ARCHITECTURAL STUDY FOR AN INTEGRATED FIXED AND FLOATING-POINT VLSI-ASIC PROCESSOR

V. G. Oklobdzija, Greg Grohosky

IBM T. J. Watson Research Center
P.O.Box 218
Yorktown Heights, NY 10598

ABSTRACT

The architecture of a single-chip processor with integrated fixed and floating point execution units is presented. A single-chip implementation is enabled by current ASIC technology offering well in excess of 50,000 gates per chip and delays on the order of 600 pS per gate [16]. The basic principles of RISC architecture are used and a fast floating-point processor is architected as an integral part of the chip. The architecture is intended to make the processor attractive for a wide range of scientific computing implemented as a part of a special purpose machine. The architecture takes advantage of the high level of integration and low power consumption of CMOS technology. The integration of the execution units and an efficient inter-unit communication protocol avoid off-chip delay penalties associated with comparable implementations which use several chips.

INTRODUCTION

One of the principles of RISC architecture is to trade complexity for the speed of execution [1],[2],[3],[4]. Speed is achieved by designing a small, simple instruction set which executes on simple hardware. The simplicity of the hardware allows for fast instruction execution.

Given such high performance one expects to find RISC machines used for solving a variety of problems including intensive floating-point computation in such applications as scientific computation. However, the floating-point performance of RISC machines has not been impressive [9],[10],[11]. This is caused by the fact that floating-point performance has not been emphasized in current RISC architectures. To improve floating-point performance RISC machines use a variety of "off the shelf" processors. These processors are either not able to match the fixed point performance or much of their power is lost in inadequate communication between the fixed and floating point units. The performance loss due to communication is a consequence of the communication protocol being adapted rather than being an integral part of the processor design.

Our goal is to design a processor with a simple architecture which allows for rapid implementation and balanced fixed and floating point performance in a wide range of computation.

The capability of currently available ASIC technology allows for the integration of complex designs on a single chip. ASIC technology also offers a shorter design cycle than a full custom VLSI design. This means that an ASIC-based RISC product can reach the market faster than a custom-designed CISC processor. Thus, an ASIC RISC processor is a logical choice.

MOTIVATION

Technology

Today many "silicon foundries" offer services to design and process gate arrays with turnaround times ranging from 2 to 10 weeks. Most of them use CMOS technology with ground rules of less than 1.5 microns. A 15 mm by 15 mm chip contains several hundred thousand transistors; these chips may contain from a few thousand gates to a hundred thousand gates or more. These gates can switch in several hundred pico-seconds depending on their loading and power dissipation. Using figures of 12-15 levels of logic per cycle and 1 nS propagation delay per level, cycle times of 20 nS are achievable.

High quality packages are available which contain several hundred I/O pins, can simultaneously switch many output pins at high clock rates, and can dissipate the power of fast, dense chips [15],[16].

Architecture and Organization

The main drawback of current-generation RISC processors is their relatively poor performance on floating-point computation. This is even more marked when one considers that due to their high performance they belong to the range of scientific work stations where intensive floating-point computation is 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 achievable by the processor. This situation has somewhat been changed by the availability of the new generation Weitek floating-point chips [12].
- Second, with a sufficiently high-performanc, floating-point unit, much of the potential performance is lost in fixed-point processor to floating-point processor communication. Instructions must be dispatched to both units, data must be transferred between the units, and the units must occasionally be synchronized to ensure proper program execution. Designers have few choices to improve the performance if the fixed and floating point processors are not specifically designed to work together. Such a combination lacks a well conceived communication protocol and supporting dataflow.
- Third, signals take a significant amount of time to propogate between chips relative to the delays of gates, and the second level of packaging introduces physical and electrical effects which cause the relative timing of signals to vary. The resulting delays and clock system skews are aggravated if the "off-the-shelf" floating-point unit consists of several chips.

The main objective of this architecture is to provide an integrated fixed/floating-point unit with a unified communication protocol. In addition the execution speeds of the two units should be comparable in order to avoid unnecessary synchronization delays. The processing units should be able to achieve as much parallelism as possible, i.e. they should be allowed to execute their instructions independently of each other most of the time. Maximizing their parallelism improves the performance by minimizing the impact of:

- a slow operation in one of the processing units on the other
- dispatching data and instructions
- synchronization

Also, by executing two (three) instructions in parallel.

First we will describe the proposed instruction set architecture of the chip. Then the logical organization of the chip will be described with emphasis on the coupled fixed and floating point execution units. The results of some initial studies on execution unit size and delay are also discussed.

