Uppsala Universitet
Instutitionen för informationsteknologi
Avdelningen för datorteknik

Kursens Namn: Datorarkitektur 2		Datum 2002-12-18
Namn (efternamn först, v.g. texta)		Utbildningsprogram (eller liknande)
Namn (namnteckning)		Personnummer (10 siffror)
Termin och år då du först registrerade kursen	Inlämningstid	Bordsnummer

X= Solution provided ☺=I have bonus point		Your Score	Max score
1			8
2			8
3			8
4			8
5			8
6			8
7			8
8			8
Summa poäng:			64
Betyg:			-

Tentamen i Datorarkitektur 2

(1DT636, Datorarkitektur MN2 och 1IT190, Datorarkitektur IT)

Onsdagen 2002-12-18

Place: Polaksbacken
Time: 14:00 – 19:00
Allowed help: Calculator, pencil, and eraser.
Language: Answer in Swedish or English

General information

- I will turn up at 15:30 and 17:00 to answer questions.
- Solve at most one "problem" on each sheet (there are eight problems in total)
- Write your name on all pages
- If anything seems unclear in the question, state your assumptions clearly.
- Not readable and/or not understandable answers result in zero points.
- If you did not bring a calculator you still need to estimate the answer value (and state that you did not bring a calculator).
- Mark each question for which you are providing a suggested solution with an "**X**" in the first column on the cover page.
- Check the bonus pages to see if you have earned any "bonus questions". You will automatically be given maximum points for the corresponding bonus questions. If so, put a "☺" in the first column for those questions on the cover page.

// ERIK

I can be reached at 070-425 0502 during the exam.

Bonus Dark2 2002

Namn	Pnr	H1	H2	H3	Simics	Stride
Anderson						
Anth						
Bemersjö		G	G	G	G	G
Benson		G	G	G	G	G
Bergsten		G				
Billquist		G	G	G	G	G
Bjurefors		G	G	G	G	G
Bran		G				
Cabric		G	G	G	G	G
Dahlqvist						
Ekström		G	G	G	G	G
Hadefjell		G	G		G	G
Hedquist		G				
Heggbrenna		G	G	G	G	G
Holmén		G	G	G	G	G
Holmgren		G			G	
Jespersen		G	G	G	G	G
Johansson		G	G	G	G	G
Johnsson		G	G	G	G	G
Jorlin		G	G		G	G
Linden		G	G	G	G	G
Lindström		G	G	G	G	G
Lundberg		G	G			
Lundgren		G	G	G	G	G
Lundström		G				
Lundqvist			G	G	G	
Lyrberg		G			G	
Lönnberg		G	G	G	G	G
Nyberg		G	G	G	G	G
Olsson, H		G	G	G	G	G
Olsson, M		G			G	
Oksvold		G				
Otto					G	
Persson		G	G		G	G
Petersson		G	G	G	G	G
Rensfelt			G	G	G	G
Rogell		G	G	G	G	G
Rosen		G	G	G	G	G
Röjdeby		G	G	G	G	G
Vernersson		G	G	G	G	G
Wennerström		G	G	G	G	G
Wibbling		G	G	G	G	G
Ylitalo		G	G	G	G	

1. (Bonus 1) Caches

a) Make a drawing of a physical cache with the following data

- Cache size: 8Mbyte
- Cache line size: 64byte
- Organization: 4-way
- CPU word size: 4 byte (this is the size of the data unit delivered from the cache to the CPU)
- Physical address size: 44 bits (i.e., 16Tbyte of address space can be addressed)

Clearly show the address bits ranges used for the indexing, the comparisons, and multiplex (mux) selects, etc. Describe, either in words or in logic, the functionality of all logic needed to resolve a read access to the cache.(4p)

b) Which bit ranges would change if the cache size was decreased to 4Mbyte while all the other parameters stayed the same? State the new bit ranges (you do not need to make a completely new drawing) (2p)

c) How would the cache change if the cache in a) also was made 2-way subblocked (2p)

2. (Bonus 2) Loop Scheduling

Consider the loop and the corresponding compiler-generated code

```
for (i=1; i<=1000; i++)
     x[i] = x[i] + s;
```

```
loop:   LD F0, 0(R1)        ;line 1
        ADDD F4, F0, F2     ;line 2
        SD 0(R1), F4        ;line 3
        SUBI R1, R1, #8     ;line 4
        BNEZ R1, loop       ;line 5
```

Assume that the array is stored in "backwards order" in the array, i.e., the address on x[i+1] is 8 smaller than the address of x[i].

