ECE 586

Homework #1

Problem 1:

Chip	Die Size (mm^2)	Estimated defect rate (per cm^2)	N	Manufacturing size (nm)	Transistors (billion)	Cores
BlueDragon	180	0.03	12	10	7.5	4
RedDragon	120	0.04	14	7	7.5	4
Phoenix[8]	200	0.04	14	7	12	8

1.2 They will sell a range of chips from that factory, and they need to decide how much capacity to dedicate to each chip. Imagine that they will sell two chips. Phoenix is a completely new architecture designed with 7 nm technology in mind, whereas RedDragon is the same architecture as their 10 nm BlueDragon. Imagine that RedDragon will make a profit of $15 per defect-free chip. Phoenix will make a profit of $30 per defect-free chip. Each wafer has a 450 mm diameter.

a. How much profit do you make on each wafer of Phoenix chips?

b. How much profit do you make on each wafer of RedDragon chips?

c. If your demand is 50,000 RedDragon chips per month and 25,000 Phoenix chips per month, and your facility can fabricate 70 wafers a month, how many wafers should you make of each chip?

Problem 2:

1.6 General-purpose processes are optimized for general-purpose computing. That is, they are optimized for behavior that is generally found across a large number of applications. However, once the domain is restricted somewhat, the behavior that is found across a large number of the target applications may be different from general-purpose applications. One such application is deep learning or neural networks. Deep learning can be applied to many different applications, but the fundamental building block of inference—using the learned information to make decisions—is the same across them all. Inference operations are largely parallel, so they are currently performed on graphics processing units, which are specialized more toward this type of computation, and not to inference in particular. In a quest for more performance per watt, Google has created a custom chip using tensor processing units to accelerate inference operations in deep learning.[1] This approach can be used for speech recognition and image recognition, for example. This problem explores the trade-offs between this process, a general-purpose processor (Haswell

E5-2699 v3) and a GPU (NVIDIA K80), in terms of performance and cooling. If heat is not removed from the computer efficiently, the fans will blow hot air back onto the computer, not cold air. Note: The differences are more than processor—on-chip memory and DRAM also come into play. Therefore statistics are at a system level, not a chip level.

a. If Google's data center spends 70% of its time on workload A and 30% of its time on workload B when running GPUs, what is the speedup of the TPU system over the GPU system?

b. If Google's data center spends 70% of its time on workload A and 30% of its time on workload B when running GPUs, what percentage of Max IPS does it achieve for each of the three systems?

c. Building on (b), assuming that the power scales linearly from idle to busy power as IPS grows from 0% to 100%, what is the performance per watt of the TPU system over the GPU system?

d. If another data center spends 40% of its time on workload A, 10% of its time on workload B, and 50% of its time on workload C, what are the speedups of the GPU and TPU systems over the general-purpose system?

e. A cooling door for a rack costs $4000 and dissipates 14 kW (into the room; additional cost is required to get it out of the room). How many Haswell-, NVIDIA-, or Tensor-based servers can you cool with one cooling door, assuming TDP in Figures 1.27 and 1.28?

f. Typical server farms can dissipate a maximum of 200 W per square foot. Given that a server rack requires 11 square feet (including front and back clearance), how many servers from part (e) can be placed on a single rack, and how many cooling doors are required?

System	Chip	TDP	Idle power	Busy power
General-purpose	Haswell E5-2699 v3	504 W	159 W	455 W
Graphics processor	NVIDIA K80	1838 W	357 W	991 W
Custom ASIC	TPU	861 W	290 W	384 W

System	Chip	Throughput			% Max IPS		
		A	B	C	A	B	C
General-purpose	Haswell E5-2699 v3	5482	13,194	12,000	42%	100%	90%
Graphics processor	NVIDIA K80	13,461	36,465	15,000	37%	100%	40%
Custom ASIC	TPU	225,000	280,000	2000	80%	100%	1%

Problem 3.

1.11 In a server farm such as that used by Amazon or eBay, a single failure does not cause the entire system to crash. Instead, it will reduce the number of requests that can be satisfied at any one time.

a. If a company has 10,000 computers, each with an MTTF of 35 days, and it experiences catastrophic failure only if 1/3 of the computers fail, what is the MTTF for the system?

b. If it costs an extra $1000, per computer, to double the MTTF, would this be a good business decision? Show your work.

c. Figure 1.3 shows, on average, the cost of downtimes, assuming that the cost is equal at all times of the year. For retailers, however, the Christmas season is the most profitable (and therefore the most costly time to lose sales). If a catalog sales center has twice as much traffic in the fourth quarter as every other quarter, what is the average cost of downtime per hour during the fourth quarter and the rest of the year?

Application	Cost of downtime per hour	Annual losses with downtime of		
		1% (87.6 h/year)	0.5% (43.8 h/year)	0.1% (8.8 h/year)
Brokerage service	$4,000,000	$350,400,000	$175,200,000	$35,000,000
Energy	$1,750,000	$153,300,000	$76,700,000	$15,300,000
Telecom	$1,250,000	$109,500,000	$54,800,000	$11,000,000
Manufacturing	$1,000,000	$87,600,000	$43,800,000	$8,800,000
Retail	$650,000	$56,900,000	$28,500,000	$5,700,000
Health care	$400,000	$35,000,000	$17,500,000	$3,500,000
Media	$50,000	$4,400,000	$2,200,000	$400,000

Problem 4.

