Lösningar till Tentamen i

Datorarkitektur 2

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

Torsdagen 1999-12-16

Place:	Polacksbacken skrivsal
Time:	15:00 – 21:00
Allowed help:	Pencil, eraser and a good attitude
Language:	Swedish or English

General information

- If you are not sure about what is described in a question, clearly state you assumptions
- Solve at most one "Problem" per page (there are 9 problems total)
- Write your name on all pages
- Do not cross-reference between the solutions of different problems
- Not readable and/or not understandable answers result in zero points.
- Check the second page to see if you already can account for maximim points for the bonus questions 5 and 7.
- Solutions can be found outside Erik's office 99-12-17
- The availability of the result and the location of the list will be announced on the course home page around Jan 15th.
- Have a nice holiday.

May the force be with you,

// ERIK

Erik Hagersten will show up twice during the exam: 16:00 and 19:00.
Erik can be reached at 070-425 0502

Problem 1, Random questions (10p)

a) Show how the expression C := A + B is computed on a accumulator architecture (2p)
Load A
Add B
Store C

b) What is key advantage of a accumulator architecture over a load/store architecture? (one sentence) (2p)
Smaller code size since the operands are implicit in the instruction format

c) *i)* What is a pipeline hazard?
Describe the different types of pipeline hazard:
ii) structural hazards,
iii) data hazards and
iv) control hazards
(one sentence each)(4p)

i) A situation that delays the next instruction in the instruction stream from being executed
ii) Conflict between instructions requiring the same resource at the same time
iii) An instruction needs results from a previous instruction sooner than it is available
iv) Instructions that changes the PC value causes hickup in the instruction fetch since the sequential stream of instructions is broken and the new PC value must be decoded and/or calculated before the instruction can be fetched.

d) What is a TLB? One sentence (2p)
A cache indexed by virtual addresses storing the most recent virtual to physical address translations.

Problem 2, Caches (8p)

a) What is a victim cache and what kind of cache misses can it potentially remove? (2p)
I is a [small and often fairly associative] cache that holds the most recently evicted (e.g., replace) cache lines. It is searched in parallel with the "normal" cache on each lookup and removes conflict misses.

b) What is the LRU replacement algorithm? (2p)
The Least Recently Used algorithm keeps track of the access history to a set in the cache (e.g., to all the four "ways" in a 4-way associative cache) and selects the way that has been least recently accessed for replacement.

c) Why are ***Spatial*** and ***Temporal*** locality and why are they important for cache-based systems? (4p)
Spatial locality: data located nearby the currently accessed data are likely to be accessed soon.
Temporal locality: the data currently accessed are likely to be accessed soon again.
A cache stores cache blocks that has been accessed in the recent past, without any temporal locality the execution would never need that data again and the cache would not be to any use. Normally, a cache holds not only the accessed data, but also some surrounding data in each cache block. Without spatial locality that would not make any sense.

d) What kind of cache misses (Mark Hill's three "C" misses) are removed by adding
i) more associativity to a cache,
ii) larger cache blocks to a cache?
iii) What kind of misses are increased in a multiprocessor with larger cache blocks (does not start with a "C")? (One sentence each) (3p)

i) More asociativity removes conflict misse
ii) Larger cache blocks removes compulsory misses
iii) False sharing

Problem 3, virtual memory system (8p)

a) What is a virtually indexed physically tagged (VIPT) cache? (2p)
A cache indexed using some portion of the virtual address, containing a physical address tag which is compared with the physical address which is a result from the TLB lookup.

b) When is the VIPT accessed? Before, in parallel with or after the TLB(2p)
In parallel with the TLB access

