blocks which are loaded from an external source during a run by G2SYS; and second, the dynamically-created blocks in the graphic data structure which had to be removed from core. The program library contains the GRIN language statement subroutines, some utility programs, and a few application routines. This section may be updated by the utility program G2LIBE, of which more will be said in Chapter 7. Note that the program library can be write-protected during a run.

4.4 Display Management

The semi-interpretive display program of Bellgraph has been maintained, with several notable changes. First the display-trap functions, previously part of the real-time subsystem, must now be considered to be part of the display management subsystem. Since such conditions as edge violations and ends of leaves can no longer cause an interrupt on the PDP-15, these events must be recognized and dealt with by the simulator package as they arise. Accordingly, the position of each vector or point must now be calculated before the scope command is sent to the display unit to prevent edge violations.

The remainder of the display management system is the same, with one exception. Only one display pass is made to refresh the display unit. And the command which had been used to start the DMA data transfer was interpretted as the command to start the translator and send the output to the PDP-8. This system is found to be quite adequate.

4.5 PDP-8 Slave Program

The slave system on the PDP-8 was written and maintained by Juhan Leemet. Since it has been changed quite drastically since last reported on (M. 4), a short description of it will be given here. The present monitor is capable of performing the following tasks, on demand from the PDP-15:

1. Receive graphics instructions, buffer them and write the buffer onto the DATA DISC when the buffer overflows, or upon receipt of an end-of-leaf flag.

2. Accept light pen hits and output the light pen flag and the current filename ID.

3. Display and manage the tracking cross.

No longer can files be blinked; it has been decided that this is not an appropriate method of drawing attention to a file in the current implementation. Similarly, it was decided that it was a waste of effort to simulate the GRAPHIC2 console keyboard; the PDP-15 keyboard is sufficient for the operation of the system. Simulation of the back-lighted pushbuttons was also abandoned for the same reason; they added no unique capabilities to the McGraph system.

The result of all these changes was a smoother-running, more efficient system. By eliminating side-issues, the real-time system response was greatly improved. The greatest change of all, however, was that the PDP-8 was made to scan the digital interface and link flags continuously, rather than checking the link at track origin and the interface the rest of the time. Response time for a light-pen hit is slightly increased, but transmission time for graphic files is greatly decreased. The net result of all these changes is a slave system which permits graphic interactions to proceed almost an order of magnitude more rapidly than previously.

CHAPTER 5 SUBROUTINE BLOCKS

5.1 McGraph Object Code

As was mentioned in Section 2.3, subroutine blocks contain McGraph object code for the program blocks. It can be seen from Fig. 5.1 that the header word always contains the block length. The body of the block, following the header, contains the object code, while the final section contains descriptor bits reporting the nature of each command in the body. Words can be designated as being absolute, relocatable, a block-pointer, or the second word of a two-word external reference. Two bits must be reserved in the descriptor section, then, for each word in the body of the block (Fig. 5.1). When a GRIN2 program is post-processed part of the printed output consists of an octal dump of the subroutine block just generated. To increase legibility, however, the descriptor section is replaced by letters following each word in the body of the subroutine, designating the word type. That is:

B = block pointer

R = relocatable; an address relative to the beginning of the block

C = second word of a two-word external reference

If no letter follows the word, it is assumed to be absolute.

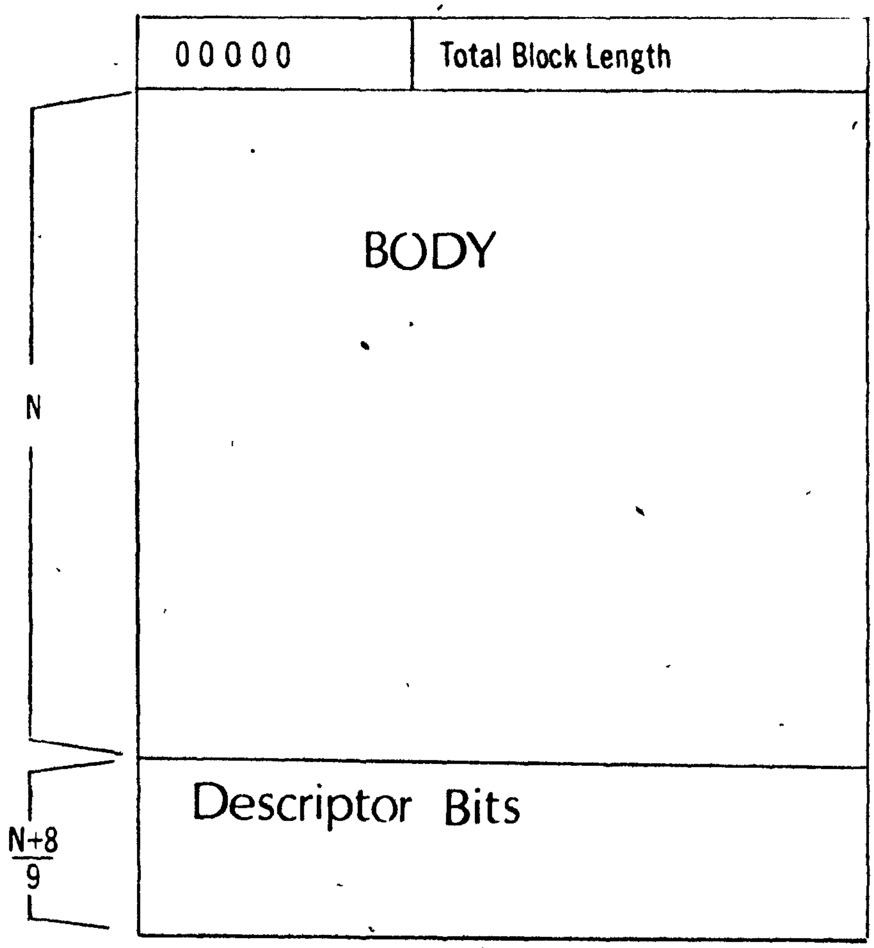
To a point, of course, McGraph object code can be simply be considered to be PDP-15 object code. An appreciation of the executive program is, however, necessary in order to understand a GRIN2 program dump. A short table of PDP-15 instructions is shown in Fig. 5-2. In McGraph, the direct CAL instruction is interpretted as denoting a trap to the error handler, while an indirect CAL instruction denotes an external reference (i.e. a reference to information contained in some other subroutine block).

Fig. 5.3 (a) shows a reference to a GRIN2 library subroutine whose ID is 165. The entry point is to relative address one within the subroutine block. Fig. 5.3 (b) shows a reference to the same subroutine, but with an entry to relative address four.

A recurring structure which may be observed in most GRIN2 dumps consists of a call to an external subroutine, followed by an argument list (Fig. 5.3 (c)). Arguments are expressed in terms of JMP instructions (6XXXXX), and take the form of pointers to information. This example shows a call to the GRIN2 library subroutine with ID106 (BRANCH), followed by seven arguments. These are, in order: (1) a branch pointer, (2) a pointer to the node the branch originates at, (3) a pointer to a two-word vector giving branch displacement, (4) a pointer to the node the branch ends at, (5) a pointer to a data block, if it exists, (6) a system pointer if the branch is a "light button", and (7) a pointer to a parameter word. Compare this, for structure, with the branch block of Fig. 2.1.

This is the general form into which GRIN2 statements are expanded.

Fig.5.1



00 RELOCATABLE

01 BLOCK PTR

10 ABSOLUTE

11 2nd WORD, TWO WORD ADDR

FIGURE 5.2

THE PDP-15 INSTRUCTION SET

MNEM	CODE	OPERATION
CAL	00	Subr. Jump to Absolute 20
DAC	04	Deposit AC in Memory
JMS	10	Jump to Subroutine
DZM	14	Deposit Zero in Memory
LAC	20	Load Accumulator From Memory
XOR	24	Exclusive-OR
ADD	30	One's Complement Addition
TAD	34	Two's Complement Addition
XCT	40	Execute Location
ISZ	44	Increment & Skip if Zero
AND	50	Logical-And
SAD	54	Skip if Accumulator = Memory
JUMP	60	Jump, Unconditional
EAE CLASS	64	Extended Arithmetic Element Op's
IOT'S	70	
OP. CLASS	74	

If Bit 5 is set, indirect addressing is requested.

FIGURE 5.3

McGRAPH OBJECT CODE

030001	
000165 C	
(a) External Ref.: RA = 1	
030004	
000165 C	
(b) External Ref.: RA = 4	

030001	
000106	C
600200	
600174	R
600177	R
600155	R
600024	
600134	R
600024	

(c) Subroutine Call With Argument List

5.2 GRIN2 Language Statement Implementation

GRIN2 language statements are implemented in the uniform fashion suggested in the previous section. First, the assembler resident on the IBM 360 is invoked to translate each statement into the form of Fig. 5.4. Registers (AC, MQ) may be set to transfer some limited information, such as the number of vectors to be generated in one call to VECTOR. Next, a call to the (external) language subroutine is issued. These subroutines are generally part of the disk-resident library which is maintained by the utility G2LIBE, and which must be loaded prior to each terminal session. Following the external reference is a list of arguments, pointing to information required by the language statement subroutine (Fig. 5.3 (c)). All these stages exist in every language-statement implementation.

Needless to say, the language-statement subroutine called does the real work of implementing the GRIN2 statement. The GRIN2 object code, on the other hand, can be seen to be graphic-device independent. Theoretically, all device dependency due to language considerations should be introduced at the language subroutine level. This considerably simplifies the task of modifying the software to meet changing hardware requirements.

Fig.5.4

Subroutine Call

CLA!CLL CMA	Set Registers
030001 000165	EXTERNAL REFERENCE
6xxxxx 6xxxxx - - - - - - - - 6zzzzz	PASS ARGUMENTS

CHAPTER 6 A SIMPLE PROGRAM

In this chapter the simple GRIN2 program written in Chapter 2 will be used to illustrate the functioning of the McGraph operating system. The object code will be analyzed, and one will see how graphic data structure blocks are built and manipulated.

