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

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

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

(3 × 8) × 4tu = 96 time units for hardware

(5 × 8) × 4tu = 160 time units for software

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

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

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

3.16 It takes B time units to get through an adder, and the adders are arranged in a tree structure. It will require log2(A) levels. An 8 bit wide word requires seven adders in three levels. 3 × 4tu = 12 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 three 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.

3.17 0x33 × 0x55 = 0x10EF. 0x33 = 51, and 51 = 32 + 16 + 2 + 1. We can shift 0x55 left five places (0xAA0), then add 0x55 shifted left four places (0x550), then add 0x55 shifted left once (0xAA), then add 0x55. 0xAA0 + 0x550 + 0xAA + 0x55 = 0x10EF. Three shifts, three adds.

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

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.

3.18 74/21 = 3 remainder 9

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

3.19 [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.)

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

74/21 = 3 remainder 11

Step	Action	Divisor	Remainder/Quotient
0	Initial Vals	010 001	000 000 111 100
1	R<<	010 001	000 001 111 000
	Rem=Rem–Div	010 001	111 000 111 000
	Rem<0,R+D	010 001	000 001 111 000
2	R<<	010 001	000 011 110 000
	Rem=Rem–Div	010 001	110 010 110 000
	Rem<0,R+D	010 001	000 011 110 000
3	R<<	010 001	000 111 100 000
	Rem=Rem–Div	010 001	110 110 110 000
	Rem<0,R+D	010 001	000 111 100 000
4	R<<	010 001	001 111 000 000
	Rem=Rem–Div	010 001	111 110 000 000
	Rem<0,R+D	010 001	001 111 000 000
5	R<<	010 001	011 110 000 000
	Rem=Rem–Div	010 001	111 110 000 000
	Rem>0,R0=1	010 001	001 101 000 001
6	R<<	010 001	011 010 000 010
	Rem=Rem–Div	010 001	001 001 000 010
	Rem>0,R0=1	010 001	001 001 000 011

3.22 [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.

3.22 0×0C000000 = 0000 1100 0000 0000 0000 0000 0000 0000

= 0 0001 1000 0000 0000 0000 0000 0000 000

sign is positive

exp = 0 × 18 = 24 − 127 = −103

there is a hidden 1

mantissa = 0

answer = 1.0×2^{-103}

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

normalize, move binary point five to the left

1.1111101×2^5

sign = positive, exp = 127 + 5 = 132

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

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

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

normalize, move binary point five to the left

1.1111101×2^5

sign = positive, exp = 1023 + 5 = 1028

Final bit pattern:

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

= 0x404FA00000000000

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?

3.41

Answer	sign	exp	Exact?
1 01111101 00000000000000000000000	–	−2	Yes

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

3.42 b+b+b+b = −1

b × 4 = −1

They are the same

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?

3.43 0101 0101 0101 0101 0101 0101

No

1.6 [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?

a. performance of P1 (instructions/sec) $= 3 \times 10^9 / 1.5 = 2 \times 10^9$

performance of P2 (instructions/sec) $= 2.5 \times 10^9 / 1.0 = 2.5 \times 10^9$

performance of P3 (instructions/sec) $= 4 \times 10^9 / 2.2 = 1.8 \times 10^9$

b. cycles(P1) $= 10 \times 3 \times 10^9 = 30 \times 10^9$ s

cycles(P2) $= 10 \times 2.5 \times 10^9 = 25 \times 10^9$ s

cycles(P3) $= 10 \times 4 \times 10^9 = 40 \times 10^9$ s

c. No. instructions(P1) $= 30 \times 10^9 / 1.5 = 20 \times 10^9$

No. instructions(P2) $= 25 \times 10^9 / 1 = 25 \times 10^9$

No. instructions(P3) $= 40 \times 10^9 / 2.2 = 18.18 \times 10^9$

$CPI_{new} = CPI_{old} \times 1.2$, then CPI(P1) $= 1.8$, CPI(P2) $= 1.2$, CPI(P3) $= 2.6$

f = No. instr. × CPI/time, then

f(P1) $= 20 \times 10^9 \times 1.8/7 = 5.14$ GHz

f(P2) $= 25 \times 10^9 \times 1.2/7 = 4.28$ GHz

f(P1) $= 18.18 \times 10^9 \times 2.6/7 = 6.75$ GHz

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 (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.

a. Class A: 10^5 instr. Class B: 2×10^5 instr. Class C: 5×10^5 instr. Class D: 2×10^5 instr.

