# HW 1

Due date: 2019.10.07 23:59

1. (10 points) Remember the difference between CISC and RISC instruction set, and fill in the table below

| | CISC | RISC |
|---|---|---|
| the length of instruction fixed/flexible | | |
| number of instructions large/small | | |
| use for RAM efficient/deficient | | |
| whether can use pipeline architecture | | |

2. (20 points) CPI calculation

   CPI is defined as 'average cycles per instruction', now calculate the CPI below. Assume the cycle time is 5ns.

| Instruction class | cycles | frequency |
|---|---|---|
| calculate | 1 | 60% |
| RAM operation | 4 | 30% |
| branch | 2 | 10% |

2.1 (5 points) calculate the CPI (no pipeline)

2.2 (2 points) If we use 5 steps pipeline architecture and the ideal CPI=1.0. However, when dealing with branch instruction, we must stall until the next pc location is calculated. Then branch instruction needs to stall ___ cycles. (Regard changing pc as write back a register)

2.3 (5 points) If the 5 stages are 1ns, 1.5ns, 4ns, 3ns and 0.5ns, then the cycle time for pipeline is ____. Now calculate CPI and speed up in situation 2.2

2.4 (3 points) If each pipeline stage also needs 0.02ns more due to register setup delay, what's the speedup?

2.5 (5 points) Branch prediction can speed up the pipeline by guessing an address when meeting branch instruction. Now assume with prediction method, 50% of

the branch instructions don't need to stall now. Calculate the new CPI and speed up in this situation.

PS: You will get some bonus score if you use branch prediction in the code, so try to learn more about it!

3. (10 points) Data dependency

   Here's a sequence of instructions:

   1. lw $s2, 0($s1)
   2. lw $s1, 40($s6)
   3. sub $s6, $s1, $s2
   4. add $s6, $s2, $s2
   5. or $s3, $s6, $zero
   6. sw $s6, 50($s1)

3.1 (5 points) Point out all data dependencies in the sequences, no matter whether there's a hazard.

   example: 3 depends on 1 ($s2)

3.2 (5 points) Assume the 5-stage pipeline with no data forwarding, and the processor stall on hazards. Fill in the table below

| cycle | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | IF | ID | EXE | MEM | WB | | | | | | | | |
| 2 | | IF | | | | | | | | | | | |
| 3 | | | | | | | | | | | | | |
| 4 | | | | | | | | | | | | | |
| 5 | | | | | | | | | | | | | |
| 6 | | | | | | | | | | | | | |

4. (10 points) Draw the picture of pipeline. It will contain the fundamental architecture of 5 steps and data forwarding. You can add branch prediction, cache, or anything you want. This question doesn't take much score and you can copy the picture in ppt. However, please draw your picture carefully and think what will you do when writing your own CPU. That's worth thinking.

# 2019 HW1

2019年10月7日 18:09

1.

| CISC | RISC |
|---|---|
| flexible | fixed |
| large | small |
| efficient | deficient |
| no | yes |

2.1 CPI = 1×60% + 4×30% + 2×10% = 2

2.2 branch IF ID EXE MEM WB
next IF
So needed to stall 4 cycles

2.3 cycle time: 4ns
CPI: 1.0 + 10% × 4 = 1.4
speed up: $\frac{\text{ideal CPI} \times \text{time unpipelined}}{\text{CPI} \times \text{time pipelined}} = \frac{1.0 \times (1+1.5+4+3+0.5)}{1.4 \times 4} = 1.79$

