Computer Organization and Design

Chapter 1

Exercise 1.1

List and describe three types of computers.

Solution

- Personal Computer: This class is made for personal use and it outcome with many kinds of workloads at reasonable performance price
- Super Computers: This class is made to outcome with the highest performance possible on specific workloads.
- Embedded Computers: This class is made to outcome with the best Watt-Performance, it outcome with a specialized and easy workload.

Exercise 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

Solution

A) Performance via Pipelining
B) Use Abstraction to Simplify Design
C) Performance via Prediction
D) Performance via Parallelism
E) Hierarchy of Memories
F) Make the Common Case Fast
G) Dependability via Redundancy

Exercise 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.

Solution

First the compiler transform the high level language to assembly code, then this assembler code pass trough the assembler to transform to machine code.

Exercise 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.

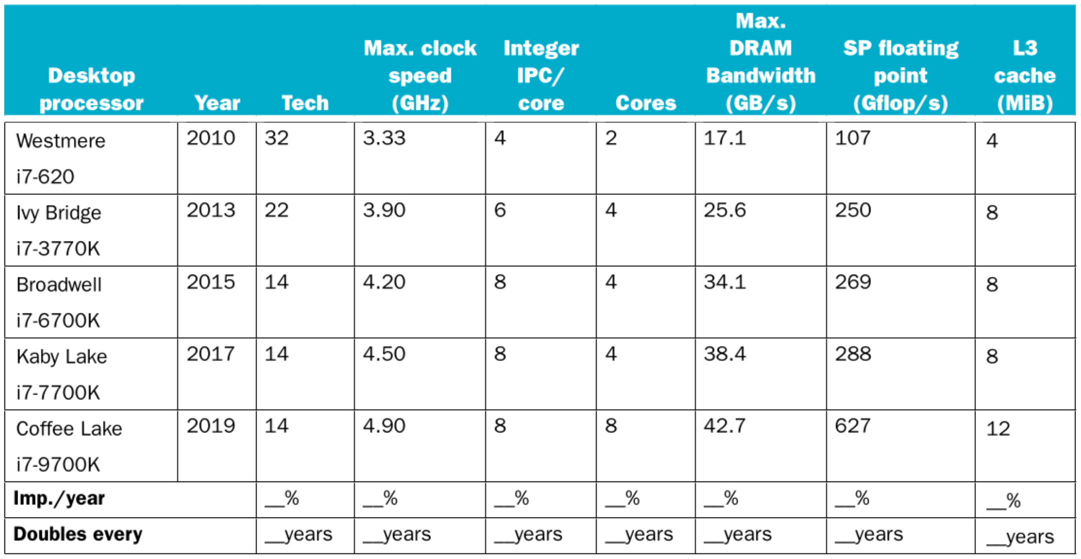
A) What is the minimum size in bytes of the frame buffer to store a frame?
B) How long would it take, at a minimum, for the frame to be sent over a 100 Mbit/s network?

Solution

A) A single frame need $1280*1024*3=3932160$ bytes, that is almost 4 MB
B) $\frac{3932160*8}{100*10^6}=0.3145728$ seconds

Exercise 1.5

Consider the table below, 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 die size has remained relatively constant, and the number of transistors comprised in 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.



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 i7-620	2010	32	3.33	4	2	17.1	107	4
Ivy Bridge i7-3770K	2013	22	3.90	6	4	25.6	250	8
Broadwell i7-6700K	2015	14	4.20	8	4	34.1	269	8
Kaby Lake i7-7700K	2017	14	4.50	8	4	38.4	288	8
Coffee Lake i7-9700K	2019	14	4.90	8	8	42.7	627	12
Imp./year		__%	__%	__%	__%	__%	__%	__%
Doubles every		__years	__years	__years	__years	__years	__years	__years

Solution

Subject	Improvement per Year	Doubles every
Tech	25.39%	7.875 Years
Max. Clock speed	16.34%	12.23 Years
Integer IPC / core	22.22%	9 Years
Cores	44.44%	4.5 Years
Max. DRAM Bandwidth	27.74%	7.2 Years
SP floating point	65.10%	3.07 Years
L3 Cache	33.33%	6 Years

Exercise 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?

Solution

A) If we want to find the number of instructions per second we need to divide the number of instructions over the time needed by the CPU to complete the task so the formula would be

$$\frac{Instructions\,count}{CPU\,time}=\frac{Clock\,rate}{CPI}$$

$$IPS_{P1}=\frac{3*10^9\,cycl/sec}{1.5\,cycl/instr}=2*10^9\,instr/sec$$

$$IPS_{P2}=\frac{2.5*10^9\,cycl/sec}{1\,cycl/instr}=2.5*10^9\,instr/sec$$

$$IPS_{P3}=\frac{4*10^9\,cycl/sec}{2.2\,cycl/instr}=1.81*10^9\,instr/sec$$

So the winner is P2

B) We can get the number of instructions using the instructions per second of before

