True or False

1. F Increasing the size of a cache results in lower miss rates and higher performance.
2. F For a given capacity and block size, a set-associative cache implementation will typically have a lower hit time than a direct-mapped implementation.
3. T Memory buses are usually picked based on the speed whereas the I/O buses are primarily adopted based on the compatibility (industry standards) and cost.
4. F Asynchronous buses cannot be long due to clock skew restrictions.
5. F 'Polling' is always a better mechanism than 'interrupts' for handling I/O operations on a network interface card.
6. F In a write-through cache, a read miss can cause a write to the lower memory level.
7. F SRAMs must be refreshed periodically to prevent loss of information.
8. F Magnetic disks are volatile storage devices.
9. T An asynchronous bus is not clocked.
10. F Victim caches decrease miss penalty while they increase miss rate.
11. F RAID 5 uses more check disks than RAID 3 for the same number of data disks.
12. T Direct-mapped cache of size N has same miss rate as 2-way set associative of size N/2.
13. F The main difference between DRAM (the technology used to implement main memory) and SRAM (the technology used to implement caches) is that DRAM is optimized for access speed while SRAM is optimized for density.
14. T If the I/O devices are connected to the CPU through main memory busses, the performance of the processor may decrease since I/O commands could interfere with CPU memory accesses.
15. F For a given capacity and block size, a set-associative cache implementation will typically have a lower hit time than a direct-mapped implementation.
16. T Memory buses are usually picked based on the speed whereas the I/O buses are primarily adopted based on the compatibility (industry standards) and cost.
17. T Asynchronous buses cannot be long due to clock skew restrictions.
18. F In a write-through cache, a read miss can cause a write to the lower memory level.
19. F RAID 5 can recover from a two-disk failure.
20. F ATM has a variable message size and Ethernet has a fixed message size.
21. F Loop-carried dependencies can be completely eliminated by a hardware mechanism at run-time.
22. F The multi-cycle data path is always faster than the single-cycle data path.
23. F Loops with intra-iteration dependencies can be executed in parallel.
24. T Software Pipelining can be used on superscalar processors.

25. F It is possible to eliminate all the stalls due to data dependencies with a special hardware mechanism.

26. T Conflict misses do not occur in fully set-associative cache memories.

27. T MIPS (million instructions per second) can be defined as $\frac{instruction\ count}{CPU\ time\ in\ seconds} \times \frac{1}{10^6}$.

28. T For a given capacity and block size, a set-associative cache implementation will typically have a higher hit time than a direct-mapped implementation.

29. F Page tables in virtual memory systems are placed on a special cache memory.

30. F "Y is p% faster than X" can be defined as: $\frac{time\ of\ Y}{time\ of\ X} = 1 + \frac{p}{100}$

31. F Virtually addressable caches would have less hit time than physically addressable cache memories.

32. T TLB misses in a virtual memory system can occur and can be handled by software.

33. F In a write-back cache, a read miss always causes a write to the lower memory level.

34. F Branch history tables typically eliminate more stall cycles than branch target buffers.

35. F Both DRAM and SRAM must be refreshed periodically using a dummy read/write operation.

36. T Memory interleaving is a technique for reducing memory access time through increased bandwidth utilization of the data bus.

37. F Increasing the size of a cache results in lower miss rates and higher performance.

38. F For a given capacity and block size, a set-associative cache implementation will typically have a lower hit time than a direct-mapped implementation.

39. F For a given capacity and block size, a set-associative cache implementation will typically have a lower miss penalty than a direct-mapped implementation.

40. T Memory buses are usually picked based on the speed whereas the I/O buses are primarily adopted based on the compatibility (industry standards) and cost.

41. F Asynchronous buses cannot be long due to clock skew restrictions.

42. F In a write-through cache, a read miss can cause a write to the lower memory level.

43. T/F A processor can have different CPIs and MIPS rates for different programs.

44. T/F MIPS = MHz/CPI

45. T/F A Memory-Memory architecture is likely to have a higher CPI than Load-Store architecture assuming all other factors to be equal.

46. T/F Data dependencies can be completely eliminated through a bypassing mechanism.

47. T/F CPUtime = Instruction Count / (MIPS x 106)

48. T/F MIPS = MHz/CPI

49. T/F Name dependencies can be completely eliminated by a hardware mechanism at run-time.

50. T/F Both simultaneous multithreading and multicore rely on parallelism to get more efficiency from a chip.

51. T/F Fine-grained multithreading reduces vertical waste, but not horizontal waste.

52. T/F SRAMs must be refreshed periodically to prevent loss of information.

53. T/F Magnetic disks are volatile storage devices.

54. T/F An asynchronous bus is not clocked.

55. T/F Victim caches decrease miss penalty while they increase miss rate.

56. T/F RAID 5 uses more check disks than RAID 3 for the same number of data disks.

57. T/F Direct-mapped cache of size N has same miss rate as 2-way set associative of size N/2.

58. T/F The main difference between DRAM (the technology used to implement main memory) and SRAM (the technology used to implement caches) is that DRAM is optimized for access speed while SRAM is optimized for density.

59. T/F If the I/O devices are connected to the CPU through main memory busses, the performance of the processor may decrease since I/O commands could interfere with CPU memory accesses.

60. T/F Increasing the size of a cache results in lower miss rates and higher performance.

61. T/F For a given capacity and block size, a set-associative cache implementation will typically have a lower hit time than a direct-mapped implementation.

62. T/F For a given capacity and block size, a set-associative cache implementation will typically have a lower miss penalty than a direct-mapped implementation.

63. T/F Memory buses are usually picked based on the speed whereas the I/O buses are primarily adopted based on the compatibility (industry standards) and cost.

64. T/F Asynchronous buses cannot be long due to clock skew restrictions.

65. T/F 'Polling' is always a better mechanism than 'interrupts' for handling I/O operations on a network interface card.

66. T/F In a write-through cache, a read miss can cause a write to the lower memory level.

67. T/F RAID 5 can recover from a two-disk failure.

68. T/F ATM has a variable message size and Ethernet has a fixed message size.

69. T/F Loop-carried dependencies can be completely eliminated by a hardware mechanism at run-time.

70. T/F The multi-cycle data path is always faster than the single-cycle data path.

71. T/F Loops with intra-iteration dependencies can be executed in parallel.

72. T/F Software Pipelining can be used on superscalar processors.

73. T/F Loops with no intra-iteration dependencies can be executed in parallel.

74. T/F Software Pipelining cannot be used on (out-of-order execution) superscalar processors.

75. T/F SIMD processors are energy-efficient for vector operations.

76. T/F Loops with inter-iteration dependencies can be executed in parallel.

77. T/F Hazard detection is done at runtime in superscalar processors.

78. T/F VLIW architectures are suitable for embedded processors.

79. T/F VLIW architectures execute instructions in parallel based on a fixed schedule determined when the code is compiled.

