

PLEASE RETURN THIS SHEET ALONG WITH ALL
THE SHEETS YOU WERE GIVEN

SURNAME _____

FIRST NAME _____

- 1) (POINTS 24/40) Consider the following snippet of code running on 4-ways out-of-order superscalar processor. Initially, all registers contain zero.

```
lab1:  LW   R2, 0(R1)
        ADDI R2, R2, 1
        MUL  R4, R2, R2
        SW   R4, 0(R1)
        ADDI R1, R1, 4
        BNE  R2, R0, lab1
```

Working hypothesis:

- * the fetch, decode and commit stages are 4 instructions wide
- * the instruction window has 18 slots
- * we have 8 physical registers in the free pool
- * the reorder buffer has unlimited size
- * the integer multiplier has 4 stages
- * the load/store queues have 3 slots each and a common effective-address calculation unit
- * there are 4 ALUs for arithmetic and logic operations and for branching
- * an ALU performs its operation in the same cycle when the operation is issued
- * reads require 1 clock cycle (after the addressing phase)
- * the register file has 4 input- and 4 output-ports
- * there are 9 logical registers (including R0 which is hardwired to 0)
- * the store operation leaves the issue stage as it is inserted in the store queue

In order to calculate the total cycles needed to execute 3 iterations of the above loop on such machine, complete the following chart until the end of the third iteration of the code fragment above, including the renamed stream the precise evolution of the free pool of the physical registers (the register map), the Instruction Window, the Reorder Buffer (ROB) and the Load Queue (LQ) and Store Queue (SQ). Calculate the total cycles needed to execute three iterations of the above loop on such machine.

- 2) (POINTS 8/40) Consider a bus-based multicore that support a) MSI or b) MESI cache coherence protocol. The cost of a read/write operation is 1 cycle, the cost of a BusRd (or BusRdX) transaction is 90 cycles; all caches are write-back, write-allocate and initially empty. For the MSI protocol the BusUpgr is not used. The cost of a BusUpgr is 60 cycles. Evaluate the total cost of executing the following streams in the cases a and b:

Stream1: R1, W1, R1, W1, R2, W2, R2, W2, R3, W3, R3, W3

Stream2: R1, R2, R3, W1, W2, W3, R1, R2, R3, W3, W1

Stream3: R1, R2, R3, R3, W1, W1, W1, W1, W2, W3

- 3) (POINTS 8/40) The following code runs on a multicore that does not impose any ordering of the memory operations. However, the machine provides a FENCE instruction that, if inserted in a code, prevents issuing memory operations that come after the fence before the memory operations that are before the fence are globally performed. For simplicity, assume that lock and unlock instructions behave like acquire and release respectively by performing a single memory operation.
- a) Insert FENCE instructions as appropriate to ensure sequential consistency;
 - b) Insert FENCE instructions as appropriate to ensure processor consistency;
 - c) Insert FENCE instructions as appropriate to ensure weak ordering;

Lock1
Load A
Store B
Unlock1
Load C
Store D
Lock2
Load E
Store F
Unlock2

(REVISED 26/10/2018)

