*** RELATÓRIO TÉNICO ***

# O PROJETO DE UM RISC PARA GERAÇÃO DE IMAGENS POR COMPUTADOR

Manuel Lois Anido

NCE 38/90
Dezembro/90

Universidade Federal do Rio de Janeiro
Núcleo de Computação Eletrônica
Caixa Postal 2324
20001 - Rio de Janeiro - RJ
BRASIL

Este relatório é cópia de um artigo publicado pelo autor na revista Microprocessors and Microsystems, butterwoth-Heinemann Publ., Vol. 14, nº 6, Agosto 1990, pp. 341-350.

O PROJETO DE UM RISC
PARA GERAÇÃO DE IMAGENS POR COMPUTADOR

## RESUMO

Este relatório descreve a análise e o projeto de um processador RISC, projetado para operar como um processador geométrico, para geração de imagens por computador. Este relatório apresenta uma nova técnica com canais DMA diretos para o conjunto de registros, juntamente com um multiplicador e um circuito de divisão de alto desempenho.

## ABSTRACT

This paper describes the design and analysis of a RISC processor, designed to operate as a geometry engine, for computer image generation. It presents a novel technique with DMA channels direct to the register file, a high-performance multiplier and a high-performance divider.

# RISC design for computer image generation

**Manuel L Anido and David J Allerton** discuss the development of a RISC-applicable instruction set for realtime computer image generation

*The paper describes the design and analysis of an instruction set which is applicable to a RISC processor operating as a geometry engine for computer image generation. A prototype LSI implementation of a fixed-point pipelined RISC is described. It provides a novel technique with DMA channels direct to the register file, a high-performance multiplier and a high-performance divider. The extension of this approach to a VLSI ASIC version using macrocells from a standard cell library is outlined together with a discussion of the testability issues. Results from an LSI implementation and an ASIC simulation are presented.*

**microsystems computer images geometric computation RISC ASICs**

RISCs have received popular acclaim in recent years for a number of reasons: first, their performance can exceed processors with complex instruction sets because their instruction decoding and execution is simpler and therefore faster, which in turn has led to simpler and more efficient compilers. Second, reduction in complexity is accompanied by a reduction in chip area. Consequently, chip area can also be utilized for floating-point processing, cache memory and memory management.

However, most RISC developments have been oriented towards the production of conventional sequential processors. Typically, these processors are judged on their relative performance in executing high-level languages for a wide range of applications. As a result of the demand for high-performance low-cost processors, the overall architectures of these processors resemble those of CISCs, particularly in terms of bus organization and memory access.

As RISC processors have become simpler in terms of internal architecture there has been a concomitant reduction in the processor design cycle in terms of man-years of development. Moreover, with advances in VLSI CAD packages, several RISCs have been produced from first-time silicon.

One common theme in the development of RISCs is the analysis of the characteristics of the application software, particularly the frequency distribution of executed instructions and their addressing modes, prior to the design of the processor. While detailed analysis results in increased speed of execution of instructions, processor performance has also been optimized by exploitation of state-of-the-art VLSI design styles, particularly full-custom design, since hardware resources are allocated to accelerate the most used instructions.

Concurrent with the development of RISC architectures, the impact of VLSI design has been most profound in the area of application-specific integrated circuits (ASICs). In order to reduce the overall design cycle, the designer enters his design at the schematic level using a predefined set of logic and arithmetic cells. The design is simulated by conventional logic simulation and the designer provides the semiconductor vendor with the schematic information and a suitable set of test vectors which define the overall operation of the device. The translation from the schematic form to a fabricated device is undertaken by the semiconductor vendor.

As ASIC technologies have progressed, the standard cells offered at the schematic stage have increased in complexity to the point where macrocells are available for ALUs, adders, multipliers, register files, shifters, etc. With processing fabrication reductions to 1 $\mu m$ technologies and improvements in the compactness of automated layouts, these macrocells are relatively compact and fast in comparison with full-custom equivalents. Thus, for the RISC designer, ASICs offer an effective method of implementing a particular design giving a relatively short time to silicon.

To summarize, there are two related themes in the developments of RISC design and ASIC technologies. First, RISC architectures can also be applied to specific applications where high processing rates are required but where a simple set of instructions is adequate for the application. Second, RISC processor design has been made more generally available by the recent development in ASIC design packages.

Department of Electronics and Computer Science, University of Southampton, University Road, Highfield, Southampton SO9 5NH, UK
Paper received: 23 January 1990. Revised 25 April 1990

0141-9331/90/06341-10 © 1990 Butterworth-Heinemann Ltd

As the use of computer workstations has become more widespread there has been a proliferation in the demands on and applications of computer graphics in workstations. The most exacting application area is realtime graphics, where images are continuously regenerated at video frame rates. One specific example is flight simulation, where the pilot's view is altered in realtime according to the changes of the position and attitude of the simulated aircraft. A second group of applications is in visualization systems, where large amounts of computer data must be processed at very high rates and the natural medium to visualize these results is interactive computer graphics.

In realtime graphics systems, objects are defined in 3D space, the image is transformed to 2D space (screen space) and this image is then formed in a framestore for subsequent display. This overall method is limited in terms of bandwidth in two respects. The arithmetic calculations in 3D graphics operations are demanding in terms of the inherent arithmetic operations and a very high data rate is required to form the transformed image in a framestore. This paper addresses the former problem, namely the execution of the graphical operations to transform 3D objects to screen space. A phrase often associated with this application is the term 'geometry engine'[1].

Visual fidelity improves with the image content. Usually, images are defined as a set of objects, each object is defined as a set of surfaces and each surface is defined by its vertices. Clearly, increasing the image content results in an increase in the processing requirement. To provide the effect of continuous motion in realtime systems the image must be regenerated at a rate of at least 25 frames/s and possibly at a rate of up to 60 frames/s[2]. This constrains the time available to generate the image.

Although commercially available microprocessors can be applied to the problems of image generation, two advantages result from the use of an application-specific RISC (ASRISC). First, the RISC can be 'tuned' to the application in terms of its instruction set and architecture, in order to enhance the performance. Second, the processor can be integrated into the system architecture in the most effective possible manner.

To transform objects from 3D space (world space) to screen space, there are three basic operations:[3]

- *Transformation*: the object vertices are transformed from the world axis system to the view-port axis system. In other words, the position and orientation of the view-port is used to redefine the axis system and each object undergoes rotation and translation to this new axis system.
- *Clipping*: as each object (or a part of each object) may not be contained within the 2D view-port, the parts of the object (in world space) that are outside the view-port must be removed (or clipped) with respect to the cone derived from the view-port.
- *Projection*: the resultant 3D object is projected onto the 2D view-port in order to take account of the perspective of the object.

