Solutions to Example Exam (X-tenta) DARK2 991206

Problem 1, random questions (8p)

a) Show how the expression C := A + B is computed on a stack machine (2p)
PUSH A
PUSH B
ADD
POP C

b) What is key advantage of a stack machine over a load/store architecture?
Smaller code size, since the operands are implicit in the instruction format

c) What is an anti dependence and an output dependence?
Antidependence: A register (or variable) is read by one instruction and written to by a later instruction, creating a write-after-read dependence (WAR).
Output dependence: A register (or variable) is written by one instruction and also written to by a later instruction, creating a write-after-write dependence (WAW).

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

Problem 2, pipelining (8p)

a) What is a branch target buffer? (2p)
A cache indexed by an instruction address that stores the most recent jump target address taken by that instruction.

b) What improvement are introduced in a folding branch-target-buffer? (2p)
It is a branch-target buffer that contains the address of the most recent target, and also the first few instructions stored at that location. Thus, the instruction fetch stage can be bypassed.

c) What purpose does the scoreboard logic in CDC6600 have? (2p)
It checks for. And resolves, write-after-read (WAR) and read-after-write (WAR) dependencies dynamically at runtime. This allows for a more aggressive issuing rate for instructions.

d) What is the name of the improved algorithm used instead of scoreboarding in IBM360/91 (2p)
Tomasulo algorithm.

Problem 3, virtual caches (8p)

Most caches are so-called physical caches, rather than virtual caches, even though virtual caches could potentially have several advantages over physical caches

a) What is a virtual cache? (2p)
It is a data or instruction cache indexed by virtual addresses and with a virtual address tag.

b) What is its potential advantage over a physical caches? (2p)
It removed the TLB from the fast critical path of a load or store operation hitting in the cache.

c) What is the inherent problem of a virtual cache? (2p)
There is an aliasing problem in that different processes may use different virtual "names" when referring to the same physical data object. The risk is that the same data item is stored in the cache using different virtual addresses, which would create a cache-internal coherence problem.

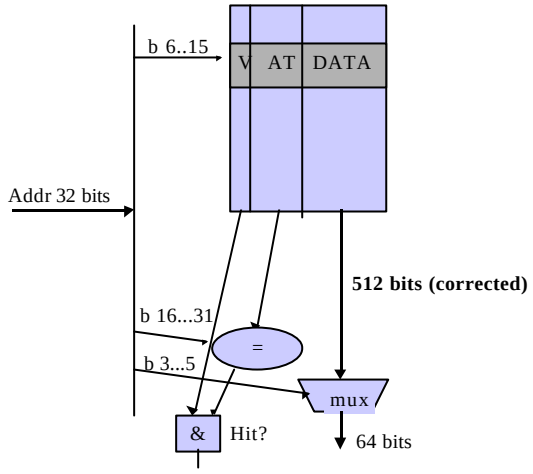
d) Describe one technique used to overcome that problem (2p)
Make the first-level cache smaller than the physical (and virtual) page size. That way, not more than one synonym can reside in the cache at once.

Problem 4, cache organization (8p)

