Name _____

Computer Architecture

EE 4720

Final Examination

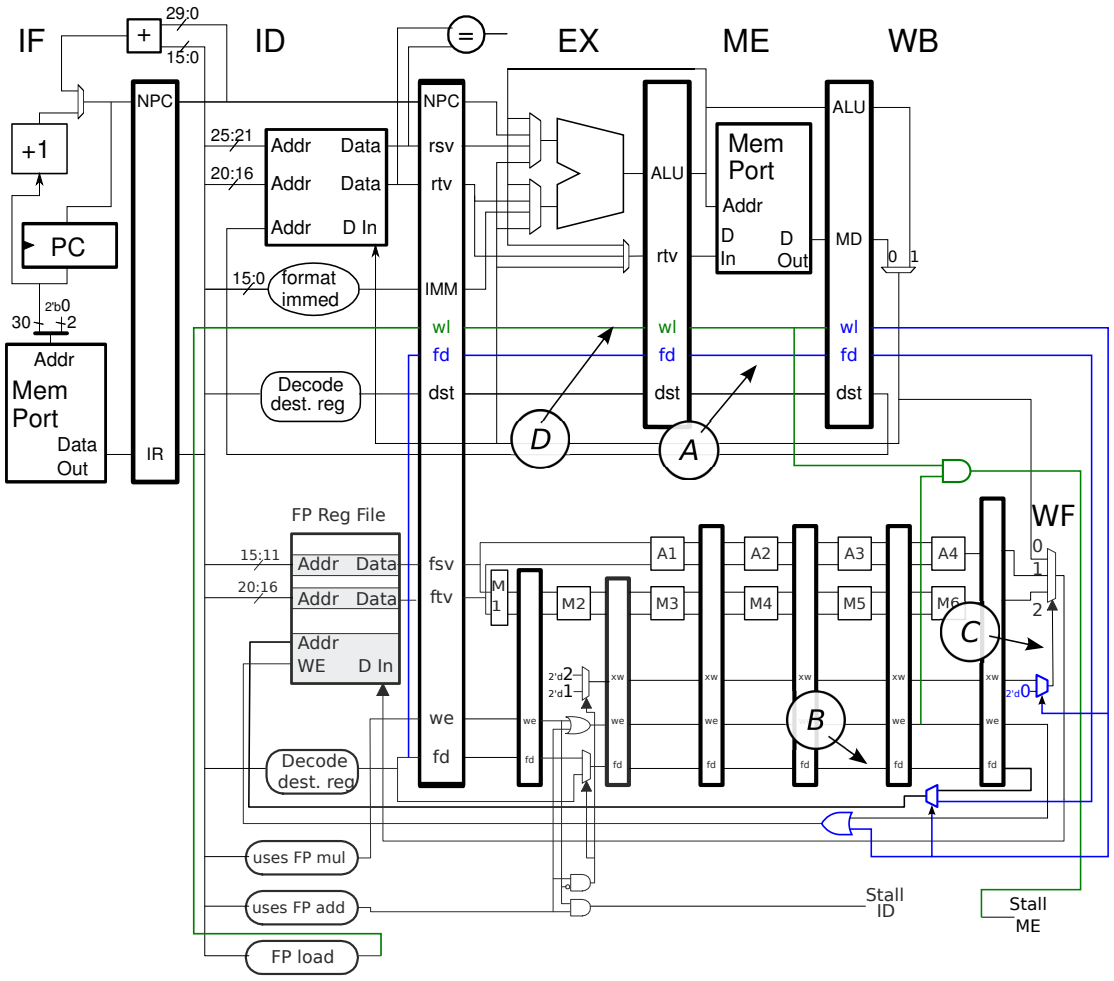
10 May 2014, 10:00–12:00 CDT

Problem 1	_____	(15 pts)
Problem 2	_____	(15 pts)
Problem 3	_____	(15 pts)
Problem 4	_____	(15 pts)
Problem 5	_____	(5 pts)
Problem 6	_____	(10 pts)
Problem 7	_____	(25 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: (15 pts) Illustrated below is the **stall-in-ME** version of our MIPS implementation, taken from the solution to the midterm exam.



