

PARALLEL INSTRUCTION EXTRACTION METHOD AND READABLE STORAGE MEDIUM

TECHNICAL FIELD OF THE INVENTION

The present invention relates to the technical field of processors, in
5 particular to a method for extracting instructions in parallel and a readable
storage medium.

BACKGROUND OF THE INVENTION

After more than 50 years of development, the architecture of
microprocessor has experienced vigorous development along with the
10 semiconductor technology, from single-core to physical multi-core and
logical multi-core; from sequential execution to out-of-order execution; from
single-issue to multi-issue. Especially in the server field, the performance of
the processor is constantly being pursued.

At present, server chips are basically superscalar out-of-order
15 execution architecture, and the processing bandwidth of processors is
getting higher and higher, up to 8 or more instructions per clock cycle.

When multiple instructions are fetched at the same time in the
instruction fetch unit, each instruction is fetched in sequence, and the logical
link is relatively long. At present, high-performance processors need to

extract 8 or more bandwidths per clock cycle, and the clock frequency requirements are relatively high. The current implementation method does not meet the requirements.

SUMMARY OF THE INVENTION

5 In view of the deficiency of the prior art, the invention discloses a parallel extraction instruction method and a readable storage medium, which is used for solving the problem that when Instruction Fetch Unit fetches multiple instructions at the same time, the logical link of serial extraction of each instruction is relatively long. At present, high-
10 performance processors need to extract 8 or more bandwidth per clock cycle, and the clock frequency is relatively high. The current implementation method can not meet the requirements.

The present invention is achieved through the following technical solutions:

15 First, the invention discloses a method for parallel extracting instructions, which is characterized in that the method generates the effective vector of the extracted instruction according to the end position vector s_mark_end of the instruction, performs parallel decoding of instructions at each position through logical "AND" and logical "OR"
20 operations, calculates the instruction address and the branch instruction target address operation, and finally fetches multiple instructions in parallel.

Further, in the method, the low 2bit of the first instruction is first determined. If the low 2bit is 00, 01, or 10:00, the first instruction length is 16bit, and if the low 2bit is 11:00, then the first instruction length is 32bit. Then the second instruction is judged from the next byte at the end position
 5 of the instruction. The judgment process is similar to that of the first instruction, and the length of the second instruction is obtained. By analogy, the length of each instruction in cacheline is obtained. After obtaining the length of each instruction, the end position vector `s_end_mark` of each instruction in the instruction stream is obtained.

10 Further, in the method, when an instruction is written, the end position vector `s_end_mark` of each instruction is calculated, and the instruction returned from the writer is in cacheline units. Each cacheline is 64byte. The high and low 32byte of the instruction calculates the end position vector of the instruction respectively, and the high 32byte
 15 instruction speculates that the instruction end position vector `s_end_mark_0` and `s_end_mark_1` with offset 0 and offset 2. According to the low 32byte instruction end position vector, a high 32byte vector is selected as the final instruction end vector of the high 32byte instruction. The instruction end position vector and the instruction are written at the same time.

20 Further, in the method, when the Instruction Fetch Unit starts fetching fingers, the read instruction simultaneously reads the instruction end position vector to verify the prediction information of the BPU and extract the instruction. The instruction end position vector `s_mark_end`

indicates whether the position is the end position of an instruction, a value of 1 means the end position of an instruction, and a value of 0 indicates the end position of an instruction.

Further, in the method, the bandwidth of the Instruction Fetch Unit is
5 each clock cycle of the 32byte, while fetching the instruction, the jump of the branch instruction is predicted, and the prediction is made according to the high 2byte of the branch instruction, and if the jump occurs in the predicted branch instruction, then jump to the target address. After retrieving the instruction from the target address, it is necessary to check the instruction
10 alias error, that is, to determine whether the branch instruction that predicts the jump is a branch instruction, and the type of the branch instruction is the same.

Further, in the method, multiple threads are supported, and all threads share the BPU prediction unit, so the prediction information
15 between threads interferes with each other, and the results of interference include:

BPU may take the middle content of an instruction, that is, not the end of the branch instruction, as the end of the branch instruction where the jump occurs.

20 The type of branch instruction does not match if this BPU information is written by a JA, but when a JALR instruction may predict based on the information of the JAL.

Further, in the method, the BPU information includes a prediction offset `pred_offset` of the BPU and an instruction type `pred_type`. The BPU generates a refresh according to the target predicted by the BPU, and refetches the instruction to detect whether the `s_mark_end` [20] is 1. If not, the position predicted by `pred_offset` is not the end position of a branch instruction, but the middle of an instruction, then a refresh is generated from the address at the end of the most recent instruction in the `pred_offset`, and the instruction is refetched, while clearing the incorrect prediction information in the BPU.

