Internal distribution code:

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

Datasheet for the decision of 6 May 2019

Case Number: T 0968/13 - 3.5.06

Application Number: 09701244.7

Publication Number: 2243076

IPC: G06F9/315, G06F9/305, G06F9/318

Language of the proceedings: EN

Title of invention:
ROTATE THEN OPERATE ON SELECTED BITS FACILITY AND INSTRUCTIONS THEREFORE

Applicant:
International Business Machines Corporation

Headword:
ROTATE THEN OPERATE ON SELECTED BITS/IBM

Relevant legal provisions:
EPC Art. 56

Keyword:
Inventive step - (no)

Decisions cited:

Catchword:

Europäisches Patentamt
European Patent Office
Office européen des brevets

Beschwerdekammern

Boards of Appeal

Chambres de recours

Boards of Appeal of the
European Patent Office
Richard-Reitzner-Allee 8
85540 Haar
GERMANY
Tel. +49 (0)89 2399-0
Fax +49 (0)89 2399-4465

Case Number: T 0968/13 - 3.5.06

D E C I S I O N
of Technical Board of Appeal 3.5.06
of 6 May 2019

Appellant: (Applicant)	International Business Machines Corporation New Orchard Road Armonk, NY 10504 (US)
Representative:	Litherland, David Peter IBM United Kingdom Limited Intellectual Property Department Hursley Park Winchester, Hampshire SO21 2JN (GB)
Decision under appeal:	**Decision of the Examining Division of the European Patent Office posted on 20 November 2012 refusing European patent application No. 09701244.7 pursuant to Article 97(2) EPC.**

Composition of the Board:

Chairman	M. Müller
Members:	A. Teale
	A. Jimenez

Summary of Facts and Submissions

I. This is an appeal against the decision, dispatched with reasons on 20 November 2012, to refuse European patent application No. 09 701 244.7 on the basis that the subject-matter of independent claims 1 and 8 did not involve an inventive step, Article 56 EPC, in view of the following documents:

D1: US 2003/0037085 A1 and
D2: GB 2 317 469 A.

II. A notice of appeal and the appeal fee were received on 16 January 2013, the appellant requesting that the decision be set aside and a patent granted.

III. With a statement of grounds of appeal, received on 20 March 2013, the appellant submitted a new set of claims and requested that the decision be set aside and a patent granted, based on said claims and the description as amended with the submission dated 29 August 2012.

IV. In an annex to a summons to oral proceedings, the board expressed doubts as to whether the subject-matter of claims 1 and 7 involved an inventive step, Article 56 EPC, starting from D1 and applying the teaching of D2.

V. In a letter received on 25 April 2019 the appellant stated that it would not attend the oral proceedings and withdrew its request for oral proceedings (the appellant had not, in fact, made such a request). No arguments as to substance or amendments were submitted. The oral proceedings were then cancelled.

VI. The application is thus being considered in the following form:

Description:
pages 1 to 3, 6 to 11, 13 to 39, 41 and 43, as published as WO 2009/087162 A2,
pages 4, 5, 12 and 40, received on 7 September 2011, and
page 42, received on 29 August 2012.

Claims: 1 to 7, received with the grounds of appeal on 20 March 2013.

Drawings:
pages 1/17 to 16/17, as published, and
page 17/17, as received on 7 September 2011.

VII. Claim 1 reads as follows:

"1. A method of operating a computer comprising: fetching a rotate-then-operate instruction in a program, the rotate-then-operate instruction defined for a computer architecture, the rotate-then-operate instruction comprising an opcode field, a first register field (R2), a second register field (R1), a T bit, and a rotate amount field (I5) specifying a rotate amount, wherein the first register field specifies one of a plurality of general registers, wherein the second register field specifies one of the plurality of general registers; executing the rotate-then-operate instruction comprising: obtaining a first operand from a first register specified by the first register field; rotating the first operand by the instruction specified rotate amount to produce a rotated value wherein the rotation effectively shifts bits towards a higher order position and effectively

shifts bits out of the high order bit position into the low order bit position; selecting a portion of the rotated value; obtaining a second operand from a second register specified by the second register field; performing a Boolean operation on the selected portion and corresponding bits of the second operand producing a result corresponding to the selected portion, the Boolean operation specified by the rotate-then-operate instruction; responsive to the T bit being 1, the execution does not change the second operand of the second register; responsive to the T bit being 0, saving the result in a second operand portion of the second operand in the second register, the second operand portion corresponding to bit positions of the selected portion, wherein all other bits of the second register other than the second operand portion are unchanged in the second register by the saving operation and continuing to a next instruction for execution; and setting a condition code, the condition code indicating the result is any one of zero or not zero."

