Discussion 5

Feb. 6th

Outline

- Midterm clarification:
  - You won't be asked to implement the register renaming and out of order execution but it will cover your understanding on register renaming and out of order execution.
- A summary on ILP
- Example on dependency and register renaming
- Week 5 quiz
  - Q 10 -> why CPI is related to throughput
- (if we have time) Example on static scheduling, loop unrolling, and multi-instruction pipelined

Static ILP and its limitations

- Done by the compiler on static code
- Dependent on the application
- Static scheduling
- Loop unrolling
- VLSI ISA

Limitation

- Can not detect dynamic dependencies
- Require complex compilers
- Larger code blocks
- Can’t deal with unpredictable delays
- …

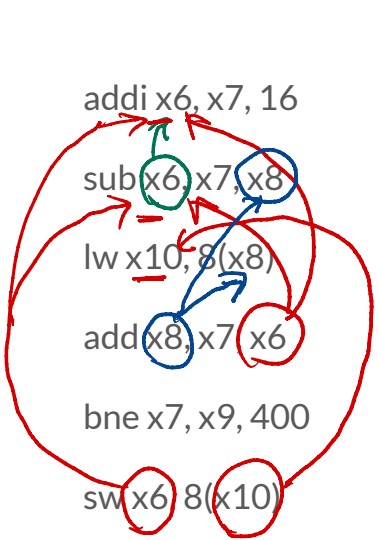
Dynamic Scheduling

- Hardware rearranges the instruction execution to reduce stall while maintaining data flow and exception behavior (Out of order execution)
- It can handle cases when dependences are unknown at compile time
- It can create parallel execution windows
- Tomasulo's algorithm is a great example of how it can be implemented
- But now we have hazard to deal with
    - The required busses and functional units are available (structural hazard)
    - RAW dependency (true dependency)
    - WAW dependency (false dependency)
    - WAR dependency (false dependency)

RAW ← true dependency

WAW ← false dependencies.
WAR ←

Solving WAW/WAR hazard with register renaming



hardware register: P1, P2, P3
x6 x8

```
addi x6, x7, 16
sub  P1  x7, x8
lw   x10, 8(x8)
add  P2, x7, P1
bne  x7, x9, 400
sw   P1, 8(10)
addi x6, x1, 10
```

Week 5 quiz

Question 1

1 pts

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

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

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'.

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

Question 5

1 pts

Mark the true statements.

- [ ] The order of instructions in the program can affect the performance
- [ ] Compilers can re-arrange instructions to increase 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 s1, (1532)s0`

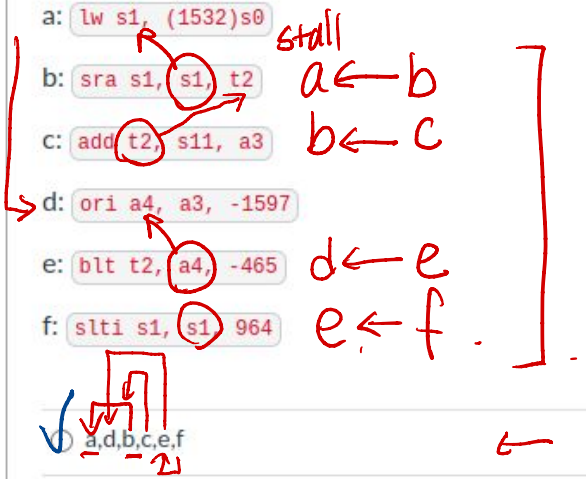
b: `sra s1, s1, t2`

c: `add t2, s11, a3`

d: `ori a4, a3, -1597`

e: `blt t2, a4, -465`

f: `slti s1, s1, 964`



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

Question 7

1 pts

Mark the true statements.

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

Question 8

1 pts

reduce dependencies, create more code windows,

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

inst x CPI x cycle time . .

