INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR
Long test 2 - 2021

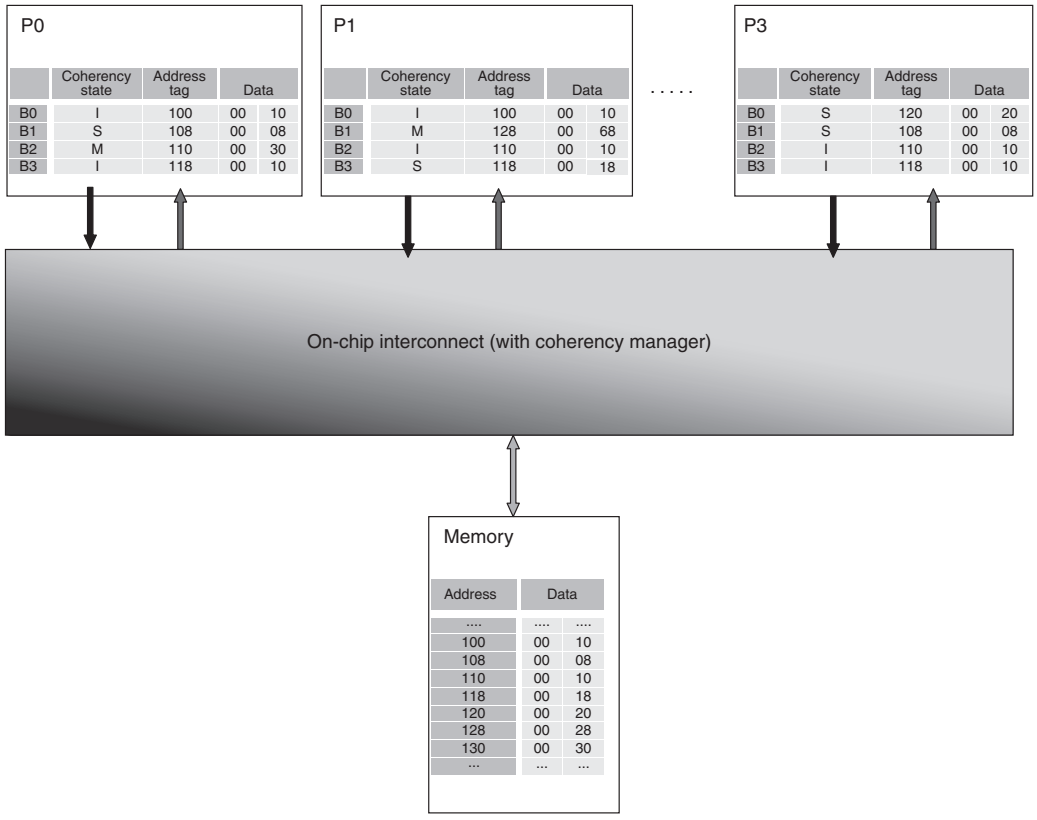
Date: 14-04-21 **Duration 1 hr 10 mins + 15 mins for submission Total Marks = 40**
Subject No : CS60003 **HIGH PERFORMANCE COMPUTER ARCHITECTURE**
Department/Centre/School : Computer Science and Engineering
Answer all questions. In case of reasonable doubt, make assumptions and state them upfront. Marks will be deducted for claims without proper reasoning. Write your answers (with all analysis, justification, calculation etc) in loose sheets, scan them and upload in moodle within the allotted time

1. Consider a 4-issue in-order pipeline executing the following loop. Considering that a loop unrolling factor

Label	Instruction	Operands		
Loop:	LW	R2	0(R1)	
	ADD	R2	R2	R3
	SW	R2	0(R1)	
	ADDI	R1	R1	-4
	BNE	R1	R5	Loop

of 3 is chosen and suitable instruction scheduling is done by the compiler, what is the effective CPI ? Assume each instruction takes one cycle to execute. **[4]**

SOL: 3/11=0.2727 (same as the case with 2 taught in class due to the restriction of 4 issue, this can shown using the instruction issue figure)



2. Consider the multi processor system illustrated in given figure. Each processor has a single, private cache with coherence maintained using invalidation based snooping coherence protocol MSI. Each cache is direct-mapped, with four blocks each holding two words. To simplify the illustration, the cache-address tag contains the full address, and each word shows only two hex characters, with the least significant word on the right. Assume that in every block, all writes only happen in the word in the right. The coherence states are denoted M, S, and I (Modified, Shared, and Invalid). Assume the initial cache and memory state as illustrated in the figure. The instruction format is: P#: $\langle op \rangle \langle address \rangle \leftarrow \langle value \rangle$ where P# designates the CPU (e.g., P0), $\langle op \rangle$ is the CPU operation (e.g., read or write), $\langle address \rangle$ denotes the memory address, and $\langle value \rangle$ indicates the new word to be assigned on a write operation.

