ARCHITECTURE FOR SINGLE-CHIP ASIC PROCESSOR WITH INTEGRATED FLOATING POINT UNIT

Vojin G. Oklobdzija

IBM T. J. Watson Research Center,
P.O.Box 218
Yorktown Heights, NY 10598
(914) 945-2607

ABSTRACT

An architecture for a single-chip processor with emphasis on ASIC implementation is presented. The basic principles of RISC architecture are applied to the processor and in addition fast floating-point and serial I/O units are included on the chip. The limitation of the processor's complexity is a 50,000 gate capacity of the chip, and 600 pS per gate delay. These are the characteristics of vendor CMOS-ASIC technology currently available. The architecture is a trade-off between the features desirable in order to make the processor attractive for a wide range of computational problems, and the possibilities of the available technology. The architecture takes advantage of a high level of integration and a low power consumption which is achievable with CMOS technology. It also benefits from the elimination of clock skews and off-chip delay penalties associated with comparable implementation using several chips.

INTRODUCTION

One of the principles of RISC architecture (RISC: Reduced Instruction Set Computer) is to trade complexity for the speed of execution [1],[2]. The speed is achieved by designing a simpler and smaller instruction set which executes on a simpler hardware. In turn the simplicity of hardware allows for a faster instruction execution. Performance achieved by RISC processors is comparable to the performance of medium size machines [3],[4]. This is true even after reducing the performance figure of a RISC machine by the factor of 2-3 in order to have a meaningful comparison of "less powerful instructions" of a RISC to "more powerful instructions" of a CISC machine (CISC: Complex Instruction Set Computer), or running a set of standard benchmark programs [5].

However, a very important and often overlooked benefit of simplicity is a shorter design cycle which offers the ability to reach the market place with a competitive performance in time.

The machine with the complex instruction set takes longer to design. Therefore, in order to be competitive with the machine which takes shorter to design, and is implemented in more recent and more competitive technology, it has to be projected for a higher performance. Suppose it takes 2-3 years to design a RISC machine and 5-6 to design a CISC machine than, given the rapid improvement in processor performance each year, the CISC will have to be targeted for almost twice the performance of RISC. This is an additional strain on a design team of a CISC machine.

Given the fact that each of the processors that has been on the market since the beginning of the microelectronics revolution of the late seventies has lived a relatively short life, it is almost irrelevant if the ultimate performance for a given architecture will or will not be achieved. The driving factor seems to be the need to reach the market in the shortest possible time, delivering a competitive performance for the time frame of the processor's lifetime.

The consequence of this line of thought is the choice of the technology and design methodology which result in the shortest turn-around time and competitive performance. Therefore the use of Application Specific Integrated Circuit Technology - ASIC seems to be a logical choice.

IMPLEMENTATION

Today there are many "silicon foundries" offering services in design and processing of gate array type designs with extremely short turn-around times, which range from 2 to 10 weeks depending on the chip size (in terms of the number of gates). Most of them are using CMOS technology with competitive ground rules ranging from 1.5 microns and below. They use different methods of placing gates or transistors on the chip, with the support of the sophisticated tools for placement and wiring. This allows the design of chips with several hundred thousands of transistors, which translated into gates ranges from a couple of thousand gates to a hundred thousand gates or more. Their internal speed can be as fast as several hundred pico-seconds depending on the loading condition and the amount of power dissipation allowed by the gate. Using the rough figures of 12-15 levels of logic and 1 nS propagation delay per gate, it is obvious that the performance achievable is not to be underestimated. These chips are supported by high quality packages, which allow several hundred I/O pins and are able to handle simultaneous switching on many output pins at very high clock rates. The package is also capable of dissipating the power generated by these fast and dense chips, due partly to the use of low-power CMOS technology and partly to the improvements made in the development of high quality ceramic packages.

This processor has been architected for implementation on a single ASIC chip containing up to 50,000 gates, and imbedded static memory arrays which are to be traded for the available gates. The goal of the architecture is to achieve high performance on both fixed-point and floating-point operations, because its intended use is for intensive scientific computation. The main issue here is how and where to use the available gates in order to maximize the performance and flexibility. Also, due to the channelless architecture of the ASIC chip, extensive bussing and wire crossing are prohibited because they take chip space and decrease the number of gates available.

