



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 understanding 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 1280×1024 .

- 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 (class 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 implementation is faster?

- 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, had 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 processor 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/cm². Assume a 20 cm diameter wafer has a cost of 15, contains 100 dies, and has 0.031 defects/cm².

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 mm².

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 largest 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

$$\text{MFLOPS} = \text{No. FP operations} / (\text{execution time} \times 1\text{E6})$$

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 programs.

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 50×10^6 FP instructions, 110×10^6 INT instructions, 80×10^6 L/S instructions, and 16×10^6 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**

include accumulator architectures, general-purpose register architectures, stack architectures, and a brief history of ARM and the x86. 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.

2.22 Exercises

Appendix A describes the MIPS simulator, which is helpful for these exercises. Although the simulator accepts pseudoinstructions, try not to use pseudoinstructions for any exercises that ask you to produce MIPS code. Your goal should be to learn the real MIPS instruction set, and if you are asked to count instructions, your count should reflect the actual instructions that will be executed and not the pseudoinstructions.

There are some cases where pseudoinstructions must be used (for example, the `li` instruction when an actual value is not known at assembly time). In many cases, they are quite convenient and result in more readable code (for example, the `li` and `move` instructions). If you choose to use pseudoinstructions for these reasons, please add a sentence or two to your solution stating which pseudoinstructions you have used and why.

2.1 [5] <§2.2> For the following C statement, what is the corresponding MIPS assembly code? Assume that the variables `f`, `g`, `h`, and `i` are given and could be considered 32-bit integers as declared in a C program. Use a minimal number of MIPS assembly instructions.

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

2.2 [5] <§2.2> For the following MIPS assembly instructions above, what is a corresponding C statement?

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

2.3 [5] <§§2.2, 2.3> For the following C statement, what is the corresponding MIPS assembly code? Assume that the variables *f*, *g*, *h*, *i*, and *j* are assigned to registers *\$s0*, *\$s1*, *\$s2*, *\$s3*, and *\$s4*, respectively. Assume that the base address of the arrays *A* and *B* are in registers *\$s6* and *\$s7*, respectively.

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

2.4 [5] <§§2.2, 2.3> For the MIPS assembly instructions below, what is the corresponding C statement? Assume that the variables *f*, *g*, *h*, *i*, and *j* are assigned to registers *\$s0*, *\$s1*, *\$s2*, *\$s3*, and *\$s4*, respectively. Assume that the base address of the arrays *A* and *B* are in registers *\$s6* and *\$s7*, respectively.

```
sll  $t0, $s0, 2      # $t0 = f * 4
add  $t0, $s6, $t0    # $t0 = &A[f]
sll  $t1, $s1, 2      # $t1 = g * 4
add  $t1, $s7, $t1    # $t1 = &B[g]
lw   $s0, 0($t0)      # f = A[f]
addi $t2, $t0, 4
lw   $t0, 0($t2)
add  $t0, $t0, $s0
sw   $t0, 0($t1)
```

2.5 [5] <§§2.2, 2.3> For the MIPS assembly instructions in Exercise 2.4, rewrite the assembly code to minimize the number of MIPS instructions (if possible) needed to carry out the same function.

2.6 The table below shows 32-bit values of an array stored in memory.

Address	Data
24	2
38	4
32	3
36	6
40	1

2.6.1 [5] <§§2.2, 2.3> For the memory locations in the table above, write C code to sort the data from lowest to highest, placing the lowest value in the smallest memory location shown in the figure. Assume that the data shown represents the C variable called `Array`, which is an array of type `int`, and that the first number in the array shown is the first element in the array. Assume that this particular machine is a byte-addressable machine and a word consists of four bytes.

2.6.2 [5] <§§2.2, 2.3> For the memory locations in the table above, write MIPS code to sort the data from lowest to highest, placing the lowest value in the smallest memory location. Use a minimum number of MIPS instructions. Assume the base address of `Array` is stored in register `$s6`.

2.7 [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 is stored starting at address 0.

2.8 [5] <§2.4> Translate `0xabcdef12` into decimal.

2.9 [5] <§§2.2, 2.3> Translate the following C code to MIPS. Assume that the variables `f`, `g`, `h`, `i`, and `j` are assigned to registers `$s0`, `$s1`, `$s2`, `$s3`, and `$s4`, respectively. Assume that the base address of the arrays `A` and `B` are in registers `$s6` and `$s7`, respectively. Assume that the elements of the arrays `A` and `B` are 4-byte words:

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

2.10 [5] <§§2.2, 2.3> Translate the following MIPS code to C. Assume that the variables `f`, `g`, `h`, `i`, and `j` are assigned to registers `$s0`, `$s1`, `$s2`, `$s3`, and `$s4`, respectively. Assume that the base address of the arrays `A` and `B` are in registers `$s6` and `$s7`, respectively.

```
addi $t0, $s6, 4
add  $t1, $s6, $0
sw   $t1, 0($t0)
lw   $t0, 0($t0)
add  $s0, $t1, $t0
```

2.11 [5] <§§2.2, 2.5> For each MIPS instruction, show the value of the opcode (OP), source register (RS), and target register (RT) fields. For the I-type instructions, show the value of the immediate field, and for the R-type instructions, show the value of the destination register (RD) field.

2.12 Assume that registers `$s0` and `$s1` hold the values `0x80000000` and `0xD0000000`, respectively.

2.12.1 [5] <\$2.4> What is the value of `$t0` for the following assembly code?

```
add $t0, $s0, $s1
```

2.12.2 [5] <\$2.4> Is the result in `$t0` the desired result, or has there been overflow?

2.12.3 [5] <\$2.4> For the contents of registers `$s0` and `$s1` as specified above, what is the value of `$t0` for the following assembly code?

```
sub $t0, $s0, $s1
```

2.12.4 [5] <\$2.4> Is the result in `$t0` the desired result, or has there been overflow?

2.12.5 [5] <\$2.4> For the contents of registers `$s0` and `$s1` as specified above, what is the value of `$t0` for the following assembly code?

```
add $t0, $s0, $s1
add $t0, $t0, $s0
```

2.12.6 [5] <\$2.4> Is the result in `$t0` the desired result, or has there been overflow?

2.13 Assume that `$s0` holds the value 128_{ten} .

2.13.1 [5] <\$2.4> For the instruction `add $t0, $s0, $s1`, what is the range(s) of values for `$s1` that would result in overflow?

2.13.2 [5] <\$2.4> For the instruction `sub $t0, $s0, $s1`, what is the range(s) of values for `$s1` that would result in overflow?

2.13.3 [5] <\$2.4> For the instruction `sub $t0, $s1, $s0`, what is the range(s) of values for `$s1` that would result in overflow?

2.14 [5] <\$\$2.2, 2.5> Provide the type and assembly language instruction for the following binary value: `0000 0010 0001 0000 1000 0000 0010 0000`_{two}

2.15 [5] <\$\$2.2, 2.5> Provide the type and hexadecimal representation of following instruction: `sw $t1, 32($t2)`

2.16 [5] <§2.5> Provide the type, assembly language instruction, and binary representation of instruction described by the following MIPS fields:

op=0, rs=3, rt=2, rd=3, shamt=0, funct=34

2.17 [5] <§2.5> Provide the type, assembly language instruction, and binary representation of instruction described by the following MIPS fields:

op=0x23, rs=1, rt=2, const=0x4

2.18 Assume that we would like to expand the MIPS register file to 128 registers and expand the instruction set to contain four times as many instructions.

2.18.1 [5] <§2.5> How this would this affect the size of each of the bit fields in the R-type instructions?

2.18.2 [5] <§2.5> How this would this affect the size of each of the bit fields in the I-type instructions?

2.18.3 [5] <§§2.5, 2.10> How could each of the two proposed changes decrease the size of an MIPS assembly program? On the other hand, how could the proposed change increase the size of an MIPS assembly program?

2.19 Assume the following register contents:

\$t0 = 0xAAAAAAAA, \$t1 = 0x12345678

2.19.1 [5] <§2.6> For the register values shown above, what is the value of \$t2 for the following sequence of instructions?

```
sll $t2, $t0, 44
or  $t2, $t2, $t1
```

2.19.2 [5] <§2.6> For the register values shown above, what is the value of \$t2 for the following sequence of instructions?

```
sll  $t2, $t0, 4
andi $t2, $t2, -1
```

2.19.3 [5] <§2.6> For the register values shown above, what is the value of \$t2 for the following sequence of instructions?

```
srl  $t2, $t0, 3
andi $t2, $t2, 0xFFEF
```

2.20 [5] <§2.6> Find the shortest sequence of MIPS instructions that extracts bits 16 down to 11 from register \$t0 and uses the value of this field to replace bits 31 down to 26 in register \$t1 without changing the other 26 bits of register \$t1.

2.21 [5] <§2.6> Provide a minimal set of MIPS instructions that may be used to implement the following pseudoinstruction:

```
not $t1, $t2      // bit-wise invert
```

2.22 [5] <§2.6> For the following C statement, write a minimal sequence of MIPS assembly instructions that does the identical operation. Assume \$t1 = A, \$t2 = B, and \$s1 is the base address of C.

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

2.23 [5] <§2.7> Assume \$t0 holds the value 0x00101000. What is the value of \$t2 after the following instructions?

```
slt  $t2, $0, $t0
bne  $t2, $0, ELSE
j    DONE
ELSE: addi $t2, $t2, 2
DONE:
```

2.24 [5] <§2.7> Suppose the program counter (PC) is set to 0x2000 0000. Is it possible to use the jump (j) MIPS assembly instruction to set the PC to the address as 0x4000 0000? Is it possible to use the branch-on-equal (beq) MIPS assembly instruction to set the PC to this same address?

2.25 The following instruction is not included in the MIPS instruction set:

```
rpt $t2, loop # if(R[rs]>0) R[rs]=R[rs]-1, PC=PC+4+BranchAddr
```

2.25.1 [5] <§2.7> If this instruction were to be implemented in the MIPS instruction set, what is the most appropriate instruction format?

2.25.2 [5] <§2.7> What is the shortest sequence of MIPS instructions that performs the same operation?

2.26 Consider the following MIPS loop:

```

LOOP: slt  $t2, $0, $t1
      beq  $t2, $0, DONE
      subi $t1, $t1, 1
      addi $s2, $s2, 2
      j    LOOP
DONE:

```

2.26.1 [5] <§2.7> Assume that the register `$t1` is initialized to the value 10. What is the value in register `$s2` assuming `$s2` is initially zero?

2.26.2 [5] <§2.7> For each of the loops above, write the equivalent C code routine. Assume that the registers `$s1`, `$s2`, `$t1`, and `$t2` are integers `A`, `B`, `i`, and `temp`, respectively.

2.26.3 [5] <§2.7> For the loops written in MIPS assembly above, assume that the register `$t1` is initialized to the value `N`. How many MIPS instructions are executed?

2.27 [5] <§2.7> Translate the following C code to MIPS assembly code. Use a minimum number of instructions. Assume that the values of `a`, `b`, `i`, and `j` are in registers `$s0`, `$s1`, `$t0`, and `$t1`, respectively. Also, assume that register `$s2` 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;

```

2.28 [5] <§2.7> How many MIPS instructions does it take to implement the C code from Exercise 2.27? 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 MIPS instructions that is executed to complete the loop?

2.29 [5] <§2.7> Translate the following loop into C. Assume that the C-level integer `i` is held in register `$t1`, `$s2` holds the C-level integer called `result`, and `$s0` holds the base address of the integer `MemArray`.

```

      addi $t1, $0, $0
LOOP: lw  $s1, 0($s0)
      add  $s2, $s2, $s1
      addi $s0, $s0, 4

```

```
addi $t1, $t1, 1
slti $t2, $t1, 100
bne  $t2, $s0, LOOP
```

2.30 [5] <§2.7> Rewrite the loop from Exercise 2.29 to reduce the number of MIPS instructions executed.

2.31 [5] <§2.8> Implement the following C code in MIPS assembly. What is the total number of MIPS instructions needed to execute the function?

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

2.32 [5] <§2.8> Functions can often be implemented by compilers “in-line.” An in-line function is when the body of the function is copied into the program space, allowing the overhead of the function call to be eliminated. Implement an “in-line” version of the C code above in MIPS assembly. What is the reduction in the total number of MIPS assembly instructions needed to complete the function? Assume that the C variable *n* is initialized to 5.

2.33 [5] <§2.8> For each function call, show the contents of the stack after the function call is made. Assume the stack pointer is originally at address 0x7fffffc, and follow the register conventions as specified in Figure 2.11.

2.34 Translate function *f* into MIPS assembly language. If you need to use registers \$t0 through \$t7, use the lower-numbered registers first. Assume the function declaration for *func* is “int *f*(int *a*, int *b*);”. The code for function *f* is as follows:

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

2.35 [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?

2.36 [5] <§2.8> Right before your function `f` from Exercise 2.34 returns, what do we know about contents of registers `$t5`, `$s3`, `$ra`, and `$sp`? Keep in mind that we know what the entire function `f` looks like, but for function `func` we only know its declaration.

2.37 [5] <§2.9> Write a program in MIPS assembly language to convert an ASCII number string containing positive and negative integer decimal strings, to an integer. Your program should expect register `$a0` to hold the address of a null-terminated string containing some combination of the digits 0 through 9. Your program should compute the integer value equivalent to this string of digits, then place the number in register `$v0`. If a non-digit character appears anywhere in the string, your program should stop with the value -1 in register `$v0`. For example, if register `$a0` points to a sequence of three bytes `50ten`, `52ten`, `0ten` (the null-terminated string “24”), then when the program stops, register `$v0` should contain the value `24ten`.

2.38 [5] <§2.9> Consider the following code:

```
lbu $t0, 0($t1)
sw  $t0, 0($t2)
```

Assume that the register `$t1` contains the address `0x1000 0000` and the register `$t2` contains the address `0x1000 0010`. Note the MIPS architecture utilizes big-endian addressing. Assume that the data (in hexadecimal) at address `0x1000 0000` is: `0x11223344`. What value is stored at the address pointed to by register `$t2`?

2.39 [5] <§2.10> Write the MIPS assembly code that creates the 32-bit constant `0010 0000 0000 0001 0100 1001 0010 0100two` and stores that value to register `$t1`.

2.40 [5] <§§2.6, 2.10> If the current value of the PC is `0x00000000`, can you use a single jump instruction to get to the PC address as shown in Exercise 2.39?

2.41 [5] <§§2.6, 2.10> If the current value of the PC is `0x00000600`, can you use a single branch instruction to get to the PC address as shown in Exercise 2.39?

2.42 [5] <§2.6, 2.10> If the current value of the PC is 0x1FFFf000, can you use a single branch instruction to get to the PC address as shown in Exercise 2.39?

2.43 [5] <§2.11> Write the MIPS assembly code to implement the following C code:

```
lock(lk);
shvar=max(shvar,x);
unlock(lk);
```

Assume that the address of the `lk` variable is in `$a0`, the address of the `shvar` variable is in `$a1`, and the value of variable `x` is in `$a2`. Your critical section should not contain any function calls. Use `ll/sc` instructions to implement the `lock()` operation, and the `unlock()` operation is simply an ordinary store instruction.

2.44 [5] <§2.11> Repeat Exercise 2.43, but this time use `ll/sc` to perform an atomic update of the `shvar` variable directly, without using `lock()` and `unlock()`. Note that in this problem there is no variable `lk`.

2.45 [5] <§2.11> Using your code from Exercise 2.43 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.

2.46 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.

2.46.1 [5] <§2.19> 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%, and the cost of increasing the clock cycle time by only 10%. Is this a good design choice? Why?

2.46.2 [5] <§2.19> 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?

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

2.47.1 [5] <§2.19> Given this instruction mix and the assumption that an arithmetic instruction requires 2 cycles, a load/store instruction takes 6 cycles, and a branch instruction takes 3 cycles, find the average CPI.

2.47.2 [5] <§2.19> 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?

2.47.3 [5] <§2.19> 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?

Answers to Check Yourself

§2.2, page 66: MIPS, C, Java

§2.3, page 72: 2) Very slow

§2.4, page 79: 2) -8_{ten}

§2.5, page 87: 4) sub \$t2, \$t0, \$t1

§2.6, page 89: Both. AND with a mask pattern of 1s will leave 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 word, with 0s in the rest of the word. Note that AND leaves the field where it was originally, and the shift pair moves the field into the rightmost part of the word.

§2.7, page 96: I. All are true. II. 1).

§2.8, page 106: Both are true.

§2.9, page 111: I. 1) and 2) II. 3)

§2.10, page 120: I. 4) $+ -128K$. II. 6) a block of 256M. III. 4) sll

§2.11, page 123: Both are true.

§2.12, page 132: 4) Machine independence.

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?

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.

*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

3.13 [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 4 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 4 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 4 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 9×6 , for example, can be written $(2 \times 2 \times 2 + 1) \times 6$, we can calculate 9×6 by shifting 6 to the left 3 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 0x0C000000 represent if it is a two's complement integer? An unsigned integer?

3.21 [10] <§3.5> If the bit pattern 0x0C000000 is placed into the Instruction Register, what MIPS instruction will be executed?

3.22 [10] <§3.5> What decimal number does the bit pattern 0x0C000000 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 -1.5625×10^{-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 -1.5625×10^{-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 -1.5625×10^{-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 2.6125×10^1 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 -8.0546875×10^0 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 8.625×10^1 divided by -4.875×10^0 . 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] <§3.9> Calculate $(3.984375 \times 10^{-1} + 3.4375 \times 10^{-1}) + 1.771 \times 10^3$ by hand, assuming each of the values are 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] <§3.9> Calculate $3.984375 \times 10^{-1} + (3.4375 \times 10^{-1} + 1.771 \times 10^3)$ by hand, assuming each of the values are 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] <§3.9> Based on your answers to 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] <§3.9> 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 are 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] <§3.9> 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 are 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] <§3.9> Based on your answers to 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] <§3.9> Calculate $1.666015625 \times 10^0 \times (1.9760 \times 10^4 + -1.9744 \times 10^4)$ by hand, assuming each of the values are 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] <§3.9> 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 are 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] <§3.9> Based on your answers to 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 4 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 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 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 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 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.

§3.2, page 182: 2.

§3.5, page 221: 3.

**Answers to
Check Yourself**



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, Rs, Rt`

Interpretation: $\text{Reg}[\text{Rd}] = \text{Reg}[\text{Rs}] \text{ AND } \text{Reg}[\text{Rt}]$

4.1.1 [5] <\$4.1> What are the values of control signals generated by the control in [Figure 4.2](#) for the above instruction?

4.1.2 [5] <\$4.1> Which resources (blocks) perform a useful function for this instruction?

4.1.3 [10] <\$4.1> Which resources (blocks) produce outputs, but their outputs are not used for this instruction? Which resources produce no outputs for this instruction?

4.2 The basic single-cycle MIPS implementation in [Figure 4.2](#) can only implement some instructions. New instructions can be added to an existing Instruction Set Architecture (ISA), but the decision whether or not to do that depends, among other things, on the cost and complexity the proposed addition introduces into the processor datapath and control. The first three problems in this exercise refer to the new instruction:

Instruction: `LWI Rt, Rd(Rs)`

Interpretation: $\text{Reg}[\text{Rt}] = \text{Mem}[\text{Reg}[\text{Rd}] + \text{Reg}[\text{Rs}]]$

4.2.1 [10] <\$4.1> Which existing blocks (if any) can be used for this instruction?

4.2.2 [10] <\$4.1> Which new functional blocks (if any) do we need for this instruction?

4.2.3 [10] <\$4.1> What new signals do we need (if any) from the control unit to support this instruction?

4.3 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 starting with a datapath from [Figure 4.2](#), where I-Mem, Add, Mux, ALU, Regs, D-Mem, and Control blocks have latencies of 400 ps, 100 ps, 30 ps, 120 ps, 200 ps, 350 ps, and 100 ps, respectively, and costs of 1000, 30, 10, 100, 200, 2000, and 500, respectively.

Consider the addition of a multiplier to the ALU. This addition will add 300 ps to the latency of the ALU and will add a cost of 600 to the ALU. The result will be 5% fewer instructions executed since we will no longer need to emulate the MUL instruction.

4.3.1 [10] <§4.1> What is the clock cycle time with and without this improvement?

4.3.2 [10] <§4.1> What is the speedup achieved by adding this improvement?

4.3.3 [10] <§4.1> Compare the cost/performance ratio with and without this improvement.

4.4 Problems in this exercise assume that logic blocks needed to implement a processor's datapath have the following latencies:

I-Mem	Add	Mux	ALU	Regs	D-Mem	Sign-Extend	Shift-Left-2
200ps	70ps	20ps	90ps	90ps	250ps	15ps	10ps

4.4.1 [10] <§4.3> If the only thing we need to do in a processor is fetch consecutive instructions ([Figure 4.6](#)), what would the cycle time be?

4.4.2 [10] <§4.3> Consider a datapath similar to the one in [Figure 4.11](#), but for a processor that only has one type of instruction: unconditional PC-relative branch. What would the cycle time be for this datapath?

4.4.3 [10] <§4.3> Repeat 4.4.2, but this time we need to support only conditional PC-relative branches.

The remaining three problems in this exercise refer to the datapath element Shift-left-2:

4.4.4 [10] <§4.3> Which kinds of instructions require this resource?

4.4.5 [20] <§4.3> For which kinds of instructions (if any) is this resource on the critical path?

4.4.6 [10] <§4.3> Assuming that we only support beq and add instructions, discuss how changes in the given latency of this resource affect the cycle time of the processor. Assume that the latencies of other resources do not change.

4.5 For the problems in this exercise, assume that there are no pipeline stalls and that the breakdown of executed instructions is as follows:

add	addi	not	beq	lw	sw
20%	20%	0%	25%	25%	10%

4.5.1 [10] <§4.3> In what fraction of all cycles is the data memory used?

4.5.2 [10] <§4.3> In what fraction of all cycles is the input of the sign-extend circuit needed? What is this circuit doing in cycles in which its input is not needed?

4.6 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). In this case we have a stuck-at-0 or a stuck-at-1 fault, and the affected signal always has a logical value of 0 or 1, respectively. The following problems refer to bit 0 of the Write Register input on the register file in [Figure 4.24](#).

4.6.1 [10] <§§4.3, 4.4> Let us assume that processor testing is done by filling the PC, registers, and data and instruction memories with some values (you can choose which values), letting a single instruction execute, then 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?

4.6.2 [10] <§§4.3, 4.4> Repeat 4.6.1 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.

4.6.3 [60] <§§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 MIPS 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. Hint: the processor is usable if every instruction “broken” by this fault can be replaced with a sequence of “working” instructions that achieve the same effect.

4.6.4 [10] <§§4.3, 4.4> Repeat 4.6.1, but now the fault to test for is whether the “MemRead” control signal becomes 0 if RegDst control signal is 0, no fault otherwise.

4.6.5 [10] <§§4.3, 4.4> Repeat 4.6.4, but now the fault to test for is whether the “Jump” control signal becomes 0 if RegDst control signal is 0, no fault otherwise.

4.7 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:

10101100011000100000000000010100.

Assume that data memory is all zeros and that the processor's registers have the following values at the beginning of the cycle in which the above instruction word is fetched:

r0	r1	r2	r3	r4	r5	r6	r8	r12	r31
0	-1	2	-3	-4	10	6	8	2	-16

4.7.1 [5] <\$4.4> What are the outputs of the sign-extend and the jump “Shift left 2” unit (near the top of Figure 4.24) for this instruction word?

4.7.2 [10] <\$4.4> What are the values of the ALU control unit's inputs for this instruction?

4.7.3 [10] <\$4.4> What is the new PC address after this instruction is executed? Highlight the path through which this value is determined.

4.7.4 [10] <\$4.4> For each Mux, show the values of its data output during the execution of this instruction and these register values.

4.7.5 [10] <\$4.4> For the ALU and the two add units, what are their data input values?

4.7.6 [10] <\$4.4> What are the values of all inputs for the “Registers” unit?

4.8 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
250ps	350ps	150ps	300ps	200ps

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

alu	beq	lw	sw
45%	20%	20%	15%

4.8.1 [5] <\$4.5> What is the clock cycle time in a pipelined and non-pipelined processor?

4.8.2 [10] <\$4.5> What is the total latency of an LW instruction in a pipelined and non-pipelined processor?

4.8.3 [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?

4.8.4 [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the data memory?

4.8.5 [10] <§4.5> Assuming there are no stalls or hazards, what is the utilization of the write-register port of the “Registers” unit?

4.8.6 [30] <§4.5> Instead of a single-cycle organization, we can use a multi-cycle organization where each instruction takes multiple cycles but one instruction finishes before another is fetched. In this organization, an instruction only goes through stages it actually needs (e.g., ST only takes 4 cycles because it does not need the WB stage). Compare clock cycle times and execution times with single-cycle, multi-cycle, and pipelined organization.

4.9 In this exercise, we examine how data dependences affect execution in the basic 5-stage pipeline described in Section 4.5. Problems in this exercise refer to the following sequence of instructions:

```
or r1,r2,r3
or r2,r1,r4
or r1,r1,r2
```

Also, assume the following cycle times for each of the options related to forwarding:

Without Forwarding	With Full Forwarding	With ALU-ALU Forwarding Only
250ps	300ps	290ps

4.9.1 [10] <§4.5> Indicate dependences and their type.

4.9.2 [10] <§4.5> Assume there is no forwarding in this pipelined processor. Indicate hazards and add `nop` instructions to eliminate them.

4.9.3 [10] <§4.5> Assume there is full forwarding. Indicate hazards and add `NOP` instructions to eliminate them.

4.9.4 [10] <§4.5> What is the total execution time of this instruction sequence without forwarding and with full forwarding? What is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

4.9.5 [10] <§4.5> Add `nop` instructions to this code to eliminate hazards if there is ALU-ALU forwarding only (no forwarding from the MEM to the EX stage).

4.9.6 [10] <§4.5> What is the total execution time of this instruction sequence with only ALU-ALU forwarding? What is the speedup over a no-forwarding pipeline?

4.10 In this exercise, we examine how resource hazards, control hazards, and Instruction Set Architecture (ISA) design can affect pipelined execution. Problems in this exercise refer to the following fragment of MIPS code:

```
sw  r16,12(r6)
lw  r16,8(r6)
beq r5,r4,Label # Assume r5!=r4
add r5,r1,r4
slt r5,r15,r4
```

Assume that individual pipeline stages have the following latencies:

IF	ID	EX	MEM	WB
200ps	120ps	150ps	190ps	100ps

4.10.1 [10] <\$4.5> For this problem, assume that all branches are perfectly predicted (this eliminates all control hazards) and that no delay slots are used. If we only have one memory (for both instructions and data), there is a structural hazard every time we need to fetch an instruction in the same cycle in which another instruction accesses data. To guarantee forward progress, this hazard must always be resolved in favor of the instruction that accesses data. What is the total execution time of this instruction sequence in the 5-stage pipeline that only has one memory? We have seen that data hazards can be eliminated by adding nops to the code. Can you do the same with this structural hazard? Why?

4.10.2 [20] <\$4.5> For this problem, assume that all branches are perfectly predicted (this eliminates all control hazards) and that no delay slots are used. 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. As a result, MEM and EX stages can be overlapped and the pipeline has only 4 stages. Change this code to accommodate this changed ISA. Assuming this change does not affect clock cycle time, what speedup is achieved in this instruction sequence?

4.10.3 [10] <\$4.5> Assuming stall-on-branch and no delay slots, what speedup is achieved on this code if branch outcomes are determined in the ID stage, relative to the execution where branch outcomes are determined in the EX stage?

4.10.4 [10] <\$4.5> Given these pipeline stage latencies, repeat the speedup calculation from 4.10.2, but take into account the (possible) change in clock cycle time. When EX and MEM are done in a single stage, most of their work can be done in parallel. As a result, the resulting EX/MEM stage has a latency that is the larger of the original two, plus 20 ps needed for the work that could not be done in parallel.

4.10.5 [10] <\$4.5> Given these pipeline stage latencies, repeat the speedup calculation from 4.10.3, taking into account the (possible) change in clock cycle time. Assume that the latency ID stage increases by 50% and the latency of the EX stage decreases by 10ps when branch outcome resolution is moved from EX to ID.

4.10.6 [10] <§4.5> Assuming stall-on-branch and no delay slots, what is the new clock cycle time and execution time of this instruction sequence if `beq` address computation is moved to the MEM stage? What is the speedup from this change? Assume that the latency of the EX stage is reduced by 20 ps and the latency of the MEM stage is unchanged when branch outcome resolution is moved from EX to MEM.

4.11 Consider the following loop.

