




## Historical Perspective and Further Reading

*An active field of science is like an immense anthill; the individual almost vanishes into the mass of minds tumbling over each other, carrying information from place to place, passing it around at the speed of light.*

Lewis Thomas, “Natural Science,” in *The Lives of a Cell*, 1974

For each chapter in the text, a section devoted to a historical perspective can be found online on a site that accompanies this book. We may trace the development of an idea through a series of computers or describe some important projects, and we provide references in case you are interested in probing further.

The historical perspective for this chapter provides a background for some of the key ideas presented in this opening chapter. Its purpose is to give you the human story behind the technological advances and to place achievements in their historical context. By studying the past, you may be better able to understand the forces that will shape computing in the future. Each Historical Perspective section online ends with suggestions for further reading, which are also collected separately online under the section “[Further Reading](#).” The rest of  [Section 1.12](#) is found online.



## Exercises

The relative time ratings of exercises are shown in square brackets after each exercise number. On average, an exercise rated [10] will take you twice as long as one rated [5]. Sections of the text that should be read before attempting an exercise will be given in angled brackets; for example, <§1.4> means you should have read [Section 1.4](#), Under the Covers, to help you solve this exercise.

**1.1** [2] <§1.1> Aside from the smart cell phones used by a billion people, list and describe four other types of computers.

**1.2** [5] <§1.2> The eight great ideas in computer architecture are similar to ideas from other fields. Match the eight ideas from computer architecture, “Design for Moore’s Law,” “Use Abstraction to Simplify Design,” “Make the Common Case Fast,” “Performance via Parallelism,” “Performance via Pipelining,” “Performance via Prediction,” “Hierarchy of Memories,” and “Dependability via Redundancy” to the following ideas from other fields:

- a. Assembly lines in automobile manufacturing
- b. Suspension bridge cables
- c. Aircraft and marine navigation systems that incorporate wind information
- d. Express elevators in buildings

- e. Library reserve desk
- f. Increasing the gate area on a CMOS transistor to decrease its switching time
- g. Adding electromagnetic aircraft catapults (which are electrically powered as opposed to current steam-powered models), allowed by the increased power generation offered by the new reactor technology
- h. Building self-driving cars whose control systems partially rely on existing sensor systems already installed into the base vehicle, such as lane departure systems and smart cruise control systems

**1.3** [2] <§1.3> Describe the steps that transform a program written in a high-level language such as C into a representation that is directly executed by a computer processor.

**1.4** [2] <§1.4> Assume a color display using 8 bits for each of the primary colors (red, green, blue) per pixel and a frame size of  $1280 \times 1024$ .

- a. What is the minimum size in bytes of the frame buffer to store a frame?
- b. How long would it take, at a minimum, for the frame to be sent over a 100 Mbit/s network?

**1.5** [4] <§1.6> Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a 3 GHz clock rate and a CPI of 1.5. P2 has a 2.5 GHz clock rate and a CPI of 1.0. P3 has a 4.0 GHz clock rate and has a CPI of 2.2.

- a. Which processor has the highest performance expressed in instructions per second?
- b. If the processors each execute a program in 10 seconds, find the number of cycles and the number of instructions.
- c. We are trying to reduce the execution time by 30%, but this leads to an increase of 20% in the CPI. What clock rate should we have to get this time reduction?

**1.6** [20] <§1.6> Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (classes A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2.

Given a program with a dynamic instruction count of  $1.0E6$  instructions divided into classes as follows: 10% class A, 20% class B, 50% class C, and 20% class D, which is faster: P1 or P2?

- a. What is the global CPI for each implementation?
- b. Find the clock cycles required in both cases.

**1.7** [15] <§1.6> Compilers can have a profound impact on the performance of an application. Assume that for a program, compiler A results in a dynamic instruction count of  $1.0E9$  and has an execution time of  $1.1$  s, while compiler B results in a dynamic instruction count of  $1.2E9$  and an execution time of  $1.5$  s.

- a. Find the average CPI for each program given that the processor has a clock cycle time of  $1$  ns.
- b. Assume the compiled programs run on two different processors. If the execution times on the two processors are the same, how much faster is the clock of the processor running compiler A's code versus the clock of the processor running compiler B's code?
- c. A new compiler is developed that uses only  $6.0E8$  instructions and has an average CPI of  $1.1$ . What is the speedup of using this new compiler versus using compiler A or B on the original processor?

**1.8** The Pentium 4 Prescott processor, released in 2004, had a clock rate of  $3.6$  GHz and voltage of  $1.25$  V. Assume that, on average, it consumed  $10$  W of static power and  $90$  W of dynamic power.

The Core i5 Ivy Bridge, released in 2012, has a clock rate of  $3.4$  GHz and voltage of  $0.9$  V. Assume that, on average, it consumed  $30$  W of static power and  $40$  W of dynamic power.

**1.8.1** [5] <§1.7> For each processor find the average capacitive loads.

**1.8.2** [5] <§1.7> Find the percentage of the total dissipated power comprised by static power and the ratio of static power to dynamic power for each technology.

**1.8.3** [15] <§1.7> If the total dissipated power is to be reduced by  $10\%$ , how much should the voltage be reduced to maintain the same leakage current? Note: power is defined as the product of voltage and current.

**1.9** Assume for arithmetic, load/store, and branch instructions, a processor has CPIs of  $1$ ,  $12$ , and  $5$ , respectively. Also assume that on a single processor a program requires the execution of  $2.56E9$  arithmetic instructions,  $1.28E9$  load/store instructions, and  $256$  million branch instructions. Assume that each processor has a  $2$  GHz clock frequency.

Assume that, as the program is parallelized to run over multiple cores, the number of arithmetic and load/store instructions per processor is divided by  $0.7 \times p$  (where  $p$  is the number of processors) but the number of branch instructions per processor remains the same.

**1.9.1** [5] <§1.7> Find the total execution time for this program on  $1$ ,  $2$ ,  $4$ , and  $8$  processors, and show the relative speedup of the  $2$ ,  $4$ , and  $8$  processors result relative to the single processor result.

**1.9.2** [10] <§§1.6, 1.8> If the CPI of the arithmetic instructions was doubled, what would the impact be on the execution time of the program on 1, 2, 4, or 8 processors?

**1.9.3** [10] <§§1.6, 1.8> To what should the CPI of load/store instructions be reduced in order for a single processor to match the performance of four processors using the original CPI values?

**1.10** Assume a 15 cm diameter wafer has a cost of 12, contains 84 dies, and has 0.020 defects/cm<sup>2</sup>. Assume a 20 cm diameter wafer has a cost of 15, contains 100 dies, and has 0.031 defects/cm<sup>2</sup>.

**1.10.1** [10] <§1.5> Find the yield for both wafers.

**1.10.2** [5] <§1.5> Find the cost per die for both wafers.

**1.10.3** [5] <§1.5> If the number of dies per wafer is increased by 10% and the defects per area unit increases by 15%, find the die area and yield.

**1.10.4** [5] <§1.5> Assume a fabrication process improves the yield from 0.92 to 0.95. Find the defects per area unit for each version of the technology given a die area of 200 mm<sup>2</sup>.

**1.11** The results of the SPEC CPU2006 bzip2 benchmark running on an AMD Barcelona has an instruction count of 2.389E12, an execution time of 750 s, and a reference time of 9650 s.

**1.11.1** [5] <§§1.6, 1.9> Find the CPI if the clock cycle time is 0.333 ns.

**1.11.2** [5] <§1.9> Find the SPECratio.

**1.11.3** [5] <§§1.6, 1.9> Find the increase in CPU time if the number of instructions of the benchmark is increased by 10% without affecting the CPI.

**1.11.4** [5] <§§1.6, 1.9> Find the increase in CPU time if the number of instructions of the benchmark is increased by 10% and the CPI is increased by 5%.

**1.11.5** [5] <§§1.6, 1.9> Find the change in the SPECratio for this change.

**1.11.6** [10] <§1.6> Suppose that we are developing a new version of the AMD Barcelona processor with a 4 GHz clock rate. We have added some additional instructions to the instruction set in such a way that the number of instructions has been reduced by 15%. The execution time is reduced to 700 s and the new SPECratio is 13.7. Find the new CPI.

**1.11.7** [10] <§1.6> This CPI value is larger than obtained in 1.11.1 as the clock rate was increased from 3 GHz to 4 GHz. Determine whether the increase in the CPI is similar to that of the clock rate. If they are dissimilar, why?

**1.11.8** [5] <§1.6> By how much has the CPU time been reduced?

**1.11.9** [10] <§1.6> For a second benchmark, libquantum, assume an execution time of 960 ns, CPI of 1.61, and clock rate of 3 GHz. If the execution time is reduced by an additional 10% without affecting the CPI and with a clock rate of 4 GHz, determine the number of instructions.

**1.11.10** [10] <§1.6> Determine the clock rate required to give a further 10% reduction in CPU time while maintaining the number of instructions and with the CPI unchanged.

**1.11.11** [10] <§1.6> Determine the clock rate if the CPI is reduced by 15% and the CPU time by 20% while the number of instructions is unchanged.

**1.12** Section 1.10 cites as a pitfall the utilization of a subset of the performance equation as a performance metric. To illustrate this, consider the following two processors. P1 has a clock rate of 4 GHz, average CPI of 0.9, and requires the execution of 5.0E9 instructions. P2 has a clock rate of 3 GHz, an average CPI of 0.75, and requires the execution of 1.0E9 instructions.

**1.12.1** [5] <§§1.6, 1.10> One usual fallacy is to consider the computer with the largest clock rate as having the highest performance. Check if this is true for P1 and P2.

**1.12.2** [10] <§§1.6, 1.10> Another fallacy is to consider that the processor executing the largest number of instructions will need a larger CPU time. Considering that processor P1 is executing a sequence of 1.0E9 instructions and that the CPI of processors P1 and P2 do not change, determine the number of instructions that P2 can execute in the same time that P1 needs to execute 1.0E9 instructions.

**1.12.3** [10] <§§1.6, 1.10> A common fallacy is to use MIPS (*millions of instructions per second*) to compare the performance of two different processors, and consider that the processor with the largest MIPS has the largest performance. Check if this is true for P1 and P2.

**1.12.4** [10] <§1.10> Another common performance figure is MFLOPS (millions of floating-point operations per second), defined as

$$\text{MFLOPS} = \text{No. FP operations} / (\text{execution time} \times 1\text{E}6)$$

but this figure has the same problems as MIPS. Assume that 40% of the instructions executed on both P1 and P2 are floating-point instructions. Find the MFLOPS figures for the processors.

**1.13** Another pitfall cited in Section 1.10 is expecting to improve the overall performance of a computer by improving only one aspect of the computer. Consider a computer running a program that requires 250 s, with 70 s spent executing FP instructions, 85 s executed L/S instructions, and 40 s spent executing branch instructions.

**1.13.1** [5] <§1.10> By how much is the total time reduced if the time for FP operations is reduced by 20%?

**1.13.2** [5] <§1.10> By how much is the time for INT operations reduced if the total time is reduced by 20%?

**1.13.3** [5] <§1.10> Can the total time can be reduced by 20% by reducing only the time for branch instructions?

**1.14** Assume a program requires the execution of  $50 \times 10^6$  FP instructions,  $110 \times 10^6$  INT instructions,  $80 \times 10^6$  L/S instructions, and  $16 \times 10^6$  branch instructions. The CPI for each type of instruction is 1, 1, 4, and 2, respectively. Assume that the processor has a 2 GHz clock rate.

**1.14.1** [10] <§1.10> By how much must we improve the CPI of FP instructions if we want the program to run two times faster?

**1.14.2** [10] <§1.10> By how much must we improve the CPI of L/S instructions if we want the program to run two times faster?

**1.14.3** [5] <§1.10> By how much is the execution time of the program improved if the CPI of INT and FP instructions is reduced by 40% and the CPI of L/S and Branch is reduced by 30%?

**1.15** [5] <§1.8> When a program is adapted to run on multiple processors in a multiprocessor system, the execution time on each processor is comprised of computing time and the overhead time required for locked critical sections and/or to send data from one processor to another.

Assume a program requires  $t = 100$  s of execution time on one processor. When run  $p$  processors, each processor requires  $t/p$  s, as well as an additional 4 s of overhead, irrespective of the number of processors. Compute the per-processor execution time for 2, 4, 8, 16, 32, 64, and 128 processors. For each case, list the corresponding speedup relative to a single processor and the ratio between actual speedup versus ideal speedup (speedup if there was no overhead).

§1.1, page 10: Discussion questions: many answers are acceptable.

§1.4, page 24: DRAM memory: volatile, short access time of 50 to 70 nanoseconds, and cost per GB is \$5 to \$10. Disk memory: nonvolatile, access times are 100,000 to 400,000 times slower than DRAM, and cost per GB is 100 times cheaper than DRAM. Flash memory: nonvolatile, access times are 100 to 1000 times slower than DRAM, and cost per GB is 7 to 10 times cheaper than DRAM.

§1.5, page 28: 1, 3, and 4 are valid reasons. Answer 5 can be generally true because high volume can make the extra investment to reduce die size by, say, 10% a good economic decision, but it doesn't have to be true.

§1.6, page 33: 1. a: both, b: latency, c: neither. 7 seconds.

§1.6, page 40: b.

§1.10, page 51: a. Computer A has the higher MIPS rating. b. Computer B is faster.


**Answers to  
Check Yourself**

Instruction class	RISC-V examples	HLL correspondence	Frequency	
			Integer	Fl. Pt.
Arithmetic	add, sub, addi	Operations in assignment statements	16%	48%
Data transfer	ld, sd, lw, sw, lh, sh, lb, sb, lui	References to data structures in memory	35%	36%
Logical	and, or, xor, sll, srl, sra	Operations in assignment statements	12%	4%
Branch	beq, bne, blt, bge, bltu, bgeu	If statements; loops	34%	8%
Jump	jal, jalr	Procedure calls & returns; switch statements	2%	0%

**FIGURE 2.41** RISC-V instruction classes, examples, correspondence to high-level program language constructs, and percentage of RISC-V instructions executed by category for the average integer and floating point SPEC CPU2006 benchmarks. Figure 3.24 in Chapter 3 shows average percentage of the individual RISC-V instructions executed.