6.1 The Object Code

If the program written in Chapter 2 (Fig. 6.1) is submitted to the GRIN2 assembly system (Chapter 7), part of the printed output consists of an octal dump of the subroutine block produced. This is shown in Fig. 6.2. According to the format of subroutine blocks (Fig. 2.2), the first word of the block should containe the word count for the block, including instruction and relocation words. In the example there are 222 (octal) words, of which the 203 (octal) instruction words are explicitely dumped. As explained in Chapter 5, the relocation words are translated into alphabetic suffixes to the appropriate instruction words. In Fig. 6.2, the eight right-most columns give the contents of eight adjacent memory locations in the subroutine block, with the starting address given by the five-digit octal address in the second column from the left. That is, the contents of location 70 (octal) of the subroutine block is 600176.

Examine Fig. 6.1 and Fig. 6.2 more carefully, recalling that all GRIN2 language statements are implemented by calls to library subroutines, in the format shown in Fig. 5.3 (c). A collection of library subroutine ID numbers and entry points is shown in Fig. 6.3. Now, one can identify words one to four of the dump as a call -

to the GRIN2 library subroutine with ID101, followed by two arguments. From Fig. 6.3, one can see that this is the LEAF library subroutine. Comparing this to the source listing of the program (Fig. 6.1), it may be seen that the first executable instruction was the LEAF declaration; this is very comforting.

Continuing in the same vein, consider the next executable instruction in the program. The CURVE statement can be used to define a figure consisting of a number of successive head-to-tail visible vectors-in this case the triangle. The statement is implemented by calling the library subroutine VECTOR, and informing it of what sort of vectors must be drawn. Examining words five to eleven of the dump, one sees that the two's complement of the number of vectors required is loaded into the accumulator, then VECTOR, ID103, is called. Entry is through relative address 1, the entry point for visible vectors (Fig. 6.2). The three arguments following the call define the vectors to be drawn. For instance, the first pointer is to location 147 relative, the start of a two-word vector containing the x- and y-deflections of the first vector to be drawn. Since 147 contains 62 (decimal) and 15Ø contains Ø, the first vector is to be drawn for 62 units in a horizontal direction. This is, of course, exactly what was specified in the source listing (Fig. 6.1).

Subsequent LEAF and CURVE statements defining the square are treated in an identical manner. NODE and BRANCH statements are translated into calls to the NODE and BRANCH library subroutines (ID 105 and 106, respectively), while the TREE statement (Fig. 6.1) is resolved into a call to the NODE subroutine (ID 105) and two calls to the BRANCH library subroutine (ID 106). In this manner, all the GRIN2 statements are expanded into executable McGraph object code.

Fig.6.1

Simple GRIN2 Program

```
STMT   SOURCE STATEMENT

   1          ICTL     1,79,16
   2          PRINT    NOGEN
   3          G2IO     501
 179          LEAF     TRIG
 180                   *,*                  000001
 185          CURVE    ((W,0),(-W,W),(0,-W))
 190                   *,*                  000005
 209          LEAF     SQUARE
 210                   *,*                  000013
 218          CURVE    ((W,0),(0,W),(-W,0),(0,-W))
 219                   *,*                  000017
 243          TREE     DNOD2,,(,(200,400),TRIG,,SFIX),(,(800,400),SQUARE,,TFI
 244                   *,*                  000026
 252                   *,*                  000032
 266                   *,*                  000043
 280          NODE     DNOD1
 281                   *,*                  000054
 289          NODE     DNOD3
 290                   *,*                  000060
 298          BRANCH   BRCHL,DNOD1,(200,400),TRIG,,SFIX
 299                   *,*                  000064
 316          BRANCH   BRCHO,DNOD3,(800,400),SQUARE,,TFIX
 317                   *,*                  000075
 334          DSPNOD   DNOD2
 335                   *,*                  000106
 343 LOOP     WHICH    ,,P1
 345                   *,*                  000115
 351          GOTO     LOOP
 352                   *,*                  000122
 354 SFIX     NEWDSP   (BRCHO)
 356                   *,*                  000123
 365          GOTO     LOOP
 366                   *,*                  000133
 368 TFIX     NEWDSP   (BRCHL)
 370                   *,*                  000134
 379          GOTO     LOOP
 380                   *,*                  000144
 382 W        EQU      62
 383          GEND     ,
 387          END
```

FIG. 6.2

A DUMP OF THE GRIN STAND-ALONE ASSEMBLY FOLLOWS:

THE ID NUMBER FOR THIS JOB IS 00001

NUMBER OF INSTRUCTION WORDS EQUALS 131 (DECIMAL)

NUMBER OF RELOCATION WORDS EQUALS 15 (DECIMAL)

		0	1	2	3	4	5	6	7
0	00000	000222	030001	001010	600146R	600024	777775	030001	001030
1	00010	600147R	600151R	600153R	030001	001010	600155R	600024	777774
2	00020	030001	001030	600156R	600160R	600162R	600164R	030001	001050
3	00030	600166R	600024	030001	001060	600200	600146R	600167R	600146R
4	00040	600024	600127R	600024	030001	001050	600200	600146R	600171R
5	00050	600155R	600024	600134R	600024	030001	001050	600173R	600024
6	00060	030001	001050	600174R	600024	030001	001060	600175R	600177R
7	00070	600176R	600146R	600024	600123R	600024	030001	001060	600200R
10	00100	600174R	600201R	600155R	600024	600134R	600024	200166R	120401
11	00110	120223	120223	440000	010107	040126	120401	120131	120131
12	00120	600100	600140	600115R	030001	001760	600126	777777	030001
13	00130	001040	600200R	600126	600115R	030001	001760	600126	777777
14	00140	030001	001040	600175R	600126	600115R	000000	000000R	000076
15	00150	000000	777702	000076	000000	777702	000000R	000076	000000
16	00160	000000	000076	777702	000000	000000	777702	000000R	000310
17	00170	000620	001440	000620	000000R	000000R	000000R	000310	000620
20	00200	000000R	001440	000620					

FIGURE 6. 3

PARTIAL LIST OF SUBROUTINE LIBRARY

ID	NAME	ENTRY POINTS
101	LEAF	LEAF
103	VECTOR	INVECT, POINT, VECTO
104	ATTACH	ATTACH
105	NODE	NODE
106	BRANCH	BRANCH
110	DETACH	DETACH
176	CLRING	CLRING
220	WHICH	WHICH

ENTRY	NAME	ID	R. A.
ATTACH	ATTACH	104	1
BRANCH	BRANCH	106	1
CLRING	CLRING	176	1
DETACH	DETACH	110	1
INVECT	INVECT	103	2
LEAF	LEAF	101	1
NODE	NODE	105	1
POINT	POINT	103	4
VECTOR	VECTOR	103	1
WHICH	WHICH	220	1

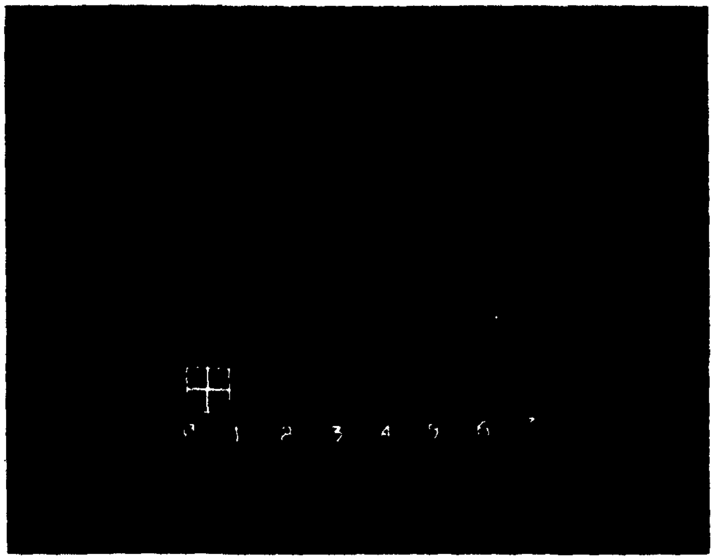
6.2 Loading and Initializing the McGraph System

Once a GRIN2 binary, loadable paper tape has been generated by means of the assembly system the program may be run on McGraph. First, though, the operating system must be loaded and initialized. Even before this can be done, however, the library must be loaded onto the auxiliary storage device. This can, of course, be done with the aid of the utility program G2LIBE (Section 7.2) but this is generally a long, tedious task. It is generally sufficient to do this once, and then to dump the contents of the disk onto a DEC-tape. At the beginning of each graphics session, no changes are required, the disk need only be loaded from DEC-tape. If the library needs editting, G2LIBE must be used in any event.

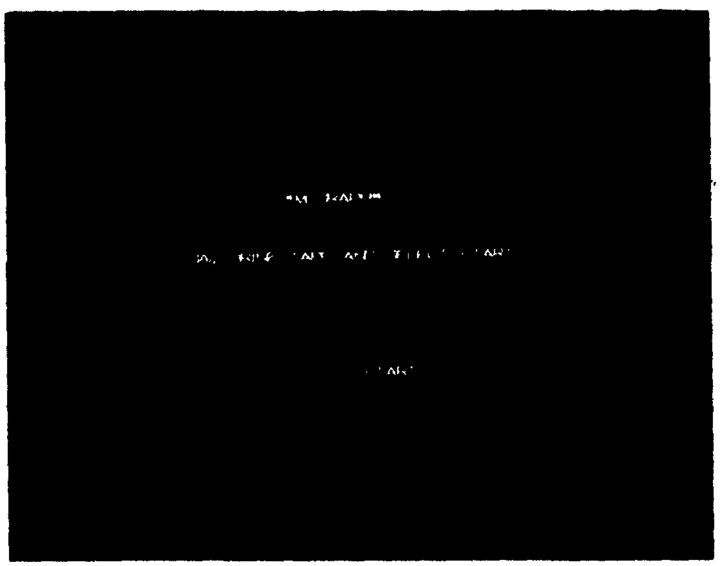
Once the program library has been loaded, and that part of the disk write protected, the executive programs must be loaded in from paper tape. Work is going on to make these loadable from DEC-tape, but this is not yet possible. Once the PDP-15 programs are loaded, the PDP-8 slave program must be loaded and initialized and a display track chosen. When the PDP-8 initialization is complete, the display shown in Fig. 6.4 (a) appears on the screen. The numbers shown are the simulated back-lighted pushbuttons, while the group of squares is the tracking cross. Although the pushbutton are effectively no longer part of the GRAPHIC2 simulation this initial display has not yet been changed to reflect this fact. After initialization the PDP-8 enters a slave mode of operation, continually scanning the link to the PDP-15 for further instructions, and watching the graphic peripherals for

Fig.6.4

System Displays



(a) Initial Display



(b) Load Program Tape

Fig.6.4 (Cont.)