The claims also comprise a claim 7 to a computer system referring to "the method of any preceding claim".

Reasons for the Decision

1. The board has no occasion to deviate from its preliminary opinion, since the appellant did not file arguments or amendments in response to the summons.

2. The admissibility of the appeal

In view of the facts set out at points I to III above, the appeal fulfills the admissibility requirements under the EPC and is consequently admissible.

3. Summary of the invention

3.1 The invention, illustrated in figure 8, relates to carrying out a "rotate-then-operate" instruction in a computer, the instruction having two operands. In the context of the application, "rotation" refers to changing the order of the bits of the first operand, for example the bits ABCD, into a rotated form, such as BCDA, CDAB or DABC. A portion of the rotated value is then selected (termed the "selected portion" in claim 1) and used as an operand in the Boolean operation.

3.2 The claims are directed to the "rotate-then-operate" embodiments illustrated in figures 6 and 7; see page 40, line 21, to the end of page 42. Figure 7 shows that the T-bit is stored in bit 0 of the I3 field; see also page 41, lines 11 to 14.

3.3 The instruction set out in claim 1 comprises an opcode field (specifying the Boolean operation on said selected portion and the second operand, such as AND, OR or XOR), first and second register fields (R1, R2, identifying the register holding the **second** and **first** operands, respectively), a T bit (determining whether the result of the Boolean operation overwrites the second operand in R1, but leaves all other bits in R1 unchanged) and a rotate amount field (I5, determining the number of bits by which the bits of the first operand are to be rotated). T=0 results in such overwriting, whilst T=1 results in no change.

3.4 In claim 1 a "condition code" is set. The board understands the condition code to be derived from the result of the Boolean operation. According to page 41,

lines 25 to 27, "As a result of executing the Rotate Then instruction, condition codes are set as follows: 0 Selected bits zero 1 Selected bits not zero".

4. The prior art on file

4.1 Document D1

4.1.1 The decision assesses inventive step starting from D1; see point 15.

4.1.2 As shown in figure 1, D1 relates to a field processing unit (160) for carrying out arithmetic and/or logical operations, the unit being connected to a register file (150) comprising a plurality of registers and a condition code register (170); see [21]. The condition codes or bits can, for example, be "carry, zero, negative and overflow bits"; see [21-22] and [39]. They are derived from the result of the operation performed by the field processing unit.

4.1.3 The field processing unit comprises an execution unit (420) for performing operations on operand words (specified by their register location) and/or "immediate data" words (contained in the instruction itself), both comprising a plurality of fields; see figure 4 and [2-3, 36-45]. The execution unit has inputs for *inter alia* operands A and B, a first barrel shifter (440) for manipulating operand A and a "field arithmetic logic unit and condition logic" unit (450) producing two outputs. The first output can be manipulated by a second, optional barrel shifter (460) and written to a register in the register file; see [41]. The second output is written to the condition code register. A field specifier selector (470) indicates a "begin" location to *inter alia* the first

barrel shifter and an "end" location to a mask generator (410) which is connected to the "field arithmetic logic unit and condition logic" unit (450). The first barrel shifter shifts the operand by a number of bits defined by the begin bit position; see [38], penultimate sentence.

4.1.4 The format of the instructions processed by the execution unit can vary; see [19]. Figure 2 illustrates an example of an instruction format, the overall structure comprising an "opcode" (operation code)(210), an "operand specifier" (250) and a "field specifier" (270). Opcodes can specify arithmetic operations (such as addition and subtraction) and logical operations (such as AND, XOR, shift left and rotate left). The operand specifier may specify two source operands (235, 240), of which the second operand is also a destination register (230), i.e. the result overwrites the second operand; see figure 2; [25], last two sentences. The "field specifier" indicates the field of the operands, otherwise referred to as the "bit boundaries", that the processing unit operates upon, the operation not affecting bits outside the boundaries; see [26]. Thus the field specifier may indicate the begin (260) and end (265) bit positions of the field.

