Faculty of Media Engineering and Technology
Dept. of Computer Science and Engineering
Dr. Milad Ghantous

CSEN 702: Microprocessors
Winter 2024

Practice assignment 1-solution

1) Convert this code into MIPS. Assume that i, j and k are in $s1, $s2 and $s3 respectively.

```
if ( i == j && i == k )
  i++ ;
else
  j-- ;
j = i + k ;
```

Solution

```
bne $s1, $s2, ELSE          # cond1: branch if !( i == j )
bne $s1, $s3, ELSE          # cond2: branch if !( i == k )
addi $s1, $s1, 1            # if-body: i++
j NEXT                      # jump over else
ELSE: addi $s2, $s2, -1     # else-body: j--
NEXT: add $s2, $s1, $s3     # j = i + k
```

==

2) Consider the following fragment of C code:

```
for (i = 0; i <= 100; i++)
   { A[i] = B[i] + C; }
```

Assume that A and B are arrays of 64-bit integers, and C and i are 64-bit integers.
Assume that all data values and their addresses are kept in memory (at addresses 1000, 3000, 5000, and 7000 for A, B, C, and i, respectively). Assume

that values in registers are lost between iterations. of the loop. You might need to load value 0 in variable i.

a) Write the code for MIPS64.
b) How many instructions are required dynamically?
c) How many memory-data references will be executed?
d) What is the code size in bytes?

Solution:
a)

```
DADD R1, R0, R0             #R0 = 0, initialize i = 0
SD R1,7000(R0)              #store i in address 7000
loop: LD R1,7000(R0)        #get value of i
DSLL R2, R1, 3              #R2 = word offset of B[i]
DADDI R3, R2,3000                   #add base address of B to R2
LD R4,0(R3)                 #load B[i]
LD R5,5000(R0)                      #load C
DADD R6, R4,R5                      #B[i] + C
DADDI R7,R2,1000                            #add base address of A to R2
SD 0(R7), R6                                #A[i] ← B[i] + C
DADDI R1, R1, 1                     #increment i
SD 7000(R0), R1                     #store i
DADDI R8, R1, -101                          #is counter at 101?
BNEZ R8, loop                               #if not 101, repeat
```

Notes:

why we shift left by 3 in (DSLL)? (equivalent to multiplying by 8)

We are working with 64 bits so memory looks like this.

Each address if for is 1 byte.

0	Array[0]
1	Array[0]
2	Array[0]

3	Array[0]
4	Array[0]
5	Array[0]
6	Array[0]
7	Array[0]
8	Array[1]
9	Array[1]
10	Array[1]
11	Array[1]
12	Array[1]
13	Array[1]
14	Array[1]
15	Array[1]
16	Array[2]
17	Array[2]
18	Array[2]
19	Array[2]
20	Array[2]
21	Array[2]
22	Array[2]
23	Array[2]

So array[0] starts at address 0 **for example**, array[1] at 8, array[2] at 16. so in iteration i=0, we want array[0],we shift 0 by 3, it stays 0, we access array[0].
When i=1, we shift by 3, it becomes 1x8 = 8, we access address 8, which is array[1]. At i=2, we shift by 3, it becomes 2x8=16, we access address 16, which is array[2] and so on.

b) We have 2 initialization instructions and then we have 12 instructions that are looping 101 times.
Total : 2 + (12 x101) = 1214 instructions

c) 1 + (5x101) = 506 instructions

d) size = 14 instructions x (4 bytes per instruction) = 56 bytes.

==

3) Of the three factors in the equation:

(*EXECUTION TIME CPU = Number of instructions * CPI * Cycle Time*)

Which is most influenced by?
(a) The technology
(b) The compiler
(c) The computer architecture

SOLUTION:

(a) The Number of Instructions: is a **Compiler** dependent function, because the Compiler is responsible for reducing the number of instructions executed for a given program and a given Instruction Set Architecture (ISA). You can also say it depends on the **architecture**.

(b) CPI: it is dependent on the **Architecture** of the microprocessor as the improvement of the CPI depends on the implementation of the processor

(c) Cycle Time: the smaller the cycle time (higher frequency) the better the

performance and this depends on the **Technology** used in manufacturing the microprocessor itself.
(Note that these factors are not completely independent)
4) When running an integer benchmark on a RISC machine, the average instruction mix was as follows:

Instructions	Average Frequency
Load	26%
Store	9%
Arithmetic	14%
Compare	13%
Cond. branch	16%
Uncond. branch	1%
Call/returns	2%
Shift	4%
Logical	9%
Other (Misc.)	6%

Note than an unconditional Branch is a jump. Also calls and returns are jumps.
The following measurements of average CPI for individual instruction categories were made:

Instruction type	Average CPI (clock cycles)
All ALU instructions	1
Load–store	1.4
Taken Conditional branches	2.0
Not taken Conditional branches	1.5
Jumps	1.2

Assume that 60% of the conditional branches are taken and that all instructions in the Misc. category are ALU instructions. What is the CPI of the benchmark on this RISC machine?

