Name Solution

Computer Architecture
EE 4720
Final Examination

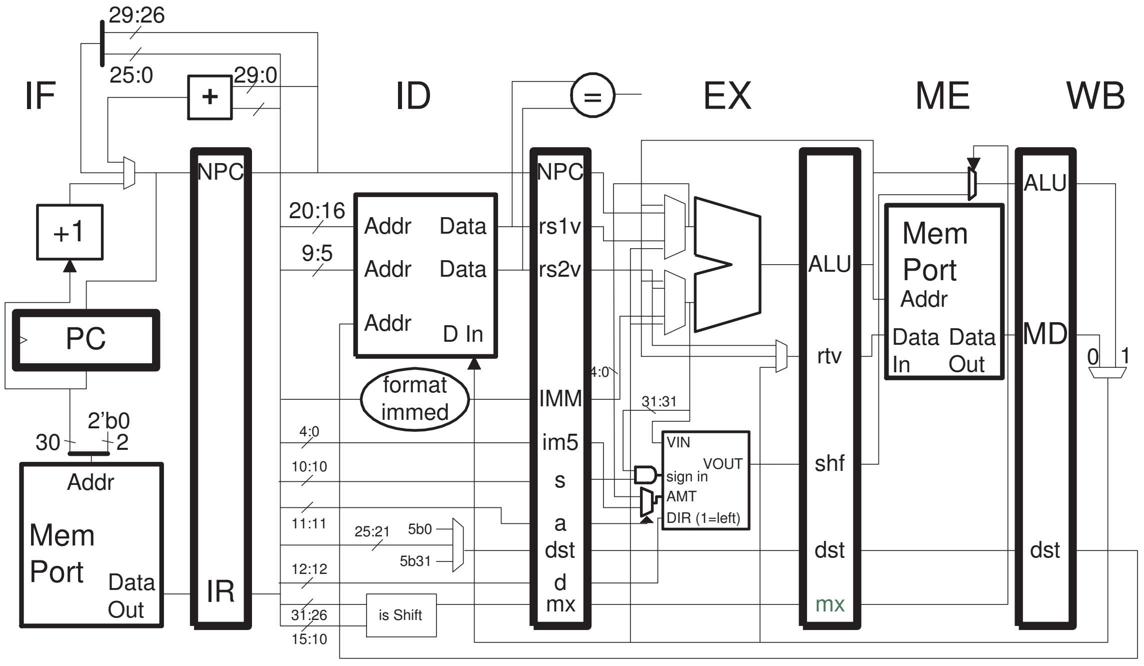
6 December 2010, 7:30–9:30 CST

Problem 1		(10 pts)
Problem 2		(14 pts)
Problem 3		(14 pts)
Problem 4		(14 pts)
Problem 5		(15 pts)
Problem 6		(15 pts)
Problem 7		(18 pts)
Exam Total		(100 pts)

Alias On File

Good Luck!

Problem 1: (10 pts) The diagram below shows a 5-stage pipeline that looks alot like our familiar MIPS implementation but it's actually an implementation of ISA X. (The diagram is based on the solution to Homework 3, in which a shift unit was added to MIPS.)



(a) ISA X instruction format T encodes the shift instructions and others, it is the equivalent of format R in MIPS. Based on the diagram above show the encoding for ISA X format T.

☑ Format T encoding, including bit positions and field names.

Solution appears below. The source registers are named `rs1` and `rs2`, those names are taken from the ID/EX pipeline latches, the bit positions at the input to the register file provide their place in the instruction encoding. The input to the $\boxed{\text{is Shift}}$ unit provide the location of the opcode fields. Through its connection to the shifter one finds that bits `4:0` are the equivalent of the MIPS `sa` field. The mux at the input to the `dst` pipeline latch provides the destination register field bits, `25:21`.

