Internal distribution code:

(A) [-] Publication in OJ
(B) [-] To Chairmen and Members
(C) [-] To Chairmen
(D) [X] No distribution

Datasheet for the decision of 22 October 2015

Case Number:	T 1941/11 - 3.5.01
Application Number:	05732166.3
Publication Number:	1741033
IPC:	G06F15/00
Language of the proceedings:	EN

Title of invention:
MULTITHREADED PROCESSOR WITH MULTIPLE CONCURRENT PIPELINES PER THREAD

Applicant:
QUALCOMM Incorporated

Headword:
Multiple concurrent pipelines / QUALCOMM

Relevant legal provisions:
EPC 1973 Art. 56

Keyword:
Inventive step - (no)

Decisions cited:
T 2241/10, T 1963/11

Case Number: T 1941/11 - 3.5.01

D E C I S I O N
of Technical Board of Appeal 3.5.01
of 22 October 2015

Appellant: (Applicant)	QUALCOMM Incorporated 5775 Morehouse Drive San Diego, CA 92121 (US)
Representative:	Beresford, Keith Denis Lewis Beresford Crump LLP 16 High Holborn London WC1V 6BX (GB)
Decision under appeal:	**Decision of the Examining Division of the European Patent Office posted on 14 April 2011 refusing European patent application No. 05732166.3 pursuant to Article 97(2) EPC.**

Summary of Facts and Submissions

I. This is an appeal against the decision of the Examining Division to refuse European Patent application 05732166.3 for lack of clarity. The application is derived from a PCT application published as WO 2005/101221, and references to the contents of the application use this published form.

II. The Examining Division considered that features relating to the threads and pipeline stages and their effect were not clear. The Division also considered, without making it a ground of refusal, that apart from these features the invention differed from D1 essentially only by the use of an "architected accumulated register file", which was an obvious routine measure in this field. During the examination, inter alia, the following documents were cited:

D1: Glossner, Hokenek, Moudgil: *The Sandbridge Sandblaster Communications Processor* in "Software Defined Radio: Baseband Technologies for 3G Handsets and Basestations", John Wiley & Sons, 4 March 2004 ISBN 0-470-86770-1.

D4: Schulte et al.: *A low-power multithreaded processor for baseband communication systems,* Lecture Notes in Computer Science, vol. 3133, November 2004, pages 393 - 402.

D5: Ungerer, Robič, Šilc: *A survey of processors with explicit multithreading*, ACM Computing Surveys, vol 35, No. 1, March 2003, pages 29 - 63.

III. The appellant submitted a new main request and three new auxiliary requests with the statement setting out the grounds of appeal. The Board set out its provisional view in a communication sent with the summons to oral proceedings. With its response, the appellant again submitted a new main request and three new auxiliary requests. The appellant also cited the following document:

D7: Laudon, Gupta, Horowitz: *Interleaving: a multithreading technique targeting multiprocessors and workstations,* International Conference on Architectural Support for Programming Languages and Operating Systems, pages 308 - 318.

IV. At oral proceedings, the appellant's four requests were discussed, after which the appellant formulated a single, new request (new main request) consisting of a new version of claim 1, based on the previous third auxiliary request, and claims 2 - 15 as in that previous request, it being understood that claim 15 would have to be amended along the same lines as claim 1. For the course of the oral proceedings, reference is made to the minutes of 22 October 2015.

V. The appellant's final formulation of its requests were that the appealed decision be set aside and that a patent be granted on the basis of claim 1 according to the (new) main request filed during the oral proceedings and claims 2 to 15 of the third auxiliary request filed with letter dated 24 September 2015.

VI. Claim 1 according the the sole request reads as follows.

1. A multithreaded processor comprising
a plurality of hardware thread units;
an instruction decoder coupled to the thread units for decoding instructions received therefrom; and

a plurality of execution units for executing the decoded instructions, said execution units including a vector multiplication and reduction unit (460) operable to produce a reduced sum;
wherein the multithreaded processor is configured for issuing instructions from a plurality of threads associated with respective ones of the hardware thread units in accordance with a plurality of interleaved multithreaded sequences each of which extends over a plurality of processor clock cycles, in each of which, on a given processor clock cycle only one of the plurality of threads is permitted to issue an instruction and in each of which a different one of the plurality of threads issues an instruction on each of said plurality of clock cycles so that each thread issues an instruction in each said sequence; and said instructions issued in each said sequence are executed concurrently;
characterised in that:
the multithreaded processor includes an accumulator register (406) and is configured so that:
(a) instructions executed by the vector multiplication and reduction unit (460) are pipelined in pipelines which have up to a predetermined number of stages, said predetermined number being greater than the number of threads N in said sequence, and which have: an instruction decode stage, a vector register file read stage, at least two multiply stages, at least two add stages, an accumulator read stage, a plurality of reduction stages, and an accumulator writeback stage; and
(b) the number of concurrently executed instructions in said sequence is such that the reduced sum produced by the vector multiplication and reduction unit (460) is stored in the accumulator register (406) for further processing and before it is needed by the next

