Name Solution

Computer Architecture
EE 4720
Final Examination

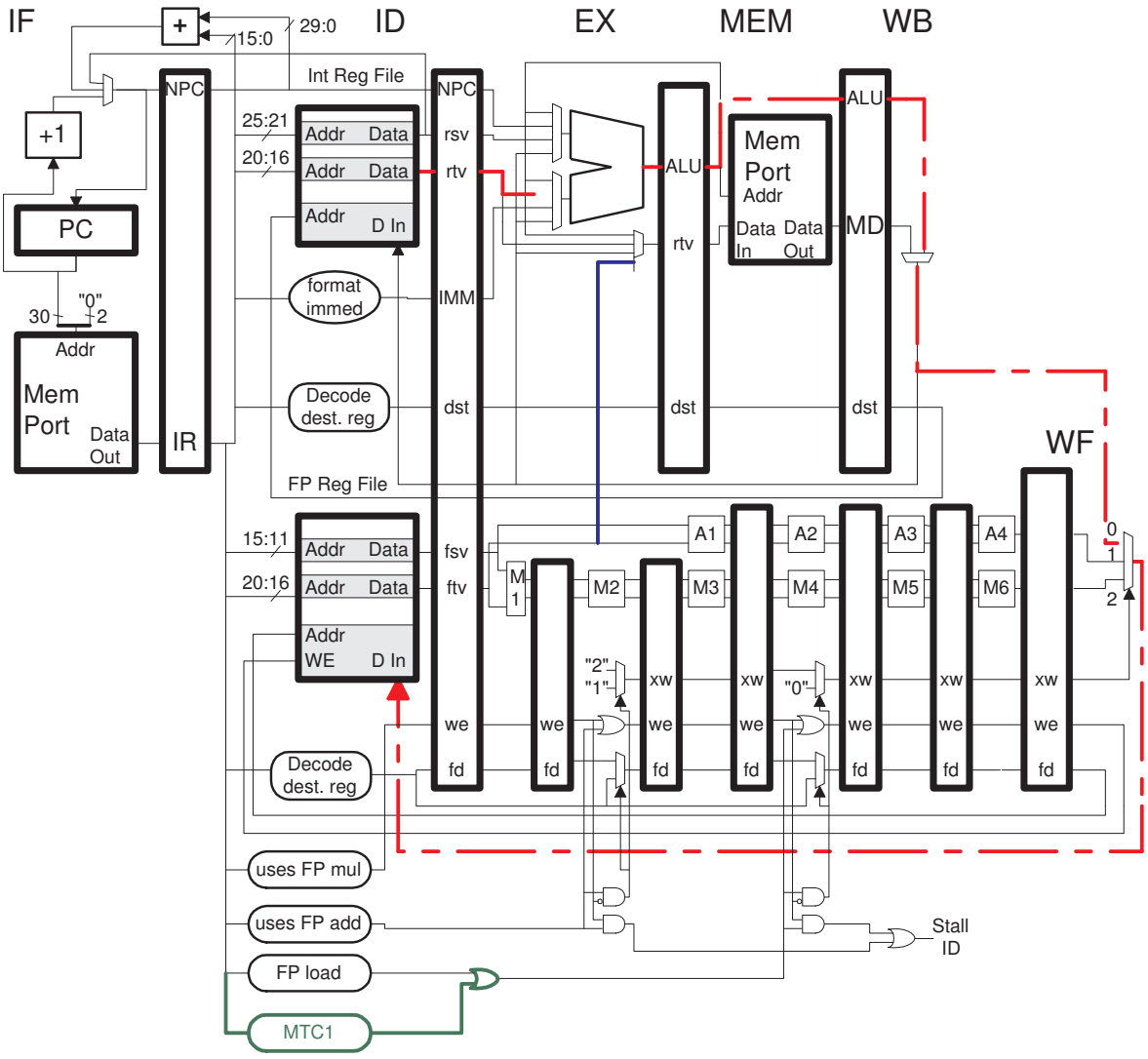
11 May 2010, 12:30–14:30 CDT

Problem 1		(15 pts)
Problem 2		(20 pts)
Problem 3		(10 pts)
Problem 4		(15 pts)
Problem 5		(15 pts)
Problem 6		(10 pts)
Problem 7		(15 pts)
Exam Total		(100 pts)

Alias A(H1N1) $\cdots$ chooo!