1.14 When making changes to optimize part of a processor, it is often the case that speeding up one type of instruction comes at the cost of slowing down something else. For example, if we put in a complicated fast floating-point unit, that takes space, and something might have to be moved farther away from the middle to accommodate it, adding an extra cycle in delay to reach that unit. The basic Amdahl's Law equation does not take into account this trade-off.

a. If the new fast floating-point unit speeds up floating-point operations by, on average, 2x, and floating-point operations take 20% of the original program's execution time, what is the overall speedup (ignoring the penalty to any other instructions)?

b. Now assume that speeding up the floating-point unit slowed down data cache accesses, resulting in a 1.5x slowdown (or 2/3 speedup). Data cache accesses consume 10% of the execution time. What is the overall speedup now?

c. After implementing the new floating-point operations, what percentage of execution time is spent on floating-point operations? What percentage is spent on data cache accesses?

Alan Palayil
A20447935

ECE 586
Homework #1

Problem 1:

1.2] a) Phoenix chips:

$$\text{Dies per wafer} = \frac{\pi \times \left(\frac{15}{2}\right)^2}{2} - \frac{\pi \times 15}{\sqrt{(2 \times 2)}}$$

$$= \frac{1589.625}{2} - \frac{141.3}{2}$$

$$= \frac{1448.325}{2}$$

$$= 724.1625 \approx 724 //$$

$$\text{Yield} = \frac{1}{(1 + (0.04 \times 2))^{14}}$$

$$= 0.34 //$$

$$\text{Profit} = 724 \times 0.34 \times 30$$

$$= \$7384.80 //$$

b) Red dragon chips:

$$\text{Dies per wafer} = \frac{\pi \times \left(\frac{15}{2}\right)^2}{2} - \frac{\pi \times 15}{\sqrt{(2 \times 1.2)}}$$

$$= \frac{1589.625}{2} - \frac{141.3}{1.55}$$

$$= 1325 - 91.25$$

$$= 1233.75 \approx 1234 //$$

$$\text{Yield} = \frac{1}{(1 + (0.04 \times 1.2))^{14}}$$

$$= 0.519 //$$

$$\text{Profit} = 1234 \times 0.519 \times 15$$

$$= \$9601.71$$

c) Phoenix chips: $\frac{25,000}{724^{(1/4)}}$

$= 34.5$ wafers needed //

Red Dragon chips: $\frac{50,000}{1234^{(1/4)}}$

$= 40.5$ wafers needed //

∴ The most lucrative split is 40 Red Dragon wafers and 30 Phoenix wafers.

Problem 2:

1.6] a) Performance gain obtained by improving some portion of the computer speed

$= \frac{\text{Execution Time for the entire task without using the enhancement}}{\text{Execution Time for entire task using the enhancement}}$

Speed up of computer A over B $= \frac{\text{Execution Time of B}}{\text{Execution Time of A.}}$ — ①

Using Table 2:

Speed up of TPU over GPU for workload A $= \frac{225000}{13461} = 16.7$ — ②

Speed up of TPU over GPU for workload B $= \frac{280000}{36465} = 7.7$ — ③

From ①, ②, & ③, considering 70% of a GPUs Execution Time is for workload A, 30% for B.

- $0.7 \times$ Ex Time (A+B) of GPU = Ex Time (A) of TPU $\times 16.7$ (for workload A)
- $0.3 \times$ Ex Time (A+B) of GPU = Ex Time (B) of TPU $\times 7.7$ (for workload B)

∴ Ex Time (A+B) of TPU $= \frac{0.7 \times \text{Ex. Time of GPU}}{16.7} + \frac{0.3 \times \text{Ex. Time of GPU}}{7.7}$

$= \left(\frac{0.7}{16.7} + \frac{0.3}{7.7}\right)$ Ex. Time (A+B) GPU.

∴ $\frac{\text{Ex. Time (A+B) GPU}}{\text{Ex Time (A+B) of TPU}} =$ speed up of TPU over GPU — ④

∴ speed up of TPU over GPU $= \frac{1}{\left(\frac{0.7}{16.7} + \frac{0.3}{7.7}\right)} = 12.37$ //

b) Data center spends 70% of its time in workload A & 30% of its time in workload B.

The CPU can accomplish only 42% of its maximum IPS in workload A and 100% of its maximum IPS in workload B.

∴ Maximum IPS can be achieved = 42% × 70% + 100% × 30%

= 0.294 + 0.3

= 0.594 ≈ 59.4% //

In GPU maximum IPS in workload A & B = 37% & 100%.

∴ 37% × 70% + 100% × 30% = 0.559 ≈ 55.9 % //

In custom ASIC (TPU), maximum IPS in workload A & B = 80% & 100%

∴ 80% × 70% + 100% × 30% = 0.86 ≈ 86 % //

c) Linear dependance of power on IPS at max. IPS, processor is consuming idle power. The formula is:

Power = Idle power + [max power − idle power] × [percentage of max IPS]

∴ For the CPU its: 159W + (455W − 159W) × (0.594) = 335W //

∴ For the GPU its: 357W + (991W − 357W) × (0.559) = 711W //

∴ For the TPU its: 290W + (384W − 290W) × (0.86) = 371W //

The ratio of system 1 to system 2 performance per watt

$$= \left[\frac{\text{Percentage of max IPS}}{\text{Power}}\right] \times \left[\frac{\text{Power}}{\text{Percentage of max IPS}}\right]$$

∴ The comparison for TPU : GPU = $\frac{0.86 \times 711}{0.559 \times 371} \approx 2.95$.

