

# VIPER: A VLIW Integer Microprocessor

*Jeffrey Gray, Andrew Naylor, Arthur Abnous,  
and Nader Bagherzadeh*

Department of Electrical and Computer Engineering  
University of California, Irvine  
Irvine, CA 92717  
(714) 856-8720

## Abstract

This paper describes the design and implementation of a very long instruction word (VLIW) microprocessor. The VIPER (VLIW Integer Processor) contains four pipelined functional units, and can achieve 0.25 cycle-per-instruction performance. The processor is capable of performing multiway branch operations, two load/store operations or up to four ALU operations in each clock cycle, with full register file access to each functional unit. Designed in twelve months, the processor is integrated with an instruction cache controller and a data cache, requiring 450,000 transistors and a die size of 12.9 by 9.1 mm in a 1.2  $\mu\text{m}$  technology.

## 1 Introduction

The VLIW architecture is considered to be one of the promising methods of increasing performance beyond standard RISC architectures. While RISC architectures take advantage of only temporal parallelism by using pipelined functional units, VLIW architectures can also take advantage of spatial parallelism by using multiple functional units to execute several operations concurrently. Similar to superscalar architectures, the VLIW architecture can reduce the cycles per instruction (CPI) factor with this added parallelism. However, superscalar processors schedule the execution order of the operations at run time and demand more hardware support in order to manage synchronization among concurrent operations. Since VLIW machines schedule operations at compile time, allowing global code optimization, they tend to have relatively simple control paths – following the RISC methodology.

In recent years, there have been several efforts to design and develop VLIW architectures [2], [3]. Although the idea of the VLIW architecture had been

known for years, there were few real implementations of such architectures, mainly due to the lack of compilation tools for such machines. Recently, however, with rapid advances in compiler optimization techniques [1], this has changed. These new fine-grain compilation techniques compute a static parallel schedule from an originally sequential program. Percolation scheduling (PS) is one of these promising techniques, and a PS compiler was developed as a parallel effort to the VLSI design of the architecture. The LIFE processor [4] is the only other VLIW microprocessor that is known to have been implemented; however, the VIPER and LIFE processors differ in the treatment of the register file, the compilation strategy, and the number of ALU and load/store units.

The VIPER processor is well-suited for embedded processing applications, as well as general purpose computing. Due to its efficient VLSI implementation, the processor core could easily be integrated with other subsystems of an application-specific controller on a single chip. This paper presents a four-processor version of the VIPER architecture, integrated with data and instruction caches for general purpose use. However, one could foresee future tools designed to tune the configurational parameters of the VIPER architecture for specific applications, such as embedded controllers for HDTV or multimedia applications.

## **2 The Processor Architecture**

The VIPER architecture has been designed to reflect the execution model assumed by the PS compiler, which includes support for multi-way branching and speculative execution. Detailed discussion of architectural design issues and analysis of hardware/software trade-offs can be found in [5]. Feasibility and efficiency considerations for a VLSI implementation of the processor architecture led to the current configuration, which includes four functional units and provides an engine that can take advantage of the parallelizing capabilities of the PS compiler.

The organization of the processor was developed to facilitate an efficient

VLSI implementation. A key aspect was an emphasis on locality of communication between different blocks of the processor. This is reflected in Figure 1 which illustrates the block diagram of the VIPER. The processor contains four integer functional units connected through a shared eight-port register file. The processor control is pipelined and can issue four operations packed into a very long instruction word in each clock cycle. The pipeline structure of the processor, shown in Figure 1, has been designed to reduce the frequency of pipeline hazards and to simplify the bypassing mechanism [5]. The control transfer units in FU0 and FU1 control the instruction fetching sequence and can execute multiway branch operations, while the load/store units in FU2 and FU3 establish a connection to a dual-port data cache subsystem. The processor has some of the typical attributes of a pipelined RISC processor, with a simple instruction set that is designed with efficient pipelining and decoding in mind. All instructions have a fixed size with only a few instruction formats, and all follow the register-to-register execution model. Arithmetic, logic, and shift operations can be executed by all of the functional units. In the VIPER architecture, load/store operations use the register indirect addressing mode instead of the more typical displacement addressing mode in order to eliminate the extra level of bypassing that would be necessary if there were an additional pipeline stage.