   (a) Treat each action below as independently applied to the initial state as given in figure. What is the resulting state (i.e., coherence state, tags, and data) of the caches and memory after the given action ? For each read/write, answer only the blocks that change; for example, P0.B0: (I, 120, 00 01) indicates that CPU P0's block B0 has the final state of I, tag of 120, and data words 00 and 01.

      1. P0: read 120
      2. P0: write 120 $\leftarrow$ 29
      3. P3: write 120 $\leftarrow$ 29
      4. P0: write 108 $\leftarrow$ 76

   [8]

   SOL:
   P0.B0: (S, 120, 0020)
   P0.B0: (M, 120, 0029); P3.B0: (I, 120, 0020)
   P3.B0: (M, 120, 0029)
   P0.B1: (M, 108, 0076) ; P3.B1: (I, 108, 0008)

   (b) For the multiprocessor illustrated in figure, consider the execution of a sequence of operations on a single CPU where

      1. CPU read and write hits generate no stall cycles.
      2. CPU read and write misses generate 150 and 60 stall cycles if satisfied by memory and cache, respectively.
      3. CPU write hits that generate an invalidate incur 22 stall cycles.
      4. A write-back of a block, due to either a conflict or another processor's request to a M block, incurs an additional 15 stall cycles.

   As an example, consider the following sequence of operations assuming the initial cache state in Figure. Also, assume that the second operation begins after the first completes (even though they are on different processors).
   P1: read 110 ; P3: read 110
   Sample solution : The first read generates 75 stall cycles because the read is satisfied by P0's cache. P1 stalls for 60 cycles while it waits for the block, and then P0 stalls for additional 15 cycles while it writes the block back to memory in response to P1's request. The second read by P3 generates 150 stall cycles because its miss is satisfied by memory, and this sequence generates a total of 225 stall cycles.
   Assuming the initial state of the figure, in each of the following *sequences* of operations, how many stall cycles are generated?

      1. P0: read 120 ; P0: read 128 ; P0: read 130
      2. P0: read 100 ; P0: write 108 $\leftarrow$ 29 ; P0: write 130 $\leftarrow$ 92
      3. P1: read 120 ; P1: read 128 ; P1: read 130
      4. P1: read 100 ; P1: write 108 $\leftarrow$ 29 ; P1: write 130 $\leftarrow$ 92

   For securing marks, write (very brief) explanations of your stall calculations in all cases.

[8]

SOL:
Q1:
P0: read 120, Read miss, satisfied by memory
P0: read 128, Read miss, satisfied by P1's cache
P0: read 130, Read miss, satisfied by memory, write back 110
150 + 60 + 15 + 150 + 15 = 390 stall cycles
Q2:
P0: read 100, Read miss, satisfied by memory
P0: write 108 ← 29, Write hit, sends invalidate
P0: write 130 ← 92, Write miss, satisfied by memory, write back 110
150 + 22 + 15 + 150 = 337 stall cycles
Q3:
P1: read 120, Read miss, satisfied by memory
P1: read 128, Read hit
P1: read 130, Read miss, satisfied by memory
150 + 0 + 150 = 300 stall cycles
Q4:
P1: read 100, Read miss, satisfied by memory
P1: write 108 ← 29, Write miss, satisfied by memory, write back 128
P1: write 130 ← 92, Write miss, satisfied by memory
150 + 150 + 15 + 150 = 465 stall cycles

3. A byte-addressable system has a 3-level hierarchical page table with 20-bit virtual addresses. The page size is 32 bytes and page table entry size is 2 bytes. Each of the inner-level and middle-level page tables must fit into single page. However, the outermost page table does not have such restriction.