Further, in the method, if the `pred_offset` is the end position of a branch instruction, but when fetching the instruction, it is also determined that the corresponding position of the `s_mark_end` is a branch instruction. If the type of branch instruction is different from the type `pred_type` predicted by BPU, it is also an alias error, and there is no error in the instruction that predicts the jump. But if the predicted destination address is incorrect, the instruction is refetched from the position where `pred_offset` plus 1 is added, and the error message corresponding to that location in the BPU is cleared. Only when the location and type predicted by BPU are correct, the prediction information of BPU is correct, otherwise it is necessary to generate a refresh and retrieve the instruction from the correct address.

Further, in the method, when each instruction has been extracted from the instruction stream, it is determined whether there is a branch instruction in the instruction and whether a jump occurs according to the

prediction information of the BPU. In the instruction, if there are multiple branch instructions, the first instruction has the highest priority, followed by the second instruction, and so on, the refresh is generated according to the target address of the branch instruction. Instruction Fetch Unit refetches the
5 instruction according to this new address, and if there is no branch instruction, all instructions are written to the instruction queue.

In a second aspect, the invention discloses a readable storage medium, which includes a memory for storing execution instructions. When the processor executes the execution instruction stored in the memory, the
10 processor hardware executes the parallel extraction instruction method described in the first aspect.

The beneficial effects of the invention are:

The present invention generates a valid vector for extracting instructions according to the end position vector `s_mark_end` of the
15 instruction, and extracts multiple instructions in parallel through logical "AND" and logical "OR" operations. It can extract multiple instructions in parallel at the same time, there is no serial dependency between each instruction, the timing is easy to converge, and a higher frequency can be obtained. The present invention is particularly suitable for high-
20 performance processors that fetch more than 8 instructions per clock cycle.

BRIEF DESCRIPTION OF THE DRAWINGS

In order to more clearly illustrate the technical scheme in the embodiment of the present invention or the prior art, the drawings that need to be used in the embodiment or the prior art description will be briefly introduced below. It is obvious that the drawings described below
5 are only some embodiments of the invention, and other drawings can be obtained according to these drawings without creative work for those skilled in the art.

Fig. 1 is a schematic diagram of the RISC V instruction mode of the present invention

10 Fig. 2 is an Instruction Fetch Unit top-level diagram of an embodiment of the present invention.

Fig. 3 is an instruction boundary identification diagram of an embodiment of the present invention.

15 Fig. 4 is a vector diagram of the instruction end position of the embodiment of the present invention.

Fig. 5 is a diagram of the cross-boundary instruction jump of the embodiment of the present invention.

Fig. 6 is an alias error check diagram of an embodiment of the present invention.

20 Fig. 7 is a parallel extraction instruction diagram of an embodiment of the present invention.

Fig. 8 is a logic diagram of instruction generation in the second embodiment of the present invention.

Fig. 9 is a diagram of an embodiment of the present invention calculating an instruction address and a branch target address.

5 Fig. 10 is a cross-boundary instruction diagram of an embodiment of the present invention.

DETAILED DESCRIPTION OF THE EMBODIMENTS

In order to make the purpose, technical scheme and advantages of the embodiment of the invention more clear, the technical scheme in the
10 embodiment of the invention will be described clearly and completely in combination with the drawings in the embodiment of the invention. It is clear that the described embodiments are some embodiments of the present invention but not all embodiments. Based on the embodiments of the invention, all other embodiments obtained by ordinary technicians in the
15 field without creative work fall within the scope of the protection of the invention.

Embodiment 1

The embodiment is a method of extracting multiple instructions in parallel according to the end position vector `s_mark_end` of the instruction
20 and extracting the effective vector of the instruction and extracting a

plurality of instructions in parallel through the logic "and" and logic OR operation.

The embodiment is not limited to chips such as CPU, GPU, DSP, etc., and is not limited to any instruction set, any implementation process and
5 other conditions.

In order to explain the principle of this method, we mainly take the RISC-V instruction set as an embodiment.

The RISC V instruction set supports 16bit instruction length, 32bit instruction length, 32bit instruction length, 48bit instruction length and 64bit
10 instruction length, as shown in Fig. 1. This paper mainly describes the method proposed in this paper by taking instructions with length of 16bit and 32bit as embodiments. In order to explain the principle of this method, it is assumed that the instruction bandwidth is 32byte each time, and 8 instructions are extracted at a time.