Because branch operations are executed by FU0 and FU1, and the pipeline architecture produces a branch delay slot between the branch instruction and the next in-line instruction, there are six available operations that can be scheduled along with the branch. In order to increase the likelihood that the compiler can schedule these open locations, a speculative execution scheme was implemented that allows instructions to complete based on the outcome of the branch. The instruction format includes a 2-bit tag field that corresponds to the location on an instruction graph of a particular operation. Tags are only used in the branch and branch delay slots.

The major difference between VLIW and superscalar processors relates to the issuing of multiple instructions simultaneously. While superscalar machines issue only as many instructions as are allowed by data dependencies and resource

conflicts, VLIW machines always issue the same number of operations per cycle, four in the case of the VIPER. The need for extensive amounts of superscalar hardware to determine these dependencies and conflicts at run time is eliminated in a VLIW architecture. All functional units operate in lockstep; if the need to stall one functional unit arises, all four will have to be stalled. A disadvantage of VLIW is that it will not always be possible to schedule four parallel operations in one instruction word, and some code expansion will be inevitable. This particular issue affects the cache system design, as is explained in section 4.

## 3 Processor Core Design

### 3.1 Operation Timing

The architecture of the VIPER requires overlapped writing and reading of operands to and from the register file within one instruction cycle. This requires a clocking scheme that can support the subdivision of the instruction cycle into distinct read and write phases. In addition, it can be seen in Figure 1 that a register must be able to be written to in the first part of the WB stage for instruction 1, so that the same data can be read in the ID stage for instruction 3. This will result in the elimination of one level of bypassing, and a more efficient VLSI implementation. For these reasons, the instruction cycle was partitioned into four phases. Table 1 shows the low-level operations of the VIPER and the clock phases to which they are assigned.

The four-phase clock drivers were designed to produce non-overlapping clock signals with a minimal amount of dead time between phases. The design used feedback from the previous phase's buffer chain to qualify assertion of the following phase. By carefully tuning the timing of the buffer chains to the load capacitances, the dead time was reduced to an average of 2 ns in a 10 ns phase. Thus, clocked circuits were designed to meet the criterion of an 8 ns-wide pulse. A total of twenty global clock signals are provided by the clock drivers: these are the four phases plus an additional signal which is active throughout phases 1, 2, and 3, used by many of the dynamic circuits in the data paths. Global

complements of all these clock signals are also generated. In addition, separate identical clocks are provided to the processor units and the caches, with the difference that the processor clocks can be stopped in the event of a cache miss. This simple stall mechanism freezes the processor with clock phase 4 high, so that no dynamic circuits will falsely discharge, while the cache controller continues to operate. Once the desired instruction or data word is available, the cache signals the clock generator to restart the processor clocks. This solution exemplifies a design decision where careful circuit design was used to eliminate control hardware cost and complexity.

### 3.2 Register File and Functional Units

At the heart of the processor core is an eight-port register file. The lower-bound of the size of the double-level metal design was limited by the need to have eight word-line and eight bit-line access paths to each register cell. As a result, very little of the register file's area is consumed by active area. Simulations in SPICE showed that the nominal read bandwidth of the register file is 12.8 Gb/s, which is more than sufficient for the required read bandwidth of 6.4 Gb/s for the original target speed of 25 MHz.

The VIPER contains four identical 32-bit data paths which are capable of executing all of the arithmetic and logic functions in the instruction set. Each data path consists of three major blocks:

1. an operand unit, which provides an interface between the data path and the register file and handles the switching functions necessary to implement the operand bypassing scheme used by the VIPER architecture;
2. an arithmetic/logic unit, which is capable of integer addition and subtraction, numerical comparison functions, and logic function computation;
3. a barrel shifter, which can execute logical and arithmetic shift operations of an arbitrary number of bit positions in a single cycle.

In addition, the data paths for functional units FU2 and FU3 include the cache interface circuitry, used for load and store operations.