INSTRUCTION SET DESIGN

The instruction set follows the principles of RISC architecture. All the operations are performed on data available in registers, and the transfer of data between the processors and memory is through the Load and Store instructions only. This applies to both fixed-point and floating-point processors.

"Transfer" instructions perform operations on data in either the fixed point or floating point register files and to transfer the result to the other unit.

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. 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.

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. With 15 bits used for addressing the registers and 7 for the opcode, 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 a page crossing is involved. Therefore our decision is to use a single instruction length of 32 bits for the sake of simplicity. The remaining bits are used conveniently for decoding the information about the type of the operands and operation.

Instruction Types

The instructions belong to the following categories:

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

LDI R1,R2,R3
Load Indexed: R1 <- M[R2 + R3]

LDI	R1	R2	R3		SD	XF	
0	7	12	17	22	30	31	32

XF = 0/1 Load Fixed/Floating point register
SD = 0/1 Single/Double Word load (FP only)

2. Address Computation (fixed-point only)

CAI R1,R2,R3
Address computation indexed : R1 <- R2 + R3

CAI	R1	R2	R3		
0	7	12	17	22	32

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

BRAX BA
Branch Absolute with execute:
Execute next instruction,
IAR <- sign extended BA

BRAX	BA	
0	7	32

4. Conditional Branches with and without link

BCNI CN , BI
Branch on condition bit specified by CN in the condition code register
Not condition is included in the condition code CN.
updated IAR <- sign extended (BI + IAR)

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

IF = 0/1 Integer/Floating Add (FP only)

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

PROCESSOR ORGANIZATION

The main components of the processor are shown in Fig. 1. The chip is divided into five loosely coupled units: fixed-point, floating-point, instruction unit, memory controller, and I/O unit [6].
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.

The processor cycle time is set for 20nS. The fixed-point unit is able to execute one instruction every cycle. It can therefore achieve a 50 MIPS peak rate. It takes 3 cycles to execute a basic floating-point operation (add/subtract or multiply).

In order to keep the design simple, there is no provision for a cache. Sophisticated branch handling is provided for by keeping branch target instructions in a branch target buffer, and

