Review Test Submission: Homework 2 (due 10/2)

User	Yuchen Wang
Course	CSE661/CIS655 - Advanced Computer Architecture - F20
Test	Homework 2 (due 10/2)
Started	9/26/20 3:21 AM
Submitted	10/2/20 11:59 PM LATE
Due Date	10/2/20 11:59 PM
Status	Completed
Attempt Score	96 out of 100 points
Time Elapsed	164 hours, 37 minutes
Results Displayed	All Answers, Submitted Answers, Correct Answers, Feedback, Incorrectly Answered Questions

Question 1

12 out of 12 points

[12] [Max 1 page] Read and comment on the paper labelled with R2a . Read pages 1-35 of the paper labeled with R2b and comment on it, particularly ranking in order of importance to ILP the features such as alias analysis, branch prediction, register renaming, and simultaneous speculative execution across different paths; and comment on the weaknesses of this study.

Selected Answer:

- R2a
    - This paper compares an example implementation from the RISC (mainly MIPS for RISC) and CISC (mainly VAX for CISC) architectures on nice of ten SPEC benchmarks in order to study on the **purely architectural advantages of RISC over CISC**
    - The authors conducted their experiments and discussed their processor performance results basing on a fundamental frame of reference: **time/program = (instructions/program) * (cycle/instruction) * (time/cycle)**
    - By conducting benchmark experiments (Ex. fpppp) and analyzing data mainly on **CPI ratio(Average VAX CPI divided by average MIPS CPI)** and **instruction radio (the radio of MIPS instruction executions to VAX instruction executions),** along with RICS factor(the net advantages of RICS), authors predict '**while VAX may catch up to current single-instruction-issue RISC performance, RISC design will push on with earlier adoption of advanced implementation techniques, achievein still higher performance'**
    - And authors conclude **'RISC as exexplified by MIPS offers a significant processor performance advantage over a VAX of comparable hardware organization(CISC)'.**
    - The paper elaborates the assumptions and limitations on their results clearly and logically. For example, the author addresses the 'three caveats go along with the results' in section five. I think those elaborations help readers to understand their results and analysis easier and correctly.
- R2b
    - Ranking in order of importance to ILP the features such as **alias analysis**, **branch prediction, register renaming**, and **simultaneous speculative execution** across different paths
        - The **most important** feature for ILP is **branch prediction (a kind of speculative execution),** as the paper describe that 'If we start with the Perfect model and remove branch prediction, the median parallelism plummets from 30.6 to 2.2'
        - The **second** and **third** most most important feature are **alias analysis** and **register renaming** respectively, as ''removing alias analysis and register renaming results in more acceptable median parallelism of 3.4, 4.8 respectively.'
        - **Jump prediction** is **the most not important** feature among these four features, as it only result 30.6 to 21.3 parallelism decrement if it is removed.
        - **Branch prediction** combines with **simultaneous speculative execution** across different branching code path is also important, as that raise the observed parallelism of 4-10 up to 7-13.
    - Comment on the **weaknesses** of this study.
        - Most of the experiments are processed under ideal situations,as the authors apply to much environment-restrictions on the experiments
        - The study made their conclusion based parallelism of 18 different program at more than 350 points. I think 18 programs may not be a sufficient experiment space to make a conclusion on.
        - I think another weakness is its extreme length to read:(

Correct Answer:	[None]
Response Feedback:	[None Given]

Question 2

8 out of 8 points

[8] In the table below, common parallel I/O buses are shown. Please search on the web or

use other resources to fill the required fields with the most up-to-date data.

