# INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

End-Spring Semester Examination 2022-23

Date of Examination: 27.04.2023 Session: (FN/AN) AN Duration: 3 hrs. Full Marks: 100

Subject No.: CS60003 Subject: High Performance Computer Architecture

Department: Computer Science and Engineering

Specific charts, graph paper, log book etc., required: None

Important Instructions:

- Use of non-programmable electronic calculators is allowed.
- 2 Year M.Tech students need to answer only Section B and all other students need to answer Section A
- No clarification to any of the questions shall be provided. In case you have any queries, you can make suitable assumptions, but please write down your assumptions clearly.
- All answers should be brief and concise. Lengthy and irrelevant answers will be penalized.

## Section A (To be answered by UG and RS Students, Not 2-year MTech Students)

**(Answer All Questions)**

1. (a) Consider a **64-bit**, byte-addressable system with **2GB** of **DRAM** installed. Each page size is **4KB**, and the system uses a **two-level page table**. For each address, the **12 bits** most-significant are used to index into the **first-level page table**, while the rest (other than the page offset) index into the **second-level page tables**. All page table entries occupy **4 bytes**.
   a. How many entries are in the **first-level page table**? **[2]**
   b. How many bits are used to index into the **second-level page table**? **[2]**
   c. How many physical pages does it take to store the **first-level page table**? **[2]**

   (b) A new company has proposed a number of different **cache layouts** for their system and you've been asked to come in and calculate the overhead for each of the different caches. Their system uses a cache with **2KB** of **data storage** capable of addressing **2GB** of **byte-addressable** memory. Stores will be handled by write-back and allocate-on-write policies. That means all subsequent writes will be done in the cache. You must show the work for your calculations to receive credit.
   a. The first design is a **fully-associative cache** with a block size of **16 bytes**, how many bits of **tags** and **LRU** bits (as required) are needed per entry? **[2]**
   b. Their next design utilizes a direct-mapped cache with **64** different cache blocks. How many bits of **tags** and **LRU** bits (as required) are needed per entry? **[2]**

2. Consider a **byte-addressable** architecture with **12-bit virtual addresses**. The system uses **1KB** of **physical memory** with **64B** page sizes. The system has a **16-byte 2-way set associative cache**

with a **2-byte block size** and a **fully-associative TLB** with **8 entries**. The **TLB, cache contents**, and a **snippet of the single level** page table are provided below:

**Page Table**

| VPN | PPN | Valid |
|---|---|---|
| 0x10 | 0x3 | 1 |
| 0x11 | 0xF | 1 |
| 0x12 | 0x3 | 0 |
| 0x13 | 0xF | 0 |
| 0x14 | 0x2 | 1 |
| 0x15 | 0x1 | 1 |
| 0x16 | 0xE | 0 |
| 0x17 | 0x2 | 0 |

| VPN | PPN | Valid |
|---|---|---|
| 0x18 | 0x4 | 0 |
| 0x19 | 0x8 | 1 |
| 0x1A | 0x7 | 1 |
| 0x1B | 0xA | 1 |
| 0x1C | 0x5 | 1 |
| 0x1D | 0x4 | 0 |
| 0x1E | 0xD | 0 |
| 0x1F | 0x1 | 1 |

**TLB**

| Tag | PPN | Valid |
|---|---|---|
| 0x05 | 0xB | 1 |
| 0x1F | 0x1 | 1 |
| 0x00 | 0xB | 0 |
| 0x17 | 0xA | 0 |
| 0x11 | 0xF | 1 |
| 0x0F | 0xA | 0 |
| 0x0D | 0x0 | 1 |
| 0x02 | 0x3 | 0 |

**Cache**

| Set Index | Tag | Valid | Byte0 | Byte1 |
|---|---|---|---|---|
| 0 | 0x37 | 1 | 0xDE | 0xAD |
| | 0x1A | 0 | 0x12 | 0xB0 |
| 1 | 0x26 | 0 | 0x0A | 0xC1 |
| | 0x33 | 1 | 0x99 | 0x1F |
| 2 | 0x1C | 1 | 0x84 | 0x92 |
| | 0x00 | 1 | 0xBE | 0xCF |
| 3 | 0x7B | 1 | 0xCC | 0xA0 |
| | 0x0A | 1 | 0x45 | 0x67 |

**VPN = Virtual Page Number, PPN = Physical Page/Frame Number**

a. Say that the processor reads one byte from virtual address **0x71A** and that the cache is **physically indexed**. Answer the following questions. Provide all numeric answers in **hex**. If the answer is unknown, write **"unknown"**.

