Quiz Week 5 (optional): Branch prediction and ILP

ⓘ This is a preview of the published version of the quiz

Started: Feb 6 at 12:31pm

Quiz Instructions

This quiz covers branch prediction and ILP.

See [https://jlpteaching.github.io/comparch/modules/processor architecture/ilp/ (https://jlpteaching.github.io/comparch/modules/processor%20architecture/ilp/)](https://jlpteaching.github.io/comparch/modules/processor%20architecture/ilp/)

Question 1 — 1 pts

For an *always not taken* predictor, what is the accuracy for the given branch outcomes?

T | T | NT | NT | T | NT | T | T | T | T | T | NT | T | T | T | T | NT | T | T | NT

Question 2 — 1 pts

For an *always taken* predictor, what is the accuracy for the given branch outcomes?

NT | T | NT | NT | T | T | NT | NT | NT | T | T | NT | NT | T | T | T | NT | T | T | NT

Question 3 — 1 pts

For a *1-bit counter* (i.e., last outcome) predictor, what is the accuracy for the given branch outcomes? The initial prediction is 'taken'.

prediction T T T T NT T T T
outcome T | T | T | NT | T | T | T | NT | T | NT | T | T | T | T | T | T | T | T | NT | NT
✓ ✓ ✓ ✗ ✗ ✓ ✓ ✗

→ make a prediction first, then change the predictor state when the outcome is known

Question 4

1 pts

For a *2-bit saturating counter* predictor, what is the accuracy for the given branch outcomes? The initial prediction is 'weakly taken'.

prediction | T | T | T | T | T | T | T | T | T | T | NT | T | T | T | NT | ...
T | T | T | T | T | T | T | NT | NT | T | T | NT | NT | T | T | T | NT | NT | NT | NT
state | WT | ST | ST | ST | ST | ST | ST | WT | WN | WT | ST | WT | WN | WT | ...

WT → weakly taken
ST → strongly taken
WN → weakly not taken
SN → strongly not taken

Question 5

1 pts

Mark the true statements.

- [ ] Compilers can re-arrange instructions to increase performance
- [ ] The order of instructions in the program can affect the performance
- [ ] Sometimes there are independent instructions that the compiler cannot find
- [ ] Compilers can re-arrange instructions to reduce hazards and stalls

Question 6

2 pts

For the following program, what is the "best" schedule without changing the program? Assume the DINO CPU pipeline without any branch prediction. I.e., there is a one cycle load to use hazard and branches are resolved in the memory stage.

a: `lw s8, (792)s0`

b: `sub s8, s8, s11`

c: `add s11, s7, a6`

d: `addi t2, a6, -750`

e: `beq s11, t2, 1681`

f: `xori s8, s8, 1141`

- b,d,c,a,e,f
- a,e,d,b,c,f
- a,c,b,d,e,f
- ✓ a,d,b,c,e,f
- a,b,c,d,e,f

→ can not reorder c and b due to WAR

→ cannot reorder e because it's a branch

→ should reorder b because a & b cause load-to-use hazard

Question 7

1 pts

Mark the true statements.

No unrolling

```
for(i=0; i<N; i++)
  a[i] = 3;
```

Loop unrolling

```
for (i=0; i<N; i+=4)
{ a[i] = 3;
  a[i+1] = 3;
  a[i+2] = 3;
  a[i+3] = 3
}
```

↳ more static inst

- ✓ When executing code with loops unrolled, the total number of dynamic instructions is about the same
- ✓ Loop unrolling reduces the number of dynamic branch instructions
- ☐ Loop unrolling makes the code have fewer static instructions
- ✓ Loop unrolling can expose instruction level parallelism
- ☐ Loop unrolling always increases performance ✗
- ✓ Loop unrolling will increase the memory footprint of the code

* 1 iteration of a loop cause 1 branch inst to be executed

→ fewer iterations, fewer branches executed

Question 8

1 pts

Compiler transformations like loop unrolling can **improve** the performance of applications by having which effect(s) on the Iron Law?

- [ ] Increase the CPI
- [ ] Increase the number of dynamic instructions
- [ ] Increase the cycle time
- [x] Reduce the CPI
- [ ] Increase the number of static instructions
- [ ] Reduce the number of static instructions
- [x] Reduce the number of dynamic instructions
- [ ] Reduce the cycle time

Question 9

2 pts

Which two instructions have a *write-after-write* dependence that would require using a new temporary register?

a: `sub s11, s11, t1`

b: `lw t0, 156(s11)`

c: `bne t1, a6, -634`

d: `xori a5, a6, 1623`

e: `sw t0, 1200(s11)`

f: `ori t0, a6, -70`

- ( ) e & f
- ( ) a & e
- ( ) b & e

- a & b
- b & c
- c & d
- b & f

Question 10

2 pts

Assume you have a processor design which is 2-wide in-order. In other words, you can fetch up to two instructions, decode up to two instructions, execute up to two instructions, send up to two instructions to memory, and write back up to two instructions each cycle. Assume that you cannot forward/bypass *in the same cycle* and have to stall any dependent instructions by at least 1 cycle.

