Errata

1. Pages 56 and 57 (last sentence that spans the two pages) (thanks to Craig Lorie, GMU):
   Change:
   > The frame pointer contains the first address on the stack that pertains to the activation record of the called procedure and never changes while this procedure is in execution.

   To:
   > The frame pointer contains the first address on the stack that pertains to the callee's portion of the activation record for the called procedure and never changes while this procedure is in execution.

2. Page 58 (first sentence on top of the page) (thanks to Craig Lorie, GMU:
   Change:
   > The frame pointer is a fixed harness on the stack (for a given procedure) and points to the first address of the activation record (AR) of the currently executing procedure."

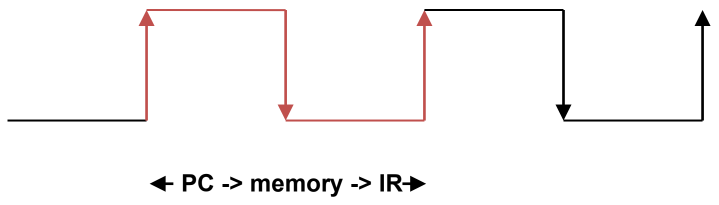
   To:
   > The frame pointer is a fixed harness on the stack (for a given procedure) and points to the first address that pertains to the callee's portion of the activation record (AR) for the currently executing procedure.

3. Page 62: decrease indent to align with the next para for the following sentence:
   > The DEC VAX 11 family and the Intel x86 family are examples of architectures with variable-length instructions. In VAX 11, the instructions can vary in size from 1 byte to 53 bytes.

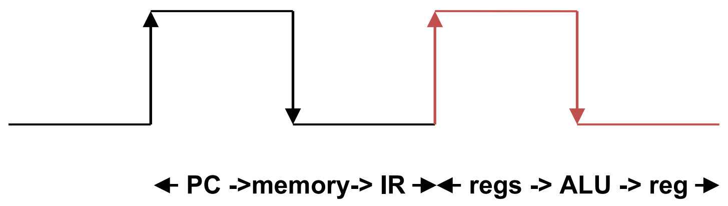
4. Page 63: Footnote 7
   Change:
   > The LC-2200 does not have a separate, unconditional jump instruction. However, it should be easy to see that we can realize such an instruction by using JALR R_{link}, $R_{dont\text{-}care}$; where R_{link} contains the address to jump to and $R_{dont\text{-}care}$ is a register whose current value you don't mind trashing.

   To:
   > The LC-2200 does not have a separate, unconditional jump instruction. However, it should be easy to see that we can realize such an instruction by using JALR R_{target}, $R_{dont\text{-}care}$; where R_{target} contains the address to jump to and $R_{dont\text{-}care}$ is a register whose current value you don't mind trashing.

5. Page 84: Figure 3.7: Align the legend under the clock to line up with beginning to end of first clock cycle



6. Page 84: Figure 3.8: Align the legend under the clock to line up properly in the first two clock cycles



7. Page 168: (thanks to TAs Alex and Sanjana)
The worked out example in the textbook (Example 5.5 on page 168 in Chapter 5) has an error. Here is the correct solution.

Example 5.5 in the textbook

Instruction	CPI
Add	2
Shift	3
Others	2
Add/Shift	4

If the sequence ADD followed by SHIFT appears in 20% of the dynamic frequency of a program, what is the speedup of the program with all {ADD, SHIFT} replaced by the new instruction?

[Hint: a) For every 10 instructions in the original program, 2 instructions are the ADD/SHIFT combo. b) The number of instructions in the new program shrinks to 90% of the original program.]

Solution:
Let N be the number of instructions in the original program. Then, the execution time of the original program.
= N*frequency of ADD*2 + N * frequency of SHIFT *3 + N*frequency of others*2
= N*0.1*2+N*0.1*3+N*0.8*2 = 2.1N

With the combo instruction replacing {ADD), SHIFT} the number of instructions in the new program shrinks to 0.9 N in the new program. The frequency of the combo instruction is 1/ 9 and the other instructions are 8/ 9.
The execution time of the new program is
= (0.9 N) * frequency of combo * 4 + (0.9 N) * frequency of others * 2
= (0.9 N) * (1/ 9) * 4 + (0.9 'N) * (8/9) * 2
= 2N

Speedup of the program = old execution time/new execution time
= 2.1 N/ 2N
= 1.05

8. Page 181: (thanks to Craig Lorie, GMU) (consider explaining this datapath some more in the book)
Figure 5.6(a) need fixing
Ex stage: The second ALU (top) computing the branch target address (PC+offset) should feedback directly to the MUX in the IF stage (not through the pipeline register of EX stage). Also, AND the output of the zero detect combination logic (that computes if the output of the first ALU is zero) with the instruction in the EX stage being a branch instruction before feeding it back to the MUX in the IF stage as a control input.

