考生注意：1. 依次序作答，只要標明題號，不必抄題。
2. 答案必須寫在答案卷上，否則不予計分。
3. 試題隨卷繳回。

The following problems may be answered in Chinese or English. You need to give all details in order to receive credit (point).

1. **(20 points)** Design a 32-bit×32-bit multiplication hardware using one 32-bit Multiplicand register, one 32-bit ALU, one 64-bit Product register, and the control circuit. Based on the multiplication hardware, derive a *signed multiplication* algorithm. Show that your algorithm works by multiplying 1011_{two} by 0010_{two}.
2. **(15 points)** Add $6.42_{ten}\times10^{1}$ to $9.51_{ten}\times10^{2}$, assuming that you have only three significant digits, first with one guard and one round digits and then without them.
3. **(15 points)** Assuming a 32-bit address, design a direct-mapped cache with 4096 blocks and a block size of 16 bytes (4 words). What block number does byte address 65713 map to?
4. **(10 points)** Use of a speculative technique may decrease performance. This is certainly true in particular when a speculative guess is wrong; performance at that point in the program is less than it would have been without speculation. However, it can also be true in general, that is, for a substantial workload such as an entire program. For a speculative technique to improve performance in general, what mathematical condition must be true?
5. **(10 points)** Assume that we have a function for an application of the form $F(i, p)$, which gives the fraction of time that exactly i processors are usable given that a total of p processors are available. This means that.

$$\sum_{i=1}^{p} F(i, p) = 1$$

Assume that when i processors are in use, the application runs i times faster. Rewrite Amdahl's Law so that it gives the speedup as a function of p for some application.

6. **(30 points)** Your task is to compare the memory efficiency of four different styles of instruction set architectures. The architecture styles are:
   - *Accumulator*—All operations occur between a single register and a memory location.
   - *Memory-memory*—All instruction addresses reference only memory locations.
   - *Stack*—All operations occur on top of the stack. Push and pop are the only instructions that access memory; all others remove their operands from the stack and replace them with the result. The implementation uses a hardwired stack for only the top two stack entries, which keeps the processor circuit very small and low cost. Additional stack positions are kept in memory locations, and accesses to these stack positions require memory references.
   - *Load-store*—All operations occur in registers, and register-to-register instructions have three register names per instruction.

   To measure memory efficiency, make the following assumptions about all four instruction sets:
   - All instructions are an integral number of bytes in length.
   - The opcode is always 1 byte (8 bits).
   - Memory accesses use direct, or absolute, addressing.
   - The variable A, B, C, D, are initially in memory.

   a. **(10 points)** Invent your own assembly language mnemonics (Figure P6 provides a useful sample to generalize), and for each architecture write the best equivalent assembly language

code for this high-level language code sequence:

$A = B + C;$

$B = A - D;$

$D = A \times B;$

b. **(10 points)** Label each instance in your assembly codes for part (a) where a value is loaded from memory after having been loaded once. Also label each instance in your code where the result of one instruction is passed to another instruction as an operand, and further classify these events as involving storage within the processor or storage in memory.

c. **(10 points)** Assume the given code sequence is from a small, embedded computer application, such as a microwave oven controller, that uses 16-bit memory addresses and data operands. If a load-store architecture is used, assume it has 16 general-purpose registers. For each architecture answer the following questions: How many instruction bytes are fetched? How many bytes of data are transferred from/to memory? Which architecture is most efficient as measured by code size? Which architecture is most efficient as measured by total memory traffic (code + data)?

Stack	Accumulator	Memory-memory	Load-store
Push *A*	Load *A*	Add *C*, *A*, *B*	Load R1, *A*
Push *B*	Add *B*		Load R2, *B*
Add	Store *C*		Add R3, R1, R2
Pop *C*			Store R3, *C*

Figure P6 The code sequence for $C = A + B$ for four classes of instruction sets.