80. T/F Superscalar architectures with speculative executions are suitable for embedded processors.

81. T/F RISC architectures are suitable for portable electronic devices.

82. T/F CPU performance is directly proportional to the system clock frequency.

83. T/F Magnetic Hard Disk Drives are nonvolatile just like Solid State Drives (SSD).

84. T/F SRAMs are optimized for access time.

85. T/F Amdahl's law does not apply to high-performance clusters.

86. T/F Directory-based protocols would better fit centralized shared-memory systems.

87. T/F Message Passing programming model can be used on small scale multiprocessors.

88. T/F Page Tables in virtual memory systems are placed on a special cache memory.

89. T/F Name dependencies can be eliminated by a hardware mechanism at run-time.

90. T/F Loops with inter-iteration dependencies cannot be executed in parallel.

91. T/F Only multicore processors can have simultaneous multithreading.

Fill in the Blanks

1. One hardware solution to control hazard stalls:
2. "Make common case faster" is the outcome of whose law?
3. A measure of performance in a computer system is: EXECUTION TIME = IC x CPI x CCT.
4. Define each term. IC ________________, CPI _________________, CCT _______________
5. Three types of Pipeline Hazards: _____________________, __________________, ___________.
6. Three types of Data Dependences: _____________________, __________________, _________.
7. Any two sources of exceptions in the Fetch and Execution stages of a pipelined processor: _____________________________________, ____
8. Two solutions to data hazard stalls can be _______________________, ____________________
9. Pipelining improves the performance by __________________ instruction throughput, as opposed to _______________________ the execution time of an individual instruction.
10. Three types of cache misses are.
11. CPU time for a program can be expressed as Instruction Count * (___________________ + Misses Per Instruction * Miss Penalty) * Clock Cycle Time.
12. One advantage of write-update protocol could be
13. One advantage of write-invalidate protocol could be
14. A solution to reduce hit time:

15. A solution to reduce miss rate:
16. A solution to reduce miss penalty:
17. A solution for higher bandwidth for main memory:
18. A strategy to select the block to replace on a cache miss:
19. An advantage of bus-based I/O interconnection can be:
20. A disadvantage of bus-based I/O interconnection can be:
21. An alternative I/O interconnection to bus-based interconnection:
22. When I/O components are considered we need alternative performance measures besides CPU time. The number of tasks completed per unit time is called
23. Which cache write mechanism allows outdated data in the main memory?
24. Goal of software techniques and hardware techniques, is to exploit
25. When instruction i and instruction j are to write the same register or memory location, it is called
26. Dependences among iterations of a loop are called
27. To be gathered in a single register, distance separating elements are called
28. Vector instructions which can be potentially executed altogether, is known as
29. If it is substantially difficult to find and exploit parallelism across branches, then it is known as
30. A special hardware buffer is required for instruction execution sequence that holds instruction results, this is known as
31. A branch-prediction cache which is used to store predicted address for upcoming instruction after branch, is called a
32. Hardware-based speculation method for executing programs, is necessarily a
33. Allowing multiple instructions for issuing in a clock cycle, is a goal of
34. Each bank register of fixed-length maintaining a single vector, is referred to as
35. Threads being blocked altogether and being executed in sets of 32 threads, is called a
36. A microprocessor clocked at a rate of 1.0 GHz has a clock cycle of _______________ seconds.
37. Types of stalls in a pipelined processor:
38. Vector instructions which can be potentially executed altogether, is known as
39. A special hardware buffer is required for instruction execution sequence that holds instruction results, this is known as
40. A branch-prediction cache which is used to store predicted address for upcoming instruction after branch, is called a
41. Threads in CUDA programming being blocked altogether and being executed in sets of 32 threads, is called a
42. If the clock and the maximum issue rates are the same, rank these three architecture approaches from fastest to slowest: **a.** Fine-grain multithreading; **b.** Superscalar; **c.** Simultaneous multithreading: ________________, _____________, ______________.

43. Three types of cache misses are:

44. CPU time for a program can be expressed as Instruction Count * (____________________ + Misses Per Instruction * Miss Penalty) * Clock Cycle Time.

45. One advantage of write-update protocol could be

46. One advantage of write-invalidate protocol could be

47. A solution to reduce hit time:

48. A solution to reduce miss rate:

49. A solution to reduce miss penalty:

50. A solution for higher bandwidth for main memory:

51. A strategy to select the block to replace on a cache miss:

52. An advantage of bus-based I/O interconnection can be:

53. A disadvantage of bus-based I/O interconnection can be:

54. An alternative I/O interconnection to bus-based interconnection:

55. When I/O components are considered we need alternative performance measures besides CPU time. The number of tasks completed per unit time is called

56. Which cache write mechanism allows outdated data in the main memory?

57. In what pipeline stage is the branch target buffer checked?

58. What needs to be stored in a branch target buffer to eliminate the branch penalty for an unconditional branch? Instruction at branch target or Address of branch target

Short Answers

1. Please do the following four questions based on the following techniques to improve performance with additional hardware and/or code.

   i. Dynamic Scheduling

   ii. Data Forwarding

   iii. Loop Unrolling

   iv. Software Pipelining

   v. Multithreading

   - Which one(s) require a great number of additional registers?
   - Which ones(s) can support multi-issue processors?
   - Which one requires most static-time preparation?
   - Which one(s) may cause code explosion?

2. Prior to the early 1980s, machines were built with more and more complex instruction sets. Why has there been a move to RISC machines away from complex instruction machines? Several possible answers: simplicity, simple design, lower CPI, low power, low cost, …

3. What are the two characteristics of program memory accesses that caches exploit? Temporal and spatial
4. What are the three types of cache misses? Compulsory, capacity, conflict
5. What is a Branch-Target Buffer? How does it differ from a Branch-History Table? What would be their sizes in a typical processor?
6. List pros and cons of VLIW architectures, and their limitations.
7. List pros and cons of Superscalar architectures, and their limitations.
8. Name two main characteristics of programs that would most benefit from GPU processors.
9. Name one limitation in the use of SIMD systems.
10. Name a major difference between SIMD vector processors and GPUs.
11. Please do the following parts based on the following techniques that improve performance with additional hardware and/or code.
    - A. VLIW
    - B. SIMD
    - C. Superscalar
    - D. GPU

    a. Which one(s) require a great number of additional registers? __________________________.

    b. Which one(s) can support multi-core processors? __________________________.

    c. Which one(s) require most compile-time preparation? __________________________.

    d. Which one(s) would be better with vector applications? ______________________.
12. **Virtual memory is an imaginary memory.** It is an alternate set of memory addresses. It allows programmers to use a very large range of memory for stored data. TLB is a special cache keeping address translations so that a memory access rarely requires a second access to translate the data. Advantages of VM include translation, protection, and sharing. Block replacement policies include LRU, FIFO, and random. DMA gives external device ability to write memory directly: much lower overhead than having processor request one word at a time.
13. **The original motivation for using virtual memory was "compatibility". What does that mean in this context? What are other motivations for using virtual memory?**

    Compatibility in this context means the ability to transparently run the same piece of (un-recompiled) software on different machines with different amounts of physical memory. This compatibility freed the application writer from worrying about how much physical memory a machine had. The motivation for using virtual memory today have mainly to do with multiprogramming, the ability to interleave the execution of multiple programs simultaneously on one processor. Virtual memory provides protection, relocation, fast startup, sharing and communication, and the ability to memory map peripheral devices.
14. **Bus is a shared communication link.** It is used to connect multiple subsystems. Buses are important technique for building large scale systems. Buses have advantages like versatility and low cost. On the other hand, they generate a communication bottleneck and their maximum speed is limited by the

length of the bus and the number of devices on the bus. Types of buses are processor-memory bus, I/O bus, backplane bus.

15. **TLB misses can be handled in software, or hardware can be used to handle them.** We can expect software to be slower due to the overhead of a context switch to the handler code, but the sophistication of the replacement algorithm can be higher for software and a wider variety of virtual memory organizations can be readily accommodated. Hardware should be faster, but less flexible. Floating-point programs often traverse large data structures and thus more often reference a large number of pages. It is thus more likely that the TLB will experience a higher rate of capacity misses.

16. **What is the principle behind RAID? What is RAID level 0 and why is it widely used.**

    A set of physical disk drives viewed by the OS as a single logical drive. Replace large-capacity disks with multiple smaller-capacity drives to improve the I/O performance (at lower price). Data are distributed across physical drives in a way that enables simultaneous access to data from multiple drives. . Redundant disk capacity is used to compensate for the increase in the probability of failure due to multiple drives. RAID level 0 refers to no redundancy and it is normal disk usage. Data are stripped across the available disks. Many personal computers and other systems do not use RAID and so fall under RAID level 0.