```
loop: lw  r1, 0(r1)
      and r1, r1, r2
      lw  r1, 0(r1)
      lw  r1, 0(r1)
      beq r1, r0, loop
```

Assume that perfect branch prediction is used (no stalls due to control hazards), that there are no delay slots, and that the pipeline has full forwarding support. Also assume that many iterations of this loop are executed before the loop exits.

4.11.1 [10] <§4.6> Show a pipeline execution diagram for the third iteration of this loop, from the cycle in which we fetch the first instruction of that iteration up to (but not including) the cycle in which we can fetch the first instruction of the next iteration. Show all instructions that are in the pipeline during these cycles (not just those from the third iteration).

4.11.2 [10] <§4.6> How often (as a percentage of all cycles) do we have a cycle in which all five pipeline stages are doing useful work?

4.12 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 4.45](#). These problems assume that, of all the instructions executed in a processor, the following fraction of these instructions have 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 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. Also, assume that the CPI of the processor is 1 if there are no data hazards.

EX to 1 st Only	MEM to 1 st Only	EX to 2 nd Only	MEM to 2 nd Only	EX to 1 st and MEM to 2 nd	Other RAW Dependences
5%	20%	5%	10%	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
150 ps	100 ps	120 ps	150 ps	140 ps	130 ps	120 ps	100 ps

4.12.1 [10] <§4.7> If we use no forwarding, what fraction of cycles are we stalling due to data hazards?

4.12.2 [5] <§4.7> If we use full forwarding (forward all results that can be forwarded), what fraction of cycles are we stalling due to data hazards?

4.12.3 [10] <§4.7> Let us assume that we cannot afford to have three-input Muxes 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). Which of the two options results in fewer data stall cycles?

4.12.4 [10] <§4.7> For the given hazard probabilities and pipeline stage latencies, what is the speedup achieved by adding full forwarding to a pipeline that had no forwarding?

4.12.5 [10] <§4.7> What would be the additional speedup (relative to a processor with forwarding) if we added time-travel forwarding that eliminates all data hazards? Assume that the yet-to-be-invented time-travel circuitry adds 100 ps to the latency of the full-forwarding EX stage.

4.12.6 [20] <§4.7> Repeat 4.12.3 but this time determine which of the two options results in shorter time per instruction.

4.13 This exercise is intended to help you understand the relationship between forwarding, hazard detection, and ISA design. Problems in this exercise refer to the following sequence of instructions, and assume that it is executed on a 5-stage pipelined datapath:

```
add r5,r2,r1
lw  r3,4(r5)
lw  r2,0(r2)
or  r3,r5,r3
sw  r3,0(r5)
```

4.13.1 [5] <§4.7> If there is no forwarding or hazard detection, insert nops to ensure correct execution.

4.13.2 [10] <\$4.7> Repeat 4.13.1 but now use `nops` only when a hazard cannot be avoided by changing or rearranging these instructions. You can assume register R7 can be used to hold temporary values in your modified code.

4.13.3 [10] <\$4.7> If the processor has forwarding, but we forgot to implement the hazard detection unit, what happens when this code executes?

4.13.4 [20] <\$4.7> If there is forwarding, for the first five cycles during the execution of this code, specify which signals are asserted in each cycle by hazard detection and forwarding units in [Figure 4.60](#).

4.13.5 [10] <\$4.7> If there is no forwarding, what new inputs and output signals do we need for the hazard detection unit in [Figure 4.60](#)? Using this instruction sequence as an example, explain why each signal is needed.

4.13.6 [20] <\$4.7> For the new hazard detection unit from 4.13.5, specify which output signals it asserts in each of the first five cycles during the execution of this code.

4.14 This exercise is intended to help you understand the relationship between delay slots, control hazards, and branch execution in a pipelined processor. In this exercise, we assume that the following MIPS code is executed on a pipelined processor with a 5-stage pipeline, full forwarding, and a predict-taken branch predictor:

```

        lw r2,0(r1)
label1: beq r2,r0,label2 # not taken once, then taken
        lw r3,0(r2)
        beq r3,r0,label1 # taken
        add r1,r3,r1
label2: sw r1,0(r2)

```

4.14.1 [10] <\$4.8> Draw the pipeline execution diagram for this code, assuming there are no delay slots and that branches execute in the EX stage.

4.14.2 [10] <\$4.8> Repeat 4.14.1, but assume that delay slots are used. In the given code, the instruction that follows the branch is now the delay slot instruction for that branch.

4.14.3 [20] <\$4.8> One way to move the branch resolution one stage earlier is to not need an ALU operation in conditional branches. The branch instructions would be “`bez rd,label`” and “`bnez rd,label`”, and it would branch if the register has and does not have a zero value, respectively. Change this code to use these branch instructions instead of `beq`. You can assume that register R8 is available for you to use as a temporary register, and that an `seq` (set if equal) R-type instruction can be used.

Section 4.8 describes how the severity of control hazards can be reduced by moving branch execution into the ID stage. This approach involves a dedicated comparator in the ID stage, as shown in Figure 4.62. However, this approach potentially adds to the latency of the ID stage, and requires additional forwarding logic and hazard detection.

4.14.4 [10] <§4.8> Using the first branch instruction in the given code as an example, describe the hazard detection logic needed to support branch execution in the ID stage as in Figure 4.62. Which type of hazard is this new logic supposed to detect?

4.14.5 [10] <§4.8> For the given code, what is the speedup achieved by moving branch execution into the ID stage? Explain your answer. In your speedup calculation, assume that the additional comparison in the ID stage does not affect clock cycle time.

4.14.6 [10] <§4.8> Using the first branch instruction in the given code as an example, describe the forwarding support that must be added to support branch execution in the ID stage. Compare the complexity of this new forwarding unit to the complexity of the existing forwarding unit in Figure 4.62.

4.15 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	BEQ	JMP	LW	SW
40%	25%	5%	25%	5%

Also, assume the following branch predictor accuracies:

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

4.15.1 [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 EX stage, that there are no data hazards, and that no delay slots are used.

4.15.2 [10] <§4.8> Repeat 4.15.1 for the “always-not-taken” predictor.

4.15.3 [10] <§4.8> Repeat 4.15.1 for for the 2-bit predictor.

4.15.4 [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 replaces a branch instruction with an ALU instruction? Assume that correctly and incorrectly predicted instructions have the same chance of being replaced.

4.15.5 [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.

4.15.6 [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?

4.16 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

4.16.1 [5] <§4.8> What is the accuracy of always-taken and always-not-taken predictors for this sequence of branch outcomes?

4.16.2 [5] <§4.8> What is the accuracy of the two-bit predictor for the first 4 branches in this pattern, assuming that the predictor starts off in the bottom left state from [Figure 4.63](#) (predict not taken)?

4.16.3 [10] <§4.8> What is the accuracy of the two-bit predictor if this pattern is repeated forever?

4.16.4 [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.

4.16.5 [10] <§4.8> What is the accuracy of your predictor from 4.16.4 if it is given a repeating pattern that is the exact opposite of this one?

4.16.6 [20] <§4.8> Repeat 4.16.4, 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.

4.17 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
BNE R1, R2, Label	LW R1, 0(R1)

4.17.1 [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.

4.17.2 [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.

4.17.3 [10] <§4.9> If the second instruction is fetched right after the first instruction, describe what happens in the pipeline when the first instruction causes the first exception you listed in 4.17.1. 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.

4.17.4 [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 4.17.3 using this modified pipeline and vectored exception handling.

4.17.5 [15] <§4.9> We want to emulate vectored exception handling (described in 4.17.4) 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.

4.18 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];
```

When writing MIPS code, assume that variables are kept in registers as follows, and that all registers except those indicated as Free are used to keep various variables, so they cannot be used for anything else.

i	j	a	b	c	Free
R5	R6	R1	R2	R3	R10, R11, R12

4.18.1 [10] <§4.10> Translate this C code into MIPS instructions. Your translation should be direct, without rearranging instructions to achieve better performance.

4.18.2 [10] <§4.10> If the loop exits after executing only two iterations, draw a pipeline diagram for your MIPS code from 4.18.1 executed on a 2-issue processor shown in [Figure 4.69](#). Assume the processor has perfect branch prediction and can fetch any two instructions (not just consecutive instructions) in the same cycle.

4.18.3 [10] <§4.10> Rearrange your code from 4.18.1 to achieve better performance on a 2-issue statically scheduled processor from [Figure 4.69](#).

4.18.4 [10] <§4.10> Repeat 4.18.2, but this time use your MIPS code from 4.18.3.

4.18.5 [10] <§4.10> What is the speedup of going from a 1-issue processor to a 2-issue processor from Figure 4.69? Use your code from 4.18.1 for both 1-issue and 2-issue, and assume that 1,000,000 iterations of the loop are executed. As in 4.18.2, assume that the processor has perfect branch predictions, and that a 2-issue processor can fetch any two instructions in the same cycle.

4.18.6 [10] <§4.10> Repeat 4.18.5, but this time assume that in the 2-issue processor one of the instructions to be executed in a cycle can be of any kind, and the other must be a non-memory instruction.

4.19 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 spend a negligible amount of energy.

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

4.19.1 [10] <§§4.3, 4.6, 4.14> How much energy is spent to execute an ADD instruction in a single-cycle design and in the 5-stage pipelined design?

4.19.2 [10] <§§4.6, 4.14> What is the worst-case MIPS instruction in terms of energy consumption, and what is the energy spent to execute it?

4.19.3 [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 LW instruction after this change?

4.19.4 [10] <§§4.6, 4.14> What is the performance impact of your changes from 4.19.3?

4.19.5 [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. What is the effect of this change on clock frequency and energy consumption?

4.19.6 [10] <§§4.6, 4.14> If an idle unit spends 10% of the power it would spend if it were active, what is the energy spent by the instruction memory in each cycle? What percentage of the overall energy spent by the instruction memory does this idle energy represent?

**Answers to
Check Yourself**

§4.1, page 248: 3 of 5: Control, Datapath, Memory. Input and Output are missing.

§4.2, page 251: false. Edge-triggered state elements make simultaneous reading and writing both possible and unambiguous.

§4.3, page 257: I. a. II. c.

§4.4, page 272: Yes, Branch and ALUOp0 are identical. In addition, MemtoReg and RegDst are inverses of one another. You don't need an inverter; simply use the other signal and flip the order of the inputs to the multiplexor!

§4.5, page 285: 1. Stall on the `lw` result. 2. Bypass the first `add` result written into `$t1`. 3. No stall or bypass required.

§4.6, page 298: Statements 2 and 4 are correct; the rest are incorrect.

§4.8, page 324: 1. Predict not taken. 2. Predict taken. 3. Dynamic prediction.

§4.9, page 332: The first instruction, since it is logically executed before the others.

§4.10, page 344: 1. Both. 2. Both. 3. Software. 4. Hardware. 5. Hardware. 6. Hardware. 7. Both. 8. Hardware. 9. Both.

§4.11, page 353: First two are false and the last two are true.

As we will see in Chapter 6, memory systems are a central design issue for parallel processors. The growing importance 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.

5.18

Exercises

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 32-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 32-bit integers can be stored in a 16-byte cache block?

5.1.2 [5] <§5.1> References to which variables exhibit temporal locality?

5.1.3 [5] <§5.1> References to which variables 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 by storing 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 [10] <§5.1> How many 16-byte cache blocks are needed to store all 32-bit matrix elements being referenced?

5.1.5 [5] <§5.1> References to which variables exhibit temporal locality?

5.1.6 [5] <§5.1> References to which variables exhibit spatial locality?

5.2 Caches are important to providing a high-performance memory hierarchy to processors. Below is a list of 32-bit memory address references, given as word addresses.

3, 180, 43, 2, 191, 88, 190, 14, 181, 44, 186, 253

5.2.1 [10] <§5.3> For each of these references, identify the binary address, the tag, and the index given a direct-mapped cache with 16 one-word blocks. Also list if 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 address, the tag, and the index given a direct-mapped cache with two-word blocks and a total size of 8 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 8 words of data: C1 has 1-word blocks, C2 has 2-word blocks, and C3 has 4-word blocks. In terms of miss rate, which cache design is the best? If the miss stall time is 25 cycles, and C1 has an access time of 2 cycles, C2 takes 3 cycles, and C3 takes 5 cycles, which is the best cache design?

There are many different design parameters that are important to a cache's overall performance. Below are listed parameters for different direct-mapped cache designs.

Cache Data Size: 32 KiB

Cache Block Size: 2 words

Cache Access Time: 1 cycle

5.2.4 [15] <§5.3> Calculate the total number of bits required for the cache listed above, assuming a 32-bit address. Given that total size, find the total size of the closest direct-mapped cache with 16-word blocks of equal size or greater. Explain why the second cache, despite its larger data size, might provide slower performance than the first cache.

5.2.5 [20] <§§5.3, 5.4> Generate a series of read requests that have a lower miss rate on a 2 KiB 2-way set associative cache than the cache listed above. Identify one possible solution that would make the cache listed have an equal or lower miss rate than the 2 KiB cache. Discuss the advantages and disadvantages of such a solution.

5.2.6 [15] <§5.3> The formula shown in 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 32-bit address and 1024 blocks in the cache, consider a different

indexing function, specifically (Block address[31:27] XOR Block address[26:22]). 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.

5.3 For a direct-mapped cache design with a 32-bit address, the following bits of the address are used to access the cache.

Tag	Index	Offset
31–10	9–5	4–0

5.3.1 [5] <§5.3> What is the cache block size (in words)?

5.3.2 [5] <§5.3> How many entries does the cache have?

5.3.3 [5] <§5.3> What is the ratio between total bits required for such a cache implementation over the data storage bits?

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

Address											
0	4	16	132	232	160	1024	30	140	3100	180	2180

5.3.4 [10] <§5.3> How many blocks are replaced?

5.3.5 [10] <§5.3> What is the hit ratio?

5.3.6 [20] <§5.3> List the final state of the cache, with each valid entry represented as a record of <index, tag, data>.

5.4 Recall that we have two write policies and 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

5.4.1 [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.

5.4.2 [20] <§§5.3, 5.8> Describe the procedure of handling an L1 write-miss, considering the component involved and the possibility of replacing a dirty block.

5.4.3 [20] <§§5.3, 5.8> For a multilevel exclusive cache (a block can only reside in one of the L1 and L2 caches), configuration, describe the procedure of handling an L1 write-miss, considering the component involved and the possibility of replacing a dirty block.

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 (byte)
250	100	0.30%	2%	64

5.4.4 [5] <§§5.3, 5.8> For a write-through, write-allocate cache, what are the minimum read and write bandwidths (measured by byte per cycle) needed to achieve a CPI of 2?

5.4.5 [5] <§§5.3, 5.8> For a write-back, write-allocate cache, assuming 30% of replaced data cache blocks are dirty, what are the minimal read and write bandwidths needed for a CPI of 2?

5.4.6 [5] <§§5.3, 5.8> What are the minimal bandwidths needed to achieve the performance of CPI=1.5?

5.5 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 address stream:

0, 2, 4, 6, 8, 10, 12, 14, 16, ...

5.5.1 [5] <§§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?

5.5.2 [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?

5.5.3 [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 is found in the prefetch buffer, it is considered as a hit and 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?

Cache block size (B) can affect both miss rate and miss latency. Assuming a 1-CPI machine with an average of 1.35 references (both instruction and data) per instruction, help find the optimal block size given the following miss rates for various block sizes.

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

5.5.4 [10] <§5.3> What is the optimal block size for a miss latency of $20 \times B$ cycles?

5.5.5 [10] <§5.3> What is the optimal block size for a miss latency of $24 + B$ cycles?

5.5.6 [10] <§5.3> For constant miss latency, what is the optimal block size?

5.6 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 memory accesses are 36% of all instructions. 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

5.6.1 [5] <§5.4> Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

5.6.2 [5] <§5.4> What is the Average Memory Access Time for P1 and P2?

5.6.3 [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?

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

5.6.4 [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?

5.6.5 [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?

5.6.6 [10] <§5.4> Which processor is faster, now that P1 has an L2 cache? If P1 is faster, what miss rate would P2 need in its L1 cache to match P1's performance? If P2 is faster, what miss rate would P1 need in its L1 cache to match P2's performance?

5.7 This exercise examines the impact of different cache designs, specifically comparing associative caches to the direct-mapped caches from Section 5.4. For these exercises, refer to the address stream shown in Exercise 5.2.

5.7.1 [10] <§5.4> Using the sequence of references from Exercise 5.2, show the final cache contents for a three-way set associative cache with two-word blocks and a total size of 24 words. Use LRU replacement. For each reference identify the index bits, the tag bits, the block offset bits, and if it is a hit or a miss.

5.7.2 [10] <§5.4> Using the references from Exercise 5.2, show the final cache contents for a fully associative cache with one-word blocks and a total size of 8 words. Use LRU replacement. For each reference identify the index bits, the tag bits, and if it is a hit or a miss.

5.7.3 [15] <§5.4> Using the references from Exercise 5.2, what is the miss rate for a fully associative cache with two-word blocks and a total size of 8 words, using LRU replacement? What is the miss rate using MRU (most recently used) replacement? Finally what is the best possible miss rate for this cache, given any replacement policy?

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 MissRate per Instruction	Second Level Cache, Direct-Mapped Speed	Global Miss Rate with Second Level Cache, Direct-Mapped	Second Level Cache, Eight-Way Set Associative Speed	Global 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%

5.7.4 [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 is doubled? If it is cut in half?

5.7.5 [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 reduce the global miss rate to 1.3%. Would this provide better performance? In general, what are the advantages and disadvantages of adding a third level cache?

5.7.6 [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 global miss rate of 4%. If each additional 512 KiB of cache lowered global 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? Of the eight-way set associative cache?

5.8 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 devices with the following metrics.

MTTF	MTTR
3 Years	1 Day

5.8.1 [5] <\$5.5> Calculate the MTBF for each of the devices in the table.

5.8.2 [5] <\$5.5> Calculate the availability for each of the devices in the table.

5.8.3 [5] <\$5.5> What happens to availability as the MTTR approaches 0? Is this a realistic situation?

5.8.4 [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?

5.9 This Exercise examines the single error correcting, double error detecting (SEC/DED) Hamming code.

5.9.1 [5] <\$5.5> What is the minimum number of parity bits required to protect a 128-bit word using the SEC/DED code?

5.9.2 [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 5.9.1. 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?

5.9.3 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.

5.10 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 16K.

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

5.10.1 [10] <\$5.7> What is the best page size if entries now become 128 bytes?

5.10.2 [10] <\$5.7> Based on 5.10.1, what is the best page size if pages are half full?

5.10.3 [20] <\$5.7> Based on 5.10.2, 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 $\text{\$/disk}/\text{accesses_per_sec}$, while the cost to keep a page in DRAM is $\text{\$/MiB}/\text{page_size}$. 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

5.10.4 [10] <\\$5.1, 5.7> What are the reuse time thresholds for these three technology generations?

5.10.5 [10] <\\$5.7> What are the reuse time thresholds if we keep using the same 4K page size? What’s the trend here?

5.10.6 [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.

5.11 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 constitutes a stream of virtual addresses as seen on a system. Assume 4 KiB pages, a 4-entry fully associative TLB, and true LRU replacement. If pages must be brought in from disk, increment the next largest page number.

4669, 2227, 13916, 34587, 48870, 12608, 49225

TLB

Valid	Tag	Physical Page Number
1	11	12
1	7	4
1	3	6
0	4	9

Page table

Valid	Physical Page or in Disk
1	5
0	Disk
0	Disk
1	6
1	9
1	11
0	Disk
1	4
0	Disk
0	Disk
1	3
1	12

5.11.1 [10] <§5.7> Given the address stream shown, and the initial TLB and page table states provided above, show the final state of the system. Also list for each reference if it is a hit in the TLB, a hit in the page table, or a page fault.

5.11.2 [15] <§5.7> Repeat 5.11.1, 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?

5.11.3 [15] <§§5.4, 5.7> Show the final contents of the TLB if it is 2-way set associative. Also show the contents of the TLB if it is direct mapped. Discuss the importance of having a TLB to high performance. How would virtual memory accesses be handled if there were no TLB?

There are several parameters that impact 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

5.11.4 [5] <§5.7> Given the parameters shown above, calculate the total page table size for a system running 5 applications that utilize half of the memory available.

5.11.5 [10] <§5.7> Given the parameters shown above, calculate the total page table size for a system running 5 applications that utilize half of the memory available, given a two level page table approach with 256 entries. Assume each entry of the main page table is 6 bytes. Calculate the minimum and maximum amount of memory required.

5.11.6 [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 2 words per block? How would the designer increase the data size of the cache?

5.12 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

5.12.1 [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?

5.12.2 [10] <§5.7> Using a multilevel 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 in this case? And how many memory references are needed for address translation if missing in TLB?

5.12.3 [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?

The following table shows the contents of a 4-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

5.12.4 [5] <§5.7> Under what scenarios would entry 2's valid bit be set to zero?

5.12.5 [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?

5.12.6 [5] <§5.7> What happens when an instruction writes to VA page 200?

5.13 In this exercise, we will examine how replacement policies impact miss rate. Assume a 2-way set associative cache with 4 blocks. To solve the problems in this exercise, you may find it helpful to draw a table like the one below, as demonstrated for the address sequence "0, 1, 2, 3, 4."

Address of Memory Block Accessed	Hit or Miss	Evicted Block	Contents of Cache Blocks After Reference			
			Set 0	Set 0	Set 1	Set 1
0	Miss		Mem[0]			
1	Miss		Mem[0]		Mem[1]	
2	Miss		Mem[0]	Mem[2]	Mem[1]	
3	Miss		Mem[0]	Mem[2]	Mem[1]	Mem[3]
4	Miss	0	Mem[4]	Mem[2]	Mem[1]	Mem[3]
...						

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

5.13.1 [5] <§§5.4, 5.8> Assuming an LRU replacement policy, how many hits does this address sequence exhibit?

5.13.2 [5] <§§5.4, 5.8> Assuming an MRU (most recently used) replacement policy, how many hits does this address sequence exhibit?

5.13.3 [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?

5.13.4 [10] <§§5.4, 5.8> Which address should be evicted at each replacement to maximize the number of hits? How many hits does this address sequence exhibit if you follow this “optimal” policy?

5.13.5 [10] <§§5.4, 5.8> Describe why it is difficult to implement a cache replacement policy that is optimal for all address sequences.

5.13.6 [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 impact could this have on miss rate?

5.14 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;

5.14.1 [10] <§§5.6, 5.7> What would happen for the given operation sequence for shadow page table and nested page table, respectively?

5.14.2 [10] <§§5.6, 5.7> Assuming an x86-based 4-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?

5.14.3 [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

5.14.4 [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)?

5.14.5 [10] <§5.6> What techniques can be used to reduce page table shadowing induced overhead?

5.14.6 [10] <§5.6> What techniques can be used to reduce NPT induced overhead?