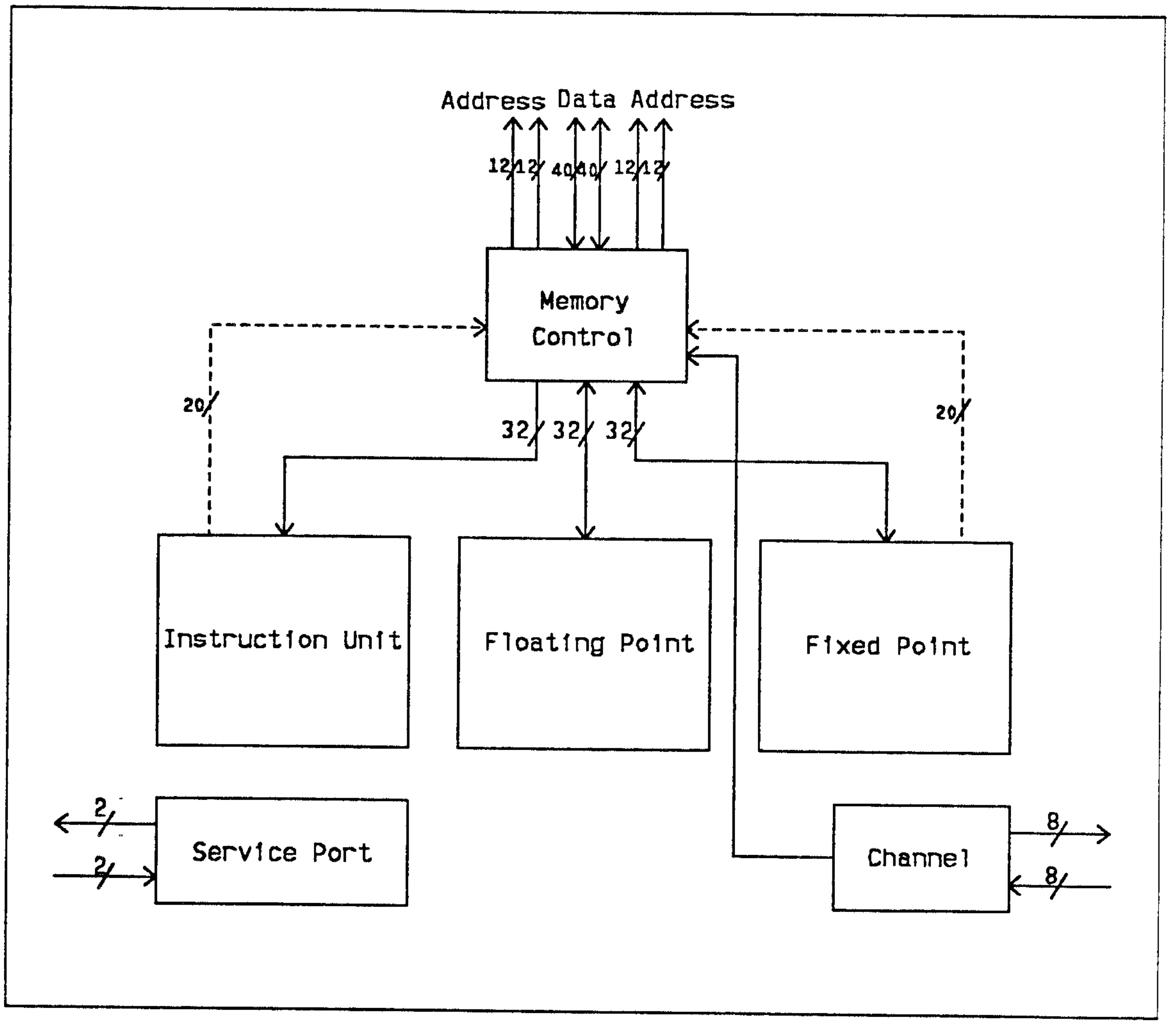

Figure 1. Processor Organization

by branching within the confines of the instruction buffer, in case the target instruction is contained in the instruction prefetch buffer.

For data references, the absence of a cache is mitigated by interleaving processor memory. If fast static RAM is used, a cache is unnecessary for our intended application. If slower dynamic RAM with static column mode is used, the interleaving still permits the memory to deliver a word every 20 nS for accesses within the same memory page.

Organization of Processing Units

Fixed Point Unit - FXU

The fixed-point unit (Fig. 2.) operates in a 20 nS cycle. Except for system control instructions, all instructions execute in one cycle.

The fixed-point unit contains a register file of 32 32-bit registers, organized as a two-port read, one-port write register file. It has a 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, the ALU, the link register, the loop counter, the shifter, the leading zero count, and the mask and merge unit.

Data is exchanged between the fixed-point and floating-point units via two staging registers. Each contains 32 bits of 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 of the source unit and then places it into one of the two staging registers, together with a 5 bit destination tag. During the next cycle, data is written from the staging register into the register file of the destination unit. There is no contention to write into the same location from any other pending operation since this transfer must be synchronized.

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 a powerful feature for bit manipulation. This operation uses the shifter unit while the mask is being generated. 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. The operation of these registers is described in the instruction unit section.

Floating-Point Unit - FPU

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

It contains a 32 by 32 bit hardware multiplier, and a 64 bit floating-point adder for the mantissa. The exponent circuitry in both the multiplier and the adder are 12 bits wide. The