Thus, the TPU delivers almost 3 times the performance per watt for this workload. //

d) For the GPU:

Speedup of GPU over general-purpose for workload A = $13461/5482 = 2.455$

Speedup of GPU over general-purpose for workload B = $36465/13194 = 2.763$

Speedup of GPU over general-purpose for workload C = $15000/12000 = 1.25$

For workload A : 0.4 × ET of general purpose = ET of GPV × 2.455
For workload B : 0.1 × ET of general purpose = ET of GPV × 2.763
For workload C : 0.5 × ET of general purpose = ET of GPV × 1.25
∴ The ET of the GPV = ET of general purpose × $\left[\frac{0.4}{2.455} + \frac{0.1}{2.763} + \frac{0.5}{1.25}\right]$

∴ The net speedup of GPV = $\frac{\text{ET of general purpose}}{\text{ET of GPV}}$

$= \frac{1}{\left[\frac{0.4}{2.455} + \frac{0.1}{2.763} + \frac{0.5}{1.25}\right]} \approx 1.666$ //

For the TPV :
Speedup of TPV over general-purpose for workload A = $225000/5482 = 44.04$
Speedup of TPV over general-purpose for workload B = $280000/13194 = 22.22$
Speedup of TPV over general-purpose for workload C = $2000/12000 = 0.166$
For workload A : 0.4 × ET of general purpose = ET of TPV × 44.04
For workload B : 0.1 × ET of general purpose = ET of TPV × 22.22
For workload C : 0.5 × ET of general purpose = ET of TPV × 0.166
∴ The ET of the TPV = ET of general purpose × $\left[\frac{0.4}{44.04} + \frac{0.1}{22.22} + \frac{0.5}{0.166}\right]$

∴ The net speedup of TPV = $\frac{\text{ET of general purpose}}{\text{ET of TPV}}$

$= \frac{1}{\left[\frac{0.4}{44.04} + \frac{0.1}{22.22} + \frac{0.5}{0.166}\right]} \approx 0.331$ //

e) The TDP is higher than the maximum power dissipation from a processor & is equal to the minimum rate of heat removal that must be exceeded by cooling.
∴ The cooling door removes 14kW of power from a rack holding multiple server boards and expansive (54K)
With a cooling door, 14kW / 504W = no. of CPUs that can be cooled = 27 //

With a cooling door, 14kW/1838W = no. of GPUs that can be cooled - 7 //
With a cooling door, 14kW/861W = no. of TPUs that can be cooled - 16 //

f) Max. power per rack in server farm = 200 W/ft² × 11 ft² = 2200 W
Max. no. of servers per rack & no. of cooling doors for this max.
CPU: 2200W/504W = 4.37 ≈ 4 servers //
GPU: 2200W/1838W = 1.19 ≈ 1 server //
TPU: 2200W/861W = 2.55 ≈ 2 servers //
∴ One cooling door is enough to dissipate 2200W //

Problem 3

1.11] a) The no. of failures needed is: $\frac{1}{3} \times 10000 = 3333.33 \approx 3333$

∴ The failures in time (FIT) = $\frac{1}{35} \times 10000 = 285.75$
≈ 285

∴ MTTF of the system = $\frac{\text{no. of failures needed}}{\text{FIT}}$

$= \frac{3333}{285} = 11.69$ days //

b) In the current system, one computer fails approximately every 5 mins. However, the amount of time before one-third of the computers have failed at the same time can be greatly extended. This is valuable as it can reduce the cost of downtime, which can be huge. There are several possible solutions to extend this time.

c) Consider the cost of downtime for first three be x & 2x for the fourth quater.
The average cost of downtime per hour is: $\frac{x+x+x+2x}{4} = 90000$

$5x = 360000$

$x = 72000$

∴ The average cost of downtime per hour during the fourth quarter is

∴ $2 \times 72000 = \$144,000$. //

Problem 4:

1.14] a) $Speedup_{enhanced} = 2$ & $Fraction_{enhanced} = 20\% \approx 0.2$

∴ Overall speedup $= \left[\frac{1}{(1 - Fraction_{enhanced}) + \left(\frac{Fraction_{enhanced}}{Speedup_{enhanced}} \right)} \right]$

$= \left[\frac{1}{(1-0.2) + \left(\frac{0.2}{2} \right)} \right]$

$= \left(\frac{1}{0.8 + 0.1} \right)$

∴ $Speedup_{overall} = 1.11$ //

b) Data cache access Speedup = 2/3.

The execution time for data cache is 10%

Data cache access Fraction = 10% ≈ 0.1

Enhanced speedup floating point = 2

Enhanced fraction floating point = 20%

∴ Overall speedup $= \frac{1}{\left(1 - fraction_{fp} - fraction_{DCA}\right) + \left(\frac{Fraction_{fp}}{Speedup_{fp}} \right) + \left(\frac{Fraction_{DCA}}{Speedup_{DCA}} \right)}$

$= \frac{1}{(1-0.2-0.1) + \left(\frac{0.2}{2} \right) + \left(\frac{0.1}{2/3} \right)}$

∴ Overall Speedup = 1.052 //

c) The percentage of execution time spent on floating point operation is:

$$= \left(\frac{0.1}{0.7+0.1+0.15}\right) \times 100$$

$$= \left(\frac{0.1}{0.95}\right) \times 100$$

$$= 10.5 //$$

The percentage of execution time spent on data cache access operation is:

$$= \left(\frac{0.15}{0.7+0.1+0.15}\right) \times 100$$

$$= \frac{0.15}{0.95} \times 100$$

$$= 15.8 //$$

ECE 586
Homework #2

Due date: 02/08/2023, Wednesday

Problem 1:

A.1 Compute the effective CPI for an implementation of an embedded RISC-V CPU using Figure A.29.