Encoding:

Opc	rd	rs1	opT	rs2	im5
31 26	25 21	20 16	15 10	9 5	4 0

(b) Consider the shift instructions `sll`, `sllv`, `srl`, `srlv`, `sra`, and `srav`. Suppose that the encoding of one of these instructions is zero (meaning that every field value is zero). Show the opcode field value(s) for each of these instructions based on the diagram above. *Hint: The control signal for each top mux input is 0, etc.*

☑ Opcode field value(s) for: `sll`, `sllv`, `srl`, `srlv`, `sra`, and `srav`.

From inspection of the diagram we see that bit 10 determines whether the shift is arithmetic (signed) (bit 10 is 1) or logical (unsigned) (bit 10 is 0). From inspection of the diagram we see that bit 11 determines whether the shift amount is obtained from the `rs1` register (possibly bypassed) or if bit 11 is 1 whether the shift amount is obtained from the `im5` field. From inspection of the diagram we see that bit 12 indicates the direction, with 1 for a left shift. Since one of the shift instructions has a zero opcode, the `opc` field must be zeros and bits `15:13` of `opT` must be zero. Putting these bits together we get the `opT` values shown below:

```
Solution:
   sll    sllv    srl    srlv    sra   srav
   110    100     010    000     011    001    <- Bits 12, 11, and 10
000110 000100  000010 000000  000011 000001    <- Full opT value.
```

(*c*) Explain why the implementation of instructions such as `sw r1,2(r3)` and `beq r1, r2 TARG` would be less elegant for ISA X than for MIPS. *Hint: It has something to do with registers.*

[✓] `sw` and `beq` less elegant because...

Consider instructions `lw r1,2(r3)` and `sw r1,2(r3)`. In MIPS these can use the same format because the `rt` field can be either a destination (as its used for `lw`) or as a source (as its used for `sw`). But in ISA X, based upon the diagram above, there is only one destination field. So the `lw` encoding might be in a format where the `rs2` field is omitted (and so bits `9:5` can be part of the immediate) while the `sw` instruction would be in a format where `rd` was omitted. As a result the `format immed` unit needs to pick different sets of bits based on format.

Problem 2: (14 pts) Answer the questions below.

(*a*) Complete the execution diagram for the MIPS code below on a two-way superscalar statically scheduled implementation of the type described in class.

```
# SOLUTION
# Cycle          0  1  2  3  4  5  6  7  8
add r1, r2, r3   IF ID EX ME WB
add r4, r5, r6   IF ID EX ME WB
add r7, r4, r8      IF ID EX ME WB
lw  r9, 0(r7)       IF ID -> EX ME WB
addi r1, r9, 3         IF -> ID -> EX ME WB
xor r10, r11, r12      IF -> ID -> EX ME WB
# Cycle          0  1  2  3  4  5  6  7  8
```

☑ Complete the diagram above.

The solution appears above. The `lw` stalls due to a dependence on the preceding `add`. Notice that the stall affects all following instructions. That is, though in cycle 3 the `lw` is stalled in $\mathtt{ID}^1$ the $\mathtt{ID}^0$ stage is empty. It would be possible to design the pipeline so that the `addi` could move into $\mathtt{ID}^0$ in cycle 3, however that would greatly complicate control logic since instructions would be out of order. (The instruction in $\mathtt{ID}^0$ would come after the one in $\mathtt{ID}^1$ in program order, whereas usually the instruction in $\mathtt{ID}^0$ is before the one in $\mathtt{ID}^1$.) So to keep instructions in order the `addi` must stall.