17. **What is the reason for using combination of first- and second- level caches rather than using the same chip area for a larger first-level cache?**

    The ultimate metric for cache performance is average access time: *tavg = thit + miss-rate* tmiss*. Multiple levels of cache are used because not all of the performance factors can be optimized in a single cache. Specifically, with *tmiss* (memory latency) given, it is difficult to design a cache which is both fast in the common case (a hit) and minimizes the costly uncommon case by having a low miss rate. These two design goals are achieved using two caches. The first level cache minimizes the hit time; therefore, it is usually small with a low-associativity. The second level cache minimizes the miss rate; it is usually large with large blocks and a higher associativity.

Multiple Choice:

1. Which of the following factors has the MOST effect in REDUCING the IC term given above? (circle only one)
    - **a.** ISA (Instruction Set Architecture)
    - **b.** Implementation
    - **c.** Technology
2. Which of the following factors has the most effect in reducing the CCT term?
    - **a.** ISA
    - **b.** Implementation
    - **c.** Technology
3. Which of the following factors has the most effect in reducing the CPI term? (circle only one)
    - **a.** ISA
    - **b.** Implementation
    - **c.** Technology

4. Considering the following performance measurements for one particular program, which computer has the higher MIPS rating? Which computer is faster?

Measurement	Computer A	Computer B
Instruction Count	9 billion	8 billion
Clock Rate	4.1 GHz	4 Ghz
CPI	1.0	1.1

a) A has higher MIPS and faster **b)** A has higher MIPS but slower

c) B has higher MIPS and faster **d)** B has higher MIPS but slower **e)** none of the above

5. Consider two different implementations, M1 and M2, of the same instruction set. There are three classes of instructions (A, B, and C) in the instruction set. M1 has a clock rate of 80 MHz and M2 has a clock rate of 100 MHz. For the given average number of cycles for each instruction class and their frequencies (for a typical program), calculate the average CPI for each machine, M1 and M2.

Instruction Class	Machine M1 Cycles/Instruction Class	Machine M2 Cycles/Instruction Class	Frequency
A	1	2	60%
B	2	3	30%
C	4	4	10%

a) 1.9 and 2.0 **b)** 1.6 and 2.3 **c)** 1.4 and 2.4 **d)** none of the others

6. Consider a memory system with a two-level cache with the following characteristics: The miss penalty from L2 cache to the main memory is 100 clock cycles, the hit time of the L2 cache is 10 clock cycles, the hit time of L1 is 1 clock cycle, and both cache memories have an average hit rate of 90%. What is the average memory access time?

a. 12 clock cycles

b. 34 clock cycles

c. 30 clock cycles

d. None of the above

7. Assume a 64KB cache with 16-byte block size and a 32-bit physical address. If a block has 16 tag bits, what is the type of this cache?

a. Direct mapped

b. 2-way set associative

c. Fully associative

d. None of the above

8. Which of the following systems is the least scalable with respect to its number of processors?

a. Cache coherent NUMA systems

b. Non-cache coherent NUMA systems

c. Symmetric multiprocessors

d. Parallel Vector Processor

9. Consider a Simultaneous Multithreading (SMT) machine with limited hardware resources. Which of the following hardware constraints would have the highest impact on the total number of threads that the machine can support.

   a. Number of functional unit
   b. Number of physical registers
   c. Data cache size
   d. Data cache associativity

10. You are asked to develop a parallel architecture that would be suitable for smartphones, laptops and datacenters. Which of the following would be most suitable? Justify your answer.

   a. A few heavy multicore processors
   b. Several lightweight cores with SoC characteristics: memory controllers and IO on die
   c. GPUs.

11. Which one of the following processors may possibly have the highest MIPS rate, assuming full compiler optimization and no cache misses?

   a. A single-issue processor driven by a 1.2 GHz clock.
   b. A 2-issue processor with a 600 MHz clock
   c. A 4-issue processor driven by a 300 MHz clock.
   d. An 8-issue VLIW processor driven by a 200 MHz clock.

12. Which of the following statements is false?

   a. A scalar processor processes one data item at a time.
   b. In a vector processor, a single instruction operates simultaneously on multiple data items.
   c. VLIW architectures execute instructions in parallel based on a fixed schedule determined when the code is compiled.
   d. Superscalar architectures with speculative executions are suitable for embedded processors

Numerical Answers:

1. On some program, the CPU takes 9 sec and the disk I/O takes 1 sec. What is the speedup using a CPU 9 times faster?

2. Performance of M1 is 100 MFLOPS and 200 MIOPS. Performance of M2 is 50 MFLOPS and 250 MIOPS. On a program using 10% floating point and 90% integer operations, which is faster? What is the speedup?

3. Find the average CPI for a program with the following statistics:

Operations	Frequency	Cycles
RR	25%	**3**
RM	50%	**4**
MM	25%	**5**

4. Assume we have a machine where the CPI is 2.2 when the all memory accessed hit in the cache. The only data accesses are loads and stores, and these total 48% of the instructions. If the miss penalty is 20 clock cycles and the miss rate is 3%, how much faster would the machine be if all instructions were cache hits?

5. Assume a 64 KB cache with 4-word block size (a word is 4 bytes) and a 32-bit address. If a block has 28 tag bits, what is the type of this cache? Fully associative

6. An embedded system has a response time of 10 ms, 50 percent of which is spent on its processor. How much faster should this processor be to reduce the overall response time to 8 ms? 1.66

7. A design team is to choose between a pipelined or nonpipelined implementation for a simple, single-issue processor. Which implementation will be faster for a program that is composed of 20% ALU instructions, 10% control instructions, and 70% memory instructions? Here are some design parameters for the two versions:

Parameter	Pipelined version	Nonpipelined version
Clock rate	500 MHz	350 MHz
CPI for ALU instructions	1	1
CPI for control instructions	2	1
CPI for memory instructions	2.7	1

8. Consider a system with a two-level cache having the following characteristics: L1 cache has an access time of 1 clock cycle with an average hit rate of 95%; L2 cache has an access time of 5 clock cycles (after L1 miss) with an average miss rate of 10%. If L2 misses take 40 clock cycles, what would be the average memory access time in clock cycles? If the clock rate is 1 Gz, what would it be in seconds? 1.45 ns

9. **Length of a bit in a 1-meter copper wire on a 1-Gbps, where the speed of propagation is $2.3x10^8$ m/s:** The width of a bit on a 1-Gbps link is 1/109(bit/sec) = 10^{-9} sec/bit. Length of a bit = 10^{-9} (sec/bit) x $2.3x10^8$ (meter/sec) = 0.23 meter/bit. One meter cable can have 1/0.23 = 4 bits.

10. **What is the average time to write or read a 32 KB block to a magnetic disk with the following properties? Rotational speed: 7,200 RPM; Transfer rate: 45 MB per second.; The advertised average seek time is 5 ms and it is three times longer than the measured seek time.; Controller overhead: 0.25 ms.**

    Average Seek Time (measured) = 5ms/3. Rotational latency: 0.5 * 1/(7200/60) / 1000 = 4.167ms

    5/3ms + 4.167ms + 32KB/45MB + 0.25ms = 6.8ms.

11. **Consider a virtual memory system with the following properties: 64-bit virtual byte address; 16-KB pages; 32-bit physical byte address. What is the total size of page table on this machine, assuming that the valid, protection, dirty, and use bits take a total of 4 bits and that all the virtual pages are in use. Please list and explain two different ways to implement this virtual memory system.**

    One page table is 16KB. So, 14 bits are required for page offset. Size of virtual page number = 64 – 14 = 50 bits. Size of physical page number = 32 – 14 = 18 bits. The number of entries of a page table is equal to the number of pages in the virtual address space. (264 bytes / 16 KB) = (2^{64} bytes / 24 $\times$ 2^{10} bytes) = 2^{50}. The width of each entry is the number of control bits plus the number of bits for physical page number: 4 + 18 = 22 bits. So, the page table contains (22×2^{50}) bits.

12. **Consider a virtual memory system with the following properties: 40-bit virtual byte address; 16-KB pages; 36-bit physical byte address. What is the total size of page table, assuming that the valid, protection, dirty, and use bits take a total of 4 bits and that all the virtual pages are in use?** (26×2^{26}) bits.

