POWER EFFICIENCY IN MICROPURCCESSORS

Inventors: Alan Mycroft, Cambridge (GB); Paul Webster, Cambridge (GB); Phil Endecott, Cambridge (GB)

Correspondence Address:
AT&T CORP.
ROOM 2A207, ONE AT&T WAY
BEDMINSTER, NJ 07921

Assignee: AT&T Corp., New York, NY (US)

Appl. No.: 12/011,891

Filed: Jan. 30, 2008

Foreign Application Priority Data

May 19, 2001 (GB) 0112276.1

Publication Classification

Int. Cl.
G06F 1/32 (2006.01)

U.S. Cl. 713/320

ABSTRACT

A method of reducing the power consumption of microprocessor system is provided, wherein said microprocessor system comprises a microprocessor (2) and a memory (4) connected by a bus (6); said memory (4) contains a plurality of data values, each represented by a number of bits, for transmission to said microprocessor (2) via the bus (6); and at least some of said data values contain unused bits; and wherein said method includes assigning values to said unused bits in such a way as to reduce the Hamming distance between successive data values by a greater extent than setting all of said unused bits to an arbitrary predetermined value.
POWER EFFICIENCY IN MICROPROCESSORS

CROSS REFERENCE TO RELATED APPLICATIONS


FIELD OF THE INVENTION

The invention relates to improved power efficiency in microprocessors.

BACKGROUND OF THE INVENTION

The concept of Hamming distance will first be described. The Hamming distance between two binary numbers is the count of the number of bits that differ between them. For example:

<table>
<thead>
<tr>
<th>Numbers in decimal</th>
<th>Numbers in binary (incl. leading zeros)</th>
<th>Hamming distance</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 and 5</td>
<td>0100 and 0101</td>
<td>1</td>
</tr>
<tr>
<td>7 and 10</td>
<td>0111 and 1010</td>
<td>3</td>
</tr>
<tr>
<td>0 and 15</td>
<td>0000 and 1111</td>
<td>4</td>
</tr>
</tbody>
</table>

Hamming distance is related to power efficiency because of the way that binary numbers are represented by electrical signals. Typically a steady low voltage on a wire represents a binary 0 bit and a steady high voltage represents a binary 1 bit. A number will be represented using these voltage levels on a group of wires, with one wire per bit. Such a group of wires is called a bus. Energy is used when the voltage on a wire is changed. The amount of energy depends on the magnitude of the voltage change and the capacitance of the wire. The capacitance depends to a large extent on the physical dimensions of the wire. So when the number represented by a bus changes, the energy consumed depends on the number of bits that have changed—the Hamming distance—between the old and the new value, and on the capacitance of the wires.

If one can reduce the average Hamming distance between successive values on a high-capacitance bus, keeping all other aspects of the system the same, the system’s power efficiency will have been increased.

The capacitance of wires internal to an integrated circuit is small compared to the capacitance of wires fabricated on a printed circuit board due to the larger physical dimensions of the latter.

EP 0,926,596 describes a method of optimizing assembly code of a VLIW processor or other processor that uses multiple-instruction words, each of which comprise instructions to be executed on different functional units of the processor.

BRIEF SUMMARY OF THE INVENTION

According to the invention there is provided a method of reducing the power consumption of a microprocessor system, a memory, a computer readable medium, computer programs, and a microprocessor system, as set out in the accompanying claims.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the invention will now be described, by way of example only, with reference to the accompanying FIGURE, which shows the interconnections between a microprocessor and its memory.

DESCRIPTION OF THE INVENTION

In the accompanying FIGURE, a microprocessor 2 is connected to memory 4 by a number of buses which are implemented on a printed circuit board (not shown).

The embodiments described here aim to reduce the average Hamming distance between successive values on the microprocessor-memory interface buses, shown as the lighter colored nets 6, 8, 10 and 12 in FIG. 1, as this will have a significant influence on power efficiency.

Even in systems where microprocessor and memory are incorporated into the same integrated circuit the capacitance of the wires connecting them will be larger than average, so even in this case reduction of average Hamming distance on the microprocessor-memory interface is worthwhile.

Processor-memory communications perform two tasks. Firstly, the processor fetches its program from the memory, one instruction at a time. Secondly, the data that the program is operating on is transferred back and forth. The embodiments described here focus on instruction fetch, which makes up the majority of the processor-memory communications, but the invention is by no means limited to instruction fetch.

