Floating Point Vector Processor for 3D Graphics Applications

AVI BLEIWEISS, HP Labs

ACM Reference Format:
Avi Bleiweiss. 1988. Floating Point Vector Processor for 3D Graphics Applications. 1, 1 (January 1988), 14 pages. https://doi.org/10.475/123_4

1 OVERVIEW

The VP is the frontend component of the Mirage architecture, a workstation (HPPA) based, high performance 3D graphics accelerator. Mirage was primarily designed with the goal of fast, high quality rendering of a large variety of primitives in complex environments. As such, the VP architectural goal was to exploit vectorization, pipelining, and parallelism to accomplish acceleration of both real-time and high-level object space algorithms. Transformation, tessellation, clipping, ray-primitive intersection, and light modeling are some of the algorithms involved. The nature of these algorithms is heavily compute bound with a high demand for floating point operations/sec (FLOPs).

The architecture addresses efficient kernel execution by a direct support of vector/matrix, scalar, and bit operations in parallel. Local data store is physically partitioned into primitive, global, and temporary high speed memory arrays, featuring a flexible multi-port and a large register file. The pipe nature of the design allows for programmable feedback data routes of pipe sections, thus minimizing fill and flush latencies. Data move, processing, and flow control are performed by a fully orthogonal VLIW (very large instruction word) microcode subsystem.

The generality of the architecture lends itself to a wider class of application areas, such as modeling (supporting double precision math), image processing, and linear algebra. A multiple VP system has a minimal switch overhead and is attractive for solving problems which can be partitioned into one 'processor per object'.

Finally, the architecture is scalable and flexible to address a variety of price/performance levels for different market needs.

The scope of the following document covers detailed descriptions of system design considerations, VP subsystems, performance benchmark figures, and design tools. Three block diagrams of the Mirage (Figure 1), VP (Figure 2), and the VP Logic unit (Figure 3) are attached as a supplement to this report and are recommended as a reference while reading the text.

2 ARCHITECTURE

The VP communicates with the host via a bi-directional mirrored system bus, supporting array (dma) and vector (memory) i/o transaction protocols for efficient display list traversing and editing, respectively. Minimal system i/o overhead is maintained by means of double buffered primitive store, simplified host/microcode handshake, combined primitive command/data block transfer, and primitive bundle handling. Full synchronization is maintained across the board by using a single system clock - the system bus master clock.

VP local store is mapped to the host memory address space, providing a mixture of virtual and physical addressing schemes.

Author's address: Avi Bleiweiss, HP Labs, Palo Alto, California.

1988. Manuscript submitted to ACM

Multi-mode context-switch implementation for multi applications executed in a windowing environment, results in a relatively negligible system performance decline. As the status log can be stored on an application/object basis and may be traced for further post exception handling. Processed image data (integer) is transferred to a subsequent graphics pipe stage (the Mirage VIP) via the 'vertex bus', using a fixed communication packet approach.

The VP is composed of three basic subsystem components: store, execution, and microcode. The components are tightly coupled to each other via a 256 bits main data path, allowing for a data move of two four element vectors (hereafter a vector) at any given clock cycle. The architecture is optimized for vector operations that support the fundamental 3D graphics homogeneous coordinate system. As such, dot products, matrix multiplications, radix-4 butterfly, and vector updates are atomic operations of the VP and can be vectorized easily.

The fully orthogonal microcode system simplifies control management, and provides modularity and ease of programming. The VLIW system exploits true parallelism and improves performance significantly. However, it is still a great challenge to have an efficient high-level language compiler, particularly one that handles diverse piping and delayed branches.

Each microcode instruction is performed within·a system clock period, or microcycle. Half system clock operations are referred as a nanocycle.

Exception detection (IEEE) and handling is directly supported by hardware and microcode.

3 STORE

The basic guide lines for the local store design were:

- *Size* - to be derived from factors such as scene complexity (up to 1024 objects/environment, shared among applications), primitive size (1024 clipped vertices/polygon), primitive bundle (up to 64 quadrilateral polygons), patch subdivision level (up to 10 levels), ray bundle (1024 rays), FFT (1024 complex points, not-in-place), and linear equations (100 x 100, double precision).
- *Flexibility* - independent control of primitive and global data is extremely important.
- *Efficiency* - enough work area, vector accessibility and ease of memory management.

Local store is therefore composed of vector and scalar memory groups, implemented in high speed SRAMs for double-buffered primitive data-vector memory (DVM), global coefficient vector memory (CVM), immediate/temporary scalar memory (SM), and tag compare code memory (CCM).