4.1.5 Figure 3A shows an example of a field operation on operands A and B, the result overwriting operand B; see [29-31 and 43]. The operation is only to be carried out on a selected portion of operands A and B, termed portions X (315) and Y (325) respectively. Portion X of operand A is at the right-hand end. In other words, field X in operand A is "right justified". In order that corresponding bits of portions X and Y align, the whole of operand A is barrel-shifted to the left, thus

moving portion X to align with portion Y. The result Z (335) overwrites portion Y of operand B. According to paragraph [40], if the second barrel shifter (445) is used, it "shifts the result back to the original bit position when the operand B is right field justified ..."; see [40].

4.1.6 Instead of shifting operand A to the left to align with operand B, D1 also mentions using a second barrel shifter (445) to shift portion Y of operand B to the right to align with portion X of operand A; see [31, 44]. In other words, portion Y in operand B is right justified, like portion X in operand A. Hence D1 discloses barrel shifting both to the right and to the left.

4.1.7 The board notes that the bit of the condition code in D1 indicating that the result is zero, for instance when the bit is "1", also implicitly indicates that the result is non-zero, in this example when the bit is "0".

4.1.8 Regarding operand A as the first operand in claim 1 and operand B as the second operand in claim 1, and the "BEGIN" value for operand B contained in the begin bit position (260) of the instruction word as a rotate amount field, D1 discloses the following features of claim 1:

A method of operating a computer comprising: fetching a rotate-then-operate instruction (see figure 2) in a program (see figure 1; instruction fetch unit 130), the rotate-then-operate instruction defined for a computer architecture, the rotate-then-operate instruction comprising an opcode field (210), a first register field (235) and a second register field (240), and a

rotate amount field specifying a rotate amount, wherein the first register field specifies one of a plurality of general registers and the second register field specifies one of the plurality of general registers; executing the rotate-then-operate instruction (see figure 3A) comprising: obtaining a first operand (A; 310) from a first register specified by the first register field; rotating the first operand (A) to produce a rotated value (330), wherein the rotation effectively shifts bits towards a higher order position and effectively shifts bits out of the higher order bit position into the low order bit position; selecting a portion (X) of the rotated value, obtaining a second operand (B) from a second register specified by the second register field; performing a Boolean operation (see "logical operations (e.g., AND, OR, XOR ..." in [24]) on the selected portion and corresponding bits of the second operand (Y) producing a result (Z, 335) corresponding to the selected portion, the Boolean operation specified by the rotate-then-operate instruction (see figure 2; opcode 210); saving the result in a portion (Z) of the second operand (B) in the second register, the second operand portion corresponding to bit positions of the selected portion (see figure 3A), wherein all other bits of the second register other than the second operand portion are unchanged in the second register by the saving operation (see [26], third sentence) and continuing to a next instruction for execution; and setting a condition code (see [21]), the condition code indicating that the result is any one of zero or not zero.

4.2 Document D2

4.2.1 According to the International Preliminary Examination Report, using a flag in an instruction to inhibit writing a result back to an operand register, while at the same time setting condition code bits, was known from D2. The board agrees.

4.2.2 According to page 1, lines 10 to 16, in the context of Reduced Instruction Set Computing (RISC), it is desirable "that those few instructions that are provided have a great degree of flexibility and utility. One instruction that is sometimes useful is to proform [sic] a calculation be [sic] not to store the result to a working register". The board understands the latter sentence to mean performing a calculation but not storing the result in a working register. D2 solves this problem by using a program instruction word comprising a destination register write disable flag for selectively disabling writing of a result data word to the destination register; see sentence bridging pages 1 and 2. By adding this flag, more data processing functions can be offered without a significant increase in the hardware overhead. The power consumption caused by unwanted register writes is also avoided; see page 2, lines 2 to 6.

4.2.3 According to page 3, lines 4 to 8, when the destination register write disable flag is set to disable writing of a result data word to the destination register, the result data word is still used to update a condition code flag comprising a zero result flag.

4.2.4 Figures 1 and 3 illustrate the use of such a program instruction word in a "Piccolo" coprocessor (4) of a DSP (Digital Signal Processor). Page 63, lines 20 to

25, discusses the CMP and CMN opcodes which carry out a subtraction and an addition operation, respectively, in which condition flags are set but register writing is disabled. According to claim 8,

"when said destination register write disable flag does not disable said writing of said result data word to said destination register, a subtract operation in which a first operand is subtracted from a second operand to yield a subtraction result data word, said subtraction result data word is written to said destination register and said at least one condition code flag, including said zero result flag, is updated;

and when said destination register write disable flag does disable said writing of said result data word to said destination register, a compare operation in which a first operand is subtracted from a second operand to yield a subtraction result data word, said subtraction result data word is not written to said destination register and said at least one condition code flag, including said zero result flag, is updated."