9. Page 183:
   Change:

   MEM stage (cycle 4):
   DBUF → MBUF; // The MEM stage has nothing to contribute toward the execution of the ADD instruction; so simply copy the DBUF to MBUF.

   To:

   MEM stage (cycle 4):
   EBUF → MBUF; // The MEM stage has nothing to contribute toward the execution of the ADD instruction; so simply copy the EBUF to MBUF.

10. Page 184: (Example 5.9)
    Change:

A (needed for R-type)	32 bits
B (needed for R-type)	32 bits
Offset (needed for I-type and J-type)	20 bits
PC value (needed for BEQ)	32 bits
Rx specifier (needed for R-, I-, and J-type)	4 bits

    To:

A (Ry contents or Rx contents depending on Instr.)	32 bits
B (Rz contents or Ry contents depending on Instr.)	32 bits
Offset/Immediate Value (needed for I-type)	20 bits
PC value (needed for BEQ and JALR)	32 bits
Dest. Reg. Specifier (Rx or Ry depending on Instr.)	4 bits

    Change:
    In the Figure at the bottom of the page,
    Rx
    To:
    Dest. Reg. Specifier

11. Page 189:
    Change:
    S1 modifies a resource read by S1.

    To:
    S1 modifies a resource read by S2.

12. Page 195: on RP bit
    Change the verbiage. Instead of RP bit, call it RP signal. This signal is generated for each register in the Register file by the ID/RR stage if the instruction in this stage finds the B bit set for

a particular register. Each of the subsequent stages (EX, MEM, WB) "forward" the register value (if they have it) if the associated RP signal is asserted by the ID/RR stage.

13. Page 239:
    Change:
        Depending on his interests, perhaps he…
    To:
        Depending on her interests, perhaps she….

14. Page 261:
    Second para:

    Change: "Figure 6.18 show"
    To: "Figure 6.18 shows"

15. Page 262:
    Figure 6.20.
    Change: "Dispatcher"
    To: "Scheduler"

    Change: "I/O Completion Trap Handler"
    To: "I/O Completion Interrupt Handler"

16. Page 318
    Jay's comment:
    "On Page 318, section 8.1.1, you consider the case of a pipelined processor where I2 (in the MEM) stage triggers a page fault.

    You state that I1 (in WB) gets to finish, and I3-I5 get "squashed" and then later say that the PC of I2 will be saved to be "re-started"

    Isn't I2 essentially squashed as well? Or is this a carefully maintained semantic difference between squashing and re-starting?"

    Response:
    "In principle you are right that re-starting I2 implies that I2 is squashed as well. But I make a distinction in that there is no "state" of I3-I5 that is needed to be saved, but in the case of I2 we have to know the PC value corresponding to I2 to enable restarting.

    BTW, LC-2200 instruction is simple and the pipeline structure is a simple 5-stage pipeline, and so re-starting an instruction is the same as executing that instruction for the first time.

However, there are ISAs (google IBM byte string instructions) and pipelined implementation of them in which the two need not be the same. In other words, there may be some state that is being saved for the instruction that is experiencing a page fault to re-start it where it left off."

17. Page 321:
    Example 8.1
    Change:

    > *before–after* pictures of the *page tables* for P1 and P2 and the *frame table* that resulted from the handling of the page fault.

    To:

    > *before–after* pictures of the *page tables* for P1 and P4 and the *frame table* that resulted from the handling of the page fault.

18. Page 370: (thanks to Craig Lorie of GMU)
    Second last para
    Change:

    > Step 1: On every write (store instruction in the LC-2200), the CPU simply writes to the cache. There is no need to check the valid bit or the cache tag. The cache updates the tag field of the corresponding entry and sets the valid bit. These actions happen in the MEM stage of the pipeline.

    To:

    > Step 1: Let us assume that the memory location is already present in the cache (i.e., a write hit). On every write (store instruction in the LC-2200), the CPU simply writes to the cache. The cache updates the tag field of the corresponding entry and sets the valid bit. These actions happen in the MEM stage of the pipeline.

