Name Solution

Computer Architecture

EE 4720

Final Examination

4 May 2016, 15:00–17:00 CDT

Problem 1		(20 pts)
Problem 2		(20 pts)
Problem 3		(20 pts)
Problem 4		(20 pts)
Problem 5		(20 pts)
Exam Total		(100 pts)

Alias Leaked

Good Luck!

Problem 1: (20 pts) Appearing below is our two-way superscalar MIPS implementation with a single, unconnected memory port in the `ME` stage. Since there is only one memory port there will have to be a slot-1 `ID`-stage stall whenever `ID` contains two memory instructions, for example:

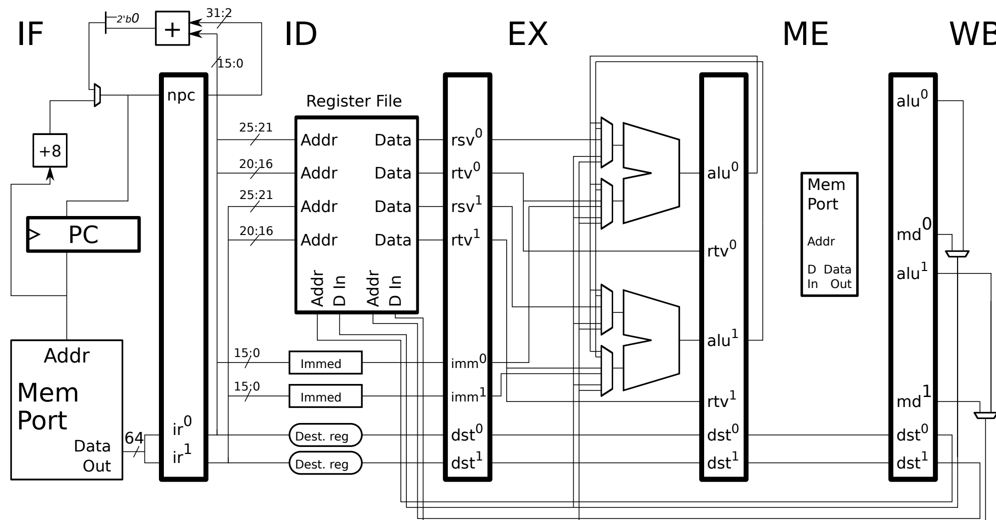
```
# Cycle        0  1  2  3  4  5
lw r1, 0(r2)   IF ID EX ME WB
lw r3, 0(r4)   IF ID -> EX ME WB
```

(*a*) On the next page add datapath to the implementation so that the memory port can be used by a load or store instruction in either slot. Pay attention to the cost of pipeline latches.

(*b*) On the next page add control logic needed for the datapath changes. The control logic should be for any multiplexors that you've added, don't include control logic for the memory port itself.

(*c*) On the next page add control logic to generate a `STALL_ID_1` signal when there are two memory instructions in `ID`, as occurs in cycle 1 to the `lw r3` in the example above.

Use next page for solution.



Use next page for solution.

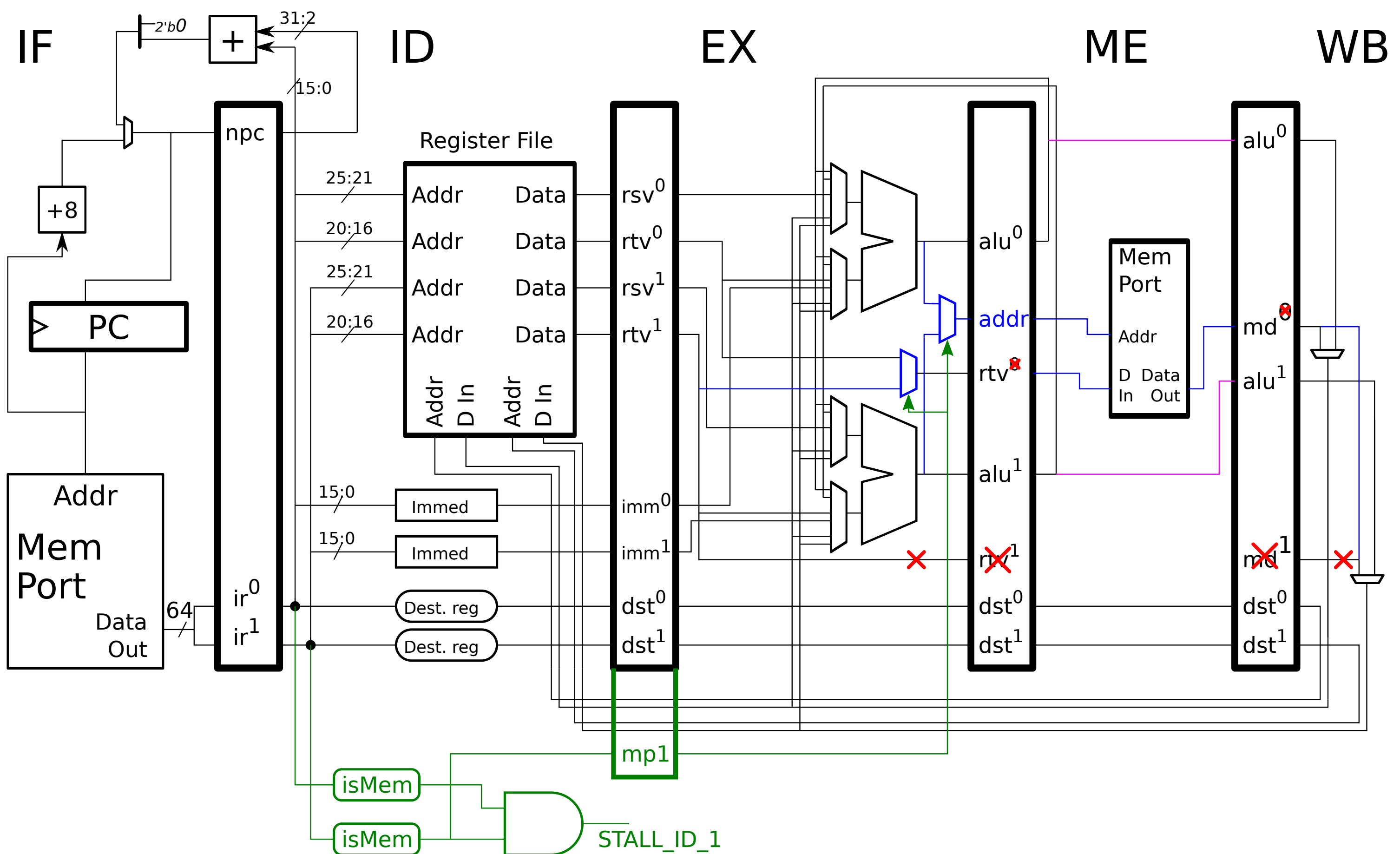
Problem 1, continued:

☑ Datapath so that memory port can be used by a ☑ load or ☑ store in either slot.

☑ Control logic for multiplexors that you've added. ☑ Control logic to generate `STALL_ID_1`.

☑ Pay attention to ☑ **pipeline latch cost** and ☑ the critical path.

