A method of locating a target value includes loading the target value into elements of a first register. The first register includes N elements (N>0). The method also includes indicating in elements of a second register, which includes N elements corresponding to the first register, whether a corresponding element from data storage matches a corresponding element of the first register.
START LOCATING TARGET VALUE

LOAD TARGET VALUE INTO A FIRST REGISTER HAVING N ELEMENTS

LOAD N ELEMENTS FROM A DATA STORAGE INTO A SECOND REGISTER

COMPARE ELEMENTS OF THE FIRST AND SECOND REGISTER

INDICATE IN A THIRD REGISTER WHICH ELEMENTS MATCH

OBTAIN THE COMPLEMENT OF THE THIRD REGISTER

LOAD A VALUE INTO POSITION VALUE FIELD

ARE THERE MATCHES?

LOAD NEXT N ELEMENTS

YES

END LOCATING TARGET VALUE

FIG. 1
START VERIFYING INITIAL EXTERMA VALUE

LOAD INITIAL EXTREMA VALUE INTO A FIRST REGISTER HAVING N FIELDS

LOAD N ELEMENTS FROM DATA STORAGE INTO A SECOND REGISTER

COMPARE FIELDS OF THE FIRST AND SECOND REGISTER

LOAD NEXT N ELEMENTS

LOAD EXTREMA OF THE FIRST AND SECOND REGISTER INTO THIRD REGISTER

IS THE INITIAL EXTREMA VALUE VALID?

YES

NO

END VERIFYING INITIAL EXTREMA VALUE

FIG. 3
FIG. 4
START VERIFYING INITIAL EXTREMA VALUE

LOAD INITIAL EXTREMA VALUE INTO A FIRST REGISTER HAVING N FIELDS

LOAD N ELEMENTS FROM DATA STORAGE INTO A SECOND REGISTER

COMPARE FIELDS OF THE FIRST AND SECOND REGISTER

LOAD NEXT N ELEMENTS

LOAD EXTREMA OF THE FIRST AND SECOND REGISTER INTO THIRD REGISTER

LOAD COUNT OF INVALID ELEMENTS

IS THE INITIAL EXTREMA VALUE VALID?

YES

NO

END VERIFYING INITIAL EXTREMA VALUE

FIG. 5
ANALYZING STORED DATA

TECHNICAL FIELD

[0001] This disclosure relates to analyzing data in data storage.

BACKGROUND

[0002] Software code may contain instructions to locate specific data in data storage (e.g., memory such as volatile memory, and non-volatile memory, and the like). For example, software code may include instructions to search for a value in memory and to specify its location. Typically, this is accomplished by comparing each value in the data storage to the value to be searched until the location containing the value is determined. For example, typical instructions to locate a value, VALUE, in an array, x, having N elements are:

```
1  pos = -1
2  for (i = 0; i < N; ++i) {
3      if (x[i] == VALUE) {
4          pos = i
5          break;
6      }
7  }
```

[0003] Other software code may contain instructions to validate extrema values such as a maximum value or a minimum value in the data storage. For example, typical instructions to verify a maximum value, MAX, in an array, y, having N elements are:

```
1  MAX = -1
2  for (i = 0; i < N; ++i) {
3      if (y[i] > MAX) {
4          MAX = y[i]
5      }
6  }
```

[0004] Each element in the array, y, is compared to the maximum value, MAX, one at a time.

DESCRIPTION OF THE DRAWINGS

[0005] FIG. 1 is a flowchart of a process for locating a target value in data storage.

[0006] FIG. 2 is a diagram of registers used in locating the target value in the data storage.

[0007] FIG. 3 is a flow chart of a process for verifying an initial extrema value in the data storage.

[0008] FIG. 4 is a diagram of registers used in verifying the initial extrema value for nonnegative integer values in the data storage.

[0009] FIG. 5 is a flow chart of a process for verifying an initial maximum value for negative integer values in the data storage.