## Historical Perspective and Further Reading

This section surveys the history of *instruction set architectures* (ISAs) over time, and we give a short history of programming languages and compilers. ISAs include accumulator architectures, general-purpose register architectures, stack architectures, and a brief history of the x86 and ARM’s 32-bit architecture, ARMv7. We also review the controversial subjects of high-level-language computer architectures and reduced instruction set computer architectures. The history of programming languages includes Fortran, Lisp, Algol, C, Cobol, Pascal, Simula, Smalltalk, C++, and Java, and the history of compilers includes the key milestones and the pioneers who achieved them. The rest of  [Section 2.21](#) is found online.



## Exercises

**2.1** [5] <§2.2> For the following C statement, write the corresponding RISC-V assembly code. Assume that the C variables *f*, *g*, and *h*, have already been placed in registers *x5*, *x6*, and *x7* respectively. Use a minimal number of RISC-V assembly instructions.

$$f = g + (h - 5);$$

**2.2** [5] <§2.2> Write a single C statement that corresponds to the two RISC-V assembly instructions below.

```
add f, g, h
add f, i, f
```

**2.3** [5] <§§2.2, 2.3> For the following C statement, write the corresponding RISC-V assembly code. Assume that the variables *f*, *g*, *h*, *i*, and *j* are assigned to registers *x5*, *x6*, *x7*, *x28*, and *x29*, respectively. Assume that the base address of the arrays *A* and *B* are in registers *x10* and *x11*, respectively.

```
B[8] = A[i-j];
```

**2.4** [10] <§§2.2, 2.3> For the RISC-V assembly instructions below, what is the corresponding C statement? Assume that the variables *f*, *g*, *h*, *i*, and *j* are assigned to registers *x5*, *x6*, *x7*, *x28*, and *x29*, respectively. Assume that the base address of the arrays *A* and *B* are in registers *x10* and *x11*, respectively.

```
slli x30, x5, 3      // x30 = f*8
add  x30, x10, x30    // x30 = &A[f]
slli x31, x6, 3      // x31 = g*8
add  x31, x11, x31    // x31 = &B[g]
ld   x5, 0(x30)      // f = A[f]

addi x12, x30, 8
ld   x30, 0(x12)
add  x30, x30, x5
sd   x30, 0(x31)
```

**2.5** [5] <§2.3> Show how the value `0xabcdef12` would be arranged in memory of a little-endian and a big-endian machine. Assume the data are stored starting at address 0 and that the word size is 4 bytes.

**2.6** [5] <§2.4> Translate `0xabcdef12` into decimal.

**2.7** [5] <§§2.2, 2.3> Translate the following C code to RISC-V. Assume that the variables *f*, *g*, *h*, *i*, and *j* are assigned to registers *x5*, *x6*, *x7*, *x28*, and *x29*, respectively. Assume that the base address of the arrays *A* and *B* are in registers *x10* and *x11*, respectively. Assume that the elements of the arrays *A* and *B* are 8-byte words:

```
B[8] = A[i] + A[j];
```

**2.8** [10] <§§2.2, 2.3> Translate the following RISC-V code to C. Assume that the variables *f*, *g*, *h*, *i*, and *j* are assigned to registers *x5*, *x6*, *x7*, *x28*, and *x29*,



respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

```
addi x30, x10, 8
addi x31, x10, 0
sd   x31, 0(x30)
ld   x30, 0(x30)
add  x5, x30, x31
```

**2.9** [20] <§2.2, 2.5> For each RISC-V instruction in Exercise 2.8, show the value of the opcode (op), source register (rs1), and destination register (rd) fields. For the I-type instructions, show the value of the immediate field, and for the R-type instructions, show the value of the second source register (rs2). For non U- and UJ-type instructions, show the funct3 field, and for R-type and S-type instructions, also show the funct7 field.

**2.10** Assume that registers x5 and x6 hold the values 0x8000000000000000 and 0xD000000000000000, respectively.

**2.10.1** [5] <§2.4> What is the value of x30 for the following assembly code?

```
add x30, x5, x6
```

**2.10.2** [5] <§2.4> Is the result in x30 the desired result, or has there been overflow?

**2.10.3** [5] <§2.4> For the contents of registers x5 and x6 as specified above, what is the value of x30 for the following assembly code?

```
sub x30, x5, x6
```

**2.10.4** [5] <§2.4> Is the result in x30 the desired result, or has there been overflow?

**2.10.5** [5] <§2.4> For the contents of registers x5 and x6 as specified above, what is the value of x30 for the following assembly code?

```
add x30, x5, x6
add x30, x30, x5
```

**2.10.6** [5] <§2.4> Is the result in x30 the desired result, or has there been overflow?

**2.11** Assume that x5 holds the value 128<sub>ten</sub>.

**2.11.1** [5] <§2.4> For the instruction add x30, x5, x6, what is the range(s) of values for x6 that would result in overflow?

**2.11.2** [5] <§2.4> For the instruction sub x30, x5, x6, what is the range(s) of values for x6 that would result in overflow?

**2.11.3** [5] <§2.4> For the instruction `sub x30, x6, x5`, what is the range(s) of values for `x6` that would result in overflow?

**2.12** [5] <§§2.2, 2.5> Provide the instruction type and assembly language instruction for the following binary value:

```
0000 0000 0001 0000 1000 0000 1011 0011two
```

Hint: [Figure 2.20](#) may be helpful.

**2.13** [5] <§§2.2, 2.5> Provide the instruction type and hexadecimal representation of the following instruction:

```
sd x5, 32(x30)
```

**2.14** [5] <§2.5> Provide the instruction type, assembly language instruction, and binary representation of instruction described by the following RISC-V fields:

```
opcode=0x33, funct3=0x0, funct7=0x20, rs2=5, rs1=7, rd=6
```

**2.15** [5] <§2.5> Provide the instruction type, assembly language instruction, and binary representation of instruction described by the following RISC-V fields:

```
opcode=0x3, funct3=0x3, rs1=27, rd=3, imm=0x4
```

**2.16** Assume that we would like to expand the RISC-V register file to 128 registers and expand the instruction set to contain four times as many instructions.

**2.16.1** [5] <§2.5> How would this affect the size of each of the bit fields in the R-type instructions?

**2.16.2** [5] <§2.5> How would this affect the size of each of the bit fields in the I-type instructions?

**2.16.3** [5] <§§2.5, 2.8, 2.10> How could each of the two proposed changes decrease the size of a RISC-V assembly program? On the other hand, how could the proposed change increase the size of a RISC-V assembly program?

**2.17** Assume the following register contents:

```
x5 = 0x00000000AAAAAAAA, x6 = 0x1234567812345678
```

**2.17.1** [5] <§2.6> For the register values shown above, what is the value of `x7` for the following sequence of instructions?

```
slli x7, x5, 4
or   x7, x7, x6
```

**2.17.2** [5] <§2.6> For the register values shown above, what is the value of `x7` for the following sequence of instructions?

```
slli x7, x6, 4
```

**2.17.3** [5] <§2.6> For the register values shown above, what is the value of `x7` for the following sequence of instructions?

```
srli x7, x5, 3
andi x7, x7, 0xFEf
```

**2.18** [10] <§2.6> Find the shortest sequence of RISC-V instructions that extracts bits 16 down to 11 from register `x5` and uses the value of this field to replace bits 31 down to 26 in register `x6` without changing the other bits of registers `x5` or `x6`. (Be sure to test your code using `x5 = 0` and `x6 = 0xffffffffffffffff`. Doing so may reveal a common oversight.)

**2.19** [5] <§2.6> Provide a minimal set of RISC-V instructions that may be used to implement the following pseudoinstruction:

```
not x5, x6      // bit-wise invert
```

**2.20** [5] <§2.6> For the following C statement, write a minimal sequence of RISC-V assembly instructions that performs the identical operation. Assume `x6 = A`, and `x17` is the base address of `C`.

```
A = C[0] << 4;
```

**2.21** [5] <§2.7> Assume `x5` holds the value `0x00000000001010000`. What is the value of `x6` after the following instructions?

```
bge x5, x0, ELSE
jal x0, DONE
ELSE: ori x6, x0, 2
DONE:
```

**2.22** Suppose the *program counter* (PC) is set to `0x20000000`.

**2.22.1** [5] <§2.10> What range of addresses can be reached using the RISC-V *jump-and-link* (`jal`) instruction? (In other words, what is the set of possible values for the PC after the jump instruction executes?)

**2.22.2** [5] <§2.10> What range of addresses can be reached using the RISC-V *branch if equal* (`beq`) instruction? (In other words, what is the set of possible values for the PC after the branch instruction executes?)

**2.23** Consider a proposed new instruction named `rpt`. This instruction combines a loop's condition check and counter decrement into a single instruction. For example `rpt x29, loop` would do the following:

```
if (x29 > 0) {
    x29 = x29 - 1;
    goto loop
}
```

**2.23.1** [5] <§2.7, 2.10> If this instruction were to be added to the RISC-V instruction set, what is the most appropriate instruction format?

**2.23.2** [5] <§2.7> What is the shortest sequence of RISC-V instructions that performs the same operation?

**2.24** Consider the following RISC-V loop:

```
LOOP: beq x6, x0, DONE
      addi x6, x6, -1
      addi x5, x5, 2
      jal x0, LOOP
DONE:
```

**2.24.1** [5] <§2.7> Assume that the register `x6` is initialized to the value 10. What is the final value in register `x5` assuming the `x5` is initially zero?

**2.24.2** [5] <§2.7> For the loop above, write the equivalent C code. Assume that the registers `x5` and `x6` are integers `acc` and `i`, respectively.

**2.24.3** [5] <§2.7> For the loop written in RISC-V assembly above, assume that the register `x6` is initialized to the value `N`. How many RISC-V instructions are executed?

**2.24.4** [5] <§2.7> For the loop written in RISC-V assembly above, replace the instruction “`beq x6, x0, DONE`” with the instruction “`blt x6, x0, DONE`” and write the equivalent C code.

**2.25** [10] <§2.7> Translate the following C code to RISC-V assembly code. Use a minimum number of instructions. Assume that the values of `a`, `b`, `i`, and `j` are in registers `x5`, `x6`, `x7`, and `x29`, respectively. Also, assume that register `x10` holds the base address of the array `D`.

```
for(i=0; i<a; i++)
    for(j=0; j<b; j++)
        D[4*j] = i + j;
```

**2.26** [5] <\$2.7> How many RISC-V instructions does it take to implement the C code from Exercise 2.25? If the variables *a* and *b* are initialized to 10 and 1 and all elements of *D* are initially 0, what is the total number of RISC-V instructions executed to complete the loop?

**2.27** [5] <\$2.7> Translate the following loop into C. Assume that the C-level integer *i* is held in register *x5*, *x6* holds the C-level integer called *result*, and *x10* holds the base address of the integer *MemArray*.

```
        addi x6, x0, 0
        addi x29, x0, 100
LOOP:   ld    x7, 0(x10)
        add  x5, x5, x7
        addi x10, x10, 8
        addi x6, x6, 1
        blt  x6, x29, LOOP
```

**2.28** [10] <\$2.7> Rewrite the loop from Exercise 2.27 to reduce the number of RISC-V instructions executed. Hint: Notice that variable *i* is used only for loop control.

**2.29** [30] <\$2.8> Implement the following C code in RISC-V assembly. Hint: Remember that the stack pointer must remain aligned on a multiple of 16.

```
int fib(int n){
    if (n==0)
        return 0;
    else if (n == 1)
        return 1;
    else
        return fib(n-1) + fib(n-2);
}
```

**2.30** [20] <\$2.8> For each function call in Exercise 2.29, show the contents of the stack after the function call is made. Assume the stack pointer is originally at address 0x7fffffff, and follow the register conventions as specified in [Figure 2.11](#).

**2.31** [20] <\$2.8> Translate function *f* into RISC-V assembly language. Assume the function declaration for *g* is `int g(int a, int b)`. The code for function *f* is as follows:

```
int f(int a, int b, int c, int d){
    return g(g(a,b), c+d);
}
```

**2.32** [5] <§2.8> Can we use the tail-call optimization in this function? If no, explain why not. If yes, what is the difference in the number of executed instructions in `f` with and without the optimization?

**2.33** [5] <§2.8> Right before your function `f` from Exercise 2.31 returns, what do we know about contents of registers `x10-x14`, `x8`, `x1`, and `sp`? Keep in mind that we know what the entire function `f` looks like, but for function `g` we only know its declaration.

**2.34** [30] <§2.9> Write a program in RISC-V assembly to convert an ASCII string containing a positive or negative integer decimal string to an integer. Your program should expect register `x10` to hold the address of a null-terminated string containing an optional “+” or “-” followed by some combination of the digits 0 through 9. Your program should compute the integer value equivalent to this string of digits, then place the number in register `x10`. If a non-digit character appears anywhere in the string, your program should stop with the value  $-1$  in register `x10`. For example, if register `x10` points to a sequence of three bytes  $50_{\text{ten}}$ ,  $52_{\text{ten}}$ ,  $0_{\text{ten}}$  (the null-terminated string “24”), then when the program stops, register `x10` should contain the value  $24_{\text{ten}}$ . The RISC-V `mul` instruction takes two registers as input. There is no “`mul i`” instruction. Thus, just store the constant 10 in a register.

**2.35** Consider the following code:

```
lb x6, 0(x7)
sd x6, 8(x7)
```

Assume that the register `x7` contains the address  $0 \times 10000000$  and the data at address is  $0 \times 1122334455667788$ .

**2.35.1** [5] <§2.3, 2.9> What value is stored in  $0 \times 10000008$  on a big-endian machine?

**2.35.2** [5] <§2.3, 2.9> What value is stored in  $0 \times 10000008$  on a little-endian machine?

**2.36** [5] <§2.10> Write the RISC-V assembly code that creates the 64-bit constant  $0 \times 1122334455667788_{\text{two}}$  and stores that value to register `x10`.