i. What is the **virtual page number** associated with that address? **[1]**

ii. Is this a **TLB** hit? **[1]**

iii. What is the **physical page number** associated with this access? **[1]**

iv. What is the **set number** where the data could be found? **[1]**

v. What is the **value of that byte** of memory being accessed? **[1]**

b. Now say that the processor reads one byte from virtual address **0x45F** and that the cache is **virtually indexed**. Answer the following questions. Provide all numeric answers in **hex**. If the answer is unknown, write **"unknown"**.

i. What is the **virtual page number** associated with that address? **[1]**

ii. Is this a **TLB** hit? **[1]**

iii. What is the **physical page number** associated with this access? **[1]**

iv. What is the **set number** where the data could be found? **[1]**

v. What is the **value of that byte** of memory being accessed? **[1]**

3. Consider a **byte-addressable memory system** with **64KB** of **virtual memory** and **4KB** of **physical memory**. The page size is **256B**. The system has a **4 entry, direct-mapped, virtually addressed cache** with block size of **4B**. The system also has a **4 entry, fully associative TLB**. The initial state of the **cache**, the **TLB**, and part of the **single level page table** are shown below. Assume that **physical memory pages 0x6** and onward are unused, and if a **page fault** occurs, the memory pages will be allocated starting at the **lowest possible page number.** The initial states of the **cache, TLB, and page table** are shown below.

Initial state of cache:

| Set | Valid | Tag |
|---|---|---|
| 0 | 1 | 0x306 |
| 1 | 1 | 0x003 |
| 2 | 1 | 0x030 |
| 3 | 1 | 0x030 |

Initial state of TLB

| Valid | VPN | PPN |
|---|---|---|
| 1 | 0x07 | 0x2 |
| 1 | 0x30 | 0x4 |
| 0 | - | - |
| 0 | - | - |

**VPN** = *Virtual Page Number,*

**PPN** = *Physical Page/Frame Number*

Initial state of Page Table

| Valid | VPN | PPN |
|---|---|---|
| 1 | 0x00 | 0x1 |
| 1 | 0x01 | 0x3 |
| 0 | 0x02 | 0x3 |
| 0 | 0x03 | 0x5 |
| 0 | 0x04 | 0x1 |
| 0 | 0x05 | 0x4 |
| 1 | 0x06 | 0x5 |
| 1 | 0x07 | 0x2 |
| (cont)... | (cont)... | (cont)... |

You want to access a virtual address: **0x0306**. Answer the questions below. Write any numerical answers in **hex**. Write **N/A** if the box does not require an answer. **[10]**

| | |
|---|---|
| Cache Tag of 0x0306 | |
| Cache Line Indez of 0x0306 | |
| Cache Block Offset of 0x0306 | |
| Cache Hit? (write Yes or No) | |
| TLB Tag of 0x0306 | |
| TLB Hit? (write Yes or No) | |
| Page offset of 0x0306 | |
| Virtual Page Number of 0x0306 | |
| Physical Page Number of 0x0306 | |
| Page Fault? (write Yes or No) | |

4. Consider a **byte-addressable system** with **20-bit virtual addresses**, and a **3-level hierarchical page table**. Memory pages are **32B** in size and **page table entries** are **2-bytes**. Further, each of the **2nd** and **3rd level page tables** must fit into a **single page**. No such restriction exists for the **1st level table**.
    a. Provide the breakdown (bit-ranges) of the **20-bit virtual address** required to carry out a **virtual-to-physical translation** in the following table: **[4]**

| 1st Level Page Table Index | 2nd Level Page Table Index | 3rd Level Page Table Index | Page Offset |
|---|---|---|---|
| | | | |

