REAL TIME PICTURE PROCESSOR: DESCRIPTION AND SPECIFICATION

NCI/IP Technical Report #7

March 31, 1976

Peter Lemkin, George Carman,*
Lewis Lipkin, Bruce Shapiro,
Morton Schultz

National Cancer Program

NATIONAL CANCER PROGRAM

U.S. Department of Health, Education, and Welfare / National Institutes of Health / National Cancer Institute

NCI/IP-76/03

REAL TIME PICTURE PROCESSOR: DESCRIPTION AND SPECIFICATION

NCI/IP Technical Report #7

March 31, 1976

Peter Lemkin, George Carman,*
Lewis Lipkin, Bruce Shapiro,
Morton Schultz

Image Processing
Division of Cancer Biology and Diagnosis
National Cancer Institute
National Institutes of Health
Bethesda, Maryland 20014

*Carman Electronics, Inc.
Corvallis, Oregon 97330

"We here highly resolve . . ."

T A B L E   O F   C O N T E N T S

L I S T   O F   F I G U R E S

LIST OF TABLES

NCI/IP-76/03

Real Time Picture Processor - Description and Specification

NCI/IP Technical Report #7

Peter Lemkin, George Carman*, Lewis Lipkin,
Bruce Shapiro, Morton Schultz

Image Processing Unit
Division of Cancer Biology and Diagnosis
National Cancer Institute
National Institutes of Health
Bethesda, Md. 20014

*Carman Electronics, Inc.
Corvallis, Oregon

March 31, 1976

ABSTRACT

The concepts and general design specifications of a new hardware picture processor are presented. The design was strongly influenced by the characteristics of biological images. This device, now in the early stages of construction at the National Institutes of Health, will meet some of the requirements for interactive design, specification and testing by biologists of algorithms for cell classification, description and measurement. The RTPP is but one component, albeit a major one, of our entire system which is intended to permit on-line description construction by the cytologist.

SECTION 1

Introduction

This paper discusses the functional hardware design specification of a Real Time Picture Processor (RTPP) ([Lem74], [Carm74]) for use in designing image processing systems for biological materials. Real time, as is used here, denotes time proportional to that required for comfortable human interaction.

The concepts and general design specifications of a new hardware picture processor are presented. The design was strongly influenced by the characteristics of biological images [Lem74]. This device, now in the early stages of construction at the National Institutes of Health, will meet some of the requirements for interactive design, specification and testing by biologists of algorithms for cell classification, description and measurement. Additional detailed hardware specifications of the Real Time Picture Processsor, RTPP, are documented [Carm76]. The RTPP is but one component, albeit a major one, of our entire system which is intended to permit on-line description construction by the cytologist. The overall discussion of the system components is in [Lem76c].

The Real Time Picture Processor in its role as a microscope controller is designed to perform at least the set of operations performed by the NCI Grain Counter-1.1 [LipL74]. These operations include joystick control of an optical microscope stage in X, Y, and focus. The RTPP, in addition, allows stepping motor control of a 4:1 microscope zoom, a rotary monochromator, and a rotary neutral density filter.

The system is designed for extremly rapid serial digital processing of digitized images to be carried out in what for the user, seems like, real time. Special purpose hardware makes this speed possible. Output from the RTPP is in the form of images to be displayed, lists of properties of objects in the image, or processed images. Because of powerful gray scale manipulation instructions, computations are not limited to processing planes of binary valued images.

These capabilities allow flexible experimentation with an on-line microscope image picture processing facility. Employing this facility, users have the ability to generate precise definitions of biological cell classifications using the RTPP as input for an interactive relational data model residing on a remote PDP10 computer. Furthermore, the system will allow the user to make measurements of cell parts and to develop heuristic measures for cell characterization.

Because processing is fast (on the order of tenths to a few seconds), the interaction between user and the system is well matched to allow for experimentation with more complex algorithms than usually attempted with serial systems. Several papers ([Lem74] and [Lem75]) give reviews of some recent special purpose image processing hardware.

1.1 Overview of this paper

Section 1 introduces the rationale on which the structure of the RTPP is based. Sections 2 through 5 discuss components: i.e. the Quantimet (QMT), buffer memories (BMs), triple line buffers, and General Picture Processor (GPP) respectively in more detail. Section 6 discusses the physical implementation of the RTPP while Section 7 comprises references. Appendix A details the GPP instruction set while Appendix B lists the GPP I/O registers and GPP I/O lights and switches. Appendix C presents the PDP8e Input Output Transfer instructions used in the RTPP. Appendix D gives several examples, including estimated running times, of several algorithms programmed for the RTPP.

1.2 RTPP design goals

There were several (sometimes conflicting) design goals used in defining what should go into a picture processing facility such as the RTPP. Long experience in picture processing on general purpose serial machines has resulted in the production of picture processing packages such as PAX [JohnE70]. Some of these operations consist of simple pixel boolean binary planar operations, 4 and 8 neighbor operations, and then more complex operations such as blob finding, etc. Obviously, as the complexity of an operation increases, so does the complexity of its hardware implementation. We decided that neighborhood processing was the upper limit of complexity that the resources availiable to our group would allow us to undertake. Several years of biological experience posed another requirement: i.e. that grayscale texture measure algorithms be easy to implement. This means doing integer arithmetic on gray scales images rather than boolean arithmetic on planes of binary images.

A general gray scale parallel processor is an unrealistic goal. The complexity of such a device, (at the current state of the hardware art and occasioned by the combinatorics of the required hardware interconnections) place it well beyond the construction abilities of a small group. The cost of such a device also would make it difficult to justify if it were to be dedicated rather than a shared resource.

As will be seen we have drawn often from the good efforts of other computer and picture processing system designs, (especially [Dec71], [Dec72a], [Dec72b], [Dec67], [Thor70]) as well as many other influences. It was hoped that some of the best (and not necessarily the most complex or costly) features of some of the above machines could be incorporated into our design.

Before going into the design of the RTPP the definitions of some of the terms used in the rest of this paper are given.

Picture - set of gray values

A image part of a picture is a square array approximation of the corresponding real image. Each element in this sampling array is said to have an integer gray value called a pixel. This gray value is a measure of the darkness or whiteness of that pixel. In our discussion, white will be zero and black 255. This representation is said to have 256 linearly resolvable gray levels or values. The gray value is generaly given in the range of the powers of a binary number as it is usually derived from the output of an analogue to digital converter. i.e. 8, 16, 32, 64, 128, 256 etc. An alternative view (which lies at the basis of such implementations as PAX) of an image is as a set of overlying binary arrays more commonly called binary planes. Thus 256 gray values is represented by 8-bits or 8 binary planes.

Neighborhood

Within a image, one usually deals with the concept of neighborhood. A neighborhood is a set of pixels close to (usually touching) a given pixel. An example of this is all pixels touching a given pixel in a array. For a square array there are 8 such pixels and consequently this 3x3 set of pixels is called the 8 neighborhood or 8 nearest neighbors. In this paper and in the RTPP the labeling of the 3x3 neighborhood is as follows:

```
3 2 1
4 8 0
5 6 7.
```

This choice of neighborhood labeling facilitates chain coding as the pixel index corresponds to its angle with the central pixel divided by 45 degrees.

Current pixel

The current pixel is the central pixel (8) in this 3x3 array. The current neighborhood is the 3x3 array surrounding the current pixel.

Neighborhood operation

A neighborhood operator is a unary or binary operator which maps neighborhoods into pixels. One of the ways the neighborhood operator works is to do various operations on parts of the neighborhood array as specified by an ordered list of pixels to be operated on. When this list is ordered in terms of the orientation of neighborhood elements, it is called a direction list.

Pixel operator

A pixel operator is a unary or binary operator which maps pixels into pixels in a 1 to 1 (x,y) mapping.

The RTPP performs neighborhood operations by executing a program on a neighborhood for each neighborhood in the plane. This is actually done using special purpose hardware called the General Picture Processor (GPP) which will be discussed below. The GPP is designed to allow pixel operations to be performed on all pixels of a 3x3 neighborhood within a single I/O access, thus allowing 3 lines to be processed at a time.

1.2.1 Picture operations

Image data processing in particular consists of doing mostly binary argument algebraic operations. These can be enumerated in an interesting way as shown below. The following are binary algebraic picture operations where "o" is a general operator. Let I1, I2 and I3 be images. Then,

I1 o I2 --> I3
I1 o -->I3
I1 o I2 --> a property list
I1 o I2 --> a relational data structure.

The operation I1 o -->I3 may be thought of as either a unary operation or a binary operation with a null second operand.

These picture operations might include such operations as boolean: bit set, bit clear, bit complement, and, or, exclusive or, equivalence; direction list processing; maxima; minima; Fourier transforms; threshold slicing; scaling; component labeling; propagation of edges; edge finding; pixel-wise addition, subtraction, multiplication, division; rotation and shifting of images; gradient; gray level histogram. Run-length/gray-scale texture histograms should also be computable by the GPP as well a large class of other picture operations. The PAX functions mentioned above are described by Johnson [LipB70].

In Appendix D sample GPP programs are shown, which compute some of the above mentioned operations and for which timing estimates are given.

1.2.2 Picture operations as binary operations

These basic binary argument relations on picture data are typical of picture processing in systems. What varies is the complexity of the operation "o". The increasing complexity of operations in the binary image plane is shown below in five increasingly complex levels I to V. Let "x" below be a pixel in a NxN neighborhood. As will be shown, an analogous hierarchy also exists for gray scale machines.

Let Ij(k) be the k'th pixel in picture j.

(I) Pixel (picture element) operations:

```
I1(k)  o I2(k)-->I3(k).
```

(II) Neighborhood operations around a pixel into a pixel:

```
xxx
xxx   o   I2(k)  --> I3(k).
xxx
   I1
```

(III) Neighborhood contextual operations into a pixel:

```
xxx      xxx
xxx   o  xxx    --> I3(k).
xxx      xxx
   I1       I2
```

(IV) Neighborhood operations into a neighborhood:

```
xxx      xxx         xxx
xxx   o  xxx    -->  xxx
xxx      xxx         xxx
   I1       I2          I3
```

(V) special high level functions:
edge detection, differentiation, smoothing, propagation, etc.

With progress from level I to IV and V, the increasing complexity (number of connections, number of nodes, number of modules) for a hardware circuit to perform such an operation increases. We use this wiring complexity as an estimate of the "complexity" of the operation and thus to characterize the complexity of a given level of operation.

Most small neighborhood parallel binary machines exhibit maximum complexity of levels II or III. For an NxN binary plane image these have complexity levels between N^2 and N^4 (e.g. for a 3x3 array this is 9 to 81). The ILLIAC III is an example of a level IV binary machine.

The same five levels of operations could be applied to the class of gray scale image machines. There is only one known existing gray scale processor, the ILLIAC IV.

It is interesting to examine the complexity of gray scale parallel processing operations. Assume a K bit gray scale (for binary images K is 1) and an NxN image. A binary machine having level II complexity would be interconnected on the order of NxNx1. Then a gray scale machine (having K bits of gray scale) of level II complexity would be KxNxN interconnected. The complexity of level III operation is on the order of (KxNxN)x(KxNxN)xN or $(K^2)(N^5)$. For level IV the complexity is of the order of (KxNxN)x(KxNxN)x(KxNxN) or $(K^3)(N^6)$. Obviously, the costs of implementing a machine rise with its complexity. The cost of debugging, documentation, and maintenance follow a similar increase with increasing complexity.

1.2.3 A level IV gray scale machine - 3 address machine

A gray scale picture processor of at least level IV complexity is needed to meet the requirements of the Real Time Picture Processor delineated above. This is the least complex machine that permits full generality. The use of such a level IV machine as a design tool will, we believe, allow the proper selection of some level V functions.

Although a K bit gray scale machine can be simulated using K binary planes, the expression of algorithms is awkward and usually inefficient.

Not surprisingly, a parallel gray scale machine of level IV is beyond the resources of our laboratory. Our solution to the design/maintenance dilemma has been to design a fast serial machine which operates on neighborhoods of rectangularly tessellated gray scale picture arrays. The machine's speed and a felicity in expressing algorithms is achieved partly through the use of triple operand instructions. In this machine a neighborhood has a serial representation.

The following example illustrates serial representation. Let us consider the images I1 and I2 (one of which may be the temporal successor of the other, or they may represent members of a set of serial sections etc). It is desired to find a measure of pattern similarity (in order to spatially or temporally align the images). The problem is solved by finding a measure of pattern similarity between neighborhoods I1 and I2 by computing their product and summing this product into the central pixel of I3. All neighborhoods are 3x3. Neighborhood I1, for example, comprises pixels I1(8) (its central pixel) and boundary pixels I1(0) through I1(7).

The operation to be performed is $I3(8) = \sum_{i=0}^{8} I1(i) I2(i)$.

The neighborhood processor could perform the following sequence of operations:

```
   Form the products array I3 from I1 and I2.
        I3(0)<--I1(0) MUL I2(0)  ; product of I1(0), I2(0) into I3(0)
        I3(1)<--I1(1) MUL I2(1)  ;            I1(1), I2(1) into I3(1)
        I3(2)<--I1(2) MUL I2(2)  ; etc.
        I3(3)<--I1(3) MUL I2(3)  ; naturally there is a way
        I3(4)<--I1(4) MUL I2(4)  ; to do this with a loop
        I3(5)<--I1(5) MUL I2(5)  ; structure
        I3(6)<--I1(6) MUL I2(6)
        I3(7)<--I1(7) MUL I2(7)
        I3(8)<--I1(8) MUL I2(8)

   Form the sum of the pixels in the I3 neighborhood.
        I3(8)<--I3(8) ADD I3(0)  ; sum the pixel product into I3(8)
        I3(8)<--I3(8) ADD I3(1)
        I3(8)<--I3(8) ADD I3(2)
        I3(8)<--I3(8) ADD I3(3)
        I3(8)<--I3(8) ADD I3(4)
        I3(8)<--I3(8) ADD I3(5)
        I3(8)<--I3(8) ADD I3(6)
        I3(8)<--I3(8) ADD I3(7)  ; The sum of the entire neighborhood
                                 ; product is now in I3(8)
```

These operations, performed over the entire set of corresponding neighborhoods and resulting in the construction of a third image, where the gray value at the point is a measure of the local similarity of the anticedent images, constitutes a solution to the problem.

These iterative neighborhood operations, prohibitively expensive in time and computer cost on standard machines are to be implemented on the General Picture Processor (GPP). This is the fast serial 3-address component of the RTPP. As will be further detained below, the 3-address machine is closest to the triple operand concept at the base of the mapping: I1oI2-->I3.

1.3 The configuration of the RTPP

The real time picture processor hardware shown in Figure 2 consists of an Imanco Quantimet 720 (QMT); a Zeiss Axiomat microscope with stepping stage, focus and light source; a galvanometer mirror precision scanner; a Dicomed model 31 64-gray level storage display; up to eight 256x256 16-bit gray level buffer memories (BM); a general picture processor (GPP); a PDP8e computer with 32K core; a four 1.5 million word disk cartridge drives; an interactive control desk shown in Figure 3, and a high speed connection to a PDP10 computer on which the modelling programs are run.

1.3.1 Axiomat microscope

A Zeiss Axiomat microscope is used to supply images to the RTPP system. It has been modified so that the focus and 4:1 zoom knob controls are under computer actuated stepping motor control. In addition, the stage X and Y positions of the slide are moved by computer driven stepping motors. The light illumination subsystem contains variable wavelength interference and variable neutral density filters. These two functions are implemented by use of continuous rotary wedge filters which are controlled by the computer via high speed stepping motors. Currently, a Quantimet plumbicon or vidicon scanner and a precision galvanometer mirror scanner are interfaced to output ports of the Axiomat. There is an additional viewing port within the viewer's reach. Thus there are three image planes accessable simultaneously.

1.3.2 Video input and output devices

The video input, display, and certain image processing functions can be performed by various I/O devices connected to the multiple output ports of a Zeiss Axiomat Microscope.

A Quantimet 720 image analyzer has been incorporated into the RTPP. The plumbicon camera video is digitized to 8-bits for sampling by the system (at a 8 MHz rate) and is then reconverted to an analogue signal for reinsertion into the Quantimet video input chain. The Quantimet includes a 10.1 frame/second television display and some minimal feature extraction and measurement hardware to be discussed below.

A precision galvanometer mirror scanner with a 1024x1024 pixel random access window and 256 gray level video output can also be used as a digitizing input device to the RTPP from the microscope. The PDP8e may route mirror scanner data to a buffer memory for display on the Quantimet or further procesing by the RTPP. In addition, a Dicomed model 31

1024x1024 picture point 64-gray level precision storage display may also be used as a storage output device in the RTPP. Presently, hardcopy images from the system are produced by photographing the Dicomed display.

The Quantimet is the preferred I/O device for real time interaction because of its rapid frame rate. The digitized picture from the plumbicon scanner video is a 256 (64 usable) gray level digitized image within a 880x680 pixel window. The QMT display may also be used to post images from buffer memory windows inserted within original scanner image data. There are two types of cameras: a plumbicon and a vidicon. The plumbicon gives better linearity for use with densitometry while the vidicon has a larger dynamic light input range. When one of these cameras is used on the Axiomat at the same time the galvanometer scanner is used the difference in dynamic range of light intensities required is a problem. This is solved by putting N.D. filters in front of the TV camera. In addition, a shutter protection circuit is used to turn off the image when no TV scanning is being done or if the light level into the camera is too great.

1.3.3 Buffer memories - BM

Although a histologic slide can be used as a random access read only memory, it is inadequate for use in processing the information it contains. It is necessary to retain images and their transforms in buffer memories, and to be able to access them and display them rapidly. In order to do this our Real Time Picture Processor needs facilities to store at least four entire images or transforms at once (e.g. to store a Fourier transform and a Fourier filter takes 4 images - 2 real and 2 imaginary), and the ability to access part of them very quickly. This rapid pixel accessing is the primary reason for the use of buffer memories.

Our approach has been to design buffer memories each of which can contain a reasonable size of working image (256x256 pixels). At a nominal maximum optical resolution of about 0.25 microns, a biologically usable picture window is approximately 50 microns or more corresponding to about 200 pixels. The RTPP uses a 256x256 pixel window.

Eight 256x256-word 16-bit/word gray scale buffer memories are being built which may be accessed by either the Quantimet video scanner/display, the GPP (general picture processor), or the PDP8e computer. Each buffer memory holds two 8-bit gray scale images, one in the low and one in the high half of the 16-bit BM word. When a BM is not reading or writing data to the GPP, it can be used to continuously recycle a picture for posting on the Quantimet display and/or as input to the Quantimet video-input chain using a fast 8-bit digital to analog converter. The PDP8e controls the scanning (into) and the posting of images (from) the buffer memories. In addition, the PDP8e computer can read/write data from/to buffer memories using rapid direct memory access (DMA) techniques. The BM can also be used as a binary data mask for Quantimet or other BM data.

A BM can acquire a selected 256x256 window of a QMT scanner image or an image loaded from the PDP8e. This image can then be accessed pixel by pixel or line by line from the GPP or PDP8e. Its contents may also be posted on the Quantimet display (using a fast D/A and digitized video multiplexing). When used in this mode, the QMT is able to process synthesized video. Using a direct memory access (DMA) channel, the PDP8e can then access the buffer memories. The GPP also can directly access the buffer memories for performing transformations on the image.

The buffer memories are organized so that it is most economical to transfer entire horizontal lines of data between them and the GPP processing unit. Vertical lines or randomly accessed pixels can also be transferred but at only 1/4 the data rate.

1.3.4 Triple line buffers for image addressing

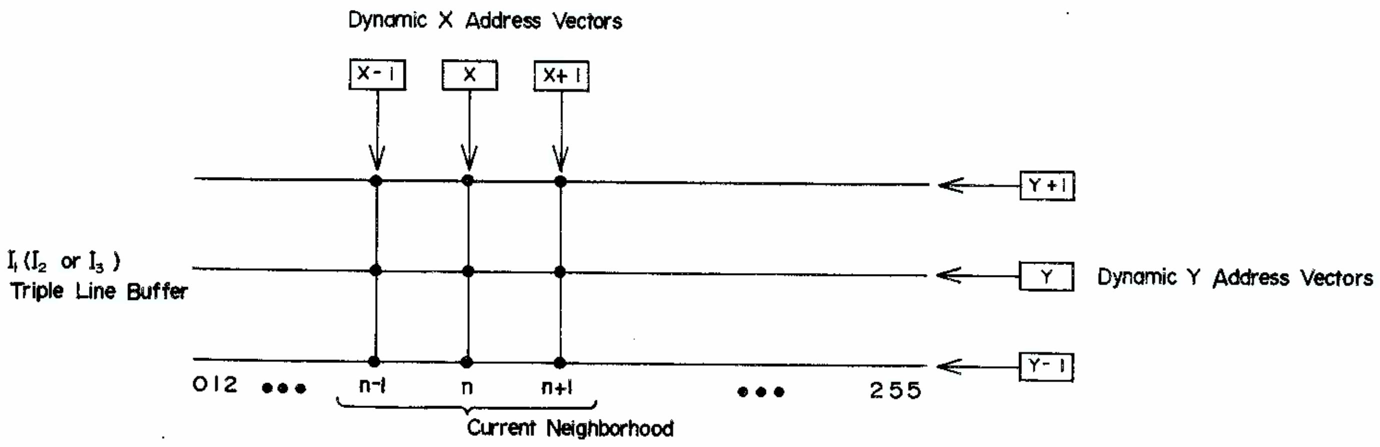
Given the basic architecture of the GPP, an addressing mode to implement the fast triple operand processing is needed. It was decided that the NxN neighborhood should be directly addressable for all I1(i), I2(j) and I3(k) in the three current NxN neighborhoods (i,j,k), letting the neighborhood be tessilated through the entire image.