2.4 cycle time: 4.02 ns CPI: 1.4
speed up: $\frac{10}{1.4 \times 4.02} = 1.78$

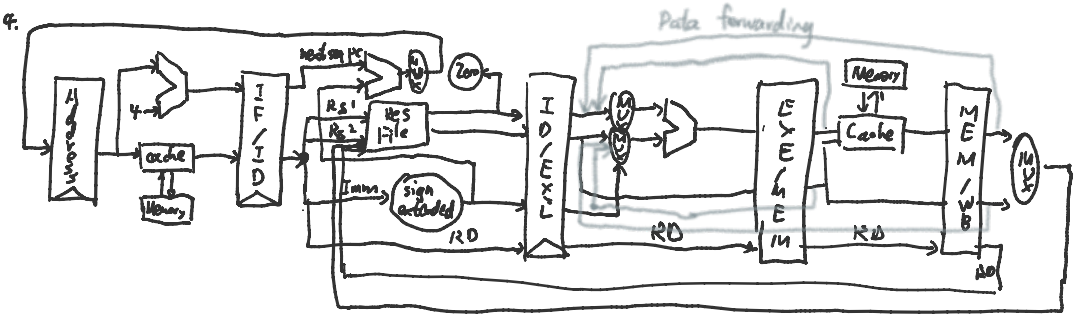
2.5 CPI: 1.0 + 10% × 50% × 4 = 1.2
speed up: $\frac{10}{1.2 \times 4} = 2.08$

3.1 3 depends on 1 ($s0)
3 depends on 2 ($s1)
4 depends on 1 ($s2)
5 depends on 4 ($s6)
6 depends on 2 ($s1)
6 depends on 4 ($s4)

3.2

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | IF | ID | EXE | MEM | WB | | | | | | |
| 2 | | IF | ID | EXE | MEM | WB | | | | | |
| 3 | | | IF | ID | | EXE | MEM | WB | | | |
| 4 | | | | IF | ID | | EXE | MEM | WB | | |
| 5 | | | | | IF | ID | | EXE | MEM | WB | |
| 6 | | | | | | IF | ID | | EXE | MEM | WB |

4.



DDL: 2019.11.17 23:59

# HW2

1. Cache

   Answer questions:

   a. If I use cache as a bridge between main memory and CPU, what will happen when I want to access a block of data in memory? Will it be faster than the case without a cache?

   b. True or false? A split cache usually divided both L1 and L2 caches into data-only cache and instruction-only cache.

   c. Calculate: design a 128KB cache using direct mapping, use 32bit address and 16 bytes per block, calculate the number of bytes used for byte offset, set(index) field, and tag?

   d. Average Memory Access Time(AMAT) = _____. For each of the following optimizations, indicate which part of AMAT is improved?
      1) Using direct-mapped cache
      2) Using larger blocks
      3) Using 4-way set-associative cache
      4) Using second-level cache
      5) Using virtually-addressed cache
      6) Using non-blocking cache

2. Memory

   Answer questions:

   a. Assume you have a 64-bit virtual address, 16KB page size. Then the virtual address 0xABCDE1234CAEF567 will have virtual page number _____, and page index _____.

   b. Based on (a), if page size increases to 1GB, the following of these will increase, decrease or remain unchanged? 1) number of entry in TLB. 2) size of VPN 3) size of physical page number

   c. Rank the time used to recover: 1) L1 cache miss, but L2 hits. 2) page fault 3) TLB miss. 4) cache miss

3. IO

   Some basic knowledge (write down the calculation and answers)

   a. Which of the part costs most of the time when accessing data on the disk?
      a. Settle time
      b. Rotational latency
      c. Seek time
      d. Waiting time

   b. Consider a hard disk with 16 recording surfaces (0-15) having 16384 cylinders (0-16383) and each cylinder contains 64 sectors (0-63). Data storage capacity in each sector is 512 bytes. Data are organized cylinder-wise. A file of size 42797 KB is stored in the disk and the starting disk location of the file is <1200, 9, 40>. What is the cylinder number of the last sector of the file, if it is stored in a contiguous manner?

   c. A file system with 300 GB disk uses a file descriptor with 8 direct block addresses, 1 indirect block address and 1 doubly indirect block address. The size of each disk block is 128 Bytes and the size of each disk block address is 8 Bytes. The maximum possible file size in this file system is ___

   d. Consider a typical disk that rotates at 15000 rotations per minute (RPM) and has a transfer rate of 50 × 106 bytes/sec. If the average seek time of the disk is twice the average rotational delay and the controller's transfer time is 10 times the disk transfer time, the average time (in milliseconds) to read or write a 512 byte sector of the disk is _____

   e. About RAID:

      RAID is the abbreviation of _____

      Which raid level provides maximum usable disk space? _____

      What is the minimum number of disks required for RAID1? _____

      From 0~6, which raid level doesn't use parity? _____

      If a disk fails in RAID level ___, rebuilding lost data is easiest. (From 0~6)