Good Luck!

Problem 1: (15 pts) The statically scheduled MIPS implementation including the floating-point pipeline is illustrated below.



(*a*) Consider the instruction `mtc1 f2, r4`. On the diagram above show the path taken by the data on its trip from `r4` to `f2`.

☑ Show path taken by value using a squiggly line on the diagram above.

The path is highlighted in red with dashes instead of squiggles. Note that `mtc1` uses the `rt` field for the integer register. The ALU will set its output to the value at its lower input.

(*b*) The control logic for the FP pipeline needs only a small change to handle `mtc1`. Make that change above. (This has nothing to do with the bypass problem below.)

☑ Control logic for `mtc1` in diagram above. (Ignore bypasses.)

Change appears in green. Both the `lwc1` and `mtc1` instructions take a value from the integer pipeline and write it to the FP register file in the integer WB stage. So for detecting `WF` structural hazards the same logic can be used with the addition of a box for detecting the `mtc1` opcode.

(*c*) Add the hardware needed to implement `swc1`. Add only datapath, not control logic.

☑ Datapath for `swc1`.

The added hardware will provide a way for a value read from the FP register file to hop over to the "Data In" connection on the ME-stage memory port. The EX stage is the critical-path friendly place to do that, the change appears in blue.

Problem 1, continued:

```
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13
lui r1, 0x4593      IF ID EX ME WB
ori r1, r1, 0x819c     IF ID EX ME WB
mtc1 f1, r1               IF ID EX ME WF
add.s f2, f2, f1             IF ID A1 A2 A3 A4 WF
mtc1 f4, r4                     IF ID EX ME WF
sub.s f6, f4, f1                   IF ID A1 A2 A3 A4 WF
swc1 f6, 0(r5)                        IF ID -------> EX ME WF
# Cycle             0  1  2  3  4  5  6  7  8  9  10 11 12 13
```

(*d*) The code fragment execution (pipeline diagram) above could not occur on the pipeline above because certain bypass paths are needed.

☑ Add those bypass paths for the code above.

The bypass paths appear in blue in the diagram below.

☑ Show the cycle in which each added bypass path is used by the code above.

