ISTITUTO NAZIONALE DI FISICA NUCLEARE

Centro Nazionale Fotogrammi

INFN-TC/86-21
19 Dicembre 1986

L. Fonti, M.L. Luvisetto, E. Ugolini, E. Remiddi, F. Semeria:

INSTALLATION AND USE OF TEX AT INFN-CNAF

Servizio Documentazione
dei Laboratori Nazionali di Frascati

Istituto Nazionale di Fisica Nucleare
Centro nazionale Analisi Fotogrammi

INFN-TC/86-21
19 Dicembre 1986

INSTALLATION AND USE OF TEX AT INFN-CNAF

L. Fonti, M.L. Luvisetto, E. Ugolini
I.N.F.N.-C.N.A.F., Bologna
E. Remiddi, F. Semeria
Dipartimento di Fisica, Università di Bologna
I.N.F.N.-C.N.A.F & I.N.F.N. Sezione di Bologna

This report describes previous work done at INFN-CNAF in installing TEX on VAX in view of its dissemination to INFNET VAXes and developping spoolers for existing devices, as well as the recent implementation of a screen previewer for the Olivetti M24-PC.

1. Introduction

In the past two-three years a dramatic drop in the cost of hardware has made possible a new dimension of computing and computer use. One of the most striking and common applications of Personal Computers is "text processing". In this field, many solutions are available with increasing capabilities from simple, basic features up to *technical and scientific* word processing (TWP) programs [1] .

Among the many TWP software products a very specialized one is TEX*, a TWP that has been especially designed to handle mathematical documents [2] . TEX diffusion is constantly increasing especially in the scientific community, mainly for the following reasons: TEX is a pubblic domain software, TEX is running on most computers ranging from mainframes to personal computers of virtually any architecture and size (above 512 Kbytes RAM), TEX produces a device independent output file, thus the very same file can be used to produce a preprint and the final output, TEX is "de facto" an international standard, thus the user can prepare the documents on a personal

* TEX is a trademark of the American Mathematical Society

computer and then use a mainframe to produce the high quality copies to be used for the creation of the printing plates, and finally TEX is "programmable" [3] .

TEX is a markup language, that reads ASCII files containing the text combined with the commands describing the formatting actions and produces a device independent output file (DVI). When the DVI file has been written, the task of TEX is accomplished. The next step is the production of a printable file by means of some software package producing the list of commands specific to the output device choosen for the particular application (i.e. memorandum, letter, internal note, report, book, etc.). Such a program is called "spooler", each output device requiring its own spooler.

Note that only the last step is printing and olny at this point the user can see his document in its final form, look for bugs, insert corrections — and repeat once more the whole procedure.

Some steps (such as modifying the source and generate the DVI file) are usually very fast even on a medium size minicomputer; spooling on the contrary can proceed very slowly depending on the graphic device at hand; that is true, in particular in the case of a dot–matrix printer connected with a serial line to the computer. For that reason the preview on a PC screen, such as Olivetti M24, is a very attractive possibility as it can display the content of the PC memory without any bottleneck due to data transmission.

In this report we describe TEX user support at INFN–CNAF with a general overview of:

- TEX usage and dissemination through INFNET,
- its implementation in past years,
- the Fortran spooler and previewers for VAX,
- and the recent port of previewing facilities to M24.

Unless the printing device is able to access down–loaded characters, the spooler step is slow due to the bit–map representation of the printed output, as described in the next sections, therefore most spoolers are two–step programs that:

1. read the DVI file and write the bit–map on disk (BIT file),
2. print the BIT file;

where the very spooler performs the first steps, while the last step is a device dependent driver and produces the final output.

On M24 the spooling steps are slow, but screen previewing is very fast; processing from TEX to DVI to BIT is anyhow almost acceptable at the present stage.

The VAX spooler is written in Fortran as that is the "default" language for INFN basic applications, thus also the M24 version is written in Fortran to minimize changes and implement the previewer without much effort and in a short time. From the present results, we expect to reach a fully satisfactory time performance with some improvements on the code and the use of the recent PC generation M28 with 80286 microprocessor.

2. Spoolers for graphic terminals on VAX. TEX at INFN–CNAF

Before going into the details of the spooling task, it is time to have a look at the way TEX handles the characters to typeset (i.e. letters, numbers, special symbols, etc.). TEX is much more than a conventional TWP, it works more or less in the same way as a human typist, thus it needs characters that have a very precise shape and size. Due to the complex way TEX acts, we need some knowledge of the typesetting technical terminology and an overview of the computer methods and algorithms required to produce high quality letters and symbols.