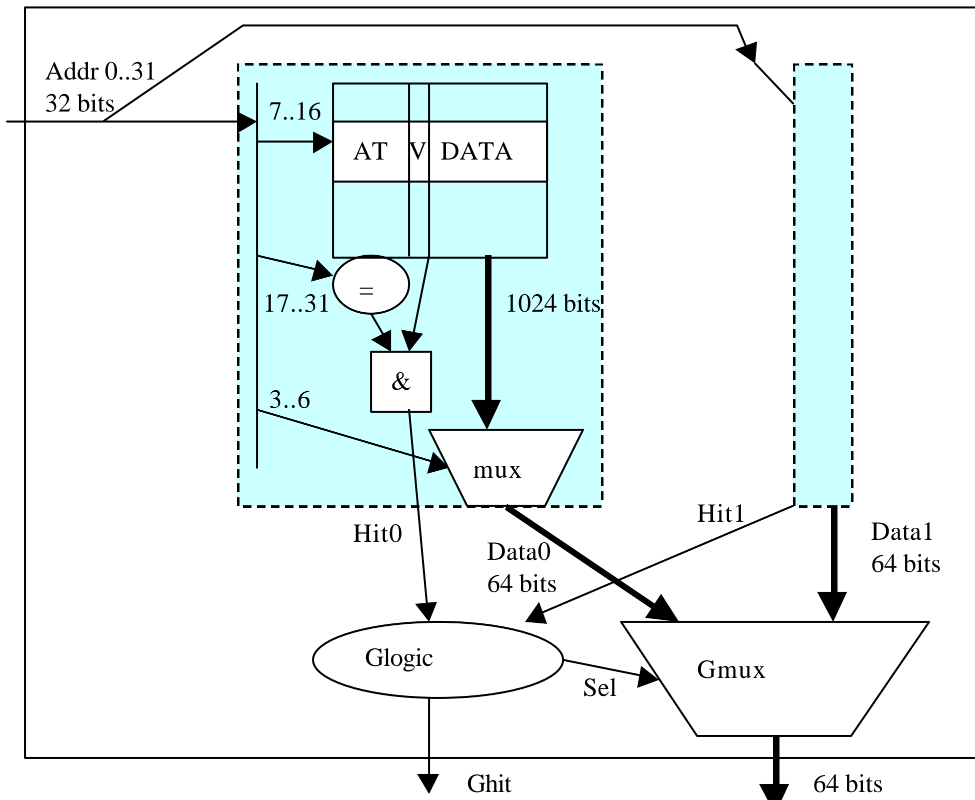
c) The virtual memory system is sometimes described as some kind of a cache system.
   i) What is its cache block size?
   ii) Is it a physically or virtually indexed cache?
(one sentence each) (2p)
i) The bock size is identical to the page size
ii) It is virtually indexed

d) Suggest one way to overcome the aliasing problem present in virtual caches. Why does your proposed technique work? (three sentences) (2p)
Make the first-level cache smaller than the physical (and virtual) page size. That way, not more than one synonym can reside in the cache at once.

Problem 4, cache organization (8p)

a) Make a drawing of a physical cache with the following data
- Cache size: 256kbyte
- Cache line size: 128B
- Organization: 2-way
- CPU word size: 8 byte (this is the size of the data unit delivered from the cache)

Physical address size: 32 bits (i.e., 4Gbyte of address space can be addressed)
Clearly show the address bits used for the indexing, the comparisons, and multiplexer (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)



The two identical blocks each produces two identical signals: hit and data, numbered from left to right: hit0, hit1 and data0, data1. At most one of the hit signals will be asserted at any one time. The new Gmux selects is controlled by the two hit signals and selects the data corresponding to the block with its hit signal asserted. The Ghit signal is the OR function of the two hit signals.

Problem 5, loop scheduling (8p)

BONUS assignment 2:
If the second page of the test indicates that you are eligible for BONUS 2nd assignment, don't do this problem.
Consider the loop and the corresponding compiler-generated code

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

```
loop:      LD F0, 0(R1)          ;line 1
           LD F2, -8(R1)         ;line 2
           ADDD F4, F0, F2       ;line 3
           SD 0(R1), F4          ;line 4
           SUBI R1, R1, #8       ;line 5
           BNEZ R1, loop         ;line 6
```

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
INT ALU OP OUTPUT to INT ALU OP INPUT:	0 cycles
FP ALU OP OUTPUT to FP ALU INPUT:	2 cycles
LD DATA to INT/FP INPUT:	2 cycles
FP ALU OUTPUT to ST INPUT:	2 cycles
Branch delay slots:	2 cycles

a) Write one-sentence comments for each line (2p)
Line1: F0, F1 is now x[i]
Line2: F2,F3 is now x[i+1]
Line3: Store the sum in F4,5
Line4: Write the result to memory x[i]
Line5: Decrement array pointer
Line6: If array pointer != 0, loop back to line 1

b) Show where the bubbles are in each and calculate the number of cycles in each iteration? (2p)

```
loop:      LD F0, 0(R1)          ;line 1
           LD F2, -8(R1)         ;line 2
           stall
           stall
           ADDD F4, F0, F2       ;line 3
           stall
           stall
           SD 0(R1), F4          ;line 4
           SUBI R1, R1, #8       ;line 5
           BNEZ R1, loop         ;line 6
           stall (or NOP)
           stall (or NOP)
```

