Improvement for Reducing Latency of Load Instructions

Wen-Jun Liu Chang-Jiu Chen

Department of Computer Science and Information Engineering
National Chiao Tung Universit

ABSTRACT

*Memory system design is one of the most challenging aspects of comput er architecture. One of the important challenges in memory system design is the problem of continually lengthening load latency. This paper proposes three methods to eliminate load latency : **FAC-like** scheme eliminates one cycle of load latency , **LTAPB** scheme reduces two cycles ,and **Hybrid** scheme combines above two methods to benefit from their advantages. We adopt an execution-driven simulator and apply Spec95 benchmarks to simulate the results.*

Index Terms : Instruction Latency , Fast Address Caculation , Pipeline

1 Introduction

Recentl , many newer emerged workloads, such as compression and multimedia data, lack the characteristic of locality and result in the poor performance of cache memory system. In addition, some trends emphasized on instruction -level parallelism or some multi-scalar processors issue multiple loads and stores per cycle [1], resulting in increased bandwidth demands on the memory system and further aggravate load latency. Therefore, reduce memory latency becomes a more and more important topic to improve the efficiency of load instruction.

Load instructions move data between the memory system and processor. A typical load instruction is analyzed as follows: two inputs, a base address and offset, are added together to form an effective address which is used to access data memory. In most modern architectures, the base is supplied from a processor register and the offset is supplied by either an immediate constant, i.e., register+constant addressing, or via a register, i.e., register+register addressing. For RISC architectures, these two addressing modes are adequate.

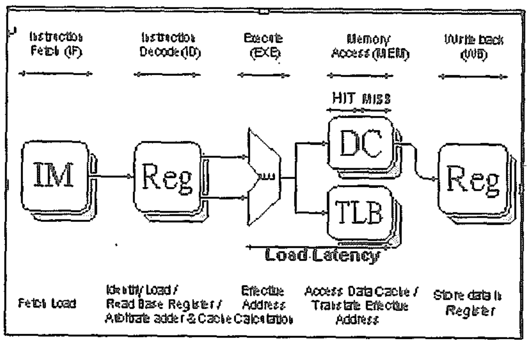


Figure 1 : pipeline Datapath of a Load Instruction and Definition of load Latenc

Figure 1 illustrates the major component operations of a load and their order of execution. Load latency is defined as the time it takes to compute the effective address of the access, access the data memory, and return a result. Obviously in Figure 1, effective address calculation takes a single cycle, and data memory access takes a single cycle if the access hits in the data cache. If the access misses in the data cache, the load latency is further increased by delays incurred with accessing lower levels of the memory hierarchy.

2 Simulation Model

We will use an execution-driven simulator to evaluate the simulation results [8]. Execution-driven simulator acts like an actual processor. It "runs" every instruction in the program, updates the contents of virtual registers, modifies the virtual state of the processor, and simulates the behaviors of functional units and each pipeline stage.

2.1 Simulation Architecture

All experiments were performed with programs compiled for the SimpleScalar architecture. This RISC architecture is derived from the MIPS -IV ISA [5]. The simulator we choose from the SimpleScalar ToolSet [8] is a detailed and complicated one, named ***sim-outorder***. This simulator supports out -of-order issue and execution, based on the Register Update Unit [2]. Figure 2 illustrates the simulated pipeline of the sim-outorder simulator.