**2.37** [10] <§2.11> Write the RISC-V assembly code to implement the following C code as an atomic “set max” operation using the `lr.d/sc.d` instructions. Here, the argument `shvar` contains the address of a shared variable which should be replaced by `x` if `x` is greater than the value it points to:

```
void setmax(int* shvar, int x) {  
    // Begin critical section  
    if (x > *shvar)  
        *shvar = x;  
    // End critical section  
}
```

**2.38** [5] <§2.11> Using your code from Exercise 2.37 as an example, explain what happens when two processors begin to execute this critical section at the same time, assuming that each processor executes exactly one instruction per cycle.

**2.39** Assume for a given processor the CPI of arithmetic instructions is 1, the CPI of load/store instructions is 10, and the CPI of branch instructions is 3. Assume a program has the following instruction breakdowns: 500 million arithmetic instructions, 300 million load/store instructions, 100 million branch instructions.

**2.39.1** [5] <§§1.6, 2.13> Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, while increasing the clock cycle time by only 10%. Is this a good design choice? Why?

**2.39.2** [5] <§§1.6, 2.13> Suppose that we find a way to double the performance of arithmetic instructions. What is the overall speedup of our machine? What if we find a way to improve the performance of arithmetic instructions by 10 times?

**2.40** Assume that for a given program 70% of the executed instructions are arithmetic, 10% are load/store, and 20% are branch.

**2.40.1** [5] <§§1.6, 2.13> Given this instruction mix and the assumption that an arithmetic instruction requires two cycles, a load/store instruction takes six cycles, and a branch instruction takes three cycles, find the average CPI.

**2.40.2** [5] <§§1.6, 2.13> For a 25% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

**2.40.3** [5] <§§1.6, 2.13> For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

**2.41** [10] <§2.19> Suppose the RISC-V ISA included a scaled offset addressing mode similar to the x86 one described in [Section 2.17](#) ([Figure 2.35](#)). Describe how you would use scaled offset loads to further reduce the number of assembly instructions needed to carry out the function given in Exercise 2.4.

**2.42** [10] <§2.19> Suppose the RISC-V ISA included a scaled offset addressing mode similar to the x86 one described in [Section 2.17](#) ([Figure 2.35](#)). Describe how you would use scaled offset loads to further reduce the number of assembly instructions needed to implement the C code given in Exercise 2.7.

§2.2, page 66: RISC-V, C, Java.

§2.3, page 73: 2) Very slow.

§2.4, page 80: 2)  $-8_{\text{ten}}$

§2.5, page 89: 3) `sub x11, x10, x9`

§2.6, page 92: Both. AND with a mask pattern of 1s will leaves 0s everywhere but the desired field. Shifting left by the correct amount removes the bits from the left of the field. Shifting right by the appropriate amount puts the field into the rightmost bits of the doubleword, with 0s in the rest of the doubleword. Note that AND leaves the field where it was originally, and the shift pair moves the field into the rightmost part of the doubleword.

§2.7, page 97: I. All are true. II. 1).

§2.8, page 108: Both are true.

§2.9, page 113: I. 1) and 2) II. 3).

§2.10, page 121: I. 4)  $\pm 4K$ . II. 4)  $\pm 1M$ .

§2.11, page 124: Both are true.

§2.12, page 133: 4) Machine independence.

## Answers to Check Yourself



observation has significant implications for the design of the processor, as we will see in [Chapter 4](#).

No matter what the instruction set or its size—RISC-V, MIPS, x86—never forget that bit patterns have no inherent meaning. The same bit pattern may represent a signed integer, unsigned integer, floating-point number, string, instruction, and so on. In stored-program computers, it is the operation on the bit pattern that determines its meaning.



## Historical Perspective and Further Reading

This section surveys the history of the floating point going back to von Neumann, including the surprisingly controversial IEEE standards effort, plus the rationale for the 80-bit stack architecture for floating point in the x86. See the rest of [Section 3.11](#) online.

*Gresham's Law ("Bad money drives out Good") for computers would say, "The Fast drives out the Slow even if the Fast is wrong."*

W. Kahan, 1992

## 3.12

## Exercises

**3.1** [5] <§3.2> What is  $5ED4 - 07A4$  when these values represent unsigned 16-bit hexadecimal numbers? The result should be written in hexadecimal. Show your work.

**3.2** [5] <§3.2> What is  $5ED4 - 07A4$  when these values represent signed 16-bit hexadecimal numbers stored in sign-magnitude format? The result should be written in hexadecimal. Show your work.

**3.3** [10] <§3.2> Convert  $5ED4$  into a binary number. What makes base 16 (hexadecimal) an attractive numbering system for representing values in computers?

**3.4** [5] <§3.2> What is  $4365 - 3412$  when these values represent unsigned 12-bit octal numbers? The result should be written in octal. Show your work.

**3.5** [5] <§3.2> What is  $4365 - 3412$  when these values represent signed 12-bit octal numbers stored in sign-magnitude format? The result should be written in octal. Show your work.

**3.6** [5] <§3.2> Assume 185 and 122 are unsigned 8-bit decimal integers. Calculate  $185 - 122$ . Is there overflow, underflow, or neither?

*Never give in, never give in, never, never, never—in nothing, great or small, large or petty—never give in.*

Winston Churchill,  
address at Harrow  
School, 1941

**3.7** [5] <§3.2> Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate  $185 + 122$ . Is there overflow, underflow, or neither?

**3.8** [5] <§3.2> Assume 185 and 122 are signed 8-bit decimal integers stored in sign-magnitude format. Calculate  $185 - 122$ . Is there overflow, underflow, or neither?

**3.9** [10] <§3.2> Assume 151 and 214 are signed 8-bit decimal integers stored in two's complement format. Calculate  $151 + 214$  using saturating arithmetic. The result should be written in decimal. Show your work.

**3.10** [10] <§3.2> Assume 151 and 214 are signed 8-bit decimal integers stored in two's complement format. Calculate  $151 - 214$  using saturating arithmetic. The result should be written in decimal. Show your work.

**3.11** [10] <§3.2> Assume 151 and 214 are unsigned 8-bit integers. Calculate  $151 + 214$  using saturating arithmetic. The result should be written in decimal. Show your work.

**3.12** [20] <§3.3> Using a table similar to that shown in [Figure 3.6](#), calculate the product of the octal unsigned 6-bit integers 62 and 12 using the hardware described in [Figure 3.3](#). You should show the contents of each register on each step.

**3.13** [20] <§3.3> Using a table similar to that shown in [Figure 3.6](#), calculate the product of the hexadecimal unsigned 8-bit integers 62 and 12 using the hardware described in [Figure 3.5](#). You should show the contents of each register on each step.

**3.14** [10] <§3.3> Calculate the time necessary to perform a multiply using the approach given in [Figures 3.3 and 3.4](#) if an integer is 8 bits wide and each step of the operation takes four time units. Assume that in step 1a an addition is always performed—either the multiplicand will be added, or a zero will be. Also assume that the registers have already been initialized (you are just counting how long it takes to do the multiplication loop itself). If this is being done in hardware, the shifts of the multiplicand and multiplier can be done simultaneously. If this is being done in software, they will have to be done one after the other. Solve for each case.

**3.15** [10] <§3.3> Calculate the time necessary to perform a multiply using the approach described in the text (31 adders stacked vertically) if an integer is 8 bits wide and an adder takes four time units.

**3.16** [20] <§3.3> Calculate the time necessary to perform a multiply using the approach given in [Figure 3.7](#) if an integer is 8 bits wide and an adder takes four time units.

**3.17** [20] <§3.3> As discussed in the text, one possible performance enhancement is to do a shift and add instead of an actual multiplication. Since  $9 \times 6$ , for example, can be written  $(2 \times 2 \times 2 + 1) \times 6$ , we can calculate  $9 \times 6$  by shifting 6 to the left three times and then adding 6 to that result. Show the best way to calculate  $0 \times 33 \times 0 \times 55$  using shifts and adds/subtracts. Assume both inputs are 8-bit unsigned integers.

**3.18** [20] <§3.4> Using a table similar to that shown in Figure 3.10, calculate 74 divided by 21 using the hardware described in Figure 3.8. You should show the contents of each register on each step. Assume both inputs are unsigned 6-bit integers.

**3.19** [30] <§3.4> Using a table similar to that shown in Figure 3.10, calculate 74 divided by 21 using the hardware described in Figure 3.11. You should show the contents of each register on each step. Assume A and B are unsigned 6-bit integers. This algorithm requires a slightly different approach than that shown in Figure 3.9. You will want to think hard about this, do an experiment or two, or else go to the web to figure out how to make this work correctly. (Hint: one possible solution involves using the fact that Figure 3.11 implies the remainder register can be shifted either direction.)

**3.20** [5] <§3.5> What decimal number does the bit pattern  $0 \times 0C000000$  represent if it is a two's complement integer? An unsigned integer?

**3.21** [10] <§3.5> If the bit pattern  $0 \times 0C000000$  is placed into the Instruction Register, what MIPS instruction will be executed?

**3.22** [10] <§3.5> What decimal number does the bit pattern  $0 \times 0C000000$  represent if it is a floating point number? Use the IEEE 754 standard.

**3.23** [10] <§3.5> Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 single precision format.

**3.24** [10] <§3.5> Write down the binary representation of the decimal number 63.25 assuming the IEEE 754 double precision format.

**3.25** [10] <§3.5> Write down the binary representation of the decimal number 63.25 assuming it was stored using the single precision IBM format (base 16, instead of base 2, with 7 bits of exponent).

**3.26** [20] <§3.5> Write down the binary bit pattern to represent  $-1.5625 \times 10^{-1}$  assuming a format similar to that employed by the DEC PDP-8 (the leftmost 12 bits are the exponent stored as a two's complement number, and the rightmost 24 bits are the fraction stored as a two's complement number). No hidden 1 is used. Comment on how the range and accuracy of this 36-bit pattern compares to the single and double precision IEEE 754 standards.

**3.27** [20] <§3.5> IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern to represent  $-1.5625 \times 10^{-1}$  assuming a version of this format, which uses an excess-16 format to store the exponent. Comment on how the range and accuracy of this 16-bit floating point format compares to the single precision IEEE 754 standard.

**3.28** [20] <§3.5> The Hewlett-Packard 2114, 2115, and 2116 used a format with the leftmost 16 bits being the fraction stored in two's complement format,

followed by another 16-bit field which had the leftmost 8 bits as an extension of the fraction (making the fraction 24 bits long), and the rightmost 8 bits representing the exponent. However, in an interesting twist, the exponent was stored in sign-magnitude format with the sign bit on the far right! Write down the bit pattern to represent  $-1.5625 \times 10^{-1}$  assuming this format. No hidden 1 is used. Comment on how the range and accuracy of this 32-bit pattern compares to the single precision IEEE 754 standard.

**3.29** [20] <\$3.5> Calculate the sum of  $2.6125 \times 10^1$  and  $4.150390625 \times 10^{-1}$  by hand, assuming A and B are stored in the 16-bit half precision described in Exercise 3.27. Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps.

**3.30** [30] <\$3.5> Calculate the product of  $-8.0546875 \times 10^0$  and  $-1.79931640625 \times 10^{-1}$  by hand, assuming A and B are stored in the 16-bit half precision format described in Exercise 3.27. Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps; however, as is done in the example in the text, you can do the multiplication in human-readable format instead of using the techniques described in Exercises 3.12 through 3.14. Indicate if there is overflow or underflow. Write your answer in both the 16-bit floating point format described in Exercise 3.27 and also as a decimal number. How accurate is your result? How does it compare to the number you get if you do the multiplication on a calculator?

**3.31** [30] <\$3.5> Calculate by hand  $8.625 \times 10^1$  divided by  $-4.875 \times 10^0$ . Show all the steps necessary to achieve your answer. Assume there is a guard, a round bit, and a sticky bit, and use them if necessary. Write the final answer in both the 16-bit floating point format described in Exercise 3.27 and in decimal and compare the decimal result to that which you get if you use a calculator.

**3.32** [20] <\$3.10> Calculate  $(3.984375 \times 10^{-1} + 3.4375 \times 10^{-1}) + 1.771 \times 10^3$  by hand, assuming each of the values is stored in the 16-bit half precision format described in Exercise 3.27 (and also described in the text). Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps, and write your answer in both the 16-bit floating point format and in decimal.

**3.33** [20] <\$3.10> Calculate  $3.984375 \times 10^{-1} + (3.4375 \times 10^{-1} + 1.771 \times 10^3)$  by hand, assuming each of the values is stored in the 16-bit half precision format described in Exercise 3.27 (and also described in the text). Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps, and write your answer in both the 16-bit floating point format and in decimal.

**3.34** [10] <\$3.10> Based on your answers to Exercises 3.32 and 3.33, does  $(3.984375 \times 10^{-1} + 3.4375 \times 10^{-1}) + 1.771 \times 10^3 = 3.984375 \times 10^{-1} + (3.4375 \times 10^{-1} + 1.771 \times 10^3)$ ?

**3.35** [30] <\$3.10> Calculate  $(3.41796875 \times 10^{-3} \times 6.34765625 \times 10^{-3}) \times 1.05625 \times 10^2$  by hand, assuming each of the values is stored in the 16-bit half precision format described in Exercise 3.27 (and also described in the text). Assume 1 guard, 1 round

bit, and 1 sticky bit, and round to the nearest even. Show all the steps, and write your answer in both the 16-bit floating point format and in decimal.

**3.36** [30] <§3.10> Calculate  $3.41796875 \times 10^{-3} \times (6.34765625 \times 10^{-3} \times 1.05625 \times 10^2)$  by hand, assuming each of the values is stored in the 16-bit half precision format described in Exercise 3.27 (and also described in the text). Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps, and write your answer in both the 16-bit floating point format and in decimal.

**3.37** [10] <§3.10> Based on your answers to Exercises 3.35 and 3.36, does  $(3.41796875 \times 10^{-3} \times 6.34765625 \times 10^{-3}) \times 1.05625 \times 10^2 = 3.41796875 \times 10^{-3} \times (6.34765625 \times 10^{-3} \times 1.05625 \times 10^2)$ ?

**3.38** [30] <§3.10> Calculate  $1.666015625 \times 10^0 \times (1.9760 \times 10^4 + -1.9744 \times 10^4)$  by hand, assuming each of the values is stored in the 16-bit half precision format described in Exercise 3.27 (and also described in the text). Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps, and write your answer in both the 16-bit floating point format and in decimal.