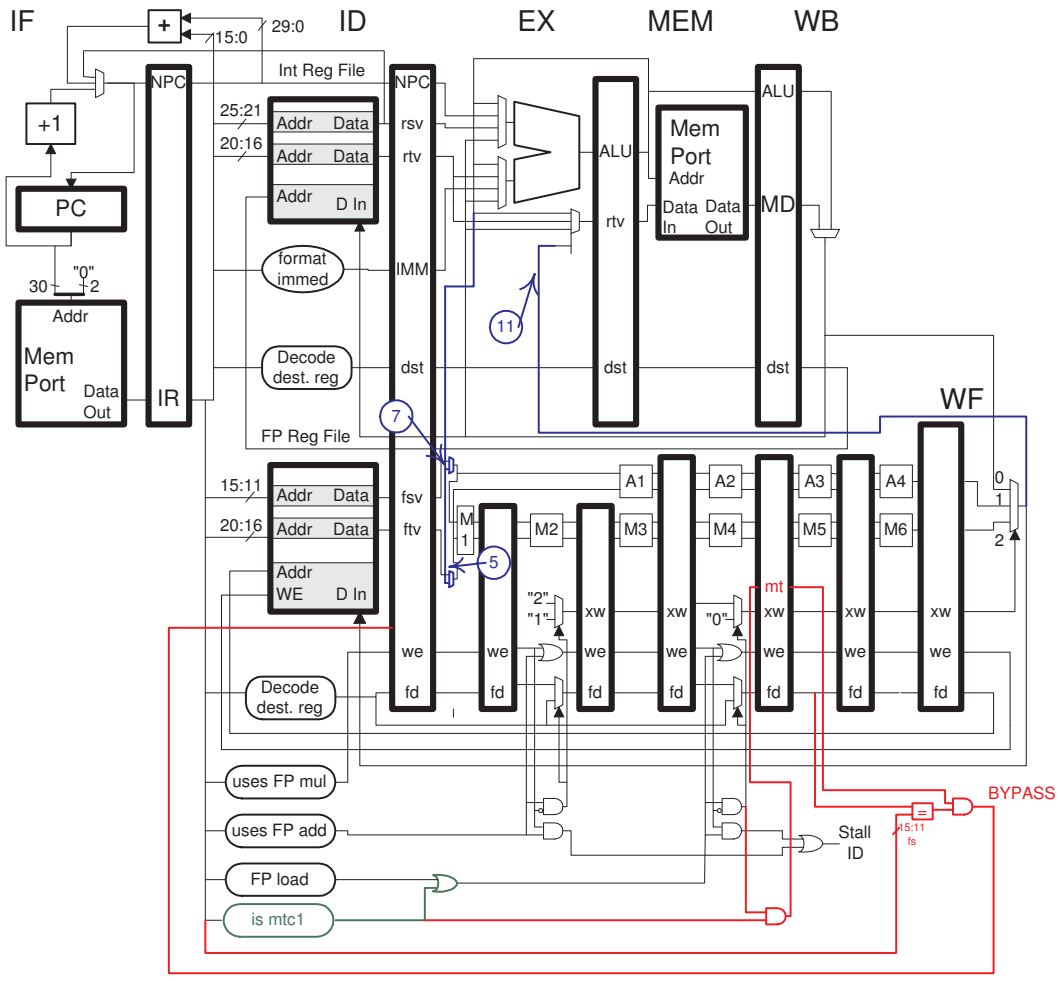
See the blue circles in the diagram below.

(*e*) Add control logic needed to detect the bypass used from `mtc1` to `sub.s`. The logic should deliver a signal, `BYPASS`, to the stage containing the bypass multiplexors. The `BYPASS` signal should be true if the bypass is needed.

☑ Logic generating `mtc1` to `sub.s` bypass signal.

Solution appears in red below; the green changes are from part (b). The dependence is detected by the [=] box, comparing the **`fs`** source of the instruction in `ID` (register `f4` of the **`sub.s`**, compared in cycle `6`) to the destination of whatever instruction is two stages from `WF`, that would be the **`mtc1`** in the example (also during cycle `6`). The comparison unit output is connected to an AND gate which makes sure that there is an **`mtc1`** two stages from `WF`. (Just checking **`we`** would tell us that something is two stages from `WF`, but for our purposes we need to know whether it's an **`mtc1`** because that is the only instruction that can use the bypass path.) The problem just asked for a signal named BYPASS, not that it be connected to anything, but the solution goes a bit further showing that BYPASS is connected to the `ID/EX` latch. Connecting BYPASS directly to the added **`fsv`** multiplexor **would be wrong**.

Diagram on next page.


IF
ID
EX
MEM
WB
WF
+
+1
PC
NPC
IR
Mem Port
Addr
Data Out
29:0
15:0
Int Reg File
25:21
20:16
Addr Data
Addr D In
format immed
Decode dest. reg
FP Reg File
15:11
Addr WE D In
rsv
rtv
IMM
dst
fsv
ftv
we
fd
ALU
Mem Port
Data In
MD
M1
M2
M3
M4
M5
M6
A1
A2
A3
A4
xw
"2"
"1"
"0"
mt
BYPASS
Stall ID
uses FP mul
uses FP add
FP load
is mtc1
11
7
5

Problem 2: (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 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.

