※ 考生請注意：本試題 □可 ☑不可 使用計算機

「計算機組織與系統」

共九題(六頁)，請在答案卷作一表格如下，並清楚地填入這九個題目的答案，否則不予計分。

題號	答案
1	
2	
3	
4	
5	
6	
7	
8	
9	

1 [10%] Translate the **beq** instructions shown in the following code into an 32-bit binary instruction, provided that the opcode of **beq** is **0x04**.

```
Loop:  add   $8,$9,$10
       add   $8,$9,$10
       add   $8,$9,$10
       add   $8,$9,$10
       add   $8,$9,$10
       add   $8,$9,$10
       beq   $0,$9,Loop
End:   add   $9, $0, $9
```

2. [10%] Let the value of the program counter (i.e., PC) be **0xCF01FC00**. What is the target address of the instruction, **j 0x20**.

3. [10%] The table (Table 1) shown as follows is the main control of a single-cycle processor which cannot execute the **j** instruction. What are the values of **RegDst**, **ALUsrc**, **MemtoReg**, **RegWrite**, **MemRead**, **Branch** and **MemWrite** for executing the **j** instruction?

(背面仍有題目,請繼續作答)

※ 考生請注意：本試題 □可 ☑不可 使用計算機

Table 1: Control of a single-cycle processor

Instruction	RegDst	ALUSrc	MemtoReg	Reg Write	Mem Read	Mem Write	Branch	ALUOp1	ALUOp0
R-format	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

4. [20%] A *multiprocessor* system is a system consisting of several (identical) processors in a single machine. A *shared-memory* multiprocessor is a multiprocessor machine in which processors share the single memory address space. The processors in a shared-memory multiprocessor communicate through shared memory variables, with all processors capable of accessing any memory location via load (e.g., `lw`) and store (or `sw`) instructions.

   Single address space multiprocessors come in two styles. The first takes the same time to access memory no matter which processor requests it and no matter which memory block is requested. Such machines are called *uniform memory access* (UMA) multiprocessors or *symmetric multiprocessors* (SMP). In the second style, some memory accesses are faster than others depending on which processor asks for which word. Such machines are called *nonuniform memory access* (NUMA) multiprocessors. Figure 1 shown as follows illustrates an example of a UMA machine in which there are n processing nodes. These n nodes are interconnected with a shared bus, and each of the nodes can access any memory location in the main memory through the bus. Communication due to coordination of sharing data among processors also relies on the bus. Notably, in Figure 1, each processor is associated with a local cache memory.

※ 考生請注意：本試題 □可 ☑不可 使用計算機

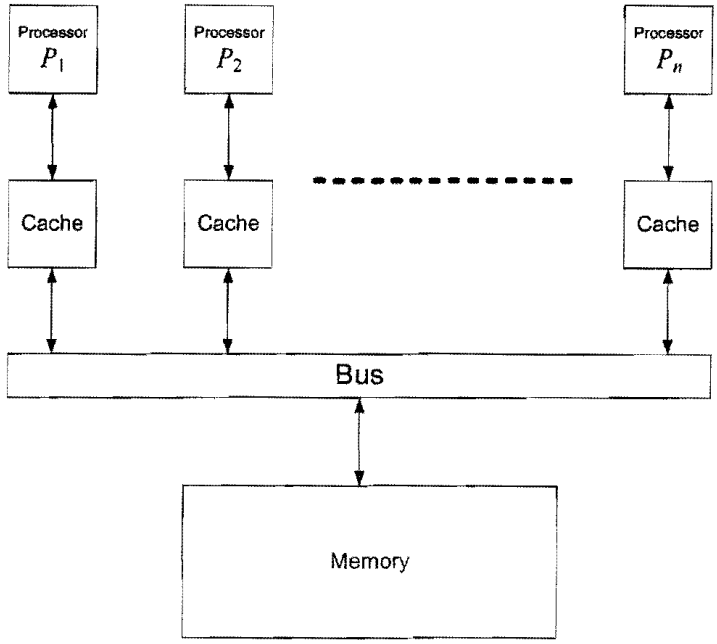

Figure 1: A UMA multiprocessor consisting *n* processing nodes

Conceptually, each processor can fetch a memory block from memory and cache the memory block in its local cache to reduce the latency of upcoming memory references to the block. *Cache coherence protocols* are the methods to guarantee the consistency of the content of a memory block replicated by different caches.

Figure 2 shows an example of a cache coherence protocol (may not be efficient) which is represented as a finite state machine. In the coherence protocol, each cache block has one of the following three states:

- **`Clean`**: This cache block is clean.
- **`Dirty`**: This cache block is dirty due to the store instruction.
- **`Invalid`**: This cache block does not have valid data.

The three states of the protocol in Figure 2 shows the state transitions based on processor actions as opposed to transitions based on bus operations. In default, each cache block is in the state of **`Invalid`**.