13. **The memory architecture of a machine X has the following characteristics: Virtual Address 54 bits; Page Size 16 K bytes; PTE Size 4 bytes. Assume that there are 8 bits reserved for the operating system functions (protection, replacement, valid, modified, and Hit/Miss- All overhead bits) other than required by the hardware translation algorithm. Derive the largest physical memory size (in bytes) allowed by this PTE format. Make sure you consider all the fields required by the translation algorithm.**

    Since each Page Table element has 4 bytes (32 bits) and the page size is 16K bytes:

    (1) we need $\log_2(16*2^{10}*23)$, ie, 17 bits to hold the page offset

    (2) we need 32 – 8 (used for protection) = 24 bits for page number

    The largest physical memory size is $2^{(17 + 24)/2(3)}$ bytes = 2^{38} bytes = 256 GB

    How large (in bytes) is the page table?

    The page table is indexed by the virtual page number which uses 54 – 14 = 40 bits. The number of entries are therefore 2^{40}. Each PTE has 4 bytes. So the total size of the page table is 2^{42} bytes which is 4 terabytes.

    Assuming 1 application exists in the system and the maximum physical memory is devoted to the process, how much physical space (in bytes) is there for the application's data and code.

    The application's page table has an entry for every physical page that exists on the system, which means the page table size is $2^{24}*4$ bytes. This leaves the remaining physical space to the process: $2^{38} - 2^{26}$ bytes

14. **A four-set associative cache-memory system consists of 32 blocks of cache. Each block holds 256 bytes of data. Each block includes a control field with two LRU bits, a valid bit and a dirty bit. The address space is 4 gigabytes. Calculate the total size of the cache, including all control, data and tag bits.**

    Total $ size = Data + Control bits + Tag; For each block: Control bits = LRU bits + Valid bit + Dirty bit = 2 + 1 + 1 = 4 b

    Address = Tag + Index + Offset; 256 = 2^8, then Offset = 8; Number of blocks in each direct mapped cache = 32/4 = 2^3, Index = 3; Tag = 32 – 8 – 3 = 21 bits. Total $ size = 32 blocks x (4 bits + 21 bits + 256 x 8 bits) = 66,336 bits = 8,292 B.

15. **Consider a system with a two-level cache having the following characteristics:**

    L1 Cache: Physically addressed; L1 hit time is 2 clock cycles; L1 average miss rate is 0.15

    L2 Cache • Physically addressed; L2 hit time is 5 clocks (after L1 miss); L2 average miss rate is 0.05

    If L2 miss takes 50 clock cycles, compute the average memory access time in clocks for the given 2-level cache system. AMAT = HT1 + MR1 x (HT2 + MR2 x MP2) = 2 + 0.15 x (5 + 0.05 x 50) = 3.125 clock cycles.

16. **Assume that a computer's address size is k bits, the cache size is S bytes, the block size is B bytes, and the cache is A-way set-associative. Suppose that B = 2b. Please find the followings in terms of S, B, A, b, and k: the number of sets in the cache and the number of index bits in the address.**

    Address size is k bits, cache size is S bytes/cache, block size is B = 2b bytes/block, and associativity is A blocks/set. The number of sets in the cache (X) can be can be defined, as follows:

    X = Sets/Cache = (Bytes/cache)/[(Blocks/set) x (Bytes/block)] = S/(AB)

    The number of address bits needed to index a particular set of the cache can be found, as follows:

    Cache set index bits = $\log_2$ (Sets/cache) = $\log_2$ (S/(AB)) = $\log_2$ (S/A) – b

17. **Please formulate the average disk access time and define each term. What do disks spend most of their time? What is the average time to write or read a 32 KB block to magnetic disks with the following properties? The disk rotates at 7,200 RPM and the transfer rate is 40 MB per second. The advertised average seek time is 4 ms and it is it is three times longer than the measured seek time. Assume that the controller overhead is 0.2 ms.**

    Ave. disk access time is the average time of between initiating a request and obtaining the first data character. Ave. disk access time is equal to average seek time + average rotational delay + transfer time + controller overhead. Controller time is the overhead the controller imposes in performing I/O access. The time for the requested sector to rotate under the head is the rotational delay. Transfer time is the time it takes to transfer of a block of bits. The time to move the arm to the desired track is called seek time. Disks spend most of their time waiting for the head to get over the data. ADAT = (4/3) + (0.5/7,200RPM) + (32KB/40MB/sec) + 0.2 ms = 1.33 + 4.17 + 0.8 + 0.2 = 6.5 ms.

18. **What is the average time to read 1KB data from a disk? Assuming advertised seek time is 6ms, transfer rate is 20MB/sec, the disk rotates 6000 RPM, and the controller overhead is 0.1 ms.** 7.15ms.

19. **The table reflects reasonable up-to-date information. However, most of these technologies and standards are currently under continuous improvements.**

	IDE/Ultra ATA	**SCSI**	**PCI**	**PCI-X**
Data Width	16 bits	8 bits (regular) 16 bits (Wide)	32 bits	64 bits (current) 32 bits (originally)
Clock Rate	Up to 100Mhz (started at 25mhz)	Up to 160Mhz (started at 5mhz)	33mhz and 66mhz	66, 100, 133mhz.
# of Bus Masters	1	Multiple	Multiple	Multiple
Bandwidth Peak	200MB/s	320MB/s	533MB/s	1066MB/s
Clocking	Asynchronous (old) Synchronous (latest)	Synchronous	Synchronous	Synchronous
Standard	ANSI standard ATA-1 through ATA-8	ANSI standard SCSI-1 through SCSI-3	PCI, PCI 1.0, 2.2, 2.3, and 3.0	PCI-X, PCI-X 1.0, and 2.0

20. **Assume a 64KB cache with four-word block size (a word is 4 bytes) and a 32-bit address. If a block has 28 tag bits, what is the type of this cache?**

    a) Direct mapped c) 2-way set associative c) Fully associative

21. **RAID 3** is unsuited to transactional processing because each read involves activity at all disks. In RAID 4 and 5 reads only involve activity at one disk. The disadvantages of RAID 3 are mitigated when long sequential reads are common, but performance never exceeds RAID 5. For this reason, RAID 3 has been all but abandoned commercially.

22. **You purchased an Acme computer with the following features:**
- **95% of all memory accesses are found in the cache**
- **Each cache block is 2 words, and the whole block is read on any miss**
- **The processor sends references to its cache at the rate of 10^9 words per second**
- **25% of those references are writes**
- **Assume that the memory system can support 10^9 words per second, reads or writes**
- **The bus reads or writes a single word at a time (the memory system cannot read or write two words at once)**
- **Assume that any one time, 30% of the blocks in the cache have been modified**
- **The cache uses write allocate on a write miss**

You are considering adding a peripheral to the system, and you want to know how much of the memory system bandwidth is already used. Calculate the percentage of the memory system bandwidth used on the average if the cache is Write Through. Be sure to state your assumptions.

Write Through				
Access Type	Access Hits Cache?	Frequency	Memory Accesses Generated	
Read	Yes	95% x 75% = 71.25%	0	The cache contains the data in question→ no need to generate a memory system access.
Read	No	5% x 75% = 3.75%	2	The cache fills the appropriate cache block from memory → requires two memory system reads as each block is two words.
Write	Yes	95% x 25% = 23.75%	1	The cache must generate a memory access to update the word in memory being written to cache.
Write	No	5% x 25% = 1.25%	1	Write No Allocate: The word of data is written to main memory only (1 access). If it were 'write allocate' cache: First, fill the appropriate cache block from memory (2 memory accesses). Then, write the word of data to main memory (1 access). → 3 accesses.

