本試題是否可以使用計算機： □可使用 ， [✓ 不可使用] （請命題老師勾選）

1. (10%) The following program tries to copy words from the address in register $a0 to the address in register $a1 and count the number of words copied in register $v0. The program stops copying when it finds a word equal to 0. You do not have to preserve the contents of registers $v1, $a0, and $a1. This terminating word should be copied but not counted.

```
Loop :   lw       $v1 , 0($a0)
         addi     $v0 , $v0 , 1
         sw       $v1 , 0($a1)
         addi     $a0 , $a0 , 1
         addi     $a1 , $a1 , 1
         bne      $v1 , $zero , loop
```

There are multiple bugs in this MIPS program. Please fix them and turn in bug-free version.

2. (20%) (a) Fill in the following table using the index provided in the keywords (1)-(5) to determine the 3-bit Booth algorithm. Assume that you have both the multiplicand and twice the multiplicand already in registers.
(1) None (2) Add the multiplicand (3) Add twice the multiplicand (4) Subtract the multiplicand (5) Subtract twice the multiplicand (8%)
(b) Assume x is "010101_2" and y is "011011_2". Please use 2-bit and 3-bit Booth algorithm to do the y*x operation. (8%)
(c) Will the 3-bit Booth algorithm always have a fewer operations than the 2-bit Booth algorithm? Justify your answer by a brief description. (4%)

Current bits		Previous bit	Operation
a_{i+1}	a_i	a_{i-1}	
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

（考生必須在"作答卷"上作答）

3. (10%) Define zero, de-normalized number, floating point number, infinity, and NaN (Not a Number) in IEEE 754 double precision format by giving the range of their exponents and significands, respectively. Fill in your answer in the following table. (考生必須在"作答卷"上作答)

	zero	de-normalized	floating point	infinity	NaN
exponent					
significand					

(背面仍有題目,請繼續作答)

本試題是否可以使用計算機： ☐可使用 ， [✓ 不可使用]　（請命題老師勾選）

4. (10%)Consider a CPU with the following instructions：

Instruction	Example	Meaning
add	add $1, $2, $3	$1 = $2+$3
sub	sub $1, $2, $3	$1 = $2-$3

There are five pipeline stages：

（1）IF-Instruction fetch
（2）ID-Instruction decode and register fetch
（3）EX-Execution or calculate effective address
（4）MEM-Access data memory
（5）WB-Write back to registers

Now, consider a program segment S：

```
add $1, $2, $3
sub $4, $1, $5
add $6, $1, $7
add $8, $4, $1
sub $2, $7, $9
```

（a）If we stall the pipeline when there is a data hazard (no forwarding), how many cycles will it take to complete this program segment. Draw the resulting pipeline. (5%)

（b）Is it possible to produce the same result in few cycles by reordering instruction? If so, show the reordering, depict the new pipeline and indicate how many cycles it will take to complete this program segment S. (5%)

本試題是否可以使用計算機：□可使用，[✓ 不可使用]（請命題老師勾選）

「作業系統」

先看這裡：　共三題，請務必將每一題的答案明顯地標示，以利評分。
請勿將計算機組織之作答區與作業系統之作答區混在一起。
（即按題目順序作答）

5. (14 %) Determining the optimal page size of a memory requires balancing some competing factors. The memory overhead of paging consists of the page table (increased as page size gets smaller) and the internal fragmentation waste (increased as page size gets larger). Suppose that the average process size is 2M bytes, page entry size is 4 bytes, the average wasted memory in the last page of each process due to internal fragmentation is a half page.
   (a) What is the optimal page size, in bytes? (10%)
   (b) What is the minimum overhead, in bytes, for an average process? (4%)
   For clearness and grading, please **highlight** or underline your answer in your answer sheet. (請在作答卷中明顯地標示你的答案，以利評分)

6. (16 %) Consider a snapshot of the system as shown below. Upon the next request, P_1 makes a request for (0 4 2 0) and is granted immediately. What is the minimum number of **initial instances** (i.e. the total number of instances before any process enters the system) for each resource type to make sure the system still in a safe state after P_1's request is granted? **(答案請以(a b c d)方式表示)**

	Allocation	Max
	A B C D	A B C D
P_0	0 0 1 2	0 1 1 2
P_1	1 0 0 0	1 7 5 0
P_2	1 3 5 4	2 4 5 6
P_3	0 4 3 2	0 7 5 2
P_4	0 0 1 4	0 6 5 6

7. (20 %) Consider the following synchronization problem. In this problem, there is one dispatcher who decides the access ordering among n tasks ($\tau_1, \tau_2, \ldots, \tau_n$). The dispatcher does so by randomly generating a total-ordering sequence **Who[1..*n*]**, in which Who[i] = j means the i-th task entering the critical section is τ_j. In addition, we provide another array **Order[1..n]** to perform the inverse function. Namely, Order[i]= j means τ_i is the j-th task in the ordering. Naturally, Order[Who[i]]=i and Who[Order[j]] = j, $\forall i, j$. Task τ_i may enter the critical section if each task τ_j whose Order[j] < Order[i] has exited the critical section. If a task tried to enter the critical section before its predecessor(s) enter the critical section, it waits. We also provide an array of status flag, **done[1..*n*]** (initialized to all false), to indicate the execution status of each task. After a task τ_i finishes its

(背面仍有題目,請繼續作答)

execution, it sets its corresponding flag done[i] to true. We also make the following assumptions:

- Only one task is allowed at a time in the critical section.
- The dispatcher and each task only execute once.
- The dispatcher finishes its execution before any task starts.
- The dispatcher calls **Generate_Sequence(Order, Who)** to set up the values in Order[1..*n*] and Who[1..*n*] arrays.

Based on the above description, the declaration of variables and the pseudo code of the dispatcher and each task are given below:

```
semaphore mutex, first-delay, second-delay;
int first-count, second-count, Order[1..n], Who[1..n];
boolean done[1..n];
Dispatcher:
        Generate_Sequence(Order, Who);

Task τi :
        wait( mutex);
        while (my_turn(i) ≠ true) do
            /* my_turn(i) is a function that uses Who[], Order[], and done[] arrays */
            /* to check whether Task τi is the next one to enter the critical session or not. */
        begin
            first-count := first-count +1;
            if second-count > 0 then signal( second-delay)
                                 else signal( mutex);
            wait( first-delay);
            first-count := first-count - 1;
            second-count := second-count +1;
            if first-count > 0   then signal( first-delay)
                                 else signal( second-delay);
            wait( second-delay);
            second-count := second-count - 1;
        end;
        S;  /* enter critical session */
        done[i]=true; /* my_turn() checks the done[] array to set the true/false value */
        your_exit_routine_here(); /* releases semaphores */
```

Please complete the code of task τ_i by writing the pseudo code of *your_exit_routine_here()*.(請在作答卷中明顯地註記答案，字體工整，以利評分作業)