※ 考生請注意：本試題 □可 ☑不可 使用計算機

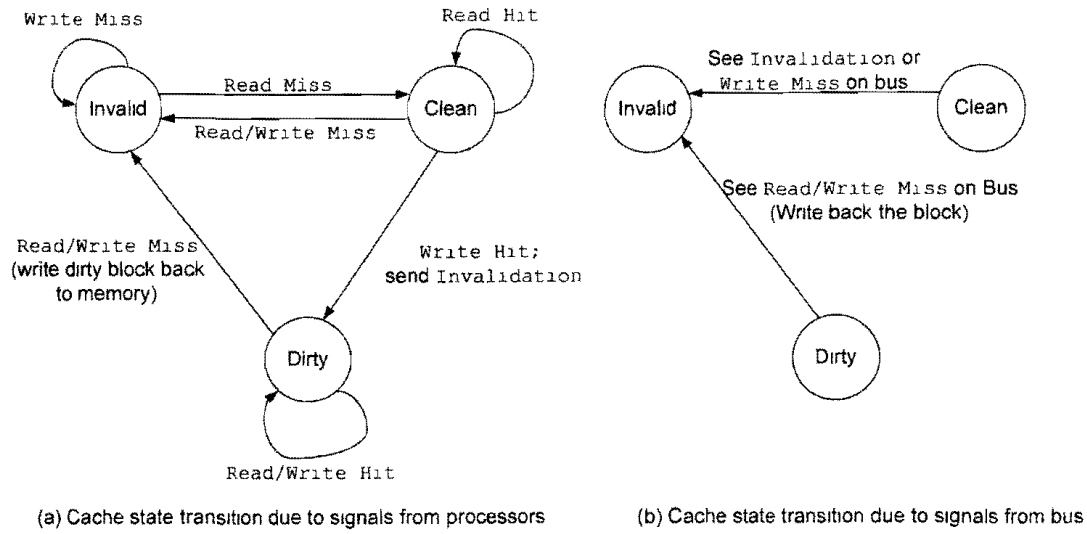

Figure 2: The write-invalidate cache coherence protocol

Transitions between the states of a cache block happen on read misses, write misses, or write hits; read hits do not change cache state. Consider a read miss by a processor *Pi* (where $1 \leq i \leq n$). If the read miss is to an **Invalid** block in *Pi*'s cache, the read miss is then satisfied by reading from memory, and the state of the fetched block is set to **Clean** in *Pi*'s cache (see Figure 2(a)). If the read miss goes to a **Clean** block (in *Pi*'s cache), then *Pi* invalidates the block and fetches the requested block from memory directly without caching. Otherwise, if the read miss is to a **Dirty** block, *Pi* writes back the **Dirty** block to memory and invalidates the block in its cache. Similarly, *Pi* then fetches the requested block from memory directly without caching. On the other hand, while *Pi* has a read miss, *Pi* injects the read miss signal onto bus (Figure 2(b)). All the caches in the other processors monitor the read miss signal to see if the memory block requested by *Pi* is in their caches. If one has a copy and it is in the **Dirty** state, then the block is written back to memory and its state is changed to **Invalid**. Notably, in this protocol the memory block is read directly from memory whether a copy is in a cache or not in this protocol.

Consider write hits. A write hit to a **Dirty** block causes no protocol action (Figure 2(a)). A write hit to a **Clean** block causes the cache to send an **Invalidation** signal onto bus to knock out any other copies, modify the portion of the cache block, and change the state of the cache block to **Dirty** (see Figures 2(a) and (b)).

Regarding write misses, a write miss due to *Pi* to an **Invalid** block in *Pi*'s cache causes *Pi* to send

a write miss signal and update the block in memory directly without caching (Figure 2(a)). Meanwhile, if there exist caches other than *Pi* having the block in the **Clean** state, then by snooping the write miss signal these caches invalidate their cached blocks (Figure 2(b)). Possibly, *Pi*'s write miss goes to a cache having the requested block in the **Dirty** state. If so, that cache writes back and then invalidates the block. A write miss by *Pi* to a **Clean** block causes *Pi*'s cache to invalidate the block, and update the requested block in memory directly (Figure 2(a)). Similarly, if there exist other caches having the block in the **Clean** state, these caches invalidate their cached blocks (Figure 2(b)). If a write miss to a **Dirty** block, then the protocol shown in Figure 2(a) writes back the **Dirty** block, resets the state of the cache block as **Invalid**, and then updates the requested block directly in memory.

Now consider the following portions of two different programs running at the same time on two processors in a 2-node UMA multiprocessor implementing the cache coherence protocol illustrated above. Assume that before this code is run, both x and y are the integer value of 15.

Processor 1. ..., $x = x + 2$; $y = x + y$; ...
Processor 2: ..., $y = x + 2$; ...

What are the possible resulting values of x and y?

5. [10%] Consider a file currently consisting 100 blocks. Assume that the file control block (and the index block, in the case of indexed allocation) is already in memory. For the **contiguous**, **linked**, and **indexed** (single-level) allocation strategies, calculate how many disk block IO operations are required to add a block to the middle of the file (i.e., the new block will become block 50 of the file). In the contiguous allocation case, assume that there is no room to grow at the beginning but there is room to grow at the end. Also assume that the block to be added is stored in memory.

6. [10%] Consider the following page reference string: 1, 2, 3, 4, 2, 1, 5, 6, 2, 1, 2, 3, 7, 6, 3, 2, 1, 2, 3, 6. How many page faults would occur for the LRU replacement algorithm, assuming 4 frames and all the frames are initially empty.

7. [10%] Consider a system running ten I/O-bound user tasks and one CPU-bound user task. Assume that the I/O-bound tasks issue an I/O operation once for every millisecond of CPU computing and that each I/O operation takes 10 milliseconds to complete. Also assume that the context switching overhead is 0.1millisecond and that all processes are long-running tasks. What is the CPU utilization

※ 考生請注意：本試題 □可 ☑不可 使用計算機

for a round-robin scheduler when the time quantum is 10 milliseconds? Note that the CPU utilization is defined as the percentage of CPU time spent on user tasks.

8. [10%] Describe the immutable-shared-file semantics for shared access to files that are stored on remote file systems. What are the advantages and disadvantages of the semantics?

9. [10%] Consider the following set of processes with different CPU-burst time and arrival time:

Process	Burst Time	Arrival Time
*P*1	10	0
*P*2	1	1
*P*3	3	2
*P*4	1	3
*P*5	5	4

What is the waiting time of the processes under the **preemptive SJF** scheduling algorithm?