5.15 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	Performance Impact to Trap to the Guest O/S	Performance Impact 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

5.15.1 [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 performance impact doubles? If it is cut in half? If a virtual machine software company wishes to obtain a 10% performance degradation, what is the longest possible penalty to trap to the VMM?

5.15.2 [10] <§5.6> I/O accesses often have a large impact 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? Explain why I/O bound applications have a smaller impact from virtualization.

5.15.3 [30] <§§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.

5.15.4 [20] <§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?

5.16 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 5.40 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 465 (Figure 5.40 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.

5.16.1 [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?

5.16.2 [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?

5.16.3 [30] <§§5.8, 5.9> Design a finite state machine to enable the use of a write buffer.

5.17 Cache coherence concerns the views of multiple processors on a given cache block. The following data shows two processors and their read/write operations on two different words of a cache block X (initially $X[0] = X[1] = 0$). Assume the size of integers is 32 bits.

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

5.17.1 [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.

5.17.2 [15] <§5.10> For a snooping protocol, list a valid operation sequence on each processor/cache to finish the above read/write operations.

5.17.3 [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 shows two processors and their read/write operations on different cache blocks (A and B initially 0).

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

5.17.4 [15] <§5.10> List the possible values of C and D for an implementation that ensures both consistency assumptions on page 470.

5.17.5 [15] <§5.10> List at least one more possible pair of values for C and D if such assumptions are not maintained.

5.17.6 [15] <§§5.3, 5.10> For various combinations of write policies and write allocation policies, which combinations make the protocol implementation simpler?

5.18 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 L1 cache misses once every 32 instructions.

	Private	Shared
Benchmark A misses-per-instruction	0.30%	0.12%
Benchmark B misses-per-instruction	0.06%	0.03%

Assume the following hit latencies:

Private Cache	Shared Cache	Memory
5	20	180

5.18.1 [15] <§5.13> Which cache design is better for each of these benchmarks? Use data to support your conclusion.

5.18.2 [15] <§5.13> Shared cache latency increases with the CMP size. Choose the best design if the shared cache latency doubles. Off-chip bandwidth becomes the bottleneck as the number of CMP cores increases. Choose the best design if off-chip memory latency doubles.

5.18.3 [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.

5.18.4 [15] <§5.13> Assume both benchmarks have a base CPI of 1 (ideal L2 cache). If having non-blocking cache improves the average number of concurrent L2 misses from 1 to 2, how much performance improvement does this provide over a shared L2 cache? How much improvement can be achieved over private L2?

5.18.5 [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?

5.18.6 [15] <§5.13> Consider the entire memory hierarchy. What kinds of optimizations can improve the number of concurrent misses?

5.19 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);
```

5.19.1 [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?

5.19.2 [10] <\$5.15> How can you reorganize the data structure to improve cache utilization and access locality? Show your structure definition code.

5.19.3 [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.

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.

a.	Mesa / gcc
b.	mcf / swim

5.19.4 [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?

5.19.5 [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.

5.19.6 [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.

**Answers to
Check Yourself**

§5.1, page 377: 1 and 4. (3 is false because the cost of the memory hierarchy varies per computer, but in 2013 the highest cost is usually the DRAM.)

§5.3, page 398: 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 417: 1.

§5.7, page 454: 1-a, 2-c, 3-b, 4-d.

§5.8, page 461: 2. (Both large block sizes and prefetching may reduce compulsory misses, so 1 is false.)

This parallel revolution in the hardware/software interface is perhaps the greatest challenge facing the field in the last 60 years. You can also think of it as the greatest opportunity, as our Going Faster sections demonstrate. This revolution will provide many new research and business prospects inside and outside the IT field, and the companies that dominate the multicore era may not be the same ones that dominated the uniprocessor era. After understanding the underlying hardware trends and learning to adapt software to them, perhaps you will be one of the innovators who will seize the opportunities that are certain to appear in the uncertain times ahead. We look forward to benefiting from your inventions!



Historical Perspective and Further Reading

This section online gives the rich and often disastrous history of multiprocessors over the last 50 years.

References

- G. Regnier, S. Makineni, R. Illikkal, R. Iyer, D. Minturn, R. Huggahalli, D. Newell, L. Cline, and A. Foong. TCP onloading for data center servers. *IEEE Computer*, 37(11):48–58, 2004.
- B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, R. Sears. Benchmarking cloud serving systems with YCSB, In: *Proceedings of the 1st ACM Symposium on Cloud computing*, June 10–11, 2010, Indianapolis, Indiana, USA, doi:10.1145/1807128.1807152.

6.16

Exercises

6.1 First, write down a list of your daily activities that you typically do on a weekday. For instance, you might get out of bed, take a shower, get dressed, eat breakfast, dry your hair, brush your teeth. Make sure to break down your list so you have a minimum of 10 activities.

6.1.1 [5] <\$6.2> Now consider which of these activities is already exploiting some form of parallelism (e.g., brushing multiple teeth at the same time, versus one at a time, carrying one book at a time to school, versus loading them all into your

backpack and then carry them “in parallel”). For each of your activities, discuss if they are already working in parallel, but if not, why they are not.

6.1.2 [5] <\$6.2> Next, consider which of the activities could be carried out concurrently (e.g., eating breakfast and listening to the news). For each of your activities, describe which other activity could be paired with this activity.

6.1.3 [5] <\$6.2> For 6.1.2, what could we change about current systems (e.g., showers, clothes, TVs, cars) so that we could perform more tasks in parallel?

6.1.4 [5] <\$6.2> Estimate how much shorter time it would take to carry out these activities if you tried to carry out as many tasks in parallel as possible.

6.2 You are trying to bake 3 blueberry pound cakes. Cake ingredients are as follows:

1 cup butter, softened
1 cup sugar
4 large eggs
1 teaspoon vanilla extract
1/2 teaspoon salt
1/4 teaspoon nutmeg
1 1/2 cups flour
1 cup blueberries

The recipe for a single cake is as follows:

Step 1: Preheat oven to 325°F (160°C). Grease and flour your cake pan.

Step 2: In large bowl, beat together with a mixer butter and sugar at medium speed until light and fluffy. Add eggs, vanilla, salt and nutmeg. Beat until thoroughly blended. Reduce mixer speed to low and add flour, 1/2 cup at a time, beating just until blended.

Step 3: Gently fold in blueberries. Spread evenly in prepared baking pan. Bake for 60 minutes.

6.2.1 [5] <\$6.2> Your job is to cook 3 cakes as efficiently as possible. Assuming that you only have one oven large enough to hold one cake, one large bowl, one cake pan, and one mixer, come up with a schedule to make three cakes as quickly as possible. Identify the bottlenecks in completing this task.

6.2.2 [5] <\$6.2> Assume now that you have three bowls, 3 cake pans and 3 mixers. How much faster is the process now that you have additional resources?

6.2.3 [5] <\$6.2> Assume now that you have two friends that will help you cook, and that you have a large oven that can accommodate all three cakes. How will this change the schedule you arrived at in Exercise 6.2.1 above?

6.2.4 [5] <\$6.2> Compare the cake-making task to computing 3 iterations of a loop on a parallel computer. Identify data-level parallelism and task-level parallelism in the cake-making loop.

6.3 Many computer applications involve searching through a set of data and sorting the data. A number of efficient searching and sorting algorithms have been devised in order to reduce the runtime of these tedious tasks. In this problem we will consider how best to parallelize these tasks.

6.3.1 [10] <\$6.2> Consider the following binary search algorithm (a classic divide and conquer algorithm) that searches for a value X in a sorted N -element array A and returns the index of matched entry:

```

BinarySearch(A[0..N-1], X) {
    low = 0
    high = N - 1
    while (low <= high) {
        mid = (low + high) / 2
        if (A[mid] > X)
            high = mid - 1
        else if (A[mid] < X)
            low = mid + 1
        else
            return mid // found
    }
    return -1 // not found
}

```

Assume that you have Y cores on a multi-core processor to run `BinarySearch`. Assuming that Y is much smaller than N , express the speedup factor you might expect to obtain for values of Y and N . Plot these on a graph.

6.3.2 [5] <\$6.2> Next, assume that Y is equal to N . How would this affect your conclusions in your previous answer? If you were tasked with obtaining the best speedup factor possible (i.e., strong scaling), explain how you might change this code to obtain it.

6.4 Consider the following piece of C code:

```

for (j=2; j<1000; j++)
    D[j] = D[j-1]+D[j-2];

```

The MIPS code corresponding to the above fragment is:

```

        addiu    $s2,$zero,7992
        addiu    $s1,$zero,16
loop:   l.d      $f0, -16($s1)
        l.d      $f2, -8($s1)
        add.d    $f4, $f0, $f2
        s.d      $f4, 0($s1)
        addiu    $s1, $s1, 8
        bne      $s1, $s2, loop

```

Instructions have the following associated latencies (in cycles):

add.d	l.d	s.d	addiu
4	6	1	2

6.4.1 [10] <\$6.2> How many cycles does it take for all instructions in a single iteration of the above loop to execute?

6.4.2 [10] <\$6.2> When an instruction in a later iteration of a loop depends upon a data value produced in an earlier iteration of the same loop, we say that there is a *loop carried dependence* between iterations of the loop. Identify the loop-carried dependences in the above code. Identify the dependent program variable and assembly-level registers. You can ignore the loop induction variable *j*.

6.4.3 [10] <\$6.2> Loop unrolling was described in Chapter 4. Apply loop unrolling to this loop and then consider running this code on a 2-node distributed memory message passing system. Assume that we are going to use message passing as described in Section 6.7, where we introduce a new operation `send(x, y)` that sends to node *x* the value *y*, and an operation `receive()` that waits for the value being sent to it. Assume that `send` operations take a cycle to issue (i.e., later instructions on the same node can proceed on the next cycle), but take 10 cycles to be received on the receiving node. Receive instructions stall execution on the node where they are executed until they receive a message. Produce a schedule for the two nodes assuming an unroll factor of 4 for the loop body (i.e., the loop body will appear 4 times). Compute the number of cycles it will take for the loop to run on the message passing system.

6.4.4 [10] <\$6.2> The latency of the interconnect network plays a large role in the efficiency of message passing systems. How fast does the interconnect need to be in order to obtain any speedup from using the distributed system described in Exercise 6.4.3?

6.5 Consider the following recursive mergesort algorithm (another classic divide and conquer algorithm). Mergesort was first described by John Von Neumann in 1945. The basic idea is to divide an unsorted list *x* of *m* elements into two sublists of about half the size of the original list. Repeat this operation on each sublist, and

continue until we have lists of size 1 in length. Then starting with sublists of length 1, “merge” the two sublists into a single sorted list.

```
Mergesort(m)
  var list left, right, result
  if length(m) ≤ 1
    return m
  else
    var middle = length(m) / 2
    for each x in m up to middle
      add x to left
    for each x in m after middle
      add x to right
    left = Mergesort(left)
    right = Mergesort(right)
    result = Merge(left, right)
    return result
```

The merge step is carried out by the following code:

```
Merge(left, right)
  var list result
  while length(left) > 0 and length(right) > 0
    if first(left) ≤ first(right)
      append first(left) to result
      left = rest(left)
    else
      append first(right) to result
      right = rest(right)
  if length(left) > 0
    append rest(left) to result
  if length(right) > 0
    append rest(right) to result
  return result
```

6.5.1 [10] <\$6.2> Assume that you have Y cores on a multicore processor to run MergeSort. Assuming that Y is much smaller than $\text{length}(m)$, express the speedup factor you might expect to obtain for values of Y and $\text{length}(m)$. Plot these on a graph.

6.5.2 [10] <\$6.2> Next, assume that Y is equal to $\text{length}(m)$. How would this affect your conclusions your previous answer? If you were tasked with obtaining the best speedup factor possible (i.e., strong scaling), explain how you might change this code to obtain it.

6.6 Matrix multiplication plays an important role in a number of applications. Two matrices can only be multiplied if the number of columns of the first matrix is equal to the number of rows in the second.

Let's assume we have an $m \times n$ matrix A and we want to multiply it by an $n \times p$ matrix B . We can express their product as an $m \times p$ matrix denoted by AB (or $A \cdot B$). If we assign $C = AB$, and c_{ij} denotes the entry in C at position (i, j) , then for each element i and j with $1 \leq i \leq m$ and $1 \leq j \leq p$. Now we want to see if we can parallelize the computation of C . Assume that matrices are laid out in memory sequentially as follows: $a_{1,1}, a_{2,1}, a_{3,1}, a_{4,1}, \dots$, etc.

6.6.1 [10] <§6.5> Assume that we are going to compute C on both a single core shared memory machine and a 4-core shared-memory machine. Compute the speedup we would expect to obtain on the 4-core machine, ignoring any memory issues.

6.6.2 [10] <§6.5> Repeat Exercise 6.6.1, assuming that updates to C incur a cache miss due to false sharing when consecutive elements are in a row (i.e., index i) are updated.

6.6.3 [10] <§6.5> How would you fix the false sharing issue that can occur?

6.7 Consider the following portions of two different programs running at the same time on four processors in a symmetric multicore processor (SMP). Assume that before this code is run, both x and y are 0.

Core 1: $x = 2$;

Core 2: $y = 2$;

Core 3: $w = x + y + 1$;

Core 4: $z = x + y$;

6.7.1 [10] <§6.5> What are all the possible resulting values of w , x , y , and z ? For each possible outcome, explain how we might arrive at those values. You will need to examine all possible interleavings of instructions.

6.7.2 [5] <§6.5> How could you make the execution more deterministic so that only one set of values is possible?

6.8 The dining philosopher's problem is a classic problem of synchronization and concurrency. The general problem is stated as philosophers sitting at a round table doing one of two things: eating or thinking. When they are eating, they are not thinking, and when they are thinking, they are not eating. There is a bowl of pasta in the center. A fork is placed in between each philosopher. The result is that each philosopher has one fork to her left and one fork to her right. Given the nature of eating pasta, the philosopher needs two forks to eat, and can only use the forks on her immediate left and right. The philosophers do not speak to one another.

6.8.1 [10] <\$6.7> Describe the scenario where none of philosophers ever eats (i.e., starvation). What is the sequence of events that happen that lead up to this problem?

6.8.2 [10] <\$6.7> Describe how we can solve this problem by introducing the concept of a priority? But can we guarantee that we will treat all the philosophers fairly? Explain.

Now assume we hire a waiter who is in charge of assigning forks to philosophers. Nobody can pick up a fork until the waiter says they can. The waiter has global knowledge of all forks. Further, if we impose the policy that philosophers will always request to pick up their left fork before requesting to pick up their right fork, then we can guarantee to avoid deadlock.

6.8.3 [10] <\$6.7> We can implement requests to the waiter as either a queue of requests or as a periodic retry of a request. With a queue, requests are handled in the order they are received. The problem with using the queue is that we may not always be able to service the philosopher whose request is at the head of the queue (due to the unavailability of resources). Describe a scenario with 5 philosophers where a queue is provided, but service is not granted even though there are forks available for another philosopher (whose request is deeper in the queue) to eat.

6.8.4 [10] <\$6.7> If we implement requests to the waiter by periodically repeating our request until the resources become available, will this solve the problem described in Exercise 6.8.3? Explain.

6.9 Consider the following three CPU organizations:

CPU SS: A 2-core superscalar microprocessor that provides out-of-order issue capabilities on 2 function units (FUs). Only a single thread can run on each core at a time.

CPU MT: A fine-grained multithreaded processor that allows instructions from 2 threads to be run concurrently (i.e., there are two functional units), though only instructions from a single thread can be issued on any cycle.

CPU SMT: An SMT processor that allows instructions from 2 threads to be run concurrently (i.e., there are two functional units), and instructions from either or both threads can be issued to run on any cycle.

Assume we have two threads X and Y to run on these CPUs that include the following operations:

Thread X	Thread Y
A1 – takes 3 cycles to execute	B1 – take 2 cycles to execute
A2 – no dependences	B2 – conflicts for a functional unit with B1
A3 – conflicts for a functional unit with A1	B3 – depends on the result of B2
A4 – depends on the result of A3	B4 – no dependences and takes 2 cycles to execute

Assume all instructions take a single cycle to execute unless noted otherwise or they encounter a hazard.

6.9.1 [10] <\$6.4> Assume that you have 1 SS CPU. How many cycles will it take to execute these two threads? How many issue slots are wasted due to hazards?

6.9.2 [10] <\$6.4> Now assume you have 2 SS CPUs. How many cycles will it take to execute these two threads? How many issue slots are wasted due to hazards?

6.9.3 [10] <\$6.4> Assume that you have 1 MT CPU. How many cycles will it take to execute these two threads? How many issue slots are wasted due to hazards?

6.10 Virtualization software is being aggressively deployed to reduce the costs of managing today's high performance servers. Companies like VMWare, Microsoft and IBM have all developed a range of virtualization products. The general concept, described in Chapter 5, is that a hypervisor layer can be introduced between the hardware and the operating system to allow multiple operating systems to share the same physical hardware. The hypervisor layer is then responsible for allocating CPU and memory resources, as well as handling services typically handled by the operating system (e.g., I/O).

Virtualization provides an abstract view of the underlying hardware to the hosted operating system and application software. This will require us to rethink how multi-core and multiprocessor systems will be designed in the future to support the sharing of CPUs and memories by a number of operating systems concurrently.

6.10.1 [30] <\$6.4> Select two hypervisors on the market today, and compare and contrast how they virtualize and manage the underlying hardware (CPUs and memory).

6.10.2 [15] <\$6.4> Discuss what changes may be necessary in future multi-core CPU platforms in order to better match the resource demands placed on these systems. For instance, can multithreading play an effective role in alleviating the competition for computing resources?

6.11 We would like to execute the loop below as efficiently as possible. We have two different machines, a MIMD machine and a SIMD machine.

```
for (i=0; i < 2000; i++)
    for (j=0; j<3000; j++)
        X_array[i][j] = Y_array[j][i] + 200;
```

6.11.1 [10] <\$6.3> For a 4 CPU MIMD machine, show the sequence of MIPS instructions that you would execute on each CPU. What is the speedup for this MIMD machine?

6.11.2 [20] <\$6.3> For an 8-wide SIMD machine (i.e., 8 parallel SIMD functional units), write an assembly program in using your own SIMD extensions to MIPS to execute the loop. Compare the number of instructions executed on the SIMD machine to the MIMD machine.

6.12 A systolic array is an example of an MISD machine. A systolic array is a pipeline network or “wavefront” of data processing elements. Each of these elements does not need a program counter since execution is triggered by the arrival of data. Clocked systolic arrays compute in “lock-step” with each processor undertaking alternate compute and communication phases.

6.12.1 [10] <§6.3> Consider proposed implementations of a systolic array (you can find these in on the Internet or in technical publications). Then attempt to program the loop provided in Exercise 6.11 using this MISD model. Discuss any difficulties you encounter.

6.12.2 [10] <§6.3> Discuss the similarities and differences between an MISD and SIMD machine. Answer this question in terms of data-level parallelism.

6.13 Assume we want to execute the DAXPY loop show on page 511 in MIPS assembly on the NVIDIA 8800 GTX GPU described in this chapter. In this problem, we will assume that all math operations are performed on single-precision floating-point numbers (we will rename the loop SAXPY). Assume that instructions take the following number of cycles to execute.

Loads	Stores	Add.S	Mult.S
5	2	3	4

6.13.1 [20] <§6.6> Describe how you will constructs warps for the SAXPY loop to exploit the 8 cores provided in a single multiprocessor.

6.14 Download the CUDA Toolkit and SDK from http://www.nvidia.com/object/cuda_get.html. Make sure to use the “emurelease” (Emulation Mode) version of the code (you will not need actual NVIDIA hardware for this assignment). Build the example programs provided in the SDK, and confirm that they run on the emulator.

6.14.1 [90] <§6.6> Using the “template” SDK sample as a starting point, write a CUDA program to perform the following vector operations:

- 1) $a - b$ (vector-vector subtraction)
- 2) $a \cdot b$ (vector dot product)

The dot product of two vectors $a = [a_1, a_2, \dots, a_n]$ and $b = [b_1, b_2, \dots, b_n]$ is defined as:

$$a \cdot b = \sum_{i=1}^n a_i b_i = a_1 b_1 + a_2 b_2 + \dots + a_n b_n$$

Submit code for each program that demonstrates each operation and verifies the correctness of the results.

6.14.2 [90] <§6.6> If you have GPU hardware available, complete a performance analysis your program, examining the computation time for the GPU and a CPU version of your program for a range of vector sizes. Explain any results you see.

6.15 AMD has recently announced that they will be integrating a graphics processing unit with their x86 cores in a single package, though with different clocks for each of the cores. This is an example of a heterogeneous multiprocessor system which we expect to see produced commercially in the near future. One of the key design points will be to allow for fast data communication between the CPU and the GPU. Presently communications must be performed between discrete CPU and GPU chips. But this is changing in AMD's Fusion architecture. Presently the plan is to use multiple (at least 16) PCI express channels for facilitate intercommunication. Intel is also jumping into this arena with their Larrabee chip. Intel is considering to use their QuickPath interconnect technology.

6.15.1 [25] <§6.6> Compare the bandwidth and latency associated with these two interconnect technologies.

6.16 Refer to [Figure 6.14b](#), which shows an n-cube interconnect topology of order 3 that interconnects 8 nodes. One attractive feature of an n-cube interconnection network topology is its ability to sustain broken links and still provide connectivity.

6.16.1 [10] <§6.8> Develop an equation that computes how many links in the n-cube (where n is the order of the cube) can fail and we can still guarantee an unbroken link will exist to connect any node in the n-cube.

6.16.2 [10] <§6.8> Compare the resiliency to failure of n-cube to a fully-connected interconnection network. Plot a comparison of reliability as a function of the added number of links for the two topologies.

6.17 Benchmarking is field of study that involves identifying representative workloads to run on specific computing platforms in order to be able to objectively compare performance of one system to another. In this exercise we will compare two classes of benchmarks: the Whetstone CPU benchmark and the PARSEC Benchmark suite. Select one program from PARSEC. All programs should be freely available on the Internet. Consider running multiple copies of Whetstone versus running the PARSEC Benchmark on any of systems described in Section 6.11.

6.17.1 [60] <§6.10> What is inherently different between these two classes of workload when run on these multi-core systems?

6.17.2 [60] <§6.10> In terms of the Roofline Model, how dependent will the results you obtain when running these benchmarks be on the amount of sharing and synchronization present in the workload used?

6.18 When performing computations on sparse matrices, latency in the memory hierarchy becomes much more of a factor. Sparse matrices lack the spatial locality in the data stream typically found in matrix operations. As a result, new matrix representations have been proposed.

One the earliest sparse matrix representations is the Yale Sparse Matrix Format. It stores an initial sparse $m \times n$ matrix, M in row form using three one-dimensional

arrays. Let R be the number of nonzero entries in M . We construct an array A of length R that contains all nonzero entries of M (in left-to-right top-to-bottom order). We also construct a second array IA of length $m + 1$ (i.e., one entry per row, plus one). $IA(i)$ contains the index in A of the first nonzero element of row i . Row i of the original matrix extends from $A(IA(i))$ to $A(IA(i+1)-1)$. The third array, JA , contains the column index of each element of A , so it also is of length R .

6.18.1 [15] <\$6.10> Consider the sparse matrix X below and write C code that would store this code in Yale Sparse Matrix Format.

```
Row 1 [1, 2, 0, 0, 0, 0]
Row 2 [0, 0, 1, 1, 0, 0]
Row 3 [0, 0, 0, 0, 9, 0]
Row 4 [2, 0, 0, 0, 0, 2]
Row 5 [0, 0, 3, 3, 0, 7]
Row 6 [1, 3, 0, 0, 0, 1]
```

6.18.2 [10] <\$6.10> In terms of storage space, assuming that each element in matrix X is single precision floating point, compute the amount of storage used to store the Matrix above in Yale Sparse Matrix Format.

6.18.3 [15] <\$6.10> Perform matrix multiplication of Matrix X by Matrix Y shown below.