Store partitioning thereof implies a general modular convention for various data type allocations. The notion of primitive reflects the lowest level of a problem breakdown i.e. line, polygon, patch, image slice, linear equation array, etc. Global stands for a matrix stack, light coefficients, twiddle factors, ray parameters, etc. Temporary work area is used to store intermediate results of not-in-place processes such as intermediate primitives in a reconstruction process, fft staging data, etc. A tag field utility is a control extension of primitive data types and is useful for intersection acceleration (clip code), subdivision level trace, and resulting attribute flags (such as front/back facing, hemi-cube face number, and a shadow polygon).

3.1 DVM

The double buffer scheme of the DVM provides a framework for concurrent 'current primitive' processing and 'next primitive' host i/o (dma). Each buffer is of four planes (x, y, z, w), as each plane size is 8Kx32 bits comprised of four 8Kx8

memory chips of 35nsec access. The utility of a bi-directional frontend pipe stage, for each plane, reflects a pipe-section data feedback that speeds up vector memory updates.

Overall, pipe data flow is maintained at the DVM node of the main data path, by performing memory read and write each nanocycle.

Two programmable address generators (AD1410), for both reads and writes, supply the address (16 bits) for the 'current primitive' buffer. An address generator executes a rich instruction set and with the core facility of multi address registers and an ALU, it generates offsets, circular buffering checks, and bit reversing. The 'next primitive' buffer is addressed by a dedicated dma address bus that is fed by the system interface unit, thus relieving microcode of host load/store control.

3.2 CVM

CVM is identical in structure to a DVM buffer with the exception that in any given microcycle, a memory read or write may take place. A single identical address generator serves as the address source for the memory either in processing or in host load/store modes. Also, the address generator provides addresses for loading/storing the LUT (described in 4.2.2).

3.3 SM

SM is a single plane memory, yet with the same four plane, bi-directional pipe register array, as described in section 3.1. SM behavior is identical to the CVM and the single address generator serves the CCM (described in section 3.4), as well.

CVM and SM are considered a single node of the main data path, where read or write from/to CVM or SM may apply at any microcycle.

3.4 CCM

The CCM is a dedicated memory of 8Kx8 bits, tightly coupled to the bit logic unit (described in section 5.2), which provides an eight bit field as a tag extension to primitive data.

4 EXECUTION

Dot product, scalar and logic units compose the VP execution unit. With the exception of the logic unit, the execution unit operates on either single or double precision variables, floating or fixed point. All units interlink to the local store memory groups through the main data path. Efficient overall data flow is maintained by overlapping pipe setup of one unit with the pipe closeup of another.

The dot product is a triple, 128 bits port component, that facilitates dot product of two vector operands (DVM and either CVM or SM) as a core operation. This basic operation is extended naturally to a vector/matrix and matrix/ matrix multiplication. In a more general form, the dot product architecture has the growth path potential of performing all intra-plane operations in parallel, i.e. concurrent execution on x, y, z, and w (described further in section 4.1).

Inter-plane (inter vector element) operations are performed in the scalar unit. The unit is closely looped to address recursion, sum of dot products and the product of sum. Exact divide and square-root operations are supported in a single microcode instruction, and a look-up table (LUT) provides rapid non exact square roots, power, and trigonometric functions.

The logic unit is functionally divided into minmax and bit-logic blocks. Minmax serves as a floating point data sorter, that derives minimum and maximum values of a vector. A 4x4 register/mux array adds on functions such as matrix

transpose and inter-vector compare, highly useful for computing primitive bounding boxes. Compare bit extraction is performed to generate a clip code for efficient geometry clip-test and intersection computations.

Two parallel bit operators, 8 bit wide, perform logic and shift operations on compare codes (clipping), or possibly on flag bits for rendering mode detection.

Control flow is activated by arithmetic and exception status lines, originating at any component of the execution unit. These status lines are sensed at the branch/micro-sequencer logic (described in section 6.1).

4.1 Dot Product

A sum of four multiplications is implemented by seven floating point processors (4x MUL AD3211/2, 3x ALU - AD3221/2). Peak performance of any of the processors is 20MFLOPs for either single or double precision add/subtract or multiply. Each processor is a triple port device with two or four deep register files at its input ports. With two pipe stages at each processor, overall dot product pipe setup takes nine microcycles.

Sign and zero status of the top stage are sensed in the dot product arithmetic branch instructions. A zero filled buffer and associated control logic are used as an on-the-fly less-than-zero test without the need to execute a branch instruction (highly useful in intensity computations).

