CRISP: A Pipelined 32b Microprocessor with 13Kb of Cache Memory

Alan D. Berenbaum†, *Brian W. Colbry*††, *David R. Ditzel*††,
R. Don Freeman††, *Hubert R. McLellan*††, *Kevin J. O'Connor*†††, *Masakazu Shoji*††

ABSTRACT

The implementation and architecture of a 172,163 transistor single-chip general purpose 32b microprocessor is described. The 16 MHz chip is fabricated using a single-metal double-poly 1.75μ CMOS technology and is capable of a peak execution rate of over 1 instruction/clock. Multiple on-chip caches, pipelining and a 1-cycle IO protocol are employed.

Introduction

High performance microprocessor design requires careful tradeoffs with the instruction set, compiler technology, hardware architecture and physical layout of the VLSI circuits. These tradeoffs have been studied since 1975 at AT&T Bell Laboratories in the design of a series of ''C Machines'', computers that efficiently support the C programming language and the Unix operating system. The design of these machines consisted of a cycle of proposing a new machine, writing a compiler for the machine, and using simulators to generate measurements for evaluation. These measurements were then used to make new tradeoffs and propose a new machine to repeat the cycle. This paper describes implementation of the latest C Machine, a 32b single chip microprocessor called CRISP.

The resulting processor contains features of both RISC[1] and CISC machines to uniquely satisfy several goals. First, the streamlined instruction set allows a high performance pipelined implementation. Second, the implementation was constrained to fit on single chip and work well without the need for external caches. Third, hardware was used to simplify the software requirements. Compiler register allocation, reduction of procedure call overhead and other optimizations traditionally required by both RISC and CISC machines have been reduced or eliminated. This allows the compiler to run faster and gives the appearance of even greater overall system performance. Fourth, the instruction set and hardware architecture provides for scalability. By eliminating programmer visible registers and relying on internal caches, implementations can scale with technology. Fifth, despite the simple instruction set, the code density is very compact.

Instruction Set Architecture

The CPU implements about 25 basic instructions and 4 simple addressing modes for accessing data. Basic operations of addition, subtraction, multiplication, division, remainder, bitwise and, bitwise or, bitwise exclusive or, and shifts may use either a 2 address form (i.e. A ← A + B), or a 2½ address form (i.e.

† AT&T Information Systems, Holmdel, NJ
†† AT&T Bell Laboratories, Murray Hill, NJ
††† AT&T Bell Laboratories, Allentown, PA

special temporary ← A + B). Multiplication, division, remainder, and shift are available in both signed and unsigned forms. Two address forms exist for compares, moves and semaphore operations. The result of a compare instruction sets a single condition code bit, which is later used by conditional branch instructions. This condition code bit is not set by any instructions other than compares. This restriction eliminates traditional pipeline dependencies caused by instructions which affect the condition codes. Control instructions for procedure calling and branching use one memory address specifier. The four modes for addressing data are absolute addressing, immediate constants stored in the instructions, stack pointer relative, and indirect though a stack pointer relative location. This 2½ address memory-to-memory architecture executes programs in fewer instructions than register-to-register load/store machines, since both operands can specify a full memory address. The CPU handles 32b words, signed or unsigned 16b halfwords, and signed or unsigned 8b bytes. A co-processor interface supports operands of 1, 2, 3 and 4 32b words. All instructions are restartable for virtual memory support. A variable length instruction encoding yields good code density, generally equal to that of the DEC VAX. Two, six and ten byte instruction lengths utilize 114 encodings of the basic instructions, as shown in Figure 1. The instruction set is summarized in Table 1. A more detailed description of the instruction set can be found elsewhere.[2]

Unlike most computers the CPU has no visible data or address registers. Instead, 32 internal *Stack Cache*[3] registers are automatically allocated by the hardware. These byte addressable registers are mapped into the address space's top-of-stack region. The low order bits of operand addresses are used as register indices, and range check circuits determine whether addressed data resides within the on-chip Stack Cache, or whether data must be fetched from off-chip. Measurements show that about 80% of all data references are to the small region near the top of the stack that can be captured by the 128 bytes of the Stack Cache. The Stack Cache eliminates software register allocation by allocating the registers in hardware, capturing a factor of three more data references on-chip than is accomplished with typical register allocation by compilers. The Stack Cache also allows procedure calls to be very fast as typically no registers need to be saved/restored across procedure call boundaries. The number of Stack Cache registers can be increased in future implementations without software changes because there are no explicit register specifiers in the instruction encoding. To the compiler and programmer, the machine fits a simple memory-to-memory model.