**3.39** [30] <§3.10> Calculate  $(1.666015625 \times 10^0 \times 1.9760 \times 10^4) + (1.666015625 \times 10^0 \times -1.9744 \times 10^4)$  by hand, assuming each of the values is stored in the 16-bit half precision format described in Exercise 3.27 (and also described in the text). Assume 1 guard, 1 round bit, and 1 sticky bit, and round to the nearest even. Show all the steps, and write your answer in both the 16-bit floating point format and in decimal.

**3.40** [10] <§3.10> Based on your answers to Exercises 3.38 and 3.39, does  $(1.666015625 \times 10^0 \times 1.9760 \times 10^4) + (1.666015625 \times 10^0 \times -1.9744 \times 10^4) = 1.666015625 \times 10^0 \times (1.9760 \times 10^4 + -1.9744 \times 10^4)$ ?

**3.41** [10] <§3.5> Using the IEEE 754 floating point format, write down the bit pattern that would represent  $-1/4$ . Can you represent  $-1/4$  exactly?

**3.42** [10] <§3.5> What do you get if you add  $-1/4$  to itself four times? What is  $-1/4 \times 4$ ? Are they the same? What should they be?

**3.43** [10] <§3.5> Write down the bit pattern in the fraction of value  $1/3$  assuming a floating point format that uses binary numbers in the fraction. Assume there are 24 bits, and you do not need to normalize. Is this representation exact?

**3.44** [10] <§3.5> Write down the bit pattern in the fraction of value  $1/3$  assuming a floating point format that uses Binary Coded Decimal (base 10) numbers in the fraction instead of base 2. Assume there are 24 bits, and you do not need to normalize. Is this representation exact?

**3.45** [10] <§3.5> Write down the bit pattern assuming that we are using base 15 numbers in the fraction of value  $1/3$  instead of base 2. (Base 16 numbers use the symbols 0–9 and A–F. Base 15 numbers would use 0–9 and A–E.) Assume there are 24 bits, and you do not need to normalize. Is this representation exact?

**3.46** [20] <§3.5> Write down the bit pattern assuming that we are using base 30 numbers in the fraction of value  $1/3$  instead of base 2. (Base 16 numbers use the symbols 0–9 and A–F. Base 30 numbers would use 0–9 and A–T.) Assume there are 20 bits, and you do not need to normalize. Is this representation exact?

**3.47** [45] <§§3.6, 3.7> The following C code implements a four-tap FIR filter on input array `sig_in`. Assume that all arrays are 16-bit fixed-point values.

```
for (i = 3; i < 128; i++)
    sig_out[i] = sig_in[i - 3] * f[0] + sig_in[i - 2] * f[1]
               + sig_in[i - 1] * f[2] + sig_in[i] * f[3];
```

Assume you are to write an optimized implementation of this code in assembly language on a processor that has SIMD instructions and 128-bit registers. Without knowing the details of the instruction set, briefly describe how you would implement this code, maximizing the use of sub-word operations and minimizing the amount of data that is transferred between registers and memory. State all your assumptions about the instructions you use.

### Answers to Check Yourself

§3.2, page 177: 2.

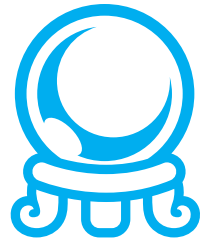
§3.5, page 215: 3.

Pipelining and multiple issue both attempt to exploit instruction-level parallelism. The presence of data and control dependences, which can become hazards, are the primary limitations on how much parallelism can be exploited. Scheduling and speculation via **prediction**, both in hardware and in software, are the primary techniques used to reduce the performance impact of dependences.

We showed that unrolling the DGEMM loop four times exposed more instructions that could take advantage of the out-of-order execution engine of the Core i7 to more than double performance.

The switch to longer pipelines, multiple instruction issue, and dynamic scheduling in the mid-1990s helped sustain the 60% per year processor performance increase that started in the early 1980s. As mentioned in [Chapter 1](#), these microprocessors preserved the sequential programming model, but they eventually ran into the power wall. Thus, the industry was forced to switch to multiprocessors, which exploit parallelism at much coarser levels (the subject of [Chapter 6](#)). This trend has also caused designers to reassess the energy-performance implications of some of the inventions since the mid-1990s, resulting in a simplification of pipelines in the more recent versions of microarchitectures.

To sustain the advances in processing performance via parallel processors, Amdahl's law suggests that another part of the system will become the bottleneck. That bottleneck is the topic of the next chapter: the **memory hierarchy**.



PREDICTION



HIERARCHY



## Historical Perspective and Further Reading

This section, which appears online, discusses the history of the first pipelined processors, the earliest superscalars, and the development of out-of-order and speculative techniques, as well as important developments in the accompanying compiler technology.

## 4.17

## Exercises

**4.1** Consider the following instruction:

Instruction: `and rd, rs1, rs2`

Interpretation: `Reg[rd] = Reg[rs1] AND Reg[rs2]`

**4.1.1** [5] <§4.3> What are the values of control signals generated by the control in [Figure 4.10](#) for this instruction?

**4.1.2** [5] <§4.3> Which resources (blocks) perform a useful function for this instruction?

**4.1.3** [10] <§4.3> Which resources (blocks) produce no output for this instruction? Which resources produce output that is not used?

**4.2** [10] <§4.4> Explain each of the “don’t cares” in [Figure 4.18](#).

**4.3** Consider the following instruction mix:

R-type	I-type (non-ld)	Load	Store	Branch	Jump
24%	28%	25%	10%	11%	2%

**4.3.1** [5] <§4.4> What fraction of all instructions use data memory?

**4.3.2** [5] <§4.4> What fraction of all instructions use instruction memory?

**4.3.3** [5] <§4.4> What fraction of all instructions use the sign extend?

**4.3.4** [5] <§4.4> What is the sign extend doing during cycles in which its output is not needed?

**4.4** When silicon chips are fabricated, defects in materials (e.g., silicon) and manufacturing errors can result in defective circuits. A very common defect is for one signal wire to get “broken” and always register a logical 0. This is often called a “stuck-at-0” fault.

**4.4.1** [5] <§4.4> Which instructions fail to operate correctly if the MemToReg wire is stuck at 0?

**4.4.2** [5] <§4.4> Which instructions fail to operate correctly if the ALUSrc wire is stuck at 0?

**4.5** In this exercise, we examine in detail how an instruction is executed in a single-cycle datapath. Problems in this exercise refer to a clock cycle in which the processor fetches the following instruction word: 0x00c6ba23.

**4.5.1** [10] <§4.4> What are the values of the ALU control unit’s inputs for this instruction?

**4.5.2** [5] <§4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

**4.5.3** [10] <§4.4> For each mux, show the values of its inputs and outputs during the execution of this instruction. List values that are register outputs at Reg [xn].

**4.5.4** [10] <§4.4> What are the input values for the ALU and the two add units?



**4.5.5** [10] <§4.4> What are the values of all inputs for the registers unit?

**4.6** Section 4.4 does not discuss I-type instructions like `addi` or `andi`.

**4.6.1** [5] <§4.4> What additional logic blocks, if any, are needed to add I-type instructions to the CPU shown in Figure 4.21? Add any necessary logic blocks to Figure 4.21 and explain their purpose.

**4.6.2** [10] <§4.4> List the values of the signals generated by the control unit for `addi`. Explain the reasoning for any “don’t care” control signals.

**4.7** Problems in this exercise assume that the logic blocks used to implement a processor’s datapath have the following latencies:

I-Mem / D-Mem	Register File	Mux	ALU	Adder	Single gate	Register Read	Register Setup	Sign extend	Control
250 ps	150 ps	25 ps	200 ps	150 ps	5 ps	30 ps	20 ps	50 ps	50 ps

“Register read” is the time needed after the rising clock edge for the new register value to appear on the output. This value applies to the PC only. “Register setup” is the amount of time a register’s data input must be stable before the rising edge of the clock. This value applies to both the PC and Register File.

**4.7.1** [5] <§4.4> What is the latency of an R-type instruction (i.e., how long must the clock period be to ensure that this instruction works correctly)?

**4.7.2** [10] <§4.4> What is the latency of `ld`? (Check your answer carefully. Many students place extra muxes on the critical path.)

**4.7.3** [10] <§4.4> What is the latency of `sd`? (Check your answer carefully. Many students place extra muxes on the critical path.)

**4.7.4** [5] <§4.4> What is the latency of `beq`?

**4.7.5** [5] <§4.4> What is the latency of an I-type instruction?

**4.7.6** [5] <§4.4> What is the minimum clock period for this CPU?

**4.8** [10] <§4.4> Suppose you could build a CPU where the clock cycle time was different for each instruction. What would the speedup of this new CPU be over the CPU presented in Figure 4.21 given the instruction mix below?

R-type/I-type (non-ld)	ld	sd	beq
52%	25%	11%	12%

**4.9** Consider the addition of a multiplier to the CPU shown in Figure 4.21. This addition will add 300 ps to the latency of the ALU, but will reduce the number of instructions by 5% (because there will no longer be a need to emulate the multiply instruction).

**4.9.1** [5] <\$4.4> What is the clock cycle time with and without this improvement?

**4.9.2** [10] <\$4.4> What is the speedup achieved by adding this improvement?

**4.9.3** [10] <\$4.4> What is the slowest the new ALU can be and still result in improved performance?

**4.10** When processor designers consider a possible improvement to the processor datapath, the decision usually depends on the cost/performance trade-off. In the following three problems, assume that we are beginning with the datapath from Figure 4.21, the latencies from Exercise 4.7, and the following costs:

I-Mem	Register File	Mux	ALU	Adder	D-Mem	Single Register	Sign extend	Single gate	Control
1000	200	10	100	30	2000	5	100	1	500

Suppose doubling the number of general purpose registers from 32 to 64 would reduce the number of `ld` and `sd` instruction by 12%, but increase the latency of the register file from 150 ps to 160 ps and double the cost from 200 to 400. (Use the instruction mix from Exercise 4.8 and ignore the other effects on the ISA discussed in Exercise 2.18.)

**4.10.1** [5] <\$4.4> What is the speedup achieved by adding this improvement?

**4.10.2** [10] <\$4.4> Compare the change in performance to the change in cost.

**4.10.3** [10] <\$4.4> Given the cost/performance ratios you just calculated, describe a situation where it makes sense to add more registers and describe a situation where it doesn't make sense to add more registers.

**4.11** Examine the difficulty of adding a proposed `lwi.d rd, rs1, rs2` (“Load With Increment”) instruction to RISC-V.

Interpretation:  $\text{Reg}[\text{rd}] = \text{Mem}[\text{Reg}[\text{rs1}] + \text{Reg}[\text{rs2}]]$

**4.11.1** [5] <\$4.4> Which new functional blocks (if any) do we need for this instruction?

**4.11.2** [5] <\$4.4> Which existing functional blocks (if any) require modification?

**4.11.3** [5] <\$4.4> Which new data paths (if any) do we need for this instruction?

**4.11.4** [5] <\$4.4> What new signals do we need (if any) from the control unit to support this instruction?

**4.12** Examine the difficulty of adding a proposed `swap rs1, rs2` instruction to RISC-V.

Interpretation: `Reg[rs2]=Reg[rs1]; Reg[rs1]=Reg[rs2]`

**4.12.1** [5] <\$4.4> Which new functional blocks (if any) do we need for this instruction?

**4.12.2** [10] <\$4.4> Which existing functional blocks (if any) require modification?

**4.12.3** [5] <\$4.4> What new data paths do we need (if any) to support this instruction?

**4.12.4** [5] <\$4.4> What new signals do we need (if any) from the control unit to support this instruction?

**4.12.5** [5] <\$4.4> Modify [Figure 4.21](#) to demonstrate an implementation of this new instruction.

**4.13** Examine the difficulty of adding a proposed `ss rs1, rs2, imm` (Store Sum) instruction to RISC-V.

Interpretation: `Mem[Reg[rs1]]=Reg[rs2]+immediate`

**4.13.1** [10] <\$4.4> Which new functional blocks (if any) do we need for this instruction?

**4.13.2** [10] <\$4.4> Which existing functional blocks (if any) require modification?

**4.13.3** [5] <\$4.4> What new data paths do we need (if any) to support this instruction?

**4.13.4** [5] <\$4.4> What new signals do we need (if any) from the control unit to support this instruction?

**4.13.5** [5] <\$4.4> Modify [Figure 4.21](#) to demonstrate an implementation of this new instruction.

**4.14** [5] <\$4.4> For which instructions (if any) is the Imm Gen block on the critical path?

**4.15** `ld` is the instruction with the longest latency on the CPU from [Section 4.4](#). If we modified `ld` and `sd` so that there was no offset (i.e., the address to be loaded from/stored to must be calculated and placed in `rs1` before calling `ld/sd`), then no instruction would use both the ALU and Data memory. This would allow us to reduce the clock cycle time. However, it would also increase the number of instructions, because many `ld` and `sd` instructions would need to be replaced with `ld/add` or `sd/add` combinations.

**4.15.1** [5] <§4.4> What would the new clock cycle time be?

**4.15.2** [10] <§4.4> Would a program with the instruction mix presented in Exercise 4.7 run faster or slower on this new CPU? By how much? (For simplicity, assume every `ld` and `sd` instruction is replaced with a sequence of two instructions.)

**4.15.3** [5] <§4.4> What is the primary factor that influences whether a program will run faster or slower on the new CPU?

**4.15.4** [5] <§4.4> Do you consider the original CPU (as shown in [Figure 4.21](#)) a better overall design; or do you consider the new CPU a better overall design? Why?

**4.16** In this exercise, we examine how pipelining affects the clock cycle time of the processor. Problems in this exercise assume that individual stages of the datapath have the following latencies:

IF	ID	EX	MEM	WB
250 ps	350 ps	150 ps	300 ps	200 ps

Also, assume that instructions executed by the processor are broken down as follows:

ALU/Logic	Jump/Branch	Load	Store
45%	20%	20%	15%

**4.16.1** [5] <§4.5> What is the clock cycle time in a pipelined and non-pipelined processor?

**4.16.2** [10] <§4.5> What is the total latency of an `ld` instruction in a pipelined and non-pipelined processor?

**4.16.3** [10] <§4.5> If we can split one stage of the pipelined datapath into two new stages, each with half the latency of the original stage, which stage would you split and what is the new clock cycle time of the processor?