(*a*) Show a pipeline execution diagram of the code below on this pipeline.

(*b*) Wires in the diagram are labeled A, B, C, and D. Under your pipeline execution diagram show the values on those wires when they are in use.

☐ Show pipeline execution diagram. ☐ Show values of A, B, C, and D.

```
add.s f4,f5,f6

sub.s f7,f8,f9

lwc1 f1, 0(r2)
```

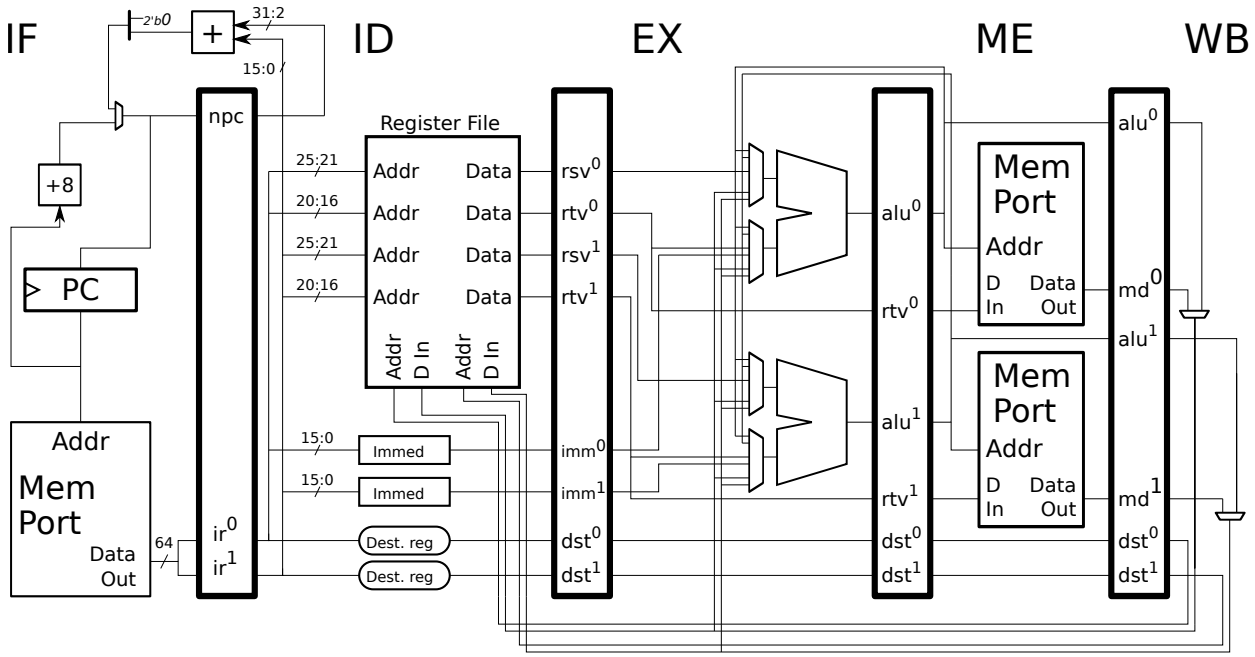
A:

B:

C:

D:

Problem 2: (15 pts) Illustrated below is a 2-way superscalar MIPS implementation. Design the hardware described below. You can use the following logic blocks (with appropriate inputs) in your solution: The output of logic block isALU is 1 if the instruction's result is computed by the integer ALU. The output of logic block rtSrc is 1 if the instruction uses the `rt` register as a source. The output of logic block isStore is 1 if the instruction is a store.



(*a*) Design logic to generate a signal named `STALL`, which should be `1` when there is a true (also called data or flow) dependence between the two instructions in `ID`.