Solution appears below. The added datapath is in blue, reconnected existing datapath is in purple, and control logic is in green. In EX multiplexors have been added to select the address and store value from the appropriate slot. By putting the store value multiplexor in EX we can eliminate the `ME.rtv1` pipeline latch, shown exed out in red. A multiplexor *and* a new pipeline latch were added for the store address in EX; putting the address multiplexor in ME was out of the question because of critical path and since we don't know whether the memory instruction will be in slot 0 or slot 1, we couldn't just connect the mux to `ME.alu0` (as we did with the store value, to `ME.rtv0`). The memory port output is connected only to `md0`, the connection to slot 1 is made in the WB stage, saving another pipeline latch. The non-memory-related `alu0` and `alu1` connections are reconnected, that's shown in purple. Control logic appears in green. The output of isMem is logic `1` if the instruction is any kind of load or store.



(*d*) Notice that the two instructions in the code below load from adjacent addresses. The superscalar implementation from the previous page would stall the `lbu r3`. But that might not be necessary because the memory port retrieves 32 b of data. A properly designed alignment network could provide data for both instructions, in some cases. Let's call such nice situations, *shared loads*.

```
lbu r1, 0(r2)
lbu r3, 1(r2)
```

Only show `ID`-stage changes. Show control logic to detect possible shared loads in `ID`. Generate a signal named `SHARED_LOAD` which is `1` if the instructions in `ID` could form a shared load, based on register numbers and immediate values.

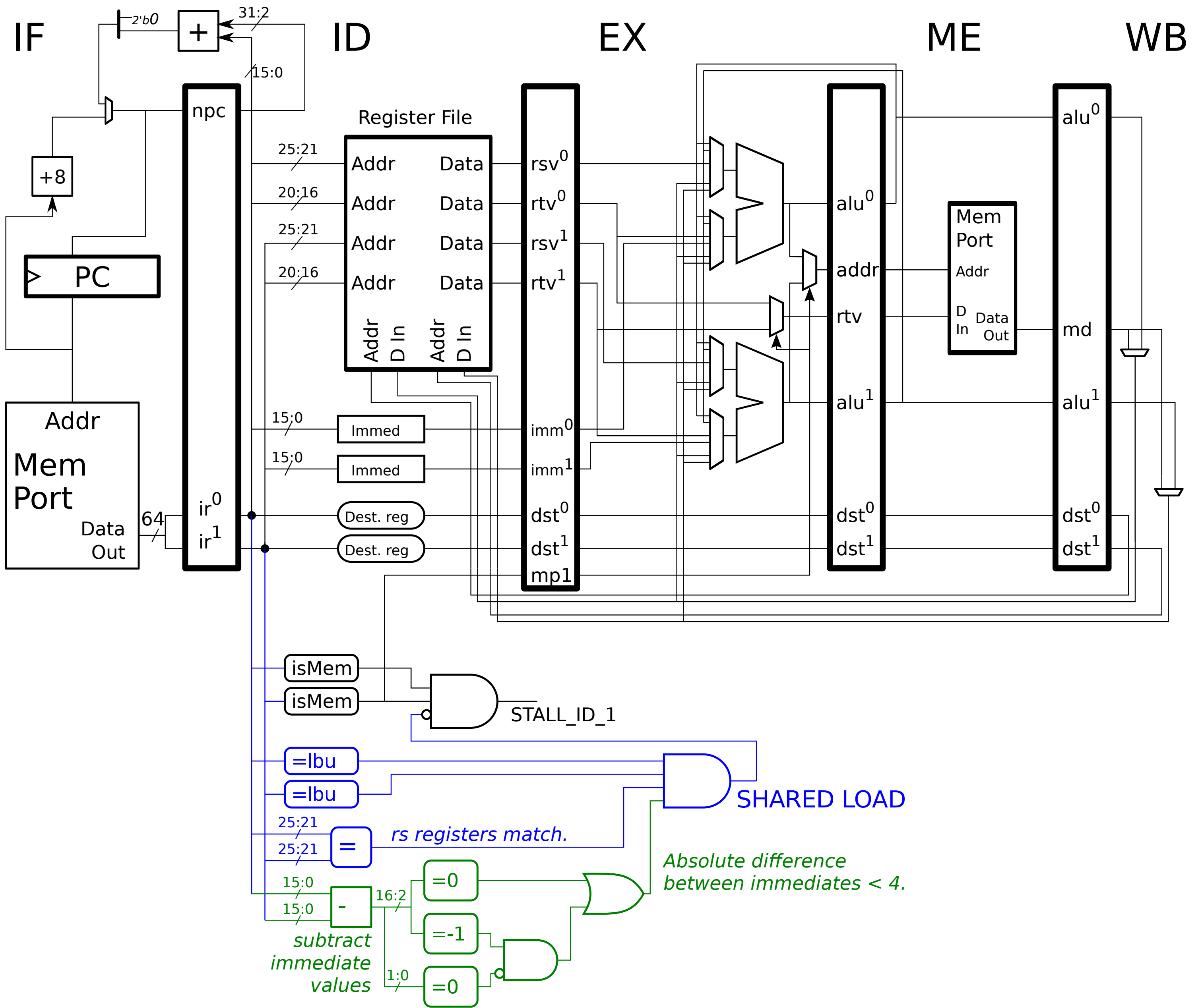
☑ Control logic to detect possible shared loads.

Solution appears below. The logic in blue checks that both instructions are `lbu` and that the `rs` register numbers of both instructions are the same. The logic in green checks that the absolute value of the difference between the immediate values is less than four. This is shown with a subtraction followed by simple gates. Full credit would have been given for a box labeled with something like `abs < 4`.

The logic from Parts a, b, and c are shown in black. The `SHARED_LOAD` signal is used to suppress the two-loads stall.

☑ Explain why alignment and critical path make it impossible to know for sure in `ID` that a shared load is possible.

Due to alignment restrictions both addresses must be identical in bits 31:2. If we assume that bits 1:0 of the base register (`r2` in the example) are 0 then all we need to do is check whether bits 15:2 of the two immediates are identical. But suppose bits 1:0 of the register are 3. Then the two addresses in the example will not be in the same four-byte aligned chunk.



IF
ID
EX
ME
WB
2'b0
+
31:2
15:0
npc
+8
PC
Addr
Mem
Port
Data
Out
64
ir0
ir1
Register File
25:21
20:16
25:21
20:16
Addr
Data
D In
rsv0
rtv0
rsv1
rtv1
15:0
Immed
imm0
imm1
Dest. reg
dst0
dst1
mp1
alu0
addr
rtv
alu1
Mem
Port
Addr
D
In
Data
Out
md
isMem
STALL_ID_1
=lbu
SHARED LOAD
rs registers match.
=
Absolute difference
between immediates < 4.
-
16:2
1:0
=0
=-1
subtract
immediate
values

The following part was NOT on the final exam as given. It will be assigned as homework in 2017.

(*e*) Notice that the two instructions in the code below load from adjacent addresses. The superscalar implementation from the previous page would stall the `lbu r3`. But that might not be necessary because the memory port retrieves 32 b of data. A properly designed alignment network could provide data for both instructions, in some cases. Let's call such nice situations, *shared loads*.

```
lbu r1, 0(r2)
lbu r3, 1(r2)
```

Here's the plan: In `ID` detect whether a shared load is possible. In `EX` check addresses. In `ME` have memory port perform only one kind of load, 32 b (stores are unaffected). In `WB` have a separate alignment network for each slot.