**4.16.4** [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the data memory?

**4.16.5** [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the write-register port of the “Registers” unit?

**4.17** [10] <§4.5> What is the minimum number of cycles needed to completely execute  $n$  instructions on a CPU with a  $k$  stage pipeline? Justify your formula.

**4.18** [5] <§4.5> Assume that `x11` is initialized to 11 and `x12` is initialized to 22. Suppose you executed the code below on a version of the pipeline from [Section 4.5](#) that does not handle data hazards (i.e., the programmer is responsible for

addressing data hazards by inserting NOP instructions where necessary). What would the final values of registers x13 and x14 be?

```
addi    x11, x12, 5
add     x13, x11, x12
addi    x14, x11, 15
```

**4.19** [10] <§4.5> Assume that x11 is initialized to 11 and x12 is initialized to 22. Suppose you executed the code below on a version of the pipeline from [Section 4.5](#) that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). What would the final values of register x15 be? Assume the register file is written at the beginning of the cycle and read at the end of a cycle. Therefore, an ID stage will return the results of a WB state occurring during the same cycle. See [Section 4.7](#) and [Figure 4.51](#) for details.

```
addi    x11, x12, 5
add     x13, x11, x12
addi    x14, x11, 15
add     x15, x11, x11
```

**4.20** [5] <§4.5> Add NOP instructions to the code below so that it will run correctly on a pipeline that does not handle data hazards.

```
addi    x11, x12, 5
add     x13, x11, x12
addi    x14, x11, 15
add     x15, x13, x12
```

**4.21** Consider a version of the pipeline from [Section 4.5](#) that does not handle data hazards (i.e., the programmer is responsible for addressing data hazards by inserting NOP instructions where necessary). Suppose that (after optimization) a typical  $n$ -instruction program requires an additional  $4*n$  NOP instructions to correctly handle data hazards.

**4.21.1** [5] <§4.5> Suppose that the cycle time of this pipeline without forwarding is 250 ps. Suppose also that adding forwarding hardware will reduce the number of NOPs from  $.4*n$  to  $.05*n$ , but increase the cycle time to 300 ps. What is the speedup of this new pipeline compared to the one without forwarding?

**4.21.2** [10] <§4.5> Different programs will require different amounts of NOPs. How many NOPs (as a percentage of code instructions) can remain in the typical program before that program runs slower on the pipeline with forwarding?

**4.21.3** [10] <§4.5> Repeat 4.21.2; however, this time let  $x$  represent the number of NOP instructions relative to  $n$ . (In 4.21.2,  $x$  was equal to  $.4$ .) Your answer will be with respect to  $x$ .

**4.21.4** [10] <\$4.5> Can a program with only  $.075 \cdot n$  NOPs possibly run faster on the pipeline with forwarding? Explain why or why not.

**4.21.5** [10] <\$4.5> At minimum, how many NOPs (as a percentage of code instructions) must a program have before it can possibly run faster on the pipeline with forwarding?

**4.22** [5] <\$4.5> Consider the fragment of RISC-V assembly below:

```
sd    x29, 12(x16)
ld    x29, 8(x16)
sub   x17, x15, x14
beqz  x17, label
add   x15, x11, x14
sub   x15, x30, x14
```

Suppose we modify the pipeline so that it has only one memory (that handles both instructions and data). In this case, there will be a structural hazard every time a program needs to fetch an instruction during the same cycle in which another instruction accesses data.

**4.22.1** [5] <\$4.5> Draw a pipeline diagram to show where the code above will stall.

**4.22.2** [5] <\$4.5> In general, is it possible to reduce the number of stalls/NOPs resulting from this structural hazard by reordering code?

**4.22.3** [5] <\$4.5> Must this structural hazard be handled in hardware? We have seen that data hazards can be eliminated by adding NOPs to the code. Can you do the same with this structural hazard? If so, explain how. If not, explain why not.

**4.22.4** [5] <\$4.5> Approximately how many stalls would you expect this structural hazard to generate in a typical program? (Use the instruction mix from Exercise 4.8.)

**4.23** If we change load/store instructions to use a register (without an offset) as the address, these instructions no longer need to use the ALU. (See Exercise 4.15.) As a result, the MEM and EX stages can be overlapped and the pipeline has only four stages.

**4.23.1** [10] <\$4.5> How will the reduction in pipeline depth affect the cycle time?

**4.23.2** [5] <\$4.5> How might this change improve the performance of the pipeline?

**4.23.3** [5] <\$4.5> How might this change degrade the performance of the pipeline?

**4.24** [10] <§4.7> Which of the two pipeline diagrams below better describes the operation of the pipeline's hazard detection unit? Why?

Choice 1:

```
ld x11, 0(x12):    IF ID EX ME WB
add x13, x11, x14:  IF ID EX..ME WB
or x15, x16, x17:   IF ID..EX ME WB
```

Choice 2:

```
ld x11, 0(x12):    IF ID EX ME WB
add x13, x11, x14:  IF ID..EX ME WB
or x15, x16, x17:   IF..ID EX ME WB
```

**4.25** Consider the following loop.

```
LOOP: ld    x10, 0(x13)
      ld    x11, 8(x13)
      add   x12, x10, x11
      subi  x13, x13, 16
      bnez  x12, LOOP
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, that the pipeline has full forwarding support, and that branches are resolved in the EX (as opposed to the ID) stage.

**4.25.1** [10] <§4.7> Show a pipeline execution diagram for the first two iterations of this loop.

**4.25.2** [10] <§4.7> Mark pipeline stages that do not perform useful work. How often while the pipeline is full do we have a cycle in which all five pipeline stages are doing useful work? (Begin with the cycle during which the `subi` is in the IF stage. End with the cycle during which the `bnez` is in the IF stage.)

**4.26** This exercise is intended to help you understand the cost/complexity/performance trade-offs of forwarding in a pipelined processor. Problems in this exercise refer to pipelined datapaths from [Figure 4.53](#). These problems assume that, of all the instructions executed in a processor, the following fraction of these instructions has a particular type of RAW data dependence. The type of RAW data dependence is identified by the stage that produces the result (EX or MEM) and the next instruction that consumes the result (1st instruction that follows the one that produces the result, 2nd instruction that follows, or both). We assume that the register write is done in the first half of the clock cycle and that register reads are done in the second half of the cycle, so “EX to 3rd” and “MEM to 3rd” dependences

are not counted because they cannot result in data hazards. We also assume that branches are resolved in the EX stage (as opposed to the ID stage), and that the CPI of the processor is 1 if there are no data hazards.

EX to 1 <sup>st</sup> Only	MEM to 1 <sup>st</sup> Only	EX to 2 <sup>nd</sup> Only	MEM to 2 <sup>nd</sup> Only	EX to 1 <sup>st</sup> and EX to 2 <sup>nd</sup>
5%	20%	5%	10%	10%

Assume the following latencies for individual pipeline stages. For the EX stage, latencies are given separately for a processor without forwarding and for a processor with different kinds of forwarding.

IF	ID	EX (no FW)	EX (full FW)	EX (FW from EX/MEM only)	EX (FW from MEM/WB only)	MEM	WB
120 ps	100 ps	110 ps	130 ps	120 ps	120 ps	120 ps	100 ps

**4.26.1** [5] <§4.7> For each RAW dependency listed above, give a sequence of at least three assembly statements that exhibits that dependency.

**4.26.2** [5] <§4.7> For each RAW dependency above, how many NOPs would need to be inserted to allow your code from 4.26.1 to run correctly on a pipeline with no forwarding or hazard detection? Show where the NOPs could be inserted.

**4.26.3** [10] <§4.7> Analyzing each instruction independently will over-count the number of NOPs needed to run a program on a pipeline with no forwarding or hazard detection. Write a sequence of three assembly instructions so that, when you consider each instruction in the sequence independently, the sum of the stalls is larger than the number of stalls the sequence actually needs to avoid data hazards.

**4.26.4** [5] <§4.7> Assuming no other hazards, what is the CPI for the program described by the table above when run on a pipeline with no forwarding? What percent of cycles are stalls? (For simplicity, assume that all necessary cases are listed above and can be treated independently.)

**4.26.5** [5] <§4.7> What is the CPI if we use full forwarding (forward all results that can be forwarded)? What percent of cycles are stalls?

**4.26.6** [10] <§4.7> Let us assume that we cannot afford to have three-input multiplexors that are needed for full forwarding. We have to decide if it is better to forward only from the EX/MEM pipeline register (next-cycle forwarding) or only from the MEM/WB pipeline register (two-cycle forwarding). What is the CPI for each option?

**4.26.7** [5] <§4.7> For the given hazard probabilities and pipeline stage latencies, what is the speedup achieved by each type of forwarding (EX/MEM, MEM/WB, for full) as compared to a pipeline that has no forwarding?



**4.26.8** [5] <§4.7> What would be the additional speedup (relative to the fastest processor from 4.26.7) be if we added “time-travel” forwarding that eliminates all data hazards? Assume that the yet-to-be-invented time-travel circuitry adds 100ps to the latency of the full-forwarding EX stage.

**4.26.9** [5] <§4.7> The table of hazard types has separate entries for “EX to 1<sup>st</sup>” and “EX to 1<sup>st</sup> and EX to 2<sup>nd</sup>”. Why is there no entry for “MEM to 1<sup>st</sup> and MEM to 2<sup>nd</sup>”?

**4.27** Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a five-stage pipelined datapath:

```
add  x15, x12, x11
ld   x13, 4(x15)
ld   x12, 0(x2)
or   x13, x15, x13
sd   x13, 0(x15)
```

**4.27.1** [5] <§4.7> If there is no forwarding or hazard detection, insert NOPs to ensure correct execution.

**4.27.2** [10] <§4.7> Now, change and/or rearrange the code to minimize the number of NOPs needed. You can assume register x17 can be used to hold temporary values in your modified code.

**4.27.3** [10] <§4.7> If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when the original code executes?

**4.27.4** [20] <§4.7> If there is forwarding, for the first seven cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in [Figure 4.59](#).

**4.27.5** [10] <§4.7> If there is no forwarding, what new input and output signals do we need for the hazard detection unit in [Figure 4.59](#)? Using this instruction sequence as an example, explain why each signal is needed.

**4.27.6** [20] <§4.7> For the new hazard detection unit from 4.26.5, specify which output signals it asserts in each of the first five cycles during the execution of this code.

**4.28** The importance of having a good branch predictor depends on how often conditional branches are executed. Together with branch predictor accuracy, this will determine how much time is spent stalling due to mispredicted branches. In this exercise, assume that the breakdown of dynamic instructions into various instruction categories is as follows:

R-type	beqz/bnez	jal	ld	sd
40%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:

Always-Taken	Always-Not-Taken	2-Bit
45%	55%	85%

**4.28.1** [10] <\$4.8> Stall cycles due to mispredicted branches increase the CPI. What is the extra CPI due to mispredicted branches with the always-taken predictor? Assume that branch outcomes are determined in the ID stage and applied in the EX stage that there are no data hazards, and that no delay slots are used.

**4.28.2** [10] <\$4.8> Repeat 4.28.1 for the “always-not-taken” predictor.

**4.28.3** [10] <\$4.8> Repeat 4.28.1 for the 2-bit predictor.

**4.28.4** [10] <\$4.8> With the 2-bit predictor, what speedup would be achieved if we could convert half of the branch instructions to some ALU instruction? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

**4.28.5** [10] <\$4.8> With the 2-bit predictor, what speedup would be achieved if we could convert half of the branch instructions in a way that replaced each branch instruction with two ALU instructions? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

**4.28.6** [10] <\$4.8> Some branch instructions are much more predictable than others. If we know that 80% of all executed branch instructions are easy-to-predict loop-back branches that are always predicted correctly, what is the accuracy of the 2-bit predictor on the remaining 20% of the branch instructions?

**4.29** This exercise examines the accuracy of various branch predictors for the following repeating pattern (e.g., in a loop) of branch outcomes: T, NT, T, T, NT.

**4.29.1** [5] <\$4.8> What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

**4.29.2** [5] <\$4.8> What is the accuracy of the 2-bit predictor for the first four branches in this pattern, assuming that the predictor starts off in the bottom left state from [Figure 4.61](#) (predict not taken)?

**4.29.3** [10] <\$4.8> What is the accuracy of the 2-bit predictor if this pattern is repeated forever?

**4.29.4** [30] <\$4.8> Design a predictor that would achieve a perfect accuracy if this pattern is repeated forever. Your predictor should be a sequential circuit with one output that provides a prediction (1 for taken, 0 for not taken) and no inputs other than the clock and the control signal that indicates that the instruction is a conditional branch.

**4.29.5** [10] <§4.8> What is the accuracy of your predictor from 4.29.4 if it is given a repeating pattern that is the exact opposite of this one?

**4.29.6** [20] <§4.8> Repeat 4.29.4, but now your predictor should be able to eventually (after a warm-up period during which it can make wrong predictions) start perfectly predicting both this pattern and its opposite. Your predictor should have an input that tells it what the real outcome was. Hint: this input lets your predictor determine which of the two repeating patterns it is given.

**4.30** This exercise explores how exception handling affects pipeline design. The first three problems in this exercise refer to the following two instructions:

Instruction 1	Instruction 2
beqz x11, LABEL	ld x11, 0(x12)

**4.30.1** [5] <§4.9> Which exceptions can each of these instructions trigger? For each of these exceptions, specify the pipeline stage in which it is detected.

**4.30.2** [10] <§4.9> If there is a separate handler address for each exception, show how the pipeline organization must be changed to be able to handle this exception. You can assume that the addresses of these handlers are known when the processor is designed.

**4.30.3** [10] <§4.9> If the second instruction is fetched immediately after the first instruction, describe what happens in the pipeline when the first instruction causes the first exception you listed in Exercise 4.30.1. Show the pipeline execution diagram from the time the first instruction is fetched until the time the first instruction of the exception handler is completed.

**4.30.4** [20] <§4.9> In vectored exception handling, the table of exception handler addresses is in data memory at a known (fixed) address. Change the pipeline to implement this exception handling mechanism. Repeat Exercise 4.30.3 using this modified pipeline and vectored exception handling.

**4.30.5** [15] <§4.9> We want to emulate vectored exception handling (described in Exercise 4.30.4) on a machine that has only one fixed handler address. Write the code that should be at that fixed address. Hint: this code should identify the exception, get the right address from the exception vector table, and transfer execution to that handler.

**4.31** In this exercise we compare the performance of 1-issue and 2-issue processors, taking into account program transformations that can be made to optimize for 2-issue execution. Problems in this exercise refer to the following loop (written in C):

