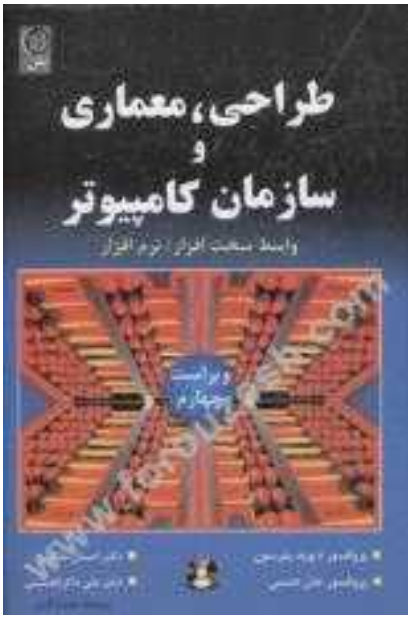
دانشگاه آزاد اسلامی واحد بافت

حل تمرین طراحی ، معماری
و
سازمان کامپیوتر

واسط سخت افزار / نرم افزار



تهیه کننده :

کاظم فریدی

دانشجوی کارشناسی ارشد مهندسی معماری سیستمهای کامپیوتری

1.1 تمرین

1.1.1
5 supercomputers

1.1.2 7 petabyte

1.1.3 3 servers

1.1.4 1 virtual worlds

1.1.5 12 RAM

1.1.6 : 13 CPU

1.1.7 8 datacenters

1.1.8 10 multi- core processors

1.1.9 4 low-end servers

1.1.10 9 embedded computers

1.1.11 11 VHDL

1.1.12 2 desktop computers

1.1.13 15 compiler

1.1.14
21 assembler

1.1.15 25 cobol

1.1.16 19 machine language

1.1.17 17 instruction

1.1.18 26 fortran

1.1.19 18 assembly language

1.1.20 14 operating system

1.1.21 24 application software

1.1.22 16 bit

1.1.23 23 system software

1.1.24 20 C

1.1.25 22 high-level language

1.1.26 6 terabyte

تمرین 1.2

1.2.1 8 bits × 3 colors = 24 bits/pixel = 4 bytes/pixel. 1280 × 800 pixels = 1,024,000 pixels. 1,024,000 pixels × 4 bytes/pixel = 4,096,000 bytes (approx 4 Mbytes).

1.2.2 2 GB = 2000 Mbytes. No. frames = 2000 Mbytes/4 Mbytes = 500 frames.

1.2.3 Network speed: 1 gigabit network ==> 1 gigabit/per second = 125 Mbytes/ second. File size: 256 Kbytes = 0.256 Mbytes. Time for 0.256 Mbytes = 0.256/125 = 2.048 ms

1.2.4 2 microseconds from cache ==> 20 microseconds from DRAM. 20 micro- seconds from DRAM ==> 2 seconds from magnetic disk. 20 microseconds from DRAM ==> 2 ms from flash memory.

تمرین 1.3

1.3.1 P2 has the highest performance

performance of P1 (instructions/sec) = $2 \times 10^9/1.5 = 1.33 \times 10^9$ performance of P2 (instructions/sec) = $1.5 \times 10^9/1.0 = 1.5 \times 10^9$ performance of P3 (instructions/sec) = $3 \times 10^9/2.5 = 1.2 \times 10^9$

1.3.2 No. cycles = time × clock

rate cycles(P1) = $10 \times 2 \times 10^9 =$

20×10^9 s
cycles(P2) = $10 \times 1.5 \times 10^9 = 15 \times 10^9$ s cycles(P3) = $10 \times 3 \times 10^9 = 30 \times 10^9$ s

time = (No. instr. × CPI)/clock rate, then No. instructions = No. cycles/CPI

instructions(P1) = $20 \times 10^9/1.5 = 13.33 \times 10^9$ instructions(P2) = $15 \times 10^9/1 = 15 \times 10^9$ instructions(P3) = $30 \times 10^9/2.5 = 12 \times 10^9$

1.3.3 $\text{time}_{\text{new}} = \text{time}_{\text{old}} \times 0.7 = 7$ s

CPI = CPI × 1.2, then CPI(P1) = 1.8, CPI(P2) = 1.2, CPI(P3) = 3

f = No. instr. × CPI/time, then

f(P1) = $13.33 \times 10^9 \times 1.8/7 = 3.42$ GHz f(P2) = $15 \times 10^9 \times 1.2/7 =$ 2.57 GHz f(P3) = $12 \times 10^9 \times 3/7 =$ 5.14 GHz

1.3.4 IPC = 1/CPI = No. instr./(time × clock rate) IPC(P1) = 1.42
IPC(P2) = 2
IPC(P3) = 3.33

1.3.5 $\text{Time}_{new}/\text{Time}_{old} = 7/10 = 0.7$. So $f_{new} = f_{old}/0.7 = 1.5\,\text{GHz}/0.7 = 2.14\,\text{GHz}$.

1.3.6 $\text{Time}_{new}/\text{Time}_{old} = 9/10 = 0.9$.
So $\text{Instructions}_{new} = \text{Instructions}_{old} \times 0.9 = 30 \times 10^9 \times 0.9 = 27 \times 10^9$.