These operations require a significant number of arithmetic operations, particularly multiplication and division, which have to be executed within the period dictated by the image update rate (usually less than 40 ms). The number of operations varies with the orientation and position of the view-port and also with image content, which makes it difficult to define an 'average' case. Moreover, the visual system has to be able to cope with near worst case situations, otherwise the user will notice a discontinuity in motion and/or a lack of synchronization. For objects represented by four-sided planar polygons, the best case and worst case arithmetic operations per polygon are shown in Table 1.

**Table 1. Arithmetic operations per polygon (best and worst cases)**

| | Best case (polygon outside screen) | | | Worst case (all edges cross screen) | | |
|---|---|---|---|---|---|---|
| Operation | + | × | ÷ | + | × | ÷ |
| Transformation | 24 | 36 | 0 | 24 | 36 | 0 |
| Clipping | 0 | 0 | 0 | 40 | 16 | 16 |
| Perspect. project | 0 | 0 | 0 | 16 | 16 | 16 |

Consider an image with 1000 quadrilaterals (4000 vertices) and an update rate of 25 times/s. In the worst case, the number of arithmetic operations per second is

$4000 \times 25 \times 68 = 6\,800\,000$ multiplications
$4000 \times 25 \times 32 = 3\,200\,000$ divisions
$4000 \times 25 \times 80 = 8\,000\,000$ additions

Unfortunately, multiplication and division instructions are typically more than 20 times slower than simple instructions on most microprocessors. Additionally, there are many more 'non-arithmetic' instructions to be executed to accomplish geometric computations, but the basic example above gives an insight into a formidable computation problem.

The image content (and consequently image quality) is largely dependent upon the speed of execution of the algorithms implicit in transformation, clipping and projection operations, and the speed of execution of the multiply and divide instructions.

As the resolution of typical screen displays is of the order 1024:1024, and the problem does not demand great magnitude or precision, it is possible that the complexity of floating-point arithmetic can be avoided. Fixed-point arithmetic can be used, provided that the overall resolution is maintained by means of *intelligent* scaling of the equations inherent in the image rendering processes. In order to avoid a larger and slower chip, it is also possible to operate with 16-bit, instead of 32-bit fixed-point arithmetic. Several implementations of fixed-point arithmetic have been demonstrated for realtime image generation systems[2,4,5].

There is one further consideration in image generation where the application differs from general-purpose applications. Objects are effectively passed to the image generation processor (stage 1) for transformation, clipping and projection (stage 2), and the 2D image is then passed to a framestore controller (stage 3).

This particular application allows a possible overlap of these three stages. While an object undergoes the above transformations, the previous object can be transferred out of the processor (to the framestore controller) and a new object can be acquired ready for processing on completion of processing the current object. This form of overlapped processing is not usually provided in conventional RISC processors, and its absence results in significant processor idle time while I/O operations are performed, and a significant overall degradation of performance. The use of data storage internal to the processor and allowing autonomous external access concurrent with processing can provide a significant speed improvement, but requires a rigid scheme to avoid possible contention for on-chip memory.

It is thus possible to gain significant improvement in overall speed for image generation over conventional processors by designing a RISC with an instruction set and architecture which is optimized for the application and which affords optimized I/O channels for the external reading of 3D objects and the writing of 2D objects.

## REALTIME IMAGE GENERATION APPLICATION

In order to assess the most used and time-demanding instructions of a realtime image generation (RTIG) application, qualitative and quantitative analyses of the University of Southampton flight simulator[4,5] (which employs MC68000 microprocessors) were carried out[6]. The results of the quantitative analysis are shown in Figure 1.

### Instruction analysis

The results of the instruction analysis are presented as a percentage of the total time per instruction rather than percentage of instructions per program because this reflects the problem more realistically. In addition, system characteristics such as addressing range, addressing modes, stack size, data space, and data structure were also analysed[6].

The quantitative results show that the time spent by the MC68000 on multiplication and division instructions represents almost 35% of the total. This is mainly because

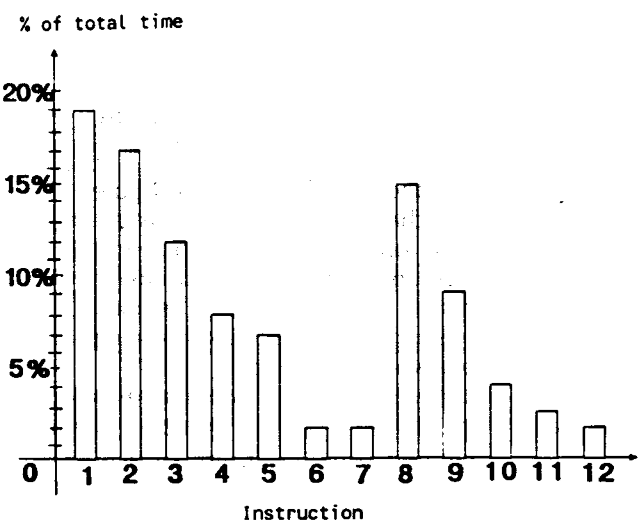


*Figure 1. Time requirements of geometric procedures on an MC68000 microprocessor (Instructions: 1, Multiplication; 2, Division; 3, Conditional branch; 4, BSR; 5, Shift; 6, Addition; 7, Subtraction; 8, Other instruction; 9, Load address; 10, Load data; 11, Move register to register; 12, Store data)*

multiplication takes 40 clock cycles (on average) to complete and division takes 80 clock cycles (on average) per operation. Thus their optimization is essential to improve system performance. Branches also consume a considerable amount of the processor time (15%). Optimizing the execution of branches is more difficult in highly pipelined architectures and usually requires hardware and/or software techniques to achieve significant improvements[7-10]. In this application, shift instructions occupy 7% of the time of the MC68000s. A shift instruction requires two clock cycles to shift an operand one bit position ($6 + 2n$ clock cycles, where $n$ is the number of positions to shift). This explains the significant time needed for this instruction.

Branch to subroutine (BSR) instructions consume 8% of the MC68000 processor time. However, the nesting depth level is small (less than six depth levels), which allows the implementation of a reduced and fast stack, internal to this RISC geometry engine. Load, store, and load address instructions represent almost 16% of the total time. This large proportion of the total time is characteristic of microprocessors with a limited number of internal registers, and leads to the use of the main memory for temporary storage. Move register to register instructions represent only 3% of the processor time, but represent 5% of the total number of instructions. Once multiplication, division, branch to subroutine and shift instructions are optimized, move register to register instructions represent a much higher percentage of the processor time. Therefore, their high frequency of occurrence supports a switch to three operand instructions.

