



(12) **EUROPEAN PATENT SPECIFICATION**

(43) Date of publication:

20.05.1998 Bulletin 1998/21

(21) Application number: 97117565.8

(22) Date of filing: 10.10.1997

(51) Int. Cl.⁶: **G06F 9/45**, G06F 9/455

(84) Designated Contracting States:

**AT BE CH DE DK ES FI FR GB GR IE IT LI LU MC
NL PT SE**

Designated Extension States:

AL LT LV RO SI

(30) Priority: **18.11.1996 DE 19647628**

(71) Applicant:

**Siemens Nixdorf
Informationssysteme AG
33106 Paderborn (DE)**

(72) Inventor: **Loderer, Johann**

85250 Altomünster (DE)

(54) Simulation of a condition code during code transformations

(57) The invention relates to a method of simulating a condition code (CC) when porting hardware-specific program code (PCU) from a source hardware (M1) to a target hardware (M2) where no hardware support of a condition code (CC) is provided.

Mapping is defined which uniquely associates each possible value (CCW) of the condition code (CC) on the source hardware (M1) with a bit pattern (BM) on the target hardware (M2). To store the bit pattern (BM) associated with the respective current value (CCW), a memory area (REG) is provided on the target hardware (M2). When porting the program code (PCU), any instruction (A_n) potentially influencing the value (CCW) is transformed by being simulated in the

program code (PCZ) by means of a command (B_1 , B_2) generating a result value in the course of the execution of the program and without using branching commands. Also, additional code (ZC_1 , ZC_2 , ZC_3) for the target hardware (M2) is produced which ascertains at runtime the bit pattern (BM) corresponding to the value (CCW) on the source hardware (M1) following execution of the instruction (A_n) by virtue of a prescribed initial bit pattern being subjected to a bit operation whose effect is dependent on the at least one result value generated in connection with the instruction (A_n). The advantage of this method is that for its implementation on the target hardware (M2) no time-consuming jump commands are required.

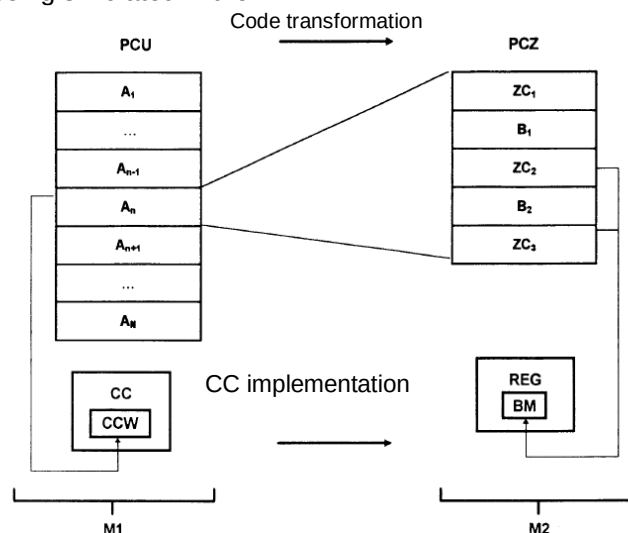


FIG. 1

Description

The invention relates to a method for simulating a condition code when porting hardware-specific program code from a source hardware to a target hardware, in which no hardware support of a condition code is provided.

In particular, on mainframe systems (e. g. VAX or IBM/390 mainframes) a special hardware flag, known as *condition code* or and usually comprising several bits, is often implemented in the computer architecture. It represents a hardware display, which is utilized, *inter alia*, in comparison operations and is set by appropriate machine commands to a value specified depending on the result of the operation. Also arithmetic overflows can be displayed. Subsequent commands, especially conditional branch instructions, can then evaluate the condition code, for example, to specify a jump target.

New computer generations, especially those with RISC architecture, usually have no condition code. As a result, when transforming hardware-specific program code from a source hardware with condition code to program code for such a target hardware without condition code, a software level conditional code must be simulated.

In particular, if the value store of the condition code comprises more than two different values, several individual queries or comparisons must also be performed on the target hardware during the simulation of an instruction in order to ascertain the value of the condition code after a sequence of commands resulting from an instruction that influenced the condition code during the transformation. To do this, the results of the individual queries or comparisons must be linked to one another in order to obtain the condition code. An obvious and commonly used method is to make an "if" cascade out of the individual queries or comparisons, i. e., to build a multi-level branch. The implementation of such a branch into machine code takes place with the aid of jump commands. In the case of such a solution, however, the jump commands which for many computers are 'time-consuming' relative to other machine commands, i.e., requiring a high number of processor clock cycles, will result in loss of speed.