The instruction fetch bus 6 is the bus on which instructions are communicated from the memory 4 to the processor 2. Embodiments described here aim to reduce the average Hamming distance on this bus, i.e. to reduce the average Hamming distance from one instruction to the next.

It is common for instruction sets to be redundant, i.e. for instructions to contain information that is ignored by the processor. In particular some instructions contain unused bits.

For example in the instruction set considered here all instructions are 32 bits long. There are three instruction formats:
Bits marked ‘X’ are unused bits. The other bits convey useful information to the processor.

In two of these formats, 31 of the 32 bits are used. In the third format, 26 of the 32 bits are used. The remaining six bits are completely ignored by the processor. Some further features of our illustrative instruction set mean that other bits will also sometimes be unused, but the exact details of this have been omitted for clarity.

Any or all of the unused bits can be assigned to a combination of ‘0’s or ‘1’s without changing the meaning of the program.

Many other common processor instruction sets also have unused bits. Typically, existing compiler tool chains will set all of the unused bits to ‘0’.

Although the processor ignores these bits, they still contribute to the average Hamming distance for instruction fetches. The embodiments described here assign values to unused bits in a way that reduces the Hamming distance and hence maximizes the power efficiency.

For example, consider the following sequence of three instructions:

A: X011100001101000110100011010011
B: X00000001011110000101000001000111
C: X00110111001110000000100101111100

Note the group of unused bits, marked ‘XXXX’ in instruction B. A conventional system might set all unused bits to ‘0’ giving the following code:

A: 0011100001101000110100011010011
B: 00000001011110000101000001000111
C: 0000101111001110001110000110011100

The Hamming distances between these instructions are:

From A to B: 16
From B to C: 22

One embodiment described here instead gives the unused bits the following values:

A: 0011100001101000110100011010011
B: 00000001011110000101000001000111
C: 000010111100111100001110000110011100

In this case the Hamming distances have been reduced to:

From A to B: 13
From B to C: 21

This technique does not require any modifications to the microprocessor. Power is saved through only changing the program bit pattern.

A first embodiment of the invention uses the following method for assigning unused bits in a sequence of instructions:

For the first instruction in the program, set any unused bits to 0.

Considering each subsequent instruction in the program sequentially:

Considering each bit in this instruction:

If this bit is unused, assign the value of the corresponding bit from the previous instruction to it.

An example of this method will now be given using the following sequence of 8-bit instructions:

Bit number: 76543210

01X01X01
X01X01X0
XX000X1X
00X1X100
1X001X00

The changes to the bits of the first three instructions will now be given in detail.

The first instruction is 01X01X01.

Bit 7 of instruction 0 ⇒Do nothing
Bit 6 of instruction 1 ⇒Do nothing
Bit 5 of instruction X ⇒Set it to 0
Bit 4 of instruction 0 ⇒Do nothing
Bit 3 of instruction 1 ⇒Do nothing
Bit 2 of instruction X ⇒Set it to 0
Bit 1 of instruction 0 ⇒Do nothing
Bit 0 of instruction 1 ⇒Do nothing

After processing the first instruction it has been changed to 01001001.

The second instruction is X001X010.

Bit 7 of instruction X ⇒Set it to 0
Bit 6 of instruction 0 ⇒Do nothing
Bit 5 of instruction X ⇒Set it to 0
Bit 4 of instruction 0 ⇒Do nothing
Bit 3 of instruction 1 ⇒Do nothing
Bit 2 of instruction X ⇒Set it to 1
Bit 1 of instruction 0 ⇒Do nothing
Bit 0 of instruction 1 ⇒Do nothing

After processing the second instruction it has been changed to 00110101.

The third instruction is XX000X1X.

Bit 7 of instruction X ⇒Set it to 0
Bit 6 of instruction 0 ⇒Do nothing
Bit 5 of instruction X ⇒Set it to 0
Bit 4 of instruction 0 ⇒Do nothing
Bit 3 of instruction 0 ⇒Do nothing
Bit 2 of instruction X ⇒Set it to 0
Bit 1 of instruction 0 ⇒Do nothing
Bit 0 of instruction 1 ⇒Do nothing

After processing the third instruction it has been changed to 00001010.

The complete sequence of output instructions, after processing according to the method of the first embodiment, is given in the following table.