(*b*) Show the execution of the code below on an 8-way superscalar statically scheduled processor of the type described in class. Branches are not predicted. Find the CPI for a large number of iterations.

```
# SOLUTION
LOOP:  # Cyc     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
and r1, r1, r5   IF ID EX ME WB
add r3, r3, r1   IF ID -> EX ME WB
lw r1, 0(r2)     IF ID -> EX ME WB
bne r2, r4 LOOP  IF ID -> EX ME WB
addi r2, r2, 4   IF ID -> EX ME WB
or               IF ID -> x
or               IF ID -> x
or               IF ID -> x
xor                 IF -> x
xor                 IF -> x # Six more instructions squashed.
LOOP:  # Cyc     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
and r1, r1, r5            IF ID EX ME WB
add r3, r3, r1            IF ID -> EX ME WB
lw r1, 0(r2)              IF ID -> EX ME WB
bne r2, r4 LOOP           IF ID -> EX ME WB
addi r2, r2, 4            IF ID -> EX ME WB
or                        IF ID -> x
or                        IF ID -> x
or                        IF ID -> x
xor                          IF -> x
xor                          IF -> x # Six more instructions squashed.
LOOP:  # Cyc     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
and r1, r1, r5                     IF ID EX ME WB
```

☑ Complete diagram above for enough iterations to determine CPI.

The solution appears above.

☑ Find the CPI for a large number of iterations.

The third iteration starts, in cycle 6, with the pipeline in the same state as the start of the second iteration, in cycle 3. (The pipeline state is `and` in `ME`, `add`, `lw`, `bne`, and `addi` are all in `EX`, etc.) Therefore the time for the second iteration, $6-3=3$ cycles will be the same as the time of the third. The CPI is then $\boxed{\frac{6-3}{5}\text{ CPI}}$.

(*c*) Complete the execution diagram for the MIPS code below on a two-way superscalar dynamically scheduled implementation of the type described in class. The execution of the first instruction is shown. The `lw` instruction uses stages `EA` and `ME` in place of `EX`.

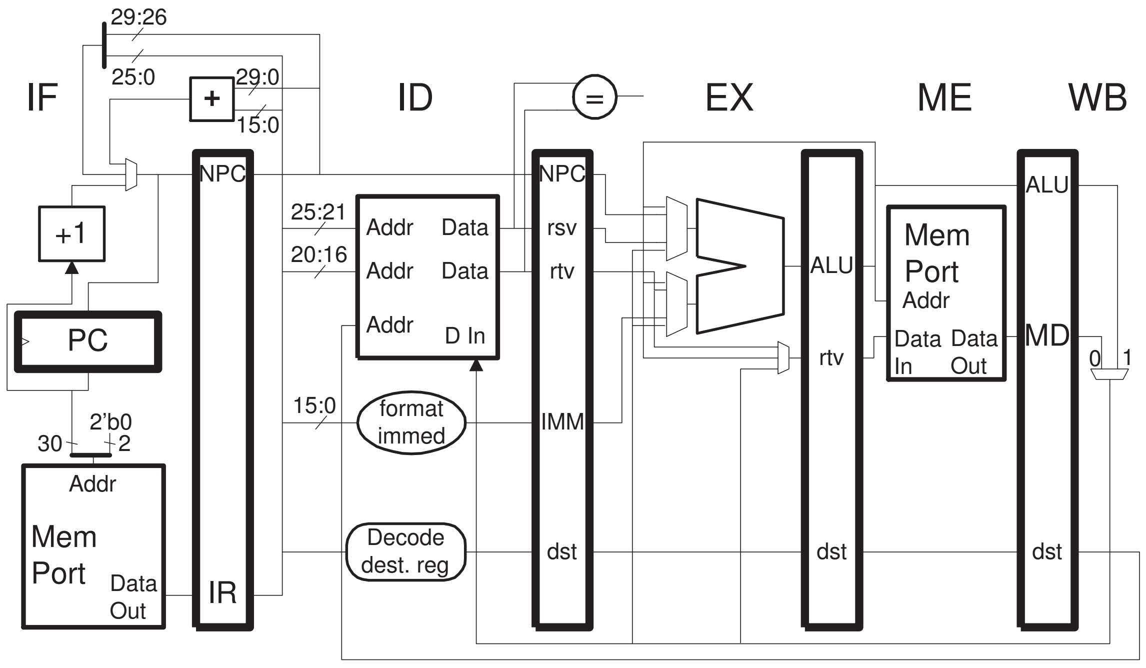
```
# Solution
LOOP:  # Cyc     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
add r1, r2, r3   IF ID Q  RR EX WB C
add r4, r5, r6   IF ID Q  RR EX WB C
add r7, r4, r8      IF ID Q  RR EX WB C
lw  r9, 0(r7)       IF ID Q     RR EA ME WB C
addi r1, r9, 3         IF ID Q        RR EX WB C
xor r10, r11, r12      IF ID Q  RR EX WB       C
LOOP:  # Cyc     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18 19
```

☑ Complete diagram above.

