國立中山大學九十學年度碩博士班招生考試試題

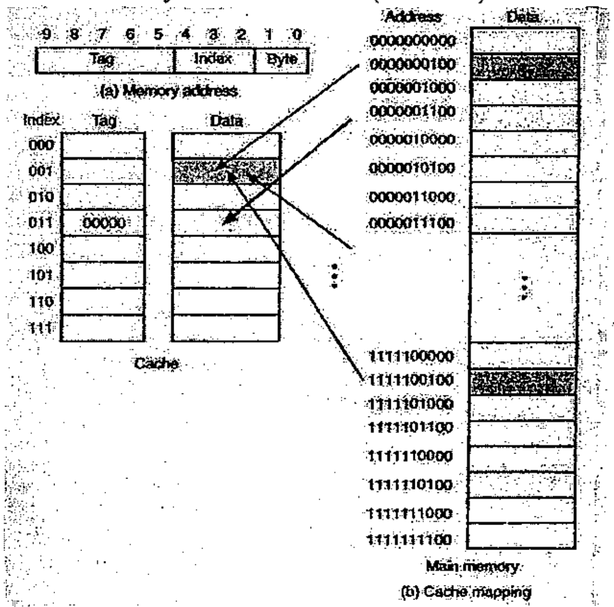
科目：計算機結構【資工系碩士班】

資訊工程研究所 碩士班考試　計算機結構

If some questions are unclear or not well-defined to you, you can make your own assumptions and state them clearly in the answer sheet.

1. Cache (20%)

In general, there are three different cache mapping methods: direct mapped cache, fully-associative cache, and set-associative mapping. A cache system contains three major parts: an index field, a tag field and a data filed. In addition, the line size (block size) defines the number of words being fetched from main memory. For example, the following figure shows the direct mapped cache with a small size of 8 words (4 bytes per word) for a 256-word memory where the line size (block size) is one word (4 bytes).



1.1 Assume a very small cache of 128 bytes (i.e., 32 words) and a small main memory with 1 K bytes (256 words). What are the numbers of bits in the index fields and the tag fields of the cache that is designed using direct mapped method with line size of one word?

1.2 Repeat 1.1 if the cache adopts two-way set associative method with one-word line.

1.3 Repeat 1.2 if the line size is 4 words.

1.4 Give at least two cache-writing strategies and compare their advantages and disadvantages.

2. Computer Architecture Terminologies (10%)

Briefly explain the following terminologies in computer architecture area:

2.1 What is a superscaler CPU? What is a superpipeline CPU? Explain their features

2.2 What is "Out-of-Order Execution In-order Completion"? What is its relationship with the reservation station?

3. Instruction (10%)

Consider the strange instruction: *GLOP a, b*, which implements the operation *AC:= 1/ (a-b)*, where *AC* (accumulator) denotes a general-purpose CPU register. Using GLOP only, it is possible to program all the four basic arithmetic operations: *a+b, a-b, a x b, a/b*. For example, the fact that $a-b=1/(1/(a-b)-0)$ implies that the following two-instruction program implements subtraction:

```
GLOP a, b
GLOP AC, 0
```

Devise a similar program for addition that uses GLOP only and does not contain minus sign among its operands.

4. Assembly Programming (10%)

The following is an assembly program for some microprocessor. What is the operation specified by the program?

```
ARRAY   EQU     $1000       ; set array address
SIZE    EQU     $0064       ; set array size (in words)
SUM     EQU     $00FC       ; set sum address
        MOVE    #ARRAY, A0  ; load array address into register A0
        MOVE    #SIZE, D0   ; load array size into register D0
        MOVE    #0, D1      ; clear D1
LOOP    ADD     (A0), D1    ; add current array element to D1
        ADD     #4, A0      ; increment A0 by 4
        SUB     #1, D0      ; decrement D0 by 1 (and update flags)
        BNE     LOOP        ; continue adding if Z != 1 (implies D0 != 0)
        MOV     D1, SUM     ; store result in memory
```