Exception detection is embedded in an 1800 gate PLD, which latches any of the following possible occurrences including overflow, underflow, invalid operation, inexact, and denormal. The latched data may be stored in the system interface register file for the subsequent exception handling phase, either managed in microcode or by the application.

The dot product structure has a potential future evolutionary path that can lead to a major system price/performance improvement. A modification for generalizing the architecture calls for replacing the frontend multiplier array with a multiplier/accumulator (MAC) array and by having each of the MAC outputs feeding back a DVM buffer plane (i.e. multiplexed with the dot product output).

4.2 Scalar

The main data path is piped by a register/mux array, DMUX, in front of the scalar unit. DMUX control logic selects any two elements of a vector, in any order, each in a nanocycle, and feeds them as scalar or logic unit operands. This flexibility is very important for unordered vector operations such as cross product and subdivision.

Two floating point processors (ALU - AD3221/2 and a MUL - AD3211/2), serving as a MAC, and a lookup table compose the execution part of the scalar unit. The outputs of the unit are fed back internally at a component level, or externally, through the DMUX, with no need to update memory.

4.2.1 MAC. The MAC performs all single cycle scalar operations- add/subtract, multiply, compare and format conversion - as well as multi-cycle exact divide and square root.

Two sign, one for each the ALU and MUL, and ALU zero status lines are tested in the scalar arithmetic branch instructions. Exception signals are transferred to the branch logic and special logic provide denormal handling in hardware.

4.2.2 LUT. A 32Kx32 bit memory array comprised of four 32Kx8 memory chips of 55nsec access time, serves as a lookup table (LUT).

Two address modes are implemented: linear and tiled. In linear mode the full LUT is addressed linearly as being a single entity. In the second mode, the LUT is sliced into 1Kx32 pieces and a tile register provides for tile selection. The

1	2	3	4	5	6
$x > w$	$x < -w$	$y > w$	$y < -w$	$z > w$	$x < 0$

Table 1. Six bit compare code corresponding to each of the clipping planes.

tiling mode is used to implement an indexed power function, which is required in the specular intensity computation. Data stored in the LUT is presented in either single floating point or fixed point.

5 LOGIC

The logic unit is functionally divided into two blocks: the minmax and the bit logic.

5.1 Minmax

The minmax block has four modes of operation, namely clip, intra-vector; inter-vector, and transpose. In the clip mode the minmax generates a clip code for an input data vector. A clip code is a 6 bit compare code extraction as defined in Table 1. In general, 8 bit code can be extracted if necessary.

In the intra-vector mode, the minmax finds the minimum and the maximum values of the components of the same vector. This operation is useful for some general purpose routines that operate on vector data, such as pivoting and sorting. Compare bits extraction provides the element index information of the minimum and maximum values in the vector.

In the inter-vector mode, the block compares the like components of four different vectors and finds the maximum and the minimum values of those components. This information is pertinent to accelerate computation of primitive bounding boxes for performing intersection of a ray with B-splines, using a divide-and-conquer mechanism.

Finally, the transpose mode is used for 4x4 matrix transpose. In this mode the unit receives the input vectors in a 'row by row' fashion and outputs the data in a 'column by column' order. No comparisons between vector components are performed in this mode.

The minmax block operates in a pipelined manner. It takes two microcycles to load a vector and correspondingly, two microcycles to produce the minimum/maximum values or a clip/compare code. The input stage of the rninrnax is a. 4x4 32-bit register array.

This array is used to store the new input vectors while the previous vectors are being processed, and also to translate the input data received in a 'vector by vector' manner into a 'column by column' output required for the inter-vector and the transpose modes. The outputs of the register file are four vector components which are applied to the array of six comparators. Simultaneously, the register array outputs are applied to four output registers each of which stores one vector component.

The six comparators compare all possible pairs of vector components in parallel. The outputs of the comparators are applied to control logic which uses those outputs to compute a clip or compare code, as well as to determine the minimum and the maximum components. The control logic then selects the appropriate output register, as determined by the microcode, and the contents of that register is placed on the main data path. The computed clip/compare code is sent to the bit logic unit and to the CCM.

5.2 Bit Logic

The bit logic block is primarily used to perform various logic operations on compare/clip codes or any byte wide data. Clip code generation significantly accelerates recurring functions such as primitive trivial accept/reject tests and in-out intersection checks.