There are several special machine registers. A stack pointer and interrupt stack pointer are used for accessing local variables on a stack frame. A program status word contains condition code bits, interrupt and other control information. A 28-bit timer can be set to count either clock cycles or instructions executed and it can generate an interrupt on overflow. A vector base register allows the specification of program vectors for handling machine interrupts, faults and other exceptions.

Implementation

The microprocessor has been fabricated using a mature 1.75μ twin-tub[4] CMOS technology. Three levels of interconnect are used: a single layer of metal, a low resistance silicided polysilicon, and a second level of non-silicided polysilicon used in the memories for local interconnect. The die size is 10.35mm x 12.23mm and contains 172,163 actual transistors, of which approximately 100,000 transistors are memory. Chip functionality has been measured at 16 MHz 85°C with a 4.75 volt supply. Power dissipation is under 500 mW. The chip has 95 active pads, 20 power pads, 19 ground, and 6 test pads. The chip is packaged in a 125 pin ceramic PGA with internal power and ground planes. A chip photograph is shown in Figure 2.

The chip contains seven static memory arrays containing 13,152 bits of data.[5] There are 2 PLAs. The PDUPLA (20 inputs, 45 outputs, 158 wordlines) translates compact macro-instructions into a more decoded, easy to execute, internal format. The SEQPLA (34 inputs, 46 outputs, 187 wordlines) supports multi-cycle sequences such as multiply, divide, exceptions, and interrupts. Multiple chips may be wired in parallel for fault-tolerant checking operation. In this mode, output circuits on slave chips are disabled and the external state as sensed at the pins is compared with internal state to signal a fault.

As shown in Figure 3, the CPU consists of two logically separate machines, a Prefetch Decode Unit (PDU) and an Execution Unit (EU); each pipelined with 3 stages. The PDU fetches two word blocks of instructions from memory and places them in a 512 byte (64-entry, 64b-wide) Prefetch Buffer Cache (PBUF). The purpose of the PBUF is to match the high bandwidth demands of on-chip instruction execution to the relatively lower bandwidth available across the pins. The organization of the PBUF, a sector cache with 2 valid bits per four word line, is a tradeoff between hit rate and latency. This organization has a higher hit rate than a single word line and increases performance by allowing other IO activity to intervene every two words. Block transfers greater than 2 words were found to degrade performance since the instruction prefetching was interfering with data transfers. Instructions move from the PBUF 64b at a time into an eight-entry 16b-wide instruction Queue. The PDU extracts one to five two-byte instruction parcels from the Queue each clock cycle and translates macro-instructions into an easier to execute internal format. This technique resembles the dynamic generation of horizontal microinstructions. The decoded instructions are placed in a 32-entry 192b-wide decoded Instruction Cache (IC), which serves to connect the PDU to the EU. The IC is implemented as three separate memory arrays; a single 32b by 32-entry array which contains the opcode and control bits and two 80b by 32-entry arrays which store two 32b operands, two 31b next-address fields and a 33b cache tag. By separating the EU and PDU with the IC, what would have been a single six stage pipeline has been turned into two three-stage pipelines. This increases performance by reducing pipeline breakage and allowing each machine to operate autonomously.

The EU reads these pre-decoded instructions from the IC. The first pipeline stage of the EU performs effective address calculations (stack pointer plus offset) with two 28b adders. Operands are then fetched from either the dual-port (32-entry, 32b-wide) Stack Cache or from off-chip. After alignment and sign extension, 32b operands are presented to the ALU and then dis-aligned and stored back either to the Stack

Cache or off-chip as necessary. Pipeline hazards are recognized with four comparators that check the destination address against the left and right operands of the first two pipeline stages. If a hazard occurs, the correct value is automatically bypassed back from the ALU into the correct position in the pipeline.