5. Control Unit (50%)

A control unit can be implemented by a block of logic that takes as inputs the current state and the opcode field of the instruction register and produces as outputs the datapath-control signals and the value of the next state. Fig. 1 is a finite state diagram for the design of the control unit in a microprocessor. With 10 states, we use 4 bits, S_3, S_2, S_1, S_0, to encode the state number with normal binary order, i.e., state 6 is encoded as 0110_2 with $S_3 = 0, S_2 = 1, S_1 = 1, S_0 = 0$. The current state number will be stored in a state register, as shown in Fig. 2. The control logic in Fig. 2 implements two parts of the finite state machine of Fig. 1. One part is the logic that determines the setting of the datapath control outputs, which depend only on the state bits. The other part implements the next-state bits based on the current-state bits and the other inputs (the 6-bit opcode). The opcode fields of the five instructions are given in the following table.

Instruction	Opcode field					
	Op5	Op4	Op3	Op2	Op1	Op0
R format	0	0	0	0	0	0
JMP	0	0	0	0	1	0
BEQ	0	0	0	1	0	0
LW	1	0	0	0	1	1
SW	1	0	1	0	1	1

Fig. 3 shows the logic equations: the top portion showing the datapath control signal outputs, and the bottom portion showing the next-state function.

5.1 Please complete the table in Fig. 3 by filling in the blank portion based on the information provided in Figs. 1 and 2. Copy to your answer sheet only the rows that you fill in.

5.2 Give the Boolean logic equation for the low-order next-state bit, NS0.

The control function can be expressed as a logic equation for each output and can be implemented in two ways: corresponding to a complete truth table (realized in a ROM), or corresponding to a two-level logic minimized structure (realized by a programmable logic array, or PLA).

5.3 What is the size of the ROM if the control function is realized using a ROM?

It is simplest if we break the control function defined in Fig. 3 into two parts: the next-state outputs, which may depend on all the inputs, and the control signal outputs, which depends only on the current-state bits. Fig. 4 shows the truth tables for all the datapath-control signals.

5.4 Fill in the blank parts of the truth tables in Fig. 4. Copy to your answer sheet the rows that you fill in.

Because these signals actually depend only on the state bits, each of the entries in a table in Fig. 4 actually represents 64 entries, with the 6 bits named Op having all the possible values; that is, the Op bits are don't-care bits in determining the datapath-control outputs. Fig. 5 shows the truth table for the next-state bits NS[3-0], which depends on the state input bits and the instruction bits, which supply the opcode.

5.5 Complete the truth table in Fig. 5 by filling the left-behind parts. Copy to your answer sheet the rows that you fill in.

By separating the datapath-control and the next-state outputs, we can design the control unit using a ROM of smaller size. Fig. 6, a re-formulation for the truth tables in Fig. 4, gives the content of the upper 17 bits of each control word in the ROM. The 4-bit input field gives the low-order four address bits of each word and the column gives the contents of the word at that address.

5.6 Complete the truth table in Fig. 6. Copy to your answer sheet the rows that you fill in.

Fig. 7, derived directly from Fig. 6, shows the upper 17 bits of the control word. These datapath-control outputs depend only on the state inputs. Fig. 8 shows the lower 4 bits of the control word corresponding to the next-state outputs. The last column of the table in Fig. 8 corresponds to all the possible values of the opcode that do not match the specified opcodes. In state 0, the next state is always 1, since the instruction was still being fetched. After state 1, the opcode field must be valid. The table in Fig. 8 indicates this by the entries marked illegal.

5.7 Complete Fig. 7. Copy to your answer sheet the rows that you fill in. What is the total size of the ROM if partitioning the control word outputs into two independent parts, as described above?

We can reduce the amount of the control storage required using programmable logic array (PLA) as shown in Fig. 9. In PLA, each output is the logical OR of one or more of the minterms (sometimes called product terms). Thus, a PLA consists of an AND plane to produce all the necessary product terms and an OR plane which generate the OR operations of the relevant product terms for each output signal. The total size of a PLA is proportional to (#inputs x #product terms) + (#outputs x #product terms).

