TESTS AND SOLUTIONS

Computer Science Department
Stanford University
Comprehensive Examination in Analysis of Algorithms

Autumn 1991

October 7, 1991

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in BLUE BOOKs. There are 4 problems in the exam; use a SEPARATE blue book for each problem. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.
2. The number of POINTS for each problem indicates how elaborate an answer is expected. The total number of points is 60, and the exam takes 60 minutes. This "coincidence" can help you plan your time.

3. This exam is CLOSED BOOK. You may not use any notes, calculators, computers, or outside help.

4. Show your work, since PARTIAL CREDIT will be given for incomplete answers.

"COMPS"
1991

Comprehensive Exam: Analysis of Algorithms (60 points) Autumn 1991
Please answer 3 out of 4 questions. If you attempt to answer all 4 questions, your grade will be the sum of the scores on the 3 best answers.

1. (20 points total) *Counting*
A group of n people comes to a party, each person carrying a hat and an umbrella. At the end of the party each person leaves with a hat and an umbrella, neither of which is his. Notice that in order to find out the number of possibilities, we can check all possible assignments of hats and umbrellas and count the number of appropriate assignments. The problem with this approach is that it takes exponential time. Can you come up with an approach that will allow us to compute the number of possibilities in polynomial time ? For example, an expression like $\sum_{i=1}^{n} i^3$ is an acceptable answer. In other words, your answer should be a formula computable in polynomial time.

2. (20 points total)
Given a sequence of numbers $a_1, a_2, \ldots, a_n$, a *subsequence* is a sequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$, where $i_j < i_{j+1}$ for all $1 \leq j \leq k-1$. You are given weights $w(a_i) \geq 0$ associated with each element of the given sequence. Describe an efficient algorithm to find an increasing subsequence of maximum weight, where the weight of the subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ is defined as $\sum_{j=1}^{k} w(a_{i_j})$. What is the running time of your algorithm? Explain.

3. (20 points total)
An ordered 2-3 tree T is used to implement a dictionary, with each element in the dictionary being assigned to a unique leaf in T. Initially, T is empty. Then the sequence of operations INSERT(a_1), INSERT(a_2), ..., INSERT(a_n) is performed, where each of the $n!$ possible orderings of the elements $a_1, a_2, \ldots, a_n$ is equally likely. Let h be the height of T following these n insertions (recall that a tree consisting of a single vertex has height 0).

   (a) (15 points)
   Give an exact formula for the maximum possible value of h (as a function of n).

   (b) (5 points)
   If $n = 30$, what is the expected value of h? [*Hint:* What are the minimum and maximum possible values of h?]

4. (20 points total) *Recurrence Relations*
Given any constant c, where $0 < c < 1$, and any positive real number N, the function $T(N, c)$ is defined as follows:

$$T(N,c) = \begin{cases} 2T(cN,c) + N^2 & \text{if } N > 1 \\ N^2 & \text{if } 0 < N \leq 1 \end{cases}$$

   (a) (10 points)
   What is the asymptotic growth rate of $T(N, 1/2)$ (to within a constant factor)?

   (b) (10 points)
   What is the largest value of c such that $T(N,c) = O(N^2 \log N)$?

2

Comprehensive Exam: Analysis of Algorithms
Autumn 1991

Problem 1: (20 points total) *Counting*

A group of n people comes to a party, each person carrying a hat and an umbrella. At the end of the party each person leaves with a hat and an umbrella, neither of which is his. Notice that in order to find out the number of possibilities, we can check all possible assignments of hats and umbrellas and count the number of appropriate assignments. The problem with this approach is that it takes exponential time. Can you come up with an approach that will allow us to compute the number of possibilities in polynomial time ? For example, an expression like $\sum_{i=1}^{n} i^3$ is an acceptable answer. In other words, your answer should be a formula computable in polynomial time.

Answer: First, observe that there is no connection between umbrellas and hats, and therefore it is sufficient to count the number of possibilities to assign, say, hats, and square the result. Notice that although the first person to leave has exactly $n-1$ choices, the second person has either $n-1$ or $n-2$ choices, depending on whether the first one has grabbed the hat that belongs to the second one or not. Therefore, this approach will lead to an exponential algorithm.

There are several valid approaches, and we will present two of them. First, notice that we are looking for the number of permutations of n elements that do not have fixed points. Let $H(n)$ denote this number. Total number of permutations on n elements is $n!$. Out of those, we have $H(n)$ permutations with no fixed points, $\binom{n}{1}H(n-1)$ permutations with a single fixed point, $\binom{n}{2}H(n-2)$ with exactly 2 fixed points, etc. Hence, we have:

$$n! = \sum_{i=0}^{n} \binom{n}{i} H(i)$$

Using this formula, one can compute $H(1)$, use it to compute $H(2)$, etc.

A different, somewhat cleaner way, is to use inclusion-exclusion. Let S_i denote the set of permutations that fix i. We would like to compute

$\cup_i |S_i|$. In order to use inclusion-exclusion, we have to be able to compute the cardinality of intersection of r different sets S_i. Given r places that we want to fix, the cardinality of such intersection is $(n-r)!$, since all the rest of the elements can be assigned arbitrary. There are $\binom{n}{r}$ possible intersections associated with each r, and hence this leads to the following formula:

$$H(n) = n! - \sum_{i=1}^{n} (-1)^{i+1} \binom{n}{r} (n-r)!$$

Problem 2: (20 points total)

Given a sequence of numbers $a_1, a_2, \ldots, a_n$, a *subsequence* is a sequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$, where $i_j < i_{j+1}$ for all $1 \leq j \leq k-1$. You are given weights $w(a_i) \geq 0$ associated with each element of the given sequence. Describe an efficient algorithm to find an increasing subsequence of maximum weight, where the weight of the subsequence $a_{i_1}, a_{i_2}, \ldots, a_{i_k}$ is defined as $\sum_{j=1}^{k} w(a_{i_j})$. What is the running time of your algorithm? Explain.

Answer: There was a confusion as to what does an "increasing subsequence" mean. It means that the numbers and not weights are increasing. If one tries to solve the problem where we require increasing weights in the output subsequence, then it is easy to see that this problem is a special case of the original problem (just set each number in the sequence to be equal to its weight). Therefore, we will present the solution to the original problem.