```
[2, 4, 1, 99, 7, 2]
```

Put this computation in a loop, and time its execution. Make sure to increase the number of times this loop is executed to get good resolution in your timing measurement. Compare the runtime of using a naïve representation of the matrix, and the Yale Sparse Matrix Format.

6.18.4 [15] <\$6.10> Can you find a more efficient sparse matrix representation (in terms of space and computational overhead)?

6.19 In future systems, we expect to see heterogeneous computing platforms constructed out of heterogeneous CPUs. We have begun to see some appear in the embedded processing market in systems that contain both floating point DSPs and a microcontroller CPUs in a multichip module package.

Assume that you have three classes of CPU:

CPU A—A moderate speed multi-core CPU (with a floating point unit) that can execute multiple instructions per cycle.

CPU B—A fast single-core integer CPU (i.e., no floating point unit) that can execute a single instruction per cycle.

CPU C—A slow vector CPU (with floating point capability) that can execute multiple copies of the same instruction per cycle.

Assume that our processors run at the following frequencies:

CPU A	CPU B	CPU C
1 GHz	3 GHz	250 MHz

CPU A can execute 2 instructions per cycle, CPU B can execute 1 instruction per cycle, and CPU C can execute 8 instructions (though the same instruction) per cycle. Assume all operations can complete execution in a single cycle of latency without any hazards.

All three CPUs have the ability to perform integer arithmetic, though CPU B cannot perform floating point arithmetic. CPU A and B have an instruction set similar to a MIPS processor. CPU C can only perform floating point add and subtract operations, as well as memory loads and stores. Assume all CPUs have access to shared memory and that synchronization has zero cost.

The task at hand is to compare two matrices X and Y that each contain 1024×1024 floating point elements. The output should be a count of the number indices where the value in X was larger or equal to the value in Y.

6.19.1 [10] <\$6.11> Describe how you would partition the problem on the 3 different CPUs to obtain the best performance.

6.19.2 [10] <\$6.11> What kind of instruction would you add to the vector CPU C to obtain better performance?

6.20 Assume a quad-core computer system can process database queries at a steady state rate of requests per second. Also assume that each transaction takes, on average, a fixed amount of time to process. The following table shows pairs of transaction latency and processing rate.

Average Transaction Latency	Maximum transaction processing rate
1 ms	5000/sec
2 ms	5000/sec
1 ms	10,000/sec
2 ms	10,000/sec

For each of the pairs in the table, answer the following questions:

6.20.1 [10] <\$6.11> On average, how many requests are being processed at any given instant?

6.20.2 [10] <\$6.11> If move to an 8-core system, ideally, what will happen to the system throughput (i.e., how many queries/second will the computer process)?

6.20.3 [10] <\$6.11> Discuss why we rarely obtain this kind of speedup by simply increasing the number of cores.

§6.1, page 504: False. Task-level parallelism can help sequential applications and sequential applications can be made to run on parallel hardware, although it is more challenging.

§6.2, page 509: False. *Weak* scaling can compensate for a serial portion of the program that would otherwise limit scalability, but not so for strong scaling.

§6.3, page 514: True, but they are missing useful vector features like gather-scatter and vector length registers that improve the efficiency of vector architectures. (As an elaboration in this section mentions, the AVX2 SIMD extensions offers indexed loads via a gather operation but *not* scatter for indexed stores. The Haswell generation x86 microprocessor is the first to support AVX2.)

§6.4, page 519: 1. True. 2. True.

§6.5, page 523: False. Since the shared address is a *physical* address, multiple tasks each in their own *virtual* address spaces can run well on a shared memory multiprocessor.

§6.6, page 531: False. Graphics DRAM chips are prized for their higher bandwidth.

§6.7, page 536: 1. False. Sending and receiving a message is an implicit synchronization, as well as a way to share data. 2. True.

§6.8, page 538: True.

§6.10, page 550: True. We likely need innovation at all levels of the hardware and software stack for parallel computing to succeed.

Answers to Check Yourself

1

Solutions

1.1 Personal computer (includes workstation and laptop): Personal computers emphasize delivery of good performance to single users at low cost and usually execute third-party software.

Personal mobile device (PMD, includes tablets): PMDs are battery operated with wireless connectivity to the Internet and typically cost hundreds of dollars, and, like PCs, users can download software (“apps”) to run on them. Unlike PCs, they no longer have a keyboard and mouse, and are more likely to rely on a touch-sensitive screen or even speech input.

Server: Computer used to run large problems and usually accessed via a network.

Warehouse scale computer: Thousands of processors forming a large cluster.

Supercomputer: Computer composed of hundreds to thousands of processors and terabytes of memory.

Embedded computer: Computer designed to run one application or one set of related applications and integrated into a single system.

1.2

- a. Performance via Pipelining
- b. Dependability via Redundancy
- c. Performance via Prediction
- d. Make the Common Case Fast
- e. Hierarchy of Memories
- f. Performance via Parallelism
- g. Design for Moore’s Law
- h. Use Abstraction to Simplify Design

1.3 The program is compiled into an assembly language program, which is then assembled into a machine language program.

1.4

- a. $1280 \times 1024 \text{ pixels} = 1,310,720 \text{ pixels} \Rightarrow 1,310,720 \times 3 = 3,932,160 \text{ bytes/frame.}$
- b. $3,932,160 \text{ bytes} \times (8 \text{ bits/byte}) / 100\text{E6 bits/second} = 0.31 \text{ seconds}$

1.5

- a. performance of P1 (instructions/sec) $= 3 \times 10^9 / 1.5 = 2 \times 10^9$
performance of P2 (instructions/sec) $= 2.5 \times 10^9 / 1.0 = 2.5 \times 10^9$
performance of P3 (instructions/sec) $= 4 \times 10^9 / 2.2 = 1.8 \times 10^9$

- b. $\text{cycles}(P1) = 10 \times 3 \times 10^9 = 30 \times 10^9 \text{ s}$
 $\text{cycles}(P2) = 10 \times 2.5 \times 10^9 = 25 \times 10^9 \text{ s}$
 $\text{cycles}(P3) = 10 \times 4 \times 10^9 = 40 \times 10^9 \text{ s}$
- c. $\text{No. instructions}(P1) = 30 \times 10^9 / 1.5 = 20 \times 10^9$
 $\text{No. instructions}(P2) = 25 \times 10^9 / 1 = 25 \times 10^9$
 $\text{No. instructions}(P3) = 40 \times 10^9 / 2.2 = 18.18 \times 10^9$
 $\text{CPI}_{\text{new}} = \text{CPI}_{\text{old}} \times 1.2$, then $\text{CPI}(P1) = 1.8$, $\text{CPI}(P2) = 1.2$, $\text{CPI}(P3) = 2.6$
 $f = \text{No. instr.} \times \text{CPI} / \text{time}$, then
 $f(P1) = 20 \times 10^9 \times 1.8 / 7 = 5.14 \text{ GHz}$
 $f(P2) = 25 \times 10^9 \times 1.2 / 7 = 4.28 \text{ GHz}$
 $f(P3) = 18.18 \times 10^9 \times 2.6 / 7 = 6.75 \text{ GHz}$

1.6

- a. Class A: 10^5 instr. Class B: 2×10^5 instr. Class C: 5×10^5 instr.
Class D: 2×10^5 instr.
 $\text{Time} = \text{No. instr.} \times \text{CPI} / \text{clock rate}$
 $\text{Total time P1} = (10^5 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 3 + 2 \times 10^5 \times 3) / (2.5 \times 10^9) = 10.4 \times 10^{-4} \text{ s}$
 $\text{Total time P2} = (10^5 \times 2 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 2 + 2 \times 10^5 \times 2) / (3 \times 10^9) = 6.66 \times 10^{-4} \text{ s}$
 $\text{CPI}(P1) = 10.4 \times 10^{-4} \times 2.5 \times 10^9 / 10^6 = 2.6$
 $\text{CPI}(P2) = 6.66 \times 10^{-4} \times 3 \times 10^9 / 10^6 = 2.0$
- b. $\text{clock cycles}(P1) = 10^5 \times 1 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 3 + 2 \times 10^5 \times 3 = 26 \times 10^5$
 $\text{clock cycles}(P2) = 10^5 \times 2 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 2 + 2 \times 10^5 \times 2 = 20 \times 10^5$

1.7

- a. $\text{CPI} = T_{\text{exec}} \times f / \text{No. instr.}$
Compiler A $\text{CPI} = 1.1$
Compiler B $\text{CPI} = 1.25$
- b. $f_B / f_A = (\text{No. instr.}(B) \times \text{CPI}(B)) / (\text{No. instr.}(A) \times \text{CPI}(A)) = 1.37$
- c. $T_A / T_{\text{new}} = 1.67$
 $T_B / T_{\text{new}} = 2.27$

1.8

$$1.8.1 \quad C = 2 \times DP / (V^2 \times F)$$

$$\text{Pentium 4: } C = 3.2\text{E-}8F$$

$$\text{Core i5 Ivy Bridge: } C = 2.9\text{E-}8F$$

$$1.8.2 \quad \text{Pentium 4: } 10/100 = 10\%$$

$$\text{Core i5 Ivy Bridge: } 30/70 = 42.9\%$$

$$1.8.3 \quad (S_{\text{new}} + D_{\text{new}}) / (S_{\text{old}} + D_{\text{old}}) = 0.90$$

$$D_{\text{new}} = C \times V_{\text{new}}^2 \times F$$

$$S_{\text{old}} = V_{\text{old}} \times I$$

$$S_{\text{new}} = V_{\text{new}} \times I$$

Therefore:

$$V_{\text{new}} = [D_{\text{new}} / (C \times F)]^{1/2}$$

$$D_{\text{new}} = 0.90 \times (S_{\text{old}} + D_{\text{old}}) - S_{\text{new}}$$

$$S_{\text{new}} = V_{\text{new}} \times (S_{\text{old}} / V_{\text{old}})$$

Pentium 4:

$$S_{\text{new}} = V_{\text{new}} \times (10/1.25) = V_{\text{new}} \times 8$$

$$D_{\text{new}} = 0.90 \times 100 - V_{\text{new}} \times 8 = 90 - V_{\text{new}} \times 8$$

$$V_{\text{new}} = [(90 - V_{\text{new}} \times 8) / (3.2\text{E}8 \times 3.6\text{E}9)]^{1/2}$$

$$V_{\text{new}} = 0.85 \text{ V}$$

Core i5:

$$S_{\text{new}} = V_{\text{new}} \times (30/0.9) = V_{\text{new}} \times 33.3$$

$$D_{\text{new}} = 0.90 \times 70 - V_{\text{new}} \times 33.3 = 63 - V_{\text{new}} \times 33.3$$

$$V_{\text{new}} = [(63 - V_{\text{new}} \times 33.3) / (2.9\text{E}8 \times 3.4\text{E}9)]^{1/2}$$

$$V_{\text{new}} = 0.64 \text{ V}$$

1.9**1.9.1**

p	# arith inst.	# L/S inst.	# branch inst.	cycles	ex. time	speedup
1	2.56E9	1.28E9	2.56E8	7.94E10	39.7	1
2	1.83E9	9.14E8	2.56E8	5.67E10	28.3	1.4
4	9.12E8	4.57E8	2.56E8	2.83E10	14.2	2.8
8	4.57E8	2.29E8	2.56E8	1.42E10	7.10	5.6

1.9.2

p	ex. time
1	41.0
2	29.3
4	14.6
8	7.33

1.9.3 3

1.10

$$1.10.1 \text{ die area}_{15\text{cm}} = \text{wafer area/dies per wafer} = \pi \cdot 7.5^2 / 84 = 2.10 \text{ cm}^2$$

$$\text{yield}_{15\text{cm}} = 1/(1 + (0.020 \cdot 2.10/2))^2 = 0.9593$$

$$\text{die area}_{20\text{cm}} = \text{wafer area/dies per wafer} = \pi \cdot 10^2 / 100 = 3.14 \text{ cm}^2$$

$$\text{yield}_{20\text{cm}} = 1/(1 + (0.031 \cdot 3.14/2))^2 = 0.9093$$

$$1.10.2 \text{ cost/die}_{15\text{cm}} = 12/(84 \cdot 0.9593) = 0.1489$$

$$\text{cost/die}_{20\text{cm}} = 15/(100 \cdot 0.9093) = 0.1650$$

$$1.10.3 \text{ die area}_{15\text{cm}} = \text{wafer area/dies per wafer} = \pi \cdot 7.5^2 / (84 \cdot 1.1) = 1.91 \text{ cm}^2$$

$$\text{yield}_{15\text{cm}} = 1/(1 + (0.020 \cdot 1.15 \cdot 1.91/2))^2 = 0.9575$$

$$\text{die area}_{20\text{cm}} = \text{wafer area/dies per wafer} = \pi \cdot 10^2 / (100 \cdot 1.1) = 2.86 \text{ cm}^2$$

$$\text{yield}_{20\text{cm}} = 1/(1 + (0.03 \cdot 1.15 \cdot 2.86/2))^2 = 0.9082$$

$$1.10.4 \text{ defects per area}_{0.92} = (1 - y^{.5}) / (y^{.5} \cdot \text{die_area}/2) = (1 - 0.92^{.5}) / (0.92^{.5} \cdot 2/2) = 0.043 \text{ defects/cm}^2$$

$$\text{defects per area}_{0.95} = (1 - y^{.5}) / (y^{.5} \cdot \text{die_area}/2) = (1 - 0.95^{.5}) / (0.95^{.5} \cdot 2/2) = 0.026 \text{ defects/cm}^2$$

1.11

$$1.11.1 \text{ CPI} = \text{clock rate} \times \text{CPU time/instr. count}$$

$$\text{clock rate} = 1/\text{cycle time} = 3 \text{ GHz}$$

$$\text{CPI}(\text{bzip2}) = 3 \times 10^9 \times 750 / (2389 \times 10^9) = 0.94$$

$$1.11.2 \text{ SPEC ratio} = \text{ref. time/execution time}$$

$$\text{SPEC ratio}(\text{bzip2}) = 9650/750 = 12.86$$

$$1.11.3. \text{ CPU time} = \text{No. instr.} \times \text{CPI/clock rate}$$

If CPI and clock rate do not change, the CPU time increase is equal to the increase in the of number of instructions, that is 10%.

1.11.4 $\text{CPU time}(\text{before}) = \text{No. instr.} \times \text{CPI}/\text{clock rate}$

$$\text{CPU time}(\text{after}) = 1.1 \times \text{No. instr.} \times 1.05 \times \text{CPI}/\text{clock rate}$$

$\text{CPU time}(\text{after})/\text{CPU time}(\text{before}) = 1.1 \times 1.05 = 1.155$. Thus, CPU time is increased by 15.5%.

1.11.5 $\text{SPECratio} = \text{reference time}/\text{CPU time}$

$$\text{SPECratio}(\text{after})/\text{SPECratio}(\text{before}) = \text{CPU time}(\text{before})/\text{CPU time}(\text{after}) = 1/1.155 = 0.86. \text{ The SPECratio is decreased by 14\%.}$$

1.11.6 $\text{CPI} = (\text{CPU time} \times \text{clock rate})/\text{No. instr.}$

$$\text{CPI} = 700 \times 4 \times 10^9 / (0.85 \times 2389 \times 10^9) = 1.37$$

1.11.7 $\text{Clock rate ratio} = 4 \text{ GHz} / 3 \text{ GHz} = 1.33$

$$\text{CPI @ 4 GHz} = 1.37, \text{ CPI @ 3 GHz} = 0.94, \text{ ratio} = 1.45$$

They are different because, although the number of instructions has been reduced by 15%, the CPU time has been reduced by a lower percentage.

1.11.8 $700/750 = 0.933$. CPU time reduction: 6.7%

1.11.9 $\text{No. instr.} = \text{CPU time} \times \text{clock rate}/\text{CPI}$

$$\text{No. instr.} = 960 \times 0.9 \times 4 \times 10^9 / 1.61 = 2146 \times 10^9$$

1.11.10 $\text{Clock rate} = \text{No. instr.} \times \text{CPI}/\text{CPU time}$.

$$\text{Clock rate}_{\text{new}} = \text{No. instr.} \times \text{CPI} / 0.9 \times \text{CPU time} = 1/0.9 \text{ clock rate}_{\text{old}} = 3.33 \text{ GHz}$$

1.11.11 $\text{Clock rate} = \text{No. instr.} \times \text{CPI}/\text{CPU time}$.

$$\text{Clock rate}_{\text{new}} = \text{No. instr.} \times 0.85 \times \text{CPI} / 0.80 \text{ CPU time} = 0.85/0.80, \text{ clock rate}_{\text{old}} = 3.18 \text{ GHz}$$

1.12

1.12.1 $T(P1) = 5 \times 10^9 \times 0.9 / (4 \times 10^9) = 1.125 \text{ s}$

$$T(P2) = 10^9 \times 0.75 / (3 \times 10^9) = 0.25 \text{ s}$$

$\text{clock rate}(P1) > \text{clock rate}(P2)$, $\text{performance}(P1) < \text{performance}(P2)$

1.12.2 $T(P1) = \text{No. instr.} \times \text{CPI}/\text{clock rate}$

$$T(P1) = 2.25 \times 10^{21} \text{ s}$$

$$T(P2) = 5 \times 10^8 \times 0.75 / (3 \times 10^9), \text{ then } N = 9 \times 10^8$$

1.12.3 $\text{MIPS} = \text{Clock rate} \times 10^{-6}/\text{CPI}$

$$\text{MIPS}(P1) = 4 \times 10^9 \times 10^{-6} / 0.9 = 4.44 \times 10^3$$

$$\text{MIPS(P2)} = 3 \times 10^9 \times 10^{-6}/0.75 = 4.0 \times 10^3$$

$\text{MIPS(P1)} > \text{MIPS(P2)}$, $\text{performance(P1)} < \text{performance(P2)}$ (from 11a)

$$\mathbf{1.12.4} \quad \text{MFLOPS} = \text{No. FP operations} \times 10^{-6}/T$$

$$\text{MFLOPS(P1)} = .4 \times 5\text{E}9 \times 1\text{E}-6/1.125 = 1.78\text{E}3$$

$$\text{MFLOPS(P2)} = .4 \times 1\text{E}9 \times 1\text{E}-6/.25 = 1.60\text{E}3$$

$\text{MFLOPS(P1)} > \text{MFLOPS(P2)}$, $\text{performance(P1)} < \text{performance(P2)}$ (from 11a)

1.13

$$\mathbf{1.13.1} \quad T_{\text{fp}} = 70 \times 0.8 = 56 \text{ s. } T_{\text{new}} = 56 + 85 + 55 + 40 = 236 \text{ s. Reduction: 5.6\%}$$

$$\mathbf{1.13.2} \quad T_{\text{new}} = 250 \times 0.8 = 200 \text{ s, } T_{\text{fp}} + T_{\text{l/s}} + T_{\text{branch}} = 165 \text{ s, } T_{\text{int}} = 35 \text{ s. Reduction time INT: 58.8\%}$$

$$\mathbf{1.13.3} \quad T_{\text{new}} = 250 \times 0.8 = 200 \text{ s, } T_{\text{fp}} + T_{\text{int}} + T_{\text{l/s}} = 210 \text{ s. NO}$$

1.14

$$\mathbf{1.14.1} \quad \text{Clock cycles} = \text{CPI}_{\text{fp}} \times \text{No. FP instr.} + \text{CPI}_{\text{int}} \times \text{No. INT instr.} + \text{CPI}_{\text{l/s}} \times \text{No. L/S instr.} + \text{CPI}_{\text{branch}} \times \text{No. branch instr.}$$

$$T_{\text{CPU}} = \text{clock cycles/clock rate} = \text{clock cycles}/2 \times 10^9$$

$$\text{clock cycles} = 512 \times 10^6; T_{\text{CPU}} = 0.256 \text{ s}$$

To have the number of clock cycles by improving the CPI of FP instructions:

$$\text{CPI}_{\text{improved fp}} \times \text{No. FP instr.} + \text{CPI}_{\text{int}} \times \text{No. INT instr.} + \text{CPI}_{\text{l/s}} \times \text{No. L/S instr.} + \text{CPI}_{\text{branch}} \times \text{No. branch instr.} = \text{clock cycles}/2$$

$$\text{CPI}_{\text{improved fp}} = (\text{clock cycles}/2 - (\text{CPI}_{\text{int}} \times \text{No. INT instr.} + \text{CPI}_{\text{l/s}} \times \text{No. L/S instr.} + \text{CPI}_{\text{branch}} \times \text{No. branch instr.})) / \text{No. FP instr.}$$

$$\text{CPI}_{\text{improved fp}} = (256 - 462)/50 < 0 ==> \text{not possible}$$

$$\mathbf{1.14.2} \quad \text{Using the clock cycle data from a.}$$

To have the number of clock cycles improving the CPI of L/S instructions:

$$\text{CPI}_{\text{fp}} \times \text{No. FP instr.} + \text{CPI}_{\text{int}} \times \text{No. INT instr.} + \text{CPI}_{\text{improved l/s}} \times \text{No. L/S instr.} + \text{CPI}_{\text{branch}} \times \text{No. branch instr.} = \text{clock cycles}/2$$

$$\text{CPI}_{\text{improved l/s}} = (\text{clock cycles}/2 - (\text{CPI}_{\text{fp}} \times \text{No. FP instr.} + \text{CPI}_{\text{int}} \times \text{No. INT instr.} + \text{CPI}_{\text{branch}} \times \text{No. branch instr.})) / \text{No. L/S instr.}$$

$$\text{CPI}_{\text{improved l/s}} = (256 - 198)/80 = 0.725$$

$$\mathbf{1.14.3} \quad \text{Clock cycles} = \text{CPI}_{\text{fp}} \times \text{No. FP instr.} + \text{CPI}_{\text{int}} \times \text{No. INT instr.} + \text{CPI}_{\text{l/s}} \times \text{No. L/S instr.} + \text{CPI}_{\text{branch}} \times \text{No. branch instr.}$$

$T_{\text{CPU}} = \text{clock cycles}/\text{clock rate} = \text{clock cycles}/2 \times 10^9$

$\text{CPI}_{\text{int}} = 0.6 \times 1 = 0.6$; $\text{CPI}_{\text{fp}} = 0.6 \times 1 = 0.6$; $\text{CPI}_{\text{l/s}} = 0.7 \times 4 = 2.8$;
 $\text{CPI}_{\text{branch}} = 0.7 \times 2 = 1.4$

$T_{\text{CPU}} (\text{before improv.}) = 0.256 \text{ s}$; $T_{\text{CPU}} (\text{after improv.}) = 0.171 \text{ s}$

1.15

processors	exec. time/ processor	time w/overhead	speedup	actual speedup/ideal speedup
1	100			
2	50	54	$100/54 = 1.85$	$1.85/2 = .93$
4	25	29	$100/29 = 3.44$	$3.44/4 = 0.86$
8	12.5	16.5	$100/16.5 = 6.06$	$6.06/8 = 0.75$
16	6.25	10.25	$100/10.25 = 9.76$	$9.76/16 = 0.61$



2

Solutions

2.1 `addi f, h, -5 (note, no subi)`
`add f, f, g`

2.2 `f = g + h + i`

2.3 `sub $t0, $s3, $s4`
`add $t0, $s6, $t0`
`lw $t1, 16($t0)`
`sw $t1, 32($s7)`

2.4 `B[g] = A[f] + A[1+f];`

2.5 `add $t0, $s6, $s0`
`add $t1, $s7, $s1`
`lw $s0, 0($t0)`
`lw $t0, 4($t0)`
`add $t0, $t0, $s0`
`sw $t0, 0($t1)`

2.6

2.6.1 `temp = Array[0];`
`temp2 = Array[1];`
`Array[0] = Array[4];`
`Array[1] = temp;`
`Array[4] = Array[3];`
`Array[3] = temp2;`

2.6.2 `lw $t0, 0($s6)`
`lw $t1, 4($s6)`
`lw $t2, 16($s6)`
`sw $t2, 0($s6)`
`sw $t0, 4($s6)`
`lw $t0, 12($s6)`
`sw $t0, 16($s6)`
`sw $t1, 12($s6)`

2.7

Little-Endian		Big-Endian	
Address	Data	Address	Data
12	ab	12	12
8	cd	8	ef
4	ef	4	cd
0	12	0	ab

2.8 2882400018

```

2.9 sll    $t0, $s1, 2    # $t0 <-- 4*g
      add    $t0, $t0, $s7 # $t0 <-- Addr(B[g])
      lw     $t0, 0($t0)   # $t0 <-- B[g]
      addi   $t0, $t0, 1   # $t0 <-- B[g]+1
      sll    $t0, $t0, 2   # $t0 <-- 4*(B[g]+1) = Addr(A[B[g]+1])
      lw     $s0, 0($t0)   # f    <-- A[B[g]+1]

```

2.10 $f = 2*(\&A);$

2.11

	type	opcode	rs	rt	rd	immed
addi \$t0, \$s6, 4	I-type	8	22	8		4
add \$t1, \$s6, \$0	R-type	0	22	0	9	
sw \$t1, 0(\$t0)	I-type	43	8	9		0
lw \$t0, 0(\$t0)	I-type	35	8	8		0
add \$s0, \$t1, \$t0	R-type	0	9	8	16	

2.12

2.12.1 50000000

2.12.2 overflow

2.12.3 B0000000

2.12.4 no overflow

2.12.5 D0000000

2.12.6 overflow

2.13

2.13.1 $128 + x > 2^{31}-1$, $x > 2^{31}-129$ and $128 + x < -2^{31}$, $x < -2^{31} - 128$
(impossible)

2.13.2 $128 - x > 2^{31}-1$, $x < -2^{31}+129$ and $128 - x < -2^{31}$, $x > 2^{31} + 128$
(impossible)

2.13.3 $x - 128 < -2^{31}$, $x < -2^{31} + 128$ and $x - 128 > 2^{31} - 1$, $x > 2^{31} + 127$
(impossible)

2.14 r-type, add \$s0, \$s0, \$s0

2.15 i-type, 0xAD490020

2.16 r-type, sub \$v1, \$v1, \$v0, 0x00621822

2.17 i-type, lw \$v0, 4(\$at), 0x8C220004

2.18

2.18.1 opcode would be 8 bits, rs, rt, rd fields would be 7 bits each

2.18.2 opcode would be 8 bits, rs and rt fields would be 7 bits each

2.18.3 more registers → more bits per instruction → could increase code size

more registers → less register spills → less instructions

more instructions → more appropriate instruction → decrease code size

more instructions → larger opcodes → larger code size

2.19

2.19.1 0xBABEF8

2.19.2 0xAAAAAA0

2.19.3 0x00005545

2.20 srl \$t0, \$t0, 11
sll \$t0, \$t0, 26
ori \$t2, \$0, 0x03ff
sll \$t2, \$t2, 16
ori \$t2, \$t2, 0xffff
and \$t1, \$t1, \$t2
or \$t1, \$t1, \$t0

2.21 nor \$t1, \$t2, \$t2

2.22 lw \$t3, 0(\$s1)
sll \$t1, \$t3, 4

2.23 \$t2 = 3

2.24 jump: no, beq: no

2.25**2.25.1** i-type

2.25.2 `addi $t2, $t2, -1`
`beq $t2, $0, loop`

2.26**2.26.1** 20

2.26.2 `i = 10;`
`do {`
 `B += 2;`
 `i = i - 1;`
`} while (i > 0)`

2.26.3 5*N

2.27 `addi $t0, $0, 0`
`beq $0, $0, TEST1`
LOOP1: `addi $t1, $0, 0`
`beq $0, $0, TEST2`
LOOP2: `add $t3, $t0, $t1`
`sll $t2, $t1, 4`
`add $t2, $t2, $s2`
`sw $t3, ($t2)`
`addi $t1, $t1, 1`
TEST2: `slt $t2, $t1, $s1`
`bne $t2, $0, LOOP2`
`addi $t0, $t0, 1`
TEST1: `slt $t2, $t0, $s0`
`bne $t2, $0, LOOP1`

2.28 14 instructions to implement and 158 instructions executed

2.29 `for (i=0; i<100; i++) {`
 `result += MemArray[s0];`
 `s0 = s0 + 4;`
`}`

```

2.30  addi $t1, $s0, 400
LOOP:  lw   $s1, 0($t1)
        add  $s2, $s2, $s1
        addi $t1, $t1, -4
        bne $t1, $s0, LOOP

```

```

2.31 fib:  addi $sp, $sp, -12      # make room on stack
            sw   $ra, 8($sp)       # push $ra
            sw   $s0, 4($sp)       # push $s0
            sw   $a0, 0($sp)       # push $a0 (N)
            bgt  $a0, $0, test2     # if n>0, test if n=1
            add  $v0, $0, $0        # else fib(0) = 0
            j    rtn               #
test2:      addi $t0, $0, 1         #
            bne  $t0, $a0, gen      # if n>1, gen
            add  $v0, $0, $t0       # else fib(1) = 1
            j    rtn               #
gen:        subi $a0, $a0, 1        # n-1
            jal  fib               # call fib(n-1)
            add  $s0, $v0, $0       # copy fib(n-1)
            sub  $a0, $a0, 1        # n-2
            jal  fib               # call fib(n-2)
            add  $v0, $v0, $s0      # fib(n-1)+fib(n-2)
rtn:        lw   $a0, 0($sp)        # pop $a0
            lw   $s0, 4($sp)        # pop $s0
            lw   $ra, 8($sp)        # pop $ra
            addi $sp, $sp, 12       # restore sp
            jr   $ra

```

```

# fib(0) = 12 instructions, fib(1) = 14 instructions,
# fib(N) = 26 + 18N instructions for N >=2

```

2.32 Due to the recursive nature of the code, it is not possible for the compiler to in-line the function call.