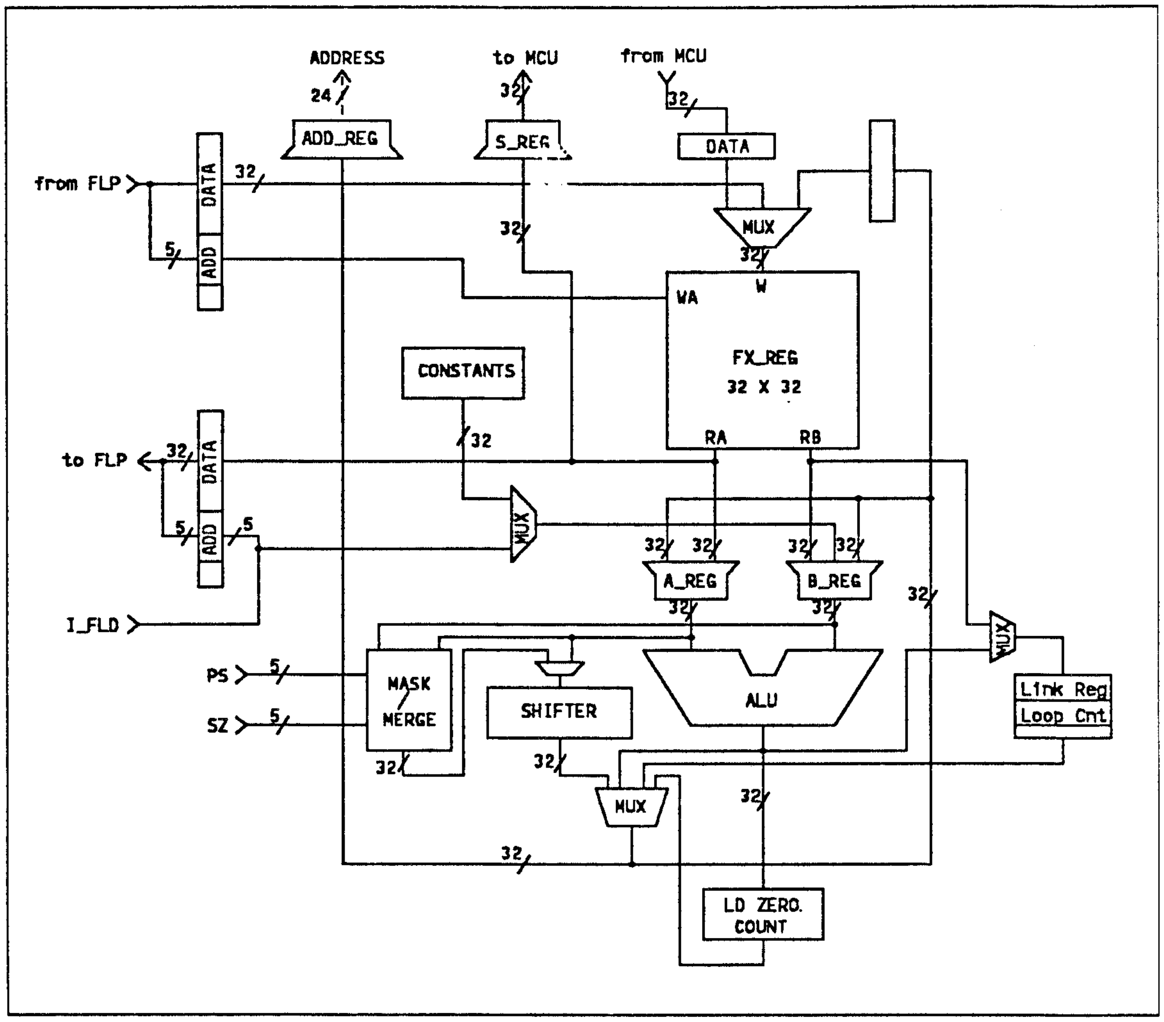

Figure 2. Fixed Point Unit

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 both 32 and 64 bit quantities into the register file. 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 20 nS boundaries.

There is a path between the floating-point multiplier and floating-point adder, so the product may be fed directly to the floating-point adder. This 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 single-precision matrix multiplication the processor can achieve a 33 MFLOPS (mega-flops) peak. The sustained operation rate depends on the code, and it typically can be anywhere from 10 to 24 MFLOPS for scientific applications.