(c) Missing Program Block

any change in status. At this point system loading is complete, and the PDP-15 core map is shown in Fig. 6.5.

To initialize the system it is necessary to execute the system bootstrap. This causes the interrupt system to be initialized, the display screen cleared and a clean directory to be copied into the system scratch area of the disk. A system message then appears on the display screen requesting that a program tape be loaded into the paper-tape reader (Fig. 6.4 (b)). At this point the initialization phase is now completed and the system is prepared to start reading and executing a user program. The core map at this point is shown in Fig. 6.6.

CORE MAPS

Fig.6.5 LOAD TIME

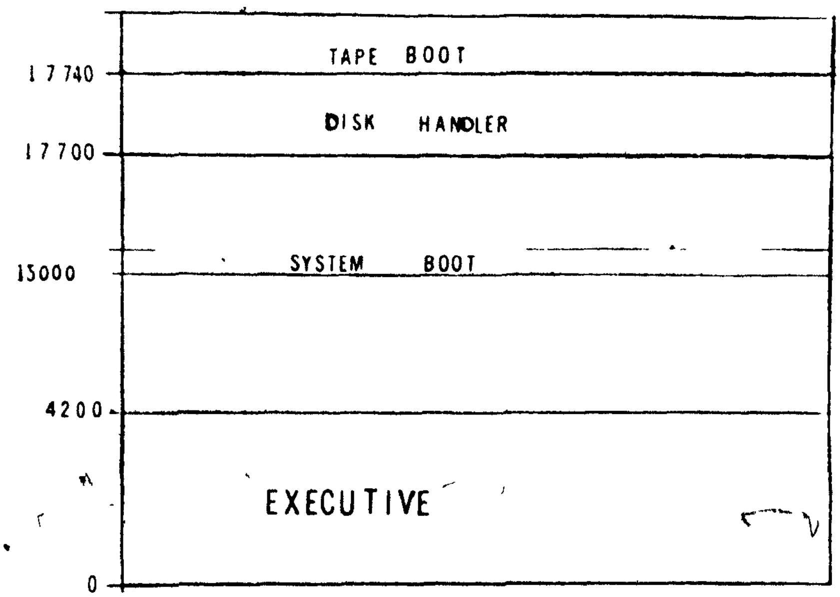
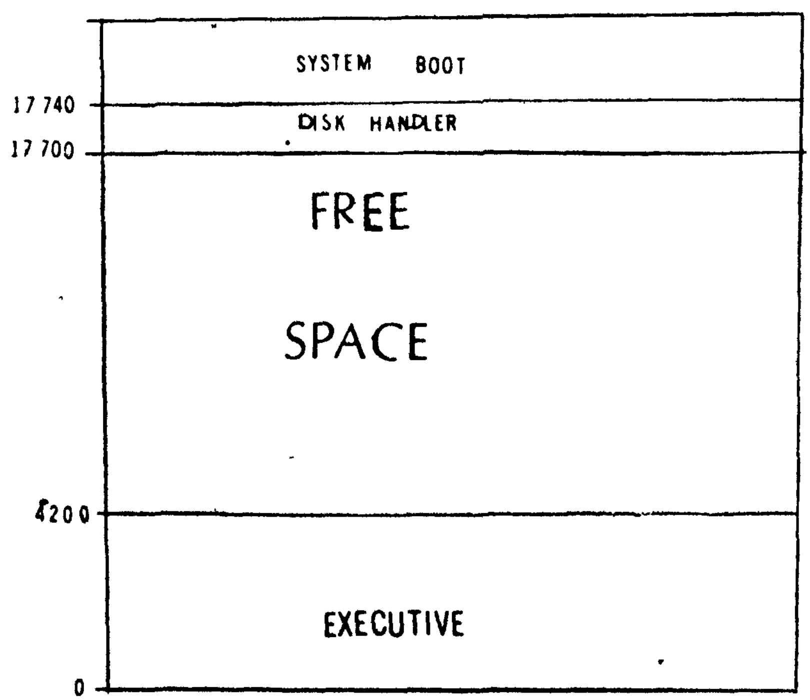


FIG.66 INITIALIZATION



6.3 Building The Data Structure Blocks

Once a program tape is loaded into the paper tape reader the user can ask the system to read the tape and start executing the program by making a light pen strike on the display screen. The tape is then read in and the subroutine block loaded in core, an entry placed in the block table, and a copy of the subroutine copied into the scratch area of disk. Execution of the program can then begin.

As was mentioned in Section 6.1, the first executable GRIN2 statement in the program is the LEAF declaration. This is translated into a call to the library subroutine LEAF, which generates an empty leaf block, as shown in Fig. 6.7 (a), gives this block the ID number Ø, and places a reference to it in the block table. Next, the CURVE statement is to be executed. This has been translated (Section 6.1) into a call to the VECTOR library subroutine, followed by a series of arguments. The VECTOR subroutine has the effect of placing display information, in the form of GRAPHIC2 scope code, into the last leaf block referenced, as in Fig. 6.7 (b).

Similarly, calls to the NODE subroutine (ID 105) cause node blocks to be created, with content specified by the argument list. Calls to the BRANCH subroutine cause branch blocks to be created, linking previously defined node blocks, and so on. Each time a graphic data structure block is created and given a distinctive ID, a new entry in the block table must also be generated. If, during an external reference, the desired block proves to be neither in core nor on disk, the picture shown in Fig. 6.3 (c) is displayed, and the

FIGURE 6. 7

LEAF BLOCK STRUCTURE

020001	HEADER
000000	DATA POINTER
777777	EOL TRAP

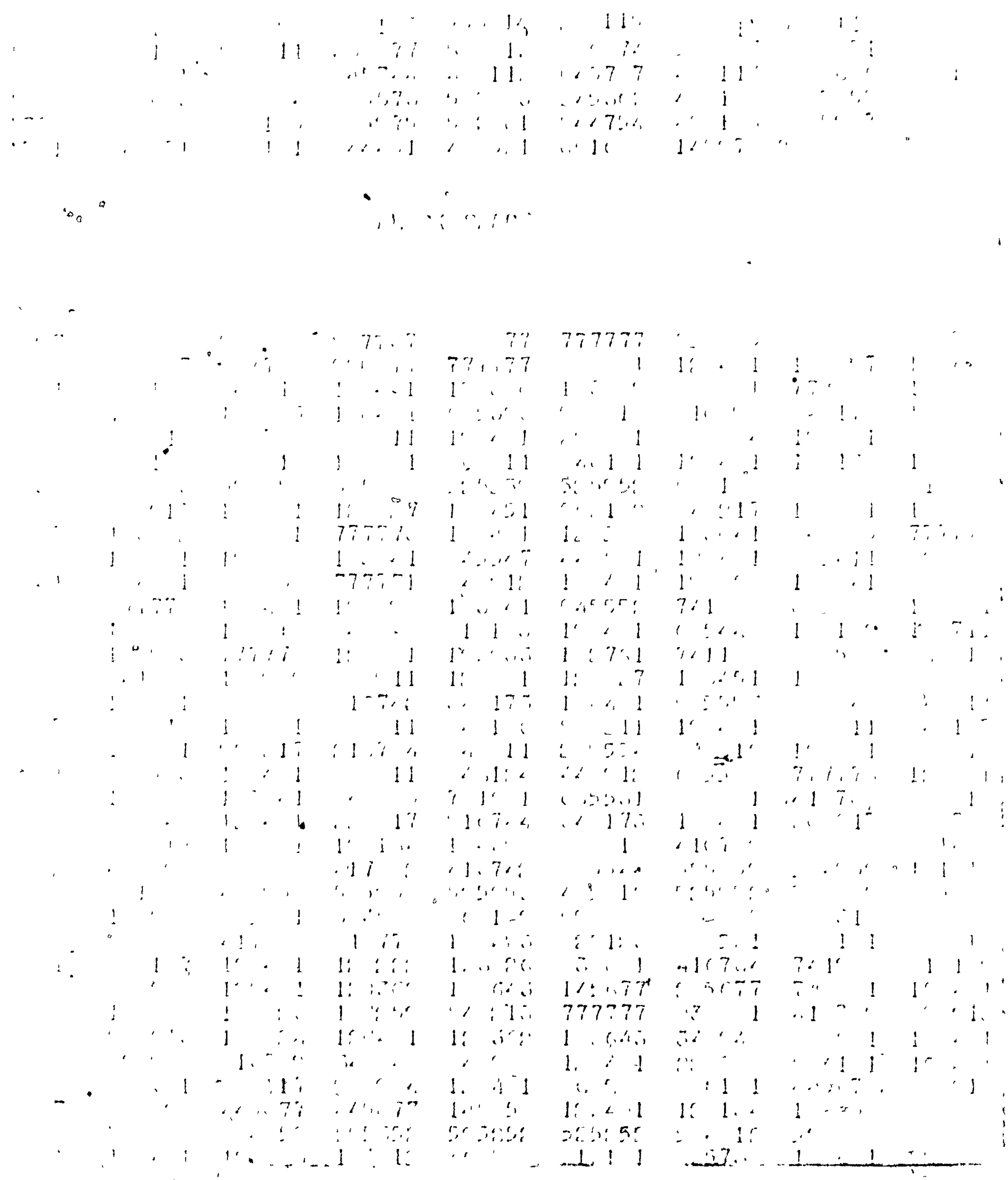
(a) Null Leaf Block

020004	HEADER
000000	DATA POINTER
223700	BODY-GRAPHIC2 CODING
227737	
220077	
777777	EOL TRAP

(b) Leaf Block, Containing Data

block must be read in by means of the paper tape reader. In this case the WHICH routine, ID 220, had been omitted. When all necessary blocks have been generated in the current example, the core dump is as shown in Fig. 6.8.