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

for loop → 10 times.

loop: bne] x10

⇒ loop x10

bne

Question 9

2 pts

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

a: `add t5, t5, t4`

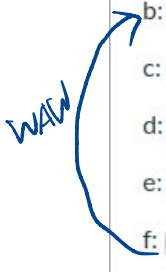
b: `lw s1, 236(t5)`

c: `bge t4, t3, -1265`

d: `ori a7, t3, -1100`

e: `sw s1, 732(t5)`

f: `xori s1, t3, -646`



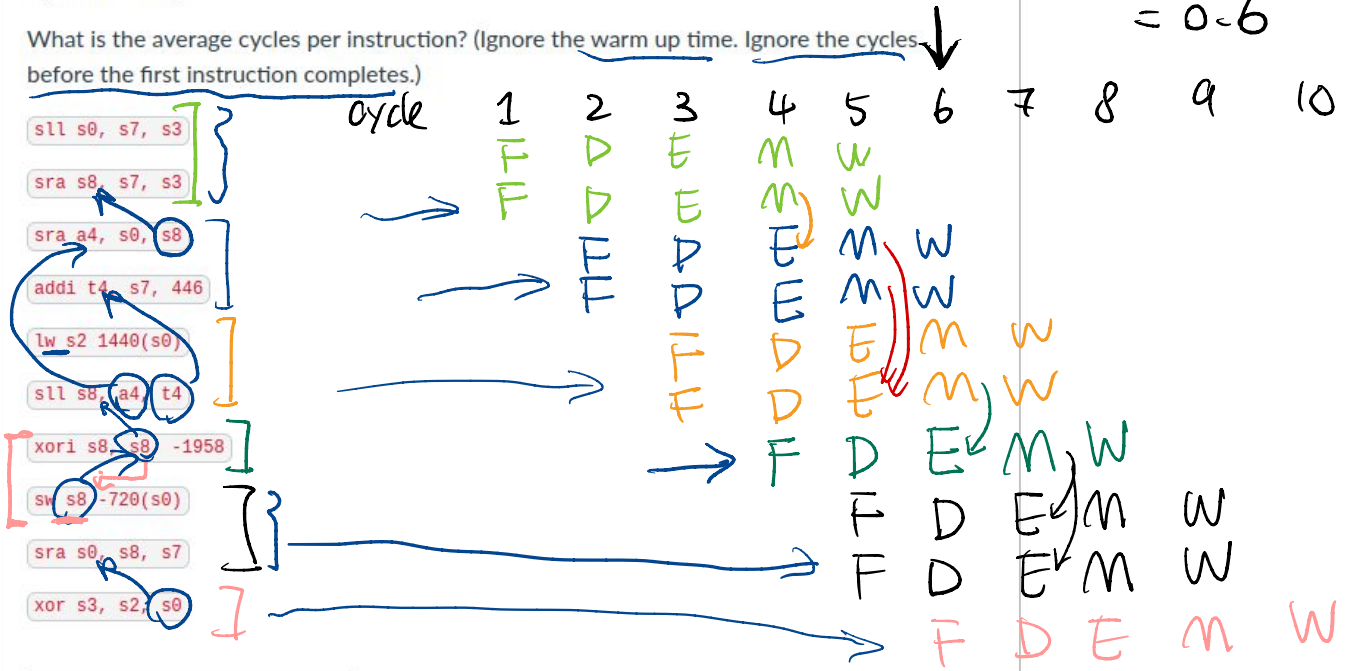
- ○ b & c
- ○ a & b
- ○ b & f
- ○ a & e
- ○ b & e
- ○ c & d
- ○ e & f

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.)



.6

CPI = IPC.

= inst/cycle.

6 cycles.

10 insts.

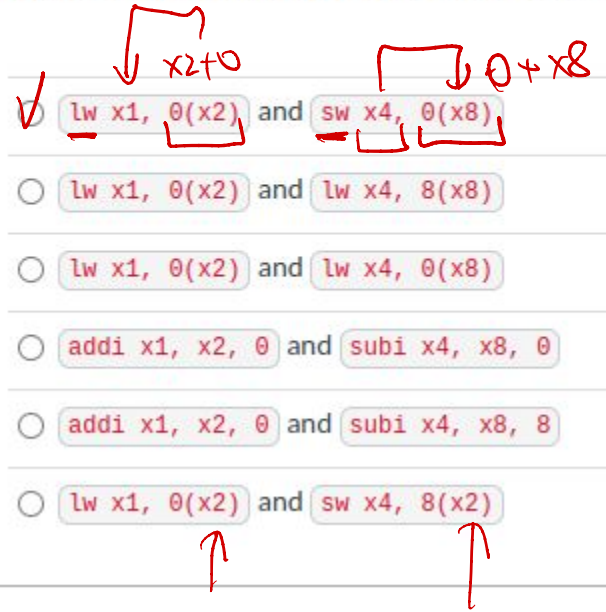
= 0.6

Question 11

1 pts

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

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



Question 12

1 pts

Mark all that are true.

