

METHOD AND APPARATUS FOR UTILIZING CACHE MISS LATENCY USING PRIORITY BASED EXECUTION

CROSS-REFERENCE TO RELATED PATENT APPLICATION

5

This application claims the benefit of Korean Patent Application No. 10-2008-0083540, filed on August 26, 2008, in the Korean Intellectual Property Office, the disclosure of which is incorporated herein in its entirety by reference.

10

BACKGROUND OF THE INVENTION

1. Field of the Invention

15

The present invention relates to utilization of cache memory latency, and more particularly, to a method and apparatus for utilizing cache miss latency by using a priority based execution for an embedded processor.

2. Description of the Related Art

20

Embedded processors today are facing increasingly larger memory latencies. With the growing disparity between processors and memory speeds, the operations that cause cache misses out to main memory take hundreds of processor cycles to complete execution. As memory access operations result in approximately 30-40% of the total of instructions in applications, reducing the average memory access time is crucial for any type of architectures. While high-end processors can use elaborate techniques like multiple issue, out-of-order execution, speculative execution, value prediction, etc., to tolerate the high memory latencies, it is often not a viable solution for the embedded processors due to the significant overheads of these techniques in terms of area, power, and chip complexity.

25

30

The fundamental problem with the single-issue processors is that the instructions which have long latency stall the pipeline, and do not allow the incoming instructions to be executed. Hardware solutions, e.g., out-of-order and speculation mechanisms perform the complex analysis to discover the independent operations in the schedule and execute them. The hardware required for the analysis often consumes too much power and has complex logic.

The most existing embedded processors are single-issue and non-speculative processors. Therefore, embedded processors require alternative mechanisms to hide the memory latency with minimal power and area costs.

SUMMARY OF THE INVENTION

The present invention provides a method and apparatus for utilizing cache miss latency by using a priority based execution so that the cache miss latency for an embedded processor is efficiently utilized with minimal power and area.

According to an aspect of the present invention, there is provided a method of utilizing cache miss latency by using a priority based execution, the method comprising: compile a program; classifying high priority instructions and low priority instructions in the compiled program; executing the classified instructions based on high priorities; and if a cache miss occurs, executing the low priority instructions until the cache miss is resolved.

According to another aspect of the present invention, there is provided an apparatus for utilizing cache miss latency by using a priority based execution, the apparatus comprising: an instructions classifying unit classifying instructions compiled by a compiler into low priority instructions and high priority instructions; and an instructions processing unit executing the classified instructions based on high priorities, and, if a cache miss occurs, executing the low priority instructions until the cache miss is resolved.

BRIEF DESCRIPTION OF THE DRAWINGS

The above and other features and advantages of the present invention will become more apparent by describing in detail exemplary embodiments thereof with reference to the attached drawings in which:

FIG. 1 is a block diagram of an apparatus for utilizing cache miss latency by using a priority based execution according to an embodiment of the present invention;

FIG. 2 is a block diagram of an instructions processing unit according to an embodiment of the present invention;

FIG. 3 is a flowchart illustrating a method of utilizing cache miss latency by using a priority based execution according to an embodiment of the present invention;

FIG. 4 illustrates an assembly instructions code and a data flow graph (DFG) according to an embodiment of the present invention; and

FIG. 5 is an exemplary diagram of an instructions processing unit of the apparatus for utilizing the cache miss by using the priority based execution according to an embodiment of the present invention.

DETAILED DESCRIPTION OF THE INVENTION

Hereinafter, the present invention will be described in detail by explaining embodiments of the invention with reference to the attached drawings.

FIG. 1 is a block diagram of an apparatus for utilizing cache miss latency by using a priority based execution according to an embodiment of the present invention. Referring to FIG. 1, the apparatus comprises an instructions classifying unit 120 and an instructions processing unit 140.

The instructions classifying unit 120 classifies instructions compiled by a compiler 100 into low priority instructions and high priority instructions. All load instructions, all branch instructions, and all instructions that define operands for the high priority instructions are classified as high priority instructions. To support priority classification, 1-bit priority information may be added to all the instructions. For example, if 0 is added to an instruction, the instruction may be classified into the low priority instructions, and if 1 is added to an instruction, the instruction may be classified into the high priority instructions.