Branch `B1` is random, and can be described by a Bernoulli random variable with $p = .5$. The outcome of branch `B3` will always be the same as the most recent outcome of branch `B1`. (That is, if an execution of `B1` is taken, the next execution of `B3` will be taken.) Branch `B2` has a repeating pattern, it repeats twice below.

```
B1: r   r       r           r               r   r       r           r
B2:  T   N   T   N   N   T   N   N   N   T    T   N   T   N   N   T   N   N   N   T
B3:   R       R           R               R    R       R           R               R
```

For the questions below accuracy is after warmup.

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

The 2-bit counter values are shown below. Note that the same counter value is present at the beginning and end, so the first occurrence of the pattern can be used to compute the prediction ratio. That would be an unimpressive $\boxed{\frac{4}{10} = .4}$ (40%).

```
# SOLUTION Work
    1  2   1   2   1   0   1   0   0   0  1    <-- Counter values
B2:  T   N   T   N   N   T   N   N   N   T    T   N   T   N   N   T   N   N   N   T
     x   x   x   x       x               x     <-- Prediction outcome.
```

☑ What is the accuracy of the bimodal predictor on `B3`?

The branch is random and uncorrelated with itself so there is no way prediction accuracy can be anything other than $\boxed{.5}$.

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

The pattern length is ten outcomes, which can easily fit within the 16-outcome local history, and so the accuracy will be $\boxed{1}$.

☑ How small can the local history size be made without affecting the accuracy of branch `B2`? Explain.

The local history can be as small as $\boxed{\text{5 outcomes}}$. A local history size of four would be too small, in that case local history pattern NTNN could be followed by both a T or N outcome, precluding 100% accuracy. The table below shows all of the five-outcome patterns that are possible (sorted and original order), followed by the outcome. We know that five are sufficient because each distinct history is followed by only one possible outcome.

```
---Sorted-----        -- Original Order--
Hist   Outcome        Hist   Outcome
NNNTT  N              TNTNN  T
NNTNN  N              NTNNT  N
NNTTN  T              TNNTN  N
NTNNN  T              NNTNN  N
NTNNT  N              NTNNN  T
NTTNT  N              TNNNT  T
TNNNT  T              NNNTT  N
TNNTN  N              NNTTN  T
TNTNN  T              NTTNT  N
TNTNN  T              TTNTN  N
TTNTN  N              TNTNN  T
```

☑ What is the accuracy of the local predictor on `B3`?

The local predictor can only "see" B3 and so is just as helpless as the bimodal predictor, the accuracy will be .5.

☑ What is the accuracy of the global predictor on B3?

Branch B3 is perfectly correlated with B1, and the global predictor can "see" B1, thanks to the generous 16-outcome global history register. So after warmup accuracy will be 100%.

☑ How small can the global history size be made without affecting the accuracy of branch B3? Explain.
Five outcomes.

Five is the size needed to hold the `r N N N T` pattern, which has the longest separation between B1 and B2.

☑ How many PHT entries are used by B2 in the system using the local predictor?

Not counting warmup, ten entries. During warmup up to 15 additional patterns would be used for the first 15 times branch B2 was encountered. What those 15 patterns are is determined by the contents of the local history field in the BHT entry for B2 before B2 was first executed. That local history field might contain the outcomes for some aliased branch, or else whatever values were set when the power was turned on (random, or perhaps intentionally initialized to 0).

☑ What is the warmup time of the global predictor on branch B3?

There are four patterns of GHR value that will be present when predicting B3, for example, `TRrNTRrNNTRrNNNT`. The `r` and `R` in this pattern can be `T` or `N`. If B1 and B3 were not correlated then the six arrs could have $2^6 = 64$ possible values. But they are and so the big R will always have the same outcome of the little r to its left. There are two such pairs in the pattern, and two unpaired ars to the total number of instances of this pattern is $2^4 = 16$. Now, assuming that there are 16 instances of each of the remaining three patterns there would be a total of 64 possible GHR values and so 64 PHT entries would be used, each of which requires two updates to warm up in the worst case. The warmup time would then be at least 128 executions of B3.