$$Instruction\,count=Instructions\,per\,second*seconds$$

$$IC_{P1}=(10\,sec)*(2*10^9\,instr/sec)=20*10^{10}\,instructions$$

$$IC_{P2}=(10\,sec)*(2.5*10^9\,instr/sec)=25*10^{10}\,instructions$$

$$IC_{P3}=(10\,sec)*(1.81*10^9\,instr/sec)=18.1*10^{10}\,instructions$$

The number of cycles needed are

$$Number\,of\,cycles=Instructions\,count*cycles\,per\,instructions$$

$$NC_{P1}=(20*10^9\,instr)*(1.5\,cycl/instr)=3*10^{10}\,cycles$$

$$NC_{P2}=(25*10^9\,instr)*(1.0\,cycl/instr)=2.5*10^{10}\,cycles$$

$$NC_{P3}=(18.1*10^9\,instr)*(2.2\,cycl/instr)=4.0*10^{10}\,cycles$$

C) $$\frac{ET_{New}}{ET_{Old}}=\frac{Instr*(1.2*CPI)*1/CR_{New}}{Instr*CPI*1/CR_{Old}}=0.7$$

$$CR_{New}=1.2*CR_{Old}/0.7$$

Exercise 1.7

Consider two different implementations of the same instruction set architecture. The instructions can be divided into four classes according to their CPI (classes A, B, C, and D). P1 with a clock rate of 2.5 GHz and CPIs of 1, 2, 3, and 3, and P2 with a clock rate of 3 GHz and CPIs of 2, 2, 2, and 2.
Given a program with a dynamic instruction count of 1.0E6 instructions divided into classes as follows: 10% class A, 20% class B, 50% class C, and 20% class D, which is faster: P1 or P2?

A) What is the global CPI for each implementation?
B) Find the clock cycles required in both cases.

Solution

A) First we need to calculate the total time of the CPU to do the calculations for every class

$$CPUT_{P1}=\frac{(0.1*1+0.2*2+0.5*3+0.2*3)*10^6\,instr/cycle}{2.5*10^9\,Hz}=1.04*10^{-3}\,sec$$

$$CPUT_{P2}=\frac{(0.1*2+0.2*2+0.5*2+0.2*2)*10^6\,instr/cycle}{3.0*10^9\,Hz}=0.66*10^{-3}\,sec$$

Obviously P2 is faster than P1

$GCPI_{P1}=(0.1*1+0.2*2+0.5*3+0.2*3)=2.6$

$GCPI_{P2}=(0.1*2+0.2*2+0.5*2+0.2*2)=2.0$

B) Clock cycles

$CC_{P1}=2.6*10^6$

$CC_{P2}=2.0*10^6$

Exercise 1.8

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?

Solution

A) $CPI=\frac{CPU\ time}{Instruction\,count*Clock\,cycle\,time}$

$$CPI_A=\frac{1.1\,sec}{1*10^9\,instr*1*10^{-9}\,sec}=1.1$$

$$CPI_B=\frac{1.5\,sec}{1.2*10^9\,instr*1*10^{-9}\,sec}=1.25$$

B) $\frac{CR_B}{CR_A}=\frac{1.2*10^9\,instr*1.25}{1*10^9\,instr*1.1}=1.36$

C) $\frac{ET_A}{ET_C}=\frac{(1.0*10^9\,instr)*1.1*(1*10^{-9}\,sec)}{(6.0*10^8\,instr)*1.1*(1*10^{-9}\,sec)}=1.66$

$$\frac{ET_A}{ET_C}=\frac{(1.2*10^9\,instr)*1.25*(1*10^{-9}\,sec)}{(6.0*10^8\,instr)*1.1*(1*10^{-9}\,sec)}=2.5$$

Exercise 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, has a clock rate of 3.4 GHz and voltage of 0.9 V. Assume that, on average, it consumed 30 W of static power and 40 W of dynamic power.

1) For each processor find the average capacitive loads.

2) Find the percentage of the total dissipated power comprised by static power and the ratio of static power to dynamic power for each technology.
3) 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.

Solution

1) $Capacitive\ Load = \dfrac{Power}{1/2 * Frecuency * Voltage^2}$

$$CL_{Pentium} = \frac{90\,W}{1/2 * 3.6\,GHz * (1.25\,V)^2} = 32 * 10^{-9}\,F$$

$$CL_{Core} = \frac{40\,W}{1/2 * 3.4\,GHz * (0.9\,V)^2} = 29.04 * 10^{-9}\,F$$

2) $Total\ Power = \dfrac{Static\ Power}{Static\ Power + Dynamic\ Power}$

$$TP_{Pentium} = 10\,W/(10\,W + 90\,W) = 0.1 = 10\,\%$$

$$TP_{Core} = 30\,W/(30\,W + 40\,W) = 0.4285 = 42.85\,\%$$

3) $Leak\ Current = \dfrac{Power}{Voltage}$

$$LC_{Pentium} = \frac{10\,W}{1.25\,V} = 8\,A$$

$$LC_{Core} = \frac{30\,W}{0.9\,V} = 33.3\,A$$

$$Total\ Power = (1/2 * Frecuency * Capacitive\ Load * Voltage^2) + (Leak\ Current * Voltage)$$

$$0.9 * 100\,W = ((1/2 * 3.6\,GHz * 32\,nF * V^2) + 8\,A * V)$$

$$0 = 57.6\,V^2 + 8\,V - 90$$

$$V = 1.1824\,V$$

$$Total\ Power = (1/2 * Frecuency * Capacitive\ Load * Voltage^2) + (Leak\ Current * Voltage)$$

$$0.9 * 70\,W = ((1/2 * 3.4\,GHz * 29.04\,nF * V^2) + 33.3\,A * V)$$

$$0 = 49.36\,V^2 + 33.3\,V - 63$$

$$V = 0.8417\,V$$

Exercise 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 × p (where p is the number of processors) but the number of branch instructions per processor remains the same.