# 2019 HW2

2019年11月17日 19:31

1. a. Cache will check whether the block in which the data required by CPU is located is in the cache.
If yes, the cache will use the data directly.
Otherwise, cache will copy that block from memory, storing which in a specific way in cache, and then tell it back to CPU.
Due to the locality of time and memory, data accessing will be faster on average.
As far as single accessing concerned, if cache hits, it becomes far faster. But it becomes slower if cache misses.

b. $L_1$ cache will split them.
But $L_2$ cache won't.

c. 128 kB / 16 bytes = 8192 $\log_2 8192 = 13$
So set(index) field uses 13 bit
$\log_2 16 = 4$ So byte select occupies 4 bit
And tag uses $32 - 13 - 4 = 15$ bit
That is byte offset uses $2^{13} \times 4 \div 8 = 4096$ bytes = 4kB
index field uses $2^{13} \times 13 \div 8 = 13$ kB
tag uses $2^{13} \times 15 \div 8 = 15$kB

d. AMAT = $T_{hit} + \eta T_{miss}$, where $\eta$ is the miss rate.
-1
1) $T_{hit}$ ↓
2) $\eta$ ↓
3) $\eta$ ↓
4) $T_{miss}$ ↓ (targeting the $L_1$-cache) or $\eta$ ↓ (considering $L_1$-cache and $L_2$-cache as a whole)
5) $T_{miss}$ ↓ 降低hit time
6) $T_{miss}$ ↓

2. a. 16kB = $2^{14}$ bytes
virtual page number: 0x2AF37848D32BB
page index: 0x3567

b. 1) remain unchanged
2) decreased
3) decreased

c. page fault > TLB miss > cache miss > L2 hit

3. a. seek time

b. We need 42797 KB / 512 bytes = 85594 sectors

85594 = 1024 × 83 + 602 and <1000, 9, 40> ~ <1082, 9, 39> contains 1024 × 83 sectors

<1083, 9, 39> ~ <1283, 15, 63> contains 408 sectors, with 194 sectors left

So the last sector is <1284, 3, 1>

c. One block may hold 128/8 = 16 block address.

So we can have at most 8 + 1·16 + 1·16·16 = 280 blocks.

And the file size: 280 × 128 ÷ 1024 = 35 KB

d. rotation delay: $\frac{60 \times 1000}{15000} = 4ms$

average rotation delay: 2ms seek time: 4ms

transfo. time $\frac{512}{50 \times 10^6} \times 1000 = 0.01024$ ms

controller's transfer time: $10 \times 0.01024 = 0.1024$ ms

average total: $2 + 4 + 0.1024 + 0.01024 = 6.11264$ ms

e. 1) Redundant Arrays of Inexpensive (Independent) Disks

2) raid 0

3) 2

4) raid 0 . raid 1

5) raid 1

# Problem Out-of-Order Execution

For this problem, Lu Jiaxin wants to schedule the following codes on an out-of-order core.

The processor is a single-issue core with integer register (x0, x1, . . . ) and float register (f0, f1, . . . ). The processor uses free ROB entries from low index to high index. Instructions can commit one cycle after writeback, and ROB entries can be reused one cycle after commit. Instructions that depend on others can issue one cycle after the instruction it depends on writes back. Loads and stores take two cycles each, floating-point multiplies take three cycles, and floating-point adds take five cycles.

All functional units are fully pipelined. All the reservation stations are large enough.

Fill out the table with the cycles at which instructions enter the ROB, issue to the functional units, write back to the ROB, and commit. Also fill out the new register names for each instruction. If the instruction producing a source register had already committed before the dependent instruction enters the ROB, use the architectural register name.

Remember that instructions must enter the ROB and commit in order. On each cycle, only one instruction can enter the ROB, one can issue, one can write back, and one can commit.