The instructions processing unit 140 executes the classified instructions based on high priorities. If a cache miss occurs, the instructions processing unit 140 executes the low priority instructions until the cache miss is resolved.

FIG. 2 is a block diagram of the instructions processing unit 140 according to an embodiment of the present invention. The instructions processing unit 140 comprises a rename table 220, a reorder buffer (ROB) 200, a rename manager 210, a priority selector 230, a multiplexer 240, and an instructions executer 250.

The rename table 220 stores mapping information of a real register into a renamed register.

The ROB 200 stores the classified instructions. The instructions stored in the ROB 200 may include the 1-bit priority information indicating that a corresponding instruction is the high priority instruction or the low priority instruction.

The rename manager 210 renames names of the instructions stored in the ROB 200. When the instructions are stored in the ROB 200 from decode stage, the rename manager 210 finds out the instructions which define the source operands of the low priority instructions, and renames the source operands and the destination
5 operands of the low priority instructions by making reference to the rename table 220.

The priority selector 230 compares a source register with a register which misses a cache and determines an execution mode of a processor. The processor has two execution modes of a high priority execution mode and a low priority
10 execution mode. In the high priority execution mode, only the high priority instructions are executed. The operands of the low priority instructions are renamed and the instructions are stored for a while in the ROB 200. As a result of comparing the source register with the register which misses the cache, when a data cache miss occurs, the processor switches to the low priority execution mode and
15 controls the multiplexer 240 to start executing the low priority instructions. During the low priority execution, if the cache miss is resolved, the processor controls the multiplexer 240 to immediately resume the high priority execution mode.

The multiplexer 240 selectively outputs the instructions stored in the ROB 200 according to the control of the priority selector 230. For example, if the cache miss
20 occurs, the multiplexer 240 outputs the low priority instructions.

The instructions executor 250 executes the instructions output by the multiplexer 240, and, if the cache miss occurs, outputs the register which misses the cache.

FIG. 3 is a flowchart illustrating a method of utilizing cache miss latency by using a priority based execution according to an embodiment of the present
25 invention.

Referring to FIG. 3, in operation 300, a program is compiled. In operation 310, the compiled instructions are classified into high priority instructions and low priority instructions. In operation 320, the classified instructions are executed based
30 on high priorities. In this regard, the low priority instructions are for a while stored in the ROB 200.

In operation 330, it is determined whether a cache miss occurs. If it is determined that the cache miss occurs, in operation 340, the low priority instructions

are executed. In operation 350, if the cache miss is resolved, the high priority instructions are immediately resumed to be executed in operation 320.

After the instructions are executed based on their priorities, in operation 360, if there are no more instructions to execute, the processor finishes the priority based execution.

Meanwhile, pending low priority instructions are executed at the end of the execution of a loop.

The method and apparatus for utilizing cache miss latency by using the priority based execution of the present invention will now be described in more detail. FIG. 4 illustrates an assembly instructions code and a data flow graph (DFG) according to an embodiment of the present invention. Referring to FIG. 4, the assembly instruction code shows the innermost loop in the compress benchmark from multimedia benchmarks.

The high priority instructions are those which are required to continue the operations of instruction fetch and data transfers. The high priority instructions are defined as: the branch instructions, all load instructions, and all instructions that generate the source operands for a high priority instruction are high priority instructions.

All branch instructions are the high priority instructions because if the branch instructions are not executed, it is not known what to fetch next. Similarly, all load instructions are the high priority instructions to keep a bus between the processor and a data memory busy. In addition, all instructions that generate the source operands for the high priority instructions also need to be executed with high priority. All the other instructions in a loop are the low priority instructions.

The priority of the instructions is determined as following steps. In a first step, all load and branch instructions of a loop are marked. In a second step, user define (UD) chains are used to find instructions that define the operands of already marked instructions and to mark them. In a third step, conditional instructions of which already marked instructions and control-dependent on are found and marked. In a fourth step, the steps are recursively continued until no more instructions can be marked.

In a fifth step, the marked instructions are high priority, and unmarked instructions are low priority.