The idea is to use dynamic programming. Instead of presenting an algorithm to compute the optimum subsequence, we will concentrate on an algorithm to compute the weight of the subsequence. It is straightforward to change this algorithm to compute the subsequence itself. The following algorithm can be imroved, but we will ommit the optimization issues and concentrate on the main ideas. Notice that, without loss of generality, we can assume that the input sequence includes only numbers from 1 to n.

Let $S_i = a_1, a_2, \ldots, a_i$ be the i-th prefix of the given sequence. Assume that we have computed the maximum-weight increasing subsequence

for S_i. Notice that this is not sufficient in order to quickly compute the maximum-weight increasing subsequence for S_{i+1} ! The reason is that it might be beneficial to use a non-optimum subsequence with respect to S_i in order to be able to use a_i as well.

For each prefix S_i, we will keep $A(i,j)$, $1 \leq j \leq n$, where $A(i,j)$ is the weight of the maximum-weight increasing subsequence out of prefix S_i, where the restriction is that the subsequence does not include elements that are larger than j.

Initialization of $A(1,j)$ is trivial. Now, assume that we have computed $A(i,j)$ for some i and for all j. Consider $k = a_{i+1}$. For all $j < k$, existence of this element does not help us, and we just copy $A(i+1,j) = A(i,j)$. Consider $A(i+1,k)$. We can add a_{i+1} to the corresponding sequence, and hence $A(i+1,k) = A(i,k) + w(a_{i+1})$. For all $j > k$, we set $A(i+1,j) = max\{A(i,j), A(i+1,k)\}$.

The running time of this algorithm is $O(n^2)$ and it uses $O(n)$ space, since we need to keep only a single (current) column of A in memory.

Problem 3: (20 points total)

An ordered 2-3 tree T is used to implement a dictionary, with each element in the dictionary being assigned to a unique leaf in T. Initially, T is empty. Then the sequence of operations INSERT(a_1), INSERT(a_2), ..., INSERT(a_n) is performed, where each of the $n!$ possible orderings of the elements $a_1, a_2, \ldots, a_n$ is equally likely. Let h be the height of T following these n insertions (recall that a tree consisting of a single vertex has height 0).

1. (15 points)
   Give an exact formula for the maximum possible value of h (as a function of n).
2. (5 points)
   If $n = 30$, what is the expected value of h? [*Hint:* What are the minimum and maximum possible values of h?]

Answer: A 2-3 tree is a tree in which each internal node has either 2 or 3 children, and every path from the root to a leaf is of the same length.

When an ordered 2-3 tree is used to implement a dictionary, the elements in the dictionary are stored in the leaves of the tree in ascending order from left to right.

1. $\lfloor \log_2 n \rfloor$.
   If the elements are inserted in increasing order, then following 2^k insertions T is a complete binary tree with height k. Therefore, $h \geq \lfloor \log_2 n \rfloor$. Furthermore, no 2-3 tree with n leaves has height greater than $\lfloor \log_2 n \rfloor$ because all leaves are at the same level and each internal node has degree at least 2, so $h \leq \lfloor \log_2 n \rfloor$.
2. 4.
   Any 2-3 tree of height h has at least 2^h leaves and at most 3^h leaves. If $h \leq 3$ then $3^h < 30$, while if $h \geq 5$ then $2^h > 30$. Therefore, any 2-3 tree with 30 leaves has height 4.

Problem 4: (20 points total) *Recurrence Relations*

Given any constant c, where $0 < c < 1$, and any positive real number N, the function $T(N, c)$ is defined as follows:

$$T(N,c) = \begin{cases} 2T(cN,c) + N^2 & \text{if } N > 1 \\ N^2 & \text{if } 0 < N \leq 1 \end{cases}$$

1. (10 points)
   What is the asymptotic complexity of $T(N, 1/2)$ (to within a constant factor)?
2. (10 points)
   What is the largest value of c such that $T(N,c) = O(N^2 \log N)$?

Answer: 1. $O(N^2)$.

The values of the recursive calls form a geometrically decreasing sequence. For example, if $N > 4$ then

$$\begin{aligned} T(N,1/2) &= 2T(N/2,1/2) + N^2 \\ &= 4T(N/4,1/2) + N^2/2 + N^2 \\ &= 8T(N/8,1/2) + N^2/4 + N^2/2 + N^2. \end{aligned}$$

Computer Science Department
Stanford University
Comprehensive Examination in Automata, Languages, and Mathematical Theory of Computation
Autumn 1991

October 7, 1991

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in BLUE BOOKS. There are four problems in the exam; use a SEPARATE blue book for each problem. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. Be sure you have all the pages of this exam. There are 3 pages.

3. The number of POINTS for each problem indicates how elaborate an answer is expected. The total number of points is 60, and the exam takes 60 minutes. This "coincidence" can help you plan your time.

4. This exam is OPEN BOOK.

5. Show your work, since PARTIAL CREDIT will be given for incomplete answers.

Comprehensive:

Automata, Languages, and Mathematical Theory of Computation (60 points) **Autumn 1991**

Problem 1 (24 points). Consider a programming language built up using assignment, composition, while, if and the local variable statement

begin new x; S end

Assume the obvious Hoare axioms

Axiom 1: Assignment Axiom

$$\{p[t/x]\}\ x := t\ \{p\}.$$

Rule 2: Composition Rule

$$\frac{\{p\}\ S_1\ \{r\}\ ,\ \{r\}\ S_2\ \{q\}}{\{p\}\ S_1\ ;\ S_2\ \{q\}}.$$

Axiom 3: *if-then-else* Rule

$$\frac{\{p \wedge e\}\ S_1\ \{q\}\ ,\ \{p \wedge \neg e\}\ S_2\ \{q\}}{\{p\}\ \textit{if } e \textit{ then } S_1 \textit{ else } S_2 \textit{ fi}\ \{q\}}.$$

Rule 4: *while* Rule

$$\frac{\{p \wedge e\}\ S\ \{p\}}{\{p\}\ \textit{while } e \textit{ do } S \textit{ od}\ \{p \wedge \neg e\}}.$$

Rule 5: Consequence Rule

$$\frac{p_0 \rightarrow p_1 \quad \{p_1\}\ S\ \{q_1\} \quad q_1 \rightarrow q_0}{\{p_0\}\ S\ \{q_0\}}$$

and

Rule 16: Variable Declaration Rule

$$\frac{\{p \wedge y = \omega\}\ S[y/x]\ \{q\}}{\{p\}\ \textit{begin new } x; S \textit{ end}\ \{q\}} \qquad \text{where } y \notin \textit{free}(p, S, q).$$

together with a background theory T (cf. Rule 5). Two programs P1 and P2 are said to provably satisfy the same triples iff for all p, q:

$$\vdash \{p\}P1\{q\} \text{ iff } \vdash \{p\}P2\{q\}.$$

We abbreviate this to $P1 \sim P2$. Justify your answers briefly in the following questions.