5.8 How many unique minterms (product terms) are in the PLA shown in Fig. 9 that implements the control unit? What is the total size of the PLA?

We observe in Fig. 9 that among the 18 unique product terms, 10 depends only on the current state and 8 others depend on a combination of the OP field and the current-state bits. Thus, we could split the PLA in two PLAs: one (PLA1) with four inputs and 10 product terms that generate the 17 control outputs, and the other (PLA2) with 10 inputs and 8 product terms that generates the 4 next-state outputs.

5.9 What is the size of the first PLA1 and what is the size of the second PLA2? Compare the results of the overall PLA size with that obtained in 5.8. What do you observe?

For the implementation of a control unit using two separate ROMs, most of the hardware cost is spent in the design of the next-state function. Imagine what the control logic unit would look like if the instruction set had many more different instruction types, some of which required many clocks to implement. There would be many more states in the finite state machine. In some states, we might be branching to a large number of different states depending on the instruction type (as we did in state 1 of the finite state machine in Fig. 1). However, many of the states would proceed in a sequential fashion, just as states 3 and 4 do in Fig. 1. An alternative implementation of a control unit is shown in Fig. 10 with the ASL block shown in Fig. 11. Fig. 10 looks remarkably like a computer and the ROM or PLA can be thought of as memory supplying instructions for the datapath. The state can be thought of as an instruction address.

5.10 What is the function of the ASL block in Fig. 10 and Fig. 11? What is the relationship between the number of dispatch ROMs in Fig. 11 and the number of states with more than one outgoing branches? Please give a well-known terminology for such a control unit design (as shown in Fig. 10).

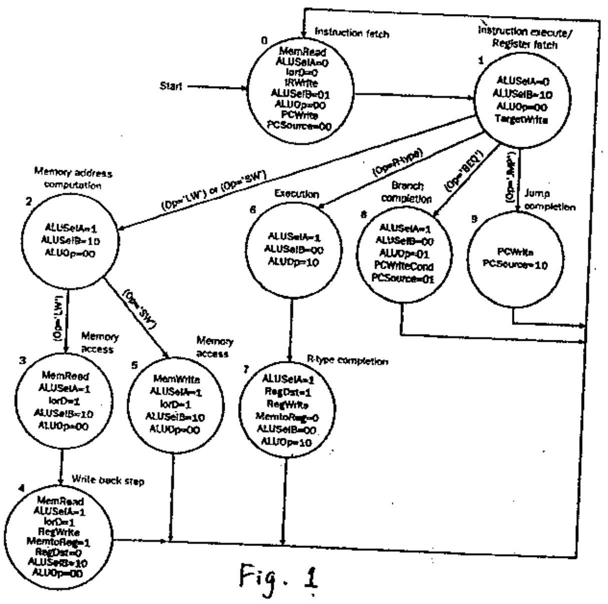

Instruction fetch
Instruction execute/ Register fetch
Start
0
MemRead ALUSelA=0 IorD=0 IRWrite ALUSelB=01 ALUOp=00 PCWrite PCSource=00
1
ALUSelA=0 ALUSelB=10 ALUOp=00 TargetWrite
Memory address computation
(Op='LW') or (Op='SW')
(Op=R-type)
(Op='BEQ')
(Op='JMP')
Execution
Branch completion
Jump completion
2
ALUSelA=1 ALUSelB=10 ALUOp=00
6
ALUSelA=1 ALUSelB=00 ALUOp=10
8
ALUSelA=1 ALUSelB=00 ALUOp=01 PCWriteCond PCSource=01
9
PCWrite PCSource=10
(Op='LW')
(Op='SW')
Memory access
Memory access
R-type completion
3
MemRead ALUSelA=1 IorD=1 ALUSelB=10 ALUOp=00
5
MemWrite ALUSelA=1 IorD=1 ALUSelB=10 ALUOp=00
7
ALUSelA=1 RegDst=1 RegWrite MemtoReg=0 ALUSelB=00 ALUOp=10
Write back step
4
MemRead ALUSelA=1 IorD=1 RegWrite MemtoReg=1 RegDst=0 ALUSelB=10 ALUOp=00