The chip is broken up in to five loosely coupled units: fixed-point, floating-point, instruction unit, memory controller and I/O unit. (Fig.1.) Processor memory consists of interleaved dynamic RAM where static column mode is used to fetch data contained within the same page. Memory can be two or four way interleaved, depending on the mode selected, and supported by the memory control unit. With the four-way interleaved memory in the static-column, mode the memory is expected to deliver a word every 20nS. The processor cycle time is also set for 20nS where the fixed-point unit is able to execute one instruction every cycle, while it takes 3 cycles to execute basic floating-point operation (add/subtract or multiply).

The floating-point processor is able to peak at 33 MFP (mega flops) by executing two instructions (add and multiply) in the same 60nS cycle, while the fixed-point should achieve 50 MIPS (mega instructions per second).

The instruction unit fetches instructions and dispatches them to the fixed and floating-point units. It contains an instruction buffer (I-cache) and it is capable of executing BRANCH instructions directly.

The I/O unit is byte-serial, operating at a 50 MHz rate. It transfers data directly to the memory in a DMA fashion and operates in a packet mode.

The memory controller directs the transfer of data and instructions to and from the memory, assigns priorities and provides queuing. It is also responsible for generating refresh and timing signals for the dynamic memory.

ARCHITECTURAL GOALS

The main drawback of RISC processors available today is their relatively poor performance on floating-point computation. This is especially emphasized, because due to their high performance they belong to the range of scientific work stations where intensive floating-point computation is often required. To remedy their deficiencies on the floating-point performance, some of them resort to the use of commercially available floating-point processors. This has several disadvantages:

- First, the floating-point performance of many "off the shelf" floating-point units is relatively poor, and this diminishes much of the high performance achieved by

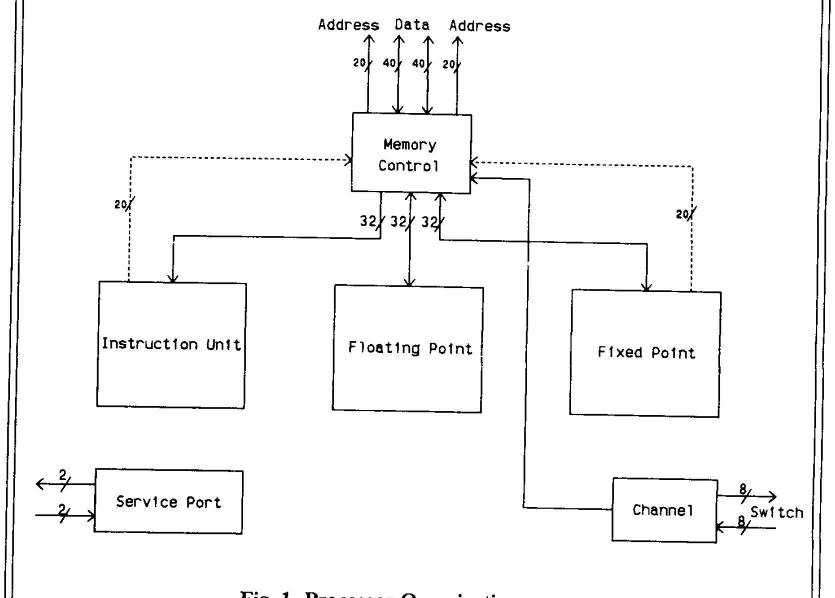


Fig. 1. Processor Organization

the processor. This situation has somehow been changed by the availability of the new generation Weitek floating-point chip [6].

- Second, with a sufficient performance floating-point unit, much of the performance is lost in the fixed-point to floating-point communication, dispatching the instructions and transferring data back and forth, as well as synchronizing the operation of fixed and floating-point units. This situation is hard to remedy because the two processors are not specifically designed to work together. Such a combination is lacking a well conceived communication protocol and supporting busses.
- Third, there are unavoidable off-chip crossing penalties. Especially if the "off-the-shelf" floating-point unit consists of more than one chip.

The main objective of this architecture is to provide an integrated fixed/floating-point unit with the communication protocol built into the instruction set and control. In addition the two units should be of comparable speeds of execution in order to avoid necessary synchronization delays.

Therefore, the floating-point unit contains a fast hardware multiplier and a floating-point adder of the size of 64 bits. The size of the multiplier is limited to 32 X 32 bits, due to the constraint of the available gates. The 64-bit size of the adder allows fast execution of double precision operations, and also pipelining of the multiplication and addition operations, so that it is possible to have two operations concurrently executed in the multiplier and the adder unit.