1a. (4 points). Clearly $\sim$ is an equivalence relation. Is it necessarily a congruence (i.e., do the programming primitives (; *while* ...) preserve this relation)?

1b. (4 points). If $P1 \sim P2$ then are they necessarily equivalent semantically (i.e., as maps from stores to stores) for all models of T?

Are the following programs $\sim$ equivalent?

1c. (4 points). x not free in P.

$P1$ = begin new x; P; end
$P2 = P$

1d. (4 points). x not free in P

$P1$ = begin new x;
$x := 0$;
P;
if $x = 0$ then diverge fi;
end
$P2$ = diverge

1e. (4 points).

$P1$ = begin new x; $P[x/y]$; $y := x$; end

$P2 = P$

where P is a program.

1f. (4 points).

$P1$ = begin new x; begin new y; $x := 0$; $y := 0$; $P[x/z_0, y/z_1]$; end

$P2$ = begin new x; begin new y; $x := 0$; $y := 0$; $P[y/z_0, x/z_1]$; end

Problem 2 (16 points). Below is text that defines the concept of ordered tree and the insert operation within the Boyer-Moore logic. The shell principle is used to introduce an abstract data type, OT. Elements of OT are either empty (ET) or constructed from a numeric label together with left and right subtrees. A predicate ORDERED.TREE is introduced to specify the elements of the OT shell that are ordered trees. Finally the INSERT function is defined and a theorem, ORDERED.TREE.INSERT is conjectured. The theorem states that INSERT applied to a number and an ordered tree returns an ordered tree.

2a. (4 points). Give at least 5 axioms assumed by the theorem prover as a consequence of the Shell Definition.

2b. (6 points). State a justification that the theorem prover might use to accept the definition of the ORDERED.TREE predicate.

2c. (6 points). Give an instance of the induction principle that could be used by the theorem prover to prove ORDERED.TREE.INSERT.

Shell Definition.

Add the shell OT of three arguments with bottom object ET, recognizer OTP, accessors LT, LABEL, and RT, type restrictions (ONE.OF OTP), (ONE.OF NUMBERP), and (ONE.OF OTP), and default values ET, ZERO, ET.

Definition.

```
(ORDERED.TREE X)
 =
(IF (NOT (OTP X))
    F
    (IF (EQUAL (ET) X)
        T
        (AND (ORDERED.TREE (LT X))
             (ORDERED.TREE (RT X))
             (OR (EQUAL (LT X) (ET)) (LESSP (LABEL (LT X)) (LABEL X)))
             (OR (EQUAL (RT X) (ET)) (LESSP (LABEL X)) (LABEL (RT X)))))))
```

Definition.

```
(INSERT L X)
 =
(IF (NOT (OTP X))
    (ET)
    (IF (EQUAL (ET) X)
        (OT (ET) L (ET))
        (IF (LESSP (LABEL X) L)
            (OT (LT X ) (LABEL X) (INSERT L (RT X)))
            (IF (LESSP L (LABEL X))
                (OT (INSERT L (LT X)) (LABEL X) (RT X ))
                X))))
```

```
Theorem ORDERED.TREE.INSERT.
    (IMPLIES (AND (NUMBERP L) (ORDERED.TREE X))
             (ORDERED.TREE (INSERT L X)))
```

Problem 3 (10 points). A minimized deterministic finite state machine accepts all strings that contain a certain fixed string s of length L. What are the largest and smallest number of states the machine might have? Justify your answer.

Problem 4 (10 points).

Consider the following program for finding a real-number approximation to the square root of a non-negataive real number r.

$z \leftarrow 0$;
$v \leftarrow max(r, 1)$;
while $\epsilon \leq v$ *do* $v \leftarrow v/2$;
    if $(z + v)^2 \leq r$ *then* $z \leftarrow z + v$;
return(z)

4a. Write a specification that expresses what this program is doing.

4b. Provide an inductive assertion, well-founded set, and partial function adequate for proving the total correctness of this program. It is not necessary to give the proof.

Comprehensive Solutions: **Autumn 1991**

Automata, Languages, and Mathematical Theory of Computation (60 points)

Problem 1 (24 points). [2 pts for correct yes/no answer; 2 pts for correct reason].

1a. Yes. Easily established by induction on proofs (i.e., last rule used).

1b. No. $P1$ is $x := 0$; $P2$ is *while* $x > 0$ *do* $x := x - 1$.

1c. Yes. Use Rule 16.

1d. Yes. Use Rule 16

1e. No.

$y := y + 1$
$\vdash \{y = 0\}\ P2\ \{y = 1\}$
$\neg (\vdash \{y = 0\}\ P1\ \{y = 1\})$

1f. Yes. (Modulo missing "end" — a typo). Again use Rule 16.

Problem 2 (16 points).

