MICROPROGRAM CONTROL ARCHITECTURE FOR THE GENERAL PICTURE PROCESSOR

NCI/IP Technical Report #22

April 22, 1977

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

National Cancer Program

NATIONAL CANCER PROGRAM

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

NCI/IP-77/05

Microprogram Control Architecture for the General Picture Processor

NCI/IP Technical Report No. 22

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

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

*Carman Electronics, Inc.
Corvallis, Oregon

April 22, 1977

Abstract

The General Picture Processor, running on the Image Processing Units' Real Time Picture Processor, will use a microprogrammable control structrue. The hardware design specifications for this control is presented.

TABLE OF CONTENTS

SECTION 1

Introduction

The General Picture Processor, GPP, is an integral part of the Real Time Picture Processor (RTPP) ([Carm74], [Lem74], [Lem76a], [Lem76b]) at the National Cancer Institute. This document explains the microprogrammable control structure of the GPP. The reader is expected to be fimiliar with the GPP from the mentioned references, most important of which is [Lem76a].

The GPP is basically a register to register transfer machine. Because the GPP is also an experimental laboratory picture processing computer, which requires a dynamic machine instruction set, a microprogrammable control structure was selected. The microprogram commands are sufficiently powerful to allow the microprogamming of detailed high level picture processing instructions.

With reference to figure 1, the GPP contains a microprogram memory, MPM, a reentrant GPP instruction program memory, PM, a data memory, DM, and a set of arithmetic logic units, ALUs. The PDP8e may read or write each of the memories. The dynamic microprogram memory controls the GPP while the program memory contains the GPP machine instructions. The data memory addresses most GPP special registers and cashe image line buffers which are interfaced directly to the RTPP buffer memories [Lem76a section 4]. Two 16-bit address and two 16-bit data buses plus microcommand signals are routed via the G-bus. Most data communications between GPP modules is done via the G-bus.

This document will present the GPP microprogram control structure, description of microcommands, and examples of microprograms.

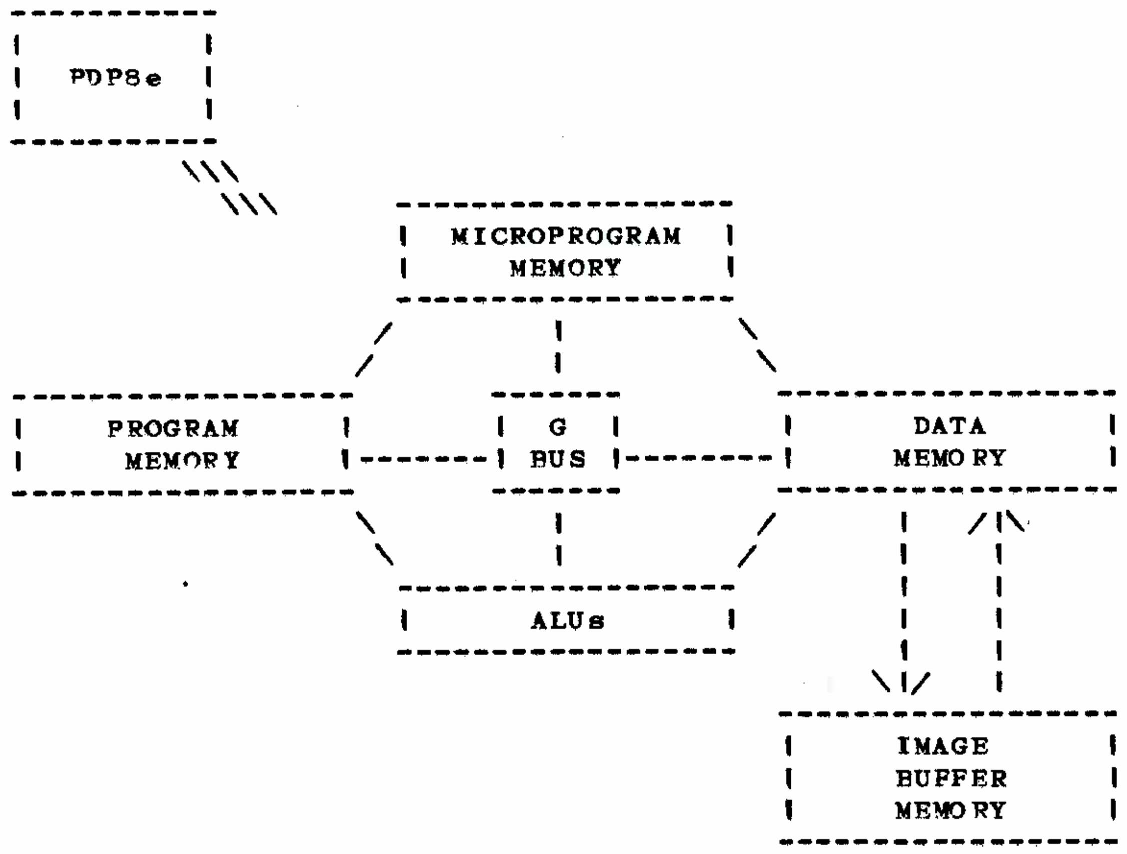

Figure 1: General Picture Processor.
The microprogram memory, MPM, program memory, PM, and data memory, DM, are all read/write via the PDP8e. Most communications between GPP modules is done via the G-bus. The image buffer memory, BM, is interfaced directly to the data memory space cashe image line buffers for fast data transfers.

SECTION 2

Microprogram structure

The GPP microprogram control structure uses a single phase clock to execute the microcommands. The microprogram commands are first decoded from the microprogram memory as a function of the microprogram address register, MPAR[0:12], and then distributed throughout the GPP (as required) to be executed on the next clock cycle. While the present microcommand is being executed the next microcommand is being decoded. This look-ahead decoding is interrupted only when a conditional micro-jump occurs.