Show control logic to detect possible shared loads in `ID`. Generate a signal named `SHARED_LOAD` which is `1` if the instructions in `ID` could form a shared load, based on register numbers and immediate values. In `EX` generate a `STALL-EX-1-SL` signal if the shared load cannot be performed. This will stall the pipeline allowing the slot-0 load to proceed normally in the current cycle and the slot-1 load to be executed in the next cycle. Show connections for the alignment units in the appropriate places.

☑ Control logic to detect possible shared loads.

Solution appears below in green and blue. The only difference with the changes in `ID` below and those from Part d is that below the low three bits of the difference between the immediates is put in a pipeline latch, `EX.di`, shown in purple. Signal `di` (delta immediates) is used in `EX` to check whether a shared load can actually be done.

☑ Explain why alignment and critical path make it impossible to know for sure in `ID` that a shared load is possible.

See solution to Part d.

☑ `EX`-stage hardware to generate `STALL-EX-1-SL` if can't do shared load based on addresses.

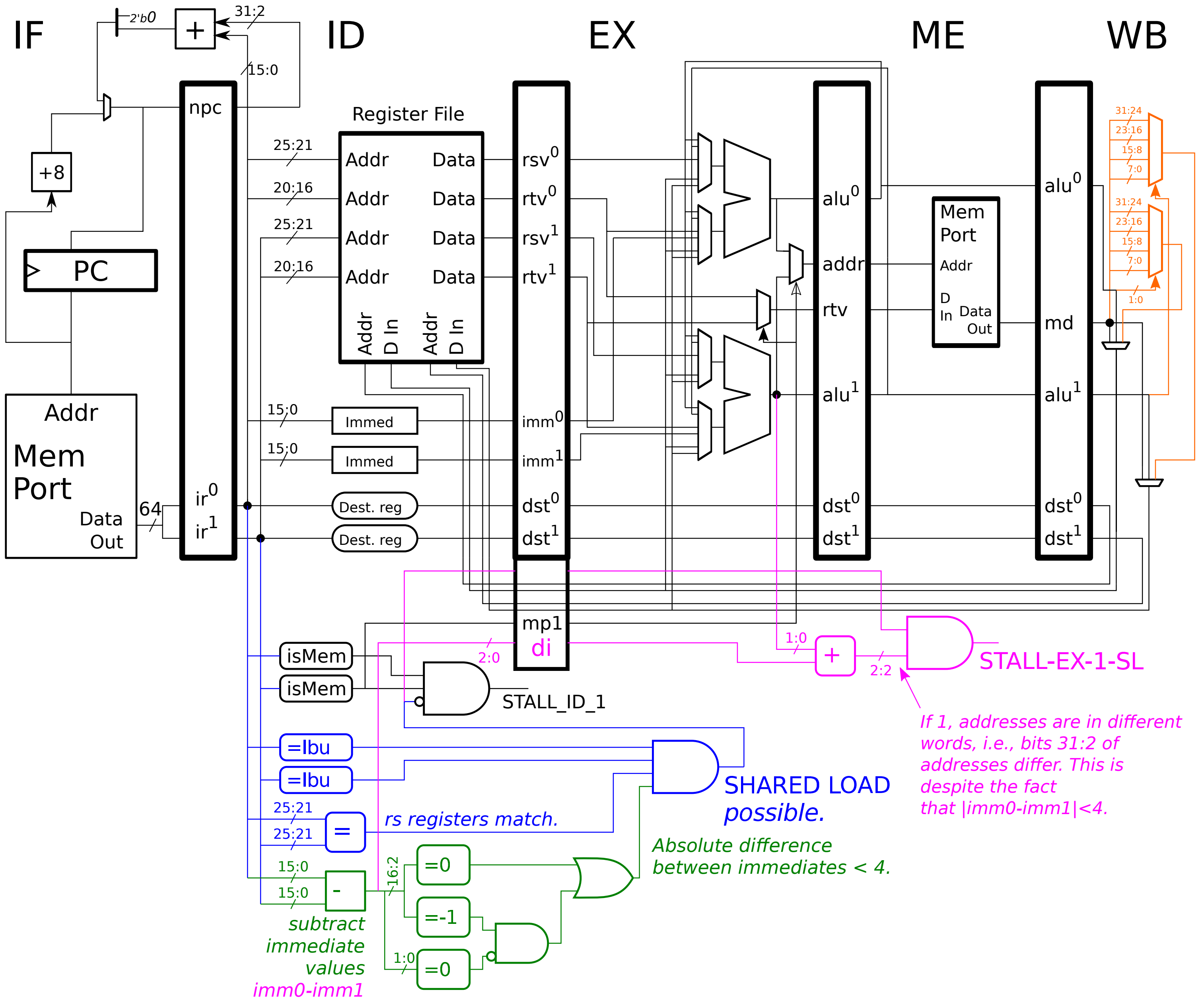
The difference `imm0-imm1` is added to the low two bits of the address of the slot-1 load. If that sum is > 3 or < 0 then bits 31:2 of the two addresses must differ. That logic is shown in purple. The `STALL-EX-1-SL` signal is generated when the sum is out of range and when there was possible shared load when the instruction was in `ID`.

For example, consider the two loads `lbu r1, 0(r2) lbu r3, 1(r2)`. For these loads `imm0-imm1` is -1 or 111_2 as a 2's complement 3-bit number. Suppose `r2` (`rsv`) is `0x1000`. The slot-1 address will be `0x1001`, the two least significant bits are 01_2. Adding these two yields: $111_2 + 01_2 = 000_2$. Bit 2 is zero so the shared load can go ahead. Next, suppose that `r2` is `0x1003`. Now, the address for the slot-1 load is `0x1003+1=0x1004`. The two least significant bits are 00_2. In the sum $111_2 + 00_2 = 111_2$ the bit at position 2 (lsb is position 0) is 1, and so a shared load is not possible.

Note that the stall determination is made by logic that examines just $3 + 2$ bits. Checking whether the high 30 bits of the two ALU outputs were the same would be correct but would stretch the critical path.

☑ Add remaining `ME`- and `WB`-stage hardware.