Fig.6.8



6.4 Real-Time I/O

No display information is sent to the PDP-8 until a GRIN2 I/O statement is executed. This class of statements include WHERE, WAIT, TYPOUT and, as in the case of this program, WHICH. These commands go through a data structure trace, sending display information to the PDP-8 as it is developed. The graphical devices are enabled, and some user interaction is expected.

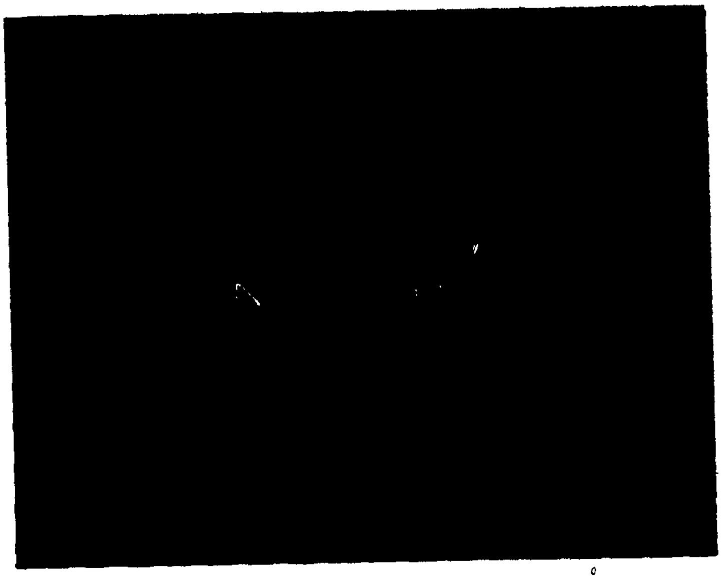
In the sample program the user is required to choose between the triangle and the square. The branches to each leaf have been declared to be light buttons. No matter which leaf is selected, then, command is transferred to an appropriate part of the program, dependent upon the choice.

Consider the data structure trace. One node is designated as the display node and is considered to be the root of the display. All information below it is displayed, any above it ignored. The system of pointers is followed until a leaf is encountered, at which point control is relinquished to the translation routine G2TRAN. The GRAPHIC2 scope code in the leaves is then translated into McGraph code and sent to the PDP-8 for display. Upon an end-of-leaf trap, control is returned to the structure tracing routine. This process continues until the entire display is visible. Note that each time a leaf appears, its contents must be translated into valid McGraph code. This is tremendously inefficient but it works, which was the initial prime consideration. In this program the problem is never encountered due to the simplicity of the display, but in larger systems it can be a definite draw-back.

Photos showing a sequence of events appear in Fig. 6.9. In the first frame, both the triangle and square are visible. After the user makes a light pen hit on the square, he is left with the situation shown in Fig. 6.9 (b): only the triangle remains visible. A subsequent light pen hit on the triangle causes it to disappear and the square to reappear, as in Fig. 6.9 (c). These last two displays alternate indefinitely, upon light pen hits, until the system is stopped and reinitialized by the operator.

Clearly this is a trivial example, but it displays, hopefully, in a simple way, how the system operates. One display generated during an interactive run is shown in Fig. 6.10. It stems from a test program, known as CITY. This program is used to test the WHICH subroutine and the graphic structure building routines. Output from it is shown since there was no time to write and debug anything more than test programs of one sort or another.

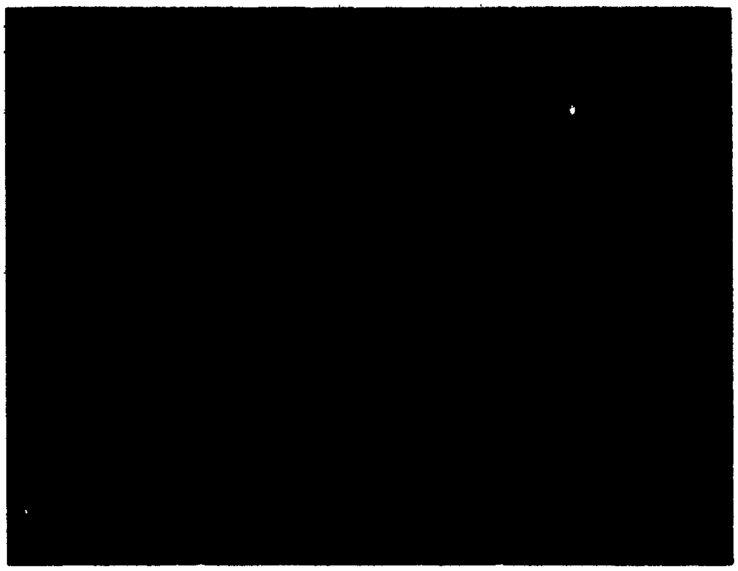
Fig.6.9
Sequence of Displays



(a) Display 1

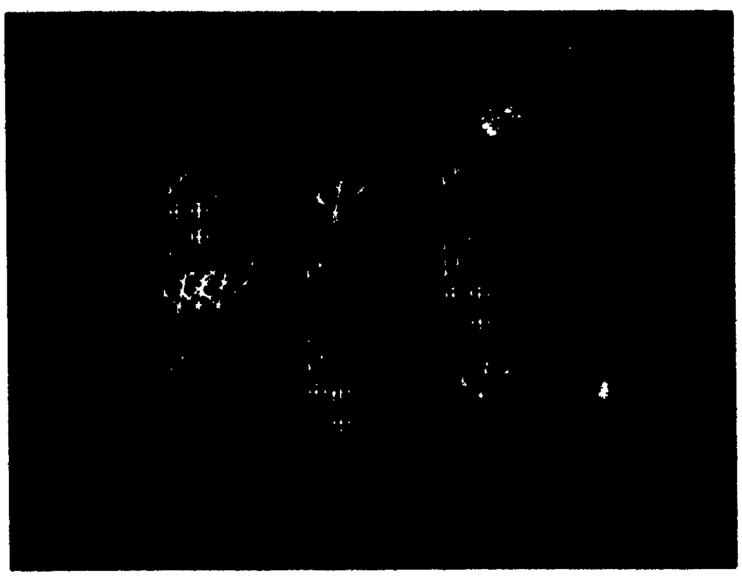
(b) Display 2

Fig.6.9(Cont.)



(c) Display 3

Fig.6.10



CITY Program

CHAPTER 7 ASSOCIATED PROCESSES

7.1 The GRIN2 Assembly System

There is more to the McGraph operating system than the executive programs and subroutine library. Most notable among the associated systems are the GRIN2 assembly system, the library editor G2LIBE, and a set of debug routines.

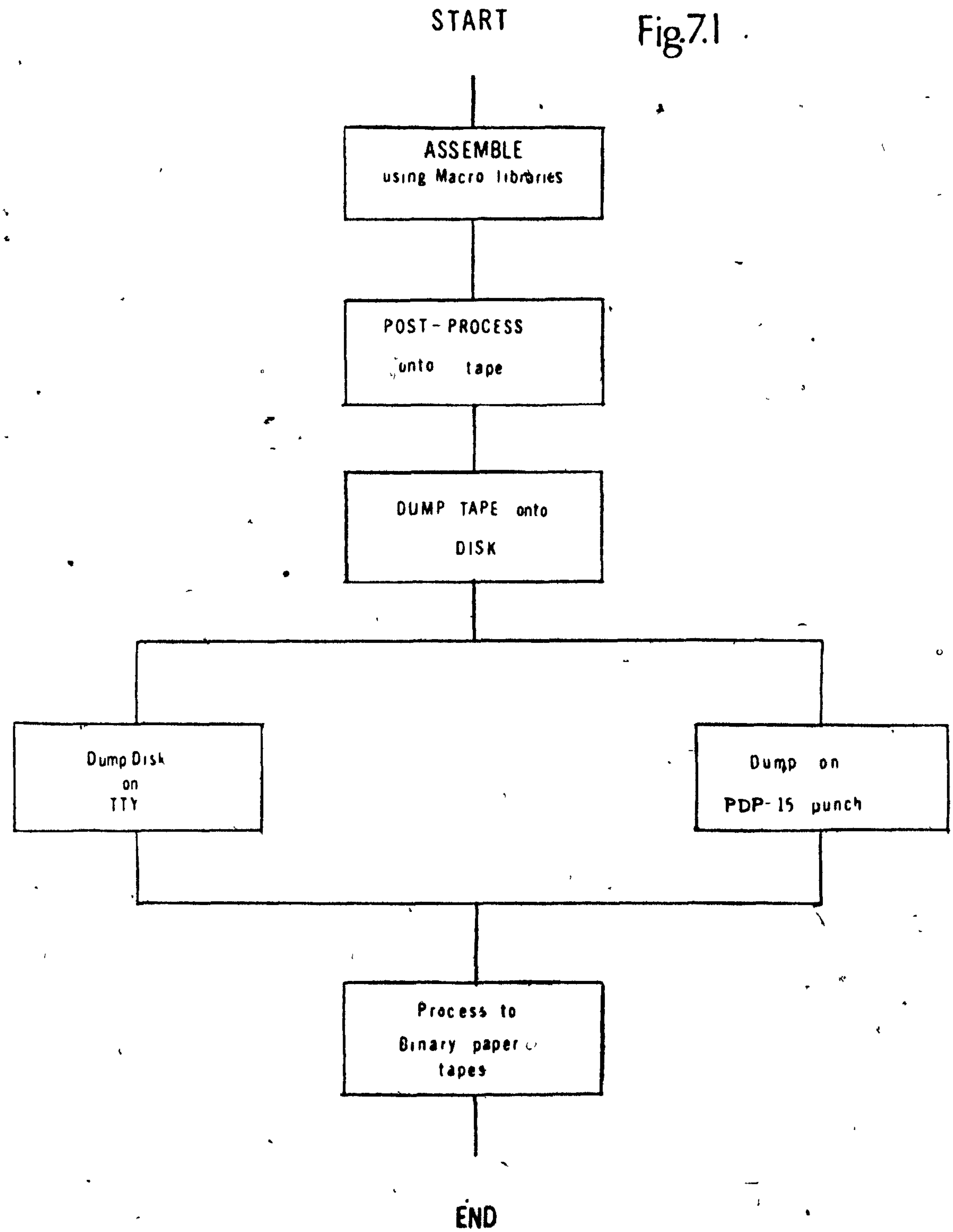
The GRIN2 assembly system is as described by McNeil(M. 4); no major changes were made to it during this project. In essence the GRIN2 language is defined by a set of macros in the IBM OS-Assembler language, so that the process of assembling a GRIN2 program reduces to that of expanding and interpretting the program in terms of these macros. Two macro packages exist: one for absolute PDP-9 assemblies (i.e. the system generation package A. EE25. P9 MACROS and the other for relocatable GRIN2 programs (A. EE25. GRIN2). The output of this assembler is recorded on punch cards. A post-processor program is then invoked to translate the information stored on punched cards into correct PDP-9 loadable block format, and to dump this information onto magnetic tape.