[0010] FIG. 6 is a diagram of registers used in verifying the initial maximum value for negative integer values in the data storage. FIG. 7 is a block diagram of a computer system on which the processes of FIGS. 1 and 3 may be implemented.

DESCRIPTION

[0011] Referring to FIGS. 1 and 2, a process 10 may be used to locate a target value in a data storage location (not shown). Instead of comparing each value in an element within the data storage location one-at-a-time with the target value, process 10 searches for the target value N elements at-a-time (N>0) and as will be described below process 10 saves processing time. Each element, for example, may include 8-bits or 16-bits. The target value may be a value required and requested during the execution of a program (e.g., from a compiler), an arbitrary value, or a user chosen value.

[0012] Process 10 may load (12) the target value into each element 30 of a first register 32 having N (N>0) elements. For example, each element may be 8 bits and a target value of 3 may be loaded into 8 elements of first register 32, a 64-bit register. In one embodiment, process 10 may load (12) the target value using a single computer instruction (e.g., in this embodiment, mov). Process 10 may load (14) the first N elements of the storage location into a second register 34. This can be done, for example, using one 8-byte load or eight 8-bit loads.

[0013] Process 10 may compare (16) each element of first register 32 with its corresponding element in second register 34. Process 10 may indicate (18) which elements match the target value by placing a nonzero value into a corresponding element of a third register 36. Process 10 may place a zero value into the corresponding value of the third register if there is no match. With eight one-byte values, the corresponding elements of third register 36 may be set to hexadecimal value 0xff to indicate a match and 0x00 to indicate no match. In one embodiment, process 10 compares (16) and indicates (18) using a single computer instruction (e.g., in this embodiment, cmp.eq).

[0014] Process 10 may obtain (20) the complement of third register 36 and place resulting corresponding values into a fourth register 38. In one embodiment, process 10 obtains (20) the complement using a single computer instruction (e.g., in this embodiment, negate). In other embodiments, (20) may be skipped.

[0015] Process 10 may load (22) a value into a position field 40 indicating if and where there is an element in fourth register 38 having a zero value. A value from “0” to “N-1” may be loaded into a position field 40 to indicate a match and the position (described below) of the element having the matching value. A value of “N” may be loaded into position field 40 to indicate no match.

[0016] Each register (first register 32, second register 34, third register 36, and fourth register 38) stores values in a little-endian format, i.e., the least-significant (“right-most”) element is the least significant. Thus, in second register 34, the least significant element has a value of “1” and the most significant element has a value of “4.” The least significant value has a position value of “0” and the most significant value has a position value of “7.” In FIG. 2, the position value of the element of fourth register 38 containing a zero value is position value “5.” Thus, a value of “5” is placed in position field 40.
[0017] If more than one zero value is in fourth register 38, process 10 may load (22) into position field 40 the position value of the least significant element in fourth register 38 having a zero value.

[0018] In one embodiment, process may load (22) a position field value using a single computer instruction (e.g., in this embodiment, cxx1).1

[0019] Process 10 may determine (24) if there is a match by reading position value 40. If there are no matches (i.e., a value of “N” in field 40, e.g., a value of “8” when there are eight elements), process 10 may load (26) the next N elements (following the first N elements) of the data storage location into second register 34, and process 10 may compare (16) each field of the second register with first register 32, as above.

[0020] If there are matches (e.g., a value from “0” to “N-1” is placed in field 40), process 10 ends.