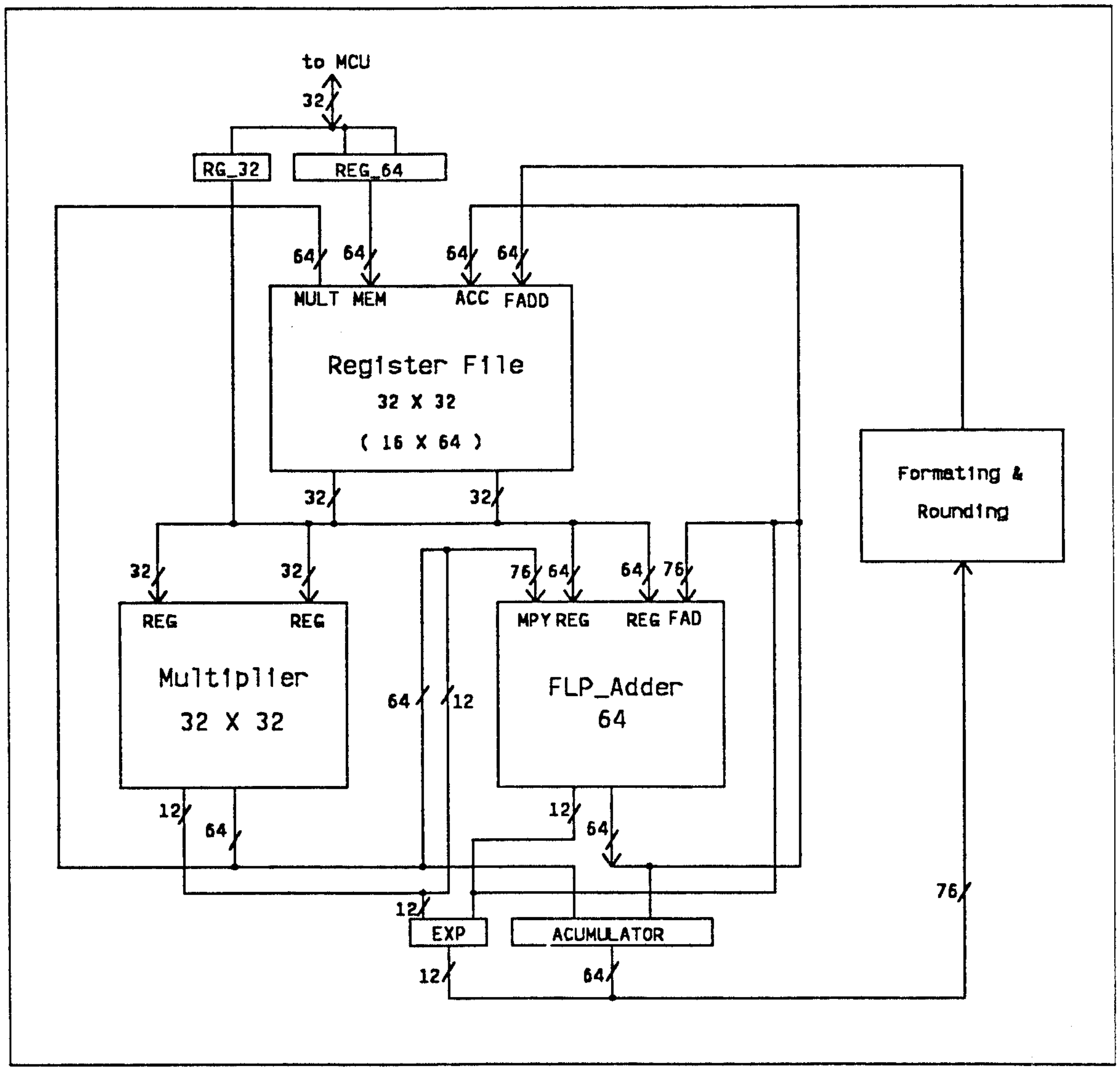

Figure 3. Floating Point Unit

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.

Floating-Point Adder

Th. Floating Point Adder is shown on Fig. 4. Due to the pre-alignment of the operands and post-normalization of the result, including the adjustment of the exponents, achieving 60 nS operation is not easy.