The pipeline has the following characteristics	Delay
FP ALU op to another FP ALU op:	3 cycles
FP ALU op to store double:	2 cycles
Store double to FP ALU op:	1 cycles
INT ALU op to another INT ALU op	0 cycles
Branch delay slots:	1 cycles

a) Show where the bubbles are and write one-sentence comments for each line (1p).
b) Show how the loop can be statically scheduled to improve the performance and calculate the number of cycles required for each iteration (no unrolling yet). How many cycles per iteration? (2p)
c) Show how loop unrolling with maximum optimizations could help avoiding <u>all</u> the stalls in this loop using the smallest amount of unrolling. You do not have to show the set-up/clean-up code before and after the loop. How many cycles per iteration? (2p)
d) Show how software pipelining can be used to remove the bubbles (no unrolling is allowed, you do not need to show the prologue and epilogue code) (3p)

3. (Bonus 3) MP

(At most three sentences per subquestion)

a) List two kinds of cache misses that exist in a cache-coherence multiprocessor, but not in a uniprocessor, and describe how they occur.(2p).
b) List the two main reasons why a snooping-based coherence scheme may be preferred over a directory-based system (2p)
c) All parallel threads in a cache-coherence multiprocessor should increment the shared variable "A" by one. Write the code (including the calls to a synchronization library) that makes that increment in an atomic fashion.(2p)
d) What is the main difference in the behavior between a MSI and a MOSI protocol.? (2p)

4. Memory Ordering and Synchronization

Dekker's algorithm is used to demonstrate why memory consistency matters:

```
                    Initially:
                       A:= 0;
                       B:= 0;
Thread 0:                                   Thread1:
   A:= 1;                                      B:=1;
   if (B==0) then                              if (A==0) then
      print("Thread 0 won");                      print("Thread 1 won");
```

a) What is the purpose of this algorithm, and under which of the memory models, sequential consistency and release consistency, would it work? (2p)
b) How would the code be changed in order to work for the other model too? (2p)

Synchronization:

c) Write a simple synchronization primitive for lock(addr) and unlock(addr) such that waiting threads do not produce any global coherence traffic while they wait for a busy lock to become free. Use the atomic function SWAP(tmp,addr), which atomically writes the value of `tmp` into memory location addr, and returns the value previously stored at addr. (2p)
d) Name two techniques that significantly can reduce the coherence traffic each time a contended lock is released.(2p)

5. Random Questions

Which of these statements are true? (8p)
(You'll get 8p if you list them all. 3p will be deducted for each one you do not list, or list even though it is false. You will never get less than 0p ☺)

a) OpenMP supports a shared memory programming paradigm
b) MPI supports a message-passing programming paradigm
c) The dynamic power consumption is proportional to the square of the chip's supply voltage.
d) All coherent shared memory system have a uniform access time to the shared memory.
e) A memory scrubber is a piece of software that frees up unused page frames in the virtual memory system
f) The latency for communicating 1mm on a chip is likely to increase with future CMOS chip technologies
g) The number of memory bits accessible in one CPU cycle is likely to decrease with future CMOS chip technologies
h) Out-of-order execution in combination with an accurate branch prediction scheme can speed up execution through speculative execution.

6. Cache Coherence

All the three RISC CPUs in a MOSI shared-memory sequentially consistent multiprocessor executes the following code almost at the same time:

```
while(A != my_id){};/* this is a primitive kind of lock*/
B := B + A * 2;
A := A + 1;         /* this is a primitive kind of unlock */
while (A != 4) {};  /* this is a primitive kind of barrier */
<after a long time …>
<some other execution replaces A and B from the caches, if still present>
```

Initially, CPU1 has its local variable my_id=1, CPU has my_id=2 and CPU3 has my_id=3 and the globally shared variables A is equal to 1 and B is equal to 0. CPU2 and 3 are starting slightly ahead of CPU1 and will execute the first while statement before CPU1. Initially, both A and B only reside in memory.