b. Assume instruction fetches will never lead to page faults. And that there is no **TLB** in the system. Assume that there is enough **physical memory** to never need a page replacement. **[6]**

```
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 **1st level table** is in memory. None of the **2nd, 3rd 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. Calculate the **total number** of **disk accesses** incurred due to accesses to "arr" in the "for loop". Show your work.
Note: Due to padding, the blob struct will occupy **16B** of memory.

5. Let's Consider an **8-bit byte-addressable** system with **256B** of physical memory partitioned into **16B** pages. The system manages its virtual memory using a separate single-level page table per process. Assume that there is **no TLB**. Page table entries are **1 byte** each, in the format below: Assume all relevant page table entries are valid. Page table entries are **1 byte** each, in the format below:

| Bits 7-4 | Bits 3-0 |
|---|---|
| <control bits> | Physical Page Number |

Tabular representation of the system's physical memory:

| Address | Data (from <Address + 0x0> to <Address + 0x7>) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x00 | 0xD7 | 0x82 | 0x3E | 0x0F | 0x50 | 0xCE | 0x38 | 0xA6 |
| 0x08 | 0x75 | 0xFF | 0x18 | 0x40 | 0xD3 | 0x19 | 0x64 | 0x22 |
| 0x10 | 0x64 | 0x33 | 0xD1 | 0x74 | 0xB0 | 0x55 | 0xA3 | 0xE5 |
| 0x18 | 0x37 | 0x5F | 0x9D | 0xC8 | 0x45 | 0x28 | 0x18 | 0x09 |
| 0x20 | 0x05 | 0xA6 | 0x17 | 0x28 | 0x75 | 0xCC | 0x29 | 0x0F |
| 0x28 | 0x49 | 0xD3 | 0xA1 | 0x82 | 0x00 | 0x4A | 0x2B | 0xEA |
| 0x30 | 0x35 | 0x12 | 0x75 | 0x76 | 0xF5 | 0xD2 | 0xFE | 0x3D |
| 0x38 | 0x72 | 0xA6 | 0x44 | 0x20 | 0x08 | 0x1C | 0x19 | 0x66 |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 0x40 | 0xCE | 0x51 | 0x6E | 0x2F | 0x1F | 0xDD | 0x61 | 0xFC |
| 0x48 | 0x11 | 0x84 | 0xD5 | 0x06 | 0x2A | 0xB5 | 0x58 | 0x77 |
| 0x50 | 0x2D | 0xF1 | 0x0F | 0x8B | 0x33 | 0xC6 | 0xD8 | 0x43 |
| 0x58 | 0x17 | 0xA4 | 0x21 | 0x3D | 0xA9 | 0xE8 | 0x91 | 0x55 |
| 0x60 | 0x58 | 0xCO | 0x9E | 0xD6 | 0x3B | 0xD1 | 0x07 | 0x58 |
| 0x68 | 0x2C | 0x84 | 0x6D | 0x11 | 0xBD | 0x05 | 0xF1 | 0x2D |
| 0x70 | 0xF1 | 0x71 | 0x08 | 0x6A | 0x22 | 0x30 | 0xAC | 0x70 |
| 0x78 | 0x55 | 0xC7 | 0x9A | 0x18 | 0x34 | 0x87 | 0x2E | 0x27 |

a. Assume a **16B, virtually indexed, fully associative data cache** with a block size of **4B** and initially empty state. It is initially empty. Assume **Page Table Base Register** (PTBR) points to address **0x10**.

Fill out the following table. **[6]**

| Virtual Address | Cache Access (Hit/ Miss?) | Physical Address | Data Returned (1 byte) |
|---|---|---|---|
| 0x34 | | | |
| 0x82 | | | |
| 0x80 | | | |
| 0x3F | | | |

b. After these accesses are finished, the processor switches to a different process. The **PTBR** now points to address **0x30**. The **state of memory** and **cache** are unchanged. Assume **data cache** has **no solution** to prevent aliasing.
Fill out the following table as you did above.
However, this time, differentiate between true **cache hits** vs. **aliased cache hits.** Provide **physical address** of **current process' data.** **[8]**

| Virtual Address | Cache Access (Hit/ Alias/ Miss) | Physical Address | Data Returned (1 byte) | Correct data (1 byte) |
|---|---|---|---|---|
| 0x45 | | | | |
| 0x36 | | | | |
| 0x47 | | | | |
| 0x83 | | | | |

6. Consider the following architecture specification:

| Functional Unit Type | Cycles in EX | Number of Functional Units |
|---|---|---|
| Integer | 1 | 1 |
| FP Adder | 5 | 1 |
| FP Multiplier | 8 | 1 |
| FP Divider | 15 | 1 |

- Assume that you have **unlimited** reservation stations.
- Memory accesses use the **integer functional unit** to **perform effective address** calculation during the EX stage. **Loads and Stores** stay in EX for 1 cycle.
- Functional units **are not pipelined**.
- If an **instruction moves** to its WB stage in cycle x, then an instruction that is waiting on the **same functional unit** (due to a structural hazard) can start executing in cycle x.
- An instruction waiting for data on the **Common Data Bus** (CDB) can move to its **EX stage** in the cycle after the **CDB broadcast**.
- Only one instruction can be written to the **CDB in one clock cycle**. Branches and stores do not need the **CDB**.
- Whenever there is a conflict for a **functional unit** or the **CDB**, assume that the oldest (by program order) of the **conflicting instructions** gets access, while others are stalled.
- Assume that the result from the **integer functional unit** is also broadcast on the **CDB** and forwarded to dependent instructions through the **CDB** (just like any floating point instruction).
- Assume that the **BNEQZ** occupies the **integer functional unit** for its computation and spends one cycle in **EX**.
- Assume that your **reorder buffer** has **12 entries** (at the beginning of execution the **ROB** is empty).

Assume **hardware speculation** and **dual-issue** in the **Tomasulo pipeline**. That is, assume that an instruction can be issued even before the branch has completed (or started) its execution (as with perfect branch and target prediction). However, assume that an instruction after a branch cannot issue in the same cycle as the branch; the earliest it can issue is in the cycle immediately after the branch (to give time to access the branch history table and/or buffer). Any other pair of instructions can be issued in the same cycle. Assume that a store calculates its **target address** in **EX** and performs its **memory access** during the **Commit stage**. Recall that stores do not write back. Furthermore, **two instructions can commit each cycle**. Fill in the cycle numbers in each pipeline stage for each instruction in the first **two iterations of the loop** represented below, assuming the branch is always taken. **[12]**

| Instruction | Issue | Execute | Write Back | Commit |
|---|---|---|---|---|
| **Iteration 1** | - | | | |
| LP: L.D F0, 0(R1) | 1 | 2 | 3 | 4 |

| ADD.D F0, F0, F6 | 1 | 4-8 | 9 | 10 |
|---|---|---|---|---|
| DIV.D F2, F2, F0 | | | | |
| L.D F0, 8(R1) | | | | |
| DIV.D F4, F0, F8 | | | | |
| S.D F4, 16(R1) | | | | |
| DADDI R1, R1, #-24 | | | | |
| BNEZ R1, LP | | | | |

| Iteration 2 | - | | | |
|---|---|---|---|---|
| LP: L.D F0, 0(R1) | | | | |
| ADD.D F0, F0, F6 | | | | |
| DIV.D F2, F2, F0 | | | | |
| L.D F0, 8(R1) | | | | |
| DIV.D F4, F0, F8 | | | | |
| S.D F4, 16(R1) | | | | |
| DADDI R1, R1, #-24 | | | | |
| BNEZ R1, LP | | | | |

7. Consider a processor equipped with speculative **out-of-order** execution **Tomasulo hardware**. Study the **processor performance** when running the following code of the operation, Y=aX+Y, for a vector length 100. Initially, R1 is set to the **base address** of array X and R2 is set to the base address of Y.