Solution appears above. Notice that the dependence chain from the `add r4` through the `lw` and `addi` does not affect when the `xor` executes, though it does delay its commit.

Some important things to remember: Commit must occur in program order. Since the processor is 2-way superscalar at most two instructions per cycle can commit (by default for classroom examples).

Problem 3: (14 pts) A deeply pipelined MIPS implementation can be constructed by dividing some stages of our familiar 5-stage statically scheduled scalar implementation (shown below) into two or more parts. In this problem the technique is applied to construct several 8-stage implementations. All have just one `ID` and `WB` stage, and in all implementations **it takes** 4.4 ns **for an instruction to pass through all 8 stages**, from the beginning of `IF1` to end of `WB1`. The stages are divided without changing what they do. For example, if an "original" MIPS stage, say `EX`, is divided into multiple stages, say `EX1 EX2 ... EXn`, then all values from `ID` and bypass paths are needed when `EX1` starts, and values reach `ME` in the cycle after `EXn`. Our familiar 5-stage implementation is shown below for reference:



(*a*) Consider the 8-stage *baseline* implementation below (indicated by the stage labels). What is the execution rate of a friendly program (one written to maximize performance) on the implementation, in units of instructions per second? The answer can be given as a formula of constants (as opposed to putting down just x as an answer).

`Baseline Implementation: IF1 IF2 ID1 EX1 EX2 ME1 ME2 WB1`

☑ Execution rate in instructions per **second**.

If it takes 4.4 ns to pass through 8 stages then the clock frequency must be $\frac{8}{4.4\,\text{ns}} = 1.82\,\text{GHz}$. So the execution rate is 1.82 billion insn per second.

(*b*) Call a program *favorable* for an implementation if it runs faster on the implementation than on the baseline (repeated below). If it runs slower, call it *unfavorable*. For each implementation below write a two- or three-instruction favorable program and a two- or three-instruction unfavorable program. Also provide a concise but clear explanation of what it is about the program and implementation that makes it favorable or unfavorable. If a program is favorable on one implementation and unfavorable on another write it once, but provide an explanation for each. *Hint: The three implementations differ in how they are affected by certain dependencies.*

```
Baseline Impl.:     IF1 IF2 ID1 EX1 EX2 ME1 ME2 WB1
Implementation 1:   IF1 IF2 IF3 ID1 EX1 ME1 ME2 WB1
```

☑ Favorable program. ☑ Explanation.

☑ Unfavorable program. ☑ Explanation.

In Implementation 1 there is one more `IF` stage than the baseline but one less `EX` stage. A favorable program would have a close dependence from ALU output to ALU input. An unfavorable program would have a dependence from anywhere to the `IF` input, which would be a control transfer, including a branch. From the pipeline diagram below for the unfavorable program one can see that because of the extra IF stage the target is fetched one cycle later and so one more instruction is squashed than would be in the baseline. Without a branch predictor these squashes would occur for every taken branch, with a branch predictor they would only happen when the branch is mispredicted, but that's still worse than the Baseline.

```
# SOLUTION
# Favorable for 1:  Second instruction has true dep with first, both are ALU.
 add r1, r2, r3
 sub r4, r1, r5

# Unfavorable for 1: Taken branch
 beq r1, r2 TARG   IF1 IF2 IF3 ID1 EX1 ME1 ME2 WB1
 and                   IF1 IF2 IF3 ID1 EX1 ME1 ME2 WB1
 or                        IF1 IF2x
 sub                           IF1x

TARG:
 xor                               IF1 IF2 IF3 ID1 EX1 ME1 ME2 WB1
```

```
Baseline Impl.:     IF1 IF2 ID1 EX1 EX2 ME1 ME2 WB1
Implementation 2:   IF1 ID1 EX1 EX2 EX3 ME1 ME2 WB1
```

☑ Favorable program. ☑ Explanation.