Fig. 1

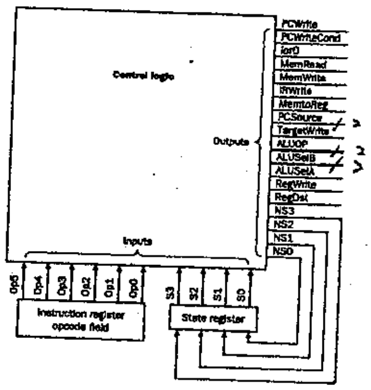


Fig. 2

Output	Current states	Op
PCWrite	state0+state9	
PCWriteCond	state8	
IorD	state3+state4+state5	
MemRead	state0+state3+state4	
MemWrite	state5	
IRWrite	state0	
MemtoReg	state4	
PCSource1	state9	
PCSource0	state8	
TargetWrite	state1	
ALUOp1	state6+state7	
ALUOp0	state8	
ALUSelB1	state1+state2+state3+state4+state5	
ALUSelB0	state0+state1	
ALUSelA	state2+state3+state4+state5+state6+state7+state8	
RegWrite	state4+state7	
RegDst	state7	
NextState0		
NextState1	state0	
NextState2	state1	(Op='lw') + (Op='sw')
NextState3	state2	(Op='lw')
NextState4	state3	
NextState5	state2	(Op='sw')
NextState6	state1	(Op='R-type')
NextState7	state6	
NextState8	state1	(Op='beq')
NextState9	state1	(Op='jmp')

✓

Fig. 3

s3	s2	s1	s0
0	0	0	0
1	0	0	1

a. Truth table for PCWrite.

s3	s2	s1	s0
1	0	0	0

b. Truth table for PCWriteCond.

s3	s2	s1	s0
0	0	1	1
0	1	0	0
0	1	0	1
0	0	1	1

c. Truth table for IorD.

s3	s2	s1	s0

✓ ✓ ✓

d. Truth table for MemRead.

s3	s2	s1	s0
0	1	0	1

e. Truth table for MemWrite.

s3	s2	s1	s0
0	0	0	0

f. Truth table for IRWrite.

s3	s2	s1	s0
0	1	0	0

g. Truth table for MemtoReg.

s3	s2	s1	s0
1	0	0	1

h. Truth table for PCSource1.

s3	s2	s1	s0
1	0	0	0

i. Truth table for PCSource0.

s3	s2	s1	s0
0	0	0	1

j. Truth table for TargetWrite.

s3	s2	s1	s0
0	1	1	0
0	1	1	1

k. Truth table for ALUOp1.

s3	s2	s1	s0
1	0	0	0

l. Truth table for ALUOp0.

s3	s2	s1	s0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1

m. Truth table for ALUSelB1.

s3	s2	s1	s0
0	0	0	0
0	0	0	1

n. Truth table for ALUSelB0.

s3	s2	s1	s0
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0

o. Truth table for ALUSelA.

s3	s2	s1	s0
0	1	0	0
0	1	1	1

p. Truth table for RegWrite.

s3	s2	s1	s0
0	1	1	1

q. Truth table for RegDst.

Fig. 4

Op5	Op4	Op3	Op2	Op1	Op0	S3	S2	S1	S0

✓ ✓

a. The truth table for the NS3 output, active when the next state is 8 or 9. This signal is activated from state 1.

Op5	Op4	Op3	Op2	Op1	Op0	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	1
1	0	1	0	1	1	0	0	1	0
X	X	X	X	X	X	0	0	1	1
X	X	X	X	X	X	0	1	1	0

b. The truth table for the NS2 output, which is active when the next state is 4, 5, 6, or 7. This situation occurs when the current state is one of 1, 2, 3, or 6.