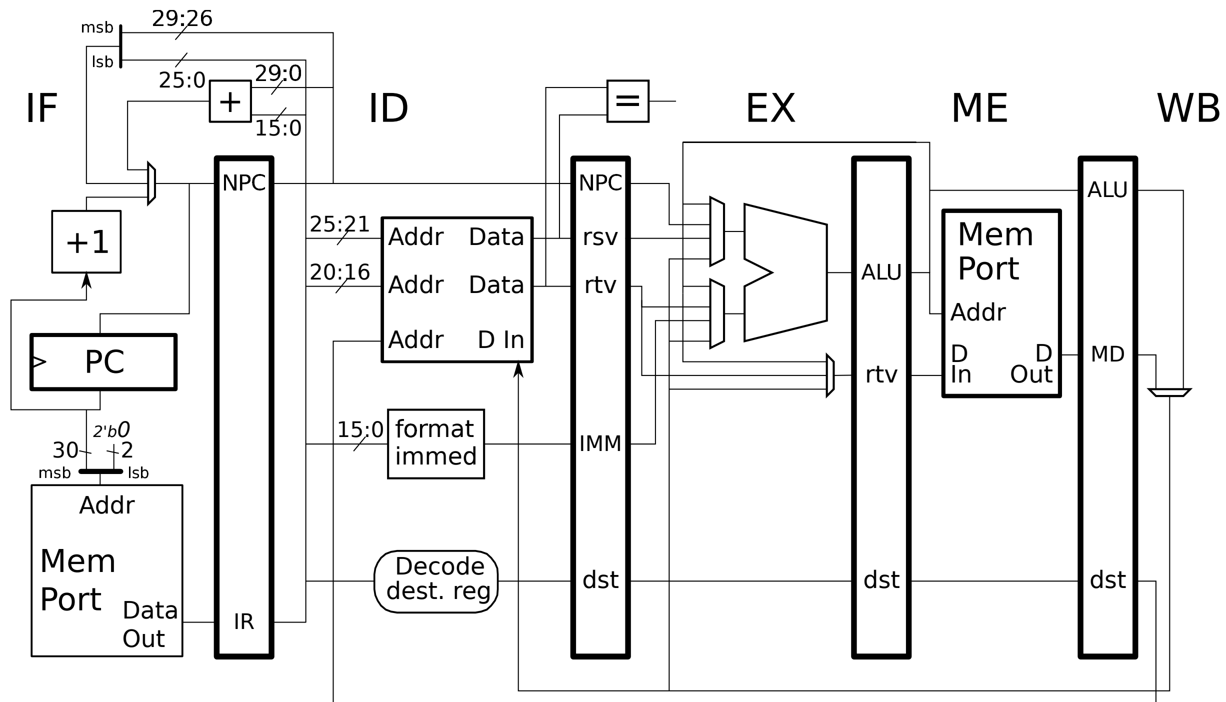
The solution appears below in orange. Multiplexors select the appropriate 8 bits from the memory port output based on the two least significant bits of the address.



IF
ID
EX
ME
WB
2'b0
+
31:2
15:0
npc
+8
PC
Addr
Mem
Port
Data
Out
64
ir0
ir1
Register File
Addr
Data
Addr
Data
Addr
Data
Addr
Data
Addr
D In
Addr
D In
25:21
20:16
25:21
20:16
15:0
Immed
15:0
Immed
Dest. reg
Dest. reg
rsv0
rtv0
rsv1
rtv1
imm0
imm1
dst0
dst1
mp1
di
alu0
addr
rtv
alu1
dst0
dst1
Mem
Port
Addr
D
In
Data
Out
alu0
md
alu1
dst0
dst1
31:24
23:16
15:8
7:0
31:24
23:16
15:8
7:0
1:0
isMem
isMem
STALL_ID_1
2:0
1:0
+
2:2
STALL-EX-1-SL
If 1, addresses are in different words, i.e., bits 31:2 of addresses differ. This is despite the fact that |imm0-imm1|<4.
=lbu
=lbu
SHARED LOAD
possible.
25:21
25:21
=
rs registers match.
Absolute difference between immediates < 4.
15:0
15:0
-
16:2
=0
=-1
1:0
=0
subtract immediate values
imm0-imm1

Problem 2: (20 pts) Show the execution of the code fragments below on the illustrated MIPS implementations. All branches are taken. Don't forget to check for dependencies.

(*a*) Show executions.



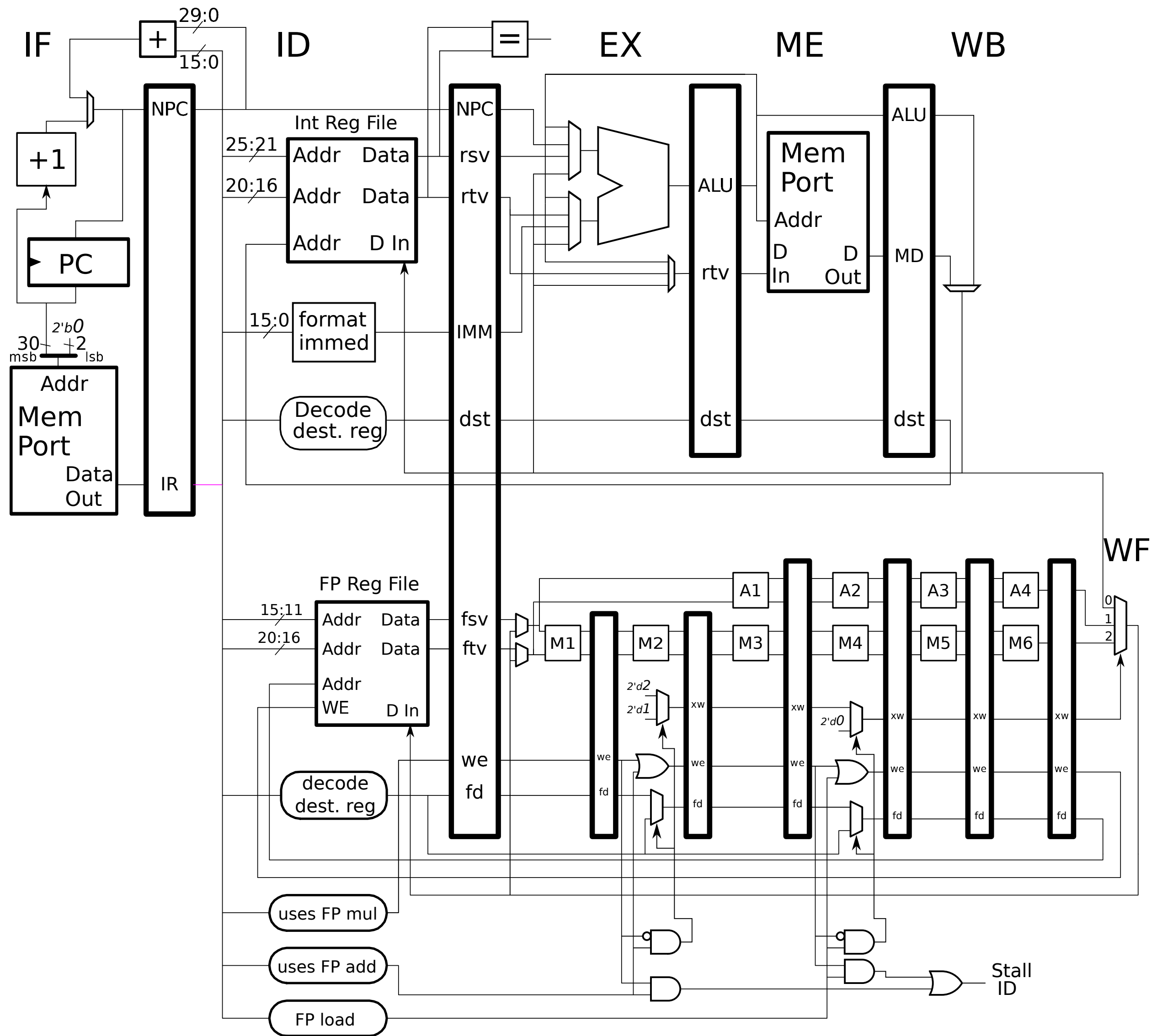
☑ Show execution of this simple code sequence.

```
# SOLUTION
# Cycle        0  1  2  3  4  5
add r1, r2, r3 IF ID EX ME WB
sub r4, r1, r5    IF ID EX ME WB
```