☑ Unfavorable program. ☑ Explanation.

Implementation 2 has one fewer `IF` stage, but one more `EX` stage. So the favorable program from the last part is unfavorable here, and the unfavorable program from the last part is favorable here.

```
Baseline Impl.:     IF1 IF2 ID1 EX1 EX2 ME1 ME2 WB1
Implementation 3:   IF1 ID1 EX1 EX2 ME1 ME2 ME3 WB1
```

☑ Favorable program. ☑ Explanation.

☑ Unfavorable program. ☑ Explanation.

Implementation 3 also has one fewer IF than the baseline, but it has one more ME than the baseline. So the favorable program is the same as the favorable program from the last part. The unfavorable program has a dependence through memory.

```
# SOLUTION - Unfavorable Program
lw r1, 0(r2)       IF1 ID1 EX1 EX2 ME1 ME2 ME3 WB1
add r3, r1, r4         IF1 ID1 --------------> EX1 EX2 ME1 ME2 ME3 WB1
```

Problem 4: (14 pts) Answer the following branch predictor questions.

(*a*) Code producing the branch patterns shown below is to run on three systems, each with a different branch predictor. All systems use a 2^{14}-entry BHT. One system uses a bimodal predictor, one system uses a local predictor with a 16-outcome local history, and one system uses a global predictor with a 16-outcome global history.

```
0x1000: B1:  T N  T T N  T T T N   T N  T T N  T T T N     ...
0x1010: B2:     T      T        N     T      T        N    ...
0x1020: B3:                      T                     T   ...
```

For the questions below accuracy is after warmup.

☑ What is the accuracy of the bimodal predictor on `B1`?

The accuracy is $\frac{6}{9}$.

☑ What is the accuracy of the local predictor on branch `B1`?

The branch `B1` pattern has a length of 9, which easily fits in the 16-outcome local history, and so $\boxed{\text{the accuracy is 100\%}}$.

☑ What is the warmup time of the local predictor on branch `B1`?

The branch must first be seen 16 times to warm up the local history, then it must be seen 2×9 times to warm up each of the 9 entries that it uses. The warmup time is $\boxed{16 + 2 \times 9 \text{ executions}}$.

☑ What is the minimum local history size needed for a local predictor to predict `B1` with 100% accuracy?

$\boxed{\text{Five outcomes.}}$

If it were four outcomes the local history `TTNT` would occur before both a taken and not-taken outcome.

☑ What is the accuracy of a global predictor with a three-outcome global history on branch `B2` (not `B1`)?

Two global histories are possible: `TTN` and `NTN`. the `NTN` global history is always followed by a taken outcome. The `TTN` global history can be followed by both a taken and a not taken outcome. The patterns occur in the following repeating sequence `TTN.N`, `TTN.T`, `NTN.T`, `TTN.N`, `TTN.T`, `NTN.T`,.... Assuming one of the two `TTN` patterns mispredicts $\boxed{\text{the accuracy would be } \frac{2}{3}}$. It is also possible that both `TTN` occurrences mispredicts, in that case the accuracy would be $\frac{1}{3}$.

☑ What is the minimum global history size needed for a global predictor to predict `B2` (not `B1`) with 100% accuracy?

Five outcomes.