15 The minimum 2bit for the 16bit instruction is 00, 01, or 10. The minimum 2bit for the 32bit instruction is 11. Therefore, when judging the length of the current instruction, only the lowest 2bit of the instruction needs to be judged. First determine the low 2bit of the first instruction. If the low 2bit is 00, 01, or 10:00, then the length of the first instruction is 16bit. If
20 the low 2bit is 11:00, then the first instruction length is 32bit. Then the second instruction is judged from the next byte at the end of the instruction. The judgment process is similar to that of the first instruction, and the

length of the second instruction can be obtained. And so on, get the length of each instruction in cacheline, as shown in Fig. 2. After getting the length of each instruction, we get the position vector s_end_mark of the end of each instruction in the instruction stream.

5 When an instruction is written from L2 to L1, the end position vector s_end_mark of each instruction is calculated. The instruction returned from L2 is in cacheline units. As shown in Fig. 3, each cacheline is 64byte, and the high and low 32byte of the instruction calculates the end position vector of the instruction respectively. The high 32byte instruction speculates that the
10 two instruction end position vectors $s_end_mark_0$ and $s_end_mark_1$ with offset 0 and offset 2 are calculated. Select a high 32byte vector according to the low 32byte instruction end position vector as the final instruction end vector of the high 32byte instruction, as shown in Fig. 3. Both the end position vector of the instruction and the instruction are written to L1.

15 **Embodiment 2**

The present embodiment is not limited to chips such as CPU, GPU, DSP, etc., and is not limited to any instruction set, any implementation process and other conditions. This paper mainly takes RISC-V instruction set as an embodiment to illustrate. When Instruction Fetch Unit starts to
20 fetch fingers, when reading instructions in L1 CACHE, the instruction end position vector is read out at the same time to verify the prediction information of BPU and extract instructions.

Instruction end position vector `s_mark_end`, indicating whether the position is the end of an instruction. A value of 1 indicates the end position of an instruction; a value of 0 indicates that it is not the end position of an instruction, that is, it may be the operation code of the instruction or the
 5 immediate number within the instruction.

In Fig. 4, the first instruction LUI length is 4byte, `s_mark_end` [28] is 1; the second instruction is C.ADDI, which is a compression instruction of 16bit, `s_mark_end` [26] is 1; the third instruction AUIPC length is 4byte, `s_mark_end` [22] is 1; the fourth instruction JAL length is 4byte, `s_mark_end`
 10 [18] is 1; the fifth instruction LB length is 4byte, `s_mark_end` [14] is 1; the sixth instruction LH length is 4byte, `s_mark_end` [10] is 1; the seventh instruction ADDI length is 4byte, `s_mark_end` [6] is 1; the eighth instruction SRAI length is 4byte, `s_mark_end` [2] is 1; the ninth instruction BNE length is 4byte, this instruction spans the 32byte, so the instruction end position of
 15 the instruction BNE is not in the current instruction block, as shown in Fig. 4.

The bandwidth of Instruction Fetch Unit is each clock cycle of 32byte. Because 16bit/32bit hybrid instructions are supported, it is possible for a branch instruction to span two adjacent instruction blocks. The low 2byte of
 20 the branch instruction is at the end of a 32byte instruction block block0, while the high 2byte is at the head of the adjacent instruction block block1, as shown in Fig. 5.

While fetching the instruction, the jump of the branch instruction is predicted. The prediction is made according to the high 2byte of the branch instruction, and if the branch instruction is predicted to jump, it will jump to the target address. After retrieving the instruction from the target address, it
 5 is necessary to check the instruction alias error, that is, to determine whether the branch instruction that predicts the jump is a branch instruction, and the type of the branch instruction is the same.

Because multiple threads are supported and all threads share the BPU prediction unit, the prediction information between threads interferes
 10 with each other.

The results of interference include: 1 the BPU may take the middle content of an instruction, that is, not the end of the branch instruction, as the end position of the branch instruction where the jump occurs.

2, the type of branch instruction does not match, if this BPU
 15 information is written by a JA, but when a JALR instruction may predict based on the information of the JAL.

The BPU prediction information includes the prediction offset pred_offset of the BPU and the instruction type pred_type. As shown in Fig. 6, pred_offset is 5'd11, that is, the position in the BPU prediction chart is the
 20 end position of a branch instruction, and the jump occurs.