The operand unit physically separates the source busses from the register file, latching the output of the register file on each instruction cycle and then conditionally providing this data onto the source busses. In the event that a bypass or operand forwarding condition has been detected, indicating that the register file value is not valid, the operand unit will instead provide the data from the destination bus of the functional unit that is the source of the valid data. If the VIPER allowed unrestricted bypassing, sixteen 32-bit busses would be needed to fully connect all four functional units, and eight of these busses would have to cross the length of the chip. To eliminate this excessive wiring, the bypassing scheme in the VIPER restricts bypassing from one functional unit to only itself and its nearest neighbor, resulting in one compact routing channel between adjacent functional units.

### **3.3 Program Counter Unit**

The VIPER's program counter unit was designed to include the architecture's multi-way branching feature. This requires three potential target addresses to be simultaneously computed for each branch; these three adders represent the majority of the cost of the PC unit. Two additional registers were included for return addresses for trap instructions and interrupts, but exception handling was not of major concern for this implementation. The organization of the PC unit is shown in the block diagram in Figure 2.

Because the addition of branch delay slots has a detrimental effect on performance, the address of the next instruction must be available by the end of the instruction decode (ID) pipeline stage. Hence the computation of the possible target addresses occurs during phase 123 of the ID stage. The branch conditions, however, are not available until phase 4. In the case that the conditions are in the register file at the time of the branch, then evaluation of the branch condition can commence once the source busses have stabilized. In the case that the branch conditions are computed by the previous instruction (and must be bypassed) the source busses will take even longer to resolve. Once the branch conditions are available, then the control logic can begin to arbitrate the branch,

requiring additional time before the end of phase 4. This case determines the critical path of the processor, as the pulse width of clock phase 4 must be wide enough to accomplish everything described above. The PC can then be latched on phase 4 and driven to the instruction cache.

### 3.4 Control Design

The control logic for the VIPER is distributed into four blocks, each of which is tightly coupled to the functional unit with which it is associated. Each control path is implemented as three instruction registers, corresponding to the ID, EX, and WB pipeline stages, with combinational logic between registers to perform decoding and other control functions. Instruction decoding is hard-wired, and the lockstep nature of execution and freezing of the processor clocks during cache wait cycles results in a minimal amount of state information. In essence, the control contains only two states, one for normal operation, and one for repeating execution of an instruction when data dependencies can not be resolved with bypassing.

The control paths were designed in standard cells using the Berkeley Octtools system. The overall approach for designing the control logic was to partition the function into blocks which could be separately described in a higher-level behavioral language, and then synthesized and recombined into one physical block. Partitioning for the instruction decoding was performed by grouping together control signals with common functions (for example, all ALU-related control signals), and assigning them as outputs of one functional block. In essence this partitioning also divides decoding by instruction class (computation, control transfer, or load/store), which allows the blocks to be included in the control for only the functional units that require it. The instruction bits then serve as inputs to many of these blocks. The four blocks were placed and routed into four rows of standard cells each in order to achieve an aspect ratio consistent with the floorplan shown in Figure 1. It is worth noting that over 40% of all cells used are dedicated to bypass detection.

## 4 Instruction Cache

One problem inherent in VLIW architectures is that of code expansion. Since the compiler can not always schedule four operations in every cycle, then the unused operation slots will have to be filled with NOP's. In addition, many of the techniques used by the compiler to exploit parallelism, such as loop unrolling and software pipelining, will also cause the code size to grow. This code expansion could have a detrimental effect on the locality of the code, which would affect the cache performance. When combined with the increased width of the instruction word, this effect requires that the instruction cache be somewhat larger than a comparable uniprocessor cache. For these reasons, the instruction cache controller was included on the same die as the processor, with the intent that the actual instruction cache RAM be implemented in external memory.

The most significant physical cost of the VIPER's instruction memory subsystem is the 128-bit width of the instruction word. In order to fetch an instruction in a single cycle, the read bandwidth of the cache must be four times greater than that of a single processor machine. Implementing this wider memory with an on-chip instruction cache does greatly impact the die area, as a cache containing the same number of words as a uniprocessor implementation will have to be four times as large. Preliminary area estimates showed that the largest possible instruction cache feasible for a reasonable chip size would be about 8K bytes. The area required for this size of a cache would be about twice that of all four functional units combined, and a cache of this size would only hold 512 long instruction words. Studies have shown that the expected hit ratio for a cache of this size would be approximately 75% [7]. This would have a serious impact on the peak performance of the VIPER. Recent research has shown that much larger off-chip instruction caches can be built out of standard SRAM that will meet the same cycle time as an on-chip cache [8]; therefore VIPER's instruction cache system employs 256 KB of external RAM. The above justification for external RAM does not apply to the tag RAM or the cache controller, however. After the tag is read out of the tag memory, it still has to be compared with the tag of the pending instruction. The result of this comparison is then