### Additional findings

To achieve high polygon throughput, parallel processing is required. The workload can be easily partitioned by a host computer which can associate one subregion of the potentially visible polygons per processor. This workload partitioning is feasible because in realtime image generation scenes are usually defined by sets of independent objects and objects are composed of sets of independent polygons. By processing one polygon at a time, the data area required for 'polygon-related' data, transformation matrix, viewpoint coordinates, and temporary variables is relatively small (less than 64 16-bit words) and can be stored in the register file of a processor with an adequate number of internal registers.

Polygon-related data, such as vertex coordinates, vertex colour, and vertex intensity can be stored using a highly regular data structure, such as a table, for easy data access. Figure 2 illustrates a possible data organization. Similar data is repeated at constant intervals for the four polygon vertices. Thus, it can be used to facilitate procedure argument passing.

The quantity of data to be transferred into and out of the system (per polygon) may vary between 20 and 60 words, depending on the polygon position in relation to the view-port. If a polygon is totally outside the view-port, it is rejected during the clipping operation and no output data transfer is required. In this case, only input (3D) coordinates are transferred. However, if a polygon 'crosses' the view-port, it is clipped, which can result in several additional vertices and thus additional data (2D coordinates) to be output.

The data transfer time is significant when compared

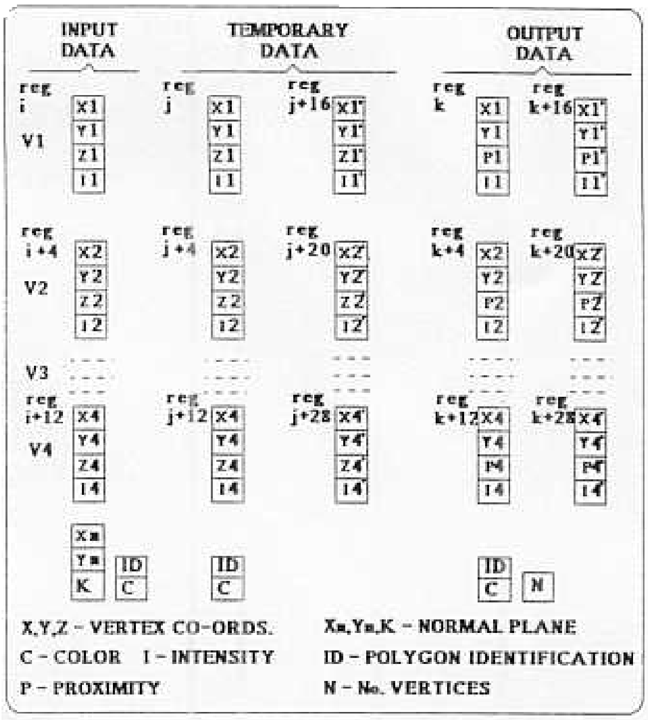


Figure 2. Polygon information and data structure

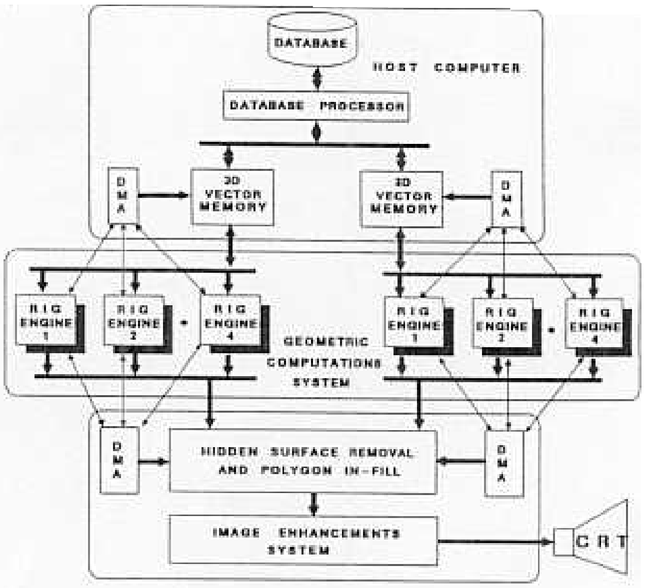


Figure 3. MIGS — a multiprocessor image generation system

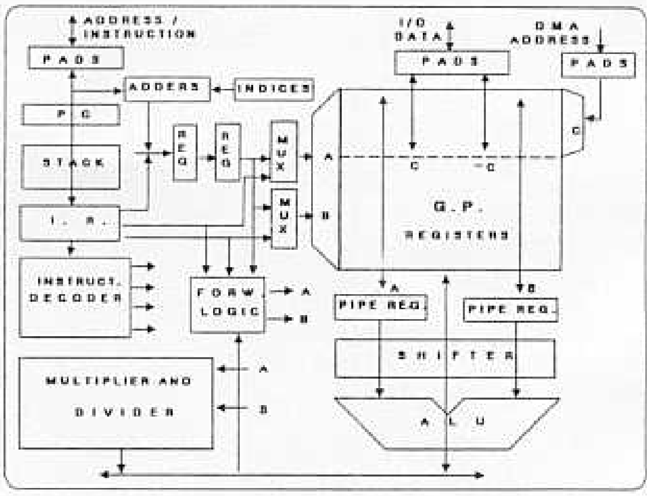


Figure 4. RIG internal architecture

with processing time because the program is small (less than 2k instructions) and there are relatively few iterations. For the same reason, no instruction cache is required. The program can be stored in a small and fast external memory. For data, the caching effect can be provided by an adequate register file. Instruction statistics have shown that the usual bit/byte/longword instructions are not necessary for this application, neither are complex addressing modes.

Using fixed-point number representation requires careful consideration of overflow, underflow and loss of accuracy. In order to avoid these problems a scaling operation is necessary, demanding the implementation of a prioritizer and a fast shifter for improved performance.

Unlike general-purpose RISCs this application does not require complex exception handling. Therefore, saving and restoring the state of condition codes is simple. Moreover, the program is small and can be hand coded, and compute instructions can be made to generate the condition code needed for a branch, suggesting the use of condition codes, instead of an explicit instruction to generate the required condition[7, 11].

## ARCHITECTURAL OVERVIEW

A parallel processing system which employs several concurrently operating RIG (RISC for image generation) geometry engines is used to achieve high polygon throughput. Its operation is discussed in more detail in a previous paper[12]. The organization of this multiprocessor image generation system is illustrated in Figure 3.

This paper concentrates on the architecture of the RIG engine which is an ASRISC and, unlike general-purpose RISC architectures, is characterized by an emphasis on

- extremely fast I/O data transfer
- the use of the internal registers as the main data storage area
- fast multiplication and division instructions
- an application driven instruction set

RIG internal architecture is illustrated in Figure 4, and its most important characteristics are discussed below. It accomplishes the processing of one polygon at a time and, by using an adequate number of internal registers (128), no external data memory is required for this application, thus avoiding data delays in the pipeline and improving processor performance.