BPU generates a refresh based on the target predicted by BPU and refetches the finger. When fetching the instruction, it is detected whether s_mark_end [20] is 1. It is found that s_mark_end [20] is 0, that is, the position predicted by pred_offset is not the end position of a branch instruction, but the middle of an instruction.

At this point, you need to add 1 from the address at the end of the most recent instruction in pred_offset, generate a refresh, and refetch the instruction, while clearing the wrong prediction information in BPU. By the same token, if pred_offset is the end position of a branch instruction, but when fetching, it is judged that the corresponding position of s_mark_end is a branch instruction, and if the type of branch instruction is different from the type pred_type predicted by BPU, it is also an alias error.

There is no error in predicting the jump of this instruction, but the predicted target address is incorrect. It is necessary to retrieve the instruction from the position where pred_offset plus 1 is added, and clear the error message of the corresponding location in BPU. Only when the location and type predicted by BPU are correct, the prediction information of BPU is correct, otherwise it is necessary to generate a refresh and retrieve the instruction from the correct address.

The embodiment parallel generates and extracts eight instruction effective vectors according to the instruction end vector, and at the same time, the 32byte instruction conjectures the decoding, calculates the

instruction address, calculates the target address of the instruction, and so on. Then, perform "AND" and "OR" logical operations on the effective vector of 8 instructions and speculative decoding, calculate the address of the instruction, calculate the target address of the instruction, etc., to obtain
 5 the extracted instruction and related attributes, as shown in Fig. 7.

Embodiment 3

The present embodiment takes the effective vector generation logic of the second instruction as an embodiment. S_ptr represents the offset of the first instruction in the 32byte instruction stream. S_mark_end represents the
 10 instruction end position vector in the 32byte instruction stream, and each bit of s_mark_end is 1 to indicate the end position of an instruction. Inst_2_val represents the valid vector of the second instruction in the 32byte instruction stream, and the position of 1 indicates the byte at the beginning of the second instruction. Taking the 4byte from this position is a complete
 15 instruction (if it is a compressed instruction of 16bit, it has also been decoded to an 32bit instruction). The effective vector inst_2_val of the second instruction and the 16 instructions obtained by speculative decoding first do the "AND" operation, and then do the "OR" operation to get the second instruction.

20 S_ptr and s_mark_end form an instruction position identification vector of 35bit, which is mapped to another vector inst_2_val in the form of

onehot. The logical mapping relationship that produces the valid vector of the second instruction is shown in the following table:

Table 1 Valid vector map of the second instruction

[illegible]

5

10

15

20

10

15

20

```
00100xxxxxxxxxxxxxxxxxxxxxx100010xxx  0000000000000000000000000000000000000000
```

10

15

20

```
10000xxxxxxxxxx100010xxxxxxxxxxxxxxxxxx 00000000000000100000000000000000
```

	10000xxxxxxxxxx1010xxxxxxxxxxxxxxxxxx	00000000000010000000000000000000
	10010xxxxxx10001000xxxxxxxxxxxxxxxxxx	00000000010000000000000000000000
	10010xxxxxxxx101000xxxxxxxxxxxxxxxxxx	00000000010000000000000000000000
	10010xxxxxxxx100010xxxxxxxxxxxxxxxxxx	00000000000100000000000000000000
5	10010xxxxxxxx1010xxxxxxxxxxxxxxxxxx	00000000000100000000000000000000
	10100xxxx10001000xxxxxxxxxxxxxxxxxx	00000001000000000000000000000000
	10100xxxxxx101000xxxxxxxxxxxxxxxxxx	00000001000000000000000000000000
	10100xxxxxx100010xxxxxxxxxxxxxxxxxx	00000000010000000000000000000000
	10100xxxxxx1010xxxxxxxxxxxxxxxxxx	00000000010000000000000000000000
10	10110xx10001000xxxxxxxxxxxxxxxxxx	00000100000000000000000000000000
	10110xxxx101000xxxxxxxxxxxxxxxxxx	00000100000000000000000000000000
	10110xxxx100010xxxxxxxxxxxxxxxxxx	00000001000000000000000000000000
	10110xxxxxx1010xxxxxxxxxxxxxxxxxx	00000001000000000000000000000000
	1100010001000xxxxxxxxxxxxxxxxxx	00010000000000000000000000000000
15	11000xx101000xxxxxxxxxxxxxxxxxx	00010000000000000000000000000000
	11000xx100010xxxxxxxxxxxxxxxxxx	00000100000000000000000000000000
	11000xxxx1010xxxxxxxxxxxxxxxxxx	00000100000000000000000000000000
	11010101000xxxxxxxxxxxxxxxxxx	01000000000000000000000000000000
	11010100010xxxxxxxxxxxxxxxxxx	00010000000000000000000000000000
20	11010xx1010xxxxxxxxxxxxxxxxxx	00010000000000000000000000000000
	111001010xxxxxxxxxxxxxxxxxx	01000000000000000000000000000000

In the same way, the effective vectors of the remaining instructions can be obtained.

