國立台灣大學九十四學年度碩士班招生考試試題

科目：計算機結構

※注意：請於答案卷上依序作答，並應註明作答之大題及其題號。

1. **(10%)** Compare two memory system designs for a classic 5-stage pipelined processor. Both memory systems have a 4-KB instruction cache. But system A has a 4K-byte data cache, with a miss rate of 10% and a hit time of 1 cycle; and system B has an 8K-byte data cache, with a miss rate of 5% and a hit time of 2 cycles (the cache is not pipelined). For both data caches, cache lines hold a single word (4 bytes), and the miss penalty is 10 cycles.
What are the respective *average memory access times* for data retrieved by load instructions for the above two memory system designs, *measured in clock cycles*?

2. **(10%)**
(a) Describe *at least one clear advantage* a Harvard architecture (separate instruction and data caches) has over a unified cache architecture (a single cache memory array accessed by a processor to retrieve both instructions and data).
(b) Describe *one clear advantage* a unified cache architecture has over the Harvard architecture.

3. **(10%)** (a) What is RAID? (b) Match the RAID levels 1, 3, and 5 to the following phrases for the *best match*. Use each only once.
__________ Data and parity striped across multiple disks
__________ Can withstand selective multiple disk failures
__________ Requires only one disk for redundancy

4. **(10%)**
(a) Explain the *differences* between a *write-through* policy and a *write back* policy.
(b) Tell which policy *cannot* be used in a *virtual memory* system, and describe the *reason*.

5. **(10%)**
(a) What is a *denormalized* number (*denorm* or *subnormal*)?
(b) Show how to use *gradual underflow* to represent a *denorm* in a Floating Point number system.

6. **(10%)** Try to show the following structure in the *memory map* of a 64-bit Big-Endian machine, by plotting the answer in a *two-raw* map where each raw contains 8 *bytes*.

```
Struct {
  int    a;    //0x11121314
  char*  c;   // "A", "B", "C", "D", "E", "F", "G"
  short  e;   //0x2122
}s;
```

7. **(15%)** Assume we have the following 3 ISA styles:
(1) *Stack*: All operations occur on top of stack where PUSH and POP are the only instructions that access memory;
(2) *Accumulator*: All operations occur between an ACCumulator and a memory location;
(3) *Load-Store*: All operations occur in registers, and register-to-register instructions use 3 registers per instruction.

接背面

(a) For each of the above ISAs, write an *assembly code* for the following program segment using LOAD, STORE, PUSH, POP, ADD, and SUB and other necessary assembly language mnemonics.

```
{  A=A+C;
   D=A-B;
}
```

(b) Some operations are not *commutative* (e.g., subtraction). Discuss what are the *advantages and disadvantages* of the above 3 ISAs when executing non-commutative operations.

8. **(25%)** The program below divides two integers through repeated addition and was originally written for a non-pipelined architecture. The divide function takes in as its parameter a pointer to the base of an array of three elements where X is the first element at `0($a0)`, Y is the second element at `4($a0)`, and the result Z is to be stored into the third element at `8($a0)`. Line numbers have been added to the left for use in answering the questions below.

```
1 DIVIDE: add $t3, $zero, $zero
2         add $t2, $zero, $zero
3         lw $t1, 4($a0)
4         lw $t0, 0($a0)
5 LOOP:   beq $t2, $t0, END
6         addi $t3, $t3, 1
7         add $t2, $t2, $t1
8         j LOOP
9 END:    sw $t3, 8($a0)
```

(a) Given a pipelined processor as discussed in the textbook, *where will data be forwarded* (ex. Line 10 EX.MEM? Line 11 EX.MEM)? Assume that forwarding is used whenever possible, but that branches have not been optimized in any way and are resolved in the EX stage.

(b) *How many data hazard stalls* are needed? Between which instructions should the stall bubble(s) be introduced (ex. Line 10 and Line 11)? Again, assume that forwarding is used whenever possible, but that branches have not been optimized in any way and are resolved in the EX stage.

(c) If X = 6 and Y = 3,
  (i) How many times is the *body of the loop executed*?
  (ii) How many times is the branch beq *not taken*?
  (iii) How many times is the branch beq *taken*?

(d) Rewrite the code assuming *delayed branches* are used. If it helps, you may assume that the answer to X/Y is at least 2. Assume that forwarding is used whenever possible and that branches are resolved in RF/ID. Do not worry about reducing the number of times through the loop, but arrange the code to use as few cycles as possible by avoiding stalls and wasted instructions.