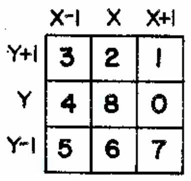
The NxN selected is a 3x3 neighborhood because of the combinatorial constraints which increase intolerably rapidly for any larger neighborhood. Three NxN neighborhoods are required for the images I1, I2 and I3; therefore 27 directly addressable pixels are needed for N=3. Forty-eight and 75 directly addressable pixels would be needed for N=4 and N=5 respectively. For any size N one could argue that some algorithm would need N+1. Since a 3x3 neighborhood is the minimum size that intrinsically handles 4 and 8 neighborhood processing (a symmetrical central pixel), at least a minimum of processing power exists with this size. If necessary N+1 can be simulated in software for any N. Therefore, it was decided to hold down the complexity of the hardware and let N=3.

Finally, to make an entire neighborhood directly addressable, the processor has three line-buffers, each capable of holding N entire lines of image data (N=3). Figure 1 illustrates an N-line buffer with K-bits/pixel and M pixels/line. In this design (N,K,M) = (3,16,256). Three triple line buffers are implemented.

The processor can transfer an entire line at a time either between a buffer memory and the line buffer or vice versa. The line buffers, being implemented with fast registers, constitute a kind of cache memory (a special type of hardware used to optimize data rates between a processor and its main data memory).

The problem of addressing a neighborhood is reduced to the problem of addressing the line-buffers. Thus, neighborhood processing of an entire image can be accomplished by a sequence of actions: first, processing is done on each 3-pixel-wide current neighborhood of an 3-line-deep line-buffer. Then a processed line (usually the oldest) is moved out of the line-buffer to a buffer memory and a new line, usually the next line in the raster source image, replaces it. The processing loop is then repeated, until the entire image has been processed. Three line buffers, providing 8 neighbor processing, maintain the full generality of level IV operations. A more complete description of the hardware is given later in this document.





	X-1	X	X+1
Y+1	3	2	1
Y	4	8	0
Y-1	5	6	7

GPP Neighborhood Addressing Map

Figure 1. GPP triple line buffer addressing

Associated with each line buffer is a set of three X and one Y dynamic address vectors. These dynamic address vectors define the current 3x3 neighborhood pixels in the line buffer. Therefore tessellation through different neighborhoods is easily performed by changing these address vectors. An instruction is available to selectively increment or decrement (X-1,X,X+1) in I1, I2, and I3 at one time thus effectively moving the current neighborhood to the left or right respectively.

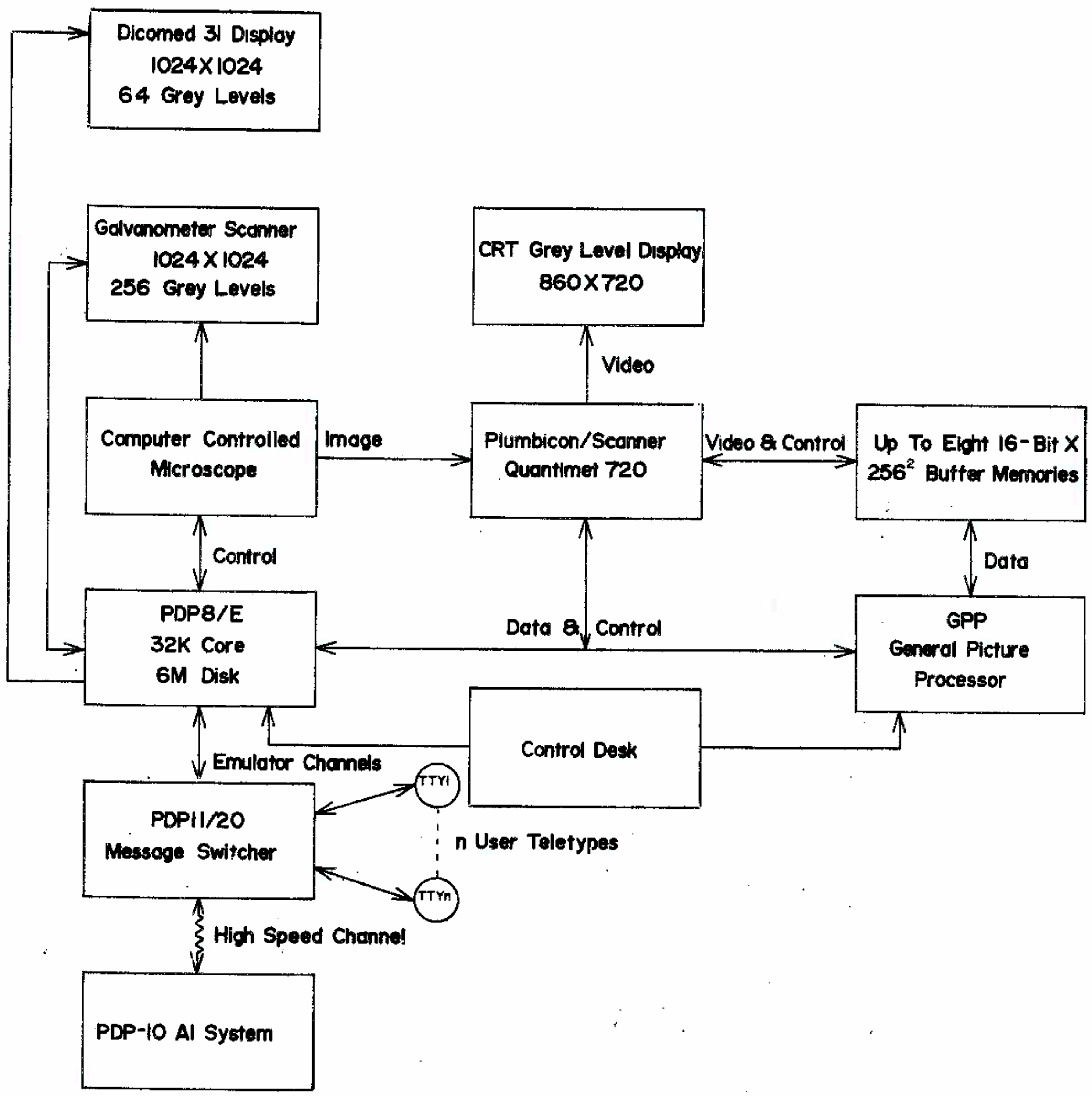
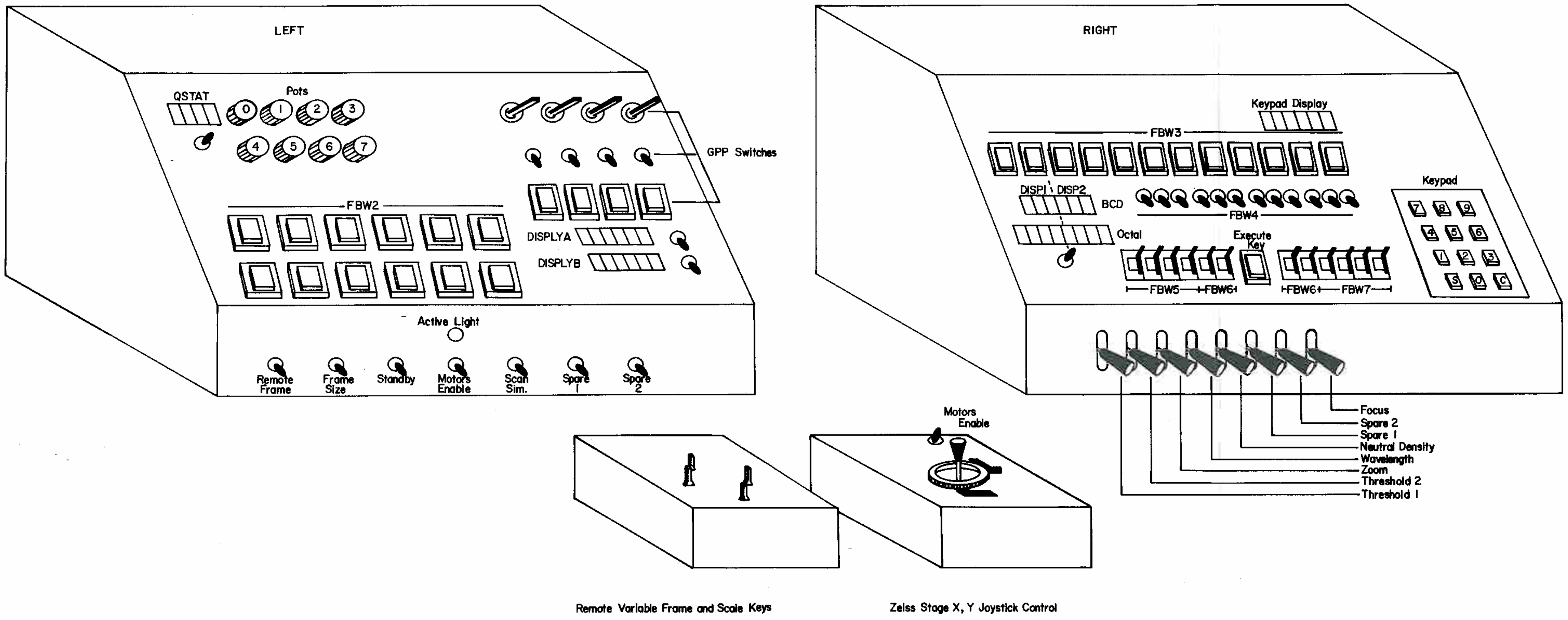


Figure 2. Block diagram of RTPP

The PDP8e computer directs the microscope stage to positions determined either manually by the operator or automatically by the PDP10 system. Images may be acquired by the buffer memories for processing by the GPP. Raw images as well as processed images may be displayed on the Quantimet 720. Precision scanning and display are implemented by the galvanometer mirror scanner and Dicomed display respectively. The user may interact with the system either through a control desk or a teletype.

Figure 3. Interactive control desk

The RTPP interactive control desk is situated next to the RTPP with the Quantimet video display to the rear of the control desk and the Axiomat microscope off to the side. A Graf-Pen spark tablet is located immediately in front of the operator with the pushbuttons and lights mounted in two large boxes to the left and right. The remote Quantimet variable frame and scale keys are located in a small box with a movable cable as is joystick for the Zeiss Axiomat (x,y) stepping stage. The latter has a long cable and may be used at the microscope for control of the stage while viewing thru the eyepiece of the microscope. The control desk controls are listed as follows going from left to right and top to bottom (for the left box first): the QSTAT lights indicate the status of the Quantimet interface; the pots are connected to A/D channels in both the PDP8e and the GPP; the GPP switches are read by the SWITCHA register; FBW2 are lighted "command" keys for the PDP8e; DISPLA/B are GPP display registers. the remove frame switch enables the remove frame and scale switches even when the PDP8e has not enabled them; the frame size switch freezes the frame and scale sizes so that a frame of fixed size may be moved around; the standby switch places the Quantimet display and system control in standby mode; the motors enable switch (also in joystick box) enables the stepping motors when the light above it is on; the scan simulation switch moves the galvanometer scanner independently of the PDP8e so that the gain/baseline controls may be setup with using the PDP8e. For the right control box, the controls are: keypad display of keypad input for the PDP8e; FBW3 "classification" keys for the PDP8e; DISP1/2 PDP8e display lights which are decoded as BCD in the top lights and as octal in the bottom lights; FBW4 PDP8e toggle switches; keypad to input 6 BCD digits to the PDP8e; FBW5/6/7 PDP8e octal digiswitches; Execute key used to execute (interpretively by the PDP8e) instructions given in the digiswitches; eight 5-position spring loaded toggle switches control various stepping motors with a fast and slow speed in both forward and reverse directions.


LEFT
QSTAT
Pots
0
1
2
3
4
5
6
7
GPP Switches
FBW2
DISPLYA
DISPLYB
Active Light
Remote Frame
Frame Size
Standby
Motors Enable
Scan Sim.
Spare 1
Spare 2
RIGHT
Keypad Display
FBW3
Keypad
7
8
9
4
5
6
1
2
3
S
0
C
DISP1
DISP2
BCD
FBW4
Octal
Execute Key
FBW5
FBW6
FBW6
FBW7
Focus
Spare 2
Spare 1
Neutral Density
Wavelength
Zoom
Threshold 2
Threshold 1
Motors Enable
Remote Variable Frame and Scale Keys
Zeiss Stage X, Y Joystick Control

1.3.5 General Picture Processor

The General Picture Processor (GPP) is a stored program very fast serial processor which can rapidly access and process image data through current picture neighborhoods. It uses the concept of triple operand instructions on up to 3 different current neighborhoods. These current neighborhoods may be taken from three diffent triple line cache memory buffers which in turn are backed by the slower buffer memories.

The programs which the GPP executes are loaded into a GPP program memory (PM) by the PDP8e which controls the GPP as a peripheral device. The GPP programs will be written and compiled on the the PDP10 system and assembled on the PDP8e. The PDP8e interacting with the PDP10 will be able to load the binary compilation files into the GPP or to save (get) these files on its own local disk for later use.

The PDP10 system, which is described subsequently in other documents ([Lem76b], [Lem76c], [Lem76d]), and in sections (1.4-1.5,1.7) is written in PDP10 SAIL [VanL73] language. It consists of a high level procedural description language PRDL [Lem76b] which will be used interactively to build morphological descriptions of biological images. PRDL will cause the low level picture processing functions to be evaluated on the RTPP rather than on the PDP10. An image processing program PROC10 [Lem76d] is currently being used on the PDP10 to emulate various image processing procedures which could run on the RTPP. Various often called procedures such as fetching a picture neighborhood have been shown (using PROC10) to be useful image processing primitive operations which are time consuming using normal PDP10 instructions. Thus such operations are committed to GPP hardware instructions (both through actual hardware and through microprogram implementation).

Running RTPP programs will be accomplished by having PRDL functions for the RTPP be compiled by the MAINSAIL cross-compiler [Wil75] on the PDP10. The output of MAINSAIL is then sent to the PDP8e where it is assembled by the RTPP assembler GPPASM [Gros76a] on the PDP8e. Thus PRDL functions which run on the RTPP can be constructed and later be called by PRDL through the RTPP monitor DDTG [Lem76a] (see section 1.6). The communication between these various processors must be intimate and the ultimate responsibility for insuring this lies with the PRDL system.

1.3.6 Quantimet controller

The interface between the Quantimet and the PDP8e consists of an interface to "program" the Quantimet (e.g. use thresholded video (by some criteria) to measure the total area); acquire QMT data; display numbers on the right QMT number display; load various QMT detector threshold and sizing limit registers and "live frame" frame and scale window coordinates.

In addition, special hardware is used to acquire a set of 2-properties/object for all objects in the scene (up to 1024 objects). The properties are taken from the set of object features computed by the intrinsic Quantimet Function Computer modules. These properties include: integrated density, area, perimeter, horizontal and vertial projections, and horizontal and vertical ferets. This acquisition is accomplished during a single scan of 0.1 second. The additional hardware constructed by us includes, a 1024 datum deep stack of the bottom most end points of blobs called Anti-Cooincidence-Points (X-ACP, Y-ACP); a stack of 1 bit/word detector-bit of the associated detected level; and a associated dual data stack which accepts two QMT data words from the function computers of 6 BCD digits each.

At the occurence of each ACP (which corresponds to the recognition of a discrete blob) all the above stacks have data pushed into them. The PDP8e can unload the data after it has been acquired by the Quantimet. In addition, data can be pushed on matching the (X,Y) coordinates of the QMT with those of the front of the (X-ACP, Y-ACP) stack. Since the ACP stack may be loaded by the PDP8e, this offers further possibilities for data acquisition by measuring the detection at the coordinates of the synthetic ACPs.

Other Quantimet related hardware includes a specially constructed Mask register for acquiring and, storing and using arbitrary convex shapes and a programmable display cursor. Both of these devices are described in more detail in Section 2.

1.3.7 Control of the RTPP by the PDP8e

The RTPP complex, (Figure 2), under control from the PDP8e minicomputer, may position the active microscope elements (eg. focus, stage, zoom, etc) and adjust the live frame image of the Quantimet according to program requirements or in response to the operator's directions at the control desk. The PDP8e has full control over the Quantimet to program it and acquire single data scans. Live frame video (within specified 256x256 windows) may be stored in any or all eight buffer memories, and displayed on the Quantimet at the direction of the PDP8e. To process image data in the buffer memories, the

PDP8e loads a specific program into the instruction memory of the GPP and initiates execution. Results may be passed back to the PDP8e for storage or transfer to other computers, or displayed on the Quantimet, or both.

1.4 The RTPP as a picture processing peripheral

The RTPP is really a specialized peripheral processor, a level of complexity beyond the now accepted class of display controllers. The latter usually consists of a programmable special purpose computer with a moderate amount of memory and a digital image display. The display controller is intended to remove from its large general purpose host computer a large portion of the burden of interactive display programs. In terms of the host computer and its associated facilities there can be little doubt that such a solution to the display problem is usually economically and computationally justified.

It may be noted that the RTPP, as a picture processing peripheral, removes a proportionately greater burden from the host computer's CPU and core memory than does the display controller. There is no need now to devote as much channel capacity for the transmission of raw picture data and their transforms. Instead, a system of distributed computing, made possible by the RTPP, allows the exchange of higher level data such as property lists, effectively resulting in marked information compression. This in turn allows more effective use of a wide range of facilities available on the general purpose host computer.

RTPP output would usually be images and/or lists of properties of objects contained in the images. The major component of the RTPP real time interaction is the General Picture Processor (GPP). This hardware processor allows extremely rapid serial digital processing of digitized gray scale images using special purpose hardware including image buffer memories and fast addressing schemes. Because the GPP/image buffer memories handle gray scale data (not merely binary black and white images) a larger set of useful picture functions can be quickly designed and executed than with a strictly binary image processor.

1.5 Cell Modeling System - CELMOD

A biological cell relational data modeling system will analyze the scene using RTPP generated picture primitives as in ([Lem72], [ShapB74]) and involves the use of procedural definition on a semantic data base. This modelling system is called CELMOD [Lem76c]. It consists of the three PDP10 programs (PRDL [Lem76b], PROC10 [Lem76c] and MAINSAIL [Wil75]) and the RTPP. A block diagram of CELMOD is given in Figure 4.

This model will be interactive to allow a biologist to define what he means by a particular class of cells and manipulate properties and relations of objects in the images. The biologist by means of iterative composition of processes can build up explicit connections between his semantic information and the image information represented at the individual pixel level. Being an interactive system, it will be easier for him to elicit the subconscious clues that he uses in making these decisions.

The decription language is called the Procedural Description Language (PRDL) which runs on a PDP10 and is described in [Lem76b]. The RTPP will thus also serve as a feature extractor for PRDL. Figure 4 shows a block diagram of the CELMOD system. The overall connection between the RTPP and PRDL is described in more detail in [Lem76b].

The CELMOD system will be able to perform image processing operations on the PDP10 using PROC10 while the RTPP is being constructed.

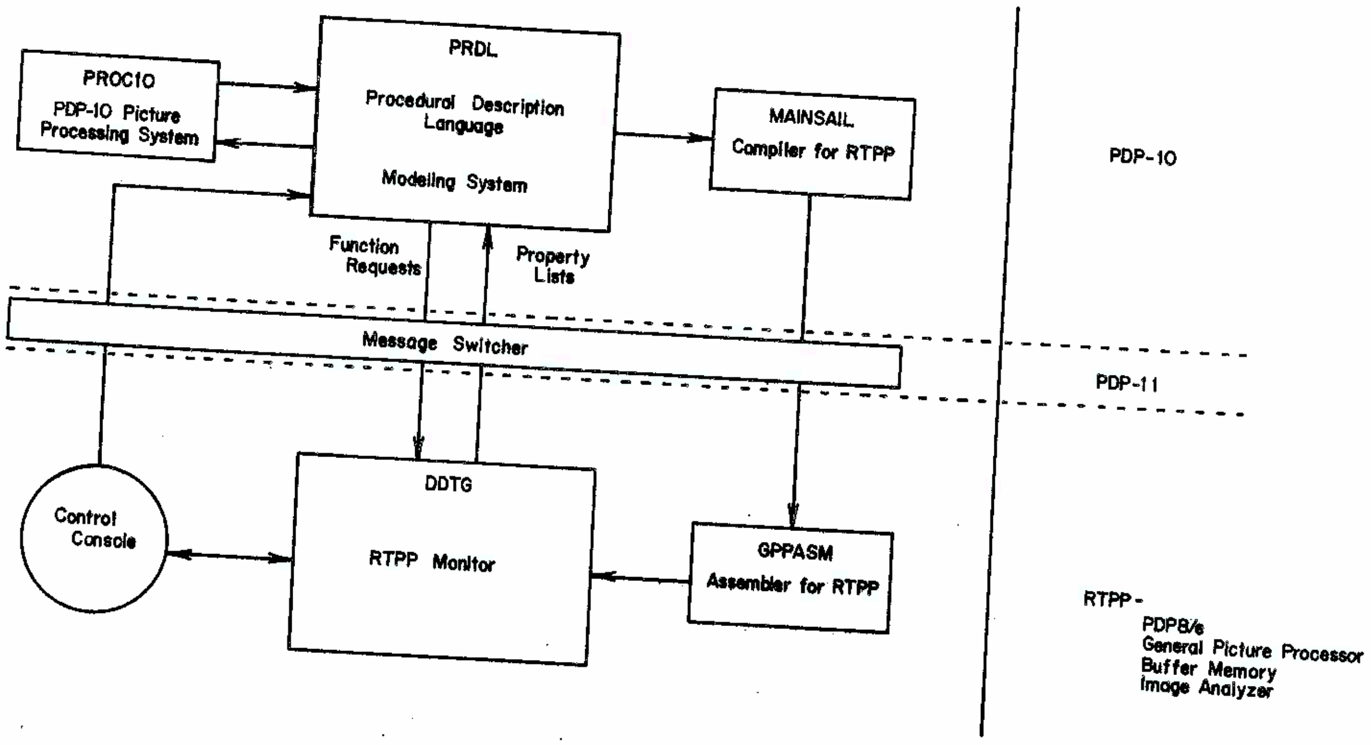


Figure 4. CELMOD system block diagram

CELMOD is composed of two major software systems running on different machines. The PRDL program runs on a DECSYSTEM-10 and is used to model cell images assuming the features are extracted. The RTPP has a resident monitor called DDTG which executes functions requested by the PRDL system and returns feature lists. The user sits at the RTPP control desk communicates with PRDL through a GT40 line graphics-teletype video terminal. The microscope data are visible on the RTPP display. The connection between the two systems is made via a PDP11/20 message switcher which is an operationally invisible high speed link between the two systems.