1) Find the total execution time for this program on 1, 2, 4, and 8 processors, and show the relative speedup of the 2, 4, and 8 processors result relative to the single processor result.
2) 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?
3) 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?

Solution

1) $$ET_{1P}=\frac{(2.56+10^9\,instr)*1+(1.28*10^9\,instr)*12+(0.256*10^9\,instr)*5}{(2*10^9)\,Hz}=9.6\,sec$$

$$ET_{2P}=\frac{\frac{(2.56+10^9\,ins)*1}{0.7*2}+\frac{(1.28*10^9\,ins)*12}{0.7*2}+(0.256*10^9\,ins)*5}{(2*10^9)\,Hz}=7.04\,sec$$

$$ET_{4P}=\frac{\frac{(2.56+10^9\,ins)*1}{0.7*4}+\frac{(1.28*10^9\,ins)*12}{0.7*4}+(0.256*10^9\,ins)*5}{(2*10^9)\,Hz}=3.84\,sec$$

$$ET_{8P}=\frac{\frac{(2.56+10^9\,ins)*1}{0.7*8}+\frac{(1.28*10^9\,ins)*12}{0.7*8}+(0.256*10^9\,ins)*5}{(2*10^9)\,Hz}=2.24\,sec$$

$Speedup_{2P}=9.6/7.04=1.36$
$Speedup_{4P}=9.6/3.84=2.5$
$Speedup_{8P}=9.6/2.24=4.28$

2) $$ET_{1P}=\frac{(2.56+10^9\,instr)*2+(1.28*10^9\,instr)*12+(0.256*10^9\,instr)*5}{(2*10^9)\,Hz}=10.88\,sec$$

$$ET_{2P}=\frac{\frac{(2.56+10^9\,ins)*2}{0.7*2}+\frac{(1.28*10^9\,ins)*12}{0.7*2}+(0.256*10^9\,ins)*5}{(2*10^9)\,Hz}=7.95\,sec$$

$$ET_{4P}=\frac{\frac{(2.56+10^9\,ins)*2}{0.7*4}+\frac{(1.28*10^9\,ins)*12}{0.7*4}+(0.256*10^9\,ins)*5}{(2*10^9)\,Hz}=4.29\,sec$$

$$ET_{8P}=\frac{\frac{(2.56+10^9\,ins)*2}{0.7*8}+\frac{(1.28*10^9\,ins)*12}{0.7*8}+(0.256*10^9\,ins)*5}{(2*10^9)\,Hz}=2.46\,sec$$

3) $$ET_{4P}=\frac{(2.56+10^9\,instr)*1+(1.28*10^9\,instr)*CPI_{LW}+(0.256*10^9\,instr)*5}{(2*10^9)\,Hz}=3.84\,sec$$

$$CPI_{LW}=\frac{(3.84\,sec*2*10^9\,Hz)-(0.256*10^9\,instr)*5-(2.56+10^9\,instr)*1}{(1.28*10^9\,instr)}=3$$

Exercise 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) Find the yield for both wafers.
2) Find the cost per die for both wafers.
3) 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.
4) 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.

Solution

1) $Die\ Area = \frac{Wafer\ Area}{Dies\ per\ Wafer}$

$Yield = \frac{1}{(1 + Defects\ per\ Area * Die\ Area/2)^2}$

I will assume that N = 2

$DA_{W1} = \frac{\pi * (15\, cm/2)^2}{84\, dies} = 2.1037\, cm2$

$Y_{W1} = \frac{1}{(1 + 0.020\, defect/cm2 * 1/2 * 2.1037\, cm2)^2} = 0.9592$

$DA_{W2} = \frac{\pi * (20\, cm/2)^2}{100\, dies} = 3.1415\, cm2$

$Y_{W2} = \frac{1}{(1 + 0.031\, defect/cm2 * 1/2 * 3.1415\, cm2)^2} = 0.9092$

2) $Cost\ per\ Die = \frac{Cost\ per\ Wafer}{Dies\ per\ Wafer * Yield}$

$CD_{W1} = \frac{12}{84 * 0.9592} = 0.1489$

$CD_{W2} = \frac{15}{100 * 0.9092} = 0.1649$

3) $DA_{W1} = \frac{\pi * (15\, cm/2)^2}{1.1 * 84\, dies} = 1.9124\, cm2$

$Y_{W1} = \frac{1}{(1 + 1.15 * 0.020\, defect/cm2 * 1/2 * 1.9124\, cm2)^2} = 0.9574$

$DA_{W2} = \frac{\pi * (20\, cm/2)^2}{1.1 * 100\, dies} = 2.8559\, cm2$

$Y_{W2} = \frac{1}{(1 + 1.15 * 0.031\, defect/cm2 * 1/2 * 2.8559\, cm2)^2} = 0.9054$

4) $Defects\ per\ Area = \frac{1/\sqrt[2]{Yield} - 1}{Die\ Area/2}$

$Defects\ per\ Area = \frac{1/\sqrt[2]{0.92} - 1}{2\, cm2/2} = 0.0425\, defects/cm2$

$Defects\ per\ Area = \frac{1/\sqrt[2]{0.95} - 1}{2\, cm2/2} = 0.0259\, defects/cm2$

Exercise 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) Find the CPI if the clock cycle time is 0.333 ns.
2) Find the SPECratio.
3) Find the increase in CPU time if the number of instructions of the benchmark is increased by 10% without affecting the CPI.
4) 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%.
5) Find the change in the SPECratio for this change.