The microprogram address register, MPAR[0:12], is loaded as a result of the contents of the program memory OPRgroup[0:8] datum. As seen in figure 2, the GPP instruction addressed by the program counter is first decoded from the program memory. The OPRgroup[0:8] is used to select the microprogram which will execute this instruction. The nine bits of the OPRgroup may address 512 individual microprograms. The starting address for these microprograms in the microprogram memory is stored in the mapping memory, MM. The OPRgroup is used as the address to the mapping memory. The decoded data from the mapping memory is stored in the microprogram address register to be used as the starting address of the microprogram which executes the GPP instruction in the program memory, PM (see figure 2).

```
-------------------------------------------------------------------

        ------------   32-bits      ---------------------
        | PROGRAM  |<--------------| PROGRAM COUNTER AND |
        |  MEMORY  |    Address    | PROGRAM FIELD REG.  |
        ------------               ---------------------
              | Output
      64-bits |          OPRgroup[0:8]
              |-------------- 9-bits
             \|/            \|/ Address
           -------      ------------
           |  G  |      | MAPPING  |<--> PDP8e
           | BUS |      |  MEMORY  |   read/write
           -------      ------------
                             | 13-bits
                            \|/ Input data
                    ------------------
                    |  MICROPROGRAM  |
                    |ADDRESS REGISTER|
                    ------------------
                             | 13-bits
                            \|/ Address
                     ----------------
                     | MICROPROGRAM |<--> PDP8e
                     |    MEMORY    |   read/write
                     ----------------
                             | 128-bits max.
                            \|/
                        MICROCOMMANDS
```

Figure 2: Fetching the microcommands.
The output of the program memory subfield OPRgroup[0:8] is used to fetch the starting address from the mapping memory of the microprogram which will execute the GPP instruction addressed by the program counter.

2.1 Program memory

Becaues the GPP is pipelined and may fetch several PM arguments while processing one GPP instruction several program memory buffering registers are required. Figure 3 depicts the program memory and the required registers and logic needed to interface with the mapping memory and GPP address, data, and control buses (all labeled the G-bus). When the program counter is loaded by a microcommand, a program memory fetch sequence is started. The program memory will then supply 64-bits of data to the program memory interface. This data is stored in either the program memory register, PMR[0:63], or the program memory buffer register, PMBR[0:63]. The program memory data segment, OPRgroup[0:8], may be used as an address by the mapping memory. The first 64k field (field 0) is connected to one of two memory ports on the program memory interface. The second memory port is made available for additional memory which may vary in access time from field 0. The program memory interface also contains four 16-bit registers (PMWRi, i=0,1,2,3) addressed in the data memory space. These registers are used when writing on the program memory with data from the PDP8e.

The program memory registers', PMR, contents is distributed to the GPP as PM arguments via the G-bus. When the PM data is stored in this register, the program memory is free to fetch another PM word. This second PM word may be saved in the program memory buffer register, PMBR, to be used by the PMR if required. A third PM word may then be fetched which could be stored in either the PMR or PMBR. Using the PMBR, two PM words can be made available to the PMR. This double buffering of the PM arguments enables the two PM datums (incremented address and branched address) in a conditional branch instruction to be immediately available to the program memory register, PMR. As can also be seen in figure 3, both the PM OPR[0:8] datum from the PM or PMBR can be used by the mapping memory to obtain the starting address of the next microprogram.

In order to fetch several PM arguments and choose which will be used as the next GPP instruction, the program counter, PC, must be double buffered. Figure 4 depicts the program counter logic and buffers. The 16-bit program counter can directly address any PM argument in a PM 64k field. The program counter buffer register, PCBR, is used to save the present PC data plus one for restoring the program counter during a conditional branch where the condition fails.

The program field register, PFR, is used to select which PM field the PC will address. The program field buffer register, PFBR, is used to save the future program field register data until it is required. For example, when a GPP jump instruction is executed the PFBR is loaded into the PFR and the next GPP instruction is fetched from that field. The PFBR is addressed in the data memory space and therefore may be loaded by a MOVE instruction prior to the JUMP instruction.

A last in first out push down list for the PC and PFR is used to implement the GPP PUSHJ and POPJ instructions (see figure 4). The push down list, PDL, is 32-bits by 1K words deep and is built with 1k RAMs. The push down list counter, PDLCTR, is used as the address to these RAMs. Microcommands are available to increment and decrement this counter as well as push and pop the PC and PFR into and from the PDL. The PC and PFBR are able to be read/written by the PDP8e, but the PFR is read only by the PDP8e. The PC and PFR are addressed in the data memory space for read only. The PFBR is addressed for both read and write in the GPP data memory space.

```
--------------------------------------------------------------------------------

    ---------------                                         ---------------
    | PM MEMORY   |                                         |  PM MEMORY  |
    |   FIELD 0   |                                         |  FIELD 1-N  |
    ---------------                                         ---------------
           |                                                       |
           -----------------------------------------------------------
                                  | 64-bits
    -----------------------------------------------------------------------
    |                      PROGRAM MEMORY INTERFACE                       |
    |  (program counter, PMWR 0-3, all read/write logic)                  |
    -----------------------------------------------------------------------
                                  | OPR, P1, P2, P3
                                  |
            -------------------------------------------------
            |                     | 64-bits                 |
            |        -------------------------              |
            |        | PM BUFFER REGISTER    |              |
            |        |        PMBR           |              |
            |        -------------------------              |
            |              | 64-bits        |               | 9-bits
            |             \|/               |              \|/ PMopr[0:8]
            |            -------            |             -------
            ------->|  OR  |                ------->|  OR  |
         64-bits     -------       PMBRopr[0:8]       -------
                        |                                 |
                       \|/ 64-bits                       \|/ 9-bits
                 -----------------                 To mapping memory
                 |  PM REGISTER  |                   address input
                 |      PMR      |
                 -----------------
                        |
                       \|/ 64-bits
                    -----------
                    |  G BUS  |
                    -----------
```