☐ Control logic to detect true dependence in `ID` and assign `STALL`.

(*b*) The code fragment below should generate a stall in our two-way superscalar implementation when the two instructions are in the same fetch group. However this particular instruction pair is a special case in which the stall is not necessary when the right bypass path(s) and control logic are provided. *Note: There was a similar-sounding problem in last year's final, but the solutions are different.*

```
0x1000: add r1, r2, r3
0x1004: sw r1, 0(r5)
```

☐ Add the bypass path(s) needed so that the code executes without a stall.

☐ Add control logic to detect this special case and use it to suppress the stall signal from the first part.

Problem 3: (15 pts) Code producing the branch patterns shown below is to run on three systems, each with a different branch predictor. All systems use a 2^{30} entry BHT. One system has a bimodal predictor, one system has a local predictor with a 12-outcome local history, and one system has a global predictor with a 12-outcome global history.

(*a*) Branch behavior is shown below. Notice that `B2`'s outcomes come in groups of three, such as 3 `q`s. The first outcome of each group is random and is modeled by a Bernoulli random variable with $p = .5$ (taken probability is .5). The second and third outcomes are the same as the first. For example, if the first `q` is `T` the second and third `q` will also be `T`. If the first `s` is `N`, the second and third `s` will also be `N`. Answer each question below, the answers should be for predictors that have already warmed up.

```
B1:  T   T   T   N   N     T   T   T   N   N    ...
B2:   r   r   r   q   q     q   s   s   s   u    ...
B3:    T   T   T   T   T     T   T   T   T   T   ...
```

☐ What is the accuracy of the bimodal predictor on branch `B1`?

☐ What is the approximate accuracy of the bimodal predictor on branch `B2`? ☐ Explain.

☐ What is the minimum local history size needed to predict `B1` with 100% accuracy?

☐ What is the accuracy of the local predictor on branch `B2`, after warmup. ☐ Explain.

☐ What is the best local history size for branch `B2`, *taking warmup into consideration.* ☐ Explain.

☐ How many different GHR values will there be when predicting `B3`?

Problem 3, continued:

In this part consider the same predictors as on the previous page, except this time the BHT has 2^{14} entries. Also, consider the same branch patterns, they are repeated below, along with the address of branches `B1` and `B2`. The branch predictors are part of a MIPS implementation.

```
0x1234: B1:  T   T   T   N   N     T   T   T   N   N    ...
0x1242: B2:   r   r   r   q   q     q   s   s   s   u    ...
        B3:    T   T   T   T   T     T   T   T   T   T   ...
```

(*b*) Choose an address for branch `B3` that will result in a BHT collision with branch `B1`.

☐ Address for `B3` that results in a collision.

(*c*) How does the collision change the prediction accuracy of the bimodal predictor on the two branches?

☐ Change in `B1` and ☐ Change in `B3`.