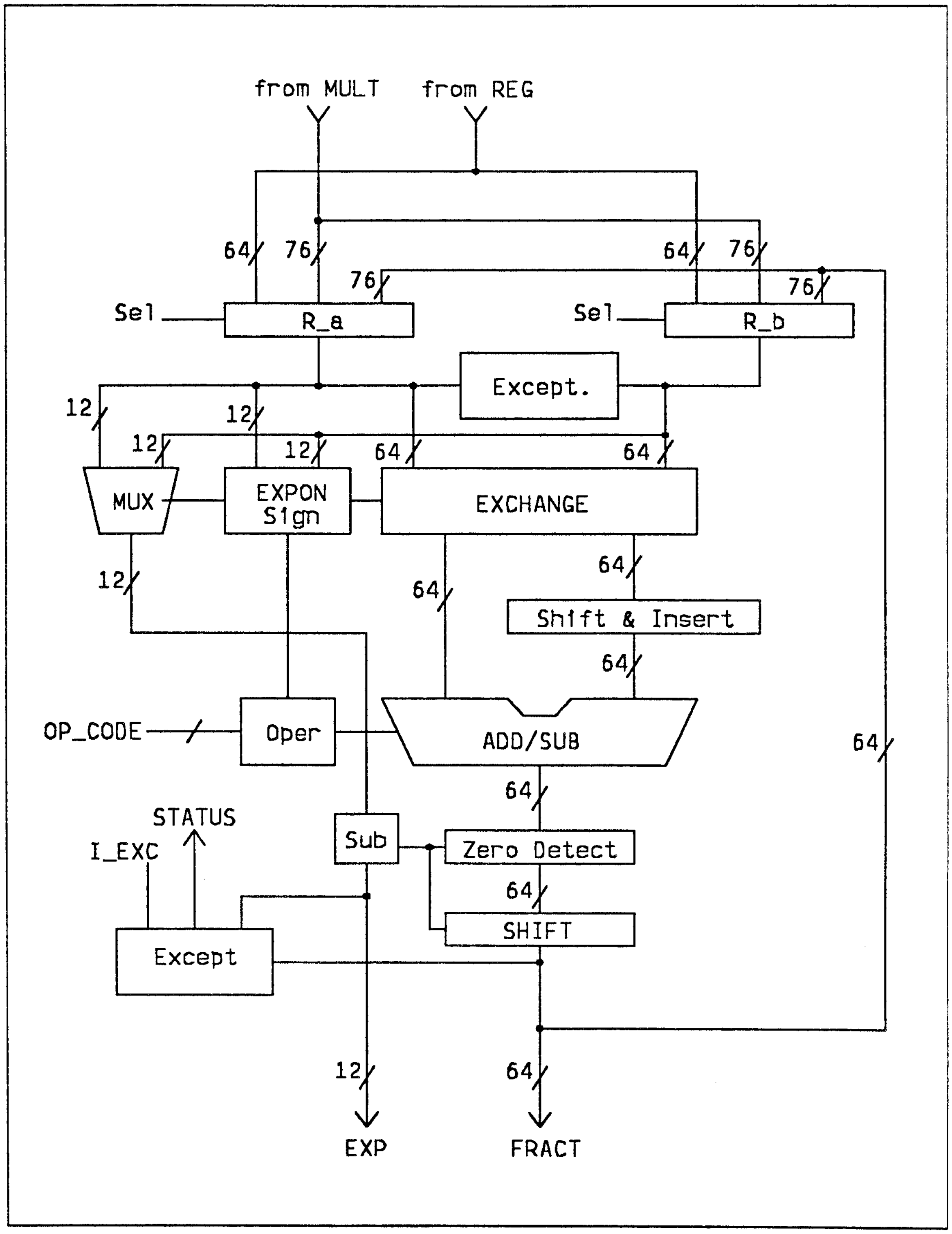

Figure 4. Floating Point Adder

The Adder (Subtractor) is implemented as a combination of a conditional-sum and variable optimal length carry-skip scheme [13]. By combining a sophisticated scheme and a design optimized for speed, a 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 and 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.

The multiplier is optimized for speed by a careful mixture of 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 a multiply-add operation without significant loss of precision.

Memory Control Unit - MCU

The processor's addressing capability is a maximum of 1 MWord (32-bit words) in 20 nS access time.

Memory access is requested by four sources. These are, in priority order, the channel unit, the floating-point unit, the fixed-point unit, and the instruction unit.

The channel unit and the instruction unit 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 a time. 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.

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

The Adder (Subtractor) is implemented as a combination of a conditional-sum and variable optimal length carry-skip scheme [13]. By combining a sophisticated scheme and a design optimized for speed, a 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 and 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.

The multiplier is optimized for speed by a careful mixture of 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 a multiply-add operation without significant loss of precision.

Memory Control Unit - MCU

The processor's addressing capability is a maximum of 1 MWord (32-bit words) in 20 nS access time.

Memory access is requested by four sources. These are, in priority order, the channel unit, the floating-point unit, the fixed-point unit, and the instruction unit.

The channel unit and the instruction unit 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 a time. 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.

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

In this mode, memory is treated as a single memory system with the 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 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 - IU

The Instruction Unit, Fig. 6., embodies the principles of decoupled architecture [5]. It receives the instructions from the MCU, four at a time, taking 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) with a maximal capacity of 16 instructions. The Instruction Buffer is organized as a 16 entry, 2 read port, 1 write port 32-bit wide register file. Two read ports are used to permit two instructions to be decoded at the same time, and if no conflict is found, dispatched and executed.

The fixed and floating-point condition status bits are passed to the IU, which performs branch condition evaluation and generates target addresses for branches.

These two instructions can be any combination of fixed and floating point instructions, including fixed (floating) loads and stores. The instructions are received by the pre-decode registers PDS1 and PDS2 and partial decoding is performed in the Pre Decode and Control

