CS433 Homework 1
(Chapter 1, Appendix C)

Assigned on 9/5/2017
Due in class on 9/19/2017

Instructions:

1. Please write your name and NetID clearly on the first page.
2. Refer to the course fact sheet for policies on collaboration.
3. Due **IN CLASS** on 9/19/2017.

Problem 1 [10 points]

You are running a program of which the expected gain is inversely proportional to the execution time (e.g., cryptocurrency mining). In other words, the faster you run the program, the more money you earn. Assume that (i) the serial execution of the program takes 20 minutes (ii) 80% of the program is parallelizable (iii) the parallelizable part is embarrassingly parallel so that it can be evenly distributed to any number of cores (iv) the expected gain is $10 / (Execution time in seconds) and (v) the cost of operation per core is $0.01 per hour.

Part A [4 points]

Given infinite resource, what is the shortest execution time possible?

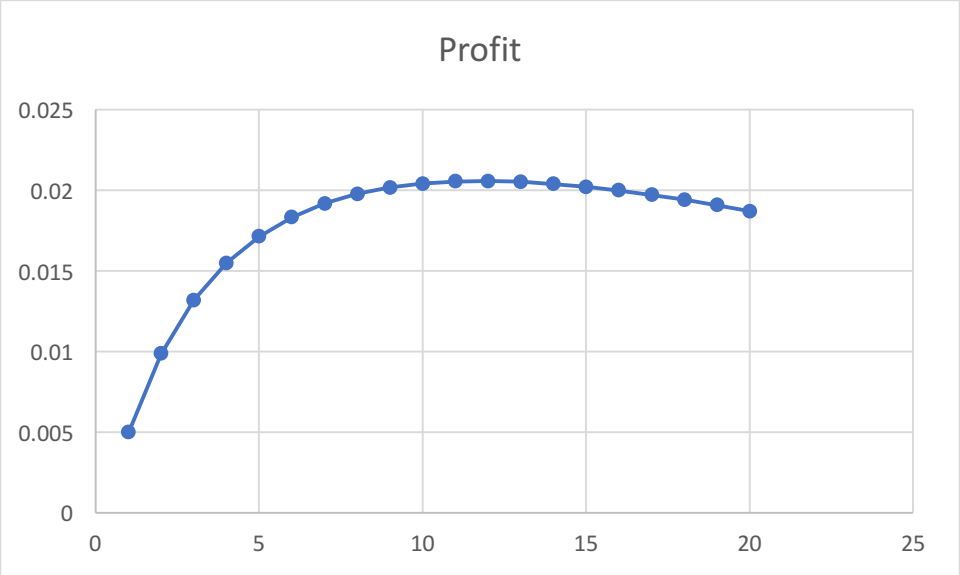
Using Amdahl's law, the execution time with N cores is:
1200 minutes * (20% + 80% / N)
Using an infinite number of cores, the parallelizable part takes no time. The serial part takes the same amount of time which is 20 minutes * 20% = 4 minutes.

[Answer] 4 minutes

Part B [6 points]

How many cores would you use to maximize the profit? What is the maximum profit possible from a single run of the program?

(# of cores) = N
(Execution time in seconds)
= ((60 seconds / 1 minute) * 20 minutes) * (20% + 80% / N)
= 1200 * (0.2 + 0.8 / N) seconds
(Expected gain) = $10 / (1200 * (0.2 + 0.8 / N))
(Cost) = ($0.01 / 3600 seconds) * N * (1200 * (0.2 + 0.8 / N))
(Profit) = (Expected gain) – (Cost)



[Answer] 12 cores, $0.02058333

Problem 2 [10 points]

You are evaluating the performance of a processor without pipelining which runs at 1 GHz. The table shows latencies of different instruction types and the number of instructions of each type in the benchmark program.