EXERCIZE 1)

```

=====
PHYSICAL REGS: 1 2 3 4 5 6 7 8
                * * *
qi: 1 1 0 0 1 0 1 1
vi: 00 04 01 08 04 01 01 01
=====
REG.FILE: Ri: 1 2 3 4 5 6 7 8
           Pi: 4 3 - 6 - - - -
           Qi: 0 0 0 0 0 0 0 0
           Vi: 00000004 00000000 00000000 00000001 00000000 00000000 00000000 00000000
=====
STAGES:          F D P I X W C RENAMED-STR  INSTRUCTION-WINDOW  REORDER-BUFFER  A M L S B F X
TOTAL SLOTS:    4 4 18 4 12 4 4 8  18  99  4 1 1 0 1 4 1
BUSY SLOTS:     0 0 0 0 0 1 0 3  0  0  0 0 0 0 0 0
STALLS:         0 11 0 21 0 0 12 11  0  0  0 0 0 2 0 0 0
=====
PC INSTRUCTION  F D P I X W C Pi,Pj Pk P1  IW#  OPCODE Pi Pj Pk I/P1  Cj Ck Cl  ROB# PC Ri  oPi x s c  +-----+
000] LW R2,0(R1) 0 1 2 3 4 6 7 P2,0(P1)  ---- LW P2 P1 - 0  2 - -  ---- 000 R2 - 0 0 1  |LQ(0) |
001] ADDI R2,R2,1 0 1 2 6 6 7 8 P3,P2,1  ---- ADDI P3 P2 - 1  6 - -  ---- 001 R2 P2 0 0 1  |PC OP Pi EFAD Ci|
002] MUL R4,R2,R2 0 1 2 7 7 12 13 P4,P3,P3  ---- MUL P4 P3 P3 - 7  7 - -  ---- 002 R4 - 0 0 1  |---- LW P2 0000 6|
003] SW R4,0(R1) 0 1 2 4 5 12 13 ,P0(P1)<-P4  ---- SW - P4 P1 0 - 2 - -  ---- 003 - - 1 0 1  |---- LW P6 0004 8|
004] ADDI R1,R1,4 1 2 3 4 4 5 13 P5,P1,4  ---- ADDI P5 P1 - 4  3 - -  ---- 004 R1 P1 0 0 1  |---- LW P1 0004 18|
005] BNE R2,R0,-6 1 2 3 4 4 5 13 ,P3,P0,-6  ---- BNE - P3 P0 -6 - 3 - -  ---- 005 - - 0 0 1  +-----+
006] LW R2,0(R1) 2 3 4 5 6 8 14 P6,0(P5)  ---- LW P6 P5 - 0  5 - -  ---- 000 R2 P3 0 0 1  +-----+
007] ADDI R2,R2,1 2 3 4 8 8 9 14 P7,P6,1  ---- ADDI P7 P6 - 1  8 - -  ---- 001 R2 P6 0 0 1  +-----+
008] MUL R4,R2,R2 2 3 4 9 9 14 15 P8,P7,P7  ---- MUL P8 P7 P7 - 9  9 - -  ---- 002 R4 P4 0 0 1  |SQ(0) |
009] SW R4,0(R1) 2 3 4 6 7 14 15 ,P0(P5)<-P8  ---- SW - P8 P5 0 - 5 - -  ---- 003 - - 1 0 1  |PC OP Pi EFAD Cl|
010] ADDI R1,R1,4 3 8 9 10 10 11 15 P2,P5,4  ---- ADDI P2 P5 - 4  9 - -  ---- 004 R1 P5 0 0 1  |---- SW P0 0000 12|
011] BNE R2,R0,-6 3 8 9 10 10 11 15 ,P7,P0,-6  ---- BNE - P7 P0 -6 - 9 - -  ---- 005 - - 0 0 1  |---- SW P0 0004 14|
012] LW R2,0(R1) 4 13 14 15 16 18 19 P1,0(P2)  ---- LW P1 P2 - 0  14 - -  ---- 000 R2 P7 0 0 1  |---- SW P0 0004 24|
013] ADDI R2,R2,1 4 14 15 18 18 19 20 P3,P1,1  ---- ADDI P3 P1 - 1  18 - -  ---- 001 R2 P1 0 0 1  +-----+
014] MUL R4,R2,R2 8 14 15 19 19 24 25 P6,P3,P3  ---- MUL P6 P3 P3 - 19 19 - -  ---- 002 R4 P8 0 0 1  +-----+
015] SW R4,0(R1) 8 14 15 16 17 24 25 ,P0(P2)<-P6  ---- SW - P6 P2 0 - 15 - -  ---- 003 - - 1 0 1  +-----+
016] ADDI R1,R1,4 13 15 16 17 17 18 25 P4,P2,4  ---- ADDI P4 P2 - 4  16 - -  ---- 004 R1 P2 0 0 1  +-----+
017] BNE R2,R0,-6 14 15 16 17 17 18 25 ,P3,P0,-6  ---- BNE - P3 P0 -6 - 16 - -  ---- 005 - - 0 0 1  +-----+
    
```

Therefore 26 cycles are needed for this configuration.

EXERCIZE 2)

2a) stream-1 MSI

Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
PrRd1	S	I	I	BusRd	Mem	90
PrWr1	M	I	I	BusRdX	Mem	90
PrRd1	M	I	I	-	-	1
PrWr1	M	I	I	-	-	1
PrRd2	S	S	I	BusRd/Flush	C1	90
PrWr2	I	M	I	BusRdX	Mem	90
PrRd2	I	M	I	-	-	1
PrWr2	I	M	I	-	-	1
PrRd3	I	S	S	BusRd/Flush	C2	90
PrWr3	I	I	M	BusRdX	Mem	90
PrRd3	I	I	M	-	-	1
PrWr3	I	I	M	-	-	1
TOTAL						546

