HIGH PERFORMANCE COMPUTER ARCHITECTURE 20-11-2025 MATRICULATION NO._____

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

SURNAME_____

FIRST NAME_____

Consider a bus-based multicore that supports a cache-coherence protocol called MEI, what was used in the PowerPC-755. Compared to the well-known MESI protocol, the MEI protocol does not have the S state. There is no need for a BusUpgr transaction, only Flush*, BusRd and BusRdX may happen.

1a) [Points 4/30] Draw the diagram of the MEI protocol according to the above description.

1b) [Points 18/30] Assuming a cost of 1cc (1 clock-cycle) for read/write operations, 90cc for BusRd or BusRdx transactions, and 20cc for Flush*. Evaluate the total cost (in clock-cycles) for the following streams:

stream-1 MEI

Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
PrRd1						
PrWr1						
PrRd1						
PrWr1						
PrRd2						
PrWr2						
PrRd2						
PrWr2						
PrRd3						
PrWr3						
PrRd3						
PrWr3						
TOTAL						

stream-2 MEI

Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
PrRd1						
PrRd2						
PrRd3						
PrWr1						
PrWr2						
PrWr3						
PrRd1						
PrRd2						
PrRd3						
PrWr3						
PrWr1						
TOTAL						

stream-3 MEI

Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
PrRd1						
PrRd2						
PrRd3						
PrRd3						
PrWr1						
PrWr1						
PrWr1						
PrWr1						
PrWr2						
PrWr3						
TOTAL						

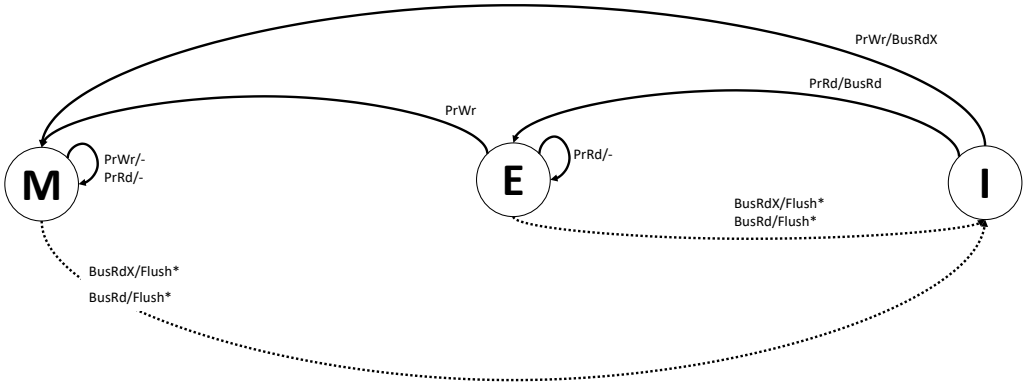
2) Let's consider a cache-coherent multiprocessor system, in which each processor executes its code in program order (no reordering in source), but the hardware may reorder memory operations according to the consistency model. The variable x, y, z, are initialized to 0. We run the following program **once**:

P0: `x = 1;` `r1 = y;`	**P1:** `y = 1;` `r2 = x;`

2a) [4/30] Under **Sequential Consistency (SC)**: i) list all possible combinations of final values for `(r1, r2)` and ii) for each combination, say whether it is **allowed or forbidden** under SC.

2b) [4/30] Under a **TSO model** (store buffering but loads cannot be reordered with older loads, and stores become visible to others later): i) is the outcome `(r1=0, r2=0)` possible? Explain informally how store buffers make this outcome possible even if each core "respects program order" locally; ii) explain why coherence is **not** violated even if `(r1=0, r2=0)` occurs.

1)



stream-1 MOSI	Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
	PrRd1	E	I	I	BusRd	Mem	90
	PrWr1	M			-		1
	PrRd1	M			-		1
	PrWr1	M			-		1
	PrRd2	I	E		BusRd/Flush*	C1	90+20
	PrWr2	I	M		-		1
	PrRd2		M		-		1
	PrWr2		M		-		1
	PrRd3		I	E	BusRd/Flush*	C2	90+20
	PrWr3		I	M	-		1
	PrRd3			M	-		1
	PrWr3			M	-		1
	TOTAL						319

stream-2 MOSI	Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
	PrRd1	E	I	I	BusRd	Mem	90
	PrRd2	I	E	I	BusRd/Flush*	C1	90+20
	PrRd3	I	I	E	BusRd/Flush*	C2	90+20
	PrWr1	M	I	I	BusRdX/Flush*	C3	90+20
	PrWr2	I	M	I	BusRdX/Flush*	C1	90+20
	PrWr3		I	M	BusRdX/Flush*	C2	90+20
	PrRd1	E	I	I	BusRd/Flush*	C3	90+20
	PrRd2	I	E	I	BusRd/Flush*	C1	90+20
	PrRd3	I	I	E	BusRd/Flush*	C2	90+20
	PrWr3	I	I	M	-		1
	PrWr1	E	I	I	BusRdX/Flush*	C3	90+20
	TOTAL						1081

stream-3 MOSI	Core Operation	C1	C2	C3	Bus Transaction	Data from	Cycles
	PrRd1	E	I	I	BusRd	Mem	90
	PrRd2	I	E	I	BusRd/Flush*	C1	90+20
	PrRd3	I	I	E	BusRd/Flush*	C2	90+20
	PrRd3	I	I	E	-		1
	PrWr1	M	I	I	BusRdX/Flush*	C3	90+20
	PrWr1	M			-		1
	PrWr1	M			-		1
	PrWr1	M			-		1
	PrWr2	I	M		BusRd/Flush*	C1	90+20
	PrWr3		I	M	BusRd/Flush*	C2	90+20
	TOTAL						644

2a) **Under Sequential Consistency (SC)**

- `(0,0)` → **Forbidden** under SC
- `(0,1), (1,0), (1,1)` → **Allowed**

SC requires a **single global interleaving** that respects per-thread program order. `(0,0)` would require each read to happen "before" the other thread's write, creating a cycle in the implied order; this cannot be linearized.

2b) **Under TSO** (with store buffering) (0,0) is possible:
• Each core first does a store, which goes into a per-core store buffer.
• The subsequent load can bypass the buffered store and read from memory, which hasn't yet seen the other core's store.
So. P0: x = 1 goes into P0's buffer (not yet visible globally). Then r1 = y reads y = 0 from memory. P1: y = 1 into its buffer. Then r2 = x reads x = 0 from memory. Later, the buffers are drained, but the reads have already seen 0. Locally, each core respects its own program order, but the visibility to other cores is delayed. Coherence is not violeted since it is per single location (e.g., just x, just y): For x: all cores agree that there is a single total order of writes to x (x=0 then x=1), and reads observe values consistent with that order (they may read either 0 or 1 depending on their position). For y: same story. In (0,0), both cores read the initial values before the other's write becomes visible. That is coherent: for x and for y there is a consistent order; what's "strange" is the cross-variable pattern, which is about consistency, not coherence.