Figure 3: Program memory logic.
Data stored in the program memory may be saved in one of two registers. The PMBR is used as a buffer register to the PMR. The OPRgroup[0:8] which is used as the address for the mapping memory may be obtained by either the program memory or the PMBR.

```
--------------------------------------------------------------------------------
```

```
                                    PDP8e                PDP8e
                                     /|\                  /|\
                                     \|/                  \|/
                                    ---------------------------
                                    |  PROGRAM FIELD BUFFER   |
                                    |  REGISTER, PFBR[0:15]   |
                                    ---------------------------
  PDP8e                G BUS                    |       PDP8e
    |                    |             16-bits \|/       /|\
  -----------------------------     ---------------------------
  | PROGRAM COUNTER,PC[0:15]  |     |     PROGRAM FIELD       |
  -----------------------------     |  REGISTER, PFR[0:15]    |
                                    ---------------------------
      |        /|\     |
     \|/        |      |                        |
  ---------     |      ------------------       |
  |   +1  |     |                       |       |
  ---------     |              16-bits  |       |
      |         |                       |       | 16-bits
     \|/        | 16-bits              \|/     \|/
  ---------------------------       ---------------------------
  | PROGRAM COUNTER BUFFER  |       |   1K PUSH DOWN LIST     |
  |    REGISTER, PCBR       |       |   FIRST IN FIRST OUT    |
  ---------------------------       ---------------------------
                                               /|\ 10-bits
                                                |
                                    ---------------------------
                                    |     PUSH DOWN LIST      |
                                    |     ADDRESS COUNTER     |
                                    ---------------------------
```

Figure 4: Program counter logic.
The program counter contents plus one can be stored in the program counter buffer register, PCBR, to be later loaded into the program counter. Under microprogram control the program field buffer register, PFBR, is loaded into the program field register, PFR. Both the program counter and program field register may be pushed and popped from the push down list.

2.2 Microprogram memory

A microinstruction is the total of all the microcommands stored in the microprogram memory addressed by the microprogram address register, MPAR[0:12]. These microcommands are active when there respective addressed memory location is a logical true. A complete discription of the microcommands may be found in section 3. In order for the microcontrol logic to address data space arguments, 32 microcommands are assigned to two microcommand fields called the MFA and MFB. Both these 16-bit microcommand fields may be used as addresses or arguments in the data memory space. The MFB[9-15] field may also be used to drive the OPRalu[9-15] bus. This allows the microcontrol logic to address or supply arguments to the ALU A or B registers and then select the ALU used for the required operation. These features enable high level GPP machine instructions, which use a combination of ALUs, to be microprogrammed.

The microprogram address register, PMAR, can be set directly to zero (see figure 5). The zero address is used as the starting address for the initial start up microprograms.

The PDP8e may start and stop the microprogram clock which allows the PDP8e to load the miroprogram and mapping memories. A monitor microprogram may then be started at MPAR address zero. This monitor microprogram may wait for instructions from the PDP8e to load the program or data memory or execute GPP programs.

The microprogram starting address to execute a GPP instruction is fetched from the mapping memory (addressed by either the program memory buffer register OPR field, PMBRopr[0:8], or the output of the program memory OPR field, PMopr[0:8],) and loaded directly into the microprogram address register.

Because the mapping memory contains 1k 12-bit words and the PMBRopr[0:8] or PMopr[0:8] are both only 9-bit addresses used by the mapping memory a third 10-bit mapping memory address is provided via the data bus, D1[6:15] (see figure 5). This allows for the program memory to supply or address the argument which will provide the starting addresses of a special instruction other than the 512 defined in the program memory OPR. The GPP APPLY instruction is implemented with this feature.

A microprogram direct jump is implemented via the microcommand field MFA[3:15] loaded into the microprogram address register. A microprogram relative jump is implemented using the present contents of the microprogram address register added to the microcommand field MFA[3:15] and loaded into the microprogram address register.

The microprogram address register is incremented by adding one to the present contents of the MPAR and then loading the result into the MPAR.

Microprogram push and pop microcommands are implemented using a four word last in first out push down stack as shown in figure 5. To execute a microprogram push-jump the present contents of the MPAR plus one is stored in the push down stack while the microcommand field MFA[3:15] are loaded into the MPAR. To execute a microprogram pop-jump the contents of the top of the push down stack is loaded into the MPAR.

Any data memory addressed device or ALU addressed device may disable or stall the microprogram clock. This allows slower devices to complete their respective functions before the beginning of a new cycle. For example, the divide ALU may require part of several cycle times to complete its function.

The divide ALU simply asserts the MHOLD signal bus, which is distributed throughout the GPP, and the microprogram control logic disables the microcycle clock. When the divide is complete the ALU releases the MHOLD signal bus whereby the microcycle is restarted. This feature not only allows ALUs to take some extra needed processing time, but also allows special slower data and program space memories time to complete their read or write function before the next cycle occurs.

```
-----------------------------------------------------------------

  ----------------       --------------        ------------
  | PMBRopr[0:8] |       | PMopr[0:8] |        | D1[6:15] |
  ----------------       --------------        ------------
     \|/ 9-bits              \|/ 9-bits      10-bits \|/
  ------------------------------------------------------------
  |          MAPPING MEMORY, MM  (3 address sources)         |
  ------------------------------------------------------------
                             \|/ 12-bits
                           -------
                           | OR |<---- MFA[3:15]
                           -------
     12-bits                 \|/ 12-bits
  --------               --------------------------------
  |     \|/ B          \|/ A                            |
  |    ---------------------------------                |
  |    |  MICROPROGRAM ADDRESS ADDER: |                |
  |    |     C = B + 1, or C = A + B   |                |
  |    ---------------------------------                |
  |      12-bits | C                                    |
  |              |---------------------------------     |
  |              |        -----------------      |      |
  |             \|/     /|\      12-bits \|/    \|/    \|/
  |    ----------------------      -------------------------
  |    |       4 WORD       |      |      MULTIPLEXER      |
  |    |  PUSH DOWN STACK   |      -------------------------
  |    ----------------------                | 12-bits
  |                                          \|/
  |                                -------------------------
  |                      ZERO ---> | MICROPROGRAM ADDRESS |
  |                                |    REGISTER, MPAR     |
  |                                -------------------------
  -------------------------------------------| 12-bits
                                            \|/
                                   -------------------------
                                   |     MICROPROGRAM      |
                                   |     MEMORY, MPM       |
                                   -------------------------
                                     | Output (128-bits max.)
                                    \|/
                                MICROCOMMANDS:
                                     GPP control, MFA, and MFB.
```

Figure 5: Microprogram memory logic.
The microprogram address register, MPAR[0:12], may be loaded from several sources. Data addressed by one of three address sources to the mapping memory or the MFA register may be loaded directly into the MPAR. The MPAR may be incremented or loaded by the output of the four word push down stack. The output of the microprogram memory addressed by the microprogram address register is used as the microcommands to control the execution of a GPP instruction.