5. Inventive step, Article 56 EPC

5.1 The appealed decision

5.1.1 Claims 1 and 8 treated in the decision are the same as present claims 1 and 7, respectively.

5.1.2 According to the reasons for the decision (see point 16), the subject-matter of claim 1 differed from the disclosure of D1 in that the "rotate-then-operate" instruction comprised:

i. a field comprising an explicit value of the rotation amount (In contrast, D1 implied that this was calculated from the "field specifier" (270), containing the begin bit position (260) and the end bit position (265), in the instruction).

ii. a bit whose value controlled whether the result of the operation was committed or not.

These differences were not functionally related, being neither complementary nor cooperative. Regarding difference feature "i", providing a rotate amount directly in the instruction word as an immediate, or computing it relative to the position of a field of interest in respect to a standard alignment were well known alternatives not yielding an unexpected technical effect. The skilled person would have chosen one of these alternatives according to considerations such as increasing the instruction length in order to accommodate the extra field in the instruction, against the necessity to execute more instructions in order to align operands with the reference point. Such choices would have been made by the person skilled in computer architectures without resorting to inventive skill.

5.1.3 Regarding difference feature "ii", the apparatus of D1 suffered from the obvious disadvantage of decreased instruction throughput and wastage of instruction memory, or register pressure, since, whenever an operation was performed solely to produce condition codes, extra instructions were required to reinstate the former content of the result register, or the compiler had to allocate or dedicate registers to accommodate the otherwise unused results of the operation. D1 taught branching according to condition

bits generated by the field processing unit. The skilled person would have attempted to solve the problem of producing condition codes without losing the content of the result register and thus have found D2. D2 disclosed an arithmetic and logic unit controlled by instructions comprising a control bit which, when set, inhibited writing of the result to the destination register, while at the same time allowing the updating of condition codes including the zero flag; see pages 1 to 3. It would have been obvious to apply the teaching of D2 to the apparatus of D1 to solve the problem at hand, this requiring no modification of the field processing unit, which comprised a context multiplexer coupled to its output which selected whether the result of the operation or an unchanged operand was to be forwarded to the destination register; see paragraph [0041].

5.1.4 Hence the subject-matter of claim 1 and, for the same reasons, that of claim 8 (the same as present claim 7), lacked inventive step.

5.2 The grounds of appeal

5.2.1 According to the appellant, D1 (see [24]) discloses rotate left/right opcodes (210) which, in view of the field specifier 270 in [26] and the barrel-shifting of a field of operand A to align it with a field of operand B before carrying out an operation, is a "barrel-shift then rotate" instruction (see [38]), in contrast to the claimed rotation followed by a Boolean operation, rotation not being a Boolean operation.

5.2.2 In contrast to the "rotate amount field" set out in claim 1, in D1 the "begin bit position" (see [26], line 9) not only specified the point after which the bits in

operand A were to be rotated to align with operand B (see figure 3A) but also implicitly specified the number of bits through which that part of operand A was to be rotated. This meant that operand A had to have the part to be operated on at the right-hand end, i.e. portion X in operand A had to be "right justified".

5.2.3 D1 did not disclose a "T" bit, set out in claim 1. In D1 the condition code depended on the entire result word (see figure 3A, portions 3, Z, 4), whereas claim 1 specified that the condition code depended on whether the result of the Boolean operation on the selected portion of the rotated first operand and corresponding bits of the second operand portion operated was zero or non-zero (see figure 8; 806).

5.2.4 According to the appellant, the prior art suffered from three problems. Firstly the portion of the first operand to be operated on had to be "right justified", i.e. at the end. Secondly the solution known from D1 did not allow the result to be discarded once the condition code had been derived from it. Thirdly the condition code set out in claim 1 only depended on the bits operated on, whilst in D1 it depended on the entire result. The invention solved all three problems by using the rotate amount field "to allow the portion of the first operand to be operated on to be in any position within the first operand". Hence the same first operand could be compared to a succession of second operands, for instance to find matching access-control bits of a memory page's storage protection key (PSW)(Program Status Word). In the embodiment of the invention on page 42, lines 8 to 19, in which in the course of a RXSBG (rotate then exclusive OR selected bits) instruction the second operand in register R4 is rotated by sixteen bits (specified in field I5), bits

56 to 63 are rotated (moved) to bits 40 to 47. Bits 40 to 43 (specified in fields I3 and I4) are then compared in a Boolean XOR operation with the corresponding bits 40 to 43 of a first operand in register R14 and the condition code set (0=bits zero, 1=bits non-zero). In the example the T-bit equals 1 (see bit 0 of I3 field), so the result is not written to the second operand. Using the approach known from D1, bits 56 to 63 of the second operand would have to be right justified, i.e. they would have to appear at the end of the second operand, and the second operand would have to be reloaded after each comparison.