used by the cache controller to decide what the next state of the processor will be. The time required for these additional operations (along with extra chip boundary crossings) makes it difficult to use off-chip RAM for the tag memory without violating the target cycle time. Furthermore, since the tag RAM is small compared to the instruction memory, it was easily included on the die with the processor.

The configuration of the cache is direct-mapped with 512 blocks of 32 words each. The direct-mapped approach was chosen for several reasons. First, the hit rate for direct-mapped caches approaches that of caches with more associativity when the size of the cache becomes large [9]. Second, one side effect of higher degrees of associativity is that the memory must be made wider, since  $N$  banks of memory are required to implement  $N$ -way set associativity. Since the VLIW already requires a fairly wide memory structure, this would merely compound the problem. Finally, the outputs of the separate sets would have to be multiplexed together in order to supply the correct data to the processor. The choice of 32 words for a block size does yield a slightly lower hit rate than a smaller block size would, but was chosen in order to keep the tag RAM at a reasonable size. For a block size of 32 words, the tag RAM is fairly large at almost 1K bytes.

## 5 Data Cache

The data memory subsystem of the VIPER processor was designed to meet several requirements unique to a VLIW architecture. Since VIPER has two parallel data ports, then any data memory subsystem must be able to service two requests per cycle. The two ports are independent, so both ports are able to load or store data regardless of the operation of the other port. A dual-port memory scheme was employed in order to service both data ports of the processor independently. The dual-port design does require special RAM hardware of greater size and complexity than standard RAM, and requires a dual-ported tag RAM to compare the tags from both ports simultaneously. The use of an

off-chip instruction cache left enough core area for 4K bytes of dual-port RAM on the processor die, and allowed use of a single port to main memory. One difficulty with implementing the data cache on-chip was the lack of available RAM generators for dual-ported memory, thus both the data and tag RAM were implemented using full-custom layout.

Unlike the direct-mapped instruction cache, a two-way set associative scheme was chosen for the data cache. Most of the arguments given for the direct-mapped approach used in the instruction cache don't apply to the data cache. First, while the performance gap between direct-mapped and set-associative caches may be small for large cache sizes, the difference can be substantial for smaller caches. In fact, studies have shown that for caches of the size of the VIPER's data cache, the performance gain of doubling the associativity can be about the same as the performance that would be gained by doubling the cache size [7]. Second, since the cache is on-chip, there is no extra chip boundary crossing necessary to choose between the memory banks. Finally, since the data word is only 32 bits wide, then moving to multiple sets does not result in the excessive width problem that would occur with the instruction cache. The cost of this performance is added complexity in both the tag RAM and the cache controller. One added advantage that two-way set associativity gives VIPER is that it makes it easier to resolve collisions between the two ports. These collisions occur when both ports try to access the same cache block, but with different tag values. With a direct-mapped cache, only one of the accesses would be able to go into the cache, and the other would have to be serviced externally. With a two-way set associative cache, each port can access a different bank of data, thus possibly avoiding the external memory reference.

## 6 Global Routing and External Interface

The external interface of the VIPER is dominated by the 128 input pins for the instruction bus, which are necessary to fetch one instruction per cycle. Another 28 output pins are dedicated to the instruction address output bus, which

connects externally to the instruction cache memory. Thirty-two bidirectional pins are used by the data cache to interface with the main memory, using these pins as a multiplexed data and address bus. In addition, twelve more pins are dedicated to various control signals, for a total of 200 signal pins. 32 pins were allocated for powering the pad frame and an additional 20 pins were allocated for powering the core blocks. Thus the total number of pins required is 252, which can be packaged in a standard 256-pin PGA. The I/O buffers used were from the MOSIS standard library, and were assigned to die edges in order to reflect the approximate aspect ratio of the placed but unrouted core macrocells. Several iterations of the final routing stage were performed, altering the pad ordering and macrocell placement to improve the routing quality. The resulting layout was within 3 microns of the minimum size allowed by the pad frame horizontally, and exactly minimum vertically. With 450,000 transistors, the final die size is 12.9 by 9.1 mm, for a total area of 116.5 mm<sup>2</sup>. Figure 3 shows the physical design of the VIPER processor die.