12 cycles

c) Show how the loop can be statically scheduled to improve the performance and calculate the number of cycles required for each iteration. (2p)

```
loop:      LD F0, 0(R1)          ;line 1
           LD F2, -8(R1)         ;line 2
           SUBI R1, R1, #8       ;line 5
```

```
            stall
            ADDD F4, F0, F2          ;line 3
            BNEZ R1, loop            ;line 6
            stall (or NOP)
            SD 8(R1), F4             ;line 4
```

Each iteration of i takes 8 cycles

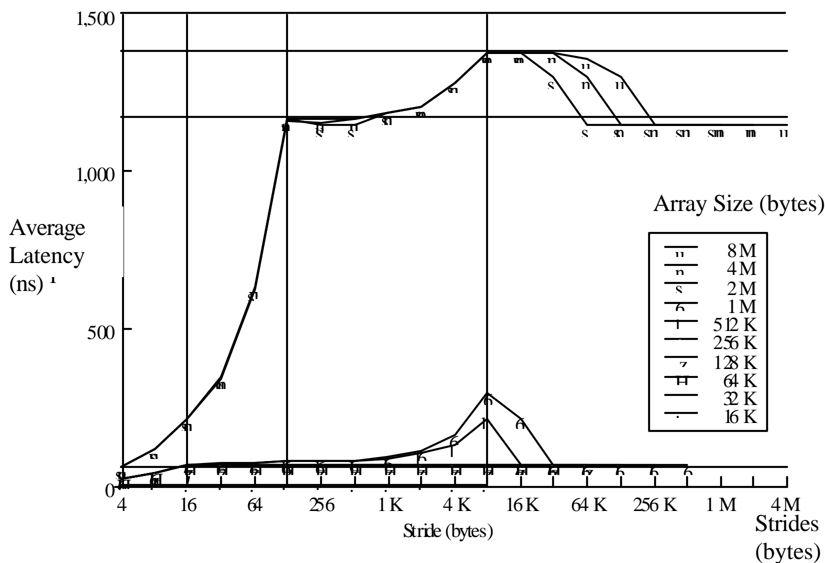
d) Show how loop unrolling with maximum optimizations could help avoiding all the stalls in this loop using the smallest amount of unrolling. You can not introduce any new instruction types. You can only re-schedule, remove or re-offset existing ones. You do not have to show the set-up/clean-up code before and after the loop (4p)

```
loop:       LD F0, 0(R1)             ;line1 setup            F0 := x[i]
            LD F2, -8(R1)            ;line2                  F2 := x[i+1]
            LD F6, -16(R1)           ;line 2'                F6 := x[i+2]
            SUBI R1, R1, #16         ;line 5 & line 5':      i := i+2
            ADDD F4, F0, F2          ;line 3:                F4 := x[i+1] + x[i]
            ADDD F8, F6, F2          ;line 3'                F8 := x[i+2]+x[i+1]
            BNEZ R1, loop            ;line 6 &line 6'        done yet?
            SD 16(R1), F4            ;line 4                 x[i]:=F4
            SD 8(R1), F8             ;line 4'                x[i+1]:=F8
```

Each iteration of i takes 4.5 cycles
(You are not allowed to use the MOV instruction in this example)

Problem 6, microbenchmarks (8p)

A computer architecture has been characterized with using a simple microbenchmark according to the graph below.



Arrays are repeatedly walked through using different distances (strides) between the subsequent accesses. For each array size {16k bytes .. 8M bytes}, the average access time is plotted as a function of the stride size.

a) What is the cache sizes for the L2 caches, what is its block size and what is the memory access time? (4p)

L2 size = 1Mbyte (The 2M array curve is the first one to sky-rocket)
L2 block = 128byte (The sky rocketing plateaus at memory access time 1200ns for strides 128, i.e., there is no longer any spatial locality since each cache line is only touched once.
Memory access time = 1200ns (as show by the Memory plateau for strides>L2 cache line size)

b) Estimate the upper and lower bound for the number of TLB entries in this architecture (2p)