2.3 Data memory

The data memory space is divided into the I/O data memory space and the general registers, GR, data memory space. The I/O data space contains addresses for most special control registers and is addressed in the top 2k of the 64k of the data memory space. The GR is a fast (100 nsec.) interleaved memory addressed in the first 62k of the data space.

Register to register data transfer in the data memory space is accomplished using a separate read and write microcommand function. An address and a microcommand to read are used to address and decode an argument to be read in the data memory space. The argument is then temporarily stored in a read data register, RDR, one of which is present on most data memory modules (see figure 6). This intermediate storage register is necessary especially when data is being moved from within the same data memory module. Data may be stored in the data memory space by supplying an address, data, and the microcommand to write on the data memory space. Therefore, to transfer data from one data memory register to another, the argument is first decoded and stored in its RDR during the first clock cycle and then is used as data with a supplied address and stored in a data memory location during the second clock cycle.

Logic is provided on each data memory module to sence that its respective RDR was loaded, for example, on the present cycle. This information is then used on the next microcycle, with the appropriate microcommands, to load the respective RDR data onto one of the required address or data buses.

To address GR double precision operands, which use consecutive addresses, the first address is placed on address bus A1 and the second address (A1 data plus 1) is placed on address bus A2 during the same read cycle. (Consecutive addressed GR data memory locations are interleaved.) Therefore, each data memory GR module (and some I/O registers like auto index registers) must provide an adder for the plus one function (see figure 6).

In order to address floating point arguments (three words) a special register, A2P1, is used to store the previous contents of the A2 bus plus one for use as an address during the next microcycle. This register is not addressed in the data memory space.

Besides random access memory the data memory I/O space may address "active" memory locations. These active memory locations, when addressed, may initiate a GPP I/O function or drive the conditional bus, CB, for one microcycle thereby informing the GPP of a flip flop (flag) set. For example, the GPP MOVEB instruction uses this feature when it moves a datum into the addressed active register and branches if the CB bus is asserted. The microprogram logic saves the CB bus if asserted and will keep the CB true until the CLRCB microcommand is executed.

```
-----------------------------------------------------------------------

 ---------------                                 ---------------
 |    G-BUS     |                                |    G-BUS      |
 |  (A1 or A2)  |                                |  (D1 or D2    |
 ---------------                                 ---------------
       \|/ 16-bits                                     \|/ 16-bits
  -----------------------------------------------------------------
  |   address                                    data input |
  |               DATA MEMORY ARRAY                          |
  |                                                          |
  |                    output                                |
  -----------------------------------------------------------------
                       \|/ 16-bits
  -----------------------------------------------------------------
  |               READ DATA REGISTER, RDR                        |
  | (plus logic to sence being loaded by the present             |
  |  microcycle in order to drive the respective G bus           |
  |  line on the next microcycle.)                               |
  -----------------------------------------------------------------
                       \|/ 16-bits
  -----------------------------------------------------------------
  |      \|/       |              |              |
  |    -------     |              |              |
  |    | +1 |--->--|              |              |
 \|/   -------    \|/            \|/            \|/
 A1               A2             D1             D2
```

Figure 6: Data memory logic.
The data memory array is addressed by the A1 or A2 address buses. Data is stored or read via the D1 or D2 data buses. The present read function is used to enable the RDR to drive one of the address or data buses during the next microcycle.

2.4 ALUs

Two sets of 16 16-bit registers are used as inputs to the ALUs. These registers, ALUA(0-15) and ALUB(0-15), are read/written and addressed in the data memory I/O space. These ALU registers are duplicated on each ALU logic card as required. The microcommands can load them directly thereby freeing an address bus for multiple or double precision data fetches and stores. One of a maximum 128 ALUs is selected by the OPRalu bus (part of the G-bus). As many as 16 16-bit ALU output words may be selected for storage in the data memory space (see figure 7). Some ALUs have a conditional output which drives the conditional bus, CB, for one cycle (after the ALU is started with a SETALU microcommand). The state of the CB will be saved by the microprogram logic until a CLRCB microcommand is executed.

```
--------------------------------------------------------------------

      G BUS                                       G BUS
       \|/                                         \|/
  ----------------                            ----------------
  |  ALUA(0-15)  |                            |  ALUB(0-15)  |
  ----------------                            ----------------
         |                                          |
         -----------                       -----------
                    \|/ Input           \|/ Input
               ------------------------------------
               |                                  | Conditional
   OPRalu      |              ALUs                |  output
   select ---->|   (subset of all GPP ALUs)       |-------
               ------------------------------------      |
                          \|/ Data output               \|/
                    ---------------------                CB
                    |     AOUT (0-15)   |
                    ---------------------
                          \|/
                          G BUS
```

Figure 7: One of several ALU logic cards.
The ALU logic cards each have a maximum of 32 duplicated 16-bit input words and can produce a maximum of 16 16-bit output words. Thus, an ALU logic card could possibly operate on 32 input registers in parallel.

2.5 GPP--PDP8e interface

When the PDP8e wishes to transfer data to or from the GPP program or data memories an address transfer counter must be loaded by the PDP8e. This GPP transfer counter, GTC, is addressed in the data memory space but can only be read by the GPP. Only the PDP8e can write on the GTC. The GPP reads and writes the mapping memory and microprogram memory using programmed I/O. But the program and data memories are read/write using PDP8e DMA. Once the PDP8e DMA has been set up and initiated then a GPP input (GPPW) register and GPP output (GPPR) register are used to pass data to the PDP8e DMA channel. Because these two registers are addressed in the data memory space the microinstructions can control the data transfers. The address of the data transfers for the data memory comes from the GTC being placed onto the address bus, A1. The INCGTC microcommand is provided to increment the GTC after each transfer.