```

2.33 after calling function fib:
old $sp -> 0x7fffffff ???
           -4          contents of register $ra for
                        fib(N)
           -8          contents of register $s0 for
                        fib(N)
$sp->      -12         contents of register $a0 for
                        fib(N)
there will be N-1 copies of $ra, $s0 and $a0

```

```

2.34 f: addi $sp,$sp,-12
        sw    $ra,8($sp)
        sw    $s1,4($sp)
        sw    $s0,0($sp)
        move  $s1,$a2
        move  $s0,$a3
        jal   func
        move  $a0,$v0
        add   $a1,$s0,$s1
        jal   func
        lw    $ra,8($sp)
        lw    $s1,4($sp)
        lw    $s0,0($sp)
        addi  $sp,$sp,12
        jr    $ra

```

2.35 We can use the tail-call optimization for the second call to `func`, but then we must restore `$ra`, `$s0`, `$s1`, and `$sp` before that call. We save only one instruction (`jr $ra`).

2.36 Register `$ra` is equal to the return address in the caller function, registers `$sp` and `$s3` have the same values they had when function `f` was called, and register `$t5` can have an arbitrary value. For register `$t5`, note that although our function `f` does not modify it, function `func` is allowed to modify it so we cannot assume anything about the of `$t5` after function `func` has been called.

```

2.37 MAIN: addi $sp, $sp, -4
        sw    $ra, ($sp)
        add   $t6, $0, 0x30 # '0'
        add   $t7, $0, 0x39 # '9'
        add   $s0, $0, $0
        add   $t0, $a0, $0
LOOP:   lb    $t1, ($t0)
        slt   $t2, $t1, $t6
        bne   $t2, $0, DONE
        slt   $t2, $t7, $t1
        bne   $t2, $0, DONE
        sub   $t1, $t1, $t6
        beq   $s0, $0, FIRST
        mul   $s0, $s0, 10
FIRST:  add   $s0, $s0, $t1
        addi  $t0, $t0, 1
        j     LOOP

```



```

DONE:  add  $v0, $s0, $0
        lw   $ra, ($sp)
        addi $sp, $sp, 4
        jr   $ra

```

2.38 0x00000011

2.39 Generally, all solutions are similar:

```

lui $t1, top_16_bits
ori $t1, $t1, bottom_16_bits

```

2.40 No, jump can go up to 0x0FFFFFFC.

2.41 No, range is $0x604 + 0x1FFFC = 0x0002\ 0600$ to $0x604 - 0x20000 = 0xFFFE\ 0604$.

2.42 Yes, range is $0x1FFFF004 + 0x1FFFC = 0x2001F000$ to $0x1FFFF004 - 0x20000 = 1FFDF004$

2.43 trylk: li \$t1,1
 ll \$t0,0(\$a0)
 bnez \$t0,trylk
 sc \$t1,0(\$a0)
 beqz \$t1,trylk
 lw \$t2,0(\$a1)
 slt \$t3,\$t2,\$a2
 bnez \$t3,skip
 sw \$a2,0(\$a1)
 skip: sw \$0,0(\$a0)

2.44 try: ll \$t0,0(\$a1)
 slt \$t1,\$t0,\$a2
 bnez \$t1,skip
 mov \$t0,\$a2
 sc \$t0,0(\$a1)
 beqz \$t0,try
 skip:

2.45 It is possible for one or both processors to complete this code without ever reaching the SC instruction. If only one executes SC, it completes successfully. If both reach SC, they do so in the same cycle, but one SC completes first and then the other detects this and fails.

2.46

2.46.1 Answer is no in all cases. Slows down the computer.

CCT = clock cycle time

ICa = instruction count (arithmetic)

ICls = instruction count (load/store)

ICb = instruction count (branch)

$$\begin{aligned}\text{new CPU time} &= 0.75 * \text{old ICa} * \text{CPIa} * 1.1 * \text{old CCT} \\ &\quad + \text{old IClS} * \text{CPIs} * 1.1 * \text{old CCT} \\ &\quad + \text{old ICb} * \text{CPIb} * 1.1 * \text{old CCT}\end{aligned}$$

The extra clock cycle time adds sufficiently to the new CPU time such that it is not quicker than the old execution time in all cases.

2.46.2 107.04%, 113.43%

2.47

2.47.1 2.6

2.47.2 0.88

2.47.3 0.533333333



3

Solutions

3.1 5730**3.2** 5730**3.3** 0101111011010100

The attraction is that each hex digit contains one of 16 different characters (0–9, A–E). Since with 4 binary bits you can represent 16 different patterns, in hex each digit requires exactly 4 binary bits. And bytes are by definition 8 bits long, so two hex digits are all that are required to represent the contents of 1 byte.

3.4 753**3.5** 7777 (−3777)**3.6** Neither (63)**3.7** Neither (65)**3.8** Overflow (result = −179, which does not fit into an SM 8-bit format)**3.9** $-105 - 42 = -128$ (−147)**3.10** $-105 + 42 = -63$ **3.11** $151 + 214 = 255$ (365)**3.12** 62×12

Step	Action	Multiplier	Multiplicand	Product
0	Initial Vals	001 010	000 000 110 010	000 000 000 000
1	lsb=0, no op	001 010	000 000 110 010	000 000 000 000
	Lshift Mcand	001 010	000 001 100 100	000 000 000 000
	Rshift Mplier	000 101	000 001 100 100	000 000 000 000
2	Prod=Prod+Mcand	000 101	000 001 100 100	000 001 100 100
	Lshift Mcand	000 101	000 011 001 000	000 001 100 100
	Rshift Mplier	000 010	000 011 001 000	000 001 100 100
3	lsb=0, no op	000 010	000 011 001 000	000 001 100 100
	Lshift Mcand	000 010	000 110 010 000	000 001 100 100
	Rshift Mplier	000 001	000 110 010 000	000 001 100 100
4	Prod=Prod+Mcand	000 001	000 110 010 000	000 111 110 100
	Lshift Mcand	000 001	001 100 100 000	000 111 110 100
	Rshift Mplier	000 000	001 100 100 000	000 111 110 100
5	lsb=0, no op	000 000	001 100 100 000	000 111 110 100
	Lshift Mcand	000 000	011 001 000 000	000 111 110 100
	Rshift Mplier	000 000	011 001 000 000	000 111 110 100
6	lsb=0, no op	000 000	110 010 000 000	000 111 110 100
	Lshift Mcand	000 000	110 010 000 000	000 111 110 100
	Rshift Mplier	000 000	110 010 000 000	000 111 110 100

3.13 62×12

Step	Action	Multiplicand	Product/Multiplier
0	Initial Vals	110 010	000 000 001 010
1	lsb=0, no op	110 010	000 000 001 010
	Rshift Product	110 010	000 000 000 101
2	Prod=Prod+Mcand	110 010	110 010 000 101
	Rshift Mplier	110 010	011 001 000 010
3	lsb=0, no op	110 010	011 001 000 010
	Rshift Mplier	110 010	001 100 100 001
4	Prod=Prod+Mcand	110 010	111 110 100 001
	Rshift Mplier	110 010	011 111 010 000
5	lsb=0, no op	110 010	011 111 010 000
	Rshift Mplier	110 010	001 111 101 000
6	lsb=0, no op	110 010	001 111 101 000
	Rshift Mplier	110 010	000 111 110 100

3.14 For hardware, it takes 1 cycle to do the add, 1 cycle to do the shift, and 1 cycle to decide if we are done. So the loop takes $(3 \times A)$ cycles, with each cycle being B time units long.

For a software implementation, it takes 1 cycle to decide what to add, 1 cycle to do the add, 1 cycle to do each shift, and 1 cycle to decide if we are done. So the loop takes $(5 \times A)$ cycles, with each cycle being B time units long.

$$(3 \times 8) \times 4tu = 96 \text{ time units for hardware}$$

$$(5 \times 8) \times 4tu = 160 \text{ time units for software}$$

3.15 It takes B time units to get through an adder, and there will be $A - 1$ adders. Word is 8 bits wide, requiring 7 adders. $7 \times 4tu = 28$ time units.

3.16 It takes B time units to get through an adder, and the adders are arranged in a tree structure. It will require $\log_2(A)$ levels. 8 bit wide word requires 7 adders in 3 levels. $3 \times 4tu = 12$ time units.

3.17 $0x33 \times 0x55 = 0x10EF$. $0x33 = 51$, and $51 = 32 + 16 + 2 + 1$. We can shift $0x55$ left 5 places ($0xAA0$), then add $0x55$ shifted left 4 places ($0x550$), then add $0x55$ shifted left once ($0xAA$), then add $0x55$. $0xAA0 + 0x550 + 0xAA + 0x55 = 0x10EF$. 3 shifts, 3 adds.

(Could also use $0x55$, which is $64 + 16 + 4 + 1$, and shift $0x33$ left 6 times, add to it $0x33$ shifted left 4 times, add to that $0x33$ shifted left 2 times, and add to that $0x33$. Same number of shifts and adds.)

3.18 $74/21 = 3$ remainder 9

Step	Action	Quotient	Divisor	Remainder
0	Initial Vals	000 000	010 001 000 000	000 000 111 100
1	Rem=Rem-Div	000 000	010 001 000 000	101 111 111 100
	Rem<0, R+D, Q<<	000 000	010 001 000 000	000 000 111 100
	Rshift Div	000 000	001 000 100 000	000 000 111 100
2	Rem=Rem-Div	000 000	001 000 100 000	111 000 011 100
	Rem<0, R+D, Q<<	000 000	001 000 100 000	000 000 111 100
	Rshift Div	000 000	000 100 010 000	000 000 111 100
3	Rem=Rem-Div	000 000	000 100 010 000	111 100 101 100
	Rem<0, R+D, Q<<	000 000	000 100 010 000	000 000 111 100
	Rshift Div	000 000	000 010 001 000	000 000 111 100
4	Rem=Rem-Div	000 000	000 010 001 000	111 110 110 100
	Rem<0, R+D, Q<<	000 000	000 010 001 000	000 000 111 100
	Rshift Div	000 000	000 001 000 100	000 000 111 100
5	Rem=Rem-Div	000 000	000 001 000 100	111 111 111 000
	Rem<0, R+D, Q<<	000 000	000 001 000 100	000 000 111 100
	Rshift Div	000 000	000 000 100 010	000 000 111 100
6	Rem=Rem-Div	000 000	000 000 100 010	000 000 011 010
	Rem>0, Q<<1	000 001	000 000 100 010	000 000 011 010
	Rshift Div	000 001	000 000 010 001	000 000 011 010
7	Rem=Rem-Div	000 001	000 000 010 001	000 000 001 001
	Rem>0, Q<<1	000 011	000 000 010 001	000 000 001 001
	Rshift Div	000 011	000 000 001 000	000 000 001 001

3.19. In these solutions a 1 or a 0 was added to the Quotient if the remainder was greater than or equal to 0. However, an equally valid solution is to shift in a 1 or 0, but if you do this you must do a compensating right shift of the remainder (only the remainder, not the entire remainder/quotient combination) after the last step.

 $74/21 = 3$ remainder 11

Step	Action	Divisor	Remainder/Quotient
0	Initial Vals	010 001	000 000 111 100
1	R<<	010 001	000 001 111 000
	Rem=Rem-Div	010 001	111 000 111 000
	Rem<0, R+D	010 001	000 001 111 000
2	R<<	010 001	000 011 110 000
	Rem=Rem-Div	010 001	110 010 110 000
	Rem<0, R+D	010 001	000 011 110 000
3	R<<	010 001	000 111 100 000
	Rem=Rem-Div	010 001	110 110 110 000
	Rem<0, R+D	010 001	000 111 100 000
4	R<<	010 001	001 111 000 000
	Rem=Rem-Div	010 001	111 110 000 000
	Rem<0, R+D	010 001	001 111 000 000

Step	Action	Divisor	Remainder/Quotient
5	R<<	010 001	011 110 000 000
	Rem=Rem-Div	010 001	111 110 000 000
	Rem>0, R0=1	010 001	001 101 000 001
6	R<<	010 001	011 010 000 010
	Rem=Rem-Div	010 001	001 001 000 010
	Rem>0, R0=1	010 001	001 001 000 011

3.20 201326592 in both cases.

3.21 jal 0x00000000

3.22

$$0 \times 0C000000 = 0000\ 1100\ 0000\ 0000\ 0000\ 0000\ 0000\ 0000$$

$$= 0\ 0001\ 1000\ 0000\ 0000\ 0000\ 0000\ 0000\ 000$$

sign is positive

$$\text{exp} = 0 \times 18 = 24 - 127 = -103$$

there is a hidden 1

$$\text{mantissa} = 0$$

$$\text{answer} = 1.0 \times 2^{-103}$$

3.23 $63.25 \times 10^0 = 111111.01 \times 2^0$

normalize, move binary point 5 to the left

$$1.1111101 \times 2^5$$

$$\text{sign} = \text{positive}, \text{exp} = 127 + 5 = 132$$

Final bit pattern: 0 1000 0100 1111 1010 0000 0000 0000 000

$$= 0100\ 0010\ 0111\ 1101\ 0000\ 0000\ 0000\ 0000 = 0x427D0000$$

3.24 $63.25 \times 10^0 = 111111.01 \times 2^0$

normalize, move binary point 5 to the left

$$1.1111101 \times 2^5$$

$$\text{sign} = \text{positive}, \text{exp} = 1023 + 5 = 1028$$

Final bit pattern:

0 100 0000 0100 1111 1010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000
0000

$$= 0x404FA00000000000$$

3.25 $63.25 \times 10^0 = 111111.01 \times 2^0 = 3F.40 \times 16^0$

move hex point 2 to the left

$$.3F40 \times 16^2$$

sign = positive, exp = 64+2

Final bit pattern: 01000010001111110100000000000000

3.26 $-1.5625 \times 10^{-1} = -.15625 \times 10^0$

$$= -.00101 \times 2^0$$

move the binary point 2 to the right

$$= -.101 \times 2^{-2}$$

exponent = -2, fraction = -.101000000000000000000000

answer: 111111111110101100000000000000000000

3.27 $-1.5625 \times 10^{-1} = -.15625 \times 10^0$

$$= -.00101 \times 2^0$$

move the binary point 3 to the right, $= -1.01 \times 2^{-3}$

exponent = -3 = -3+15 = 12, fraction = -.0100000000

answer: 1011000100000000

3.28 $-1.5625 \times 10^{-1} = -.15625 \times 10^0$

$$= -.00101 \times 2^0$$

move the binary point 2 to the right

$$= -.101 \times 2^{-2}$$

exponent = -2, fraction = -.101000000000000000000000

answer: 10110000000000000000000000000101

3.29 $2.6125 \times 10^1 + 4.150390625 \times 10^{-1}$

$$2.6125 \times 10^1 = 26.125 = 11010.001 = 1.1010001000 \times 2^4$$

$$4.150390625 \times 10^{-1} = .4150390625 = .011010100111 = 1.1010100111 \times 2^{-2}$$

Shift binary point 6 to the left to align exponents,

GR

1.1010001000 00

1.0000011010 10 0111 (Guard 5 1, Round 5 0,
Sticky 5 1)

1.1010100010 10

In this case the extra bit (G,R,S) is more than half of the least significant bit (0).

Thus, the value is rounded up.

$$1.1010100011 \times 2^4 = 11010.100011 \times 2^0 = 26.546875 = 2.6546875 \times 10^1$$

$$\mathbf{3.30} \quad -8.0546875 \times -1.79931640625 \times 10^{-1}$$

$$-8.0546875 = -1.0000000111 \times 2^3$$

$$-1.79931640625 \times 10^{-1} = -1.0111000010 \times 2^{-3}$$

$$\text{Exp: } -3 + 3 = 0, 0 + 16 = 16 \text{ (10000)}$$

Signs: both negative, result positive

Fraction:

$$\begin{array}{r} 1.0000000111 \\ \times 1.0111000010 \\ \hline \end{array}$$

0000000000

10000000111

0000000000

0000000000

0000000000

0000000000

10000000111

10000000111

10000000111

0000000000

10000000111

1.01110011000001001110

1.0111001100 00 01001110 Guard = 0, Round = 0, Sticky = 1:NoRnd

$$1.0111001100 \times 2^0 = 0100000111001100 \text{ (} 1.0111001100 = 1.44921875 \text{)}$$

$$-8.0546875 \times -.179931640625 = 1.4492931365966796875$$

Some information was lost because the result did not fit into the available 10-bit field. Answer (only) off by .0000743865966796875

3.31 $8.625 \times 10^1 / -4.875 \times 10^0$

$$8.625 \times 10^1 = 1.0101100100 \times 2^6$$

$$-4.875 = -1.0011100000 \times 2^2$$

$$\text{Exponent} = 6 - 2 = 4, 4 + 15 = 19 \text{ (10011)}$$

Signs: one positive, one negative, result negative

Fraction:

$$\begin{array}{r}
 \\
 10011100000. 10101100100.00000000000000000000 \\
 -10011100000. \\
 \hline
 10000100.0000 \\
 -1001110.0000 \\
 \hline
 1100110.00000 \\
 -100111.00000 \\
 \hline
 1111.0000000 \\
 -1001.1100000 \\
 \hline
 101.01000000 \\
 -100.11100000 \\
 \hline
 000.011000000000 \\
 - .010011100000 \\
 \hline
 .0001001000000000 \\
 - .000010011100000 \\
 \hline
 .0000100001000000 \\
 - .0000010011100000 \\
 \hline
 .00000011011000000 \\
 - .00000010011100000 \\
 \hline
 .00000000110000000
 \end{array}$$

1.000110110001001111 Guard=0, Round=1, Sticky=1: No Round, fix sign

$$-1.0001101100 \times 2^4 = 1101000001101100 = 10001.101100 = -17.6875$$

$$86.25 / -4.875 = -17.692307692307$$

Some information was lost because the result did not fit into the available 10-bit field. Answer off by .00480769230

3.32 $(3.984375 \times 10^{-1} + 3.4375 \times 10^{-1}) + 1.771 \times 10^3$

$$3.984375 \times 10^{-1} = 1.1001100000 \times 2^{-2}$$

$$3.4375 \times 10^{-1} = 1.0110000000 \times 2^{-2}$$

$$1.771 \times 10^3 = 1771 = 1.1011101011 \times 2^{10}$$

shift binary point of smaller left 12 so exponents match

$$(A) \quad 1.1001100000$$

$$(B) \quad + 1.0110000000$$

$$10.1111100000 \text{ Normalize,}$$

$$(A+B) \quad 1.0111110000 \times 2^{-1}$$

$$(C) \quad + 1.1011101011$$

$$(A+B) \quad .0000000000 \quad 10 \quad 111110000 \quad \text{Guard} = 1, \\ \text{Round} = 0, \text{Sticky} = 1$$

$$(A+B)+C \quad + 1.1011101011 \quad 10 \quad 1 \quad \text{Round up}$$

$$(A+B)+C = 1.1011101100 \times 2^{10} = 0110101011101100 = 1772$$

3.33 $3.984375 \times 10^{-1} + (3.4375 \times 10^{-1} + 1.771 \times 10^3)$

$$3.984375 \times 10^{-1} = 1.1001100000 \times 2^{-2}$$

$$3.4375 \times 10^{-1} = 1.0110000000 \times 2^{-2}$$

$$1.771 \times 10^3 = 1771 = 1.1011101011 \times 2^{10}$$

shift binary point of smaller left 12 so exponents match

$$(B) \quad .0000000000 \quad 01 \quad 0110000000 \quad \text{Guard} = 0, \\ \text{Round} = 1, \text{Sticky} = 1$$

$$(C) \quad + 1.1011101011$$

$$(B+C) \quad + 1.1011101011$$

```

(A)      .0000000000  011001100000
      -----
A+(B+C) +1.1011101011  No round
A+(B+C) +1.1011101011   $\times 2^{10} = 0110101011101011 = 1771$ 

```

3.34 No, they are not equal: $(A+B)+C = 1772$, $A+(B+C) = 1771$ (steps shown above).

Exact: $.398437 + .34375 + 1771 = 1771.742187$

3.35 $(3.41796875 \times 10^{-3} \times 6.34765625 \times 10^{-3}) \times 1.05625 \times 10^2$

(A) $3.41796875 \times 10^{-3} = 1.1100000000 \times 2^{-9}$

(B) $4.150390625 \times 10^{-3} = 1.0001000000 \times 2^{-8}$

(C) $1.05625 \times 10^2 = 1.1010011010 \times 2^6$

Exp: $-9-8 = -17$

Signs: both positive, result positive

Fraction:

```

(A)      1.1100000000
(B)       $\times 1.0001000000$ 
      -----

```

```

      11100000000
      11100000000
      -----
      1.110111000000000000000000
A×B      1.1101110000 00 00000000
Guard = 0, Round = 0, Sticky = 0: No Round

```

$A \times B$ $1.1101110000 \times 2^{-17}$ UNDERFLOW: Cannot represent number

3.36 $3.41796875 \times 10^{-3} \times (6.34765625 \times 10^{-3} \times 1.05625 \times 10^2)$

(A) $3.41796875 \times 10^{-3} = 1.1100000000 \times 2^{-9}$

(B) $4.150390625 \times 10^{-3} = 1.0001000000 \times 2^{-8}$

(C) $1.05625 \times 10^2 = 1.1010011010 \times 2^6$

Exp: $-8+6 = -2$

Signs: both positive, result positive

Fraction:

(B) 1.0001000000

(C) \times 1.1010011010

```

-----
          10001000000
        10001000000
        10001000000
       10001000000
      10001000000
     10001000000
    10001000000
   10001000000
  10001000000
 10001000000
-----

```

1.110000001110100000000

1.1100000011 10 100000000 Guard 5 1, Round 5 0, Sticky

5 1: Round

$B \times C = 1.1100000100 \times 2^{-2}$

Exp: $-9 - 2 = -11$

Signs: both positive, result positive

Fraction:

(A) 1.1100000000

(B \times C) \times 1.1100000100