The hardware multiplier is used for the division operation by applying a multiplicative division algorithm (IBM 360/91) and using a table look-up for the first iteration.

Both the multiplier and adder are of a larger size than required by the IEEE-754 standard to which the operand conform in their formats. The multiplier size is 32 X 32 bits for the fractions, instead of 24 bits, and the adder size is 64 bits, instead of 52. This is done for two reasons. The first is to use the multiplier for the integer multiplication and division, and the second to postpone rounding in the sequence of iterative multiply-add operations, as will be explained later.

The operands are passed between the fixed-point and floating-point processors through the buffer stage registers, under the control of the transfer instruction.
The fixed-point unit is used for the address computation for the floating-point during the course of floating-point intensive computation. It executes in a 20 nS cycle and is capable of achieving a 50 MIPS peak. It contains a set of powerful instruction for the bit manipulations like Insert, Mask, Extract and Merge, which are suitable for signal processing applications. All the instructions are register-to-register in accordance with the RISC principles, operating on 32 fixed-point registers, which are 32-bits long. The shift left and right operation is designed to extend the sign bit on the left, or zeroes or ones on the right.

In order to keep the design simple, there is no provision for a cache. To make up for the absence of a cache, an instruction buffer is extended to 32 registers capable of containing 32 instructions. A sophisticated branch handling is provided for keeping branch target instructions or branching from the buffer, in case the target instruction is contained in the instruction prefetch buffer.
On the data side, the absence of a cache is supplemented by interleaving and by using a static column mode if inexpensive dynamic RAM is used for memory. It is also possible to switch the memory controller into the mode in which fast static RAM is used, thus eliminating the need for cache.

INSTRUCTION SET DESIGN

The instruction set follows the principles of RISC architecture. All the operations are performed on the registers, and the transfer of data between the processor and memory is through the Load and Store instructions. This applies to both fixed-point and floating-point processors, as well as the operations which are performed on the data found in both fixed-point and floating-point registers. In order to perform the operation, the data must first be transferred into the register file of the processor which is to perform the operation (fixed-point or floating-point). This is done using a special "transfer" instruction, the purpose of which is to transfer data between fixed-point and floating-point register files.

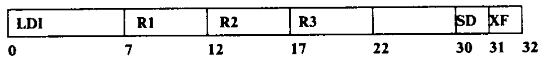
The instructions are chosen so that the instruction set represents a small set of carefully selected instructions which are simple to implement and which execute in one cycle.

Instruction Types

The instructions fall into the following categories:

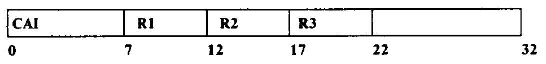
1. Load and Store Instructions (fixed and floating-point)

LDI R1,R2,R3 Load Indexed: R1 $\leftarrow$ (R2 + R3)



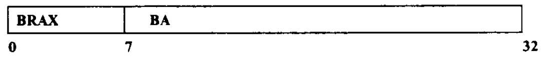
2. Address Computation (fixed-point only)

CAI R1,R2,R3 Address computation indexed : R1 $\leftarrow$ R2 + R3



3. Branch with or without executing the next instruction and link

BRAX BA Branch Absolute with execute: Execute next instruction, IAR $\leftarrow$ sign extended BA



4. Conditional Branches with and without link

BCNI CN , BI Branch on condition bit specified by CN in the condition code register CR. upd.IAR ← sign extended BI + IAR
Not condition is included in the condition code CN.

BCNI	CN	BI	
0	7	12	32

5. Bit Manipulations (fixed-point only)

MRG R1,R2,R3,PS,SZ : Merge SZ of R2 and PS from left and right of R3 into R1

MRG	R1	R2	R3	PS	SZ	
0	7	12	17	22	27	32

6. Shift (fixed-point only)

SHR R1,R2,R3,S : R1 ← R2 shifted right for R3 position. If S=1 sign extended if S=0 zero extended.

SHR	R1	R2	R3		S	
0	7	12	17	22	31	32

7. Arithmetic Instructions

ADD R1,R2,R3 Add: R1 ← R2 + R3

ADD	R1	R2	R3		IF	SD	XF	
0	7	12	17	22	29	30	31	32

8. Logical Operations

AND R1, R2, R3 AND: R1 ← R2 .AND. R3

AND	R1	R2	R3		
0	7	12	17	22	32

9. Transfer Instructions

TFFL Transfer Fixed Point Register R2 into Floating Point Register F1. (32-bit quantities)

TFFL	F1	R2		
0	7	12	17	32

11. Multiply fixed, floating or integer operands

MLT F1,F2,F3 : Multiply F1 ← F2 X F3

MLT	F1	F2	F3		IF	SD		
0	7	12	17	22	29	30	31	32

12. I/O instructions

Whenever the implementation seems to complicate the hardware and add complexity, the decision is made to emulate the function in software. This applies to both fixed and floating-point units.

