The Development of iHARP: A Multiple Instruction Issue Processor Chip

G.B. Steven , R.G. Adams, P.A. Findlay and S.A. Trainis

1. Introduction

During the last decade RISC ideas on processor architecture have become widely accepted. RISC architectures achieve significant performance advantages over CISC architectures by striving to execute one instruction per cycle. However, a traditional RISC architecture can never execute more than one instruction per cycle. Achieving further performance improvements beyond RISC depends on developing processors which fetch and execute more than one operation in each processor cycle.

The objective of the HARP project is to design, build and test a processor which executes non-numeric benchmarks at a sustained execution rate in excess of two instructions per cycle. Earlier work at Hatfield centered around the design of an abstract HARP architectural model [1-3]. This paper describes iHARP [4,5], a physical realisation of the HARP architectural model within the constraints of a single VLSI chip.

The iHARP processor fetches a 128-bit long instruction word from an instruction cache every processor cycle. Each long instruction consists of four, 32-bit short RISC primitives which are dispatched to four integer pipelines for parallel execution. The HARP compiler is responsible for detecting groups of instructions which can be executed in parallel and for placing them in a single long instruction word at compile time. Multiple-instruction-issue architectures which rely on the compiler to schedule instructions at compile time are often called VLIW (Very Long Instruction Word) architectures[6-8]. More recently processors which use hardware techniques to execute multiple instructions in parallel have been termed superscalar processors[9].

iHARP is a single chip, multiple-instruction-issue processor which is targeted at non-numeric applications. Non-numeric code differs from numeric code in several ways. First, there is less parallelism to exploit. Consequently, long instruction latencies are less easily tolerated, and the compiler must employ more aggressive instruction scheduling techniques. Second, a typical operand is no longer an element of a large array. The latency of a typical memory load instruction can therefore be significantly reduced by providing a data cache.

This paper introduces the iHARP chip design. Particular attention is paid to the development of the pipeline structure, to the register file organisation and to iHARP features which minimise instruction latencies. The paper concludes with a summary of the current status of the project.

2. Instruction Set

iHARP supports a conventional RISC load and store architecture. Computational instructions use two general-purpose register operands and return a result to a third register. In all cases a signed 11-bit literal or a long 32-bit literal can be substituted for the second operand. Load and store instructions use the same two operands to compute memory addresses.

Conventional condition codes are replaced by eight, one-bit Boolean registers which are set explicitly by relational or compare instructions. For example

```
NE B1, R2, R3
```

sets Boolean register B1 to TRUE if register R2 is not equal to register R3. As well as being tested by conditional branch instructions, the Boolean registers are used to directly control instruction

Hatfield Polytechnic, Hatfield, Herts AL10 9AB

execution. All instructions, including conditional branch instructions, specify a Boolean register and a truth value. An instruction is only executed if the value held in the Boolean register corresponds to the specified truth value. For example, the instruction

```
T B2 ADD R1,R2,R3
```

will only add register R2 to R3 and place the result in R1 if Boolean register B2 holds the value TRUE. Conditional instruction execution allows instructions which are normally executed after a branch to be executed as soon as the Boolean condition which controls the branch is computed. Unconditional instruction execution is achieved by testing B0 which is always FALSE.

3. Pipeline Structure

All iHARP instructions are executed in the following four stage pipeline:

IF	Fetch next instruction from instruction cache
RF	Fetch register operands from register file
ALU/MEM	Perform operation or access data cache
WB	Return results to register file

Computational instructions use all four stages: reading two operands in the second stage, performing the actual computation in the third stage and returning a result to a register in the final stage. Relational instructions use a dedicated high-speed comparator in each pipeline to compute a Boolean value during the first half of the ALU/MEM stage. As a result new Boolean values are available for control purposes before the end of the ALU/MEM stage. Memory reference instructions compute a memory address in the RF stage and access a separate off-chip data cache in the ALU/MEM stage. Load instructions also return a result to a register during the WB phase. All iHARP branch instructions have a branch delay of one. Branch instructions use dedicated adders to compute branch targets in the RF stage, thus allowing branch resolution at the end of the RF stage.

Not all instructions are supported by all four pipelines. Memory reference instructions, for example, can only be executed in pipelines zero and two. While two pipelines are able to generate a memory address, only one access can be made to the data cache in each cycle. Two memory reference instructions may therefore only be placed in a single long instruction word if they are guarded by mutually exclusive Boolean conditions. Similarly, branch instructions are restricted to pipelines one and three. Since a branch instruction in pipeline three is always given priority, two branch instructions executed in parallel have the same effect as two branches executed consecutively in sequential code.

4. Register File Organisation

The iHARP processor provides 32 general-purpose registers which are shared by all four pipelines. Register zero is read only and always delivers zero. While each short instruction normally requires two register operands, a store instruction may require three, two to compute the memory address and a third to provide the data to be stored. As a result iHARP ideally requires a fourteen-ported register file which is capable of reading ten operands and writing four results in every processor cycle.