vector multiplication and reduction instruction from the same thread,
whereby at least in a given one of the threads, execution of a given instruction may begin before completion of execution of a preceding vector multiplication and reduction instruction issued by said given thread in the preceding sequence, and said accumulator read stage of the given pipeline executes later than the accumulator writeback stage of said preceding pipeline, said given thread thereby supporting multiple concurrent instruction pipelines.

VII. The appellant's arguments were as follows.

D1 was the best starting point for the assessment of inventive step. The processor disclosed in D1 was basically the same as in the present invention. However, the vector processor of the invention had a accumulator, which was not disclosed in D1. Nor did D1 disclose the overlapping of instructions from a single thread in a multi-thread machine.

In a single-thread machine, beginning execution of an instruction only after the previous instruction had completely finished was one possibility (Figure 1A of the application). Another was to use pipelining so that more intensive use of processing units was possible. With pipelining (Figures 1B and 2), the execution of an instruction might start before the execution of a previous one had finished. A problem arose, however, when the second instruction needed data produced by the first. It might happen that the second had to wait, so that some clock cycles of the processor went unused (Figure 2). These were called "bubbles", and it was the aim of the invention to avoid them.

Some techniques were known for avoiding bubbles. One was to provide at least as many threads as pipeline stages, and to interleave instructions from each thread in turn. That guaranteed that the first instruction of a thread would have finished execution before the next instruction of that thread started. It was, therefore, possible to use the pipeline fully, without needing to check for dependencies between instructions or taking special measures such as "bypassing". According to the invention, the number of threads needed to avoid bubbles by interleaving threads could be fewer than the number of stages in the pipeline.

The accumulator in the vector unite made some operations faster by providing memory access within a single cycle, whereas using general vector registers would require two cycles. This, as explained in D4 (a document published later than the earliest priority date of the present application but which nevertheless gave useful information as to the technology), was the purpose in the XFER stages in the instructions I_Mul and V_MUl (Figure 6 of the application). The V_Mul REDUCE instruction did not need an XFER stage in its pipeline, because the writeback stage was to the accumulator rather than to the vector register file. The effect was to make the pipeline shorter and to make it possible to overlap instructions from one thread in a pipeline.

Reasons for the Decision

Procedural issues

1. The appellant filed a new main request as the sole request during oral proceedings, after lengthy discussion of the previous main request and a short discussion as to whether auxiliary requests I to III should be admitted. Although late filed, the Board admitted that request into the proceedings so that a decision could be taken on the substance of the claimed invention.

The Invention

2. The application is one of several that concern the "Sandbridge Sandblaster" processor. Two others, at least, have been the subject of appeal proceedings (T 2241/10 *Pipelined multioperand adder/QUALCOMM* and T 1963/11 *Multithreaded processor/QUALCOMM*). D1 and D4 also address aspects of this processor. The processor in question is designed for use in mobile devices, especially in view of specific requirements for digital signal processing (see, for example, D1, section 6.1). The application does not identify this context, let alone explain it. As a consequence, it has been difficult for the appellant to formulate requests that reflect the true technical background.

3. The processor, shown in Figure 4 of the application, has three processing sections. One deals with branch instructions, one with integer and load/store operations, and one with vector operations. The appellant has now restricted claim 1 to a particular situation which arises in the vector processor. It

concerns the interactions between a vector processing pipeline and a plurality of threads, when a particular instruction is issued twice in succession to the vector processor (this is described in more detail, below).

4. The application sets out a problem with pipelined processors generally: when two instructions are simultaneously in the pipeline and the second requires some data generated by the first, the second will have to wait (otherwise, the second instruction will read the wrong data). This is shown in Figure 2 of the application. The pipeline is idle for one or more clock cycles, which means the processor cannot be fully utilised. The application calls such idle cycles "bubbles" and seeks to avoid them.