[0021] A representative example of program code (i.e., machine-executable instructions) for an INTEL® ITanium® processor to implement process 10 is as follows:

```
1  mov  RA = addr of the 1st element of x
2  mov  RPOS = 0
3  mov  VAL = @best
4  L:  
5  ld8  RCONT = [RA], 8  ; //Post-increment by 8 bytes
6  pcmpl eq  RRES = RVAL, RCONT  ;
7  negate NR=RRES  ; //Using e.g. xor
8  cxx1  RIND=RNR  ;
9  cmp  eq p0, p0=RIND, 0  ;
10  out  
11  add  RPOS=8, RPOS/increment RPOS by 8
12  br    L  ;
13  out:  
14  /RIND is different from 8, the value was found
15  /Then, its position pos in array x equals RPOS+RIND
```

[0022] In the above code, “RVAL” corresponds to first register 32, “RCONT” corresponds to second register 34, “RRES” corresponds to third register 36, “NR” corresponds to fourth register 38, and “RIND” corresponds to position field 40. Of course, other code (or even hardware) may be used to implement process 10.

[0023] Referring now to FIGS. 3 and 4, another process is shown for validating extrema values. In more detail, a process 60 (FIG. 3) searches data storage and verifies that an initial extrema value, such as a maximum value or a minimum value, is valid. Process 60 may load (62) an initial extrema value into each element of a first register 82 having N (N=0) elements, e.g., eight elements (FIG. 4). In one embodiment, process 60 may load (62) an initial extrema value using a single computer instruction (e.g., in this embodiment, mxx).

[0024] The initial extrema value is a guess of the actual extrema value for the data storage. Process 60 may be used to determine if that guess is correct. The initial extrema value can come from a user input or the initial extrema value can be determined by a compiler via a compiler optimization setting. For example, a compiler, prior to executing process 60, may read the first 10% of the values in the data storage and may take the extrema from those values. The compiler may then process the remaining 90% of the data storage elements using process 60.

[0025] Process 60 may load (64) N (N=0) elements from the data storage into a second register 84. Process 60 may compare (66) each element’s value in second register 84 to the initial extrema value loaded in first register 82. Process 60 may load (68) the extrema value between the first register and the second register into third register 86. For example, if the initial extrema value is a maximum, the larger of the first register element and the second register element is placed in a corresponding third register element. If the initial extrema value is a minimum, the smaller of the first register element and the second register element is placed in a corresponding third register element.

[0026] In one embodiment, process 60 compares (66) and loads (68) third register 86 using a computer instruction (e.g., in this embodiment, pmx) if the initial extrema value is a maximum and another computer instruction (e.g., in this embodiment, pmn) is used if the initial extrema value is a minimum.

[0027] Process 60 may determine (70) if the initial extrema value is valid by comparing elements from third register 86 to the initial extrema value. If all values match the initial extrema value, then the initial extrema value is valid. If at least one value in the third register does not match the initial extrema value, the initial extrema value is invalid.

[0028] If the initial extrema value is valid, process 60 may load (72) the next N (N=0) elements into second register 84. If the initial extrema value is invalid, process 60 ends.

[0029] A representative example of program code (i.e., machine-executable instructions) for an INTEL® ITanium® processor to implement process 60 is as follows:

```
1  //Process first elements using method in prior art
2  //At this point, MAX contains the local maximum
3  //IA = addr of the 1st element of x on which this method is applied
4  mxx  RMAX = MAX, @best
5  L:  
6  ld8  RVAL = [RA], 8  ; //8 values are loaded in one step
7  pmxl u  RRES = RVAL, RMAX  ;
8  cmp  eq p2,g2=RRES, RMAX, 0  ; //Are all values in RVAL lower than or equal to MAX?
9  pmx eq p3, p3=RRES, RMAX, 0  ; //Are all values in RVAL lower than or equal to MAX?
10  out  
11  out:  
12  /Yes. Process 60 can proceed.
```

[0030] In the above code, “RMAX” corresponds to first register 82, “RVAL” corresponds to second register 84 and “RRES” corresponds to third register 86. The code can be pipelined with an initiation interval of one using the following br.top instruction:

```
1  L:  
2  (p16)  ld8t32 = [RA], 8  // r32 serves a RVAL
3  (p17)  pmxl u  r34 = r32, RMAX  // r32 rotated into r33, r34
4  (p19)  cmpl eq p2, r3, r36, RMAX  //r34 rotated into r36
5  (p3)  bcond method_of_prior_art  
6  (p2)  bcond L:  ; //Yes. Process 60 can proceed.
```