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.
7) This CPI value is larger than obtained in 1.12.1 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?
8) By how much has the CPU time been reduced?
9) 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.
10) 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.
11) Determine the clock rate if the CPI is reduced by 15% and the CPU time by 20% while the number of instructions is unchanged.

Solution

1) $CPI=\frac{750\,s}{2.389*10^{12}*0.333*10^{-9}\,s}=0.9427$

2) $SPECratio=\frac{Reference\,Time}{Execution\,Time}$

$SPECratio=\frac{9650\,s}{750\,s}=12.8666$

3) $\frac{ET_{New}}{ET_{Old}}=\frac{1.1*2.389*10^{12}\,instr*0.9427\,cyc/instr*0.333*10^{-9}\,s}{750\,s}=1.1$

4) $\frac{ET_{New}}{ET_{Old}}=\frac{1.1*2.389*10^{12}\,instr*(1.05*0.9427\,cyc/instr)*0.333*10^{-9}\,s}{750\,s}=1.155$

5) $SPECratio=\frac{9650\,s}{1.155*750\,s}=11.1399$

6) $CPI=\frac{700\,s*4*10^{9}\,Hz}{0.85*(2.389*10^{12}\,instr)}=1.3788$

7) When we increase the clock rate we improve by 4/3 = 1.3333. But CPI increase 1.3788/0.9427 = 1.4626. Both clock rate and CPI are relative closer but is better to prioritize CPI

8) $\frac{ET_{Old}}{ET_{New}}=\frac{750\,s}{700\,s}=1.0714$

9) $IC=\frac{0.9*960\,sec*(4*10^{9}\,Hz)}{1.61\,cycles/instr}=2146.58*10^{9}\,Instr$

10) $CR=\frac{2146.58*10^{9}\,Instr*1.61\,cycles/instr}{0.8*960\,sec}=4.5\,GHz$

11) $CR=\frac{2146.58*10^{9}\,Instr*0.85*1.61\,cycles/instr}{0.8*960\,sec}=3.825\,GHz$

Exercise 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) One usual fallacy is to consider the computer with the larges clock rate as having the highest performance. Check if this is true for P1 and P2.
2) 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.
3) 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.
4) 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 processors.

Solution

1) $ET_{P1}=\frac{0.9\, cycles/instr*(5*10^9\, instr)}{4*10^9\, Hz}=1.125\, sec$

   $ET_{P2}=\frac{0.75\, cycles/instr*(1*10^9\, instr)}{3*10^9\, Hz}=0.25\, sec$

   P2 is faster than P1

2) $ET_{P1}=\frac{0.9\, cycles/instr*(1*10^9\, instr)}{4*10^9\, Hz}=0.225\, sec$

   $IC_{P2}=\frac{0.225\, sec*(3*10^9\, Hz)}{0.75\, cycles/instr}=0.9*10^9\, instr$

3) $MIPS_{P1}=\frac{(5*10^9\, instr)}{1.125\, sec*10^6}=4.44*10^3$

   $MIPS_{P2}=\frac{(1*10^9\, instr)}{0.25\, sec*10^6}=4*10^3$

4) $ET_{P1}=\frac{0.9\, cycles/instr*0.4*(5*10^9\, instr)}{4*10^9\, Hz}=0.45\, sec$

   $ET_{P2}=\frac{0.75\, cycles/instr*0.4*(1*10^9\, instr)}{3*10^9\, Hz}=0.1\, sec$

   $MFLOPS_{P1}=\frac{0.4*(5*10^9\, instr)}{0.45\, sec*10^6}=4.44*10^3$

   $MFLOPS_{P2}=\frac{0.4*(1*10^9\, instr)}{0.1\, sec*10^6}=4*10^3$

Exercise 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 40 s spent executing branch instructions.

1) By how much is the total time reduced if the time for FP operations is reduced by 20%?

2) By how much is the time for INT operations reduced if the total time is reduced by 20%?
3) Can the total time can be reduced by 20% by reducing only the time for branch instructions?

Solution

1) $RTAI = 250\,s - \left(\frac{70\,s}{1/0.8} + (250\,s - 70\,s)\right) = 14\,s$
2) I will assume the rest of seconds are INT operations 250s-70s-85s-40s=55s
3) $AOI = \frac{55\,s}{0.8 * 250\,s - (250 - 55\,s)} = 11$
4) No it can't. Because a 20% reduction is 200s and we can only reduce 40s

Exercise 1.15

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) By how much must we improve the CPI of FP instructions if we want the program to run two times faster?
2) By how much must we improve the CPI of L/S instructions if we want the program to run two times faster?
3) 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%?

