(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 November 2010 (04.11.2010)

PCT

(10) International Publication Number
WO 2010/125336 A1

(51) International Patent Classification:
G06F 9/38 (2006.01)

(21) International Application Number:
PCT/GB2010/000832

(22) International Filing Date:
27 April 2010 (27.04.2010)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
0907286.9 28 April 2009 (28.04.2009) GB

(71) Applicant *(for all designated States except US)*: **IMAGINATION TECHNOLOGIES LIMITED** [GB/GB]; Home Park Estate, Kings Langley, Hertfordshire WD4 8LZ (GB).

(72) Inventor; and

(75) Inventor/Applicant *(for US only)*: **WEBBER, Andrew, David** [GB/GB]; 17 Fields End, Tring, Hertfordshire HP23 5ER (GB).

(74) Agent: **REDDIE & GROSE**; 16 Theobalds Road, London WC1X 8PL (GB).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *with international search report (Art. 21(3))*

— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

(54) Title: METHOD AND APPARATUS FOR SCHEDULING THE ISSUE OF INSTRUCTIONS IN A MULTITHREADED MICROPROCESSOR

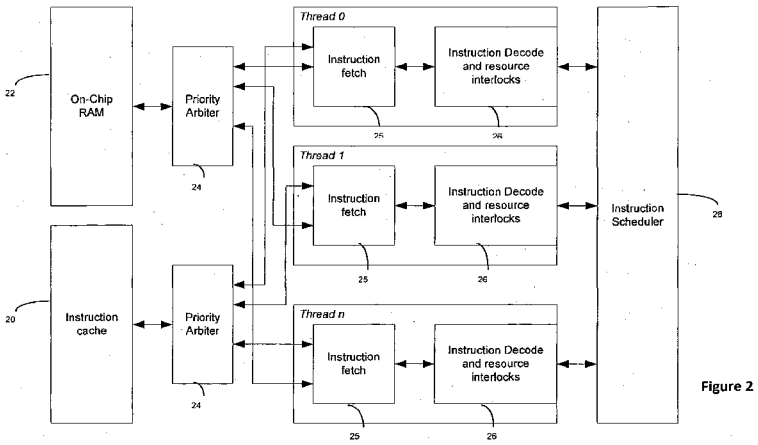

Figure 2

(57) Abstract: There is provided a method to dynamically determine which instructions from a plurality of available instructions to issue in each clock cycle in a multithreaded processor capable of issuing a plurality of instructions in each clock cycle, comprising the steps of: determining a highest priority instruction from the plurality of available instructions; determining the compatibility of the highest priority instruction with each of the remaining available instructions; and issuing the highest priority instruction together with other instructions compatible with the highest priority instruction in the same clock cycle; wherein the highest priority instruction cannot be a speculative instruction. The effect of this is that speculative instructions are only ever issued together with at least one non- speculative instruction.

WO 2010/125336 A1

Method And Apparatus For Scheduling The Issue Of Instructions In A Multithreaded Microprocessor

Field of the Invention

The invention relates to a method and apparatus for dynamically optimising the issue of instructions in a multithreaded microprocessor. Specifically, the invention relates to the issuing of speculative instructions in a multithreaded microprocessor which is capable of issuing instructions from more than one thread in each clock cycle.

Background of the Invention

A multi-threaded processor is a microprocessor with execution units, caches and memories, etc. as per any other processor, but that additionally incorporates the concept of multiple threads of execution. A thread of execution consists of a stream of instructions acting upon the resources of the microprocessor, some of which are private to that specific thread and some of which may be shared between threads. Resources include registers, bandwidth and space in the cache or memories. Figure 1 is a schematic illustration of a multithreaded microprocessor core, known in the art. In a multithreaded processor core as shown in Figure 1, a number of threads 2 can operate in parallel, subject to their ability to issue instructions via the thread instruction scheduler 1. The threads may be given a priority order to allow the scheduler to determine which instructions to issue first.

It is possible for the processor to issue instructions from multiple threads in the same clock cycle provided that they do not use the same resources, such as the same data cache 3, at the same time. The scheduler must be configured to determine which instructions can be issued together in each clock cycle.

Speculative instructions are a particular type of instruction. The concept of speculative instruction fetching and issue is well understood in the field of microprocessors. Speculative instruction fetch and issue takes the form of making a prediction of the outcome of an earlier program instruction and then fetching and issuing instructions based on that prediction before it is known whether the prediction was correctly made. The most common example of an

action to predict the outcome is of a branch in a program, where a choice is made as to what code path to take based on a conditional test. Other examples include return predictions and jump predictions. If it is later determined that the prediction was correct, the processor continues. If it is determined that the prediction was incorrect, then it is necessary to delete the predicted instructions and their side effects, and restart the instruction fetch and issue from the newly determined instruction fetch address.

The term speculative is used because the processor needs to act in advance of knowing the outcome of any recently issued instructions. For example, a branch may well follow immediately after a compare or condition testing instruction. In a pipelined processor the pipeline may be many stages long and the result of the condition test typically produces a result a long way down the pipeline. However, branches directly affect instruction fetch and will therefore have an impact right at the start of the pipeline. This leads to a bubble or stall as the branch has to wait for the condition test result to be known, and the alternative to this wait is to guess which branch is likely to be taken and take that guess as the route to follow until it is known whether the guess was right or wrong. The instructions fetched and/or issued between guessing and knowing the correctness of that guess are not necessarily correct and may have to be unwound – hence the term speculative. Guesses are either made based upon simple rules (e.g. backwards branches taken, forwards branches not taken) or historical data that indicates what a given section of code did most commonly in the past.