	SATA	SCSI	PCI-X	Choice of yours
Clock Rate				
Data Width				
Bandwidth				

Selected Answer:

	SATA	SCSI	PCI-X	USB 2.0		
Clock Rate	150 MHz	320 MHz	533 MHz	960 MHz		
Data Width	4	4	64/lane	2		
Bandwidth	600 MB/s	640 MB/s	4266 MB/s	60MB/s		

Correct Answer: [None]

Response Feedback: [None Given]

Question 3

8 out of 8 points

[8] Suppose we have a deeply pipelined processor, for which we implement a branch-target buffer for the conditional branches only. Assume that the misprediction penalty is always four cycles and the buffer miss penalty is always three cycles. Assume a 90% hit rate, 80% accuracy, and 10% branch frequency. How much faster is the processor with the branch-target buffer versus a processor that has a fixed two-cycle branch penalty? Assume a base clock cycle per instruction (CPI) without branch stalls of 1.

Selected Answer:

Assume a base clock cycle per instruction (CPI) without branch stalls of 1.
1. For the processor that has a fixed two-cycle branch penalty, the branch frequency is 10%.

- ExecutionTime = ExecutionTimeOld * ((1-p) + p/s) = CPI * ((1-branchFrequency) + branchFrequency/(CPI/(branchPenalty+CPI))) = 1 * ((1 - 0.1) + 0.1/(1/3)) = 1 * (0.9 + 0.3) = 1.2

2. For the processor with the branch-target buffer, assume a 90% hit rate, 80% accuracy, and also 10% branch frequency.

- As utilizing BTB, if hit and instruction is branch predicted taken, can fetch target **immediately**(hit time = 0). When branch prediction is taken, AMAT = Hit time + (Miss rate * Miss penalty) = 0 + (0.1 * 3) = 0.3
- Mean time of branch (MTOB) = 0.8 * 0 + (1-0.8) * 4 = 0.2 * 4 = 0.8
- ExecutionTime = ExecutionTimeOld * ((1-p) + p/s) = CPI * ((1- branchFrequency) + branchFrequency/(CPI/(MTOB+ AMAT + CPI))) = 1 * ((1 - 0.1) + 0.1/(1/2.1)) = 1 * (0.9 + 0.21) = 1.11

3. SpeedUp = ExecutionTimeOld/ExecutionTimeNew = 1.2/1.11 = 1.08

- The processor with the branch-target buffer is **(1.08 - 1) = 8% faster** than a processor that has a fixed two-cycle branch penalty.

OR:
1. For the processor that has a fixed two-cycle branch penalty, the branch frequency is 10%.

- ExecutionTime = ExecutionTimeOld + BranchTime = CPI + branchPenality * branchFrequency = 1 + 2 * 0.1 = 1.2

2. For the processor with the branch-target buffer, assume a 90% hit rate, 80% accuracy, and also 10% branch frequency.

- As utilizing BTB, if hit and instruction is branch predicted taken, can fetch target **immediately**(hit time = 0). When branch prediction is taken, AMAT = Hit time + (Miss rate * Miss penalty) = 0 + (0.1 * 3) = 0.3
- Mean time of branch (MTOB) = 0.8 * 0 + (1-0.8) * 4 = 0.2 * 4 = 0.8
- ExecutionTime = ExecutionTimeOld + BranchTime = CPI + branchPenality * branchFrequency = CPI + (AMAT + MTOB) * branchFrequency = 1 + (0.8+0.3) * 0.1 = 1.11

3. SpeedUp = ExecutionTimeOld/ExecutionTimeNew = 1.2/1.11 = 1.08

- The processor with the branch-target buffer is **STILL (1.08 - 1) = 8% faster** than a process.....................

Correct Answer:	[None]
Response Feedback:	[None Given]

Question 4

8 out of 8 points

✓ [8] In a server farm such as that used by Amazon or Google, 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. If a company has 1,000 computers, each with a MTTF of 25 days, and it experiences catastrophic failure only if 1/8 of the computers fail, what is the MTTF for the system?

Selected Answer:

MTTF = Total hours of operation/Total assets in use
For each computer in the computer, we have a MTTF(computer) of 25 days.