AccessesAvg = (71.25% x 0) + (23.75% x 1) + (3.75% x 2) + (1.25% x 1) = 0.325

Bandwidth Used = 0.325 x $(10^9)/10^9$ = 32.5%

23. **What is the bottleneck in the following system setup, the CPU, memory bus, or the disk set?**
- **The user program continuously performs reads of 64KB blocks, and requires 2 million cycles to process each block.**
- **The operating system requires 1 million cycles of overhead for each I/O operation.**
- **The clock rate is 3GHz.**
- **The maximum sustained transfer rate of the memory bus is 640MB/sec**
- **The read/write bandwidth of the disk controller and the disk drives is 64MB/sec, disk average seek plus rotational latency is 9ms.**
- **There are 20 disks attached to the bus each with its own controller. (Assume that each disk can be controlled independently and ignore disk conflicts.)**

Amount of time CPU takes to process each 64 KB block = $((2 * 10^6)/ (3 * 10^9)) * 10^3$ = 0.67 ms

Amount of time spent in memory transfer for 64 KB block = $((64 * 2^{10})/(640*2^{20}))*1000$ = 0.097 ms

Amount of time spent in I/O transfer for a 64 KB block = seek time + rot. delay + transfer time + controller overhead = $9 + ((64*2^{10})/(64*2^{20}))*10^3 + ((10^6 * 3)/10^9)*10^3$ = 12.97 ms. The main bottleneck is I/O in the above system.

18. We have a program core consisting of four conditional branches. The program core will be executed thousands of times. Below are the outcomes of each branch for one execution of the program core (T for taken, N for not taken).

Branch 1: T – T –T
Branch 2: N – N – N – N
Branch 3: T – N – T – N –T
Branch 4: T – T – T – N – T

Assume the behavior of each branch remains the same for each program core execution. For dynamic schemes, assume each branch has its own prediction buffer and each buffer initialized to the same state before each execution. List the predictions for the following branch predication schemes:

a. Always taken

b. Always not taken

c. 1-bit predictor, initialized to predict taken

What are the **prediction accuracies**?

	Always taken	Always not taken	1-bit predictor, initialized to predict taken
T – T –T	100%	0%	100%
N – N – N – N	0%	100%	75%
T – N – T – N –T	60%	40%	20%
T – T – T – N – T	80%	20%	60%

19. Suppose we have two different I/O systems A and B. A has data transfer rate: 5KB/s and has access delay: 5 sec. While B has data transfer rate: 3 KB/s and has access delay: 4 sec. Now we have a 3M I/O request, taking performance into consideration, which I/O system will you use? What about for a 3KB request?

3M request

Case 1: t (in sec) = 5 + (3 * 1024 * 1024) / (5 * 1024) = 5 + 614.4 = 619.4

Case 2: t (in sec) = 4 + (3 * 1024 * 1024) / (3 * 1024) = 4 + 1024 = 1028. So, system 1 will be chosen.

3K request

Case 1: t (in sec) = 5 + (3 * 1024) / (5 * 1024) = 5.6

Case 2: t (in sec) = 4 + (3 * 1024) / (3 * 1024) = 5 So, system 2 will be chosen.

Loop Optimizations:

1. Given the following code, list all the **dependences** by their types in the provided table. Is this a parallel loop? Is it parallelizable? If so, how? Devise a *dream machine* to execute this loop fastest. Please use equations and block diagrams to elaborate on your approach.

```
    for (i=2;i<100;i=i+1)
        a[i] = b[i] + a[i];        /* S1 */
        c[i-1] = a[i] + d[i];      /* S2 */
        a[i-1] = 2 * b[i];         /* S3 */
        b[i+1] = 2 * b[i];         /* S4 */
```

2. Given the following code:

```
    for (i=2;i<100;i=i+1)
        a[i] = b[i] + a[i];        /* S1 */
        c[i-1] = a[i] + d[i];      /* S2 */
        a[i-1] = 2 * b[i];         /* S3 */
        b[i+1] = 2 * b[i];         /* S4 */
```

   a. List all the dependencies by their types (TD: true-data, AD: anti-data, OD: output-data dependencies).

   b. Show how Software Pipelining can exploit parallelism in this code to its fullest potential. How many functional units would be sufficient to achieve the fastest execution time?

   c. Is this a parallel loop? If not, can it be parallelized? How?

3. Identify all of the data dependencies in the following code. Which dependencies are data hazards that will be resolved via forwarding? Which **dependencies** are data hazards that will cause a stall?

```
    S1:    add $3, $4, $6
    S2:    sub $5, $3, $2
    S3:    lw $7, 100($5)
    S4:    add $8, $7, $2
```

4. Linear loop transformations are useful in reductions of cache misses and increase possible parallelization. Consider the following two loops.

```
A)                                   B)
For I = 1, 1000                      For J = 1, 1000
 For J = 1, 1000                      For I = 1, 1000
  a(J) = c(J) + a(I, J)*b(J)            c(J) = c(J) + a(I, J)*b(J)
```

   a. Which of these loops would yield better performance? Why?

   b. Is it possible to schedule the loop you picked in the part a) onto a dual-core processor? How? How about onto a quad-core processor?

5. Consider the following piece of code for the following parts.

```
        R1 = 0
        F2 = R1
        F3 = R1
loop:   F4 = 2012
        F2 = A[R1]
        R1 = R1 + 1
        F3 = F3 + F2
        While F2 != F4
```

a. List all the dependencies by their types.

b. Is it possible to rewrite this code to eliminate some of these dependencies? How?

c. Explain if the loop in this code is parallel? If not, can it be parallelized? How?

d. Show how Software Pipelining can exploit parallelism in this code to its fullest potential. How many functional/execution units would be sufficient to achieve the fastest execution time?

6. Determine the loop-carried (inter-iteration) dependencies in the following loops. Are they parallel? If not, can you rewrite to make them parallel?

```
LOOP 1:
for (i = 1; i<=99; i++)
a[i] = b[i] + c[i];  /S1/
b[i] = a[i] + d[i];  /S2/
a[i+1] = a[i] + e[i]; /S3/
LOOP 2:
for (i=0; i<100; i=i+1)
A[i] = A[i] + B[i];   /* S1 */
B[i+1] = C[i] + D[i]; /* S2 */
```

7. Identify all of the data dependencies in the following code. Which dependencies are data hazards that will be resolved via forwarding? Which **dependencies** are data hazards that will cause a stall?

```
S1:     add $3, $4, $6
S2:     sub $5, $3, $2
S3:     lw $7, 100($5)
S4:     add $8, $7, $2
```

Branch Prediction

For this problem, we are interested in the following snippet of code:

```
int array[N] = {....};
for(int i = 0; i < N; i++)
    if (array[i] != 0)
        array[i] = array[i] + 1;
```

Using the disassembler we get:

```
        addi $n, $0, N
        addi $i, $0, 0
loop:
        ld $a, array($i)
        beqz $a, endif
        addi $a, $a, 1
        st $a, array($i)
endif:
        addi $i, $i, 4
        addi $n, $n, -1
        bnez $n, loop
```

(where N is some integer used to specify the size of array).

1. **For this piece of code, would it be better to spend hardware resources on branch prediction or register renaming?**

   Branch prediction. Especially for this code, there are too many dependencies between instructions to allow for register renaming to provide a benefit, and without branch prediction we cannot speculate past branches anyways to allow for register renaming to break the antidependencies between iterations.

2. **The processor that this code runs on uses a 512-entry branch history table (BHT), indexed by PC [10:2]. Each entry in the BHT contains a 2-bit counter, initialized to the 00 state. Each 2-bit counter works as follows: the state of the 2-bit counter decides whether the branch is predicted taken or not taken, as shown in Table 3. If the branch is actually taken, the counter is incremented (e.g., state 00 becomes state 01). If the branch is not taken, the counter is decremented. The counter saturates at 00 and 11 (a not-taken branch while in the 00 state keeps the 2-bit counter in the 00 state). Two-bit counter states: 00,01 for Not Taken; 10,11 for Taken.**

3. **If array = {0, 1, -3, 4, 1}, what is the prediction accuracy for the two branches found in the above code for five iterations of the loop, using the 512-entry BHT described above?**

   With a 512-entry BHT, both branches will safely index different entries in the BHT. 5 iterations.