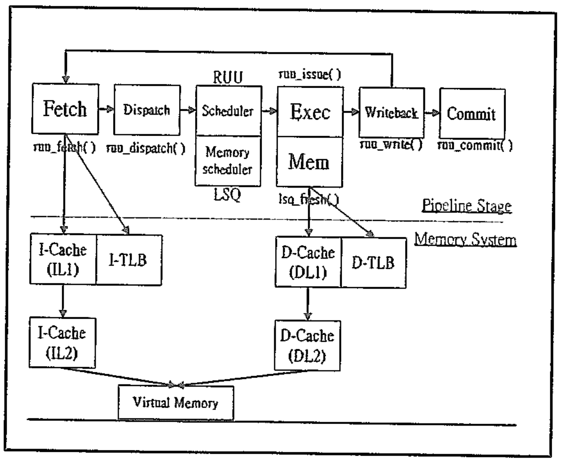


Figure2: Out-of-Order Processor Timing Simulation

Benchmark Programs

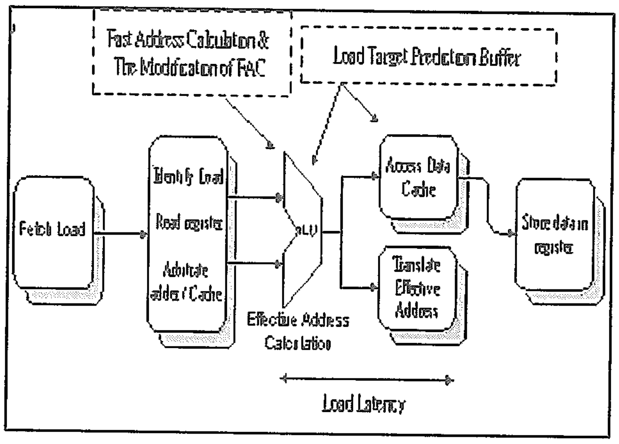
Benchmark	Language	Input
Compress	C	Test.in
GCC	C	Cccp.i
Lisp	C	Test.lsp
M88ksim	C	ctl.raw
Su2cor	Fortran	su2cor.model
SWIM	Fortran	swim.in
Fpppp	Fortran	natoms.in
Hydro2d	Fortran	hydro2d.model

Table 1: Benchmark Programs

Table 1 details the programs analyzed, the language they are written in, and their inputs and options. The top group is integer codes and the bottom group is floating point codes. These eight programs are from the SPEC '95 benchmark suite [6].

3 Approaches for Reducing Load Latency

This section introduces three approaches to reduce load latency. Figure 3 shows the approaches of this paper and the load latency components that each addresses.



Fast Address Calculation & The Modification of FAC
Load Target Prediction Buffer
Fetch Load
Identify Load
Read register
Arbitrate adder / Cache
ALU
Effective Address Calculation
Access Data Cache
Translate Effective Address
Store data in register
Load Latency

Figure 3:The component that each approach addresses

The following subsections give a brief description of each approach.

3.1 FAC-like Scheme

Based on the original *Fast Address Calculation* mechanism [7], we propose a modification named ***FAC-like*** to increase the hit ratio of original design. Figure 4 shows the conceptual model of FAC.

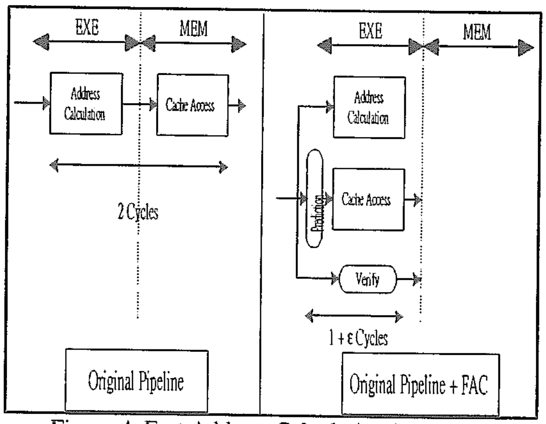


Figure 4: Fast Address Calculation in Pipeline

The idea we adopt in the FAC-like modification is to add an additional prediction circuit decoupled from the normal pipeline desig n. After the base register is read from the register file, the value of base register and the constant offset (known in Instruction Fetch stage) can be sent into the prediction circuit immediately. The prediction circuit is a simple combinational circuit. Thereby the calculation of effective address and validation of prediction circuit can be completed quickly, and then offered control logic to select the prediction result or ignore it. In the following cycle, the data cache access can be performed in time. Figure5 shows the conceptual model.

