Student Name: _____

This question will explore your understanding of performance, power, and energy consumption trade-offs in modern processors. It will probably help you to write down the basic equations that calculate the execution time (T), consumed power (P), and consumed energy (E) when running some application on a processor:

T = Execution Time =

P = Power Consumption =

E = Energy Consumption =

a) What is the impact of decreasing the processor clock frequency on T, P, and E? What is the impact of decreasing the processor power supply on T, P, and E? Explain briefly.

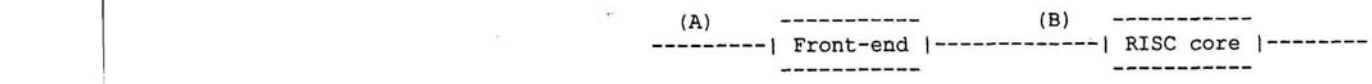
b) Assume an application that is easily parallelizable and that you are not interested in performance improvement. How can the use of two processors (parallel processing) help you achieve significant reductions in P and E? Explain briefly.

c) In terms of performance, RISC instructions sets are typically preferred over CISC instructions sets: compilers can produce better executable code for RISC and it should be easier to build a high frequency RISC processor. From the point of view of power consumption, how would you compare RISC and CISC instruction sets?

d) Modern superscalar processors use speculative execution based on a number of prediction techniques (branch prediction, memory dependence prediction, etc). What is the likely impact of speculation on T, P, and E? Explain briefly.

NAME: Christos Kozyrakis

Modern CISC processors (e.g. the superscalar Pentium 4 or Athlon designs) are not as CISC as they used to be. Often, a RISC core lies at the heart of the processor and a "front-end" translates a CISC instruction to a sequence of one or more RISC-like micro-instructions.



1) Designers of CISC processors are faced with the decision of placing an instruction cache a location (A) or at location (B). Discuss the advantages and disadvantages of each alternative, particularly with respect to wide superscalar microarchitectures.

A particular CISC processor has a cache only at location (A). Its RISC core could achieve CPI of 1.0, if it was given a steady stream of micro-instructions. However, the front-end takes 5 cycles (unpipelined) to translate a CISC instruction. On the average, each CISC instruction translates to 3 RISC microinstructions.

A (B) cache is proposed as an alternative. On a hit, this cache can produce 1 translation per cycle. Misses take 12 cycles. What hit rate would you need for the (B) cache in order to operate the RISC core at its peak performance? What speedup does this give you over the original processor with the cache in location (A)?

Tip: assume that both alternatives have sufficient buffering at proper locations in order to stream-line execution.

EE Quals 2005 Name: Christoforos Kozyrakis

Vector architectures use instructions (load/store and arithmetic) that define operations on arrays of numbers (vectors). The following vectorizable loop:

for (i=0; i<64; i++) c[i] = a[i] + b[i];

can be expressed with two vector loads for a and b, an element-wise vector add, and a vector store for c. Each instruction specifies 64-bit independent element operations. A vector processor typically executes multiple of these operations per cycle (e.g. 8 elements adds per cycle for the vector add in our example).

For vector processors to be useful, the loop must be vectorizable. For the following loops:

- Identify if they are vectorizable: yes/no, why, how, under which conditions, or with what hardware or instruction set extensions/requirements…

You can discuss the loops in any order you want. Assume N is large (100s or 1000s).

1	for (i=0; i<N; i++)	c[i] = c[i+1] + b[i];
2	for (i=0; i<N; i++)	c[i] = c[i-1] + b[i];
3	for (i=0; i<N; i++)	if (b[i]!=0) c[i] = a[i] / b[i];
4	while (b[i]!=0)	{ c[i] = a[i] / b[i]; i++; }
5	for (i=0; i<N; i++)	t += a[i] * b[i];
6	for (i=0; i<N; i++)	c[i] = a[d[i]] + b[i];
7	for (i=0; i<N; i++)	c[d[i]] = a[i] + b[i];
8	for (i=0; i<N; i++)	c[d[i]] += a[i] + b[i];
9	for (i=0; i<N; i++)	c[2*i] = a[2i+1];

1	for (i=0; i<N; i++) c[i] = c[i+1] + b[i]; Always vectorizable
2	for (i=0; i<N; i++) c[i] = c[i-1] + b[i]; NOT vectorizable (loop-carried dependency distance 1)
3	for (i=0; i<N; i++) if (b[i]!=0) c[i] = a[i] / b[i]; Vectorizable Requires support for conditional vector execution
4	while (b[i]!=0) {c[i] = a[i] / b[i]; i++} Partially vectorizable (vector search for zero, set vl, vector divide) Fully vectorizable with speculative vector support
5	for (i=0; i<N; i++) t += a[i] * b[i]; Partially vectorizable by using tree reduction (vector temporary) Fully vectorizable if reduction permutation supported
6	for (i=0; i<N; i++) c[i] = a[d[i]] + b[i]; Vectorizable Requires support for indexed instructions
7	for (i=0; i<N; i++) c[d[i]] = a[i] + b[i]; Vectorizable if - d[i] elements are disjoint OR - HW executes vector stores in order
8	for (i=0; i<N; i++) c[d[i]] += a[i] + b[i]; Vectorizable ONLY if d[i] elements are disjoint
9	for (i=0; i<N; i++) c[2*i] = a[2i+1]; Vectorizable with strided accesses

Comments:

NAME Christos Kozyrakis DATE _____

- **What is the base idea behind the stored-program concept?**

The idea is to use a single storage (memory) in a computer to store both instructions and their data. It was introduced by von Neumann in the 40s,

- **What are the advantages of the stored-program concepts? Provide some examples.**

Better utilization of the storage

We can write programs that write programs (compilers, assemblers, JITs etc).

- **What are the disadvantages of the stored program concept? How can we address this problem?**

Security: a malicious (or buggy) program can produce data that are later treated as code and lead to crashes or other serious security issues for the application or the whole system. Buffer overflows that overwrite return addresses in stacks are a good example.

Various protection mechanisms could help: non-execute bits, virtual memory, etc.