<table>
<thead>
<tr>
<th>Input instruction</th>
<th>Output instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>00X1X100</td>
<td>00001000</td>
</tr>
<tr>
<td>1X001X00</td>
<td>10001100</td>
</tr>
<tr>
<td>1X001X00</td>
<td>10001100</td>
</tr>
<tr>
<td>00001X00</td>
<td>00001100</td>
</tr>
<tr>
<td>01X0XX10</td>
<td>01001110</td>
</tr>
<tr>
<td>10000XX1</td>
<td>10000111</td>
</tr>
<tr>
<td>01X100X0</td>
<td>01001010</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input instruction</th>
<th>Output instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>00X1X100</td>
<td>00001000</td>
</tr>
<tr>
<td>1X001X00</td>
<td>10001100</td>
</tr>
<tr>
<td>1X001X00</td>
<td>10001100</td>
</tr>
<tr>
<td>00001X00</td>
<td>00001100</td>
</tr>
<tr>
<td>01X0XX10</td>
<td>01001110</td>
</tr>
<tr>
<td>10000XX1</td>
<td>10000111</td>
</tr>
<tr>
<td>01X100X0</td>
<td>01001010</td>
</tr>
<tr>
<td>10000XX1</td>
<td>10000111</td>
</tr>
<tr>
<td>01X100X0</td>
<td>01001010</td>
</tr>
</tbody>
</table>
In this example, the mean inter-instruction Hamming distance after the unused bits have been assigned is 2.61. If all unused bits had been assigned to zero, then the mean inter-instruction Hamming distance would be 2.92, indicating a power saving of around 5%.

This method produces optimal results for straight-line code, i.e. code that has no branches in the flow-of-control. To take into account non-linear flow-of-control a more sophisticated method is required, as will be described below.

The following table shows an example of a fragment of pseudo high-level code and corresponding pseudo assembly language instructions:

<table>
<thead>
<tr>
<th>Input instruction</th>
<th>Output instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>1XX00X0X0</td>
<td>11100100</td>
</tr>
<tr>
<td>01X01000</td>
<td>01101000</td>
</tr>
<tr>
<td>XX001X0X0</td>
<td>01001000</td>
</tr>
<tr>
<td>11001X0X0</td>
<td>11000100</td>
</tr>
</tbody>
</table>

There are two possible paths that control-flow can take through this code. If a is equal to 1 then the sequence of instructions executed is 1 ⇒ 2 ⇒ 3 ⇒ 4 ⇒ 5 ⇒ 6 ⇒ 7. If a is not equal to 1 then the sequence is 1 ⇒ 2 ⇒ 5 ⇒ 6 ⇒ 7.

The simple algorithm presented above would assign unused bits as if the execution sequence were 1 ⇒ 2 ⇒ 3 ⇒ 4 ⇒ 5 ⇒ 6 ⇒ 7. This is not necessarily the optimal assignment for either of the actual execution sequences.

A second embodiment of the invention incorporates an unused bit assignment method that takes into account flow of control.

When an unused bit is both preceded and followed by used bits in the adjacent instructions, setting the unused bit to the value of either the preceding bit or the following bit will optimize Hamming distance. For example:

<table>
<thead>
<tr>
<th>Preceding bit</th>
<th>Unused bit</th>
<th>Following bit</th>
</tr>
</thead>
<tbody>
<tr>
<td>A:</td>
<td>0</td>
<td>X</td>
</tr>
<tr>
<td>B:</td>
<td>1</td>
<td>X</td>
</tr>
</tbody>
</table>

In example A, the value of the preceding bit can be copied into the unused bit giving the bit-sequence 001, or the value of the following bit can be copied into the unused bit giving the bit-sequence 011. In both cases there is exactly one transition overall. In example B, whichever bit's value is copied into the unused bit it will be a 1, giving no transitions in either case.

The first embodiment described above always copies from the preceding instruction. A modification of the first embodiment could run in reverse and always copy from the following instruction. The method of the second embodiment may copy from either.

In the example, instruction 2 is a point of divergence, because the following instruction can be either instruction 3 or instruction 5. Instruction 6 is a point of convergence, because the preceding instruction can be either instruction 4 or instruction 5.

Instructions at points of convergence have more than one possible preceding instruction. If we used only preceding instructions to determine how to assign unused bits we would have to make a decision about which of the two possible preceding instruction to use. Instead in these cases we can use the following instruction as the basis for unused bit assignment. So in the example, unused bits in instruction 6 are filled in from instruction 7.

The method of the second embodiment is based on the following three rules:

1) If an instruction has more than one preceding instruction, i.e. it is at a point of convergence, assign unused bits based on the following instruction.

2) If an instruction has more than one following instruction, i.e. it is at a point of divergence, assign unused bits based on the preceding instruction.