The characters are created as raster images, i.e. they are represented by dots, where a dot corresponds to a bit set to one for each black spot that should appear at some point in the page. The more dots per inch can be drawn on a page, the higher quality level will result. This device capability is referred to as "resolution".

To produce high quality pages, different styles, shapes and sizes for each character are required. The complete set of characters of the same style ans size is called "font". To be able to "draw" fonts of different shape and size TEX refers to the information related to a raster image of each character created by METAFONT [4] and stored in bit–map form on a storing device (disk).

From this point of view each page can be regarded as a giant matrix of dots. Therefore, to visualize the DVI file, a graphic device is needed. For high quality printing a resolution of at least 300 dot/inch is required, but for proof–reading and previewing a video terminal with a resolution of some 600 pixels in the horizontal direction and 400 in the vertical one can offer a fast and economic tool of invaluable help to produce high quality documents.

At INFN–CNAF use of TEX began in spring 1982 in view of its later dissemination to the other VAXes of the INFN network INFNET. At that time the smallest machines able to run TEX were superminis, and the only low–price device for which a spooler was at hand was the **Versatec** family of electrostatic printer plotters, with resolution ranging from 100 to 200 dots/inch. TEX was installed on a VAX 11/780 running VMS, but no facility to print the DVI files was at hand, except for a very low resolution dot–matrix printer (70 dots/inch). Fortunately INFN–CNAF was equipped with a high resolution graphic VT (Tektronix model 4014, with a visible raster range from 0–4095 in 'x' and from 0–3120 in 'y') and a dedicated spooler was implemented general enough to handle in a simple and versatile way a larger set of output devices.

A program that reads the DVI file and produces a graphic output is not a particularly difficult task to achieve, provided that the target machine is equipped:

1. with enough memory (either real or virtual),
2. a good compiler for some high level language,
3. a some what fast link to a graphic device.

The VAX superminis meet the above requirements for point 1. and 2. in their standard configuration and, in principle, could be equipped with some DMA interface for fast data transfer. Furthermore the VMS Run Time Librery (RTL) offers tools to implement a very friendly user interface with the operating system and the site environment. As we will discuss later, point 3. is critical for printers and graphic terminals linked to serial lines.

As our Institute is mainly concerned with high energy physics research, the "official" programming language is Fortran and, at the same time, VAX–VMS can be regarded as a Fortran machine, the spooler was developped in Fortran 77 [5] using the VAX–VMS compiler extensions for the general layout and referencing the RTL routines to handle the "virtual memory" allocation holding the bit–map raster information for each font used in the current DVI.

The DVI file is a kind of metafile that describes the graphical actions required to produce a hard copy output of the document; it contains commands that give the current x–y coordinates on the page, and commands that specify which character of the current font family must be typeset at that position. Each DVI file contains also summary information listing the names of all the different fonts in use for the document.

The shape and size of the characters is described in the form of pixel tables, one file per font family. Therefore the spooler must read the DVI summary, setup a table of required font files, allocate enough memory for those files and read the pixel information, then read the DVI description related to the selected pages and produce a bit–map for the whole page. As there is no garantee that the x–y coordinates are stored in increasing order in the DVI file, random access to the bit–map page is required, resulting in a big buffer in main storage.

The time required to read and interpret one TEX page on VAX 11/780 ranges from 1 to 2 seconds depending on page complexity plus the time required to setup the bit image of the characters in the page. For a VAX 11/750 the time required to process the same document ranges from 2 to 4 seconds, while on a VAX 8800 one page is processed in 0.15–0.33 seconds. When a character definition command is met in the DVI file, the address of the raster image in the pixel description is computed. Each character is stored as a variable size cell whose width is an integer multiple of 32 bits and whose height is made of a variable number of 32–bit words, thus cell columns are represented by bits, while cell rows are represented by words. To build the full bit–map, one needs to shift the pixels in one cell column to the current bit position, and to store the cell rows at the same beginning bit position in a 32–bit word whose address is incremented of an amount that represents the total page width in the output device.

Bit handling is done using OR, AND and SHIFT statements that are time consuming and represent a heavy load on the CPU. Usually such operations are implemented in machine language thus reducing the program portability. Programming in Fortran 77 has the advantage that bit handling functions are supported as in–line code by several compilers.

Bit handling is required to create the full bit–map of one page and, unfortunaltely, also for drawing the results on graphic devices which do not support direct raster access. Besides the overhead caused by bit operations, one must take into account the transmission time required to write the graphic protocol compliant information from main storage to output device.