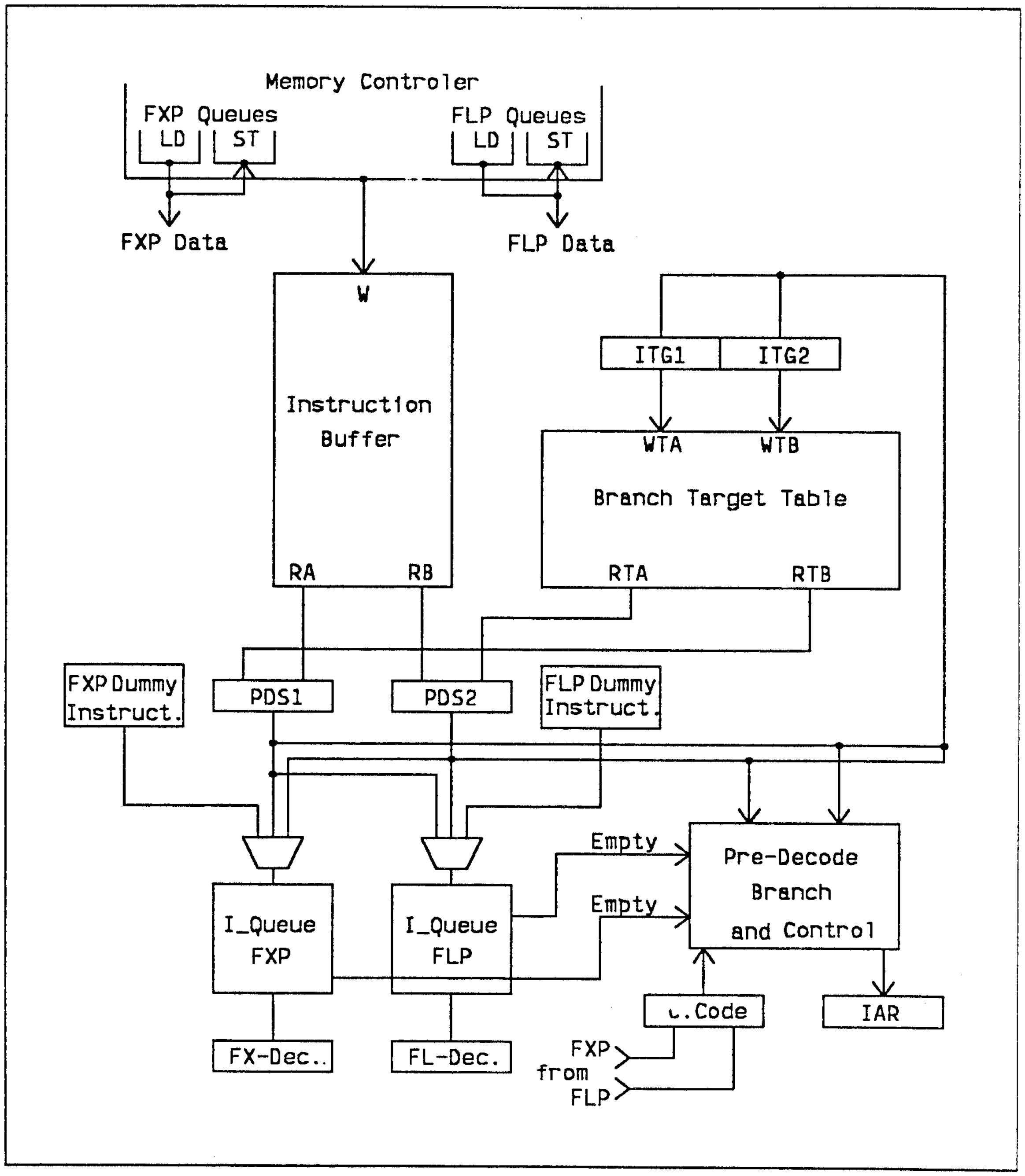

Figure 6. Instruction Unit

Unit (PDC). The PDC unit determines if the instruction is Load, Store or Branch and if it requires synchronization between fixed and floating point units.

Normally, Load and Store (fixed and floating) require the units to be synchronized to assure that one of the processors is not using "old" data, i.e. data that would be otherwise overwritten had the processors been in synch. Given the frequency of Load and Store instructions, about 30% of the instruction mix [1], it would practically be required that the processors run in a lockstep most of the time. Other solution would be to use rather complex hardware, which would keep track of the instrcution dependencies and correct sequencing of dependent instructions [7].

In many high-end commercial procesors (IBM 360/91, 3033, etc.) this problem is solved by using address comparison logic to compare the address of a load with addresses of stores from the floating point unit which have not been performed. This address comparison logic is used infrequently and consumes valuable area on a chip [7],[8].

Here our decision has been to allow the processors not to synchronize on Load and Store but not to allow them to share data through the memory. The only exchange of data between fixed and floating point processors can happen using the transfer instruction (TFFL, TFLF) passing the data between the general purpose register files of the corresponding units. In addition we introduce a SYNCH instruction whose sole purpose is to provide synchronization between the units.