```

-----
          11100000000
        11100000000
        11100000000
       11100000000
      11100000000
     11100000000
    11100000000
   11100000000
  11100000000
 11100000000
-----

```

11.00010001110000000000

Normalize, add 1 to exponent

1.1000100011 10 0000000000 Guard=1, Round=0, Sticky=0:

Round to even

$A \times (B \times C) = 1.1000100100 \times 2^{-10}$

3.37 b) No:

$$A \times B = 1.1101110000 \times 2^{-17} \text{ UNDERFLOW: Cannot represent}$$

$$A \times (B \times C) = 1.1000100100 \times 2^{-10}$$

A and B are both small, so their product does not fit into the 16-bit floating point format being used.

3.38 $1.666015625 \times 10^0 \times (1.9760 \times 10^4 - 1.9744 \times 10^4)$

$$(A) \quad 1.666015625 \times 10^0 = 1.1010101010 \times 2^0$$

$$(B) \quad 1.9760 \times 10^4 = 1.0011010011 \times 2^{14}$$

$$(C) \quad -1.9744 \times 10^4 = -1.0011010010 \times 2^{14}$$

Exponents match, no shifting necessary

$$(B) \quad 1.0011010011$$

$$(C) \quad -1.0011010010$$

$$(B+C) \quad 0.0000000001 \times 2^{14}$$

$$(B+C) \quad 1.0000000000 \times 2^4$$

$$\text{Exp: } 0+4 = 4$$

Signs: both positive, result positive

Fraction:

$$(A) \quad 1.1010101010$$

$$(B+C) \quad \times 1.0000000000$$

$$11010101010$$

$$1.10101010100000000000$$

$$A \times (B+C) \quad 1.1010101010 \quad 0000000000 \quad \text{Guard} = 0, \text{ Round} = 0, \text{ sticky} = 0: \text{ No round}$$

$$A \times (B+C) \quad 1.1010101010 \times 2^4$$

3.39 $1.666015625 \times 10^0 \times (1.9760 \times 10^4 - 1.9744 \times 10^4)$

$$(A) \quad 1.666015625 \times 10^0 = 1.1010101010 \times 2^0$$

$$(B) \quad 1.9760 \times 10^4 = 1.0011010011 \times 2^{14}$$

$$(C) -1.9744 \times 10^4 = -1.0011010010 \times 2^{14}$$

$$\text{Exp: } 0+14 = 14$$

Signs: both positive, result positive

Fraction:

$$(A) \quad 1.1010101010$$

$$(B) \quad \times 1.0011010011$$

$$\begin{array}{r}
11010101010 \\
11010101010 \\
11010101010 \\
11010101010 \\
11010101010 \\
11010101010 \\
11010101010 \\
11010101010 \\
\hline
10.0000001001100001111 \text{ Normalize, add 1 to} \\
 \text{exponent}
\end{array}$$

$$A \times B \quad 1.0000000100 \ 11 \ 00001111 \text{ Guard} = 1, \text{ Round} = 1, \\ \text{Sticky} = 1: \text{ Round}$$

$$A \times B \quad 1.0000000101 \times 2^{15}$$

$$\text{Exp: } 0+14=14$$

Signs: one negative, one positive, result negative

Fraction:

$$(A) \quad 1.1010101010$$

$$(C) \quad \times 1.0011010010$$

$$\begin{array}{r}
11010101010 \\
11010101010 \\
11010101010 \\
11010101010 \\
11010101010 \\
11010101010 \\
\hline
10.0000000111110111010
\end{array}$$

Normalize, add 1 to exponent

$$A \times C \quad 1.0000000011 \ 11 \ 101110100$$

Guard = 1, Round = 1, Sticky = 1: Round

$$A \times C \quad -1.0000000100 \times 2^{15}$$

$$A \times B \quad 1.0000000101 \times 2^{15}$$

$$A \times C \quad -1.0000000100 \times 2^{15}$$

$$A \times B + A \times C \quad .0000000001 \times 2^{15}$$

$$A \times B + A \times C \quad 1.0000000000 \times 2^5$$

3.40 b) No:

$$A \times (B+C) = 1.1010101010 \times 2^4 = 26.65625, \text{ and } (A \times B) + (A \times C) = 1.0000000000 \times 2^5 = 32$$

$$\text{Exact: } 1.666015625 \times (19,760 - 19,744) = 26.65625$$

3.41

Answer	sign	exp	Exact?
1 01111101 000000000000000000000000	-	-2	Yes

3.42 $b+b+b+b = -1$

$$b \times 4 = -1$$

They are the same

3.43 0101 0101 0101 0101 0101 0101

No

3.44 0011 0011 0011 0011 0011 0011

No

3.45 0101 0000 0000 0000 0000 0000

0.5

Yes

3.46 01010 00000 00000 00000

0.A

Yes

3.47 Instruction assumptions:

- (1) 8-lane 16-bit multiplies
- (2) sum reductions of the four most significant 16-bit values
- (3) shift and bitwise operations
- (4) 128-, 64-, and 32-bit loads and stores of most significant bits

Outline of solution:

```
load register F[bits 127:0] = f[3..0] & f[3..0] (64-bit load)
load register A[bits 127:0] = sig_in[7..0] (128-bit load)
```

```
for i = 0 to 15 do
  load register B[bits 127:0] = sig_in[(i*8+7..i*8]
  (128-bit load)

  for j = 0 to 7 do
    (1) eight-lane multiply C[bits 127:0] = A*F
    (eight 16-bit multiplies)
    (2) set D[bits 15:0] = sum of the four 16-bit values
    in C[bits 63:0] (reduction of four 16-bit values)
    (3) set D[bits 31:16] = sum of the four 16-bit
    values in C[bits 127:64] (reduction of four 16-
    bit values)
    (4) store D[bits 31:0] to sig_out (32-bit store)
    (5) set A = A shifted 16 bits to the left
    (6) set E = B shifted 112 shifts to the right
    (7) set A = A OR E
    (8) set B = B shifted 16 bits to the left
  end for
end for
```


4

Solutions

4.1

4.1.1 The values of the signals are as follows:

RegWrite	MemRead	ALUMux	MemWrite	ALUop	RegMux	Branch
0	0	1 (Imm)	1	ADD	X	0

ALUMux is the control signal that controls the Mux at the ALU input, 0 (Reg) selects the output of the register file, and 1 (Imm) selects the immediate from the instruction word as the second input to the ALU.

RegMux is the control signal that controls the Mux at the Data input to the register file, 0 (ALU) selects the output of the ALU, and 1 (Mem) selects the output of memory.

A value of X is a “don’t care” (does not matter if signal is 0 or 1)

4.1.2 All except branch Add unit and write port of the Registers

4.1.3 Outputs that are not used: Branch Add, write port of Registers

No outputs: None (all units produce outputs)

4.2

4.2.1 This instruction uses instruction memory, both register read ports, the ALU to add Rd and Rs together, data memory, and write port in Registers.

4.2.2 None. This instruction can be implemented using existing blocks.

4.2.3 None. This instruction can be implemented without adding new control signals. It only requires changes in the Control logic.

4.3

4.3.1 Clock cycle time is determined by the critical path, which for the given latencies happens to be to get the data value for the load instruction: I-Mem (read instruction), Regs (takes longer than Control), Mux (select ALU input), ALU, Data Memory, and Mux (select value from memory to be written into Registers). The latency of this path is $400 \text{ ps} + 200 \text{ ps} + 30 \text{ ps} + 120 \text{ ps} + 350 \text{ ps} + 30 \text{ ps} = 1130 \text{ ps}$. 1430 ps ($1130 \text{ ps} + 300 \text{ ps}$, ALU is on the critical path).

4.3.2 The speedup comes from changes in clock cycle time and changes to the number of clock cycles we need for the program: We need 5% fewer cycles for a program, but cycle time is 1430 instead of 1130, so we have a speedup of $(1/0.95) \cdot (1130/1430) = 0.83$, which means we actually have a slowdown.

4.3.3 The cost is always the total cost of all components (not just those on the critical path, so the original processor has a cost of I-Mem, Regs, Control, ALU, D-Mem, 2 Add units and 3 Mux units, for a total cost of $1000 + 200 + 500 + 100 + 2000 + 2 \cdot 30 + 3 \cdot 10 = 3890$.

We will compute cost relative to this baseline. The performance relative to this baseline is the speedup we previously computed, and our cost/performance relative to the baseline is as follows:

New Cost: $3890 + 600 = 4490$

Relative Cost: $4490/3890 = 1.15$

Cost/Performance: $1.15/0.83 = 1.39$. We are paying significantly more for significantly worse performance; the cost/performance is a lot worse than with the unmodified processor.

4.4

4.4.1 I-Mem takes longer than the Add unit, so the clock cycle time is equal to the latency of the I-Mem:

200 ps

4.4.2 The critical path for this instruction is through the instruction memory, Sign-extend and Shift-left-2 to get the offset, Add unit to compute the new PC, and Mux to select that value instead of PC+4. Note that the path through the other Add unit is shorter, because the latency of I-Mem is longer than the latency of the Add unit. We have:

$200 \text{ ps} + 15 \text{ ps} + 10 \text{ ps} + 70 \text{ ps} + 20 \text{ ps} = 315 \text{ ps}$

4.4.3 Conditional branches have the same long-latency path that computes the branch address as unconditional branches do. Additionally, they have a long-latency path that goes through Registers, Mux, and ALU to compute the PCSrc condition. The critical path is the longer of the two, and the path through PCSrc is longer for these latencies:

$200 \text{ ps} + 90 \text{ ps} + 20 \text{ ps} + 90 \text{ ps} + 20 \text{ ps} = 420 \text{ ps}$

4.4.4 PC-relative branches.

4.4.5 PC-relative unconditional branch instructions. We saw in part c that this is not on the critical path of conditional branches, and it is only needed for PC-relative branches. Note that MIPS does not have actual unconditional branches (bne zero,zero,Label plays that role so there is no need for unconditional branch opcodes) so for MIPS the answer to this question is actually “None”.

4.4.6 Of the two instructions (BNE and ADD), BNE has a longer critical path so it determines the clock cycle time. Note that every path for ADD is shorter than or equal to the corresponding path for BNE, so changes in unit latency

will not affect this. As a result, we focus on how the unit's latency affects the critical path of BNE.

This unit is not on the critical path, so the only way for this unit to become critical is to increase its latency until the path for address computation through sign extend, shift left, and branch add becomes longer than the path for PCSrc through registers, Mux, and ALU. The latency of Regs, Mux, and ALU is 200 ps and the latency of Sign-extend, Shift-left-2, and Add is 95 ps, so the latency of Shift-left-2 must be increased by 105 ps or more for it to affect clock cycle time.

4.5

4.5.1 The data memory is used by LW and SW instructions, so the answer is:

$$25\% + 10\% = 35\%$$

4.5.2 The sign-extend circuit is actually computing a result in every cycle, but its output is ignored for ADD and NOT instructions. The input of the sign-extend circuit is needed for ADDI (to provide the immediate ALU operand), BEQ (to provide the PC-relative offset), and LW and SW (to provide the offset used in addressing memory) so the answer is:

$$20\% + 25\% + 25\% + 10\% = 80\%$$

4.6

4.6.1 To test for a stuck-at-0 fault on a wire, we need an instruction that puts that wire to a value of 1 and has a different result if the value on the wire is stuck at zero:

If this signal is stuck at zero, an instruction that writes to an odd-numbered register will end up writing to the even-numbered register. So if we place a value of zero in R30 and a value of 1 in R31, and then execute ADD R31,R30,R30 the value of R31 is supposed to be zero. If bit 0 of the Write Register input to the Registers unit is stuck at zero, the value is written to R30 instead and R31 will be 1.

4.6.2 The test for stuck-at-zero requires an instruction that sets the signal to 1, and the test for stuck-at-1 requires an instruction that sets the signal to 0. Because the signal cannot be both 0 and 1 in the same cycle, we cannot test the same signal simultaneously for stuck-at-0 and stuck-at-1 using only one instruction. The test for stuck-at-1 is analogous to the stuck-at-0 test:

We can place a value of zero in R31 and a value of 1 in R30, then use ADD R30,R31,R31 which is supposed to place 0 in R30. If this signal is stuck-at-1, the write goes to R31 instead, so the value in R30 remains 1.

4.6.3 We need to rewrite the program to use only odd-numbered registers.

4.6.4 To test for this fault, we need an instruction whose MemRead is 1, so it has to be a load. The instruction also needs to have RegDst set to 0, which is the case for loads. Finally, the instruction needs to have a different result if

MemRead is set to 0. For a load, MemRead=0 result in not reading memory, so the value placed in the register is “random” (whatever happened to be at the output of the memory unit). Unfortunately, this “random” value can be the same as the one already in the register, so this test is not conclusive.

- 4.6.5** To test for this fault, we need an instruction whose Jump is 1, so it has to be the jump instruction. However, for the jump instruction the RegDst signal is “don’t care” because it does not write to any registers, so the implementation may or may not allow us to set RegDst to 0 so we can test for this fault. As a result, we cannot reliably test for this fault.

4.7

4.7.1

Sign-extend	Jump's shift-left-2
000000000000000000000000010100	0001100010000000000001010000

4.7.2

ALUOp[1-0]	Instruction[5-0]
00	010100

4.7.3

New PC	Path
PC+4	PC to Add (PC+4) to branch Mux to jump Mux to PC

4.7.4

WrReg Mux	ALU Mux	Mem/ALU Mux	Branch Mux	Jump Mux
2 or 0 (RegDst is X)	20	X	PC+4	PC+4

4.7.5

ALU	Add (PC+4)	Add (Branch)
-3 and 20	PC and 4	PC+4 and 20*4

4.7.6

Read Register 1	Read Register 2	Write Register	Write Data	RegWrite
-----------------	-----------------	----------------	------------	----------

4.8

4.8.1

Pipelined	Single-cycle
350 ps	1250 ps

4.8.2

Pipelined	Single-cycle
1750 ps	1250 ps

4.8.3

Stage to split	New clock cycle time
ID	300 ps

4.8.4

a.	35%
----	-----

4.8.5

a.	65%
----	-----

4.8.6 We already computed clock cycle times for pipelined and single cycle organizations, and the multi-cycle organization has the same clock cycle time as the pipelined organization. We will compute execution times relative to the pipelined organization. In single-cycle, every instruction takes one (long) clock cycle. In pipelined, a long-running program with no pipeline stalls completes one instruction in every cycle. Finally, a multi-cycle organization completes a LW in 5 cycles, a SW in 4 cycles (no WB), an ALU instruction in 4 cycles (no MEM), and a BEQ in 4 cycles (no WB). So we have the speedup of pipeline

	Multi-cycle execution time is X times pipelined execution time, where X is:	Single-cycle execution time is X times pipelined execution time, where X is:
a.	$0.20 \times 5 + 0.80 \times 4 = 4.20$	$1250 \text{ ps} / 350 \text{ ps} = 3.57$

4.9**4.9.1**

Instruction sequence	Dependences
I1: OR R1,R2,R3 I2: OR R2,R1,R4 I3: OR R1,R1,R2	RAW on R1 from I1 to I2 and I3 RAW on R2 from I2 to I3 WAR on R2 from I1 to I2 WAR on R1 from I2 to I3 WAW on R1 from I1 to I3

4.9.2 In the basic five-stage pipeline WAR and WAW dependences do not cause any hazards. Without forwarding, any RAW dependence between an instruction and the next two instructions (if register read happens in the second half of the clock cycle and the register write happens in the first half). The code that eliminates these hazards by inserting NOP instructions is:

Instruction sequence	
OR R1,R2,R3 NOP NOP OR R2,R1,R4 NOP NOP OR R1,R1,R2	Delay I2 to avoid RAW hazard on R1 from I1 Delay I3 to avoid RAW hazard on R2 from I2

- 4.9.3** With full forwarding, an ALU instruction can forward a value to EX stage of the next instruction without a hazard. However, a load cannot forward to the EX stage of the next instruction (by can to the instruction after that). The code that eliminates these hazards by inserting NOP instructions is:

Instruction sequence	
OR R1,R2,R3	No RAW hazard on R1 from I1 (forwarded) No RAW hazard on R2 from I2 (forwarded)
OR R2,R1,R4	
OR R1,R1,R2	

- 4.9.4** The total execution time is the clock cycle time times the number of cycles. Without any stalls, a three-instruction sequence executes in 7 cycles (5 to complete the first instruction, then one per instruction). The execution without forwarding must add a stall for every NOP we had in 4.9.2, and execution forwarding must add a stall cycle for every NOP we had in 4.9.3. Overall, we get:

No forwarding	With forwarding	Speedup due to forwarding
$(7 + 4) * 180 \text{ ps} = 1980 \text{ ps}$	$7 * 240 \text{ ps} = 1680 \text{ ps}$	1.18

- 4.9.5** With ALU-ALU-only forwarding, an ALU instruction can forward to the next instruction, but not to the second-next instruction (because that would be forwarding from MEM to EX). A load cannot forward at all, because it determines the data value in MEM stage, when it is too late for ALU-ALU forwarding. We have:

Instruction sequence	
OR R1,R2,R3	ALU-ALU forwarding of R1 from I1 ALU-ALU forwarding of R2 from I2
OR R2,R1,R4	
OR R1,R1,R2	

4.9.6

No forwarding	With ALU-ALU forwarding only	Speedup with ALU-ALU forwarding
$(7 + 4) * 180 \text{ ps} = 1980 \text{ ps}$	$7 * 210 \text{ ps} = 1470 \text{ ps}$	1.35

4.10

- 4.10.1** In the pipelined execution shown below, *** represents a stall when an instruction cannot be fetched because a load or store instruction is using the memory in that cycle. Cycles are represented from left to right, and for each instruction we show the pipeline stage it is in during that cycle:

Instruction	Pipeline Stage	Cycles
SW R16,12(R6)	IF ID EX MEM WB	11
LW R16,8(R6)	IF ED EX MEM WB	
BEQ R5,R4,Lb1	IF ID EX MEM WB	
ADD R5,R1,R4	*** ** IF ID EX MEM WB	
SLT R5,R15,R4	IF ID EX MEM WB	

We can not add NOPs to the code to eliminate this hazard – NOPs need to be fetched just like any other instructions, so this hazard must be addressed with a hardware hazard detection unit in the processor.

- 4.10.2** This change only saves one cycle in an entire execution without data hazards (such as the one given). This cycle is saved because the last instruction finishes one cycle earlier (one less stage to go through). If there were data hazards from loads to other instructions, the change would help eliminate some stall cycles.

Instructions Executed	Cycles with 5 stages	Cycles with 4 stages	Speedup
5	$4 + 5 = 9$	$3 + 5 = 8$	$9/8 = 1.13$

- 4.10.3** Stall-on-branch delays the fetch of the next instruction until the branch is executed. When branches execute in the EXE stage, each branch causes two stall cycles. When branches execute in the ID stage, each branch only causes one stall cycle. Without branch stalls (e.g., with perfect branch prediction) there are no stalls, and the execution time is 4 plus the number of executed instructions. We have:

Instructions Executed	Branches Executed	Cycles with branch in EXE	Cycles with branch in ID	Speedup
5	1	$4 + 5 + 1*2 = 11$	$4 + 5 + 1*1 = 10$	$11/10 = 1.10$

- 4.10.4** The number of cycles for the (normal) 5-stage and the (combined EX/MEM) 4-stage pipeline is already computed in 4.10.2. The clock cycle time is equal to the latency of the longest-latency stage. Combining EX and MEM stages affects clock time only if the combined EX/MEM stage becomes the longest-latency stage:

Cycle time with 5 stages	Cycle time with 4 stages	Speedup
200 ps (IF)	210 ps (MEM + 20 ps)	$(9*200)/(8*210) = 1.07$

4.10.5

New ID latency	New EX latency	New cycle time	Old cycle time	Speedup
180 ps	140 ps	200 ps (IF)	200 ps (IF)	$(11*200)/(10*200) = 1.10$

- 4.10.6** The cycle time remains unchanged: a 20 ps reduction in EX latency has no effect on clock cycle time because EX is not the longest-latency stage. The change does affect execution time because it adds one additional stall cycle to each branch. Because the clock cycle time does not improve but

the number of cycles increases, the speedup from this change will be below 1 (a slowdown). In 4.10.3 we already computed the number of cycles when branch is in EX stage. We have:

	Cycles with branch in EX	Execution time (branch in EX)	Cycles with branch in MEM	Execution time (branch in MEM)	Speedup
a.	$4 + 5 + 1 \cdot 2 = 11$	$11 \cdot 200 \text{ ps} = 2200 \text{ ps}$	$4 + 5 + 1 \cdot 3 = 12$	$12 \cdot 200 \text{ ps} = 2400 \text{ ps}$	0.92

4.11

4.11.1

LW R1,0(R1)	WB
LW R1,0(R1)	EX MEM WB
BEQ R1,R0,Loop	ID *** EX MEM WB
LW R1,0(R1)	IF *** ID EX MEM WB
AND R1,R1,R2	IF ID *** EX MEM WB
LW R1,0(R1)	IF *** ID EX MEM
LW R1,0(R1)	IF ID ***
BEQ R1,R0,Loop	IF ***

4.11.2 In a particular clock cycle, a pipeline stage is not doing useful work if it is stalled or if the instruction going through that stage is not doing any useful work there. In the pipeline execution diagram from 4.11.1, a stage is stalled if its name is not shown for a particular cycles, and stages in which the particular instruction is not doing useful work are marked in blue. Note that a BEQ instruction is doing useful work in the MEM stage, because it is determining the correct value of the next instruction's PC in that stage. We have:

Cycles per loop iteration	Cycles in which all stages do useful work	% of cycles in which all stages do useful work
8	0	0%

4.12

4.12.1 Dependences to the 1st next instruction result in 2 stall cycles, and the stall is also 2 cycles if the dependence is to both 1st and 2nd next instruction. Dependences to only the 2nd next instruction result in one stall cycle. We have:

CPI	Stall Cycles
$1 + 0.35 \cdot 2 + 0.15 \cdot 1 = 1.85$	46% (0.85/1.85)

4.12.2 With full forwarding, the only RAW data dependences that cause stalls are those from the MEM stage of one instruction to the 1st next instruction. Even this dependences causes only one stall cycle, so we have:

CPI	Stall Cycles
$1 + 0.20 = 1.20$	17% (0.20/1.20)

4.12.3 With forwarding only from the EX/MEM register, EX to 1st dependences can be satisfied without stalls but any other dependences (even when together with EX to 1st) incur a one-cycle stall. With forwarding only from the MEM/WB register, EX to 2nd dependences incur no stalls. MEM to 1st dependences still incur a one-cycle stall, and EX to 1st dependences now incur one stall cycle because we must wait for the instruction to complete the MEM stage to be able to forward to the next instruction. We compute stall cycles per instructions for each case as follows:

EX/MEM	MEM/WB	Fewer stall cycles with
$0.2 + 0.05 + 0.1 + 0.1 = 0.45$	$0.05 + 0.2 + 0.1 = 0.35$	MEM/WB

4.12.4 In 4.12.1 and 4.12.2 we have already computed the CPI without forwarding and with full forwarding. Now we compute time per instruction by taking into account the clock cycle time:

Without forwarding	With forwarding	Speedup
$1.85 * 150 \text{ ps} = 277.5 \text{ ps}$	$1.20 * 150 \text{ ps} = 180 \text{ ps}$	1.54

4.12.5 We already computed the time per instruction for full forwarding in 4.12.4. Now we compute time-per instruction with time-travel forwarding and the speedup over full forwarding:

With full forwarding	Time-travel forwarding	Speedup
$1.20 * 150 \text{ ps} = 180 \text{ ps}$	$1 * 250 \text{ ps} = 250 \text{ ps}$	0.72

4.12.6

EX/MEM	MEM/WB	Shorter time per instruction with
$1.45 * 150 \text{ ps} = 217.5$	$1.35 * 150 \text{ ps} = 202.5 \text{ ps}$	MEM/WB

4.13

4.13.1

```

ADD R5,R2,R1
NOP
NOP
LW R3,4(R5)
LW R2,0(R2)
NOP
OR R3,R5,R3
NOP
NOP
SW R3,0(R5)

```

4.13.2 We can move up an instruction by swapping its place with another instruction that has no dependences with it, so we can try to fill some NOP slots with such instructions. We can also use R7 to eliminate WAW or WAR dependences so we can have more instructions to move up.

I1: ADD R5,R2,R1	Moved up to fill NOP slot
I3: LW R2,0(R2)	
NOP	Had to add another NOP here, so there is no performance gain
I2: LW R3,4(R5)	
NOP	
NOP	
I4: OR R3,R5,R3	
NOP	
NOP	
I5: SW R3,0(R5)	

4.13.3 With forwarding, the hazard detection unit is still needed because it must insert a one-cycle stall whenever the load supplies a value to the instruction that immediately follows that load. Without the hazard detection unit, the instruction that depends on the immediately preceding load gets the stale value the register had before the load instruction.

Code executes correctly (for both loads, there is no RAW dependence between the load and the next instruction).

4.13.4 The outputs of the hazard detection unit are PCWrite, IF/IDWrite, and ID/EXZero (which controls the Mux after the output of the Control unit). Note that IF/IDWrite is always equal to PCWrite, and ED/ExZero is always the opposite of PCWrite. As a result, we will only show the value of PCWrite for each cycle. The outputs of the forwarding unit is ALUin1 and ALUin2, which control Muxes that select the first and second input of the ALU. The three possible values for ALUin1 or ALUin2 are 0 (no forwarding), 1 (forward ALU output from previous instruction), or 2 (forward data value for second-previous instruction). We have:

Instruction sequence	First five cycles					Signals
	1	2	3	4	5	
ADD R5,R2,R1	IF	ID	EX	MEM	WB	1: PCWrite=1, ALUin1=X, ALUin2=X
LW R3,4(R5)		IF	ID	EX	MEM	2: PCWrite=1, ALUin1=X, ALUin2=X
LW R2,0(R2)			IF	ID	EX	3: PCWrite=1, ALUin1=0, ALUin2=0
OR R3,R5,R3				IF	ID	4: PCWrite=1, ALUin1=1, ALUin2=0
SW R3,0(R5)					IF	5: PCWrite=1, ALUin1=0, ALUin2=0

4.13.5 The instruction that is currently in the ID stage needs to be stalled if it depends on a value produced by the instruction in the EX or the instruction in the MEM stage. So we need to check the destination register of these two instructions. For the instruction in the EX stage, we need to check Rd for R-type instructions and Rd for loads. For the instruction in the MEM stage, the destination register is already selected (by the Mux in the EX stage) so we need to check that register number (this is the bottommost output of the EX/MEM pipeline register). The additional inputs to the hazard detection unit

are register Rd from the ID/EX pipeline register and the output number of the output register from the EX/MEM pipeline register. The Rt field from the ID/EX register is already an input of the hazard detection unit in Figure 4.60.

No additional outputs are needed. We can stall the pipeline using the three output signals that we already have.

4.13.6 As explained for part e, we only need to specify the value of the PCWrite signal, because IF/IDWrite is equal to PCWrite and the ID/EXzero signal is its opposite. We have:

Instruction sequence	First five cycles					Signals
	1	2	3	4	5	
ADD R5,R2,R1	IF	ID	EX	MEM	WB	1: PCWrite=1
LW R3,4(R5)		IF	ID	***	***	2: PCWrite=1
LW R2,0(R2)			IF	***	***	3: PCWrite=1
OR R3,R5,R3					***	4: PCWrite=0
SW R3,0(R5)						5: PCWrite=0

4.14

4.14.1

Executed Instructions	Pipeline Cycles													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW R2,0(R1)	IF	ID	EX	MEM	WB									
BEQ R2,R0,Label12 (NT)		IF	ID	***	EX	MEM	WB							
LW R3,0(R2)						IF	ID	EX	MEM	WB				
BEQ R3,R0,Label11 (T)							IF	ID	***	EX	MEM	WB		
BEQ R2,R0,Label12 (T)								IF	***	ID	EX	MEM	WB	
SW R1,0(R2)										IF	ID	EX	MEM	WB

4.14.2

Executed Instructions	Pipeline Cycles													
	1	2	3	4	5	6	7	8	9	10	11	12	13	14
LW R2,0(R1)	IF	ID	EX	MEM	WB									
BEQ R2,R0,Label12 (NT)		IF	ID	***	EX	MEM	WB							
LW R3,0(R2)			IF	***	ID	EX	MEM	WB						
BEQ R3,R0,Label11 (T)						IF	ID	EX	MEM	WB				
ADD R1,R3,R1							IF	ID	EX	MEM	WB			
BEQ R2,R0,Label12 (T)								IF	ID	EX	MEM	WB		
LW R3,0(R2)									IF	ID	EX	MEM	WB	
SW R1,0(R2)										IF	ID	EX	MEM	WB

4.14.3