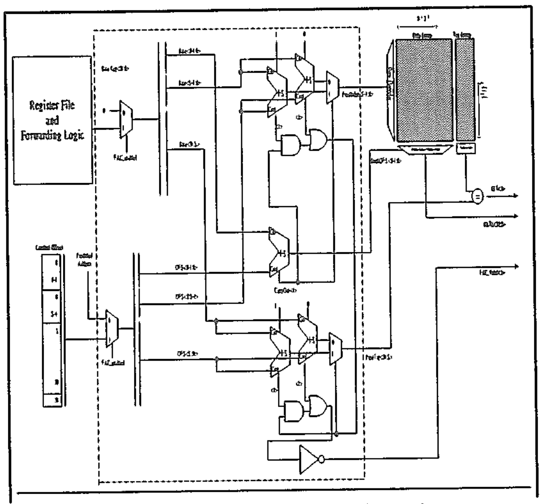


Figure 5: The Circuit of FAC-like scheme

We adopt ***carry-select adder*** [9] illustrated as Figure 6 to accomplish our design. The asymptotic time and space requirements for the carry-select adder are O(√n) and O(n). And n is the bits of precision. We can adopt 32 1-bit carr -select adders to design the prediction circuit or three different carry-select adders for block offset portion, set index portion, and tag portion respectively. This is a cost/performance trade-off problem. As shown in Figure 6, the carr -out of the adder that computes the block offset portion of the effective address is propagated to select the output of the set index portion computation and its carry-out. And then the carry-out of the set index portion adder continues to select the output of the adder that computes the tag portion of the effective address and its carr -out.

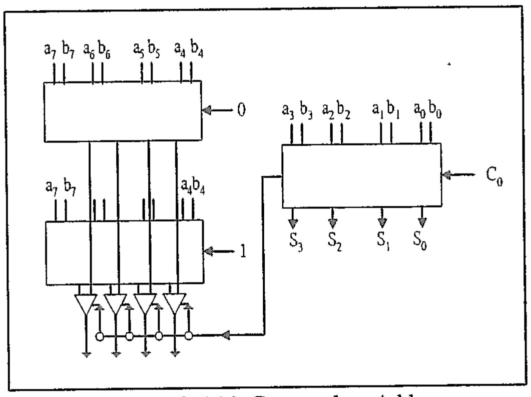

$a_7\ b_7\ a_6 b_6\ \ a_5 b_5\ \ a_4 b_4$
0
$a_3\ b_3\ a_2 b_2\ \ a_1 b_1\ \ a_0 b_0$
C_0
$a_7\ b_7$
$a_4 b_4$
1
$S_3\ \ S_2\ \ S_1\ \ S_0$

Figure 6: 4-bit Carr -select Adder

Since this FAC-like conceptual model is an additional circuit decoupled from the normal pipeline design, there must be a signal to choose the normal effective address or predicted effective address while accessing the data cache in the following clock cycle. Figure 7 shows the conception. There are two inputs that are chosen to access the data c ache. One is the normal effective address computed by the normal ALU logic in EXE (EXEcute) stage when the prediction circuit fails, and the other is the predicted effective address generated from the prediction circuit when the prediction logic succeeds. In addition, the signal ***FAC_Vali*** denotes the validity of the prediction outcome. And the signal ***FAC_enable2*** is applied to enable the selection of predicted effective address.

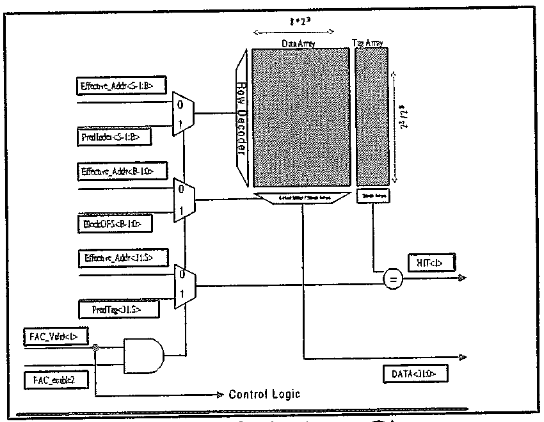


Figure 7: Data Cache Access Diagram

3.2 LTAPB Scheme

FAC-like was introduced as a technique to overlap address calculation with data cache access, thereby eliminating the extra cycle needed for address calculation. For most pipeline designs, address calculation latency comprises at most half of the latency of loads that hit in the data cache, leaving one or more cycles of data cache access latency still exposed to extend execution critical paths or stall instruction issue. Thus we propose this approach named ***LTAPB (Load Target Address Prediction Buffer)*** placed in IF stage of the pipeline to predict the effective address that the load instruction needs to access the data cache.

The LTAPB, loosely based on a branch target buffer [3], uses the address of a load instruction to predict the effective address early in the pipeline. Therefore, the LTAPB is indexed with the virtual address of the load instruction. If the virtual address is hit in LTAPB, the virtual effective target address is extracted from the hit entry of the LTAPB. And the cache access is overlapped with non -speculative instruction decoding operation. Otherwise, the cache is accessed again in EXE stage in later second cycle using the correct effective address. If predict successfully, in the following ID stage, the acquired address can be used to access the data cache up to two cycles earlier than the traditional pipeline path. In other words, the LTAPB can reduce two cycles load latency at the maximum.