3) If an instruction has exactly one preceding and exactly one following instruction, i.e. it is neither a point of convergence nor divergence, then assign unused bits based on either the preceding or the following instruction.

The following table shows how this can be applied to the example shown above:

<table>
<thead>
<tr>
<th>Possible preceding instructions</th>
<th>Possible following instructions</th>
<th>Assign unused bits based on</th>
</tr>
</thead>
<tbody>
<tr>
<td>compare 'a' with 1 (1)</td>
<td>(None)</td>
<td>2</td>
</tr>
<tr>
<td>branch if not equal, to L1 (2)</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>set 'b' to 0 (3)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>jump to L2 (4)</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>L1: set 'c' to 1 (5)</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>L2: add 1 to 'd' (6)</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td>set 'e' to 'a' (7)</td>
<td>6</td>
<td>7</td>
</tr>
</tbody>
</table>

In a variant embodiment an instruction's unused bits are based on a non-adjacent instruction. For example, instruction (5) could be based on instruction (2). This will typically occur at the target of an unconditional branch. In an implementation of this algorithm, such an assignment may be less practical than an assignment from an adjacent instruction because it requires that the implementation compute the target of the branch. In some cases, this may even be impossible, particularly when the branch target is computed at run time.

The following are hypothetical machine code bit patterns corresponding to these instructions, including some unused bits:

<table>
<thead>
<tr>
<th>Compare with 1</th>
<th>Branch if not equal to L1</th>
<th>Set b to 0</th>
<th>Jump L2</th>
<th>Set c to 1</th>
<th>Add 1 to d</th>
<th>Set e to a</th>
</tr>
</thead>
<tbody>
<tr>
<td>00001X00</td>
<td>01XX0X10</td>
<td>1000XX1X</td>
<td>01X0X0X0</td>
<td>1XX0X0X0</td>
<td>01X01000</td>
<td>XX001X0X</td>
</tr>
</tbody>
</table>
In accordance with the second embodiment, combining the above two tables gives the following assignments for unused bits, in which the arrows show how unused bits are assigned from adjacent instructions.

Where an arrow joins a used bit to an unused bit the unused bit can be assigned from that used bit. For example, bit 3 of instruction (2) can be assigned from bit 3 of instruction (1). When a used bit is connected to an unused bit, and that unused bit is connected in turn to other unused bits, the value can be propagated to all of them. For example, bit 1 of instruction (2) can be propagated to unused bit 1 in instructions (3) and (4).

Here is the complete list of the assignments in accordance with the second embodiment:
Some unused bits can remain unassigned after the method of the second embodiment has been carried out. This will occur when a bit is unused in all instructions between a point of convergence and a point of divergence. In the example, bit 2 of instructions (1) and (2) are unassigned for this reason. A third embodiment, given later, will “seed” the group of unused bits using a used bit in an adjacent instruction. In this example, bit 2 of instruction (2) could be seeded with a ‘0’ from instruction (3).

The mean inter-instruction Hamming distances for this example are:

<table>
<thead>
<tr>
<th>Instruction sequence</th>
<th>Mean Hamming distance with unused bits...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set to ‘0’</td>
<td></td>
</tr>
<tr>
<td>Set taking flow of control into account</td>
<td></td>
</tr>
</tbody>
</table>

-ideal-continued-

Possible paths through this code are:

-ideal-continued-

Difficulties arise when an instruction is both a point of convergence and a point of divergence. This will occur when a branch instruction leads to another branch instruction. The example in the following table illustrates this.
In a non-pipelined processor, the possible execution sequences for this code are 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 and 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 6 \rightarrow 7. For a pipelined processor that fetches one extra instruction after taken branches the possible execution sequences are 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 and 1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7, where [n] indicates the fetched-but-discard instruction.

The second and third embodiments can function correctly for pipelined processors with only a minor change. “Points of divergence”, which were previously considered to be branch instructions, arm now the instructions an appropriate distance after the branch instructions.

Let C be the set of instructions that are at points of divergence. This means all instructions that are labeled as branch targets.

Let D be the set of instructions that are at points of divergence. This means all branch instructions, or if pipelining is being taken into account, all instructions that are the appropriate distance after a branch instruction.

Let E be the set of pairs (I, J) where I and J are instructions that satisfy either or both of the following conditions:

1. J is not an element of C, and I immediately precedes J
2. J is not an element of D, and I immediately follows J

(Note: Each element of E corresponds to an arrow in FIGURE, with I being the instruction at the tail of the arrow and J being the instruction at the head of the arrow).