Most of the fixed-point and floating-point instructions execute in a single cycle, where the cycle time for the fixed-point processor is 20 nS, while floating-point processor cycle is 60 nS (3 basic cycles). The reason for a longer floating-point cycle is simply because it takes longer to perform a basic floating-point operation, and 3 basic cycles are the minimum time required. Exceptions are interrupts and subroutine calls which take several cycles due to the need of saving the complete processor status.

There are 32 registers in the fixed-point unit and 32 registers in the floating-point, organized as 32 32-bit registers or 16 64-bit registers. Therefore we need 5-bits for the register address in the instruction. The target register of every instruction is named explicitly, so that each operation is of the R1 , R2 type where the result is destined for R3. Exceptions are instructions for monadic operation. With 15 bits used for addressing the registers and 7 for the opcode, the instruction lengths of 24 and 40 bits would provide a better utilization of memory and code space. However, this requires a substantial overhead for demultiplexing the instructions from the Instruction Buffer, checking if the entire instruction is contained in the Buffer, and if it is not, checking if the page crossing is involved. Therefore our decision is to use a single instruction length of 32-bits for the sake of simplicity. The extra available bits are used conveniently for decoding the information about the type of the operands and operation.

SYSTEM ORGANIZATION

The main components of the processor are shown on Fig. 1. All data paths between the memory controller and the processing units are 32-bits wide. The fixed-point unit is responsible for address generation for the floating-point unit, and the instruction unit is capable of handling branch instructions. The address inputs to the memory controller originate from the instruction and fixed-point units. Instructions and data are dispatched from the memory controller to the instruction unit, fixed and floating-point processors. The channel unit is connected directly to the memory controller, since all the I/O communication is performed in a DMA fashion.

Fixed Point Unit - FXU

The fixed-point unit (Fig.2.) operates in a 20 nS cycle and all the instructions are executed in one cycle. Exceptions are interrupts and system control instructions.

The fixed-point unit contains a register file of 32 registers 32-bit long, organized as a two-port read - one-port write, with the nominal access time of 9.2 nS. The outputs from the register file are staged in two registers with imbedded input multiplexers, A and B. The input to the staging registers comes from various sources: the constant register supplying constants, ALU, link register, loop counter, shifter, leading zero count, and mask and merge unit.

The exchange of data between fixed-point and floating-point units is through two staging registers containing the 32-bit data and 5 address bits of the destination register in the fixed-point / floating-point register file. The transfer instruction first reads data from the register file and then place it into one of the staging registers, together with a 5 bit destination tag. In the next cycle, data is written from the staging register into the register file of the destination unit, assuming there is no contention to write into the same location.