   The for loop will be {T,T,T,T,N} (taken, not-taken). However, the 2-bit counter will predict {N,N,T,T,T} as it takes two cycles to learn the branch is taken. So the predictor is only accurate on 2 out of the 5 predictions. The if branch will be {T,N,N,N,N}. The 2-bit counter will predict {N,N,N,N,N}. So it is accurate 4 out of 5 times. BEQZ (if) (4/5) = 80%; BNEZ (for loop) (2/5) = 40%.

4. **If array = {0,1,-3,4,1}, what is the prediction accuracy for the two branches found in the above code for five loop iterations, using a ONE-entry BHT (i.e., all branches map to the same two-bit entry).**

   With a one-entry BHT, both branches will alias to the same entry in the BHT. The for loop will be {T,T,T,T,N}. The if branch will be {T,N,N,N,N}. Here's a table tracking the 2-bit predictor state ("00→01" means that it started in the 00 state, branch was taken, so it ended up in the 01 state):

	0	1	-3	4	1
if	00→01	10→01	10→01	10→01	10→01
for	01→10	01→10	01→10	01→10	01→00

   And here's a table tabulating when the predictor was correct:

	0	1	-3	4	1
if	X	X	X	X	X
for	X	X	X	X	√

   We can see that the if-branch was 0 for 5, and the for-loop-branch was 1 out of 5.

   BEQZ (if) (0/5) = 0%; BNEZ (for loop) (1/5) = 20%.

5. **For this part, assume that the compiler can specify statically which way the processor should predict the branch will go. If the processor sees a "branch-likely" hint from the compiler, it predicts the branch is taken and does NOT update the BHT with this branch (i.e., any branches the compiler can analyze do not pollute the BHT). Which branches, if any, can the compiler provides hints for, if the input array for the compiler's test runs varies widely (assume the compiler must be fairly confident in the accuracy of a branch to be predicted statically before it labels a branch)?**

   a) The BEQZ branch (if)

   b) The BNEZ branch (for loop)

   c) Both branches

   d) Neither branch

   The answer is b. The if-branch is data dependent, and the test arrays are specified as being relatively random and thus hard to predict. The for-loop will always be taken except for the last iteration, which makes it easy to predict.

6. **Adding a BTB (branch target buffer): The BTB can hold two entries (fully-associative), is indexed by the current PC, and allows the processor's fetch stage to immediately fetch along the target PC's path if the entry tagged by the current PC is found in the BTB. For this piece of code, and assuming the baseline is a processor with only one entry for a BHT, would it be more advantageous to add a BTB, or would it be better to add static hints from the compiler?**

   Depending on your justifications, the answer could be either the BTB or the static hints.

   BTB: The key is to identify that while the actual prediction rates of both would be about the same (especially regarding the for loop), the BTB can redirect the instruction stream much earlier (as it only relies on the current PC), and thus hide a lot of the branch latency. Meanwhile, the BHT/static hints must wait until branch address calculation and instruction decode, which can be many, many cycles later. Thus, the BTB is a win.

-- or --

Static Hints: Some students recognized while both schemes have about the same prediction accuracy, the BTB takes up some area and may not be worth the area/power/timing costs, if that is what we are considering to be the more important metric.

-- or --

Static Hints: Depending upon the interpretation of the interaction of the BTB and BHT, one can make a case for using the one-entry BHT augmented with static hints. The argument is as follows: the BTB can redirect immediately and will accurately (and quickly) redirect the for loop branch. However, the BHT has the power to overrule the BTB later in the pipeline, if it disagrees. The BHT for this problem has only one entry, and because it is aliased by both the always-taken for loop and the unpredictable if branch, the BHT will provide unpredictable predictions. If the BHT predicts "not-taken" for the for loop, it will undo all of the correct predictions from the BTB, which won't be resolved until the Execute stage (if you assume the BHT is allowed to overrule the BTB).

A real processor will make sure the BHT is much larger (in address coverage) than the BTB to prevent this very issue (although addresses can still alias), because the BHT should technically provide much better prediction abilities than the BTB. It is also possible that the BHT may only overrule the BTB if the BTB predicts not-taken.

Note: The BTB can have its own prediction bits, so it can invalidate entries it learns are not taken (so it won't always mispredict the if branch is it sees one zero, followed by a bunch of non-zeros).

VLIW machines

The program we will use for this problem is listed below (In all questions, you should assume that arrays **A**, **B** and **C** do not overlap in memory).

C code

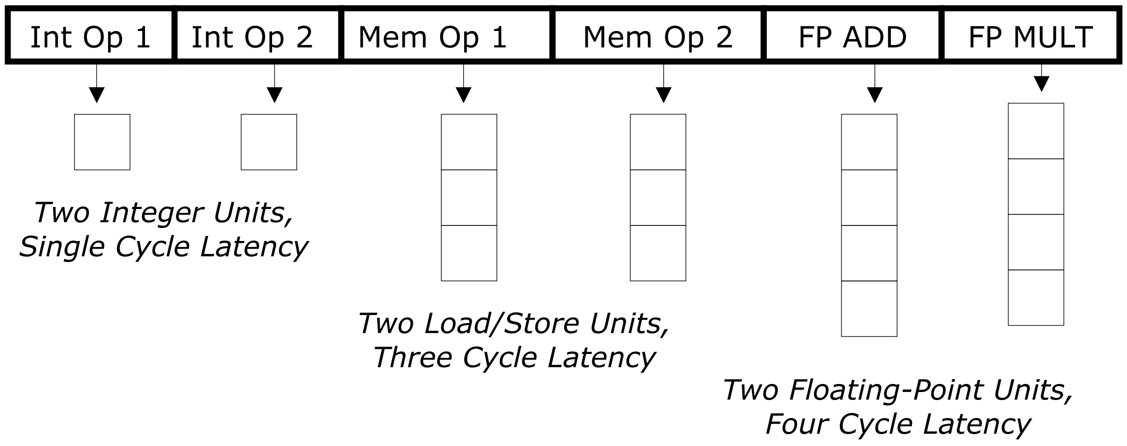
```
for (i=0; i<328; i++)
   A[i] = A[i] * B[i];
   C[i] = C[i] + A[i];
```

In this problem, we will deal with the code sample on a VLIW machine. Our machine will have six execution units:

- Two ALU units, latency one cycle, also used for branch operations
- Two memory units, latency three cycles, fully pipelined, each unit can perform either a store or a load
- Two FPU units, latency four cycles, fully pipelined, one unit can perform **fadd** operations, the other **fmul** operations.

Our machine has no interlocks. The result of an operation is written to the register file immediately after it has gone through the corresponding execution unit: one cycle after issue for ALU operations, three cycles for memory operations and four cycles for FPU operations. The old values can be read from the registers until they have been overwritten.