This geometry engine also provides an extremely fast mechanism for transferring data into and out of the processor. Data is transferred in direct memory access (DMA) burst mode directly between the external environment and the internal registers, concurrent with processing. This is possible because the register file is organized in three independent sections:

- general-purpose registers and polygon data structure section
- input registers section
- output registers section

A program space of 2k instructions is compatible with the application, which allows the specification of an *absolute address* or an *offset* in the instruction field for branch instructions. Branch instructions (unconditional and conditional) use the absolute address for fast branching.

The target application does not require a large stack. Never the less a very fast call/return mechanism is necessary for improved performance. A small stack of depth 8 is used. Because all global variables, local variables, and procedure parameters can be stored in the register file, call/return instructions can be very fast. The program is small and can be hand coded in assembly language, therefore the task of ensuring that there is no stack overflow or underflow is left to the programmer.

RIG uses a fast Booth-encoded combinational multiplier to speed-up multiplications. A multiplication instruction is executed in two clock cycles. Division is accomplished by using a radix-4 signed division method that provides two quotient bits per iteration, instead of the usual single quotient bit per division step.

## Instruction format

The architecture of the RIG geometry engine is register-oriented because fast operand access and efficient use of the register file is essential to achieve high performance and also to minimize the overall area. Three-address instructions allow nondestructive register-to-register or register with immediate operations and are employed to achieve maximum register utilization. All RIG instructions have a fixed width of one word for simplicity of the fetch unit, with operand address fields at fixed locations, for direct and fast instruction decoding. This scheme also allows register access in parallel with instruction decoding. Constants and branch addresses can be embedded in the instruction field, providing immediate access to constants and fast branch execution.

## Instruction set

The instruction set is illustrated in Figure 5, reflecting the findings of the quantitative and qualitative analyses discussed above.

| OPERATION | OPERANDS | DESCRIPTION |
|---|---|---|
| ADD | sc1,sc2,dst | dst := sc1+sc2 |
| ADDc | sc1,sc2,dst | dst := sc1+sc2+cy |
| SUB | sc1,sc2,dst | dst := sc1-sc2 |
| SUBc | sc1,sc2,dst | dst := sc1-sc2-cy |
| AND | sc1,sc2,dst | dst := sc1 & sc2 |
| OR | sc1,sc2,dst | dst := sc1 v sc2 |
| XOR | sc1,sc2,dst | dst := sc1 + sc2 |
| LSL | sc1,sc2,dst | dst := sc1 sl sc2 posit; lsb = 0 |
| LSR | sc1,sc2,dst | dst := sc1 sr sc2 posit.; msb = 0 |
| ASR | sc1,sc2,dst | dst := sc1 sr sc2 posit. ; msb = msb |
| SCALELl | sc1,sc2,dst | dst := low(sc1 concat sc2 << n) |
| SCALELh | sc1,sc2,dst | dst := high(sc1 concat sc2 << n) |
| SCALERl | sc1,sc2,dst | dst := low(sc1 concat sc2 >> n) |
| SCALERh | sc1,sc2,dst | dst := high(sc1 concat sc2 >> n) |
| BRA | addr,cc | If cc = true then PC := addr else PC := PC+1 |
| CMPBRA | sc1,offset,cc | If (sc1=0 & cc=0) v (sc1<0 & cc=1) -> PC:=PC+offset |
| CALL | addr | PC := addr ; Stack := PC+1; StackP := StackP +1 |
| RET | -------- | StackP := StackP -1 ; PC := Stack |
| MULs | sc1,sc2,dst | dst := low(sc1 * sc2) |
| DIVSTEP | - - | Division Step |
| PRIOR | sc1,dst | dst := priority (sc1) |
| RDREG | sc2,dst | dst := register(sc2) |
| WRREG | sc1,sc2 | Register(sc2) := sc1 |
| MOVEQ | n,dst | dst := n |
| NOP | -------- | No operation |

*Figure 5. RIG instruction set*

The SCALE instruction performs a multiple bit shift of two concatenated registers (left or right), using a funnel-shifter. The number of bits to shift is specified by the 'scaling factor' register, which is implicitly loaded by the prioritize (PRIOR) instruction or can be explicitly loaded using the WRREG instruction. This instruction is used to scale operands in fixed-point arithmetic so that overflow, underflow and loss of accuracy problems are avoided or minimized.

The PRIOR instruction determines the 'priority' of the most significant bit (MSB). It determines how many positions an operand has to be left-shifted to align the MSB bit to the left, next to the sign bit. It operates on positive or negative numbers. The instruction result is stored in a general-purpose register (explicit in the instruction) and in the 'scaling factor' register (implicit in the instruction).

RDREG and WRREG instructions control the read/write operations on the, multiplier, divider and priortizer circuits, as well as I/O control. Therefore, operations such as 'load dividend', 'read quotient' or 'set DMA output request bit' are accomplished by these instructions.

The CMPBRA instruction performs a 'quick compare and branch' operation[9], which makes it possible to suppress a significant number of conditional branch instructions.

## Four-stage pipeline

Pipelining has been an essential technique in improving processor performance. To balance the utilization of pipeline stages (and also take advantage of the pipeline organization), it is necessary to evaluate the time required by each operation in the early stages of the design. Instruction decoding in the RIG engine is simple (a common RISC feature) and register read/write operations are fast, when compared with instruction execution or instruction fetch operations (approximately twice faster). The use of a three-operand instruction format allows instruction decoding and register read operations to be effected in parallel (because register addresses and instruction opcode are ready simultaneously).

RIG uses a four-stage pipeline, with half-clock cycle times available for instruction decoding and register read/write operations (see Figure 6). All functional units (fetch, decode and execute units) are 100% occupied. One instruction is executed every clock cycle and there are three overlapped instructions in the pipeline.

This pipeline is much simpler than that of general-purpose RISCs because no external data access is required (data can always be read from or written to the internal registers). Therefore, no address computation and data access pipeline stages are necessary. Only one level of internal forwarding (or bypassing) is required because the instruction result is written in the register file in the following cycle. In order to provide one cycle load instructions, some RISCs[10] delay the write result operation by one cycle, which demands two levels of data forwarding to avoid pipeline interlocks.