- [ ] Dynamic methods for finding ILP are more flexible to runtime dependencies (e.g., address dependencies)
- [ ] Increasing the window of instructions can increase the ILP, but it also increases the complexity, power, and area
- [ ] Dynamic ILP techniques implemented in hardware uses less power and area than static techniques implemented in the compiler
- [ ] 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 issue in order to make sure that exceptions/interrupts happen precisely for the right instruction.
- [ ] Must issue in order to determine their dependencies.
- [ ] Must commit instructions in order to make sure that exceptions/interrupts happen precisely for the right instruction.
- [ ] Must commit instructions 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: 1, 3. Destination reg: 2

i2: source regs: 4, 2. Destination reg: 3

i3: source regs: 0, 4. Destination reg: 7

i4: source regs: 0, 6. Destination reg: 6

Registers 1 and 5 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.
- [ ] (iv) The source or destination register will be written by a prior instruction
- [ ] (iii) The destination register is used as a source for a prior instruction
- [ ] (ii) The registers are busy.

Question 15

2 pts

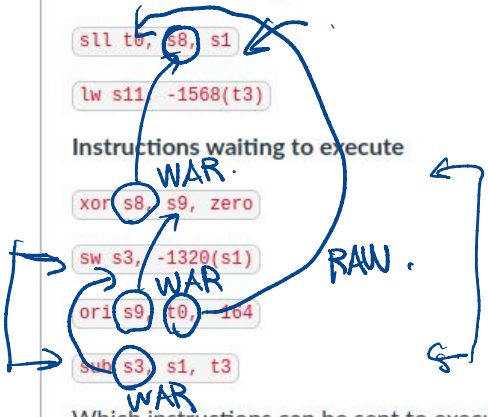
Assume the following hardware state. There are some instructions currently executing, and others that have been decoded and are waiting to execute. Use this information to answer the questions below.

Currently executing instructions

```
sll t0, s8, s1
lw s11, -1568(t3)
```

Instructions waiting to execute

```
xor s8, s9, zero
sw s3, -1320(s1)
ori s9, t0, -164
sub s3, s1, t3
```



F D E M W.

Oxor s8.

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

- [x] xor s8, s9, zero
- [x] sw s3, -1320(s1)
- [ ] ori s9, t0, -164
- [ ] sub s3, s1, t3

Question 16

2 pts

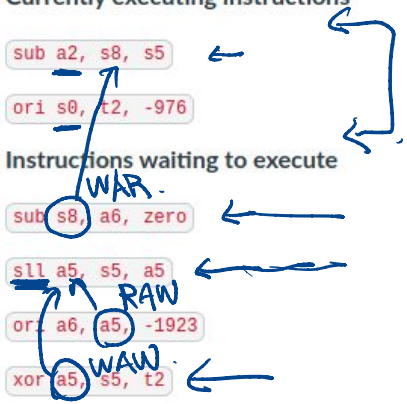
Assume the following hardware state. There are some instructions currently executing, and others that have been decoded and are waiting to execute. Use this information to answer the questions below.

Currently executing instructions

```
sub a2, s8, s5
ori s0, t2, -976
```

Instructions waiting to execute

```
sub s8, a6, zero
sll a5, s5, a5
ori a6, a5, -1923
xor a5, s5, t2
```



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

- [ ] ori a6, a5, -1923
- [x] xor a5, s5, t2
- [x] sub s8, a6, zero
- [x] sll a5, s5, a5

Question 17

2 pts

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

- [ ] Write after read
- [ ] Write after write
- [ ] Rename
- [ ] Read after write
- [ ] Read after read
- [ ] Control
- [ ] Structural