1.6 DDTG - the RTPP debugger/monitor

DDTG [Lem76a], a monitor/debugger is constructed for user and/or computer control of the RTPP. Functionally, DDTG is more than a simple combination of monitor and debugging facilities. As the RTPP operating system, it is required to interpret direct (i.e. via control console) user commands, or a string of commands generated by user-PDP10 interaction. In addition, it is required to provide access and control (at machine language word level) of major memory structures (PDP8e core, GPP program memory, GPP general register memory (GR) and buffer memories). It provides full control for a variety of image acquisition and low level analytic peripheral devices (e.g. Axiomat microscope stage, focus, etc. the Quantimet 720 and a variety of its plug in modules, a special mirror scanner, a sonic tablet, and a rapid scan spectrometer). Display control functions of DDTG also include the Dicomed and Quantimet displays.

DDTG has the ability to store, retrieve and execute (from PDP8e disks) PDP8e and/or GPP programs (user or PDP10 generated) (cf. Figure 4). Since in stand alone mode, DDTG is to be used as much by biologists as by computer scientists, the user interface allows many high level and seeming English command constructs. This in turn permits easy construction of understandable control programs for stand alone use, exploration and debugging at a very high level.

DDTG, written in standard PDP8e FortranII-Sabr consists of 4 major parts: an interpreter, parser, symbol table, and a large set of worker routines. The last include such features as loaders for the various component computers of the RTPP, as well as extensive and flexible disk I/O routines, a wide variety of stylized data structures.

DDTG has capabilities which allow it to load and run programs in the RTPP and to monitor their activity. An extensive set of commands are implemented to facilitate image data acquisition and display, and the running of small picture operation programs called "special segments". The General Picture Processor (GPP) portion of the RTPP will, in performing picture operations, require special segment support from DDTG. In addition, mechanisms are available for RTPP program interaction with OS/8 in order to facilitate the implementation of image processing programs (exclusive of DDTG) to process DDTG produced data.

1.7 RTPP Compiler/Assembler - MAINSAIL/GPPASM

The RTPP will have access to a very powerful cross-compiler MAINSAIL [Wil75] and a low leve assembler GPPASM [Gros76]. MAINSAIL will run on the PDP10 and compile a dialect of PDP10 SAIL which includes teletype I/O and records but not LEAP or file structured I/O. As the initial RTPP has no floating point hardware, the initial MAINSAIL to be used has no real arithmetic constructs. Registers (such as line buffers Ij(k)) will be represented as reserved symbols (Ijk). MAINSAIL will output assembly language source code for GPPASM.

GPPASM is a low level assembler which produces non-relocatable load modules. It has labels and limited arithmetic capabilities as it was designed to run efficiently on a PDP8e. This last aspect is important for the debugging and maintenance of the RTPP. The GPPLDR which loads the absolute binary files produced by GPPASM is part of DDTG. GPPASM permits the use of REQUIRE <file> LOAD or SOURCE statements and thus a run time library can be assembled or loaded with main programs.

The GPPASM grammar was designed to be easily parsed by a simple finite state acceptor with a small auxillary stack. The assembler has either 2 or 3 passes depending on whether an assembly listing file is to be generated. The first pass incorporates declaration, PM and GR label resolution; the 2nd pass generates the PM and GR output code segments while the 3rd pass optionally generates an assembly listing. The PDP8e special (PAL8) segment is stripped out during the 1st pass and is later assembled by a modified version of PAL8 and concatinated with the PM and GR segments. The GPPLDR in DDTG is able to analyze and load such GPP binary load files. The BNF grammar for the GPPASM assembly language is given in Appendix E.

1.8 GPP Microprogram Assembler - MICROP

The actual GPP implementation employs microprogram control. The microprogram is stored in a 60-bitx4K RAM by the PDP8e. Microprograms are assembled on the PDP8e using the MICROP assembler [Gros76b]. The microassembler is discussed in more detail in Section 6.1.1.

SECTION 2

Quantimet subsystem

The Quantimet 720 [Fish71] is a modular low level image processor which is used primarily as an I/O device rather than as an stand alone image processor. In addition to the basic scanner and display modules of the QMT, other hardware modules are used to accomplish elementary functions such as object area as determined by thresholded detected video: perimeter, number of objects in a field, integrated density under a mask, and several intercept properties. The Quantimet functions are controlled and programmed by the PDP8e minicomputer or may be used as a stand alone device.

The Quantimet part of the Real Time Picture Processor consists of the following Imanco Quantimet 720 modules:

Plumbicon (or vidicon) non-interlaced TV raster scanner

System Control module

System Display module

Variable Frame and Scale module

Standard Detector module

Digitizer/Densitometer module

1-Dimensional detector module

Amender module

Standard Computer module

MS3 Computer module

2 Function Computer modules (with Density option)

Light pen module

Classifier Collector module

The Classifier Collector module is added primarily for maintenance of the Function Computer modules.

2.1 Quantimet modules

The basic functions of these modules are enumerated here as an aid in understanding the design of the real time picture processor system. For further details see [Fish71]. The basic design for some of the Quantimet modifications was done initially for the NCI Grain Counter-I [LipL74].

2.1.1 Scanner - System Control module

The TV scanner (either a Plumbicon or vidicon camera) and system control produce a 10.1/second 880x720 digitally derived TV image. Actually, a smaller window is used which is called the big frame consisting of 860x680 lines. Inside the big frame is still a smaller frame called the live frame. Data inside the live frame is the data which is used by the rest of the Quantimet. The live frame is derived from many sources, some of which are the variable frame and scale, detected video, light pen mask output, buffer memory derived detected video and computed video of the various modules.

The TV camera is physically connected to the Zeiss Axiomat microscope through an IPU designed and constructed mechanical interface. This interface includes a Zeiss electromechanical shutter which is closed either by the computer (which can also open it) or by an automatic shut-off circuit which senses too much light entering the Quantimet camera. This latter feature prevents camera damage with too much light. The interface also has room for several neutral density filters so that the light levels can be balanced for running both the galvanometer scanner and the Quantimet camera at the same time.

2.1.2 System display module

The system display module is used to mix the system input video with the output display signals from the Standard computer, MS3 computer, Amender, Frame and Scale, 1-D detector, Digitizing detector, Function computers, Classifier Collector, and light pen modules. These display signals will either fully intensify or partially blank the system display. This mixed video signal is then displayed on a 10.1 rasters/second orange phosphor TV monitor. This phosphor is designed for slow decay times to reduce flicker. However, it does so at cost to the gray scale resolution.

The RTPP will be able to blank out the scanner video and substitute synthesized video in its place inside of buffer memory 256x256 pixel windows. This is discussed in more detail in Section 3 on the buffer memories. The Quantimet display cursor and mask register displays are added to the Quantimet display inputs (Scale mixer knob and Guard mixer knobs respectively).

2.1.3 Variable Frame and Scale module

The Variable Frame and Scale module generates a live Variable Frame by generating a rectangular computing window specified by (hor-position, hor-size, vert-position, vert-size). When in variable frame mode, any data inside this window will be enabled for detection by the Quantimet hardware; otherwise detection takes place within the standard live frame.

2.1.4 Detector modules

There are three detector modules: the Digitizer/Densitometer, the 1-D detector and the Standard detector modules. The Digitizer can be used as a detector (high speed comparator) to transform the analog video into a detected/not-detected digital signal according to the selected threshold values. The 1-D detector performs auto thresholding in 1 dimension (X-axis) to detect objects in a background phase. The Standard detector performs simple thresholding. The Digitizer and 1D detectors have a 64 gray level range while the Standard Detector maps up the white to black range into 4096 divisions. The thresholds in the Digitizer/detector and Standard Detector have been modified so as to be able to be loaded by the PDP8e.

2.1.5 Light pen

The light pen module will generate a mask of a detected object selected with the light pen. There is a restriction that the object be vertically convex so as to generate the complete mask.

2.1.6 Digitizer/densitometer

The Digitizer/densitometer also functions as a densitometer. It integrates detected gray scale data in the range of [0:63]. It does this by using a fast 125 nanosecond A/D converter and taking an optional log of the signal. It gives relative density directly. Using this module one can perform densitometry when suitably calibrated.

2.1.7 MS3 Computer

The MS3 computer module uses the detected video from the detector modules to compute global area, perimeter, intercept and count. The "pattern recognition" mode of operation activates two Function computer modules for acquiring object specific data.

2.1.8 Function Computer module

The Function computer computes for each detected object area, integrated gray-value or log-gray-value, perimeter, vertical projection, horizontal projection, horizontal feret, and vertical feret. An object is marked by the occurance of its anti-coincidence point (ACP). Data is not acquired for partially represented objects (in which the ACP falls outside of the live frame).

2.1.9 Classifier Collector module

A Classifier Collector module may be used to accept 2 Function Computer module outputs for stand alone operation. It is also able to compare function values from the function computer output against a range of values to determine whether the ACP (object label) should be used.

2.1.10 Standard Computer module

The Standard Computer module is similar to the MS3 Computer except that it does not compute perimeter and has no pattern recognition mode.

2.2 Microprogramming QMT modules

Many of the QMT modules mentioned above may be microprogrammed to select either modes of operation for functions to compute. This is done by enabling a rear status register on each of the modules called the "programmer" input. On the RTPP this is done by loading, for each QMT scan, a program word which consists of eight 12-bit status words called the Quantimet Program registers denoted QPROG1 through QPROG8. These registers and their allocation in the Quantimet subsystem is discussed in Appendix C.3.

2.3 PDP8e control of the Quantimet

The Quantimet may be controlled by the PDP8e in a single step data acquistion mode. When the System Control module 'Continuous/Auto' switch is put in 'Auto', it is placed under control of the PDP8e. Essentially, PDP8e control is effected by manipulation of the eight 12-bit static programming words and the QSTAT status register. The PDP8e STQMT instruction will start the Quantimet data acquistion when the Quantimet scanner reaches the start (top) of the next scan frame. The QMSKP skip instruction can be used to test when the data acquisition is done. At this time, the whole field Quantimet data is available to the PDP8e in the QDAT1, QDAT2, and QDAT3 input instructions which will read 7 BCD digits (LSD to MSD) into the PDP8e accumulator. This whole field data is displayed automatically in the left Quantimet display. The right Quantimet display may be loaded by the PDP8e load instructions LQDT1, LQDT2 and LQDT1 (MSD to LSD).

Function computer data as well as ACP (x,y) coordinates may be acquired in a single Quantimet scan through a 1024 word (69-bits/word) shift register. This is described in more detail in Appendix C.1.2.

2.3.1 Quantimet status register - QSTAT

A status register QSTAT controls operation of the Quantimet/PDP8e interface and various options on the Quantimet. This control includes enabling the Frame and Scale control desk switches, shift register data acquistion system for Function Computer data, standby and camera shutter activation. These are discussed in Appendix C.1.7.

2.4 The mask register module

Detected video is used to generate a 720 picture line Quantimet mask of entry and exit line intercept positions stored in a hardware Mask register. This Mask register can be used to supply the QMT with a detection region mask by reading the mask as the QMT goes through a scan. The mask register is a (1024 word) RAM which is loaded either from with the QMT detected video on a command from the PDP8e (GETMSK) or from the PDP8e directly. On the PDP8e issuing the GETMSK instruction, detected video is used to determine "DET-ENTRY" and "DET-EXIT" for the entire QMT live frame.

When the mask register is not being loaded, it cycles with the QMT and outputs a (QPROG selected) mask of either variable frame, mask or various logical comginations of variable frame and mask (see QPROG2[0:3]). The QPROG2 selected mask display intensity is controlled by the display "guard" knob. A display of the Mask register output alone (without the frame and scale) is also available and is selected by QPROG7[0:1]. The display intensity for this display is controlled by the display "scale and figures" knob.

The addressing in the mask register is the same as that of the frame and scale window . The QMT VTRIG, SYNC, (HTRIG), and Clock signals are used to generate the actual addresses. The entrance/exit acquisition algorithm works as follows. When detected video first goes true (enter a solid blob) the "X" coordinate at the Y'th line inside the frame is entered into the Y'th (0 to 719) "DET-ENTER" of the mask register. When the detected video first goes false again (leaving a blob), the "DET-EXIT" "X" coordinate is entered for the Y'th line. By default a line's "DET-ENTER" when "getting" a mask is assumed to , be 1023 before a line is processed. This takes care of the case where no data or only entry data is present. Note that only the first object in a line is detected and that objects with concave inward tops or bottoms will not be acquired properly because horizontal slices of such objects have multiple entrance/exit points.

The mask register can be loaded or read from the PDP8e. The main use of the mask register would be in masking QMT data with synthesized masks or in accessing actual perimeter X-Y coordinates.

Control of the mask register from the PDP8e is done with the GETMSK, MSKADR, RMASKE, RMASKX, LMASKE, and LMASKX PDP8e instructions. GETMSK enables the acquiring the next QMT detected video mask into the mask register. Now doing a STQMT command starts the actual mask acquisition from the input detected video. MSKADR loads the line number for subsequent I/O. Then RMASK- and LMASK- do I/O on that MSKADR's entrance/exit pixels.

Note that RMASK- and LMASK- are allowed at any time except during mask acqusition. the mask generated for an object will be a fairly good approximation to its boundry given that the object has no concave upward or downward regions. The following OS8 Fortran II program will get a mask from the detected video inside the current frame and save it in the PDP8e memory.

```
        DIMENSION IENTER(720),IEXIT(720)
S       GETMSK /enable a mask to be accessed
S       STQMT /start mask acquisition
S NOTDONE,      QMSKP
S       JMP NOTDONE
        DO 100 I=1,720
S       TAD  I
S       MSKADR
S       RMASKE
S       DCA  JENTER
S       RMASKX
S       DCA  JEXIT
        IENTER(I)=JENTER
100     IEXIT(I)=JEXIT
```

2.5 QMT cursor

An X-Y cursor is available which can be loaded from the PDP8e using the same coordinate system as the Frame and Scale and mask register devices. It appears as a 10 pixels line segment to the right on the actual (x,y) position on the "scale" display control. It is programmed in DDTG software to be used with the Graf-Pen to track the pen on the QMT screen whether the pen is actually using the data or not. To make the cursor disappear, load XP or YP. The PDP8e commands LDXP and LDYP load a binary coordinate pair into the cursor controller. The cursor display intensity on the Quantimet display is controlled by the "scale and figures" knob.

SECTION 3

Buffer memory

The RTPP is designed to use up to eight gray scale buffer memories (BM). The memories may be simultaneously selected by the PDP8e for posting of images (displaying on the Quantimet display) or acquiring QMT scanner data. However, there is a restriction that no two active BM windows can intersect during display or scanning as data will be lost from one of the memories.

A BM consists of 65K (256x256 pixels) 16-bit/words with gray scale data being stored in both the high and low 8-bit bytes of a 16-bit word. Binary detected video images are stored as signed 8-bit bytes. Since each BM is synchronized with the QMT independently, up to 512 X 1024 pixels "could" be posted in real-time (if the Quantimet display were large enough). Usually, a 256x256 window will be displayed and the other BMs may contain variants of that displayed window. One may select the high or low 8-bit bytes in a given buffer memory. Each 8-bit slice may then be used to store separate images or alternatively, for example the high and low 8-bit fields may be used to store the real and immaginary portions of a Fourier transform.

Synthesized Quantimet video

The QMT video (level adjusted by the system control module) is digitized using the fast A/D converter to 8 bits (using 8-bits of the 9-bit Computer Labs model 7910 10MHZ A/D). This digitized image is them multiplexed digitally with the buffer memory data to be synthesized as Quantimet input video - that is substituting buffer memory data for Quantimet TV scanner data. The digitized video is then resynthesized using a fast D/A converter (Computer Labs RDA-0815A, 15 MHZ).

Synthesized video, selected for display posting, is substituted in the window (Quantimet active frame) [[XB:XB+255],[YB:YB+255]] for the QMT scanner video input to the rest of the QMT. This results from our modification of the video train which allows the QMT to process synthesized video!

In order to synthesize QMT video, the BM first synchronizes with the QMT. That is, it waits until the QMT reaches the (XB,YB) pixel and then starts dumping 256 pixels into the digital video multiplexer. This digitized video is then converted with a fast D/A to analogue video for Quantimet video input. Successive lines are then synchronized similarly.

GPP use of BMs

The GPP can perform I/O with the BMs. That is, BMs are used for storing and retrieving intermediate pictures while doing picture processing. This is done synchronously by line after the line's row or column address is specified by the GPP LINE instruction. The GPP can also random access the BM in single word or neighborhood mode.

It is possible for two different devices to access two different BM's simultaneously under certain conditions. The eight BMs are divided into parallel memory systems (group A - BMs 0 to 3, and group B - BMs 4 to 7). Provided each device access is to a different group of 4 memories then parallel access is possible. The BM access priorities from highest to lowest are: BM input from QMT scanner, GPP I/O, PDP8e I/O, BM posting on the QMT display.

If two devices require I/O on the same BM group memories simultaneously, then the highest priority device obtains access. For example, if the GPP requests I/O during a posting on the QMT sequence, then the "would be" BM posted data becomes the actual QMT scanner data. If the scanner data were a dark object then the resulting posted image would have light holes in it representing a higher priority access conflict.

BM scan and display selection

Two types of images may be acquired: 8-bit grayscale and binary images. Binary images are stored in the buffer memories as signed 8-bit bytes. Images are stored and used by selecting a high or low byte to be accessed. PDP8e commands GETA, GETB, POSTA, and POSTB perform these control functions.

They all load 12-bit command registers (from the PDP8e AC) which selects which byte to use, which buffer memories to enable, and whether to perform gray scale or binary (leading bit of a byte) I/O. The actual scan is started with the STQMT operation. The command register format (the same for all four GET-/POST- instructions) is as follows:

```
[0:3] - (0) use low byte for pix, high byte for binary mask
        (1) use high byte for pix, low byte for binary mask

[4:7] - (0) use gray scale data
        (1) use binary mask data

[8:11] - (0) don't select a BM
         (1) select a BM
```

BM window and scan acquisition

A buffer memory can accept a 256 X 256 x 8-bit gray scale image from the Quantimet video (digitized via Computer Labs 8-bit A/D) starting at the upper left hand corner coordinates denoted (XB,YB). This is done in less than one QMT scan cycle of 0.1 second. During this time the BM group is unavailiable to both the GPP and the synthesized video generator But other BM I/O operations, GPP or PDP8e I/O, may take place. The lesser priority operation will be disconnected only during the part of the horizontal scan line that the BM recieves data from the QMT. All other lines in that BM group during this 'GET' function are available for lower priority I/O except posting.

At the end of acquiring the QMT image, the BM is disconnected from the digitizing video channel. During a scan, the BM accepts 256 sequential pixels/line starting at the X coordinate XB of each line for each of 256 lines beginning with line YB. The BM window so defined is determined by comparing (XB,YB) with the real-time (xreal,yreal) coordinates. Each BM has a pair of window coordinates (XB(i), YB(i)) which may be loaded by the PDP8e. The eight window coordinate pairs are loaded from the PDP8e by the commands (BMXi,BMYi) where i=[0:7].

In addition, the detected Quantimet video may be acquired as a binary mask into buffer memories using the GETA/B command with the appropriate bits [4:7] set. It may be inserted into the Quantimet frame input using the POSTA/B instruction with the appropriate bits [4:7] set. The resulting mask may be used for further Quantimet data acquistion.

It should be noted to avoid confusion that BM window positions are independent of the frame and scale window position.

PDP8e accessing of BM data

The PDP8e can transfer up to 4K PDP8e 12-bit words using direct memory accessing (DMA) to/from a BM. Thus the transfer of a complete BMs contents would take several DMA transfers. Four PDP8e/BM DMA packing modes are available, all of which optimize the 12/16 bit word size differential. The first 2 modes transfer three 8-bit bytes (either high or low byte) in three BM words into two (OS8 packed format) PDP8e words. The third mode transfers three 16-bit BM words in four (OS8 packed format) PDP8e words. The fourth mode transfers 16-bit BM data in two PDP8e 12-bit words in packed sign-extended EAE (extended arithmetic element of the PDP8e) double precision format so that the PDP8e can do arithmetic on full BM words without extensive packing and unpacking.

3.1 Physical BM memory addressing

Although it is unnecessary to understand the physical implementation of the BMs to use them, the underlying structure is presented here for those who are interested. The 19-bit BM address specifies a buffer memory location and may be broken down into the concatination (&) of subaddresses: UBM[0:2]&YXADDR[0:15].

UBM[0:2] selects a BM unit.

Within a BM, YXADDR selects the data:

```
        YXADDR[0:15]=YADDR[0:7]&XADDR[0:7].
```

Then the

```
        row address = YADDR[0:7],
        Column address = XADDR[0:7].
```

Each buffer memory is divided into four physical memory cards. YADDR[0:1] selects the memory card. The BM word size is 16-bits. The memory cards are four way interleaved so that 64-bits of data may be transfered in one 500 nsec memory cycle. The buffer memory is constructed in such a way that only data addressed modulo XADDR[6:7]=00 are obtained in one memory cycle. Therefore, random addressing of BM words requires 500 nsec/word minimum; wherea horizontal raster or line by line transfer (i.e . continuous incrementing of YXADDR) will result in an effective 125 nsec./word transfer rate.

3.1.1 BM controller accessing priorities

During a BM-QMT scan acquisition, the posting of data to the QMT is discontinued during that entire scan for that respective BM group. For example, posting of BM data from one BM group may take place during the same scan that acquisition of QMT video data by the other BM group. In fact, video data being posted by one BM group may be loaded into the second BM group during the acquisition scan.