2a. Axioms added include the following.

```
(OR (EQUAL (OTP X) T) (EQUAL (OTP X) F))
(OTP (OT X1 X2 X3))
(OTP (ET))
NOT (EQUAL (OT X1 X2 X3) (ET)))
(IMPLIES (AND (OTP X) (NOT (EQUAL X (ET))))
         (EQUAL (OT (LT X) (LABEL X) (RT X)) X))
(IMPLIES (OTP X1) (EQUAL (LT (OT X1 X2 X3)) X1))
(IMPLIES (NUMBERP X2) (EQUAL (LABEL (OT X1 X2 X3)) X2))
(IMPLIES (OTP X3) (EQUAL (RT (OT X1 X2 X3)) X3))
(IMPLIES (OR (NOT (OTP X))
             (EQUAL X (ET))
             (AND (NOT (OTP X1)) (EQUAL X (OT X1 X2 X3))))
         (EQUAL (LT X) (ET)))
(IMPLIES (OR (NOT (OTP X))
             (EQUAL X (ET))
             (AND (NOT (NUMBERP X2)) (EQUAL X (OT X1 X2 X3))))
         EQUAL (LABEL X) (ZERO)))
(IMPLIES (OR (NOT (OTP X))
             (EQUAL X (ET))
             (AND (NOT (OTP X3)) (EQUAL X (OT X1 X2 X3))))
         (EQUAL (RT X) (ET)))
(NOT (OTP T))
(NOT (OTP F))
(IMPLIES (OTP X) (NOT (r' X)))  ;; r' previously introduce recognizer
```

2b. There must be a well-founded relation r and a function m such that

```
(IMPLIES (AND (OTP X) (NOT (EQUAL X (ET))))  (r (m (LT X)) (m X)))
(IMPLIES (AND (OTP X) (NOT (EQUAL X (ET))))  (r (m (RT X)) (m X)))
```

are proveable. Take m = COUNT, r = LESSP.

2c. Let (p L X) =

```
(IMPLIES (AND (NUMBERP L) (ORDERED.TREE X))
              (ORDERED.TREE (INSERT L X)))
```

To instantiate the induction principal it is sufficient to find terms q, sX1, sL1, sX2, sL2 with at most L and X free, a well-founded relation r, and a measure m meeting the conditions of the induction principle that

```
(IMPLIES q (r (m sL1 sX1) (m L X)))
(IMPLIES q (r (m sL2 sX2) (m L X)))
(IMPLIES (NOT q) (p L X))
(IMPLIES (AND q (p sL1 sX1)  (p sL2 sX2)) (p L X))
```

are provable. This is satisfied by taking r and m as in B (m now ignoring its first argument) and

```
q =  (AND (OTP X) (NOT (EQUAL X (ET))))
sL1 = sL2 = L
sX1 = (LT X)
sX2 = (RT X)
```

Problem 3 (10 points). Distinct prefixes of s are inequivalent, so they must go to distinct states, requiring at least $L+1$ states. If the state encodes the longest prefix of s that is a suffix of the already-consumed input, $L+1$ suffices.

Problem 4 (10 points).

2a. [4 pts]

$$\{r \geq 0 \wedge \epsilon > 0\}\ P\ \{\sqrt{r} - \epsilon < z \leq \sqrt{r}\}.$$

2b. [6 pts] The inductive assertion [3 pts] is given by

$$(z \leq r) \wedge [(\sqrt{r} < z+v) \vee (r = z+v = 1)]$$

and the well-founded set and partial function [3 pts] are $(\mathcal{N}, <)$ and $\lfloor \frac{u}{\epsilon} \rfloor$, respectively.

COMPUTER ARCHITECTURE

Comprehensive Exam: Architecture (60 points) **Autumn 1991**

1. (15 points total) *Instruction Set Design.*

Your company currently produces a load/store processor. The table below lists the instruction distributions (along with the instruction latencies) for your most important application–SPICE.

	latency (cycles)	frequency
intALU	1	45%
branch	2	5%
load	2	27%
store	1	8%
FPadd	4	7%
FPmult	6	8%

(a) (3 points) What is the CPI of execution (CPIe) for this machine?

ANSWER: CPIe = (0.45+0.08) + (0.05+0.27)2 + (0.07)4 + (0.08)6
= 1.93 cycles/instr

(b) (2 points) Now the wonderful group in the technology lab have provided the next generation of the chip with more transistors, and the VLSI people say that they can use these transistors to reduce the latency of the FPadd to 2 cycles and the FPmult to 4 cycles. What is the new CPIe?

ANSWER: CPIe = (0.45+0.08) + (0.05+0.27)2 + (0.07)2 + (0.08)4
= 1.63 cycles/instr

(c) (2 points) What is the resulting speedup, assuming no change in the cycle time?

ANSWER: speedup = 1.93 / 1.63 = 1.18x faster

(d) (5 points) Some other daring VLSI designers in the company claim that they can use the same extra transistors to implement some new, complex instructions. They claim that the FP, intALU, and memory port are essentially independent functional units, and thus, can operate in parallel. They suggest two new types of instructions: (1) an instruction that performs a FPop and a R-R intALU operation; and (2) an instruction that performs a register-indirect load/store and a R-R intALU operation. The bottom line is that 20% of the intALU operations can be collapsed with a load/store operation and that 12% of the intALU operations can be collapsed with a FP operation. What is the resulting CPIe of this technique? State any assumptions that you make.

ANSWER: CPIe = (0.45-0.20-0.12+0.08)/0.68 + ((0.05+0.27)/0.68)2
+ (0.07/0.68)4 + (0.08/0.68)6 = 2.37 cycles/instr

8

(e) (3 points) Again, assuming no change in the cycle time, which use of the extra transistors gives better performance (please show a speedup comparison)?

ANSWER: speedup = 1.93(1) / 2.37(0.68) = 1.20x faster
therefore (d) is better

2. (20 points total) *Memory System Design*

You've been given a promotion to memory system designer, and your company has the following questions for you.

(a) (4 points) The current OS uses a 4KB page size. Your boss is thinking about doing the virtual address translation in parallel with the instruction cache (Icache) access so that she can build a physical cache (i.e. no flushes or PIDs necessary). The VLSI designers tell you that you can have an Icache with an associativity of at most 5 ways. What is the biggest-sized, physical Icache that you can build? How many bits of virtual address (VA) are used to index into this Icache if the block size is 8 words (1 word = 32-bits)?

```
ANSWER: MaxSize = 5 * 4KB = 20KB
        index = 12 - 2 - 3 = 7 bits
```

(b) (2 points) Your boss's boss is really interested in building a 64-bit processor, i.e. a processor with a flat 64-bit virtual address space. If you designed the Icache as a virtually-tagged cache, how many bits of the 64-bit virtual address are kept in the tags if the data portion of the Icache is 32KB in size and only 2-way set associative?

```
ANSWER: tag = 64 - 15 + 1 = 50 bits
```