Instruction Type	Latency	# of instructions in the Benchmark Program
Memory	5 cycles	40,000,000
Branch	4 cycles	20,000,000
ALU	4 cycles	30,000,000
I/O	10 cycles	10,000,000

Part A [3 points]

Calculate (i) the execution time (ii) CPI and (iii) MIPS of the processor running the benchmark program.

(total # cycles)
= 5 * 40M + 4 * 20M + 4 * 30M + 10 * 10M
= 200M + 80M + 120M + 100M
= 500M
(execution time)
= (total # cycles) / (frequency)
= 0.5 sec
(total # instructions)
= 40M + 20M + 30M + 10M
= 100M
(CPI)
= (total # cycles) / (total # instructions)
= 500M / 100M
= 5
(MIPS)
= ((total # instructions) / 10^6) / (execution time)
= 100 / 0.5
= 200

[Answer] Highlighted above.

Part B [6 points]

You found out that the application often jumps to an instruction address stored in the memory. To optimize for this case, you are adding a new instruction which combines a load instruction and a branch instruction. The new instruction takes 7 cycles and can be applied to 40% of branch instructions. Note that you can replace two original instructions (a load and a branch) with one new instruction. Calculate (i) the execution time (ii) CPI and (iii) MIPS of the processor enhanced with the new instruction.

(# new branch instructions) = 20M * 0.4 = 8M
(# original branch instructions) = 20M * 0.6 = 12M
Since a new branch instruction replaces one original branch instruction and one load instruction,
(# memory instructions) = 40M – 8M = 32M
(# total cycles)
= 5 * 32M + 4 * 12M + 4 * 30M + 10 * 10M + 7 * 8M
= 160M + 48M + 120M + 100M + 56M
= 484M
(execution time)
= (total # cycles) / (frequency)
= 0.484 sec [Answer] Highlighted above.
(total # instructions)
= 32M + 12M + 30M + 10M + 8M
= 92M
(CPI)
= (total # cycles) / (total # instructions)
= 484M / 92M
= 5.26
(MIPS)
= ((total # instructions) / 10^6) / (execution time)
= 92 / 0.484
= 190

[Answer] Highlighted above.

Problem 3 [10 points]

Identify the RAW, WAW and WAR dependences (potential data hazards) in the code below. State whether the dependence will cause a stall. Consider a 5-stage RISC pipeline with IF ID EX MEM WB stages as in Appendix C. Branches are resolved in the ID stage. All stages take 1 cycle. Assume full forwarding.

```
1: ADD R1, R2, R3
2: LD R4, 0(R1)
3: ADD R1, R4, R5
4: SUB R4, R6, R7
5: BEQZ R4, done
```

[Answer]
1->2 RAW (R1)
1->3 WAW (R1)
2->3 WAR (R1)
2->3 RAW (R4) stall (MEM->EX)
2->4 WAW (R4)
2->5 RAW (R4)
3->4 WAR (R4)
4->5 RAW (R4) stall (EX->ID)

Problem 4 [10 points]

We will add support for register-memory ALU instructions to the classic 5-stage RISC pipeline. For example, the register-memory instruction `ADD R4,R5,(R1)` means adding the content of the register R5 to the content of the memory location of the address equal to the value in the register R1 and putting the sum into the register R4. Register-register ALU instructions are unchanged. To offset this increase in complexity, all memory instructions (including original load/store instructions and new register-memory ALU instructions) will be limited to register indirect addressing (i.e., an address can only be provided as a variable held in a register and separate instructions are needed to calculate the effective address and put it into the register).

Part A [1 point]

List a rearranged order of the five traditional stages of the RISC pipeline that will support register-memory instructions.

[Answer] IF-ID-MEM-EX-WB

Part B [3 points]

Describe what forwarding paths are needed for the rearranged pipeline by stating the source, destination, and when that path is used. Include forwarding paths which were also necessary in the original design.