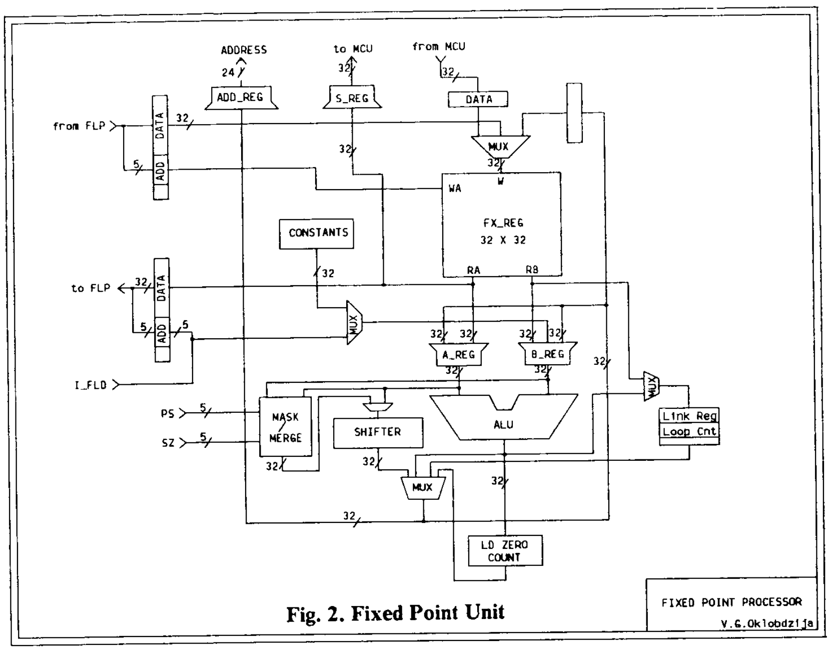


Fig. 2. Fixed Point Unit

The shifter unit performs a shift for any number of bit positions (1 to 31) left or right with the variations on the leftmost / rightmost positions. When shifting right, the rightmost bits are either sign extended or filled up with zeroes. In case of a left shift, the leftmost side is filled up with either ones or zeroes. This is dependent on the S bit (in position 31) of the shift instruction. We did not find any strong reason for implementing a rotate feature and, given the overhead for wiring complexity, decided against it.

The Mask and Merge unit can extract any number of bits from one operand and place or merge it into any position of the other operand. This is found to be a very powerful feature for bit manipulation. This operation uses the shifter unit, however the operation is performed in parallel with the mask generation. Therefore, it is possible to accomplish the operation in a single 20 nS cycle.

The Leading Zero Count is taken at the output of the ALU, and the result is placed into the staging register or written back into the register file.

The Link Register is used to save the return address after the Branch and Link instruction is executed.

The Loop Count Register is used to store the loop count during the execution of a loop terminated with the Branch on Condition instruction, where the exiting condition can be a predetermined number of iterations or the specific condition - whichever occurs first.

Between the MCU and fixed-point units, there are three staging register. Two are for data to and from the MCU and one is for the address generated by the fixed-point unit.

Floating-Point Unit - FPU

The Floating-Point unit operates on a 60 nS cycle - three 20 nS clocks (Fig.3.). The operand formats are consistent with the IEEE-754 standards single and double precision formats, in the length of the mantissa end exponent parts. However, since it does not comply with all the prescribed modes of operation required by the standard, it is therefore not compatible. In addition to the floating-point numbers, it operates on the integers of the lengths of 32 and 64 bits.

It contains a hardware multiplier, 32 by 32-bits in size and a floating-point adder of the size of 64 bits for the mantissa.

The exponent circuitry in both the multiplier and the adder are of 12-bit size. The size of the multiplier is determined by the number of available gates and the chip area. It is the multiplier that determines the basic cycle of the floating-point unit.

The register file contains 32 registers of 32-bits. It has two 32-bit read-ports and one 64-bit write-port, made as a combination of two 32-bit write ports. In this way it is possible to write into the register quantities of both 32 and 64 bit size. Single precision operands are read simultaneously from the register file and fed into the multiplier or upper portion of the 64-bit staging registers of the floating-point adder. The double precision operands are read as a register pair, one at the time, and stored into the staging registers of the floating-point adder. Reading of the register file is accomplished on the 20 nS boundaries.

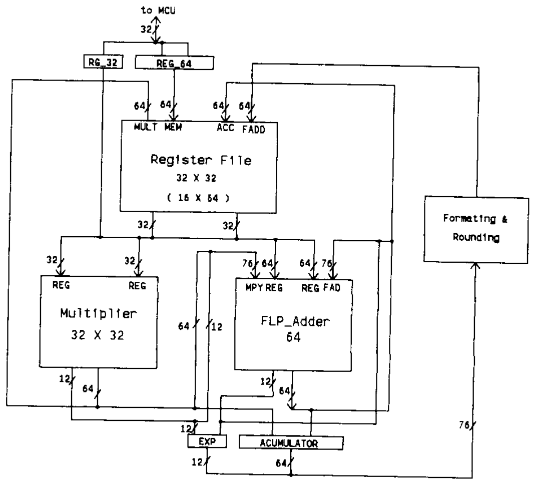


Fig. 3. Floating Point Unit

There is a path between the floating-point multiplier and floating-point adder, so that the product is fed directly to the floating-point adder. The Multiply-Add operation is pipelined so that both units (adder and multiplier) are executing an instruction in a 60 nS cycle. On some problems like matrix multiplication (single precision) the processor can achieve a 33 MFP (mega-flops) peak. A sustained operation rate depends on the code, and it can be anywhere between 10 and 24 MFPs.