## 7 Final Thoughts

The VIPER VLIW microprocessor we have presented is capable of executing four 32-bit integer operations concurrently. The short design time of twelve months with a small design team demonstrates the power of designing hardware with relatively simple, repeatable components. The small, simple control paths are a distinct advantage over the significantly more complex control found in superscalar architectures due to run-time scheduling hardware. With a simulated operating speed of 25 MHz in 1.2  $\mu$ m technology, the processor can achieve a peak throughput of 100 million native operations per second. While lack of funding has prevented fabrication, we hope that the VIPER will prove to be a useful example of a general-purpose architecture with strong promise for use in other applications that demand high-performance computation.

## References

- [1] A. S. Aiken, "Compaction-Based Parallelization," Ph.D. Dissertation, Cornell University, August 1988.
- [2] K. Ebcioglu, "Some Design Ideas for a VLIW Architecture for Sequential-Natured Software," *Proceedings IFIP*, 1988.
- [3] R. P. Colwell, R. P. Nix, J. J. O'Donnell, D. P., Papworth, and P. K. Rodman, "A VLIW Architecture for a Trace Scheduling Compiler," *IEEE Transactions on Computers*, Vol. 37, 1988, pp. 967-979.
- [4] J. Labrousse and G. A. Slavenburg, "CREATE-LIFE: A Modular Design Approach for High Performance ASIC's," *COMPCON* 1990.
- [5] A. Abnous, "Architectural Design and Analysis of a VLIW Processor," M.S. Thesis, University of California, Irvine, 1991.
- [6] A. Abnous, C. Christensen, J. Gray, J. Lenell, A. Naylor, and N. Bagherzadeh, "VLSI Design of the TinyRISC Microprocessor," *Proceedings of the 1992 Custom Integrated Circuits Conference*, pp. 30.4.1-30.4.5, Boston, May 1992.
- [7] J. Hennessey and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufmann Publishers, Inc., San Mateo, CA: 1990.
- [8] J. Lotz, B. Miller, E. Delano, J. Lamb, M. Forsyth, and T. Hotchkiss, "A CMOS RISC CPU Designed for High Performance on Large Applications," *IEEE Journal of Solid-State Circuits*, vol. SC-25, pp. 1190-1198, October 1990.
- [9] M.D. Hill, "The Case For Direct-Mapped Caches," *IEEE Computer*, vol. 21, no. 12, pp. 25-41, December 1988.

## A Captions

Captions for Gray, Naylor, Abnous, Bagherzadeh; **VIPER – A VLIW Integer Processor**:

Table 1. Phase-by-phase operation of the VIPER.

Table 2. Die area consumed by various processor elements.

Fig. 1. Block diagram and pipeline structure of the processor.

Fig. 2. Block diagram of the program counter unit.

Fig. 3. Physical design of the VIPER processor.

<b>Clock phase</b>	<b>Pipeline stage</b>	<b>Operation</b>
Phase 1	IF	Program counter driven to instruction cache
	ID	Instruction decoding begins Bypass condition detection begins
	EX	Evaluation of ALU elements begins Data cache RAM decoders evaluate Data tag and status RAMs read and compared
	WB	Destination busses written into register file
Phase 2	IF	Instruction cache tags read and compared
	ID	Source register specifiers are decoded Register file precharges
	EX	Evaluation of ALU elements in progress Data RAM is read
Phase 3	IF	Stall signal sent to clock generator in event of instruction cache miss
	ID	Source operands read from register file Outputs of instruction decoders latched Computation of possible PC targets completes
	EX	ALU outputs latched Stall signal sent to clock generator in event of data cache miss
Phase 4	IF	Instruction latched in instruction pipeline
	ID	Bypassing paths activated, source operands latched Source busses driven Data address driven to cache Next PC selected
	EX	Destination busses driven and latched Destination register specifiers decoded

