Computer Science 605.611

Final Exam

1. A computer system contains a byte addressable memory system with 8 separate memory modules. Each memory module contains 134217728 cells. Recall that a zero-based numbering scheme is used for memory modules, cells and bytes.

a) (5) If the memory employs big endian storage order, high order interleaving and 32-bit memory cells, what is the 32-bit memory address of cell 1048578 within module 3? Express your answer in hex.

To determine the answer to this we need to look at a few pieces of info. To start we know that to find the address it is Module number (3 bits) then Cell Number (29 bits). The module number only needs 3 bits as that can represent modules 0-7 in binary. Then we need module 3 which is 11 in binary and 3 in hex. And the cell number is 0x100002 in Hex and 100000000000000000010 in binary. We need to extend the front part of the cell number to 29 bits so 00000000100000000000000000010 is our second part, then adding 011 to the front leaves us with 01100000000100000000000000000010 which is **0x60100002**

b) (3) What 32-bit address would you specify as your answer in part a) if the only change is to use little-endian storage order instead of big-endian storage order? Express your answer in hex.

The main difference here is that we need to reverse the order of the previous question. So 0x60100002 can be split into 60 10 00 02, reversing that order we get **0x02001060**

c) (5) Suppose that a different big endian memory system uses low order interleaving and contains 16 modules each of which contains 268435456 eight-bit cells. What is the 32-bit memory address of cell 1048577 within module 7 for this system? Express your answer in hex.

For low order interleaving we swap the direction of our address calculation, cell number then module number. The module number is 7 or 0111 (needs 4 binary digits since there are 16 modules), and the cell number is 100000000000000000001. Extend the cell number to 28 bits is 0000000100000000000000000001. Then we tack on the module number

0000000100000000000000000001 0111 which is **0x01000017**

2. Consider three different MIPS systems each of which runs at the same 200 MHz clock rate and each system employs one or more 5-stage instruction pipelines. In cycle 1 all three systems start fetching and executing a stream of instructions. One of the systems is a scalar system, another is a degree-4 super pipelined system and the third is a degree-4 superscalar system. Superscalar and superpipelined systems were defined and described in module 8. Assume the same instruction sequence containing only independent R-type instructions is executed by each of the three systems.

a) (3) By the end of clock cycle 10, how many of the instructions can the scalar system complete?

**There would be in total 2 instructions** that have completed. The first would complete at cycle 5 and the second would enter at cycle 6 completing at cycle 10.

b) (3) By the end of clock cycle 10, how many of the instructions can the degree-4 superpipelined system complete?

To determine this lets think of how this works. We know that a degree 4 super pipelined system can handle 4 instructions in each stage. In Cycle 1 4 instructions enter the first stage, in Cycle 2 4 new instructions are added to the first stage as the first 4 move to the next stage. Cycle 3-5 this cycle continues and by 5 the first instructions complete. Cycle 6-10 continues this and the next sets continue to complete. So cycles 5-10 complete 4 instructions so, 4 \* 6 = **24 instructions total**.

c) (3) By the end of clock cycle 10, how many of the instructions can the degree-4 superscalar system complete?

This system can handle 4 instructions per cycle. So in an ideal scenario, we would complete **40 instructions**.

3. a) (5) The ARM instruction BLX R3 calls the function whose memory address is in register R3 and saves the function return address in the ARM linkage register. Is there a single MIPS true-op instruction that has the same effect on the MIPS processor using MIPS register $3? If so, show the equivalent MIPS true-op instruction. If not, explain why this can’t be done on the MIPS processor.

First lets look at what BLX R3 actually does, it is branch and link meaning it jumps to address at R3 and then saves that return address in the link register. There is not a single MIPS true-op instruction that has the same affect as BLX R3. The main reason is that MIPS doesn’t have a dedicated linking register, the jal instruction uses an immediate value instead of a register but does accomplish a similar thing to the BLX instruction. There would however be a way to do this with more than one instruction.

b) (3) The ARM instruction MOVT R2, #253 copies the constant 253 into the leftmost 16 bits of register R2 without changing the rightmost 16 bits in the register.