Problem 4, continued:

(*b*) Consider code producing the patterns below running on two systems, one using a global predictor and the other using a gshare predictor. Both systems use a 2^{16}-entry BHT and a 12-outcome global history. The hexadecimal numbers indicate the address of the branch instruction. For example, `B5: 0x1100` indicates that the instruction we call `B5` is at address `0x1100`.

```
Pattern L:  T T T ... N    (A hundred iteration loop.)

B5: 0x1100:  L       L       L       L
B6: 0x1110:    N       N       N       N
B7: 0x1200:      L       L       L       L
B8: 0x1210:        T       T       T       T
```

☑ Why is the accuracy of the gshare predictor so much better than the global predictor on this example?

The difference is on branches B6 and B8. Branch B6 is highly biased not taken and B8 is highly biased taken, a bimodal predictor could easily predict each of these branches accurately since it uses the branch address to find a two-bit counter. In contrast the global predictor uses the global history for an address in the PHT to retrieve a two-bit counter. The global history for both branches will be TTTTTTTTTTTN, because branches B5 and B7 have the same long pattern. Therefore branches B6 and B8 will share an entry and so be predicted inaccurately. The gshare predictor uses the global history exclusive-ored with the branch address as an address in the PHT. Because the PC is used branches B6 and B8 will use different entries and so be predicted accurately.

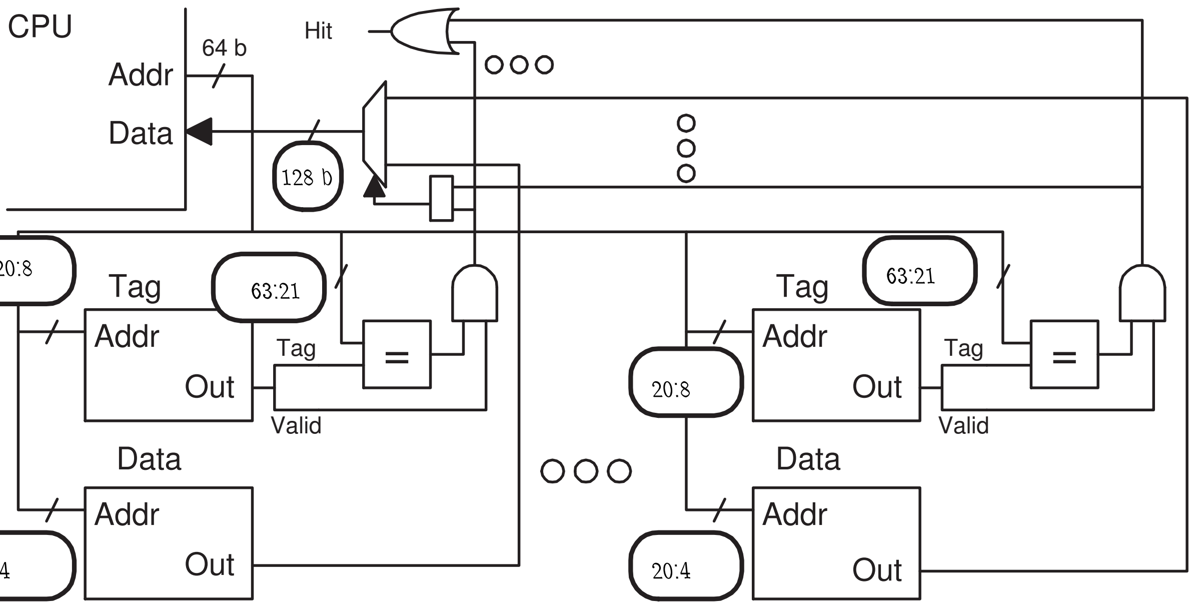
☑ What is the minimum number of BHT entries for which the gshare predictor will outperform global on this code? *Hint: Look at the branch addresses.*

The minimum number is 2^7 entries. If there are any fewer than address B7 and B6 will map to the same entry because the lower $6 + 2 = 8$ bits of their addresses are identical. (The $+2$ is needed because the two least significant bits, which are always zero, are not used in computing the PHT address.)

Problem 5: (15 pts) The diagram below is for a **4-way** set-associative cache with a capacity of 8 MiB (2^{23} bytes). The system has the usual 8-bit characters.