```

LW R2,0(R1)
Label11: BEZ R2,Label12 ; Not taken once, then taken
LW R3,0(R2)
BEZ R3,Label11 ; Taken
ADD R1,R3,R1
Label12: SW R1,0(R2)

```

4.14.4 The hazard detection logic must detect situations when the branch depends on the result of the previous R-type instruction, or on the result of two previous loads. When the branch uses the values of its register operands in its ID stage, the R-type instruction's result is still being generated in the EX stage. Thus we must stall the processor and repeat the ID stage of the branch in the next cycle. Similarly, if the branch depends on a load that immediately precedes it, the result of the load is only generated two cycles after the branch enters the ID stage, so we must stall the branch for two cycles. Finally, if the branch depends on a load that is the second-previous instruction, the load is completing its MEM stage when the branch is in its ID stage, so we must stall the branch for one cycle. In all three cases, the hazard is a data hazard.

Note that in all three cases we assume that the values of preceding instructions are forwarded to the ID stage of the branch if possible.

4.14.5 For part a we have already shows the pipeline execution diagram for the case when branches are executed in the EX stage. The following is the pipeline diagram when branches are executed in the ID stage, including new stalls due to data dependences described for part d:

Executed Instructions	Pipeline Cycles														
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
LW R2,0(R1)	IF	ID	EX	MEM	WB										
BEQ R2,R0,Label2 (NT)		IF	***	***ID	EX	MEM	WB								
LW R3,0(R2)						IF	ID	EX	MEM	WB					
BEQ R3,R0,Label1 (T)							IF	***	***	ID	EX	MEM	WB		
BEQ R2,R0,Label2 (T)										IF	ID	EX	MEM	WB	
SW R1,0(R2)											IF	ID	EX	MEM	WB

Now the speedup can be computed as:

$$14/15 = 0.93$$

4.14.6 Branch instructions are now executed in the ID stage. If the branch instruction is using a register value produced by the immediately preceding instruction, as we described for part d the branch must be stalled because the preceding instruction is in the EX stage when the branch is already using the stale register values in the ID stage. If the branch in the ID stage depends on an R-type instruction that is in the MEM stage, we need forwarding to ensure correct execution of the branch. Similarly, if the branch in the ID stage depends on an R-type of load instruction in the WB stage, we need forwarding to ensure correct execution of the branch. Overall, we need another forwarding unit that takes the same inputs as the one that forwards to the EX stage. The new forwarding unit should control two Muxes placed right before the branch comparator. Each Mux selects between the value read from Registers, the ALU output from the EX/MEM pipeline register, and the data value from the MEM/WB pipeline register. The complexity of the new forwarding unit is the same as the complexity of the existing one.

4.15

4.15.1 Each branch that is not correctly predicted by the always-taken predictor will cause 3 stall cycles, so we have:

Extra CPI
$3 * (1 - 0.45) * 0.25 = 0.41$

4.15.2 Each branch that is not correctly predicted by the always-not-taken predictor will cause 3 stall cycles, so we have:

Extra CPI
$3 * (1 - 0.55) * 0.25 = 0.34$

4.15.3 Each branch that is not correctly predicted by the 2-bit predictor will cause 3 stall cycles, so we have:

Extra CPI
$3 * (1 - 0.85) * 0.25 = 0.113$

4.15.4 Correctly predicted branches had CPI of 1 and now they become ALU instructions whose CPI is also 1. Incorrectly predicted instructions that are converted also become ALU instructions with a CPI of 1, so we have:

CPI without conversion	CPI with conversion	Speedup from conversion
$1 + 3 * (1 - 0.85) * 0.25 = 1.113$	$1 + 3 * (1 - 0.85) * 0.25 * 0.5 = 1.056$	$1.113 / 1.056 = 1.054$

4.15.5 Every converted branch instruction now takes an extra cycle to execute, so we have:

CPI without conversion	Cycles per original instruction with conversion	Speedup from conversion
1.113	$1 + (1 + 3 * (1 - 0.85)) * 0.25 * 0.5 = 1.181$	$1.113 / 1.181 = 0.94$

4.15.6 Let the total number of branch instructions executed in the program be B. Then we have:

Correctly predicted	Correctly predicted non-loop-back	Accuracy on non-loop-back branches
$B * 0.85$	$B * 0.05$	$(B * 0.05) / (B * 0.20) = 0.25$ (25%)

4.16**4.16.1**

Always Taken	Always not-taken
$3/5 = 60\%$	$2/5 = 40\%$

4.16.2

Outcomes	Predictor value at time of prediction	Correct or Incorrect	Accuracy
T, NT, T, T	0,1,0,1	I,C,I,I	25%

4.16.3 The first few recurrences of this pattern do not have the same accuracy as the later ones because the predictor is still warming up. To determine the accuracy in the “steady state”, we must work through the branch predictions until the predictor values start repeating (i.e., until the predictor has the same value at the start of the current and the next recurrence of the pattern).

Outcomes	Predictor value at time of prediction	Correct or Incorrect (in steady state)	Accuracy in steady state
T, NT, T, T, NT	1 st occurrence: 0,1,0,1,2 2 nd occurrence: 1,2,1,2,3 3 rd occurrence: 2,3,2,3,3 4 th occurrence: 2,3,2,3,3	C,I,C,C,I	60%

4.16.4 The predictor should be an N-bit shift register, where N is the number of branch outcomes in the target pattern. The shift register should be initialized with the pattern itself (0 for NT, 1 for T), and the prediction is always the value in the leftmost bit of the shift register. The register should be shifted after each predicted branch.

4.16.5 Since the predictor’s output is always the opposite of the actual outcome of the branch instruction, the accuracy is zero.

4.16.6 The predictor is the same as in part d, except that it should compare its prediction to the actual outcome and invert (logical NOT) all the bits in the shift register if the prediction is incorrect. This predictor still always perfectly predicts the given pattern. For the opposite pattern, the first prediction will be incorrect, so the predictor’s state is inverted and after that the predictions are always correct. Overall, there is no warm-up period for the given pattern, and the warm-up period for the opposite pattern is only one branch.

4.17**4.17.1**

Instruction 1	Instruction 2
Invalid target address (EX)	Invalid data address (MEM)

4.17.2 The Mux that selects the next PC must have inputs added to it. Each input is a constant address of an exception handler. The exception detectors

must be added to the appropriate pipeline stage and the outputs of these detectors must be used to control the pre-PC Mux, and also to convert to NOPs instructions that are already in the pipeline behind the exception-triggering instruction.

- 4.17.3** Instructions are fetched normally until the exception is detected. When the exception is detected, all instructions that are in the pipeline after the first instruction must be converted to NOPs. As a result, the second instruction never completes and does not affect pipeline state. In the cycle that immediately follows the cycle in which the exception is detected, the processor will fetch the first instruction of the exception handler.
- 4.17.4** This approach requires us to fetch the address of the handler from memory. We must add the code of the exception to the address of the exception vector table, read the handler's address from memory, and jump to that address. One way of doing this is to handle it like a special instruction that computer the address in EX, loads the handler's address in MEM, and sets the PC in WB.
- 4.17.5** We need a special instruction that allows us to move a value from the (exception) Cause register to a general-purpose register. We must first save the general-purpose register (so we can restore it later), load the Cause register into it, add the address of the vector table to it, use the result as an address for a load that gets the address of the right exception handler from memory, and finally jump to that handler.

4.18

4.18.1