Solution

1) $ET_{Total} = \frac{((1*50)+(1*110)+(4*80)+(2*16))*10^6}{2*10^9} = 256*10^{-3}\,sec$

$ET_{FP} = \frac{1\,cyc/instr\;50*10^6\,instr}{2*10^9} = 25*10^{-3}\,sec$

$0.5*256*10^{-3}\,s = (25*10^{-3}\,s)/n + (256-25)*10^{-3}\,s$

$n = \frac{(25*10^{-3}\,s)}{(0.5*256*10^{-3}\,s)-(256-25)*10^{-3}\,s} = -0.2427$

$CPI_{FP} = 1/-0.2427*1\,cyc/instr = -4.12\,cyc/instr$

No possible improvement can be done to achieve 2x performance

2) $ET_{L/S} = \frac{4\,cyc/instr\;80*10^6\,instr}{2*10^9} = 160*10^{-3}\,sec$

$0.5*256*10^{-3}\,s = (160*10^{-3}\,s)/n + (256-160)*10^{-3}\,s$

$n = \frac{(160*10^{-3}\,s)}{(0.5*256*10^{-3}\,s)-(256-160)*10^{-3}\,s} = 5$

$CPI_{L/S} = 1/5*1\,cyc/instr = 0.2\,cyc/instr$

3) $ET_{New\,Total} = \frac{((0.6*1*50)+(0.6*1*110)+(0.7*4*80)+(0.7*2*16))*10^6}{2*10^9} = 171.2*10^{-3}\,sec$

$\frac{ET_{Old}}{ET_{New}} = \frac{256\,s*10^{-3}}{171.2\,s*10^{-3}} = 1.4953$

Exercise 1.16

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).

Solution

Number of cores	Execution time	Speedup
1	100s	100s/100s=1
2	100s/2+4s=54s	100s/54s=1.8518
4	100s/4+4s=29s	100s/29s=3.4482
8	100s/8+4s=16.5s	100s/16.5s=6.0606
16	100s/16+4s=10.25s	100s/10.25s=9.756
32	100s/32+4s=7.125s	100s/7.125s=14.035
64	100s/64+4s=5.5625s	100s/5.5625s=17.9775
128	100s/128+4s=4.78125s	100s/4.78125s=20.915

Chapter 2

Exercise 2.1

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

```
f = g + (h − 5);
```

Solution

```
addi    x7, x7, -5
add     x5, x6, x7
```

Exercise 2.2

Write a single C statement that corresponds to the two RISC-V assembly instructions below.

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

Solution

```
f = i + g + h;
```

Exercise 2.3

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

```
B[8] = A[i−j];
```

Solution

```
sub     x30, x28, x29  // x30 = i-j
slli    x30, x30, 2    // x30 = 4*i_word – 4* j_word
add     x30, x30, x10  // x30 = &A[i-j]
lw      x30, 0(x30)    // x30 = A[i-j]
sw      x30, 32(x11)   // B[8] = A[i−j]
```

Exercise 2.4

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

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

addi    x12, x30, 8
lw      x30, 0(x12)
add     x30, x30, x5
```

```
lw      x30, 0(31)
```

Solution

```
addi    x12, x30, 8     // x12 = &A[f]+8
lw      x30, 0(x12)     // A[f+2] = &A[f]
add     x30, x30, x5    // x30 = &A[f] + 8 + f
lw      x30, 0(x31)     // B[g] = A[f+2+f/4]
```

Exercise 2.5

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.

Solution

Little endian:

0C	08	04	00
AB	CD	EF	12

Big Endian

0C	08	04	00
12	EF	CD	AB

Exercise 2.6

Translate 0xabcdef12 into decimal.

Solution

$10*16^7+11*16^6+12*16^5+13*16^4+14*16^3+15*16^2+1*16+2=2882400018$

Exercise 2.7

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

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

Solution

```
slli    x30, x29, 3     // x30 = j*8
add     x30, x10, x30   // x30 = A[] + j_word
ld      x30, 0(x30)     // x30 = A[j]
slli    x31, x28, 3     // x31 = i*8
add     x31, x10, x31   // x31 = A[] + i_word
ld      x31, 0(x31)     // x31 = A[i]
add     x30, x31, x30   // x30 = A[i] + A[j]
sd      x30, 64(x11)    // B[8] = A[i] + A[j]
```

Exercise 2.8

Translate the following RISC-V code to C. Assume that the variables f, g, h, i, and j are assigned to registers x5, x6, x7, x28, and x29, respectively. Assume that the base address of the arrays A and B are in registers x10 and x11, respectively.

```
addi    x30, x10, 8
addi    x31, x10, 0
sw      x31, 0(x30)
```

```
lw      x30, 0(x30)
add     x5, x30, x31
```

Solution

```
addi    x30, x10, 8     // x30 = &A[0]+8
addi    x31, x10, 0     // x31 = &A[0]
sw      x31, 0(x30)     // A[2] = &A[0]
lw      x30, 0(x30)     // A[2] = A[2]
add     x5, x30, x31    // f = A[2] + &A[0]
```

Exercise 2.9

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

Solution

```
Imm11:0        rs1   funct3  rd     op          I-Type
000000001000_01010_000_11110_0010011          addi   x30, x10, 8

Imm11:0        rs1   funct3  rd     op          I-Type
000000000000_01010_000_11111_ 0010011         addi   x31, x10, 0

Imm11:5   rs2   rs1  funct3  Imm4:0   op        S-Type
0000000_11111_11110_010_00000_0100011         sw     x31, 0(x30)