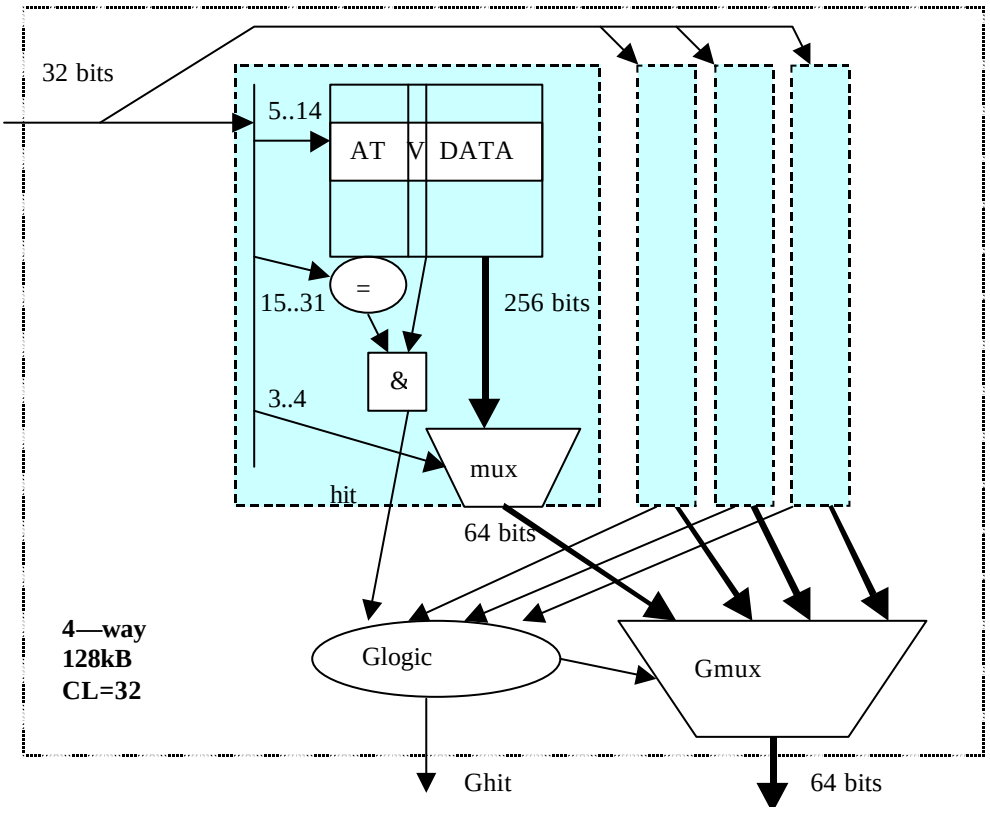
The following picture shows a direct-mapped cache for 64 byte large cache lines.

a) What is the cache size, i.e., what is its data capacity (2p)

The cache is indexed using bits 6..15, i.e., it takes 16 bits of addresses to identify data in the cache. The size is 2^{16} = 64kbyte.



b) Make your own drawing of one cache of twice that capacity, with 32 byte cache lines, and with a 4-way associative organization (4p)



The four identical blocks each produces two identical signals: hit and data, numbered from left to right: hit 0,1,2,3 and data 0,1,2,3. At most one of the hit signals will be asserted at any one time. The new Gmux selects is controlled by the four hit signals and selects the data corresponding to the block with its hit signal asserted. The Ghit signal is the or function of the four hit signals. (Note the new bit range on the index, mux select and comparator fields in each block)

c) What are conflict, compulsory, and capacity misses, and describe cache properties used to remove each category. (2p)

- **Capacity: The cache is not large enough for the data to remain in the cache the next time it is needed. → get a larger cache**
- **Conflict: The cache capacity is large enough for the data to remain in the cache the next time it is needed, but it was replace by data stired in its location. → Increase the cache associativity and/or improve the replacement algorithm.**
- **Compulsory: The data is touched for the first time → Increase the cache block (i.e., cache line) size and hope for spatial locality.**

Problem 5, loop scheduling (8p)

Consider the loop and the corresponding compiler-generated code

```
for (i=1; i<=1000; i = i+1)
    x[i] = x[i] + 10;
```

```
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
```

a) Write one-sentence comments for each line? (2p)

Line1: F0 and F1 is now x[i]
Line2: Add scalar constant stored in F2,F3 to F0,1 and store to F4,F5
Line3: Save result in x[i]
Line4: Decrement array pointer
Line5: If array pointer != 0, loop back to line 1

b) Show where the bubbles are in each iteration, assuming an architecture with one cycle branch-delay slot and a load delay of two cycles? (2p)

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

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

Each iteration takes 6 cycles

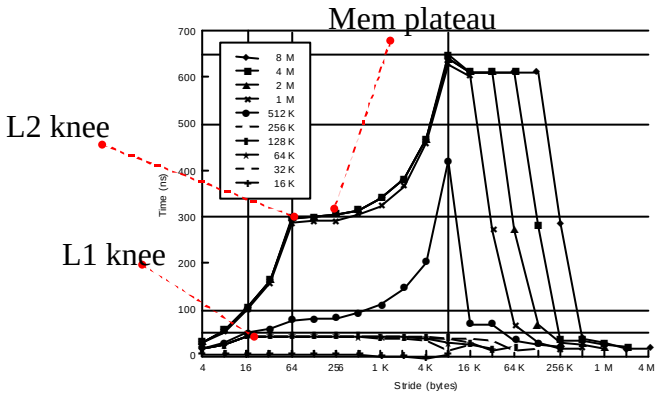
d) Show how loop unrolling could help to avoid all the stalls in this loop (2p)

```
loop:   LD F0, 0(R1)        ;line1
        LD F6, -8(R1)       ;line1'
        SUBI R1, R1, #16    ;line 4 + line 4'
        ADDD F4, F0, F2     ;line 2
        ADDD F8, F6, F2     ;line 2'
        SD 16(R1), F4       ;line 3
        BNEZ R1, loop       ;line 5
        SD 8(R1), F8        ;line 3'
```