```
for(i=0; i!=j; i+=2)
    b[i]=a[i]-a[i+1];
```

A compiler doing little or no optimization might produce the following RISC-V assembly code:

```

        li    x12, 0
        jal   ENT
TOP:     slli   x5, x12, 3
        add   x6, x10, x5
        ld    x7, 0(x6)
        ld    x29, 8(x6)
        sub   x30, x7, x29
        add   x31, x11, x5
        sd    x30, 0(x31)
        addi  x12, x12, 2
ENT:     bne   x12, x13, TOP
```

The code above uses the following registers:

i	j	a	b	Temporary values
x12	x13	x10	x11	x5–x7, x29–x31

Assume the two-issue, statically scheduled processor for this exercise has the following properties:

- 1. One instruction must be a memory operation; the other must be an arithmetic/logic instruction or a branch.
- 2. The processor has all possible forwarding paths between stages (including paths to the ID stage for branch resolution).
- 3. The processor has perfect branch prediction.
- 4. Two instruction may not issue together in a packet if one depends on the other. (See page 324.)
- 5. If a stall is necessary, both instructions in the issue packet must stall. (See page 324.)

As you complete these exercises, notice how much effort goes into generating code that will produce a near-optimal speedup.

**4.31.1** [30] <§4.10> Draw a pipeline diagram showing how RISC-V code given above executes on the two-issue processor. Assume that the loop exits after two iterations.

**4.31.2** [10] <§4.10> What is the speedup of going from a one-issue to a two-issue processor? (Assume the loop runs thousands of iterations.)

**4.31.3** [10] <§4.10> Rearrange/rewrite the RISC-V code given above to achieve better performance on the one-issue processor. Hint: Use the instruction “beqz x13,DONE” to skip the loop entirely if  $j = 0$ .

**4.31.4** [20] <§4.10> Rearrange/rewrite the RISC-V code given above to achieve better performance on the two-issue processor. (Do not unroll the loop, however.)

**4.31.5** [30] <§4.10> Repeat Exercise 4.31.1, but this time use your optimized code from Exercise 4.31.4.

**4.31.6** [10] <§4.10> What is the speedup of going from a one-issue processor to a two-issue processor when running the optimized code from Exercises 4.31.3 and 4.31.4.

**4.31.7** [10] <§4.10> Unroll the RISC-V code from Exercise 4.31.3 so that each iteration of the unrolled loop handles two iterations of the original loop. Then, rearrange/rewrite your unrolled code to achieve better performance on the one-issue processor. You may assume that  $j$  is a multiple of 4.

**4.31.8** [20] <§4.10> Unroll the RISC-V code from Exercise 4.31.4 so that each iteration of the unrolled loop handles two iterations of the original loop. Then, rearrange/rewrite your unrolled code to achieve better performance on the two-issue processor. You may assume that  $j$  is a multiple of 4. (Hint: Re-organize the loop so that some calculations appear both outside the loop and at the end of the loop. You may assume that the values in temporary registers are not needed after the loop.)

**4.31.9** [10] <§4.10> What is the speedup of going from a one-issue processor to a two-issue processor when running the unrolled, optimized code from Exercises 4.31.7 and 4.31.8?

**4.31.10** [30] <§4.10> Repeat Exercises 4.31.8 and 4.31.9, but this time assume the two-issue processor can run two arithmetic/logic instructions together. (In other words, the first instruction in a packet can be any type of instruction, but the second must be an arithmetic or logic instruction. Two memory operations cannot be scheduled at the same time.)

**4.32** This exercise explores energy efficiency and its relationship with performance. Problems in this exercise assume the following energy consumption for activity in Instruction memory, Registers, and Data memory. You can assume that the other components of the datapath consume a negligible amount of energy. (“Register Read” and “Register Write” refer to the register file only.)

I-Mem	1 Register Read	Register Write	D-Mem Read	D-Mem Write
140pJ	70pJ	60pJ	140pJ	120pJ

Assume that components in the datapath have the following latencies. You can assume that the other components of the datapath have negligible latencies.

I-Mem	Control	Register Read or Write	ALU	D-Mem Read or Write
200ps	150ps	90ps	90ps	250ps

**4.32.1** [5] <§§4.3, 4.6, 4.14> How much energy is spent to execute an `add` instruction in a single-cycle design and in the five-stage pipelined design?

**4.32.2** [10] <§§4.6, 4.14> What is the worst-case RISC-V instruction in terms of energy consumption? What is the energy spent to execute it?

**4.32.3** [10] <§§4.6, 4.14> If energy reduction is paramount, how would you change the pipelined design? What is the percentage reduction in the energy spent by an `ld` instruction after this change?

**4.32.4** [10] <§§4.6, 4.14> What other instructions can potentially benefit from the change discussed in Exercise 4.32.3?

**4.32.5** [10] <§§4.6, 4.14> How do your changes from Exercise 4.32.3 affect the performance of a pipelined CPU?

**4.32.6** [10] <§§4.6, 4.14> We can eliminate the `MemRead` control signal and have the data memory be read in every cycle, i.e., we can permanently have `MemRead=1`. Explain why the processor still functions correctly after this change. If 25% of instructions are loads, what is the effect of this change on clock frequency and energy consumption?

**4.33** When silicon chips are fabricated, defects in materials (e.g., silicon) and manufacturing errors can result in defective circuits. A very common defect is for one wire to affect the signal in another. This is called a “cross-talk fault”. A special class of cross-talk faults is when a signal is connected to a wire that has a constant logical value (e.g., a power supply wire). These faults, where the affected signal always has a logical value of either 0 or 1 are called “stuck-at-0” or “stuck-at-1” faults. The following problems refer to bit 0 of the Write Register input on the register file in [Figure 4.21](#).

**4.33.1** [10] <§§4.3, 4.4> Let us assume that processor testing is done by (1) filling the PC, registers, and data and instruction memories with some values (you can choose which values), (2) letting a single instruction execute, then (3) reading the PC, memories, and registers. These values are then examined to determine if a particular fault is present. Can you design a test (values for PC, memories, and registers) that would determine if there is a stuck-at-0 fault on this signal?

**4.33.2** [10] <§§4.3, 4.4> Repeat Exercise 4.33.1 for a stuck-at-1 fault. Can you use a single test for both stuck-at-0 and stuck-at-1? If yes, explain how; if no, explain why not.

**4.33.3** [10] <§§4.3, 4.4> If we know that the processor has a stuck-at-1 fault on this signal, is the processor still usable? To be usable, we must be able to convert any program that executes on a normal RISC-V processor into a program that works on this processor. You can assume that there is enough free instruction memory and data memory to let you make the program longer and store additional data.

**4.33.4** [10] <§§4.3, 4.4> Repeat Exercise 4.33.1; but now the fault to test for is whether the MemRead control signal becomes 0 if the branch control signal is 0, no fault otherwise.

**4.33.5** [10] <§§4.3, 4.4> Repeat Exercise 4.33.1; but now the fault to test for is whether the MemRead control signal becomes 1 if RegRd control signal is 1, no fault otherwise. Hint: This problem requires knowledge of operating systems. Consider what causes segmentation faults.

§4.1, page 240: 3 of 5: Control, Datapath, Memory. Input and Output are missing.  
 §4.2, page 243: false. Edge-triggered state elements make simultaneous reading and writing both possible and unambiguous.  
 §4.3, page 250: I. a. II. c.  
 §4.4, page 262: Yes, Branch and ALUOp0 are identical. In addition, you can use the flexibility of the don't care bits to combine other signals together. ALUSrc and MemtoReg can be made the same by setting the two don't care bits of MemtoReg to 1 and 0. ALUOp1 and MemtoReg can be made to be inverses of one another by setting the don't care bit of MemtoReg to 1. You don't need an inverter; simply use the other signal and flip the order of the inputs to the MemtoReg multiplexor!  
 §4.5, page 275: 1. Stall due to a load-use data hazard of the `ld` result. 2. Avoid stalling in the third instruction for the read-after-write data hazard on `x11` by forwarding the `add` result. 3. It need not stall, even without forwarding.  
 §4.6, page 288: Statements 2 and 4 are correct; the rest are incorrect.  
 §4.8, page 314: 1. Predict not taken. 2. Predict taken. 3. Dynamic prediction.  
 §4.9, page 321: The first instruction, since it is logically executed before the others.  
 §4.10, page 334: 1. Both. 2. Both. 3. Software. 4. Hardware. 5. Hardware. 6. Hardware. 7. Both. 8. Hardware. 9. Both.  
 §4.12, page 344: First two are false and the last two are true.

**Answers to  
Check Yourself**

As we will see in [Chapter 6](#), memory systems are a central design issue for parallel processors. The growing significance of the memory hierarchy in determining system performance means that this important area will continue to be a focus for both designers and researchers for some years to come.



## Historical Perspective and Further Reading

This section, which appears online, gives an overview of memory technologies, from mercury delay lines to DRAM, the invention of the memory hierarchy, protection mechanisms, and virtual machines, and concludes with a brief history of operating systems, including CTSS, MULTICS, UNIX, BSD UNIX, MS-DOS, Windows, and Linux.

## 5.19

## Exercises

Assume memory is byte addressable and words are 64 bits, unless specified otherwise.

**5.1** In this exercise we look at memory locality properties of matrix computation. The following code is written in C, where elements within the same row are stored contiguously. Assume each word is a 64-bit integer.

```
for (I=0; I<8; I++)
    for (J=0; J<8000; J++)
        A[I][J]=B[I][0]+A[J][I];
```

**5.1.1** [5] <§5.1> How many 64-bit integers can be stored in a 16-byte cache block?

**5.1.2** [5] <§5.1> Which variable references exhibit temporal locality?

**5.1.3** [5] <§5.1> Which variable references exhibit spatial locality?

Locality is affected by both the reference order and data layout. The same computation can also be written below in Matlab, which differs from C in that it stores matrix elements within the same column contiguously in memory.