The 2M array still has problems with the TLB at strides 16k, i.e., there are not enough entries in the TLB to avoid thrashing. There must be less than 2M/16k entries, i.e., less than 128 entries. It is getting luckier with the TLB for 32k strides (~64 entries) and hits again all the time for stride 64k (~32 entries). Based on this it is safe to say that there are between 128 and 32 entries in the TLB. (The TLB organization and replacement algorithm blurs the picture a bit)

Problem 7, Bus bandwidth (8p)

BONUS assignment 1
If the second page of the test indicates that you are eligible for BONUS 1st assignment, don't do this problem

You have a system with the following properties

- 10% of the instructions are stores
- 20% of the instructions are loads
- 90% of the load hit in the 1st level data cache
- 90% of the load accesses to the 2nd level cache hit
- 80% of the stores hit in the 1st level data cache
- 75% of the stores accesses to the 2nd level cache hit
- All the instruction fetches hit in the separate L1 cache instruction cache
- The CPU is a "perfect" 1 CPI pipeline, if all the loads and stores hit in the 1st level cache.
- The CPU clock frequency is 100MHz, in-order, no write buffer, and with blocking caches.
- There is a 10 cycle stall if you miss in L1 data cache and hit in the L2 cache, both for loads and stores.
- There is a 100 cycle stall if you miss in L1 and L2 caches and have to access memory, both for loads and stores
- There is unlimited bandwidth at all levels of the hierarchy
- The cache line size is 64 bytes

a) What is the CPI for the system if the overhead in the memory system is also taken into account(4p)
b) What is the bandwidth of the memory traffic for a copy-back cache, assuming that 20% of the L2 misses causes a write-back (and that there is inclusion between the L1 and L2)? (2p)
c) What is the bandwidth of the memory traffic for a write-through cache, assuming no allocate on write and that all memory writes are 64 bytes? (2p)

a) $CPI=CPI_{BASE} + CPI_{LD} + CPI_{ST}$
CPI_{LD} =fraction_loads * fraction_L1misses * (fraction_L2hits * L2hit_time + fraction_Lmisses * mem_time) =
= 0.2 x 0.1 x (0.9 x 10 + 0.1 * 100) = 0.38
CPI_{ST} =fraction_stores * fraction_L1misses * (fraction_L2hits * L2hit_time + fraction_Lmisses * mem_time) =
= 0.1 x 0.2 x (0.75 * 0.10 + 0.25 x 100) = 0.65
CPI = 1 + 0.38 + 0.65 = 2.03 i.e., approximately 2

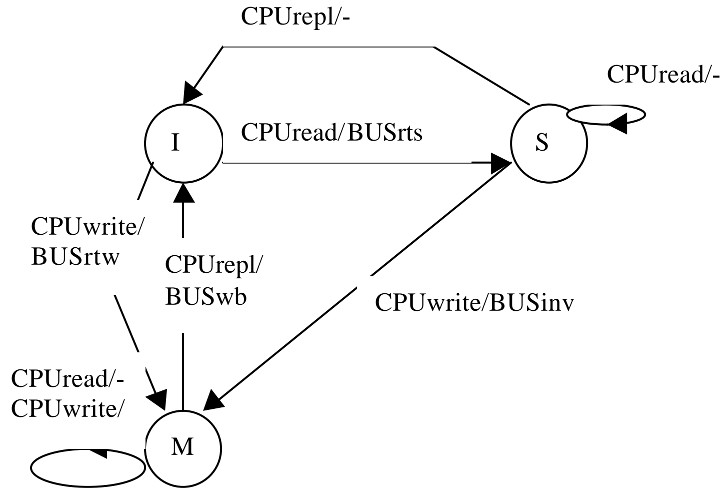
b) The fraction of instructions that produce a memory access (L2 miss)are:
MEM_LD = 0.2 x 0.1 x 0.1 = 0.002
MEM_ST = 0.1 x 0.2 x 0.25 = 0.005
Write-back = 0.2 x (MEM_LD + MEM_ST) = 0.0014
The average number of memory accesses per instruction is 0.002+0.005+0.0014 = 0.0084
The CPI is roughly 2, i.e., each instruction takes two cycles on average ➔ average number of mem accesses per cycle is 0.0084/2 = 0.0042
Every cycle takes 10 ns, every data transfer is 64 bytes ➔
➔ Mem bandwidth = 0.0042 x 64 bytes / (10×10^{-9} s) = 27MB/s