If the transfer is to take place between the PDP8e and the program memory then the GTC is first transferred to the program counter via the data bus D1. Program memory data is then transferred to the program memory write registers, PMWR(0-4) from the GPPW register and a PMW microcommand is executed which writes the PMWR(0-4) registers onto the PM. To read the PM, a PM fetch cycle is initiated by loading the PC and storing the result in the program memory register, PMR, where the results are passed to the GPPR register to be read by the PDP8e DMA channel. Both these function are fully controlled by microcommands (located in the microprogram monitor starting at microprogram address zero).

SECTION 3

Microcommand definitions

The following section lists the microcommands and their definitions. Many of the microcommands are distributed throughout the GPP via the G-bus. There are also several logic cards which some of the microcommands are routed directly to. The [ ] notation will contain the following symbols; GB for GPP G-bus, MC for microcontrol logic card, 8e for PDP8e interface card, PM for the program memory card, and MFA or MFB for the microprogram field cards A or B respectively.

A1 <group..[GB]..The following 8 commands with the prefix of A1 < are coded using 3 microcommands. The code is specified by the ( ) characters.

A1 <P1...[GB]..(0)..Load program memory P1 onto address bus A1.

A1 <P2...[GB]..(1)..Load program memory P2 onto address bus A1.

A1 <P3...[GB]..(2)..Load program memory P3 onto address bus A1.

A1 <CA1..[GB]..(3)..Load the previous addressed data as a function of address bus A1 onto the address bus A1.

A1 <ALU..[GB]..(4)..Load the selected ALU output register onto the address bus A1. Select the output register from the ALU10-3 microcommands. The contents of the OPR bus during the last SETALU microcommand determins the ALU selected.

A1 <PA2..[GB]..(5)..Load previous address bus A2 data plus 1 to the address bus A1.

A1 <MFA..[GB]..(6)..Load the microcommand field, MFA, onto the address bus A1.

A1 <MFB..[GB]..(7)..Load the microcommand field, MFB, onto the address bus A1.

A2 <group..[GB]..The following 8 commands with the prefix of A2 < are coded using 3 microcommands. The code is specified by the ( ) characters.

A2 <P1...[GB]..(0)..Load program memory P1 onto address bus A2.

A2 <P2...[GB]..(1)..Load program memory P2 onto address bus A2.

A2 <P3...[GB]..(2)..Load program memory P3 onto address bus A2.

A2 <CA2..[GB]..(3)..Load the previous addressed data as a function of address bus A2 onto the address bus A2.

A2 <ALU..[GB]..(4)..Load the selected ALU output register onto the address bus A2. Select the output register from the ALU20-3 microcommands. The contents of the OPR bus during the last SETALU microcommand determins the ALU selected.

A2 <MFA..[GB]..(5)..Load the microcommand field, MFA, onto the address bus A2.

A2 <MFB..[GB]..(6)..Load the microcommand field, MFB, onto the address bus A2.

A2 <A1P..[GB]..(7)..Load the address data now loaded on the address bus A1 plus one onto the address bus A2.

D1 <group..[GB]..The following 8 commands with the prefix of D1 < are coded using 3 microcommands. The code is specified by the ( ) characters.

D1 <P1...[GB]..(0)..Load program memory P1 onto data bus D1.

D1 <P2...[GB]..(1)..Load program memory P2 onto data bus D1.

D1 <P3...[GB]..(2)..Load program memory P3 onto data bus D1.

D1 <CA1..[GB]..(3)..Load the previous addressed data as a function of address bus A1 onto the data bus D1.

D1 <ALU..[GB]..(4)..Load the selected ALU output register onto the data bus D1. Select the output register from the ALU10-3 microcommands. The contents of the OPR bus during the last SETALU microcommand determins the ALU selected.

D1 <MFA..[GB]..(5)..Load the microcommand field, MFA, onto the data bus D1.

D1 <MFB..[GB]..(6)..Load the microcommand field, MFB, onto the data bus D1.

D1 <OPR..[GB]..(7)..Load the program memory register OPR onto the data bus D1.

D2 <group..[GB]..The following 7 commands with the prefix of D2 < are coded using 3 microcommands. The code is specified by the ( ) characters.

D2 <P1...[GB]..(0)..Load program memory P1 onto data bus D2.

D2 <P2...[GB]..(1)..Load program memory P2 onto data bus D2.

D2<P3...[GB]..(2)..Load program memory P3 onto data bus D2.

D2<CA2..[GB]..(3)..Load the previous addressed data as a function of address bus A2 onto the data bus D2.

D2<ALU..[GB]..(4)..Load the selected ALU output register onto the data bus D2. Select the output register from the ALU20-3 microcommands. The contents of the OPR bus during the last SETALU microcommand determins the ALU selected.

D2<MFA..[GB]..(5)..Load the microcommand field, MFA, onto the data bus D2.

D2<MFB..[GB]..(6)..Load the microcommand field, MFB, onto the data bus D2.

WA1D1T..[GB]..Write the contents of the data bus D1 onto the data memory space addressed by the contents of address bus A1 if the conditional bus, CB, is true.

WA1D1F..[GB]..Write the contents of the data bus D1 onto the data memory space addressed by the contents of address bus A1 if the conditional bus, CB, is false.

WA2D2T..[GB]..Write the contents of the data bus D2 onto the data memory space addressed by the contents of address bus A2 if the conditional bus, CB, is true.

WA2D2F..[GB]..Write the contents of the data bus D2 onto the data memory space addressed by the contents of address bus A2 if the conditional bus, CB, is false.

READA1..[GB]..Load the addressed data memory module read data register, RDR, with data addressed by address bus A1.

READA2..[GB]..Load the addressed data memory module read data register, RDR, with data addressed by address bus A2.

ALA<D1..[GB]..Load the selected ALU "A" register from the contents of the data bus D1. Select the "A" register from the ALU10-3 and the Program memory OPR bus for the ALU.