   (a) Provide the breakdown (bit-ranges) of the 20-bit virtual address required to carry out a virtual-to-physical translation as per the following table: **[2]**

   SOL:

Outermost Page Table Index	Middle Level Page Table Index	Inner Level Page Table Index	Page Offset
19-13 (7 bits)	**12-9** (4 bits)	**8-5** (4 bits)	**4-0** (5 bits)

   (b) Assume instruction fetches will never lead to page faults, and there is no TLB in the system. Assume that there is enough physical memory to never need a page replacement. Consider the following code snippet of a program:

```
typedef struct blob{
   int data[3];  // 12 Bytes
   bool valid;   // 1 Byte
}blob;

blob arr[4096];

for(int i=0; i < 4096; i++){
    arr[i].valid = true;
}
```

   At the program start, all of the outermost page-tables are in memory. None of the inner and middle level page tables, or the data pages (containing `arr`) are in memory. The array "arr" starts at virtual address 0x00000, has been allocated, and is on disk. Assume due to padding the blob struct

will occupy 16 bytes of memory. Calculate the total number of disk accesses incurred due to accesses to "arr" in the for loop. **[6]**

SOL: Recognize that due to padding the blob struct will occupy 16 bytes of memory. As a result, each data page (32 bytes) can contain two blob structs. Further, since PTEs are 2 bytes, each page table will contain 16 PTEs. We are told that only the outermost table is in memory at the beginning of execution. The array has been allocated on disk. There are 4096 elements in the array. Each entry in the outermost table accounts for $16 \times 16 \times 2 = 512$ elements. of the array. Thus, the program would use $4096/512 = 8$ outermost table entries to walk the array. Each new outermost table entry will incur a disk access. Each new middle and innermost level page tables would themselves incur disk accesses. Thus the total number of disk accesses is given by: Disk accesses $= 8 \times (1 + 16 + 16 \times 16) = 2184$.

4. A system having physical address space of 512 B uses a single-level virtual memory system. The virtual address space is 2KB and the page size is 8 B. Assume that the page tables are always in physical memory.

   (a) How many bits of each virtual address is the virtual page number? How many bits of each physical address is the physical frame number? **[2]**

   SOL: $\log(2048/8) = 8$, $\log(512/8) = 6$.

   (b) Assume there is a 128 B *write-through* virtually-indexed physically-tagged cache (VIPT). In other words, the cache is indexed using virtual address and the tag comparison is done using physical address. Also assume the cache is direct-mapped and the block size is 2 B. Based on the above assumptions, answer the following question:

   1. How many bits of a virtual address are used to index into the cache? **[1]**
      SOL: log(cache_size/block_size) = 6
   2. What is the size of the memory required to store the tags (in bytes)? **[3]**
      SOL: Per cache block we have tag + valid bit (no dirty bit for write-through cache).
      Tag size = 6 + 1 (valid bit)
      Tag store = 64 (blocks) * 7 = 56 bytes

   (c) Now suppose there are two processes (process 0 and process 1) running simultaneously on the system. These processes share some portion of the physical memory. Some of the virtual page-physical frame mappings of each process are given below:

PROCESS 0	
Virtual Page	Physical Frame
Page 0	Frame 0
Page 3	Frame 7
Page 7	Frame 1
Page 15	Frame 3

PROCESS 1	
Virtual Page	Physical Frame
Page 0	Frame 4
Page 1	Frame 5
Page 7	Frame 3
Page 11	Frame 2

   1. Give a complete physical address of the first byte whose data can exist in two different locations in the cache. **[1]**
      SOL: Any byte in frame 3 is seen as 2 different virtual pages from processes 0, 1. Physical address of the first byte in frame 3 = 011000.
   2. Give the indexes of those two different locations in the cache. **[2]**
      SOL: Corresponding virtual address from process 0 (Page 15) = 00001111000 (index in bold)
      Corresponding virtual address from process 1 (Page 7) = 00000111000 (index in bold)
   3. In order to avoid a situation where the same physical address is stored in two different locations, a common strategy is to increase the associativity of the VIPT cache (convert direct mapped to set-associative). What is the minimum associativity required? **[3]**
      SOL: 4 bits of the index come from the virtual page number. All of these might change during translation. To eliminate all these bits from the index, we should have at least 2^4 ways in the cache. Minimum associativity = 16 ways.