2b) stream-1 MESI

Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
PrRd1	E	I	I	BusRd (/S)	Mem	90
PrWr1	M	I	I	-	-	1
PrRd1	M	I	I	-	-	1
PrWr1	M	I	I	-	-	1
PrRd2	S	S	I	BusRd(S) /Flush	C1	90
PrWr2	I	M	I	BusUpgr	-	60
PrRd2	I	M	I	-	-	1
PrWr2	I	M	I	-	-	1
PrRd3	I	S	S	BusRd(S) /Flush	C2	90
PrWr3	I	I	M	BusUpgr	-	60
PrRd3	I	I	M	-	-	1
PrWr3	I	I	M	-	-	1
TOTAL						397

2c) stream-2 MSI

Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
PrRd1	S	I	I	BusRd	Mem	90
PrRd2	S	S	I	BusRd	Mem	90
PrRd3	S	S	S	BusRd	Mem	90
PrWr1	M	I	I	BusRdX	Mem	90
PrWr2	I	M	I	BusRdX/Flush	C1	90
PrWr3	I	I	M	BusRdX/Flush	C2	90
PrRd1	S	I	S	BusRd/Flush	C3	90
PrRd2	S	S	S	BusRd	Mem	90
PrRd3	S	S	S	-	-	1
PrWr3	I	I	M	BusRdX	Mem	90
PrWr1	M	I	I	BusRdX/Flush	C3	90
TOTAL						901

2d) stream-2 MESI

Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
PrRd1	E	I	I	BusRd (/S)	Mem	90
PrRd2	S	S	I	BusRd(S)	C1	90
PrRd3	S	S	S	BusRd(S)	Mem	90
PrWr1	M	I	I	BusUpgr	-	60
PrWr2	I	M	I	BusRdX/Flush	C1	90
PrWr3	I	I	M	BusRdX/Flush	C2	90
PrRd1	S	I	S	BusRd(S) /Flush	C3	90
PrRd2	S	S	S	BusRd(S) /Flush	C3	90
PrRd3	S	S	S	-	-	1
PrWr3	I	I	M	BusUpgr	-	60
PrWr1	M	I	I	BusRdX/Flush	C3	90
TOTAL						841

(REVISED 26/10/2018)

2e) stream-3 MSI

Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
PrRd1	S	I	I	BusRd	Mem	90
PrRd2	S	S	I	BusRd	Mem	90
PrRd3	S	S	S	BusRd	Mem	90
PrRd3	S	S	S	-	-	1
PrWr1	M	I	I	BusRdX	Mem	90
PrWr1	M	I	I	-	-	1
PrWr1	M	I	I	-	-	1
PrWr1	M	I	I	-	-	1
PrWr2	I	M	I	BusRdX/Flush	C1	90
PrWr3	I	I	M	BusRdX/Flush	C2	90
TOTAL						544

2f) stream-3 MSI

Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
PrRd1	E	I	I	BusRd (/S)	Mem	90
PrRd2	S	S	I	BusRd (S)	C1	90
PrRd3	S	S	S	BusRd (S)	C1/C2	90
PrRd3	S	S	S	-	-	1
PrWr1	M	I	I	BusUpgr	-	60
PrWr1	M	I	I	-	-	1
PrWr1	M	I	I	-	-	1
PrWr1	M	I	I	-	-	1
PrWr2	I	M	I	BusRdX/Flush	C1	90
PrWr3	I	I	M	BusRdX/Flush	C2	90
TOTAL						514

EXERCIZE 3)

3a) SEQUENTIAL CONSISTENCY.

Every memory operation is followed by a FENCE instruction:

Lock1
 FENCE
 Load A
 FENCE
 Store B
 FENCE
 Unlock1
 FENCE
 Load C
 FENCE
 Store D
 FENCE
 Lock2
 FENCE
 Load E
 FENCE
 Store F
 FENCE
 Unlock2
 FENCE

3a) PROCESSOR CONSISTENCY.

Every memory operation is followed by a FENCE instruction, except the UNLOCK since it can only succeed on the same core where the LOCK succeeded:

Lock1
 FENCE
 Load A
 FENCE
 Store B
 FENCE
 Unlock1

 Load C
 FENCE
 Store D
 FENCE
 Lock2
 FENCE
 Load E
 FENCE
 Store F
 FENCE
 Unlock2

3a) WEAK ORDERING.

We only need to enforce global consistency at the entrance and exit of locked regions and before loads outside locked regions:

Lock1
 FENCE
 Load A

 Store B
 FENCE
 Unlock1
 FENCE
 Load C

 Store D
 FENCE
 Lock2
 FENCE
 Load E

 Store F
 FENCE
 Unlock2