The object of the invention is therefore to specify a method for simulating a condition code when porting hardware-specific program code from a source hardware to a target hardware that is not providing for condition code, in which the above-mentioned disadvantages are avoided.

The object is achieved by a method for simulating a condition code having the features specified in claim 1.

The simulation of the condition code on the target hardware according to the invention is carried out as follows:

First, mapping is defined which uniquely associates each possible value of the condition code on the source hardware with a bit pattern on the target hardware. To store the bit pattern associated with the respective current value of the condition code, a memory area or a separate register is provided on the target hardware.

When porting the program code of the source hardware, any instruction potentially influencing the value of the condition code is transformed by being simulated in the program code for the target hardware by means of a command generating a result value in the course of the execution of the program and without using branching commands.

The command accomplishes both the determination of the result value as a function of the program run and the setting and outputting of this result value, e. g. in a register. This combination of two sub-functions in a single machine command is crucial to avoid branching in the program flow. To simulate an instruction, it may also be necessary to combine several, even different ones, of such result value-generating commands.

For example, a comparison instruction that compares two values on the source hardware and outputs the result of the comparison by setting the condition code to a corresponding value, can be simulated on the target hardware by using a command implementing the elementary comparison operation ">" ("greater than") and providing a result value characterizing the comparison result. Linking several such commands is required if the original comparison instruction can generate more than two different results represented by the condition code (e. g., "greater", "less", "equal").

During the transformation, additional code, hereinafter referred to as "additional code", is generated for the target hardware, by which at runtime the bit pattern corresponding to the value of the condition code on the source hardware after execution of the instruction is ascertained. This is done by subjecting a given output bit pattern to a bit operation, the effect of which depends on the at least one result value generated in connection with the instruction.

The advantage of this method results from the fact that when it is implemented in machine-readable program code on the target hardware, unlike, for example, a branch cascade ("if cascade") controlled by individual queries, no jump commands are required:

The individual result values of the commands for the simulation of an instruction are determined independently of each other

and utilized in the bit operation on the output pattern in a fixed manner for the determination of the bit pattern. Branching of the program flow (and thus jumps) are not required for this purpose.

Advantageous embodiments and further developments of the invention are the subject of the dependent claims.

According to a first configuration of the invention - claim 2 -, the generation of the additional code for ascertaining the bit pattern corresponding to the value of the condition code for an instruction only takes place when porting, if there is the possibility that there is an evaluation of this bit pattern taking place in the program flow before the possible influence of the condition code by subsequent instructions. If, on the other hand, it is clear that regardless of a specific program run the condition code generated in the execution of the relevant instruction on the source hardware would be overwritten in any case, before it could be used in the further course of the program, the generation of the additional code for this instruction when porting is omitted. Thus, unnecessary program code and resulting runtime can be saved without incurring disadvantages.

According to an embodiment of the invention - claim 3 -, at least one of the commands used for the simulation of instructions influencing the value of the condition code, is a comparison command. It can be used to simulate mainly the frequently required comparison operations of the source hardware

A further configuration of the method according to the invention - claim 4 -, relates to a special representation of the simulated condition code on the target hardware. In this case, each value i of the original condition code is associated, on the target hardware, with a bit pattern in which only a bit position uniquely associated with the value i is set to "1" and all other bits of the bit pattern to "0". This special association of bit patterns is particularly advantageous when these bit patterns correspond to the jump masks of the program branching instructions on the source hardware (e.g., in case of IBM/390 systems). This then makes possible a very high-performance query of the simulated condition code by subsequent branching commands resulting from the source code when porting.

According to a further embodiment of the invention - claim 5 -, the bit operation is implemented by using at least one shift command

In a particular variant of this embodiment - claim 6 -, the output bit pattern for the bit operation is associated with a prescribed value of the condition code, and the bit operation involves the initial bit pattern being shifted by a prescribed number of bit positions depending on the at least one result value which appears at runtime in

connection with the simulation of an instruction. In this case, different shifting directions can also occur depending on the at least one result value.

Details of the invention will be explained in more detail by means of exemplary embodiments with reference to the drawing.