Table 1: Phase-by-phase operation of the VIPER (Gray, Naylor, Abnous, Bagherzadeh)

Cell	Percentage of total chip area
Register file	5.1
Branch units	2.8
Load/store units	3.2
Program counter unit	0.9
Control	6.7
Local routing	9.4
<b>Processing core total</b>	<b>28.1</b>
Data cache RAM	25.7
Data cache control	7.1
Instruction cache control	4.4
<b>Cache total</b>	<b>37.2</b>
Clock generator	1.8
Pads	7.3
Global routing	25.6

Table 2: Die area consumed by various processor elements (Gray, Naylor, Abnous, Bagherzadeh)

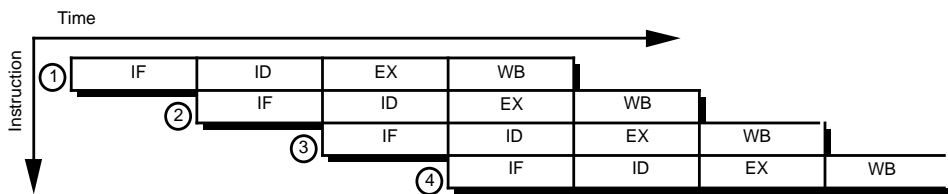
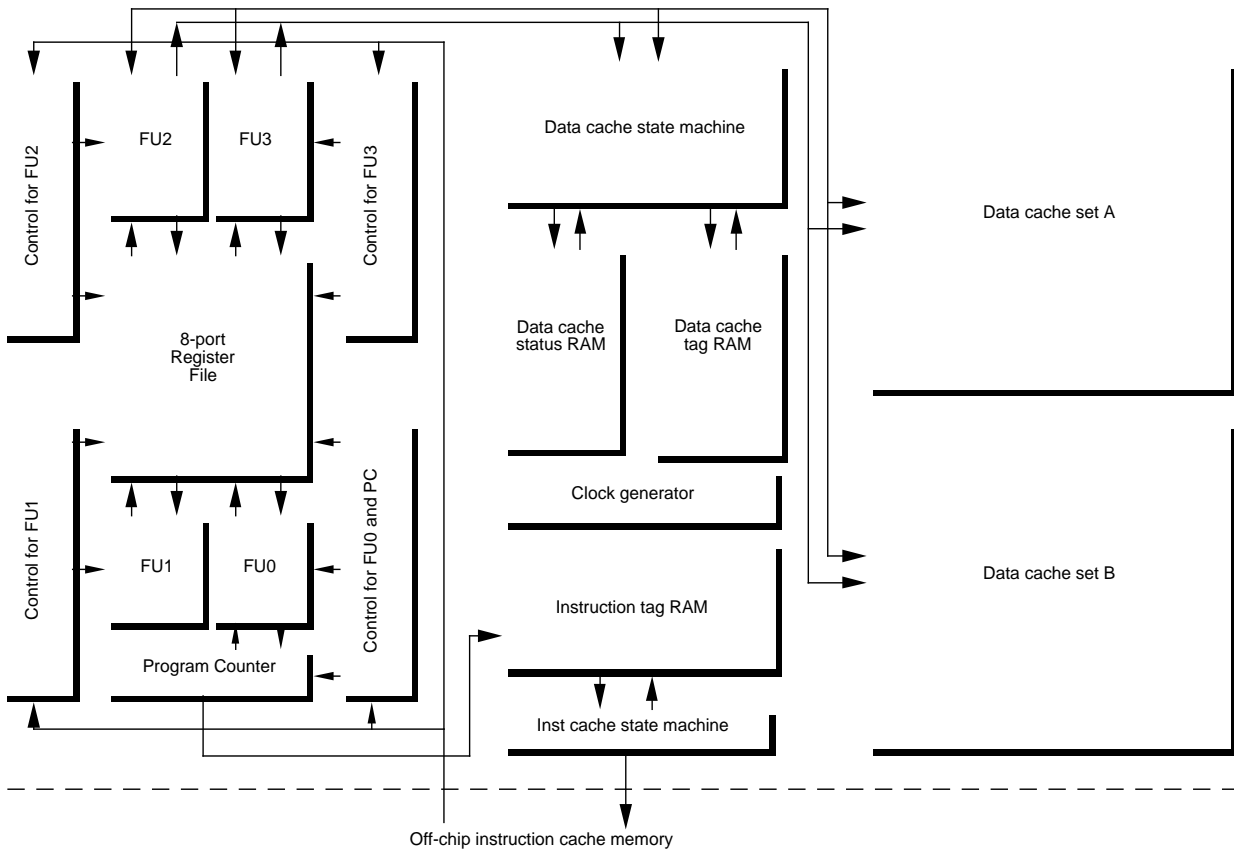


