1.12 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.

1.13 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 d

f. Increasing the ga

g. Adding electron
as opposed to curr
generation offered b

h. Building self-dri
systems already inst
smart cruise contro

1.3 [2] <§1.3> De
language such as C
processor.

1.4 [2] <§1.4> As
(red, green, blue) p

a. What is the min

b. How long woul
Mbit/s network?

1.5 [4] <§1.6> C
the same instructi
2.5 GHz clock rat
of 2.2.

a. Which processo

b. If the processo
cycles and the nur

c. We are trying t
of 20% in the CPI.

1.6 [20] <§1.6>
set architecture.
their CPI (class A,
and 3, and P2 wit
Given a program
into classes as fo
which implement

a. What is the gl

b. Find the clock

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 x *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
what would the impa
processors?

1.9.3 [10] <§§1.6, 1
reduced in order for a
using the original CP

1.10 Assume a 15 c
0.020 defects/cm^2. As
dies, and has 0.031 de

1.10.1 [10] <§1.5>

1.10.2 [5] <§1.5> F

1.10.3 [5] <§1.5> I
defects per area unit i

1.10.4 [5] <§1.5> A
0.95. Find the defect
area of 200 mm^2.

1.11 The results of
Barcelona has an ins
reference time of 965

1.11.1 [5] <§§1.6, 1

1.11.2 [5] <§1.9> F

1.11.3 [5] <§§1.6, 1
of the benchmark is

1.11.4 [5] <§§1.6, 1
of the benchmark is

1.11.5 [5] <§§1.6,

1.11.6 [10] <§1.6>
Barcelona processor
instructions to the i
has been reduced b
SPECratio is 13.7. F

1.11.7 [10] <§1.6>
rate was increased f
CPI is similar to tha

1.11.8 [5] <§1.6>

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

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

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

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

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

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

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

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 to 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

MFLOPS = No. FP operations / (execution time × 1E6)

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>
total time is reduced

1.13.3 [5] <§1.10>
the time for branch i

1.14 Assume a pr
110 × 106 INT ins
instructions. The C
Assume that the pro

1.14.1 [10] <§1.10
we want the prograr

1.14.2 [10] <§1.1(
if we want the progr

1.14.3 [5] <§1.10:
if the CPI of INT a
Branch is reduced t

1.15 [5] <§1.8>
a multiprocessor s
computing time an
to send data from

Assume a program
p processors, each
irrespective of the
time for 2, 4, 8, 16,
speedup relative tc
ideal speedup (spe

§1.1, page 10: Disc
§1.4, page 24: DRA
and cost per GB i
to 400,000 times
DRAM. Flash me
DRAM, and cost
§1.5, page 28: 1, 3
high volume can
economic decisio
§1.6, page 33: 1. a
§1.6, page 40: b.
§1.10, page 51: a.

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 × 106 FP instructions, 110 × 106 INT instructions, 80 × 106 L/S instructions, and 16 × 106 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).

Answers to Check Yourself

§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.

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 2.19 is found online.

2.20 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 `la` 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 if 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>
code to sort the data fr
smallest memory locatio
represents the C variable
the first number in the
that this particular mach
of four bytes.

2.6.2 [5] <§§2.2, 2.3> F
code to sort the data from
memory location. Use a n
address of Array is store

2.7 [5] <§2.3> Show ho
of a little-endian and a b
address 0.

2.8 [5] <§2.4> Translate

2.9 [5] <§§2.2, 2.3> Tr
variables f, g, h, i, and
respectively. Assume that
and $s7, respectively. A
words:

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

2.10 [5] <§§2.2, 2.3> T
variables f, g, h, i, and
respectively. Assume tha
and $s7, respectively.

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

2.11 [5] <§§2.2, 2.5> F
(OP), source register (RS
show the value of the in
value of the destination

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: $\texttt{0000 0010 0001 0000 1000 0000 0010 0000}_{two}$

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> Pr
representation of ins

```
op=0, rs=3,
```

2.17 [5] <§2.5> Pr
representation of ins

```
op=0x23, rs
```

2.18 Assume that v
and expand the instr

2.18.1 [5] <§2.5> I
the R-type instructio

2.18.2 [5] <§2.5> I
the I-type instructio

2.18.3 [5] <§§2.5,
the size of an MIPS a
change increase the s

2.19 Assume the fo

```
$t0 = 0xAAA
```

2.19.1 [5] <§2.6> I
for the following seq

```
sll $t2, $t
or  $t2, $t
```

2.19.2 [5] <§2.6> F
for the following seq

```
sll  $t2, $
andi $t2, $
```

2.19.3 [5] <§2.6> F
for the following seq

```
srl  $t2, $
andi $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 follo

```
LOOP: slt  $t2
      beq  $t2
      subi $t1
      addi $s
      j    LO
DONE:
```

2.26.1 [5] <§2.7> Assu
is the value in register $

2.26.2 [5] <§2.7> Fo
routine. Assume that th
temp, respectively.

2.26.3 [5] <§2.7> Fo
the register `$t1` is ini
executed?

2.27 [5] <§2.7> Tran
minimum number of
registers `$s0`, `$s1`, `$t0`
the base address of the

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

2.28 [5] <§2.7> How
code from Exercise 2.
elements of D are init
executed to complete

2.29 [5] <§2.7> Tra
integer `i` is held in re
`$s0` holds the base ad

```
       addi
LOOP:  lw
       add
       addi
```

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 0x7ffffffc, 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 w
explain why not. If yes, w
in f with and without th

2.36 [5] <§2.8> Right b
we know about content
we know what the entire
its declaration.

Exercises 2.37 and 2.38

2.39 [5] <§2.9> Write
`0010 0000 0000 00`
register `$t1`.

2.40 [5] <§§2.6, 2.9>
a single jump instructi

2.41 [5] <§§2.6, 2.9>
a single branch instruc

2.42 [5] <§§2.6, 2.9>
a single branch instruc

2.43 [5] <§2.10> Wr
code:

```
    lock(l
    shvar=
    unlock
```

Assume that the add
variable is in `$a1`, and
not contain any funct
operation, and the un

2.44 [5] <§2.10> R
an atomic update of
`unlock()`. Note that

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.

Exercises 2.37 and 2.38 have been deleted from this edition.

2.39 [5] <§2.9> Write the MIPS assembly code that creates the 32-bit constant `0010 0000 0000 0001 0100 1001 0010 0100`$_{two}$ and stores that value to register `$t1`.

2.40 [5] <§§2.6, 2.9> 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, 2.9> 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, 2.9> 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] <§2.10> 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] <§2.10> 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] <§2.10> 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] <§2.17> 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] <§2.17> 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] <§2.17> 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] <§2.17> 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] <§2.17> For a 50% improvement in performance, how many cycles, on average, may an arithmetic instruction take if load/store and branch instructions are not improved at all?

§2.2, page 66: MIPS, C,
§2.3, page 72: 2) Very sl
§2.4, page 79: 2) -8_{ten}
§2.5, page 87: 4) sub $t
§2.6, page 89: Both. AN
the desired field. Shiftir
of the field. Shifting rig
most bits of the word,
field where it was origi
part of the word.
§2.7, page 96: I. All are
§2.8, page 106: Both ar
§2.9, page 115: I. 4) +
§2.10, page 118: Both