5. It was known completely to eliminate bubbles by processing a number of different threads. Instructions from different threads are assumed to be independent, so by interposing enough instructions from other threads, it can be assumed that a first instruction has completely finished its execution before a second instruction from the same thread enters the pipeline. If the pipeline has N stages, and there are at least N threads, then each instruction of each thread will have finished execution, before the next instruction of the same thread. This, however, assumes that each thread takes a turn at most once every N threads. As explained in D5 (page 35, right-hand column), but, unfortunately, not in the application, this has a drawback: each thread gets one N-th of the processor time, and there is no flexibility for threads that need to issue instructions more frequently or which can get by with less. D5 mentions two ways of ameliorating this. Firstly, the compiler may indicate dependencies between instructions, so that the instructions from one thread

can safely be fed in to a pipeline more frequently (a sort of dependency checking). Secondly, the caching technique set out in D7 may be used. The application seeks another way of allowing instructions from a single thread safely to overlap in the pipeline.

6. As the application describes it, the invention consists in allowing one thread out of several to have two or more instructions in the pipeline at one one time, and this is achieved by having a number of threads which is smaller than the number of the pipeline stages.

7. In its full generality, the invention does not use round-robin scheduling of threads but "token triggered threading", which (again unfortunately) is not described in the application (see page 9, lines 12 - 18 of the published application), but in another application (US6842848). Although this technique is described as one possible embodiment, it is in fact the most general form of thread scheduling envisaged in terms of the order in which different threads issue instructions to the pipeline. In particular, it can implement round-robin scheduling. A central difficulty with the application has been the lack of basis for a formulation of thread scheduling in relation to pipeline structure.

8. Claim 1 according to the sole request is restricted to the situation in which one thread issues consecutive V_Mul_REDUCE instructions. There are various versions of this instruction, as the application attempts to explain (page 11, lines 14 - 23), but they all use the pipeline structure shown in the bottom row of Figure 6. Although claim 1 does not define this pipeline structure, it is the only example given and the claim must be read so as to include at least this structure.

The pipeline consists of thirteen stages. When implemented in the processor of D1 (the "Sandbridge Sandblaster", also set out in US2003/0120901, to which the application refers at lines 19 - 23 on page 6) so that there are eight threads, each of which issues an instruction every eight clock cycles, the second V_Mul_REDUCE instruction enters the pipeline before the first has finished. Nevertheless, there are no bubbles. This is because the accumulator read stage (the sixth stage of the pipeline) necessarily occurs after the previous instruction has written to the accumulator. That, in turn, is because there are sufficient intervening threads.

Inventive step

9. The Board agrees with the Examining Division and with the appellant, that D1 is a reasonable starting point for the skilled person as it relates to a very similar multithreaded processor having a vector processing unit (Figure 6.1 cf. Figure 4 of the application). According to the appellant, the vector processor of D1 does not have an accumulator, and does not have a sequence such that one thread can have two instructions in the pipeline at one time. In particular, it does not have that in the context of V_Mul_REDUCE instructions.

10. It is not straightforward to understand the clause, in claim 1, that specifies the sequence of threads, or to see its basis in the application as filed. The appellant explained that, when there are eight threads, as in the example set out in the description of Figure 7 (page 12, from line 9 of the published application), each of the "plurality of interleaved multithreaded sequences" extended over eight clock cycles, and in

each sequence, each of the threads issued one instruction. The order of the instructions from one of sequences to the next was the same. Thus, it could well be the repeating sequence shown in Figure 6.3 of D1. The important point is that this sequence, being constant, guarantees that two consecutive V_Mul_REDUCE instructions from one thread enter the pipeline exactly eight clock cycles apart, as shown in Figure 7. This guarantees the the stage in which the accumulator is read by the second instruction (Figure 7, second line, stage 13) necessarily comes after the stage in which data are written to the accumulator in the first (first line, stage 10). In fact, reading occurs three stages after writing.

11. Computer processors are complicated things, as the appellant argued during oral proceedings. The skilled person possesses a wealth of background knowledge. In the present case, the skilled person knew about pipelining and about vector processors and about the DSP operations needed in mobile devices.

12. In particular, the skilled person knew about pipelining in single-thread machines, as shown in Figures 1A, 1B, and 2 of the application. She knew that if a second instruction read data before the first instruction had written it, then there would be errors. That is why she would delay execution of some stages in the second pipeline (Figure 2). It is the reason there are bubbles at all: they prevent the second instruction reading the wrong value.

13. As mentioned above, the skilled person also knew that one way of ensuring data was not read before it was written was to interpose instructions from other threads, but she also knew the costs of providing a

complete set of registers for each thread.