The instruction fetch unit decodes 32 bytes each time, and the RISC-V instruction length is 2 or 4. So the opcodes for instructions start at the even-numbered positions 0, 2, 4, ... 30. Similarly, the end position of the instruction is odd-numbered positions 1, 3, 5, ..., 31.

5 The effective vector `inst_2_val[0]` for the instruction is 1 if the instruction starts at position 0. At the same time, the instruction `inst0` obtained by speculative decoding is fetched, and the length is 4 bytes. When the instruction is a C extension instruction, it has been decoded into an instruction with a length of 4 bytes during speculative decoding.

10 If the instruction starts from position 2, the effective vector `inst_2_val[2]` of the instruction is 1; meanwhile, the instruction `inst1` obtained by speculative decoding is fetched.

 If the instruction starts from position 4, then the effective vector `inst_2_val[4]` of the instruction is 1; meanwhile, the instruction `inst2`
 15 obtained by speculative decoding is fetched.

 If the instruction starts from position 6, then the effective vector `inst_2_val[6]` of the instruction is 1; meanwhile, the instruction `inst3` obtained by speculative decoding is fetched.

 If the instruction starts from position 8, the effective vector
 20 `inst_2_val[8]` of the instruction is 1; meanwhile, the instruction `inst4` obtained by speculative decoding is fetched.

If the instruction starts from position 10, the effective vector `inst_2_val[10]` of the instruction is 1; meanwhile, the instruction `inst5` obtained by speculative decoding is fetched.

If the instruction starts from position 12, the effective vector
 5 `inst_2_val[12]` of the instruction is 1; meanwhile, the instruction `inst6` obtained by speculative decoding is fetched.

If the instruction starts from position 14, the effective vector `inst_2_val[14]` of the instruction is 1; meanwhile, the instruction `inst7` obtained by speculative decoding is fetched.

10 If the instruction starts from position 16, then the effective vector `inst_2_val[16]` of the instruction is 1; meanwhile, the instruction `inst8` obtained by speculative decoding is fetched.

If the instruction starts from position 18, the effective vector `inst_2_val[18]` of the instruction is 1; meanwhile, the instruction `inst9`
 15 obtained by speculative decoding is fetched.

If the instruction starts from position 20, the effective vector `inst_2_val[20]` of the instruction is 1; meanwhile, the instruction `inst10` obtained by speculative decoding is fetched.

If the instruction starts from position 22, the effective vector
 20 `inst_2_val[22]` of the instruction is 1; meanwhile, the instruction `inst11` obtained by speculative decoding is fetched.

If the instruction starts from position 24, then the effective vector `inst_2_val[24]` of the instruction is 1; meanwhile, the instruction `inst12` obtained by speculative decoding is fetched.

If the instruction starts from position 26, the effective vector
 5 `inst_2_val[26]` of the instruction is 1; meanwhile, the instruction `inst13` obtained by speculative decoding is fetched.

If the instruction starts from position 28, the effective vector `inst_2_val[28]` of the instruction is 1; meanwhile, the instruction `inst14` obtained by speculative decoding is fetched.

10 If the instruction starts from position 30, if the current instruction does not cross the boundary, then the effective vector `inst_2_val[30]` of the instruction is 1; meanwhile, the instruction `inst15` obtained by speculative decoding is fetched.

If the current instruction crosses the boundary, the current
 15 instruction is invalid, and this instruction is not fetched until the next 32byte instruction stream is valid.

If the offset of the 1st instruction is not 0, it starts at a non-zero offset. Then the starting position of the 1st instruction is the position of this offset. The positions of other instructions start with the same offset in sequence.