Imm11:0        rs1   funct3  rd     op          I-Type
000000000000_11110_010_11110_ 0000011         lw     x30, 0(x30)

funct7    rs2   rs1  funct3  rd    op          R-Type
0000000_11111_11110_000_00101_0110011         add    x5, x30, x31
```

Exercise 2.10

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

1. What is the value of x30 for the following assembly code?
   add x30, x5, x6
2. Is the result in x30 the desired result, or has there been overflow?
3. For the contents of registers x5 and x6 as specified above, what is the value of x30 for the following assembly code?
   sub x30, x5, x6
4. Is the result in x30 the desired result, or has there been overflow?
5. For the contents of registers x5 and x6 as specified above, what is the value of x30 for the following assembly code?
   add x30, x5, x6
   add x30, x30, x5
6. Is the result in x30 the desired result, or has there been overflow?

Solution

1. 0x5000000000000000
2. Is the desired output because the register work signed
3. 0xB000000000000000
4. Yes is the desired outcome
5. 0xD000000000000000
6. No, it had an overflow

Exercise 2.11

Assume that x5 holds the value 128.

1. For the instruction add x30, x5, x6, what is the range(s) of values for x6 that would result in overflow?
2. For the instruction sub x30, x5, x6, what is the range(s) of values for x6 that would result in overflow?
3. For the instruction sub x30, x6, x5, what is the range(s) of values for x6 that would result in overflow?

Solution

1. If we assume that the length of the registers are 32-bits and is on twos complement format then we take the twos complement of 0x80 and its 0xFFFFFF80 and we remove the sign bit so it start to overflow from 0x7FFFFF80 to 0x7FFFFFFF
2. Because if we know that when we add 0x7FFFFF80 do the sign flip then we take twos complement of that number which is 0x80000080 so the range of overflow is 0x8000007F to 0x80000000
3. Now we need to know tha range of number which will overflow when add 0xFFFFFF80, the range is from 0x80 to the most positive number 0x7FFFFFFF

Exercise 2.12

Provide the instruction type and assembly language instruction for the following binary value:
0000_0000_0001_0000_1000_0000_1011_0011

Solution

0000000_00001_00001_000_00001_0110011
First we check the opcode=0110011 and we know is a R-Type instruction. We follow with rd=00001 equivalent to x1, then funct3=000 so is and addition or substraction, then rs1=00001, that means x1, then rs2= 00001 also means x1, and finally funct7=0000000 so is an addition.

```
add x1, x1, x1
```

Exercise 2.13

Provide the instruction type and hexadecimal representation of the following instruction:

```
sw x5, 32(x30)
```

Solution

0000001_00101_11110_010_00000_0100011	Format as S-Type instruction
0000_0010_0101_1111_0010_0000_0010_0011	Format bin to hex
0x025F2023	Hex format

Exercise 2.14

Provide the instruction type, assembly language instruction, and binary representation of instruction described by the following RISC-V fields:
opcode=0x33, funct3=0x0, funct7=0x20, rs2=5, rs1=7, rd=6

Solution

The opcode indicate an R-Type instruction. The funct3 indicate add or sub. Funct7 indicate a subtraction.

```
sub     x6, x7, x5
```

Exercise 2.15

Provide the instruction type, assembly language instruction, and binary representation of instruction described by the following RISC-V fields:
opcode=0x3, funct3=0x3, rs1=27, rd=3, imm=0x4

Solution

The opcode indicate a load instruction (I-Type). Funct3 indicate a ld instruction.

```
ld      x3, 4(x27)
```

Exercise 2.16

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

1. How would this affect the size of each of the bit fields in the R-type instructions?
2. How would this affect the size of each of the bit fields in the I-type instructions?
3. How could each of the two proposed changes decrease the size of a RISC-V assembly program? On the other hand, how could the proposed change increase the size of an RISC-V assembly program?

Solution

1. Well the opcode probably won't need changes however if we need 4 times more instructions probably if we encode it to funct7 we wouldn't need to add more bits, however if it couldn't be then we need add 2 bits to funct3. And to the register we would need 7 bits to encode all those register so rd, rs1 and rs2 need 7 bits. So in worst case scenario we need 7+7+5+7+7+7=40 bits best case scenario 38 bits.
2. Same opcode is the same, funct3 need 2 bits more, rs1 and rd would be 7 bits each. So 7+7+5+7+12=38 bits
3. The only way to decrease the size with those changes is making those new instructions a complex set of instructions. But if this change is not taken then the size of programs will increase

Exercise 2.17

Assume the following register contents:
x5 = 0x00000000AAAAAAAA, x6 = 0x1234567812345678

1. For the register values shown above, what is the value of x7 for the following sequence of instructions?

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

2. For the register values shown above, what is the value of x7 for the following sequence of instructions?
```
        slli    x7, x6, 4
```
3. For the register values shown above, what is the value of x7 for the following sequence of instructions?
```
        srli    x7, x5, 3
        andi    x7, x7, 0xFEF
```

Solution

1. Value of x7=0x1234FEFABABE5678
```
slli    x7, x5, 4       // x7=0x0000AAAAAAAA0000
or      x7, x7, x6      // x7=0x1234FEFABABE5678
```
2. Value of x7=0x5678123456780000
```
slli    x7, x6, 4       // x7=0x5678123456780000
```
3. Value of x7=0x0000000000000AAA
```
srli    x7, x5, 3       // x7=0x00000000000AAAAA
andi    x7, x7, 0xFEF // x7=0x0000000000000AAA
```

Exercise 2.18

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

Solution