## Branches

Branch instructions use an absolute address embedded in the instruction register. There is no PC-relative addressing mode for branch instructions because no relocatable

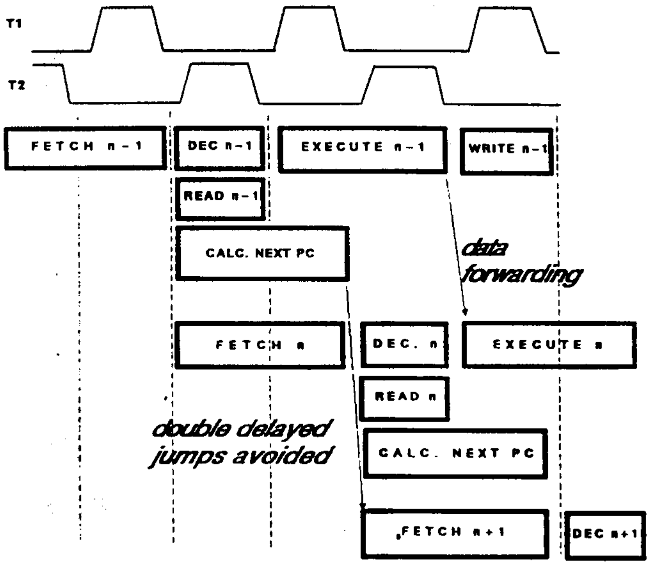


*Figure 6. RIG four-stage pipeline*

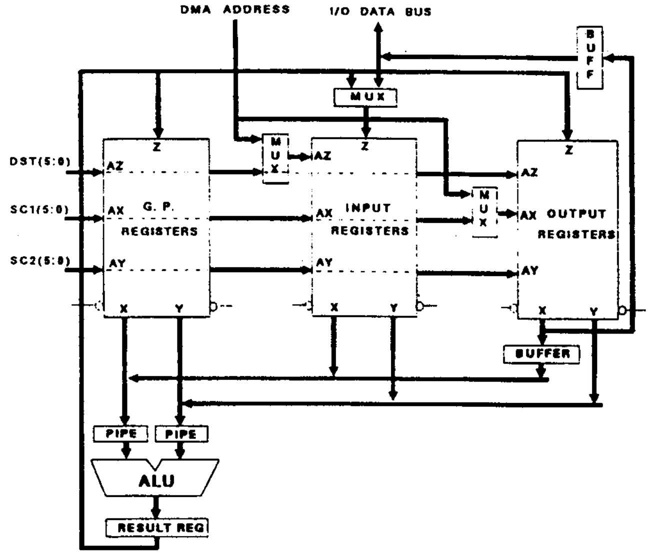


*Figure 7. Register file organization*

object code is required. Conditional branches are performed according to the condition codes, which are set by the arithmetic and logic instructions. The branch delay for these instructions is only one cycle, because there is no need to use the ALU to calculate the next PC address.

A significant percentage of branch instructions (approximately 20%) can be eliminated by employing a quick compare and branch (CMPBRA) instruction[9]. The instruction field incorporates an offset which is added to the program counter, providing a PC-relative addressing mode for this instruction. This addition is performed in the instruction fetch unit (IFU), rather than the ALU. Instead of a full comparison of two registers, which would demand the completion of an ALU cycle (consequently adding an extra clock cycle), a quick comparison of a register with zero or sign comparison is effected. This comparison is realized at the end of the register file read cycle by a simple and fast circuit and thus avoids double delayed jumps in the four-stage pipeline.

To implement half-clock cycle operation, the IFU master and slave clocks are swapped with the master and slave clocks of the rest of the system. Therefore, it is not possible to connect all system registers in a single shift register for scan-path testing. Two solutions were considered: the use of two extra scan-in and scan-out pins for the instruction fetch unit or multiplexing the master and slave clocks (controlled by the test signal). In order to avoid extra test pins, the latter solution — which allows all system registers to be connected in a long shift register — was adopted.

## HARDWARE RESOURCES

### Register file and indices

A register file with three sections is provided: one section for the general-purpose registers and polygon data structure, one section for data input, and one section for data output. These input and output registers can be read/written directly by the processor, and can also be accessed by an external DMA device, concurrent with processing. In a previous paper[13] several options to implement such a register file were discussed. It was shown that (for this application) a register file with three-port/three-access cells, allowing read/write operations on all ports, would afford advantages over a first in, first out (FIFO) memory or register banks in terms of area, regularity and the number of buses required. However, standard cell libraries do not normally provide such register files and this option is only feasible using a full-custom implementation.

To use proprietary register files or register files available in standard cell libraries (two read/write ports and a third write port), a scheme with 64 general-purpose registers, 32 input registers and 32 output registers was chosen. These registers are organized in a linear addressing space, which requires 7-bit in the instruction register (IR) operand fields. As illustrated in Figure 7, input and output registers can be addressed by the processor (using the SC1, SC2 and DST address buses) or by an external DMA controller (using the DMAADDR address bus). The input data required during processing can be accessed directly from the input registers, dispensing with additional instructions or storage space to move data to the general-purpose registers. The same reasoning applies to output. Additionally, I/O registers can be used as general-purpose registers.

To achieve an effective parameter passing mechanism which maximized register usage, two register file organization techniques were examined. First, a fixed-size multi-window register file organization[9] was considered. Although the nesting depth level in this application is small (which favours windows), this scheme lacks efficiency because it does not provide sufficient registers for some procedures and wastes registers for others. While the translation, transformation, and perspective procedures require few parameters and local variables, the clipping procedure (which is frequently used) requires a large number of parameters and local variables. Second, a variable-size multi-window register file (or register stack) organization[9, 14] was examined. In this scheme each window is only as large as needed, making effective use of the register file. The RIG processor uses a variation of this technique, which is described in more detail in Reference 12. The data structure is regular, thus making the use of indices a feasible choice. The penalty for using indices is the extra time required to perform the addition of a register address with an index, obtaining a new register address. However, the required adders are only 7-bit wide and can perform the addition in a time that does not compromise the system clock rate. In many cases, passing parameters to a procedure and retrieving results can be performed by a single instruction. This method is advantageous in this particular case, but cannot be used as a broad scheme for argument passing in general-purpose RISCs.

### Funnel shifter and prioritizer

To support fast scaling in fixed-point arithmetic and provide fast shift instructions, a funnel-shifter and a prioritizer circuit were used. The funnel-shifter performs multiple-bit shifts of two concatenated operands ($2n$-bit input and $n$-bit output), in one clock cycle. The prioritizer circuit determines the position of the most significant bit of positive or negative numbers. It determines the number of positions left an operand must be shifted so that the most significant bit will be aligned with the sign bit.

### Combinational multiplier

Multiplication (16-bit × 16-bit with 32-bit result) is performed by a fast Booth-encoded combinational multiplier, which is regular, iterative and uses a smaller area than an array multiplier[15, 16]. It takes two clock cycles to accomplish a multiplication operation. In the first instruction cycle operands are loaded in the multiplier and the lower result bits are read. The higher part of the result is available only on the following clock cycle. In order to minimize loss of accuracy, arithmetic operations are effected on 32-bit operands, converting the final result to 16 bit.

Performed in the conventional manner, binary multiplication requires the summation of as many partial products as there are bits in the multiplier. Techniques to reduce the number of partial products have been developed and fall under the heading of signed-digit recoding schemes[16]. The modified Booth algorithm is one of these techniques, encoding the multiplier bits in such a way that the number of partial products required is only half that of the 'AND' gate approach[16].