```
for I=1:8
    for J=1:8000
        A(I,J)=B(I,0)+A(J,I);
    end
end
```



**5.1.4** [5] <§5.1> Which variable references exhibit temporal locality?

**5.1.5** [5] <§5.1> Which variable references exhibit spatial locality?

**5.1.6** [15] <§5.1> How many 16-byte cache blocks are needed to store all 64-bit matrix elements being referenced using Matlab's matrix storage? How many using C's matrix storage? (Assume each row contains more than one element.)

**5.2** Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 64-bit memory address references, given as word addresses.

0x03, 0xb4, 0x2b, 0x02, 0xbf, 0x58, 0xbe, 0x0e, 0xb5,  
0x2c, 0xba, 0xfd

**5.2.1** [10] <§5.3> For each of these references, identify the binary word address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list whether each reference is a hit or a miss, assuming the cache is initially empty.

**5.2.2** [10] <§5.3> For each of these references, identify the binary word address, the tag, the index, and the offset given a direct-mapped cache with two-word blocks and a total size of eight blocks. Also list if each reference is a hit or a miss, assuming the cache is initially empty.

**5.2.3** [20] <§§5.3, 5.4> You are asked to optimize a cache design for the given references. There are three direct-mapped cache designs possible, all with a total of eight words of data:

- C1 has 1-word blocks,
- C2 has 2-word blocks, and
- C3 has 4-word blocks.

**5.3** By convention, a cache is named according to the amount of data it contains (i.e., a 4 KiB cache can hold 4 KiB of data); however, caches also require SRAM to store metadata such as tags and valid bits. For this exercise, you will examine how a cache's configuration affects the total amount of SRAM needed to implement it as well as the performance of the cache. For all parts, assume that the caches are byte addressable, and that addresses and words are 64 bits.

**5.3.1** [10] <§5.3> Calculate the total number of bits required to implement a 32 KiB cache with two-word blocks.

**5.3.2** [10] <§5.3> Calculate the total number of bits required to implement a 64 KiB cache with 16-word blocks. How much bigger is this cache than the 32 KiB cache described in Exercise 5.3.1? (Notice that, by changing the block size, we doubled the amount of data without doubling the total size of the cache.)

**5.3.3** [5] <§5.3> Explain why this 64 KiB cache, despite its larger data size, might provide slower performance than the first cache.

**5.3.4** [10] <§§5.3, 5.4> Generate a series of read requests that have a lower miss rate on a 32 KiB two-way set associative cache than on the cache described in Exercise 5.3.1.

**5.4** [15] <§5.3> Section 5.3 shows the typical method to index a direct-mapped cache, specifically (Block address) modulo (Number of blocks in the cache). Assuming a 64-bit address and 1024 blocks in the cache, consider a different indexing function, specifically (Block address[63:54] XOR Block address[53:44]). Is it possible to use this to index a direct-mapped cache? If so, explain why and discuss any changes that might need to be made to the cache. If it is not possible, explain why.

**5.5** For a direct-mapped cache design with a 64-bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset
63–10	9–5	4–0

**5.5.1** [5] <§5.3> What is the cache block size (in words)?

**5.5.2** [5] <§5.3> How many blocks does the cache have?

**5.5.3** [5] <§5.3> What is the ratio between total bits required for such a cache implementation over the data storage bits?

Beginning from power on, the following byte-addressed cache references are recorded.

Address												
Hex	00	04	10	84	E8	A0	400	1E	8C	C1C	B4	884
Dec	0	4	16	132	232	160	1024	30	140	3100	180	2180

**5.5.4** [20] <§5.3> For each reference, list (1) its tag, index, and offset, (2) whether it is a hit or a miss, and (3) which bytes were replaced (if any).

**5.5.5** [5] <§5.3> What is the hit ratio?

**5.5.6** [5] <§5.3> List the final state of the cache, with each valid entry represented as a record of <index, tag, data>. For example,

<0, 3, Mem[0xC00]–Mem[0xC1F]>

**5.6** Recall that we have two write policies and two write allocation policies, and their combinations can be implemented either in L1 or L2 cache. Assume the following choices for L1 and L2 caches:

L1	L2
Write through, non-write allocate	Write back, write allocate

**5.6.1** [5] <§§5.3, 5.8> Buffers are employed between different levels of memory hierarchy to reduce access latency. For this given configuration, list the possible buffers needed between L1 and L2 caches, as well as L2 cache and memory.

**5.6.2** [20] <§§5.3, 5.8> Describe the procedure of handling an L1 write-miss, considering the components involved and the possibility of replacing a dirty block.

**5.6.3** [20] <§§5.3, 5.8> For a multilevel exclusive cache configuration (a block can only reside in one of the L1 and L2 caches), describe the procedures of handling an L1 write-miss and an L1 read-miss, considering the components involved and the possibility of replacing a dirty block.

**5.7** Consider the following program and cache behaviors.

Data Reads per 1000 Instructions	Data Writes per 1000 Instructions	Instruction Cache Miss Rate	Data Cache Miss Rate	Block Size (bytes)
250	100	0.30%	2%	64

**5.7.1** [10] <§§5.3, 5.8> Suppose a CPU with a write-through, write-allocate cache achieves a CPI of 2. What are the read and write bandwidths (measured by bytes per cycle) between RAM and the cache? (Assume each miss generates a request for one block.)

**5.7.2** [10] <§§5.3, 5.8> For a write-back, write-allocate cache, assuming 30% of replaced data cache blocks are dirty, what are the read and write bandwidths needed for a CPI of 2?

**5.8** Media applications that play audio or video files are part of a class of workloads called “streaming” workloads (i.e., they bring in large amounts of data but do not reuse much of it). Consider a video streaming workload that accesses a 512 KiB working set sequentially with the following word address stream:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9 . . .

**5.8.1** [10] <§§5.4, 5.8> Assume a 64 KiB direct-mapped cache with a 32-byte block. What is the miss rate for the address stream above? How is this miss rate sensitive to the size of the cache or the working set? How would you categorize the misses this workload is experiencing, based on the 3C model?

**5.8.2** [5] <§§5.1, 5.8> Re-compute the miss rate when the cache block size is 16 bytes, 64 bytes, and 128 bytes. What kind of locality is this workload exploiting?

**5.8.3** [10] <§5.13> “Prefetching” is a technique that leverages predictable address patterns to speculatively bring in additional cache blocks when a particular cache block is accessed. One example of prefetching is a stream buffer that prefetches sequentially adjacent cache blocks into a separate buffer when a particular cache block is brought in. If the data are found in the prefetch buffer, it is considered

as a hit, moved into the cache, and the next cache block is prefetched. Assume a two-entry stream buffer; and, assume that the cache latency is such that a cache block can be loaded before the computation on the previous cache block is completed. What is the miss rate for the address stream above?

**5.9** Cache block size ( $B$ ) can affect both miss rate and miss latency. Assuming a machine with a base CPI of 1, and an average of 1.35 references (both instruction and data) per instruction, find the block size that minimizes the total miss latency given the following miss rates for various block sizes.

8: 4%	16: 3%	32: 2%	64: 1.5%	128: 1%
-------	--------	--------	----------	---------

**5.9.1** [10] <§5.3> What is the optimal block size for a miss latency of  $20 \times B$  cycles?

**5.9.2** [10] <§5.3> What is the optimal block size for a miss latency of  $24 + B$  cycles?

**5.9.3** [10] <§5.3> For constant miss latency, what is the optimal block size?

**5.10** In this exercise, we will look at the different ways capacity affects overall performance. In general, cache access time is proportional to capacity. Assume that main memory accesses take 70 ns and that 36% of all instructions access data memory. The following table shows data for L1 caches attached to each of two processors, P1 and P2.

	L1 Size	L1 Miss Rate	L1 Hit Time
P1	2 KiB	8.0%	0.66 ns
P2	4 KiB	6.0%	0.90 ns

**5.10.1** [5] <§5.4> Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

**5.10.2** [10] <§5.4> What is the Average Memory Access Time for P1 and P2 (in cycles)?

**5.10.3** [5] <§5.4> Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 and P2? Which processor is faster? (When we say a “base CPI of 1.0”, we mean that instructions complete in one cycle, unless either the instruction access or the data access causes a cache miss.)

For the next three problems, we will consider the addition of an L2 cache to P1 (to presumably make up for its limited L1 cache capacity). Use the L1 cache capacities and hit times from the previous table when solving these problems. The L2 miss rate indicated is its local miss rate.

L2 Size	L2 Miss Rate	L2 Hit Time
1 MiB	95%	5.62 ns

**5.10.4** [10] <\$5.4> What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

**5.10.5** [5] <\$5.4> Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 with the addition of an L2 cache?

**5.10.6** [10] <\$5.4> What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P1 without an L2 cache?

**5.10.7** [15] <\$5.4> What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P2 without an L2 cache?

**5.11** This exercise examines the effect of different cache designs, specifically comparing associative caches to the direct-mapped caches from [Section 5.4](#). For these exercises, refer to the sequence of word address shown below.