We feel this is a reasonable approach given:

- the gain in parallelism having the units running concurrently
- that the attempt to use each others data is infrequent

Having the synchronization on conditional branch and transfer instructions only, the units are made less dependent and they are allowed to operate with higher degree of parallelism.

Floating Point Load

When a floating point load is detected at the pre-decode stage the following occurs:

- The load instruction is issued to the fixed point unit for address generation.
- A dummy Load (RECEIVE) instruction is issued to the floating point instruction queue (FLP-IQ), containing the destination register.
- Fixed point unit issues the load request to the memory controller by placing the request into the aforementioned memory request registers. The data from the requested memory location is then placed on the data queue inside the memory control unit.
- The RECEIVE instruction, when executed by the floating point unit, takes the data from the data queue and writes the data into the destination register.

Given that there are no out of sequence instructions, data need not to be tagged.

Floating point stores are performed in a similar way issuing dummy SEND instruction which place data on the FLP Store Data Queue. The destination address is generated by the FXP unit and tagged to the data in the store queue. Data can not be written to the memory by the MCU until both data and address are valid.

Branching

Fetching of instructions from the IB is suspended when the Branch instruction is encountered awaiting the generation of the Branch Target Address (BTA). In the case of Branch and Execute the instruction following the Branch is dispatched to the corresponding unit. The Branch Target Table (BTT) is associatively searched to determine if the branch target instruction is already in BTT. If a match is found, two instructions are placed in the pre-decode registers PDS1 and PDS2 with the target instruction in PDS1. The target instruction might be in the IB and if it is found in IB, fetching of instructions continues from IB. Mechanism for matching and keeping circular pointers are implemented to support this feature.

The BTT contains four target instructions, each followed by up to three subsequent instructions. The pointer to the BTI in the BTT is 20-bit BTA which has valid bit VF associated with it. The BTT is organized as a two-port read two-port write register file containing 16 instructions. It is possible to simultaneously read or write two instructions at a time in adjacent locations.

If the BTI is not found in IB or BTT, than the following occurs:

- The IB is cleared by resetting pointers.
- The BTA is generated and a request to fetch a block of instructions from that location is issued to the MCU.
- The Instructions are written into IB and passed through into the pre-decode stage.
- The BTI and the subsequent three instructions are pre-decoded and passed from the pre-decode stage into IQ of FXP and FLP units. They are also writen into the BTT. If another Branch is not encountered at the end of this process, the valid bit VF is set to 1.

When BTI is found in BTT, IU issues request to the MCU to fetch a block of four instructions starting with the memory location BTA+4, only if VF=1. A conditional branch is always treated as if it will not be taken (instructions following it are brought to IB).

The partially decoded instructions are forwarded to the IQs 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 ISSUES

Currently the largest available ASIC chip allows packing of up to 100,000 gates [15]. This number is highly dependent on the regularity of the structure, the amount of imbedded memory arrays, and the size and number of busses implemented on the chip. 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 multiplexor cell, which due to its pass-transistor implementation yields very low gate count and propagation delay. This turns out to be advantageous, since the multiplexor is a very common and versatile building block in any computer. 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 [13]. 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 [14]. Yet, it implements itself in a small and regular structure.

The fixed point ALU of 32 bits was implemented with 513 gates resulting in 16.7 nS nominal time for the critical path.

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. Simplicity is 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 many ideas. Jaime Moreno contributed to the memory control unit.

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] 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.

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

[5] J.E.Smith et al, *A Simulation Study of Decoupled Architecture Computers*, IEEE-TC, Vol. C-35, No. 8., August 1986.

[6] V.G.Oklobdzija, *Architecture for a Single-Chip ASIC Processor with Integrated Floating Point Unit*, 21st Hawaii International Conference on System Sciences, Kailua-Kona, Hawaii, January 5-8, 1988.

[7] R.M.Tomasulo, *An Efficient Algorithm for Exploiting Multiple Arithmetic Units*, IBM Journal of Research and Development, January 1967.

[8] D.W.Anderson at al, *The IBM System/360 Model 91: Machine Philosophy and Instruction-Handling*, IBM Journal of Research and Development, January 1967.

[9] *IBM-RT Personal Computer Technology*, Publication No. SA23-1057, IBM Corporation 1986.

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

[11] *CLIPPER, 32-bit Microprocessor, User's Manual*, Fairchild 1987.

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

[13] 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.

[14] 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.

[15] *LSI Packs 100K Gates On Chip*, Electronics Engineering Times, October 26, 1987.

[16] *LCA 100000 Compacted Array Series* , LSI Logic Corp., June, 1986.