Below is a diagram of our VLIW machine:


Int Op 1
Int Op 2
Mem Op 1
Mem Op 2
FP ADD
FP MULT
Two Integer Units, Single Cycle Latency
Two Load/Store Units, Three Cycle Latency
Two Floating-Point Units, Four Cycle Latency

The program for this problem translates to the following VLIW operations:

```
loop:   1.  ld f1, 0(r1)        ; f1 = A[i]
        2.  ld f2, 0(r2)        ; f2 = B[i]
        3.  fmul f4, f2, f1     ; f4 = f1 * f2
        4.  st f4, 0(r1)        ; A[i] = f4
        5.  ld f3, 0(r3)        ; f3 = C[i]
        6.  fadd f5, f4, f3     ; f5 = f4 + f3
        7.  st f5, 0(r3)        ; C[i] = f5
        8.  add r1, r1, 4       ; i++
        9.  add r2, r2, 4
       10.  add r3, r3, 4
       11.  add r4, r4, -1
       12.  bnez r4, loop       ; loop
```

1. **Table shows our program rewritten for our VLIW machine, with some operations missing (instructions 2, 6 and 7). We have rearranged the instructions to execute as soon as they possibly can, but ensuring program correctness. Please fill in missing operations.**

ALU1	ALU2	MU1	MU2	FADD	FMUL
add r1, r1, 4	add r2, r2, 4	ld f1, 0(r1)	**ld f2, 0(r2)**		
add r3, r3, 4	add r4, r4, -1	ld f3, 0(r3)			
					fmul f4, f2, f1
			st f4, -4(r1)	**fadd f5, f4, f3**	
	bnez r4, loop	**st f5, -4(r3)**			

2. **How many cycles are required to complete one iteration of the loop in steady state? What is the performance (flops/cycle) of the program?**

   12 cycles, 2/12=0.17 flops per cycle

3. **How many VLIW instructions would the smallest software pipelined loop require? Explain briefly. Ignore the prologue and the epilogue. Note: You do not need to write the software pipelined version.**

   3 instructions, because there are 5 memory ops and 5 ALU ops, and we can only issue 2 of those per instruction. The answer of is not 4 (longest latency operation FMUL), because the memory unit will become the bottleneck first. Here is the resulting code:

add r1, r1, 4	add r2, r2, 4	ld f1, 0(r1)	ld f2, 0(r2)		fmul f4, f2, f1
add r3, r3, 4	add r4, r4, -1	ld f3, -4(r3)	st f4, -8(r1)	fadd f5, f4, f3	
	bnez r4, loop		st f5, -12(r3)		

   for a particular **i**, white background corresponds to first iteration of the loop, grey background to the second iteration, yellow background to third, and blue to fourth. Note, one does not need to write the code to get an answer, because it's just a question of how many instructions are needed to express all the operations.

4. **What would be the performance (flops/cycle) of the program? How many iterations of the loop would we have executing at the same time?**

   (2 flops)/(3 cycles per iteration)=0.67 flops per cycle, 4 iterations at a time (total iteration length)/(min cycles per iteration) = 12/3=4

5. **If we unrolled the loop once, would that give us better performance? How many VLIW instructions would we need for optimal performance? How many flops/cycle would we get? Explain.**

If we unrolled once, while still software pipelining, we need 5 instructions to execute two iterations – we get 4/5=0.8 flops/cycle.

6. **What is the maximal performance in flops/cycle for this program on this architecture? Explain.**

Same as above: 0.8 flops/cycle. We are fully utilizing the memory units (they are the bottleneck), so we can't execute more loops/cycle.

7. **If our machine had a rotating register file, could we use fewer instructions than in Question F and still achieve optimal performance? Explain.**

No. We need to unroll the loop once to have an even number of memory ops. Using rotating registers would not allow us to squeeze more memory ops per iteration, so we'd still need 5 instructions per iteration of the program's flow. The code size in memory might be made smaller.

8. **Imagine that memory latency has just increased to 100 cycles. Circle how many instructions (approximately) an optimal loop would require. (no rotating register file, ignoring prologue/epilogue). Explain briefly. 5 50 100 200**

The correct answer is 5, because increasing the latency of one unit doesn't hurt its throughput, so it can still sustain the same throughput as before. The only complication is if we have enough registers (in this case we do). Without interlocks, we can use the registers just as values come in for them, using the execution units to "store" the loops (inside pipeline registers).

Multithreading

This problem evaluates the effectiveness of multithreading using a simple database benchmark. The benchmark searches for an entry in a linked list built from the following structure, which contains a key, a pointer to the next node in the linked list, and a pointer to the data entry.

```
        struct node
                int key;
                struct node *next;
                struct data *ptr;
```

The following MIPS code shows the core of the benchmark, which traverses the linked list and finds an entry with a particular key. Assume MIPS has no delay slots.

```
        ; R1: a pointer to the linked list
        ; R2: the key to find
loop:   LW      R3, 0(R1)           ; load a key
        LW      R4, 4(R1)           ; load the next pointer
        SEQ     R3, R3, R2          ; set R3 if R3 == R2
        BNEZ    R3, End         ; found the entry
        ADD     R1, R0, R4
        BNEZ    R1, Loop            ; check the next node
End:
         ; R1 contains a pointer to the matching entry or zero if not found
```

We run this benchmark on a single-issue in-order processor. The processor can fetch and issue (dispatch) one instruction per cycle. If an instruction cannot be issued due to a data dependency, the processor stalls. Integer instructions take one cycle to execute and the result can be used in the next cycle. For example, if SEQ is executed in cycle 1, BNEZ can be executed in cycle 2. We also assume that the processor has a perfect branch predictor with no penalty for both taken and not-taken branches.

1. **Assume that our system does not have a cache. Each memory operation directly accesses main memory and takes 100 CPU cycles. The load/store unit is fully pipelined, and non-blocking. After the processor issues a memory operation, it can continue executing instructions until it reaches an instruction that is dependent on an outstanding memory operation. How many cycles does it take to execute one iteration of the loop in steady state?**

   Since there is no penalty for conditional branches, instructions take one cycle to execute unless there is a dependency problem. The following table summarizes the execution time for each instruction. From the table, the loop takes **104 cycles** to execute.

Instruction	Start Cycle	End Cycle
LW R3, 0(R1)	1	100
LW R4, 4(R1)	2	101
SEQ R3, R3, R2	101	101
BNEZ R3, End	102	102
ADD R1, R0, R4	103	103
BNEZ R1, Loop	104	104

2. **Now we add zero-overhead multithreading to our pipeline. A processor executes multiple threads, each of which performs an independent search. Hardware mechanisms schedule a thread to**

execute each cycle. In our first implementation, the processor switches to a different thread every cycle using fixed round robin scheduling. Each of the N threads executes one instruction every N cycles. What is the minimum number of threads that we need to fully utilize the processor, i.e., execute one instruction per cycle?

If we have N threads and the first load executes at cycle 1, SEQ, which depends on the load, executes at cycle $2 \cdot N + 1$. To fully utilize the processor, we need to hide 100-cycle memory latency, $2 \cdot N + 1 \geq 101$. The minimum number of thread needed is **50**.

3. **How does multithreading affect throughput (number of keys the processor can find within a given time) and latency (time processor takes to find an entry with a specific key)? Assume the processor switches to a different thread every cycle and is fully utilized. Check the correct boxes.**

	Throughput	Latency
Better	√	
Same		
Worse		√

4. **We change the processor to only switch to a different thread when an instruction cannot execute due to data dependency. What is the minimum number of threads to fully utilize the processor now? Note that the processor issues instructions in-order in each thread.**

In steady state, each thread can execute 6 instructions (SEQ, BNEZ, ADD, BNEZ, LW, LW). Therefore, to hide 98 (104-6) cycles between the second LW and SEQ, a processor needs $\lceil 98/6 \rceil + 1 = \mathbf{18}$ threads.

Multithreading

Consider a single-issue in-order multithreading processor. Each cycle, the processor can fetch and issue one instruction that performs any of the following operations:

- **load/store, 12-cycle latency (fully pipelined)**
- **integer add, 1-cycle latency**
- **floating-point add, 5-cycle latency (fully pipelined)**
- **branch, no delay slots, 1-cycle latency**

The processor **does not have a cache**. Each memory operation directly accesses main memory. If an instruction cannot be issued due to a data dependency, the processor stalls. We also assume that the processor has a perfect branch predictor with no penalty for both taken and not-taken branches.

You job is to analyze the processor utilizations for the following two thread-switching implementations:

Fixed Switching: the processor switches to a different thread every cycle using fixed round robin scheduling. Each of the N threads executes an instruction every N cycles.

Data-dependent Switching: the processor only switches to a different thread when an instruction cannot execute due to a data dependency.