A typical speed for such devices is 4800 baud on a serial line. The bit–map information amounts to some 300 Kbytes per page in standard TEX format at 200 dots/inch resolution and referred to the whole raster (i.e. full and empty areas of the page), thus the time required to draw the whole page ranges from 4 to 6 minutes on a VAX 11/750 equipped with a VT240 connected at 4800 bauds with pages containing a typical mix of text and mathematics. The time decreases for pages filled with formulas, but can consistently increase for very small font texts.

3. Use of TEX on a PC linked to a VAX

Since 1985 TEX is also running on personal computers equipped with Intel 8086 or 8088 micro-processors. Several program versions are available, the ones that implement the full system produce standard DVI files according to the format described in [6] . To run TEX the minimum system requirements are 512 Kbytes memory (or 640 Kbytes depending on the operating system release and the amount of macros used by the application) and a 10 Mbytes hard disk. Olivetti M24 is already equipped with a graphic card (640 $\times$ 400 x–y pixel resolution), for other PCs the graphic card could be not standard thus requiring a hardware upgrade.

PCs are frequently used as intelligent terminals connected to bigger machines like VAXes. This solution improves computer response both on PC and on VAX. The user can do editing on the PC and then run big applications on VAX using the link and some communication utility. In this case, as the PC DVI file is in standard format, one can run TEX on VAX and then down–load the DVI on the PC disk for local previewing.

Communication between the PC and a VAX are usually achieved through serial lines. In this way there is no problem in exchanging ASCII files. A simple protocol can be easily implemented on PC. On VAX DCL standard commands are used, thus avoiding the need to implement special drivers. In fact to read a file from PC to VAX, the CREATE command is invoked. To write a file on PC from VAX, the TYPE command is used.

To run the spooler on PC with down–loaded DVI files, there is no need to have TEX installed on PC, but the pixel files (PXL) must be resident on the personal hard disk. In this case also the PXL files must be down–loaded from VAX. As both DVI and PXL are binary files, a small program must be implemented on VAX that translates the file contents from binary to the corresponding hexadecimal representation in ASCII form. The opposite operation will be done on PC. This kind of transaction is necessary also to change the file blocking factor that differs on the two machines.

File transfer could also be handled by Kermit, a public domain protocol that allows file transfer between different computers. As Kermit is public domain there are available literally hundreds versions for many different machines and configurations. And also reports and articles describing Kermit application [7] have large diffusion.

Kermit is designed to operate file transfer over normal asynchronous lines using a packed oriented protocol. Transimission is in ASCII with "embedded control" characters prefixed to allow data conversion. Furthermore Kermit provides a mechanism by which file attributes can be negotiated, thus a wide range of file types can be transferred between the PC and the VAX.

Having a communication packet, the user can choose among different solutions such as either running TEX on VAX and down–loading the DVI on the PC or doing the whole job on the PC.

To preview the DVI the user must have the PXL files on the hard disk either from PC–TEX or down–loaded (once for all) from the host. Once the DVI is on the PC disk, the previewing process runs completely on the PC, thus avoiding a poor answer in case of a heavy loaded host.

4. TXMAPPER on VAX

The whole package is written in Fortran 77 using the VAX–VMS extensions. The package consists not only of the spooler but also of some other utilities with the aim of easing the debugging stage of the spooler development and updates and also of checking the TEX site implementation and fonts. As part of the routines are common to the whole package, they are included in a general purpose library. The definitions are stored on separate files and made modular and flexible through extensive use of the "PARAMETER" statement.

To achieve a high degree of modularity, the spooler can either write the bit map on a file (BIT) or display graphically the TEX page on a large number of graphic devices in view of portability and support for the INFN community. In case the bit map is written on file, it can be displayed later on the graphic device using the related "driver". In this way new device drivers can be easily implemented and included in the spooler only when they run correctly. On the other hand the user that has access to more than one device can produce the BIT file and then view or print it without rerunnig the spooler.

The spooler can run using different values for such parameters as magnification, device type and resolution, page selection and pixel compression. Default values are setted but these can be changed at user request and supplied in the form of qualifiers. The spooler has an internal help facility and can detect typing errors. Invalid or conflicting qualifiers force a new inquiry of values, thus avoiding problems and misleading behaviour of the output device. Furthermore the spooler addresses a set of logical symbols that can assume a class of predefined values to alter the internal defaults and customize the run environment.

The spooler must handle the following input–output files and allocate enough memory for the related data: the DVI file, the PXL files as input and the BIT file as output. The DVI file is written with fixed record length, each record 512 byte long. The record does not contain control information and its format is such that each byte can represent a command code indicating at the same time the operation to be performed and the number of following bytes to process for that command. In this way DVI commands can span one record. The memory required to handle DVI information is stored in two contiguous buffers 512 bytes each.