Problem 6, microbenchmarks (8p)

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

```
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 */
```



a) What is the cache line sizes for the L1 and L2 caches? (2p)
L1: 16B as shown by the L1 knee
L2: 64B as shown by the L2 knee

b) What is the access time to the memory? (2p)
300ns (as show by the Memory plateau)

c) The curve corresponding to the array size of 512k has one point much higher than its other points, still it is much lower than the corresponding points for Array sizes > 512k. Describe what is happening at that point. (2p)
These accesses miss in the TLB and hit in the L2 cache. Smaller array sizes hit in the L2 and TLB and larger array sizes miss in both the TLB and the L2.

d) Estimate the upper and lower bound for the number of TLB entries in this architecture (2p)
Based on c) we can assume that the TLB reach is less than 512kB, but larger than 256kB. The page size is 8kB (that is the stride for which the 512k curve got in trouble) . Lower bound TLB entries is 256k/8k = 32, upper bound TLB entries is 512k/8k = 64.

Problem 7, Storage systems (8p)

a) A disk is marked with an average seek time of 8ms, it rotates at 7200RPM and has a data transfer rate of 5Mbyte/s. What is the average time for accessing a file of 8kbytes? (2p)

Rotation frequenzy: 7200/60 turns per second. → half a turn 0.5/120 sec → avg rotate latency = 4.2 ms
Transfer time: 8k x 10^3 / 5 x 10^6 = 1.6 ms
Access time = seek time + rotate time + transfer time = 8ms + 4.2ms + 1.6 ms = 13.8 ms

b) Describe how RAID3 techniques can be used to improve the availability of data stored in a disk array (2p)

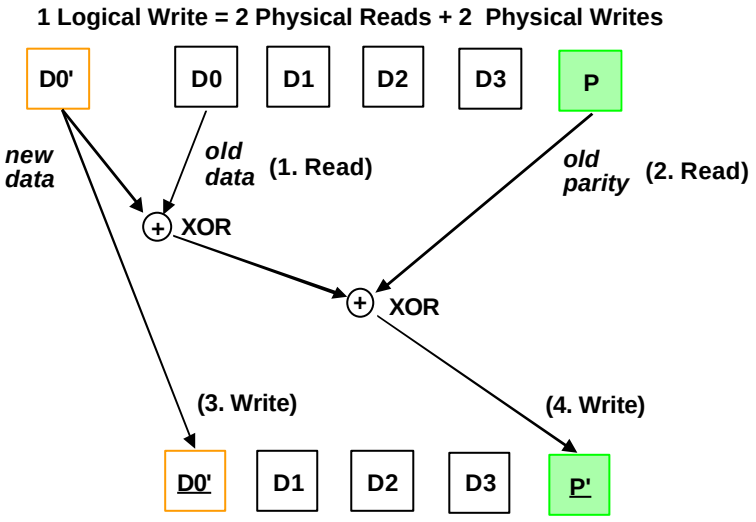
The "words" written are striped across many disks, e.g., 8 disks, and one extra disk stores the bit-wise parity of each word. If one of the disk is removed (including the parity disk) its contents can easily be calculated and a new disk can be initialized to replace the faulty one.

c) Describe how RAID5 techniques can be used to improve the "small writes" to a disk array, compared with a RAID 3 system.

All the disks are utilized as parity disks in a cyclic manner and the job of storing the parity is evenly distributed between the disks. Writes of small data item only need to access the disk where the data is stored and the parity disk.

d) Describe the disk reads, disk writes and bit-logical operations for small writes in a RAID 5 system (2p)