As used herein, the term "speculative instruction" means an instruction fetched and/or issued that may not be necessary, because the outcome of an earlier program instruction is not yet known.

The present invention deals with the problem of how to schedule the issue of speculative instructions in a processor that can issue instructions from multiple threads in the same clock cycle.

Summary of the Invention

According to a first aspect of the invention, there is provided a method to dynamically determine which instructions from a plurality of available instructions to issue in each clock cycle in a multithreaded processor capable of issuing a plurality of instructions in each clock cycle, comprising the steps of:

determining a highest priority instruction from the plurality of available instructions;

determining the compatibility of the highest priority instruction with each of the remaining available instructions, wherein instructions are compatible with each other if they do not require the same resources; and

issuing the highest priority instruction together with other instructions compatible both with the highest priority instruction and with each other in the same clock cycle;

wherein the highest priority instruction cannot be a speculative instruction.

The effect of this is that speculative instructions are only ever issued together with at least one non-speculative instruction.

According to a second aspect of the invention, there is provided a system for issuing instructions in a multithreaded processor capable of issuing a plurality of instructions in each clock cycle, comprising:

means for determining a highest priority instruction from the plurality of available instructions;

means for determining the compatibility of the highest priority instruction with each of the remaining available instructions, wherein instructions are compatible with each other if they do not require the same resources; and

means for issuing the highest priority instruction together with other instructions compatible both with the highest priority instruction and with each other in the same clock cycle;

wherein the highest priority instruction cannot be a speculative instruction.

Preferred embodiments of the present invention address the problem of efficient scheduling of speculative instructions in a multithreaded microprocessor capable of issuing instructions from a plurality of threads on each clock cycle.

This and other aspects of the invention are defined in the appended claims, to which reference should now be made.

Brief description of the drawings

There now follows a detailed description of a preferred embodiment of the invention, which is provided by way of example with reference to the accompanying drawings, in which:

Figure 1 illustrates a multithreaded processor core;

Figure 2 illustrates an instruction fetch subsystem suitable for use in a system in accordance with the present invention;

Figure 3 illustrates a system for speculative instruction fetching;

Figure 4 illustrates a system for speculative instruction fetch and execution in accordance with the present invention; and

Figure 5 illustrates the process steps taken to schedule the issue of instructions in accordance with an aspect of the invention.

Detailed description

Figure 1 is a schematic illustration of a multithreaded microprocessor. The microprocessor supports a plurality of threads that operate in parallel, subject to their ability to issue instructions via the instruction scheduler 1. In this system, different threads may issue instructions simultaneously provided that they are not attempting to use the same resources at the same time. The resources

used by thread instructions include arithmetic units 4, 5, the cache subsystem 3, including instruction cache and data cache, and the coprocessors. In addition, the system may include small quantities of on-chip random access memory (RAM) that can be accessed instead of the cache subsystem based upon a simple address mapping. Systems may incorporate instruction RAM as well as data RAM.

The microprocessor includes a number of pipelines or execution units that operate in parallel. If one thread is issuing instructions to one pipeline, it is possible to allow other threads to issue instructions to other pipelines on the same clock cycle. In order to do this, instructions from multiple threads have to be available to the scheduler at each clock cycle and the scheduler must be able to prevent any conflicting resource requirements.

Figure 2 illustrates the elements involved in fetching instructions in order that instructions from a plurality of threads are available to the scheduler every clock cycle. This sub-system is described in co-pending application GB0802314.5. In the embodiment illustrated in Figure 2, there are two main sources of instruction data – the instruction cache 20 and the on chip RAM 22. To optimize performance, both sources may be accessed at the same time (although each thread will only access one or the other, never both). Having two sources of instruction data enables up to two instructions for two different threads to be obtained per cycle.

In order to determine which thread is to access instructions from either the instruction cache 20 or the on chip RAM 22, each source of instructions has a priority arbiter 24 coupled between it and the instruction fetch units 25 for each of the threads.

Each priority arbiter is coupled to an instruction fetch unit 25 for each of the threads (thread 0 to thread n). Via this connection, the priority arbiter can receive requests for instructions with associated prioritisation data and can provide fetched instructions to an instruction fetch unit 25.

Each instruction fetch unit 25 includes an instruction buffer which can hold up to eight instructions in readiness to feed to an instruction decoder and resource interlocking unit 26 before issuing the instruction to an instruction scheduler 28 which will pass each instruction in turn to a processor for execution.

Having obtained instructions for multiple threads the instruction scheduler 28 may then be in a position to issue those instructions for multiple threads on the same cycle.

In order for the instruction scheduler 28 to be able to issue instructions from multiple threads in the same clock cycle, it needs to test the resources required by each thread's instruction against those needed for other instructions. This then ensures that conflicting instructions do not issue at the same time. This can be managed by the instruction decode unit 26 performing pre-decode of each instruction as it comes out of the corresponding instruction fetch unit 25 and then ANDing the stated resources required for this instruction against those determined in parallel for all the other threads active in the device.

A reduced set of these resource requirements may be tested. Picking a simple example, a thread may wish to use an execution unit to send a request for a data memory fetch to load or store some data. If another thread wishes to use either the same execution unit or wishes to use the data memory port for the data memory fetch, then it will be incompatible with execution of the first thread such that the two cannot be issued at the same time. If, however, another thread wishes to perform a set of different instructions such as a program branch which does not require those resources, it will be deemed to be compatible and may be issued for execution at the same time as a thread is used to request a data memory fetch.