14. Starting from D1, the skilled person had a machine with hardware support for eight threads. She has to accommodate a V_Mul_REDUCE instruction (D1, section 6.2.4) because it is needed when computing, for example, "a sum of squares function common to many signal processing kernels." D1 describes something of the V_Mul_REDUCE instruction (D1, penultimate paragraph of section 6.2.4), but does not give detailed pipeline stages. Nevertheless, it is clear that this instruction will take more than eight stages.

15. According to Figures 6 and 7 of the application, the V_Mul_REDUCE instruction takes 11 stages. It is useful to compare it with the V_Mul instruction also shown in Figure 6. The final two stages of V_Mul are "XFER" and "WB". The description has this to say (page 11, lines 12 - 13): "In stage 6 (Xfer), the results are transferred back to the vector register file, and in stage 7 (WB), the results are written back." The skilled reader could be forgiven for wondering why the results are written to the register file twice. The explanation can be found in D4, on which the appellant drew for explaining the invention, but which was published after the earliest priority date. D4 sets out a pipeline for a "vector multiply and accumulate" instruction, which includes stages labelled "Xfer" and "WB" (D4, Figure 6) and says: "The Transfer stage is needed due to the long wiring delay between the bottom of the VPU [the vector processing unit] and the VRF [the vector register file]." Thus, the XFER stage is there to account for the amount of time it takes to get data from the processor to the register file. The XFER and WB stages are simply this: during XFER, the data can be seen as en route, during WB, they arrive in the

register. In contrast, the V_Mul_REDUCE instruction shown in Figures 6 and 7 of the application does not have an XFER stage. The appellant explained that this was because the accumulator did not have the "long wiring delay", so only a single cycle was needed. The accumulator, therefore, has the effect of shortening the V_Mul_REDUCE pipeline by one cycle. Without it, there would have been twelve stages.

16. Even using the accumulator, the V_Mul_REDUCE instruction takes three stages more than the number of threads (eleven stages, eight threads). What is the skilled person to do to avoid bubbles? She might redesign the processor to provide hardware support for more threads. She might accept that there will be bubbles and provide the dependency checking that is needed to insert idle cycles into the pipeline when they are needed or insert idle cycles just in case they are needed. The latter is the solution adopted in the application for ALU and I_Mul instructions shown in Figure 6: each of the pipelines includes a WAIT cycle, and a WAIT cycle is a bubble.

17. For consecutive V_Mul_REDUCE instructions, however, the skilled person need actually do nothing. With eleven cycles in the pipeline and eight threads, and with the accumulator being read only in stage 5, bubbles just do not arise. Indeed, as shown in Figure 7, the second instruction could even start two cycles earlier, and still there would be no bubbles. (At least, there would be no bubbles caused by reading the accumulator; the reading of the register file is another matter, but the invention does not deal with that.)

18. The claimed invention does not solve the problem of removing (or avoiding) bubbles. There are no bubbles to

remove.

19. While it is true that the accumulator shortens the pipeline, the Board finds it difficult to accept this as an alternative objective technical problem. Firstly, the claim does not specify any particular pipeline structure or relationship between pipeline structure and the use of the accumulator. Secondly, the application as filed does not explain the accumulator's role in shortening the pipeline even when restricted to the particular form of V_Mul_REDUCE shown in Figures 6 and 7 (but not claimed). That role is only apparent from D4, a document of which the reader of the application as filed would have been unaware both because the application does not refer to it and because it was not published until later. However, even if the Board did acknowledge this technical problem, it would not help the appellant. The provision of a local memory for faster access was standard practise.

20. Since the invention does not address any problem of bubbles, the Board considers that the subject matter of Claim 1 does no more than provide an accumulator in the context of a processor which often needs to accumulate. The provision of an accumulator is not a necessary consequence; the register files could be used, for example. But the provision of a specialised accumulator would have been an obvious option as stated by the examining division.

21. The Board concludes that the subject matter of claim 1 does not involve an inventive step (Article 56 EPC 1973).

Conclusion

22. The appellant's request does not comply with the requirements of the EPC and cannot be allowed.

Order

For these reasons it is decided that:

The appeal is dismissed.

The Registrar: The Chairman:

Beschwerdekammern des Europäischen Patentamts ✦ Boards of Appeal of the European Patent Office ✦ Chambres de recours de l'Office européen des brevets ✦

T. Buschek W. Chandler

Decision electronically authenticated