The output stored on this tape is processed into loadable paper tapes required by the McGraph operating system, as described in Appendix B.

This process is depicted in Fig. 7-1.

GRIN2 Assembly System

Fig.7.1



7.2 The Library Editor G2LIBE

The library editor G2LIBE is a utility program inherited from the Bellgraph system. This is not a real-time program, nor does it require the operation of any graphical devices. Accordingly, very few changes were required to make it work.

The editor can be used for many purposes. First, G2LIBE can be used to load GRIN2 program blocks from paper tape onto auxiliary storage. These disk files can then be dumped onto a teletype, octal patched, renamed, or purged altogether from mass storage.

The editor is not an essential part of the McGraph operating system. It can, however, facilitate the running and debugging of graphics programs, and as such has a definite value.

7.3 Debug Packages

Two debug packages are associated with McGraph: the primitive Octal Debug Package, and the more elaborate utility, G2BUG. These are designed for two separate applications.

The utility G2BUG is actually a well-debugged GRIN2 program, which is used to help debug other GRIN2 programs. An entire range of advanced operations is possible with G2BUG, including imbedding break-points in the program, single-stepping, and so on. This utility has not yet been worked on at McGill, since it can only be used in the debugging of application programs. It assumes a working operating system, and so is of limited use in debugging the operating system.

The Octal Debugging Package, on the other hand is used in the debugging of the operating system. It assumes nothing and does very little! It can be used to obtain core dumps, on the TTY patch parts of core, patch the registers, start execution at a given place in core, or make a subroutine jump to a given core location.

This rather limited instruction set puts a real load on the user, who must be totally familiar with the machine's instruction set in order to use it. It is, however, a great improvement over manually toggling in changes.

SECTION C - CONCLUSIONS

CHAPTER 8 EVALUATION OF THE McGRAPH SYSTEM

A computer system can best be judged by how well it meets user requirements, and by how easily it can be adapted to meet changing demands. In this chapter the McGraph operating system will be evaluated on the basis of programming and debugging ease, run-time flexibility, and its real-time characteristics. A discussion of the techniques used in system modification and expansion will be postponed until the next chapter.

8.1 Programming in GRIN2

Experience with the GRIN2 language has been quite positive, both at McGill and at BTL [C.1]. The existence of a graphical data structure enables complicated, iterative designs to be expressed is a compact, logical and readable form. And, of course, legible programs are maintainable programs. GRIN2 programs may easily be segmented into linked blocks of the type recommended by disciples of the structured programming concept [M.5], a rather nice side-effect.

Over the course of the years there have been many proponents and detractors of fixed data structures in graphics applications. It is clear that a graphic data structure is not essential to an effective language; it does facilitate operations such as duplication of subpictures, associating data with graphic information, and moving pictures or subpictures wholesale. It also has the added effect of forcing the user to think about the existence of structure in his display, frequently a useful task. It is no accident, therefore, that the GRIN2 experiment has left "... in-house users apparently satisfied and some outsiders dissatisfied, with such features as a fixed (ring) data structure, an unwieldy programming language, not enough local core, etc." (V.1).

All GRIN2 programs also have the added benefit of being, to a large degree, "idiot-proof" (W.1). The user cannot easily bomb the system by making an incorrect response at any point, because the language forces the programmer to think of these possibilities. Left to their own devices, "application programmers almost never consider the consequences of a system failure in the middle of processing their programs" (Y.2), or indeed" anything out of the ordinary happening" (W.1). GRIN2 forces thought about these things, and permits recovery in case of error. It has, then, some of the characteristics of a good interactive language, and a good operating system.

On the other hand, most modern graphical languages have provisions for interfacing to a high-level algorithmic language, a provision sadly lacking in GRIN2. For applications requiring on-line scientific computation, the present implementation is entirely inadequate.

8.2 GRIN2 Assembly System

The current assembly system (M.4) is unacceptably slow and costly. It has been shown that the GRIN2 assembler could reside on mass storage on the PDP-15 (MAN-1) in a much simplified form. Indeed, the GRAPHIC2 terminals have now progressed further around the "wheel of reincarnation" and have all been given disks, containing the resident assembler (V. 1). The interface to the main computer has been abandoned and the BTL system, like the McGraph system, is functioning entirely in stand-alone mode.

If GRIN2 is ever to be used to full advantage at McGill the same project must be undertaken. The assembler program is really just another GRIN2 program, functioning under the McGraph monitor, which accepts symbolic input from a disk file, assembles it and places object code on a named file on disk. This program was also donated to McGill by BTL, but could not, of course, be run until the operating system was working. The time has now come when it can be tested.

8.3 Run-Time Characteristics

At present the graphics hardware is operating at somewhat less than peak efficiency, chiefly due to the numerous patches required in order to make McGraph operative. As was noted earlier, the GRAPHIC2 IOT simulator and scope-code translator (M. 4) is enormously inefficient. The PDP-8 slave system is similarly burdened with simulation tasks, and so further slows down the response time. The result is that the system has rather sluggish characteristics for a high-powered system. Some solutions to this problem will be proposed in the succeeding chapter.

A light-pen interrupt is immediately serviced-the PDP-15 is informed of the situation within 5μsec. The link is then sealed out for half a second to prevent multiple hits (the PDP-8 cannot be run in the interrupt mode due to DISC interface problems). Meanwhile, processing continues on the PDP-15, and within a second of the interrupt, activity can be seen on the screen. Each leaf is sent individually from the PDP-15 to the PDP-8, and written on the DISC. Clearly, then, only one leaf can be written at each rotation (1/30 second). Complex pictures can easily take five or six seconds to display, then. Work can obviously be done to speed this process up.

With a few notable exceptions, all the run-time operations available under the Bellgraph operating system have been maintained. First, since it was not essential to the operation of the system, the GRAPHIC2 console keyboard has not been simulated (McNeil). All typed operator responses must be made via the PDP-15 console keyboard. Second, the backlighted pushbuttons have ceased to be simulated, although they still appear, at system load time, as a set of numbers on the face of the display.

Finally, the class of "real-time" programs has become obsolete. In these programs it was assumed that core-based refresh was taking place, and that, therefore, one could change the display in real-time by changing strategic core locations. A disc-refreshed system effectively eliminates this possibility, and make the entire strategy obsolete.

Apart from these changes the Bellgraph Programmer's Manual can be applied directly.

8.4 Debugging

Although much of this project reduced to executive program debugging, very little experience with applications program debugging was obtained. These processes are, of course, very different. It was immediately found that debugging in large systems was no easy metter. Standard approaches such as using the Dynamic Debugging Technique (DDT), a DEC-supplied utility could not be applied, since these programs competed with the executive program for control of the same core locations. Nor could one use the Bellgraph utility G2BUG, since it ran under the operating system which was to be examined. Eventually it was found that the only available package which could be applied to the problem was the primitive Octal Debugging Package, described in Chapter 7.

Octal debugging is demanding, and the user must have a good grasp of his machine and of the process he wishes to examine. This package cannot produce corrected paper tape, but merely patch core. From this comes the restriction that the modified code must not write over other, valid, code. Deletions are simple; additions can be rather complex.

Some debugging information has been built into McGraph. When a program error is encountered the computer rings the bell on the teletype and traps to an error-handling routine. The user is then provided with the error number, the location where it was trapped, the ID number of the block in which it was detected, and its relative address within the block. This information, taken together with a block-table dump, can usually help the user to detect his error.

The utility program G2BUG is a GRIN2 program, which should run under McGraph. This program can be used to aid the debugging of application programs through a series of high-level commands. This has not yet been tested at McGill, but will become necessary if this system is pursued.

8.5 Run-Time Flexibility

A strong point of the McGraph system is its run-time flexibility. Because of the graphic data structure, sub-pictures can easily be duplicated, shifted, rotated, scaled or deleted. Structures can be built and tested on-line in a natural way. Application programs can even be written permitting leaves to be built on-line from user-supplied information.

The true strength of GRIN2 lies in the ease with which complicated structures can be handled. Used wisely, the system can be made to enhance rather than constrain human creativity. Run-time errors will not normally bomb the system. Nor will a system failure cause the loss of all the data base built up; copies of most of the files exist on disk as well as in core. The user, then, is given a great deal of freedom, in return for some additional sweat by the application programmers.

CHAPTER 9 SYSTEM EXTENSION AND MODIFICATION

The current McGraph operating system cannot be regarded as a finished product. A flexible system has been provided which is capable of supporting applications programs written in a high-level graphics-oriented language, GRIN2. It would, however, be unrealistic to claim that the McGraph peripherals are operating in an efficient manner. Much still remains to be done in order to achieve optimal system performance. This chapter outlines some possible improvements which could be implemented.

Consider what must be done. First, the GRAPHIC2 simulator must be eliminated from the McGraph software. Second, the system should be extended to include other graphical devices available at McGill. Third, the GRIN2 assembler could be made to reside on the PDP-15, and, if necessary, an interface to a high-level, algorithmic language (i.e. FORTRAN) could be established. These goals will be studied, in that order.

9.1 The Device Translation Subsystem

Logically, the GRAPHIC2 simulator consists of two distinct parts. One part, the IOT simulator, translates GRAPHIC2 IOT commands into McGraph IOT's and McGraph scope code. The other part, the scope code translator changes GRAPHIC2 scope code into appropriate sequences of McGraph scope code. Each part of the simulator can be changed (or eliminated) independently of the other. One can, then, speak of modifying the device translation subsystem, the PDP-8 slave subsystem and the real-time subsystem (which, of course, includes the IOT simulator), rather than of eliminating the GRAPHIC2 simulation.