[0031] Of course, other code (or even hardware) may be used to implement process 60.
[0032] Heretofore, comparing each element one-at-a-time to validate an initial extremum value took N (N>0) cycles plus a fixed amount of time (e.g., time to load instructions, etc.), assuming the processing is pipelined with an initiation interval of one. For an array x having N elements, assuming that the values are stored using 8 bits per element, assuming process 60 is applied to the last f(N-1) elements of the data storage where f is the portion of the data storage analyzed by a compiler before executing process 60 and assuming the maximum value was in the first (1-f)N elements, then process 60 takes:

\[(1-f)N \times 8n/a \text{cycles;}
\]

[0033] where a is a constant. Assuming that N is sufficiently large, process 60 takes 7s/8 cycles.

[0034] Referring to FIGS. 5 and 6, other embodiments process values in data storage that may be negative integers instead of nonnegative integers. Process 60 may be modified into a process 80 to account for nonnegative integers.

[0035] Actions 62, 64 and 66 in process 80 (FIG. 5) are the same as actions 62, 64 and 66 of process 60 (FIG. 3).

[0036] For each element in which an initial extremum is false, process 80 may load (88) a hexadecimal value of 0x0f in the corresponding element of a third register 86. For each element in which the initial extremum is true, process 80 may load (88) a hexadecimal value of 0x00 into the corresponding element of third register 86. If the initial extremum value is a maximum, process 80 may determine that the initial extremum value is valid if values in first register 82 are greater than or equal to values in second register 84. If the extremum value is a minimum, process 80 determines that the initial extremum value is valid if values in first register 82 are lower than or equal to the values in second register 84. In one embodiment, process 80 may compare (88) the values using a single computer instruction (e.g., in this embodiment, pcpl.g).

[0037] Process 80 may load (90) into an invalid count field 94 a count of the elements in third register 86 where the initial extremum value is invalid (i.e., elements having a hexadecimal value of 0x0f). In one embodiment, process 80 may load (90) invalid count field 94 by using a single computer instruction (e.g., in this embodiment, popcnt).

[0038] Process 80 may determine (92) if the initial extremum value is invalid by determining if there is a nonzero value in invalid count field 94.

[0039] If the initial extremum value is valid (i.e., a zero value in invalid count field 94), process 80 may load (72) the next N (N>0) elements into second register 84. If the initial extremum value is invalid (i.e., invalid count field 94 contains a nonzero value), process 80 ends.

[0040] A representative example of program code (i.e., machine-executable instructions) for an INTEL® TITANIUM® processor to implement process 80 is as follows:

```
1 RA = addr of the 1st element of x on which process 80 is applied
2 RMX = MAX, @best
3 L:
4 ldr RVAL = [x], #;
5 pcmp. gt RMS = RVAL, RMAX, #;
6 cmp.eq p2, p3 = RCNT, 0;
7 becond method_of_prior;_int
8 b L:
```

[0041] In the above code, “RMAX” corresponds to first register 82, “RVAL” corresponds to second register 84, “RRMS” corresponds to third register 86 and “RCNT” corresponds to invalid count field 94. The instruction above can be pipelined with an initiation interval of one and described as:

```
1 L:
2 p2 (a) ldr RVAL = [x], #;
3 pcmp. gt RMS = RVAL, RMAX, #;
4 p2 (a) popcnt RCNT = RRMS;
5 p2 (a) cmp.eq p2, p3 = RCNT, 0;
6 p3 becond method_of_prior;_int
7 p3 be,ecmp L;
```

[0042] Of course, other code (or even hardware) may be used to implement process 80.

