Advanced Computer Architecture

Final Test

Date: 26 July 2019 **Time: 14:00 - 15:20**

Instructions:

- Duration: 80 minutes
- This test has 7 assignments.
- Write your answers to the point.
- If you need to make any assumptions, state them explicitly.
- Write your name and student number on any piece of paper you hand in.
- In total 50 points can be achieved.

Question No.	1	2	3	4	5	6	7	Total
Max points	8	10	6	10	8	4	4	50
Scored								

Name: Student Number:

1) Performance (8 points)

An application executes 20% integer, 50% floating point and 30% memory instructions.

a) (6 points) We want to improve overall performance by 25%, while only optimizing a **single** instruction type. Calculate what speedup of integer, floating point, and memory instructions is necessary to achieve this overall improvement. If this is not possible for some instruction types, explain why. If you do not have a calculator handy, it is okay to give an approximate result (as long as it is clear how the exact one needs to be calculated.)

b) (2 points) 2 billion (2×10^9) instructions with average CPI of 2.0 are executed on a processor running at frequency 4GHz. How much time does the execution take?

Name: Student Number:

2) Dependencies and pipelined execution (10 points)

Consider following MIPS assembly code

```
    ADD  R2, R3, R4
    SLL  R5, R2, #2
    LW   R6, 12(R5)
    SUB  R8, R6, R3
    BNE  R8, R1, L2
    ADDI R3, R3, #1
L2: ADD  R2, R3, R4
```

a) **(5 points)** Identify all control and true data dependencies in this code. Draw an arrow between 2 instructions if the second instruction depends on the first instruction and label the arrow with the type of dependency.
You do not need to indicate transitive dependencies.

```
    ADD  R2, R3, R4

    SLL  R5, R2, #2

    LW   R6, 12(R5)

    SUB  R8, R6, R3

    BNE  R8, R1, L2

    ADDI R3, R3, #1

L2: ADD  R2, R3, R4
```

Name: Student Number:

b) (5 points) Show how this code will be scheduled on a processor with following attributes:

- 5-stage canonical pipelined architecture
- support for hardware forwarding unit
- branch conditions and target address are evaluated in ID-stage

Complete the table below using following guidelines:

- Place IF, ID, EX, MEM, or WB in the box if the instruction in the 1st column is in that stage in the clock cycle in the first row.
- If a stall happens, place an X in the box
- If forwarding happens, draw an arrow between the corresponding pipeline stages

Instruction	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9	CC 10	CC 11	CC 12	CC 13
ADD R2,R3,R4													
SLL R5,R2, 2													
LW R6,12(R5)													
SUB R8,R6,R3													
BNE R8,R1,L2													

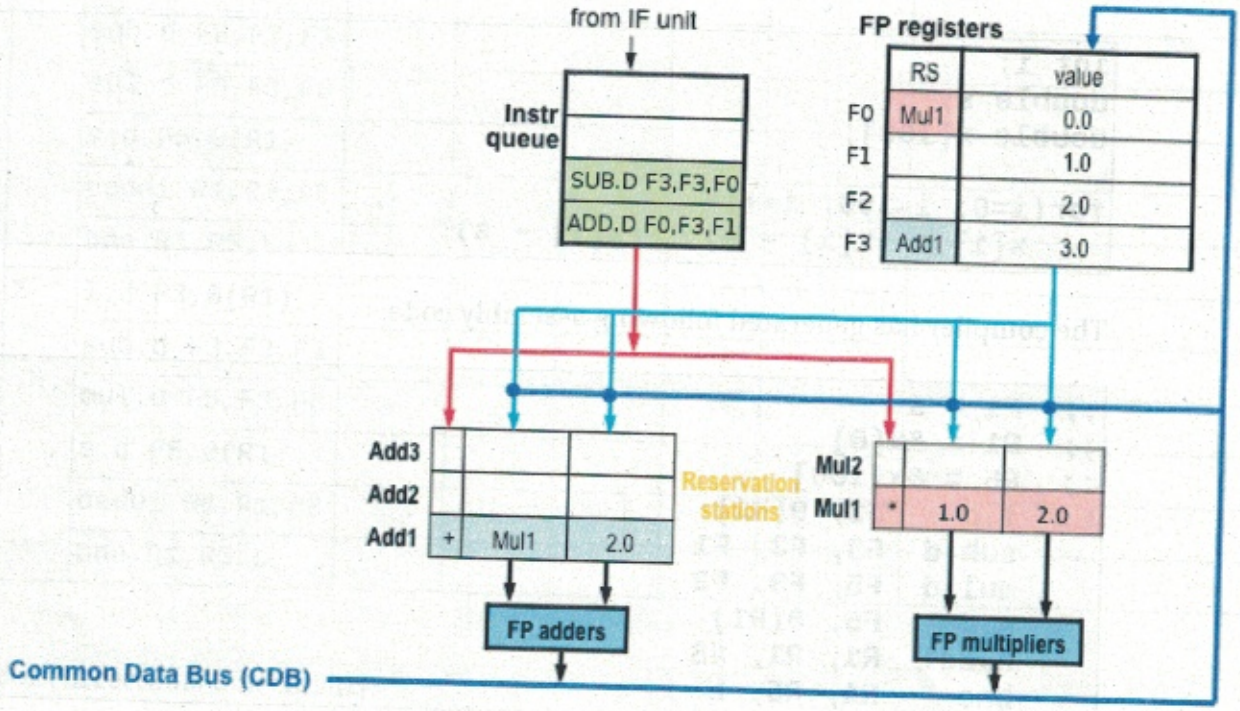
For corrections if necessary

Instruction	CC 1	CC 2	CC 3	CC 4	CC 5	CC 6	CC 7	CC 8	CC 9	CC 10	CC 11	CC 12	CC 13
ADD R2,R3,R4													
SLL R5,R2, 2													
LW R6,12(R5)													
SUB R8,R6,R3													
BNE R8,R1,L2													

Name: Student Number:

3) Tomasulo's algorithm (6 points)

The following figure shows a snapshot of Tomasulo's algorithm.