20 The logical expression to get the second instruction is:

$$\text{Inst_2} = ((32\{\text{inst_2_val}[0]\}) \& \text{inst0}) |$$

```

({32{ inst_2_val[2]}}&inst1) |

({32{ inst_2_val[4]}}&inst2) |

({32{ inst_2_val[6]}}&inst3) |

({32{ inst_2_val[8]}}&inst4) |

5   ({32{ inst_2_val[10]}}&inst5) |

    ({32{ inst_2_val[12]}}&inst6) |

    ({32{ inst_2_val[14]}}&inst7) |

    ({32{ inst_2_val[16]}}&inst8) |

    ({32{ inst_2_val[18]}}&inst9) |

10  ({32{ inst_2_val[20]}}&inst10) |

    ({32{ inst_2_val[22]}}&inst11) |

    ({32{ inst_2_val[24]}}&inst12) |

    ({32{ inst_2_val[26]}}&inst13) |

    ({32{ inst_2_val[28]}}&inst14) |

15  ({32{ inst_2_val[30]}}&inst15));

```

Inst0, inst1, ... inst15 are 16 speculatively generated instructions. The circuit implemented by the second instruction is implemented by logic

"AND" and logic "OR" gates, as shown in Fig. 8. Other instructions, according to the same principle, can obtain logical expressions and logical circuit diagrams.

Embodiment 4

5 When calculating the address and target address of an instruction in this embodiment, it is also speculative calculation. The Instruction Fetch Unit fetches 32 bytes each time, and the fetch address is `fetch_address`, which is the base address for calculating the instruction address. Because the length of the RISC V instruction is 2 or 4, the instruction addresses for
10 the speculative calculation of the 16 positions are: `base_address`, `base_address+2`, `base_address+4`, `base_address+8`, `base_address+10`, `base_address+12`, `base_address+14`, `base_address+16`, `base_address+18`, `base_address+20`, `base_address+22`, `base_address+24`, `base_address+28` and `base_address+30`. The address `inst_2_addr` of the second instruction is also
15 obtained using the logic similar to the second instruction. As follows:

$$\text{Inst_2_addr} = (\{64\{ \text{inst_2_val}[0] \} \} \& \text{base_address}) |$$

$$(\{64\{ \text{inst_2_val}[2] \} \} \& (\text{base_address}+2)) |$$

$$(\{64\{ \text{inst_2_val}[4] \} \} \& (\text{base_address}+4)) |$$

$$(\{64\{ \text{inst_2_val}[6] \} \} \& (\text{base_address}+6)) |$$

$$20 \quad (\{64\{ \text{inst_2_val}[8] \} \} \& (\text{base_address}+8)) |$$

```

({64{ inst_2_val[10]}}&( base_address+10)) |

({64{ inst_2_val[12]}}&( base_address+12)) |

({64{ inst_2_val[14]}}&( base_address+14)) |

({64{ inst_2_val[16]}}&( base_address+16)) |

5   ({64{ inst_2_val[18]}}&( base_address+18)) |

({64{ inst_2_val[20]}}&( base_address+20)) |

({64{ inst_2_val[22]}}&( base_address+22)) |

({64{ inst_2_val[24]}}&( base_address+24)) |

({64{ inst_2_val[26]}}&( base_address+26)) |

10  ({64{ inst_2_val[28]}}&( base_address+28)) |

({64{ inst_2_val[30]}}&( base_address+30)));

```

Instructions fetched in the Instruction Fetch Unit, the branch instructions in the instructions include JAL, JALR, BEQ, BNE, BLT, BGE, BLTU, BGEU, C.JAL, CJ, C.BEQZ, C.BNEZ, C.JR and C. JALR. Among them,

15 the destination address of the instructions JAL, BEQ, BNE, BLT, BGE, BLTU, BGEU, C.JAL, C.J, C.BEQZ, C.BNEZ is the addition of the instruction address and the offset. Similarly, it is assumed that each offset of 2 byte is a branch instruction, so it is also speculated that the target address of each instruction can be obtained by parallel computation. It is speculated that the

target addresses of the instructions at 16 locations are: $\text{base_address} + \text{offset}$,
 $\text{base_address} + 2 + \text{offset}$, $\text{base_address} + 4 + \text{offset}$, $\text{base_address} + 8 + \text{offset}$,
 $\text{base_address} + 10 + \text{offset}$, $\text{base_address} + 12 + \text{offset}$, $\text{base_address} + 14 + \text{offset}$,
 $\text{base_address} + 16 + \text{offset}$, $\text{base_address} + 18 + \text{offset}$, $\text{base_address} + 20 + \text{offset}$,
5 $\text{base_address} + 22 + \text{offset}$, $\text{base_address} + 24 + \text{offset}$, $\text{base_address} + 28 + \text{offset}$
and $\text{base_address} + 30 + \text{offset}$. Offset is the offset of the branch instruction.
Inst is an instruction.