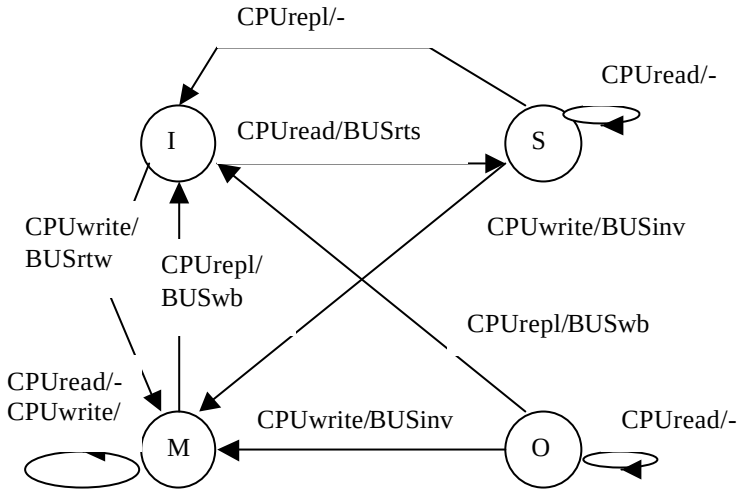
- **The target location of the target disk and the target parity is read, e.g., one byte from each.**
- **The new parity byte is calculated as the bit-wise XOR of the new byte, the old byte and the old parity.**
- **The new data and the new parity is stored to the target disk and the parity disk.**



Problem 8, Cache Coherence (12p)

a) Draw a state transition diagram of a write invalidate MOSI 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

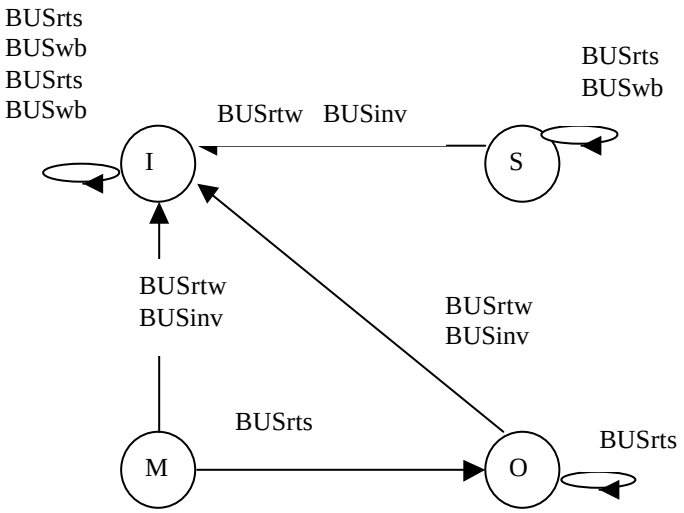
BUSrts: ReadtoShare (reading the data with the intention to read it)
BUSrtw, ReadToWrite (reading the data with the intention to modify it)
BUSwb: Writing data back to memory
BUSinv: Invalidating other caches copies (4p)



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 MOSI protocol (4p)



C) Compulsory, Capacity and Conflict misses have got a new companion miss category (starting on the same letter "C") in a shared memory system. What is its name and how does it occur? (2p)

Coherence misses occur when an access to data, that otherwise would have remained in a cache, causes a miss because it got invalidated by the actions caused by a foreign write request.

D) What is false sharing? (2p)

Two CPUs that do not share data still fight over a cache line because they both access data in the same cache line and at least one of them writes to the data.

Problem 9, Synchronization (8p)

The CPUs of a shared-memory multiprocessor system can perform an atomic swap operation SWAP(R, M) 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 smallest possible amount of BUS traffic, assuming a heavily contended lock variable. (4p)

```
proc lock (addr)
        R := 1;
        while (mem[addr] =! 0) {};      /test first if the lock is contended/
        SWAP(addr, R)                           /this operation is really called SWAP!!/
        while R!= 0 {
          while mem[addr] !=0 {};       /spin on this line/
          SWAP(addr, R) };

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

b) What is a barrier synchronization? (2p)

A blocking synchronization primitive that causes the process (or thread) to wait until all the companion processes (threads) also have reached the barrier.

c) Write the pseudo C-code for a barrier using the lock/unlock primitives you (hopefully) completed under item a. The algorithm should minimize BUS traffic (2p)

Global variables:

```
barrier_cnt = barrier_release = barrier_lock=0;
N = /number of threads /
```

Local variables:

```
local_sense=0;

proc barrier(barrier_cnt, barrier_lock, barrier_release, N)
        local_sense := !local_sense
        lock(barrier_lock)
        barrier_cnt++;
        if (barrier_cnt ==N) {          /Is this the last thread?/
          barrier_cnt := 0;
          barrier_release := local_sense;
          unlock(barrier_lock);}
        else {
           unlock(barrier_lock)
           while(barrier_release != local_sense)};
```