(c) (8 points) If we kept a 9-bit PID and a valid bit in the tag along with the rest of the virtual address, what percentage of the total Icache space is consumed by the tag array for a 32-bit VA and for a 64-bit VA? The data portion of the Icache is still 32KB in size and 2-way set associative. Please count only bits, not transistors, and ignore control logic.

```
ANSWER: set size = 32KB / (2 * 32B) = 512 entries => 1024 tags

32-bit VA: tag bits = 1024 * (18+9+1) = 28672 bits
           data bits = 32K * 8 = 262144 bits
               28672 / (28672+262144) = 9.9% overhead

64-bit VA: tag bits = 1024 * (50+9+1) = 61440 bits
           data bits = 32K * 8 = 262144 bits
                 61440 / (61440+262144) = 19.0% overhead
```

(d) (4 points) To fully support this 64-bit addressing, someone suggests adding 64-bit instructions to the instruction set architecture so that large immediates can be generated quickly. A problem with 64-bit instructions is that they can fall across Icache line boundaries because instructions only have to be word-aligned. If the Icache miss rate is 2%, the miss penalty is 12 cycles, and probability that a 64-bit instruction falls across an Icache boundary is 6%, what is the average memory access time (AMAT) for fetching an instruction? Assume that a 32-bit or a 64-bit

instruction can be fetched in 1 cycle if you hit in the Icache and if you do not cross a line boundary.

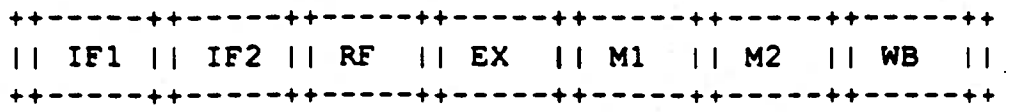
ANSWER: AMAT = 1 + (0.02)(12) + (0.06)(1) = 1.30 cycles

(e) (2 points) This splitting increases the AMAT of the Icache, but it also affects the virtual memory system. Assume your processor uses pages and a TLB, what is the ugly effect of these split instructions?

ANSWER: 2 TLB accesses are necessary for split fetches that miss meaning that a page fault can occur in the middle of an instruction miss. What do you do with the first half of the instruction fetch in the meantime?

3. (25 points total) *Pipelining*.

The pipeline shown below has been designed to work at a very high clock rate. The pipeline ticks twice as fast as the memory latency, although the memory is pipelined so that it returns an item on every clock tick. The consequence is that both the instruction fetch and the memory load/store operations have had to be split into two stages.




```
IF1   - first cycle of instruction fetch
IF2   - second cycle of instruction fetch
RF    - instruction decode and register fetch
EX    - ALUop OR memory address calcuation
M1    - first cycle of data memory load or store
M2    - second cycle of data memory load or store
WB    - write back result into register file
```

(a) (3 point) For such a high performance machine, is it a good idea to have the PC as one of the general purpose registers? If not, why not?

ANSWER: It is not a good idea because it interferes with the ability to efficiently pipeline the machine.

(b) (10 points) We are considering the tradeoffs between the use of delayed branches and squashing branches. For the squashing branch case, assume that the machine executes along the not-taken path. If the branch is actually taken, then the write-back and memory operations corresponding to the incorrectly fetched instructions are suppressed. For the non-squashing case, the following table gives the probabilities of filling the branch delay slots from code above the branch. If the probability that a branch is taken is 60%, which method do you recommend? Justify your decision quantitatively.

slot	probability of fill
1	75%
2	25%
3	5%
4	2%
5	1%

ANSWER: (NOTE this answer is for 3 branch delay slots, but 2 slots is reasonable)

```
CPI-branch-sq     = fraction_taken * 4 + fraction_not_taken * 1
                  = .6 * 4 + .4 = 2.8

CPI-branch-non-sq = 4 - (prob_slot1_full + prob_slot2_full
                    + prob_slot3_full)
                  = 4 - ( .75  + .25 + .05) = 2.95
```

Therefore, it is better to have squashing branches.

(c) (6 points) Assuming that the processor has no hardware interlocks, rewrite the code sequence below inserting the necessary NOPs to avoid hazards. Assume that branches are not squashing.

```
OR   R5,R6 -> R7
LD   1(R1) -> R4
ADD  R1,R5 -> R5
ADD  R3,R4 -> R2
ST   R5 -> 2(R1)
ADD  R7,R5 -> R4
BEQ  R2,R5,target
AND  R1,R6 -> R1
...

ANSWER:
OR   R5,R6 -> R7
LD   1(R1) -> R4
ADD  R1,R5 -> R5
NOP
ADD  R3,R4 -> R2
ST   R5 -> 2(R1)
ADD  R7,R5 -> R4
BEQ  R2,R5,target
NOP
NOP
NOP
AND  R1,R6 -> R1
...

or, without any by-passing:
OR   R5,R6 -> R7
LD   1(R1) -> R4
ADD  R1,R5 -> R5
NOP
NOP
NOP
ADD  R3,R4 -> R2
ST   R5 -> 2(R1)
ADD  R7,R5 -> R4
NOP
NOP
BEQ  R2,R5,target
NOP
```

```
NOP
NOP
AND R1,R6 -> R1
...
```

(d) (6 points) Reorganize the above code to use the fewest number of NOPs.

```
ANSWER:
LD   1(R1) -> R4
OR   R5,R6 -> R7
ADD  R1,R5 -> R5
ADD  R3,R4 -> R2
BEQ  R2,R5,target
ST   R5 -> 2(R1)
ADD  R7,R5 -> R4
NOP
AND  R1,R6 -> R1
...
```

Computer Science Department
Stanford University
Comprehensive Examination in Numerical Analysis
Autumn 1991

October 9, 1991

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. The number of POINTS for each problem indicates how elaborate an answer is expected. For example, an essay-type question worth 6 points or less doesn't deserve an extremely detailed answer, even though a person can expound at length on just about any topic in computer science.

3. The total number of points is 30, and the exam takes 30 minutes. This "coincidence" can help you plan your time.

4. This exam is OPEN BOOK. You may use notes, articles, or books—but no help from other sentient agents such as other humans or robots.

5. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

Problem I:

(16 points). **Linear Systems and Matrices.**

Consider the matrix

$$A = \begin{bmatrix} 4 & -2 & 1 \\ -2 & 1 & 0 \\ 3 & 3 & 1 \end{bmatrix}.$$

(a) (3 points). Show that the LU factorization of A does not exist. Give necessary and sufficient conditions for the LU factorization to exist.