```
      ADD R5,R0,R0
Again: BEQ R5,R6,End
      ADD R10,R5,R1
      LW  R11,0(R10)
      LW  R10,1(R10)
      SUB R10,R11,R10
      ADD R11,R5,R2
      SW  R10,0(R11)
      ADDI R5,R5,2
      BEW R0,R0,Again
End:
```

4.18.2

Instructions	Pipeline
ADD R5,R0,R0	IF ID EX ME WB
BEQ R5,R6,End	IF ID ** EX ME WB
ADD R10,R5,R1	IF ** ID EX ME WB
LW R11,0(R10)	IF ** ID ** EX ME WB
LW R10,1(R10)	IF ** ID EX ME WB
SUB R10,R11,R10	IF ** ID ** ** EX ME WB
ADD R11,R5,R2	IF ** ** ID EX ME WB
SW R10,0(R11)	IF ** ** ID ** EX ME WB
ADDI R5,R5,2	IF ** ID EX ME WB
BEW R0,R0,Again	IF ** ID ** EX ME WB
BEQ R5,R6,End	IF ** ID EX ME WB
ADD R10,R5,R1	IF ** ID ** EX ME WB
LW R11,0(R10)	IF ** ID EX ME WB
LW R10,1(R10)	IF ** ID ** EX ME WB
SUB R10,R11,R10	IF ** ID ** EX ME WB
ADD R11,R5,R2	IF ** ID ** ** EX ME WB
SW R10,0(R11)	IF ** ** ID EX ME WB
ADDI R5,R5,2	IF ** ** ID EX ME WB
BEW R0,R0,Again	IF ID EX ME WB
BEQ R5,R6,End	IF ID ** EX ME WB

4.18.3 The only way to execute 2 instructions fully in parallel is for a load/store to execute together with another instruction. To achieve this, around each load/store instruction we will try to put non-load/store instructions that have no dependences with the load/store.

ADD R5,R0,R0 Again: ADD R10,R5,R1 BEQ R5,R6,End LW R11,0(R10) ADD R12,R5,R2 LW R10,1(R10) ADDI R5,R5,2 SUB R10,R11,R10 SW R10,0(R12) BEQ R0,R0,Again End:	Note that we are now computing $a+i$ before we check whether we should continue the loop. This is OK because we are allowed to "trash" R10. If we exit the loop one extra instruction is executed, but if we stay in the loop we allow both of the memory instructions to execute in parallel with other instructions
---	---

4.18.4

Instructions	Pipeline
ADD R5,R0,R0	IF ID EX ME WB
ADD R10,R5,R1	IF ID ** EX ME WB
BEQ R5,R6,End	IF ** ID EX ME WB
LW R11,0(R10)	IF ** ID EX ME WB
ADD R12,R5,R2	IF ID EX ME WB
LW R10,1(R10)	IF ID EX ME WB
ADDI R5,R5,2	IF ID EX ME WB
SUB R10,R11,R10	IF ID ** EX ME WB
SW R10,0(R12)	IF ** ID EX ME WB
BEQ R0,R0,Again	IF ** ID EX ME WB
ADD R10,R5,R1	IF ** ID EX ME WB
BEQ R5,R6,End	IF ** ID ** EX ME WB
LW R11,0(R10)	IF ** ID EX ME WB
ADD R12,R5,R2	IF ** ID EX ME WB
LW R10,1(R10)	IF ID EX ME WB
ADDI R5,R5,2	IF ID EX ME WB
SUB R10,R11,R10	IF ID ** EX ME WB
SW R10,0(R12)	IF ID ** EX ME WB
BEQ R0,R0,Again	IF ** ID EX ME WB
ADD R10,R5,R1	IF ** ID ** EX ME WB
BEQ R5,R6,End	IF ** ID EX ME WB

4.18.5

CPI for 1-issue	CPI for 2-issue	Speedup
1.11 (10 cycles per 9 instructions). There is 1 stall cycle in each iteration due to a data hazard between the second LW and the next instruction (SUB).	1.06 (19 cycles per 18 instructions). Neither of the two LW instructions can execute in parallel with another instruction, and SUB stalls because it depends on the second LW. The SW instruction executes in parallel with ADDI in even-numbered iterations.	1.05

4.18.6

CPI for 1-issue	CPI for 2-issue	Speedup
1.11	0.83 (15 cycles per 18 instructions). In all iterations, SUB is stalled because it depends on the second LW. The only instructions that execute in odd-numbered iterations as a pair are ADDI and BEQ. In even-numbered iterations, only the two LW instruction cannot execute as a pair.	1.34

4.19

4.19.1 The energy for the two designs is the same: I-Mem is read, two registers are read, and a register is written. We have:

$$140 \text{ pJ} + 2 \cdot 70 \text{ pJ} + 60 \text{ pJ} = 340 \text{ pJ}$$

4.19.2 The instruction memory is read for all instructions. Every instruction also results in two register reads (even if only one of those values is actually used). A load instruction results in a memory read and a register write, a store instruction results in a memory write, and all other instructions result in either no register write (e.g., BEQ) or a register write. Because the sum of memory read and register write energy is larger than memory write energy, the worst-case instruction is a load instruction. For the energy spent by a load, we have:

$$140 \text{ pJ} + 2 \cdot 70 \text{ pJ} + 60 \text{ pJ} + 140 \text{ pJ} = 480 \text{ pJ}$$

4.19.3 Instruction memory must be read for every instruction. However, we can avoid reading registers whose values are not going to be used. To do this, we must add RegRead1 and RegRead2 control inputs to the Registers unit to enable or disable each register read. We must generate these control signals quickly to avoid lengthening the clock cycle time. With these new control signals, a LW instruction results in only one register read (we still must read the register used to generate the address), so we have:

Energy before change	Energy saved by change	% Savings
$140 \text{ pJ} + 2 \cdot 70 \text{ pJ} + 60 \text{ pJ} + 140 \text{ pJ} = 480 \text{ pJ}$	70 pJ	14.6%

4.19.4 Before the change, the Control unit decodes the instruction while register reads are happening. After the change, the latencies of Control and Register Read cannot be overlapped. This increases the latency of the ID stage and could affect the processor's clock cycle time if the ID stage becomes the longest-latency stage. We have:

Clock cycle time before change	Clock cycle time after change
250 ps (D-Mem in MEM stage)	No change ($150 \text{ ps} + 90 \text{ ps} < 250 \text{ ps}$)

4.19.5 If memory is read in every cycle, the value is either needed (for a load instruction), or it does not get past the WB Mux (or a non-load instruction that writes to a register), or it does not get written to any register (all other instructions, including stalls). This change does not affect clock cycle time because the clock cycle time must already allow enough time for memory to be read in the MEM stage. It does affect energy: a memory read occurs in every cycle instead of only in cycles when a load instruction is in the MEM stage.

I-Mem active energy	I-Mem latency	Clock cycle time	Total I-Mem energy	Idle energy %
140 pJ	200 ps	250 ps	$140 \text{ pJ} + 50 \text{ ps} \cdot 0.1 \cdot 140 \text{ pJ} / 200 \text{ ps} = 143.5 \text{ pJ}$	$3.5 \text{ pJ} / 143.5 \text{ pJ} = 2.44\%$



5

Solutions

5.1**5.1.1** 4**5.1.2** I, J**5.1.3** $A[I][J]$ **5.1.4** $3596 = 8 \times 800/4 \times 2 - 8 \times 8/4 + 8000/4$ **5.1.5** I, J**5.1.6** $A(J, I)$ **5.2****5.2.1**

Word Address	Binary Address	Tag	Index	Hit/Miss
3	0000 0011	0	3	M
180	1011 0100	11	4	M
43	0010 1011	2	11	M
2	0000 0010	0	2	M
191	1011 1111	11	15	M
88	0101 1000	5	8	M
190	1011 1110	11	14	M
14	0000 1110	0	14	M
181	1011 0101	11	5	M
44	0010 1100	2	12	M
186	1011 1010	11	10	M
253	1111 1101	15	13	M

5.2.2

Word Address	Binary Address	Tag	Index	Hit/Miss
3	0000 0011	0	1	M
180	1011 0100	11	2	M
43	0010 1011	2	5	M
2	0000 0010	0	1	H
191	1011 1111	11	7	M
88	0101 1000	5	4	M
190	1011 1110	11	7	H
14	0000 1110	0	7	M
181	1011 0101	11	2	H
44	0010 1100	2	6	M
186	1011 1010	11	5	M
253	1111 1101	15	6	M

5.2.3

Word Address	Binary Address	Tag	Cache 1		Cache 2		Cache 3	
			index	hit/miss	index	hit/miss	index	hit/miss
3	0000 0011	0	3	M	1	M	0	M
180	1011 0100	22	4	M	2	M	1	M
43	0010 1011	5	3	M	1	M	0	M
2	0000 0010	0	2	M	1	M	0	M
191	1011 1111	23	7	M	3	M	1	M
88	0101 1000	11	0	M	0	M	0	M
190	1011 1110	23	6	M	3	H	1	H
14	0000 1110	1	6	M	3	M	1	M
181	1011 0101	22	5	M	2	H	1	M
44	0010 1100	5	4	M	2	M	1	M
186	1011 1010	23	2	M	1	M	0	M
253	1111 1101	31	5	M	2	M	1	M

Cache 1 miss rate = 100%

Cache 1 total cycles = $12 \times 25 + 12 \times 2 = 324$

Cache 2 miss rate = $10/12 = 83\%$

Cache 2 total cycles = $10 \times 25 + 12 \times 3 = 286$

Cache 3 miss rate = $11/12 = 92\%$

Cache 3 total cycles = $11 \times 25 + 12 \times 5 = 335$

Cache 2 provides the best performance.

5.2.4 First we must compute the number of cache blocks in the initial cache configuration. For this, we divide 32KiB by 4 (for the number of bytes per word) and again by 2 (for the number of words per block). This gives us 4096 blocks and a resulting index field width of 12 bits. We also have a word offset size of 1 bit and a byte offset size of 2 bits. This gives us a tag field size of $32 - 15 = 17$ bits. These tag bits, along with one valid bit per block, will require $18 \times 4096 = 73728$ bits or 9216 bytes. The total cache size is thus $9216 + 32768 = 41984$ bytes.

The total cache size can be generalized to

$\text{totalsize} = \text{datasize} + (\text{validbitsize} + \text{tagsize}) \times \text{blocks}$

$\text{totalsize} = 41984$

$\text{datasize} = \text{blocks} \times \text{blocksize} \times \text{wordsize}$

$\text{wordsize} = 4$

$\text{tagsize} = 32 - \log_2(\text{blocks}) - \log_2(\text{blocksize}) - \log_2(\text{wordsize})$

$\text{validbitsize} = 1$

Increasing from 2-word blocks to 16-word blocks will reduce the tag size from 17 bits to 14 bits.

In order to determine the number of blocks, we solve the inequality:

$$41984 \leq 64 \times \text{blocks} + 15 \times \text{blocks}$$

Solving this inequality gives us 531 blocks, and rounding to the next power of two gives us a 1024-block cache.

The larger block size may require an increased hit time and an increased miss penalty than the original cache. The fewer number of blocks may cause a higher conflict miss rate than the original cache.

5.2.5 Associative caches are designed to reduce the rate of conflict misses. As such, a sequence of read requests with the same 12-bit index field but a different tag field will generate many misses. For the cache described above, the sequence 0, 32768, 0, 32768, 0, 32768, ..., would miss on every access, while a 2-way set associate cache with LRU replacement, even one with a significantly smaller overall capacity, would hit on every access after the first two.

5.2.6 Yes, it is possible to use this function to index the cache. However, information about the five bits is lost because the bits are XOR'd, so you must include more tag bits to identify the address in the cache.

5.3

5.3.1 8

5.3.2 32

5.3.3 $1 + (22/8/32) = 1.086$

5.3.4 3

5.3.5 0.25

5.3.6 <Index, tag, data>

<000001₂, 0001₂, mem[1024]>

<000001₂, 0011₂, mem[16]>

<001011₂, 0000₂, mem[176]>

<001000₂, 0010₂, mem[2176]>

<001110₂, 0000₂, mem[224]>

<001010₂, 0000₂, mem[160]>

5.4

5.4.1 The L1 cache has a low write miss penalty while the L2 cache has a high write miss penalty. A write buffer between the L1 and L2 cache would hide the write miss latency of the L2 cache. The L2 cache would benefit from write buffers when replacing a dirty block, since the new block would be read in before the dirty block is physically written to memory.

5.4.2 On an L1 write miss, the word is written directly to L2 without bringing its block into the L1 cache. If this results in an L2 miss, its block must be brought into the L2 cache, possibly replacing a dirty block which must first be written to memory.

5.4.3 After an L1 write miss, the block will reside in L2 but not in L1. A subsequent read miss on the same block will require that the block in L2 be written back to memory, transferred to L1, and invalidated in L2.

5.4.4 One in four instructions is a data read, one in ten instructions is a data write. For a CPI of 2, there are 0.5 instruction accesses per cycle, 12.5% of cycles will require a data read, and 5% of cycles will require a data write.

The instruction bandwidth is thus $(0.0030 \times 64) \times 0.5 = 0.096$ bytes/cycle. The data read bandwidth is thus $0.02 \times (0.13 + 0.050) \times 64 = 0.23$ bytes/cycle. The total read bandwidth requirement is 0.33 bytes/cycle. The data write bandwidth requirement is $0.05 \times 4 = 0.2$ bytes/cycle.

5.4.5 The instruction and data read bandwidth requirement is the same as in 5.4.4. The data write bandwidth requirement becomes $0.02 \times 0.30 \times (0.13 + 0.050) \times 64 = 0.069$ bytes/cycle.

5.4.6 For CPI=1.5 the instruction throughput becomes $1/1.5 = 0.67$ instructions per cycle. The data read frequency becomes $0.25 / 1.5 = 0.17$ and the write frequency becomes $0.10 / 1.5 = 0.067$.

The instruction bandwidth is $(0.0030 \times 64) \times 0.67 = 0.13$ bytes/cycle.

For the write-through cache, the data read bandwidth is $0.02 \times (0.17 + 0.067) \times 64 = 0.22$ bytes/cycle. The total read bandwidth is 0.35 bytes/cycle. The data write bandwidth is $0.067 \times 4 = 0.27$ bytes/cycle.

For the write-back cache, the data write bandwidth becomes $0.02 \times 0.30 \times (0.17 + 0.067) \times 64 = 0.091$ bytes/cycle.

Address	0	4	16	132	232	160	1024	30	140	3100	180	2180
Line ID	0	0	1	8	14	10	0	1	9	1	11	8
Hit/miss	M	H	M	M	M	M	M	H	H	M	M	M
Replace	N	N	N	N	N	N	Y	N	N	Y	N	Y

5.5

5.5.1 Assuming the addresses given as byte addresses, each group of 16 accesses will map to the same 32-byte block so the cache will have a miss rate of 1/16. All misses are compulsory misses. The miss rate is not sensitive to the size of the cache or the size of the working set. It is, however, sensitive to the access pattern and block size.

5.5.2 The miss rates are 1/8, 1/32, and 1/64, respectively. The workload is exploiting temporal locality.

5.5.3 In this case the miss rate is 0.

5.5.4 AMAT for $B = 8$: $0.040 \times (20 \times 8) = 6.40$

AMAT for $B = 16$: $0.030 \times (20 \times 16) = 9.60$

AMAT for $B = 32$: $0.020 \times (20 \times 32) = 12.80$

AMAT for $B = 64$: $0.015 \times (20 \times 64) = 19.20$

AMAT for $B = 128$: $0.010 \times (20 \times 128) = 25.60$

$B = 8$ is optimal.

5.5.5 AMAT for $B = 8$: $0.040 \times (24 + 8) = 1.28$

AMAT for $B = 16$: $0.030 \times (24 + 16) = 1.20$

AMAT for $B = 32$: $0.020 \times (24 + 32) = 1.12$

AMAT for $B = 64$: $0.015 \times (24 + 64) = 1.32$

AMAT for $B = 128$: $0.010 \times (24 + 128) = 1.52$

$B = 32$ is optimal.

5.5.6 $B=128$

5.6**5.6.1**

P1	1.52 GHz
P2	1.11 GHz

5.6.2

P1	6.31 ns	9.56 cycles
P2	5.11 ns	5.68 cycles

5.6.3

P1	12.64 CPI	8.34 ns per inst
P2	7.36 CPI	6.63 ns per inst

5.6.4

6.50 ns	9.85 cycles	Worse
---------	-------------	-------

5.6.5 13.04

5.6.6 $P1 \text{ AMAT} = 0.66 \text{ ns} + 0.08 \times 70 \text{ ns} = 6.26 \text{ ns}$

$P2 \text{ AMAT} = 0.90 \text{ ns} + 0.06 \times (5.62 \text{ ns} + 0.95 \times 70 \text{ ns}) = 5.23 \text{ ns}$

For P1 to match P2's performance:

$5.23 = 0.66 \text{ ns} + MR \times 70 \text{ ns}$

$MR = 6.5\%$

5.7

5.7.1 The cache would have $24 / 3 = 8$ blocks per way and thus an index field of 3 bits.

Word Address	Binary Address	Tag	Index	Hit/Miss	Way 0	Way 1	Way 2
3	0000 0011	0	1	M	T(1)=0		
180	1011 0100	11	2	M	T(1)=0 T(2)=11		
43	0010 1011	2	5	M	T(1)=0 T(2)=11 T(5)=2		
2	0000 0010	0	1	M	T(1)=0 T(2)=11 T(5)=2	T(1)=0	
191	1011 1111	11	7	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11	T(1)=0	
88	0101 1000	5	4	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5	T(1)=0	
190	1011 1110	11	7	H	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5	T(1)=0	
14	0000 1110	0	7	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5	T(1)=0 T(7)=0	
181	1011 0101	11	2	H	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5	T(1)=0 T(7)=0	

44	0010 1100	2	6	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5 T(6)=2	T(1)=0 T(7)=0	
186	1011 1010	11	5	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5 T(6)=2	T(1)=0 T(7)=0 T(5)=11	
253	1111 1101	15	6	M	T(1)=0 T(2)=11 T(5)=2 T(7)=11 T(4)=5 T(6)=2	T(1)=0 T(7)=0 T(5)=11 T(6)=15	

5.7.2 Since this cache is fully associative and has one-word blocks, the word address is equivalent to the tag. The only possible way for there to be a hit is a repeated reference to the same word, which doesn't occur for this sequence.

Tag	Hit/Miss	Contents
3	M	3
180	M	3, 180
43	M	3, 180, 43
2	M	3, 180, 43, 2
191	M	3, 180, 43, 2, 191
88	M	3, 180, 43, 2, 191, 88
190	M	3, 180, 43, 2, 191, 88, 190
14	M	3, 180, 43, 2, 191, 88, 190, 14
181	M	181, 180, 43, 2, 191, 88, 190, 14
44	M	181, 44, 43, 2, 191, 88, 190, 14
186	M	181, 44, 186, 2, 191, 88, 190, 14
253	M	181, 44, 186, 253, 191, 88, 190, 14

5.7.3

Address	Tag	Hit/Miss	Contents
3	1	M	1
180	90	M	1, 90
43	21	M	1, 90, 21
2	1	H	1, 90, 21
191	95	M	1, 90, 21, 95
88	44	M	1, 90, 21, 95, 44
190	95	H	1, 90, 21, 95, 44
14	7	M	1, 90, 21, 95, 44, 7
181	90	H	1, 90, 21, 95, 44, 7
44	22	M	1, 90, 21, 95, 44, 7, 22
186	143	M	1, 90, 21, 95, 44, 7, 22, 143
253	126	M	1, 90, 126, 95, 44, 7, 22, 143

The final reference replaces tag 21 in the cache, since tags 1 and 90 had been re-used at time=3 and time=8 while 21 hadn't been used since time=2.

$$\text{Miss rate} = 9/12 = 75\%$$

This is the best possible miss rate, since there were no misses on any block that had been previously evicted from the cache. In fact, the only eviction was for tag 21, which is only referenced once.

5.7.4 L1 only:

$$.07 \times 100 = 7 \text{ ns}$$

$$\text{CPI} = 7 \text{ ns} / .5 \text{ ns} = 14$$

Direct mapped L2:

$$.07 \times (12 + 0.035 \times 100) = 1.1 \text{ ns}$$

$$\text{CPI} = \text{ceiling}(1.1 \text{ ns} / .5 \text{ ns}) = 3$$

8-way set associated L2:

$$.07 \times (28 + 0.015 \times 100) = 2.1 \text{ ns}$$

$$\text{CPI} = \text{ceiling}(2.1 \text{ ns} / .5 \text{ ns}) = 5$$

Doubled memory access time, L1 only:

$$.07 \times 200 = 14 \text{ ns}$$

$$\text{CPI} = 14 \text{ ns} / .5 \text{ ns} = 28$$

Doubled memory access time, direct mapped L2:

$$.07 \times (12 + 0.035 \times 200) = 1.3 \text{ ns}$$

$$\text{CPI} = \text{ceiling}(1.3 \text{ ns} / .5 \text{ ns}) = 3$$

Doubled memory access time, 8-way set associated L2:

$$.07 \times (28 + 0.015 \times 200) = 2.2 \text{ ns}$$

$$\text{CPI} = \text{ceiling}(2.2 \text{ ns} / .5 \text{ ns}) = 5$$

Halved memory access time, L1 only:

$$.07 \times 50 = 3.5 \text{ ns}$$

$$\text{CPI} = 3.5 \text{ ns} / .5 \text{ ns} = 7$$

Halved memory access time, direct mapped L2:

$$.07 \times (12 + 0.035 \times 50) = 1.0 \text{ ns}$$

$$\text{CPI} = \text{ceiling}(1.1 \text{ ns} / .5 \text{ ns}) = 2$$

Halved memory access time, 8-way set associated L2:

$$.07 \times (28 + 0.015 \times 50) = 2.1 \text{ ns}$$

$$\text{CPI} = \text{ceiling}(2.1 \text{ ns} / .5 \text{ ns}) = 5$$

$$\mathbf{5.7.5} \quad .07 \times (12 + 0.035 \times (50 + 0.013 \times 100)) = 1.0 \text{ ns}$$

Adding the L3 cache does reduce the overall memory access time, which is the main advantage of having a L3 cache. The disadvantage is that the L3 cache takes real estate away from having other types of resources, such as functional units.

5.7.6 Even if the miss rate of the L2 cache was 0, a 50 ns access time gives

AMAT = $.07 \times 50 = 3.5 \text{ ns}$, which is greater than the 1.1 ns and 2.1 ns given by the on-chip L2 caches. As such, no size will achieve the performance goal.

5.8

5.8.1

1096 days	26304 hours
-----------	-------------

5.8.2

0.9990875912%

5.8.3 Availability approaches 1.0. With the emergence of inexpensive drives, having a nearly 0 replacement time *for hardware* is quite feasible. However, replacing file systems and other data can take significant time. Although a drive manufacturer will not include this time in their statistics, it is certainly a part of replacing a disk.

5.8.4 MTTR becomes the dominant factor in determining availability. However, availability would be quite high if MTTF also grew measurably. If MTTF is 1000 times MTTR, the specific value of MTTR is not significant.

5.9

5.9.1 Need to find minimum p such that $2^p \geq p + d + 1$ and then add one. Thus 9 total bits are needed for SEC/DED.

5.9.2 The (72,64) code described in the chapter requires an overhead of $8/64=12.5\%$ additional bits to tolerate the loss of any single bit within 72 bits, providing a protection rate of 1.4%. The (137,128) code from part a requires an overhead of $9/128=7.0\%$ additional bits to tolerate the loss of any single bit within 137 bits, providing a protection rate of 0.73%. The cost/performance of both codes is as follows:

$$(72,64) \text{ code} \Rightarrow 12.5/1.4 = 8.9$$

$$(136,128) \text{ code} \Rightarrow 7.0/0.73 = 9.6$$

The (72,64) code has a better cost/performance ratio.

5.9.3 Using the bit numbering from section 5.5, bit 8 is in error so the value would be corrected to 0x365.

5.10 Instructors can change the disk latency, transfer rate and optimal page size for more variants. Refer to Jim Gray's paper on the five-minute rule ten years later.

5.10.1 32 KB

5.10.2 Still 32 KB

5.10.3 64 KB. Because the disk bandwidth grows much faster than seek latency, future paging cost will be more close to constant, thus favoring larger pages.

5.10.4 1987/1997/2007: 205/267/308 seconds. (or roughly five minutes)

5.10.5 1987/1997/2007: 51/533/4935 seconds. (or 10 times longer for every 10 years).

5.10.6 (1) DRAM cost/MB scaling trend dramatically slows down; or (2) disk \$/access/sec dramatically increase. (2) is more likely to happen due to the emerging flash technology.

5.11

5.11.1

Address	Virtual Page	TLB H/M	TLB		
			Valid	Tag	Physical Page
4669	1	TLB miss PT hit PF	1	11	12
			1	7	4
			1	3	6
			1 (last access 0)	1	13
2227	0	TLB miss PT hit	1 (last access 1)	0	5
			1	7	4
			1	3	6
			1 (last access 0)	1	13
13916	3	TLB hit	1 (last access 1)	0	5
			1	7	4
			1 (last access 2)	3	6
			1 (last access 0)	1	13
34587	8	TLB miss PT hit PF	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 2)	3	6
			1 (last access 0)	1	13
48870	11	TLB miss PT hit	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 2)	3	6
			1 (last access 4)	11	12
12608	3	TLB hit	1 (last access 1)	0	5
			1 (last access 3)	8	14
			1 (last access 5)	3	6
			1 (last access 4)	11	12
49225	12	TLB miss PT miss	1 (last access 6)	12	15
			1 (last access 3)	8	14
			1 (last access 5)	3	6
			1 (last access 4)	11	12

5.11.2

Address	Virtual Page	TLB H/M	TLB		
			Valid	Tag	Physical Page
4669	0	TLB miss PT hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 0)	0	5
2227	0	TLB hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 1)	0	5
13916	0	TLB hit	1	11	12
			1	7	4
			1	3	6
			1 (last access 2)	0	5
34587	2	TLB miss PT hit PF	1 (last access 3)	2	13
			1	7	4
			1	3	6
			1 (last access 2)	0	5
48870	2	TLB hit	1 (last access 4)	2	13
			1	7	4
			1	3	6
			1 (last access 2)	0	5
12608	0	TLB hit	1 (last access 4)	2	13
			1	7	4
			1	3	6
			1 (last access 5)	0	5
49225	3	TLB hit	1 (last access 4)	2	13
			1	7	4
			1 (last axcess 6)	3	6
			1 (last access 5)	0	5

A larger page size reduces the TLB miss rate but can lead to higher fragmentation and lower utilization of the physical memory.

5.11.3 Two-way set associative

Address	Virtual Page	Tag	Index	TLB H/M	TLB			
					Valid	Tag	Physical Page	Index
4669	1	0	1	TLB miss PT hit PF	1	11	12	0
					1	7	4	1
					1	3	6	0
					1 (last access 0)	0	13	1
2227	0	0	0	TLB miss PT hit	1 (last access 1)	0	5	0
					1	7	4	1
					1	3	6	0
					1 (last access 0)	0	13	1
13916	3	1	1	TLB miss PT hit	1 (last access 1)	0	5	0
					1 (last access 2)	1	6	1
					1	3	6	0
					1 (last access 0)	1	13	1
34587	8	4	0	TLB miss PT hit PF	1 (last access 1)	0	5	0
					1 (last access 2)	1	6	1
					1 (last access 3)	4	14	0
					1 (last access 0)	1	13	1
48870	11	5	1	TLB miss PT hit	1 (last access 1)	0	5	0
					1 (last access 2)	1	6	1
					1 (last access 3)	4	14	0
					1 (last access 4)	5	12	1
12608	3	1	1	TLB hit	1 (last access 1)	0	5	0
					1 (last access 5)	1	6	1
					1 (last access 3)	4	14	0
					1 (last access 4)	5	12	1
49225	12	6	0	TLB miss PT miss	1 (last access 6)	6	15	0
					1 (last access 5)	1	6	1
					1 (last access 3)	4	14	0
					1 (last access 4)	5	12	1

Direct mapped

Address	Virtual Page	Tag	Index	TLB H/M	TLB			
					Valid	Tag	Physical Page	Index
4669	1	0	1	TLB miss PT hit PF	1	11	12	0
					1	0	13	1
					1	3	6	2
					0	4	9	3
2227	0	0	0	TLB miss PT hit	1	0	5	0
					1	0	13	1
					1	3	6	2
					0	4	9	3
13916	3	0	3	TLB miss PT hit	1	0	5	0
					1	0	13	1
					1	3	6	2
					1	0	6	3
34587	8	2	0	TLB miss PT hit PF	1	2	14	0
					1	0	13	1
					1	3	6	2
					1	0	6	3
48870	11	2	3	TLB miss PT hit	1	2	14	0
					1	0	13	1
					1	3	6	2
					1	2	12	3
12608	3	0	3	TLB miss PT hit	1	2	14	0
					1	0	13	1
					1	3	6	2
					1	0	6	3
49225	12	3	0	TLB miss PT miss	1	3	15	0
					1	0	13	1
					1	3	6	2
					1	0	6	3

All memory references must be cross referenced against the page table and the TLB allows this to be performed without accessing off-chip memory (in the common case). If there were no TLB, memory access time would increase significantly.

5.11.4 Assumption: “half the memory available” means half of the 32-bit virtual address space for each running application.

The tag size is $32 - \log_2(8192) = 32 - 13 = 19$ bits. All five page tables would require $5 \times (2^{19/2} \times 4)$ bytes = 5 MB.

5.11.5 In the two-level approach, the 2^{19} page table entries are divided into 256 segments that are allocated on demand. Each of the second-level tables contain $2^{(19-8)} = 2048$ entries, requiring $2048 \times 4 = 8$ KB each and covering 2048×8 KB = 16 MB (2^{24}) of the virtual address space.

If we assume that “half the memory” means 2^{31} bytes, then the minimum amount of memory required for the second-level tables would be $5 \times (2^{31} / 2^{24}) \times 8 \text{ KB} = 5 \text{ MB}$. The first-level tables would require an additional $5 \times 128 \times 6 \text{ bytes} = 3840 \text{ bytes}$.

The maximum amount would be if all segments were activated, requiring the use of all 256 segments in each application. This would require $5 \times 256 \times 8 \text{ KB} = 10 \text{ MB}$ for the second-level tables and 7680 bytes for the first-level tables.

5.11.6 The page index consists of address bits 12 down to 0 so the LSB of the tag is address bit 13.

A 16 KB direct-mapped cache with 2-words per block would have 8-byte blocks and thus $16 \text{ KB} / 8 \text{ bytes} = 2048 \text{ blocks}$, and its index field would span address bits 13 down to 3 (11 bits to index, 1 bit word offset, 2 bit byte offset). As such, the tag LSB of the cache tag is address bit 14.

The designer would instead need to make the cache 2-way associative to increase its size to 16 KB.

5.12

5.12.1 Worst case is $2^{(43-12)}$ entries, requiring $2^{(43-12)} \times 4 \text{ bytes} = 2^{33} = 8 \text{ GB}$.

5.12.2 With only two levels, the designer can select the size of each page table segment. In a multi-level scheme, reading a PTE requires an access to each level of the table.

5.12.3 In an inverted page table, the number of PTEs can be reduced to the size of the hash table plus the cost of collisions. In this case, serving a TLB miss requires an extra reference to compare the tag or tags stored in the hash table.

5.12.4 It would be invalid if it was paged out to disk.

5.12.5 A write to page 30 would generate a TLB miss. Software-managed TLBs are faster in cases where the software can pre-fetch TLB entries.

5.12.6 When an instruction writes to VA page 200, and interrupt would be generated because the page is marked as read only.

5.13

5.13.1 0 hits

5.13.2 1 hit

5.13.3 1 hits or fewer

5.13.4 1 hit. Any address sequence is fine so long as the number of hits are correct.

5.13.5 The best block to evict is the one that will cause the fewest misses in the future. Unfortunately, a cache controller cannot know the future! Our best alternative is to make a good prediction.

5.13.6 If you knew that an address had limited temporal locality and would conflict with another block in the cache, it could improve miss rate. On the other hand, you could worsen the miss rate by choosing poorly which addresses to cache.

5.14

5.14.1 Shadow page table: (1) VM creates page table, hypervisor updates shadow table; (2) nothing; (3) hypervisor intercepts page fault, creates new mapping, and invalidates the old mapping in TLB; (4) VM notifies the hypervisor to invalidate the process's TLB entries. Nested page table: (1) VM creates new page table, hypervisor adds new mappings in PA to MA table. (2) Hardware walks both page tables to translate VA to MA; (3) VM and hypervisor update their page tables, hypervisor invalidates stale TLB entries; (4) same as shadow page table.

5.14.2 Native: 4; NPT: 24 (instructors can change the levels of page table)

Native: L ; NPT: $L \times (L + 2)$

5.14.3 Shadow page table: page fault rate.

NPT: TLB miss rate.

5.14.4 Shadow page table: 1.03

NPT: 1.04

5.14.5 Combining multiple page table updates

5.14.6 NPT caching (similar to TLB caching)

5.15

5.15.1 $\text{CPI} = 1.5 + 120/10000 \times (15 + 175) = 3.78$

If VMM performance impact doubles $\Rightarrow \text{CPI} = 1.5 + 120/10000 \times (15 + 350) = 5.88$

If VMM performance impact halves $\Rightarrow \text{CPI} = 1.5 + 120/10000 \times (15 + 87.5) = 2.73$

5.15.2 Non-virtualized $\text{CPI} = 1.5 + 30/10000 \times 1100 = 4.80$

Virtualized $\text{CPI} = 1.5 + 120/10000 \times (15 + 175) + 30/10000 \times (1100 + 175) = 7.60$

Virtualized CPI with half I/O $= 1.5 + 120/10000 \times (15 + 175) + 15/10000 \times (1100 + 175) = 5.69$

I/O traps usually often require long periods of execution time that can be performed in the guest O/S, with only a small portion of that time needing to be spent in the VMM. As such, the impact of virtualization is less for I/O bound applications.

5.15.3 Virtual memory aims to provide each application with the illusion of the entire address space of the machine. Virtual machines aims to provide each operating system with the illusion of having the entire machine to its disposal. Thus they both serve very similar goals, and offer benefits such as increased security. Virtual memory can allow for many applications running in the same memory space to not have to manage keeping their memory separate.

5.15.4 Emulating a different ISA requires specific handling of that ISA's API. Each ISA has specific behaviors that will happen upon instruction execution, interrupts, trapping to kernel mode, etc. that therefore must be emulated. This can require many more instructions to be executed to emulate each instruction than was originally necessary in the target ISA. This can cause a large performance impact and make it difficult to properly communicate with external devices. An emulated system can potentially run faster than on its native ISA if the emulated code can be dynamically examined and optimized. For example, if the underlying machine's ISA has a single instruction that can handle the execution of several of the emulated system's instructions, then potentially the number of instructions executed can be reduced. This is similar to the case with the recent Intel processors that do micro-op fusion, allowing several instructions to be handled by fewer instructions.

5.16

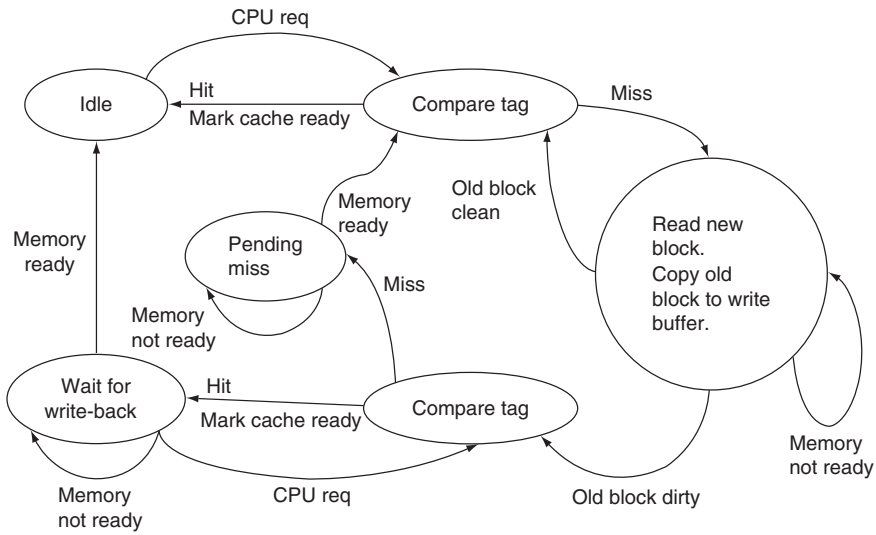
5.16.1 The cache should be able to satisfy the request since it is otherwise idle when the write buffer is writing back to memory. If the cache is not able to satisfy hits while writing back from the write buffer, the cache will perform little or no better than the cache without the write buffer, since requests will still be serialized behind writebacks.

5.16.2 Unfortunately, the cache will have to wait until the writeback is complete since the memory channel is occupied. Once the memory channel is free, the cache is able to issue the read request to satisfy the miss.

5.16.3 Correct solutions should exhibit the following features:

1. The memory read should come before memory writes.
2. The cache should signal "Ready" to the processor before completing the write.

Example (simpler solutions exist; the state machine is somewhat underspecified in the chapter):

**5.17**

5.17.1 There are 6 possible orderings for these instructions.

Ordering 1:

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

Results: (5,5)

Ordering 2:

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

Results: (5,5)

Ordering 3:

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

Results: (6,3)

Ordering 4:

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

Results: (5,3)

Ordering 5:

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

Results: (6,5)

Ordering 6:

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

(6,3)

If coherency isn't ensured:

P2's operations take precedence over P1's: (5,2)

5.17.2

P1	P1 cache status/ action	P2	P2 cache status/action
		X[0]=5	invalidate X on other caches, read X in exclusive state, write X block in cache
		X[1] += 2;	read and write X block in cache
X[0]++;	read value of X into cache		X block enters shared state
	send invalidate message		X block is invalidated
	write X block in cache		
X[1] = 3;	write X block in cache		

5.17.3 Best case:

Orderings 1 and 6 above, which require only two total misses.

Worst case:

Orderings 2 and 3 above, which require 4 total cache misses.

5.17.4 Ordering 1:

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

Result: (3,3)

Ordering 2:

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

Result: (2,3)

Ordering 3:

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

Result: (2,3)

Ordering 4:

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

Result: (0,3)

Ordering 5:

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

Result: (0,3)

Ordering 6:

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

Result: (2,3)

Ordering 7:

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

Result: (2,3)

Ordering 8:

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

Result: (0,3)

Ordering 9:

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

Result: (0,3)

Ordering 10:

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

Result: (2,1)

Ordering 11:

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

Result: (0,1)

Ordering 12:

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

Result: (0,1)

Ordering 13:

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

Result: (0,1)

Ordering 14:

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

Result: (0,1)

Ordering 15:

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

Result: (0,0)

5.17.5 Assume B=0 is seen by P2 but not preceding A=1

Result: (2,0)

5.17.6 Write back is simpler than write through, since it facilitates the use of exclusive access blocks and lowers the frequency of invalidates. It prevents the use of write-broadcasts, but this is a more complex protocol.

The allocation policy has little effect on the protocol.

5.18

5.18.1 Benchmark A

$$AMAT_{\text{private}} = (1/32) \times 5 + 0.0030 \times 180 = 0.70$$

$$AMAT_{\text{shared}} = (1/32) \times 20 + 0.0012 \times 180 = 0.84$$

Benchmark B

$$AMAT_{\text{private}} = (1/32) \times 5 + 0.0006 \times 180 = 0.26$$

$$AMAT_{\text{shared}} = (1/32) \times 20 + 0.0003 \times 180 = 0.68$$

Private cache is superior for both benchmarks.

5.18.2 Shared cache latency doubles for shared cache. Memory latency doubles for private cache.

Benchmark A

$$AMAT_{\text{private}} = (1/32) \times 5 + 0.0030 \times 360 = 1.24$$

$$AMAT_{\text{shared}} = (1/32) \times 40 + 0.0012 \times 180 = 1.47$$

Benchmark B

$$AMAT_{\text{private}} = (1/32) \times 5 + 0.0006 \times 360 = 0.37$$

$$AMAT_{\text{shared}} = (1/32) \times 40 + 0.0003 \times 180 = 1.30$$

Private is still superior for both benchmarks.

5.18.3

	Shared L2	Private L2
Single threaded	No advantage. No disadvantage.	No advantage. No disadvantage.
Multi-threaded	Shared caches can perform better for workloads where threads are tightly coupled and frequently share data.	Threads often have private working sets, and using a private L2 prevents cache contamination and conflict misses between threads.
	No disadvantage.	
Multiprogrammed	No advantage except in rare cases where processes communicate. The disadvantage is higher cache latency.	Caches are kept private, isolating data between processes. This works especially well if the OS attempts to assign the same CPU to each process.
Having private L2 caches with a shared L3 cache is an effective compromise for many workloads, and this is the scheme used by many modern processors.		

5.18.4 A non-blocking shared L2 cache would reduce the latency of the L2 cache by allowing hits for one CPU to be serviced while a miss is serviced for another CPU, or allow for misses from both CPUs to be serviced simultaneously. A non-blocking private L2 would reduce latency assuming that multiple memory instructions can be executed concurrently.

5.18.5 4 times.

5.18.6 Additional DRAM bandwidth, dynamic memory schedulers, multi-banked memory systems, higher cache associativity, and additional levels of cache.

- f. Processor: out-of-order execution, larger load/store queue, multiple hardware threads;

Caches: more miss status handling registers (MSHR)

Memory: memory controller to support multiple outstanding memory requests

5.19

5.19.1 `srcIP` and `refTime` fields. 2 misses per entry.

5.19.2 Group the `srcIP` and `refTime` fields into a separate array.

5.19.3 `peak_hour (int status); // peak hours of a given status`

Group `srcIP`, `refTime` and `status` together.

5.19.4 Answers will vary depending on which data set is used.

Conflict misses do not occur in fully associative caches.

Compulsory (cold) misses are not affected by associativity.

Capacity miss rate is computed by subtracting the compulsory miss rate and the fully associative miss rate (compulsory + capacity misses) from the total miss rate. Conflict miss rate is computed by subtracting the cold and the newly computed capacity miss rate from the total miss rate.

The values reported are miss rate per instruction, as opposed to miss rate per memory instruction.

5.19.5 Answers will vary depending on which data set is used.

5.19.6 `apsi/mesa/ammp/mcf` all have such examples.

Example cache: 4-block caches, direct-mapped vs. 2-way LRU.

Reference stream (blocks): 1 2 2 6 1.