The input stage of the bit logic unit has three register files: A (4x8 bits), B (4x8 bits), and M (8x8 bits). The register files A and B are used to store bit oriented data (such as clip/compare code, mode), and the M registers store immediate values (such as masks). Processing of data is performed in the logic operators 1 and 2 (Figure 3) concurrently. Each of the operators performs the following logic functions, namely XOR, AND, OR, or bypassing data to the output without change. The outputs of the logic operators are applied to a barrel shifter, which is capable of shifting data to the left or to the right by 1, 2, or 4 bit positions. In addition, each logic operator generates a status signal which indicates if the result of a logic operation is an all-zero quantity.

The bit logic also includes a test register which can be loaded and read from the host. Each operation of the bit logic, including shifting, takes one microcycle.

6 MICROCODE

System control is governed by the microcode subsystem comprised of a micro-sequencer, branch logic, and a piped 296 bit writable control store, WCS. The microcode word is divided into seven control fields and provides for a fully orthogonal control system.

A mode register (8 bits) provides control lines that do not change every microcycle. Among them are rounding, fast/IEEE math, and LUT address modes. Single or double precision is controlled either by the mode register or by microcode, on a microcycle basis.

6.1 Control Flow

The micro-sequencer (AD1401) supplies the address to the WCS and conduct control flow by sensing external flag and interrupt (4) lines. It has a 64 word (16 bits) deep stack memory, which is controlled by subroutine and global stack pointers. Three event counters are utilized for scheduling purposes.

The micro-sequencer data port is connected to the system control bus (16 bits) that links address generators, micro-sequencer, logic unit, and WCS immediate field (described in section 6.2). Host load/store data are provided on the system control bus to its residents. Dynamic stack management is facilitated by means of a bi-directional link, connecting the DVM 'w' plane to the system control bus.

Host/microcode handshake is conducted by the microcode 'busy' and 'host ready' signals. Active 'host ready' basically informs the microcode system that the next primitive data block is ready for process. Microcode 'busy' informs the system interface that the system is currently in the processing phase. Inactive microcode 'busy' allows the system interface to arbitrate for the micro-sequencer instruction port, primarily for WCS load/store transactions and state save under a context switch.

Arithmetic status lines and the 'host ready' signal are multiplexed for generating the flag input to the micro-sequencer. System wise, the assumption was that arithmetic checks and host handshakes may be performed frequently. Therefore the micro-sequencer flag, as opposed to interrupts, that has a single microcycle taken branch latency was selected for this purpose.

Flag is the output of two multiplexing levels, namely group and system. Three group mux's of the logic, scalar and dot product units receive arithmetic and exception status lines, the latter are the result of floating point compare operation. Every group mux output as well as 'host ready' reaches the system status mux that outputs the global flag signal.

Interrupts have a minimum of four microcycle latency and are handled as exceptions. Four interrupt lines were implemented for the following events:

- *Context switch*: originated at the system interface.
- *Divide by zero*: scalar MAC divide.
- *Denormal*: scalar MAC floating point multiply.
- *VIP acknowledge*: means of arbitration in a multi VP system.

A dedicated 1800 gate PLD stores the exception signals that are sourced at both the dot product and scalar units, and delivers a fourteen bit status word to the system interface. The status word is divided into three exception fields, namely dot product, scalar ALU, and scalar MUL. A 'device error' signal reflects the overall system exception status.

The system pipe may be frozen at any given microcycle by disabling the system clock source to any of the execution unit components, namely dot product, scalar, and logic, individually. A side benefit of the freeze feature is the ability to control system power dynamically by leveraging CMOS active components to increase reliability.

6.2 WCS

The WCS size is 32Kx296 bits and is composed of thirty seven 32Kx8 memory chips of 55nsec access. A microcode word in a multiport pipe register is used as a sync point to compensate for micro-sequencer address delay and processor setups. The pipe register serves also as a serial host link to load/store WCS data and supplies a microcode word upon power up initialization.

The seven control fields of the microcode word are assigned as follows:

- *micro-sequencer* (bits 0 - 7): busy and sequencer.
- *immediate* (bits 8 - 23): pointer data for static stack management.
- *branch* (bits 24 - 39): arithmetic and exception branch.
- *store* (bits 40 - 119): address generators, read/write memories, data direction.
- *dot product* (bits 120 - 199): multiplier-array ALU stages.
- *scalar* (bits 200 - 255): DMUX operand select and MAC.
- *logic unit* (bits 256 - 295): minmax and bit logic.