With LTAPB, the load instructions produce a result by the time they reach the EXE stage of the pipeline. Subsequent dependent instructions can enter the EXE stage of the pipeline at the same time (with forwarding technique) unencumbered by load instruction hazards. Programs executing on a processor supportingLTAPB will have fewer pipeline stalls caused by load instruction and increased overall performance. Figure 8 shows the conceptual model of LTAPB.

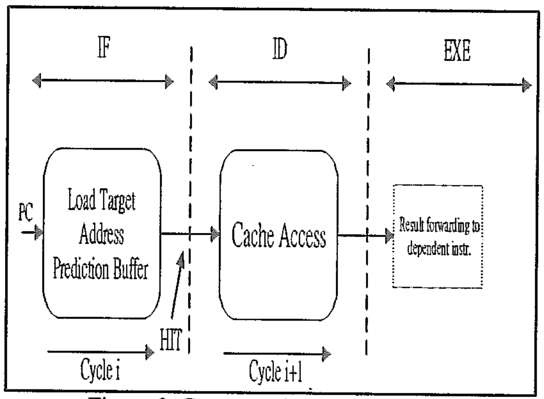

IF
ID
EXE
PC
Load Target Address Prediction Buffer
Cache Access
Result forwarding to dependent instr.
HIT
Cycle i
Cycle i+1

Figure 8: Conceptual Model of LTAPB

The hardware structure we craft for the LTAPB concept is shown in Figure 9. There are six fields in the LTAPB. The value of PC is compared with the value of ***tag*** field to determine a hit/miss. The ***vali*** bit indicates the validation of the entry. The ***reserved*** bit signifies that the entry is reserved to wait for the result of the effective address computation to perform updating when a LTAPB miss is caused by a load instruction. The value of the ***Base_Num*** field denotes the number of base register, providing for dependencies checking. The ***Ref_Count*** field expresses the reference/hit number of the entry. The replacement policy is implemented according to Ref_Count field when the LTAPB is full. At last, the ***Effec_Addr*** field is extracted to apply the data cache access in the following clock cycle.

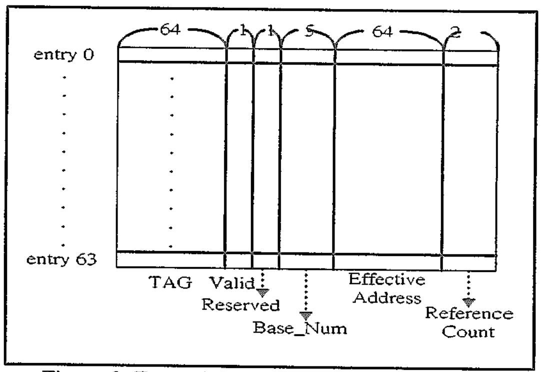


Figure 9: Example Structure Diagram of LTAPB

In the IF stage of the processor, theLTAPB and the instruction cache are accessed in parallel with the address of the current PC. There are three conditions described as follows.

HIT Condition

If the current program counter hits in the LTAPB, the effective address extracted from the hit entry can be used immediately to access the data cache. And the Ref_Count field increases by one.

MISS Condition

If the LTAPB misses, an entry is reserved for this instruction. In the following ID stage, check the dependency relation between the instruction and the LTAPB, and identify whether the instruction is a load instruction. If there are no any existed dependency and this current instruction is not a load instruction, the reserved bit of the reserved entry will be clear or set to zero. If this current instruction is indeed a load instruction, the Effec_Addr field of the entry reserved early in the IF stage will be filled in the end of the EXE stage. The computation result of the effective address is forwarded to update the Effec_Addr field of the reserved entry.

Dependency Resolution Condition

If the LTAPB misses and there are some dependency relations checked out in the ID stage, the central control logic of the LTAPB turns into the dependency checking mode to resolve the dependency relation precisely. "Dependency relation" means that the destination field of the current instruction is the same as one or more entries' Base_Num field. Therefore, the entry must be set invalid (that is, clear the valid bit or set the valid bit to zero) to avoid the abuse of following instructions. And t he Effec_Addr field of the related/conflicted entry must be updated when the computation of the conflicted base register's newer value completes. After updating the Effec_Addr field, the valid bit of the entry is set back to 1 to denote its validity.