Instruction and data fetches share a single 32b data bus, and a 30b address bus with 4 byte strobes to give an effective 32b of byte addressing. The IO is fully synchronous and can complete a new 32b IO transaction every clock cycle, for a bus bandwidth of 64 Mbytes/sec at 16MHz operation. Because of extensive on-chip caching, performance degrades gracefully as off-chip memory wait states are added. The IO protocol also supports co-processors, slow tri-stating peripherals, bus cycle retry, and interlocked bus operations.

The extensive use of on-chip caches allows on-chip access to data and instructions at very high bandwidths. Two 32b Stack Cache reads and one write per cycle, one 64b PBUF read or write per cycle, and a read and write of the 192b IC per cycle give an effective bandwidth equivalent to 1,088 Mbytes/sec.

Branch Folding

An instruction may be executed several times from the Decoded Instruction Cache and each time (except for conditional branches) the next instruction address is always the same. Instead of recalculating this address every time the instruction is executed, the next address logic was moved to the input side of the Decoded Instruction Cache. As instructions are decoded and placed in the IC, their "next address" value is stored with them in an additional cache field. When an instruction is read from the IC by the EU, its next address is then immediately available to retrieve the next instruction from the IC.

Providing a next address field for every instruction in the IC has the same effect as turning every instruction into a branch. Since every instruction in the IC can specify a branch address, there is no need for separate branch instructions as seen by the EU. The logic in the PDU decodes two instructions simultaneously. When it recognizes that a non-branching instruction is followed by a branch instructions it ‘‘folds’’ the two instructions together. This single instruction is then placed into the decoded IC. The separate branch instruction disappears entirely from the execution pipeline and the program behaves as if the branch were executed in zero time. This technique is referred to as *Branch Folding*.[6]

Since branches constitute as much as 30% of the instructions typically executed in computers, Branch Folding provides significant performance advantages. Conditional branches are handled in a similar manner by providing a second alternate next-address field in the Decoded Instruction Cache. Branch prediction mechanisms, in both hardware and software,[7] chose which execution path to follow.

Circuit Techniques

The clock strategy is shown in Figure 4. Two externally provided, 90 degree out of phase, CMOS level clocks are required. After the clocks are buffered with a single inverter stage, one of them is further passed through a processing independent delay line to yield a third internal clock. These three clocks are distributed throughout the chip and locally decoded to provide one of four phases. The clocks are routed from the IO frame to the local decoders in continuous metal to reduce the internal clock skew. The internally delayed clock is used to generate non-overlapping phases. A 1x input clock and minimal decode delay reduces skew between internal and external circuits and simplifies system design at higher frequencies.

Static, Domino and Zipper CMOS circuits are used. Zipper CMOS,[8] also known as n-p-CMOS[9] or modified Domino,[10] is a dynamic circuit technique consisting of alternating NFET and PFET logic stages. Dynamic circuits operate by initially precharging an output node and later conditionally discharging it by an array of transistors implementing the desired logic function. Replacing static CMOS's complementary pair of FETs with a single precharge/discharge FET eliminates half the capacitive load from every input and significantly improves performance. The discharge path has traditionally been implemented with NFETs due to better electron mobility. However, with increasing electric fields from shorter channels, velocity saturation occurs and the advantage of electron mobility over hole mobility decreases. The performance of dynamic PFET logic thereby approaches that of dynamic NFET logic. It is now practical to eliminate the static inverter from Domino CMOS[11] and cascade alternating dynamic NFET and PFET logic stages directly. The resulting circuit is Zipper CMOS.

Zipper CMOS requires that all inputs be either stable before the end of the precharge period, or make a single transition from the inactive to active state during the evaluate period. Otherwise, precharged nodes could be falsely discharged. If all inputs comply with this rule, internal signals are guaranteed to make at most a single transition from the precharged state and the circuit will be race free.[12] Fortunately, these conditions can be satisfied in most microprocessor datapaths. Furthermore, Zipper CMOS is unaffected by clock skew.