In order to make an appropriate determination, all of the threads present have all of their stated resource requirements tested against each other in parallel. This process is symmetrical, in that if a thread A is compatible with a thread B then by definition thread B will also be compatible with thread A. Therefore, the test only needs to be made once between each pair of threads. As a result of this compatibility testing, a flag is generated for each pairing of threads and the

status of this flag is determines whether or not threads are compatible. Therefore, the result of testing each thread against all other threads is that for each pair of threads a compatibility flag is generated, the status of which determines whether or not threads are compatible for execution, i.e. whether or not only two threads overlapping resource requirements at the same time.

In order to determine which instructions to then issue, each thread is given a priority ranking. The priority ranking may be based on any suitable metric, such as the automatic MIPS allocation (AMA) metric described in EP1639554. The thread with the highest priority ranking that is not prevented from issuing instructions due to register or resource interlocks, is then chosen as the thread to issue. Any other threads that are found not to require the same resources may be grouped with the instruction from the highest priority thread that is free to issue (called the lead instruction) into a set of instructions to be issued on that clock cycle. Instructions may be added to the set in priority order subject to their compatibility with instructions already included in the set.

The determination of the resources required by each instruction is carried out by an instruction pre-decode operation in the instruction decode unit 26. The decode unit 26 produces a short summary of the resources required by each instruction. The instruction scheduler then tests the resource requirements against other threads for inter-thread compatibility and against machine state for resource and register availability. The scheduler combines the results of these tests with the priority ranking to provide final sets of instructions to schedule on any given cycle.

The concept of 'speculation' is well understood within the art. This takes the form of making a prediction and then issuing instructions based upon that prediction before it can be known whether that prediction was made correctly. The most common example of an action to predict the outcome of an instruction is a branch where a choice is made as to what code path to take based upon a conditional test. Speculation only applies for short periods, after which the outcome of the instruction is known and normal scheduling can resume.

In accordance with an aspect of the present invention, instructions that are speculatively executed do not use the normal thread instruction prioritisation rules described above. They should not use these rules because it is not known at the point of issue whether the speculation is correct or not. An alternative to using the normal prioritisation rules is to give these threads the minimum priority. However, this can lead to a number of design challenges. Therefore the preferred arrangement is that speculatively executed instructions should have no opportunity to issue except when paired with another thread.

A preferred example of the invention consists of a multi-threaded microprocessor with two or more threads (specifically 2, 3 or 4 threads). Each thread in the microprocessor possesses its own program counter (PC) which in turn is used to control instruction fetch for that thread, such that every thread has a piece of hardware to fetch instructions from instruction memory or an instruction cache. It is possible for every thread to have its own instruction memory or instruction cache as well. However, this requires each thread to be built with the resources it needs regardless of whether a thread in a system needs less memory than other threads in the same system. This is undesirable. A preferable strategy is to share the instruction memories and instruction cache between all of the threads and arbitrate between them for the right to access that memory or cache.

Because each thread has a certain amount of hardware specific to its own instruction fetch, it is possible to augment that hardware to make predictions as to the flow of instructions. In the first instance, for example, the prediction hardware could watch for branch instructions in the instruction stream being sent to issue and predict whether that branch would be taken or not. Thereafter, the instruction fetch could change the fetch pattern based upon that prediction (e.g. taking or not taking a branch). At this stage it is possible for the system to be configured such that instruction fetch waits until the outcome of the branch itself (at issue) can be determined and then feeds further instructions or restarts the fetch at the corrected location. Predicted instruction fetch of this nature is shown in Figure 3.

As can be seen from Figure 3, the instruction fetch system for each thread consists of a state machine 30 to generate new requests ('address feed') and a state machine 32 to feed instructions to the thread and its instruction scheduler ('data feed'). State machine 32 includes the functionality of the instruction pre-decode and resource interlocks tests 26 of Figure 2. Also, shown in Figure 3 is a branch filter and prediction unit 34 that picks out branch instructions from the returning fetch data and which can update the address feed as to where to be reading instructions from based upon predictions of where those branches will go.

In non-multithreaded microprocessors, the instruction fetch pipeline could be stalled if the consumption of instructions was less than the maximum rate of request. However, in multithreaded systems, where the instruction cache is shared between the threads, such stalling is not usually possible because it would introduce a dependency between threads such that a thread that has requested a number of instructions but not used them might then stall all the other threads in the system. Therefore, each thread instruction fetch unit incorporates an instruction buffer able to hold the returning data for a fixed number of instruction requests. As many instruction requests may be in transit as there are slots in the instruction buffer may be stored. The instruction buffer is shown as block 36. From the instruction buffer, instructions pass via an instruction data feed unit 32 to the instruction scheduler.

In this system predictions can be made based upon simple rules such as the direction of the branch (forwards or backwards) or using some form of abridged history as to which route was taken for this branch in the past. The actual issue of the branch instruction at the issue register 38 is held up until such time as the outcome is known (i.e. the predicates such as condition testing can be determined). Once the outcome can be determined the branch can be issued and then the flow of instructions can either pick up the predicted fetch data or cause the instruction fetch to restart with the correct instruction address.

Such a system is useful for hiding some of the latency involved in taking branches. However, it does have the significant limitation that there may

always be stalls introduced while waiting to determine the result of the branch condition test.