1.4 تمرین

1.4.1 P2

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

P1: Time class A = 0.66×10^{-4}
Time class B = 2.66×10^{-4}
Time class C = 10×10^{-4}
Time class D = 5.33×10^{-4}

Total time P1 = 18.65×10^{-4}

P2: Time class A = 10^{-4}
Time class B = 2×10^{-4}
Time class C = 5×10^{-4}
Time class D = 3×10^{-4}

Total time P2 = 11×10^{-4}

1.4.2 CPI = time × clock rate/No. instr. CPI(P1) = $18.65 \times 10^{-4} \times 1.5 \times 10^9/10^6 = 2.79$
CPI(P2) = $11 \times 10^{-4} \times 2 \times 10^9/10^6 = 2.2$

1.4.3

clock cycles(P1) = $10^5 \times 1 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 3 + 2 \times 10^5 \times 4 = 28 \times 10^5$
clock cycles(P2) = $10^5 \times 2 + 2 \times 10^5 \times 2 + 5 \times 10^5 \times 2 + 2 \times 10^5 \times 3 = 22 \times 10^5$

1.4.4

$(500 \times 1 + 50 \times 5 + 100 \times 5 + 50 \times 2) \times 0.5 \times 10^{-9} = 675\,\text{ns}$

1.4.5 CPI = time × clock rate/No. instr.

$\text{CPI} = 675 \times 10^{-9} \times 2 \times 10^9/700 = 1.92$

1.4.6

$\text{Time} = (500 \times 1 + 50 \times 5 + 50 \times 5 + 50 \times 2) \times 0.5 \times 10^{-9} = 550\,\text{ns}$

Speed-up = 675 ns/550 ns = 1.22

$\text{CPI} = 550 \times 10^{-9} \times 2 \times 10^9/700 = 1.57$

1.5 تمرین

1.5.1

a.	1G, 0.75G inst/s
b.	1G, 1.5G inst/s

1.5.2

a.	P2 is 1.33 times faster than P1
b.	P1 is 1.03 times faster than P2

1.5.3

a.	P2 is 1.31 times faster than P1
b.	P1 is 1.00 times faster than P2

1.5.4

a.	2.05 µs
b.	1.93 µs

1.5.5

a.	0.71 µs
b.	0.86 µs

1.5.6

a.	1.30 times faster
b.	1.40 times faster

1.6 تمرین

1.6.1

	Compiler A CPI	Compiler B CPI
a.	1.00	1.17
b.	0.80	0.58

1.6.2

a.	0.86
b.	1.37

1.6.3

	Compiler A speed-up	Compiler B speed-up
a.	1.52	1.77
b.	1.21	0.88

1.6.4

	P1 peak	P2 peak
a.	4G Inst/s	3G Inst/s
b.	4G Inst/s	3G Inst/s

1.6.5 Speed-up, P1 versus P2:

a.	0.967105263
b.	0.730263158

1.6.6

a.	6.204081633
b.	8.216216216

1.7 تمرین

1.7.1

Geometric mean clock rate ratio = $(1.28 \times 1.56 \times 2.64 \times 3.03 \times 10.00 \times 1.80 \times 0.74)^{1/7} = 2.15$
Geometric mean power ratio = $(1.24 \times 1.20 \times 2.06 \times 2.88 \times 2.59 \times 1.37 \times 0.92)^{1/7} = 1.62$

1.7.2

Largest clock rate ratio = 2000 MHz/200 MHz = 10 (Pentium Pro to Pentium 4 Willamette)

Largest power ratio = 29.1 W/10.1 W = 2.88 (Pentium to Pentium Pro

1.7.3

Clock rate: $2.667 \times 10^9/12.5 \times 10^6 = 212.8$
Power: 95 W/3.3 W = 28.78

1.7.4 $C = P/V^2 \times$ clockrate

80286: $C = 0.0105 \times 10^{-6}$
80386: $C = 0.01025 \times 10^{-6}$
80486: $C = 0.00784 \times 10^{-6}$
Pentium: $C = 0.00612 \times 10^{-6}$
Pentium Pro: $C = 0.0133 \times 10^{-6}$
Pentium 4 Willamette: $C = 0.0122 \times 10^{-6}$
Pentium 4 Prescott: $C = 0.00183 \times 10^{-6}$

Core 2: $C = 0.0294 \times 10^{-6}$

1.7.5 3.3/1.75 = 1.78 (Pentium Pro to Pentium 4 Willamette)

1.7.6

Pentium to Pentium Pro: 3.3/5 = 0.66
Pentium Pro to Pentium 4 Willamette: 1.75/3.3 = 0.53
Pentium 4 Willamette to Pentium 4 Prescott: 1.25/1.75 = 0.71
Pentium 4 Prescott to Core 2: 1.1/1.25 = 0.88
Geometric mean = 0.68

تمرین 1.8

1.8.1 $Power_1 = V^2 \times clock\ rate \times C$. $Power_2 = 0.9\ Power_1$

1.8.2 $Power_2/Power_1 = V_2^2 \times clock\ rate_2/V^2 \times clock\ rate_1$

1.8.3

$Power_2 = V_2^2 \times 1 \times 10^9 \times 0.8 \times C_1 = 0.6 \times Power_1$

$Power_1 = 5^2 \times 0.5 \times 10^9 \times C_1$

$V_2^2 \times 1 \times 10^9 \times 0.8 \times C_1 = 0.6 \times 5^2 \times 0.5 \times 10^9 \times C_1$

$V_2 = ((0.6 \times 5^2 \times 0.5 \times 10^9)/(1 \times 10^9 \times 0.8))^{1/2} = 3.06\ V$

1.8.4 $Power_{new} = 1 \times C_{old} \times V^2_{old}/(2^{-1/4})^2 \times clock\ rate \times 2^{1/2} = Power_{old}$. Thus, power scales by 1.