Zipper CMOS is particularly attractive for the carry look-ahead circuits of 32b adders since all inputs have the same timing characteristics. The chip contains eight Zipper adders and incrementers from 10 to 32 bits. All of these circuits are derivative sub-circuits of the main 32b ALU. The ALU provides 32b addition and the 32b parallel logic operations: OR, AND, and XOR.

Figure 5 shows the carry propagate and carry generate circuit used for the odd bits of the Zipper ALU. The circuit for the even bits is similar, except the inverter is deleted. The even bits require active low propagate and generate signals and the odd bits need active high signals. During precharging **pre_L** is held low, and the data inputs **a** and **b** are held low by the input circuits to the ALU. Holding the data inputs low during precharge allows the elimination of the discharge FET as would be typically found in Domino

circuits.

Figure 6 depicts the remainder of a four bit section of the ALU. During precharging **pre** is held high and **pre_L** is held low. During evaluation **pre** is held low and **pre_L** is held high. The alternating active high and active low propagate (**p0_L, p1, p2_L, p3**) and generate (**g0_L, g1, g2_L, g3**) terms control the corresponding NFET and PFET stages. Addition is performed unless **logic_L** is asserted. The Domino style inverter connecting C_i to each XOR gate is critical for correct operation. Otherwise, the Zipper stage is capacitively connected to the XOR output via gate to drain overlap parasitics. If the XOR switches, the dynamic node's voltage can change and improperly trigger other connected Zipper stages. Zipper CMOS performance is achieved because switching occurs at the transistor threshold, V_T, instead of switching at Vdd/2. Switching at V_T requires precharging (or discharging) all internal nodes to eliminate charge sharing and false switching.

The full 32b carry generation is accomplished by look-ahead over four bits and ripple over eight 4b-nibbles. Carry generation over four bits is achieved in 5.0 nS and full 32b carry propagation occurs in 22.3 nS. Zipper CMOS provides a good trade-off between area and performance. In the current microprocessor, a second level of carry look-ahead was not required since the compact layout for a single level of carry look-ahead met the timing goals.

Performance Characteristics

Since the PDU can combine both a branch and a non-branch instruction in a single decoded IC entry using Branch Folding, peak throughput is greater than 1 instruction/cycle. For very large programs, cache misses, multiple cycle instructions and other effects cause typical execution rates of 2.5 to 3.5 cycles/instruction. Multiply instructions take from 5 to 35 cycles using a modified Booth algorithm. For large Unix programs performance is approximately 9 times the performance of a VAX-11/780. Dhrystone benchmark performance without optimization is 13,560 Dhrystones at 16 MHz with 0 wait states.

Performance for several other benchmarks is shown in Table 2. Comparison is made with a CISC machine, the DEC VAX-11/780, and with another RISC machine, the MIPS Computer Systems R2000 processor as implemented in the M/500 Development System. The figures for CRISP show a system running at 16 MHz with no wait states. A second mask of CRISP using the same technology is expected to run above 20 MHz. The numbers show CRISP to be substantially faster than the VAX, and slightly faster than the MIPS R2000.

For smaller systems CRISP provides a distinct advantage in having the caches on-chip. A one wait-state system can be built at 16 Mhz interfacing directly to dynamic memory with a loss in performance of only about 20% compared to a zero wait-state system. CRISP saves substantially in board area and parts cost since it is a complete CPU on a single chip, compared to the R2000 which requires two external caches and various support chips, for a total of about 30 chips. CRISP's good code density is also important in

some applications where memory costs are still a large portion of system costs, compared to the poorer code density typically found in load/store RISC machines.

Benchmark	VAX-780	R2000	CRISP	CRISP/VAX	CRISP/R2000
ackerman	20.9 sec	1.6 sec	1.1 sec	19.0	1.5
word count	55.0 sec	5.2 sec	4.2 sec	13.1	1.2
quicksort	36.2 sec	4.0 sec	3.4 sec	10.6	1.2
tty driver	17.4 sec	2.2 sec	1.2 sec	14.5	1.8
symbol table	14.6 sec	1.3 sec	1.2 sec	12.2	1.1
buffer release	9.9 sec	0.9 sec	0.8 sec	12.4	1.1
arithmetic	12.8 sec	2.7 sec	1.6 sec	8.0	1.7

Table 2. Relative performance of a 16 MHz CRISP.