3.3 Hybrid Scheme

Since the FAC-like scheme and LTAPB scheme are adopted in distinct stages, they can be directly put together to gain the best efficiency of load latency reduction. The conceptual model is shown in Figure 10.

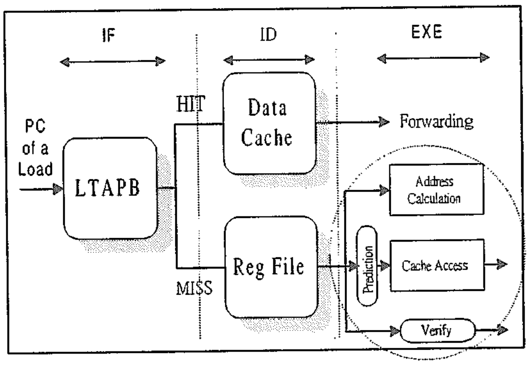


Figure 10: Conceptual Model of Hybrid Approach

The conceptual model of hybrid approach shows the process of a load instruction in the pipeline. If the LTAPB hits, the load will complete in one cycle. If unsuccessful, the load instruction enters the ID stage ordinarily. After decoding, the base register value read from the register file can be used to execute the access using FAC-like in the execute stage. If FAC -like fails, a no -speculative effective address can be computed in the execute stage of the pipeline, with subsequent data cache access in the memory stage. If an interlock condition exists, the load instruction must stall until it clears, at which point the access can proceed, possibly employing FAC (or FAC-like) if the interlock condition clears before address generation completes. In the worse case, the LTAPB will miss, forcing re-execution in the execute stage, where FAC-like will fail, resulting in re-execution in the memory stage of the processor – a worse case latency of two cycles (given that mispredictions can be recovered in the following cycle and there is sufficient data cache bandwidth).

4 Simulation Result

This section gives a description of simulation evaluation result. First of all, we show the percentage of load instruction of total executed instructions in each benchmark we employ in Figure 11. And then the following subsections describe the experiment results briefl .

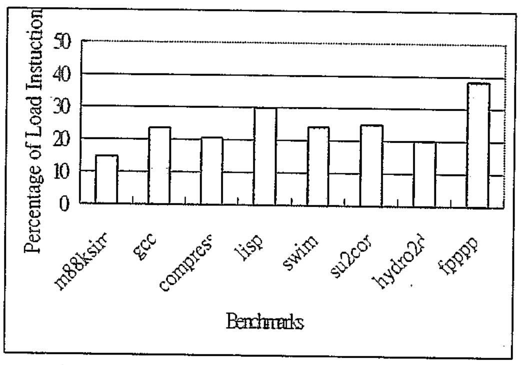


Figure 11: Percentage of Load Instruction in Benchmarks

The left group is for integer codes and the right group is for floating point codes. In this f igure, the speculative loads are not counted. Therefore, the counted load instructions are committed regularly. We make an apparent assumption, that is, the higher the percentage of load instruction, the more influential the load instructions.

4.1 FAC-like Scheme

This section evaluates the effectiveness of FAC -like by examining the efficiency of programs. The analyzed programs include both integer and floating point codes.

Prediction Accuracy

For FAC-like, there is not any address prediction failure in each ben chmark. The only failure situation in FAC-like is when a carry is generated from the effective address calculation. A propagated carry generated from the predictor means the effective address is out of illegal address space. In traditional system, this condition will result in an operating system trap or hardware trap. Generally, this executing program will be aborted. Therefore, our predictor should be tagged as a failure prediction to represent this situation. Consequently, the predictor achieves a zero m isprediction rate in normal circumstances and may improve the program performance significantly.

Program Performance

Prediction performance does not translate directly into program run -time improvements. A successful effective address prediction may or ma not improve program performance, depending on whether or not the access is on the program's critical path. To gauge the performance of FA -like in the context of a realistic processor model, baseline program performance. was compared to the performance of programs running on the baseline timing simulator extended to support FAC-like.

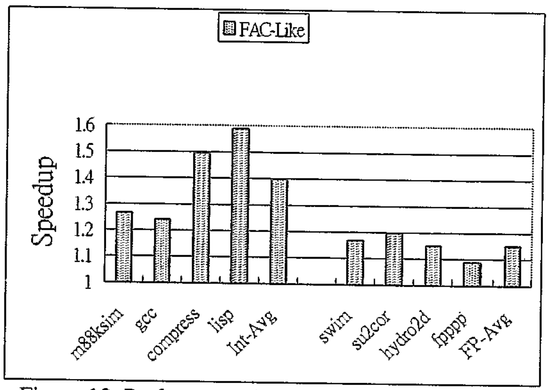


Figure 12: Performance Speedup. Speedups shown are over baseline model execution time.