(b) (4 points). Show how a permutation can be introduced to obtain a matrix B for which the LU factorization exists. Compute the LU factorization of this matrix B.

(c) (4 points). Show how the LU factorization can be used to compute the third column of A^{-1} without computing A^{-1} completely.

(d) (5 points). Suppose that the function $f(x)$ was measured in 4 points with the following results:

x	-2	-1	0	1
$f(x)$	-1	1	1	4

We want to approximate $f(x)$ by $p_2(x) = a + bx + cx^2$ with the method of least squares. Establish the corresponding linear system of equations and show that it does not possess a solution.

Problem II:

(14 points). **Interpolation.**

(a) (5 points). Construct the polynomial

$$p_2(x) = a_2x^2 + a_1x + a_0$$

of degree 2 which interpolates the function

$$f(x) = \frac{1}{1+x}$$

at the points $x_0 = 0, x_1 = \frac{1}{2}, x_2 = 1$.

(b) (5 points). Give an upper bound for the interpolation error $|f(x) - p_2(x)|$ on $[0, 1]$.

(c) (4 points). Show that the polynomial $p_n(x)$ of degree $\leq n$ interpolating a function $f(x)$ in $n+1$ points $x_0 < x_1 < \cdots < x_n$ is unique.

Autumn 1991

Solution to the Numerical Analysis Examination

I.

a) The LU factorization does not exist since the 2×2 submatrix

$$A_2 = \begin{bmatrix} 4 & -2 \\ -2 & 1 \end{bmatrix}$$

is singular. A necessary and sufficient condition is that all the submatrices $A_k = (a_{ij})_{1 \le i,j \le k}$, $k = 1, \ldots, n$ are nonsingular.

b) A permutation of the second and third row gives

$$B = \begin{bmatrix} 4 & -2 & 1 \\ 3 & 3 & 1 \\ -2 & 1 & 0 \end{bmatrix} \quad \text{with the } LU \text{ factorization}$$

$$B = \begin{bmatrix} 1 & & \\ 3/4 & 1 & \\ -1/2 & 0 & 1 \end{bmatrix} \begin{bmatrix} 4 & -2 & 1 \\ 0 & 9/2 & 1/4 \\ 0 & 0 & 1/2 \end{bmatrix}$$

c) The third column of A^{-1} can be computed by solving the linear system $Ax = e_3$ which is equivalent to $Bx = e_3$. This is done by solving the triangular system $Ly = e_3$, and then $Ux = y$.

d)

$$\begin{aligned} a - 2b + 4c &= -1 \\ a - b + c &= 1 \\ a &= 1 \\ a + b + c &= 4 \end{aligned}$$

The corresponding linear system is

$$\begin{bmatrix} 1 & -2 & 4 \\ 1 & -1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} -1 \\ 1 \\ 1 \\ 4 \end{bmatrix}$$

Since $a = 1$, we have

$$\begin{aligned} -2b + 4c &= -2 \\ -b + c &= 0 \\ b + c &= 3 \end{aligned}$$

The second and third equation lead to $b = c = 3/2$ but then the first equation is not fulfilled ($-2 \cdot 3/2 + 4 \cdot 3/2 \neq -2$!).

II.

a)

$$\begin{aligned} a_0 &= 1 \\ a_0 + \frac{1}{2}a_1 + \frac{1}{4}a_2 &= \frac{2}{3} \\ a_0 + a_1 + a_2 &= \frac{1}{2} \end{aligned}$$

$a_0 = 1$:

$$\left.\begin{aligned} \frac{1}{2}a_1 + \frac{1}{4}a_2 &= -\frac{1}{3} \\ a_1 + a_2 &= -\frac{1}{2} \end{aligned}\right\} \Rightarrow \begin{aligned} a_1 &= -\frac{5}{6} \\ a_2 &= \frac{1}{3} \end{aligned}$$

The interpolating polynomial is $p_2(x) = \frac{1}{3}x^2 - \frac{5}{6}x + 1$.

b)

$$|f(x) - p_2(x)| \le \frac{1}{6} \cdot \max_{x\in[0,1]} |f^{(3)}(x)| \cdot \max_{x\in[0,1]} |w_2(x)|$$

$$f^{(3)}(x) = -\frac{1}{(1+x)^4} \quad \Rightarrow \quad \max_{x\in[0,1]} |f^{(3)}(x)| \le 1$$

$$w_2(x) = x(x - \frac{1}{2})(x - 1), \quad w_2'(x) = 3x^2 - 3x + \frac{1}{2}$$

$$\max_{x\in[0,1]} |w_2(x)| = |w_2(\frac{1}{2} \pm \sqrt{\frac{1}{12}})| = \frac{1}{6}\sqrt{\frac{1}{12}}$$

c) Suppose p_n and $\tilde{p}_n$ are two polynomials interpolating $f(x)$ at the points $x_0 < x_1 < \cdots < x_n$. Then, $\hat{p}_n = p_n - \tilde{p}_n$ is a polynomial of degree n with $n+1$ zeros. Therefore, $\hat{p}_n \equiv 0$ and $p_n \equiv \tilde{p}_n$.

Computer Science Department
Stanford University
Comprehensive Examination in Software Systems
Autumn 1991

October 11, 1991

READ THIS FIRST!

1. You should write your answers for this part of the Comprehensive Examination in a BLUE BOOK. Be sure to write your MAGIC NUMBER on the cover of every blue book that you use.

2. Be sure you have all the pages of this exam. There are 2 pages.

3. The number of POINTS for each problem indicates how elaborate an answer is expected. For example, an essay-type question worth 6 points or less doesn't deserve an extremely detailed answer, even though a person can expound at length on just about any topic in computer science.

4. The total number of points is 60, and the exam takes 60 minutes. This "coincidence" can help you plan your time.

5. This exam is OPEN BOOK. You may use notes, articles, or books—but no help from other sentient agents such as other humans or robots.

6. Show your work, since PARTIAL CREDIT will be given for incomplete answers. For example, you can get credit for making a reasonable start on a problem even if the idea doesn't work out; you can also get credit for realizing that certain approaches are incorrect. On a true/false question, you might get partial credit for explaining why you think something is true when it is actually false. But no partial credit can be given if you write nothing.