FIG. 1 shows schematically the basic principle of the method according to the invention for simulating a condition code;

FIG. 2 shows the value assignment for the condition code in a comparison command;

FIG. 3 shows the structure of a branching cascade in previous solutions;

FIG. 4 shows a specific example of simulating an instruction for a source hardware on a target hardware.

The basic principle of the method according to the invention for simulating a condition code is shown schematically in FIG. 1. A program code PCU to be ported from a source hardware M1 to a target hardware M2 is made up of individual instructions $A_1, A_2, \dots, A_N$. Among these instructions is at least one, e. g. A_n , which influences the value CCW of the condition code CC provided in the machine architecture of the source hardware M1. In the case of computers using a condition code, this is almost always the case in practice, since most of the computer commands influence the value CCW of the condition code CC and the conditional branches which are often required at least in large part rely on the evaluation of the condition code CC.

In a code transformation with which the program code PCU to be ported is converted into program code PCZ for a target hardware M2, each of the instructions $A_1, A_2, \dots, A_N$ is implemented as a command (or a semantically equivalent command sequence) in the program code PCZ. For instructions potentially influencing the value CCW of the condition code CC, additional program code (additional code) is generated for the simulation of the condition code CC on the target hardware M2. In the example shown in FIG. 1, in this way, the instruction A_n in the program code PCU is implemented as a command sequence in the program code PCZ. This command sequence consists of two commands B_1 and B_2 required for the semantic representation of the instruction A_n and an additional code consisting of three subsections ZC_1, ZC_2 and ZC_3 .

In the machine architecture of the source hardware M1, the condition code CC is typically implemented as a special hardware display. The condition code must be simulated on the target

hardware. In addition to the above-mentioned generation of additional code in the code transformation, it is also necessary to provide for a memory area - for performance reasons, a processor register REG is advantageous - in which the respective current value of the condition code simulated by the additional code is stored in the form of an associated bit pattern BM.

The figures FIG. 2, FIG. 3 and FIG. 4 refer to a specific exemplary embodiment in which an IBM/390 computer system is selected as source hardware M1 and a signed "a" value is to be compared to zero.

Different comparison instructions are available on this source hardware for such comparisons, which compare a first numerical value OP1 with a second numerical value OP2 and, - as depicted in FIG. 2, columns one and two - depending on the result of this comparison set the value CCW of the condition code CC. In accordance with a configuration of the invention, bit values BM are respectively uniquely associated with the values CCW thus specified when simulating the condition code on a target hardware (e. g. a Mips R4400 RISC machine) - FIG. 2 third column - where one bit position is set to "1" and all other bit positions are set to "0". In the example, these are the bit patterns "1000" for equality of both numerical values OP1 and OP2, "0100" for the case $OP1 < OP2$ and "0010" for the case $OP1 > OP2$. In addition, the value $CCW = 3$ is defined for the condition code CC or the associated bit pattern "0001" to indicate a numeric overflow. However, in comparison operations in which both operands (numerical values) are well-defined, an overflow does not matter, so that it will not be discussed in more detail below.

FIG. 3 depicts, first, a method obvious to the person skilled in the art for generating the bit pattern associated with the condition code. In a branching cascade, the comparison of the numerical value a selected as the first operand OP1 with the numerical value "0" selected as the second operand OP2 is subdivided into two individual queries. The first individual query checks the equality of both operands and sets the bit pattern BM to value "1000" set in FIG. 2, if the equality is fulfilled. Otherwise, the second individual query follows, which checks whether the numerical value a is greater than the second numerical value "0". According to the comparison result, the bit pattern BM is then set. During the implementation of the branching cascade in program code there is a jump command, with which the second individual query is called in case of $OP1 \neq OP2$ ($a \neq 0$). However, with this method, the jump command which for many computers is 'time-consuming' relative to other machine commands, i.e., requiring

a high number of processor clock cycles, results in significant loss of speed.

In contrast, the method according to the invention for simulating a condition code in a code transformation does not require jumps. For the above example of a comparison instruction, FIG. 4 depicts the command sequence generated in the program code PCZ for the target hardware M2 and consisting of the subsections ZC_1 , ZC_2 , ZC_3 of the additional code and the commands B_1 and B_2 . The last three columns of the table shown in FIG. 4 indicate in each line in each case for the cases $a = 0$, $a > 0$ and $a < 0$ the current value of the bit pattern BM or the contents of two register variables t_1 and t_2 after execution of the command appearing in the line.