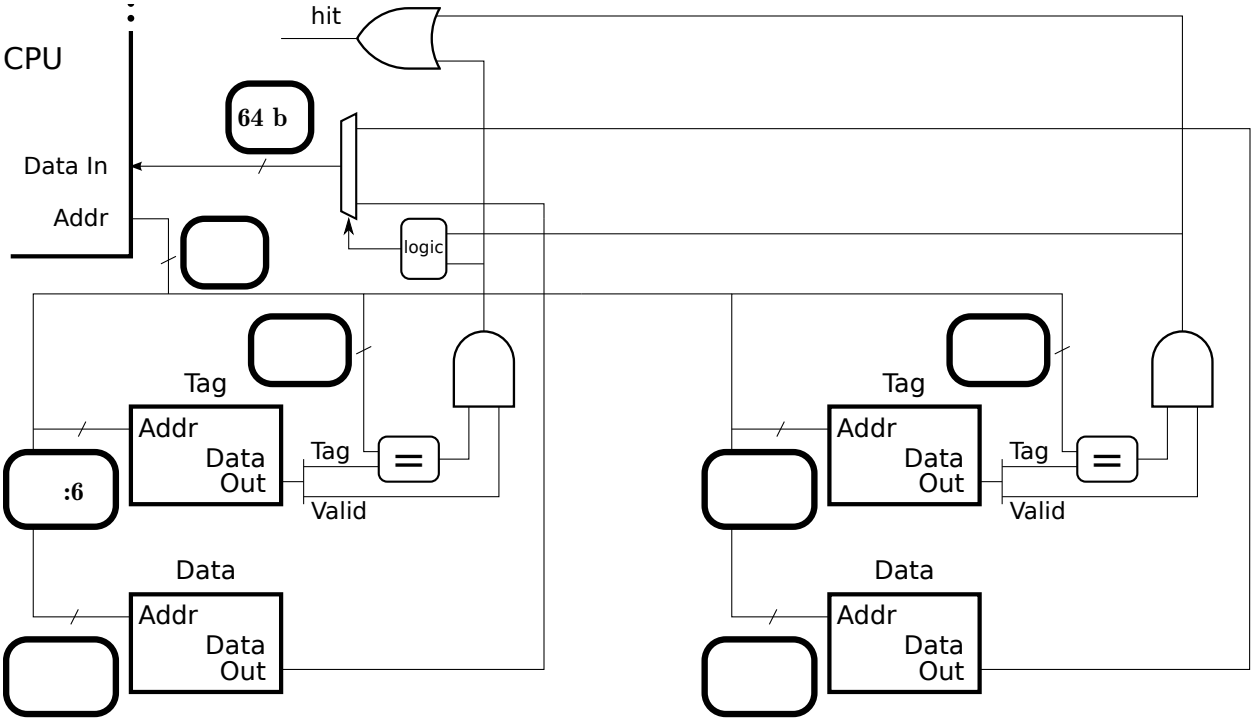
(*d*) (The answer to the following question does not depend on the sample branch patterns above.) Suppose we detect a BHT collision (perhaps by using tags). Why should we predict not taken?

☐ Reason for predicting not-taken for a collision.

Problem 4: (15 pts) The diagram below is for a 256 kiB (2^{18} B) set-associative cache. Hints about the cache are provided in the diagram.

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

☐ Fill in the blanks in the diagram.



☐ Complete the address bit categorization. Label the sections appropriately. (Index, Offset, Tag.)

Address: ☐ ☐ ☐

31 0

☐ Associativity:

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

☐ Line Size (Indicate Unit!!):

☐ Show the bit categorization for a **fully associative** cache with the same capacity and line size.

Address: ☐ ☐ ☐

Problem 4, continued: The problems on this page are **not** based on the cache from the previous page. The code in the problems below run on a 4 MiB (2^{22} byte) 4-way set-associative cache with a line size of 128 bytes.

Each code fragment starts with the cache empty; consider only accesses to the arrays.

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

```
double sum = 0;
double *a = 0x20000000; // sizeof(double) == 8
int i;
int ILIMIT = 1 << 11;    // = 2^11

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

☐ What is the hit ratio running the code above? Show formula and briefly justify.

(*c*) Find the largest value for `BSIZE` for which the second `for` loop will enjoy a 100% hit ratio.

```
struct Some_Struct {
  double val;        // sizeof(double) = 8
  double norm_val;
  double a[14];
 };