Problem 3: (10 pts) Suppose that in a MIPS-I system using a bimodal predictor there were BHT collisions on 5% of the predictions. (A BHT collision occurs when two branches use the same BHT entry.) A BHT entry stores both a 2-bit counter and the branch's 16-bit displacement. *Grading Note: The contents of a BHT entry was not in the original exam.*

Consider a design alternative in which a tag were used to detect BHT collisions, in the same way a cache uses a tag to detect hits (or misses).

(*a*) How large would the tag have to be to perfectly detect misses on a MIPS-I system using a 2^{14}-entry BHT?

☑ Tag size needed, reason:

The BHT is indexed using the low bits of the branch PC, omitting alignment bits of course. That would be bits 15:2 in this case. The higher bits, 31:16 would form the tag, and so that tag size would be 16 bits.

(*b*) Suppose that the storage budget for the BHT was fixed at the number of bits in a 2^{14}-entry BHT without tags.

☑ About how many entries would there be in a BHT with tags?

Two solutions appear below. One is for a BHT entry that only stores a 2-bit counter. That was what the original question implied. The second is for a BHT that stores the branch displacement.

Solution Ignoring Displacement: The existing BHT uses $2^{14} \times 2 = 2^{15}$ bits. If the BHT held the tag, not just the two bit counter, its capacity would be $2^x \times 18 \approx= 2^{x+4}$ bits, where 2^x is the number of BHT entries. Solving $2^{x+4} = 2^{15}$ for x yields $x = 11$, so the tagged BHT would have 2048 entries, one eighth its original size. The net result would be an increase in collisions, but now at least we can detect those collisions. *Note: That last sentence needs to be read with an air of foolish confidence.*

Solution Using Displacement: An existing BHT entry requires 18 bits, 16 for the displacement and 2 for the counter. Also including the tag would nearly double the size. To keep the costs equal the number of entries in the tagged BHT would have to be halved. Therefore, The tagged BHT would have $2^{13} = 8192$ entries.

(*c*) Based on the answer to the previous part, could we use a tagged BHT if the benefit of detecting collisions were small, moderate, or large?

☑ Benefit needs to be small, medium, or large. Explain.

Solution Ignoring Displacement: With a factor of eight reduction in BHT size the benefit of detecting a collision needs to be large. Extra large, maybe.

Solution Using Displacement: Since the BHT is half the size we would expect more collisions, exactly how much more is hard to say. Therefore there would have to be at least a moderate benefit in doing so.

(*d*) What is the benefit of detecting BHT collisions on a four-way superscalar statically scheduled MIPS implementation?

☑ Benefit for 4-way MIPS:

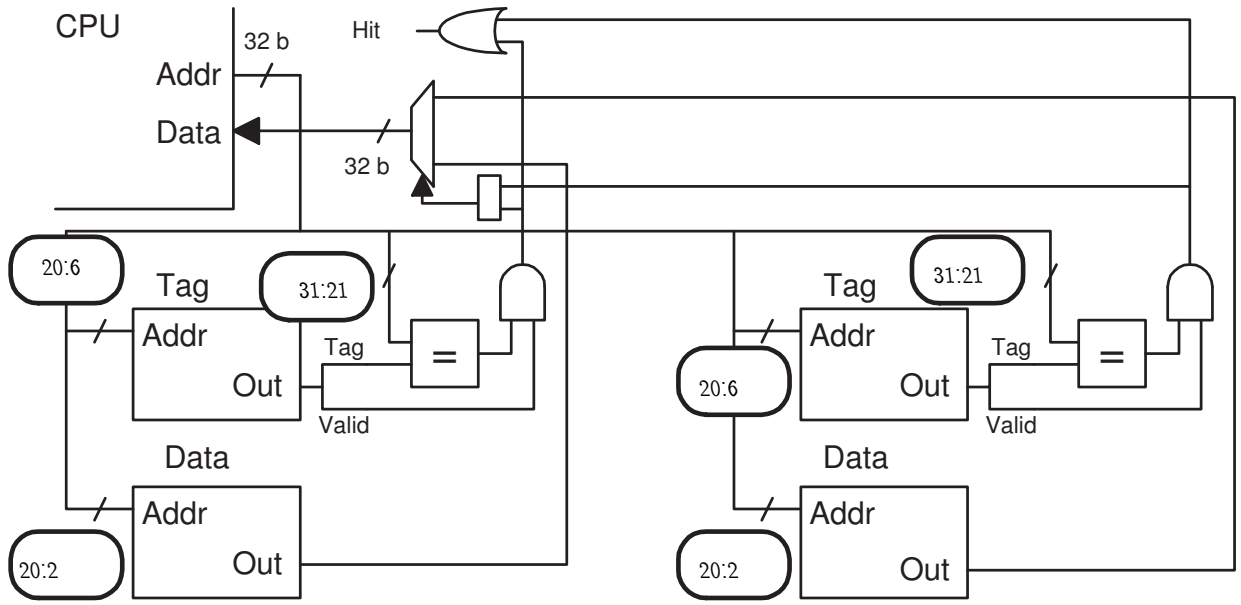
Solution Ignoring Displacement: Here we are assuming that a correct branch target is available despite the BHT collision. There is no benefit at all because knowing that there is a collision does not help in making a prediction. It's better to use an unreliable prediction that might be right than to stall the pipeline until the branch resolves (thereby guaranteeing performance loss).