- Total hours of operation(computer)/Total assets in use(computer) = 25

Within the system, as the system experiences catastrophic failure only if 1/8 of the computers fail, total hours of operation(computer) = MTTF(System) * 1000; Total assets in use(computer) = 1000 * 1/8
Hence,

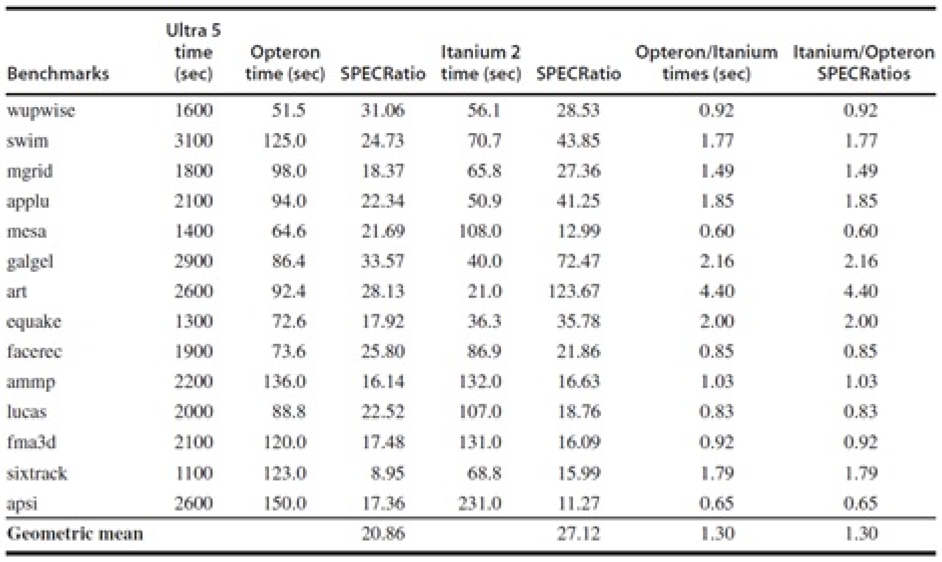
- MTTF(computer) = MTTF(System) * 1000/(1000 * 1/8)
- MTTF(System) = MTTF(computer) /8 = 25/8 = 3.125

Correct Answer:	[None]
Response Feedback:	[None Given]

Question 5

8 out of 8 points

[8] Your company is trying to choose between purchasing the Opteron or Itanium 2. You have analyzed your company's applications, and 60% of the time it will be running applications similar to facerec, 20% of the time applications similar to applu, and 20% of the time applications similar to sixtrack. (See Figure 1) If you were choosing just based on overall SPEC performance, which would you choose and why? What is the weighted average of execution time ratios for this mix of applications for the Opteron and Itanium 2?



Benchmarks	Ultra 5 time (sec)	Opteron time (sec)	SPECRatio	Itanium 2 time (sec)	SPECRatio	Opteron/Itanium times (sec)	Itanium/Opteron SPECRatios
wupwise	1600	51.5	31.06	56.1	28.53	0.92	0.92
swim	3100	125.0	24.73	70.7	43.85	1.77	1.77
mgrid	1800	98.0	18.37	65.8	27.36	1.49	1.49
applu	2100	94.0	22.34	50.9	41.25	1.85	1.85
mesa	1400	64.6	21.69	108.0	12.99	0.60	0.60
galgel	2900	86.4	33.57	40.0	72.47	2.16	2.16
art	2600	92.4	28.13	21.0	123.67	4.40	4.40
equake	1300	72.6	17.92	36.3	35.78	2.00	2.00
facerec	1900	73.6	25.80	86.9	21.86	0.85	0.85
ammp	2200	136.0	16.14	132.0	16.63	1.03	1.03
lucas	2000	88.8	22.52	107.0	18.76	0.83	0.83
fma3d	2100	120.0	17.48	131.0	16.09	0.92	0.92
sixtrack	1100	123.0	8.95	68.8	15.99	1.79	1.79
apsi	2600	150.0	17.36	231.0	11.27	0.65	0.65
Geometric mean			20.86		27.12	1.30	1.30