(*a*) Answer the following, formulæ are fine as long as they consist of grade-time constants.

☑ Fill in the blanks in the diagram.



☑ Show the address bit categorization. Label the sections appropriately. (Alignment, Index, Offset, Tag.)

	Tag	Index	Offset
Address:	63 … 21	20 … 8	7 … 4, 3 … 0

☑ Memory Needed to Implement (Indicate Unit!!):

It's the cache capacity, 2^{23} characters, plus $4 \times 2^{21-8}\,(64 - 21 + 1)$ bits.

☑ Line Size (Indicate Unit!!):

Line size is $2^{8} = 256$ characters.

☑ Show the bit categorization for a direct mapped cache with the same capacity and line size.

	Tag	Index	Offset
Address:	63 … 23	22 … 8	7 … 4, 3 … 0

Problem 5, continued: The problems on this page are **not** based on the cache from the previous page. The code fragments below run on a 32 MiB (2^{25} byte) direct-mapped cache with a 64-byte line size. Each code fragment starts with the cache empty; consider only accesses to the array, `a`.

(*b*) Find the hit ratio executing the code below.

☑ What is the hit ratio running the code below? Explain

```
float sum = 0.0;
float *a = 0x2000000; // sizeof(float) == 4
int i;
int ILIMIT = 1 << 11;    // = 2^11

for (i=0; i<ILIMIT; i++) sum += a[ i ];
```

The line size of 2^6 characters is given, the size of an array element is $4 = 2^2$ characters, and so there are $2^{6-2} = 2^4$ elements per line. The first access, at `i=0`, will miss but bring in a line with 2^4 elements, the next $2^4 - 1$ accesses will be to data on the line, but sixteenth access after the miss will miss again. Therefore the hit ratio is $\frac{15}{16}$.

(*c*) Find the smallest value of `STRIDE` for which the cache hit ratio in the program below will be zero.

☑ Fill in smallest value for `STRIDE` for which hit ratio 0.

☑ Briefly explain.

The value of `STRIDE` should be chosen so that the index of `a[i]` and `a[i+STRIDE]` are the same (but of course their tags will be different). We want the difference in addresses to be 2^{25}. Since an array element is 4 bytes that means that `STRIDE` must be $\frac{2^{25}}{4} = 2^{23}$.

```
float sum = 0.0;
float *a = 0x2000000; // sizeof(float) == 4
int i;
int ILIMIT = 1 << 11;    // = 2^11


int STRIDE =      1 << 23;   // <-- FILL IN.   SOLUTION FILLED IN.


for (i=0; i<ILIMIT; i++) sum += a[ i ] + a[ i + STRIDE ];
```

Problem 6: (15 pts) Answer each question below.

(*a*) What is the difference between instruction-level parallelism (ILP) and explicit parallelism?

☑ ILP and explicit parallelism difference.

Short answer, sufficient to get full credit: ILP is a property of machine language code that is written for a serial system, it is a measure of how many instructions can execute at the same time. Explicit parallelism is something specified by a programmer or compiler.

More detailed explanation: Explicit parallelism is something specified by the programmer, perhaps using a special language or more often an API such as pthreads, OpenMP, or MPI to specify how parts of the program should execute in parallel. Explicit parallelism might also be specified by a parallelizing compiler. In contrast, instruction level parallelism is something that is present in serial (non-parallel) programs at the machine-language level. It is a measure of how many instructions can execute at the same time. Programs with lots of ILP execute well on superscalar processors.

(*b*) A company has a large customer base for its ISA Y products, the most advanced of which is a 4-way superscalar dynamically scheduled implementation. The company is considering three possible next-generation systems: Develop an 8-way superscalar dynamically implementation of ISA Y, develop a chip with 16 scalar implementations of ISA Y, or develop a VLIW ISA and implementation. For each strategy indicate how much effort it will take for customers to use the new system.

☑ Customer effort to use 8-way superscalar dynamically scheduled system. Explain.