Time = No. instr. × CPI/clock rate

Total time P1 = $(10^5 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 3 + 2 \times 10^5 \times 3)/(2.5 \times 10^9) = 10.4 \times 10^{-4}$ s

Total time P2 = $(10^5 \times 2 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 2 + 2 \times 10^5 \times 2)/(3 \times 10^9) = 6.66 \times 10^{-4}$ s

CPI(P1) = $10.4 \times 10^{-4} \times 2.5 \times 10^9/10^6 = 2.6$

CPI(P2) = $6.66 \times 10^{-4} \times 3 \times 10^9/10^6 = 2.0$

b. clock cycles(P1) = $10^5 \times 1 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 3 + 2 \times 10^5 \times 3 = 26 \times 10^5$

clock cycles(P2) = $10^5 \times 2 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 2 + 2 \times 10^5 \times 2 = 20 \times 10^5$

1.8 [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.

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?

a. CPU time = Instruction count × CPI × Clock cycle time
CPI = CPU time / (Instruction count × Clock cycle time)

Compiler A CPI = 1.1 / (1.0E9 × 1.0E-9) = 1.1
Compiler B CPI = 1.5 / (1.2E9 × 1.0E-9) = 1.25

b. 1/ Clock cycle time = Instruction count × CPI / CPU time
Clock rate (Frequency) = Instruction count × CPI / CPU time

For same CPU or execution time on both processors,
f_B/f_A = (Instruction count(B) × CPI(B)) / (Instruction count (A) × CPI(A)) = 1.37

c. For the original processor with a clock cycle time of 1 ns:

$(\text{CPU time})_A / (\text{CPU time})_{NEW} = (\text{Instruction count} \times \text{CPI})_A / (\text{Instruction count} \times \text{CPI})_{NEW}$
$= (1.0E9 \times 1.1)_A / (6.0E8 \times 1.1)_{NEW}$
$= 1.67$

$(\text{CPU time})_B / (\text{CPU time})_{NEW} = (\text{Instruction count} \times \text{CPI})_B / (\text{Instruction count} \times \text{CPI})_{NEW}$
$= (1.2E9 \times 1.25)_B / (6.0E8 \times 1.1)_{NEW}$
$= 2.27$

or

a. $CPI = T_{exec} \times f/\text{No. instr.}$

Compiler A CPI = 1.1

Compiler B CPI = 1.25

b. $f_B/f_A = (\text{No. instr.(B)} \times \text{CPI(B)})/(\text{No. instr.(A)} \times \text{CPI(A)}) = 1.37$

c. $T_A/T_{new} = 1.67$

$T_B/T_{new} = 2.27$

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.

1.9.1 [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.

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

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

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

1.8.2 Pentium 4: $10/100 = 10\%$

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

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

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

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

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

Therefore:

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

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

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

Pentium 4:

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

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

$V_{new} = [(90 - V_{new} \times 8)/(3.2E8 \times 3.6E9)]^{1/2}$

$V_{new} = 0.85\,V$

Core i5:

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

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

$V_{new} = [(63 - V_{new} \times 33.3)/(2.9E8 \times 3.4E9)]^{1/2}$

$V_{new} = 0.64\,V$

(Numbering may vary, but correct only)

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 $0.7 \times p$ (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 processors 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?

1.9.1

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

1.9.2

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

1.9.3 3

(Numbering may vary, but correct only)

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

1.11.1 [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 mm^2.