1.8.5 $1/2^{-1/2} = 2^{1/2}$

1.8.6 Voltage = $1.1 \times 1/2^{-1/4} = 0.92$ V. Clock rate = $2.667 \times 2^{1/2} = 3.771$ GHz

تمرین 1.9

1.9.1

a.	1/49 × 100 = 2%
b.	45/120 × 100 = 37.5%

1.9.2

a.	I_{leak} = 1/3.3 = 0.3
b.	I_{leak} = 45/1.1 = 40.9

1.9.3

a.	$Power_{st}/Power_{dyn}$ = 1/49 = 0.02
b.	$Power_{st}/Power_{dyn}$ = 45/57 = 0.6

1.9.4 $Power_{st}/Power_{dyn} = 0.6 \Rightarrow Power_{st} = 0.6 \times Power_{dyn}$

a.	$Power_{st} = 0.6 \times 40\ W = 24\ W$
b.	$Power_{st} = 0.6 \times 30\ W = 18\ W$

1.9.5

a.	$I_{lk} = 24/0.8 = 30\ A$
b.	$I_{lk} = 18/0.8 = 22.5\ A$

1.9.6

	$Power_{st}$ at 1.0 V	I_{lk} at 1.0 V	$Power_{st}$ at 1.2 V	I_{lk} at 1.2 V	Larger
a.	119 W	119 A	136 W	113.3 A	I_{lk} at 1.0 V
b.	93.5 W	93.5 A	110.5 W	92.1 A	I_{lk} at 1.0 V

1.10 تمرین

1.10.1

a.	**Processors**	**Instructions per processor**	**Total instructions**
	1	4096	4096
	2	2048	4096
	4	1024	4096
	8	512	4096

b.	**Processors**	**Instructions per processor**	**Total instructions**
	1	4096	4096
	2	2278	4556
	4	1464	5856
	8	1132	9056

1.10.2

a.	**Processors**	**Execution time (μs)**
	1	4.096
	2	2.048
	4	1.024
	8	0.512

b.	**Processors**	**Execution time (μs)**
	1	4.096
	2	3.203
	4	3.164
	8	3.582

1.10.3

a.	Processors	Execution time (μs)
	1	5.376
	2	2.688
	4	1.344
	8	0.672

b.	Processors	Execution time (μs)
	1	5.376
	2	3.878
	4	3.564
	8	3.882

1.10.4

a.	Cores	Execution time (s) @ 3 GHz
	1	4.00
	2	2.17
	4	1.25
	8	0.75

b.	Cores	Execution time (s) @ 3 GHz
	1	4.00
	2	2.00
	4	1.00
	8	0.50

1.10.5

a.	Cores	Power (W) per core @ 3 GHz	Power (W) per core @ 500 MHz	Power (W) @ 3 GHz	Power (W) @ 500 MHz
	1	15	0.625	15	0.625
	2	15	0.625	30	1.25
	4	15	0.625	60	2.5
	8	15	0.625	120	5

b.	Cores	Power (W) per core @ 3 GHz	Power (W) per core @ 500 MHz	Power (W) @ 3 GHz	Power (W) @ 500 MHz
	1	15	0.625	15	0.625
	2	15	0.625	30	1.25
	4	15	0.625	60	2.5
	8	15	0.625	120	5

1.10.6

a.	Processors	Energy (J) @ 3 GHz	Energy (J) @ 500 MHz
	1	60	15
	2	65	16.25
	4	75	18.75
	8	90	22.5
b.	**Processors**	**Energy (J) @ 3 GHz**	**Energy (J) @ 500 MHz**
	1	60	15
	2	60	15
	4	60	15
	8	60	15

تمرین 1.11

1.11.1 Wafer area = $\pi \times (d/2)^2$

a.	Wafer area = $\pi \times 7.5^2 = 176.7$ cm^2
b.	Wafer area = $\pi \times 12.5^2 = 490.9$ cm^2

Die area = wafer area/dies per wafer

a.	Die area = 176.7/90 = 1.96 cm^2
b.	Die area = 490.9/140 = 3.51 cm^2

Yield = $1/(1 + (\text{defect per area} \times \text{die area})/2)^2$

a.	Yield = 0.97
b.	Yield = 0.92

1.11.2 Cost per die = cost per wafer/(dies per wafer × yield)

a.	Cost per die = 0.12
b.	Cost per die = 0.16

1.11.3

a.	Dies per wafer = 1.1 × 90 = 99 Defects per area = 1.15 × 0.018 = 0.021 defects/cm^2 Die area = wafer area/Dies per wafer = 176.7/99 = 1.78 cm^2 Yield = 0.97
b.	Dies per wafer = 1.1 × 140 = 154 Defects per area = 1.15 × 0.024 = 0.028 defects/cm^2 Die area = wafer area/Dies per wafer = 490.9/154 = 3.19 cm^2 Yield = 0.93

1.11.4 Yield = $1/(1 + (\text{defect per area} \times \text{die area})/2)^2$

Then defect per area = (2/die area)($y^{-1/2} - 1$) Replacing values for T1 and T2 we get

T1: defects per area = 0.00085 defects/mm^2 = 0.085 defects/cm^2
T2: defects per area = 0.00060 defects/mm^2 = 0.060 defects/cm^2
T3: defects per area = 0.00043 defects/mm^2 = 0.043 defects/cm^2
T4: defects per area = 0.00026 defects/mm^2 = 0.026 defects/cm^2

1.11.5 no solution provided

تمرین 1.12

1.12.1 CPI = clock rate × CPU time/instr.

count clock rate = 1/cycle time = 3 GHz

a.	CPI(pearl) = $3 \times 10^9 \times 500/2118 \times 10^9 = 0.7$
b.	CPI(mcf) = $3 \times 10^9 \times 1200/336 \times 10^9 = 10.7$

1.12.2 SPECratio = ref. time/execution time.

a.	SPECratio(pearl) = 9770/500 = 19.54
b.	SPECratio(mcf) = 9120/1200 = 7.6

1.12.3

$(19.54 \times 7.6)^{1/2} = 12.19$

1.12.4 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%.