Program	Loads	Stores	Branches	Jumps	ALU operations
astar	28%	6%	18%	2%	46%
bzip	20%	7%	11%	1%	54%
gcc	17%	23%	20%	4%	36%
gobmk	21%	12%	14%	2%	50%
h264ref	33%	14%	5%	2%	45%
hmmer	28%	9%	17%	0%	46%
libquantum	16%	6%	29%	0%	48%
mcf	35%	11%	24%	1%	29%
omnetpp	23%	15%	17%	7%	31%
perlbench	25%	14%	15%	7%	39%
sjeng	19%	7%	15%	3%	56%
xalancbmk	30%	8%	27%	3%	31%

Figure A.29 RISC-V dynamic instruction mix for the SPECint2006 programs

Assume we have made the following measurements of average CPI for each of the instruction types:

Instruction	Clock cycles
All ALU operations	1.0
Loads	5.0
Stores	3.0
Branches	
Taken	5.0
Not taken	3.0
Jumps	3.0

Average the instruction frequencies of astar and gcc to obtain the instruction mix. Assume 60% of branches are “Taken”.

Problem 2:

C.2 Suppose the branch frequencies (as percentages of all instructions) are as follows:

Conditional branches	15%
Jumps and calls	1%
Taken conditional branches	60% are taken

a. We are examining a four-stage pipeline where the branch is resolved at the end of the second cycle for unconditional branches and at the end of the third cycle for conditional branches. Assuming that only the first pipe stage can always be completed independent of whether the branch is taken and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?
b. Now assume a high-performance processor in which we have a 15-deep pipeline where the branch is resolved at the end of the fifth cycle for unconditional branches and at the end of the tenth cycle for conditional branches. Assuming that only the first pipe stage can always be completed independent of whether the branch is taken and ignoring other pipeline stalls, how much faster would the machine be without any branch hazards?

Problem 3:

C.3 We begin with a computer implemented in single-cycle implementation. When the stages are split by functionality, the stages do not require exactly the same amount of time. The original machine had a clock cycle time of 7 ns. After the stages were split, the measured times were IF, 1 ns; ID, 1.5 ns; EX, 1 ns; MEM, 2 ns; and WB, 1.5 ns. The pipeline register delay is 0.1 ns.

a. What is the clock cycle time of the 5-stage pipelined machine?

b. If there is a stall every four instructions, what is the CPI of the new machine?

c. What is the speedup of the pipelined machine over the single-cycle machine?

d. If the pipelined machine had an infinite number of stages, what would its speedup be over the single-cycle machine?

Problem 4:

C.7 In this problem, we will explore how deepening the pipeline affects performance in two ways: faster clock cycle and increased stalls due to data and control hazards. Assume that the original machine is a 5-stage pipeline with a 1 ns clock cycle. The second machine is a 12-stage pipeline with a 0.6 ns clock cycle. The 5-stage pipeline experiences a stall due to a data hazard every five instructions, whereas the 12-stage pipeline experiences three stalls every eight instructions. In addition, branches constitute 20% of the instructions, and the misprediction rate for both machines is 5%.

a. What is the speedup of the 12-stage pipeline over the 5-stage pipeline, taking into account only data hazards?

b. If the branch mispredict penalty for the first machine is 2 cycles but the second machine is 5 cycles, what are the CPIs of each, taking into account the stalls due to branch mispredictions?

Alan Palayil ECE 586

A20447935 Homework #2 Due Date: 2/8/23

Problem 1

A.1 The average percentage of the following instructions assuming 60% of branches are Taken.

ALU instructions: $\frac{46+36}{2} = \frac{82}{2} = 41\%$

Load instructions: $\frac{28+17}{2} = \frac{45}{2} = 22.5\%$

Store instructions: $\frac{6+23}{2} = \frac{29}{2} = 14.5\%$

Branch instructions: $\frac{18+20}{2} = \frac{38}{2} = 19\%$

Jump instructions $= \frac{2+4}{2} = 3\%$

$\therefore$ The average CPI $= 0.41 \times 1 + 0.225 \times 5 + 0.145 \times 3 + 0.19 \times \frac{(5+3)}{2} + 0.03 \times 3$

$= 0.41 + 1.125 + 0.435 + 0.76 + 0.09$

$= 2.82$ //

Problem 2

C.2 a) Pipeline Speedup $= \frac{1}{1 + \text{Pipeline stalls}} \times \text{Pipeline depths}$ — (1)

For ideal cases where no control hazards and stalls:

$\text{Pipeline Speedup}_{ideal} = \frac{1}{1+(0)} \times 4$ — (2)

Using given data:

$\text{Pipeline stalls}_{real} = (1 \times 1\%) + (2 \times 9\%) + (1 \times 6\%)$