[Answer]
Forwarding paths that need to be added in the new pipeline:
EX → MEM, ALU Op to Load/Store
MEM → MEM, Load/Store to Load/Store
Forwarding paths that were there in the earlier design and that are also needed in the new pipeline:
EX → EX, ALU Op to ALU Op
MEM → ID, Load/Store to Branch
EX → ID, ALU Op to Branch

Part C [2 points]

For the reordered stages of the RISC pipeline, what new data hazards are created by this addressing mode? Give an instruction sequence illustrating each new hazard.

[Answer]

There is a new hazard when an ALU operation is followed by an instruction using the ALU operation's result to perform a memory operation.

Example 1:
ADD R1, R2, R3
LD R4 (R1)

The load could be replaced by any instruction using indirection (LD, ST, ADD)

Example 2:
ADD R1, R2, R3
ST R1, (R4)

Example 3:
ADD R1, R2, R3
ADD R4, R5, (R1)

Part D [2 points]

Show that the RISC pipeline with register-memory ALU instructions can take more or fewer instruction for a given program than the original RISC pipeline. Find an instruction sequence for the new architecture that is shorter than any equivalent sequence of instructions on the old architecture, and an instruction sequence for the old architecture that is shorter than any equivalent instruction sequence on the new architecture.

[Answer]
Shorter sequence on the new machine when loading from zero offset addresses.
Original code:
LD R1, 0(R2)
ADD R4, R1, R3
New code:
ADD R4, R3, (R2)

Shorter sequence on the original machine when loading from non-zero offset addresses.
Original code:
LD R1, 4(R2)
New code:
ADD R2, R2, #4
LD R1 (R2)

Part E [2 points]

Assume all instructions take 1 clock cycle per stage. Many instructions are common to both architectures, but even compatible programs may have different performance. Give one compatible instruction sequence which will run with a higher CPI on the new design than on the original pipeline, and one instruction sequence which will run with a lower CPI on the new design.

[Answer]
CPI is affected by stalls. The new pipeline has a higher CPI when memory operations have to wait for ALU result:
ADD R1, R2, R3
LD R4 (R1)

The original pipeline has a higher CPI when ALU operations have to wait for memory operation results. This stall is avoided in the new pipeline:
LD R4 (R1)
ADD R6, R4, R5

Problem 5 (*for graduate students only*) [10 points]

Consider three different 5-stage RISC pipeline machines with IF ID EX MEM WB stages. The first machine has an oracle branch predictor which always predicts the correct target. In other words, there is no stall due to branch misprediction. The second machine resolves branches in the EX stage using a predict-not-taken scheme. The third machine resolves branches in the ID stage using one branch delay slot. Assume that (i) the first machine's CPI is 1 (ii) they run at the same frequency (iii) 20% of the instructions are branches (iv) 25% of branches are taken and (v) stalls are due to branches alone.

Part A [4 points]

Calculate the CPI of the second machine.

It mispredicts 25% of branches which are 20% * 25% = 5% of the total instructions. As branches are solved in the EX stage, the branch penalty is 2 cycles. CPI is increased since those 5% of mispredicted branches waste 2 cycles each.

(CPI of the second machine) = 1 + 0.05 * 2 = 1.1

[Answer] 1.1

Part B [4 points]

Assuming that the compiler is able to fill 30% of the delay slots with useful instructions, calculate the CPI of the third machine.

70% of branch delay slots are wasted and increase the total number of cycles.

(CPI of the third machine) = 1 + 0.2 * 0.7 = 1.14

[Answer] 1.14

Part C [2 points]

What percentage of delay slots should be filled with useful instructions for the third machine so that the third machine run faster than the second machine?

Assuming X% of the delay slots filled with useful instructions,

(CPI of the third machine) = 1 + 0.2 * (1 – X%) = 1.2 – 0.2 * X%

With the same instruction set, lower CPI means faster execution.

1.2 – 0.2 * X% < 1.1
0.1 < 0.2 * X%
0.5 < X%

[Answer] 50%