## 1 Question A

Code A :

```
fmul.d  f0, f0, f1
fadd.d  f2, f2, f0
```

The ROB has two entries. Use r0-r1 for the two ROB entries.

| | Time | | | | Instruction | | | |
|---|---|---|---|---|---|---|---|---|
| | Enter ROB | Issue | WB | Commit | OP | Dest | Src1 | Src2 |
| $I_1$ | -1 | 0 | 3 | 4 | FMUL.D | r0 | f0 | f1 |
| $I_2$ | | | | | FADD.D | | | |

## 2 Question B

Code B :

```
fmul.d  f0, f0, f1
fadd.d  f2, f2, f0
fadd.d  f2, f2, f0
```

The ROB has two entries. Use r0-r1 for the two ROB entries.

| | Time | | | | Instruction | | | |
|---|---|---|---|---|---|---|---|---|
| | Enter ROB | Issue | WB | Commit | OP | Dest | Src1 | Src2 |
| $I_1$ | -1 | 0 | 3 | 4 | FMUL.D | r0 | f0 | f1 |
| $I_2$ | | | | | FADD.D | | | |
| $I_3$ | | | | | FADD.D | | | |

## 3 Question C

Code C :

```
fld     f0, 0(x1)
fld     f1, 8(x1)
fmul.d  f0, f0, f1
fadd.d  f2, f2, f0
fld     f0, 16(x1)
fadd.d  f2, f2, f0
```

The ROB has four entries. Use r0-r3 for the four ROB entries.

| | Time | | | | Instruction | | | |
|---|---|---|---|---|---|---|---|---|
| | Enter ROB | Issue | WB | Commit | OP | Dest | Src1 | Src2 |
| $I_1$ | -1 | 0 | 2 | 3 | FLD | r0 | x1 | |
| $I_2$ | 0 | 1 | 3 | 4 | FLD | r1 | x1 | |
| $I_3$ | 1 | | | | FMUL.D | | | |
| $I_4$ | | | | | FADD.D | | | |
| $I_5$ | | | | | FLD | | | |
| $I_6$ | | | | | FADD.D | | | |

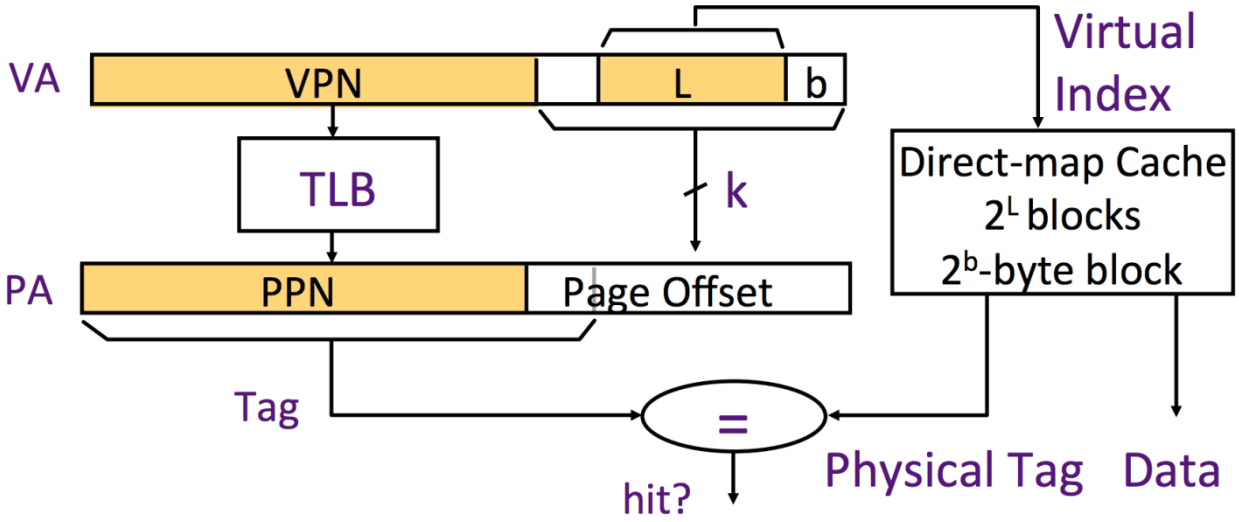
# Problem Virtual Memory & Aliasing Problem

## 1 Question A

Suppose Lu Jiaxin has a virtual memory system with 2048-byte pages. Assume that physical addresses are 64 bits, each page table entry takes up 64 bits, and each level of the page table takes up a single page total. How many levels of paging would Lu Jiaxin need in order to cover 32 GB of virtual memory?

## 2 Question B

What is the aliasing problem? Describe it in your words. Let's consider a feasible solution for the aliasing problem, Virtual Indexed & Physical Tagged Cache.