The following four bus transaction types can be seen on the snooping bus connecting the CPUs:

- RTS: ReadtoShare (reading the data with the intention to read it)
- RTW, ReadToWrite (reading the data with the intention to modify it)
- WB: Writing data back to memory
- INV: Invalidating other caches copies

Rip out and fill in the solution sheet at the end of the exam. Show every state change and/or value change of A and B in each CPU's cache according to one possible interleaving of the memory accesses. After the parallel execution is done for all of the CPUs, the cache lines still in the caches will be replaced. These actions should also be shown. For each line, also state what bus transaction occurs on the bus (if any) as well as which device is providing the corresponding data (if any). (8p)

As a guide to how you should fill out the table, this small example shows a parallel program where CPU1 executes a load(A) instruction after which CPU2 executes a load(B) instruction, followed by the replacement actions to empty the caches. A and B are initially 1 and 0.

CPU action	**Bus Transaction (if any)**	**State/value after the CPU action**						**Data is provided by** [CPU 1, 2, 3 or Mem] (if any)
		CPU1		CPU2		CPU3		
		A	B	A	B	A	B	
Initially		I	I	I	I	I	I	
CPU1: LD A	RTS(A)	S/1						Mem
CPU2: LD B	RTS(B)				S/0			Mem
…some time elapses .								
CPU1: replace A	-	I						-
CPU2: replace B	-				I			-

7. VLIW Scheduling

Consider the following code fragment:

```
            tmp = a+1;
            if (--i != 0) then {
                 b=b+tmp;
            }
            else {
                 c=c+tmp;
            }
            e = e+tmp;
```

which is compiled into this un-optimized pseudo-assembler code:

```
            (Initially R6 contains the value "i")

            LD R1, [A_ADDR]
            ADD R1, R1, #1        ; R1 = tmp = a + 1
            SUB R6, R6, #1        ; i = i - 1
            BREZ R6, #ELSE
            NOP
            LD R2 [B_ADDR]        ; R2 = b
            ADD R2, R2, R1
            ST R2 [B_ADDR]
            BR #CONT
            NOP
ELSE:       LD R3 [C_ADDR]        ; R3 = c
            ADD R3, R3, R1
            ST R3 [C_ADDR]
CONT:       LD R5 [E_ADDR]        ; R5 = e
            ADD R5, R1, R5
            ST R5 [E_ADDR]
```

The the pipeline has the following characteristics:
One branch F-unit with one delay slot after all branches
Two LD/ST, (both can perform both loads and stores) F-units that each can start a memory access each cycle with a latency from LD to usage of 2 cycles
One integer ADD/SUB F- unit that can start one operation per cycle, with a latency from ADD or SUB to usage of 0 cycles (i.e., the result can be used during the next cycle

Rewrite an optimize the code for each of these assumptions, show where the stall/NOP cycles are and estimate how many cycles each example takes to execute along the "then" path. (You may use the paper at the solution sheet at the end of the exam to fill in you answer.) The new code should not cause any exceptions that the old code would not have produced.
Assumptions:

a) a pipeline that only can issue one instruction per cycle. Make the obvious optimizations (1p)
b) a VLIW pipeline that can issues four instructions, but at most one instruction per F-unit per cycle. Schedule the program with one column per F-unit. Fill out with NOPs if not all F-units can be used during one cycle. You can not use any other instruction types than the ones in the example (2p)
c) the same as b), but you may use speculative loads (LD.s) and checks (LD.c), which are both handled by the LD/ST units. The LD.c has a zero cycle latency to usage (2p)
d) same as b), but using predicate execution instructions instead. Assume a new instruction `PREDZ R1, P2` that sets P2=1, P3=0 if R1 is equal to zero and P2=0, P3=1 otherwise. Write each instruction INST that depends on the predicate register P1 as `P1::INST`. The latency from the PREDZ instruction to the usage of the predicate registers is 0 cycles and is handled by the branch F-unit in this example (2p)

e) As illustrated by this example, traditional VLIW can suffer from huge code expansion. Briefly mention how this has been overcome in the EPIC architecture (i.e., in the IA-64 and Itanium implementations) (1p)