☑ Show execution of the following code sequence.

```
# SOLUTION
# Cycle          0  1  2  3  4  5  6
beq r1, r1 TARG  IF ID EX ME WB
or  r2, r3, r4      IF ID EX ME WB
sub r5, r6, r7
xor r8, r9, r10
TARG:
lw r10, 0(r11)         IF ID EX ME WB
# Cycle          0  1  2  3  4  5  6
```

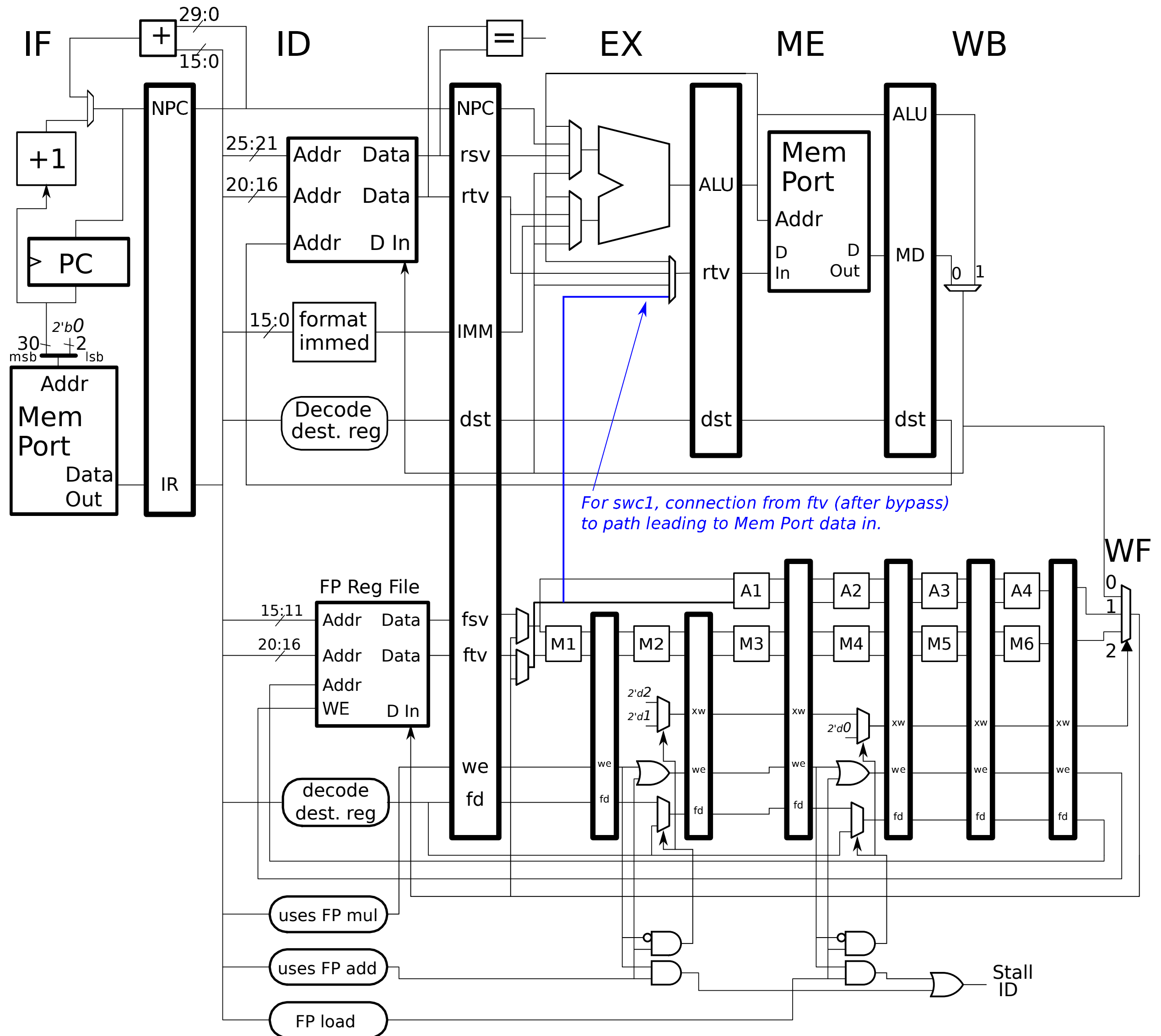
(*b*) Show the execution of the code sequences below on the illustrated MIPS implementation. Don't forget to check for dependencies.



☑ Show execution of the following code sequence.

```
# SOLUTION
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12
mul.s f5, f6, f7    IF ID M1 M2 M3 M4 M5 M6 WF
add.s f1, f2, f3       IF ID A1 A2 A3 A4 WF
sub.s f4, f8, f5          IF ID ----------> A1 A2 A3 A4 WF
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12
```

(*c*) Add any datapath hardware needed to execute the code sequence below, and show its execution. (Control logic is not needed.) Don't forget to check for dependencies.



☑ Add hardware needed by, and ☑ show execution of, the following code sequence.

☑ Consider both stores and, as always, avoid unnecessary costs.

Added hardware appears above in blue, and the execution appears below. The added hardware consists of a connection from the M1/A1 stage to the mux leading to the memory port data in. For `swc1 f1` the connection is used in cycle 3, carrying data from the `IF/ID.ftv` pipeline latch. For `swc1 f2` the connection is used in cycle 6, where it is carrying data bypassed from `WF`.

```
# SOLUTION
# Cycle              0  1  2  3  4  5  6  7  8
add.s f2, f3, f4     IF ID A1 A2 A3 A4 WF
swc1 f1, 0(r7)          IF ID EX ME WB
swc1 f2, 4(r8)             IF ID ----> EX ME WB
```

Problem 3: (20 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^{14} entry BHT. One system has a bimodal predictor, one system has a local predictor with a 10-outcome local history, and one system has a global predictor with a 10-outcome global history.

(*a*) Branch behavior is shown below. Two repetitions of a repeating sequence are shown for branches `B1` and `B2`. Branch `B2` generates the following repeating sequence: the eight outcomes `TNTNTNTN` followed by either `NN`, probability .2, or `TT`, probability .8. These last two outcomes are shown below as `r r` and `q q`.

Answer each question below, the answers should be for predictors that have already warmed up. Show work or provide brief explanations.

```
B1:   N   N   T   T   T   N   T    N   N   T   T   T   N   T ···

B2:    T   N   T   N   T   N   T    N   r   r   T   N   T   N T N T N q q ···
```

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

Branch `B1` repeats every 7 outcomes. Based on the analysis below there are 4 mispredicts and 3 correct predictions, for **an accuracy of $\frac{3}{7} = .4286$**.

```
 SOLUTION:  Analysis of counter values and outcomes for branch B1.
                                  ! <-------------------------!<--- matching ctr
    0   0   0   1   2   3   2    3    2   1   2   3   3   2   3  <- 2 bit counter
B1:   N   N   T   T   T   N   T    N   N   T   T   T   N   T    ···
                                   x   x   x           x         <- prediction outcome

B2:    T   N   T   N   T   N   T    N   r   r   T   N   T   N T N T N q q ···
```

☑ What is the accuracy of the bimodal predictor on branch B2?