The company's current product is a 4-way superscalar chip. Code should run on the 8-way chip without any modification and so the customer effort (in preparing the code) is zero.

☑ Customer effort to use chip with 16 scalar implementations. Explain.

Presumably the company's customers have not parallelized their code. Since the chip has sixteen separate processors customers will have to parallelize their code, otherwise the code will just run on one core and so run slowly (since its scalar). Parallelizing code is hard, and so customer effort is high.

☑ Customer effort to use VLIW implementation.

The term VLIW refers to a style of ISA. To use the new chip customers will have to re-compile their code. That will require moderate to low effort.

(*c*) Consider the three systems from the previous part. For each system indicate an advantage over the others. The advantage should specify an assumed workload, an answer might start "The advantage, those customers that run programs that are __________, is ....

☑ Advantage of 8-way superscalar dynamically scheduled system.

It can run existing code without modification. Dynamic scheduling is well suited to typical "integer" workload, with its many medium difficulty branches and irregular data access patterns.

☑ Advantage of 16 scalar system chip.

Because the processors are simple, many can fit on a chip. Workload that can be efficiently parallelized will run fastest on this system.

☑ Advantage of VLIW implementation.

The intended advantage of VLIW is simpler implementations compared to superscalar techniques. So far this has only achieved success in DSP (digital signal processing) chips. So lets assume the workload consists of DSP programs.

Problem 7: (18 pts) Answer each question below.

(*a*) Why might a 1% change in branch prediction accuracy have a larger impact on the performance of a 4-way superscalar processor than on a 2-way superscalar system?

☑ Larger impact on 4-way over 2-way because...

Because the 4-way system will fetch twice as many instructions per cycle. Since the time to resolve the branch will likely be the same on the two systems, the 4-way system will waste more of its execution potential. For example, suppose that the 4-way system takes 10 seconds to execute a program and the 2-way system takes 15 seconds. Suppose that the time to resolve mispredicted branches on both systems (in total) is 3 seconds. That 3 seconds is a bigger percentage of 10 than 15, so the 1% improvement will have a bigger impact on the 4-way system.

(*b*) Dynamically scheduled systems use a technique called register renaming in which an instruction's architected registers are renamed into physical ones. Provide a brief example illustrating why register renaming is necessary.

☑ Example illustrating need for register renaming and explanation.

```
# Example for solution.
mul.s f0, f1, f2   IF ID Q  RR M1 M2 M3 M4 M5 M6 WB C
add.s f3, f0, f5      IF ID Q                  RR A1 A2 ..
lwc1  f0, 0(r1)          IF ID Q  EA ME WB
sub.s f6, f0, f7            IF ID Q  RR A1 A2
```

In the example above the lw writes back before the mul. Without register renaming the value in the `f0` register would be wrong after the `mul.s` writes back. Any later instruction would get the `mul.s` result, not the `lwc1` result which is correct.

(*c*) The SPECcpu rules require that the compilers used to prepare a run of the suite be real products that the company (or some other company) makes an honest effort to sell (or give away). Explain how the SPECcpu scores might be inflated if the compilers could be products in a technical sense, but not something the company is trying to put in customer hands.

☑ Scores inflated by unmarketed compilers because...

The compilers might include buggy optimizations which work for the SPECcpu benchmarks, because those specific problems have been fixed, but would still be buggy for other code. Actual users would be reluctant to use a buggy compiler, even if avoiding the bugs would yield a performance benefit.

☑ If the compilers produce faster programs, why not promote them?

Based on the answer above, because that would alienate customers.

(*d*) What is the difference between a trap and an exception?

☑ Difference between trap and exception.

A trap is an instruction which is intended to start a handler. It's usually used to implement the interface to an operating system, among other purposes. The programmer might insert a trap instruction to request that the operating system, say, open a file. An exception is the response of the system to something going wrong with an instruction, for example, a bad opcode or memory address. As with a trap the handler is called, but unlike a trap an exception can happen to most any instruction.