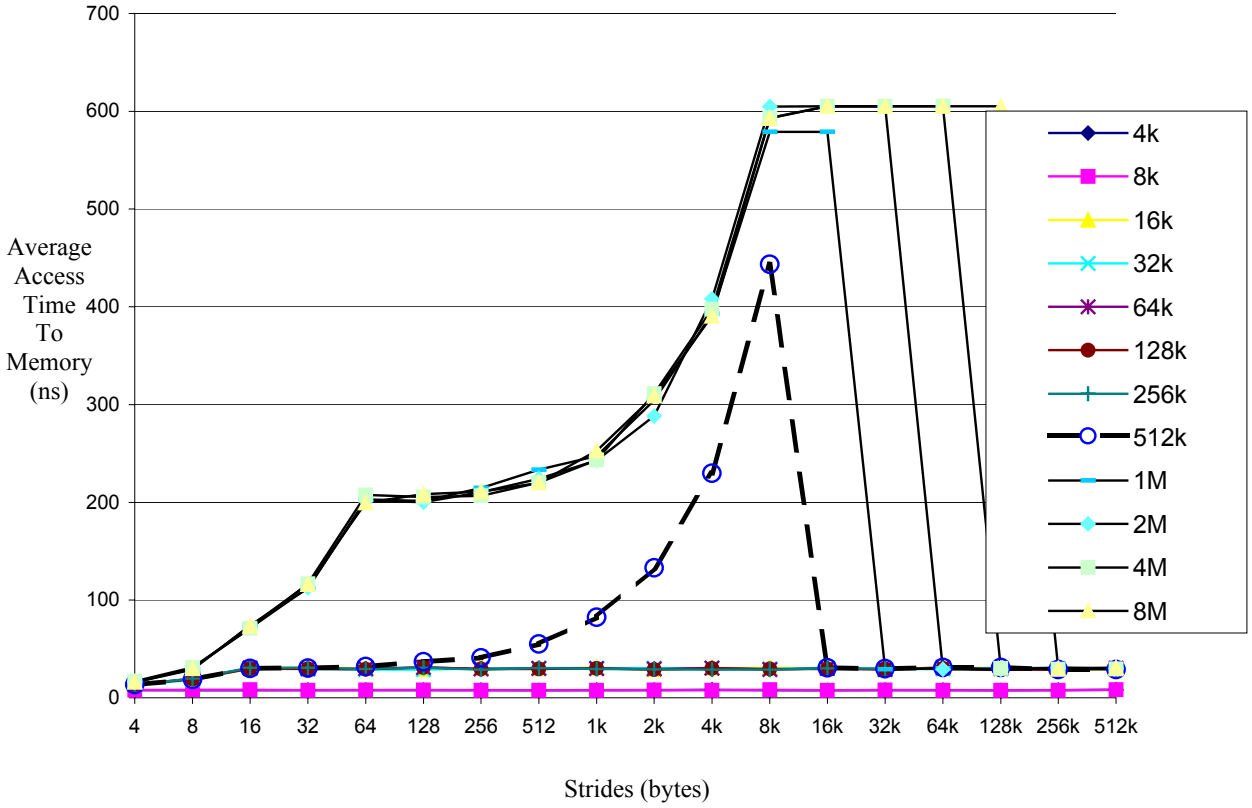
8. Microbenchmarks

A simple microbenchmark code below is used to draw curves that characterizes the properties of the underlying memory system .

```
for (times = 0; times < Max; time++) /* many times*/
  for (i=0; i < ArraySize; i = i + Stride)
    dummy = A[i]; /* touch an item in the array */
```

This page shows such a curve for a computer system. Curves 4k – 32k are on top of each other furthest down.

a) What is the cache line size and cache size for the 1st level cache? (1p)
b) What is the cache line size and cache size for the 2nd level cache? (1p)
c) What is the page size , how many entries does the TLB have and what can you say about its associativity(2p)
d) How would the graph change if the TLB was made twice as large, but everything else stayed the same? (2p)
e) How would the graph change if the TLB size did not change, but the pages were half the size? (2p)



Solution sheet Question 6 Your Name ____

CPU action	Bus transaction	State/value after the bus transaction						Data is provided by [CPU 1, 2, 3 or Mem]
		CPU1		CPU2		CPU3		
		A	B	A	B	A	B	
Initially		I	I	I	I	I	I	

Name:_____

Solution sheet 7a)

cycle	Instruction
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
1	
2	
13	
14	
15	
16	
17	
18	
19	
20	
21	
22	

7b)

Cycle	LD/ST	LD/ST	ADD/SUB	Branch
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
1				
2				
13				
14				
15				
16				
17				
18				
19				
20				
21				

Name:_____

Solution sheet 7c)

Cycle	LD/ST	LD/ST	ADD/SUB	Branch
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
1				
2				
13				
14				
15				
16				
17				
18				
19				
20				
21				

7d)

Cycle	LD/ST	LD/ST	ADD/SUB	Branch
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				
11				
1				
2				
13				
14				
15				
16				
17				
18				
19				
20				
21				