As a first step, an output bit pattern is defined in the subsection ZC_1 , which corresponds to the case $OP1 < OP2$, ($a < 0$) and is thus associated with a permissible value ($CCW = 1$) of the condition code CC.

Command B_1 represents a first command needed to simulate the comparison instruction present in the program code PCU and serves to check the equality of both operands ($OP1 = a$, $OP2 = 0$). The result of this first individual query is stored as the result value "1" (for equality) or "0" (for inequality) in the register variable t_1 . In the following step, subsection ZC_2 , a first part of a two-part bit operation acting on the (output) bit pattern BM takes place. The bit operator in the example is the shift operator "<<", which causes the bit pattern BM to be shifted by the number of bit positions (0 or 1) stored in a register variable t_1 in a first direction (to the left).

The subsequent command B_2 represents the second command needed to simulate the comparison instruction present in the program code PCU. It checks whether the first operand ($OP1 = a$) is greater than the second one ($OP2 = 0$). The result of this second individual query is stored as result value "1" (for $a > 0$) or "0" (for $a < 0$) in the register variable t_2 . In the final step, subsection ZC_3 , the second part of the two-part bit operation takes place. It acts on the bit pattern BM already subjected to the first part of the bit operation. The bit operator is now the shift operator ">>" complementary to the first bit operator, which causes the bit pattern BM to be shifted by the number of bit positions (0 or 1) stored in the register variable t_2 in the direction opposite to the first shift direction (i. e. to the right). The bit pattern BM resulting in this step is uniquely associated with the value CCW of the condition code CC occurring after execution of the original comparison instruction on the source hardware M1 and can be evaluated by subsequent instructions in the program code PCZ instead of the original condition code CC of the source hardware M1.

Claims

1. A method for simulating a condition code (CC), used for the execution of hardware-specific program code (PCU) for a source hardware (M1), in the program code (PCZ), obtained following transformation, for a differing target hardware (M2) with a machine architecture which operates without a condition code (CC),
wherein

- a memory area (REG) holding the respective current value (CCW) of the simulated condition code (CC) in the form of an associated bit pattern (BM) is provided on the target hardware (M2), and
- each instruction (A_n) influencing the value (CCW) of the condition code (CC) in the program code (PCU) for the source hardware (M1) is simulated in the program code (PCZ) for the target hardware (M2) without using branching commands and using at least one command (B1, B2) which generates a result value during the program execution, and
- additionally code (ZC_1, ZC_2, ZC_3) is produced which ascertains at runtime the bit pattern (BM) corresponding to the value (CCW) of the condition code (CC) on the source hardware (M1) following execution of the instruction (A_n) by virtue of a defined initial bit pattern being subjected to a bit operation whose effect is dependent on the at least one result value generated in connection with the instruction.

2. The method according to any one of the preceding claims,
characterized
in that during the transformation the additional code (ZC_1, ZC_2, ZC_3) is produced for the purpose of ascertaining the bit pattern (BM) corresponding to the value (CCW) of the condition code (CC) following execution of an instruction (A_n) only if there is no exclusion to evaluation of this bit pattern (BM) taking place in the program flow prior to the possible influencing of the condition code (CC) by subsequent instructions.

3. The method according to any one of the preceding claims,
characterized
in that at least one of the commands (B1, B2) used to simulate instructions ($A_1, A_2, \dots, A_N$) influencing the value (CCW) of the condition code (CC) is a comparison command.

4. The method according to any one of the preceding claims,
characterized
in that a current value (CCW) of the original condition code is associated, on the target hardware, with a bit pattern (BM) for which only a bit position clearly associated with the value (CCW) is set to "1" and all other bits of the bit pattern (BM) are set to "0".

5. The method according to any one of the preceding claims,
characterized
in that the bit operation is implemented by using at least one shift command

6. The method according to claim 5,
characterized
in that the initial bit pattern for the bit operation is associated with a prescribed value (CCW) of the condition code (CC), and the bit operation involves the initial bit pattern being shifted by a prescribed number of bit positions depending on the at least one result value which appears at runtime in connection with the simulation of an instruction (A_n).