Figure 1. SPECfp2000 execution times (in seconds) for the Sun Ultra 5—the reference computer of SPEC2000— and execution times and SPECRatios for the AMD Opteron and Intel Itanium 2

Selected Answer:

- If you were choosing just based on overall SPEC performance, which would you choose and why?
    - I will choose Itanium 2, as it has a higher overall SPECRadio than Opteron.
- What is the weighted average of execution time ratios for this mix of applications for the Opteron and Itanium 2?
    - As 60% of the time it will be running applications similar to facerec, 20% of the time applications similar to applu, and 20% of the time applications similar to sixtrack
    - weighted average of execution time(Opteron) = 0.6 * facerecTime + 0.2 * appluTime + 0.2 * sixtrackTime = 0.6 * 73.6 + 0.2 * 94.0 + 0.2 * 123.0 = 44.16 + 18.8 + 24.6 = 87.56
    - weighted average of execution time(Itanium) = 0.6 * facerecTime + 0.2 * appluTime + 0.2 * sixtrackTime = 0.6 * 86.9 + 0.2 * 50.9 + 0.2 * 68.8 = 52.14 + 10.8 + 13.76 = 76.7
    - the weighted average of execution time ratio = weighted average of execution time(Opteron)/weighted average of execution time(Itanium) = 87.56/76.7 = 1.14

Correct Answer:	[None]
Response Feedback:	[None Given]

Question 6

10 out of 12 points

[12] Assume a five-stage single-pipeline microarchitecture (fetch, decode, execute, memory, write- back) and the code given below. All ops are one cycle except LW and SW, which are 1+2 cycles, and branches, which are 1+1 cycles. There is no forwarding. Show the phases of each instruction per clock cycle for one iteration of the loop.

```
Loop: lw x1,0(x2)
      addi x1,x1, 1
      sw x1,0(x2)
      addi x2,x2,4
      sub x4,x3,x2
      bnz x4,Loop
```

a. How many clock cycles per loop iteration are lost to branch overhead?
b. Assume a static branch predictor, capable of recognizing a backward branch in the Decode stage. Now how many clock cycles are wasted on branch overhead?
c. Assume a dynamic branch predictor. How many cycles are lost on a correct prediction?

Selected Answer:

As there are some of ops taking more than one cycle, we can treat them as having n extra stage (Ex. LW has 5 + 2 = 7 cycles).

a. How many clock cycles per loop iteration are lost to branch overhead?
    a. **BNZ is the branch instruction in the loop and have 5 stages + 1 extra stages**
    b. **We don't treat any data and structure hazard as part of branch overhead**
    c. If CPU allows to **get branch result at end of ID stage**, and **the extra stage (now can be EX, MEM, WB) happens after the ID stage**, there will be **one clock cycle lost per loop iteration** to branch overhead.
    d. If CPU allows to **get branch result at end of ID stage**, and **the extra stage (now can be IF, ID) happens before or with the ID stage**, there will be **two clock cycles lost per loop iteration** to branch overhead.
    e. If CPU requires to **get branch result at end of EX stage**, and **the extra stage (now can be MEM, WB) happens after the EX stage**, there will be **two clock cycle lost per loop iteration** to branch overhead.
    f. If CPU requires to **get branch result at end of EX stage**, and **the extra stage (now can be IF, ID, EX) happens before or with the EX stage**, there will be **three clock cycles lost per loop iteration** to branch overhead.

b. Assume a static branch predictor, capable of recognizing a backward branch in the Decode stage. Now how many clock cycles are wasted on branch overhead?
    a. If CPU requires to **get branch result at end of EX stage**
        a. If **the extra stage (now can be EX, MEM, WB) happens after the ID stage**, there will be **one clock cycle lost per loop iteration** to branch overhead **on a correct prediction.**
        b. If **the extra stage (now can be IF, ID) happens before or with the ID stage**, there will be **two clock cycles lost per loop iteration** to branch overhead **on a correct prediction.**
        c. If **the extra stage (now can be MEM, WB) happens after the EX stage**, there will be **two clock cycle lost per loop iteration** to branch overhead **on a incorrect prediction.**
        d. If **the extra stage (now can be IF, ID, EX) happens before or with the EX stage**, there will be **three clock cycles lost per loop iteration** to branch overhead **on a incorrect prediction.**
    b. If CPU allows to **get branch result at end of ID stage**, same situation will happen as what happen on **correct prediction.**

c. Assume a dynamic branch predictor. How many cycles are lost on a correct prediction?
    a. There will be **no cycle lost on a correct prediction.**