The PXL files can be of any size and number depending on fonts and magnification. They are addressed randomly according to the characters used in each font family and on the overall text style. Therefore, for each DVI page, the PXL files required for that page should reside in memory at least for the time required to create the related bit–map. To know how many PXL are used in the current document, TEX stores at the end of the DVI (in a command called "postamble") the font information. From the postamble, doing some computation for magnification handling, the spooler is able to set–up a table of file names and directories pointing to PXL (fonts at different magnifications are stored in dedicated subdirectories). On VAX, as the RTL supports Virtual Memory allocation, a tricky method that associates the size to the file names is implemented and from the above table the required storage is computed and allocated as contiguous virtual memory blocks. This area is accessed by the program through two variables stored in a specialized COMMON, i.e. the total size (LOC_VM) and the beginning address of the virtual memory area (ADR_VM) as assigned by the RTL routine "LIB$GET_VM". To access this area the user must declare an adjustable array of size LOC_VM in the routines that use the PXL information and specify the array address in the form of %VAL(ADR_VM) in the calling sequence.

It is very difficult to estimate the total memory required for the PXL as such information, for a given font, can range from a minimum of 3000 bytes to a maximum of some 160k at 200 dots/inch. A typical size for the most commonly used fonts is 10 kbytes, and for technical documents some 7–10 fonts per document are generally used, thus requiring some 70–100 kbytes storage. With VAX and virtual memory management this does not represent a critical request and it is completely transparent to the end user.

More critical is the memory allocation to store the output page in bit–map form. A TEX page is 6.5 inches wide and 8.9 inches tall and that size must be multiplied by the PXL resolution in dots/inch. The standard value for low–resolution devices is 200, but for laser printers it is already 300. Furthermore the page size can be enlarged if a magnification factor is required when running the spooler. Thus in the VAX version the page size is set to 8 inches width and 10 inches height and the resolution is set to 320 (this is not just magic, but it simplifies the computation as memory location are 32–bit words). A two–dimensional integer array defined as INTEGER*4 BITMAP(80,3200) is allocated in a COMMON area thus reserving more than 1 Mbyte for each page.

Also this big memory size is not critical on VAX–VMS that manages in a completely transparent way virtual memory allocation also for COMMON arrays, that is for those arrays whose sizes are declared already at compilation time.

It is important to point out the difference between the memory allocation methods used for PXL and for the BITMAP array. In the first case the program does not know anything about memory dimension until PXL size is computed and the routine LIB$GET_ VM is called, while BITMAP array is defined as a common area whose size is declared in the source file and assigned already at compile time.

All files are defined as "fixed record length". On VAX there is no need to specify also "direct access" when using fixed record length files. The main difference between sequential and direct access when the information is used as sequential is in "end–of–file" (EOF) detection. For sequential access EOF is detected in the status or in the end options of the Fortran READ statement, for direct access files EOF is meaningless and the ERR information must be checked instead to monitor a read operation on a non–existent record.

PXL information is read only in sequential order, thus the file is opened with the default access and record type parameters. BIT file is used as sequential in TXMAPPER, direct access is needed only by the stand–alone drivers for fast page selection. As this file is opened for output the access parameter is not required, but the fixed record specification is needed to override the default.

The DVI file contains the font information at its end and uses a pointer system to loop through document pages, therefore for maximum speed it is opened specifying the direct access parameter.

The enviroment setting through qualifier specification, the file operation and the memory management are strictly computer dependent, the rest of the program is easily portable. The program is highly structured and modular therefore the VAX–VMS code is hidden from the main structure and executed by specialized routines that can be eliminated or changed as a whole to fit other compilers and operating systems.

The spooler operation can be easily summarized as follows.

1. Set up run dependent parameters, i.e. magnification, page selection, output device, pixel compression: handled by TXGETQUAL;
2. Read the postamble, allocate virtual memory, read the PXL files: handled by TXRDPST and the library routine package TXVMPXL;
3. Decode the DVI and produce a bit–mapped page: handled by the MAIN program in the steering IF–THEN–ELSE structure and by the bit handling routine TXGET;
4. Write the current bit–map either to file or to the selected device as specified by the qualifiers: handled by TXOUT.

The spooler is made of nearly 3350 Fortran lines, 2000 for the spooler, 450 for the drivers as callable routines (not counting the stand–alone versions) and 900 for the library package, i.e. memory management, bit compression and general purpose routines for byte handling and the like as we will see in greater detail when describing the M24 implementation.