The (XB(i), YB(i)) position for each buffer memory is used to select where the data shall be taken and the image posted on the display. This allows the operator to position the transform window exactly where it is needed to operate on data. The hardware priority networks resolves conflicts if two or more BM windows intersect (in case the software does not prevent their intersection).

If while posting BM video data on the QMT a conflict occurs with BM I/O other than acquiring QMT video, then during the duration of the conflict the origonal scanner video is posted rather than the BM video.

SECTION 4

Triple line buffering

As was discussed in the introduction, it is important that current neighborhood addressing be fast and efficient. To make maximal use of the triple operand instruction scheme in processing current neighborhoods, data from the image buffer memories is transfered to a cache line-buffer shown in figures 1 and 5. The GPP contains three cache line-buffers named I1, I2 and I3 respectively.

Each of the line buffers stores three lines (256 picture pixels/each) of image data usually derived from a buffer memory. In order to facilitate local 3x3 neighborhood processing, each line buffer has its own dynamic address vectors (X-1,X,X+1) which are simply a set of three 8-bit address vector registers as shown in figure 5. The current neighborhood is the 3x3 neighborhood currently accessed through the dynamic address vectors. The dynamic address vectors are (IjY,IjXP,IjX,IjXM) for line buffers.

4.1 Programming the Triple Line Buffer

For example, if the leftmost neighborhood of a triple-line-buffer were to be accessed, the dynamic address vectors would be appropriately set to (-1,0,+1), thereby directly addressing each current 3x3 neighborhood pixel around the pixel X=0. This is done automatically using the instruction:

```
        XRST(<X-1,X,X+1>,<I1,I2,I3>);
```

which also resets a 256 pixel counter.

Each of the line buffers have associated with them nine direct addresses which are remapped by the appropriate dynamic address vectors to allow the instruction field to directly address each of the nine current neighborhood picture pixels. Instructions are availiable to selectively increment or decrement (X-1,X,X+1) of I1, I2, and I3 at one time thus moving the current neighborhoods right or left respectively.

```
        XCLK(<up/down>,<X-1,X,X+1>,<I1,I2,I3>);
or
   "done label" <== XCLKB(<up/down>,<X-1,X,X+1>,<I1,I2,I3>);
```

Similar instructions selectively advances the Y dynamic address vector in I1, I2 and I3 so that new lines may be easily added replacing the oldest lines with minimum program bookkeeping.

```
        YRST (<I1,I2,I3>);
```

resets the 256 line counter and ring (3-line) counter. Lines are advanced in the triple line buffer by the YCLK instruction:

```
        YCLK (<I1,I2,I3>);
or
   "256 line done label" <== YCLKB (<I1,I2,I3>);
```

A line L of BMj data is transfered to or from a triple line buffer Ik line Yl using the LINE instruction as follows:

```
        LINE (<high/low>,<in/out>,<hor/vert>,BMj,Ik,Yl), L;
```

The triple address instruction might address two of the triple line buffers as sources, (i.e. I1 and I2) and after the operation deposit the result in third line buffer, I3. After this operation is iterated for all the local neighborhoods in a line, the I3 line buffer data might be returned to one of the eight buffer memories. For example, consecutive lines of images from two different buffer memories may be loaded into the I1 and I2 line buffers and the picture processing algorithm repeats itself until all local neighborhoods of a line are processed. Then the resultant line in I3 is stored in one of the image buffer memories. Note that accessing horizontal lines is 4 times faster than accessing vertical lines.

This line by line processing continues until all lines have been processed. Then the buffer memory data resulting from the computations on I3 are available for display and futher processing by the GPP or QMT if desired.

4.2 Alternate Neighborhood Accessing Methods

In addition to accessing the line buffers through the current neighborhoods, the 9 lines in the 3 triple line buffers may be directly addressed as 16-bit addresses in the top 2304 decimal (9x256) addresses in the GPP general register (GR) address space. These addresses are given in Appendix B.

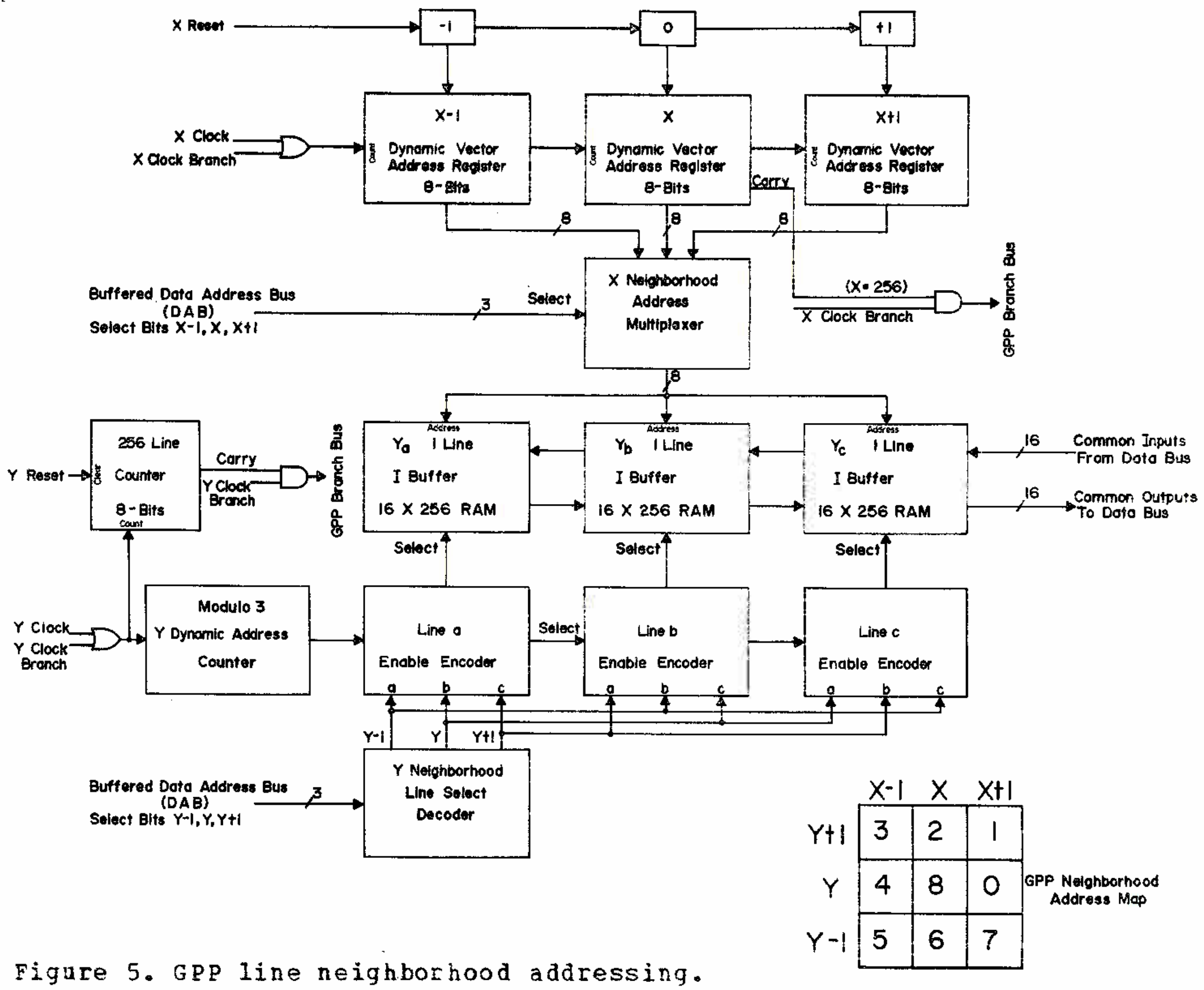
Alternatively, when random neighborhood accessing is required, it is possible to fetch 9 current neighborhood pixels into either I1, I2 or I3 using the GETI1, GETI2 or GETI3 instructions. The cost of fetching a random neighborhood is that each pixel is accessed at 500 nsec./pixel rather than the effective 125 nsec./pixel in the LINE raster I/O mode. The random neighborhood fetch is more efficient than the LINE when fewer neighborhoods are required and their relative locations are random (such as with a boundary follower). For example:

```
        GETI1 (<high/low>,<BMj>),YXADDR;
```

The instruction MAKYXADDR may be used to pack two 8-bit x and y BM address pointers into an effective 16-bit YXADDR necessary for GETIi instructions (as well as for the indirect BM pointers PBMi).

```
        YXADDR <== x MAKYXADDR y;
```

There are other ways than of using line buffers of accessing data in the BM for use by the GPP. In computing Fourier and other global transforms, it is necessary to operate on picture lines located far apart. This facility is implemented by the ability to random access and copy one or more random BM lines into the general registers, GR, and then operate from there. Random BM picture pixel words may be addressed by the indirect BM addresses PBM0 through PBM7 given in Appendix B.


X Reset
-1
0
+1
X Clock
X Clock Branch
X-1
Dynamic Vector Address Register 8-Bits
X
Dynamic Vector Address Register 8-Bits
X+1
Dynamic Vector Address Register 8-Bits
Count
Carry
8
Buffered Data Address Bus (DAB) Select Bits X-1, X, X+1
3
Select
X Neighborhood Address Multiplexer
(X=256)
X Clock Branch
GPP Branch Bus
Y Reset
256 Line Counter 8-Bits
Clear
Count
Carry
Y Clock Branch
GPP Branch Bus
Address
Y_a 1 Line I Buffer 16 X 256 RAM
Y_b 1 Line I Buffer 16 X 256 RAM
Y_c 1 Line I Buffer 16 X 256 RAM
16
Common Inputs From Data Bus
Common Outputs To Data Bus
Select
Y Clock
Y Clock Branch
Modulo 3 Y Dynamic Address Counter
Line a Enable Encoder
Line b Enable Encoder
Line c Enable Encoder
a b c
Y-1
Y
Y+1
Buffered Data Address Bus (DAB) Select Bits Y-1, Y, Y+1
Y Neighborhood Line Select Decoder
GPP Neighborhood Address Map

	X-1	X	X+1
Y+1	3	2	1
Y	4	8	0
Y-1	5	6	7

Figure 5. GPP line neighborhood addressing.

Line buffer addressing in the GPP. Associated with the three lines is an effective Y dynamic address used to order the three lines as to (Y-1,Y,Y+1). In reading a raster line pattern into the triple line buffer, the oldest line must be replaced with the newest line. Similarly, the other two lines need to be adjusted as (Y-1,Y,Y+1) ==> (Y,Y+1,Y+2). By selecting the effective line address with a modulo 3 dynamic Y address counter, a dynamic Y address vector can be implemented. This is similar to the concept of ring buffer. The three X dynamic address vectors point to a 3x3 neighborhood array in the line buffer. This neighborhood is called the current neighborhood. All of the dynamic address vectors are easily and efficiently programmed in the GPP to tessellate the current neighborhood along the three lines in the line buffer.

SECTION 5

GPP - general picture processor

The general picture processor, GPP, is primarily used to perform serially rapid neighborhood processing on image data stored in the buffer memories. The basic GPP structure is illustrated in Figure 6. The GPP is fully controlled by the PDP8e DDTG monitor system discussed in Section 1.6. GPP program memory (PM) is distinct from its data memory (general registers - GR) so as to more easily implement reentrant software.

The program counter (PC) bus is 16-bits wide which allows directly addressing 65K words of program memory. Thus a program address can address all of the program memory (via branch or jump instruction). Data addresses are also 16-bits wide so that all of the 65K word data address space may be directly addressed. A 60-bit GPP instruction (stored in the PM) is organized into 3 data address fields (P1, P2, and P3) and a 12-bit operation field.

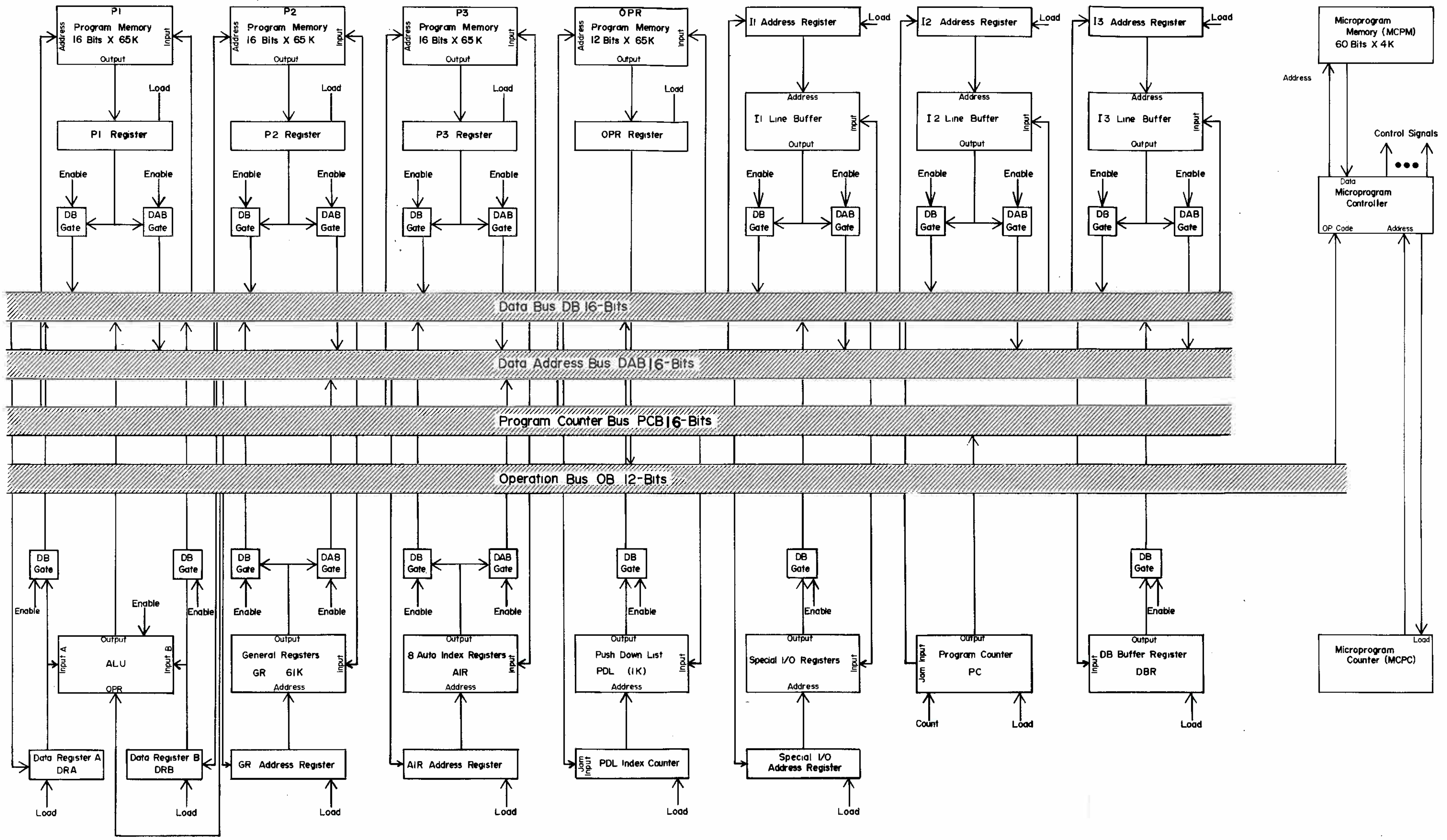
Because of the problem of packaging the high-speed PM 60-bit memory close enough so that the cable delays are not appreciable, the full 65K memory will not be built at this time without slowing down the basic cycle time of the GPP. Therefore, a 32K PM will be constructed in the initial implementation. A full 65K machine could be built as higher bit packing densty RAMs are introduced by the industry or by slowing down the cycle time of the machine.

The triple-address instruction format allows the GPP to rapidly access two sources operands in parallel, operate on them, and deposit the result in a sink operand with a single instruction. Parallel data fetches are performed when no data space addressing conflict is found. This conflict analysis is performed during instruction lookahead. An address field may be used to specify an immediate, direct, or indirect effective address depending on its associated addressing mode bits in the operator field.

This section defines the GPP architecture while Appendix A gives the detailed instruction set.

Figure 6. GPP bus structure

General picture processor bus structure. The instruction addressing sequence is done serially. That is, P1 is addressed, then P2, then P3. Let "c(.)" denote "contents of '.'". If any address is immediate no memory fetch is done. Rather, the PM data, c(P), is enabled onto the data bus DB. If direct addressing mode c(c(P)) is used, the PM data c(P) is enabled onto the data address bus, DAB, and then loaded into the approriate data field address register. The memory then enables its data, c(c(P)), onto the data bus, DB. If indirect mode is used then the same sequence is repeated as for direct mode, but c(c(P)) is enabled back onto the DAB instead of the DB. Then the data address field register addressed is loaded and the c(c(c(P))) from that memory is enabled onto the DB. A conflict may occur in the use of the indirect mode with the "MOVE" instruction. This is resolved by storing the source data in the data bus register, DBR, temporarily. Various devices and memories are connected to the bus structure and interact when the control section activates them. The average GPP instruction time is designed to be on the order of 250 nanoseconds.


P1 Program Memory 16 Bits X 65K
P2 Program Memory 16 Bits X 65 K
P3 Program Memory 16 Bits X 65 K
OPR Program Memory 12 Bits X 65K
Address
Input
Output
Load
P1 Register
P2 Register
P3 Register
OPR Register
Enable
DB Gate
DAB Gate
I1 Address Register
I2 Address Register
I3 Address Register
I1 Line Buffer
I2 Line Buffer
I3 Line Buffer
Microprogram Memory (MCPM) 60 Bits X 4K
Control Signals
Data
Microprogram Controller
OP Code
Data Bus DB 16-Bits
Data Address Bus DAB 16-Bits
Program Counter Bus PCB 16-Bits
Operation Bus OB 12-Bits
ALU
Input A
Input B
OPR
General Registers GR 61K
8 Auto Index Registers AIR
Push Down List PDL (1K)
Special I/O Registers
Program Counter PC
Jam Input
Count
DB Buffer Register DBR
Microprogram Counter (MCPC)
Data Register A DRA
Data Register B DRB
GR Address Register
AIR Address Register
PDL Index Counter
Special I/O Address Register

5.1 Operation of the GPP

The operation of the GPP is similar to that of a four field - three address machine. The three address fields are called P1,P2,P3. An instruction is a 4-tuple (o,P1,P2,P3), where "o" is the operator. Note that P1, P2, and P3 are effective address fields of the instruction while I1, I2 and I3 refer to specific I/O devices (the three triple line buffers). Any address field may address any image line buffer, auto-index registers, status registers,I/O registers, and general registers (GR) etc.

P1 o P2=>P3.

An execute-only program memory, PM, is used to store the ordered set of 4-tuples which constitute a program. All three address fields have an indirect bit (denoted "'") which causes a defered addressing mode to be entered. P1 (P2) have an immediate mode bit ("#") which specifies the P1 (P2) contents be used as data rather than as an address. The PM is loaded from the PDP8e.

In order to implement the various arithmetic operations, several different 16-bit and 32-bit "A"rithmetic "L"ogic "U"nits use two directly addressable registers: data register A (DRA) and data register B (DRB). For example, during an "ADD" operation, data effectively addressed by P1 of the instruction field is deposited in DRA, data effectively addressed by P2 of the instruction field is deposited in DRB. DRA and DRB are summed together to obtain the result, which is deposited in a data register effectively addressed by P3.

A bus structure is used to control and implement data transfers between sections of the GPP. The program counter bus is used to point to the present instruction in the PM under execution. The operation bus (OPRB) contains the specific code for an instruction. The data address bus (DAB) passes an address from the instruction, or an indirect address from the data field, to one of the data field's memory units such as the line buffers or general registers. The data bus (DB) passes data to and from various sections of the GPP. Actually there are three DAB/DB bus pairs: which may be used in parallel for P1, P2 and P3 (as was mentioned before) if the addresses do not conflict (i.e. access the same memory device).

The individual bus transfers are driven by a microprogrammable control processor which is itself driven by the operation bus (OPRB). The microprogram memory which defines the GPP instruction set is loaded from the PDP8e into RAMs when the GPP is first powered on each day. The microprogram instuction word is 60-bits and the microprogram memory is 4K.

Since the GPP program counter register (PC) may be loaded under program control, the machine can execute branches. The GPP also has a 16-bit X 1024 word push down list (PDL) for use in procedure calls. Arguments for these procedure calls are not passed through the PDL (used here only to return from procedure calls) but by various other standard methods such as an auxillary stack specifically used for passing arguments.

A GPP operation consists of reading P1, P2 data as specified by the addressing conventions and performing the operation "o", then storing the result in P3 according to its address conventions.

Instructions are divided into two types. Type 1 instructions are the class of instructions such as jumps and branches which may change the PC by more than +1. Type 2 instructions are the class of instructions which do not modify the PC by more than +1 such as ADD, SUB, MOVE, etc. When a type 2 instruction is being executed, a lookahead of the PM is made to fetch the next instruction as it is known what its address will be. Instructions are further classified into groups which have similar control sequences. This is discussed in appendix A.

The use of the triple DAB/DB bus pairs depends on the early recognition of P1/P2/P3 data address space conflicts. These potential conflicts are analyzed during the lookahead phase. If there is no lookahead phase (such as with type 1 branch instructions) or the binary instruction (i.e. P1oP2) which is looked ahead contains either (normal, indirect), (indirect, normal), or (indirect, indirect) (P1, P2) address then parallel (P1, P2) fetching is not performed. To do so would slow the machine down while further conflict analysis is performed after the indirect address was fetched. Additional conflict analysis is performed on P1/P3 and P2/P3 to see whether the effective P3 address may be set up on the 3rd DAB/DB bus pair. Since all bus use is controlled by the microprogram controller, it is possible to run the machine with 1, 2 or 3 DAB/DB bus pairs. This is useful both for bypassing bad buses as well as for faster implementaion of the GPP microcode for easier debugging.