```
        DADDIU    R3, R1,#800    ;  R1 = upper bound on X
foo:    L.D       F2, 0(R1)      ;  (F2) = X(i)
        MUL.D     F4,F2,F0       ;  (F4) = a*X(i)
        L.D       F6,0(R2)       ;  (F6) = Y(i)
        ADD.D     F6,F4,F6       ;  (F6) = a*X(i) + Y(i)
        S.D       F6,0(R2)       ;  Y(i) = a*X(i) + Y(i)
        DADDIU    R1,R1,#8       ;  increment X index
        DADDIU    R2,R2,#8       ;  increment Y index
        DSLTU     R3,R1,R4       ;  test: continue loop?
        BNEZ      R3,foo         ;  loop if needed
```

| FU Type | Cycles in EX | Number of FUs | Number of reservation stations |
|---|---|---|---|
| Integer | 1 | 1 | 5 |

| | | | |
|---|---|---|---|
| FP Adder | 10 | 1 | 3 |
| FP Multiplier | 15 | 1 | 2 |

Assume the following:

- **Functional units** are not pipelined.
- There is **no forwarding** between functional units; results are communicated by the **common data bus (CDB)**.
- The **execution stage (EX)** does both the **effective address calculation** and the **memory access** for loads and stores. Loads require one clock cycle.
- The **issue** (IS) and **write-back** (WB) result stages each require **one clock cycle**.
- There are **five load** buffer slots and **five store** buffer slots.
- Assume that the **Branch on Not Equal to Zero** (BNEZ) instruction requires one clock cycle.
- Assume a large enough **ROB** to accommodate all **in-flight instructions**.

For this part, use the **single-issue speculative** Tomasulo pipeline with the pipeline latencies from the list above. Fill up the table for **two iterations** of the loop. You can ignore the first instruction (DADDIU R4, R1, #800). **[12]**

| Loop Seq. | Instruction | Issue | Execute | Write Back | Commit |
|---|---|---|---|---|---|
| 1 | L.D F2, 0(R1) | 1 | 2 | 3 | 4 |
| | | | | | |

8. In this problem, we consider a **scalar processor** with **in-order fetch, out-of-order dispatch,** and **in-order commit** execution engine that employs **Tomasulo's algorithm**. This processor behaves as follows:
   - The processor has **four** main pipeline stages: Fetch (F), Decode (D), Execute (E), and Write-back (W).
   - The processor implements a **single-level data cache**.
   - The processor has the following two types of execution units **but it is unknown how many of each type the processor** has.
     - **Integer ALU:** Executes integer instructions (i.e., addition, multiplication, move, branch).
     - **Memory Unit:** Executes load/store instructions.
   - The processor is connected to a **main memory** that has a fixed access latency.
   - **Load/store instructions** spend cycles in the E stage exclusively for accessing the data cache or the main memory.
   - There are **two** reservation stations, **one** for each execution unit type.

   The reservation stations are all **initially empty**. The processor executes an arbitrary program. From the beginning of the program until the program execution finishes, **seven dynamic instructions** enter the processor pipeline. Table below shows the **seven instructions** and their execution diagram.

- **Instruction semantics**:
  - MV R0 ← #0x1000: moves the hexadecimal number **0x1000** to register R0.
  - LD R1 ← [R0]: loads the value stored at memory address R0 to register R1.
  - BL R1, #100, #LB1: a branch instruction that conditionally takes the path specified by label "#LB1" if the content of register R1 is smaller than integer value 100.
  - MUL R1 ← R1, #5: multiplies R1 and 5 and writes the result to R1.
  - ST [R0] ← R1: stores R1 to memory address specified by R0.
  - ADD R1 ← R1, R0: adds R1 and R0 and writes the result to R1.

| Instruction/Cycle: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1: MV R0 ←#0x1000 | F | D | EI | E2 | E3 | E4 | W | | | | | | | | | | | | | | | | | | | |
| 2: LD R1 ← [R0] | | F | D | - | - | - | EI | E2 | E3 | E4 | E5 | E6 | E7 | E8 | W | | | | | | | | | | | |
| 3: BL R1 #100, #LB1 | | | | | | F | D | - | - | - | - | - | - | - | EI | E2 | E3 | E4 | W | | | | | | | |
| 4: MUL R1 ← R1, #5 | | | | | | | | | | | | | | F | D | EI | E2 | E3 | // squashed (i.e. killed) | | | | | | | |
| 5: ST [R0] ← R1 | | | | | | | | | | | | | | | F | D | - | - | // squashed (i.e. killed) | | | | | | | |
| 6: ADD R1 ←R1, R0 | | | | | | | | | | | | | | | | | | | F | D | E1 | E2 | E3 | E4 | W | |
| 7: ST [R0] ← R1 | | | | | | | | | | | | | | | | | | | | F | D | - | - | - | E1 | W |