5. Bit–map compression

As we have already mentioned a TEX page is 6.5 inches wide and 8.9 inches tall. Fonts cannot be represented with an acceptable quality at resolutions below 200 dots/inch, and this is specially true for "slanted" fonts. As a consequence to display a page on a raster device we need $200 \times 6.5 = 1300$ dots per line (horizontal) and $200 \times 8.9 = 1780$ dots per page (vertical). As low resolution VT screens have at most 800 dots per line, to display a whole page one has either to reduce the page size at TEX level or to operate some kind of data reduction either in the spooler or running the driver.

If TEX standard page size is used and no compression is operated this results in a severe limitation on line width as the end of line is lost. Page length does not represent a problem as a new screen is displayed for each page section, that is that fraction of a whole page that fits the vertical resolution of the output device.

Using Tektronix 4014 with the high resolution qualifier on, it is possible to display the whole line length. But with other devices the only solution at hand is a smaller page size, as also scrolling is not easy to implement and in any case it represents a slow and time consuming process.

At TEX level any page size can be specified by use of the \hsize and \vsize commands at the beginning of the document. As a drawback TEX can have problems in formatting the page with lines that are either too empty or too full, thus resulting in "ragged" right margins and, last but not least, the previewed document has a completely different page make–up with respect to the final high quality print.

To avoid a different set–up for TEX, some kind of data reduction must be devised. Taking into account the bit–map structure of each character, the only solution is to operate a "pixel compression" and divide the total amount of dots by 4, this results (at 200 dots/inch) in simulating fonts at an approximate 100 dots/inch resolution. This method is also fast enough and the compression step does not appreciably increase the total preview time.

Some care must be taken using bit compression as low resolution devices have usually not only a limited number of pixels but also the pixel size is a limiting factor, therefore the resulting output can be confused and difficult to read especially with regard to subscripts and supersctipts in mathematical formulas, that make use of smaller fonts.

When VT240 terminals (800 pixels per line) are used on VAX, to improve the readability, the ouput file can be written using a magnification qualifier that requests 240 dots/inch fonts, i.e. 1560 pixels per line, that divided by 2 at the compression stage still fit in 800 pixel. With lower resolution VTs some compromise between compression, page size and output quality is to be used depending on the specific needs of the text for which previewing is required. As in general previewing is the most useful aid for TEX macro development, page size and quality are not mandatory, therefore it is up to the user's experience to set–up an adequate choice for the aimed application.

A description of the method used to produce the compressed output follows. As the compressed output can be produced in two ways:

1. writing the BIT file on disk,
2. displaying the bit–map on VT,

different methods are needed in the two cases.

Attention must be paid to the horizontal and vertical dimensions, as they play different roles and need different programming techniques for data compression, even if the operation is logically symmetric. This is due to the fact that rows are addressed as 32–bit words while columns are addressed as single bits to produce a raster image.

The row handling part of the code is very simple and it is the same in both cases. Let us say that the j-th row is stored in the integer array TREC and the next one is tored in TREC1 and TEX is the integer array that contains the compressed output. Let LWDOUT be the number of columns expressed as 32–bit words, then the following DO loop will compress the rows and divide by 2 the number of pixels per page.

```
        DO 100 JJ=1,LWDOUT
        TEX(JJ)=IOR(TEC(JJ),TREC1(JJ))
100     CONTINUE
```

Once we have compressed the pixels in the y–direction, to reduce the data in the x–direction we must use two different algorithms, depending on the output type we will use, either disk file or graphic device. When accessing a graphic device that is not bit–map addressable, we have to extract the pixel information and generate vectors for the black dots or compute a new vector initial position for the empty areas of the page. In this case we will test two consecutive bits and add 1 to one of the following counters: NONE (i.e. a contribution to the current vector), NZERO (i.e. a contribution to the space between vectors) depending on the following conditions:

- if both bits are off, add 1 to NZERO,
- if both or one of the bits is on, add 1 to NONE.

In this way we divide by 2 the number of pixels in each line and plot the page with a very small time overhead with respect to the full page algorithm.

On the contrary, when we want to write the compressed bit–map on disk, to operate a fast data reduction, we generate a configuration table and address this table on a byte basis. As a byte can range from 0 to 255 in absolute value, we create a table of 256 elements that contains all possible compressed configurations for a byte. In this way we reduce 8 bits to 4. To create the new bit–map we fit 2 bytes in one, thus to simplify and speed–up the process we store the compressed information in two tables, one for the leftmost 4 bits and the other for the righmost ones.