An improvement upon this system is to issue the predicted instructions before the outcome of the branch, and therefore the correctness of the branch prediction, has been determined. In this case the instructions issued immediately after the branch are speculatively executed (as at the point of issue it is unknown whether the branch prediction was correct or not). As the branch outcome is calculated in the processor pipeline it will eventually become apparent whether the prediction was made correctly or not. When predictions are made correctly the issue of instructions ceases to be speculative and the thread may carry on as normal. When predictions are found to have been incorrectly predicted it is then necessary to delete the instructions issued after the branch and their side-effects and restart the instruction fetch from the newly determined instruction fetch address. This speculative instruction fetch and issue is shown in Figure 4.

Figure 4 shows the same features of a fetch system as Figure 3, but with the difference that speculative instructions are issued. Figure 4 shows a three stage execution pipeline 40 following the instruction scheduler 28, and additionally a branch prediction verification data path 42 back to branch prediction unit 34. As can be seen from Figure 4 the evaluation of branch instructions can take several cycles. To avoid stalling for these cycles the thread needs to be able to continue issuing instructions that followed on from the branch. This may be done provided the actions of those instructions can be unwound. As can be seen from Figure 4, the number of instructions in question is typically fairly short – as shown it would be at most three or four instructions.

In a multi-threaded microprocessor there may well be several threads that have instructions available to be issued on any given cycle. Therefore, it may be better to issue non-speculative instructions from other threads where there is no question of the correctness of the instructions to be issued (in terms of whether they should be issued or not). However, if speculative instructions are never issued there will always be stalls in a thread's instruction stream (exactly as per the arrangement where only the fetch is predicted). The advantage of

speculatively issuing instructions is that if good predictions can be made, more useful work can be done with fewer stalls in the instruction stream for if/then decisions.

As such it is beneficial to be able to speculatively execute instructions where possible, provided that issue does not impact on other threads that have other instructions ready to run. One way to do this is to give threads which are speculatively executing the lowest priority. However, this can be hard to achieve in a system with a feedback loop between current instruction issues and future priority levels. An alternative that can be achieved more easily is to mark speculative execution instructions as unsuitable for being the first or principal or lead issuer. This is achieved by not presenting speculatively executed instructions as available instructions to the main instruction scheduler – instead the threads may mark themselves as interlocked or unavailable at this stage.

In parallel with the choice of the highest priority thread with an instruction that is ready to run, the decision has to be made as to what other threads might be allowed to issue at the same time as this one. This can be calculated for all of the threads on every cycle, as there are fewer gates involved when compared to working out a full set of load-balancing metrics. The gates required test a thread's stated resources against all of the other threads in the system, giving a list of other threads that this thread's instruction is compatible with. This information can then be combined with the pre-prepared thread priority ordering into an absolute list of which threads would issue with this thread if it wins the initial scheduling decision.

Figure 5 illustrates the steps taken in determining which instructions to use, in accordance with an aspect of the invention. At step 500 a priority ranking is determined for each of the threads having available instructions. At this stage threads that are prevented from issuing an instruction owing to resource or register interlocks are removed from consideration. Threads having speculative instructions mark themselves as ineligible for consideration at this stage as well. At step 510 the highest priority thread able to issue an instruction is selected. The instruction issued from this thread is referred to as

the lead instruction. As stated, this lead instruction cannot be a speculative instruction. At step 520, a list of compatible threads is compiled for each thread with an available instruction. From these lists and the priority ranking, a set of instructions for issue with the lead instruction is determined, at step 530. At step 540, the instruction set is issued by the instruction scheduler.

Therefore, the invention provides a simple means by which a thread can take itself out of the normal instruction scheduling rules when speculatively executing instructions, while still maintaining the possibility of improved throughput due to being able to issue instructions where previously the scheduler would have had to stall while it waited for a branch outcome to be determined. This solution uses few additional gates and as a result does not have a negative impact on the clock speed that can be achieved for a given multi-threaded microprocessor design.

Claims

1. A method to dynamically determine which instructions from a plurality of available instructions to issue in each clock cycle in a multithreaded processor capable of issuing a plurality of instructions in each clock cycle, comprising the steps of:

determining a highest priority instruction from the plurality of available instructions;

determining the compatibility of the highest priority instruction with each of the available instructions, wherein instructions are compatible with each other if they do not require the same resources; and

issuing the highest priority instruction together with other instructions compatible both with the highest priority instruction and with each other in the same clock cycle;

wherein the highest priority instruction cannot be a speculative instruction.

2. A method according to claim 1, wherein the step of determining the highest priority instruction comprises determining a highest priority thread that is able to issue an instruction, and selecting the instruction from that thread as the highest priority instruction.

3. A method according to claim 1, further comprising the step of marking speculative instructions, or threads with speculative instructions, as unavailable for the step of determining a highest priority instruction.

4. A method according to claim 1, 2 or 3, further comprising the step of determining a priority ranking for the plurality of available instructions, wherein compatible instructions are issued with the highest priority instruction in order of priority ranking.

5. A method according to claim 4, wherein speculative instructions are given a lower priority ranking than non-speculative instructions.

6. A method according to claim 4 or 5, wherein the step of determining a priority ranking for the plurality of available instructions comprises determining a priority ranking for each thread having an available instruction.

7. A method according to any preceding claim, further comprising the step of determining a list of compatible or incompatible instructions for each of the available instructions.

8. A system for issuing instructions in a multithreaded processor capable of issuing a plurality of instructions in each clock cycle, comprising:

means for determining a highest priority instruction from the plurality of available instructions;

means for determining the compatibility of the highest priority instruction with each of the remaining available instructions, wherein instructions are compatible with each other if they do not require the same resources; and

means for issuing the highest priority instruction together with other instructions compatible both with the highest priority instruction and with each other in the same clock cycle;

wherein the highest priority instruction cannot be a speculative instruction.