(a) Using the information provided above, answer the following questions regarding the processor design. If a question has more than one correct answer or a correct answer cannot be determined using the information provided in the question, answer the question as specifically as possible. For example, use phrases such as **"at least/at most"** and try to narrow down the answer using the information that is provided in the question and can be inferred from the table. If nothing can be inferred, write **"Unknown"** as an answer. Explain your reasoning briefly, in all cases.

a. What is the **cache hit latency**? **[2]**
b. What is the **cache miss latency**? **[2]**
c. What is the **cache line size**? **[1]**
d. What is the number of entries in each **reservation station** (R)? **[1]**
e. How many ALUs does the processor have? **[1]**
f. Is the integer ALU pipelined? **[1]**
g. Does the processor perform **branch prediction**? **[1]**
h. At which pipeline stage is the correct outcome of a **branch evaluated**? **[1]**

9. Consider the instruction sequence that executes according to **Tomasulo's algorithm** with **Reorder Buffer (RoB)** and **Load-Store Queue (LSQ)**.

| Label | Instruction | Operands | | |
|---|---|---|---|---|
| Loop: | LD | R1 | 0(R3) | |
| | LD | R2 | 0(R4) | |
| | MUL | R5 | R1 | R2 |
| | ADD | R5 | R5 | R6 |
| | ST | R5 | 0(R3) | |
| | SUB | R2 | R2 | R0 |
| | BNE | R2 | R7 | Loop |

- The instructions are issued and committed **in order**. Assume there is one execution unit for ADD/SUB with a **6-entry** reservation station and one for **DIV/MUL** operations with a **4-entry** reservation station.
- There is also a Load Store unit with **store-to-load** forwarding enabled. The **ROB** can hold up to **12 instructions**, and the **LSQ** can hold up to **6 Instructions**.
- The load/store instruction takes **2 cycles**, addition/subtraction instruction takes 1 cycle, and multiplication takes **8 cycles**, whereas the branch decision takes **2 cycles** to finish (branch instruction does not require write back).
- Assume that the cache is **write-through** and the **cache miss penalty** is **5 cycles**. Therefore, for a LD instruction that suffered a cache miss, the total execution time will be (2+5=) **7 cycles**. Similarly, for all SD instructions, the execution time will be **7 cycles**.
- The **initial** content of the registers R0, R3, R4, R6 and R7 are 5, 10, 20, 50 and 10 respectively. Assume that the **initial** value in any memory location is the same as the **address** value. For example, the value at MEM[10] is 10.
- We also assume that the first two LD instructions will face a **cache miss** and the **Branch Predictor** takes the decision of "**Always Taken**" in every invocation.
- Same cycle **broadcast** and capture is not allowed.
- Assume the **store instruction** also passes through the **write-back** stage (of one cycle).
- Reservation entries are **freed** on dispatch.

Based on the above information, answer the following questions. Show your work for full credit.

a. What will be the cycle number at which the **first** occurrence of the instruction **BNE** is issued, executed and committed? **[6]**

b. What will be the cycle number at which the **second** occurrence of the instruction **BNE** is issued, executed and committed ? **[6]**

| Instruction | Issue | Execute | Write Back | Commit |
|---|---|---|---|---|
| LD R1, 0(R3) | 1 | 2 | | |

## Section B (To be answered by 2-Year M.Tech Students)

**(Answer All Questions)**

1. Suppose you want to run a program that has four separate threads of execution on a multithreaded processor. Two of these threads experience only one L2 cache miss (L2 miss penalty = **80** cycles) every **20** instructions, and incur no L1 cache miss. The other two threads suffer only one L1 cache misses (L1 miss penalty = **4** cycles) every **20** instructions, and incur no L2 cache misses. Would this program run faster: **a)** On a processor using fine-grained **or b)** On a similar processor using coarse-grained multithreading? Justify your answer using one short sentence. **[2]**

2. Is there any loop-carried dependence(s) in the following code? If so, identify with which loop (that is, i-loop or j-loop) are these associated. For each loop-carried dependence, identify whether it is cyclic. Clearly show your work out. **[3]**

