Chapter 1

Find the best match of the seven great ideas from computer architecture to these real world examples:

1.Reducing time to do laundry by washing the next load while drying the last load

2.Hiding a spare key in case you lose your front door key

3.Checking the weather forecast for cites you drive through when deciding which route to take on a long winter trip

4.Express checkout lanes in grocery stores for 10 items or less

5.The local branch of a large city library system

6.Au powered by an electric motor on all four wheels

7.Optional self-driving automobiles automoblie mode that requires purchase of self-parking navigation

How do you measure fastest? Consider three different processors P1, P2, and P3 executing the same instruction set. P1 has a clock cycle time of 0.33 ns and a CPI of 1.5; P2 has a clock cycle time of 0.40 ns and a CPI of 1.0; P3 has clock cycle time of 0.25 ns and a CPI of 2.2.

1.Which has the highest clock rate? What is it?

2. Which is the fastest computer? If answer is different from above, explain why. Which is the slowest?

3. How do the answers to (1) and (2) reflect the importance of benchmarks?

Amdahl's Law and brotherhood. Amdahl's Law is basically the Law of Diminishing Returns, which applies to investments as well as computer architecture. Here is an example that helps explain the law— Your brother has joined a start-up and is trying to convince you to invest some of your savings, since, he claims, "it's a sure thing!"

1. You decide to invest 10% of your savings. What must be the return on your investment in the start-up to double your overall wealth, assuming the start-up was your only investment?

2. Assuming the start-up investment delivers the return you calculated in (1), how much of your savings would you have to invest to realize 90% of start-up's increase? How about 95%?

3. How do the results relate to Amdahl's observation about computers? What does it say about brotherhood?

DRAM price versus cost. Figure 1.21 plots the price of DRAM chips from 1975 to 2020, while Figure 1.11 shows capacity per DRAM chip over the same period. They show a 1,000,000-fold increase in capacity (16 Kbit to 16 Gbit) and a 25,000,000-fold reduction in price per gigabyte ($100 million to $4). Note that the price per GiB fluctuates up and down over time, while capacity per chip has a smooth growth curve.

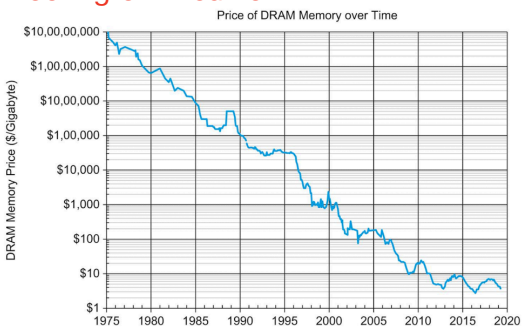

FIGURE 1.21 Price of memory per gigabyte between 1975 and 2020. (Source: https://jcmit.net/memoryprice.htm)

1. Can you see evidence of the slowing of Moore's Law in Figure 1.21?

2. Why might price improve by a factor of 25 higher than the improvement in capacity per chip? What might be other reasons than the increasing chip capacity?

3. Why do you think price per gigabyte fluctuates over 3- to 5-year periods? Is it be related to the chip cost formulas on page 28 or to other forces in the marketplace?

MORE PRACTICE PROBLEMS

1.1 [2] <§1.1> List and describe three types of computers.

1.2 [5] <§1.2> The seven great ideas in computer architecture are similar to ideas from other fields. Match the seven ideas from computer architecture, “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. 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.

Desktop processor	Year	Tech	Max. clock speed (GHz)	Integer IPC/core	Cores	Max. DRAM Bandwidth (GB/s)	SP floating point (Gflop/s)	L3 cache (MiB)
Westmere	2010	32	3.33	4	2	17.1	107	4
i7-620								
Ivy Bridge	2013	22	3.90	6	4	25.6	250	8
i7-3770K								
Broadwell	2015	14	4.20	8	4	34.1	269	8
i7-6700K								
Kaby Lake	2017	14	4.50	8	4	38.4	288	8
i7-7700K								
Coffee Lake	2019	14	4.90	8	8	42.7	627	12
i7-9700K								
Imp./year		__%	__%	__%	__%	__%	__%	__%
Doubles every		__years	__years	__years	__years	__years	__years	__years