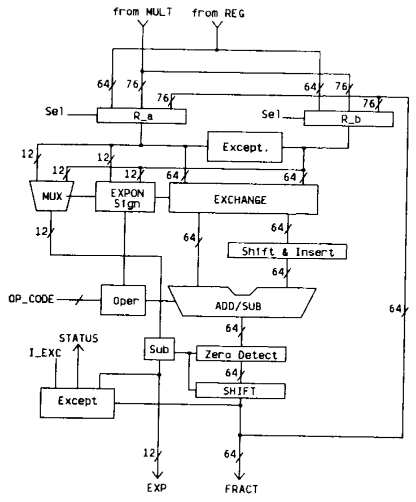


Fig. 4. Floating Point Adder

The operands can be passed between the multiplier and the adder before being stored into the register file. Before the result is written into the register file, it is passed through the Formatting and Rounding Unit which will perform a rounding operation as prescribed by the standard. It will format the operand into a single or double word, depending on the type of the operand to be written into the register file. The Formatting Unit will signal an exception, if one occurs. Not all of the exception conditions and formats as prescribed by the standard are implemented.

Floating-Point Adder

The Floating Point Adder is shown on Fig. 4. Due to the pre-alignment and post-alignment of the operands, including the adjustment of the exponents, achieving 60 nS operation is not an easy task. The Adder (Subtracter) is implemented as a combination of a conditional-sum and variable optimal length carry-skip scheme [7]. By combining a sophisticated scheme and a design optimized for speed, the 64-bit addition is performed in 15.9 nS nominal time.

Floating-Point Multiplier

The Floating-Point Multiplier is shown in Fig. 5. Its main components are an integer multiplier, an exponent adder and a normalizer used for post-normalization of a product. When operating in the integer mode the exponent unit is ignored. The product is 64-bits long and it is fed directly to the floating-point adder as one operand. The inputs to the multiplier are staged in a combination of first-stage L1 and second-stage L2, L3 latches, so that the next set of operands is ready in L1, while the multiplication is in progress on the outputs of either L2 or L3, selected by the multiplexer.

Multiplication of double length integers (floating-point numbers) is performed by performing 3 add and 4 multiply, operations using a multiply-add combination. This operation takes four 60 nS cycles.

Memory Control Unit - MCU

The processor's addressing capability is a maximum of 1 MWord, 32 bit words with 20 nS access time.

Memory access is requested by four sources. These are the channel unit, the floating-point unit, the fixed-point unit, and the instruction unit. The priority assigned to them is in a decreasing order of priority, as listed (i.e. the channel unit has the highest priority).

The channel unit and the instruction unit are issue requests to transfer blocks of data, up to 32 words long. These words must be delivered in order. The requests include the starting memory address and the length of the block.

The floating-point unit and the fixed-point unit request transfers of a single memory word at the time. As a consequence, vector transfers are presented to the memory controller as a sequence of single word transfers. The advantage of doing it this way is that these requests may be fulfilled out of order, though they need to be tagged for such purposes. Since the destination of these requests is one of the 32 registers within either the floating-point or fixed-point unit, the corresponding register number is used as the tag. Thus, the request includes the memory address and the register number.

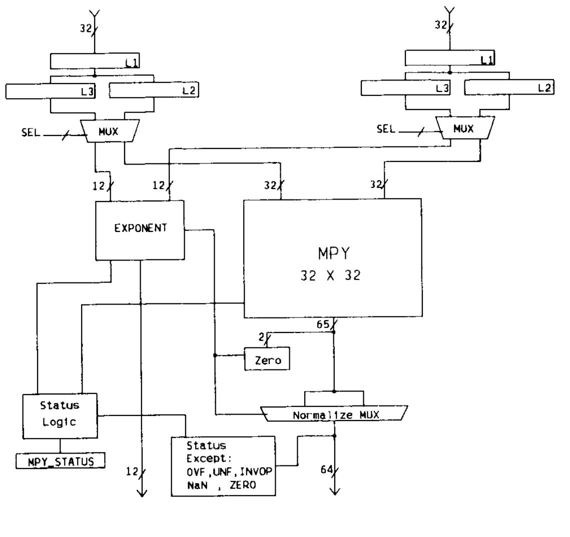


Fig. 5. Floating Point Multiplier

The Memory Controller operates in one of two modes, supporting the use of either dynamic or static memory devices. In either mode, the sources of memory requests compete for access to the MCU. Their conflicts are resolved using the priorities indicated before. The unit with the highest priority is enabled to present its request to the MCU. These modes are:

Use of Dynamic Memory

In this mode, the memory system is 4-way interleaved, with 256 KWords of dynamic memory per bank, using static-column (page-mode) devices. 1 Mbit dynamic memory devices are used, organized as 512 rows of 512 by 4 bits. These are static-column RAM devices, with 80 nS access time to locations within the same row. Each memory bank has its own independent address bus. Each of these busses includes 9 address lines for multiplexed Row Address and Column Address, plus 3 control signals. In this mode, the memory address presented to the MCU is used to identify the required memory bank, and the status of such a bank is checked. If the bank is available, the request is acknowledged and processed. The requester, after being enabled to present its request to the MCU, expects an acknowledgement signal. If that signal is not received during the next clock cycle, it implies that the memory bank needed is not available. The source of the request must again present it to the MCU.

To increase the utilization of the memory system in this mode, each one of the memory banks has a queue of pending requests. Therefore, the acceptance of a request is actually subject to the availability of space in the corresponding queue. As a consequence, and depending on the status of the different queues, requests may be serviced out of order, as it has been already stated.

Use of Static Memory Devices

In this mode, memory is treated as a single memory system with the a 32-bit address bus. It is designed for 20 nS access time static memory devices. The 32 address lines are obtained by grouping the four sets of 9 address lines described above into a single bus (with 4 unused lines). In this mode, the address received from the requester is placed in the proper output pins. No further processing of the request is performed.

An additional operating mode with separate address spaces for instructions and data (Harward Architecture) can also be supported, but requires modifications of the MCU logic.

The Memory Controller performs the following functions:

- data transfers
- 4-way interleaved addressing
- selection of highest priority request and management of request priorities
- support for the static-column mode allowing fast access to the locations within the same row
- generation of timing and memory refresh signals
- single error correction, double error detection

The memory controller consists of two subsystems:

- A subsystem to accept and process requests. It includes the logic to select requests according to their priorities, the logic to process those requests, the queues and the logic to implement the dynamic memory refresh schedule.
- A subsystem to access the memory devices. It includes the logic to generate the control signals to the memory devices, the error correction logic, and the required data interfaces.

Single error correction and double error detection are accomplished using a modified Hamming code. Seven parity bits are computed from the 32 bits of data, and stored together with it.

Implementation consists of two internally connected data busses to reduce the timing requirements imposed by the 50 Mhz clock.

Instruction Unit

The Instruction Unit IU receives the instructions from the MCU, eight at a time, in order to take advantage of the fact that they are likely to be on the same page, since page crossing penalties are costly. They are stored in the Instruction Buffer (IB) whith a maximal capacity of 32 instructions. The Instruction Buffer is organized as a 32-bit register file containing 32 registers and two read and one write port. The reason for two read ports is so that two instructions can be decoded at the same time, and if no conflict is found, dispatched and executed. These two instructions can be one fixed and one floating-point operation, or fixed (floating) point operation and fixed (floating) point load (store). Branches are also handled by the instruction unit. The instructions are partially decoded, facilitated by their formats, and they are forwarded to the corresponding fixed (floating) point units to be further decoded and executed.

Four target instructions of the most recently taken branches with the three following instructions (total of four per branch) are kept in a separate register file TIR (Target Instruction Register), containing 16 32-bit registers. This contains two read-ports which are connected to the instruction decode registers through the multiplexers so that they can be easily swapped if the branch target is found to be contained in TIR. The write-port of TIR is 64-bitd wide, allowing the ability to write two instructions at the same time, following the most recently encountered branch instruction. The replacement policy for the branch target instructions is "the last recently encountered". This is done by keeping circular pointers pointing to the last location which was written, and therefore replacing the four target instructions which were the last written in the buffer. The target instructions that are found in the buffer are not affected i.e. they are not written back and their position in the queue is not changed. There are four pointers to the TIR, each having an instruction address appended to it for comparison purposes.

The fixed and floating-point status bits are passed to the IU.

The partially decoded instructions are forwarded to the control units of FPU and FXU for further decode and execution. Since the IU takes part in instruction handling and execution, the control units of FXU and FPU are made simpler.

I■O Unit

The I/O unit operates in a byte-serial mode between the memory and the I/O device. It is a 50 MHz byte-serial packet consisting of a header with the information of the packet length (in the number of bytes) and the destination. The last byte is a CRC code. When the packet is sent to the processor it interrupts the operation and it is transferred to the memory having the highest priority given by the MCU.