```
for   (i=1;   i   <   n-1;   i++)
        for   (j=1;   j   <   n-1;   j++)
                A[i+2][2*j+1]   =   A[i+2][4*j-1]   +   A[i+2][4*j+1];
```

3. Consider the following **for** loop that iterates **100,000** times. Out of the total number of iterations, in how many iterations are the two branches (branch1 and branch2) correctly predicted using **2**-bit predictors, each initialized to strongly not taken? Assume that the size of the BPB (BHT) is extremely large. **[3]**

```
for (i=0; i<100,000; i++) {
        if (i%2==1) {     // branch 1: if i is odd
            ...   ...   ...   // do something
        }
}                         // branch 2: loop
```

4. The clock rate of a simple MIPS processor is **200** MHz. Simulation results of the processor with a perfect cache system indicate a CPI of **1** is achieved while running a certain program. The processor is next fitted with a cache system having a hit time of **1** clock cycle and a hit rate of **95%.** It takes **50 ns** to access the first word from main memory and **80 ns** to load the complete cache line. The cache uses critical word first technique. What is the average utilization of the main memory? **[6]**

5. Simulation experiments on the **Dolphin** processor for an important workload indicated that two branch instructions at the following two 32-bit addresses (in binary) execute frequently:
   - **Address A**:1000 1101 1100 0011 0110 1011 0100 1001
   - **Address B**: 1000 1101 0110 1001 1110 1011 0100 1001

   a) The low branch prediction accuracy was analyzed by the processor designers. They suspected that the above two branch instructions must be conflicting (hashing to the same entry) in the BPB (BHT). For a simple "bimodal" predictor of two-bit saturating counters, at most how many entries must the predictor have in order to prevent these two branches from

interfering (conflicting) with each other? In this case, **at least** how much storage space (in bytes) is required for implementing the BPB? **[1+1]**

b) Simulation results indicated that a predictor with just **2048** entries (**512** bytes) would be sufficient, if it wasn't for the above two troublesome branches. The processor designers had a brilliant idea: "Why not create a set-associative BPB?" They considered a two-way set-associative predictor that uses a straightforward tagging strategy (each two-bit counter has a tag, instructions have 32-bit addresses). How large (in KBs) is a two-way set-associative **2048**-entry tagged predictor? Assume that an LRU replacement would be used. **[4]**

6. In a certain multiprocessor, the private L2 caches at various nodes are connected using a snooping bus. Assume that each node is a multiple-issue, out-of-order processor and can achieve an average IPC of **1.1**. The average memory access per instruction is **1.2**. Assume that the global miss rate of each L2 cache is **1.5%.** The bus speed (cycle time) is four times slower than the processor speed (cycle time). The bus can serve one request every **2** bus cycles. Furthermore, on the average **30%** of the replaced blocks are dirty and need to be written back to the memory. Calculate at most how many processors can be supported by the snooping bus. You can assume that the data bus connecting the memory as well as the memory modules themselves are not bottlenecks. **[6]**

7. A designer is investigating two different cache designs with similar costs for realizing the memory subsystem of a new processor.

   **Option 1:** A direct-mapped cache with a **90%** hit rate and **2 ns** hit time.

   **Option 2:** A set-associative cache with a **93%** hit rate and **10 ns** hit time. To improve the performance of the cache, the designer has incorporated a way predictor. If the way predictor is correct, cache hits take only **3 ns**, while way prediction misses that still produce cache hits take **12 ns**.

   Assume that the cache miss penalty is **200 ns**, regardless of the type of cache being used. How accurate must the way predictor be for the option 2 to result in better overall performance? **[6]**

8. The ideal CPI for a processor when all memory requests are cache hits is **1** cycle. The average memory access time (AMAT) is **3** clock cycles. The L1 cache miss penalty is **80** clock cycles. Designers are trying to achieve **50%** improvements to AMAT, by adding an on-chip L2 cache. The L2 cache hit time would be **6** clock cycles and the miss penalty would be **80** cycles. To obtain the desired speedup, how often must data be found in the L2 cache ( that is, what must be local hit rate for L2)? **[6]**

9. A computer system with in-order execution runs at **1GHz** and has a CPI of **1**, with perfect caches. It was later fitted with a split L1 cache. Both the L1 I-cache and the D-cache are direct mapped and hold **32KB** each with block size **64 bytes**. The I-cache has a **2%** miss rate, and the D-cache is write-through with **5%** miss rate. The hit time for both the I-cache and the D-cache is **1ns**. The L2 cache is a **512KB** unified write-back cache having block size of **64 bytes**. The hit time of the L2 cache is **15ns**. The local hit rate of the L2 cache is **80%**. The **64-bit** wide main memory has an access

latency of **20ns**, after which any number of bus words may be transferred at the rate of one bus word (**64-bit**) per bus cycle on the **64-bit** wide **100MHz** main memory bus. Compute the AMAT for instruction and data. **[6]**

10. Consider a simple bus-based symmetric multiprocessor. Each processor has a single private **write-through** cache. Cache coherence maintained with a snooping, invalidate protocol. How many states would the state machine implementing the protocol have? Name the states. Draw the state machine diagram showing the events causing the transitions among the states and the associated actions if any. **[2+4]**