Op5	Op4	Op3	Op2	Op1	Op0	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	1
1	0	0	0	1	1	0	0	0	1
1	0	1	0	1	1	0	0	0	1
1	0	0	0	1	1	0	0	1	0
X	X	X	X	X	X	0	1	1	0

c. The truth table for the NS1 output, which is active when the next state is 2, 3, 6, or 7. The next state is one of 2, 3, 6, or 7 only if the current state is one of 1, 2, or 6.

Op5	Op4	Op3	Op2	Op1	Op0	S3	S2	S1	S0
X	X	X	X	X	X	0	0	0	0
1	0	0	0	1	1	0	0	1	0
1	0	1	0	1	1	0	0	1	0
X	X	X	X	X	X	0	1	1	0
0	0	0	0	1	0	0	0	0	1

d. The truth table for the NS0 output, which is active when the next state is 1, 3, 5, 7, or 9. This happens only if the current state is one of 0, 1, 2, or 6.

Fig. 5

Outputs	Input values (S[3-0])									
	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001
PCWrite	1	0	0	0	0	0	0	0	0	1
PCWriteCond	0	0	0	0	0	0	0	0	1	0
IorD	0	0	0	1	1	1	0	0	0	0
MemRead										
MemWrite	0	0	0	0	0	1	0	0	0	0
IRWrite	1	0	0	0	0	0	0	0	0	0
MemtoReg	0	0	0	0	1	0	0	0	0	0
PCSource1	0	0	0	0	0	0	0	0	0	1
PCSource0	0	0	0	0	0	0	0	0	1	0
TargetWrite	0	1	0	0	0	0	0	0	0	0
ALUOp1	0	0	0	0	0	0	1	1	0	0
ALUOp0	0	0	0	0	0	0	0	0	1	0
ALUSelB1	0	1	1	1	1	1	0	0	0	0
ALUSelB0	1	1	0	0	0	0	0	0	0	0
ALUSelA	0	0	1	1	1	1	1	1	1	0
RegWrite	0	0	0	0	1	0	0	1	0	0
RegDst	0	0	0	0	0	0	0	1	0	0

✓

Fig. 6

Lower 4 bits of the address	Bits 20–4 of the word
0000	10010100000001000
0001	00000000010011000
0010	00000000000010100
0011	00110000000010100
0100	00110010000010110
0101	00101000000010100
0110	00000000001000100
0111	00000000001000111
1000	01000000100100100
1001	

✓

Fig. 7

	Op [5–0]					
Current state S[3–0]	000000 (R format)	000010 (jmp)	000100 (beq)	100011 (lw)	101011 (sw)	Any other value
0000	0001	0001	0001	0001	0001	0001
0001	0110	1001	1000	0010	0010	Illegal
0010	XXXX	XXXX	XXXX	0011	0101	Illegal
0011	0100	0100	0100	0100	0100	Illegal
0100	0000	0000	0000	0000	0000	Illegal
0101	0000	0000	0000	0000	0000	Illegal
0110	0111	0111	0111	0111	0111	Illegal
0111	0000	0000	0000	0000	0000	Illegal
1000	0000	0000	0000	0000	0000	Illegal
1001	0000	0000	0000	0000	0000	Illegal