The conditional instruction immediate number cond_imm of the
32bit instruction is: cond_imm : {inst[31], inst[7], inst[30:25], inst[11:8],
10 1'b0};

The immediate data of unconditional instruction of 32bit instruction
is: uncond_imm : {inst[31], inst[19:12], inst[20], inst[30:21], 1'b0};

The conditional instruction immediate number cond_imm_c of the
16bit compressed instruction is: cond_imm_c : {inst[12], inst[6:5], inst[2],
15 inst[11:10], inst[4:3], 1'b0};

The unconditional instruction immediate number uncond_imm_c of
the 16bit compressed instruction is: uncond_imm_c : {inst[12], inst[8],
inst[10:9], inst[6], inst[7], inst[2], inst[11], inst[5:3], 1'b0};

Each location may be these four branch instructions, so each location
20 first determines the instruction type, and then calculates a different type of
offset offset. The target address of the second instruction,

Inst_2_target_addr, can also get a logical expression similar to Inst_2_addr, as shown in Fig. 9.

Embodiment 5

The present embodiment determines that a specific branch
 5 instruction type br_type for each location is obtained. Br_type [0] is the conditional instruction of 32bit instruction, br_type [1] is the unconditional instruction of 32bit instruction, br_type [2] is the conditional instruction of 16bit instruction, and br_type [3] is the unconditional instruction of 16bit instruction. Therefore, the offset offset of the branch instruction is obtained
 10 according to br_type and cond_imm, uncond_imm, cond_imm_c and uncond_imm_c.

Since both 16bit and 32bit instructions are supported, there are mixed
 16bit and 32bit instructions in the instruction stream. Each 32byte instruction consists of 8-16 instructions, so an 32bit instruction may exist
 15 across consecutive adjacent 32byte instruction streams. In the instruction extraction module, a 2byte register is used to store the high 2byte of the 32byte instruction stream, and the 2byte is used as the 2byte of cross-boundary instructions.

At the same time, it is determined whether the cross-boundary
 20 instruction occurs in the current 32byte instruction stream, and if so, it is necessary to generate a valid indication signal of the cross-boundary

instruction. When the adjacent 32byte instruction blocks reach the instruction fetch pipeline stage, if the cross-boundary instruction effectively indicates that the signal is 1, it indicates that the first instruction has a cross-boundary situation. At this point, the first instruction consists of two parts,
5 as shown in Fig. 10.

If the cross-boundary instruction effectively indicates that the signal is 0, it means that the first instruction does not cross the boundary. The first instruction is the first instruction of the current 32byte instruction block. Other instructions are taken in turn from the subsequent instruction stream
10 of the first instruction. When the instruction is a branch instruction across the boundary, the BPU prediction information of the instruction also needs to be saved until the adjacent instruction stream is valid, the prediction information of the first instruction is obtained, which is similar to the processing method of getting the first instruction.

15 When each instruction has been extracted from the instruction stream, it is judged whether there are branch instructions in the 8 instructions and whether the jump occurs according to the prediction information of the BPU. Among the 8 instructions, if there are multiple branch instructions, the first instruction has the highest priority, followed by
20 the second instruction, and so on. A refresh is generated according to the target address of the branch instruction, and the Instruction Fetch Unit refetches the instruction according to this new address. If there are no branch instructions, all instructions are written to the instruction queue.

Embodiment 6

The present embodiment discloses a readable storage medium, including a memory for storing execution instructions. When the processor executes the execution instruction stored in the memory, the processor
5 hardware executes a method of extracting instructions in parallel.

In summary, according to the end position vector `s_mark_end` of the instruction, the invention generates a method of extracting the effective vector of the instruction and extracting a plurality of instructions in parallel through the logic "and" and logic OR operation. Multiple instructions can be
10 extracted in parallel at the same time, there is no serial dependency between each instruction, the timing is easy to converge, and a higher main frequency can be obtained. It is especially suitable for high-performance processors that extract more than 8 instructions per clock cycle.

The above embodiments are only used to illustrate the technical
15 scheme of the present invention and not to restrict it. Although the invention is explained in detail with reference to the above-mentioned embodiments, ordinary technicians in the field should understand that it can still modify the technical scheme recorded in the above-mentioned embodiments, or equivalent replacement of some of the technical features;
20 and these modifications or replacements do not deviate the essence of the corresponding technical scheme from the spirit and scope of the technical scheme of the embodiments of the present invention.