```
srli    x27, x5, 11         // shift 16:11 to 4:0
andi    x27, x27, 0x3F      // turn 31:5 on 0s and keeping 4:0
slli    x27, x27, 26        // shift 4:0 to 31:26
lui     x28, 0xFC000        // twos complement of 0x03FFF
xori    x28, x28, 0xFFF     // 0xFC000000 XOR 0xFFFFFFFF = 0x03FFFFFF
and     x6, x6, x28         // turn 31:26 to 0s and keeping 25:0
or      x6, x6, x27         // Mix x27 31:26 bits with x28 25:0 bits
```

Exercise 2.19

Provide a minimal set of RISC-V instructions that may be used to implement the following pseudoinstruction:
```
not x5, x6      // bit-wise invert
```

Solution

```
addi    x27, x0, 0xFFE      // x27 = 0xFFFFFFFE
xor     x5, x6, x27         // invert bits from 31:1 and keep bit 0
```

Exercise 2.20

For the following C statement, write a minimal sequence of RISC-V assembly instructions that performs the identical operation. Assume x6 = A, and x17 is the base address of C.
```
A = C[0] << 4;
```

Solution

```
lw      x27, 0(x17)
```

```
        slli    x6, x27, 4
```

Exercise 2.21

Assume x5 holds the value 0x00000000001010000. What is the value of x6 after the following instructions?

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

Solution

x6=0x2

Exercise 2.22

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

1. What range of addresses can be reached using the RISC-V jump-and-link (jal) instruction? (In other words, what is the set of possible values for the PC after the jump instruction executes?)
2. What range of addresses can be reached using the RISC-V branch if equal (beq) instruction? (In other words, what is the set of possible values for the PC after the branch instruction executes?)

Solution

1. From 0x1FF00002 to 0x200FFFFE
2. From 0x1FFFF002 to 0x20000FFE

Exercise 2.23

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

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

1. If this instruction were to be added to the RISC-V instruction set, what is the most appropriate instruction format?
2. What is the shortest sequence of RISC-V instructions that performs the same operation?

Solution

1. The best format is I-Type
2. The following:

```
LOOP:
    bge     x0, x29, DONE
    addi    x29, x20, -1
    jal     x0, LOOP
DONE:
```

Exercise 2.24

Consider the following RISC-V loop:

LOOP:

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

1. Assume that the register x6 is initialized to the value 10. What is the final value in register x5 assuming the x5 is initially zero?
2. For the loop above, write the equivalent C code. Assume that the registers x5 and x6 are integers acc and i, respectively.
3. For the loop written in RISC-V assembly above, assume that the register x6 is initialized to the value N. How many RISC-V instructions are executed?
4. For the loop written in RISC-V assembly above, replace the instruction “beq x6, x0, DONE” with the instruction “blt x6, x0, DONE” and write the equivalent C code.

Solution

1. x5 = 20
2. Equivalent code:
```
while(i!=0) {
    accum = accum + 2;
    i = i – 1;
}
```
3. 4*N+1
4. Replaced Line:
```
while(i>=0) {
    accum = accum + 2;
    i = i – 1;
}
```

Exercise 2.25

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

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

Solution

```
        add     x7, x0, x0              // i = 0
        add     x29, x0, x0             // j = 0
LOOP1:
        bge     x7, x5, DONE1           // if (i<a)

LOOP2:
        bge     x29, x6, DONE2          // if (j<b)

        add     x30, x7, x29            // x30 = i+j
        slli    x31, x29, 4             // x31 = 4*(4*j)
        add     x31, x31, x10           // x31 = &D[0] + 16 * j
        sw      x30, 0(x31)             // D[4*j] = i + j
        addi    x29, x29, 1             // j++
        jal     x0, LOOP2

DONE2:
        addi    x7, x7, 1               // i++
        jal     x0, LOOP1
DONE1:
```

Exercise 2.26

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

Solution

It takes 12 instructions, I take 4 instructions for doing "D[4*j] = i + j" which repeats a*b times. For the second loop it takes (3*b+1)*a. Then the first loop 3*a+1. Finally 2 instructions more for set i and j to 0, makes the count of (10*1)+((3*1+1)*10)+(3*10+1)+2 = 83 instructions

Exercise 2.27

Translate the following loop into C. Assume that the C-level integer i is held in register x5, x6 holds the C-level integer called result, and x10 holds the base address of the integer MemArray.

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

Solution

/*Pending*//*Because it doesn't make sense the values, what is the value of x5 at first? Is i and result registers switched?*/

Exercise 2.28

Rewrite the loop from Exercise 2.27 to reduce the number of RISC-V instructions executed. Hint: Notice that variable i is used only for loop control.

Solution

/*Pending*//*Well...*/

Exercise 2.29

Implement the following C code in RISC-V assembly. Hint: Remember that the stack pointer must remain aligned on a multiple of 16.

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

Solution