ALB<D1..[GB]..Load the selected ALU "B" register from the contents of the data bus D1. Select the "B" register from the ALU10-3 and the Program memory OPR bus for the ALU.

ALA<D2..[GB]..Load the selected ALU "A" register from the contents of the data bus D2. Select the "A" register from the ALU20-3 and the Program memory OPR bus for the ALU.

ALB<D2..[GB]..Load the selected ALU "B" register from the contents of the data bus D2. Select the "B" register from the ALU20-3 and the Program memory OPR bus for the ALU.

SETALU..[GB]..Start the ALU operation as a function of the operation bus And also enable the selected ALU to drive the condition bus. The conditional bus, CB, is asserter for two (2) microcycles if the condition output of the ALU is true. Select the ALU from the OPR bus.

ALU10...[GB]..Bit 0 of the ALU address and data bus, ALU1, select register.

ALU11...[GB]..Bit 1 of the ALU address and data bus, ALU1, select register.

ALU12...[GB]..Bit 2 of the ALU address and data bus, ALU1, select register.

ALU13...[GB]..Bit 3 of the ALU address and data bus, ALU1, select register.

ALU20...[GB]..Bit 0 of the ALU address and data bus, ALU2, select register.

ALU21...[GB]..Bit 1 of the ALU address and data bus, ALU2, select register.

ALU22...[GB]..Bit 2 of the ALU address and data bus, ALU2, select register.

ALU23...[GB]..Bit 3 of the ALU address and data bus, ALU2, select register.

MFA[0:15]...[MFA]..Microcommand field MFA bits 0 through 15.

MFB[0:15]...[MFB]..Microcommand field MFB bits 0 through 15.

OP<MFB..[MFB and PM]..Load the microcommand field, MFB, bits 8 through 15 onto the OPR bus bits 8 through 15. Disable the program memory OPR data from driving the OPR bus bits 8 through 15.

JMPT....[MC]..If the conditional bus, CB, is true then branch to the next microinstruction addressed by the microcommand field, MFA.

JMPF....[MC]..If the conditional bus, CB, is false then branch to the next microinstruction addressed by the microcommand field, MFA.

PUSHJT..[MC]..If the conditional bus, CB, is true then branch to the next microinstruction addressed by the microcommand field and load the microprogram address plus one in the microprogram push down stack.

PUSHJF..[MC]..If the conditional bus, CB, is false then branch to the next microinstruction addressed by the microprogram data register and load the microprogram address plus one in the microprogram push down stack.

POPJT...[MC]..If the conditional bus, CB, is true then branch to the next microinstruction addressed by the microprogram push down Stack.

POPJF...[MC]..If the conditional bus, CB, is false then branch to the next microinstruction addressed by the microprogram push down Stack.

MPCPMT..[MC]..If the conditional bus, CB, is true then load the microprogram starting address from the PM via the mapping memory into the microprogram address register. The microinstruction addressed by the MM datum will be used after the next microcycle is executed. If the PM fetch is not complete then the microcycle is stalled by the PM interface.

MPCPMF..[MC]..If the conditional bus, CB, is false then load the microprogram starting address from the PM via the mapping memory into the microprogram address register. The microinstruction addressed by the MM datum will be used after the next microcycle is executed. If the PM fetch is not complete then the microcycle is stalled by the PM interface.

MPCPBT..[MC]..If the conditional bus, CB, is true then load the microprogram starting address from the PM buffer register via the mapping memory into the microprogram address register. The microinstruction addressed by the MM datum will be used after the next microcycle is executed.

MPCPBF..[MC]..If the conditional bus, CB, is false then load the microprogram starting address from the PM buffer register via the mapping memory into the microprogram address register. The microinstruction addressed by the MM datum will be used after the next microcycle is executed.

MPC<D1..[MC]..Load the microprogram starting address from the mapping memory addressed by data bus, D1, into the microprogram address register. This microcommand will add approximately 100 nsecond to the microcycle.

INCPDL..[MC]..Add one to the push down list address counter.

DECPDL..[MC]..Subtract one from the push down list address counter.

PC<D1T..[MC]..Load the contents of the data bus D1 into the program counter if the conditional bus, CB, is true. Also update the program field register, PFR, with last stored data by a GPP instruction. When the PC is loaded a program memory fetch sequence is executed.

PC<D1F..[MC]..Load the contents of the data bus D1 into the program counter if the conditional bus, CB, is false. Also update the program field register, PFR, with last stored data by a GPP instruction. When the PC is loaded a program memory fetch sequence is executed.

INCPCT..[MC]..If the conditional bus, CB, is true then increment the program counter. When the PC is incremented a program memory fetch sequence is executed.

INCPCF..[MC]..If the conditional bus, CB, is false then increment the program counter. When the PC is incremented a program memory fetch sequence is executed.

PCBRPC..[MC]..Load the program counter buffer register with the program counter data plus 1.

PCPCBT..[MC]..If the CB is true, load the contents of the program counter buffer register, PCBR, into the PC and start a PM fetch sequence.

PCPCBF..[MC]..If the CB is false, load the contents of the program counter buffer register, PCBR, into the PC and start a PM fetch sequence.

PDL<PC..[MC]..Load the program counter, PC, and the program field register, PFR, into the push down list addressed by the push down list counter, PDLCTR.

PC<PDL..[MC]..Load the PC and the PFR from the PDL addressed by the PDLCTR. When the PC is loaded a program memory fetch sequence is executed.

LPFR....[MC]..Load the program field register, PFR, with the contents of the program field buffer register, PFBR. When the PFR is loaded, A PM fetch sequence is executed.

PMRPMT..[PM]..If the conditional bus, CB, is true then save the program memory data locations addressed by the program counter into the program memory output registers. If the PM fetch is not complete then the microcycle is stalled by the PM interface.

PMRPMF..[PM]..If the conditional bus, CB, is false then save the program memory data locations addressed by the program counter into the program memory output registers. If the PM fetch or write (see microcommand PMW) is not complete then the microcycle is stalled by the PM interface.

PMRPBT..[PM]..If the conditional bus, CB, is true then load the program memory buffer register, PMBR, into the program memory registers, PMR.