Comprehensive Exam: Software Systems (60 points) **Autumn 1991**

Note: If you are convinced you need to make an assumption to answer the question, state your assumption(s) as well as the answer.

1. (20 points total) *Concurrency*

   (a) (7 points) Bulgarian Computer & Communication Inc. (BCCI) has just produced a multiprocessor machine which lacks any type of indivisible read-modify-write instructions. John Meredith, a well-known computer consultant, has claimed that the whole thing has to be scrapped (at the cost of much wasted time and money) because of the above omission. You are called as a second opinion. Describe what you would recommend and why, both for this machine and what they do in the future, including any issues with scale and applications.

   (b) (7 points) Garn Smidley, a young, somewhat overconfident programmer, claims that he needs no synchronization in the sense of locks, semaphores, etc. in the concurrent program he is writing for you because there is only a single writer/update process for each shared data structure — the other processes just read/query the shared data structures. Give an example of how Smidley can still get himself in trouble if not careful, and an example of a non-trivial type of trick he might be using to make concurrent operations correct.

   (c) (6 points) You have the pleasure of meeting Paula Abdul at a party, and discover to your surprise that she has been studying concurrent programming between hit records! There is one thing she is confused on (which she proceeds to ask you). "How can signaling on a condition variable inside a monitor avoid resuming the waiting process inside the monitor, so there are multiple processes in the monitor (the signaler and the signaled), while still assuring the signaled process that the condition is true when it enters the monitor". she asks, explaining to the roadies that you are in the Ph.D. program at Stanford. As all look on in awe, what do you say?

2. (20 points total) **Virtual Memory** Consider the simulation of a large wind tunnel on a uniprocessor machine with 100 Megabytes of memory. (For the purposes of this question, assume all the memory is available for application data, and the code size is negligible.) The simulation works by scanning and updating the entire wind tunnel state, represented as K megabytes of data, on each time step. Suppose that a page fault costs 10 milliseconds, with a page size of 4 kilobytes, 8.3. milliseconds rotational latency and 1.7 milliseconds transfer time (for round numbers). A timestep takes approximately 1 second of CPU time for each 10 megabytes of wind tunnel without page faults.

   (a) (7 points) Calculate an estimate of the elapsed time per timestep with 130 megabytes of windtunnel, assuming a simple LRU scheme being used in the virtual memory system. Describe any factors/assumptions that might affect your estimate.

(b) (7 points) Suppose your job was to speed up this program (without getting more hardware), and that the operating system provided virtual memory controls to prepage and flush out portions of the address space. How would you (re)structure the program to take advantage of these facilities, and what sort of performance might you expect when done? Assume each prepage and flush operation cost 1 millisecond of CPU time to start/complete I/O transfers, etc.

(c) (6 points) Describe all the additional (real world) factors that might further limit the performance to be below what seems reasonable from the basic numbers above.

3. (20 points total) **File Systems** Himmel Smackly, a new Masters student, claims that file systems are totally irrelevant for the future because, with 64-bit addressing in processors, we can finally have true single-level store systems, and keep all "objects", including files in the virtual memory system. Your roommate laughs heartily at this, knowing that you missed some great parties to study how file systems work for this exam, and now the technology is obsolete. Put them both in their place by describing:

(a) (7 points) the key functions of file systems.

(b) (9 points) why the basic modules and techniques won't disappear just because of 64-bit addressing.

(c) (4 points) why single-level store may not be the great salvation even for users, applications, etc. that Smackly claims, even if every machine converted to 64-bit addressing tomorrow.

(Note: you may answer this question as three separate parts, or as one combined answer covering all at once.)

Solutions

Comprehensive Exam: Software Systems (60 points) **Autumn 1991**

1. (20 points total) *Concurrency*

   (a) (7 points) Synchronization can be implemented using Dekker's algorithm. Lamport's algorithm.. etc. assuming that BCCI has designed the machine with atomic read and write operations (so a read operation never returns the result of a partial write). Thus. the machine can still be useful but they should market the machine according to the costs of this approach. Namely. the machine would be best with fewer processors (ideally 2) to simplify the Dekker's implementation. Also. the machine would be best with applications with low-contention locks because of the cost of contention on locks. and even the cost of acquiring the locks without contention. (The extreme is to use the machine just for multi-programming. when only the operating system executes with coupled parallelism.) Of course. all this assumes that the Bulgarians dont surprise us by announcing they provided a "split" compare-and-swap instruction like in the MIPS R4000. For the future. you advise them to provide hardware support for locking to support fine-grain concurrency. but no fancy hardware semaphores. etc. (assuming you think that is a danger). Their omission does affect whether processes do busy-waiting. because typical read-modify-write operations simply efficiently support busy-waiting. Process blocking is implemented by the operating system.

   (b) A read-only process can run into problems in a variety of ways. even if there is only one writer. For example, it might be report generator needs to balance income and expense statements. An update during its scan of the data could produce inconsistent results. On the other hand. Smidley might implement a singly-linked list out of memory records that are either free or in the list. A record can be inserted as a single write (after initialization) and removed with a single write. The updater can also increment a generation number for the data structure so the reader processes can determine if the data structure might have changed during their access. and then simply redo the operation if so. Another good example is having the writer update a shadow copy of a data structure and then replace the real data structure with the shadow copy in one atomic write to a pointer, for instance. Answers to the first part that include faking multiple writers, such as having the readers ask another process to do the writes. are considerable suboptimal because they indicate a lack of appreciation of common readers/single writer conflicts, which are important. Answers to the second part that basically have Smidley implementing synchronization (in the sense of causing readers to wait) are also suboptimal because they clearly do not recognize the wide range of scenarios in which blocking can be avoid altogether.

   (c) Paula, Hoare has the signaler suspend, the signaled acquire the monitor lock immediately. and the signaler then resume as soon at the signaled had blocked again or left the monitor. Thus. the signaled process seems the state of the

45

monitor at the time of the signal. but the signaler has to be careful when it resumes. Brinch-Hansen. in actually trying to implement this nightmare (which Hoare never bothered with) in Concurrent Pascal. required that the signal be the last action taken by the signaler before leaving the monitor. But. Paula. just like there are musicologists. and then there are those that produce hit records. in practice. it is more common to implement monitors so as to reschedule the signaled processes but have them wait for the monitor lock. like all other processes. and in particular wait until the signaler departs the monitor. Consequently. a signaled process has to recheck the condition to ensure it is true. despite the original intentions of Tony Hoare. Thus. the signal is a *hint*. not a guarantee. (This reduces the coupling of the monitor mechanism to the scheduler. and reduces the typical number of context switches. all important things in practice.)