CAD Methodology

The complex nature of the pipelined implementation required an extensive use of CAD tools.[13] The specification, implementation and verification of the chip was performed in a hierarchical fashion, with each level being more accurate at the expense of more CPU time required for simulation. At the highest level, a C compiler and instruction set interpreter provided a semantically correct implementation of the instruction set. Register transfer level simulation was provided with an architectural simulator. A functional simulator[14] provided simulation correct to individual clock phase boundaries, and modeled the physical chip partitioning of the separate logic blocks. Hierarchical schematics captured the exact connectivity of each transistor on the chip. The functional simulator was capable of driving a transistor level (0/1) simulator for individual blocks, or the entire chip, without the explicit use of test vectors.

Layout was done using a symbolic virtual grid compaction system[15] except for the cache memories, which used a physical geometry layout editor to define second level poly and non-Manhattan geometries. Logical correctness of the layout was verified by circuit extracting[16] the final mask data and making a 1-to-1 comparison with a transistor level netlist generated from schematics.[17] Timing of extracted layout was verified with a new static critical path analysis tool.[18] Transistor sizing of random logic control blocks was done by starting with minimum transistor sizes, then resizing only those transistors constraining the critical path. The result was a smaller layout and lower capacitances than would have been possible by fanout based approaches. This approach was reasonable only because the automatic recompaction allowed quick layout changes, and the timing verifier could re-evaluate the effect of transistor size changes over the entire chip in a matter of seconds. Because of the enormous size of the database, a significant part of the methodology was to have all the tools capable of handling the entire chip at one time, rather than doing piecewise verification.

Chip test is accomplished by a sequence of macro-instructions; no LSSD or special test circuits are used. The on-chip cache memories may be disabled under software control.

Conclusion

CRISP is a single chip microprocessor fabricated in 1.75µ CMOS technology. Peak performance is greater than 16 MIPS at 16 MHz operation. Branches can be executed in zero time. Registers are allocated dynamically by the hardware rather than by a compiler. Autonomous execute and decode units work cooperatively to execute a common program. The overhead for procedure calls has been reduced dramatically from that of typical register oriented machines. All these features are available on a machine with an instruction set that is simple to understand and for which it is easy to generate code. CRISP achieves the instruction efficiency and code compaction of CISC machines, while reaching beyond the performance levels of pure RISC machines without resorting to extremes of compiler optimization.

Acknowledgements

L. Chen, R. Cmelik, R. Franzo, R. Heaton, D. Inglis, G. Janac, K. Maletsky, F. Remite, and M. Zalonis made significant contributions to the implementation. Special thanks to T. Szymanski for his state of the art CAD tools, and to A. G. Fraser, V. A. Vyssotsky and A. A. Penzias for their support. Many people with many skills were associated with this project. To all who helped, our thanks.