Each thread executes the following MIPS code:

```
loop:       L.D         F2, 0(R1)          ; load data into F2
            ADDI        R1, R1, 4          ; bump source pointer
            FADD        F3, F3, F2         ; F3 = F3 + F2
            BNE         F2, F4, loop       ; continue if F2 != F4
```

5. **What is the minimum number of threads that we need to fully utilize the processor for each implementation?**
   Fixed Switching: __________________ Thread(s)
   Data-dependent Switching: __________________ Thread(s)

   Fixed Switching: _________6________ Thread(s)

   If we have N threads and L.D. executes at cycle 1, FADD, which depends on the load executes at cycle 2N + 1. To fully utilize the processor, we need to hide 12-cycle memory latency, $2N + 1 \geq 13$. The minimum number of thread needed is 6.

   Data-dependent Switching: _________4_________ Thread(s)

   In steady state, each thread can execute 4 instructions (FADD, BNE, LD, ADDI). Therefore, to hide 11 cycles between ADDI and FADD, a processor needs $\lceil 11/4 \rceil + 1 = 4$ threads.

6. **What is the minimum number of threads that we need to fully utilize the processor for each implementation if we change the load/store latency to 1-cycle (but keep the 5-cycle floating-point add)?**
   Fixed Switching: __ Thread(s); Data-dependent Switching: ______ Thread(s)

   Fixed Switching: ___2___ Thread(s). Each FADD depends on the previous iteration's FADD. If we have N threads and the first FADD executes at cycle 1, the second FADD executes at cycle 4N + 1. To fully utilize the processor, we need to hide 5-cycle latency, $4N + 1 \geq 6$. The minimum number of thread needed is 2.

   Data-dependent Switching: __2__ Thread(s). In steady state, each thread can execute 4 instructions (FADD, BNE, LD, ADDI). Therefore, to hide 1 cycle between ADDI and FADD, a processor needs $\lceil 1/4 \rceil + 1 = 2$ threads.

7. **Consider a Simultaneous Multithreading (SMT) machine with limited hardware resources. Circle the following hardware constraints that can limit the total number of threads that the machine can support. For the item(s) that you circle, briefly describe the minimum requirement to support N threads.**
   a) **Number of Functional Unit:**
   b) **Number of Physical Registers:**
   c) **Data Cache Size:**
   d) **Data Cache Associatively:**

(A) Number of Functional Unit:	Since not all the treads are executed each cycle, the number of functional unit is not a constraint that limits the total number of threads that the machine can support.
(B) Number of Physical Registers:	**We need at least [N × (number of architecture registers)] physical registers for an in-order system. Since it is SMT, it is actually least [N × (number of architecture registers) + 1] physical registers, because you can't free a physical register until the next instruction commits to that same architectural register.**
(C) Data Cache Size:	This is for performance reasons.
(D) Data Cache Associatively:	This is for performance reasons.

Out of Order Superscalar Processors

Mark whether the following modifications will cause each of the categories to increase, decrease, or whether the modification will have no effect. You can assume the baseline processor is a standard out-of-order, superscalar processor with register renaming and branch prediction. Explain your reasoning to receive full credit. Assume that in each case the rest of the machine remains unchanged.

	Instructions / Program	Seconds / Cycle	Cycles / Instruction
wider instruction issue	no change doesn't affect the ISA	increases a wider instruction issue complicates the wiring of the issue logic	decreases Ideally issuing more instructions per cycle will decrease CPI (i.e., IPC is increasing). This is the goal of widening issue.
more physical registers	no change doesn't affect the ISA	increases larger register file will mean longer to read from and write to. physical register specifiers will also have to be larger, increasing the size of the micro-ops, tag compares, etc.	decreases More physical registers provide more renaming resources, which will help prevent stalls due to running out of available physical registers.
add more entries to the branch target buffer (BTB)	no change doesn't affect the ISA	increases BTB entries are large in size and require an expensive tag comparison - or - no change, if BTB is argued to not be on the critical path	decreases Ideally, adding more entries allows the processor to accurately redirect the PC on a larger number of branches earlier in the pipeline, cutting down on mispredicts and bubbles caused by waiting on the target address calculation.
add static branch hints to the compiler	no change, since we are just adding a new opcode/type of branch to provide a hint on existing branches - or - increase, as the compiler re-factors and possibly duplicates code to make branches more predictable	increases slightly more complicated decode - or - unchanged, doesn't change logic much and decode is probably not on the critical path, or can assume no changes to ISA	decreases Static hints are added to help the processor better predict branches that can be analyzed at compile-time.
recompile software with a newer, better version of an optimizing compiler	most likely increase many performance optimizations (loop unrolling, register blocking, software pipelining, etc.) will cause the code to inflate - or - however, one could argue that some optimizations, particularly if aiming for "-Os" type optimizations, could shrink the code size, or that optimizing processors will be better at removing redundant code	no change no change is made to the actual hardware	decreases Ideally, your new compiler does a better job than the last one at providing better performance, in the way of hiding instruction latencies (e.g., loop unrolling) or making branches easier to predict, which will affect the measured CPI for a given program. - or - if you argued that your new compiler optimizes for code size, then one could easily imagine the tradeoff was smaller code size for a higher CPI - or - increases, as CPI becomes worse because unoptimized code is easier for hardware to accelerate.

7. Given the following instruction sequence of three threads, fill out the scheduling table for each of the three types of multithreading techniques and determine the total number of cycles.

Thread 1:	Thread 2:	Thread 3:
□ □	◇ ◇	○
□ □ □	◇◇◇	○ ○
□ □	(stall)	○ ○ ○
□ □ □	◇ ◇ ◇	(stall)
(stall)	◇	○ ○
(stall)	◇	(stall)
□ □	◇ ◇	○ ○ ○
(stall)	(stall)	○ ○
□□□	◇	

	Course-grained Multithreading			Fine-grained Multithreading			Simultaneous Multithreading		
1.									
2.									
3.									
4.									
5.									
6.									
7.									
8.									
9.									
10.									
11.									
12.									
13.									
14.									
15.									

Importance of Features

For the following snippets of code, select the single architectural feature that will most improve the performance of the code. Explain your choice, including description of why the other features will not improve performance as much and your assumptions about the machine design. The features you have to choose from are: out-of-order issue with renaming, branch prediction, and superscalar execution. Loads are marked whether they hit or miss in the cache.

```
        ADD.D F0, F1, F8
        ADD.D F2, F3, F8
        ADD.D F4, F5, F8
        ADD.D F6, F7, F8
```

Circle one:

- Out-of-Order Issue with Renaming
- Branch Prediction
- Superscalar

Superscalar because the instructions have no dependencies so they could be issued in parallel (superscalar could deliver speedup). The other two techniques will waste hardware because they will offer no speedup. Out-of-Order with renaming will not help because there are no dependencies. Branch prediction is useless since there are no branches.

```
loop:   ADD R3 R4 R0
        LD R4, 8(R4) # cache hit
        BNEQZ R4, LOOP
```

Circle one:

- Out-of-Order Issue with Renaming
- Branch Prediction
- Superscalar

Branch prediction is necessary with a tight loop to prevent bubbles in the pipeline. Superscalars will be limited not only by the branches, but also the WAR and RAW hazards. They will limit how much ILP it can achieve. Out-of-order with renaming will be limited by the branches. Renaming will take care of the WAR hazard, but the

RAW hazard will still limit how much improvement is possible.

```
LD R1 0(R2) # cache miss
ADD R2 R1 R1
LD R1 0(R3) # cache hit
LD R3 0(R4) # cache hit
ADD R3 R1 R3
ADD R1 R2 R3
```

Circle one:

- Out-of-Order Issue with Renaming
- Branch Prediction
- Superscalar

Out-of-order with renaming will let the third, fourth, and fifth instruction run while the cache miss is being handled. When the miss completes it only needs to do the second and sixth instruction. Branch prediction won’t help with getting around the cache miss and there are enough hazards to limit a superscalar.