A low priority selection algorithm operates on a DFG of the loop. The loop is

represented as a DFG $G = (N, E)$, where each $n \in N$ is an operation in the loop body. There exists a directed edge $e = (n_i, n_j) \in E$ if operation n_i of any loop iteration is data dependent on operation n_j of any loop iteration. Thus there will be a directed edge between two operations even if they have a loop carried dependency.

5 The DCF shown in FIG. 4 shows a data dependence graph of the innermost loop of the compress benchmark. The annotation on each edge is a dependent operand.

All the load instructions are considered as the high priority instructions. Therefore, instructions 1, 2 and 3 are high priority instructions. The branch instruction at the end of the loop is important to continue to fetch new instructions.

10 Therefore, instruction 10 is high priority instruction. Then more high priority instructions are detected by using second and thirds steps. All instructions which compute the source operands of branch are also high priority instructions.

Instruction 7 subtracts register $r7$ and sets a zero flag in the condition code, which is used by the branch instruction 10 to decide an execution flow. In addition,

15 instruction 9 computes the register $r0$, which will be used by the high priority instructions, 1, 2, and 3, therefore instruction 9 is also a high priority instruction.

Finally, among the rest of the instructions, the memory operands of store instructions are checked. If the store instruction feeds a high priority load, it is not marked as a low priority instruction. Referring to FIG. 4, the instructions 4, 5, 6 and

20 8 are marked as low priority instructions.

The instructions only in the loops are considered. When the processor finishes the execution of a loop, the processor flushes an instruction window by executing the pending low priority instructions. Since a cache miss rarely occurs outside the loop body, all the pending low priority instructions are executed rather than let them wait in the ROB until a next cache miss. For a transformation, there should be no other branch instructions inside the loop. If there are branch constructs like “if-then-else”, they have to be converted into a conditional execution, or predicated instructions. Predication is a mechanism of converting control dependencies into data dependencies, and allows more candidates of the low

30 priority instruction in the loop to be considered.

For the memory operations, a compiler statically disambiguates the memory dependency. If the static memory disambiguation approaches cannot disambiguate memory operations in the loop, those operations are not considered as the low priority instructions. It is worth mentioning that the priority based execution

technique is orthogonal to the memory disambiguation; memory latency can be hidden even without memory disambiguation, but it can perform better if memory dependencies can be further disambiguated.

To support flushing operation and priority classification, it is necessary to modify an instruction set architecture (ISA). 1-bit priority information is annotated for every instruction to make the processor perform a priority based execution, and a *flushLowPriority* instruction is added to execute the pending low priority instructions at the end of the loop execution.

The priority based instruction execution of the present invention will now be described. The method of utilizing the cache miss latency by using the priority based execution uses two modes of the execution; the high priority execution mode and the low priority execution mode.

In the high priority execution mode, only the high priority instructions are executed. The operands of the low priority instructions are renamed and the instructions are stored for a while in a ROB. When a data cache miss occurs, the high priority instructions are flushed from a processor pipeline. Then, the processor switches to the low priority execution mode and starts executing the low priority instructions. During the low priority execution, if the cache miss is resolved, the processor immediately resumes the high priority execution mode. If there are no more low-priority instructions to execute even before the cache miss is serviced, the processor switches back to high priority execution mode and stalls until the miss is resolved.

The low priority instructions cannot stall the processor pipeline due to the cache miss since no load instruction is low priority. This scheme results in performance improvements due to two main reasons. Firstly, since the low priority instructions are skipped during the loop iteration, the number of effective instructions in a loop is reduced and therefore the loop executes faster. Secondly, the low priority instructions are executed on the cache miss and the cache miss latency is therefore utilized for a useful work.

The processor micro-architecture should provide support for renaming a full register so as not to violate the dependencies between the instructions.

FIG. 5 is an exemplary diagram of an instructions processing unit of the apparatus for utilizing the cache miss by using the priority based execution according to an embodiment of the present invention. Referring to FIG. 5, the instructions

processing unit comprises an ROB, a rename manager, a rename table, a priority selector, a MUX, a memory unit, and an FU. The memory unit and the FU correspond to the instructions executor 120 shown in FIG. 2.