Desktop processor	Year	Tech	Max. clock speed (GHz)	Integer IPC/core	Cores	Max. DRAM Bandwidth (GB/s)	SP floating point (Gflop/s)	L3 cache (MiB)
Westmere	2010	32	3.33	4	2	17.1	107	4
i7-620								
Ivy Bridge	2013	22	3.90	6	4	25.6	250	8
i7-3770K								
Broadwell	2015	14	4.20	8	4	34.1	269	8
i7-6700K								
Kaby Lake	2017	14	4.50	8	4	38.4	288	8
i7-7700K								
Coffee Lake	2019	14	4.90	8	8	42.7	627	12
i7-9700K								
Imp./year		__%	__%	__%	__%	__%	__%	__%
Doubles every		__years	__years	__years	__years	__years	__years	__years

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 X 1024.

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 [5] Consider the table given next, which tracks several performance indicators for Intel desktop processors since 2010. The “Tech” column shows the minimum feature size of each processor’s fabrication process. Assume that the die size has remained relatively constant and the number of transistors that comprise each processor scales at (1/t)2, where t = the minimum feature size. For each performance indicator, calculate the average rate of
improvement from 2010 to 2019, as well as the number of years required to double each at that corresponding rate.

1.7 [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 is faster: P1 or P2?
a. What is the global CPI for each implementation?
b. Find the clock cycles required in both cases.

1.8 [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.
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?

1.9 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.

1.9.1 [5] <§1.7> For each processor find the average capacitive loads.
1.9.2 [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.
1.9.3 [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.

1.10 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.

1.10.1 [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.

1.10.2 [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?

1.10.3 [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?

1.11 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.

1.11.1 [10] <§1.5> Find the yield for both wafers.

1.11.2 [5] <§1.5> Find the cost per die for both wafers.

1.11.3 [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.

1.11.4 [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.

1.12 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.

1.12.1 [5] <§§1.6, 1.9> Find the CPI if the clock cycle time is 0.333 ns.

1.12.2 [5] <§1.9> Find the SPEC ratio.

1.12.3 [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.

1.12.4 [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%.

1.12.5 [5] <§§1.6, 1.9> Find the change in the SPEC ratio for this change.

1.12.6 [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 700s and the new SPEC ratio is 13.7. Find the new CPI.

1.12.7 [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?

1.12.8 [5] <§1.6> By how much has the CPU time been reduced?

1.12.9 [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.

1.12.10 [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.

1.12.11 [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.

1.13 Section 1.11 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.

1.13.1 [5] <§§1.6, 1.11> 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.

1.13.2 [10] <§§1.6, 1.11> 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.

1.13.3 [10] <§§1.6, 1.11> 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.

1.13.4 [10] <§1.11> Another common performance figure is MFLOPS (millions of floating-point operations per second), defined as but this figure has the same problems as MIPS. Assume that 40% of the instructions executed on both P1 and P2 are floating-point instructions. Find the MFLOPS figures for the processors

1.14 Another pitfall cited in Section 1.11 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 40s spent executing branch instructions.

1.14.1 [5] <§1.11> By how much is the total time reduced if the time for FP operations is reduced by 20%?

1.14.2 [5] <§1.11> By how much is the time for INT operations reduced if the total time is reduced by 20%?

1.14.3 [5] <§1.11> Can the total time can be reduced by 20% by reducing only the time for branch instructions?

1.15 Assume a program requires the execution of 50 X 106 FP instructions, 110 X 106 INT instructions, 80 X 106 L/S instructions, and 16 X 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.

1.15.1 [10] <§1.11> By how much must we improve the CPI of FP instructions if we want the program to run two times faster?

1.15.2 [10] <§1.11> By how much must we improve the CPI of L/S instructions if we want the program to run two times faster?

1.15.3 [5] <§1.11> 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%?

1.16 [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).