5.2 General register address space

A general register (GR) memory consisting of up to 65K of 16-bit words is used in the GPP for intermediate results and working data. The high 4K words of this are used for special registers such as the GPP data registers, neighborhood addresses, control desk switches, I/O addresses, etc. All registers are directly addressable by the instruction data address fields since the group of 4K words is in the address range [170000:177777].

The registers may also contain indirect addresses used by the instruction field; this addressing facility is extended further by the eight auto-index registers which may increment or decrement automatically when used as indirect address registers depending on their address.

The source operand data address fields P1 and P2 may each hold one immediate, direct (normal), or indirect address. The sink operand data address field P3 may hold either a direct or indirect address. All registers, including the special registers mentioned above, are directly addressable as seen in figure 6. These include the current neighborhood pixels of the line buffers, the dynamic address vector registers, general registers, auto-index registers, input-output, and internal GPP status, data, and control registers.

5.2.1 GR Address Allocation

The 16-bit operand addressing specified in P1, P2, P3 is allocated as follows.

1. 000000 to 167777 denotes the 16-bit general registers which are simply high speed memory.

2. 170000 to 170077 denotes internal status registers, including the auto increment/decrement registers at

```
170000-170007 called A0D to A7D. - Autodec. On ', -1 before use
170010-170017 called A0 to A7.   - Unmodified on '
170020-170027 called A0I to A7I. - Autoinc. On ', +1 After use
```

3. 170030-170077 GPP control panel I/O (lights, switches, knobs) see Appendix B.

4. 170100 to 170110 denotes I1 16-bit data (reserve 170100-170177)

```
3 2 1          Maps to 170103   170102   170101
4 8 0 neigh.           170104   170110   170100
5 6 7.                 170105   170106   170107
```

5. 170200 to 170210 denotes I2 16-bit data (reserve

170200-170277)

```
3 2 1           Maps to 170203  170202  170201
4 8 0 neigh.            170204  170210  170200
5 6 7.                  170205  170206  170207
```

6. 170300 to 170310 denotes I3 16-bit data (reserve 170300-170377)

```
3 2 1           Maps to 170303  170302  170301
4 8 0 neigh.            170304  170310  170300
5 6 7.                  170305  170306  170307
```

7. 170400 to 170777 denotes special registers including all I/O registers, triple line buffer dynamic address vectors (IjY, IjXM, IjX, IjXP) registers for line buffers j (j=1,2,3), and other special registers.

8. 173400 to 177777 are allocated for direct addressing of triple line buffers.

```
173400:173777 I1(y-1)
174000:174377 I1(y)
174400:174777 I1(y+1)
175000:175377 I2(y-1)
175400:175777 I2(y)
176000:175377 I2(y+1)
176400:176777 I3(y-1)
177000:177377 I3(y)
177400:177777 I3(y+1)
```

The internal status registers and the I/O registers are listed in Appendix B.1.

5.2.2 Addressing sequence

Data is addressed by P1, P2, P3 operand fields either as immediate ("#") i.e. C(p), or normal addressing i.e. C(C(p)), or indirect addressing (') i.e. C(C(C(p))). Note that P3 does not have immediate mode addressing as an immediate operand for the "sink" field makes no sense.

The addressing sequence (P1oP2==>P3) is done serially. That is, P1 is addressed, then P2, then P3. If any address is immediate (#), no memory fetch is done. Rather, the PM data, C(p), is enabled onto the data bus DB. If direct addressing mode C(C(p)) is used, the PM data C(p) is enabled onto the data address bus, DAB, and then loaded into the approriate data field address register. The memory then enables its data, C(C(p)), onto the data bus, DB. If indirect mode is used, "'", then the same sequence is repeated as for direct mode, but C(C(p)) is enabled back onto the DAB instead of the DB. Then the data address field register addressed is loaded and the C(C(C(p))) from that memory is enabled onto the DB. A conflict may occur in the use of the indirect mode with the "MOVE" instruction. This is resolved by storing the source data in the data bus register, DBR, temporarily.

5.2.3 Auto-index addressing

Eight 16-bit auto-index registers (A0 through A7) are availiable for use as auto-increment or auto-decrement memory registers. Normal addressing is used to load an auto-index register. After which, when it is referenced indirectly, its contents is used as an indirect address and then its contents is decremented, left unmodified, or incremented. The auto-index contents modification is determined by how A0 through A7 are referenced. The same set of 8 registers may be addressed by three different sets of addresses. These are A0D to A7D (170000 to 170007), A0 to A7 (170010 to 170017), and A0I to A7I (170020 to 170027).

In normal mode addressing, all three sets of addresses address the same 8 registers. In indirect mode, the address is used to specify whether the register contents should be modified before (DECRement) or after (INCRement) it is used.

A data push and pop instruction are easily implemented using the auto index registers.

For example using auto index register A0:

```
"A0<==PUSH P1"  is equivalent to  "'A0I <== MOVE P1"
"P3<==POP A0"   is equivalent to  "P3 <== MOVE 'A0D".
```

In order to perform addressing as quickly as possible, high speed random access memories (RAMs) are used for both the PM, the three triple line memories I1, I2, I3, and the general registers.

5.3 GPP input/output

The GPP can transfer data from either the PDP8e or buffer memories. The GPP has access to the three line buffers. Each line buffer itself consists of three 256 pixel lines of 16-bit gray scale. These line buffers are usually used as two input picture lines and one output picture line. I1 and I2 are usually denoted as input line buffers while I3 is denoted as an output line buffer. Because 16-bit arithmetic is used, the line buffers hold 16-bit data. However, 8-bit data could be used or generated for interaction with the Quantimet. These I/O line buffers are illustrated in Figures 1 and 5.

The boundry handling problem is not in hardware, but is left to be handled in software by various commonly used schemes such as filling the boundry with a specific value or computing inside of the 256 square region.

5.3.1 Line buffer dynamic address vector registers

As discussed in Section 4, lines of picture data may be moved between buffer memories and faster triple line buffers which are directly addressable by the GPP. The 3x3 neighborhood which is directly addressable in each of the three triple line buffers is called the current neighborhood.

The current pixel in a line is determined by a set of three X dynamic address vector registers for each line buffer. That is (X-1, X, X+1) must be specified. These X dynamic address vector registers are called (I1XM, I1X, I1XP) through (I3XM, I3X, I3XP).

Program I/O consists of incrementing the line dynamic address vectors registers after the "main" program is finished executing for the current pixel. This is what is meant by the "haltpoint" subroutine.

The Y dynamic address vectors I1Y through I3Y (also called the line select registers) are modulo 3 counters which with the Y dynamic address vector logic allows dynamic "ring buffer" type addressing of the 3-line buffers. Thus to read in the Y+2 line when (Y-1,Y,Y+1) lines are already in the buffer the Y dynamic address vector advances the pointers to the lines so that Y+2 is put into the old Y-1, then the old Y is addressed as Y-1 and the old Y+1 as Y while the Y+2 is addressed as Y+1. That is, the newest line and the most recent 2 of the old 3 lines addresses are automatically adjusted.

The current neighborhood may be loaded directly with a 3x3 neighborhoold from a selected buffer memory using the GETI1, GETI2 or GETI3 instructions. Thus loading only the actual neighborhoods required saves the overhead of loading 3 lines if only a few neighborhoods (or random access of neighborhoods) is desired. The current neighborhood used is the leftmost neighborhood (i.e. IjXM=1, IjX=2, IjXP=3).

5.3.2 PDP8e/GPP-BM DMA interface

There are up to eight bidirectional DMA I/O channels between the PDP8e and the rest of the RTPP. These include the PM, BM, general register, and X8e (auxillary PDP8e) channels. The PDP8e DMA I/O commands are given in appendix C. These DMA channels are set up by the PDP8e as to word count (DMAWC) and current address (DMACA). The peripheral device starting address is specified by a 24-bit address word broadcast to all of the devices from the PDP8e (EXDMA1,EXDMA2). Note that only 1 DMA hardware channel is actually used since the other channels are multiplexed over this single channel. There is no synchronizing conflict created since single string processing is performed by the PDP8e and all bookeeping for the DMA channels is done in the PDP8e.

The GPP I/O instructions (see Appendix A) are used in two different ways. Data may be forced by the PDP8e to/from the GR or used with "programmed" GPP I/O GIN/GOUT instructions which force a program synchronization between the two machines. Therefore in the latter mode, the PDP8e must have previously set up the (WC,CA) and an GR DMA enable. It is thought that since single thread processing is to be used, that the use of interrupts is an unnecessary evil and will be avoided.

The DMA status word loaded through the PDP8e AC using the DMAGO command has various I/O options which are listed in the table in appendix C. This status word also selects which BM packing mode that is to be used in the case of BM/PDP8e DMA.

5.4 PDP8e/GPP synchronization

The GPP behaves as a PDP8e peripheral device which the PDP8e programs, starts (GPPCLR, GPPCONT), interrogates (STATG1, STATG2), and stops (GPPHLT) without assistance from residual programs or bootstraps in the GPP. At all times, the PDP8e has complete control of the GPP run flip flop (FF) and can thus stop the GPP at the end of the GPP's current instruction.

While the GPP run FF is off, the contents of the PM and/or GR may be transfered from the PDP8e through the PDP8e GPP DMA interface to the PM. The GPP PC is loaded by an PDP8e instruction GPPLAD which loads EXDMA1, EXDMA2 into the GPP PC to tell the GPP where to start its program. The run FF is turned on by doing a GPP continue (GPPCONT). To power clear the GPP, a GPP clear (GPPCLR) is issued.

5.4.1 Software synchronizing system for general I/O

To transfer data between the GPP's general (GR) I/O channel and the PDP8e memory, both processors must be in phase and synchronize with their respective software. A PDP8e DMA interface provides the logic for the transfers. The PDP8e loads the PDP8e current address and word count (DMACA, DMAWC) registers for each block transfer that will be requested by the GPP (general I/O channel) and starts the GR channel with a DMAGO command. Once this is done, the GPP starts and stops the DMA under GPP program request (GIN, GOUT). When all of the DMA transfers are complete, PDP8e DMASKP instruction detects this condition.

5.5 Running the GPP

A typical RTPP picture operation to operate on one or more BM's data would proceed as follows.

[1] Direct DMA I/O is performed from the PDP8e to the GR and PM

memories of the GPP to set up the program and data memories.

[2] The PDP8e starts the GPP by loading the GPP PC starting address using the GPPLAD instruction, then doing a GPPCLR, GPPCONT to start the GPP.

[3] If PDP8e/GR I/O is required, the PDP8e starts the general I/O (GR) channel for each interaction with the GPP GIN/GOUT instructions.

[4] The GPP PM program is exectuted on the current neighborhood picture pixel for the current line, advancing the I1, I2 and I3 current neighborhood dynamic pointers as required. (See example of I/O - Appendix D.5). This changing of the current neighborhood may be done by a GPP subroutine in which case it is called the "haltpoint" subroutine.

[5] When the end of a line is reached, output the I3 line is output to the destination BM.

[6] When the GPP is at the end of a picture or subpicture (i.e. 256 X m lines) then halt the GPP. (Then notify the PDP8e through the status register STATG1/2 otherwise continue processing by going to [4].)

[7] If necessary, the PDP8e transfers BM images by DMA to the PDP8e backing disk for later retreival. The PDP8e may also transfer general DMA data to other devices such as the PDP8e disk or the PDP10.

This scheme of operation lets the user do picture operations with easy access to GR storage as is necessary for communication with external processors.

SECTION 6

Implementation of the RTPP

This section discusses the construction of the various electronic and mechanical components of the RTPP.

6.1 The GPP control - microprogram control

The GPP control logic is implemented as a microprogrammable controller. The microprogram is loaded into the microprogram memory by the PDP8e. The various bus and register enables, function enables etc. are driven by the microprogram controller output bus. This bus is 60-bits wide so that all necessary enables may be performed in parallel. The microprogram control store is a maximum of 4K words. Note that the microprogram memory (MCPM) is distinct from the GPP program memory (PM) although both are 60-bits wide. These microprograms are loaded into the microprogram control store RAMs through the PDP8e DMA. Thus experimentation with new instruction sets is easily performed as is debugging of the initial instruction set. A microprogram assembler MICROP will assemble microprograms on the RTPP PDP8e to facilitate rapid instruction set debugging. The grammar for MICROP is given here:

6.1.1 Microprogram instruction BNF grammar

A BNF grammar specification is given for the GPP microprogram control programs. As can be seen from the MICROP grammar, assembly consists of doing the inclusive-or of all <M> symbols in a statement up to the ";". Thus each <M> symbol has an associated bit in the microprogram controller output bus.

```
<microprogram>::=<microprogram><microstatement> |
                 <microstatement> | <microprogram>$

<microstatement>::=<label><M-list><address>; |
                 /<comment>; | <M>=<number>; | <origin def>;

<origin def>::=ORIGIN <number>

<address>::= GOTO <label symbol> | <label symbol>
<label>::=<label symbol>: | <null>

<label symbol>::=new symbol | <number>

<M-list>::=<delim><M-list> | <M-list><delim><M> |
                <M-list><delim>

<delim>::= space | ,

<M>::=P1DABE | P1DBE | P2DABE | P2DBE | P3DABE | P3DBE |
                PCDBE | DRADBE | DRBDBE | ALUDBE | DRCDBE |
                DRCDAB  |  READ  |  WRITE  |  LODCTR | INCCTR |
                DECCTR | FC0000 | ... | FC1111 | BROVF | BRUDF |
                BRGT | BRLT | BREQ | BRLE | BRGE | JUMP | PUSHJ |
                POPJ | ...
```

The microprogram instruction will have a right justified 12-bit address field which is used for jumps (JUMP, PUSHJ, POPJ) and manipulating counters (LODCTR, INCCTR, DECCTR). Microprogram subroutines are performed using a 64 word 12-bit address stack.

6.2 Internal Control Logic Design

The method of Richards [Rich73] for designing complex controllers is used in various parts of the system including the Quantimet controller, BM controller, GPP microprogram controller, DMA controllers, etc. Richards employs a method of automatically defining sequential logical states such that a sequential synchronous counter implementation is easily realized. It is thus possible to easily and rationally construct finite state machine (FSM) controllers. To ease hardware debugging and maintenance of the RTPP, the "new" state is displayed in octal LED's on the cards where the FSMs reside.

6.3 Physical construction of the RTPP

The RTPP consists of several subsections as mentioned in Section 1.3. Much of the system can be bought ready made off the shelf. Other parts such as the GPP, buffer memory, control desk and QMT I/O to the PDP8e are specially built. We would like to minimize the amount of special purpose mounting hardware needed and thus are led to the implementation of the basic RTPP control and computational hardware as a set of rack mounted card files each 19 by 10.5 inches. These card files hold 16 double height (140 pin) high density wire-wrap cards made by Cambion. The wire-wrap cards plug into a semi-automatic wire-wrapable power back plane which holds two 70-pin semi-automatic wire-wrappable card sockets for each card.

The various buses can then be implemented by wire-wraping twisted pairs on these power back planes. Interconnection between card files is by plug connection with Scotchflex (3-M Corp.) cables directly to the 70 pin card sockets. Any cabling between wire-wrap cards is avoided by all connections being done through the backplanes. The wire-wrap cards and power back planes are being constructed outside of NIH already wire-wrapped with by pass capacitors mounted.

The Quantimet and control desk logic (RQC) is housed in one standard 19" wide, 6' high cabinet, the buffer memories in another and the GPP in a third cabinet. These cabinets are located between the control-desk/Quantimet-display and the PDP8e cabinets.

6.4 Buffer memory implementation

The hardware of the BM is divided into three sections: 1) memory cards; 2) dual memory card controller; and 3) BM interfaces.

The memories use Cambion specially constructed semi-automatic wire-wrap cards. The Texas Instruments TMS4030 (N channel MOS 4096x1 bit 22 pin dynamic RAM) is used as the main memory element. Each card contains a total of 64 TMS4030 memory chips and locations for 35 16-pin TTL support chips. Four memory cards make up one memory and four memories are housed in one 16 slot Cambion card enclosure. There are a total of two card enclosures used to house the eight BMs.

The dual memory controller logic is contained on one Cambion wirewrap card. Both controllers are totally independent and may operate in parallel. The dual contoller contains the refresh logic for the dynamic RAMs and the priority logic to service a total of five I/O requests from various BM interfaces.

Each BM interface requires a minimum of three card slots to communicate with the dual BM address, data and control buses. Five separate interface locations or a total of 15 card slots plus the dual memory controller card make up the third 16 slot Cambion card enclosure. A fourth Cambion card enclosure is used for additional BM interface logic. All I/O on the buffer memories may be in 16-bit word or 8-bit byte modes.

SECTION 7

References

Carm74. Carman G, Lemkin P, Lipkin L, Shapiro B, Schultz M, Kaiser P:A real time picture processor for use in biological cell identification - II hardware implementation. J. Hist. Cyto. Vol 22, 1974, 732:740.

Carm76. Carman G, Lemkin P, Schultz M:RTPP - System Documentation, Vol I: Microscope, scanner and Quantimet Controller. NCI/IP Technical Report #13. In prep.

Dec67. Digital Equipment coproration:LINC8 small computer handbook. Maynard, Mass, 1967.

Dec71. Digital Equipment Corporation:PDP11/20 processor handbook. Maynard, Mass, 1971.

Dec72a. Digital Equipment Corporation:PDP8e and PDP8m small computer handbook. Maynard, Mass. 1972.

Dec72b. Digital Equipment Corporation:DEC system 10 assembly language handbook. Maynard, Mass. 1972.

Fish71. Fisher C:The new QUANTIMET 720. The Microscope Vol 19 No 1, 1971, 1:20.

Gros76a. Grosfeld G, Lemkin P, Shapiro B:GPPASM - assembler for the GPP.NCI/IP Technical Report #16. In prep.

Gros76b. Grosfeld G, Lemkin P:MICROP - Microprogram Assembler for the GPP.NCI/IP Technical Report #19. In prep.

JohnE70. Johnston E:The PAX II picture processing system. In [LipB70].

Kir69. Kirsch R:Computer determination of the constituent structure of biological images part I. NBS report 10 173, DEC,1969.

Lem72. Lemkin P:A simplified biological cell world model for question answering using functional description. Univ. of Maryland, Scholarly Paper #75, May, 1972.

Lem74. Lemkin P, Carman G, Lipkin L, Shapiro B, Schultz M, Kaiser P:A real time picture processor for use in biological cell identification - I systems design. J. Hist. Cyto, 22, 1974, 725:731.

Lem75. Lemkin P:A Literature Survey of the Technological Basis for Automated Cytology. Univ. Md. TR-386, June, 1975.

Lem76a. Lemkin P:Functional specifications for the RTPP monitor - debugger DDTG. NCI/IP Technical Report #2, Feb 1976.

Lem76b. Lemkin P, Shapiro B:PRDL - Procedural Description Language. NCI/IP Technical Report #13. In prep.

Lem76c. Lemkin P, Shapiro B, Lipkin L:The CELMOD Biological Image Modelling System. NCI/IP Technical Report #14. In prep.

Lem76d. Lemkin P, Gordon R, ShapiroB:PROC10 - 'PROCES' image processing system for the PDP10. NCI/IP Technical Report #20. In prep.

LipB70. Lipkin B, Rosenfeld A (Eds):Picture processing and Psychopictorics. Academic Press, 1970.

LipL74. Lipkin L, Lemkin P, Carman G:Automated Grain Counting in Human Determined Context. J. Hist. Cyto. Vol 22, 1974, 755.

Rich73. Richards C:An easy way to design complex program controllers. Electronics, Feb 1, 1973, 107:113.

ShapB74. Shapiro B, Lemkin P, Lipkin L:The application of artificial intelligence techniques to biological cell identification. J. Hist. Cyto. Vol 22, 1974, 741:750.

Thor70. Thorton J E:The control data 6600 - design of a computer. Scott, Foresman and Co, 1970.

VanL73. VanLehn K:Sail User Manual. Stanford Artifical Intelligence Laboratory memo AIM-204, July 1973.

Wil75. Wilcox C:MAINSAIL - MAchine INdependent SAIL. DECUS meeting, Languages in Review Session, 1975.

SECTION A

GPP instruction set

A.1 GPP operators

The instruction set of the GPP allows for not only the usual set of operators needed in a triple address type machine, but also specific operators oriented toward neighborhood processing and I/O image. These include the following five classes of instructions for which examples are given in table 1.

Table 1. Examples of some RTPP instructions

```
   1) Register-to-register transfers,
        P3<==MOVE P1       ; P1 to P3
        P3<==INC P1        ; increment P1 by 1 into P3
        P3<==ADDST P1      ; add P1 and DRA register, store in P3
        P3<==GESTA P1      ; if P1 > DRA store P1 in P3
   2) P1 operated on by P2 to be deposited in P3,
        P3<==P1 ADD P2     ; sum of P1 and P2 stored in P3
        P3<==P1 MUL P2     ; product of P1 and P2 stored in P3
        P3<==P1 AND P2     ; Logical AND of P1 and P2 stored in P3
        P3<==P1 MOVBIT P2 ; bitset P3 from P1 under mask P2
   3) Conditional branch instructions
        P3<==P1 BEQ P2     ; if P1=P2 then goto P3 else do next instr.
        P3<==P1 BGT P2     ; if P1>P2 then goto P3 else do next instr.
   4) Control instructions
        P3<==PUSHJ         ; procedure entry, stack return address
        POPJ               ; procedure exit to stacked return address
        P3<==JUMP          ; unconditional GOTO
   5) I/O instructions.
        XRST (<X-1,X,X+1>,<I1,,>)   ; reset I1 X dynam. vectors
        XCLK (<right>,<X-1,X,X+1>,<,I2,I3>) ; incr. X dynam. vec. of I2, I3
        YCLK (<I1,I2,>)    ; advance Y lines in I1, I2
        LINE (lowbyte,read,hor,BM3,I1,Y+1),lineaddress ; read a line
        LINE (16bit,write,vert,BM2,I3,Y),lineaddress ; write a line
        GETI2 (lowbyte,BM3),YXADDR   ; Fetch a neighborhood into
                                               I2 line buffer
        P3<==MOVE 'PBM2    ; read BM2 datum into P3
        'PBM2<==MOVE P1    ; write P1 into  BM2
        P3<==MOVE GIN      ; read word from PDP8e channel
        GOUT<==MOVE P1     ; write word to PDP8e channel
        P3<==MOVE KRB      ; read GPP teletype input channel
```

Among the I/O instructions are whole line and single pixel transfers from/to the buffer memories. The latter type of instruction allows random access of the buffer memories. The line transfer allows entire horizontal or vertical 256 (16-bit) pixel lines to be transfered. The PDP8e can access the

GPP through direct memory access to the GPP program memory, line buffers (I1, I2, and I3), and data field (through data field address).