Show how the same effect can be produced for MIPS register $2 using no more than three MIPS true-op instructions. Your solution should use at most one MIPS register other than the result register $2.

The same effects would be accomplished with these 3 true op instructions:

**Lui $t0, 253**

**Andi $2, $2, 0xFFFF**

**Or $2, $2, $t0**

4. Suppose that you had a choice of using one of three different branch prediction schemes:

I predict that each branch is never taken

II predict that each branch is always taken

III use a dynamic predictor that correctly predicts 90% of the time

Scheme I has a one-cycle penalty for a misprediction.

Scheme II has a two-cycle penalty for a misprediction.

Scheme III has a three-cycle penalty for a misprediction.

What is the total number of penalty cycles incurred for each of these schemes by a conditional branch instruction that executes 12000 times and actually branches 60% of the time?

(3) total penalty cycles for scheme I = \_\_**7200**\_\_\_

Branch is taken 69% of the time or is wrong 60% of the time. There is a misprediction penalty of 1 cycle. Penalty = num of misprediction \* penalty of misprediction. 7200 \* 1 = 7200

(3) total penalty cycles for scheme II = \_\_\_**9600**\_\_\_

40% chance is wrong, 2 cycle misprediction penalty. 4800 \* 2 = 9600

(3) total penalty cycles for scheme III = \_\_\_**3600**\_\_\_

10% wrong chance (1200 times). 3 cycle penalty. 1200 \* 3 = 3600 cycles

5. “Access pattern” is defined as the order in which the elements within a data structure are referenced. The access pattern can have a major impact on performance when using a memory system that employs caches.

Consider a system that has a direct-mapped D-cache (data cache) containing 4096 lines. Each line is 256 bytes in size. The D-cache is initially empty. The matrix X with 256 rows and 256 columns is to be processed. The matrix resides in memory starting at address 0x10040000. Assume that each matrix element is 32 bits. Also assume that registers are used for the array indices and that no code optimizations are performed.

a) (3) If the matrix is stored in row major order, what is the miss ratio for the data cache based on the matrix accesses made by the following code?

for (i=0; i<256; i=i+2;) {

for {j=0; j<256; j=j+2}

X[i,j] = X[i,j] + X[i+1,j];

}

}

So cache line size is 256 bytes, elements per cache line is 256/4 = 64 elements. Matrix size is 256x256. Based on the code, i and j are incremented in 2 so we are accessing every other element. Thus we have a miss per row calculation of 128/64 = 2 cache lines accessed per row, or 2 misses per row. We access a total of 256/2 rows or 128 rows. 128 \* 2 = 256 misses. There are 256 \* 128 = 32,768 iterations \* 2 = 65,536 accesses. Our miss ratio = total misses / total accesses = 256 /65536 = 0.0039 or **0.39% miss ratio**.

b) (3) If the matrix is stored instead in column major order, what is the hit ratio for the cache based on the matrix accesses made by the following code?

for (j=0; j<256; j=j+2;) {

for {i=0; i<256; i=i+2}

X[i,j] = X[i,j] + X[i+1,j];

}

}

There are 256 column, 127 hits per column (first access is a miss or itd be 128). Total hits are 127 \* 128. We have the same amount of accesses 65536. So (127 \* 128)/65536 = **24.80% hit ratio**.

6. A processor has a 4294967296-byte data memory and a single data cache (cache A) that contains 131072 lines. Cache A is 1-way set associative with a 256-byte line size. When the MIPS load byte instruction lb $8,0($4) is executed, CPU register $4 contains the address of the final byte within memory block number 2928610.

a) (3) Show the 8-digit hex address contained in register $4.