6.3 Instruction Set

The VP instruction set defines the setting of the 296 microcode control bits in a microcode word. A microcode word consists of one or more VP instructions, which implement the pipelining and parallelism of the machine. Eighty percent of the instruction set has been defined and assembled by the software microcode tool. Performance estimates for graphics applications were obtained by microcoding the algorithms with the VP instruction set.

The instruction set can be partitioned into the following categories, namely data movement, execution, address generation, and control flow. Next, a brief description of the instruction types provides the functionality and syntax of the VP instruction set, as In Table 2, we list our instruction microword terminology.

The 'mov' instruction is for data movement that controls the diverse flow of data in the VP. The 'mov' operands are source and destination, each specified by the name of the VP component, such as 'dvm()' for DVM and 'v_mul()' for

Notation	Description
useq()	micro-sequencer
adr2_dvm()	read address generator
dvm()	DVM
dvm_regs	DVM pipe registers
cc_reg	minmax compare code register
ccm_reg	CCM pipe registers
v_mul()	dot product vector multipliers
v_alu12()	dot product vector adders, first stage
v_alu3()	dot procluct vector adder, top stage
bit()	bit logic unit

Table 2. Microword instruction terminology.

the dot product vector multiplier array. Parameters may appear in parenthesis following the component name thus referring to a processor register, memory plane, etc. The following two examples clarify this description:

(1) mov dvm(xyzw), dvm_regs
(2) mov dvm_regs, v_mul(a0)

The first instruction moves a data vector from DVM at a specified address to the DVM pipe registers. The second instruction moves the data from the former pipe registers to the input port register, a0, of the dot product vector multipliers.

The syntax of the instructions for execution is 'unit_name(action)', where 'unit_name' is the execution unit component and 'action' is the instruction for the unit to perform. For example:

s_mul(a0*b1)

which multiplies the contents of input port registers a0 and b1 in the scalar MUL. The action instruction executes the AD3211/12/21/22 instruction set and those specifically defined for the logic unit.

Address generation instructions follow the same format as the ones for execution: 'unit.name(action)'. 'unit.name' is one of the four address generators and the action specified is encoded to represent the instruction set of the AD1410. The AD1410 contains instructions for incrementing and decrementing with offsets, plus internal register transfers and internal control word settings.

The microcode control flow includes instructions for the micro-sequencer and the branch checks. Similarly, the micro-sequencer actions are specified by the instruction set of the AD1401. The AD1401 instructions include address increment, jump and branch, and the control of the internal stack, status register, counter, and interrupts.

The sequence of instructions we show in Table 3 are part of the polygon vertex transformation procedure. A list of vertices is transformed by a transformation matrix M (4x4). In addition, a clip code is calculated for each vertex in parallel with transformation. These instructions are being executed in a single microcycle and illustrate how the dot product is programmed when fully utilizing the hardware pipeline. It also shows that logic operations are performed in parallel with the transformation.

Instruction	Description
useq(c0−)	decrement register c0 in micro-sequencer
adr2_dvm(r0 = +b0)	output transformed vertex list address and increment with offset b0
mov dvm_regs, v_mul(a3)	vertex<x,y,z,w> → register a3 in vmuls
mov dvm_regs, dvm(z)	z → DVM plane z
mov vmul(), v_alu12(al,bl)	$x * M10$ → a1 register in valu1 $y * M11$ → b1 register in valu1 $z * M12$ → a1 register in valu2 $w * M12$ → b1 register in valu2
mov v_alu12(), v_alu3(a2,b2)	$x * M20 + y * M21$ → a2 register in valu3 $z * M22 + w * M23$ → b2 register in valu3
mov v_alu3(), dvm_regs	dot product → DVM pipe registers
v_mul(a1*b3)	$x * M30$, $y * M31$, $z * M32$, $w * M33$
v_alu12(a0+b0)	$(x * M00) + (y * M01)$, $(z * M02) + (w * M03)$
v_alu3(a1+b1)	$((x * M10) + (y * M11)) + ((z * M12) + (w * M13))$
mov cc_reg, bit(a0,b0)	clip code → bit logic registers a0,b0
mov cc_reg, ccm_reg	clip code → CCM pipe registers
bit(pass acc)	pass accumulator value

Table 3. Vertex transformation microcode sequence. All instructions execute in a single microcycle.

7 PERFORMANCE

System benchmarks were performed for the entire Mirage architecture. The goal was to evaluate the right balance among system components in terms of delivery and process power, and to provide overall system performance figures. Application areas that were addressed include 3D graphics for real-time, refinement, and high-level algorithms (further detailed in the next paragraph), linear algebra, and image processing. The performance benchmarks we discuss are those where the VP was the critical path.