9. A system according to claim 8, wherein the means for determining the highest priority instruction comprises means for determining a highest

priority thread that is able to issue an instruction and selecting the instruction from that thread as the highest priority instruction.

10. A system according to claim 8 or 9, further comprising means for marking speculative instructions, or threads with speculative instructions, as unavailable for consideration as the highest priority instruction.

11. A system according to any one of claims 8, 9 or 10, further comprising means for determining a priority ranking for the plurality of available instructions, wherein the means for issuing is configured to issue compatible instructions with the highest priority instruction in order of priority ranking.

12. A system according to claim 11, wherein the means for determining a priority ranking for the plurality of available instructions comprises means for determining a priority ranking for each thread having an available instruction.

13. A system according to any one of claims 8 to 12, further comprising means for determining a list of compatible or incompatible instructions for each of the available instructions.

14. A method to dynamically determine which instructions from a plurality of available instructions to issue in each clock cycle in a multithreaded processor capable of issuing a plurality of instructions in each clock cycle substantially as herein described with reference to the accompanying drawings.

15. A system for issuing instructions in a multithreaded processor capable of issuing a plurality of instructions in each clock cycle substantially as herein described with reference to the accompanying drawings.

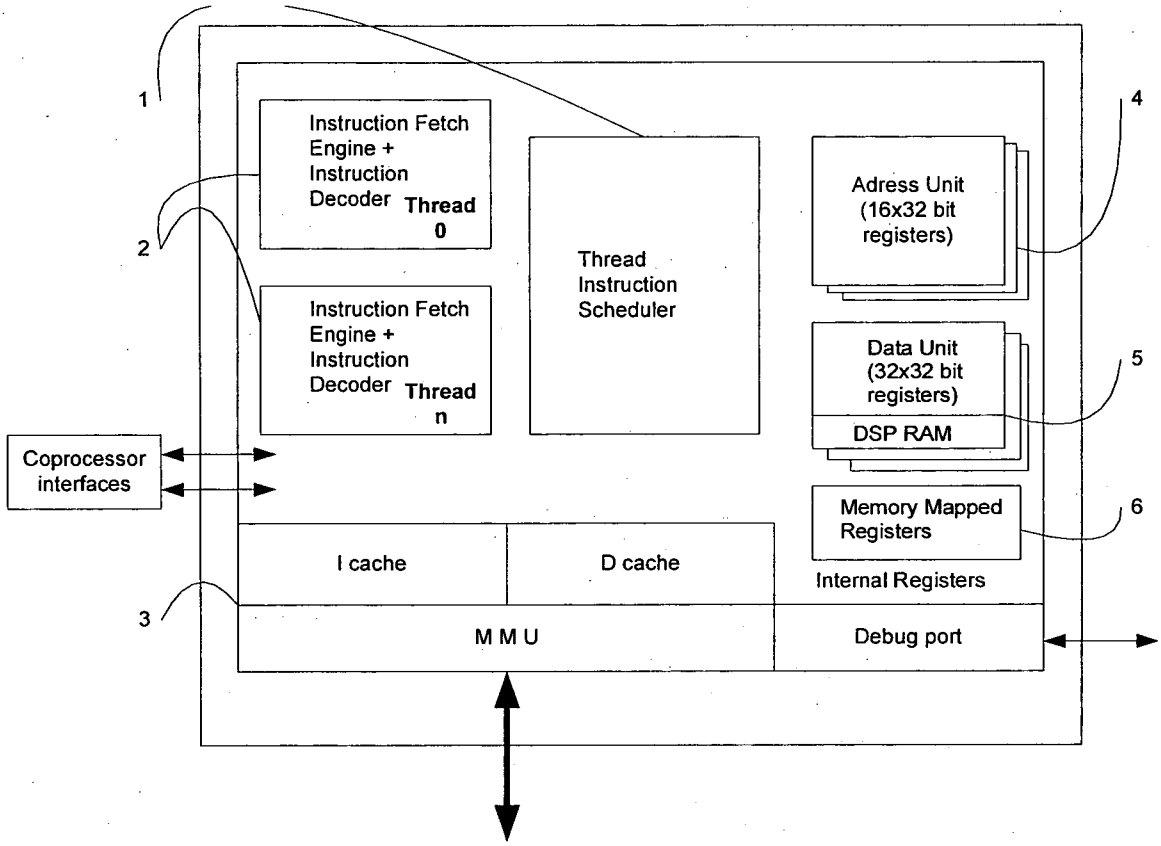
1
2
3
4
5
6
Instruction Fetch Engine + Instruction Decoder Thread 0
Instruction Fetch Engine + Instruction Decoder Thread n
Thread Instruction Scheduler
Adress Unit (16x32 bit registers)
Data Unit (32x32 bit registers)
DSP RAM
Memory Mapped Registers
Internal Registers
Coprocessor interfaces
I cache
D cache
M M U
Debug port