[0043] FIG. 7 shows a computer 100 for using processes 10, 60 and 80. Computer 100 includes a processor 102, a memory 104, and a storage medium 106 (e.g., hard disk). Storage medium 106 stores operating system 110, data storage 112 and registers 116, and computer instructions 114 which are executed by processor 102 out of memory 104 to perform processes 10, 60 and 80.

[0044] Processes 10, 60 and 80 are not limited to use with the hardware and software of FIG. 7, they may find applicability in any computing or processing environment and with any type of machine that is capable of running a computer program. Processes 10, 60 and 80 may be implemented in hardware, software, or a combination of the two. For example, processes 10, 60 and 80 may be implemented in a circuit that includes one or a combination of a processor, a memory, programmable logic and logic gates. Processes 10, 60 and 80 may be implemented in computer programs executed on programmable computers/machines that each includes a processor, a storage medium or other article of manufacture that is readable by the processor (including volatile and non-volatile memory and/or storage elements), at least one input device, and one or more output devices. Program code may be applied to data entered using an input device to perform processes 10, 60 and 80 and to generate output information.

[0045] Each such program may be implemented in a high level procedural or object-oriented programming language to communicate with a computer system. However, the programs can be implemented in assembly or machine language. The language may be a compiled or an interpreted language. Each computer program may be stored on a storage medium or device (e.g., CD-ROM, hard disk, or magnetic diskette) that is readable by a general or special purpose programmable computer for configuring and operating the computer when the storage medium or device is read by the computer to perform processes 10, 60 and 80. Processes 10, 60 and 80 may also be implemented as one or
more machine-readable storage media, configured with a computer program(s), where upon execution, instructions in the computer program(s) cause a computer to operate in accordance with processes 10, 60 and 80.

[0046] Processes 10, 60 and 80 are not limited to the specific embodiments described herein. For example, the elements are not limited to 8-bit or 16-bit, nor are the registers limited to 64 bits. Rather, the elements and registers can be any combination of sizes that are consistent with the processes described herein.

[0047] In another example, processes 60 and 80 are not limited to the actions described herein. For example, after determining that an extrema value is invalid by another value in the data storage, processes 60 and 80 can overwrite the elements of the first register with a new extrema value and continue processes 60 and 80 with the rest of the data storage elements.

[0048] In still another example, overwriting the registers with the new values may reduce the number of registers used to execute processes 10, 60 and 80.

[0049] Processes 10, 60 and 80 are not limited to the specific processing order of FIGS. 1, 3 and 5. Rather, the blocks of FIGS. 1, 3 and 5 may be re-ordered, as necessary, to achieve the results set forth above.

[0050] Other embodiments not described herein are also within the scope of the following claims.

What is claimed is:

1. A method of locating a target value, comprising:
   loading the target value into elements of a first register,
   the first register comprising N elements (N>0); and
   indicating in elements of a second register, comprising N elements corresponding to the first register, whether a corresponding element from data storage matches a corresponding element of the first register.

2. The method of claim 1, further comprising:
   indicating a position of at least one element in the data storage containing the target value based on the contents of the second register.

3. The method of claim 1, further comprising:
   indicating a position of at least significant element containing the target value.

4. The method of claim 1, further comprising:
   taking a complement of the second register; and
   loading the complement of the second register into a third register comprising corresponding N elements.

5. The method of claim 1, further comprising:
   overwriting the second register with its complement.

6. A method of verifying if an initial extrema value is valid, comprising:
   loading the initial extrema value into elements of a first register, the first register comprising N elements (N>0); and
   indicating in a second register, comprising N elements, the extrema values between corresponding elements in data storage and corresponding elements in the first register.

7. The method of claim 6, further comprising:
   indicating when an initial extrema value is invalid.

8. The method of claim 7, wherein the extrema value comprises a maximum; and
   wherein indicating comprises determining if an element in the second register is greater than the initial extrema value.