Some RISC designers have implemented multiplication based on the observation that most multiplications involve a small constant known at compile time[17], and this can be used to speed up multiplication. However, in this application most of the multiplications have variable magnitude operands. Therefore, this mechanism is not suitable for this application.

### Radix-4 divider

Division is intrinsically more complex than multiplication and demands considerable effort for its optimization. The two most common methods for division are based on repeated subtractions, as exemplified by the 'paper and pencil' method, or are based on repeated multiplications, e.g. the Newton-Raphson method[16]. A major disadvantage of the Newton-Raphson method is the need for a large ROM to obtain a first good approximation of the quotient digits to converge in a few iterations.

A radix-4 signed division method which employs 3 ×divisor multiples and uses a reduced next divisor multiple estimate table[18] is implemented using compact and fast combinational logic. Several higher radix division implementations use 2 × divisor multiples (they are easier to generate) but this implies the use of fairly large, slow PLAs, and has precluded their application in the integer unit of microprocessors[19, 20]. The PLA time adds to the ALU carry-propagate adder time, slowing down the system.

The division method employed in this research project provides two quotient bits per iteration, instead of the usual single quotient bit per division step. This method also deals directly with signed two's complement numbers, eliminating the need for additional instructions for sign conversion. This is achieved at negligible additional cost because commonly used ALU hardware (also needed for other instructions) is employed. In order to use the ALU carry-propagate adder, partial remainders and quotient bits are kept in irredundant form.

Figure 8 illustrates the circuit employed to calculate the partial remainders. The divisor multiple 3D is obtained by adding the divisor multiple 2D and D in the first division instruction step. The logic equations for the m0 and m1 signals are simply

$$
\begin{aligned}
m0 &= (S.\overline{Y2} + \overline{Y3}.\overline{Y2}.Y0 + \overline{Y3}.S.\overline{Y0} + \overline{Y2}.\overline{Y1}.Y0 \\
&\quad + S.Y1.\overline{Y0} + \overline{S}.Y2) + \text{divstep1}; \\
m1 &= (Y2.\overline{Y1} + S.\overline{Y2} + \overline{S}.Y1).\overline{\text{divstep1}}; \qquad (1)
\end{aligned}
$$

These equations can be implemented using a single 16:1 multiplexer (for m0) and an and/or logic gate (for m1), as illustrated in Figure 8. This logic is more compact and faster than solutions employing divisor multiples chosen

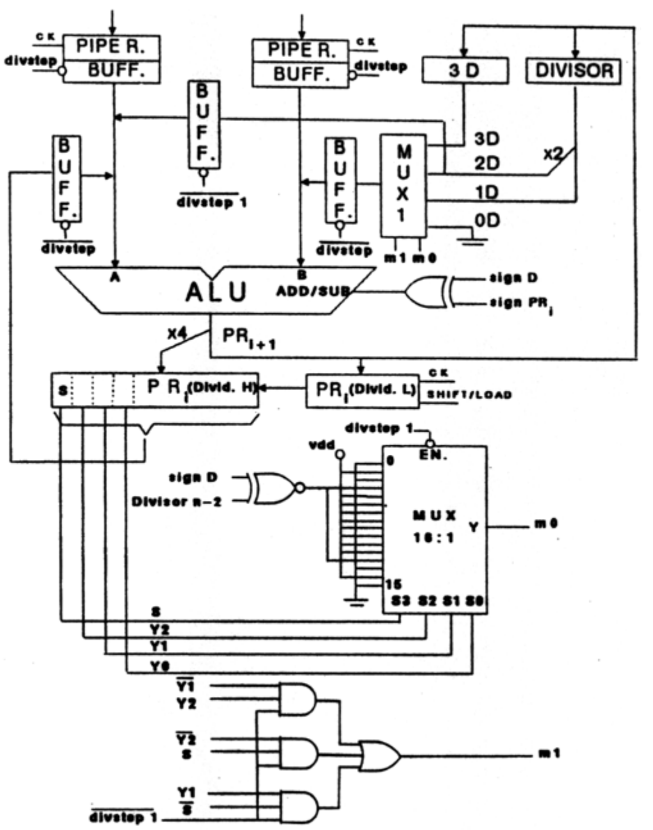


*Figure 8. Circuit for division partial remainder calculation*

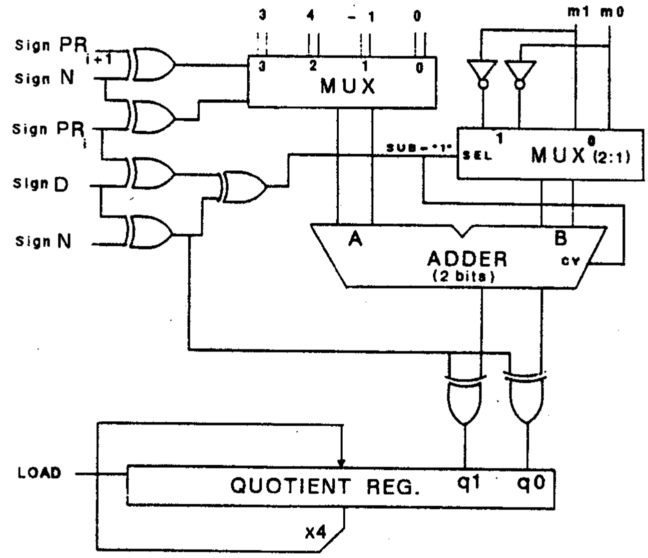


*Figure 9. Circuit for division quotient calculation*

from the set {−2,−1,0,1,2}[19, 20], and allows its use in the arithmetic unit because no significant time is added to the critical path.

Figure 9 illustrates one possible implementation to calculate the quotient digits, which are stored in irredundant form. If the quotient is negative, quotient digits are generated in one's complement and must be incremented at the end of the division process.

## EMULATION AND VLSI DESIGN

Prior to VLSI design, the system was emulated. Both software emulation and hardware emulation were considered. While software emulation allows easier changes in the architecture and usually involves less cost, a hardware emulation makes it possible to evaluate the system's realtime capabilities. Moreover, a low-level software emulation of a complex system can take a long time, even for a small program. A switch-level simulator running a design with approximately 80k transistors on a Micro Vax II is reported to execute one clock cycle every minute[10]. This implies that it takes 16.6 hours to run a program with 1000 instructions. On the other hand, if the target is a full-custom implementation, a software emulation can provide input vectors to be applied to the extracted circuit, and in this case the software approach is virtually mandatory. Thus each case must be analysed separately to decide upon the appropriate method of emulation.