A GPP instruction cycle consists of fetching an instruction from the PM, incrementing the PC instruction program counter, and then executing that instruction. The contents of the PC is the index of the next instruction to be executed in the PM. The instruction is executed by fetching the two input operands, performing the operation, and then storing the result in the output operand if so directed.

A.2 Effective address notation

The following notation is used in the description of the GPP instructions. If p is an address field of the PM instruction register, then C(p) is the contents (immediate address "#") of this field. Any immediate data have the 4 most significant bits equal to zero. Therefore, there are no negative immediate data. C(C(p)) is the (normal addressing) contents of the location pointed to by the address field. C(C(C(p))) is the (indirect address "'") contents of the address which is pointed to by the contents which is pointed to by the contents of the address field p. Because all 3 types of address are generally legal, the effective address (or effective contents) is used in the following instruction descriptions and is denoted as P1, P2 or P3. The C(...) Notation is used where appropriate.

The following prefixes define the addressing mode for various instructions.

No prefix does normal addressing.
Does immediate addressing on P1, and P2 (but not P3).
' Does indirect addressing on P1, P2, and P3.

A.3 Instruction lookahead - two types

There are two types of basic instructions. Type 1 and type 2. Type 1 is an instruction such as a jump or branch where the PC changes by more than +1 at the end of the instruction. Type 2 instructions are used when the PC is advanced by only +1 (automatically) at the end of the instruction. Type 2 is exemplified by ADD, SUB, etc. During the execution of type 2 instructions, the PM is accessed to lookahead for the next instruction, (which is the motivation for the distinction). Input/output instructions may be of either type.

The two types of instructions are composed of 13 groups of operators, each with specific properties. These properties are dictated by the control logic. In the following lists of instructions, the group each instruction belongs to is specified by a number within "< >".

The 12-bit PM "operator" field is assigned as follows: where bit 0 = msb, bit 11 = lsb. In the group table below, X denotes instruction address mode while Y denotes operator subgroup. Bits [0:4] are used for the address mode bits (note that P3 does not have immediate addressing as it does not make sense).

Bit 0	= 0	--> P3 normal addressing,
	= 1	--> P3 indirect addressing.
Bits 1,2	= 0 0	--> P1 normal addressing,
	= 0 1	--> P1 immediate addressing,
	= 1 0	--> P1 indirect addressing,
	= 1 1	--> illegal - trapped,
Bits 3,4	= 0 0	--> P2 normal addressing,
	= 0 1	--> P2 immediate addressing,
	= 1 0	--> P2 indirect addrssing,
	= 1 1	--> illegal - trapped.

Table 2. GPP Instruction Group Selection

Instruction group	Opr code												Max instr.	Basic opcode
Bits	0	1	2	3	4	5	6	7	8	9	10	11		
1	X	X	X	X	X	0	0	0	0	0	0	Y	2	0000
2	X	X	X	X	X	0	0	0	0	0	1	Y	2	0002
3	X	X	X	X	X	1	0	1	Y	Y	Y	Y	16	0120
4	X	X	X	X	X	0	0	0	1	Y	Y	Y	8	0010
5	X	X	X	X	X	0	0	1	0	Y	Y	Y	8	0020
6	X	X	X	X	X	0	0	1	1	Y	Y	Y	8	0030
7	X	X	X	X	X	0	1	0	0	0	Y	Y	4	0040
8	X	X	X	X	X	0	1	0	0	1	Y	Y	4	0044
9	X	X	X	X	X	0	1	0	1	Y	Y	Y	8	0050
10	X	X	X	X	X	1	0	0	Y	Y	Y	Y	16	0100
11	X	X	X	X	X	1	1	0	0	0	Y	Y	4	0140
12	X	X	X	X	X	1	1	0	0	1	Y	Y	4	0144
13	X	X	X	X	X	1	1	0	1	0	Y	Y	4	0150

The following sequence descriptions of the instruction groups are given to aid in understanding the structure of the GPP control system. Using Figure 6 while reading these descriptions is useful.

Additional instruction groups may be added later in the opr code range of 154 to 177. These instructions might include floating point and trigonometric functions.

A.4 Operator groups

Group 1 (0000)

P3<==P1.

Description:

Data in P1 or addressed by P1 is moved to location addressed by P3. P1 and P3 may address all of data address space.

Instruction:

```
Opcode
------
0   MOVE - P3<==P1.
        MOVE uses the DBR register when P3 is in indirect mode.
        i.e. if P1 is "'" then DBR<=='P1, P3<==DBR.
                               else P3<==P1;
```

Group 2 (0002)

```
DRA<==P1, DRB<==P2, DBR<==P3.
If P1.AND.[P2/256].NE.0
                    then P3<==DRB*DBR.
        *=One of GPP arithmetic unit (ALU) operations.
```

Description:

Data from or addressed" by P1 is deposited in DRA. Data from or addressed" by P2 is deposited in DRB. Data from or addressed by P3 is deposited in the DBR. DRA, DRB and DBR are the inputs to the selected arithmetic unit of the GPP called for as a function of the operation. The output of the ALU is deposited in the data register addressed by P3. The instructions MOVBIT and MOVBS are bit-set operations for copying 8-bit PAX type image planes. Note that C(P2) is used in an unusual way. The high order 8-bits of the 16-bit P2 mask are used to select (via a logical 'AND') the low order 8-bits of P1 to be tested. If any bit is on, MOVBIT loads the low 8-bits of P2 into P3 (zeroing the high 8-bits of P3). Similarly, if any bit is on, MOVBS (bitset) will logically OR the low order P2 8-bits with the low order P3 8-bits (destroying the high order P3 8-bits). MOVBIT will also store P3<==0 and MOVBS will store P3<==P3 if the test condition is NOT met.

Instructions:

```
Opcode
------
0   MOVBIT - if P1.AND.[P2/256].NE.0
                then P3<==255.AND.P2
                else P3<==0.
1   MOVBS - if P1.AND.[P2/256].NE.0
                then P3<==(255.AND.P2).OR.P3
                else P3<==P3.
```

Group 3 (0120)

DRA<==P1, DRB<==P2, P3<==DRA*DRB.
*=One of GPP arithmetic unit (ALU) operations.

Description:
Data from or addressed" by P1 is deposited in DRA. Data from or addressed" by P2 is deposited in DRB. DRA and DRB are the inputs to the selected arithmetic unit of the GPP called for as a function of the operation. The output of the ALU is deposited in the data register addressed by P3.

Instructions:
Opcode

For arithmetic, a 16 bit result is derived from the computation for the effective P3 address by truncation to 16-bits. The high order part (or remainder) is stored in the EXAR. ADD and SUB are 16-bit adders. MUL is carried to 32 bits (16x16) but only the bottom 16 bits of result are used. DIV is 32-bits (EXAR!c(P)) divided by 16-bits with the remainder going into the EXAR. The high order 16-bits are stored in the extended arithmetic register "EXAR".

```
0    ADD - P3<==P1+P2, (EXAR<==overflow).
1    SUB - P3<==P1-P2, (EXAR<==underflow).
2    MUL - (EXAR!P3)<==P1*P2, (EXAR<==hi order, 32 bit product).
3    DIV - P3<==(EXAR!P1)/P2, (32-bit dividend, EXAR<==remainder).
```

In the following instructions, bit selection register, BSR[0:15] is AND'ed with the logical result before it is output into P3. The BSR is a special status register in upper address space and may be loaded directly (with a move etc.).

```
4    AND - P3<== P1.AND.P2.
5    NAND - P3<== P1.NAND.P2.
6    XOR - P3<== P1.XOR.P2.
7    IMPLIES - P3<== P1.IMPLIES.P2.
10   OR - P3<== P1.OR.P2.
11   NOR - P3<== P1.NOR.P2.
12   EQV - P3<== P1.EQV.P2.

13   SHIFTR - P3<== rightshift P1 by P2 MOD 16-bits; 0 fill.
14   SHIFTL - P3<== leftshift P1 by P2 MOD 16-bits; 0 fill.
15   ROTR - P3<== P1 rotate 16-bits right P2 bits .
16   ROTL - P3<== P1 rotate 16-bits left P2 bits .
```

Group 4 (0010)

DRA<==P1, P3<==DRA*DRB.
*=One of GPP ALU operations.

Description:
Data from or addressed by P1 is deposited in DRA. DRA and DRB are the inputs to the selected logic unit called for by the instruction. The output of the logic unit is deposited in the data register addressed by P3. Note that the ALU is not symmetric and that only DRA is loaded.

Instructions:

```
Opcode
------
0    MOVEN       - 2's complement P3<==P1.
1    MOVEC       - 1's complement of P3<==P1.
2    INC         - P3<==P1+1.
3    DEC         - P3<==P1-1.
4    MAKYXADDR   - P3<== (P1 AND #377) OR ((P2 AND #377) SHIFTL 8)
```

Group 5 (0020)

```
        DRB<==P1, P3<==DRA*DRB.
                *=One of GPP ALU  operations.
```

Description:

Data from or addressed by P1 is deposited in DRB. DRA and DRB are the inputs to the selected logic unit called for by the instruction. The output of the logic unit is deposited in the data register addressed by P3. Note that the ALU is not symmetric and that only DRB is loaded. These instructions are the arithmetic list processing instructions and take the form of "operator 'ST'ore". It is useful when a previous result can be left in the DRA.

Instructions:

```
Opcode
------
0   ADDST - DRB<==P1, P3<==DRA+DRB, (EXAR<==overflow).
1   SUBST - DRB<==P1, P3<==DRA-DRB, (EXAR<==underflow).
2   MULST - DRB<==P1, P3<==DRA*DRB, (EXAR<==hiorder).
```

Group 6 (0030)

```
        DRB<==P1, condition<==DRA*DRB
                If condition is true then P3<==DRB.
                *=One of GPP ALU  compare operations.
```

Description:

Data from or addressed by P1 is deposited in DRB. DRA and DRB are the inputs to the selected logic unit called for by the instruction. If the condition is true, the DRB is deposited in the data register addressed by P3.

Instructions:

```
Opcode
------
0   GTST - (find max) DRB<==P1, if DRB .GT. DRA
                                   then P3<==DRB
                                   else continue.
1   LTST - (find min) DRB<==P1, if DRB .LT. DRA
                                   then P3<==DRB
                                   else continue.
2   GEST - (max partial) DRB<==P1, if DRB .GE. DRA
                                   then P3<==DRB
                                   else continue.
3   LEST - (min partial) DRB<==P1, if DRB .LE. DRA
                                   then P3<==DRB
                                   else continue.
```

AC, i.e. address of first transfer.

5. DMACLR clear the DMA channels. Note that a PDP8e CLF (6007) or PDP8e front panel lear also causes a DMACLR.

6. Each DMA peripheral device (BM, GR, etc.) requires an additional address at which to perform the DMA in the peripheral device address space.

   EXDMA1 - loads the high 12-bits of I/O device address.
   EXDMA2 - loads low 12-bits of I/O device address.

C.3 PDP8e IOT Instructions for BM controller

1. Eight pairs of BM (XB(i),YB(i)) coordinate registers (0:11 bit BCD each).

```
BMX0 - load the BM0 X coord. Reg<==8e acc.
BMY0 - load the BM0 Y coord. Reg<==8e acc.
BMX1 - load the BM1 X coord. Reg<==8e acc.
BMY1 - load the BM1 Y coord. Reg<==8e acc.
BMX2 - load the BM2 X coord. Reg<==8e acc.
BMY2 - load the BM2 Y coord. Reg<==8e acc.
BMX3 - load the BM3 X coord. Reg<==8e acc.
BMY3 - load the BM3 Y coord. Reg<==8e acc.
BMX4 - load the BM4 X coord. Reg<==8e acc.
BMY4 - load the BM4 Y coord. Reg<==8e acc.
BMX5 - load the BM5 X coord. Reg<==8e acc.
BMY5 - load the BM5 Y coord. Reg<==8e acc.
BMX6 - load the BM6 X coord. Reg<==8e acc.
BMY6 - load the BM6 Y coord. Reg<==8e acc.
BMX7 - load the BM7 X coord. Reg<==8e acc.
BMY7 - load the BM7 Y coord. Reg<==8e acc.
```

2. Two commands (GETA, GETB) are required to enable the acquiring of gray scale or detected (binary mask) video data into buffer memories. Issuing a STQMT instruction after one of these commands will cause video scan data to be acquired on the next scan. The BMs are divided into two groups (A and B) each with their own sub-controller. Thus one group can be posting or acquiring data while the other is being used by the GPP or PDP8e to compute in. Group A includes BMs (0,1,2,3) and group B includes (4,5,6,7). A status word is loaded by the PDP8e from the AC to specify these enables.

Bit	Function	BMi
0	1 ==>sel. high byte pix/low byte bin mask	0 (A), 4 (B)
	0 ==>sel. low byte pix/high byte bin mask	0 (A), 4 (B)
1		1 (A), 5 (B)
2		2 (A), 6 (B)
3		3 (A), 7 (B)
4	1 ==>enab. bin mask acquisiton in BM	0 (A), 4 (B)
	0 ==>disable bin mask acquisiton in BM	0 (A), 4 (B)
5		1 (A), 5 (B)
6		2 (A), 6 (B)
7		3 (A), 7 (B)
8	1 ==> select BM	0 (A), 4 (B)
	0 ==> deselect BM	0 (A), 4 (B)
9		1 (A), 5 (B)
10		2 (A), 6 (B)
11		3 (A), 7 (B)

Note that one may acquire the mask byte of a BM while storing the gray scale data in the other byte. The status of the two gets may be tested by reading in the GET-DONE bits for the two groups

RGETA - read done bits for 0,1,2,3 into [0:3].
RGETB - read done bits for 4,5,6,7 into [0:3].

3. Two commands POSTA and POSTB are used to specify which BMs to post on the Quantimet display (at the corresponding coordinate register specified windows). Either the high or low BM byte may be displayed according to bits [0:3]. In addition the BM binary mask may be generated if the correspond bits are set [4:7]. The BMs to be displayed are selected from bits [8:11]. So the table given for GETA/GETB also applies to POSTA/POSTB.

C.4 PDP8e IOT Instructions for GPP controller

1. GPP STATUS[0:15] is read/write by the PDP8e.

   STATG2[8:11] - read GPP status register bits 0:3.
   STATG1[0:11] - read GPP status register bits 4:15

2. GPPCLR - does a GPP clear to clear GPP registers including all those done by a GPP IOCLR and also clears the GPP PDLCNT, PC and DAB trap enables.

3. GPPCONT - does a GPP continue to turn on the GPP run flip flop.

4. GPPHLT - does a GPP halt to turn off the GPP run flip flop.

4. Load the PC trap register. The trap will halt the GPP if the trap address appears in the GPP PC with the trap found bit set in the GPP status register. The panel switch must be up to enable.

   PCTRP - load EXDMA1:EXDMA2 into the GPP PC trap
           register and enable the trap.
   PCTDS - disable PC trap.

5. Load the GPP PC register.

   GPPLAD - load EXDMA1:EXDMA2 into the GPP PC register.
   RGPPCH[8:11] - read GPP PC high
   RGPPCL[0:11] - read GPP PC low.

6. Load the DAB trap register. The trap will halt the GPP if the trap address appears in the GPP DAB with the trap found bit set in the GPP status register. The panel switch must be up to enable.

   DABTRP - load EXDMA1:EXDMA2 into the GPP DAB trap
            register and enable the trap.
   DABTDS - disable DAB trap.

C.5 PDP8e IOT Instructions for X8E controller

In controlling the X8E auxillary PDP8e processor, two output words are used by the X8e Controller.

1. X8ECTL - load the X8e control word from the PDP8e AC. When a bit is on in any one of these control functions the function will be executed. There is no need to clear the bit.

Bits	function
0	Halt X8e
1	Clear X8e
2	Cont X8e
3	Addr load X8e
4	Extd addr load X8e
5	Dep X8e
9-11	Extended current address for X8e.

2. X8ECA - X8e current address and switch register. Used for X8e current address on DMA and the X8e switch register when the X8ECTL instruction is used.

C.6 Allocation of PDP8e IOTs for the RTPP

The following lists are the PDP8e device codes allocated for the QMT, control desk, GPP, BM, parts of the RTPP.

C.6.1 Alphabetic listing of PDP8e IOTs

DEVICE	CODE	CARD	FUNCTION
ADVSR	6314	RQCA10	ADVANCE SHIFT REG. ONE
BMIN	6542	BMD25	BM maintenance (spare input)
BMOUT	6526	BMD25	BM maintenance (spare output)
BMX0	6500	BMD29	load the BM0 X coord reg<==8e Acc.
BMX1	6501	BMD29	load the BM1 X coord reg<==8e Acc.
BMX2	6502	BMD29	load the BM2 X coord reg<==8e Acc.
BMX3	6503	BMD29	load the BM3 X coord reg<==8e Acc.
BMX4	6510	BMD27	load the BM4 X coord reg<==8e Acc.
BMX5	6511	BMD27	load the BM5 X coord reg<==8e Acc.
BMX6	6512	BMD27	load the BM6 X coord reg<==8e Acc.
BMX7	6513	BMD27	load the BM7 X coord reg<==8e Acc.
BMY0	6504	BMD29	load the BM0 Y coord reg<==8e Acc.
BMY1	6505	BMD29	load the BM1 Y coord reg<==8e Acc.
BMY2	6506	BMD29	load the BM2 Y coord reg<==8e Acc.
BMY3	6507	BMD29	load the BM3 Y coord reg<==8e Acc.
BMY4	6514	BMD27	load the BM4 Y coord reg<==8e Acc.
BMY5	6515	BMD27	load the BM5 Y coord reg<==8e Acc.
BMY6	6516	BMD27	load the BM6 Y coord reg<==8e Acc.
BMY7	6517	BMD27	load the BM7 Y coord reg<==8e Acc.
CLKACK	6302	RQCA4	CLEAR 200 HZ CLOCK FLAG
CLKSKP	6303	RQCA4	SKIP ON 200 HZ CLOCK FLAG SET
CSRGI	6315	RQCA10	CLEAR SHIFT REG. INDEX REG.
DABTDS	-	-	disable GPP DAB trap.
DABTRP	-	-	load EXDMA1:EXDMA2 into the GPP DAB trap
DET1	6422	RQCA8	1-D DETECTOR THRESHOLD 1 <== C(AC) (*NOT USED)
DET2	6423	RQCA8	1-D DETECTOR THRESHOLD 2 <== C(AC) (*NOT USED)
DETDIG	6424	RQCA8	DENSITOMETER 6-bit T1 (LEFT BYTE) AND T2 (RIGHT BYTE) THRESH <== C(AC)
DETB	6420	RQCA8	DETECTOR THRESHOLD B <== C(AC)
DETC	6421	RQCA8	DETECTOR THRESHOLD C <== C(AC)
DISP1	6435	RQCA22	CONTROL DESK DISPLAY 1 <== C(AC)
DISP2	6436	RQCA22	CONTROL DESK DISPLAY 2 <== C(AC)
DMACA	6073	RDMA	load RTPP DMA current address register
DMACLR	6074	RDMA,BMB31	clear the RTPP DMA interface
DMAGO	6070	RDMA,BMB31	start the RTPP DMA PDP8e<==>DMA
DMASKP	6071	RDMA	skip on RTPP DMA done
DMAWC	6072	RDMA	load the DMA word count register
EXADR	6450	RQCB/EXIN/EXOUT	C(AC)==>C(XADR)
EXDMA1	6524	EBUS2 (also BMD31,BMD25,BMB31)	load high RTPP DMA address bus
EXDMA2	6525	EBUS2 (also BMD31,BMD25,BMB31)	load low RTPP DMA address bus