9. The method of claim 6, wherein the initial extrema value is determined by a user.

10. The method of claim 6, wherein the initial extrema value is determined by a compiler.

11. An apparatus comprising:
   circuitry, for locating a value, to:
   load the target value into elements of a first register, the first register comprising N elements (N>0); and
   indicate in elements of a second register, comprising N elements corresponding to the first register, whether a corresponding element from data storage matches a corresponding element of the first register.

12. The apparatus of claim 11, further comprising:
   indicating a position of at least one element in the data storage containing the target value based on the contents of the second register.

13. The apparatus of claim 11, further comprising:
   indicating a position of at least significant element containing the target value.

14. The apparatus of claim 11, further comprising:
   taking a complement of the second register; and
   load the complement of the second register into a third register comprising corresponding N elements.

15. The apparatus of claim 11, further comprising:
   overwrite the second register with its complement.

16. An apparatus comprising:
   circuitry, for locating a value, to:
   load the initial extrema value into elements of a first register, the first register comprising N elements (N>0); and
   indicate in a second register, comprising N elements, the extrema values between corresponding elements in data storage and corresponding elements in the first register.

17. The apparatus of claim 16, further comprising:
   indicate when an initial extrema value is invalid.

18. The apparatus of claim 17, wherein the extrema value comprises a maximum; and
   wherein indicating comprises determining if an element in the second register is greater than the initial extrema value.

19. The apparatus of claim 16, wherein the initial extrema value is determined by a user.

20. The apparatus of claim 16, wherein the initial extrema value is determined by a compiler.
21. An article comprising a machine-readable medium that stores executable instructions for locating data, the instructions causing a machine to:

load the target value into elements of a first register, the first register comprising N elements (N>0); and

indicate in elements of a second register, comprising N elements corresponding to the first register, whether a corresponding element from data storage matches a corresponding element of the first register.

22. The article of claim 21, further comprising instructions causing a machine to:

indicate a position of at least one element in the data storage containing the target value based on the contents of the second register.

23. The article of claim 21, further comprising instructions causing a machine to:

indicate a position of a least significant element containing the target value.

24. The article of claim 21, further comprising instructions causing a machine to:

take a complement of the second register; and

load the complement of the second register into a third register comprising corresponding N elements.

25. The article of claim 21, further comprising instructions causing a machine to:

overwrite the second register with its complement.

26. An article comprising a machine-readable medium that stores executable instructions for locating data, the instructions causing a machine to:

load the initial extrema value into elements of a first register, the first register comprising N elements (N>0); and

indicate in a second register, comprising N elements, the extrema values between corresponding elements in data storage and corresponding elements in the first register.

27. The article of claim 26, further comprising instructions causing a machine to:

indicate when an initial extrema value is invalid.

28. The article of claim 27, wherein the extrema value comprises a maximum; and

wherein indicating comprises determining if an element in the second register is greater than the initial extrema value.

29. The article of claim 26, wherein the initial extrema value is determined by a user.

30. The article of claim 26, wherein the initial extrema value is determined by a compiler.

31. A system, comprising:

at least one processor;

memory; and

logic coupled to the processing device and the memory, usable by the at least one processor to:

load a target value into elements of a first register, the first register comprising N elements (N>);

indicate in elements of a second register, comprising N elements corresponding to the first register, whether a corresponding element from data storage matches a corresponding element of the first register;

load the initial extrema value into elements of a third register, the third register comprising N elements (N>0); and

indicate in a fourth register, comprising N elements, the extrema values between corresponding elements in data storage and corresponding elements in the third register.

32. The system of claim 31 wherein the first register is the third register and the second register is the fourth register.

33. The system of claim 31, further comprising logic to:

indicate a position of at least one element in the data storage containing the target value based on the contents of the second register.

34. The system of claim 31, further comprising logic to:

indicate when an initial extrema value is invalid.

* * * * *