Example on Static Scheduling and Loop Unrolling

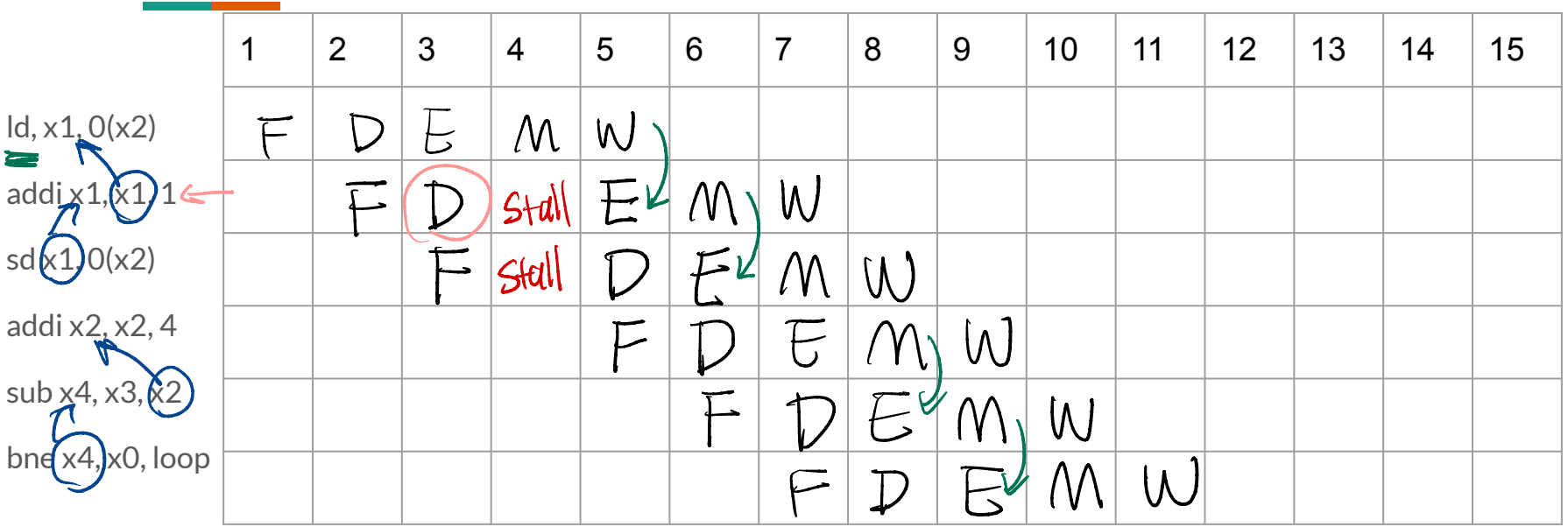
Example program:

```
void foo (size_t n, int x[], int y[]) {
        for (size_t i = 0; i < n; i ++) {
                y[ i ] = 10 * x[ i ] + y[ i ];
        }
}
```

```
loop:
  lw x2, 0(x1) // get x[ i ]
  muliw x2, x2, 10 // 10 * x[ i ]
  lw x4, 0(x3) // get y[ i ]
  addw x4, x4, x2 // 10 * x[ i ] + y[ i ]
  sw x4, 0(x3)
  addi x11, x11, 1 // i ++
  addi x1, x1, 4 // int is 4 bytes so addr of x + 4
  addi x3, x3, 4 // addr of y + 4
  bne x11, x10, loop // if i != n, then continue looping
```

```
loop:
    lw x2, 0(x1) // get x[ i ]
    muliw x2, x2, 10 // 10 * x[ i ]
    lw x4, 0(x3) // get y[ i ]
    addw x4, x4, x2 // 10 * x[ i ] + y[ i ]
    sw x4, 0(x3)
    addi x11, x11, 1 // i ++
    addi x1, x1, 4 // int is 4 bytes so addr of x + 4
    addi x3, x3, 4 // addr of y + 4
    bne x11, x10, loop // if i != n, then continue looping
```

F D E M W.


1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
ld, x1, 0(x2)
addi x1, x1, 1
sd x1, 0(x2)
addi x2, x2, 4
sub x4, x3, x2
bne x4, x0, loop
F D E M W
F D Stall E M W
F Stall D E M W
F D E M W
F D E M W
F D E M W