EXIN	6333	RQCB/EXIN	C(AC)<==C(C(XADR))
EXOUT	6451	RQCB/EXOUT	C(AC)==>C(C(XADR))
FBW1	6341	RQCB2	C(AC) <== CONTROL DESK DATA WORD 1
FBW2	6342	RQCB2	C(AC) <== CONTROL DESK DATA WORD 2
FBW3	6343	RQCB2	C(AC) <== CONTROL DESK DATA WORD 3
FBW4	6344	RQCB2	C(AC) <== CONTROL DESK DATA WORD 4
FBW5	6345	RQCB2	C(AC) <== CONTROL DESK DATA WORD 5
FBW6	6346	RQCB2	C(AC) <== CONTROL DESK DATA WORD 6
FBW7	6347	RQCB2	C(AC) <== CONTROL DESK DATA WORD 7
FBW10	6350	RQCB4	C(AC) <== CONTROL DESK DATA WORD 10
FBW11	6351	RQCB4	C(AC) <== CONTROL DESK DATA WORD 11
FBW12	6352	RQCB4	C(AC) <== CONTROL DESK DATA WORD 12
GETA	6522	BMD27	enable acquiring group A BM pix or binary masks
GETB	6523	-	enable acquiring group B BM pix or binary masks
GETMSK	6304	RQCA14	LOAD MASK REG FROM QMT ON NEXT STQMT
GPPCLR	-	-	clear the GPP registers
GPPCONT	-	-	continue the GPP
GPPHLT	-	-	HLT the GPP
GPPLAD	-	-	load GPP PC from EXDMA1,2
HPL	6360	RQCA6	FRAME HOR. POSITION COUNTER <== C(AC)
HPR	6320	RQCA6	C(AC) <== FRAME HOR. POSITION COUNTER
HSL	6361	RQCA6	FRAME HOR. SIZE COUNTER <== C(AC)
HSR	6321	RQCA6	C(AC) <== FRAME HOR. SIZE COUNTER
IZSKP	6317	RQCA10	SKIP IF INDEX 10-bits = ZERO.
LDXP	6443	RQCA14	X COORDINATE CURSOR REG. <== C(AC)
LDYP	6444	RQCA14	Y COORDINATE CURSOR REG. <== C(AC)
LFBW2	6437	RQCA22	CONTROL DESK SWITCH LIGHTS <== C(AC)
LGALX	6456	RQCB8	MIRROR SCANNER X COORDINATE <== C(AC)
LGALY	6457	RQCB8	MIRROR SCANNER Y COORDINATE <== C(AC)
LMASKE	6441	RQCA14	MASK ENTRANCE REG. <== C(AC)
LMASKX	6442	RQCA14	MASK EXIT REG. <== C(AC)
LPENX	6445	RQCA16	LIGHT PEN X COORDINATE <== C(AC)
LPENY	6446	RQCA16	LIGHT PEN Y COORDINATE <== C(AC)
LQDT1	6375	RQCA4	QMT RIGHT DISPLAY LSW <== C(AC)
LQDT2	6376	RQCA4	QMT RIGHT DISPLAY MIDDLE WORD <== C(AC)
LQDT3	6377	RQCA4	QMT RIGHT DISPLAY MSW <== C(AC)
LSRGB	6367	RQCA10	SHIFT REG. LOADING REG. <== C(AC)
MSKADR	6440	RQCA14	MASK ADDRESS REG. <== C(AC)
MSTAG	6366	RQCB10	STAGE DIRECTION REG. <== C(AC)
PCTDS	-	-	disable the PC address trap
PCTRP	-	-	load EXDMA1:EXDMA2 into the GPP PC trap
PENST	6447	RQCA16	LIGHT PEN STATUS REG. <== C(AC)
POSTA	6520	BMD29	post selected group A BMs
POSTB	6521	BMD27	post selected group B BMs
QDAT1	6324	RQCA4	C(AC) <== QMT BCD DATA LSW
QDAT2	6325	RQCA4	C(AC) <== QMT BCD DATA MIDDLE WORD
QDAT3	6326	RQCA4	C(AC) <== QMT BCD DATA MSW
QMSKP	6301	RQCA4	SKIP WHEN QMT DATA SCAN DONE
QPROG1	6370	RQCA4	QMT PROGRAM WORD 1 <== C(AC)
QPROG2	6371	RQCA4	QMT PROGRAM WORD 2 <== C(AC)
QPROG3	6372	RQCA4	QMT PROGRAM WORD 3 <== C(AC)
QPROG4	6373	RQCA4	QMT PROGRAM WORD 4 <== C(AC)
QPROG5	6431	RQCA22	QMT PROGRAM WORD 5 <== C(AC)
QPROG6	6432	RQCA22	QMT PROGRAM WORD 6 <== C(AC)
QPROG7	6433	RQCA22	QMT PROGRAM WORD 7 <== C(AC)

QPROG8	6434	RQCA22	QMT PROGRAM WORD 8 <== C(AC)
QSTAT	6374	RQCA4	RQC PROGRAM WORD <== C(AC)
RBMSP1	6543	-	spare input (not implemented)
RBMSP2	6544	-	spare input (not implemented)
RBMSP3	6545	-	spare input (not implemented)
RBMSP4	6546	-	spare input (not implemented)
RBMSP5	6547	-	spare input (not implemented)
RFC1H	6334	RQCA12	C(AC) <== FUNCTION COMPUTER 1 MSW
RFC1L	6335	RQCA12	C(AC) <== FUNCTION COMPUTER 1 LSW
RFC2H	6336	RQCA12	C(AC) <== FUNCTION COMPUTER 2 MSW
RFC2L	6337	RQCA12	C(AC) <== FUNCTION COMPUTER 2 LSW
RGETA	6540	BMD29	Read the status of done bits for BM GETA
RGETB	6541	BMD27	Read the status of done bits for BM GETB
RGPPCH	-	-	read GPP PC high
RGPPCL	-	-	read GPP PC low
RKYPDH	6340	RQCB6	C(AC) <== CONTROL DESK KEY PAD MSW
RKYPDL	6353	RQCB6	C(AC) <== CONTROL DESK KEY PAD LSW
RMASKE	6354	RQCA14	C(AC) <== MASK ENTRANCE DATA AS F(MSKADR)
RMASKX	6355	RQCA14	C(AC) <== MASK EXIT DATA AS F(MSKADR)
RPENX	6356	RQCA16	C(AC) <== LIGHT PEN X COORDINATE
RPENY	6357	RQCA16	C(AC) <== LIGHT PEN Y COORDINATE
RQSTAT	6327	RQCA4	C(AC) <== QSTAT
RSRGI	6332	RQCA10	C(AC) <== SHIFT REG. INDEX REG.
RSRGX	6330	RQCA10	C(AC) <== X CORDINATE SHIFT REG. DATA
RSRGY	6331	RQCA10	C(AC) <== Y CORDINATE SHIFT REG. DATA
SIZEA	6425	RQCA8	AMENDER SIZE REG. <== C(AC) BCD
SIZEC	6426	RQCA8	CLASS/COLLEC SIZE REG. <== C(AC) BCD
SIZEM	6427	RQCA8	MS3 COMPUTER SIZE REG. <== C(AC) BCD
SIZES	6430	RQCA22	STD COMP. SIZE REG. <== C(AC) BCD
SKPKPD	6313	RQCB6	SKIP ON CONTROL DESK KEYPAD, CLEAR FLAG ON SKIP
SMACP	6310	RQCA10	SIMULATE QMT ACP AS F(QSTAT BIT 5)
SMCLK	6311	RQCA10	SIMULATE QMT CLOCK AS F(QSTAT BIT 5)
SMHLD	6307	RQCA4	SIMULATE QMT HOLD AS F(QSTAT BIT 5)
SMOTR	6365	RQCB10	SPARE MOTOR REGISTER<==C(AC)
SMSYN	6312	RQCA10	SIMULATE QMT SYNC AS F(QSTAT BIT 5)
SMVTG	6306	RQCA4	SIMULATE QMT VERT. TRIG AS F(QSTAT BIT 5)
STEP	6305	RQCB10	MOVE STAGE AS F(MSTAG)
STQMT	6300	RQCA4	START QMT DATA SCAN
VPL	6362	RQCA6	FRAME VERT. POSITION COUNTER <== C(AC)
VPR	6322	RQCA6	C(AC) <== FRAME VERT. POSITION COUNTER
VSL	6363	RQCA6	FRAME VERT. SIZE COUNTER <== C(AC)
VSR	6323	RQCA6	C(AC) <== FRAME VERT. SIZE COUNTER
X8ECA	-	-	LOAD X8E CURRENT ADDRESS
X8ECTL	-	-	LOAD X8E CONTROL REGISTER
ZSRGI	6316	RQCA10	SEND SHIFT REG. DATA TO FRONT OF SHIFT REG.

C.7 PDP8e Device code allocation by decade

The actual device codes for particular DEC devices may be found in the various versions of the "Small Computer Handbook".

```
00       - PDP8e CPU
01       - paper tape reader
02       - paper tape punch
03       - Decwriter keyboard (or PDP11/20 emulator)
04       - Decwriter printer (or PDP11/20 emulator)
07       - RTPP DMA I/O channel
------------------------------------
10       - Dicomed
11       - DC02 serial interface keyboard
12       - DC02 serial interface printer
14       - Graf-Pen
16       - HSP: input channel (PDP11/20 emulator)
17       - HSP: output channel (PDP11/20 emulator)
------------------------------------
20:27    - PDP8e extended memory control
------------------------------------
30:37    - RQC - EBUS #1
------------------------------------
42:45    - RQC - EBUS #1
------------------------------------
50:52    - EBUS #2
53       - PDP8e A/D multiplexor AD8e/AM8e
54:57    - EBUS #2
------------------------------------
60:62    - EBUS #2
63       - (Decwriter KBD - not used since PTR used as LPT)
65       - PDP8e Floating point processor FPP
66       - PDP8e line printer LP08 (Decwriter PTR) or
           PDP11/20 emulator
67       - EBUS #2
------------------------------------
70:72    - PDP8e TC58 magtape control
74       - PDP8e RK8e disk control
76:77    - PDP8e TC08 Dectape control
```

C.7.1 Numerical listing of PDP8e IOTs

RDMA "OUTPUT" DEVICE CODES - C(PDP8e ACC)<==C(channel)

DEVICE CODE		CARD	FUNCTION
6073	DMACA	RDMA	load RTPP DMA current address register
6074	DMACLR	RDMA,BMB31	clear the RTPP DMA interface
6070	DMAGO	RDMA,BMB31	start the RTPP DMA PDP8e<==>DMA
6071	DMASKP	RDMA	skip on RTPP DMA done
6072	DMAWC	RDMA	load the DMA word count register

PDP8e auxillary devices

DEVICE CODE		CARD	FUNCTION (on DEC 1709 card)
6101	DICSKP	PFL1	skip on Dicomed ready for next command
6102	DICLR	PFL1	set Dicomed ready
6106	DICO	PFL1	send Dicomed command <==AC[0,3:11]

RQC "PULSE" DEVICE CODES - note: does not affect the PDP8e ACC

DEVICE CODE		CARD	FUNCTION
6300	STQMT	RQCA4	START QMT DATA SCAN
6301	QMSKP	RQCA4	SKIP WHEN QMT DATA SCAN DONE
6302	CLKACK	RQCA4	CLEAR 200 HZ CLOCK FLAG
6303	CLKSKP	RQCA4	SKIP ON 200 HZ CLOCK FLAG SET
6304	GETMSK	RQCA14	LOAD MASK REG FROM QMT ON NEXT STQMT
6305	STEP	RQCB10	MOVE STAGE AS F(MSTAG)
6306	SMVTG	RQCA4	SIMULATE QMT VERT. TRIG AS F(QSTAT BIT 5)
6307	SMHLD	RQCA4	SIMULATE QMT HOLD AS F(QSTAT BIT 5)
6310	SMACP	RQCA10	SIMULATE QMT ACP AS F(QSTAT BIT 5)
6311	SMCLK	RQCA10	SIMULATE QMT CLOCK AS F(QSTAT BIT 5)
6312	SMSYN	RQCA10	SIMULATE QMT SYNC AS F(QSTAT BIT 5)
6313	SKPKPD	RQCB6	SKIP ON CONTROL DESK KEYPAD, CLEAR FLAG ON SKIP
6314	ADVSR	RQCA10	ADVANCE SHIFT REG. ONE
6315	CSRGI	RQCA10	CLEAR SHIFT REG. INDEX REG.
6316	ZSRGI	RQCA10	SEND SHIFT REG. DATA TO FRONT OF SHIFT REG.
6317	IZSKP	RQCA10	SKIP IF INDEX 10-bits = ZERO.

RQC "INPUT" DEVICE CODES - C(channel)==>C(PDP8e ACC)

DEVICE CODE		CARD	FUNCTION
6320	HPR	RQCA6	C(AC) <== FRAME HOR. POSITION COUNTER
6321	HSR	RQCA6	C(AC) <== FRAME HOR. SIZE COUNTER
6322	VPR	RQCA6	C(AC) <== FRAME VERT. POSITION COUNTER
6323	VSR	RQCA6	C(AC) <== FRAME VERT. SIZE COUNTER
6324	QDAT1	RQCA4	C(AC) <== QMT BCD DATA LSW

6325	QDAT2	RQCA4	C(AC) <== QMT BCD DATA MIDDLE WORD
6326	QDAT3	RQCA4	C(AC) <== QMT BCD DATA MSW
6327	RQSTAT	RQCA4	C(AC) <== QSTAT
6330	RSRGX	RQCA10	C(AC) <== X CORDINATE SHIFT REG. DATA
6331	RSRGY	RQCA10	C(AC) <== Y CORDINATE SHIFT REG. DATA
6332	RSRGI	RQCA10	C(AC) <== SHIFT REG. INDEX REG.
6333	EXIN	RQCB/EXIN	C(C(XADR)==>C(AC)
6334	RFC1H	RQCA12	C(AC) <== FUNCTION COMPUTER 1 MSW
6335	RFC1L	RQCA12	C(AC) <== FUNCTION COMPUTER 1 LSW
6336	RFC2H	RQCA12	C(AC) <== FUNCTION COMPUTER 2 MSW
6337	RFC2L	RQCA12	C(AC) <== FUNCTION COMPUTER 2 LSW
6340	RKYPDH	RQCB6	C(AC) <== CONTROL DESK KEY PAD MSW
6341	FBW1	RQCB2	C(AC) <== CONTROL DESK DATA WORD 1
6342	FBW2	RQCB2	C(AC) <== CONTROL DESK DATA WORD 2
6343	FBW3	RQCB2	C(AC) <== CONTROL DESK DATA WORD 3
6344	FBW4	RQCB2	C(AC) <== CONTROL DESK DATA WORD 4
6345	FBW5	RQCB2	C(AC) <== CONTROL DESK DATA WORD 5
6346	FBW6	RQCB2	C(AC) <== CONTROL DESK DATA WORD 6
6347	FBW7	RQCB2	C(AC) <== CONTROL DESK DATA WORD 7
6350	FBW10	RQCB4	C(AC) <== CONTROL DESK DATA WORD 10
6351	FBW11	RQCB4	C(AC) <== CONTROL DESK DATA WORD 11
6352	FBW12	RQCB4	C(AC) <== CONTROL DESK DATA WORD 12
6353	RKYPDL	RQCB6	C(AC) <== CONTROL DESK KEY PAD LSW
6354	RMASKE	RQCA14	C(AC) <== MASK ENTRANCE DATA AS F(MSKADR)
6355	RMASKX	RQCA14	C(AC) <== MASK EXIT DATA AS F(MSKADR)
6356	RPENX	RQCA16	C(AC) <== LIGHT PEN X COORDINATE
6357	RPENY	RQCA16	C(AC) <== LIGHT PEN Y COORDINATE

RQC "OUTPUT" DEVICE CODES - C(PDP8e ACC)<==C(channel)

DEVICE CODE		CARD	FUNCTION
6360	HPL	RQCA6	FRAME HOR. POSITION COUNTER <== C(AC)
6361	HSL	RQCA6	FRAME HOR. SIZE COUNTER <== C(AC)
6362	VPL	RQCA6	FRAME VERT. POSITION COUNTER <== C(AC)
6363	VSL	RQCA6	FRAME VERT. SIZE COUNTER <== C(AC)
6364			
6365	SMOTR	RQCB10	SPARE MOTOR REGISTER<==C(AC)
6366	MSTAG	RQCB10	STAGE DIRECTION REG. <== C(AC)
6367	LSRGB	RQCA10	SHIFT REG. LOADING REG. <== C(AC)
6370	QPROG1	RQCA4	QMT PROGRAM WORD 1 <== C(AC)
6371	QPROG2	RQCA4	QMT PROGRAM WORD 2 <== C(AC)
6372	QPROG3	RQCA4	QMT PROGRAM WORD 3 <== C(AC)
6373	QPROG4	RQCA4	QMT PROGRAM WORD 4 <== C(AC)
6374	QSTAT	RQCA4	RQC PROGRAM WORD <== C(AC)
6375	LQDT1	RQCA4	QMT RIGHT DISPLAY LSW <== C(AC)
6376	LQDT2	RQCA4	QMT RIGHT DISPLAY MIDDLE WORD <== C(AC)
6377	LQDT3	RQCA4	QMT RIGHT DISPLAY MSW <== C(AC)
6420	DETB	RQCA8	DETECTOR THRESHOLD B <== C(AC)
6421	DETC	RQCA8	DETECTOR THRESHOLD C <== C(AC)
6422	DET1	RQCA8	1-D DETECTOR THRESHOLD 1 <== C(AC)

			(*NOT USED)
6423	DET2	RQCA8	1-D DETECTOR THRESHOLD 2 <== C(AC)
			(*NOT USED)
6424	DETDIG	RQCA8	DENSITOMETER 6-bit T1 AND T2 THRESHOLDS <== C(AC)
6425	SIZEA	RQCA8	AMENDER SIZE REG. <== C(AC)
6426	SIZEC	RQCA8	CLASSIFIER COLLECTOR SIZE REG. <== C(AC)
6427	SIZEM	RQCA8	MS3 COMPUTER SIZE REG. <== C(AC)
6430	SIZES	RQCA22	STANDARD COMPUTER SIZE REG. <== C(AC)
6431	QPROG5	RQCA22	QMT PROGRAM WORD 5 <== C(AC)
6432	QPROG6	RQCA22	QMT PROGRAM WORD 6 <== C(AC)
6433	QPROG7	RQCA22	QMT PROGRAM WORD 7 <== C(AC)
6434	QPROG8	RQCA22	QMT PROGRAM WORD 8 <== C(AC)
6435	DISP1	RQCA22	CONTROL DESK DISPLAY 1 <== C(AC)
6436	DISP2	RQCA22	CONTROL DESK DISPLAY 2 <== C(AC)
6437	LFBW2	RQCA22	CONTROL DESK SWITCH LIGHTS <== C(AC)
6440	MSKADR	RQCA14	MASK ADDRESS REG. <== C(AC)
6441	LMASKE	RQCA14	MASK ENTRANCE REG. <== C(AC)
6442	LMASKX	RQCA14	MASK EXIT REG. <== C(AC)
6443	LDXP	RQCA14	X COORDINATE CURSOR REG. <== C(AC)
6444	LDYP	RQCA14	Y COORDINATE CURSOR REG. <== C(AC)
6445	LPENX	RQCA16	LIGHT PEN X COORDINATE <== C(AC)
6446	LPENY	RQCA16	LIGHT PEN Y COORDINATE <== C(AC)
6447	PENST	RQCA16	LIGHT PEN STATUS REG. <== C(AC)
6450	EXADR	RQCB/EXIN/EXOUT C(AC)==>C(XADR)	
6451	EXOUT	RQCB/EXOUT C(AC)==>C(C(XADR))	
6452			
6453			
6455			
6456	LGALX	RQCB8	MIRROR SCANNER X COORDINATE <== C(AC)
6457	LGALY	RQCB8	MIRROR SCANNER Y COORDINATE <== C(AC)

BM "OUTPUT" DEVICE CODES - C(PDP8e ACC)<==C(channel)

DEVICE CODE		CARD	FUNCTION
6500	BMX4	-	load BM0 X coord reg<==8e ACC.
6501	BMX5	-	load BM0 X coord reg<==8e ACC.
6502	BMX6	-	load BM0 X coord reg<==8e ACC.
6503	BMX7	-	load BM0 X coord reg<==8e ACC.
6504	BMY4	-	load BM0 Y coord reg<==8e ACC.
6505	BMY5	-	load BM0 Y coord reg<==8e ACC.
6506	BMY6	-	load BM0 Y coord reg<==8e ACC.
6507	BMY7	-	load BM0 Y coord reg<==8e ACC.
6510	BMX0	BMD27	load BM0 X coord reg<==8e ACC.
6511	BMX1	BMD27	load BM0 X coord reg<==8e ACC.
6512	BMX2	BMD27	load BM0 X coord reg<==8e ACC.
6513	BMX3	BMD27	load BM0 X coord reg<==8e ACC.
6514	BMY0	BMD27	load BM0 Y coord reg<==8e ACC.
6515	BMY1	BMD27	load BM0 Y coord reg<==8e ACC.
6516	BMY2	BMD27	load BM0 Y coord reg<==8e ACC.
6517	BMY3	BMD27	load BM0 Y coord reg<==8e ACC.

6520	POSTA	BMD29	post selected group A BMs
6521	POSTB	BMD27	post selected group B BMs
6522	GETA	BMD29	enable acquiring group A BM pix or binary masks
6523	GETB	BMD27	enable acquiring group B BM pix or binary masks
6524	EXDMA1	EBUS2 (also BMD31,BMD25,BMB31)	load high RTPP DMA address bus
6525	EXDMA2	EBUS2 (also BMD31,BMD25,BMB31)	load low RTPP DMA address bus
6526	BMOUT	BMD25	BM maintenance output (spare)

BM "INPUT" DEVICE CODES - C(channel)==>C(PDP8e ACC)

DEVICE CODE		CARD	FUNCTION
6540	RGETA	BMD29	Read the status of done bits for BM GETA
6541	RGETB	BMD27	Read the status of done bits for BM GETB
6542	BMIN	BMD25	BM maintenance input (spare)
6543	RBMSP1	-	spare input (not implemented)
6544	RBMSP2	-	spare input (not implemented)
6545	RBMSP3	-	spare input (not implemented)
6546	RBMSP4	-	spare input (not implemented)
6547	RBMSP5	-	spare input (not implemented)

SECTION D

Examples of programming the GPP

Several examples of programming the GPP are given here as justification that a minimal set of GPP operators can be used to compute a fairly broad class of picture neighborhood functions.

A "communications" assembly language is used in the following examples. It mimics the GPPASM assembly language but is easier to read since the code is in infix notation rather than prefix. That is sink and source operands are denoted here by being on the left and right side of an arrow "<==".