We first need to convert 2928610 to binary which is 10110010101111111100010, then to hex so 002CAFE2. $4 contains **0x002CAFE2**.

b) (3) Suppose that register $6 contains some unspecified memory address. Write down a sequence of MIPS true-op instructions that compute the cache line number to which the address in register $6 maps. The line number computed should be left in register $3. Your instruction sequence should contain no more than two MIPS true-op instructions and should work for any memory address contained in $6 and should not change $6.

The two instructions that would achieve this are:  
**srl $3, $6, 8**

**and $3, $3, 0x1FFFF**

c) (3) Consider a different cache that we can call cache B. Cache B contains a total of 131072 lines and is organized as a 4-way set associative cache. Each cache line is 256 bytes in size. To what set within this 4-way set associative cache does the address 0xC306DEFC map?

We know that each cache line is 256 bytes, so 8 bits for block offset. There are 32768 sets in cache which, so set index requires 15 bits. We know the address is 32 bits, so we subtract 15 bits and 8 bits leaving us 9 bits for the tag. Converting 0xC306DEFC to binary we get 110000110 000011011011110 11111100, 000011011011110is the set index bits converted to decimal would be **1758**

d) (3) Assuming that the reference to address 0xC306DEFC causes a cache hit, use hex to show the tag for the cache line in which the hit occurs within Cache B.

Since the tag is the first 9 bits which are 110000110, we know the hex would need to be **0x186**

7. A certain program makes 176,000,840 data memory accesses (reads or writes). Each memory access takes 128 ns. To improve performance, a proposal is made to use a data cache for the system with an access time of 10 ns. The cache has a hit ratio of 90%.

a) (3) What is the total number of nano-seconds required to perform all 176,000,840 data accesses if the cache operates in look-through mode?

For look through mode we can determine that time to hit is 10 ns, but a time to miss is 10 + 128ns. We then know that (.10 \* total accesses) \* (cache access time + main memory access time), (.10 \* 176,000,840) \* (10 + 128) = 2,428,811,592 ns. (.90 \* total accesses) \* (cache access time + main memory access time) = (.9 \* 176,000,840) \* (10 + 128) = 1,584,007,540. So 2,428,811,592 ns + 1,584,007,540ns = **4,012,819,152 ns**