Short Answer: The counter must be either 0 or ≥ 2 at the start of TNTNTNTN yielding 4 correct predictions and ending with a value of 0 (prob .2) or 2 (prob .8). Summing the weighted number of correct predictions of the next two outcomes yields $.2 \times .2 \times 2 + .2 \times .8 \times 0 + .8 \times .2 \times 1 + .8 \times .8 \times 2 = 1.52$ correct predictions. That's $4 + 1.52$ correct predictions for 10 outcomes, or 5.52% prediction accuracy.

Long Answer: This problem is easy to solve by considering a twelve outcome sequence rrTNTNTNTNqq and computing the accuracy based on the last 10 of these outcomes, TNTNTNTNqq. There are four cases to consider: rr=NN,qq=NN, rr=NN,qq=TT, rr=TT,qq=NN, and rr=TT,qq=TT. First suppose rr=NN. The counter value after the second r must be zero (since there was an N before the first r). With a counter value of zero the TNTNTNTN sequence yields four correct predictions and ends with a counter value still of 0. If the qq sequence is NN there will be two more correct predictions, for a total of 6. The accuracy is 60% and the probability of the twelve outcome sequence NN TNTNTNTN NN is $.2 \times .2 = .04$. This analysis is shown in the first row of the table below. The C columns show counter values. In the first column the counter values are shown as a range, from 0 to 2 (3 is omitted because it cannot occur at that point because the preceding outcome is an N). The last column shows the prediction accuracy weighted by the probability. The remaining three rows show the other three cases, and bottom line shows a sum of the weighted probabilities, for an overall prediction accuracy of $.552$.

```
C          C              C     C   Acc   Prob            Weighted Acc
----       ---            -     -   ---   -------------   ---------------
0-2   NN   0    TNTNTNTN  0  NN 0   60%   .2 * .2 = .04   .6 * .04 = .024
                x x x x

0-2   NN   0    TNTNTNTN  0  TT 2   40%   .2 * .8 = .16   .4 * .16 = .064
                x x x x      xx

0-2   TT   2-3  TNTNTNTN  2  NN 0   50%   .8 * .2 = .16   .5 * .16 = .08
                 x x x x     x

0-2   TT   2-3  TNTNTNTN  2  TT 3   60%   .8 * .8 = .64   .6 * .64 = .384
                 x x x x
----       ---            -     -   ---   -------------   ---------------
                                                    1.0            = .552
```

Grading Note: Most looked at cases starting with the TNTNTNTN *part and just assumed without justification a single counter value at the start this sequence.*

☑ What is the accuracy of the local predictor on B2?

The local history size is ten outcomes, and so the local predictor can see where it is in this repeating sequence. The only outcome it can't predict is the first of the two random outcomes (the second always matches the first). Since that's biased taken 80% of the time we can estimate that it will be correctly predicted 80% of the time. The other 9 outcomes will be predicted every time for an overall prediction accuracy of $\frac{9+.8}{10} = .98 = 98\%$.

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

The minimum local history size is three outcomes. Two outcomes would be too few. For example, if the local history were TT then the next outcome could either be T or N. With three outcomes we'd have TTT which can only be followed by N in the pattern above. The minimum local history size can be found methodically by constructing a tree in which an edge represents an outcome. The root is a zero length history. Edges are labeled with N or T and the possible positions (see below) at which the N or T can occur. The tree is constructed until all leaf nodes are connected to edges labeled with one position. The minimum local history size is the maximum number of edges from the root to a leaf (the tree height).

```
0   1   2   3   4   5   6  Numbering of positions.
N   N   T   T   T   N   T

N     @ 0,1,5   All length-1 histories N.  (Three positions.)
  N   @ 1       All length-2 histories NN  (One position.)  Leaf
  T   @ 2,6     All length-2 histories NT  (Two positions [of the T].)
    N @ 0       All length-3 histories NTN (One position.)  Leaf
    T @ 3
T     @ 2,3,4,6
  N   @ 5,0
    N @ 1
    T @ 6
  T   @ 3,4
    N @ 5
    T @ 4
```

☑ What is the minimum local history size needed to predict B2 with maximum accuracy?

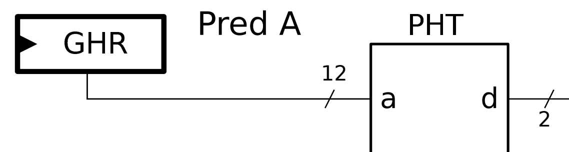
```
    B2:   TNTNTNTNrrTNTNTNTNqq
                     --------     Eight outcomes not enough. Look at q=T
                   --------       and r=N.
```

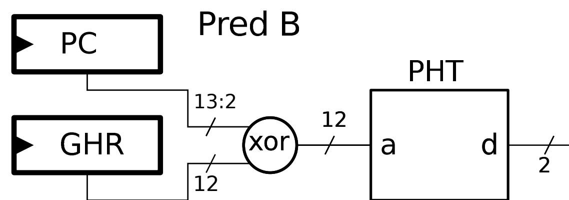
The local predictor needs to know where it is, in particular to identify the first and second random outcomes. That would require a local history size of 9. See the diagram above.

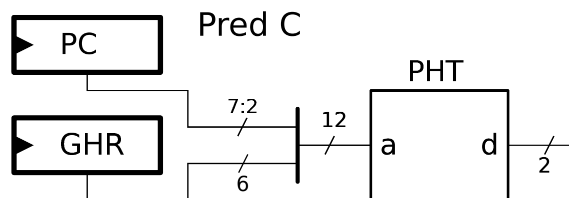
Problem 3, continued:

(*b*) The diagram below shows four branches. Branches B1 and B3 are loop branches, each in a 30-iteration loop. Branches B2 and B4 are perfectly biased branches (B2 always taken, B4 never taken). Consider their execution on the three variations of the global predictor shown below.

```
B1:   TT ··· TN                      TT ··· TN                    TT ··· TN
B2:            T                              T                             T
B3:                 TT ··· TN                      TT ··· TN
B4:                           N                              N
```







☑ Which variation (A, B, or C) will predict B2 or B4 poorly? [A boxed]

☑ Why does the variation do poorly on one or both of these simple to predict branches?

In A the PHT is indexed only by the GHR. The GHR value present when predicting B2 will be TTTTTTTTTTTN, the GHR value when predicting B4 will also be TTTTTTTTTTTN, and so they will share a PHT entry. Since they have a different bias the branch will be mispredicted.

☑ Which variation (A, B, or C) is best for B2 and B4 and also for branches with **repeating patterns**? [B boxed]

☑ Show an example that one variation predicts accurately that the other can not.