Figure 12 shows execution speedups with FAC-like hardware support respectively. Shown are the average speedups for the integer and floating point codes, weighted by the run -time (in cycles) of the program. All speedup are computed with respect to the execution time (in cycles) of the baseline program (no FAC-like specific optimizatio ns) running on the baseline simulator.

On the average, FAC-like improves the performance of integer programs by 39%. The floatin -point programs show a smaller speedup of

15.1% for FAC-like. This is a very positive result. One could expect program performa nce to consistently improve.

4.2 LTAPB Scheme

This section examines the performance of programs running on a detailed timing simulator extended to support the LTAPB mechanism. Key parameter, the entry of LTAPB scheme, is varied from 4 to 16, 32, 64 to see what effects these changes had on the efficacy of LTAPB design.

Prediction Accuracy

Figure 13 shows the prediction miss ratio for loads. The figure shows the failure rates as a percentage of total reg+const mode loads when running on the baseline simulator supporting 4-, 16-, 32-, and 64-entry LTAPB, respectively.

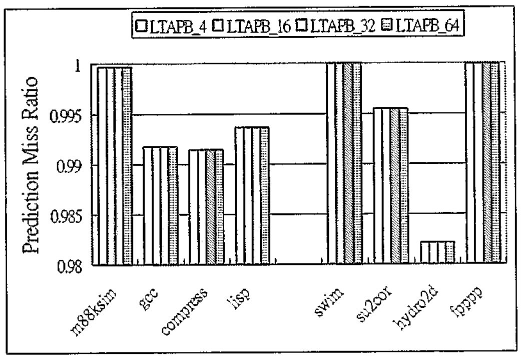


Figure 13: Prediction Miss Ratio for Variable-entry LTAPB

From 4-entry to 64 -entry LTAPB, the miss ratios are all very high. On the average, the miss ratio is more than 99.3% for the in teger codes and 99.9% for the floating point codes. One important reason is that the entry count is too few. However, the entry count of LTAPB is limited from 4 to 64 due to its complicated hardware complexity [4]. Another reason is that we do not classify the load instructions in LTAPB.

Program Performance

Figure 14 shows execution speedups with LTAPB hardware support. Shown are the average speedups for the integer and floating point codes, weighted by the run-time (in cycles) of the program. All speedup are computed with respect to the execution time (in cycles) of the baseline program (no LTAPB specific optimizations) running on the baseline simulator.

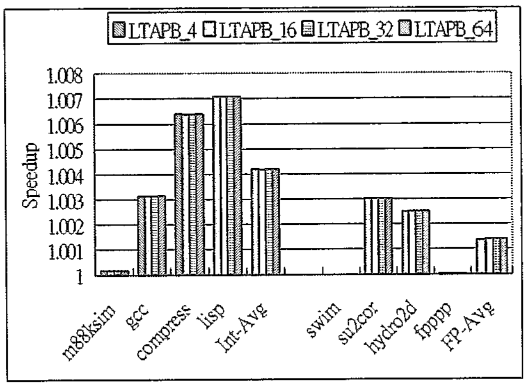


Figure 14: Speedup of Baseline Simulator with Variable-entry LTAPB

On the average, the LTAPB scheme i mproves the performance of integer codes by about 0.42%. The floatin - point codes have much smaller improvement of about 0.14%. The speedup of swim and fpppp is approximate to be zero due to its very high prediction miss ratio.

4.3 Hybrid Scheme

Baseline simulator extended to support LTAPB and FAC-like altogether is applied to evaluate the hybrid method.

Program Performance

Figure 15 shows the speedup for hybrid method. Predictably, the baseline simulator supporting FAC-like and 64 -entry LTAPB has the best av erage speedup of about 42% for integer codes. For the floatin -point codes, the speedup was smaller than integer codes. The floating-point codes have many other long latencies, such as float point computation and cache misses, which are effectively tolerat ed by the out-of-order issue mechanism, but benefit little from FAC-like+LTAPB load support. Shown in this figure, the influence of variable LTAPB entry is tiny.

Thus the major performance promotion benefits from supporting the FAC -like scheme.