Chapter 2

2.1 [5] <§2.2> For the following C statement, what is the corresponding MIPS assembly code? Assume that the C variables f, g, and h, have already been placed in registers $s0, $s1, and $s2, respectively. Use a minimal number of MIPS assembly instructions. f = g + (h – 5);

2.2 [5] <§2.2> Write a single C statement that corresponds to the two MIPS assembly instructions below.
add f, g, h
add f, i, f

2.3 [5] <§§2.2, 2.3> For the following C statement, write the corresponding 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 above, 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] <§2.3> Show how the value 0xabcdef12 would be arranged in memory of a little-endian and a big endian machine. Assume the data are stored starting at address 0 and that the word size is 4 bytes

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

2.7 [5] <§§2.2, 2.3> Translate the following C code to 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 8- byte words:

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

2.8 [10] <§§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
```

2.9 [20] <§§2.3, 2.5> For each MIPS instruction in Exercise 2.8, show the value of the opcode (op), source register (rs) and funct field, and destination register (rd) fields. For the I-type instructions, show the value of the immediate field, and for the R-type instructions, show the value of the second source register (rt).

2.10 Assume that registers $s0 and $s1 hold the values 0x8000000000000000 and 0xD000000000000000, respectively.

2.10.1 [5] <§2.4> What is the value of $t0 for the following assembly code? add $t0, $s0, $s1

2.10.2 [5] <§2.4> Is the result in $t0 the desired result, or has there been overflow?

2.10.3 [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

2.10.4 [5] <§2.4>Is the result in $t0 the desired result, or has there been overflow?

2.10.5 [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
```

2.10.6 [5] <§2.4> Is the result in $t0 the desired result, or has there been overflow?

2.11 Assume that $s0 holds the value 128 ten.

2.11.1 [5] <§2.4> For the instruction add $t0, $s0, $s1, what is the range(s) of values for $s1 that would result in overflow?

2.11.2 [5] <§2.4> For the instruction sub $t0, $s0, $s1, what is the range(s) of values for $s1 that would result in overflow?

2.11.3 [5] <§2.4> For the instruction sub $t0, $s1, $s0, what is the range(s) of values for $s1 that would result in overflow?

2.12 [5] <§§2.4, 2.5> Provide the type and assembly language instruction for the following binary value:0000 0010 0001 0000 1000 0000 0010 0000two. Hint: Figure 2.20 may be helpful.

2.13 [5] <§§2.4, 2.5> Provide the type and hexadecimal representation of following instruction: sw $t1, 32($t2)

2.14 [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

2.15 [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

2.16 [5] <§2.5> 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.

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

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

2.16.3 [5] <§§2.5, 2.10> How could each of two propose 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?

2.17 Assume the following register contents: $t0=0xAAAAAAAA, $t1=0x12345678
2.17.1 [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
```

2.17.2 [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

2.17.3 [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

2.18 [10] <§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 bits of registers $t0 and $t1 (Be sure to test your code using $t0 = 0 and $t1 = 0xffffffffffffffff . Doing so may reveal a common oversight.).

2.19 [5] <§2.6> Provide a minimal set of MIPS instructions that may be used to implement the following pseudo instruction: not $t1, $t2 // bit-wise invert

2.20 [5] <§2.6> For the following C statement, write a minimal sequence of MIPS assembly instructions that does the identical operation. Assume $t0 = A and $s0 is the base address of C.
A = C[0] << 4;

2.21 [5] <§2.7> Assume $t0 holds the value 0x010100000. 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:
```

2.22 Suppose the program counter (PC) is set to
0x20000000.
2.22.1 [5] <§2.10>What range of addresses can be reached using the MIPS jump-and-link (jai) instruction? (In other words, what is the set of possible values for the PC after the jump instruction executes?)

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

2.23 Consider a proposed new instruction named rpt. This instruction combines a loop's condition check and counter decrement into a single instruction. For example, rpt $s0, loop would do the following:

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

2.23.1 [5] <§2.7, 2.10> If this instruction were to be implemented in the MIPS instruction set, what is the most appropriate instruction format? 2.23.2 [5] <§2.7> What is the shortest sequence of MIPS instructions that performs the same operation?

2.24 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:
```

2.24.1 [5] <§2.7> Assume that the register $t1 is initialized to the value 10. What is the value in register $s0 assuming the $s0 is initially zero?

2.24.2 [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.

2.24.3 [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?

2.25 [10] <§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;
```

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

2.27 [5] <§2.7> Translate the following loop into C. Assume that the C-level integer i is held in register $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
```

2.28 [10] <§2.7> Rewrite the loop from Exercise 2.27 reduce the number of MIPS instructions executed. Hint: Notice that variable i is used only for loop control.

2.29 [30] <§2.8> Implement the following C code in MIPS assembly. Hint: Remember that the stack pointer must remain aligned on a multiple of 16.

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

2.31 [20] <§2.8> 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);
}
```

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

2.33 [5] <§2.8> Right before your function f from Exercise 2.31 returns, what do we know about contents of registers $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.