For each bit in turn:

Let B be the bit-number of the current bit
While there are any instructions where bit B is unused
While there is an element (I, J) of E, such that bit B of instruction J is unused and bit B of I is used
Set bit B of instruction J to the value of bit B of instruction I
End-while
If there are still any instructions where bit B is unused
(Note: This step implements the “seeding” process mentioned earlier)
Find any two instructions I and J, such that bit B of instruction I is used, bit B of J is unused, and I and J are adjacent instructions (for the purposes of this step, the first instruction in the program should be considered to be preceded by and the last instruction followed by an instruction containing all zeros)
Set bit B of instruction J to the value of bit B of instruction I
End-if
End-while
End-for

What is claimed is:

1. A method of reducing the power consumption of a microprocessor system which comprises a microprocessor and a memory connected by at least one bus, said memory containing a plurality of data values, each represented by a plurality of bits, for transmission to said microprocessor via said at least one bus, and at least some of said data values containing unused bits, the method comprising the steps of:
   - reading an initial data value from memory;
   - setting unused bits in the initial data value to a predetermined bit value;
   - separately considering each remaining data value in sequence, assigning a bit value to each unused bit in the considered data value, the assigned bit value being equal to a bit value of a corresponding bit in an adjacent data value.

2. A method as claimed in claim 1, wherein the data values are instructions for execution by the microprocessor.

3. A method as claimed in claim 1, wherein said adjacent data value is a data value to be transmitted to the microprocessor via the at least one bus immediately before or after transmission of the considered data value.

4. A method as claimed in claim 1, wherein said adjacent data value is stored immediately before the considered data value in said memory.

5. A method as claimed in claim 1, wherein said adjacent data value is stored immediately after the considered data value in said memory.

6. A method of reducing the power consumption of a microprocessor system which comprises a microprocessor and a memory connected by at least one bus, said memory containing a plurality of instructions, each represented by a number of bits, for transmission in a sequence for execution to said microprocessor via said at least one bus, and at least some of said instructions containing unused bits, the method comprising the steps of:
   - for an initial instruction, assigning unused bits to a predetermined bit value;
   - for remaining instructions, taken in the sequence:
     - if the instruction has more than one possible following instruction, assigning unused bits in the instruction to be equal to a bit value of a corresponding bit in a preceding instruction;
     - if the instruction has more than one possible preceding instruction, assigning unused bits in the instruction to be equal to a bit value of a corresponding bit in a following instruction; and
     - in other instructions, assigning unused bits in the instruction to be equal to a bit value of a corresponding bit in an adjacent instruction; and storing the instructions in the memory.

7. A method as claimed in claim 6, wherein, in the other instructions, the unused bits in the instruction are assigned to be equal to a bit value of a corresponding bit in a following instruction.

8. A method as claimed in claim 6, wherein, in the other instructions, the unused bits in the instruction are assigned to be equal to a bit value of a corresponding bit in a preceding instruction.

9. A method as claimed in claim 6, wherein assignment of unused bits in the instruction is based on a consideration of the probabilities of different paths to and from said instruction.

10. A method as claimed in claim 6, wherein in the case of an instruction having more than one possible preceding instruction and more than one possible following instruction, assignment of unused bits in the instruction is based on a consideration of bits in multiple preceding and following instructions.
11. A method as claimed in claim 6, wherein in the case of an instruction having more than one possible preceding instruction and more than one possible following instruction, assignment of unused bits in the instruction is based on relative probabilities of each path.

12. A method of reducing the power consumption of a microprocessor system which comprises a microprocessor and a memory connected by at least one bus, said memory containing a plurality of instructions, each represented by a number of bits, for fetching via said at least one bus in a sequence for execution by said microprocessor, and at least some of said instructions containing unused bits, said microprocessor being a pipelined microprocessor that fetches n extra instructions ahead in the sequence, the method comprising the steps of:

- for an initial instruction, assigning unused bits to a predetermined bit value;
- for remaining instructions, taken in the sequence:
  - if an instruction has more than one possible following instruction, assigning unused bits in a subsequent instruction that is n instructions after the instruction, to be equal to a bit value of a corresponding bit in an instruction preceding the subsequent instruction;
  - if the instruction has more than one possible preceding instruction, assigning unused bits in the instruction to be equal to a bit value of a corresponding bit in a following instruction; and
  - in remaining instructions with unused bits, assigning unused bits in the instruction to be equal to a bit value of a corresponding bit in an adjacent instruction; and

storing the instructions in the memory.

* * * * *