$= 0.25$ — (3)

Substituting the value from (3) to equation (1)

$\therefore \text{Pipeline Speed}_{real} = \frac{1}{1+0.25} \times 0.4 = 3.2$ — (4)

from equation ② & ④

$$\text{Pipeline speedup}_{\text{w/o control hazards}} = \frac{4}{3.2} = 1.25 //$$

∴ The presense of control hazards in the pipeline loses approximately 25% of the speedup you achieve w/o such hazards.

b) From the given data:

$$\text{Pipeline stalls}_{\text{real}} = (4 \times 1\%) + (9 \times 9\%) + (8 \times 6\%)$$
$$= 0.04 + 0.81 + 0.48$$
$$= 1.33$$

$$\text{Pipeline speed up}_{\text{real}} = \frac{1}{1 + 1.33} \times 4$$
$$= \frac{4}{2.33} = 1.72$$

$$\text{Pipeline speed up}_{\text{w/o control hazard}} = \frac{4}{1.72}$$
$$= 2.33 //$$

Problem 3

C.3 a) Clock cycled time of the pipelined machine = max time + delay
$$= 2 + 0.1$$
$$= 2.1 \text{ ns} //$$

b) Effective CPI of the new machine $= 1 + \frac{1}{4}$
$$= 1.25 //$$

c) Speedup of pipelined machine over the single-cycle machine is equal to

$$\frac{\text{average time per instruction of single cyle}}{\text{average time per instruction of pipelined}}$$

In single cycle processor:
CPI = 1 & Clock period = 7ns.
Pipelined processor:

CPI = 1.25 & Clock period = 2.1 ns

$$\therefore \text{Speedup} = \frac{7 \times 1}{1.25 \times 2.1} = \frac{7}{2.625} = 2.67 //$$

d) As the no. of stages approach infinity, the speedup = k, where k is the no. of stages in the machine.

Problem 4

C.7 a) For 5-stage pipeline:

$$CPI_{5\text{-stage}} = \frac{5+1}{5} = 1.2 \quad -①$$

Clock cycle time = 1 ns —②

For 12-stage pipeline:

$$CPI_{12\text{-stage}} = \frac{3+8}{8} = 1.375 \quad -③$$

Clock cycle time = 0.6 ns —④

Execution time$_{old}$ = IC × CPI × Cycle Time
= IC × 1.2 × 1 —⑤ (From ① & ②)

Execution time$_{new}$ = IC × CPI × Cycle Time
= IC × 1.375 × 0.6 —⑥ (From ③ & ④)

Using ⑤ & ⑥

$$\text{Speedup} = \frac{\text{Execution time}_{old}}{\text{Execution time}_{new}} = \frac{IC \times 1.2 \times 1}{IC \times 1.375 \times 0.6} = \frac{1.2}{0.825} \approx 1.45 //$$

The speedup of 12-stage over 5-stage pipeline for data hazard is 1.45 //

b) For 5-st pipeline:

$$CPI_{5\text{-stage}} = \left(\frac{6}{5}\right) + (0.2 \times 0.05 \times 2)$$

$$= (1.02) + (0.02)$$

$$= 1.22 \quad -①$$

Clock cycle time = 1 ns $-②$

For 12-stage pipeline:

$$CPI_{12\text{-stage}} = \left(\frac{3+8}{8}\right) + (0.2 \times 0.05 \times 5)$$

$$= (11/8) + 0.05$$

$$= 1.425 \quad -③$$

Clock cycle time = 0.6 ns $-④$

$$\text{Execution time}_{old} = IC \times CPI \times \text{Cycle Time}$$

$$= IC \times 1.22 \times 1 \quad -⑤ \quad (\text{From } ① \& ②)$$

$$\text{Execution time}_{new} = IC \times CPI \times \text{Cycle Time}$$

$$= IC \times 1.425 \times 0.6 \quad -⑥ \quad (\text{From } ③ \& ④)$$

Using ⑤ & ⑥

$$\text{Speedup} = \frac{\text{Execution time}_{old}}{\text{Execution time}_{new}}$$

$$= \frac{IC \times 1.22 \times 1}{IC \times 1.425 \times 0.6} = \frac{1.22}{0.855} \approx 1.426 //$$

The speedup of 12-stage over 5-stage pipeline for the branch mispredictions is 1.426//

ECE 586
Homework #3

Due date: 02/27/2023, Monday

Latencies beyond single cycle	
Memory LD	+3
Memory SD	+1
Integer ADD, SUB	+0
Branches	+1
fadd.d	+2
fmul.d	+4
fdiv.d	+10

```
Loop:   fld     f2,0(Rx)
I0:     fmul.d  f2,f0,f2
I1:     fdiv.d  f8,f2,f0
I2:     fld     f4,0(Ry)
I3:     fadd.d  f4,f0,f4
I4:     fadd.d  f10,f8,f2
I5:     fsd     f4,0(Ry)
I6:     addi    Rx,Rx,8
I7:     addi    Ry,Ry,8
I8:     sub     x20,x4,Rx
I9:     bnz     x20,Loop
```

Figure 3.47 Code and Latencies for Exercises 3.1-3.5

Problem 1:

3.1 What is the baseline performance (in cycles, per loop iteration) of the code sequence in Figure 3.47 if no new instruction's execution could be initiated until the previous instruction's execution had completed? Ignore front-end fetch and decode. Assume for now that execution does not stall for lack of the next instruction, but only one instruction/cycle can be issued. Assume the branch is taken, and that there is a one-cycle branch delay slot.