Figure 1: Block diagram and pipeline structure of the processor (Gray, Naylor, Abnous, Bagherzadeh)

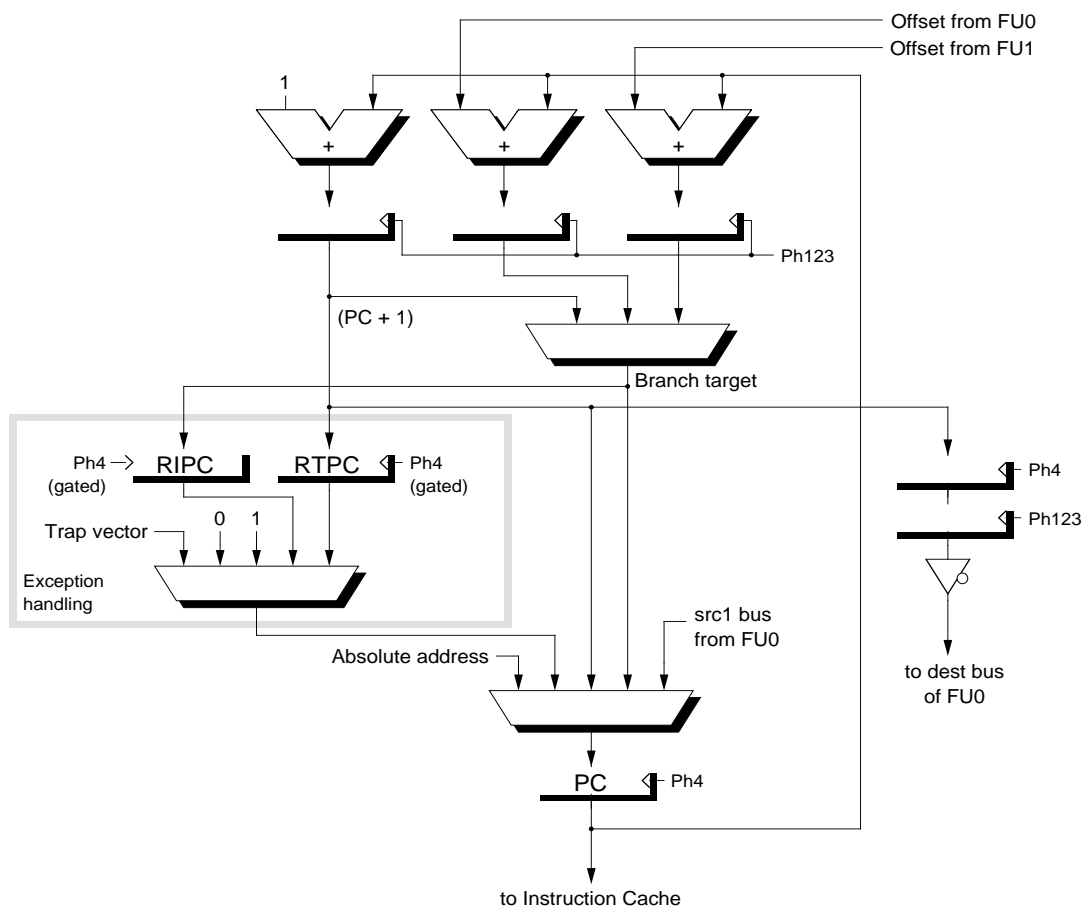


Figure 2: Block diagram of the program counter unit (Gray, Naylor, Abnous, Bagherzadeh)

Figure 3: Physical design of the VIPER processor (Gray, Naylor, Abnous, Bagherzadeh)