Solution:
CPI = (0.26 * 1.4) + (0.09 * 1.4) + (0.14 * 1) + (0.13 * 1) + (0.16 * 0.6 * 2) + (0.16 * 0.4 * 1.5) + (0.01 * 1.2) + (0.02 * 1.2) + (0.04 * 1) + (0.09 * 1) + (0.06 * 1) = 1.27

5) Consider an un-pipelined processor in which branch instructions require 2 cycles, store instructions require 4 cycles, and all other instructions require 5 cycles.
Assuming a branch frequency of 12% and a store frequency of 10%, Calculate the average CPI.

Solution:
Average CPI = 0.12*2 + 0.1*4 + (1-0.12-0.1)*5 = 4.54

===

6) Consider the un-pipelined processor that has a 1.2 ns clock cycle and that it uses 3 cycles for ALU operations and branches and 4 cycles for memory operations. Assume that the relative frequencies of these operations are 40%, 20%, and 40%, respectively.
Suppose that due to clock skew and setup, pipelining the processor adds 0.1 ns of overhead to the clock.

How much speedup in the instruction execution rate will we gain from a pipeline? (Assume ideal pipeline without any stalls)

Solution:
Un-pipelined average instruction execution time = Clock cycle time x Average CPI
= 1.2 ns x [(40% + 20%) x 3 + 40% x 4]
= 1.2 ns x 3.4
= 4.08 ns
Pipelined average instruction execution time = Clock cycle time x Average CPI
= (1.2+0.1) ns x 1
= 1.3 ns
Speedup = (avg. inst. time un-pipelined)/ (avg. inst. time pipelined)
= 4.08/1.3 = **3.13 times**

===

7) Consider a 5-stage pipelined microprocessor that has an average stalls per instruction equal to 1.5.
Calculate the speedup of this processor over its un-pipelined version.
Solution: speedup = (pipeline depth) / (1+ stalls) = 5/(2.5) = 2

==

8) We begin with a computer implemented in single-cycle implementation. When the stages are split by functionality (pipelining), 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;
WB 1.5 ns.

Also, the pipeline register delay is 0.1 ns.

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

Solution: slowest stage among all + delay = 2 + 0.1 = 2.1 ns

b) If there is a stall every 4 instructions, what is the CPI of the new machine?

Solution:
CPI = 1 + pipeline stall cycle(s) per instruction (from the slides)
In this exercise, for every 4 instructions, 1 stall, this means we have 0.25 stalls per instruction
CPI = 1 +0.25 = 1.25.

c) Calculate the speedup of the pipelined over the un-pipelined:

solution:
speedup = (avg execution time un-pipelined) /(avg execution time pipelines) (from the slides)
The number of instructions is the same for both, so we will call it n
Speedup = (n x CPI $_{\text{unpipe}}$ x cycle time $_{\text{unpipe}}$) / (n x CPI $_{\text{pipe}}$ x cycle time $_{\text{pipe}}$)
= (n x 1 x 7) / (n x 1.25 x 2.1) = 2.67

==

9) 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 1 stall due to a data hazard every 5 Instructions
- The 12-stage pipeline experiences 3 stalls every 8 instructions due to data hazards.

A) **What is the speedup of the 12-stage pipeline over the 5-stage pipeline, taking into account only data hazards?**

Solution:
Calculate CPI of 5-stage = 1 + stall cycles per instruction = 1 + (1/5) = 1.2
Calculate CPI of 12-stage = 1 + stall cycles per instruction = 1 + (3/8) = 1.375

Speedup =
(n x CPI x Cycle Time) of 5 stage / ((n x CPI x Cycle Time) of 12 stage
= (n x 1.2 x 1) / (n x 1.375 x 0.6) = 1.45.
So the the 12-stage is 1.45 times faster than 5-stage

B) Assume branches constitute 20% of the instructions, and the mis-prediction rate for both machines is 5%.
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 in addition to the data hazards stalls in part A?**
Solution:

For the 5 stage: new CPI = old CPI + branch freq x branch penalty = 1.2 + (0.2x0.05) x 2 = 1.22

For the 12-stage: new CPI = old CPI + branch freq x branch penalty = 1.375 + (0.2x0.05) x 5 = 1.425

Note: 20% of the time is branch, and 5% of those is mispredicted so that's why we have 0.2x0.05 above.

C) **Calculate the speedup in part B <u>and</u> discuss the effects.**
New Speedup = n x 1.22 x1 / n x 1.425 x 0.6 = 1.42
Discussion: deeper pipelines have deeper penalties that's why the speedup less when we included branches.