5.2.5 According to the appellant, D1 did not disclose the following features set out in the claims:

i. a "rotate then operate" instruction (D1 disclosed a "barrel-shift-then-operate" instruction);

ii. a rotate field specifying a rotate amount and

iii. a mechanism such as a T bit for determining whether the result operand is to be updated.

5.2.6 The appellant has stated that feature "iii" is known from D2, which discloses a destination register write disable flag and the updating of a condition code flag, even when said writing is disabled.

5.2.7 The appellant has also argued that neither D1 nor D2 discloses a rotation specified in a rotate amount field or rotation, as opposed to barrel-shifting, before an operation.

6. The board's finding on inventive step

6.1 The interpretation of the term "rotation"

6.1.1 Claim 1 sets out the rotation effectively shifting "bits out of the high order bit position into the low order bit position", also termed "wrapping", and thus sets out rotation without data loss, as opposed to merely shifting bits towards the higher order end and filling bits at the lower order end with, for instance, zeros, known as "padding", thereby losing data. The appellant has drawn a distinction between the claimed "rotating" and the "barrel-shifting" known from D1; see grounds of appeal, points 2.19 to 2.21. The board does not agree with the appellant on this point. D1 discloses a first and a second, optional barrel shifter shown in figure 4 (see 440, 445 and [38]) for shifting bits in operand A or the result either towards or away from the high order bit position. D1 does not explicitly explain what happens when bits "drop off the end" of a data word during said shifting. However D1 does mention that, if the optional second barrel shifter 455 is used, it "shifts the result back to the original bit position when the operand B is right field justified ..."; see [40]. In the board's view this is only possible if neither the first (440) nor the second (455) barrel shifter incurs a data loss (i.e. "pads"). Hence, in this case in D1, both barrel shifters must shift "bits out of the high order bit position into the low order bit position" and thus "wrap".

6.1.2 The board does not accept the appellant's argument (grounds, point 2.9) that in D1 the condition code is set depending on the entire result word (portions 3, Z and 4 in figure 3A). In the board's view, in both the application and D1 the condition code is derived from

only the result portion of the second operand; see D1, figure 3A; portion Z. According to paragraph [39] of D1, "The field ALU 450 has a condition logic to generate the condition codes or condition bits such as carry, zero, negative, and overflow bits **according to the result of the operation.**" (Emphasis added by the board). In this context, the "result" in D1 is portion Z of operand B; see figure 3A.

6.2 The differences between claim 1 and D1

6.2.1 The subject-matter of claim 1 differs from the disclosure of D1 only in that, responsive to a T bit of the instruction being 1, the execution does not change the second operand of the second register and, responsive to said T bit being 0, carrying out said saving step.

6.2.2 Consequently the board does not agree with the appellant that differences "i" and "ii" in the list of difference features, set out in point 2.17 of the grounds of appeal, exist. It is however common ground that difference "iii" in that list exists.

6.2.3 The board agrees with the finding in the appealed decision that the only difference feature over D1 is obvious. The skilled person would have recognised that the method and apparatus known from D1 suffered from the obvious disadvantage of decreased instruction throughput and wastage of instruction memory, or register pressure, caused by unnecessarily writing the result of the Boolean operation to a register when all that was needed was to derive condition codes from the operation result, as set out in point 18 of the decision. The skilled person would have sought to solve this problem and thus have found D2. It is common

ground between the appellant and the decision, and the board agrees, that D2 discloses an arithmetic and logic unit controlled by instructions comprising a control bit which, when set, inhibits writing of the result to the destination register, while at the same time allowing the updating of condition codes including the zero flag; see pages 1 to 3. It would consequently have been obvious for the skilled person to apply the teaching of D2 to the method and apparatus known from D1 to solve the problem at hand, thus adding the difference feature.

6.3 Consequently the board finds that the subject-matter of claim 1 does not involve an inventive step, Article 56 EPC, starting from D1 and applying the teaching of D2.

Order

For these reasons it is decided that:

The appeal is dismissed.

The Registrar: The Chairman:

N. Schneider M. Müller

Decision electronically authenticated