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

CPU time(after) = 1.1 × No. instr. × 1.05 × CPI/clock rate
CPU times(after)/CPU time(before) = 1.1 × 1.05 = 1.155. Thus, CPU time is increased by 15.5%

1.12.6 SPECratio = reference time/CPU time

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

تمرین 1.13

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

a.	CPI = $450 \times 4 \times 10^9/(0.85 \times 2118 \times 10^9) = 0.99$
b.	CPI = $1150 \times 4 \times 10^9/(0.85 \times 336 \times 10^9) = 16.10$

1.13.2 Clock rate ratio = 4 GHz/3 GHz = 1.33.

a.	CPI @ 4 GHz = 0.99, CPI @ 3 GHz = 0.7, ratio = 1.41
b.	CPI @ 4 GHz = 16.1, CPI @ 3 GHz = 10.7, ratio = 1.50

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

1.13.3

a.	450/500 = 0.90. CPU time reduction: 10%.
b.	1150/1200 = 0.958. CPU time reduction: 4.2%.

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

a.	No. instr. = $820 \times 0.9 \times 4 \times 10^9/0.96 = 3075 \times 10^9$
b.	No. instr. = $580 \times 0.9 \times 4 \times 10^9/2.94 = 710 \times 10^9$

1.13.5 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.

1.13.6 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.

تمرین 1.14

1.14.1 No. instr. = 10^6

$T_{cpu}(P1) = 10^6 \times 1.25/4 \times 10^9 = 0.315 \times 10^{-3}$ s

$T_{cpu}(P2) = 10^6 \times 0.75/3 \times 10^9 = 0.25 \times 10^{-3}$ s

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

1.14.2

P1: 10^6 instructions, $T_{cpu}(P1) = 0.315 \times 10^{-3}$ s

P2: $T_{cpu}(P2) = N \times 0.75/3 \times 10^9$ then $N = 1.26 \times 10^6$

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

MIPS(P1) = $4 \times 10^9 \times 10^{-6}/1.25 = 3200$

MIPS(P2) = $3 \times 10^9 \times 10^{-6}/0.75 = 4000$

MIPS(P1) < MIPS(P2), performance(P1) < performance(P2) in this case (from 1.14.1)

1.14.4

a.	FP op = $10^6 \times 0.4 = 4 \times 10^5$, clock $\text{cyles}_{\text{fp}} = \text{CPI} \times \text{No. FP instr.} = 4 \times 10^5$ $T_{fp} = 4 \times 10^5 \times 0.33 \times 10^{-9} = 1.32 \times 10^{-4}$ then MFLOPS = 3.03×10^3

b.	FP op = $3 \times 10^6 \times 0.4 = 1.2 \times 10^6$, clock cyles$_{fp}$ = CPI × No. FP instr. = $0.70 \times 1.2 \times 10^6$ $T_{fp} = 0.84 \times 10^6 \times 0.33 \times 10^{-9} = 2.77 \times 10^{-4}$ then MFLOPS = 4.33×10^3

1.14.5 CPU clock cycles = FP cycles + CPI(L/S) × No. instr. (L/S) + CPI(Branch) × No. instr. (Branch)

a.	5×10^5 L/S instr., 4×10^5 FP instr. and 10^5 Branch instr. CPU clock cycles = $4 \times 10^5 + 0.75 \times 5 \times 10^5 + 1.5 \times 10^5 = 9.25 \times 10^5$ $T_{cpu} = 9.25 \times 10^5 \times 0.33 \times 10^{-9} = 3.05 \times 10^{-4}$ MIPS = $10^6/(3.05 \times 10^{-4} \times 10^6) = 3.2 \times 10^3$
b.	1.2×10^6 L/S instr., 1.2×10^6 FP instr. and 0.6×10^6 Branch instr. CPU clock cycles = $0.84 \times 10^6 + 1.25 \times 1.2 \times 10^6 + 1.25 \times 0.6 \times 10^6 = 3.09 \times 10^6$ $T_{cpu} = 3.09 \times 10^6 \times 0.33 \times 10^{-9} = 1.01 \times 10^{-3}$ MIPS = $3 \times 10^6/(1.01 \times 10^{-3} \times 10^6) = 2.97 \times 10^3$

1.14.6

a.	performance = $1/T_{cpu} = 3.2 \times 10^3$
b.	performance = $1/T_{cpu} = 9.9 \times 10^2$

The second program has the higher performance and the higher MFLOPS figure, but the first program has the higher MIPS figure.