The ROB contains decoded instructions and an extra 1-bit indicator P . The indicator '0' means the low priority. The indicator '1' means the high priority. The rename table keeps mapping information of a real register into a rename register. The instructions in the ROB are renamed dynamically by the rename manager by using information stored in the rename table.

The priority selector consists of 3 comparators that compare the source registers with a register which misses a cache, and decides the execution mode of the processor.

When the instructions are stored in the ROB from a decode stage, the rename manager first finds out the instructions which define the source operands of the low priority instructions. Then the rename manager renames the source operands of the low priority instruction and the destination operands of the parent instructions according to the rename table.

There are two cases that the rename manager can encounter; a parent instruction which defines the source operand resides in the ROB, and the parent instruction has already been issued to the operation bus and is not in the ROB. In the second case, to suspend the execution of the low priority instructions, values of the source registers are preserved before the values are changed by the already issued instruction. To do this, a *mov* instruction which shifts the values from the real register to the rename register is inserted to the ROB. Then the source operand of the low priority instruction is renamed.

In this manner, the low priority instructions are accumulated in the ROB during the high priority execution mode. The low priority instructions start their execution if the priority selector generates '0' and gives it to the MUX, or if the ROB is full of the low priority instructions (even if the cache miss has not occurred yet). During the high priority execution, the priority selector keeps generating '1' until the cache miss occurs. When the processor cannot issue the instruction, the priority selector gives inverted P to the MUX, and the low priority instructions start their executions. The low priority execution continues until the missed value is available in the cache. When the priority selector detects that the cache miss is now serviced, the priority selector generates a high priority signal to the MUX and the processor resumes the

high priority execution.

The present invention can also be embodied as computer readable code on a computer readable recording medium. The computer readable recording medium is any data storage device that can store data which can be thereafter read by a computer system. Examples of the computer readable recording medium include read-only memory (ROM), random-access memory (RAM), CD-ROMs, magnetic tapes, floppy disks, optical data storage devices, and carrier waves.

The method and apparatus for utilizing cache miss latency by using a priority based execution according to the present invention can increase system performance by using efficiently utilizing memory latency with minimal power and area costs.

While the present invention has been particularly shown and described with reference to exemplary embodiments thereof, it will be understood by those of ordinary skill in the art that various changes in form and details may be made therein without departing from the spirit and scope of the present invention as defined by the following claims.

What is claimed is:

1. A method of utilizing cache miss latency by using a priority based execution, the method comprising:

compile a program;

5 classifying high priority instructions and low priority instructions in the compiled program;

executing the classified instructions based on high priorities; and

if a cache miss occurs, executing the low priority instructions until the cache miss is resolved.

10 2. The method of claim 1, wherein the classifying comprises:

marking load and branch instructions as the high priority instructions; and

finding instructions that define the operands of the instructions marked as the high priority instructions and marking them as the high priority instructions.

15 3. The method of claim 1, wherein 1-bit priority information is annotated for each of the instructions.

4. The method of claim 1, wherein the low priority instructions are stored
20 in a buffer during the execution of the high priority instructions.

5. The method of claim 1, wherein the executing of the classified instructions comprises: executing pending low priority instructions at the end of the execution of a loop.

25 6. An apparatus for utilizing cache miss latency by using a priority based execution, the apparatus comprising:

an instructions classifying unit classifying instructions compiled by a compiler into low priority instructions and high priority instructions; and

30 an instructions processing unit executing the classified instructions based on high priorities, and, if a cache miss occurs, executing the low priority instructions until the cache miss is resolved.

7. The apparatus of claim 6, wherein the instructions processing unit comprises:

a rename table storing mapping information of a real register into a renamed register;

5 a reorder buffer (ROB) storing the classified instructions;

a rename manager renaming the instructions stored in the ROB;

a priority selector comparing a source register with a register which misses a cache and deciding an execution mode of a processor;

a multiplexer selectively outputting the instructions stored in the ROB

10 according to the control of the priority selector; and

an instruction executor executing the instructions output by the multiplexer and, if a cache miss occurs, outputting the register which misses the cache.

8. The apparatus of claim 7, wherein the instructions stored in the ROB
15 include 1-bit priority information indicating that the instructions are high priority instructions or low priority instructions.