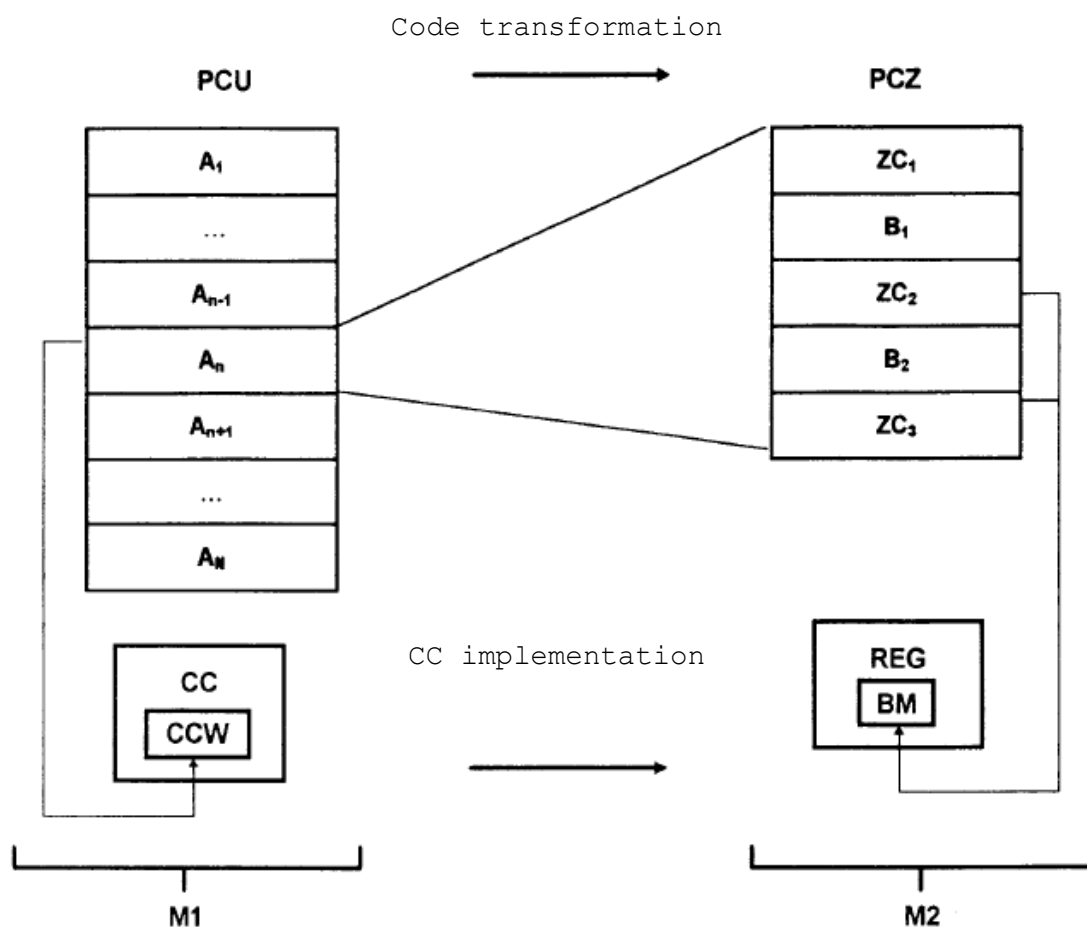


FIG. 1

Comparison result	CCW	BM
OP1 equal to OP2	0	1000
OP1 < OP2	1	0100
OP1 > OP2	2	0010
Overflow	3	0001

FIG. 2

a equal to 0 ?		
yes	no	
Set BM to '1000'	a > 0 ?	
	yes	no
	Set BM to '0010'	Set BM to '0100'

FIG. 3

Command sequence in PCZ		a = 0	a > 0	a < 0
ZC ₁	Set BM to '0100'	BM = 0100	BM = 0100	BM = 0100
B1	If a = 0, set t ₁ = 1, otherwise t ₁ = 0	t ₁ = 1	t ₁ = 0	t ₁ = 0
ZC ₂	BM << t ₁	BM = 1000	BM = 0100	BM = 0100
B2	If a > 0, set t ₂ = 1, otherwise t ₂ = 0	t ₂ = 0	t ₂ = 1	t ₂ = 0
ZC ₃	BM >> t ₂	BM = 1000	BM = 0010	BM = 0100

FIG. 4



Certification of Accuracy of Translation

Sun IP Project # 22-1811

German to English translation of European Patent Office Publication Number EP 0 843 256 A2

Sun IP hereby certifies that the attached translation has been translated by experienced and qualified translators and that, to the best of our knowledge, the translated text reflects the content, meaning and style of the original text and constitutes a true and accurate translation of the original document.

May 11, 2022

Lucia Pacini

PROJECT MANAGER