تمرین 1.15

1.15.1

a.	$T_{fp} = 35 \times 0.8 = 28$ s, $T_{p1} = 28 + 85 + 50 + 30 = 193$ s. Reduction: 3.5%
b.	$T_{fp} = 50 \times 0.8 = 40$ s, $T_{p4} = 40 + 80 + 50 + 30 = 200$ s. Reduction: 4.7%

1.15.2

a.	$T_{p1} = 200 \times 0.8 = 160$ s, $T_{fp} + T_{l/s} + T_{branch} = 115$ s, $T_{int} = 45$ s. Reduction time INT: 47%
b.	$T_{p4} = 210 \times 0.8 = 168$ s, $T_{fp} + T_{l/s} + T_{branch} = 130$ s, $T_{int} = 38$ s. Reduction time INT: 52.4%

1.15.3

a.	$T_{p1} = 200 \times 0.8 = 160$ s, $T_{fp} + T_{int} + T_{l/s} = 170$ s. NO
b.	$T_{p4} = 210 \times 0.8 = 168$ s, $T_{fp} + T_{int} + T_{l/s} = 180$ s. NO

1.15.4

Clock cyles = CPI_{fp} × No. FP instr. + CPI_{int} × No. INT instr. + $CPI_{l/s}$ × No. L/S instr. + CPI_{branch} × No. branch instr.

T_{cpu} = clock cycles/clock rate = clock cycles/2×10^9

a.	1 processor: clock cycles = 8192; T_{cpu} = 4.096 s

b.	8 processors: clock cycles = 1024; T_{cpu} = 0.512 s

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

$CPI_{improved\ fp}$ × No. FP instr. + CPI_{int} × No. INT instr. + $CPI_{l/s}$ × No. L/S instr. + CPI_{branch} × No. branch instr. = clock cycles/2

$CPI_{improved\ fp}$ = (clock cycles/2 − (CPI_{int} × No. INT instr. + $CPI_{l/s}$ × No. L/S instr. + CPI_{branch} × No. branch instr.))/No. FP instr.

a.	1 processor: $CPI_{improved\ fp}$ = (4096 – 7632)/560 < 0 ==> not possible
b.	8 processors: $CPI_{improved\ fp}$ = (512 – 944)/80 < 0 ==> not possible

1.15.5 Using the clock cycle data from 1.15.4:

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

CPI_{fp} × No. FP instr. + CPI_{int} × No. INT instr. + $CPI_{improved\ l/s}$ × No. L/S instr. + CPI_{branch} × No. branch instr. = clock cycles/2

$CPI_{improved\ l/s}$ = (clock cycles/2 − (CPI_{fp} × No. FP instr. + CPI_{int} × No. INT instr. + CPI_{branch} × No. branch instr.))/No. L/S instr

a.	1 processor: $CPI_{improved\ l/s}$ = (4096 – 3072)/1280 = 0.8
b.	8 processors: $CPI_{improved\ l/s}$ = (512 – 384)/160 = 0.8

1.15.6

Clock cyles = CPI_{fp} × No. FP instr. + CPI_{int} × No. INT instr. + $CPI_{l/s}$ × No. L/S instr. + CPI_{branch} × No. branch instr.

T_{cpu} = clock cycles/clock rate = clock cycles/2 × 10^9

CPI_{int} = 0.6 × 1 = 0.6; CPI_{fp} = 0.6 × 1 = 0.6; $CPI_{l/s}$ = 0.7 × 4 = 2.8; CPI_{branch} = 0.7 × 2 = 1.4

a.	1 processor: T_{cpu}(before improv.) = 4.096 s; T_{cpu}(after improv.) = 2.739 s
b.	8 processors: T_{cpu}(before improv.) = 0.512 s; T_{cpu}(after improv.) = 0.342 s

تمرین 1.16

1.16.1 Without reduction in any routine:

a.	total time 2 proc = 185 ns
b.	total time 16 proc = 34 ns

Reducing time in routines A, C and E:

a.	2 proc: T(A) = 17 ns, T(C) = 8.5 ns, T(E) = 4.1 ns, total time = 179.6 ns ==> reduction = 2.9%

b.	16 proc: T(A) = 3.4 ns, T(C) = 1.7 ns, T(E) = 1.7 ns, total time = 32.8 ns ==> reduction = 3.5%

1.16.2

a.	2 proc: T(B) = 72 ns, total time = 177 ns ==> reduction = 4.3%
b.	16 proc: T(B) = 12.6 ns, total time = 32.6 ns ==> reduction = 4.1%

1.16.3

a.	2 proc: T(D) = 63 ns, total time = 178 ns ==> reduction = 3.7%
b.	16 proc: T(D) = 10.8 ns, total time = 32.8 ns ==> reduction = 3.5%

1.16.4

# Processors	Computing time	Computing time ratio	Routing time ratio
2	176		
4	96	0.55	1.18
8	49	0.51	1.31
16	30	0.61	1.29
32	14	0.47	1.05
64	6.5	0.46	1.13

1.16.5 Geometric mean of computing time ratios = 0.52. Multiply this by the computing time for a 64-processor system gives a computing time for a 128- processor system of 3.4 ms.

Geometric mean of routing time ratios = 1.19. Multiply this by the routing time for a 64-processor system gives a routing time for a 128-processor system of 30.9 ms.