Consider first the device translation subsystem. Until now, no effect has been made to make this system efficient so it is, naturally, massively inefficient. Display information can exist in several forms in the system. First, alphanumeric information can exist as part of a leaf, or as a system message. Second, branch blocks contain initialization information about a leaf (Fig. 2.1): a current parameter word, and a pair of deflection words. And third, visible display information is stored in leaf blocks in the form of GRAPHIC2 scope coding.

This third class of information is most easily dealt with. All the GRIN2 language statements which generate visible display information inevitably rely upon the VECTOR or PARAM library subroutines. By rewriting these routines for each display unit used it is possible to take a giant step towards generalizing the device translator subsystem. One minor point should be noted here:

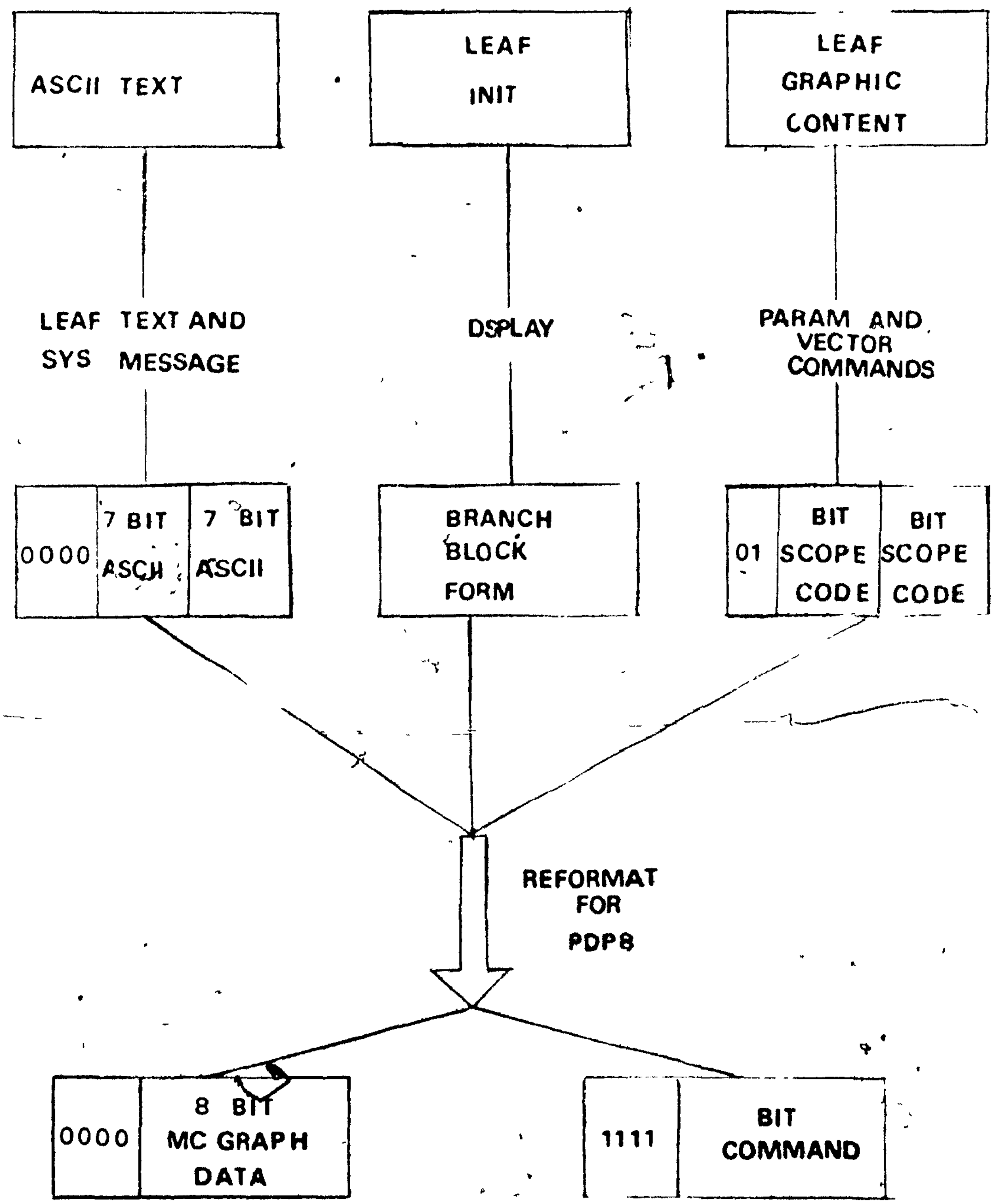
it is possible to pack two 8-bit McGraph words into one PDP-15 word in order to save on storage.

Branch blocks contain some display initialization information, as shown in Fig. 2.1. This is examined and translated into display code by the core-resident executive program DSPLAY. Under this philosophy, then, DSPLAY must be rewritten and the executive system regenerated for each device used. Alternatively, some part of the external GRAPHIC2 simulator must be maintained to handle these blocks, and must be rewritten for each device used.

Alphanumeric information causes more problems in McGraph. It was assumed, in Bellgraph, that an explicit character-generator scope code was available. This assumption, although deeply imbedded in the operating system, is invalid in the McGraph environment. No hardware character-generator is available here. To maintain current text-handling routines and still eliminate a scope code translator requires a change in file transmission methods. There would be two distinct classes of information in a leaf: display unit instructions, packed two to a PDP-15 word, and character information, stored as two seven-bit ASCII codes to a PDP-15 word. By appropriately marking the higher-order bits of these words one can differentiate between them (Fig. 9.1). Then, as the words are transmitted to the PDP-15 they can be analyzed and the character words expanded into McGraph scope code.

FIG.9.1

Graphical I/O Requirements



9.2 Modifications to the Real-Time Subsystem

The real-time subsystem could be signicantly more efficient by eliminating the GRAPHICS2-IOT simulation package. One way that this could be done is to redefine the macros associated with these IOT's. Commands refering to equipment no longer used (e.g. pushbuttons) would cease to generate PDP-9 code, while the useful IOT's would be translated into meaningful PDP-15 code. The immediate result would be to straighten long, winding subroutine paths into a sequence of commands directly imbedded into the program flow. This would cut execution time, probably at the cost of increased storage requirements in the first bank of core.

Similarly, the real-time subroutines WHICH and WHERE could be rewritten to take advantage of McGraph facilities. This would have similar advantages and penalties associated with it. Each of these steps is a partial answer to the problem of developing a shorter response time. Each is independent. And neither, alone, would totally eliminate the body of coding which is currently known as the GRAPHICS2 IOT simulator, G2SIM.

9.3 The PDP-8 Slave Program

An effort has already been made to simplify the PDP-8 slave program, as shown in Chapter 5. The GRAPHIC2 keyboard, the pushbuttons, and the blink option all have been dropped from the slave program. Only display code buffering, light-pen handling and tracking-cross manipulation remain as tasks to be done on the PDP-8. Of these, the tracking-cross manipulation routine (M. 4) must be improved to eliminate jumping, and the buffering technique could be improved to speed display generation.

Briefly, graphics commands are transmitted to, and buffered at, the PDP-8 as they are generated. The buffer is dumped onto the disk when it is full, or at the end of a leaf. Response time could be significantly shortened if the buffer was dumped when full, or at the end of a picture. In actual fact, this demands more of a change to the display routine DSPLAY than it does to the PDP-8 slave program.

9.4 Extension to Other Devices

The McGraph operating system will ultimately prove to be of value only if it is (easily) extensible to other graphics devices. A few remarks should, then, be made about this point.

First, some devices are more easily integrated into the system than others. Currently, a display unit with the following characteristics would be favoured:

(1) It is directly interfaced to, and capable of interrupting, the PDP-15

(2) It uses 18-bit words, or less

(3) It has a hardware character generator

(4) It has single-word parameter commands

(5) It is capable of returning X and Y co-ordinates upon a display interrupt (light pen or joy stick)

Probably the next device to be used in McGraph would be the TEKTRONIX 4002 display scope, already interfaced to the PDP-15. Extension to this device should be relatively easy, since it fulfils most of the conditions listed above. It is, however, a storage unit, so the real-time interrupt routines would have to be rethought. But less rethinking will be necessary than for interfacing the original McGraph display unit.