The tables can be stored on disk and addressed only when compression is requested by the spooler qualifier. The following code fragment shows the use of the table where the integer arrays TAB_ L and TAB_ R contain the table information and TEXB is a byte array with the original information, the compressed datum is stored in the byte array REDB.

```
        DO 100 JL=1,LWDREC*4,2
        I_L=ZEXT(TEXB(JL))
        I_R=ZEXT(TEXB(JL+1)
        ITOT=IOR(TAB_L(I_L),TAB_R(I_R)
        REDB(JB)=ITB(1)
        JB=JB+1
100     CONTINUE
```

The above code fragment makes use of the VAX–VMS in–line Fortran function ZEXT to perform unsigned operations on byte data. For other Fortran compilers a different approach may be required as ZEXT is not a standard Fortran 77 feature.

6. Spoolers for PCs and M24

As already said TEX is a public domain software in its source form and regarding some implementations, i.e. for VAX and other machines. For PCs a big amount of work is required to fit a huge Pascal program like TEX in 512K and to achieve a reasonable speed, therefore the PC TEX implemetations and related software like the spoolers are not public domain with package costs nearly at the same level of other high quality commercial software running on PCs. PC prerequisites are at least 512K (640K for specialized macro packages) and a hard disk.

The TEX PC packages usually include the program, the general font information for TEX, some macro package and usually the spooler for a dot–matrix printer with resolution ranging from 70 to 180 dots/inch or analogous. Some implementations offer also a screen previewer as an optional facility at more or less the same price as the TEX package, furthermore most previewers require a graphic card as a hardware add–on.

TEX package on VAX is delivered with a complete set of PXL files with resolutions ranging from 200 to over 800 dots/inch. In case of PC limited PXL sets are delivered depending on the spooler. For dot–matrix printers usually only the 240 resolution is present, while for the screen previewers smaller fonts are delivered as the screen can not handle in a fast and satisfactory way the standard fonts due to its limited resolution and overall size.

In our case no spooler facility was running on the M24 as we use TEX on VAX. Therefore we decided to implement a dedicated TXMAPPER version. The PXL files are down–loaded from VAX at the smallest available resolution, that is 200 dots/inch and stored on the PC hard disk. At the same time only the PXL file sets commonly used are stored on disk. This is not a limitation as the down loading process is fast and easy to use, so that newly created fonts can be transferred at will to the PC mass storage.

On M24 no graphic card add–on is required as the standard screen resolution is 640×400, as it is also an all–point–addressable (APA) device, no bit handling is needed once TXMAPPER has created the page raster information.

7. TXMAPPER on M24

When we decided to implement TXMAPPER for the Olivetti M24 PC, the first step was to choose a Fortran compiler as we wanted to avoid the more lengthy approach of translating our spooler into another language, namely either C or Pascal. We had at disposal 2 Fortran compilers: the Microsoft Fortran Compiler * and Professional FORTRAN **.

As the VAX version makes large use of the VMS Fortran extensions, the Microsoft compiler was not adequate to our needs as it adheres strictly to the ANSI standard. On the other hand Professional FORTRAN offers a large set of extensions and even more important they have their counterparts in the VAX ones; this compiler requires the 8087 coprocessor, already added to the M24.

Last but not least, Professional FORTRAN offers a very efficient symbolic debugger, an invaluable tool for fast error detection and correction.

For TXMAPPER the most needed extensions are:

* Microsoft Fortran is a registered trademark of Microsoft Inc.

** Professional FORTRAN is a registred trademark of Ryan–McFarland Corporation

- symbolic name length up to 31 characters,
- some special character in symbolic names (_ for VAX, $ for M24),
- the INCLUDE statement for multifile source,
- additional data types such as INTEGER*2,
- hexadecimal constants,
- the Z field descriptor for hexadecimal values in the FORMAT statement,
- the bit handling intrinsic functions BTEST, IBSET.

The very first operation when converting programs from one machine to another is the design of the target layout exerting special care in mapping the machine dependent routines from the old to the new environment. In our case we had to change the user interface and the virtual memory allocation, and moreover to check the array dimensions for a 640K machine. Due to storage limitations the spooler for M24 offers a limited user interface in that no magnification and no output device facility is provided.

To limit the implementation phase on M24, as first step we developped a modified version on VAX by changing the RTL dependent routines TXGETQUAL and TXVMPXL. TXGETQUAL is the user interface manager, thus it was simply eliminated and replaced by interactive I/O to inquire the allowed M24 options; such as DVI name, page selection and data compression.