The iHARP register file implementation uses standard RAM primitives with two read ports and two write ports. Multiple read ports are provided by replicating the register file and associating one file with each pipeline. An additional register file, associated with the address unit, provides the third operand for store instructions. Only two of the four ALUs are allowed to return a result to a general-purpose register in each processor cycle.

To reduce the scheduling pressure on the write ports, results can be bypassed to the next sequential instruction without being written to a register. A write-back bit provided in each short instruction controls this facility. If the write-back bit is not set, the hardware makes no attempt to return a result to the specified register. However, the result is still automatically forwarded to any short instruction in the next instruction word which uses the same register operand. This facility avoids

ɔring temporary results which are immediately used.

It is felt that two write ports in conjunction with iHARPs distinctive optional write-back facility will not significantly reduce the parallelism realised by the iHARP instruction scheduler. Many instructions, including relational, branch and store instructions, do not produce integer results. Furthermore two instructions scheduled on mutually exclusive Boolean conditions only generate a single write to the register file. Such instruction pairs occur frequently as a byproduct of the code motion undertaken by the HARP instruction scheduler.

5. Instruction Latency

The iHARP processor is specifically targeted at non-numeric applications. In such an environment the available parallelism may be as low as two. It is therefore essential not to squander any hard-won parallelism gained by the compiler on increased instruction latencies. This section discusses aspects of the iHARP design which are related to instruction latency.

5.1 Bypassing

Since iHARP instructions are executed in a four-stage pipeline, a result produced by one instruction is not written to the general-purpose register file in time to be read as an operand by the next sequential instruction. Immediate re-use of data is only possible if a bypass path is provided from ALU output to ALU input. To minimise instruction latencies, 32-bit bypass paths are provided from all four ALU outputs to all eight ALU inputs. Similarly a value loaded from the data cache can be bypassed to the ALU inputs for immediate use.

5.2 ORed Indexing

Many RISC processors provide delayed load instructions. As a result data loaded from the data cache can not be used by the next instruction. Recent simulations [10] suggest that introducing a one cycle load delay in iHARP would degrade the instruction-issue rate by approximately 25%. iHARP therefore implements the following four cycle load instruction:

IF	Instruction Fetch
RF	Fetch address components and compute memory address
ALU/MEM	Access data cache
WB	Load data into register file

This timing allows data loaded from the data cache to be bypassed directly to the next instruction. The drawback is that memory addresses must be computed in the RF stage. Since insufficient time is available to access the register file and perform a 32-bit addition of the address components, some simplification of the addressing mechanism is essential.

iHARP implements a distinctive ORed indexing mechanism, where a bitwise OR operation is performed between the two address components to form the effective address[11]. This simple mechanism is equivalent to an addition, providing the calculation is actually a concatenation. HARP compilers enforce this requirement by starting all procedure activation records on a power-of-two boundary[12,13]. The least significant bits of the stack pointer are forced to zero on procedure entry and variables are accessed relative to the stack pointer using ORed indexing. The power-of-two boundary used is adjusted from procedure to procedure to avoid excessive memory fragmentation. Effectively a new variable-sized 'page' is allocated on the stack on procedure entry.

5.3 Boolean Registers

All iHARP instructions are conditionally executed. Each short instruction specifies one of eight Boolean registers and an associated truth value. At run-time an instruction is only allowed to change the machine state if the Boolean register holds the specified value. The Boolean values themselves are generated using a set of relational instructions. Conditional instruction execution can be viewed as a mechanism for reducing branch latency. Instructions which are normally executed after a branch instruction can now be conditionally executed as soon as the branch condition is resolved. Thus the code fragment

```
        NE B1,R1,R2          /* Calculates a Boolean value */
        BT B1,label          /* Branch if Boolean TRUE */
        NOP                  /* Branch delay slot */
        Instr1
        Instr2
        Instr3
        :
label:  Instr4
        Instr5
        Instr6
```

can be scheduled in parallel as:

```
        NE B1,R1,R2
        BT B1,label+2;  F B1 Instr1;    T B1 Instr4
                        F B1 Instr2;    T B1 Instr5
        Instr3
        :
label+2: Instr6
```

Both instruction streams following a branch instruction can therefore be moved up by at least two instruction slots. Further code motion is possible if the relational instruction can be separated from its branch. Note that the introduction of pairs of instructions which are executed on a mutually exclusive Boolean condition will reduce the write-back pressure on the general-purpose registers.

Now consider the timing of the first two long instructions. The initial comparison for equality is carried out by a dedicated comparator in the first portion of the ALU pipeline stage, and the Boolean result is loaded immediately into the appropriate register. In contrast, the whole of the ALU pipeline stage is required to compute the two following ALU operations, and the integer results are not returned to the register file until the final write-back stage of the pipeline, two cycles after the value of B1 is updated. These observations suggest that it should be possible to move the ALU operations in parallel with the relational instruction. As a result the iHARP instruction scheduler can produce the following parallel code:

```
        NE B1,R1,R2;    F B1 ALUInstr1; T B1 ALUInstr4
        BT B1,label+3;  F B1 Instr2;    T B1 Instr5
                        F B1 Instr3;    T B1 Instr6
        :
label+3: :
```

Both original sequential instructions streams have now been moved up by three instruction slots. Only computational instructions can be scheduled in this fashion. The remaining instruction types can not be placed in parallel with an instruction which computes their Boolean control value.