Fig. 8

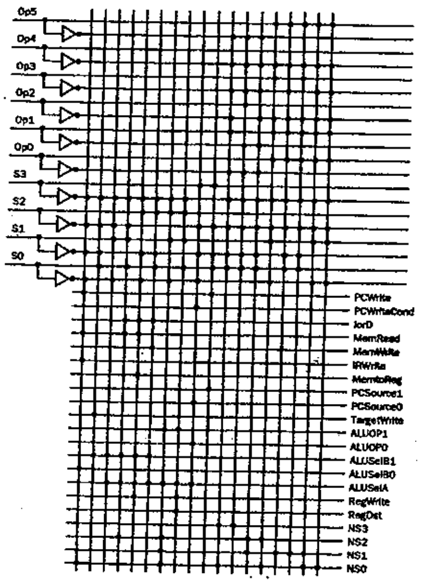


Fig. 9

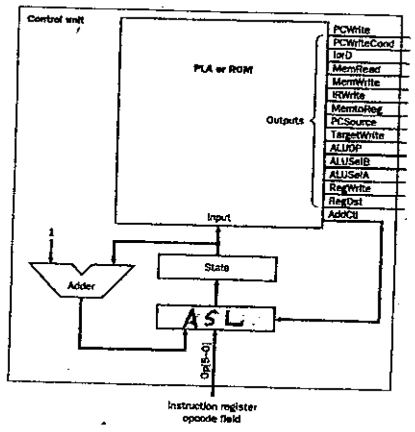


Fig. 10

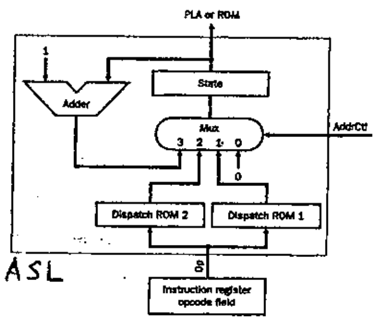


Fig. 11

國立中山大學九十學年度碩博士班招生考試試題

科目：作業系統與資料結構【資工系碩士班】

1. (a)Draw the binary tree whose *inorder* sequence is *ABCDEFGHI* and whose postorder sequence is *ACBGFIHED*. (6%)
   (b) *ACBGFIHED* is called a postorder permutation of *ACBGFIHED*. Please give all postorder permutations of the inorder sequence *ABC* and their corresponding binary trees. (8%)
2. Explain each of the following terms. (9%)
   (a) topological ordering of an acyclic graph
   (b) B-tree
   (c) hashing
3. Let $P=(p_1,p_2,\ldots,p_n)$ be a permutation of $\{1,2,\ldots,n\}$. b_i, $2\le i\le n$, denotes a permutation operator that $b_i(P)=(p_1,p_2,\ldots,p_{i-3},p_{i-2},p_i,p_{i-1},p_{i+1},p_{i+2},\ldots,p_n)$. For example, $b_3(13254)=(12354)$ and $b_5(13254)=(13245)$. A subset of the permutation operators b_i's can be used to sort a permutation into the identity permutation, which is (1,2,3,…,*n*). The *bubble sort* uses a subset of b_i's to perform sorting. For example, we can sort (13254) by applying b_3 followed by b_5, or b_5 followed by b_3. Now, a new set of permutation operators k_i's, $2\le i\le n$, are defined as that $k_i(P)=(p_i,p_{i-1},\ldots,p_3,p_2,p_1,p_{i+1},p_{i+2},\ldots,p_n)$. For example, k_3(13254)= (23154) and k_5(13254)=(45231). Answer the following questions by applying k_i, $2\le i\le n$.
   (a) How do you sort (23514) into (12345)? Please give the sequence of k_i's you use in this sorting. (7%)
   (b) Give a general algorithm which sorts any permutation $P=(p_1,p_2,\ldots,p_n)$ into the identity permutation (1,2,3,…,*n*). (8%)
4. The data elements stored in a *binary search tree* is ordered. For a node *x* in the binary search tree, the elements stored in the left subtree of node *x* are all *less than or equal to* that in *x*, and the elements stored in the right subtree of node *x* are all *greater than* that in *x*. For an input element *y*, please write a C or Pascal program to search the tree to find whether there exists an element equal to *y*. Your program should print out *the level of the node* on the tree if it finds the element and print out "not find" if otherwise. Note that the level of the root is defined as 1 and the level of the sons of the root is 2, and so on. Please point out which language (C or Pascal) you are using to write the program. You can use the following declaration in your C program (You can also use your own declaration). (12%)

```
struct nodetype {
    int   value;
    struct nodetype *left, *right;
}
int binsearch(struct nodetype   p)   /* return level or false */
```