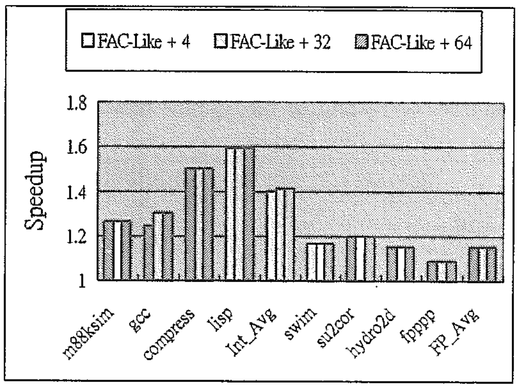


Figure 15: Speedup of Baseline Simulator with Variable
Hybrid Methods for Integer Codes

5 Conclusions

For many codes, especially integer codes with good cache performance, exposed address calculation latencies account for a significant fraction of total execution time. This paper introduces a pipeline optimization, called FAC-like, that permits effective address calculation to proceed in parallel with data cache access, thereby eliminating the extra cycle required for address calculation. The technique employs a simple circuit to quickly predict the portion of the effective address needed to access the data cache. If the address is predicted correctly, the cache access completes without an extra cycle for address calculation. If the address is mispredicted, the cac he is accessed again using the correct effective address. The predictor is an additional combinational circuit decoupled from the traditional pipeline path, therefore, has minimal impact on cache access latency. This predictor applies carr -select adders to calculate the effective address. Prediction verification is very easy to be examined. Only one bit signal is acquired to signify the validation of predictor, ensuring that pipeline control logic impacts are minimal.

FAC-like, however, is at most one hal of the latency of loads, leaving one or more cycles of cache access latency exposed to extend execution critical paths and stall instruction issue. This paper further enhances the latency reduction capability to completely hide the latency when this load instruction hits in accessing data cache. The resulting designs, named LTAPB, are capable of reducing the latency of load instructions by up to two cycles. For a pipeline with one cycle data cache access, loads can complete before reaching the execute stage of the pipeline. This scheme allows subsequent dependent instructions to issue unencumbered by load instruction hazards, resulting in fewer pipeline stalls and increased overall performance.

The major problem of LTAPB is the very low hit ratio. We have some alternatives to improve LTAPB scheme. One of them is to classify the load instructions. For the load instructions, some particular registers are used very often, such as stack point, frame point, and global point. If we augment additional spaces for t hese distinctive registers, the utilization of LTAPB may be improved greatly. Another alternative method is to tightly design the LTAPB as the BTB structure. So the entry count of LTAPB can be raised to 512, 1024, or even more. Intuitively, the prediction accuracy may be promoted. The hurdle is the recovery procedure needed to straighten the entire system to act precisely. If the LTAPB hits but the target address is not the "real" effective address, there must be a recovery mechanism to squash back to the precise checkpoint. This recovery mechanism is similar as the one employed in the branch prediction schemes.

6 References

[1] Gurindar S. Sohi," *Instruction Issue Logic for High-Performance," Interruptible, Multiple Functional Unit, Pipelined Computers* . IEEE Transactions on Computers, 39(3):349-359, March 1990

[2] Mike Johnson , *Superscalar Microprocessor Design*, Prentice Hall, New Jersey, 1991.

[3] C.H. Perleberg, and A.J. Smith, "Branch Target Buffer Design and Optimization," IEEE Transactions on Computer, Vol. 42, No. 4, April 1993.

[4] J.L. Hennessy, and D.A. Patterson, *Computer Organization and Design*, Margon Kaufmann Publishers, Inc., 1994.

[5] Charles Price. *MIPS IV Instruction Set. Revision 3.1.* MIPS Technologies, Inc., Mountain View, CA, January 1995.

[6] *SPEC95 Benchmark Suite Release 1.0* , September 1995.

[7] Todd Michael Austin, *Hardware and Software Mechanisms for Reducing Load Latency* , University of Wisconsin – Madison, 1996.

[8] Doug Burger, Todd M. Austin, and Steven Bennett. *Evaluating Future Microprocessors: the SimpleScalar Tool Set*. Technical Report 1308, Computer Science Department, University of Wisconsin, Madison, WI, July 1996.

[9] J.L. Hennessy, and D.A. Patterson, *Computer Architecture: A Quantitative Approach*, Margon Kaufmann Publishers, Inc., 1996.