```
0x03, 0xb4, 0x2b, 0x02, 0xbe, 0x58, 0xbf, 0x0e, 0x1f,
0xb5, 0xbf, 0xba, 0x2e, 0xce
```

**5.11.1** [10] <\$5.4> Sketch the organization of a three-way set associative cache with two-word blocks and a total size of 48 words. Your sketch should have a style similar to [Figure 5.18](#), but clearly show the width of the tag and data fields.

**5.11.2** [10] <\$5.4> Trace the behavior of the cache from Exercise 5.11.1. Assume a true LRU replacement policy. For each reference, identify

- the binary word address,
- the tag,
- the index,
- the offset
- whether the reference is a hit or a miss, and
- which tags are in each way of the cache after the reference has been handled.

**5.11.3** [5] <\$5.4> Sketch the organization of a fully associative cache with one-word blocks and a total size of eight words. Your sketch should have a style similar to [Figure 5.18](#), but clearly show the width of the tag and data fields.

**5.11.4** [10] <\$5.4> Trace the behavior of the cache from Exercise 5.11.3. Assume a true LRU replacement policy. For each reference, identify

- the binary word address,
- the tag,
- the index,
- the offset,

- whether the reference is a hit or a miss, and
- the contents of the cache after each reference has been handled.

**5.11.5** [5] <§5.4> Sketch the organization of a fully associative cache with two-word blocks and a total size of eight words. Your sketch should have a style similar to Figure 5.18, but clearly show the width of the tag and data fields.

**5.11.6** [10] <§5.4> Trace the behavior of the cache from Exercise 5.11.5. Assume an LRU replacement policy. For each reference, identify

- the binary word address,
- the tag,
- the index,
- the offset,
- whether the reference is a hit or a miss, and
- the contents of the cache after each reference has been handled.

**5.11.7** [10] <§5.4> Repeat Exercise 5.11.6 using MRU (*most recently used*) replacement.

**5.11.8** [15] <§5.4> Repeat Exercise 5.11.6 using the optimal replacement policy (i.e., the one that gives the lowest miss rate).

**5.12** Multilevel caching is an important technique to overcome the limited amount of space that a first-level cache can provide while still maintaining its speed. Consider a processor with the following parameters:

Base CPI, No Memory Stalls	Processor Speed	Main Memory Access Time	First-Level Cache Miss Rate per Instruction**	Second-Level Cache, Direct-Mapped Speed	Miss Rate with Second-Level Cache, Direct-Mapped	Second-Level Cache, Eight-Way Set Associative Speed	Miss Rate with Second-Level Cache, Eight-Way Set Associative
1.5	2 GHz	100 ns	7%	12 cycles	3.5%	28 cycles	1.5%

\*\*First Level Cache miss rate is per instruction. Assume the total number of L1 cache misses (instruction and data combined) is equal to 7% of the number of instructions.

**5.12.1** [10] <§5.4> Calculate the CPI for the processor in the table using: 1) only a first-level cache, 2) a second-level direct-mapped cache, and 3) a second-level eight-way set associative cache. How do these numbers change if main memory access time doubles? (Give each change as both an absolute CPI and a percent change.) Notice the extent to which an L2 cache can hide the effects of a slow memory.

**5.12.2** [10] <\$5.4> It is possible to have an even greater cache hierarchy than two levels? Given the processor above with a second-level, direct-mapped cache, a designer wants to add a third-level cache that takes 50 cycles to access and will have a 13% miss rate. Would this provide better performance? In general, what are the advantages and disadvantages of adding a third-level cache?

**5.12.3** [20] <\$5.4> In older processors, such as the Intel Pentium or Alpha 21264, the second level of cache was external (located on a different chip) from the main processor and the first-level cache. While this allowed for large second-level caches, the latency to access the cache was much higher, and the bandwidth was typically lower because the second-level cache ran at a lower frequency. Assume a 512 KiB off-chip second-level cache has a miss rate of 4%. If each additional 512 KiB of cache lowered miss rates by 0.7%, and the cache had a total access time of 50 cycles, how big would the cache have to be to match the performance of the second-level direct-mapped cache listed above?

**5.13** *Mean time between failures (MTBF), mean time to replacement (MTTR), and mean time to failure (MTTF)* are useful metrics for evaluating the reliability and availability of a storage resource. Explore these concepts by answering the questions about a device with the following metrics:

MTTF	MTTR
3 Years	1 Day

**5.13.1** [5] <\$5.5> Calculate the MTBF for such a device.

**5.13.2** [5] <\$5.5> Calculate the availability for such a device.

**5.13.3** [5] <\$5.5> What happens to availability as the MTTR approaches 0? Is this a realistic situation?

**5.13.4** [5] <\$5.5> What happens to availability as the MTTR gets very high, i.e., a device is difficult to repair? Does this imply the device has low availability?

**5.14** This exercise examines the *single error correcting, double error detecting (SEC/DED) Hamming code*.

**5.14.1** [5] <\$5.5> What is the minimum number of parity bits required to protect a 128-bit word using the SEC/DED code?

**5.14.2** [5] <\$5.5> [Section 5.5](#) states that modern server memory modules (DIMMs) employ SEC/DED ECC to protect each 64 bits with 8 parity bits. Compute the cost/performance ratio of this code to the code from Exercise 5.14.1. In this case, cost is the relative number of parity bits needed while performance is the relative number of errors that can be corrected. Which is better?

**5.14.3** [5] <\$5.5> Consider a SEC code that protects 8 bit words with 4 parity bits. If we read the value 0x375, is there an error? If so, correct the error.

**5.15** For a high-performance system such as a B-tree index for a database, the page size is determined mainly by the data size and disk performance. Assume that, on average, a B-tree index page is 70% full with fix-sized entries. The utility of a page is its B-tree depth, calculated as  $\log_2(\text{entries})$ . The following table shows that for 16-byte entries, and a 10-year-old disk with a 10 ms latency and 10 MB/s transfer rate, the optimal page size is 16 K.

Page Size (KiB)	Page Utility or B-Tree Depth (Number of Disk Accesses Saved)	Index Page Access Cost (ms)	Utility/Cost
2	6.49 (or $\log_2(2048/16 \times 0.7)$ )	10.2	0.64
4	7.49	10.4	0.72
8	8.49	10.8	0.79
16	9.49	11.6	0.82
32	10.49	13.2	0.79
64	11.49	16.4	0.70
128	12.49	22.8	0.55
256	13.49	35.6	0.38

**5.15.1** [10] <\$5.7> What is the best page size if entries now become 128 bytes?

**5.15.2** [10] <\$5.7> Based on Exercise 5.15.1, what is the best page size if pages are half full?

**5.15.3** [20] <\$5.7> Based on Exercise 5.15.2, what is the best page size if using a modern disk with a 3 ms latency and 100 MB/s transfer rate? Explain why future servers are likely to have larger pages.

Keeping “frequently used” (or “hot”) pages in DRAM can save disk accesses, but how do we determine the exact meaning of “frequently used” for a given system? Data engineers use the cost ratio between DRAM and disk access to quantify the reuse time threshold for hot pages. The cost of a disk access is \$Disk/accesses\_per\_sec, while the cost to keep a page in DRAM is \$DRAM\_MiB/page\_size. The typical DRAM and disk costs and typical database page sizes at several time points are listed below:

Year	DRAM Cost (\$/MiB)	Page Size (KiB)	Disk Cost (\$/disk)	Disk Access Rate (access/sec)
1987	5000	1	15,000	15
1997	15	8	2000	64
2007	0.05	64	80	83

**5.15.4** [20] <\$5.7> What other factors can be changed to keep using the same page size (thus avoiding software rewrite)? Discuss their likeliness with current technology and cost trends.

**5.16** As described in [Section 5.7](#), virtual memory uses a page table to track the mapping of virtual addresses to physical addresses. This exercise shows how this



table must be updated as addresses are accessed. The following data constitute a stream of virtual byte addresses as seen on a system. Assume 4 KiB pages, a four-entry fully associative TLB, and true LRU replacement. If pages must be brought in from disk, increment the next largest page number.

Decimal	4669	2227	13916	34587	48870	12608	49225
hex	0x123d	0x08b3	0x365c	0x871b	0xbee6	0x3140	0xc049

TLB

Valid	Tag	Physical Page Number	Time Since Last Access
1	0xb	12	4
1	0x7	4	1
1	0x3	6	3
0	0x4	9	7

Page table

Index	Valid	Physical Page or in Disk
0	1	5
1	0	Disk
2	0	Disk
3	1	6
4	1	9
5	1	11
6	0	Disk
7	1	4
8	0	Disk
9	0	Disk
a	1	3
b	1	12

**5.16.1** [10] <§5.7> For each access shown above, list

- whether the access is a hit or miss in the TLB,
- whether the access is a hit or miss in the page table,
- whether the access is a page fault,
- the updated state of the TLB.

**5.16.2** [15] <§5.7> Repeat Exercise 5.16.1, but this time use 16 KiB pages instead of 4 KiB pages. What would be some of the advantages of having a larger page size? What are some of the disadvantages?

**5.16.3** [15] <§5.7> Repeat Exercise 5.16.1, but this time use 4 KiB pages and a two-way set associative TLB.

**5.16.4** [15] <§5.7> Repeat Exercise 5.16.1, but this time use 4 KiB pages and a direct mapped TLB.

**5.16.5** [10] <§§5.4, 5.7> Discuss why a CPU must have a TLB for high performance. How would virtual memory accesses be handled if there were no TLB?

**5.17** There are several parameters that affect the overall size of the page table. Listed below are key page table parameters.

Virtual Address Size	Page Size	Page Table Entry Size
32 bits	8 KiB	4 bytes

**5.17.1** [5] <§5.7> Given the parameters shown above, calculate the maximum possible page table size for a system running five processes.

**5.17.2** [10] <§5.7> Given the parameters shown above, calculate the total page table size for a system running five applications that each utilize half of the virtual memory available, given a two-level page table approach with up to 256 entries at the 1<sup>st</sup> level. Assume each entry of the main page table is 6 bytes. Calculate the minimum and maximum amount of memory required for this page table.

**5.17.3** [10] <§5.7> A cache designer wants to increase the size of a 4 KiB virtually indexed, physically tagged cache. Given the page size shown above, is it possible to make a 16 KiB direct-mapped cache, assuming two 64-bit words per block? How would the designer increase the data size of the cache?

**5.18** In this exercise, we will examine space/time optimizations for page tables. The following list provides parameters of a virtual memory system.

Virtual Address (bits)	Physical DRAM Installed	Page Size	PTE Size (byte)
43	16 GiB	4 KiB	4

**5.18.1** [10] <§5.7> For a single-level page table, how many *page table entries* (PTEs) are needed? How much physical memory is needed for storing the page table?

**5.18.2** [10] <§5.7> Using a multi-level page table can reduce the physical memory consumption of page tables by only keeping active PTEs in physical memory. How many levels of page tables will be needed if the segment tables (the upper-level page tables) are allowed to be of unlimited size? How many memory references are needed for address translation if missing in TLB?

**5.18.3** [10] <§5.7> Suppose the segments are limited to the 4 KiB page size (so that they can be paged). Is 4 bytes large enough for all page table entries (including those in the segment tables)?

**5.18.4** [10] <§5.7> How many levels of page tables are needed if the segments are limited to the 4 KiB page size?

**5.18.5** [15] <§5.7> An inverted page table can be used to further optimize space and time. How many PTEs are needed to store the page table? Assuming a hash table implementation, what are the common case and worst case numbers of memory references needed for servicing a TLB miss?

**5.19** The following table shows the contents of a four-entry TLB.

Entry-ID	Valid	VA Page	Modified	Protection	PA Page
1	1	140	1	RW	30
2	0	40	0	RX	34
3	1	200	1	RO	32
4	1	280	0	RW	31

**5.19.1** [5] <§5.7> Under what scenarios would entry 3's valid bit be set to zero?

**5.19.2** [5] <§5.7> What happens when an instruction writes to VA page 30? When would a software managed TLB be faster than a hardware managed TLB?

**5.19.3** [5] <§5.7> What happens when an instruction writes to VA page 200?

**5.20** In this exercise, we will examine how replacement policies affect miss rate. Assume a two-way set associative cache with four one-word blocks. Consider the following word address sequence: 0, 1, 2, 3, 4, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0.

Consider the following address sequence: 0, 2, 4, 8, 10, 12, 14, 16, 0

**5.20.1** [5] <§§5.4, 5.8> Assuming an LRU replacement policy, which accesses are hits?

**5.20.2** [5] <§§5.4, 5.8> Assuming an MRU (*most recently used*) replacement policy, which accesses are hits?

**5.20.3** [5] <§§5.4, 5.8> Simulate a random replacement policy by flipping a coin. For example, "heads" means to evict the first block in a set and "tails" means to evict the second block in a set. How many hits does this address sequence exhibit?

**5.20.4** [10] <§§5.4, 5.8> Describe an optimal replacement policy for this sequence. Which accesses are hits using this policy?

**5.20.5** [10] <§§5.4, 5.8> Describe why it is difficult to implement a cache replacement policy that is optimal for all address sequences.

**5.20.6** [10] <§§5.4, 5.8> Assume you could make a decision upon each memory reference whether or not you want the requested address to be cached. What effect could this have on miss rate?

**5.21** One of the biggest impediments to widespread use of virtual machines is the performance overhead incurred by running a virtual machine. Listed below are various performance parameters and application behavior.

Base CPI	Privileged O/S accesses per 10,000 instructions	Overhead to trap to the guest O/S	Overhead to trap to VMM	I/O access per 10,000 instructions	I/O access time (includes time to trap to guest O/S)
1.5	120	15 cycles	175 cycles	30	1100 cycles

**5.21.1** [10] <§5.6> Calculate the CPI for the system listed above assuming that there are no accesses to I/O. What is the CPI if the VMM overhead doubles? If it is cut in half? If a virtual machine software company wishes to limit the performance degradation to 10%, what is the longest possible penalty to trap to the VMM?

**5.21.2** [15] <§5.6> I/O accesses often have a large effect on overall system performance. Calculate the CPI of a machine using the performance characteristics above, assuming a non-virtualized system. Calculate the CPI again, this time using a virtualized system. How do these CPIs change if the system has half the I/O accesses?

**5.22** [15] <§§5.6, 5.7> Compare and contrast the ideas of virtual memory and virtual machines. How do the goals of each compare? What are the pros and cons of each? List a few cases where virtual memory is desired, and a few cases where virtual machines are desired.

**5.23** [10] <§5.6> [Section 5.6](#) discusses virtualization under the assumption that the virtualized system is running the same ISA as the underlying hardware. However, one possible use of virtualization is to emulate non-native ISAs. An example of this is QEMU, which emulates a variety of ISAs such as MIPS, SPARC, and PowerPC. What are some of the difficulties involved in this kind of virtualization? Is it possible for an emulated system to run faster than on its native ISA?

**5.24** In this exercise, we will explore the control unit for a cache controller for a processor with a write buffer. Use the finite state machine found in [Figure 5.39](#) as a starting point for designing your own finite state machines. Assume that the cache controller is for the simple direct-mapped cache described on page 453 ([Figure 5.39](#) in [Section 5.9](#)), but you will add a write buffer with a capacity of one block.

Recall that the purpose of a write buffer is to serve as temporary storage so that the processor doesn't have to wait for two memory accesses on a dirty miss. Rather than writing back the dirty block before reading the new block, it buffers the dirty block and immediately begins reading the new block. The dirty block can then be written to main memory while the processor is working.

**5.24.1** [10] <§§5.8, 5.9> What should happen if the processor issues a request that *hits* in the cache while a block is being written back to main memory from the write buffer?

**5.24.2** [10] <§§5.8, 5.9> What should happen if the processor issues a request that *misses* in the cache while a block is being written back to main memory from the write buffer?

**5.24.3** [30] <§§5.8, 5.9> Design a finite state machine to enable the use of a write buffer.

**5.25** Cache coherence concerns the views of multiple processors on a given cache block. The following data show two processors and their read/write operations on two different words of a cache block X (initially  $X[0] = X[1] = 0$ ).

P1	P2
$X[0]++$ ; $X[1] = 3$ ;	$X[0] = 5$ ; $X[1] += 2$ ;

**5.25.1** [15] <§5.10> List the possible values of the given cache block for a correct cache coherence protocol implementation. List at least one more possible value of the block if the protocol doesn’t ensure cache coherency.

**5.25.2** [15] <§5.10> For a snooping protocol, list a valid operation sequence on each processor/cache to finish the above read/write operations.

**5.25.3** [10] <§5.10> What are the best-case and worst-case numbers of cache misses needed to execute the listed read/write instructions?

Memory consistency concerns the views of multiple data items. The following data show two processors and their read/write operations on different cache blocks (A and B initially 0).

P1	P2
$A = 1$ ; $B = 2$ ; $A += 2$ ; $B++$ ;	$C = B$ ; $D = A$ ;

**5.25.4** [15] <§5.10> List the possible values of C and D for all implementations that ensure both consistency assumptions on page 455.

**5.25.5** [15] <§5.10> List at least one more possible pair of values for C and D if such assumptions are not maintained.

**5.25.6** [15] <§§5.3, 5.10> For various combinations of write policies and write allocation policies, which combinations make the protocol implementation simpler?

**5.26** Chip multiprocessors (CMPs) have multiple cores and their caches on a single chip. CMP on-chip L2 cache design has interesting trade-offs. The following

table shows the miss rates and hit latencies for two benchmarks with private vs. shared L2 cache designs. Assume the L1 cache has a 3% miss rate and a 1-cycle access time.

	Private	Shared
Benchmark A miss rate	10%	4%
Benchmark B miss rate	2%	1%

Assume the following hit latencies:

Private Cache	Shared Cache	Memory
5	20	180

**5.26.1** [15] <§5.13> Which cache design is better for each of these benchmarks? Use data to support your conclusion.

**5.26.2** [15] <§5.13> Off-chip bandwidth becomes the bottleneck as the number of CMP cores increases. How does this bottleneck affect private and shared cache systems differently? Choose the best design if the latency of the first off-chip link doubles.

**5.26.3** [10] <§5.13> Discuss the pros and cons of shared vs. private L2 caches for both single-threaded, multi-threaded, and multiprogrammed workloads, and reconsider them if having on-chip L3 caches.

**5.26.4** [10] <§5.13> Would a non-blocking L2 cache produce more improvement on a CMP with a shared L2 cache or a private L2 cache? Why?

**5.26.5** [10] <§5.13> Assume new generations of processors double the number of cores every 18 months. To maintain the same level of per-core performance, how much more off-chip memory bandwidth is needed for a processor released in three years?

**5.26.6** [15] <§5.13> Consider the entire memory hierarchy. What kinds of optimizations can improve the number of concurrent misses?

**5.27** In this exercise we show the definition of a web server log and examine code optimizations to improve log processing speed. The data structure for the log is defined as follows:

```
struct entry {
    int srcIP; // remote IP address
    char URL[128]; // request URL (e.g., "GET index.html")
    long long refTime; // reference time
    int status; // connection status
    char browser[64]; // client browser name
} log [NUM_ENTRIES];
```

Assume the following processing function for the log:

```
topK_sourceIP (int hour);
```

This function determines the most frequently observed source IPs during the given hour.

**5.27.1** [5] <\$5.15> Which fields in a log entry will be accessed for the given log processing function? Assuming 64-byte cache blocks and no prefetching, how many cache misses per entry does the given function incur on average?

**5.27.2** [5] <\$5.15> How can you reorganize the data structure to improve cache utilization and access locality?

**5.27.3** [10] <\$5.15> Give an example of another log processing function that would prefer a different data structure layout. If both functions are important, how would you rewrite the program to improve the overall performance? Supplement the discussion with code snippet and data.

**5.28** For the problems below, use data from “Cache Performance for SPEC CPU2000 Benchmarks” (<http://www.cs.wisc.edu/multifacet/misc/spec2000cache-data/>) for the pairs of benchmarks shown in the following table.

<b>a.</b>	Mesa/gcc
<b>b.</b>	mcf/swim

**5.28.1** [10] <\$5.15> For 64 KiB data caches with varying set associativities, what are the miss rates broken down by miss types (cold, capacity, and conflict misses) for each benchmark?

**5.28.2** [10] <\$5.15> Select the set associativity to be used by a 64 KiB L1 data cache shared by both benchmarks. If the L1 cache has to be directly mapped, select the set associativity for the 1 MiB L2 cache.

**5.28.3** [20] <\$5.15> Give an example in the miss rate table where higher set associativity actually increases miss rate. Construct a cache configuration and reference stream to demonstrate this.

**5.29** To support multiple virtual machines, two levels of memory virtualization are needed. Each virtual machine still controls the mapping of *virtual address* (VA) to *physical address* (PA), while the hypervisor maps the *physical address* (PA) of each virtual machine to the actual *machine address* (MA). To accelerate such mappings, a software approach called “shadow paging” duplicates each virtual machine’s page tables in the hypervisor, and intercepts VA to PA mapping changes to keep both copies consistent. To remove the complexity of shadow page tables, a

hardware approach called *nested page table* (NPT) explicitly supports two classes of page tables ( $VA \Rightarrow PA$  and  $PA \Rightarrow MA$ ) and can walk such tables purely in hardware.

Consider the following sequence of operations: (1) Create process; (2) TLB miss; (3) page fault; (4) context switch;

**5.29.1** [10] <§§5.6, 5.7> What would happen for the given operation sequence for shadow page table and nested page table, respectively?

**5.29.2** [10] <§§5.6, 5.7> Assuming an x86-based four-level page table in both guest and nested page table, how many memory references are needed to service a TLB miss for native vs. nested page table?

**5.29.3** [15] <§§5.6, 5.7> Among TLB miss rate, TLB miss latency, page fault rate, and page fault handler latency, which metrics are more important for shadow page table? Which are important for nested page table?

Assume the following parameters for a shadow paging system.

TLB Misses per 1000 Instructions	NPT TLB Miss Latency	Page Faults per 1000 Instructions	Shadowing Page Fault Overhead
0.2	200 cycles	0.001	30,000 cycles

**5.29.4** [10] <§5.6> For a benchmark with native execution CPI of 1, what are the CPI numbers if using shadow page tables vs. NPT (assuming only page table virtualization overhead)?

**5.29.5** [10] <§5.6> What techniques can be used to reduce page table shadowing induced overhead?

**5.29.6** [10] <§5.6> What techniques can be used to reduce NPT induced overhead?

§5.1, page 369: 1 and 4. (3 is false because the cost of the memory hierarchy varies per computer, but in 2016 the highest cost is usually the DRAM.)

§5.3, page 390: 1 and 4: A lower miss penalty can enable smaller blocks, since you don't have that much latency to amortize, yet higher memory bandwidth usually leads to larger blocks, since the miss penalty is only slightly larger.

§5.4, page 409: 1.

§5.8, page 449: 2. (Both large block sizes and prefetching may reduce compulsory misses, so 1 is false.)

**Answers to  
Check Yourself**