TXVMPXL allocates virtual memory and reads the PXL files for the whole document. For M24 we have restricted the page size to TeX standard and 200 dots/inch, this means that to store the page bit–map we need an array dimensioned as INTEGER*4 BITMAP(42,1860) corresponding to 312480 bytes. Therefore we have 330K left for PXL , working areas and instructions. The DVI input buffers are halved with respect to the VAX version as the record size on PC is 256 bytes. Thus an array of 190K is reserved for the PXL information. This size is more than enough for common technical documents, in fact, as we have already mentioned, usually 100K is the upper limit. In summary, 510 kbytes are used for the data and 130K for the program. The code is really very small (less than 9K) and does not represent a problem. With such a memory allocation, it is not difficult to rewrite TXVMPXL as the RTL calls are eliminated and the I/O part is changed to access an array in common.

The above changes were tested on VAX, after which the source files were transferred on the PC hard disk and we begun to work at the non–portable modifications.

The remaining changes were very simple and straightforward, namely some differences related to: syntax, argument addressing of BTEST and IBSET, OPEN statement and I/O organization. The syntax difference that requires most changes is related to byte handling, i.e. on VAX there exists the data type BYTE, while on M24 we have only LOGICAL*1, this means that one has to play with EQUIVALENCE to be able to use bytes in arithmetic operations.

The other significant changes involve I/O. The files are binary fixed length record and can be correctly handled on PC only if opened as “direct access” and “unformatted”, therefore we had to modify all the OPEN statements accordingly and the related I/O operations. On VAX only DVI files are used as direct access, the others are defined “fixed” but accessed as sequential.

The whole package, including the implementation of “ad hoc” small utilities to down–load DVI and PXLs, required one month of part–time work. The execution time depends heavily on the document being processed. The time required to load the executable image in memory from hard disk and to read the related PXL for not less than 7 files amounts to 7–10 secs. One full TEX page is processed in a varying time ranging from 30 to 75 secs. In this way the BIT file is written on the PC mass storage.

The previewing step is handled in another program and it is almost istantaneous. In this way the user does not wait to see the page and is not bothered by the spooler that runs seprately and does not interfere with the document examination phase. The graphic previewer provides a full window facility with up–down and left–right scrolling and very fast page selection, thus offering an overall performance much better than on the VAX. In fact the total CPU time on VAX is much less but the previewing step, due both to the low speed connection of the graphic terminal and to its protocol, is slower and even worse depends on the global VAX load.

8. M24 previewer characteristics

The following section describes the VT driver and its use in greater detail. To the user's benefit, an overview of the complete step sequence from document creation to previewing in our environment is listed below.

- create the TEX source code using your favourite editor either on PC or on VAX,
- run and debug TEX to produce the DVI (on PC or on VAX, in the last case down–load the DVI from VAX to M24),
- run TXMAPPER on M24,
- run the preview driver on M24.

If we run TEX on VAX and use the spooler on the PC, we have to transfer binary files between the two machines using a serial link and running a special terminal emulator program on the PC. As only ASCII characters can be transfered, to down–load binary files such as DVI or PXL from VAX to M24 we must convert them from binary to ASCII hexadecimal format and, at the same time, take into account the file size needed on the target disk.

For data transmission we have implemented on the VAX an elementary Fortran program which types hexadecimal data with the proper record length on the screen. In this phase the M24 emulates the VAX terminal: it receives characters from the VAX, displays them on its screen and stores them in a local disk file with optional conversion of each pair of hexadecimal characters into a binary byte. As this kind of operation is not supported by standard terminal emulators, we developped an "ad hoc" emulator written in Turbo Pascal *.

A specialized driver (MAP) written in C is called to preview the BIT file produced by TXMAPPER. Referencing general purpose routines, the driver handles high resolution graphics and windowing with scrolling capabilities and plots individual pixels as a raster image.

The driver is invoked typing MAP followed by the current file name. MAP reads a full TEX page in bit–map format and stores it in memory. To that aim the minimum required memory size must contain one page raster as defined in the BIT file itself, i.e. page_length × line_length expressed in 32–bit words. Page and line size are computed by TXMAPPER from the DVI postamble information and written to the BIT file normalized to pixel units and expressed in 32–bit word count.

The maximum required size is 312480 bytes, but TXMAPPER writes BIT files that are formatted depending on the page size declared in TEX, i.e. smaller if a smaller page size is specified

* Turbo Pascal is a trademark of Borland International Inc.

using the \hsize and \vsize commands. It should be pointed out that, in principle, no limitation is set on the BIT file size, as TXMAPPER could be run on the VAX to a larger page format, then down–loaded to the PC for the previewing step. Usually this method is not pactical due to the file size, in fact BIT files are much bigger than DVI ones thus involving transfer time overhead.