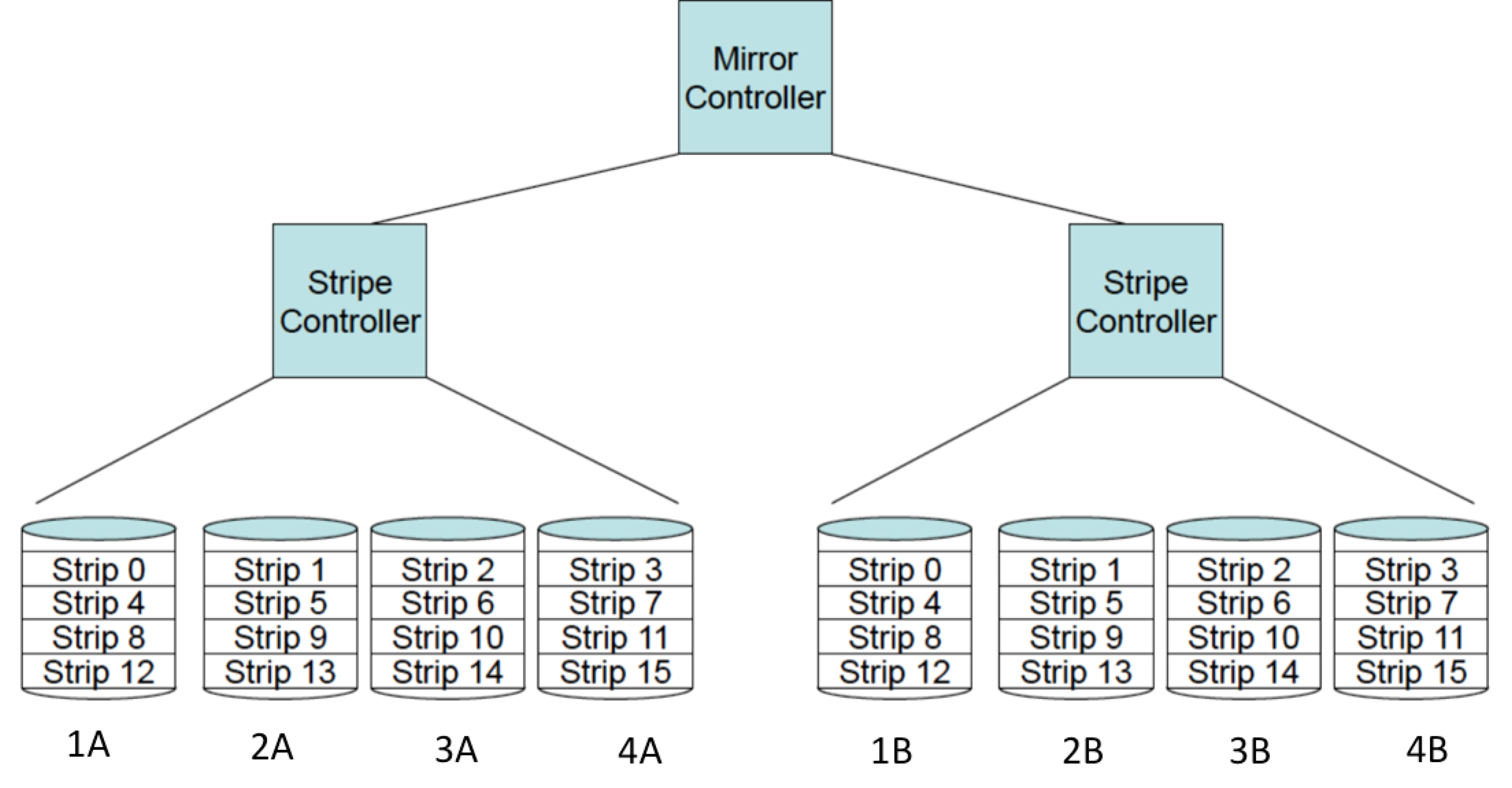
b) (3) What is the total number of nano-seconds required to perform all 176,000,840 data accesses if instead the cache operates in look-aside mode?

We do a similar

This is similar to the previous question but we don’t need to do the extra 10ns + 128ns instead it would just be this. (.9 \* total access) \* (cache access time) + (.1 \* total accesses) \* (main memory access time) = (.90 \* 176,000,840) \* (10ns) + (.10 \* 176,000,840) \* (128ns) = **3,836,818,312 ns**

8. (3) The diagram shown below represents a RAID0-1 system.

The mirror controller makes the system appear to be two separate RAID0 systems: A and its mirror image B (each of which contains 4 disks). Suppose that disk 2A fails and can’t be used. The system continues to operate using just the mirror image RAID0 system B. Some time later, a new disk is installed to replace disk 2A. Once the replacement is made, list all disks that must be read and all disks that must be written to restore the system to full operation.



The disks that need to be read is only Disk 2B, this is because disk 2B contains the mirrored images from the failed 2A.

The disks that need to be written are just 2A as the other disks continued to transfer normally just read data from the B side instead of the A side.  
This is because the other disks did not fail and continued to operate normally

9. Recall that our MIPS 5-stage scalar pipelined system includes 32 general purpose CPU registers and employs delayed branching. Suppose that the system is modified to take advantage of hyperthreading and includes control logic to alternately fetch an instruction from three different threads to enter the pipeline.

a) (3) What is the minimum number of general purpose CPU registers required for the modified system?