  const int BSIZE =                          ; // <-  FILL IN
  Some_Struct *b;
  for ( int i = 0;  i < BSIZE;  i++ ) sum += b[i].val;
  for ( int i = 0;  i < BSIZE;  i++ ) b[i].norm_val = b[i].val / sum;
```

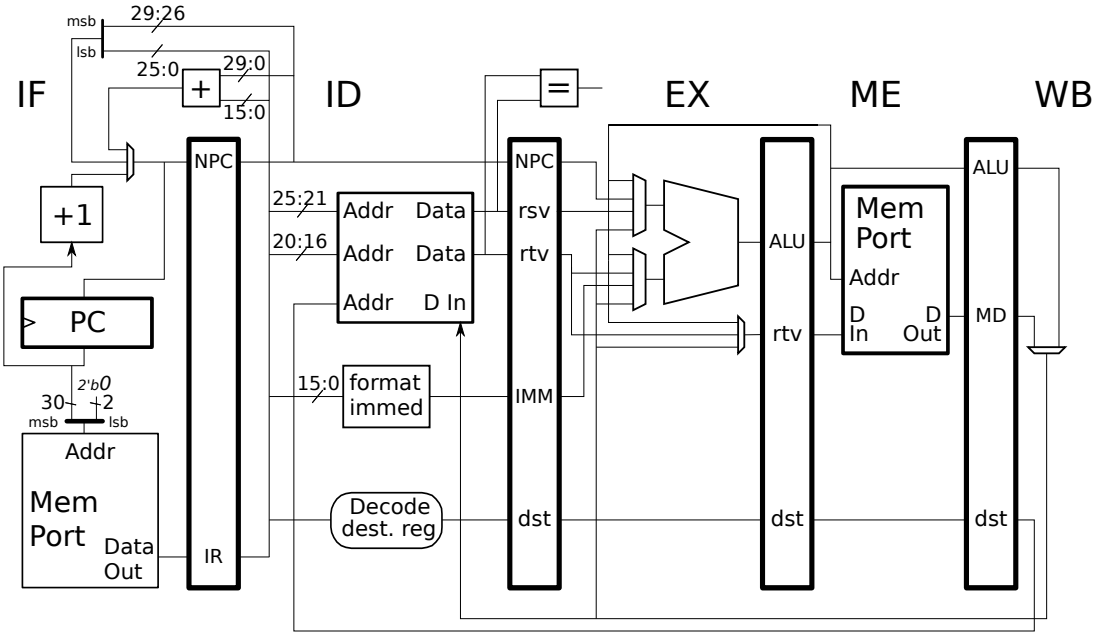
Problem 5: (5 pts) The displacement in MIPS branches is 16 bits. Consider a new MIPS branch instruction, `bfeq rsn, rtn` (branch far), where `rsn` and `rtn` are 2-bit fields that refer to registers 4-7. As with `beq`, branch `bfeq` is taken if the contents of registers `rsn` and `rtn` are equal. With six extra bits `bfeq` can branch 64 times as far.

(*a*) Show an encoding for this instruction which requires as few changes to existing hardware as possible.

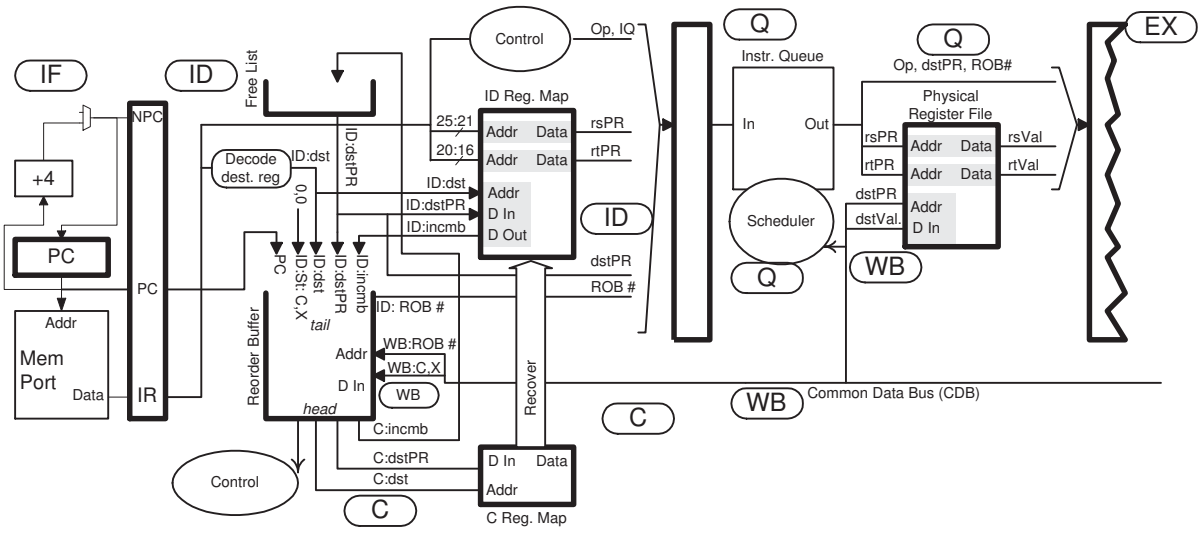
☐ Encoding for `bfeq`. ☐ Explain how minimizes changes.

(*b*) Modify the pipeline below to implement the new instruction. Use as little hardware as possible.

☐ Briefly show changes.



Problem 6: (10 pts) Illustrated below is a dynamically scheduled four-way superscalar MIPS implementation and the execution of code on that implementation.