Let comments be denoted by text enclosed in quotes "...". Let an instruction consist of 4 fields: the P3 field followed by "<==", followed by the P1 field, followed by the opr code, followed by the P2 field. Non-existant fields may be ignored. Thus,

```
        <P3> <== <P1> <OPR> <P2>;
or
        <P3> <== <OPR> <P1>;
```

D.1 Gradient used in Kirsch algorithm

The following gradient algorithm is given by Kirsch [Kir69]. Given a 3x3 I1 subarray, the following GPP program will compute the 2-D gradient by the formula "MAX [(3*SUM a(i - (5*SUM b(i)]". This PM program is written by repeating the code instead of using loops. It could be written with loops instead. The 8 permutations of the 8 neighbors are:

```
a1 a2 a3           b5 a1 a2           a2 a3 b1
b5 .  b1           b4 .  a3           a1 .  b2
b4 b3 b2           b3 b2 b1           b5 b4 b3
--------           ---------          ---------
Permutation 1      permutation 2 ...  Permutation 8
```

The following GPP program will compute the gradient at the current pixel. The neighborhood indexing scheme is restated for clarity.

```
                   3 2 1
                   4 8 0
                   5 6 7
```

```
let R1 be (3*SUM a(i)-5*SUM b[i]);
    R2 be MAX variable;
    R10 to R17 the local permutation values;
```

The computation for permutation 1 is given and the other 7 permutations are similar.

```
Step    Instruction
----    -----------
[1]     DRA <== I1(0) ADD I1(1)
[2]     R1 <== ADDST I1(2)
[3]     R1 <== #3 MUL R1
[4]     DRA <== I1(3) ADD I1(4)
[5]     DRA <== ADDST I1(5)
[6]     DRA <== ADDST I1(6)
[7]     DRA <== ADDST I1(7)
[8]     DRB <== MULST #5
[9]     R10 <== R1 SUB DRB
        .
        .
        .
```

These 9 steps give the value for 1 permutation. The 8 permutations take 72 steps. The code for the 8 permutations is just concatinated. Although it is not elegant, repeated code is obviously faster. We are able to avoid loops and increase processing speed at the expense of program memory.

To compute the maximum in R2 the following algorithm is used. Again, the PM implementation is repeated code.

```
max<==0;
offset<==9;
FOR i<=1 TO 8
  DO IF R(i+offset).GT.max
         THEN max <==R(i+offset);
```

```
Step    Instruction
----    -----------
[1]     DRA <== MOVE R10
[2]     DRA <== GTST R10
[3]     DRA <== GTST R12
[4]     DRA <== GTST R13
[5]     DRA <== GTST R14
[6]     DRA <== GTST R15
[7]     DRA <== GTST R16
[8]     DRA <== GTST R17
[9]     R2 <== MOVE DRA
[10]    Haltpoint.
```

Therefore, it takes 72+10=82 instructions/pp to compute the gradient related function. At approximately 300 nsec/GPP instruction, it would take about 30 usec/pp or 256 X 256 x 30usec. or 1.9 seconds to do the entire picture exclusive of I/O. A 128 X 128 picture would take on the order of .3 seconds.

D.2 Eight neighbor direction list processing

This is an example of 8 neighbor direction list processing. Since all 8+ (9) neighbors are directly addressable, various functions may be computed by iterating in a FOR loop construction using a direction list in the general purpose registers R(i).

For example a simple angle finder might consist of the following algorithm:

```
I3(8)<==0;
For i<==0 Step 1 Until 8 Do
  If [SUM a(i)*R(i) ] > threshold
        Then I3(8)<==1;
```

This will give a first order approximation to a 135 degree line finder.

A neighborhood direction list R(i) might look like

```
+1 -1 -2
-1 +1 -1
-2 -1 +1.
```

Let R1 through R9 be the direction list in the GR using The same addressing scheme as the current Neighborhood.
Let R10 be a 9 counter and pointer to R1:R9.
Let A0 auto-index register point to I1.
Let R12 be a temporary register
Let R13 be the threshold.

```
Step    Instruction
----    -----------
[1]     I3(8) <== MOVE #0
[2]     R10 <== MOVE #9  "form the filter list pointer"
[3]     R12 <== MOVE #0
[4]     A0 <== INC #I1(8) "form the I1 list pointer"
[5]     DRA <== 'R10 MUL 'A0D  "process the lists"
[6]     R12 <== ADDST R12  "sum the result"
[7]     R10 <== R10 DECB 9  "test if done"
[8]     [5] <== JUMP
[9]     [11] <== R12 BLE R13
[10]    I3(8) <== MOVE #1
[11]    Haltpoint.
```

D.3 Edge and curve detection

The following is an example of edge and curve detection using the algorithm in Rosenfeld, Lee and Thomas [JohnE70]. In their article, differences of averages are used as measures of texture differences. They discuss a 2-D texture measure D(rs,hk).

```
D(rs,hk)=ABS(
        [a(h+r,k+s)+...+a(h+r,k-s)+...+
         a(h+1,k+s)+...+a(h+1),k-s) ]
       -[a(h,k+s)+...+a(h,k-s)+...+
         a(h-r+1,k+s)+...+a(h-r+1,k-s) ])/(r*(2s+1.
```

They suggest using the measure

```
D(23,hk)*D(43,hk)*D(83,hk)
```

to separate regions by horizontal edges. For edges in other orientations, analogous operations would be optained by rotating D(rs,hk).

For use as an example to measure the complexity of coding these functions in the GPP. Let us look at D(23,hk). For each (h,k) in the picture, transform its position to (0,0). Then D(23,hk) reduces to D(23,00).

```
D(23,00)=ABS(
        [a(2,3)+a(2,2)+a(2,1)+a(2,0)+
         a(2,-1)+a(2,-2)+a(2,-3)+
         a(1,3)+a(1,2)+a(1,1)+a(1,0)+
         a(1,-1)+a(1,-2)+a(1,-3) ]
       -[a(0,3)+a(0,2)+a(0,1)+a(0,0)+
         a(0,-1)+a(0,-2)+a(0,-3)+
         a(-1,3)+a(-1,2)+a(-1,1)+a(-1,0)+
         a(-1,-1)+a(-1,-2)+a(-1,-3) ])/(2(2*3+1.
```

This corresponds to the 4x7 array:

-	-	+	+
-1,3	0,3	1,3	2,3
-1,2	0,2	1,2	2,2
-1,1	0,1	1,1	2,1
-1,0	0,0	1,0	2,0
-1,-1	0,-1	1,-1	2,-1
-1,-2	0,-2	1,-2	2,-2
-1,-3	0,-3	1,-3	2,-3

The program to compute D23 is given in terms of macros which the compiler/assembler for the RTPP might have implemented.

```
Define D23 = <  RY <== MOVE #256;
        A:      DOLINE (y);
                RY <==RY DECB B;
                A <== JUMP;
        B:      HLT;>

Define DOLINE (y) = < "get data";
                GETLINE(y,#3,R0);
                GETLINE(y,#2,R258);
                GETLINE(y,#1,R516);
                GETLINE(y,#0,R774);
                GETLINE(y,#-1,R1032);
                GETLINE(y,#-2,R1290);
                GETLINE(y,#-3,R1548);
        "Set up line pointers - note buffers are 258"
                A1 <== MOVE #table;
                'A1I <== MOVE #R0;
                'A1I <== MOVE #R258;
                'A1I <== MOVE #R516;
                'A1I <== MOVE #R774;
                'A1I <== MOVE #R1032;
                'A1I <== MOVE #R1290;
                'A1I <== MOVE #R1548;
        "Set up output buffer"
                'A1I <== MOVE #R1806;
        "Do 256 direction list operations/line"
                RCOUNT <== MOVE #256;
        A:      TXT47(table)
                RCOUNT <== RCOUNT DECB B;
                A <== JUMP;
        B:      SAVELINE(y,R1807);>

Define SAVELINE (sline,r) = < "copy line buffer "r" in gr to line
        'Line' in buffer memory BM3"
                IOCLR;
                XRST #(<I3>,<x>);
                A1 <== MOVE #R;
        A:      I3(5) <== MOVE 'A1I;
                A <== XCLKB #(<I3>,<x>);
                (16-bit,out,hor,BM3,I3) LINE ysline; >

Define GETLINE (sline,offset,r) = < "copy BM1 line 'line' into gr
        Buffer at 'r'.
                Gline <==line ADD offset;
                IOCLR;
                XRST #(<I1>,<X-1,X,X+1>);
                (low,in,hor,BM1,I1,y) LINE sline;
                A1I <== MOVE #R;
        " Get the x=-1 boundrypoint "
                'A1I <== MOVE I1(4);
        A:      'A1I <== MOVE I1(0);
                A <== XCLKB #(<I1>,<X-1,X,X+1>);
        " Get pixels x=256,257 boundries "
                'A1I <== MOVE I1(4);
                'A1I <== MOVE I1(0); >

Define TXT47 (table) = < "do a 4 X 7 local texture operation"
                RCOUNT <== MOVE #7;
                A2 <== MOVE #table; "buffer pointer"
                rsum <== MOVE #0;
        A:      A3 <== MOVE 'A2I; "buffer data"
                DRA <== MOVEN 'A3I;
                DRA <== SUBST 'A3I;
                DRA <== ADDST 'A3I;
                DRA <== ADDST 'A3I;
                rsum <==ADDST rsum;
                RCOUNT <== RCOUNT DECB B;
                A <== JUMP;
        B:      A3 <== INC  'A2I;
                'A3I <== MOVE rsum;>
```

The timing for the complete algorithm may be determined as follows: each output line has 7 inputs or a total of 8 BM random I/O accesses/line. Since the total maximum 1 way access and transfer time for a BM is 147 msec, the I/O time is 8*0.147 seconds = 1.2 seconds. Assuming an average instruction time of 300 nsec/instruction, exclusive of I/O, the following times for the above macros are computed:

Macro	#Instr. Execs./Call	#Times called	Total # instr
TXT47	5 + 8*7 = 61	256x256=65536	4.00xE+5
GETLINE	8 + 2*256 = 520	7*256 = 1792	9.31xE+5
SAVELINE	4 + 2*256 = 516	256	1.32xE+5
DOLINE	10 + 2*256=522	256	1.22xE+5
D23	1 + 2*256 = 513	1	513

The total number of instructions is on the order of 1.59 million and takes (at 300 nsec/instruction) about 0.47 seconds. Therefore, the total D23 macro execution takes on the order of 1.7 seconds it should be noted that the D23 algorithm could be done more directly from the line buffers for any size d(rs,00) and would run faster because data would not have to be copies into the GR.

D.4 Histogram computation

The following program will compute the gray scale histogram of the I1 picture and leave the results in general registers R0 (address 0) to R63 (address 63). It is assumed that R[0:63] are initialized to 0 by the PDP8e. After the GPP is finished, the PDP8e may read the general registers to access the resultant histogram. The algorithm is as follows:

```
        For all picture pixels
                DO R[I1(8)]<==R[I1(8)]+1;
```

step	Instruction
1	'I1(8) <== INC 'I1(8)
2	Haltpoint.

D.5 Haltpoint I/O example

This example shows roughly how "haltpoint" I/O would be done for I1==>I3. First obtain buffer memory data line by line and compute on a 3x3 neighborhood with some function. Then output I3 into BM3.

Raster I/O algorithm

```
[1.0] Load I1 buffer Y-1, Y, and Y+1 with the first 3 lines
      of BM data.

      IOCLR;
      YRST <I1,I3>;
      (low,in,hor,BM1,I1,Y+1) LINE I1Y;

      YCLK <I1>;
      (low,in,hor,BM1,I1,Y+1) LINE I1Y;

      YCLK <I1>;
      (low,in,hor,BM1,I1,Y+1) LINE I1Y;
[1.1] Reset the X counters to the front of the line.

      XRST #(<I1,I3>,<X-1,X,X+1>);

[1.2] Process first line  and  store  in  I3  buffer.  That  is
      Perform I1*==>I3.

      [Neighborhood procedure];
      [1.2] <== XCLKB #(<I1,I3>,<X-1,X,X+1>);

[2.0] Write out the I3 buffer and then load next BM line in  I1
      buffer.

      (low,out,hor,BM3,I3,Y)  LINE I3Y;
      [2.1] <== YCLKB #(<I1,I3>);
      [3.0] <== JUMP;

[2.1]  (low,in,hor,BM1,I3,Y) LINE I1Y;
      [1.1] <== JUMP;

[3.0] End. Stop GPP and notify PDP8e.
      HLT
```

D.6 Random Neighborhood Accessing

In the case where one whishes to random access a neighborhood, quite a savings may be realized by using the GETIj neighborhood fetch instruction to fetch BM data into the current neighborhood of line buffer j. The following sequence will process neighborhood (x,y) of BM3 and store it into pixel (x,y) of BM7.

```
        yxaddress <== x MAKYXADDR y;
        GETI1 (low,BM3), yxaddress;

        [Process the current I1 neighborhood==> I3(8) ]

        PBM7 <== MOVE yxaddress;
        'PBM7 <== MOVE I3(8);
```

D.7 Area and perimeter computation

Example 6 shows how one might compute total area (sum of pixels of an object > threshold) and total perimeter (total count of entrance and exit pixels of detected regions) of all objects in the scene.

```
" Compute area "
START:  AREA <== MOVE #0;
A:      C <== I1(8) BGE th;
B:      Haltpoint;
        A <== JUMP;
C:      AREA <== INC  AREA;
        B <== JUMP;



"Compute perimeter of BMi under mask BMj.
 note that SWITCH is true inside of a blob".
Define TRUE=1, FALSE=0;
Variable MASKEDI1, PERIMETER, INBLOB, THRESHOLD,

        PERIMETER <==MOVE #0;
        IOCLR;

"Get the next line for the image and its mask"
GETLINE: (low,in,hor,BMi,I1,Y) LINE I1Y;
         (low,in,hor,BMj,I2,Y) LINE I2Y;
         SWITCH <== MOVE FALSE;

"Position the line buffers' dynamic address vectors at left"
         XRST #(<I1,I2>,<X>);
TEST:    MASKEDI1 <==I1(8) AND I2(8);

"Test if in a blob"
         INBLOB <== MASKEDI1 BGE THRESHOLD;

"Test if leaving a blob"
         YES <== SWITCH BNE #0;
NEXTX:   TEST <== XCLKB #(<I1,I2>,<X>);

"Reset switch after each line"
         SWITCH <== MOVE FALSE;
         GETLINE <== YCLKB <I1,I2>;

"Send data back to PDP8e which sends it to PRDL"
         GOUT <== MOVE PERIMETER;

"Signal DDTG that the function is done"
         HALT;

"Test if just entered an object"
INBLOB: YES <== SWITCH BEQ #0;
         NEXTX <== JUMP;

"Yes, a perimeter point, increment perimeter
```

```
          and reverse the switch"
YES:      PERIMETER <==INC PERIMETER;
          SWITCH <==  MOVE TRUE;
          NEXTX <== JUMP;
```

Each pixel in computing the perimeter takes at most 8 instructions in addition to 256 input instructions at 300 nsec/avg-instruction (exclusive of I/O). This gives a computation time of about 0.15 seconds and 0.147 seconds I/O. The total time is about 0.3 seconds. As the I/O cost in time is comparable to the computation time, it would be more economical to combine several simple primitives together such as area, perimeter, etc. while doing a single I/O traversal through the BM.

D.8 Run length code of detected object

Example 7 computes the run length code of a single object extracted from an isolated gray scale image by thresholding it similarly to the perimeter program and then saving entrance and exit pixels (instead of incrementing the perimeter primitive as in example 6).

```
        "Compute run-end codes"
                Define TRUE=1, FALSE=0;
        Variable SWITCH,EN,EX,TH;
        "Initial the stacks"
                A1 <== MOVE #STACK;
                A2 <== A1 ADD #256;
                IOCLR;
     GETLINE:   (low,in,hor,BM1,I1,y) LINE I1Y;
                SWITCH <== MOVE FALSE;
                XRST #(<I1>,<x>);
                EN <== MOVE #256;
                EX <== MOVE #256;
        TEST:   INBLOB <== I1(8) BGE TH;

        "TEST if leaving a blob"
                YES <== SWITCH BGE #0;
        NEXT:   TEST <== XCLKB #(<I1>,<x>);

        " Reset SWITCH after each line"
                SWITCH <== MOVE FALSE;

        " Push the entrance and exit pixels"
                'A1I <== MOVE EN;
                'A2I <== MOVE EX;
                GETLINE <==YCLKB <I1>;
                HLT;

        " TEST if just entered object"
       INBLOB:  YES <== SWITCH BEQ #0;
                NEXTX <== JUMP;

        "YES, save the run-end codes"
        YES:    ENTRY <==SWITCH BEQ #0;
        EXIT:   EX <== MOVE I1X;
                REVERSE <== JUMP;
        ENTRY:  EN <== MOVE I1X;
        REVERSE: SWITCH <== MOVEC SWITCH;
                NEXTX <== JUMP;
```

SECTION E

GPPASM BNF Grammar Specification

The BNF grammar specification is given for the GPPASM assembler to be used to assemble RTPP programs. Note that MAINSAIL will generate GPPASM assembly language output. The PDP8e segment will be assembled by the PAL8 assembler on the PDP8e under control of GPPASM.

Note that ID is any identifier which is not a keyword in the grammar, INT is any integer, and <text> is any text not including the symbols , ", or ;. EPSILON is the null string.

```
<program>::= <GPPsegment> <PDP8Esegment> END

<PDP8Esegment>::= BEGINPDP8E <PAL8program> ENDPDP8E | EPSILON
<PAL8program>::= PDP8e program - see PDP8e OS/8 handbook

<GPPsegment>::= <statement> <GPPsegment>| <statement> ENDGPP |
                 DECIMAL | OCTAL | EPSILON

<statement>::= <PM-label> <PM-substatement> |
                <GR-label><GR-substatement> |
                <statement> ; <text>
<PM-substatement>::= <comment> | REQUIRE <file> SOURCE  |
                REQUIRE <file> LOAD  | DEFINE ID = <value> |
                PMORIGIN <value> | GRORIGIN <value> |
                <opcode> <P1> , <P2> , <P3>

<GR-substatement>::= GRBLOCK <GR-list>

<file>::= <device> <fname> . <ename>
<device>::= SYS: | DSK: | DSKB: | DSKC: | DSKD: | DSKE: | DSKF: |
                DSKG: | DSKH: | DTA0: | DTA1:
<fname>::= ID
<ename>::= ID

<comment>::=  text ; | " text "

<P1>::= ' <GR-address> | <GR-address> | # <value> | <I/O list>
<P2>::= ' <GR-address> | <GR-address> | # <value>
<P3>::= ' <GR-address> | <GR-address> | <PM-address>

<value>::= <land> | <value> ! <value1> | <ae>
<value1>::= <ae> | <land>
<land>::= <value> & <value1>

<ae>::= <sae>
<sae>::= <term> | <sae> + <term> | <sae> - <term>
<term>::= <factor> | <term> * <factor> | <term> / <factor>
<factor>::= <primary>
<primary>::= <PM-label> | <GR-label> | ( <value> ) |
                + <primary> | - <primary>
```

```
<PM-label>::= <label> | EPSILON
<GR-label>::= <label> | EPSILON
<label>::= ID :

<PM-address>::= <PM-label>

<opcode>::= MOVE | JUMP | PUSHJ | POPJ | INCB | DECB | BEQ | BGE |
            BLT | BGT | BLE | BNE | HLT | AND | NAND | XOR |
            IMPLIES | OR | NOR | EQV | MOVE | MOVBIT | MOVBS |
            SHIFTR | SHIFTL | ROTR | ROTL | GTST | LTST | GEST |
            LEST | DMOVE | DSWP | ADD | SUB | MUL | DIV |
            ADDST | SUBST | MULST | DADD | DSUB | DMUL |
            INC | DEC | MOVEN | MOVEC | DMOVEC | IOCLR | YRST |
            XRST | XCLKB | XCLK | YCLKB | YCLK | LINE |
            MAKYXADDR | GETI1 | GETI2 | GETI3


<GR-address>::= <value> | <GR-I/O-address>
<GR-I/O-address>::= <neighborhood-pixels> | <auto-index> |
            <indirect-BM-addresses> | <TTY-I/O> | <control-desk> |
            <status-registers> | <dynamic-address-vectors> |
            <GR-I/O-registers>

<neighborhood-pixels>::= I10 | I11 | I12 | I13 | I14 | I15  |  I16  |
            I17 | I18 | I20 | I21 | I22 | I23 | I24 | I25 | I26 |
            I27 | I28 | I30 | I31 | I32 | I33 | I34 | I35 | I36 |
            I37 | I38
<auto-index>::= A0D | A0 | A0I | A1D | A1 | A1I | A2D | A2 | A2I |
            A3D | A3 | A3I | A4D | A4 | A4I | A5D | A5 | A5I |
            A6D | A6 | A6I | A7D | A7 | A7I
<indirect-BM-addresses>::= PBM0 | PBM1 | PBM2 | PBM3 | PBM4 | PBM5 |
            PBM6 | PBM7 |
<TTY-I/O>::= KRB | KSTATUS | TLS | TSTATUS
<control-desk>::= SW1 | SW2 | SW3 | SWA | DSPLYA | DSPLYB | DSPLYC |
            KNOB01 | KNOB23 | KNOB45 | KNOB67 |
<status-regists>::= PDLCNT | PDL | FXAR | DRA | DRB | EXAR | EEXAR |
            EDRB | STATUS
<dynamic-address-vectors>::= I1XM | I1X | I1XP | I1Y | I2XM | I2X |
            I2XP | I2Y | I3XM | I3X | I3XP | I3Y
<GR-I/O-registers>::= GIN | GOUT

<I/O list>::= ( <leftbracket> <list> <rightbracket> ) |
            <I/O list> ! <I/O symbol> | <I/O symbol>

<list>::= <list> , <I/O symbol> | <list> , INT | <I/O symbol> |
            INT

<leftbracket>::= <
<rightbracket>::= >
<I/O  symbol>::=  I1  | I2 | I3 | XP | X | XM | YP | Y | YM | right |
            left | BM0 | BM1 | BM2 | BM3 | BM4 | BM5 | BM6 | BM7

<GR-list>::= <GR-allocation-size> | 0 <preload>
<GR-allocation-size>::= INT
<preload>::= <list-of-values> | TEXT   <text>
<list-of-values>::= <list-of-values> , <value> | <value> |
              <repeat-times> [ <list-of-values> ]
```

```
<repeat-times>::= <value>
<text>::= text string containing no   , ", or ;
```

INDEX