11. The designers of the **R-Star** Processor are trying to improve the performance of their branch resolution logic. Their base processor design achieves an IPC of **0.8** for the SPEC benchmark suite. Their proposed change shortens the execution pipeline, shaving **2 cycles** off the average branch misprediction penalty. However, the design change increases the clock period by **10%.** Over the entire benchmark suite, **20%** of instructions are branches. However, we unfortunately do not know the prediction accuracy of the branch predictor. For what range of prediction accuracies will the proposed change result in performance improvement? Clearly show the details of your work out.**[6]**

12. Consider a multi-processor having the following characteristics:
    - **8**-processor bus-based multiprocessor with private: L1 instruction and L1 data caches, and private L2 cache. However, the L3 cache is shared.
    - Snoopy invalidation protocol on write back caches is used for cache coherence.
    - Each processor is hyperthreaded and executes one memory operation (load or store) per cycle
    - The global L1 data cache miss rate is **20%**, global L1 instruction miss rate is **10%**, and global L2 cache local miss rate is **25%**.

    In this system, for any of the L2 caches, much more tag accesses were observed than actual data access. Please explain why this is so using one sentence. How many tag accesses occur per cycle? **[2+4]**

13. Consider a pipelined processor that has an average CPI of **1.8** when there are no memory stalls. The instruction cache has a hit rate of **95%** and the data cache has a hit rate of **98%**. Assume that memory reference instructions account for **30%** of all the instructions executed. Out of these data memory access instructions, **80%** are loads and **20%** are stores. On the average, the read-miss penalty is **20** cycles and the write-miss penalty is **5** cycles. Compute the effective CPI of the processor by taking into account the memory stalls. **[6]**

14. Assume that the base CPI for a pipelined datapath in a single core processor is **1**, when there are no *cache misses.*

    Now first consider a *single core chip* with only a split L1 caches (code named C1). A benchmark suite run on C1 indicates that for every **10,000,000** accesses to the L1 data cache, there are **300,000** misses.

- If data is found in the L1 cache, it can be accessed in **1** clock cycle, and there are no pipe stalls
- If data is not present in the L1 cache, it can be accessed in **10** clock cycles

Next, consider a dual-core version of the processor in which each processor has private L1 cache and a shared L2 cache. It is code named C2:

- Coherency of data is maintained by deploying an MSI coherency protocol.

By running the same benchmark suite on the dual-core system it was found that, on an average, there are now **450,000** misses (considering both the L1 caches) per total of **10,000,000** accesses.

- If data is found in a cache, it can be accessed in **1** clock cycle
- However, on the average, **14** cycles are now required when there is an L1 cache miss.

What must the CPI of the multi-core system with a perfect cache, so that C2 will outperform the single core C1 system? **[6]**

**15.** Consider the following memory hierarchy for a processor:

- Level 1 cache: **90%** hit rate, 1-cycle hit time.
- Level 2 cache: **98%** hit rate, 15-cycle hit time.
- Main memory: **140**-cycle hit time.

**a)** What is the average memory access time for the given memory system? **[2]**

b) Assume that the L2 cache is write-allocate and write-back and that there is no write buffer. If D is the percentage of evicted blocks that are dirty, how many cycles (in terms of D) do L2 write-back operations add to the average memory access time? **[4]**

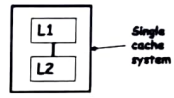
16. A simple **2.2 GHz** MIPS-like processor C1 is used to execute a program P1. However, unlike simple MIPS, branch outcomes are computed in C1 in the "decode" stage by incorporating additional logic circuits. The instruction statistics for P1 are as follows: Branches: **20%,** Loads: **20%,** Stores: **10%,** and Arithmetic Instructions: **50%.** Assume that the program P1 has no data dependencies. C1 uses a tournament dynamic branch predictor to predict the outcome of the branch instructions with a prediction accuracy of **80%**. Also assume that C1 uses a cache system in which **100%** of the instruction fetches and **X%** of the data accesses hit in the cache. The cache system miss penalty is **10** cycles. A customer requires that the processor C1 must achieve a throughput of **2** billion instructions per second. Calculate the minimum value of **X** (data hit rate in the cache system) that would satisfy this requirement. **[8]**

**17.** A processor being designed by a team of engineers has a clock rate of **2.5 GHz**. A CPI of **3** was observed from simulation experiments conducted with a perfect cache (no cache miss). A **32KB** L1 cache having miss rate is **6%** and cache miss penalty is **40ns** is proposed to be used. **[2+4+2+4]**

(a) What would be the CPI with the **32KB** L1 cache?

(b) A **512KB** L2 cache having hit time **10ns** and local miss rate **50%** was subsequently decided to be added. What would be the speedup achieved by using the **512KB** L2 cache compared to when no L2 cache is used?

(c) If we were to model the two-level cache system as a single cache (see diagram below), what would be the miss rate and miss penalty for this single cache system?




(d) Changing the mapping scheme of the **512KB** L2 cache from direct to two-way set associative, without changing its size, can improve its local miss rate to **20%** while its hit time would increase to **12 ns** due to the more complex access scheme. What is the effective CPI after this change? Is this change a good idea?

**---- The End ----**