Correct Answer:	[None]
Response Feedback:	Please draw a diagram of pipeline phases.

Question 7

12 out of 12 points

[12] In this exercise, we assume that the following MIPS code is executed on a pipelined processor with a 5-stage pipeline (IF, ID, EX, MEM, WB), full forwarding, and a predict-taken branch predictor.
Hints: First find data dependencies. Then examine if these dependencies can cause data hazard. Branch instruction can cause control hazards. Instructions will be flushed if the prediction is wrong.

```
        LW R2, 0(R1)
LABEL1: BEQ R2, R0, LABEL2 #Not taken once, then always taken
        LW R3, 0(R2)
        BEQ R3, R0, LABEL1 #Always taken
        ADD R1, R3, R1
LABEL2: SW R1, 0(R2)
        ADD R4, R5, R6
```

Draw the pipeline execution diagram for this code, assuming that branches execute in the EX stage. Please upload your legible diagram as a PDF or JPG.

Selected Answer: Q7.PNG
Response Feedback: [None Given]

Question 8

10 out of 12 points

[12] Consider a single-issue design with a single execution pipeline with five-stages (Fetch, Decode, Execute, Memory, Write Back) capable of beginning execution of one instruction per cycle with data forwarding and bypassing hardware, all memory accesses take 1 (one) clock cycle, and all memory references hit in the cache. Assume that the branch is handled by predicting it not taken. Use the following code fragment and assume that the initial value of R4 is R2+440. Please show the timing of the instruction sequence for this pipeline using the given pipeline chart. How many cycles does this loop take to execute?

```
Loop  LD    F0, 0(R2)
      LD    F4, 0(R3)
      MULTD F0, F0, F4
      ADDD  F2, F0, F2
      ADDI  R2, R2, #8
      ADDI  R3, R3, #8
      SUB   R5, R4, R2
      BNEZ  R5, Loop
```

Latencies	
LD, SD	+1
ADDI, SUB	+0
BNEZ	+1
ADDD	+1
MULTD	+3

Selected Answer: Q8.PNG

Response Feedback: You also need to calculate the number of iterations so that you can calculate the overall cycles of this loop instead of just one iteration.

Hint: you can use the initial value of R4 is R2+440 to calculate the number of iterations.

Question 9

20 out of 20 points

[20] In this part, you are expected to implement multiplication of two 2-dimensional matrices in both C (or Python) and an assembly language of your choice to compare their time and space performances (you can use `command time -v <executable>`, and `du -sh --apparent-size <file>`.) Just another hint:see this link and make a note that `gcc -S filename.c` or `gcc -masm=intel -S filename.c` would generate `filename.s` with the compiled assembly code.

You can generate random numbers to populate large arrays of data to test your programs. Remember to get the time before and after executing your programs to measure their execution times. Please submit your written report to include

a. [1] your choice of environment, OS, IDE, editor, etc.,
b. [1] your algorithm,
c. [2] a well-commented C/Python code,
d. [2] a well-commented assembly code,
e. [8] screenshots of execution times, showing also your user name,
f. [4] performance comparisons of both implementations, including execution time, code size and memory space requirements,
g. [2] and your remarks on any differences.
h. [Bonus 5] For your own assembly code, rather than using the compiled assembly code directly from your C code.

Selected Answer: HW2.zip

Response Feedback: [None Given]

Thursday, October 8, 2020 7:24:00 AM EDT

← OK