Figure 1

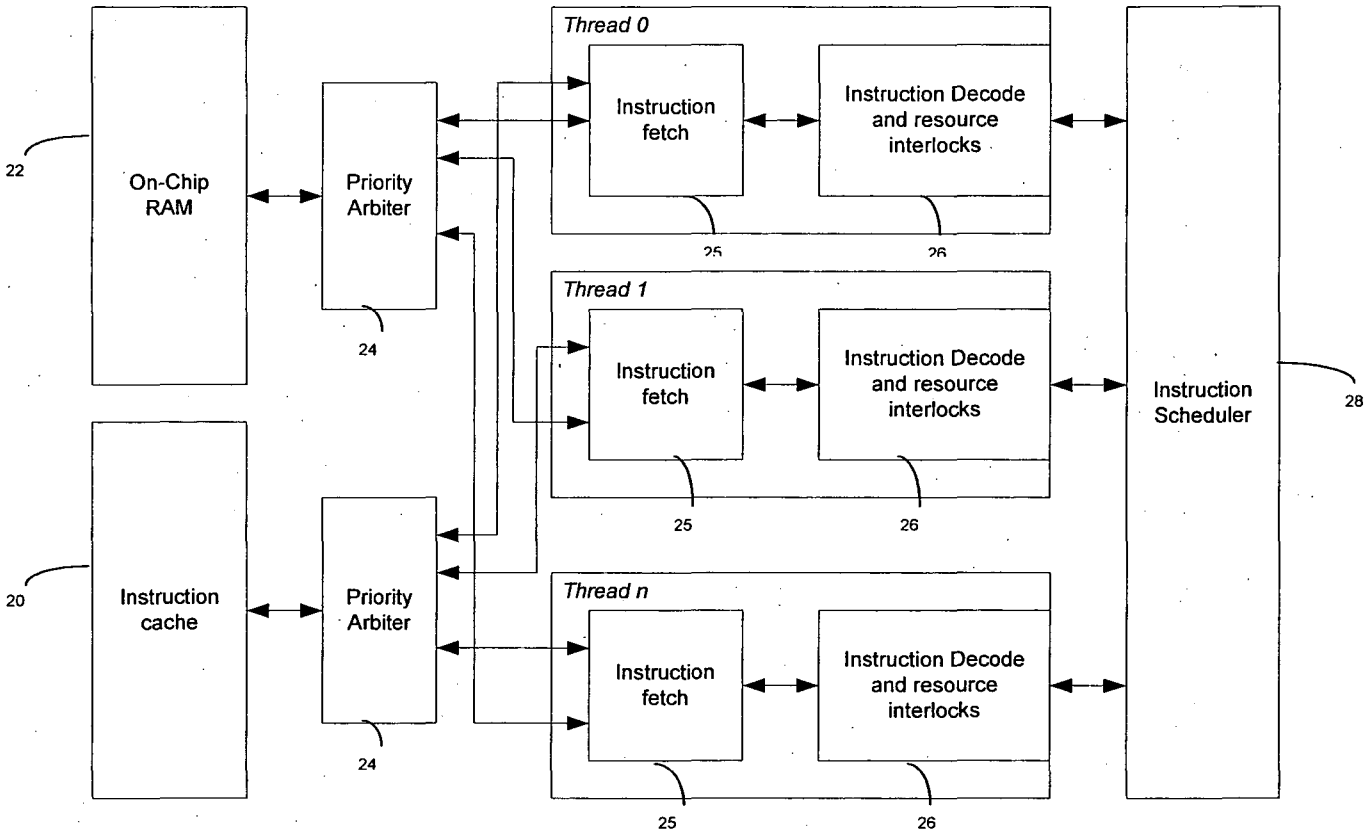
22
On-Chip RAM
Priority Arbiter
24
20
Instruction cache
Priority Arbiter
24
Thread 0
Instruction fetch
Instruction Decode and resource interlocks
25
26
Thread 1
Instruction fetch
Instruction Decode and resource interlocks
25
26
Thread n
Instruction fetch
Instruction Decode and resource interlocks
25
26
Instruction Scheduler
28

Figure 2

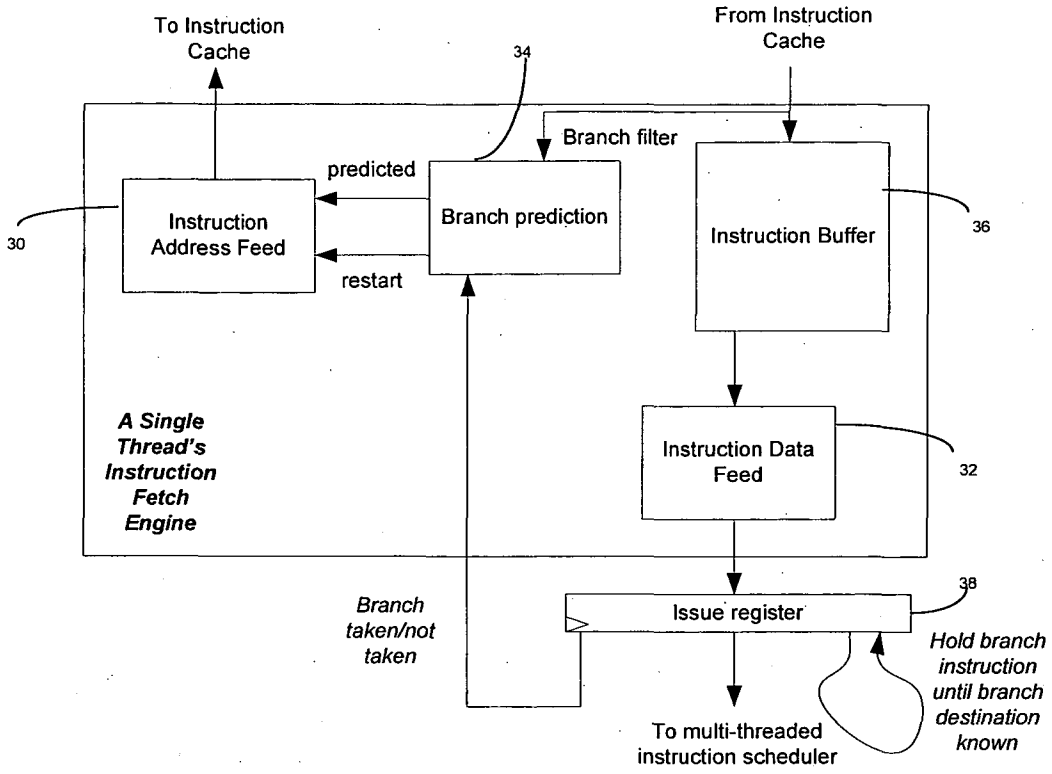
To Instruction Cache
From Instruction Cache
34
Branch filter
predicted
Branch prediction
Instruction Address Feed
30
restart
Instruction Buffer
36
A Single Thread's Instruction Fetch Engine
Instruction Data Feed
32
38
Issue register
Branch taken/not taken
Hold branch instruction until branch destination known
To multi-threaded instruction scheduler