Problem 2:

3.2 Think about what latency numbers really mean—they indicate the number of cycles a given function requires to produce its output. If the overall pipeline stalls for the latency cycles of each functional unit, then you are at least guaranteed that any pair of back-to-back instructions (a “producer” followed by a “consumer”) will execute correctly. But not all instruction pairs have a producer/consumer relationship. Sometimes two adjacent instructions have nothing to do with each other. How many cycles would the loop body in the code sequence in Figure 3.47 require if the pipeline detected true data dependences and only stalled on those, rather than blindly stalling everything just because one functional unit is busy? Show the code with < stall> inserted where necessary to accommodate stated latencies. (*Hint:* an instruction with latency + 2 requires two < stall> cycles to be inserted into the code sequence.) Think of

it this way: a one-cycle instruction has latency 1 + 0, meaning zero extra wait states. So, latency 1 + 1 implies one stall cycle; latency 1 + *N* has *N* extra stall cycles.

Problem 3 :

3.3 Consider a multiple-issue design. Suppose you have two execution pipelines, each capable of beginning execution of one instruction per cycle, and enough fetch/decode bandwidth in the front end so that it will not stall your execution. Assume results can be immediately forwarded from one execution unit to another, or to itself. Further assume that the only reason an execution pipeline would stall is to observe a true data dependency. Now how many cycles does the loop require?

Problem 4:

3.4 In the multiple-issue design of Exercise 3.3, you may have recognized some subtle issues. Even though the two pipelines have the exact same instruction repertoire, they are neither identical nor interchangeable, because there is an implicit ordering between them that must reflect the ordering of the instructions in the original program. If instruction *N* + 1 begins execution in Execution Pipe 1 at the same time that instruction *N* begins in Pipe 0, and *N* + 1 happens to require a shorter execution latency than *N*, then *N* + 1 will complete before *N* (even though program ordering would have implied otherwise). Recite at least two reasons why that could be hazardous and will require special considerations in the microarchitecture. Give an example of two instructions from the code in Figure 3.47 that demonstrate this hazard.

Problem 5:

3.5 Reorder the instructions to improve performance of the code in Figure 3.47. Assume the two-pipe machine in Exercise 3.3 and that the out-of-order completion issues of Exercise 3.4 have been dealt with successfully. Just worry about observing true data dependences and functional unit latencies for now. How many cycles does your reordered code take?

Alan Palayil

A20447935

Homework #3

ECE 586

Due Date: 2/27/23

Problem 1

Instruction	Clock Execution	Clk Completion
fld f2, 0(Rx)	1	4
MULTD f2, f0, f2	5	9
DiV f8, f2, f0	10	20
ld f4, 0(Ry)	21	24
Add f4, f0, f4	25	27
Add f10, f8, f2	28	30
sd f4, 0(Ry)	31	32
Addi Rx, Rx, 8	33	33
Addi Ry, Ry, 8	34	34
Sub x20, x4, Rx	35	35
bnz x20, Loop	36	37

It takes 37 cycles to complete an iteration.

Clk execution starts from 1 and add the memory ID latencies to clk completion.

There are two execution pipes available. The design also assumes forwarding to make results available immediately.

Clk cycle	Execution pipeline 1	Execution pipeline 2
1	LD $F_2, 0(R_x)$	NOP
2	<stall>	NOP
3	<stall>	NOP
4	<stall>	NOP
5	MULTD F_2, F_0, F_2	NOP
6	<stall>	NOP
7	<stall>	NOP
8	<stall>	NOP
9	<stall>	NOP
10	DIVD F_8, F_2, F_0	LD $F_4, 0(R_y)$
11	<stall>	<stall>

Clk cycle	Execution pipeline 1	Execution pipeline 2
12	<stall>	<stall>
13	<stall>	<stall>
14	<stall>	ADD F_4, F_0, F_4
15	<stall>	<stall>
16	<stall>	<stall>
17	<stall>	NOP
18	<stall>	NOP
19	<stall>	NOP
20	<stall>	NOP
21	ADDD F_0, F_8, F_2	SD $F_4, 0(R_y)$
22	ADDI $R_z, R_x, 8$	ADDI $R_y, R_y, 8$
23	SUB X_{20}, X_{14}, R_x	BNZ x20, Loop
24	<stall>	NOP

The iteration takes 24 cycles to complete in the 2 issue design.

Problem 2

From the figure, the latency cycles used to allow instruction to complete

LOOP: LD	$F_2, 0(R_x)$	1+3	1
<stall>			2
<stall>			3
<stall>			4
MULTD	F_2, F_0, F_2	1+4	5
<stall>			6
<stall>			7
<stall>			8
<stall>			9
DIVD	F_8, F_2, F_0	1+10	10
LD	$F_4, 0(R_y)$	1+3	11

<stall due to LD latency>			12
<stall due to LD latency>			13
<stall due to LD latency>			14
ADD D	F_4, F_0, F_4	1+2	15
<stall due to DIVD latency>			16
<stall due to DIVD latency>			17
<stall due to DIVD latency>			18
<stall due to DIVD latency>			19
<stall due to DIVD latency>			20
ADD	F_{10}, F_8, F_2	1+2	21
SD	$F_4, 0(R_y)$	1+1	22
ADDI	$R_x, R_x, \#8$	1	23
ADDI	$R_y, R_y, \#8$	1	24
SUB	R_{20}, R_4, R_x	1	25
BNZ	R_{20}, LOOP	1+1	26
<stall due to BNZ			27