PMRPBF..[PM]..If the conditional bus, CB, is false then load the program memory buffer register, PMBR, into the program memory registers, PMR.

PMBRPM..[PM]..Save the program memory data locations addressed by the program counter into the program memory buffer register. If the PM fetch is not complete then the microcycle is stalled by the PM interface.

PMW.....[PM]..Write the program memory write registers, PMW0-4, onto the program memory. Test when done by executing the PMRPMF which will stall the microcycle until the write operation is complete.

INCGTC..[8E]..Increment the GPP transfer counter.

CLRCB...[MC]..Clear the conditional bus, CB.

SECTION 4

Microprogramming examples

The following section will introduce examples of microprogramming GPP instructions. The purpose of these examples is to aquaint the reader with the microcommands and there use. The reader is advised to study the NCI/IP Technical Report #16, the GPPASM--A PDP8e Assembler For The General Picture Processor [Lem76b].

In writing microprograms, all comments are enclosed in quotation marks ("). All microcommands embedded between "/...\" are executed during the same clock cycle. All lables are terminated by a colon (:). The microcommands MFA[0:15] and MFB[0:15] are represented by [MFA] and (MFB) respectively. For example, if a [243] simbol is used it represents the value of 243 in the MFA[0:15] microcommand field.

4.1 Example 1

The first example executes a GPP instruction when the GPP RUN-HALT flip flop is set to halt. The PDP8e first loads the program counter, PC, and the program field buffer register, PFBR, with the desired starting address of the GPP instruction in the program memory. The PDP8e then sets the GPP RUN-HALT flip flop to RUN. The microprogram must sense that the RUN condition exists and execute the GPP instruction addressed by the PC. In order to do this a monitor microprogram starting at microprogram address zero is executed. The PDP8e has complete control of starting the microprogram monitor at address zero via the GMHLT (GPP microprogram halt) and GPPCLR (GPP clear) PDP8e opcodes. The following microinstructions form the microprogram for example one.

```
        MPARORIGIN 0
"The assembler will set the origin to zero."
/MONITOR:        A1<MFA
                 READA1
                 [RHFF]\
```

This microinstruction addresses the GPP RUN-HALT flip flop. If the run state is on, then the condition bus, CB, will be asserted until a CLRCB microcommand is executed.

```
/EX1L2:          JMPT
                 [GO]\
```

If the RUN-HALT flip flop was set to RUN then the CB bus will be asserted and the next microinstruction will start from the label "GO". If the CB was false then the microinstruction which follows is executed.

```
/EX1L3:          JMPF
                 [MONITOR]\
```

If the CB is false then the next microinstruction will come from the origin labeled MONITOR. This loop, then, will keep testing for the RUN-HALT flip flop.

```
/GO:             LPFR
                 CLRCB\
```

Once the RUN-HALT flip flop was set to RUN the microprogram jumped to the label GO. Here the program field register, PFR, is loaded by the program field buffer register which was inturn loaded by the PDP8e. A PM fetch sequence is initiated and the CB is cleared.

```
/EX1L4:          MPCPMF
                 PMBRPM
                 INCPCF\
```

The microcycle will stall until the previous PM fetch is complete. Then, the microprogram counter is loaded with the mapping memory data addressed by the PM OPRgroup. The program memory data is saved in the program memory buffer register, PMBR. The program counter is incremented and a new PM fetch is initiated.

```
/EX1L5:          PMRPBF\
```

The saved program memory data in the PMBR is loaded into the PMR to be used during the next microprogram. The next microprogram executes the GPP instruction addressed by the program counter which the PDP8e loaded.

4.2 Example 2

The second example is the microprogram of the GPP instruction;

```
MOVE p1,,p3.
```

The MOVE instruction is a transfer of the datum in the data space location, p1, to the data space location, p3. The notation of no # or ' means normal addressing. It may be helpful to think of this GPP instruction as the one being addressed by the PC via the PDP8e in example 1. Therefore, the next microinstruction is the result of the microcommand, MPCPMF, in example 1.

The first microinstruction of the MOVE instruction is as follows.

```
/MOVENORMAL:     A1<P1
                 READA1
                 MPCPMF
                 PMBRPM
                 INCPCF\
```

The datum addressed by P1 is read and stored in its read data register, RDR, for use in the next microinstruction. Because the microprogram for this MOVE instruction is only two microinstructions long, the MPCPMF microcommand is executed. This alows the fetching of the next GPP instruction microprogram during the completion of the MOVE microprogram. The MPCPMF microcommand will cause the microcycle to stall if the previous program memory fetch was not complete. The program memory data is stored in the program memory buffer register, PMBR, so that the program counter may be incremented and a new PM fetch started.

```
/EX2L2:          D1<CA1
                 A1<P3
                 WA1D1F
                 PMRPBF\
```

The data stored in the RDR in the last microinstruction is placed onto the D1 bus by D1<CA1. Also the address of the sink location, P3, is placed onto the address bus, A1, by A1<P3. A write is executed resulting in the data on the D1 bus being stored at the location in the data memory space addressed by the A1 bus. The program memory register is loaded with the saved program memory data for the next microprogram from the program memory buffer register, PMBR.

4.3 Example 3

The third example is the microprogram of the GPP instruction;

```
        ADD p1,p2,p3.
```

This GPP instruction will add the two arguments addressed by p1 and p2. The result will be deposited into the location addressed by p3. This GPP instruction can be thought of as following the last MOVE instruction of example 2. Therefore, the next microinstruction is the result of the microcommand, MPCPMF, in example 1.

```
/ADD             A1<P1
                 READA1\
```

The data addressed by the contents of P1 is loaded into its data read register, RDR.

```
/EX3L2:            A2<P2
                   READA2
                   D1<CA1
                   ALA<D1\
```

The data addressed by the contents of P2 is loaded into its data read register, RDR. The data previously loaded into P1's RDR is loaded into the ALUA register zero. The ALUA(0) was selected by default. Since none of the ALU1 0-3 microcommands are present they are considered to be zero which selects ALUA(0) register in the ALA<D1 microinstruction.

```
/EX3L3:            D2<CA2
                   ALB<D2
                   SETALU
                   MPCPMF
                   PMBRPM
                   INCPCF\
```

The data addressed by the contents of P2 is loaded into the ALU register ALUB(0). The SETALU microcommand starts the ALU operation selected by the OPRalu bus which contains the OPR[9:15]. The ADD alu will be selected by this bus. The MPCPMF microcommand is executed to set up for the next GPP instruction. The contents of the PM used for the next microprogram is stored in the program memory buffer register, PMBR, so the PC can be incremented and another PM fetch initiated.

```
/EX3L4:            A1<P3
                   D1<ALU
                   WA1D1F
                   PMRPBF\
```

The result of of the ALU add is placed onto the D1 bus and stored at address selected by the contents of P3. The next program memory data is stored in the program memory register, PMR.