The above work area is dynamically allocated by the program, and in case of insufficient memory an error message is displayed and the program aborts.

If successful, the program plots the first window that represents the upper left side of the first page, then the user can move around using the scrolling and page selection built–in facilities that are activated by the "arrows" and other special keys on the PC keyboard. The following table describes the provided functions activated by the corresponding special keys. In the table the keys are represented either by special symbols (i.e. →, etc.) or by bold face characters. Some keys have double functions in combination with the "Shift" key, therefore one command is executed whenever the key is pressed by itself, and the other whenever the key is pressed after the shift key.

	Key	*Function*
	→	move to the rightmost side of the page
	←	move to the leftmost side of the page
	↓	move half screen down
	↑	move half screen up
Shift	→	move a small amount to the right
Shift	←	move a small amount to the left
Shift	↓	move a small amount downwards
Shift	↑	move a small amount upwards
	HOME	resume from the left upper corner of the current page
	END	display the left bottom corner of the current page
	PG DN	load and display the next page
	PG UP	load and display the previous page

The driver most time consuming task is the input of the bit–map page from disk that depends on page size and disk access time. This operation requires 3–5 secs. Window and scrolling functions are almost instantaneous due to the raster information direct addressing the screen memory.

In fact plotting of single dots on the PC screen does not require to write an individual information for each dot or vector on the screen, and the BIT file is formatted with the graphic information in PC raster protocol, that is a full horizontal line can be directly copied to the screen memory without further processing. Screen memory addresses need to be computed only for the first byte of every new line.

High speed is thus achieved in that graphics on M24 is easy to hanlde, and consequently the driver code can be very compact (less than 150 lines), additionally the C compiler produces very effective and fast images. As an example, the following code fragment shows how simple and brief is the routine that executes the plotting operation.

```
window (hor,ver)
#define SCREEN 0XB8000L
/* SCREEN is the VT memory address */
int hor,ver;
```

```
{
  int line,hoff;
  long verx;
  char *vadd, *padd;
  hoff = hor / 8; verx = ver; verx *= lineleng;
  padd = (char *)(page + verx + hoff);
  for (line=0; line<400; line++) {
    vadd = (char *)(SCREEN + (line % 4)*8192 + (line / 4)*80);
    movmem (padd,vadd,80); padd += lineleng; }
}
```

As shown for each new line the VT address is computed (vadd), then 80 bytes, corresponding to 640 pixels, are directly moved from PC to VT memory.

9. Conclusions

Current PCs offer performances competitive with bigger machines and represent very appreciated tools as single user stations. Their memory addressing capability together with reliable and reasonably fast hard disk offer a wide range of demanding applications. Modern compilers with high level facilities and advanced extensions are available, thus program conversion from bigger machines does not represent a difficult problem, especially when the package has been written in a modular way.

In our case a limited number of changes was necessary and the total time required to port the spooler has been more than acceptable. The bottleneck has been the compilation time on M24; even if we had divided the source code in smaller files, the time required for the biggest one ranges from 2 to 4 minutes. Unfortunaltely, due to DVI format, the steering part of the program is a long structure that cannot be divided into smaller units.

The timing of the whole spooling process compares already quite well with the VAX 11/750, but further improvements seem at hand. It is our feeling that the conversion of TXMAPPER from Fortran to C would not represent a difficult task, while the resulting package could run 2–3 times faster. Furthermore a spooler totally developped in C represents a very interesting solution, because this would eliminate the need of the 8087 processor, and allow the inclusion of the previewing driver in the same program, thus enabling its access to a larger number of users.

References

[1] R.S. Palais, Mathematical Text Processing, Notices of the American Mathematical Society, Vol. 33, No. 1, January 1986

[2] D.E. Knuth, The TEXbook, Addison–Wesley Publishing Co. Inc., Reading, Mass.

[3] R.S. Palais, An Introduction to TEX and AMS–TEX, Notices of the American Mathematical Society, Vol. 33, No. 2, March 1986

[4] D.E. Knuth, The METAFONTbook, Addison–Wesley Publishing Co. Inc., Reading, Mass.

[5] M.L. Luvisetto and E. Ugolini, A TEX82 spooler for VT and dot matrix printers, TUGboat, Vol. 6, No. 1, March 1985

[6] D.E. Knuth, TEX: The Program, Addison–Wesley Publishing Co. Inc., Reading, Mass.

[7] B. Nelson, The Kermit protocol and the PDP-11, DECUS U.S. Chapter SIGs Newsletters, Vol. 1, No. 11, July 1986