```
LOOP: #   Cycle     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
 lwc1 f2, 0(r1)     IF ID Q  RR EA ME WB
 add.s f4, f4, f2   IF ID Q        RR A1 A2 A3 A4 WB
 bne  r1, r2, LOOP  IF ID Q  RR B  WB
 addi r1, r1, 4     IF ID Q  RR EX WB
LOOP: #             0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
 lwc1 f2, 0(r1)        IF ID Q  RR EA ME WB
 add.s f4, f4, f2      IF ID Q                 RR A1 A2 A3 A4 WB
 bne  r1, r2, LOOP     IF ID Q  RR B  WB
 addi r1, r1, 4        IF ID Q  RR EX WB
#         Cycle     0  1  2  3  4  5  6  7  8  9  10 11 12 13 14
```

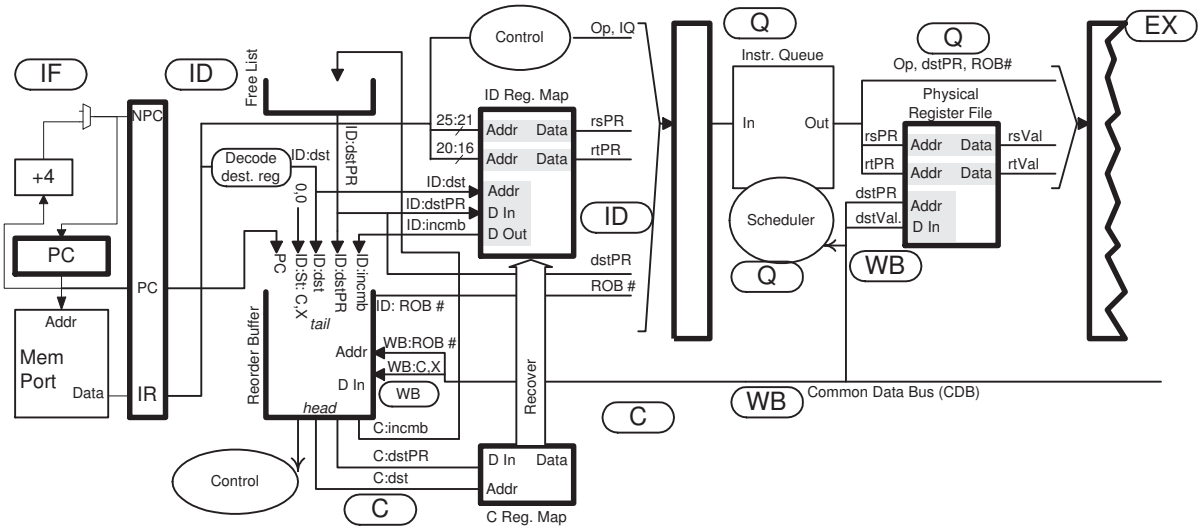
(*a*) On the diagram above indicate when each instruction will commit.

☐ Show commits on diagram above.

(*b*) What is the execution rate, IPC, for the code above for a large number of iterations assuming perfect branch prediction. Note that the system is dynamically scheduled.

☐ IPC for code above for large number of iterations.

(*c*) On the next page there is a table showing the values of selected signals during the execution of the code, the signals are related to register renaming. Show values where indicated on the table. Note that `ID:incmb` is already shown in cycle 1, show its values for later cycle(s).



☐ Show values where indicated.

