國立中山大學九十三學年度博士班招生考試試題

科目：計算機系統【資工系】

1. (15%)Let $x = a_1a_2\cdots a_m$ and $y = b_1b_2\cdots b_n$ be two strings over some alphabet Σ. Suppose we want to transform x into y by using minimum number of insertions, deletions, and replacements. For example "convert" can be transformed into "counter" by inserting "u" after "o", replacing "v" by "t" and deleting "t". Let $c_{i,j}$ be the minimum number of insertions, deletions, and replacements to transform $a_1a_2\cdots a_i$ into $b_1b_2\cdots b_j$.
   (a) State what the trivial subproblems are, and give their c values.
   (b) Show that $c_{i,j} = \min\{c_{i-1,j} + 1, c_{i,j-1} + 1, c_{i-1,j-1} + \delta_{i,j}\}$, where $\delta_{i,j} = 0$ if $a_i = b_j$ and $\delta_{i,j} = 1$, otherwise.
   (c) Describe an efficient algorithm for determining the minimum number of insertions and deletions to transform x into y.
   (d) Find the time complexity of the algorithm.

2. (15%)Quick sort is a sorting algorithm for an array $A = [a_1, a_2, \ldots, a_n]$. It first swaps elements of the array and splits the array into two parts $A_l = [a'_1, a'_2, \ldots, a'_s]$ and $A_r = [a'_{s+1}, a'_{s+2}, \ldots, a'_n]$ so that $a'_i \le a'_j$ for every $a'_i \in A_l$ and every $a'_j \in A_r$. It then recursively sorts A_l and A_r. Suppose that you are going to design a nonrecursive version of quick sort, and choose to use a stack to store the ranges of the array yet to be sorted. The new algorithm sorts the part of the array whose range is stored on the top of the stack until the stack is empty. Initially, the stack contains only one range, $[1..n]$, which means that the entire array is to be sorted.
   (a) Describe the nonrecursive version of the algorithm.
   (b) Show that if you store the smaller part of the array on top of the stack after each split, then the size of the stack will be bounded by $\lceil \log n \rceil + 1$.

3. (20%)Let S be a set of n elements. Initially, each element in S is a set by itself. The *union-find* problem on the set S consists of a sequence of $\mathtt{union}(x, y)$ and $\mathtt{find}(z)$ operations. The $\mathtt{union}(x, y)$ operation makes the set containing x and the set containing y into one set. The $\mathtt{find}(z)$ operation reports the name of the set to which z belongs. The name of a set can be anything you like, but it must not be changed, unless the set is unioned into another set.
   (a) Design data structures and algorithms for the union-find problem so that both union and find operations can be done efficiently for large $|S|$.
   (b) Analize the time complexity of $\mathtt{union}(x, y)$ and $\mathtt{find}(z)$.
   (c) Analize the time complexity for a sequence of m union and find operations.
   (d) Describe an application of the union-find problem.

國立中山大學九十三學年度博士班招生考試試題

科目：計算機系統【資工系】

4. (20%) Cache performance can be improved by reducing the miss penalty, miss rate, or hit time.
4.1 (5%) Describe five techniques to reduce the miss penalty.
4.2 (5%) Describe five techniques to reduce the miss rate.
4.3 (5%) Describe three techniques to reduce the miss penalty or miss rate *via parallelism*.
4.4 (5%) Describe three techniques to reduce the hit time.

5. (10%) What are the differences between general-purpose CPU and digital signal processors (DSP)? Explain their differences in addressing modes, types of sizes of operands, operations, compiler support, ...etc.

6. (20%)The instruction set architectures can be classified into four categories:

- accumulator : all operations occur between a single register and a memory
- memory-memory : all instruction addresses reference only memory locations.
- Stack : all operands occur on top of the stack. Push and Pop are the only instructions that access memory; all other 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 access 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.

Invent your own assembly language mnemonics (the following figure provides a useful sample to generalize), and for each architecture, write the best equivalent assembly language code for this high-level language code sequence:

$$A = B + C;$$
$$B = A + C;$$
$$D = A - B;$$

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	Register (register-memory)	Register (load-store)
Push A	Load A	Load R1,A	Load R1,A
Push B	Add B	Add R3,R1,B	Load R2,B
Add	Store C	Store R3,C	Add R3,R1,R2
Pop C			Store R3,C

Fig: The code sequence for C = A + B for four classes of instruction sets.