Figure 3

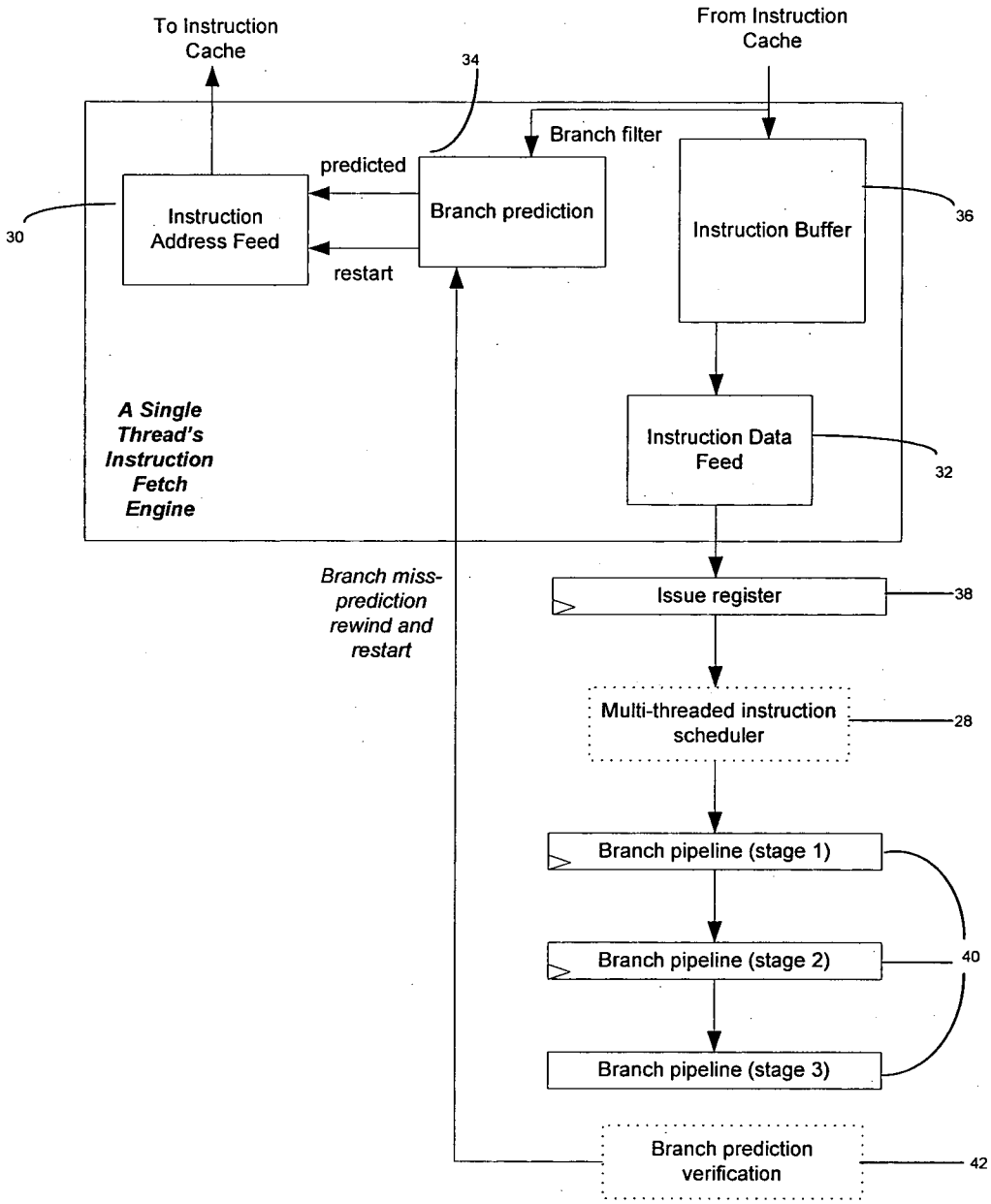
To Instruction Cache
From Instruction Cache
34
Branch filter
predicted
Branch prediction
Instruction Address Feed
30
restart
Instruction Buffer
36
A Single Thread's Instruction Fetch Engine
Instruction Data Feed
32
Branch miss-prediction rewind and restart
Issue register
38
Multi-threaded instruction scheduler
28
Branch pipeline (stage 1)
Branch pipeline (stage 2)
40
Branch pipeline (stage 3)
Branch prediction verification
42