### Using proprietary system components

In order to validate the instruction set and provide realistic performance evaluation, the RIG geometry engine was emulated using proprietary LSI building blocks, which allowed the system to be constructed with a moderate number of ICs. This gave an insight into the main problem areas and bottlenecks.

The RIG geometry engine was emulated using Am29300 chips (32-bits), including an ALU and funnel-shifter (Am29332), a register file (Am29C334) and a combinational multiplier. The control is very simple (as usual in RISC-like systems) and is performed by several PAL and PROM chips. The system architecture is similar to the final VLSI design using a standard-cell approach. Subsequent mapping of this hardware emulation to a standard-cell implementation was relatively straightforward.

**Table 2. Comparison of 12 MHz MC68000 and 16 MHz RIG for certain instructions**

| Instruction | MC68000 | RIG | Improvement |
|---|---|---|---|
| Add | 166 ns | 62.5 ns | 2.6 × |
| Shift | 1.66 μs | 62.5 ns | 26.6 × |
| Call (Bsr) | 1.33 μs | 62.5 ns | 21.3 × |
| Multiply | 4 μs | 125 ns | 32.0 × |
| Divide | 10 μs | 2 μs | 5.0 × |
| Branch | 666 ns | 62.5 ns | 10.6 × |

**Table 3. Time measurements on RIG geometric procedures**

| Procedure | Best case (μs) | Average (μs) | Worst case (μs) |
|---|---|---|---|
| Scaling | 2.8 | 7.8 | 11.2 |
| Transform. | 13.5 | 13.5 | 13.5 |
| Clipping | 15.6 | 26.2 | 94.0 |
| Perspective | 4.4 | 18.4 | 50.0 |
| Total time per polygon | 36.3 | 65.9 | 168.7 |

### *Performance measurements*

Table 2 depicts a time comparison between 12 MHz MC68000 instructions and the equivalent RIG 16 MHz instructions. Measured performance indicates that a 16 MHz RIG geometry engine (16 MIPS peak processor) emulation executes the geometric procedures 15 times faster than a 12 MHz MC68000 microprocessor. Unfortunately, no performance figures comparing RIG with more recent processors are available.

Table 3 illustrates some time measurements realized on RIG geometric procedures. They indicate that, on average, this RIG emulation is capable of delivering a transformed polygon every 66 μs, which corresponds to 15 000 polygons per second or 600 polygons in realtime (considering an update rate of 25 times/s). As is shown below, the VLSI design achieves twice this performance.

### VLSI design

The RIG geometry engine has been designed in VLSI using a 1 μm CMOS standard cell library, which has parameterized silicon compiler datapaths, such as ALU, barrel-shifter, register file and multiplier. This approach provides packing densities and speed close to full-custom designs, with the design turnaround times of ASIC standard cells.

A two-phase non-overlapping clock scheme is employed (see Figure 6). Because the design is synchronous, this clock scheme avoids any race conditions (the only races are with clocks). By generating the master and slave phases external to the chip it is possible to control the clock skew immunity and the frequency of operation.

Table 4 shows the time consumed by the major pipeline elements. It shows that the slowest element in the

Table 4. **Speed of certain pipeline elements**

| Operation | Time required (ns) |
|---|---|
| Memory access | 33 |
| Register read/write | 14 |
| Instruction decode | 11 |
| Branch test | 10 |
| ALU | 28 |
| Shifter | 27 |
| Multiplier | 2 × 30 |
| Prioritizer | 13 |

Table 5. **Area of certain basic system blocks**

| Block | Area ($mm^2$) |
|---|---|
| Register file | 17.80 |
| Multiplier | 3.82 |
| ALU | 1.11 |
| Shifter | 1.30 |
| Stack | 0.30 |
| Bus buffers | 0.62 |
| Adders | 0.65 |
| Registers | 2.50 |
| Mux and decoder | 0.20 |
| Other logic | 1.10 |

pipeline is the external memory. Using standard 2k × 8 — 25 ns external static RAMs to store the program implies a total access time of 33 ns. To speed up execution time, a faster memory is necessary. It is feasible to store the program in an internal ROM and eliminate the memory access time bottleneck but the speed improvement will be marginal. The significant advantage of this approach is to provide a geometry engine system in a single chip.

It can be seen from Table 4 that the VLSI design runs twice as fast as the hardware emulation discussed previously. This implies a geometry engine that can run the geometric procedures 30 times faster than a 12 MHz MC68000 microprocessor. The table also shows that the register read time and instruction decoding time represent less than half the memory access time, proving that the allocation of half clock cycle time for these operations was appropriate.

Table 5 presents the area of the major system blocks. It can be seen that the register file and the multiplier consume a large area. However, datapath modules are very compact, with minimal routing between them. On the other hand, the interconnection between those modules and the rest of the logic occupies a considerable area.

The VLSI standard cell design is validated by running the geometric procedures in a logic simulator and comparing the perspected co-ordinates with the co-ordinates generated by the University of Southampton flight simulator. This is accomplished by loading the object code generated by the assembler in a file that the logic simulator 'sees' as the RIG program memory. Running on an Apollo 3500 workstation with a 25 MHz MC68030 microprocessor, the simulation takes approximately three seconds per clock cycle.

### *Built-in testability*

Observability and controllability can be built into a circuit by incorporating the ability to load and store the state of

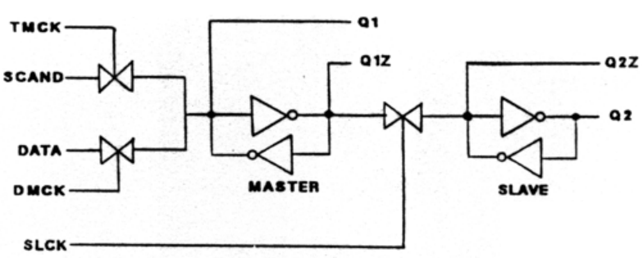


*Figure 10. Basic latch for scan-path testing*

every flip-flop (or latch) in a circuit from the device pin. The test problem is then reduced to testing the combinational logic between the memory elements, i.e. the state of a circuit loaded. The combinational logic is exercised by setting the internal memory elements to a new state which can be stored and compared to the result expected from a functionally correct device. In a scan design, all the latches and flip-flops are joined together into a serial shift register[21]. By shifting data serially (scanning) in the shift register the state of each flip-flop can be controlled and observed.

The standard cell library used in this project has special cells that use this testing philosophy. The test method is based on the scan-path testing philosophy. Figure 10 illustrates a latch with testability support for scan-path testing.

All large blocks in the system (ALU, funnel-shifter, multiplier, etc.) incorporate I/O registers, which allows them to be tested separately from the rest of the circuit. This modular scan design approach simplifies test vector generation and gives rise to a highly structured test methodology.