19. Page 373:
    First para
    Change:

    > Thus, there are write stalls incurred because the missing memory block need not be brought from the memory.

    To:

    > Thus, the write does not have to wait for the memory block to be brought into the cache from the memory. Write stalls incurred (if any) will only be due to the write buffer being full at the time of the write.

(Changes 20 through 36 thanks to Alex Daglis of Georgia Tech!)

20. Page 390:
    First line
    Change
        cache lines (or sets)
    To
        cache rows (called *sets*)

21. Page 390:
    In the entire page starting from the second line
    Change
        All occurrences of the string “cache line”
    To
        set

22. Page 390:
    Second line
    Delete footnote 4:

23. Page 390
    Line 15
    Change
        number or cache lines
    To
        number of sets

24. Page 391:
    Line 8
    Change
        cache lines
    To
        sets

25. Page 391:
    Line 23
    Change
        line
    To
        set

26. Page 391:
    Line 24
    Change
        cache lines
    To
        sets

27. Page 391:
    Line 24
    Change
        bits/cache line
    To
        bits/set

28. Page 391:
    Line 24
    Change
        cache lines
    To
        sets

29. Page 392:
    Line 20
    Change
        cache line
    To
        set

30. Page 392:
    Line 21
    Change
        cache lines
    To
        sets

31. Page 392:
    Line 22
    Change
        bits/cache line
    To
        bits/set

32. Page 392:
    Line 22
    Change
        cache lines
    To
        sets

33. Page 394:
    Line 21
    Delete
        (or cache line)

34. Page 396:
    Line 6
    Change
        lines
    To
        sets

35. Page 417
    Line 16
    Change
        Number of cache lines (L)
    To
        Number of sets (L)

36. Page 417
    Line 25-26
    Change
        cache line
    To
        set

37. Page 558:
    Second last line on the page:
    Change
        thread_cond_signal(res_not_buys);
    To
        thread_cond_signal(res_not_busy);

38. Page 645:

In Figure 13.15, as soon as an ACK for the first red packet in the active window is received, the active window moves one step to the right (the first white packet becomes a blue packet). Since the active window slides over the space of sequence numbers (from left to right) as time progresses, we refer to this as a *sliding window protocol*. Whereas this discussion presents sequence numbers as monotonically increasing, as a practical matter, the space of sequence numbers is circular and wraps down to 0 in any real implementation.

Change colors (red, blue) to gray scale in the above text

39. Page 647:

    packets (red ones in Figure 13.15) for which acknowledgements have not yet been received.

    Change red to gray scale in the above text

40. Page 443: Example 10.1

    Include

    - 2 surfaces per platter

41. Page 625:

    Last Para:

    Change:

    Today, you connect to your computer….

    To (delete "to"):

    Today, you connect your computer…"

42. Page 655: Table 13.3 incorrect.

    Correct table below: (Route for F changed in row 2, 4, 5)

Iteration Count	New node to which least-cost route known	B Cost/route	C Cost/route	D Cost/route	E Cost/route	F Cost/route
Init	A	2/AB	1/AC	4/AD	5/AE	∝
1	AC	2AB	**1/AC√**	**3/ACD**	**4/ACE**	**6/ACF**
2	ACB	**2/AB√**	√	3/ACD	**3/ABE**	**4/ABEF**
3	ACBD	√	√	**3/ACD√**	3/ABE	4/ABEF
4	ACBDE	√	√	√	**3ABE√**	4/ABEF
5	ACBDEF	√	√	√	√	**4/ABEF√**

43. Page 656: Table 13.4 incorrect (Much thanks to CS 2200 Spring 2022 student Archishmaan Peyyety, and Professor Alex Daglis for catching the typos in this table)
Correct table below: Entries for Destinations
(1) A through B changed; (2) B through F changed; (3) F through B changed; (4) F through C changed

	Cost through immediate neighbors			
Destination	A	B	C	F
A	5(EA)	3(EBA)	4(ECA)	5(EFDCA)
B	7(EAB)	1(EB)	5(ECB)	3(EFEB)
C	6(EAC)	3(EBC)	3(EC)	4(EFDC)
D	8(EACD)	4(EBEFD)	5(ECD)	2(EFD)
F	9(EABEF)	3(EBEF)	6(ECDF)	1(EF)

44. Page 657: Typo.
Third para:
Change
“converge”
To
“convergence”