Performance benchmarks were extended beyond the common scope of simple real-time 3D graphics algorithms. Our real-time figures for general line and polygon rendering (single directional light-source) are shown in Table 4. In addition, rendering specular environments with multiple light sources (Table 5), while exploiting dot product vectorization, and patch tessellation, a pivotal B-spline operation (Table 6) were of main interest. Algorithms such as anti-aliasing, texture mapping, shadowing, and phong shading belong to the refinement class. This class represents a multi-pass rendering process for which the system critical path was the VIP in most of the cases. Ray tracing, radiosity and CSG are considered hghi-level graphics algorithms and are characterized by a main repeated kernel operation, such as ray/object intersection (Table 7) or hemi-cube rendering. Our double precision Linpack (100) performance is shown in Table 8), and our 1024 complex-point radix-4 FFT is presented in Table 9).

The performance figures shown represent a single VP system running at 8MHz and at 16MHz system clock. For 16MHz implementation, faster memory chips, address generators, and micro-sequencer are required. Price wise, a full VP system (with 64K deep WCS) runs $1824 at a 8MHz clock and $1969 at 16MHz.

The evolution path of the dot product unit we discussed previously has an estimated cost of $150 and is expected to increase VP performance in 3D graphics applications by about the following percentage points:

- Multiple directional light sources: 40%
- Point light source: 65%
- Patch tessellation: 65%
- Patch rendering: 45%

System Clock	KLines/sec	KPolygons/sec (1)	KPolygons/sec (2)	KPolygons/sec (3)
8MHz	295	33.2	26.6	17.8
16MHz	560	63	55	33

Table 4. Real-time performance of 3D geometry for lines and polygons. Polygons are quadrilaterals in a scene of a single directional light source with figures shown for three computational scenarios: (1) Trivial accept clip test and diffuse reflection, (2) trivial accept clip test and both diffuse and specular reflections, and (3) clip intersections of two out of four edges and diffuse reflection.

System Clock	KPolygons/sec (1)	KPolygons/sec (2)
8MHz	14.1	9.1
16MHz	26.75	17.2

Table 5. Real-time performance of 3D trivially accepted polygons in an environment of multiple directional light sources with both diffuse and specular reflections. Polygons are quadrilaterals with figures shown for (1) five and (2) ten lights.

System Clock	Patches/sec (1)	Patches/sec (2)
8MHz	125	115
16MHz	237	218

Table 6. Real-time performance of trivially accepted 3D patch tessellation and rendering. A patch comprises 16x16 quadrilaterals with figures shown for (1) non-rational and (2) rational B-splines in an environment of a single directional light source.

System Clock	Intersections/sec (1)	Intersections/sec (2)	Intersections/sec (3)
8MHz	135	50	83.3
16MHz	256	95	158

Table 7. 3D ray-primitive intersection performance. Figures shown for (1) ray/bounding-box, (2) ray/polygon, and (3) ray/sphere intersections.

8 DESIGN TOOLS

8.1 Hardware

EDS running on HP 350 workstation has been used extensively for the VP design. EDS provides a user friendly graphics editor for schematic capture and various interfaces for simulation (HiLo) and physical layout design (Calay).

The design was laid out on three boards - VP1, VP2, and VP3 - with a total of 650 chips. The majority of the components used are of Fairchild's ACT (CMOS) family, most of them in SM packaging. A component library has been built from scratch and ten new simulation functional models were written for memory and processor chips.

The design methodology was of a hierarchical nature and simulation was carried on from the lowest functional model level up to the board level. Test vectors were manually assigned and worst case analysis has been performed in every session.

The design data file size is a good measure of the design complexity. The VP consumes about 18MBytes primarily due to the wide buses.

The physical layout design addressed board density of 0.4 $inch^2$/16dip. The major challenge was to avoid a double sided surface mount component load in a hybrid environment of both plated thru and surface mount devices. Eight to

System Clock	MFLOPS
8MHz	1.4 (4)
16MHz	2.4 (8)

Table 8. Linear algebra: double precision Linpack (100). Number of coded BLAS routines shown in parenthesis.

System Clock	msec
8MHz	2.8
16MHz	1.5

Table 9. Image processing: 1024 complex-point radix-4 FFT.

twelve layer boards were assumed of which two pairs are VCC/GND layers, with very strict routing rules to minimize crosstalk and system ground bounce.