c) The fraction of instructions that produce a memory access (L2 miss)are:
MEM_LD = 0.2 x 0.1 x 0.1 = 0.002
MEM_ST = 0.1 (every write goes to memory)
Write-back = 0 (there are no write backs from a write-through cache)
The averge number of mem accesses per instruction is 0.002 + 0.1 = 0.102
If we assume no write buffers, the CPI will be much higher in this case since the CPU stalls for 100 cycles for each write, CPI_{ST} = fraction_stores * store_latency = 0.1 * 100 = 10 ➔ CPI = 11.38 i.e., each instruction takes eleven cycles on average ➔ average number of mem accesses per cycle is 0.102/11.4 = 0.0089
Every cycle takes 10 ns, every data transfer is 64 bytes➔
➔ Mem bandwidth =0.0089 x 64 bytes / (10×10^{-9} s)= 57MB/s (but the execution takes 5 times as longer…)

Problem 8, Cache Coherence (8p)

a) Draw a state transition diagram of a write invalidate MSI protocol showing state transitions forced by the CPU operations: CPUread, CPUwrite and CPUrepl (replacement). If there is no state change, show that as a loop-back to the same state. Each state transition should be clearly marked with "CPU input signel/BUS output signal" (if any). There are four different BUS transactions in this system

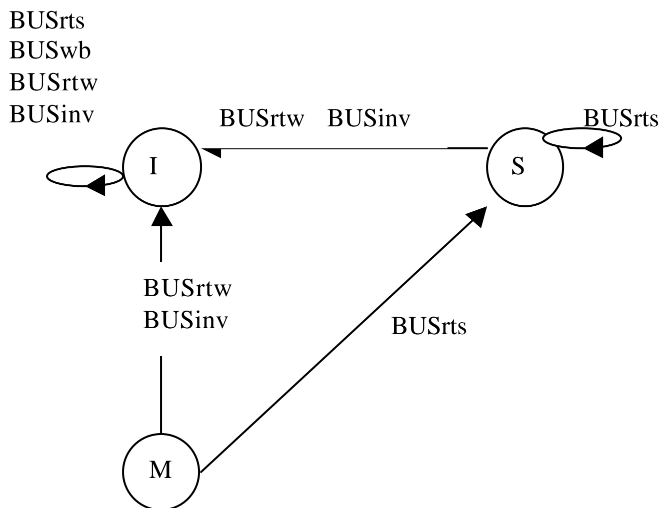
- BUSrts: ReadtoShare (reading the data with the intention to read it)
- BUSrtw, ReadToWrite (reading the data with the intention to modify it)
- BUSwb: Writing data back to memory
- BUSinv: Invalidating other caches copies (4p)



b) Show in a new drawing how external the BUS transactions

- BUSrts: ReadtoShare (reading the data with the intention to read it)
- BUSrtw, ReadToWrite (reading the data with the intention to modify it)
- BUSwb: Writing data back to memory
- BUSinv: Invalidating other caches copies

forces state changes in the MSI protocol (4p)



Problem 9, Synchronization (6p)

The CPUs of a shared-memory multiprocessor system can perform an atomic swap operation SWAP(A, R) that atomically switches the contents of memory location A and register R.

a) Show how pseudo C-code for the **lock** and **unlock** primitives with a single input variable ADDR. They should produce the shortest handover time (i.e., should allow for the shortest delay to get into the critical section) while producing a modest amount of bus traffic, assuming a lightly contended lock variable. (4p)

```
proc lock (addr)
        R := 1;
        SWAP(addr, R);              /SWAP!!/
        while R!= 0 {
           while mem[addr] !=0 {};  /spin on this line without producing any bus traffic/
           SWAP(addr, R) };         /we fell out of the spin so the lock appears to be free →SWAP/

proc unlock(addr)
        mem[addr] := 0;
```

b) What is a queue-based lock and what is its advantage over the spin lock you designed above? (2p)

It is a fair lock algorithm where each new contender will get into the critical region in the order they arrived at the lock. Each process spins on a variable that is modified by the process ahead of it in the queue. Key advantages: it is a "fair" algorithm (regardless of you physical location in the machine, everybody has the same chance to get the lock), the contenders are serviced in-order and it creates less traffic, since only the next in line will get its spin variable modified when the lock gets released.