In the original configuration there were 32 general purpose CPU registers. With Hyperthreading the CPU can execute multiple threads (3). Each thread requires its own set of registers, thus we would need 32 \* 3 = **96 registers as a minimum**.

b) (3) Explain why each of the following units IS or IS NOT required with this modified system:

a forwarding unit

This is still required, this is because it is responsible for forwarding crucial data so there are not as many data hazards/without it there could be incorrect or stale data that is being used.

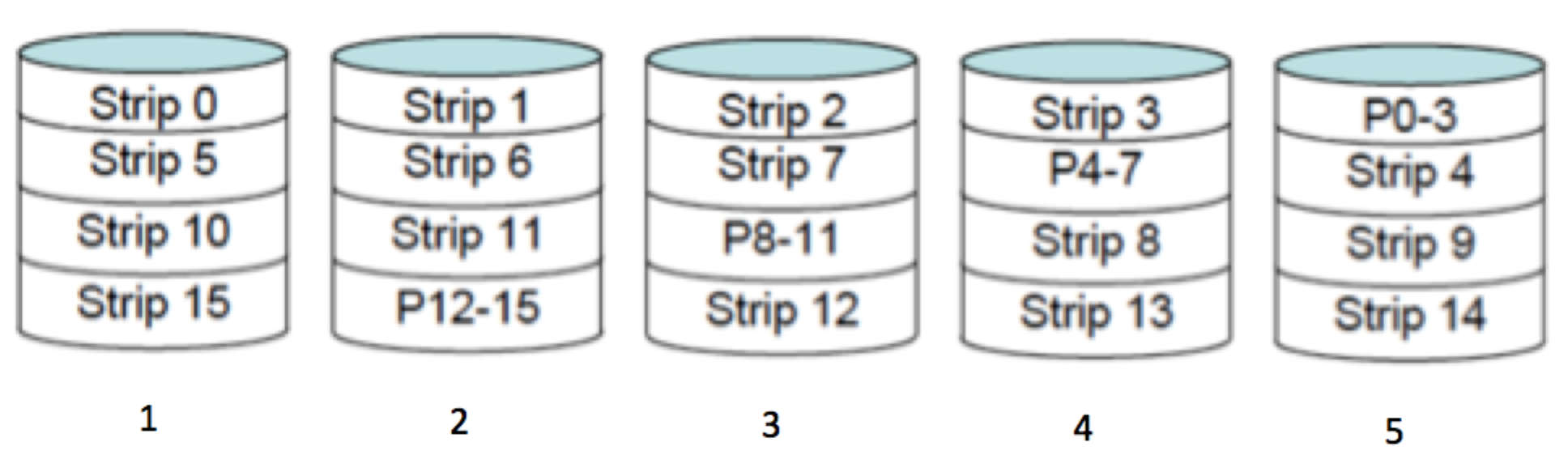
a hazard detection unit

This is required. Its role may slightly differ. This is still required since hazards may occur across threads due to shared resources. Without this too many pieces of data could have issues.

a branch prediction unit

Still necessary but may need modification. This is because even with delayed branching, we need to be able to predict. It will enable the multi threaded system to run much more efficiently than without it.

10. Consider the RAID5 system depicted in the diagram below:



a) (3) Each strip or block contains data and each parity block is computed as the XOR of the strips within that stripe (or row). Assume that disk 3 fails and is later replaced with a new disk. Once disk 3 is replaced, it must be repopulated by writing all of the required data and parity. Show how Strip 2 on the new disk 3 can be computed.

Strip2 =\_\_P0-3 XOR strip 0 XOR strip 1 XOR strip 3\_\_

b) (3) If all five disks are operating properly, can strip6 and strip14 be written in parallel on this system? Explain why or why not.

No, looking at this question it would be easy to say yes since 14 and 6 are on separate disks. However, the system also needs to write the parity of the strips. Strip14’s parity strip is on disk 2, the same disk that strip6 is on. It is possible that there wouldn’t be a conflict as long as strip 6 completes its information write before 14, but if 14 finished first it will cause errors.