## CONCLUSIONS

By careful analysis of the algorithms inherent in realtime image generation, it is possible to design a RISC processor which is capable of executing the algorithms significantly faster than conventional microprocessor systems. The use of a novel technique that employs autonomous I/O channels direct to the register file further enhances the processing speed of the RISC in applications as a geometry engine. The resultant instruction set contains 25 instructions and 128 registers. All arithmetic instructions contain three register fields and a special-purpose register file was designed to avoid contention between processor access to the register file and DMA transfers to and from the register file.

A further substantial improvement in speed, when compared with CISC computers, has been obtained by means of a four-stage pipeline. The implementation is relatively straightforward due to the simplicity of the instruction set. Because the processor does not have to access external data, dispensing with load and store instructions, and because no delayed result write operation is required, one level of data forwarding is sufficient in the four-stage pipeline. The quick-compare and branch instruction eliminates a significant percentage of conditional branch instructions and avoids double-delayed branches, which would be required in a full compare instruction using the ALU.

Analysis of the graphics algorithms emphasized the need for high-speed fixed-point multiplication and division. Problems of integer scaling are ameliorated by the implementation of a funnel-shifter. Multiplication is achieved by means of a variant of the Booth encoded algorithm. High-speed division is achieved by means of an adaptation of a two-bits-at-a-time method.

The overall simplicity and regularity of the architecture lends itself to implementation using ASIC macrocells. However, prior to this development, a prototype version was constructed using equivalent LSI components. This development confirmed the target processing speed and provided a ready means of identifying possible bottlenecks. Recently, a VLSI ASIC design has been undertaken using a 1 $\mu$m CMOS standard cell library. A performance approximately 30 times faster than a 12 MHz MC68000 processor is predicted for the VLSI implementation, with a chip area of approximately 50 mm$^2$.

## ACKNOWLEDGEMENTS

The authors acknowledge the financial support provided by the Brazilian organization CNPQ, the Federal University of Rio de Janeiro and the ORS award scheme (UK) for M L Anido's research at the University of Southampton.

## REFERENCES


1 **Clark, J H** 'The geometry engine: A VLSI geometry system for graphics' *Proc. ACM Siggraph* (1982) pp 127–133

2 **Schachter, B J** 'Computer image generation for flight simulation' *IEEE Comput. Graph. Applic.* Vol 1 No 4 (October 1981) pp 29–68

3 **Foley, J D and Van Dam, A** *Fundamentals of Interactive Computer Graphics* Addison Wesley, Reading, MA, USA (1984)

4 **Allerton, D J and Zaluska, E J** 'Computer image generation in real time' *Proc. Electron. Displays* (1985) pp 17–31

5 **Allerton, D J and Zaluska, E J** 'A multi-processor approach to image generation' *IEEE Int. Conf. Simulators* Warwick, UK (1986) pp 226–231

6 **Anido, M L** *The design and implementation of a RISC processor for real-time image generation geometric computations* PhD thesis, Dept. of Electronics and Computer Science, University of Southampton (1990)

7 **Hennessy, J L** 'VLSI processor Architecture' *IEEE Trans. Comput.* Vol C-33 No 12 (December 1984) pp 1221–1246

8 **McFarling, S and Hennessy, J L** 'Reducing the cost of branches' *Proc. IEEE/ACM 13th Annual Symp. Comput. Architect.* (1986) pp 396–403

9 **Katevenis, M G H** *Reduced Instruction Set Computer Architectures for VLSI* MIT Press, London, UK (1985)

10 **Horowitz, M, Chow, P, Stark, D *et al*** 'MIPS-X: a 20 MIPS peak, 32-bit microprocessor with on-chip cache' *IEEE J. Solid-State Circuits* Vol sc-22 No 5 (October 1987) pp 790–799

11 **Hennessy, J L and Jouppi, N** 'Design of a high performance VLSI processor' *3rd VLSI Conf.* Caltech, USA (1983) pp 33–54

12 **Anido, M L, Allerton, D J and Zaluska, E J** 'MIGS — a multiprocessor image generation system using RISC-like microprocessors' *Proc. Comput. Graph. Int.-CGI '89* Springer-Verlag, Berlin, FRG (June 1989) pp 321–331

13 **Anido, M L, Allerton, D J and Zaluska, E J** 'A three-port three-access register file for concurrent processing and I/O communication in a RISC-like graphics engine' *Proc. IEEE/ACM — 16th Annual Int. Symp. Comput. Architect.* Jerusalem, Israel (1989) pp 354–361

14 **Ditzel, D and McLellan, H** 'Register allocation for free: the C machine stack cache' *Symp. Architect. Support Prog. Lang. and Oper. Syst. ACM: SIGARCH CAN* Vol 10 No 2, *SIGPLAN Notices* Vol 17 No 4 (March 1982) pp 48–56

15 **Henlin, D A, Mazin, M, Fertsch, M T and Lewis, E T** 'A 16 × 16 pipelined multiplier macrocell' *IEEE J. Solid-State Circuits* (April 1985) pp 542–547

16 **Hwang, K** *Computer Arithmetic: Principles, Architecture and Design* John Wiley, New York, NY, USA (1978)

17 **Magenheimer, D J, Peters, L, Pettis, K W and Zuras, D** 'Integer multiplication and division on the HP precision architecture' *IEEE Trans. Comput.* Vol 37 No 8 (August 1988) pp 980–990

18 **Anido, M L and Allerton, D J** 'Radix 4 division using maximally redundant divisor multiples and a fast divisor multiple estimate logic' paper in preparation

19 **Atkins, D E** 'Higher-Radix division using estimates of the divisor and partial remainders' *IEEE Trans. Comput.* Vol C-17 No 10 (October 1968) pp 925–934

20 **Taylor, G S** 'Compatible hardware for division and square root' *Proc. 5th IEEE Symp. Comput. Arithmetic* (May 1981) pp 127–134

21 **Williams, T W** 'VLSI Testing' *IEEE Comput.* Vol 7 No 10 (October 1984) pp 126–136


*Manuel L Anido was born in Spain and received BSc and MSc degrees from the Federal University of Rio de Janeiro (UFRJ), Brazil in 1976 and 1980 respectively. He worked at the Computing Center of UFRJ (NCE/UFRJ) until 1986 as a lecturer and a senior design engineer. Presently he is completing a PhD in computer architecture/computer graphics at the University of Southampton, UK. His research interests include computer architecture, VLSI design, computer graphics and computer arithmetic.*

*David J Allerton is currently senior lecturer in computer science in the Department of Electronics and Computer Science at the University of Southampton, UK. He joined the Department in 1981 after working as a principal engineer for Marconi Space and Defence Systems. His research interests include realtime computer graphics, computer architectures, silicon compilation, flight simulation and operating systems. He has published over 20 papers in computer science and application areas.*