To avoid the aliasing problem, what condition should meet among L, k, b? Why?
(L is cache index. k is page offset. b is block offset)

## 3 Question C

You are asked to design a virtually indexed, physically tagged cache. A page is 4096 bytes. The cache must have 256 lines of 64 bytes each. What associativity must the cache have to not worry about aliases? (Hint : cache size is bigger than page size, so the cache should be set- associative)

# Problem Branch Prediction

For the following question, we are interested in the performance of branches when executing the following code. For each part, assume that N = 4 and the array A has the values {1, 7, 2, 5}.

| C code | RISC-V Assembly |
|---|---|
| `for (int i = 0; i < N; i++) {`<br>`  int b = A[i];`<br>`  if (b >= 5)`<br>`  c += b;`<br>`  if (b < 4)`<br>`  c -= b;`<br>`}` | `  li x1, A`<br>`  li x4, N`<br>`loop:`<br>`  lw x2, 0(x1)`<br>`  li x5, 5`<br>`  blt x2, x5, skip1`<br>`  add x3, x3, x2`<br>`skip1:`<br>`  li x5, 4`<br>`  bge x2, x5, skip2`<br>`  sub x3, x3, x2`<br>`skip2:`<br>`  addi x1, x1, 4`<br>`  addi x4, x4, -1`<br>`  bnez x4, loop` |

## 1 Question A

Fill out the table to show what the predictions will be if the branch predictor is a BHT indexed by PC with two-bit counters. If the most- significant bit of a counter is 1, the predictor predicts taken. Otherwise it predicts not taken. The counters are initialized to weakly not-taken (01). The Counter column in the table shows the state of the counter before the branch is executed. Assume that the BHT is large enough that no aliasing of instruction addresses will occur.

| | Instruction | Counter | Prediction | Actual |
|---|---|---|---|---|
| i = 0 | blt x2 x5, skip 1 | 01 | not taken | taken |
| | bge x2, x5, skip2 | 01 | not taken | not taken |
| | bnez x4, loop | 01 | not taken | taken |
| i = 1 | blt x2 x5, skip 1 | | | |
| | bge x2, x5, skip2 | | | |
| | bnez x4, loop | | | |
| i = 2 | blt x2 x5, skip 1 | | | |
| | bge x2, x5, skip2 | | | |
| | bnez x4, loop | | | |
| i = 3 | blt x2 x5, skip 1 | | | |
| | bge x2, x5, skip2 | | | |
| | bnez x4, loop | | | |

What is the prediction accuracy for each branch? What is the prediction accuracy overall?

| Branch | Accuracy |
|---|---|
| blt | |
| bge | |
| bnez | |
| overall | |

## 2 Question B

Now assume we change the branch predictor to a BHT indexed by PC and a single bit of global history. Assume the global history is initialized to 0, the counters are initialized to 01 (weakly not-taken), and there is no aliasing. The Counter columns in the table show the state of the counters before the branch is executed. Fill out the table with the predictions.

| | Instruction | Global History | Counter 0 | Counter 1 | Prediction | Actual |
|---|---|---|---|---|---|---|
| i = 0 | blt x2 x5, skip 1 | 0 | 01 | 01 | not taken | taken |
| | bge x2, x5, skip2 | | | | | |
| | bnez x4, loop | | | | | |
| i = 1 | blt x2 x5, skip 1 | | | | | |
| | bge x2, x5, skip2 | | | | | |
| | bnez x4, loop | | | | | |
| i = 2 | blt x2 x5, skip 1 | | | | | |
| | bge x2, x5, skip2 | | | | | |
| | bnez x4, loop | | | | | |
| i = 3 | blt x2 x5, skip 1 | | | | | |
| | bge x2, x5, skip2 | | | | | |
| | bnez x4, loop | | | | | |

What is the prediction accuracy for each branch? What is the prediction accuracy overall?

| Branch | Accuracy |
|---|---|
| blt | |
| bge | |
| bnez | |
| overall | |

## 3 Question C

If you run this code with a large array containing uniformly randomly distributed values, which branch do you expect to get the most benefit from global history and why?

## 4 Question D

Explain the motivation for using both BHT and BTB branch-prediction structures in the same implementation.