```
FIB:
        addi    x2, x2, -16
        lw      x1, 0(x2)
        lw      x18, 4(x2)
        lw      x19, 8(x2)

        addi    x28, x0, 0
        bne     x10, x28, IFELSE
        addi    x10, x0, 0      // return 0
        jal     x1, FIB
        jal     x0, DONE
IFELSE:
        addi    x28, x0, 1
        bne     x10, x28, ELSE
        addi    x10, x0, 1      // return 1
        jal     x1, FIB
        jal     x0, DONE
ELSE:
        add     x18, x0, x10    // x18 = num
        addi    x10, x18, -1
        jal     x1, FIB
        add     x19, x0, x10    // sum = fib(n-1)
        addi    x10, x18, -2
        jal     x1, FIB
        add     x19, x19, x10   // sum = sum + fib(n-2)
        add     x10, x0, x19    // return sum
DONE:
        sw      x1, 0(x2)
        sw      x18, 4(x2)
        sw      x19, 8(x2)
        addi    x2, x2, 16
        jalr    x0, 0(x1)
```

Exercise 2.30

For each function call in Exercise 2.29, show the contents of the stack after the function call is made. Assume the stack pointer is originally at address 0x7ffffffc, and follow the register conventions as specified in Figure 2.11.

Solution

This is the initial state

```
0x7ffffffc:     0x0     <-x2<-x8
```

Now we will suppose that x1= 0x2000, x18 = 0x20, x19 = 0x30 and we pass x10=0x1

```
0x7ffffffc:     0x0     <-x8
```

```
0x7ffffff8:     0x0
0x7ffffff4:     0x30
0x7ffffff0:     0x20
0x7fffffec:     0x2000<-x2
```
Now let’s suppose that our program run on 0x8000 and we hit the fib(1) were lies 12 instructions down from our starting point so the return address will be 0x8034 (12*4+4), also x18 and x19 will be stored in the stack. I know the values of those register haven’t change yet but imagine if not and have other values, just for demonstration so x18=0x90 and x19=0xFF.
```
0x7ffffffc:     0x0     <-x8
0x7ffffff8:     0x0
0x7ffffff4:     0x30
0x7ffffff0:     0x20
0x7fffffec:     0x2000
0x7fffffec:     0x0
0x7fffffe8:     0x0
0x7fffffe4:     0xFF
0x7fffffe0:     0x90
0x7fffffdc:     0x8034<-x2
```
Then it will return to x10 the value of 1 and it will load all values to the variables in case if they were modified. So x1= 0x8034, x18=0x90, x19=0xFF and add 16 to the stack
```
0x7ffffffc:     0x0     <-x8
0x7ffffff8:     0x0
0x7ffffff4:     0x30
0x7ffffff0:     0x20
0x7fffffec:     0x2000<-x2
```
Finally when fib(num) is finally calculated we load the original values before the program was run. x1=0x2000, x18=0x20 and x19=0x30 and add 16 to the stack
```
0x7ffffffc:     0x0     <-x2<-x8
```

Exercise 2.31

Translate function f into RISC-V assembly language. Assume the function declaration for g is int g(int a, int b). The code for function f is as follows:
```
        int f(int a, int b, int c, int d){
                return g(g(a,b), c+d);
        }
```

Solution

```
F:
        addi    x2, x2, -16
        lw      x1, 0(x2)
        lw      x18, 4(x2)

        add     x18, x12, x13           // x18 = c+d
        jal     x1, G                   // x10 = g(a,b)
        add     x11, x0, x18
        jal     x1, G                   // x10 = g(g(a,b), c+d)

        sw      x1, 0(x2)
        sw      x18, 4(x2)
        addi    x2, x2, 16
        jalr    x0, 0(x1)            // return g(g(a,b), c+d)
```

Exercise 2.32

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?

Solution

/*Pending*/

Exercise 2.33

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

Solution

/*Pending*/

Exercise 2.34

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

Solution

/*Pending*/

Exercise 2.35

Consider the following code:

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

Assume that the register x7 contains the address 0×10000000 and the data at address is 0×1122334455667788.

1. What value is stored in 0×10000008 on a big-endian machine?
2. What value is stored in 0×10000008 on a little-endian machine?

Solution

/*Pending*/

Exercise 2.36

Write the RISC-V assembly code that creates the 32-bit constant 0x12345678 and stores that value to register x10.

Solution

/*Pending*/

Exercise 2.37

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

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

Solution

/*Pending*/

Exercise 2.38

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.

Solution

/*Pending*/

Exercise 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.

1. Suppose that new, more powerful arithmetic instructions are added to the instruction set. On average, through the use of these more powerful arithmetic instructions, we can reduce the number of arithmetic instructions needed to execute a program by 25%, while increasing the clock cycle time by only 10%. Is this a good design choice? Why?
2. 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?

Solution

/*Pending*/

Exercise 2.40

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

1. Given this instruction mix and the assumption that an arithmetic instruction requires two cycles, a load/store instruction takes six cycles, and a branch instruction takes three cycles, find the average CPI.
2. 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?

3. 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?

Solution

/*Pending*/

Exercise 2.41

Suppose the RISC-V ISA included a scaled offset addressing mode similar to the x86 one described in Section 2.19 (Figure 2.39). 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.

Solution

/*Pending*/

Exercise 2.42

Suppose the RISC-V ISA included a scaled offset addressing mode similar to the x86 one described in Section 2.19 (Figure 2.39). 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

Solution

/*Pending*/