11. Assume that the 32-bit variables X, Y and Z correspond to memory addresses 0x10040004, 0x2004008 and 0x300400C, respectively. Show a sequence of MIPS assembly language instructions containing only true-op instructions that implements the high level language statement Z = X / Y ; for the following two cases:

a) (4) X, Y and Z are signed integer variables all of which are non-zero.

lw $t0, 0x10040004

lw $t1, 0x2004008

div $t0, $t1

mflo $t2

sw $t2, 0x300400C

b) (5) X, Y and Z are single-precision floating point variables all of which are non-zero..

lwc1 $f0, 0x10040004

lwc1 $f1, 0x2004008

div.s $f2, $f0, $f1

swc1 $f2, 0x300400C

12. (5) Assume that our MIPS scalar processor uses big-endian memory storage order and shares a data memory with a processor (such as an Intel Pentium processor) that employs little-endian memory order. The Pentium processor operating in little-endian memory mode writes the IEEE-754 single precision 32-bit floating point number representation of the decimal value -31.25 into the word at address 0x100400A0 within the shared data memory.

The MIPS processor operating in big-endian memory mode then executes the following instructions:

lui $2, 4100 ; put 0x10040000 into $2

lh $4, 162($2) ; read the first number into $4

What two’s complement value is represented by the 32-bit pattern placed into register $4 by these MIPS instructions? Express your answer as a decimal number.

Well lets start by seeing the values in each step. For lui it loads the upper 16 bits of the address into register $2, 0x1004000. Then load half word reads the 16 bits from address (0x10040000 + 162 = 0x100400A2) and stores it in $4. We also can figure out that in IEEE-754 32 bit fpn in hex is C1FA0000. Since C1FA0000 is stored at 100400A0 and lh $4, 162($2) retrieves the first half of that value we know that FAC1 is stored in $4 (due to the endian storages). The twos compliment decimal representation of that is **-1343**.

13. (5) Each of the 4 processors in a shared memory multi-processor system is rated at 800 MIPS. A program contains a purely sequential part that accounts for 42% of the program’s execution time on a single processor. The remaining code can be partitioned into three independent parts (A, B, and C). Running on a single processor, part A accounts for 20% of the program’s execution time, part B accounts for 18% of the execution time, and part C accounts for 20% of the execution time.

All four processors can be used to execute the program, but the sequential part must be completed before the remaining independent parts (A, B or C) can run in parallel.

Compared to running on the single processor system, what is the speedup ratio provided by running the program on the 4-processor system? Express your answer to 3 decimal places (d.ddd).

We know that speedup can be calculated by doing 1/(S + P/N), S = fraction of sequential execution time, P is fraction of parallelized time, N is number of processors. So S is 42%, P is 100-42 = 58%, and there are 4 processors. Plugging all of this in we get 1/(.42 + .58/4) = **~1.770**.

14. (3) On a different SMP 4-processor system, each of the four cores has a separate L1 cache, but the 4 processors share a single L2 cache. This system executes a different program that contains 4 independent parts which can execute in parallel. One option for improving the performance of the system is to provide each core with its own L2 cache so that the system contains four separate L2 caches instead of a single shared L2 cache. Each of the four L2 caches has the same size and organization as the original shared L2 cache. Can a speedup greater than 4 be achieved by making this improvement? Explain your answer.

This is a tricky question to get a true answer to without having a little more information. Lets start by determining what will help/hinder this speedup. If the workload is fully parallelizable then it helps the speedup. If the L2 caches are used optimally that can help. The system will need negligible overheads, and the memory and bus architecture need to support fast and concurrent access. If we assume a perfect scenario than the answer is yes, a speedup greater than 4 is possible. However, in a normal scenario in the real world it is highly unlikely that a speedup greater than 4 would ever happen.