In this figure a reservation station with the fields

op	RS1	RS2	Val1	Val2	Imm/addr	busy

is abbreviated as

op	RS1 or Val1	RS2 or Val2

Furthermore, the imm/addr field is not used and a reservation station is busy if it is not blank.

Now the two instructions at the head of the instruction queue (first `ADD.D` and then `SUB.D`) are issued. Show the contents of the different data structures after the instructions have been issued. You can draw the new state into the above figure. You may cross out and replace existing values.

Reservation stations should be filled from the bottom to the top. When assigning a reservation station, pick the bottom-most empty slot.

Name: Student Number:

4) Dynamically scheduled superscalar execution and speculation (10points)

Consider following C code:

```
int i;
double s;
double x[100];
…
for(i=0; i<100; i++)
    x[i] = (x[i] - s) * (x[i] - s);
```

The compiler has generated following assembly code:

```
;;  F1 = s
;;  R1 = &x[0]
;;  R5 = &x[100]
L:  l.d    F3, 0(R1)
    sub.d  F3, F3, F1
    mul.d  F5, F3, F3
    s.d    F5, 0(R1)
    daddi  R1, R1, #8
    bne    R1, R5, L
```

This code is executed on a dynamically scheduled, dual-issue superscalar processor with following attributes:

- 1 integer FU (used for ALU operations and effective address calculations)
- Separate FU to evaluate branch conditions
- Separate pipelined FP adder
- Separate pipelined FP multiplier
- Issue and Write Result take 1 clock cycle (cc) each
- The number of instructions that can commit every cycles is equal to the issue width
- Perfect branch prediction with speculation
- latency between producing and consuming instruction is 1 cc for integer ALU, 2 for load, 3 for FP adder, 4 for FP mul

Complete the table below (on the next page) indicating:

- the clock cycle (cc) at which the instruction in the 2nd column is issued
- the cc at which it starts executing
- the cc at which the memory access take place (if applicable)
- the cc at which the CDB is written
- the cc at which the instruction commits

Write a brief comment in the last column that explains your choices.

Name: Student Number:

Iter. #	Instruction	Issues at	Starts Exec. at	Mem. Access at	Write CDB at	Commits at	Comment
1	l.d F3,0(R1)						
	sub.d F3,F3,F1						
	mul.d F5,F3,F3						
	s.d F5,0(R1)						
	daddi R1,R1,#8						
	bne R1,R5,L						
2	l.d F3,0(R1)						
	sub.d F3,F3,F1						
	mul.d F5,F3,F3						
	s.d F5,0(R1)						
	daddi R1,R1,#8						
	bne R1,R5,L						

For corrections if necessary:

Iter. #	Instruction	Issues at	Starts Exec. at	Mem. Access at	Write CDB at	Commits at	Comment
1	l.d F3,0(R1)						
	sub.d F3,F3,F1						
	mul.d F5,F3,F3						
	s.d F5,0(R1)						
	daddi R1,R1,#8						
	bne R1,R5,L						
2	l.d F3,0(R1)						
	sub.d F3,F3,F1						
	mul.d F5,F3,F3						
	s.d F5,0(R1)						
	daddi R1,R1,#8						
	bne R1,R5,L						

Name: Student Number:

5) Multicore cache coherence (8 points)

The following table illustrates a consecutive sequence of CPU events. For example, at time t_1 Thread 1 (running on core 1) reads address A and at time $t_2 > t_1$ Thread 1 writes A.

time	Thread 1	Thread 2	Thread 3
t_1	Read A		
t_2	Write A		
t_3		Read A	
t_4			Write A
t_5			Write A

This sequence is executed on a cache-coherent SMP architecture that utilizes the MSI snooping protocol.

a) (6 points) complete the following table that indicates the state of the cache block containing A in each of the 3 private caches after every time step. Initially, at time t_0 before t_1, the cache block is invalid (not present) in any of the private caches. Also indicate the transaction that takes place on the bus, which can be read-miss, write-miss, or invalidate.

time	Bus transaction	state in cache 1	state in cache 2	state in cache 3
t_0		I	I	I
t_1				
t_2				
t_3				
t_4				
t_5				

b) (2 points) Which bus transaction(s) (if any) would be saved if the SMP architecture would employ the MESI protocol instead of the MSI protocol?

Name: Student Number:

6) Cache performance (4 points)

A 32bit processor has an L1 direct-mapped cache with the following parameters:

- Size = 16KB (1K = 1024)
- Block size = 64 bytes

a) (2 points) The processor generates the following byte addresses:
16384, 16408, 2048, 2064
For each of these byte addresses, determine the **index** and indicate whether it will yield a **hit** or a **miss**. (assume that we start from an empty cache)

c) (2 points) With 4% miss-rate and assuming cache hit-time is 1 clock-cycle and miss-penalty is 100 clock-cycles, what is the overall average memory access time?

Name: Student Number:

7) Branch Prediction (4 points)

a) (2 points) Explain these branch prediction schemes:
1-bit Branch Prediction Buffer' (or 'Branch History Table')
2-bit Branch Prediction Buffer' (or 'Bimodal Predictor')

b) (2 points) Draw the state diagram of a 2-bit bimodal predictor (branch history table), showing transitions between different states with appropriate labels.