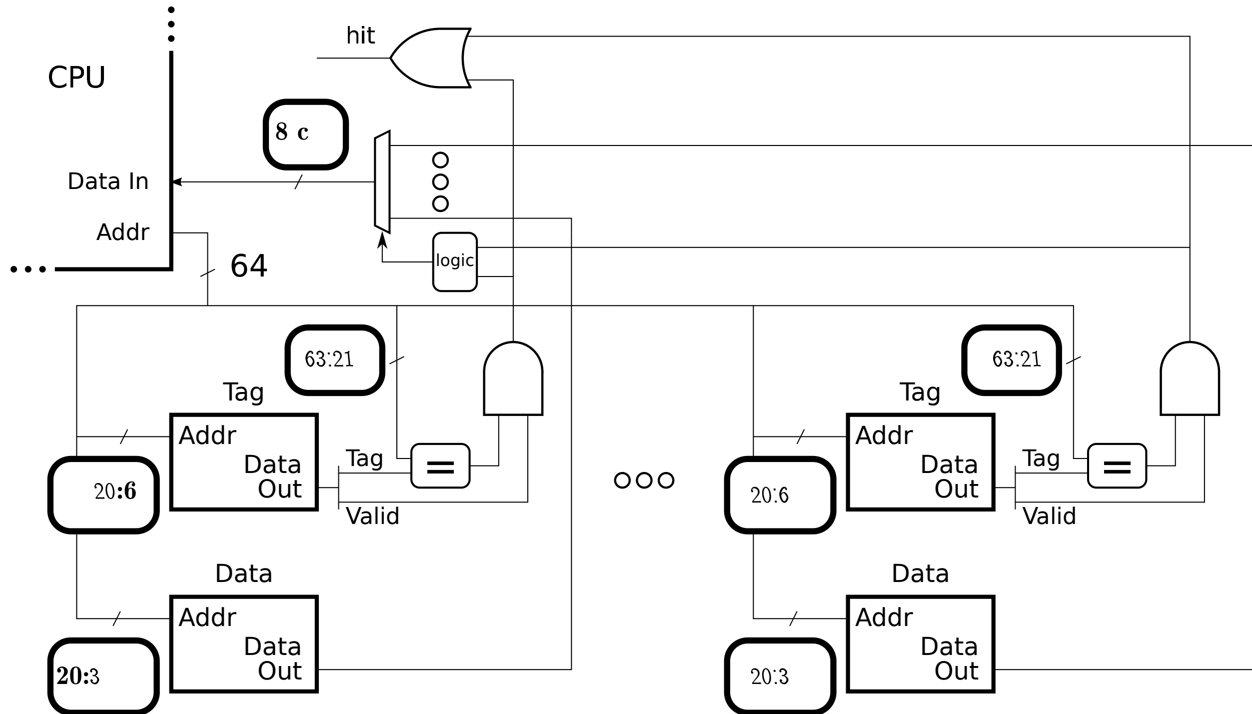
If a branch has repeating patterns then to be predicted accurately the history needs to be large enough to hold the branch's outcomes. Predictor B uses 12 bits of history, whereas Predictor C uses only 6 bits, and so Predictor B is better. Based on the first answer we know that Predictor A will not work well on B2 and B4.

Note: Predictor A is the global predictor, Predictor B is the gshare predictor, and Predictor C is called the *gselect* predictor.

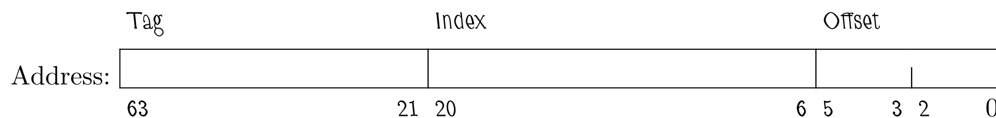
Problem 4: (20 pts) The diagram below is for a three-way set-associative cache. Hints about the cache are provided in the diagram.

(*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.)



☑ Cache Capacity ☑ Indicate Unit!!:

The cache capacity can be determined from the following pieces of information: The lowest tag bit position is 21, which means that the size of the combined index and offset is 21 bits, and so each data store holds 2^{21} characters. Nothing was said about the character size and so we can assume that it is 8 bits, which everyone alive today calls a byte. We were told that the cache is 3-way set associative, and so there are 3 data stores, and so the cache capacity is $3 \times 2^{21} = 6\,\mathrm{MiB}$.

☑ Memory Needed to Implement ☑ Indicate Unit!!:

It's the cache capacity, $6\,\mathrm{MiB}$, plus $3 \times 2^{21-6}\,(64 - 21 + 1)$ bits.

☑ Line Size ☑ Indicate Unit!!:

Lower bit position of the address going into the tag store gives the line size, $2^6 = 64$ characters.

☑ Show the bit categorization for the **smallest** direct mapped cache with the same line size and which can hold at least as much data as the cache above.

The cache above is 6 MiB, which is not a power of two. Since a direct-mapped cache must be a power of two we'll make it 8 MiB $= 2^{23}$ B. To achieve that in a direct-mapped cache of the same line size the low tag bit position must be 23, the other bit positions remain the same.

Problem 4, continued: The problems on the following pages are **not** based on the cache from Part a. The code in the problems below run on a 16 MiB (2^{24} byte) two-way set-associative cache with a line size of $2^8 = 256$ bytes. Each code fragment starts with the cache empty; consider only accesses to the arrays.

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

```
long double sum = 0;
long double *a = 0x2000000; // sizeof(long double) == 16
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.

The line size of $2^8 = 256$ bytes is given. The size of an array element, which is of type long double, is $16 = 2^4$ B, and so there are $2^8/2^4 = 2^{8-4} = 2^4 = 16$ elements per line. The first access, at `i=0`, will miss but bring in a line with 2^4 elements, and so the next $2^4 - 1 = 15$ accesses will be to data on the line, hits. The access at `i=16` will miss and the process will repeat. Therefore the hit ratio is $\frac{15}{16}$.

Problem 4, continued: The code in the problem below runs on a 16 MiB (2^{24} byte) two-way set-associative cache with a line size of $2^8 = 256$ bytes. Each code fragment starts with the cache empty; consider only accesses to the arrays.

(*c*) The code below scans through an array (or arrays) twice, once in the `First Loop` and a second time in the `Second Loop`. Three different variations are shown, Plan A, Plan C, and Plan D. For each Plan, find the largest value of `BSIZE` for which the second loop will have a 100% hit ratio.

```
// Note:  sizeof(double) == 8

// Plan A
struct Some_Struct   { double val, norm_val;   double stuff[14];   };

// Plan C
struct Some_Struct   { double val, norm_val; };
struct Some_Struct_2 { double stuff[14];};

// Plan D
double *vals, *norm_vals;
struct Some_Struct_2 { double stuff[14]; };

// Plan A and Plan C
  Some_Struct *b;
  for ( int i = 0;  i < BSIZE;  i++ ) sum += b[i].val + b[i].norm_val;   // First Loop.
  for ( int i = 0;  i < BSIZE;  i++ ) b[i].norm_val = b[i].val / sum;   // Second Loop.