2.34 [30] <§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 $t0 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. The RISC-V mul instruction takes two registers as input. There is no “muli” instruction. Thus, just store the constant 10 in a register

2.35 [5] <§2.9> For the following code: lbu $t0, 0($t1)
sw $t0, 0($t2) Assume that the register $t1 contains the address 0x10000000 and the data at address is 0x11223344.

2.35.1 [5] <§2.3, 2.9> What value is stored in 0x10000004 on a big-endian machine?

2.35.2 [5] <§2.3, 2.9> What value is stored in
0x10000004 on a little-endian machine?

2.36 [5] <§2.10> Write the MIPS assembly code thatcreates the 32-bit constant ll/sc and stores that value to register $t1.

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

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

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

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

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

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

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

2.40.1 [5] <§2.21> Given the 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.

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

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

2.41 [10] <§2.21> Suppose the MIPS ISA included a scaled offset addressing mode similar to the ×86 one described in Section 2.19 (Figure 2.35). Describe how you would use scaled offset loads to further reduce the number of assembly instructions needed to carry out the function given in Exercise 2.4.

2.42 [10] <§2.21> Suppose the MIPS ISA included a scaled offset addressing mode similar to the ×86 one described in Section 2.19 (Figure 2.35). Describe how you would use scaled offset loads to further reduce the number of assembly instructions needed to implement the C code given in Exercise 2.7.

Chapter 3

3.1 [5] <§3.2> What is 5ED4 2 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 2 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 2 122. 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 2 214 using saturating arithmetic. The result should be written in decimal. Show your work.

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

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

3.13 [20] <§3.3> Using a table similar to that shown in Figure 3.6, calculate the product of the octal 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 3.5 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 × 2 × 2 + 1) × 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 × 33 × 0 × 55 using shifts and adds/subtracts. Assume both inputs are 8-bit unsigned integers.

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

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

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

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

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

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

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

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

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

3.27 [20] <§3.5> IEEE 754-2008 contains a half precision that is only 16 bits wide. The leftmost bit is still the sign bit, the exponent is 5 bits wide and has a bias of 15, and the mantissa is 10 bits long. A hidden 1 is assumed. Write down the bit pattern to represent −1.5625 × 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 × 101 and 4.150390625 × 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 × 100 and −1.79931640625 × 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 × 101cdivided by −4.875 × 100. 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 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 × 10−1 + 3.4375 × 10−1) + 1.771 × 103 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 × 10−1 + (3.4375 × 10−1 + 1.771 × 103) 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 andc3.33, does (3.984375 × 10−1 + 3.4375 × 10−1) + 1.771c× 103 = 3.984375 × 10−1 + (3.4375 × 10−1 + 1.771 ×c103)?

3.35 [30] <§3.9> Calculate (3.41796875 × 10−3 × 6.34765625 × 10−3) × 1.05625 × 102 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 × 10−3 × (6.34765625 × 10−3 × 1.05625 × 102) 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 × 10−3 × 6.34765625 × 10−3) × 1.05625 × 102 = 3.41796875 × 10−3 × (6.34765625 × 10−3 × 1.05625 × 102)?

3.38 [30] <§3.9> Calculate 1.666015625 × 100 × (1.9760 × 104 + −1.9744 × 104) 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 × 100 × 1.9760 × 104) + (1.666015625 × 100 × −1.9744 × 104) 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 × 100 × 1.9760 × 104) + (1.666015625 × 100 × −1.9744 × 104) = 1.666015625 × 100 × (1.9760 × 104 + −1.9744 × 104)?

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 × 4? Are they the same? What should they be? 3.43 [10] <§3.5> Write down the bit pattern in the fraction of value 1/3 assuming a floating point format that uses binary numbers in the fraction. Assume there are 24 bits, and you do not need to normalize. Is this representation exact?

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

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

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

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

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

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