4.4 Example 4

The next example is the microprogram of the GPP instruction;

```
DAND p1,p2,p3.
```

This double precision GPP instruction will compute the bit AND between the two double precision arguments addressed by p1 and p2. The result will be deposited into the location addressed by p3. All double precision and floating point arguments are in the GR data space only. Because the GR is interleaved the microinstructions can address both the high and low order words of the double precision argument in parallel.

```
/DAND:          A1<P1
                READA1
                A2<A1P
                READA2\
```

The data addressed by the contents of P1 and P1+1 is loaded into their respective data read registers.

```
/EX4L2:         D1<CA1
                ALA<D1
                D2<CA2
                ALA<D2
                ALU23
                A1<P2
                READA1
                A2<A1P
                READA2\
```

The data read registers' data is placed on the D1 and D2 buses to be stored in the ALUA(0) and ALUA(1) ALU registers respectively. The data addressed by the contents of P2 and P2+1 is loaded into their respective data read registers.

```
/EX4L3:         D1<CA1
                ALB<D1
                D2<CA2
                ALB<D2
                ALU23
                SETALU
                MPCPMF
                PMBRPM
                INCPCF\
```

The data read registers' data addressed by P2 and P2+1 is stored into the ALUB(0) and ALUB(1) registers respectively. The microcommand, SETALU, starts the double precision AND (supplied by PMRalu[0:7]) and the MPCPMF microcommand loads the microprogram counter with the starting address of the next microprogram to be executed after the next microinstruction. The contents of the PM used for the next microprogram is stored in the program memory buffer register, PMBR, so the PC can be incremented and another PM fetch initiated.

```
/EX4L4:         A1<P3
                D1<ALU
                WA1D1F
                A2<A1P
                D2<ALU
                ALU23
                WA2D2F
                PMRPBF\
```

The result of the ALU function, AOUT(0) and AOUT(1), are placed onto the data buses D1 and D2 respectively. The addresses of the sink locations for this data is placed on the address buses, A1 and A2, and the write function is executed. The next program memory data is stored in the program memory register, PMR.

4.5 Example 5

The next example is a GPP contional branch instruction;

```
GTB p1,p2,p3.
```

The GTB instruction takes the arguments, p1 and p2, and places them in the ALUA(0) and ALUB(0) registers respectively. The greater-than alu is used: if p1 is greater than p2 then the program counter will be loaded with the contents of P3 causing a GPP instruction branch. Otherwise the next GPP instruction is executed.

```
/GTBNORMAL:     A1<P1
                READA1\
```

The CB is cleared and the argument addressed by the contents of P1 is stored in its RDR.

```
/EX5L2:         A2<P2
                READA2
                D1<CA1
                ALA<D1
                PCBRPC
                PMBRPM
                D1<P3
                PC<D1F\
```

The argument addressed by the contents of P2 is stored in its RDR. Also the last argument addressed by P1 is stored into ALUA(0) register. The program counter data plus 1 is stored into the program counter buffer register, PCBR, and a PM fetch of the branch PM target data is initiated.

```
/EX5L3:         D2<CA2
                ALB<D2
                SETALU
                MPCPBF\
```

The ALUB(0) register is loaded with the argument addressed by P2 and the ALU function is started. The microprogram counter is loaded with the data in the mapping memory addressed by the saved PM OPRgroup in the program memory buffer register. This microcommand is executed to start the next microprogram microinstruction fetch in case the ALU obtains a false conditional output.

```
/EX5L4:         PMRPBF
                PCPCBF
                PMRPMT
                MPCPMT
                INCPCT
                CLRCB\
```

If the CB is false then the PM data stored in the PMBR is loaded into the PMR and the program counter is restored for the next sequential GPP instruction. But, if the CB is true then the branch target GPP instruction will be executed. In order to do this the PMR is loaded with the PM data addressed by the branch address in the PC. The microprogram address for this instruction is loaded into the microprogram counter and the next microinstruction is executed from that address. The PC is also incremented to initiate the next PM fetch. Finally the CB bus is cleared.

4.6 Example 6

The following microprogram example is the GPP push jump instruction;

```
PUSHJ ,,p3.
```

The PUSHJ instruction stores the present value of the program counter, PC, and the program field register, PFR, in the push down list, PDL. It also increments the push down list address counter, PDLCTR, and loads the PC with the argument from P3.

```
/PUSHJNORMAL:   PDL<PC
                D1<P3
                PC<D1F
                LPFR\
```

First the present value of the PC and PFR are stored into the PDL addressed by the PDLCTR. In the same microinstruction the new contents of the PC is loaded from the contents of P3. The program field register, PFR, is loaded from the program field buffer register, PFBR. If the PFBR was previously loaded with a MOVE instruction (or any other GPP instruction) then the new program field will be used for the execution of the next instruction. The loading of the PC and PFR initiates a PM fetch sequence.

```
/EX6L2:         MPCMMF
                PMRPMF
                INCPCF\
```

The microcycle will stall until the PM fetch is complete. Then, the PM fetched data will be saved in the PMR and the PC is incremented which starts another PM fetch cycle. The microcommand, MPCMMF, is executed to load the microprogram counter with the address of the next microprogram to be executed after the next microinstruction.

/EX6L3: INCPDL\
The push down list address counter, PDLCTR, is incremented so that during another PUSHJ instruction the counter would address the next location in the PDL.