```
LOOP: #   Cycle      0   1   2   3   4   5   6   7   8   9   10  11  12  13  14
 lwc1 f2, 0(r1)      IF  ID  Q   RR  EA  ME  WB
 add.s f4, f4, f2    IF  ID  Q           RR  A1  A2  A3  A4  WB
 bne  r1, r2, LOOP   IF  ID  Q   RR  B   WB
 addi r1, r1, 4      IF  ID  Q   RR  EX  WB
LOOP: #   Cycle      0   1   2   3   4   5   6   7   8   9   10  11  12  13  14
 lwc1 f2, 0(r1)          IF  ID  Q   RR  EA  ME  WB
 add.s f4, f4, f2        IF  ID  Q                       RR  A1  A2  A3  A4  WB
 bne  r1, r2, LOOP       IF  ID  Q   RR  B   WB
 addi r1, r1, 4          IF  ID  Q   RR  EX  WB
#         Cycle      0   1   2   3   4   5   6   7   8   9   10  11  12  13  14
ID:dstPR 0 (lwc1)       65  62
ID:dstPR 1 (add.s)      97  69
ID:dstPR 3 (addi)       60  79
# Show values for signals below, including incmb.
#         Cycle      0   1   2   3   4   5   6   7   8   9   10  11  12  13  14

ID:incmb 0 (lwc1)       83
ID:incmb 1 (add.s)      20
ID:incmb 3 (addi)       67

ID:rsPR 0 (lwc1)

ID:rsPR 1 (add.s)

ID:rsPR 3 (addi)

ID:rtPR 1 (add.s)                                                   <- rt, not rs
#         Cycle      0   1   2   3   4   5   6   7   8   9   10  11  12  13  14
WB:dstPR 0 (lwc1)

WB:dstPR 1 (add.s)

WB:dstPR 3 (addi)

#         Cycle      0   1   2   3   4   5   6   7   8   9   10  11  12  13  14
```

Problem 7: (25 pts) Answer each question below.

(*a*) Describe how cost and performance limit the practical largest value of width (value of n) in an n-way superscalar implementation.

☐ Cost limiter.

☐ Performance limiter.

(*b*) What is the most important factor in determining the size of a level 1 cache?

☐ Most important factor in L1 cache size.

(*c*) Suppose the 16-bit offset in MIPS `lw` instructions was not large enough. Consider two alternatives. In alternative 1 the offset in the existing `lw` instruction is the immediate value times 4. So, for example, to encode instruction `lw r1, 32(r2)` the immediate would be 8. In alternative 2 the behavior of the existing `lw` is not changed but there is a new load `lws r1, 32(r2)`, in which the immediate is multiplied by 4. Note that alternative 2 requires a new opcode. Which instruction should be added to a future version of MIPS?

☐ Should choose alternative 1 or alternative 2? ☐ Explain.

(*d*) The SPECcpu suite comes with the source code for the benchmark programs. How does that help with the goal of measuring new ISAs and implementations?

☐ Source code helps with testing new implementations because:

(*e*) What's the difference between a 4-way superscalar implementation (of say MIPS) and a VLIW system with a 4-slot bundle?

☐ Difference between superscalar and VLIW.