If you write a Pascal program, you can use the following declaration (You can also use your own declaration).

```
type nodeptr=^nodetype;
    nodetype=record
        value: integer;
        left, right: nodeptr;
    end;
function binsearch(p: nodeptr):integer;    (*return level or 0 for 'not find' *)
```

5. Suppose that a 32-bit virtual address is broken up into four fields, *a*, *b*, *c*, and *d*. The first three are used for a three-level page table system. The fourth field, *d*, is the offset. Does the number of pages depend on the sizes of all four fields? If not, which ones matter and which ones do not? (10%)
6. Can a system be in a state that is neither deadlocked nor safe? If so, give an example. If not, prove that all states are either deadlocked or safe. (10%)
7. Explain the difference between internal fragmentation and external fragmentation. Which one occurs in paging systems? Which one occurs in systems using pure segmentation? Explain your answer. (10%)
8. A soft real-time system has periodic events with periods of 50, 100, 200, and 250 ms each. Suppose that the four events require 35, 20, 10 and x ms of CPU time, respectively. What is the largest value of x for which the system is schedulable? (10%)
9. The following pseudocode illustrates a method of implementing mutual exclusion. The strategy uses the shared variable "lock" which is initially set to 0.

```
while (lock)
;
lock = 1;
<Critical Section>
lock = 0;
```

Is the method correct? Justify your answer. (10%)

國立中山大學九十學年度碩博士班招生考試試題

科目：離散數學【資工系碩士班】

只有答案，沒有求解過程不計分。

1. A teacher wants to distribute n different books to the n students in the class. The books are returned and distributed again latter on. In how many ways can the books be distributed so that every student will get different books in the 2 rounds? (10%)

2. A *line segment* in $\mathbf{R}^2$ is a line joining 2 different points in $\mathbf{R}^2$. Show that in any 6 line segments in $\mathbf{R}^2$, either there are 3 line segments in which any pair of them intersect, or there are 3 line segments in which no pair of them intersect. Can this be generalized to line segments in $\mathbf{R}^3$? (15%)

3. Let the sequence of numbers $g_0, g_1, \ldots, g_n, \ldots$ be defined by $g_0 = 1$, $g_1 = 1$ and, for every $n > 1$, $g_n = g_{n-1} + 2g_{n-2} + (-1)^n$. Express g_n in terms of n. (15%)

4. A k-cube is a graph whose vertices are the ordered k-tuples of 0's and 1's, two vertices are adjacent if and only if they differ in exactly one coordinate. A *bipartite graph* is a graph whose vertices can be partitioned into two subsets X and Y so that each edge has one end in X and the other end in Y.

   (a) Draw k-cubes for $k = 1, 2, 3, 4$. (5%)

   (b) Show that every k-cube is bipartite. (10%)

5. A *lattice path* from (x_0, y_0) to (x_n, y_n) in the xy plane is defined as a sequence of points $(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n)$ such that each $x_{i+1} = x_i + 1$, and each $y_{i+1} = y_i \pm 1$, $i = 1, 2, \ldots, n-1$. How many lattices paths are there from $(0, 1)$ to $(10, 3)$? How many of them do not touch or cross the x axis? (15%)

6. Let $S = \{1, 2, \ldots, n\}$.

   (a) Prove that if n is even then any $n/2 + 1$ subset of S contains two numbers whose sum is $n + 1$. (10%)

   (b) In general, determine the value k such that any k subset of S contains two numbers whose sum is $n + 1$. (5%)

7. A theorem of a propositional calculus is a proposition that is always true. Determine which of the followings are theorems. Justify your answers. (15%)

   (a) $(x \to x)$

   (b) $\sim (x \leftrightarrow x)$

   (c) $(((x \to y) \land (\sim x \to y)) \to y)$