The outgoing transfer is initiated by the MCU, which supplies the starting and ending address in the memory and the destination address.

IMPLEMENTATION TRADE-OFFS

Currently the largest available ASIC chip allows packing of up to 70,000 gates. This number is very much dependent on the regularity of the structure, the amount of imbedded memory arrays, the size and number of busses implemented on the chip etc. There are two versions of logic cells available for the designer: a fast and slow version. The fast version consumes more power and has higher gate count (area) than the slow version and should be used very judiciously. Therefore, the fast cells are used only in the time-critical paths, where the speed-up is absolutely necessary in order to meet the 20nS cycle time. For the rest of the logic, we use the standard "slow" version.

We identified the cells having a low gate count and short propagation time, and used them extensively throughout the design. One of them is a multiplexer cell, which due to its pass-transistor implementation yields very low gate count and propagation delay. This turns out to be advantageous, since the multiplexer is a very common and versatile building block in any computer hardware. In addition, we attempted to identify and create new building blocks.

The fixed-point unit ALU and floating-point adder are built using a variable-length carry propagation scheme [7]. The advantage of this scheme for VLSI implementation is that the fan-in and fan-outs are relatively small (2-3). The wiring overhead is minimized since the carry-bypass circuitry is minimal and only one wire is used for bypassing the carry signal. This scheme in a small and regular structure with a relatively small gate count. This scheme is actually comparable in speed with the Carry-Lookahead scheme as shown by simulation [7]. Yet, it implements itself in a small and regular structure.

We used 513 gates for the 32-bit ALU, which resulted in 16.7 nS nominal time for the critical path.

The multiplier implemented in the FPU is also optimized for speed by a careful selection of the mixture of the fast and standard gates. Booth's encoding is applied in order to minimize the number of partial-product terms, which are reduced to two operands by the network of 3-2 counters (implemented as full adder cells). The final addition of 64-bits takes 16.1 nS nominal. The placement of the cells is critical in order to achieve a compact and fast implementation. The result of the multiplier is an operand of a 64-bit adder which facilitates multiply-add operation. By keeping the product 64-bits long, we introduced extra precision, which allows us several passes through the multiply-add operation without significant loss of precision.

All the modules are designed in a strictly hierarchical fashion, always attempting to identify and use common building blocks. This makes repetition possible and reduces the design time and effort. Maintaining hierarchy is of outmost importance in order to manage complexity and to achieve short design time.

CONCLUSION

The basic principles of RISC architecture are applied to design a processor in fast turnaround ASIC technology. The architecture is a trade-off between the features desirable in order to make the processor attractive for a wide range of computational problems, and the possibilities of available technology. We feel that simplicity is a very important and often overlooked. The benefit of a shorter design cycle offers the ability to reach the market place with competitive performance in a timely fashion.

ACKNOWLEDGMENT

Monty Denneau instigated this work and provided the key ideas. Jaime Moreno contributed to the memory control unit. Greg Grohowski lent his expertise on high-performance processor architecture during many pleasant discussions.

REFERENCES

[1] George Radin, *The 801 Minicomputer,* IBM T.J.Watson Research Report RC 9125, November 11, 1981.

[2] J.L.Hennessy, *VLSI Processor Architecture* , IEEE Transaction on Computers, Vol. C-33, No.12, December 1984.

[3] C.Rowen et al. *RISC VLSI Design for System-Level Performance* VLSI System Design, March 1984.

[4] *AMD 2900 User's Manual,* Advanced Micro Devices 1987.

[5] D.A. Patterson, C.H. Sequin, *RISC I: A Reduced Instruction Set VLSI Computer* , Proceedings of 8th Annual Symposium on Computer Architecture, Minneapolis, Minnesota, May 1981.

[6] *How Weitek Chips Run FORTRAN at 25 Megaflops* , Electronics, October 30, 1986.

[7] V.G.Oklobdzija, E.R.Barnes, *Some Optimal Schemes for ALU Implementation in VLSI Technology* , Proceedings of 7th Symposium on Computer Arithmetic, June 4-6, 1985, University of Illinois, Urbana, Illinois.

[8] V.G.Oklobdzija, E.R.Barnes, "*Simple and Efficient Scheme for VLSI Implementation of Addition*", Submitted to the special issue of Journal of Parallel Processing and Distributed Computing, April 1988.