2. (20 points total) **Virtual Memory**

(a) The CPU time is clearly 15 seconds per pass. With LRU and a sequential scan of the data corresponding to a more or less sequential scan of the pages. one would expect a page fault on each of the 37.500 pages of data. for a cost of 375 seconds. (There is no overlap of CPU and disk transfer time because we assume we are just brainlessly tripping across each missing page.) Thus, a scan in the simple approach takes 390 seconds. A factor that might make this worst is the mapping of the data onto pages. For example. imagine that the data is arranged as two stripes across all the pages such that we cycle through all the pages twice per timestep. (Sounds bad, but some vectoring processing data organizations do in fact lead to such bad locality behavior!) A simple improvement on this approach is to use an "elevator" approach, where one scans back and forward. so on each scan, one only pages in the "last" 50 megabytes of the scan. so the cost is 125 seconds for page-in plus 15 seconds CPU, for 140 seconds per timestep.

(b) The obvious thing is to prepage and post-flush the pages as we use them. Simplistically, that means we have 2 milliseconds per page-in and out and 37.500 pages. so it takes 75 seconds for CPU to page in pages. and 15 seconds to compute. so 90 seconds. However, the latency for page-in/out in 10 milliseconds it still takes at least 375 seconds to complete page-ins for one scan. or 125 seconds using the elevator trick. (Here, we assume that we can page-in and page-out concurrently. else the times are doubled.) Thus. the page-ahead/behind purely allows the CPU time to be overlapped with the disk activity. but the elapsed time is the maximum of the two. which is the disk in both cases. Clearly. this elapsed time would be reduced if we could prepage multiple pages simultaneously, like if we had four independent disk channels, two simultaneous page-in and two for simultaneous page out. Then. the disk latency is reduced to 75 seconds with the evalator approach. and CPU time dominates, so a timestep is 90 seconds. A similar effect occurs if we can do multi-page transfers, with a single charge of 1 ms for initiating. one charge for rotational latency, and double or whatever the transfer time. (There

are generally limits on the number of pages of contiguous transfer without a seek or rotational delay though.) Clearly, that could get the disk I/O fully overlapped with the CPU time as well. There is at least one additional scheme for minimizing paging (without the elevator/reverse scan) but that is beyond what we consider here (Contact David Cheriton if interested.)

(c) A key real world factor not mentioned is seek time, which can be 10's of milliseconds, so significant. Without significant locality, seek time could easily triple the disk access time, and thus triple the elapsed time for the timestep. Another factor is the available I/O bandwidth. As described above, the elapsed time with 4 channels is far less than one channel. The final other factor is contention with other applications and system services running on the machine, which could place additional competing demands on the memory, I/O bandwidth and CPU, further slowing down the execution of the program. There is some effect because of paging of the operating system data structures themselves, such as page tables. However, with any reasonable VM design, the page tables are a small percentage of the size of the virtual memory, like 5 percent or less, so their overall effect should be minimal. Also, the page tables for a read-ahead page are naturally referenced as part of the read-ahead, so one would not expect random, unpredictable page faults from these structures either.

3. (20 points total) **File Systems**

(a) The key functions of a file system include file access (i.e. open, close, read, write) implementation — the basic file abstract data type implementation — file data buffering, disk allocation of space for files, file directory implementation, file protection, and facilities for backup and recovery. As part of the implementation, there are generally heuristics and algorithms that recognize common file properties, such as most files are short, most files are read sequentially in total, and most files are short-lived.

(b) The 64-bit addressing allows one to address all files as part of this large address space in theory (perhaps) but there are problems in practice. First, most files are small but some are very large, and incrementally grow to a large size, like log files. Locating gigabytes of address space per file could use up even a 64-bit address space, but less means that a file might have to change its name as it got bigger. Second, users will want to identify files by character-string names anyway, so there still has to be a file directory system. This directory system or some additional mechanism would still be required to implement file protection and security, which is typically lacking from a virtual memory system (because only one address space can access its segments, in the conventional model). Thirdly, there is an evolution problem in practice. Some machines may have 64-bit addressing now, but when will all have it? One could argue that the existing mechanisms in a typical virtual memory mechanism for buffering (the page pool), disk allocation

and disk transfer can subsume that of the file system. but these tend to be the same basic mechanisms used in the file system. so one is considering a reduction on software. not elimination of the techniques I have carefully studied. Moreover. the trend is actually to have file systems mechanisms subsume virtual memory mechanisms (including the unification of the file and page buffer pools) because the file system mechanisms tend to be more powerful. In particular. one would like to retain the file optimizations for read-ahead with sequential access. etc. regards of whether you regard this as virtual memory system or file system. Also. virtual memory systems have not conventionally provided facilities for persistent storage in the area of backup and recovery. so file functionality there would also be required. By the way. 64-bit addressing does not necessarily imply any changes in the amount of data kept in physical memory. just that kept in virtual memory. So. yes I missed some great parties. but I still learned some important stuff!

(c) Himmel. too bad you were watching Sesame Street when the dudes at MIT were developing Multics. which basically did all this stuff 20 years ago. It didnt exactly catch on. did it? More seriously. many applications and programmers seem to just prefer the stream read/write interface rather than a memory-mapped I/O interface to files. For one. it is safer because if your program goes wild. it is less likely to corrupt the file. One might view that it throws file access into the memory allocation problem. and memory corruption bugs are some of the hardest for programmers. Also. lots of I/O refers to things other than disk files. such as terminals. pipes. communication lines. etc These object do not have convenient fixed-size blocks that are compatible with the virtual memory page size (necessarily). so these devices would have to be handled by a separate mechanism. a major step back from the Unix uniform I/O mechanisms. In general. history is on my side: the idea of single-level store has been around for a long time. and while not a failure completely. it has failed to displace the stream model of I/O. and conceptual separation between virtual memory and secondary storage.