What is the average cycles per instruction? (Ignore the warm up time. Ignore the cycles before the first instruction completes.)

inst α : slti a1, a1, ...
inst β : sw a1, s9 ...

```
sra s9, s7, s3
sll a1, s7, s3
sll s1, s9, a1
xori s2, s7, -124
lw s5 1728(s9)
sub a1, s1, s2
slti a1, a1, 105
sw a1 -620(s9)
add s9, a1, s7
add s3, s5, s9
```

cycle 1
cycle 2
cycle 3
cycle 4 } RAW
cycle 5 } WAR
cycle 6
cycle 7 } RAW

α → slti a1, a1, 105
β → sw a1 -620(s9)

7/10

Question 11

1 pts

Which of the following instruction pairs can you *not* execute in the same cycle?

- ○ `addi x1, x2, 0` and `subi x4, x8, 8`
- ○ `addi x1, x2, 0` and `subi x4, x8, 0`
- ○ `lw x1, 0(x2)` and `sw x4, 8(x2)`
- ○ `lw x1, 0(x2)` and `lw x4, 0(x8)`
- ✓ ○ `lw x1, 0(x2)` and `sw x4, 0(x8)`
- ○ `lw x1, 0(x2)` and `lw x4, 8(x8)`

they can be in the Execute stage at the same time but they must not be issued in the same cycle.

Question 12

1 pts

Mark all that are true.

- ☐ Dynamic ILP techniques implemented in hardware uses less power and area than static techniques implemented in the compiler
- ☐ Increasing the window of instructions can increase the ILP, but it also increases the complexity, power, and area
- ☐ Dynamic methods for finding ILP are more flexible to runtime dependencies (e.g., address dependencies)
- ☐ VLIW ISAs are better than RISC ISAs when implementing hardware for dynamic ILP

Question 13

1 pts

In out-of-order processor you can only execute instructions out of order, you still must issue instructions in order and complete (or commit) instructions in order. Why?

- ☐ Must commit instructions in order to determine their dependencies.

- [ ] Must commit instructions in order to make sure that exceptions/interrupts happen precisely for the right instruction.
- [ ] Must issue in order to make sure that exceptions/interrupts happen precisely for the right instruction.
- [ ] Must issue in order to determine their dependencies.

Question 14

2 pts

Assume you have the following 4 instructions that are decoded and waiting to execute. Assume the machine has 8 registers like the example in lecture.

i1: source regs: 7, 1. Destination reg: 0

i2: source regs: 3, 0. Destination reg: 1

i3: source regs: 5, 3. Destination reg: 2

i4: source regs: 5, 6. Destination reg: 6

Registers 7 and 4 are currently busy

Because of which rules can i2 *not* be executed? (It may help to draw out the matrices)

- [ ] (i) The required busses and functional units are available.
- [ ] (iii) The destination register is used as a source for a prior instruction
- [ ] (iv) The source or destination register will be written by a prior instruction
- [ ] (ii) The registers are busy.

Question 15

2 pts

Assume the following hardware state. There are some instructions currently executing, and others that have been decoded and are waiting to execute. Use this information to answer the questions below.

Currently executing instructions

See @ 194 on Piazza

```
add t6, s8, s1
```

`lw t0, -1060(a3)` (2) add t6, s8, s1 (1)

Instructions waiting to execute

`sub s8, a4, zero` (3)

`sw a5, -1096(s1)` (4)

`ori a4, t6, 144` (5)

`sra a5, s1, a3` (6)

(1) (3): WAR

(2)(4): unknown effective address of (4)

Which instructions can be sent to execute at this time (assume there are enough execution/functional units and busses)?

(1)(5): RAW

(4)(6): WAR

- [ ] sub s8, a4, zero
- [ ] sra a5, s1, a3
- [ ] sw a5, -1096(s1)
- [ ] ori a4, t6, 144

(1)(3) RAW in O3!

Question 16

2 pts

Assume the following hardware state. There are some instructions currently executing, and others that have been decoded and are waiting to execute. Use this information to answer the questions below.

Currently executing instructions

WAR / WAW hazards

`sub s9, t3, a1` (1)

`addi s0, t6, 1164` (2)

(1)(3) WAR

Instructions waiting to execute

(4)(5) RAW

`sub t3, t1, zero` (3)

`sra t5, a1, t5` (4)

`xori t1, t5, -1066` (5)

`xor t5, a1, t6` (6)

With register renaming Which instructions can be sent to execute at this time (assume there are enough execution/functional units and busses)?

- [ ] sub t3, t1, zero
- [ ] xori t1, t5, -1066
- [ ] xor t5, a1, t6
- [ ] sra t5, a1, t5

Question 17

2 pts

Mark all of the types of hazards that can occur in an out-of-order superscalar processor design.

- [x] Write after read
- [ ] Rename
- [x] Control
- [x] Write after write
- [x] Structural → limitation of resources, e.g.
- [x] Read after write
- [ ] Read after read

when the hazard can be solved using reg renaming but the cpu doesn't have enough physical regs

Quiz saved at 4:47pm

Submit Quiz