The design was carried on with off the shelf components, yet future potential integration was in mind. Particularly the logic unit, multiple address generators, and a 32 bit bi-directional register/buffer are the prime candidates. Applying integration of the scale above could conceivably lead to a single VP board (9u by 440mm).

8.2 Software

The VP instruction set is currently implemented with the FLAME microcode assembler. FLAME is an internal program developed at HP Labs in 1982 that was designed for horizontally-microprogrammed machines up to 128 bits. The program was extended to allow for our wide instruction word. FLAME accepts a description of the instruction set using a definition language and assembles micro-programs into the bit instruction representation with that definition language.

FLAME runs on the VAX computers and is written in PASCAL. The program did not port to the INDIGO or 9000-350 computers. It was found necessary to begin development of an assembler that implements efficient compilation algorithms for a large instruction word and runs on local machines. In addition, it was also intended to develop a linker for separate micro-program assembly.

ACKNOWLEDGMENTS

We would like to thank Dick Lampman and Joel Birnbaum for their unwavering support of the Mirage research project, as the technology developed thereof became the foundation of HP's showcase graphics 3D workstation.

REFERENCES

[1] Gordon Bell and William S. Worley. 1988. The Graphics Supercomputer: A New Class of Computer. *Information Processing* (1988), 727 – 734.