1.10

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

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

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

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

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

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

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

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

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

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

1.10.4 defects per area$_{0.92}$ = $(1-y^{-5})/(y^{-5} \times die\_area/2) = (1 - 0.92^{-5})/(0.92^{-5} \times 2/2) = 0.043$ defects/cm^2

defects per area$_{0.95}$ = $(1-y^{-5})/(y^{-5} \times die\_area/2) = (1 - 0.95^{-5})/(0.95^{-5} \times 2/2) = 0.026$ defects/cm^2

(Numbering may vary, but correct only)

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 SPECratio.

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 SPECratio 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 700 s and the new SPECratio 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 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.

1.11

1.11.1 CPI = clock rate × CPU time/instr. count

clock rate = 1/cycle time = 3 GHz

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

1.11.2 SPEC ratio = ref. time/execution time

SPEC ratio(bzip2) = 9650/750 = 12.86

1.11.3 CPU time = No. instr. × CPI/clock rate

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

1.11.4 CPU time(before) = No. instr. × CPI/clock rate

CPU time(after) = 1.1 × No. instr. × 1.05 × CPI/clock rate

CPU time(after)/CPU time(before) = 1.1 × 1.05 = 1.155. Thus, CPU time is increased by 15.5%.

1.11.5 SPECratio = reference time/CPU time

SPECratio(after)/SPECratio(before) = CPU time(before)/CPU time(after) = 1/1.1555 = 0.86. The SPECratio is decreased by 14%.

1.11.6 CPI = (CPU time × clock rate)/No. instr.

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

1.11.7 Clock rate ratio = 4 GHz/3 GHz = 1.33

CPI @ 4 GHz = 1.37, CPI @ 3 GHz = 0.94, ratio = 1.45

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

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

1.11.9 No. instr. = CPU time × clock rate/CPI

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

1.11.10 Clock rate = No. instr. × CPI/CPU time.

Clock rate_{new} = No. instr. × CPI/0.9 × CPU time = 1/0.9 clock rate_{old} = 3.33 GHz

1.11.11 Clock rate = No. instr. × CPI/CPU time.

Clock rate_{new} = No. instr. × 0.85 × CPI/0.80 CPU time = 0.85/0.80, clock rate_{old} = 3.18 GHz

(Numbering may vary, but correct only)

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 highest 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

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

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

1.12

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

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

clock rate(P1) > clock rate(P2), performance(P1) < performance(P2)

1.12.2 T(P1) = No. instr. × CPI/clock rate

T(P1) = 2.25 3 1021 s

T(P2) 5 N × 0.75/(3×10^9), then N = 9×10^8

1.12.3 MIPS = Clock rate × 10^{-6}/CPI

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

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

MIPS(P1) > MIPS(P2), performance(P1) < performance(P2) (from 11a)

1.12.4 MFLOPS = No. FP operations × 10^{-6}/T

MFLOPS(P1) = .4 × 5E9 × 1E-6/1.125 = 1.78E3

MFLOPS(P2) = .4 × 1E9 × 1E-6/.25 = 1.60E3

MFLOPS(P1) > MFLOPS(P2), performance(P1) < performance(P2) (from 11a)

(Numbering may vary, but correct only)

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.

1.14.1 [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?

1.13

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

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

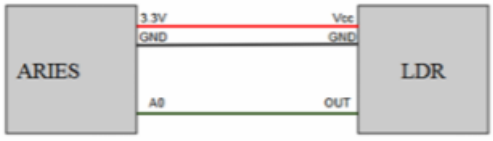
1.13.3 $T_{new} = 250 \times 0.8 = 200$ s, $T_{fp} + T_{int} + T_{l/s} = 210$ s. NO

(Numbering may vary, but correct only)

---

a.Interface LDR/Temp Sensor and compare with reference and operate a relay

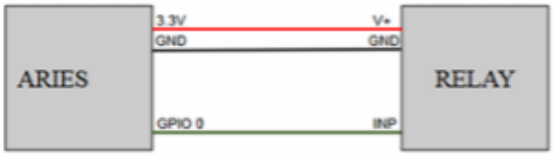
7. LDR SENSOR



ADC -> LDR_Sensor

b.Interface a LDR sensor and display the converted values on UART terminal

10. DC RELAY MODULE



GPIO -> DC_Relay_Demo

(Yes, this will be tested in exam as well)