CLAIMS

1. A parallel method for extracting instructions, wherein the method generates an effective vector of extracting instructions according to the end position vector s_mark_end of the instructions, and carries
5 out parallel decoding of instructions at each position through logical and logical OR operations, computing instruction addresses and branch instruction target addresses, and finally extracting multiple instructions in parallel.
2. The method for extracting instructions in parallel according to claim
10 1, wherein the low 2bit of the first instruction is first judged, if the low 2bit is 00, 01, or 10:00, the first instruction length is 16 bits; if the low 2bit is 11:00, then the first instruction length is 32bit, then the second instruction is judged from the next byte at the end position of the instruction, the judgment process is similar to that of the first
15 instruction, and the length of the second instruction is obtained, by analogy, the length of each instruction in cacheline is obtained, after obtaining the length of each instruction, the end position vector s_end_mark of each instruction in the instruction stream is obtained.
- 20 3. The method for extracting instructions in parallel according to claim 1, wherein when writing an instruction, the end position vector s_end_mark of each instruction is calculated, and the instruction

returned from the writer is in the unit of cacheline, and each cacheline is 64byte, the high and low 32byte of the instruction calculates the end position vector of the instruction respectively, the high 32byte instruction speculates that the instruction end position vectors s_end_mark_0 and s_end_mark_1 with offset 0 and offset 2 are calculated, according to the low 32byte instruction end position vector, a high 32byte vector is selected as the final instruction end vector of the high 32byte instruction, the instruction end position vector and the instruction are written at the same time.

4. The method for fetching instructions in parallel according to claim 1, wherein when the Instruction Fetch Unit starts to fetch the instruction, the instruction end position vector is read at the same time to verify the prediction information of the BPU and extract the instruction, the instruction end position vector s_mark_end indicates whether the position is the end of an instruction, a value of 1 means the end position of an instruction; and a value of 0 indicates the end position of an instruction.

5. The method for fetching instructions in parallel according to claim 4, wherein the bandwidth of the Instruction Fetch Unit is 32byte each clock cycle, while fetching the instruction, the jump of the branch instruction is predicted, and the prediction is carried out according to the high 2byte of the branch instruction, if the jump occurs in the predicted branch instruction, it jumps to the target

address, after retrieving the instruction from the target address, it is necessary to check the instruction alias error, that is, to determine whether the branch instruction that predicts the jump is a branch instruction, and the type of the branch instruction is the same.

- 5 6. The method for extracting instructions in parallel according to claim 1, wherein multiple threads are supported, and all threads share the BPU prediction unit, so the prediction information between threads interferes with each other, and the interference results include:

10 BPU may take the middle content of an instruction, that is, not the end of the branch instruction, as the end position of the branch instruction where the jump occurs;

the type of branch instruction does not match if this BPU information is written by a JA, but when a JALR instruction may predict based on the information of the JAL.

- 15 7. The method for fetching instructions in parallel according to claim 6, wherein the BPU information includes a prediction offset pred_offset of the BPU and an instruction type pred_type, the BPU generates a refresh according to the target predicted by the BPU, and refetches the instruction to detect whether the s_mark_end [20]
- 20 is 1, if not, the position predicted by pred_offset is not the end position of a branch instruction, but the middle of an instruction, then a refresh is generated from the address at the end position of

the most recent instruction in the `pred_offset`, and the instruction is refetched, while clearing the incorrect prediction information in the BPU.

8. The method for fetching instructions in parallel according to claim 1, wherein, if the `pred_offset` is the end position of a branch instruction, but when fetching the instruction, it is also judged that the corresponding position of the `s_mark_end` is a branch instruction, if the type of branch instruction is different from the type `pred_type` predicted by BPU, it is also an alias error, and there is no error in the instruction that predicts the jump, but if the predicted destination address is incorrect, the instruction is refetched from the position where `pred_offset` plus 1 is added, and the error message corresponding to that location in the BPU is cleared, only when the location and type predicted by BPU are correct, the prediction information of BPU is correct, otherwise it is necessary to generate a refresh and retrieve the instruction from the correct address.
9. The method for extracting instructions in parallel according to claim 1, wherein when each instruction has been extracted from the instruction stream, it is determined whether there is a branch instruction in the instruction and whether a jump occurs according to the prediction information of the BPU, in the instruction, if there are multiple branch instructions, the first instruction has the highest

priority, followed by the second instruction, and so on, the refresh is generated according to the target address of the branch instruction, and the Instruction Fetch Unit refetches the instruction according to this new address, if there are no branch instructions, all instructions
5 are written to the instruction queue.

10. A readable storage medium includes a memory for storing execution instructions, and when a processor executes the execution instructions stored in the memory, the processor hardware executes the method of fetching instructions in parallel according to any one
10 of claims 1-9.