Figure 4

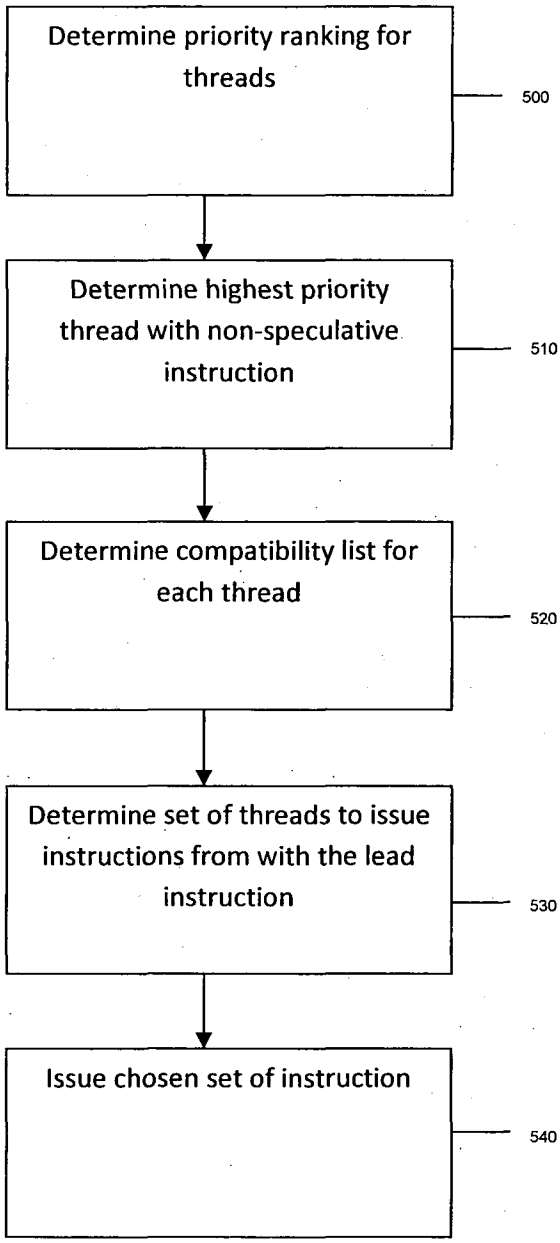
Determine priority ranking for threads
500
Determine highest priority thread with non-speculative instruction
510
Determine compatibility list for each thread
520
Determine set of threads to issue instructions from with the lead instruction
530
Issue chosen set of instruction
540

Figure 5

INTERNATIONAL SEARCH REPORT

International application No
PCT/GB2010/000832

A. CLASSIFICATION OF SUBJECT MATTER
INV. G06F9/38
ADD.

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC, IBM-TDB, WPI Data

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2006/136915 A1 (AINGARAN KATHIRGAMAR [US] ET AL) 22 June 2006 (2006-06-22) paragraphs [0010], [0011], [0022], [0023], [0038], [0043], [0045] - [0048]; figures 6,9	1-13

-/--

[X] Further documents are listed in the continuation of Box C.

[X] See patent family annex.

* Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
13 September 2010	17/09/2010

Name and mailing address of the ISA/	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL - 2280 HV Rijswijk Tel. (+31-70) 340-2040, Fax: (+31-70) 340-3016	Thibaudeau, Jean

INTERNATIONAL SEARCH REPORT

International application No
PCT/GB2010/000832

C(Continuation). DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	IL PARK ET AL: "Implicitly-multithreaded processors" PROCEEDINGS OF THE 30TH. INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE. ISCA 2003. SAN DIEGO, CA, JUNE 9 - 11, 2003; [INTERNATIONAL SYMPOSIUM ON COMPUTER ARCHITECTURE.(ISCA)], LOS ALAMITOS, CA : IEEE COMP. SOC, US, 9 June 2003 (2003-06-09), pages 39-50, XP010796919 ISBN: 978-0-7695-1945-6 Page 2, left column, second paragraph; page 4, left column, first and second full paragraphs; page 9, left column, last three lines to right column, line 15. -----	1-13

INTERNATIONAL SEARCH REPORT

International application No.
PCT/GB2010/000832

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:

2. ☒ Claims Nos.: 14, 15
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:
see FURTHER INFORMATION sheet PCT/ISA/210

3. ☐ Claims Nos.:
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.

2. ☐ As all searchable claims could be searched without effort justifying an additional fees, this Authority did not invite payment of additional fees.

3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:

4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.

☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.

☐ No protest accompanied the payment of additional search fees.

FURTHER INFORMATION CONTINUED FROM PCT/ISA/ 210

Continuation of Box II.2

Claims Nos.: 14, 15

Claims 14 and 15 are so unclear (Article 6 PCT) that no meaningful search is possible for said claims. Indeed the word "substantially" comprised in claims 14 and 15 is vague and a reference to the drawings introduces unclarity (PCT Guidelines, Rule 6.2(a)).

The applicant's attention is drawn to the fact that claims relating to inventions in respect of which no international search report has been established need not be the subject of an international preliminary examination (Rule 66.1(e) PCT). The applicant is advised that the EPO policy when acting as an International Preliminary Examining Authority is normally not to carry out a preliminary examination on matter which has not been searched. This is the case irrespective of whether or not the claims are amended following receipt of the search report or during any Chapter II procedure. If the application proceeds into the regional phase before the EPO, the applicant is reminded that a search may be carried out during examination before the EPO (see EPO Guideline C-VI, 8.2), should the problems which led to the Article 17(2) declaration be overcome.

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No
PCT/GB2010/000832

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2006136915 A1	22-06-2006	GB 2421328 A	21-06-2006