6.0 Project Status

The iHARP design is targeted at the 1.5 micron CMOS ES2 process. The chip has a target size of 150 sq mm and will be packaged in a 180 pin grid array. To squeeze the chip into a 180 pin package, the 128-bit wide instruction bus is time multiplexed. Short instructions one and three are returned to the chip from the instruction cache at the start of the RF cycle, while instructions zero and two are returned mid-way through the cycle. No time penalty is incurred, since only the branch instruction offsets from pipeline one and three are actually required on-chip during the first half of the RF pipeline stage.

The chip is being developed using Cascade, formerly Seattle Silicon, ChipCrafter software running on an Apollo workstation. To reduce development time, heavy reliance is being placed on the high-level building blocks provided by Chipcrafter. The iHARP design data base is complete and layout, timing analysis and simulation are now in progress. Chip fabrication is scheduled for the final quarter of 1992.

prototype chip will be incorporated in a small test-bed system. At the heart of the system will a custom designed processor board consisting of the iHARP processor chip, off-the-shelf Motorola cache chips and a VME bus interface. Main memory will be provided by a standard, dual-ported, VME RAM card. These two boards will be embedded in a standard VME development system which will be used to test the iHARP processor and to provide an interface to the Hatfield campus network. This test bed will allow evaluation of the iHARP architecture using C and Modula-2 benchmarks.

7. Conclusions

The success of the project ultimately depends on the ability of the HARP compiler system to schedule iHARP instructions in parallel. Initial results obtained using an ELLA simulation of the abstract HARP model are encouraging[12]. Using a suite of ten integer benchmarks, a straightforward instruction scheduling algorithm reduced dynamic execution times by an average factor of 2.07.

On iHARP the instruction latency timings are more aggressive, especially with respect to relational instructions. Also the scheduling techniques employed to date have been very straightforward. As a result, we are confident that, with the use of more advanced instruction scheduling techniques, iHARP will be able to execute non-numeric benchmarks at a sustained instruction execution rate in excess of two instructions per cycle.

Acknowledgements

The HARP project is supported by SERC Research Grant GR/F88018. Part of this work is also supported by an SERC studentship.

References

[1] Steven,G.B., Gray,S.M. and Adams,R.G. "HARP: A Parallel Pipelined RISC Processor", Microprocessors and Microsystems, Vol.13, No.9 (November 1989), pp 579-587.

[2] Adams,R.G., Gray,S.M. and Steven,G.B. "Utilising Low Level Parallelism in General Purpose Code: The HARP Project," Microprocessing and Microprogramming, Vol.29, No.3 October 1990, pp137-149.

[3] Steven,G.B. and Gray,S.M. "Specification of a Machine Model for the HARP Architecture and Instruction Set - Version 3", Hatfield Polytechnic, Division of Computer Science, TR117, January 1991, pp1-52.

[4] Trainis,S.A. "A Device Specification for the iHARP Processor", Hatfield Polytechnic, Division of Computer Science, TR118, February 1991, pp1-46.

[5] Findlay,P.A., Trainis,S.A., Steven,G.B. and Adams,R.G. "HARP: A VLIW RISC Processor", CompEuro91, Bologna, May 1991, pp368-372.

[6] Colwell,R.P., Nix,R.P., O'Donnell,J.J., Papworth,D.B., and Rodman,P.K. "A VLIW Architecture for a Trace Scheduling Compiler" IEEE Transactions on Computers, Vol.37, No. 8, August 1988, pp967-979.

[7] Annaratone,M., Arnould,E., Gross,T., Kung,H.T., Lam.M., Menzilcioglu,O. and Webb,J.A. "The Warp Computer: Architecture, Implementation, and Performance," IEEE Transactions on Computers, Vol.C-36, No. 12, December 1987, pp1523-1538.

[8] Rau,B.R., Yen,D.W.L., Yen,W. and Towle,R.A. "The Cydra 5 Departmental Supercomputer: Design Philosophies, Decisions and Trade-offs," Computer, January 1989, pp12-35.

[9] Johnson,M. "Superscalar Microprocessor Design", Prentice-Hall, New Jersey, 1991.

[10] Chang,P.P., Mahlke,S.A., Chen,W.Y., Warter,N.J. and Hwu,W.W. "IMPACT: An Architectural Framework for Multiple-Instruction-Issue Processors", 18th Annual International Symposium on Computer Architecture, Toronto, May 1991, pp266-275.

[11] Steven,G.B. "A Novel Effective Address Calculation Mechanism for RISC Microprocessors", SIGARCH, September 1988, pp150-6.

[12] Gray,S.M. "Code Generation for Long Instruction Word Architectures", PhD Thesis, Hatfield Polytechnic, expected December 1991.

[13] Wang,L. "Crafting a C Compiler for the iHARP Chip using the GNU Compiler Compiler", Division of Computer Science, Hatfield Polytechnic, TR121, April 1991, pp1-43.