1.16.6 Computing time = 176/0.52 = 338 ms. Routing time = 0, since no com-

2

2.1 تمرین

2.1.1

a.	add f, g, h
	add f, f, i
	add f, f, j

b.	addi f, h, 5 addi f, f, g

2.1.2

a.	3
b.	2

2.1.3

a.	14
b.	10

2.1.4

a.	f = g + h
b.	f = g + h

2.1.5

a.	5
b.	5

2.2 تمرین

2.2.1

a.	add f, f, f add f, f, i
b.	addi f, j, 2 add f, f, g

2.2.2

a.	2
b.	2

2.2.3

a.	6
b.	5

2.2.4

a.	`f += h;`
b.	`f = 1–f;`

2.2.5

a.	4
b.	0

2.3 تمرین

2.3.1

a.	`add f, f, g`
	`add f, f, h`
	`add f, f, i`
	`add f, f, j`
	`addi f, f, 2`

b.	`addi f, f, 5` `sub f, g, f`

2.3.2

a.	5
b.	2

2.3.3

a.	17
b.	-4

2.3.4

a.	`f = h – g;`
b.	`f = g – f – 1;`

2.3.5

a.	1
b.	0

2.4 تمرین

2.4.1

a.	`lw $s0, 16($s7)`
	`add $s0, $s0, $s1`
	`add $s0, $s0, $s2`

b.	`lw $t0, 16($s7)`
	`lw $s0, 0($t0)`
	`sub $s0, $s1, $s0`

2.4.2

a.	3
b.	3

2.4.3

a.	4
b.	4

2.4.4

a.	`f += g + h + i + j;`
b.	`f = A[1];`

2.4.5

a.	no change
b.	no change

2.4.6

a.	5 as written, 5 minimally
b.	2 as written, 2 minimally

2.5 تمرین

2.5.1

a.	Address Data 12 1 8 6 4 4 0 2	temp = Array[3]; Array[3] = Array[2]; Array[2] = Array[1]; Array[1] = Array[0]; Array[0] = temp;

b.	Address Data
	16 1
	12 2
	8 3
	4 4
	0 5
		temp = Array[4];
		Array[4] = Array[0];
		Array[0] = temp;
		temp = Array[3];
		Array[3] = Array[1];
		Array[1] = temp;

2.5.2

a.	Address Data 12 1 8 6 4 4 0 2	temp = Array[3]; Array[3] = Array[2]; Array[2] = Array[1]; Array[1] = Array[0]; Array[0] = temp;	lw $t0, 12($s6) lw $t1, 8($s6) sw $t1, 12($s6) lw $t1, 4($s6) sw $t1, 8($s6) lw $t1, 0($s6) sw $t1, 4($s6) sw $t0,

b.	Address Data
	16 1
	12 2
	8 3
	4 4
	0 5
		temp = Array[4];
		Array[4] = Array[0];
		Array[0] = temp;
		
		temp = Array[3];
		Array[3] = Array[1];
		Array[1] = temp;
			lw $t0,
			16($s6) lw $t1,
			0($s6) sw $t1,
			16($s6) sw $t0,
			0($s6)
			
			lw $t0,
			12($s6) lw $t1,
			4($s6) sw $t1,

2.5.3

a.	Address Data 12 1 8 6 4 4 0 2	temp = Array[3]; Array[3] = Array[2]; Array[2] = Array[1]; Array[1] = Array[0]; Array[0] = temp;	lw $t0, 12($s6) lw $t1, 8($s6) sw $t1, 12($s6) lw $t1, 4($s6) sw $t1, 8($s6) lw $t1, 0($s6) sw $t1, 4($s6) sw $t0,	8 mips instructions, +1 mips inst. for every non-zero offset lw/sw pair (11 mips inst.)

b.	Address Data
	16 1
	12 2
	8 3
	4 4
	0 5
		temp = Array[4];
		Array[4] = Array[0];
		Array[0] = temp;
		
		temp = Array[3];
		Array[3] = Array[1];
		Array[1] = temp;
			lw $t0,
			16($s6) lw $t1,
			0($s6) sw $t1,
			16($s6) sw $t0,
			0($s6)
			
			lw $t0,
			12($s6) lw $t1,
			4($s6) sw $t1,
				8 mips instructions, +1 mips inst. for every non-zero offset lw/sw pair (11 mips inst.)

2.5.4

a.	305419896
b.	3199070221

2.5.5

	Little-Endian		Big-Endian	
a.	Address 12 8 4 0	Data 12 34 56 78	Address 12 8 4 0	Data 78 56 34 12
b.	Address 12 8 4 0	Data be ad f0 0d	Address 12 8 4 0	Data 0d f0 ad be

2.6 تمرین

2.6.1

a.	lw $s0, 4($s7)
	sub $s0, $s0, $s1
	add $s0, $s0, $s2

b.	add $t0, $s7, $s1
	lw $t0, 0($t0)
	add $t0, $t0, $s6
	lw $s0, 4($t0)

2.6.2

a.	3
b.	4

2.6.3

a.	4
b.	5

2.6.4

a.	f = 2i + h;
b.	f = A[g – 3];

2.6.5

a.	$s0 = 110
b.	$s0 = 300

2.6.6

a.

	Type	opcode	rs	rt	rd	immed
add $s0, $s0, $s1	R-type	0	16	17	16	
add $s0, $s3, $s2	R-type	0	19	18	16	
add $s0, $s0, $s3	R-type	0	16	19	16	

b.