References

1. Ditzel, D. R., Patterson, D. A., ‘‘The Case for the Reduced Instruction Set Computer,’’ *Computer Architecture News* **8**(7) (1980).
2. Berenbaum, A. D., Ditzel, D. R., McLellan, H. R., *Introduction to the CRISP Instruction Set Architecture,* Proceedings of the Spring COMPCON (February, 1987).
3. Ditzel, D., McLellan, R., ‘‘Register Allocation for Free: The C Machine Stack Cache,’’ *Proceedings of the Symposium on Architectural Support for Programming Languages and Operating Systems*, Palo Alto, CA, pp. 48-55 (March, 1982).
4. Parillo, L. C., Payne, R. S., Davis, R. E., Reutlinger, C. W., Fields, R. L., *Twin-tub CMOS - A Technology for VLSI Circuits,* IEDM (December, 1980).
5. O’Connor, K. J., *Modular Embedded Cache Memories for a 32b Pipelined RISC Processor,* Proceedings of ISSCC87 (1987).
6. Ditzel, D. R., McLellan, H. R., ‘‘Branch Folding in the CRISP Microprocessor: Reducing Branch Delay to Zero,’’ *Proceedings of the 14th Annual Symposium on Computer Architecture* (June, 1987).
7. Bandyopadhyay, S., Begwani, V., Murray, R., *Compiling for the CRISP Microprocessor,* Proceedings of the Spring COMPCON (February, 1987).
8. Lee, C. M., Szeto, E. W., ‘‘Zipper CMOS,’’ *IEEE Circuits and Devices*, pp. 10-16 (May 1986).
9. Goncalves, N. F., DeMan, H. J., ‘‘n-p-CMOS: A Racefree Dynamic CMOS Technique for Pipelined Logic Structures,’’ *ESSCIRC Dig. Tech. Papers*, pp. 141-144 (Sept. 1982).
10. Weste, N., Eshraghian, K., *Principles of CMOS VLSI Design,* Addison Wesley (1985).
11. Krambeck, R. H., Lee, C. M., Law, H. F. S., ‘‘High-Speed Compact Circuits with CMOS,’’ *IEEE J. of Solid-State Circuits* **SC-17**(3) (June 1982).
12. Goncalves, N. F., Man, H. J. De, ‘‘NORA: A Racefree Dynamic CMOS Technique for Pipelined Logic Structures,’’ *IEEE J. of Solid-State Circuits* **SC-18**(3) (June 1983).
13. Ditzel, D. R., Berenbaum, A. D., ‘‘Using CAD Tools in the Design of CRISP,’’ *IEEE Design and*

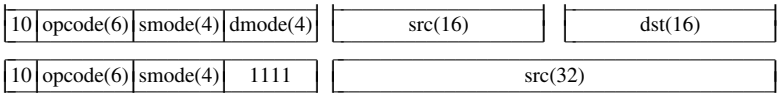
Test (June, 1987).

14. Berenbaum, A. D., Heaton, R. A., ‘‘Functional Simulation for the CRISP Microprocessor,’’ *Proceedings of VLSI87* (October, 1987).
15. Weste, N. H. E., ‘‘MULGA - An Interactive Symbolic Layout System for the Design of Integrated Circuits,’’ *The Bell System Technical Journal* **60**(6, part 1), pp. 823-857 (July-August 1981).
16. Szymanski, T. G., Wyk, C. J. Van, ‘‘Goalie: A Space Efficient System for VLSI Artwork Analysis,’’ *IEEE Design & TEST* **2**(3), pp. 64-72 (June, 1985).
17. Ebeling, C., Zajicek, O., *Validating VLSI Circuit Layout by Wirelist Comparison,* Proceedings of the Design Automation Conference (1983).
18. Szymanski, T. G., ‘‘LEADOUT: A Static Timing Analyzer for MOS Circuits,’’ *Proceedings of the ICCAD Conference*, pp. 130-133 (October 1986).

Ten Byte



Six Byte



Two Byte

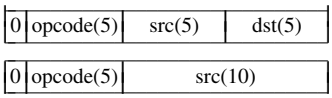


Figure 1. Instruction Encoding Formats.

Table 1. CRISP Instructions

Address Type	Name	Function	Addressing Modes	Data Types
2 and 2½ address	add	addition	Immediate Absolute Stack Offset Indirect	unsigned byte signed byte unsigned halfword signed halfword word
	sub	subtraction		
	mul	multiplication		
	quo	division		
	and	bitwise and		
	or	bitwise or		
	xor	bitwise exclusive or		
	shr	arithmetic shift right		
	ushl	unsigned shift left		
	ushr	unsigned shift right		
2 address	umul	unsigned multiply		
	uquo	unsigned divide		
	urem	unsigned remainder		
	cmp.=	equality comparison		
	cmp.s<	signed less than comparison		
	cmp.u<	unsigned less than comparison		
	move	move		
	mova	move effective address		
	addi	bitwise add interlocked		
	andi	bitwise and interlocked		
	ori	bitwise or interlocked		
1 address	jmp	unconditional jump	Absolute Absolute Indirect Stack Offset Indirect PC Relative	
	ifTjmp	conditional jump if True		
	ifFjmp	conditional jump if False		
	ifCjmp	conditional jump if Carry		
	ifOjmp	conditional jump if Overflow		
	call	procedure call		
	kcall	kernel call	Immediate	
	enter	allocate new stack space		
	return	de-allocate space and return		
	catch	restore stack cache		
0 address	nop	no operation		
	cpu	internal register access		
	kret	kernel return		