9.5 Odds and Ends

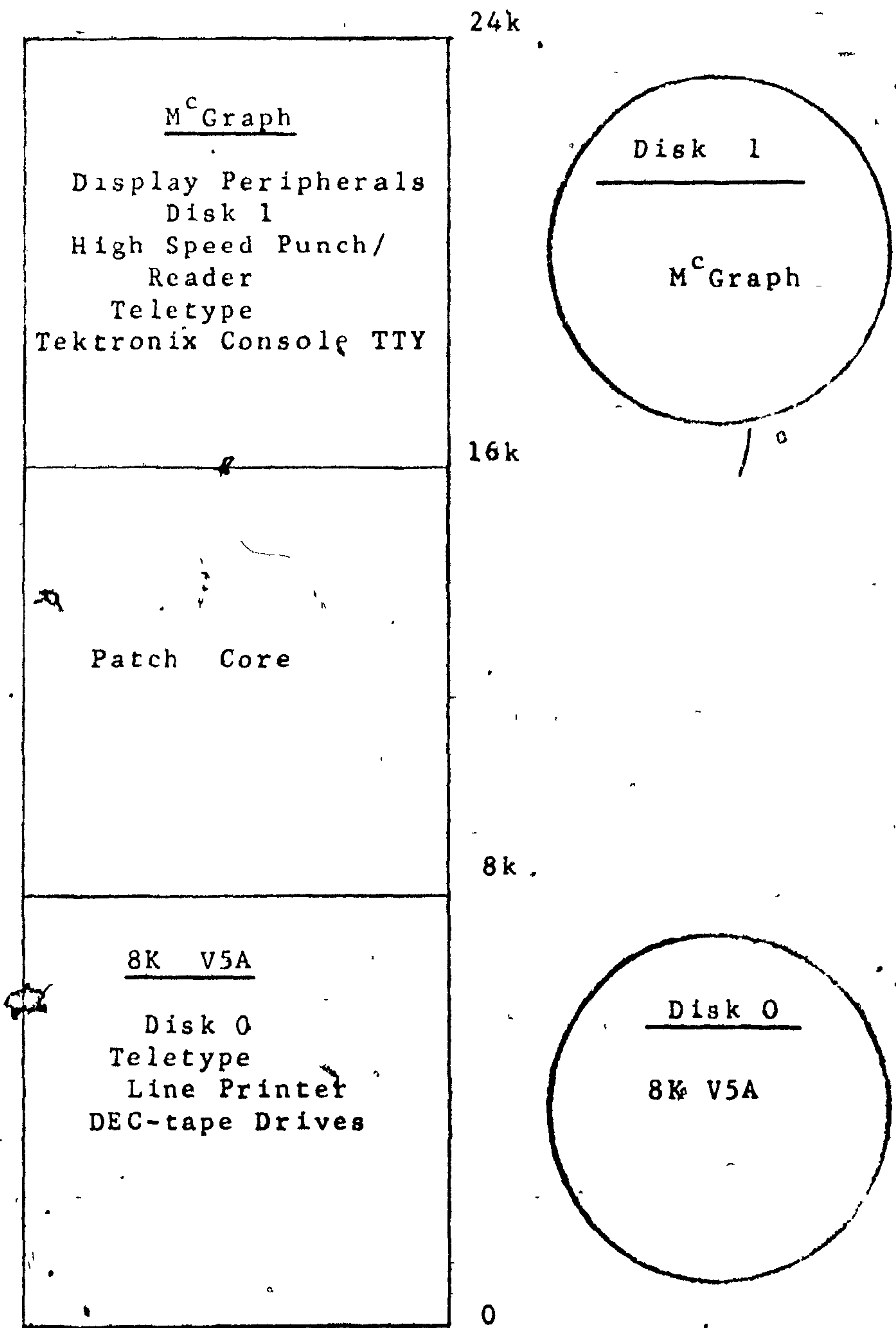
Further goals, which could be set if demand is sufficient, must include transferring the GRIN assembler to the PDP-15. This could greatly simplify the current assembly process. Bell Telephone Labs included the appropriate software to do this in the package donated to McGill. It, however, presupposed a working operating system, and so could not be tested until now. Assembling this program on the IBM 360 and placing it in the subroutine library should not be too difficult a project, and could provide a major improvement in service.

Another possibility that could be investigated is that of interfacing the McGraph operating system to standard DEC systems. The most promising arrangement is shown in Fig. 9-2. Here, an 8K version of V5A* has been generated, using one disk and residing in the lower 8K of core. McGraph can be modified to sit in the top 8K, and claims the other disk. Now each system controls independent peripherals and stays in entirely different banks of core. By allowing communication between banks to take place only through the use of a given assembler routine controlled communication could be set up between the systems. The computational power of FORTRAN would then be available to the GRIN2 programmer. This is clear a nontrivial, if worthy, project.

* Digital Equipment Corporation Operating System for the PDP - 15 computer.

FIG.9.2

PROPOSED INTERFACE BETWEEN McGRAPH & V5A



CHAPTER 10 CONCLUSIONS

This project was a continuation of the effort [M. 4] to provide a graphics language for the users of the McGraph dis-oriented graphics display facility [M-1, F. 1]. In order to achieve this goal, an existing graphics-oriented operating system [C. 1] had to be significantly modified and debugged. Most of the meaningful differences between McGraph and its predecessor, Bellgraph, lie in their respective real-time subsystems. Since the structure of the McGraph facility is far different from that of the GRAPHIC2 terminal, much of the real-time interaction philosophy had to be rethought. Due to time and manpower limitations, McGraph does not work as efficiently as possible, at this time. Further system work, in the area indicated, should make McGraph a very respectable system.

It can be honestly said now that GRIN2 is running on the McGraph facility. Application packages can be written to fully test the strengths and weaknesses of the system, while modifications can be made with confidence to the operating system. There is nothing to lose any more; there exists a working system to fall back upon. In this manner, then, McGraph can be developed and adapted to fit the special needs of users at McGill.

APPENDIX A - THE GRIN2 ASSEMBLY SYSTEM

A. 1. Preprocessing A GRIN2 Program (or PDP-9 Program)

This job must be run in 300K of core on the IBM 360/75. The input sourcedeck may contain any number of individual GRIN or PDP-9 programs. Output will be on punched cards.

```
// PREPRO     EXEC    PGM = PRE, PARM = 'DECK'
// STEPLIB    DD      DSNAME = A. EE25. PREPRO, DISP =
                      (OLD, KEEP)
// SYSPRINT   DD      SYSOUT = A
// SYSPUNCH   DD      SYSOUT = B, DCB = (RECFM = FB,
                      LRECL = 80, BLKSIZE = 3200)
// PREIN      DD      UNIT = ONLN, SPACE = (7280, (35, 4)),
                      DISP = (NEW, DELETE), DCB =
                      (RECFM = FB, LRECL = 80, BLKSIZE =
                      7280)
// ASMIN      DD      UNIT = ONLN, SPACE = (7280, (35, 4)),
                      DISP = (NEW, PASS), DCB = (RECFM =
                      FB, LRECL = 80, BLKSIZE = 7280)
// SYSIN      DD      *

               GRIN2  SOURCE DECK

/*
//
```

A. 2 Assembling A GRIN2 Program

This job may be run in 100K of core in most cases. Output will be punched object decks. The input source deck may contain any number of programs, since they are run in batch mode.

```
// ASM            EXEC    ASMGC, PARM. ASM = 'NOLOAD,
                          DECK, BATCH'
// ASM. SYSLIB    DD      DSNAME = A. EE25. GRIN2, DISP =
                          OLD
// SYSIN          DD      *

                  GRIN2  SOURCE DECKS

/*
//
```

A PDP-9 program can be assembled in the same manner if the library card is changed to

```
// ASM. SYSLIB    DD      DSNAME = A. EE25. P9MACROS,
                          DISP = OLD
```

A. 3 Post-Processing GRIN2/PDP9 Programs

Input is assumed to be in the form of punched object decks, while the resulting output appears on 9-track, 800 BPI magnetic tape. The input stream may consist of any number of individual GRIN2 programs, followed by any number of PDP-9 program blocks. The post-processor generates one block for each GRIN2 program and a load-module containing all the PDP-9 programs.

```
//            EXEC    SETUP, PARM = 'T8 = PDP9 (RING
                      IN, NL, SLOT E43)
// POST       EXEC    PGM = POST, PARM = ' options, as below'
// STEPLIB    DD      DSNAME = A. EE25. G2POST, DISP =
                      (OLD, KEEP)
// SYSPRINT   DD      SYSOUT = A
// MTAPE      DD      VOL = SER = PDP9, LABEL = (, NL),
                      UNIT = TAPE8, DCB = (RECFM = F,
                      BLKSIZE = 300, LRECL = 300, DEN = 2)
// PUNCH      DD      SYSOUT = B, DCB = (RECFM = F,
                      BLKSIZE = 1600, LRECL = 80)
// SYSIN      DD      *

              PDP-9 & GRIN2 OBJECT DECKS

/*
//
```

OPTIONS:

ORG = nnnn specifies octal starting address of absolute load modules. Ignored on input of GRIN programs.

LIST produces a listing of PDP-9 mnemonics and contents of each word.

NODUMP suppresses the printing of each word in the program. The dump cannot be suppressed on PDP-9 absolute load modules.

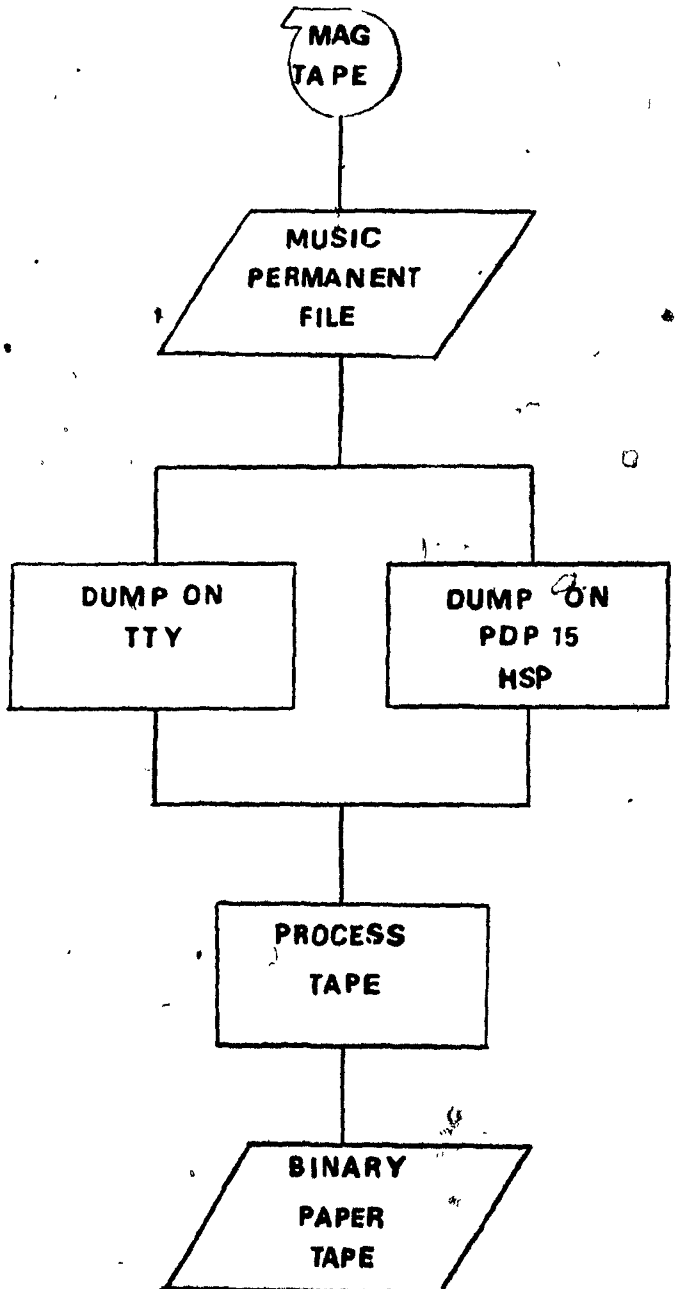
APPENDIX B - GENERATION OF GRIN2 PAPER TAPES

The output of the GRIN2 post-processor consists in part of blocks of ASCII code, dumped on a 9-track, 800BPI magnetic tape. The McGraph facility, however, lacks a magnetic tape drive, or, indeed, any other OS- compatible medium. The solution currently being used is shown in Fig. B.1.