# 2019 HW3

2019年12月18日 19:21

100-0+1=101->100（此处没有溢出分）

1. A

| | ROB | Issue | WB | Commit | OP | Dest | Src1 | Src2 |
|---|---|---|---|---|---|---|---|---|
| $I_1$ | -1 | 0 | 3 | 4 | FMUL.D | r0 | f0 | f1 |
| $I_2$ | 0 | 4 | 9 | 10 | FADD.D | r1 | f2 | r0 |

B

| | ROB | Issue | WB | Commit | OP | Dest | Src1 | Src2 |
|---|---|---|---|---|---|---|---|---|
| $I_1$ | -1 | 0 | 3 | 4 | FMUL.D | r0 | f0 | f1 |
| $I_2$ | 0 | 4 | 9 | 10 | FADD.D | r1 | f2 | r0 |
| $I_3$ | 5 | 10 | 15 | 16 | FADD.D | r0 | r1 | f0 |

C

| | ROB | Issue | WB | Commit | OP | Dest | Src1 | Src2 |
|---|---|---|---|---|---|---|---|---|
| $I_1$ | -1 | 0 | 2 | 3 | FLD | r0 | x1 | |
| $I_2$ | 0 | 1 | 3 | 4 | FLD | r1 | x1 | |
| $I_3$ | 1 | 4 | 7 | 8 | FMUL.D | r2 | r0 | r1 |
| $I_4$ | 2 | 8 | 13 | 14 | FADD.D | r3 | f2 | r2 |
| $I_5$ | 4 | 5 | 8 | 15 | FLD | r0 | x1 | |
| $I_6$ | 5 | 14 | 19 | 20 | FADD.D | r1 | r3 | r0 |

✓

2. A 32GB = $2^{35}$ bit 2048-byte = $2^{14}$ bit 64 bits = $2^6$ bit A single page may include $2^{14}/2^6 = 2^8$ table entry

First level: $2^{35}/2^{11} = 2^{24}$ pages Second level: $2^{24}/2^8 = 2^{16}$ pages fitting in a single page in third level

So we need 3 levels of pages

✓

B Aliasing: Several virtual addresses are mapped to the same physical address.

Thus, several different entries pointing to same physical data may exist simultaneously in cache

As va[k-1:0] = pa[k-1:0] is ensured, we should have $L+b \le k$

Otherwise, va[L+b-1:k] = pa[L+b-1:k] should be additionally ensured by hardware.

写的很清晰 +1

C 256 × 64 ÷ 2048 = 4 So at least 4-way SA is needed.

3. A

| Inst | Counter | Pred | Actual |
|---|---|---|---|
| blt | 01 | not taken | taken |
| bge | 01 | not taken | not taken |
| bnez | 01 | not taken | taken |
| blt | 10 | taken | not taken |
| bge | 00 | not taken | taken |
| bnez | 10 | taken | taken |
| blt | 01 | not taken | taken |
| bge | 01 | not taken | not taken |
| bnez | 11 | taken | taken |
| blt | 10 | taken | not taken |
| bge | 00 | not taken | taken |
| bnez | 11 | taken | not taken |

| Branch | Accuracy |
|---|---|
| blt | 0 |
| bge | 1/2 |
| bnez | 1/2 |
| overall | 1/3 |

✓

B

| Inst | Global | Counter 0 | Counter 1 | Prediction | Actual |
|---|---|---|---|---|---|
| blt | 0 | 01 | 01 | not taken | taken |
| bge | 1 | 01 | 01 | not taken | not taken |
| bnez | 0 | 01 | 01 | not taken | taken |
| blt | 1 | 10 | 01 | not taken | not taken |
| bge | 0 | 01 | 00 | not taken | taken |
| bnez | 1 | 10 | 01 | not taken | taken |
| blt | 1 | 10 | 00 | not taken | taken |
| bge | 1 | 10 | 00 | not taken | not-taken |
| bnez | 0 | 10 | 10 | taken | taken |
| blt | 1 | 10 | 01 | not taken | not-taken |
| bge | 0 | 10 | 00 | taken | taken |
| bnez | 0 | 11 | 10 | taken | not taken |

| Branch | Accuracy |
|---|---|
| blt | 1/2 |
| bge | 3/4 |
| bnez | 1/4 |
| overall | 1/2 |

C b<4 is strongly associated with last branch b>5, so bge benefits most.

D The number of entries in BTB is limited. BHT may help predict if pc misses in BTB.