	Type	opcode	rs	rt	rd	immed
addi $s6, $s6, –20	I-type	8	22	22		-20
add $s6, $s6, $s1	R-type	0	22q	17	22	
lw $s0, 8($s6)	I-type	35	22	16		8

2.7 تمرین

2.7.1

a.	-1391460350
b.	-19629

2.7.2

a.	2903506946
b.	4294947667

2.7.3

a.	AD100002
b.	FFFFB353

2.7.4

a.	01111111111111111111111111111111
b.	1111101000

2.7.5

a.	7FFFFFFF
b.	3E8

2.7.6

a.	80000001
b.	FFFFFC18

2.8 تمرین

2.8.1

a.	7FFFFFFF, no overflow
b.	80000000, overflow

2.8.2

a.	60000001, no overflow
b.	O, no overflow

2.8.3

a.	EFFFFFFF, overflow
b.	C0000000, overflow

2.8.4

a.	overflow
b.	no overflow

2.8.5

a.	no overflow
b.	no overflow

2.8.6

a.	overflow
b.	no overflow

2.9 تمرین

2.9.1

a.	overflow
b.	no overflow

2.9.2

a.	overflow
b.	no overflow

2.9.3

a.	no overflow
b.	overflow

2.9.4

a.	no overflow
b.	no overflow

2.9.5

a.	1D100002
b.	6FFFB353

2.9.6

a.	487587842
b.	1879028563

2.10 تمرین

2.10.1

a.	sw $t3, 4($s0)
b.	lw $t0, 64($t0)

2.10.2

a.	I-type
b.	I-type

2.10.3

a.	AE0B0004
b.	8D080040

2.10.4

a.	0x01004020

b.	0x8E690004

2.10.5

a.	R-type
b.	I-type

2.10.6

a.	op=0x0, rd=0x8, rs=0x8, rt=0x0, funct=0x0
b.	op=0x23, rs=0x13, rt=0x9, imm=0x4

Solution 2.11

2.11.1

a.	$1010\ 1110\ 0000\ 1011\ 1111\ 1111\ 1111\ 1100_{two}$
b.	$1000\ 1101\ 0000\ 1000\ 1111\ 1111\ 1100\ 0000_{two}$

2.11.2

a.	2920022012
b.	2366177216

2.11.3

a.	sw $t3, –4($s0)
b.	lw $t0, –64($t0)

2.11.4

a.	R-type
b.	I-type

2.11.5

a.	add $v1, $at, $v0
b.	sw $a1, 4($s0)

2.11.6

a.	0x00221820
b.	0xAD450004

تمرین 2.12

2.12.1

	Type	opcode	rs	rt	rd	shamt	funct	
a.	R-type	6	3	3	3	5	6	total bits = 26
b.	R-type	6	5	5	5	5	6	total bits = 32

2.12.2

	Type	opcode	rs	rt	immed	
a.	I-type	6	3	3	16	total bits = 28
b.	I-type	6	5	5	10	total bits = 26

2.12.3

a.	less registers → less bits per instruction → could reduce code size less registers → more register spills → more instructions
b.	smaller constants → more lui instructions → could increase code size smaller constants → smaller opcodes → smaller code size

2.12.4

a.	17367056
b.	2366177298

2.12.5

a.	`add $t0, $t1, $0`
b.	`lw $t1, 12($t0)`

2.12.6

a.	R-type, op=0×0, rt=0×9
b.	I-type, op=0×23, rt=0×8

Solution 2.13

2.13.1

a.	0x57755778
b.	0xFEFFFEDE

2.13.2

a.	0x55555550
b.	0xEADFEED0

2.13.3

a.	0x0000AAAA
b.	0x0000BFCD

2.13.4

a.	0x00015B5A
b.	0x00000000

2.13.5

a.	0x5b5a0000
b.	0x000000f0

2.13.6

a.	0xEFEFFFFF
b.	0x000000F0

2.14 تمرین

2.14.1

a.	add $t1, $t0, $0 srl $t1, $t1, 5 andi $t1, $t1, 0x0001ffff
b.	add $t1, $t0, $0 sll $t1, $t1, 10 andi $t1, $t1, 0xffff8000

2.14.2

a.	add $t1, $t0, $0 andi $t1, $t1, 0x0000000f
b.	add $t1, $t0, $0 srl $t1, $t1, 14 andi $t1, $t1, 0x0003c000

2.14.3

a.	add $t1, $t0, $0 srl $t1, $t1, 28
b.	add $t1, $t0, $0 srl $t1, $t1, 14 andi $t1, $t1, 0x0001c000

2.14.4

a.	add $t2, $t0, $0 srl $t2, $t2, 11 and $t2, $t2, 0x0000003f and $t1, $t1, 0xffffffc0 or $t1, $t1, $t2

b.	add $t2, $t0, $0 sll $t2, $t2, 3 and $t2, $t2, 0x000fc000 and $t1, $t1, 0xfff03fff ori $t1, $t1, $t2

2.14.5

a.	add $t2, $t0, $0 and $t2, $t2, 0x0000001f and $t1, $t1, 0xffffffe0 ori $t1, $t1, $t2
b.	add $t2, $t0, $0 sll $t2, $t2, 14 and $t2, $t2, 0x0007c000 and $t1, $t1, 0xfff83fff ori $t1, $t1, $t2

2.14.6

a.	add $t2, $t0, $0 srl $t2, $t2, 29 and $t2, $t2, 0x00000003 and $t1, $t1, 0xfffffffc or $t1, $t1, $t2
b.	add $t2, $t0, $0 srl $t2, $t2, 15 and $t2, $t2, 0x0000c000 and $t1, $t1, 0xffff3fff ori $t1, $t1, $t2

2.15 تمرین

2.15.1

a.	0x0000a581
b.	0x00ff5a66

2.15.2

a.	nor $t1, $t2, $t2 and $t1, $t1, $t3
b.	xor $t1, $t2, $t3 nor $t1, $t1, $t1

2.15.3

a.	nor $t1, $t2, $t2 and $t1, $t1, $t3	000000 01010 01010 01001 00000 100111 000000 01001 01011 01001 00000 100100
b.	xor $t1, $t2, $t3 nor $t1, $t1, $t1	000000 01010 01011 01001 00000 100110 000000 01001 01001 01001 00000 100111