First, the mag-tape is dumped onto a MUSIC permanent disk file. This file can subsequently be dumped onto paper tape from a teletype or from the PDP-15. Either way an ASCII tape is produced. This can finally be processed on the PDP-15, resulting in a loadable binary tape.

The exact procedure required is shown below.

FIG. B1


MAG TAPE
MUSIC PERMANENT FILE
DUMP ON TTY
DUMP ON PDP 15 HSP
PROCESS TAPE
BINARY PAPER TAPE

B. 1 Dumping Tape Onto Music File

This must be run as a batch job on the MUSIC system. File consists of blocks of 5 records. Each record consists of 60 characters (10 PDP-15 words).

```
/ ID
/ PAUSE         TAPEJOB, VOL = PDP9, SLOT = E43
/ FILE          DISK = (1, EE25 POST), VOL = RAXOO6,
                DISP = OLD
/ FILE          TAPE = (2, BLK = 300), RSIZ = 60, VOL =
                PDP9, DISP = SHR
/ INCLUDE       UTIL
$ OUTPUT = 1
$ INPUT  = 2
/ ENDRUN
```

B. 2 Dumping Disk from Teletype

SIGN ONTO MUSIC SYSTEM, THEN

```
/ INPUT
/ FILE DISK = (1, EE25 POST), VOL = RAXOO6
/ INCLUDE UTIL
$ INPUT  =  1
/ ENDRUN
```

~AT WHICH POINT THE USER MUST TURN ON THE TTY PAPER PUNCH

B. 3 Developing Binary GRIN2 Tapes

1. Load ASCII tape, output from the previous stage in PDP-15 high speed reader, at start of first GRIN/PDP9 block.

2. Execute Program 'PROG1' on PDP-15 (Tape 162)

3. ASCII tape will read in and punch start; System will pause after first block. Make sure tape is clear on reader, and type ↑P on teletype. Process repeats until tape is completed.

BIBLIOGRAPHY

C. 1 Christensen, C. and E. N. Pinson, "Multi-Function Graphics for a large Computer System", Proc. AFIPS Fall Joint Computer Conf., Vol. 31, pp. 697-771, 1967.

C. 2 Cuttle, G. and P. B. Robinson (ed.), "Executive Programs and Operating Systems", American Elsevier Inc., New York, N. Y., 1971.

D. 1 Dawdy, G. M., "An I-O Programming System for Small Computers", Proc. of the Canadian Computer Conf., pp. 321301-321313, 1972.

D. 2 Dudley, T. K., "Xerox Computer Graphics", Proc. of the Third Man-Computer Communications Seminar, Ottawa, pp. 13.1-13.5, 1973.

D. 3 Dunn, R. M., "Graphics, Problem Solving and Virtual Systems", Proc. AFIPS National Computer Conf., Vol. 42, pp. 23-30, 1973.

F. 1 Fabi, R. J., "The Design and Construction of a Disk-Oriented Graphics System", Dept. of Elect. Eng., McGill Univ., 1971.

F. 2 Foley, J. D. and V. L. Wallace, "The Art of Natural Man-Machine Conversation", Proc. of the IEEE, Vol. 62, pp. 462-471, April, 1974.

F. 3 Franklin, J. and E. B. Dean, "Interactive Graphics for Computer Aided Network Design," Proc. AFIPS National Computer Conf., Vol. 42, pp. 677-683, 1973.

G. 1 Gammill, R. C., "Graphics and Interactive Systems - Design Considerations of a Software System", Proc. AFIPS National Computer Conf., Vol. 42, pp. 657-663, 1973.

G. 2 George, I. R. L., "Software and Data Structures for Graphical C. A. D.," Proc. of the International Conf. on Computer Aided Design, Southampton, U. K., 1969.

G. 3 Groner, G. F., "Display Terminals Can Help People Use Computers", Proc. AFIPS National Computer Conf., Vol. 42, pp. M39-M42, 1973.

G. 4 Gwynn, J. W., "CRT Terminal Access From High Level Languages", SID Digest of Technical Papers, pp. 46-47, 1972.

L. 1 Lecarme, O., "A System for Interactive Graphic Programming", Proc. IFIP Congress, Vol. 1, pp. 440-444, 1971.

L. 2 Link, R. W. and S. S. Yau, "A Simplified Data Structure for Interactive Graphics on a Small Computer", Proc. Canadian Computer Conf., Montreal, pp. 413401-413422, 1972.

M. 1 Malowany, A., M. D. Levine, et al., "A Disc-Oriented Graphics Display System", Proc. Second Man-Computer Communications Seminar, Ottawa, 1968.

M. 2 Malowany, A., K. C. Campbell, T. McNeil, "The McGraph System", Proc. Third Man-Computer Communications Seminar, Ottawa, pp. 30.1-30.5, 1973.

M. 3 McLean, R. S. and W. P. Olivier, "Initial Use of a Plasma Display Panel", Proc. Third Man-Computer Communications Seminar, Ottawa, pp. 33.1-33.7, 1973.

M. 4 McNeil, T. O'B., "A General-Purpose Graphics System for a Small Computor", M. Eng. Thesis, Dept. of Elect. Eng., McGill Univ., 1974.

M. 5 Moyle, F. J., "Graphical Support Software GTA for PDP-9-339 System", TR-EE-71-6, Purdue Univ., 1971.

N. 1 Needham, R. M. and D. F. Hartley, "Theory and Practice in Operating System Design", Proc. Second Symposium on Operating System Principles, A. C. M., Princeton, N. J., 1969.

N. 2 Newman, W. M. and R. F. Sproull, "An Approach to Graphics System Design", Proc. of the IEEE, Vol. 62, pp. 471-483, April, 1974.

O. 1 O'Brien C. D., "Implementation of the ICPL Graphics Language on the PDP-15 Computer", Proc. Third Man-Computer Communications Seminar, Ottawa, pp. 9.1-9.8, 1973.

P. 1 Penney, J. P. and G. F. P. Deecker, "On General Purpose Software for Interactive Graphics," Proc. Fifth Australian Computer Conf., Brisbane, 1972.

P. 2 Pieke, B. and G. Schrack, "Implementation of an Interactive Graphics Language", Proc. Third Man-Computer Communications Conf., Ottawa, pp. 7.1-7.9, 1973.

P. 3 Poole, P. C. and W.M. Waite, "Machine Independent Software", Proc. Second Symposium on Operating Systems Principles, A.C.M., Princeton, N. J., 1969.

S. 1 Sackman, H., "Some Exploratory Experience with Easy Computer Systems", Proc. AFIPS National Computer Conf., Vol. 42, pp. M30-M33, 1973.

S. 2 Stockenberg, J. E. et al, "Operating System Design Considerations for Microprogrammed Minicomputer Satellite Systems", Proc. AFIPS National Computer Conf., Vol. 42, pp. 555-562, 1973.

S. 3 Sutherland, I. E., "Sketchpad: A Man-Machine Graphical Communications System", Proc. AFIPS. Spring Joint Computer Conf., Vol. 23, pp. 329-346, 1963.

T. 1 Thibault, P. C., "A Disc-Oriented Graphics System Applied to Interactive Regression Analysis", M. Eng. Thesis, Dept. of Elect. Eng., McGill Univ., 1972.

T. 2 Tinker, R. W. and I. L. Avrunin, "The COMRADE Executive System", Proc. AFIPS National Computer Conf., Vol. 42, pp. 331-337, 1973.

T. 3 Trivett, R., "PICADE - Prompt Interactive Creation of Active Display Elements", Proc. Third Computer Communications Seminar, Ottawa, pp. 15.1-15.8, 1973.

U. 1 Uzgalis, R. et al, "PL/OT 71: An Interactive Machine Independent Graphics Language", SID Digest of Technical Papers, pp. 48-49, 1972.

V. 1 Van Dam, A. and G. M. Stabler, "Intelligent Satellites for Interactive Graphics", Proc. of IEEE, Vol. 62, No. 4, pp. 483-492, April 1974.

W. 1 Wasserman, A. L., "The Design of 'Idiot-Proof' Interactive Programs", Proc. AFIPS National Computer Conf., Vol. 42, pp. M34-M38, 1973.

W. 2 Williams, D. L., "GRAPPLE - Graphics Applications Programming Language", Proc. Third Man-Computer Communications Seminar, Ottawa, pp. 5.1-5.9, 1973.

W. 3 Williams, R., "A Survey of Data Structures for Computer Graphics Systems", Computing Surveys of the A.C.M., Vol. 3, No. 1, pp. 1-21, March 1971.

Y. 1 Yan, G., "Implementation of Design Aids for Electronics in A Nuclear Research and Development Establishment", Proc. Third Man-Computer Communications Seminar, Ottawa, pp. 29.1-29.8, 1973.

Y. 2 Yourdon, E., "Design of On-Line Computer Systems", Englewood Cliffs: Prentice Hall, 1972.

LIST OF MANUALS, REFERENCE BOOKS, ETC. USED:

MAN-1	Bellgraph Computer System Programmer's Manual, BTL, Murray Hill, N. J., 1970.
MAN-2	DEC-9A-MRZB-D PDP-9 Background/Foreground Monitor Reference Manual
MAN-3	DEC-15-GWSA-D Graphic-15 Reference Manual
MAN-4	DEC-15-ZFSA-D Graphic-15 Programming Manual
MAN-5	DEC-15-BRZC-D PDP-15 Systems Reference Manual
MAN-6	DEC-15-AMZA-D Macro-15 Assembler Programmer's Reference Manual
MAN-7	GC28-6514-8 IBM OS Assembler Language
MAN-8	C28-6646-0 IBM System/360 Operating System, Supervisor & Data Management Services
MAN-9	Music User's Guide, McGill Univ. Computing Centre, 1972.
MAN-10	Computer Display Review, GML Corp., Lexington, Mass, 1974, Vol. 1-4.
MAN-11	Tekgraphics, April 74, Number 9