Total cycles per loop iterations: 27

Problem3 Considering data from the figures

Instructions Execution pipe 0	Execution pipe 1	Cycle
LOOP: LD F_2, $0(R_x)$	NOP	1
<stall due to LD latency>	NOP	2
<stall due to LD latency>	NOP	3
<stall due to LD latency>	NOP	4
<stall due to LD latency>	NOP	5
MULTD F_2, F_0, F_2	NOP	6
<stall for MULTD latency>	NOP	7
<stall for MULTD latency>	NOP	8
<stall for MULTD latency>	NOP	9
<stall for MULTD latency>	NOP	10

Instructions Execution pipe 0	Execution pipe 1	Cycle
DIVD F_8, F_2, F_0	LD $F_4, 0(R_y)$	11
<stall due to LD latency>	NOP	12
<stall due to LD latency>	NOP	13
<stall due to LD latency>	NOP	14
ADDD F_4, F_0, F_4	NOP	15
<stall due to DIVD latency>	NOP	16
<stall due to DIVD latency>	NOP	17
<stall due to DIVD latency>	NOP	18
<stall due to DIVD latency>	NOP	19
<stall due to DIVD latency>	NOP	20
ADDD F_{10}, F_8, F_2	SD $F_4, 0(R_y)$	21
ADDI $R_x, R_x, \#8$	ADDI $R_y, R_y, \#8$	22
SUB R_{20}, R_4, R_x	BNZ R_{20}, LOOP	23
<stall due to BNZ>	NOP	24

Total cycles per loop iteration: 24//

Problem 4

Based on the previous question, if the instructions are performed it is hazardous and will require special considerations such as:

- N and N+1 look to work on same register or memory.
- N+1 cannot be allowed to change arch state in case N is to be retired.
- If an interrupt occurs between N and N+1, then N+1 must not have been allowed to write its results to any permanent architectural state.

The two instructions that demonstrate this hazard are:

- DIVD F_8, F_2, F_0 //
- LD $F_4, 0(R_y)$ //

Problem 5

The instruction code can be reordered to improve performance using two pipeline machines. The first pipeline adds LD, MULTD, DIVD, ADDI, SUB, and ADDD instructions, while the second pipeline adds LD, ADDD, SD, ADDI, and BNZ instructions in parallel. The second pipeline doesn't execute any instruction if the next instruction depends on a register used by the first pipeline.

Instructions Execution pipeline 0	Execution pipeline 1	Cycle
LOOP: LD F_2, 0(R_x)	LD F_4, 0(R_y)	1
<stall>	NOP	2
<stall>	NOP	3
<stall>	NOP	4
MULTD F_2, F_0, F_2	ADDD F_4, F_0, F_4	5
<stall>	NOP	6
<stall>	NOP	7
<stall>	SD F_4, 0(R_y)	8
<stall>	NOP	9
DIVD F_8, F_2, F_0	NOP	10
<stall>	NOP	11
<stall>	NOP	12
<stall>	NOP	13
<stall>	NOP	14
<stall>	NOP	15
<stall>	NOP	16
<stall>	NOP	17
<stall>	NOP	18
ADDI R_x, R_x, #8	ADDI R_y, R_y, #8	19
SUB R_{20}, R_4, R_x	NOP	20
ADDD F_{10}, F_8, F_2	BNZ R_{20}	21
<stall>	NOP	22

Total cycles per loop iteration is 22

ECE 586
Homework #4

Due date: 04/12/2023, Wednesday

Problem 1:

B.5 You are building a system around a processor with in-order execution that runs at 1.1 GHz and has a CPI of 1.35 excluding memory accesses. The only instructions that read or write data from memory are loads (20% of all instructions) and stores (10% of all instructions). The memory system for this computer is composed of a split L1 cache that imposes no penalty on hits. Both the I-cache and D-cache are direct-mapped and hold 32 KB each. The I-cache has a 2% miss rate and 32-byte blocks, and the D-cache is write-through with a 5% miss rate and 16-byte blocks. There is a write buffer on the D-cache that eliminates stalls for 95% of all writes. The 512 KB write-back, unified L2 cache has 64-byte blocks and an access time of 15 ns. It is connected to the L1 cache by a 128-bit data bus that runs at 266 MHz and can transfer one 128-bit word per bus cycle. Of all memory references sent to the L2 cache in this system, 80% are satisfied without going to main memory. Also, 50% of all blocks replaced are dirty. The 128-bit-wide main memory has an access latency of 60 ns, after which any number of bus words may be transferred at the rate of one per cycle on the 128-bit-wide 133 MHz main memory bus.

a. What is the average memory access time for instruction accesses?
b. What is the average memory access time for data reads?
c. What is the average memory access time for data writes?
d. What is the overall CPI, including memory accesses?