// Plan D
  for ( int i = 0;  i < BSIZE;  i++ ) sum += vals[i] + norm_vals[i];   // First Loop.
  for ( int i = 0;  i < BSIZE;  i++ ) norm_vals[i] = vals[i] / sum;   // Second Loop.
```

☑ Largest `BSIZE` for Plan A. ☑ Show work or explain.

Each iteration loads one struct, size is $(1 + 1 + 14) \times 8 = 2^7$ B. There are consecutive accesses to array `b` and so the cache used efficiently (except for the fact that only 16 bytes of the 128-byte structure are used). Because the structure size is smaller than a cache line the cache can hold $2^{24}/2^7 = 2^{17}$ elements, which is what `BSIZE` should be.

☑ Largest `BSIZE` for Plan C. ☑ Show work or explain.

Size of struct (the one we need) is now $(1 + 1) \times 8 = 2^4$ B. So BSIZE is $2^{24}/2^4 = 2^{20}$. `BSIZE` is much larger because the cache holds only what we need, `val` and `norm_val`.

☑ Largest `BSIZE` for Plan D: ☑ Show work or explain.

Each iteration loads one element from `val` and one from `norm_val`. Total size is 2^4 B. Since the cache is two-way set-associative we don't need to worry about conflicts. So `BSIZE` is $2^{24}/2^4 = 2^{20}$, the same as for Plan C.

(*d*) What's wrong with the statement below:

> *In the first iteration of the First Loop under Plan C there will be one miss but with Plan D there will be two misses. Therefore Plan C is better, at least for the First Loop.*

☑ What's wrong with the statement?

What's wrong with the statement is that it is basing a conclusion about the entire First Loop on an analysis of only the first iteration of the First Loop. Yes, with C there is one miss while with D there are two. But with C there will be hits in the next 15 iterations, (line size divided by struct size) while with D there will be hits for the next 31 iterations. Overall, the number of misses is the same.

Problem 5: (20 pts) Answer each question below.

(*a*) Suppose that a new ISA is being designed. Rather than requiring implementations to include control logic to detect dependencies the ISA will require that dependent instructions be separated by at least six instructions. As a result, less hardware will be used in the first implementation.

☑ Explain why this is considered the wrong approach for most ISAs.

Short Answer: Because ISA features should be tied as little as possible to implementation features. The separation requirement in the new ISA is based on a particular implementation, one with five or six stages and that lacks dependency checking.

Long Answer: An ISA should be designed to enable good implementations over a range of cost and performance goals. It is reasonable to assume that low hardware cost was a goal of the first implementation and, as stated in the problem, the ISA's separation requirement helped achieve that goal. So the separation requirement would be a good idea **if there was only going to be one implementation or if cost constraints and technology wouldn't change**. However, ISAs are usually designed for a wide product line and a long lifetime. The separation requirement would become a burden if in the future a higher-performance, higher-cost implementation was needed.

☑ What is the disadvantage of imposing this separation requirement?

Short Answer: It would make a higher-performance implementation that included bypass paths pointless since the compiler would be forced to separate dependent instructions, possibly by inserting `nops`, and so the bypass paths would never be used.

Long Answer: Call the ISA with the separation requirement, ISA S (strict), and one that lacked the separation requirement but was otherwise identical, ISA N (normal). Consider a low-cost implementation of each ISA. Neither implementation would include bypass paths but the implementation of ISA N would need to check for dependencies and stall if any were found, making the cost slightly higher than that of the implementation if ISA S. Now consider a code fragment compiled for both ISAs in which the version for ISA S requires `nops` to meet the separation requirement. (The version for ISA N has fewer instructions.) Now consider their execution on their respective implementations. The execution times should be the same because for each `nop` in the ISA S implementation there will be a stall in the ISA N implementation.

So the performance of the low-cost implementations of the ISAs are about the same, but the cost of the ISA N version is slightly higher. (Note that the hardware for checking dependencies is of relatively low cost since it operates on register numbers, 5 bits each in many ISAs. That is in contrast to the cost of the hardware for bypasses, which are 32 or 64 bits wide and include multiplexors.)

Next, consider high-performance implementations. For ISA N we can add bypass paths, as we've done in class. As a result, some instructions that would stall in the low-cost implementation would not stall in the high-performance implementation, and so code would run faster. In contrast, code for ISA S would still have `nop` instructions since the ISA itself imposes the instruction separation requirement. That would make it much harder to design higher performance implementations.

Therefore, the disadvantage of the instruction separation requirement is that it leads to much higher-cost high-performance implementations with only a small cost benefit for the low-cost implementation.

Grading Note: Several students incorrectly answered that a disadvantage of the separation requirement is that it would be tedious and error prone for hand (human) coding, and that it would require that the compiler schedule instructions rather than having the hardware check for dependencies and stall when necessary.

It is 100% true that hand-coding assembly language with such a requirement would be tedious, especially considering branch targets. However, an ISA is designed to facilitate efficient implementations, not to improve assembly language programmers' productivity. (An assembler can still be helpful by pointing out likely separation issues.) Also, if something can be done equally well in the compiler and in hardware, it should be done in the compiler because the cost of compiling is borne beforehand, by the developer. In contrast, the higher cost of the hardware is payed by every customer and the energy of execution is expended each time the program is run.

(*b*) In the execution below the `add.s` instruction encounters an arithmetic overflow when in the `A4` stage and as a result raises an exception in cycle 6.

```
# Cycle              0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
add.s  f1, f2, f3   IF ID A1 A2 A3 A* WFx
lwc1 f3, 0(r2)         IF ID EX ME WB
lw r4, 0(r5)              IF ID EX ME WBx
add r6, r7, r8               IF ID EX MEx

HANDLER:
sw                                     IF ID ..
# Cycle              0  1  2  3  4  5  6  7  8  9  10 11 12 13 14 15 16 17 18
```

☑ Explain why this exception can't be precise.

Because the `lwc1` wasn't squashed and it wrote a register. To be precise nothing after the faulting instruction (`add.s`) can modify registers or memory.

☑ Describe something the handler could do if the exception were precise.

If the exception were precise then after the handler was finished it could return to the `add.s` giving it a second chance to execute. It can't return to the `add.s` in the imprecise execution shown above because the value of `f3` was overwritten by the `lwc1`.

(*c*) Chip A has 60 2-way superscalar cores and Chip B has 20 4-way superscalar cores. Both chips run at 3 GHz, and so Chip A has a higher peak instruction throughput. The cost and power consumption of the two chips are identical. Why might someone prefer Chip B over Chip A?

☑ Chip B preferred, despite lower peak throughput, because:

Because their workload consists of one single-threaded program. (Or maybe a program that has no more than 20 threads.) It will run faster on Chip B.

(*d*) The SPECcpu suite is used to test new chips that might have cost hundreds of millions of dollars to develop, and test results are closely watched.

☑ Why might we trust the SPECcpu selection of benchmarks and rules for building and running the benchmarks?

Because major chip makers are members of SPEC. Company X would not be a member of SPEC if Company Y twisted the benchmark selection and rules in favor of Company Y's products.

☑ Why might we trust the SPECcpu scores that Company X publishes?

Because any published score is part of a *disclosure* that includes a config file. With the config file anyone can run the test in exactly the same way as Company X does. Anything used in the test must be publicly available.