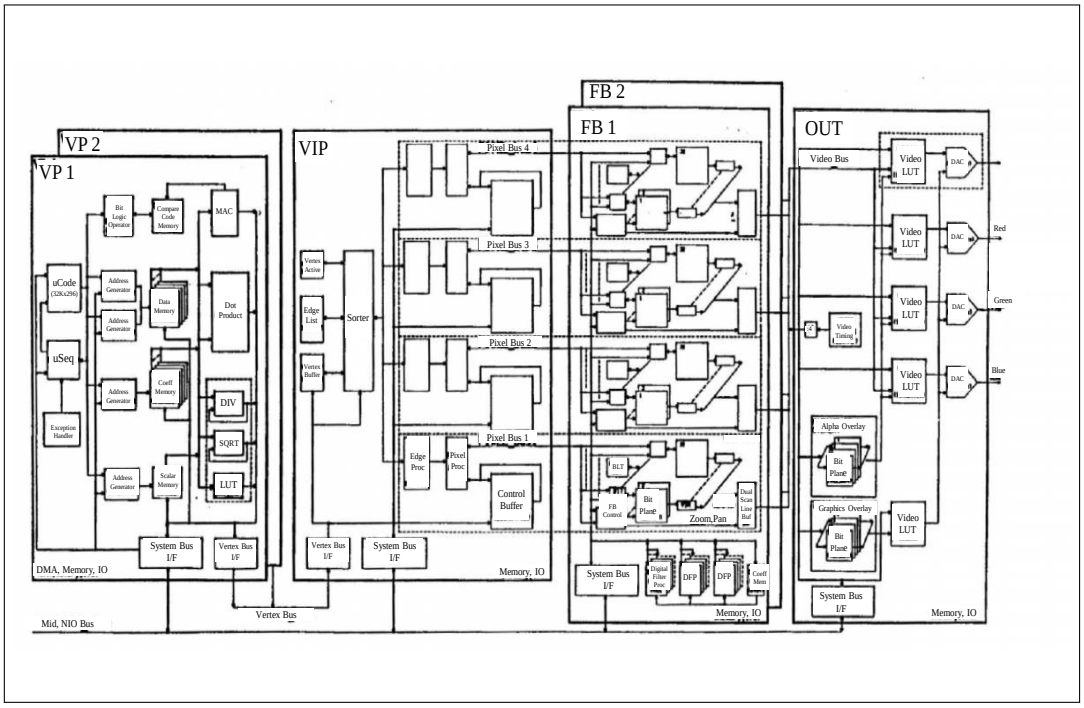

Fig. 1. Mirage architecture overview

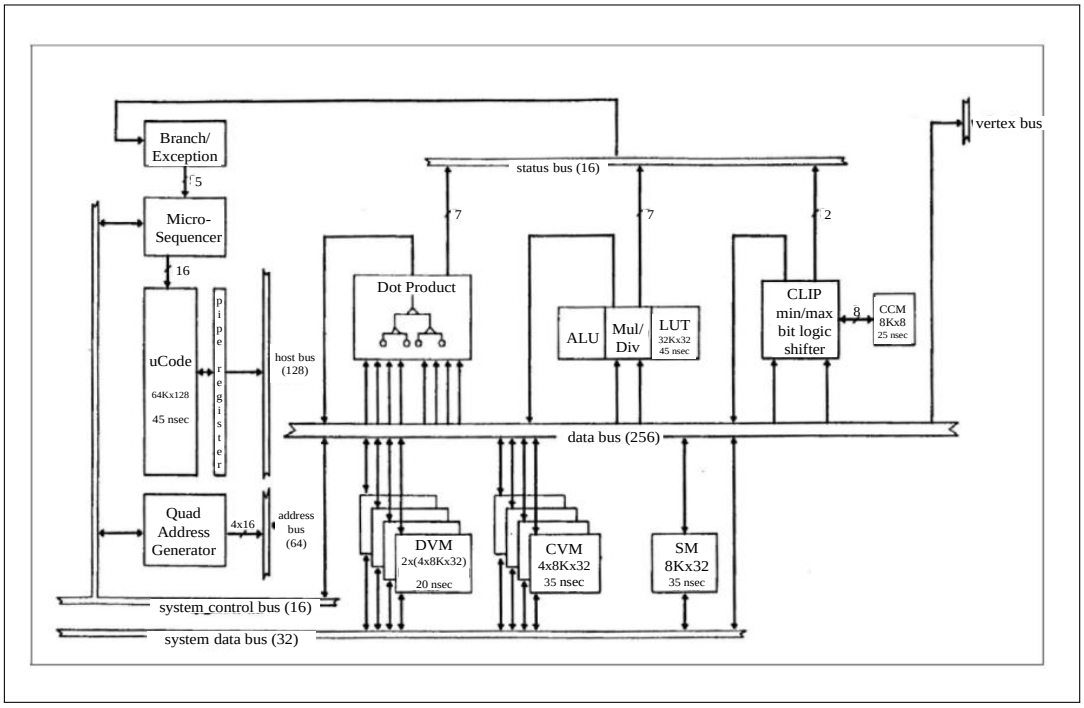

Fig. 2. VP architecture overview. All buses are 32 bit wide unless otherwise stated.

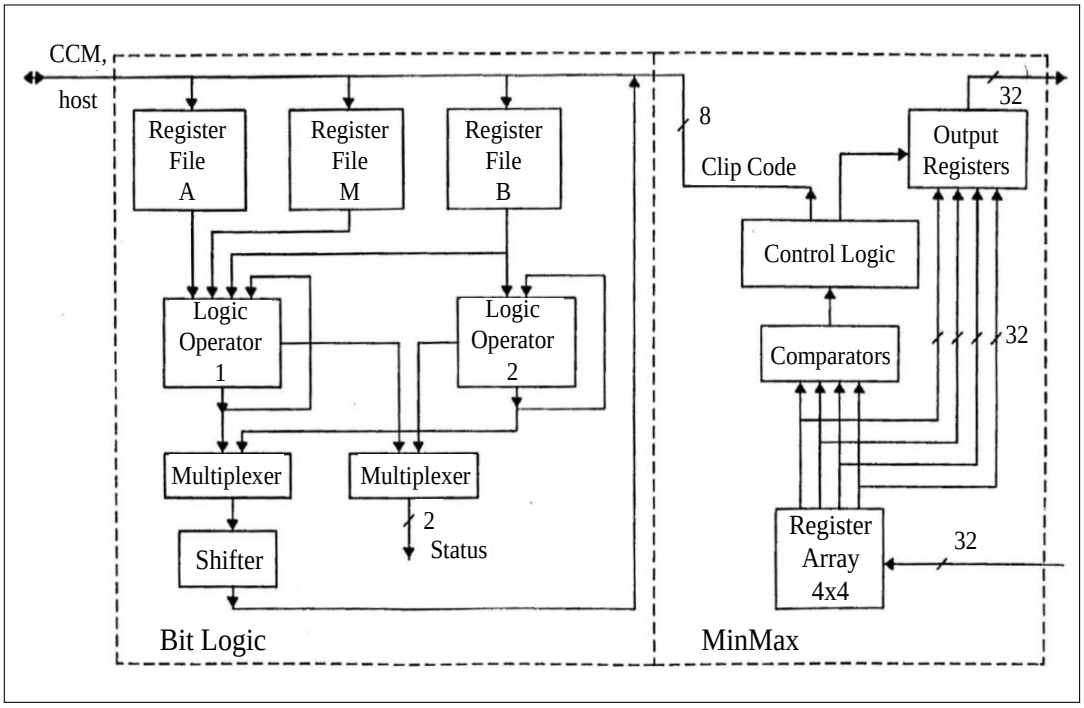

Fig. 3. VP Logic unit.