Problem 2:

B.13 A program is running on a computer with a four-entry fully associative (micro) translation lookaside buffer (TLB) (Figure B.33):

VP#	PP#	Entry valid
5	30	1
7	1	0
10	10	1
15	25	1

FIGURE B.33 TLB contents (problem B.12)

The following is a trace of virtual page numbers accessed by a program. For each access indicate whether it produces a TLB hit/miss and, if it accesses the page table, whether it produces a page hit or fault. Put an X under the page table column if it is not accessed (Figures B.34 and B.35)

Virtual page index	Physical page #	Present
0	3	Y
1	7	N
2	6	N
3	5	Y
4	14	Y
5	30	Y
6	26	Y
7	11	Y
8	13	N
9	18	N
10	10	Y
11	56	Y
12	110	Y
13	33	Y
14	12	N
15	25	Y

FIGURE B.34 Page table contents.

Virtual page accessed	TLB (hit or miss)	Page table (hit or fault)
1		
5		
9		
14		
10		
6		
15		
12		
7		
2		

FIGURE B.35 Page access trace.

Problem 3:

2.20 Consider the usage of critical word first and early restart on L2 cache misses. Assume a 1 MB L2 cache with 64-byte blocks and a refill path that is 16 bytes wide. Assume that the L2 can be written with 16 bytes every 4 processor cycles, the time to receive the first 16 byte block from the memory controller is 120 cycles, each additional 16 byte block from main memory requires 16 cycles, and data can be bypassed directly into the read port of the L2 cache. Ignore any cycles to transfer the miss request to the L2 cache and the requested data to the L1 cache.

a. How many cycles would it take to service an L2 cache miss with and without critical word first and early restart?

b. Do you think critical word first and early restart would be more important for L1 caches or L2 caches, and what factors would contribute to their relative importance?

Problem 4:

2.21 You are designing a write buffer between a write-through L1 cache and a write-back L2 cache. The L2 cache write data bus is 16 B wide and can perform a write to an independent cache address every four processor cycles.

a. How many bytes wide should each write buffer entry be?

b. What speedup could be expected in the steady state by using a merging write buffer instead of a nonmerging buffer when zeroing memory by the execution of 64-bit stores if all other instructions could be issued in parallel with the stores and the blocks are present in the L2 cache?

c. What would the effect of possible L1 misses be on the number of required write buffer entries for systems with blocking and nonblocking caches?

Alan Pulayil

A20447935

ECE 586

Homework #4

Due Date: 4/12/23

Problem 1. a) Given: Access Latency = 60 ns = 160 clocks

Bus transfer/access = 64 bytes × 8 bits/byte / 128 bits/transfer = 4 bus/transfer

$t_{avg} = t_{HIT} + \text{miss percentage} \times t_{miss}$

$t_{hit,m}$ = access time + transfer time

Processor clocks per transfer = 2.66 GHz p / 132 MHz b

= 20 processor clocks per transfer

$\therefore t_{hit,m} = 160 + 80$

$= 240$ clocks

$\therefore t_{avg,L2} = t_{Hit,L2} + \text{miss percentage}_{L2} \times t_{miss,L2}$

$= 40 + 0.2 \times 240$

$= 88$ clocks

$t_{avg,I} = t_{hit,I} + \text{miss percentage}_I \times t_{miss,I}$

$= 0 + 0.02 \times 88$

$= 1.76$ clocks //

b) $t_{avg,D} = t_{hit,D} + \text{misspercentage}_D \times t_{Miss,D}$

$= 0 + 0.05 \times 88$

$= 4.4$ clocks //

c) There is no reason given for writes to have different access time than reads.

d) Overall CPI = base CPI + Instruction cache CPI + data cache CPI + data cache CPI

$= 0.7 + 1.76 + 0.2 \times 4.4 + 0.05 \times 4.4$

$= 3.56$ //

Problem 2. When a process is created, paging and page table creation takes place. The TLB is checked for the page first and if it's valid, the corresponding frame is accessed from physical memory. If not, the page table is checked, and if found, the corresponding frame is accessed. If the frame is not in main memory, a page fault occurs. If a page is in TLB, it's in main memory and doesn't need to be checked in the page table.

Virtual Page Accessed	TLB (hit/miss)	Page-table (hit/fault)
1	miss	fault
5	hit	-
9	miss	fault
14	miss	fault
10	hit	-
6	miss	hit
15	hit	-
12	miss	hit
7	miss	hit
2	miss	fault

Problem 3. a) To transfer a 64-byte block to L2 cache, it takes 168 cycles without early restart and 120 cycles with early restart, where the first 16 bytes take 120 cycles and remaining 48 bytes take 16 cycles each.

No. of 16 bytes chunk = 64/16 = 4.

No. of cycles = (1 × 120) + (16 × 3) = 168 (without critical)

No. of cycles = (1 × 120) = 120 (with critical)

b) Critical word first and early restart techniques reduce miss time and have a larger impact on performance in L2 caches due to their larger block sizes and better work rate compared to L1 caches. The impact depends on the number of misses in L1 or L2.

Problem 4. a) Each write buffer entry should be 16 B wide //

b) 64-bit = 8 Bytes

For L2 merging write buffer, assume the width is 16 B. Then it will take 2 cycles to filled up one entry because the merging write buffer

is taking 4 cycles to write to the next level.
So for non-merging buffer, it should take → 4+4=8 cycles.
∴ the speed for using merging write buffer is doubled.

c) • With blocking caches, the CPU stalls when cache miss happens. So, the number of required write buffer entries would not be different.
• With non-blocking cache, writes can be processed from the write buffer during misses and so it requires less write buffer entries.