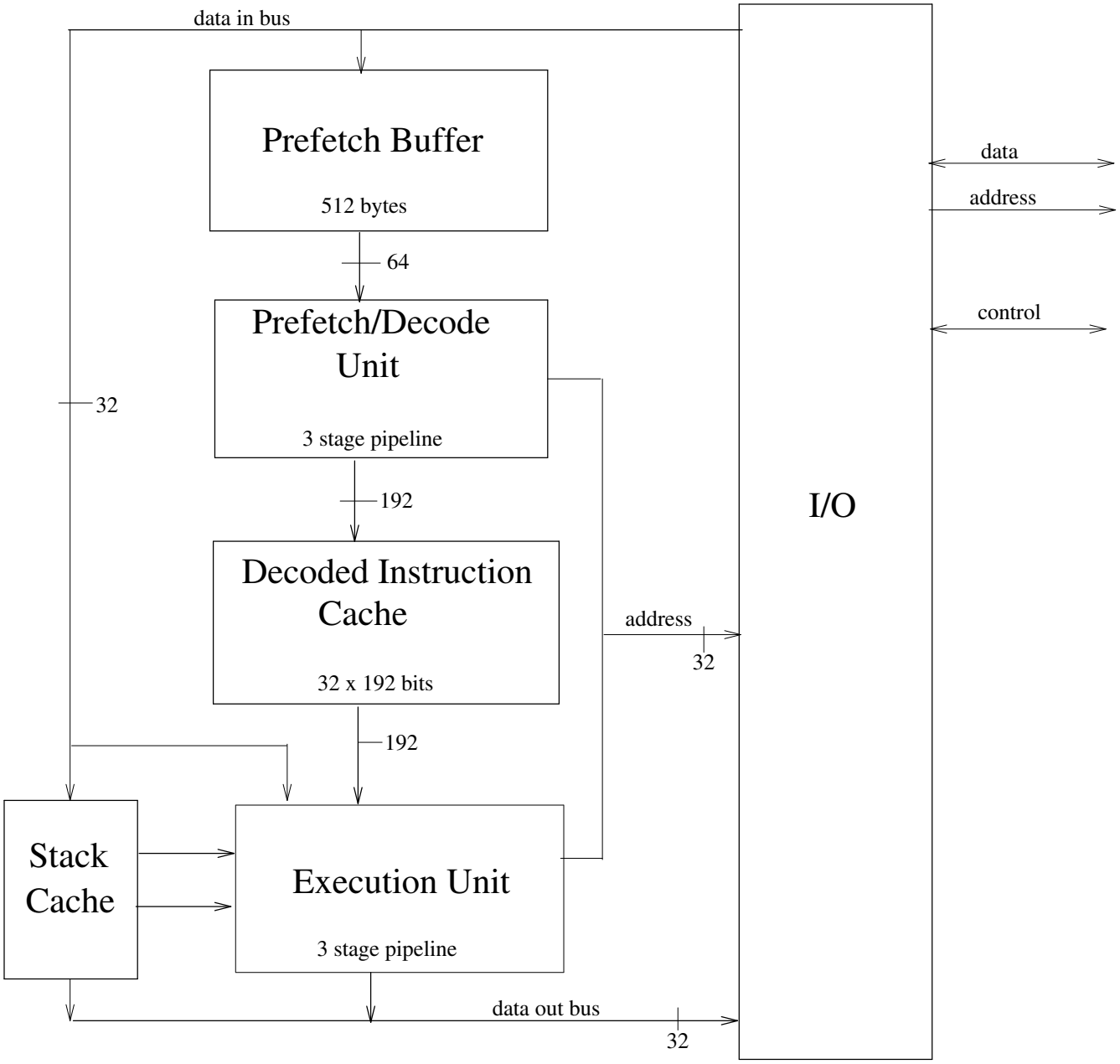


Figure 3. CRISP Microprocessor Block Diagram