2.15.4

a.	0x00000220

b.	0x00001234

2.15.5 Assuming $t1 = A, $t2 = B, $s1 = base of Array C

a.	lw $t3, 0($s1) and $t1, $t2, $t3
b.	beq $t1, $0, ELSE add $t1, $t2, $0 beq $0, $0, END ELSE: lw $t2, 0($s1) END:

2.15.6

a.	lw $t3, 0($s1) and $t1, $t2, $t3	100011 10001 01011 0000000000000000 000000 01010 01011 01001 00000 100100
b.	beq $t1, $0, ELSE add $t1, $t2, $0 beq $0, $0, END ELSE: lw $t2, 0($s1) END:	000100 01001 00000 0000000000000010 000000 01010 00000 01001 00000 100000 000100 00000 00000 0000000000000001 100011 10001 01010 0000000000000000

2.16 تمرین

2.16.1

a.	$t2 = 1
b.	$t2 = 1

2.16.2

a.	all, 0x8000 to 0x7FFFF
b.	0x8000 to 0xFFFE

2.16.3

a.	jump—no, beq—no
b.	jump—no, beq—no

2.16.4

a.	$t2 = 2
b.	$t2 = 2

2.16.5

a.	$t2 = 0
b.	$t2 = 1

2.16.6

a.	jump—yes, beq—no
b.	jump—yes, beq—yes

تمرین 2.17

2.17.1 The answer is really the same for all. All of these instructions are either supported by an existing instruction, or sequence of existing instructions. Looking for an answer along the lines of, “these instructions are not common, and we are only making the common case fast”.

2.17.2

a.	could be either R-type of I-type
b.	R-type

2.17.3

a.	ABS: sub $t2,$zero,$t3 # t2 = – t3 ble $t3,$zero,done # if t3 < 0, result is t2 add $t2,$t3,$zero # if t3 > 0, result is t3 DONE:
b.	slt $t1, $t3, $t2

2.17.4

a.	20
b.	200

2.17.5

a.	i = 10;
	do {
	B += 2;
	i = i – 1;
	} while (i > 0)

b.	i = 10;
	do {
	temp = 10;
	do {
	B += 2;
	temp = temp – 1;
	} while (temp > 0)
	i = i – 1;
	} while (i > 0)

2.17.6

a.	5 × N + 3
b.	33 × N

2.19 تمرین

2.19.1

a.

```
compare:
        addi  $sp,  $sp,  –4
        sw    $ra,  0($sp)

        add   $s0,  $a0,  $0
        add   $s1,  $a1,  $0

        jal   sub
        addi  $t1,  $0,  1
        beq   $v0,  $0,  exit
        slt   $t2,  $0,  $v0
        bne   $t2,  $0, exit
        addi  $t1,  $0,  $0
exit:
        add   $v0,  $t1,  $0
        lw    $ra,  0($sp)
        addi  $sp,  $sp,  4
        jr    $ra

sub:
        sub   $v0,  $a0,  $a1
        jr    $ra
```

b.

```
fib_iter:
        addi  $sp,  $sp,  –16
        sw    $ra,  12($sp)
        sw    $s0,  8($sp)
        sw    $s1,  4($sp)
        sw    $s2,  0($sp)

        add   $s0,  $a0,  $0
        add   $s1,  $a1,  $0
        add   $s2,  $a2,  $0

        add   $v0,  $s1,  $0,
        bne   $s2,  $0, exit

        add   $a0,  $s0,  $s1
        add   $a1,  $s0,  $0
        add   $a2,  $s2,  –1
        jal   fib_iter
exit:
        lw    $s2,  0($sp)
        lw    $s1,  4($sp)
        lw    $s0,  8($sp)
        lw    $ra,  12($sp)
        addi  $sp,  $sp,  16
        jr    $ra
```

2.19.2

a.

```
compare:
        addi  $sp,  $sp,  –4
        sw    $ra,  0($sp)

        sub   $t0,  $a0,  $a1
        addi  $t1,  $0,  1
        beq   $t0,  $0, exit
        slt   $t2,  $0,  $t0
        bne   $t2,  $0, exit
        addi  $t1,  $0,  $0
exit:
        add   $v0,  $t1,  $0
        lw    $ra,  0($sp)
        addi  $sp,  $sp,  4
        jr    $ra
```

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

2.19.3

a.	after calling function compare: old $sp => 0x7ffffffc ??? $sp => –4 contents of register $ra after calling function sub: old $sp => 0x7ffffffc ??? –4 contents of register $ra $sp => –8 contents of register $ra #return to compare
b.	after calling function fib_iter: old $sp => 0x7ffffffc ??? –4 contents of register $ra –8 contents of register $s0 –12 contents of register $s1 $sp => –16 contents of register $s2

2.19.4

a.	f: $sp,$sp,–8
	sw $ra,4($sp)
	sw $s0,0($sp)
	move $s0,$a2
	jal func
	move $a0,$v0
	move $a1,$s0
	jal func
	lw $ra,4($sp)
	lw $s0,0($sp)
	addi $sp,$sp,8
	jr $ra

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

2.19.5

a.	We can use the tail-call optimization for the second call to func, but then we must restore $ra and $sp before that call. We save only one instruction (jr $ra).
b.	We can NOT use the tail call optimization here, because the value returned from f is not equal to the value returned by the last call to func.

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