Solution Using Displacement: If there is a BHT collision then the branch displacement is likely wrong. Therefore, predicting the branch taken is likely going to result in a target misprediction. Therefore, on a collision we should predict the branch not taken, which might have just a .5 chance of being correct, but that's better than predicting taken and having a nearly certain chance of getting the target wrong.

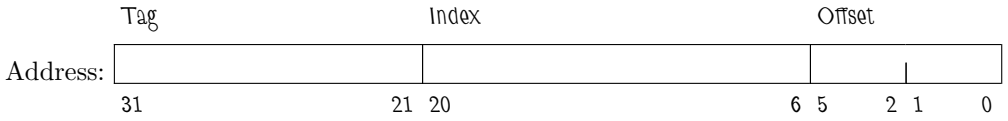
Problem 4: (15 pts) The diagram below is for a set-associative cache with a line size of 64 bytes and a tag size of 11 bits. The system has the usual 8-bit characters.

(*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.)



☑ Cache Capacity (Indicate Unit!!):

The cache capacity can be determined from the following pieces of information: the 11-bit tag (given explicitly in the first paragraph), the 2-way associativity (by noting the diagram has 2 "ways"), and the 32-bit address space (from the number of address bits shown in the diagram near the upper-left). From the 11-bit tag and 32-bit address space we know that the low tag bit is $32-11=21$, and so the capacity of a way is 2^{21} bytes or 2 MiB. Since the cache is 2-way the **total capacity is $2^{21} \times 2 = 2^{22}$ characters or 4 MiB**.

☑ Associativity:

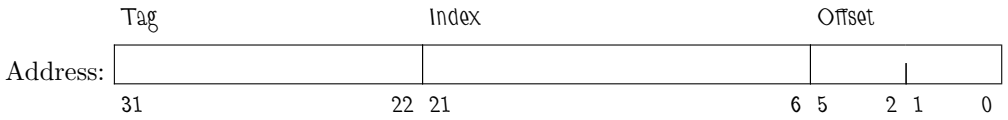
Associativity is 2

by inspection of the diagram.

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

It's the cache capacity, 2^{22} characters, plus $2 \times 2^{21-6}\,(32-21+1)$ bits.

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


Tag
Index
Offset
Address:
31
22
21
6
5
2
1
0

Problem 4, continued:

(*b*) The code below runs on a 32 MiB direct-mapped cache with a 256-byte line size. Initially the cache is empty; consider only accesses to the array.

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

```
double sum = 0.0;
double *a = 0x2000000; // sizeof(double) == 8
int i;
int ILIMIT = 1 << 11;    // = 2^11

for (i=0; i<ILIMIT; i++) sum += a[ 4 * i ];
```

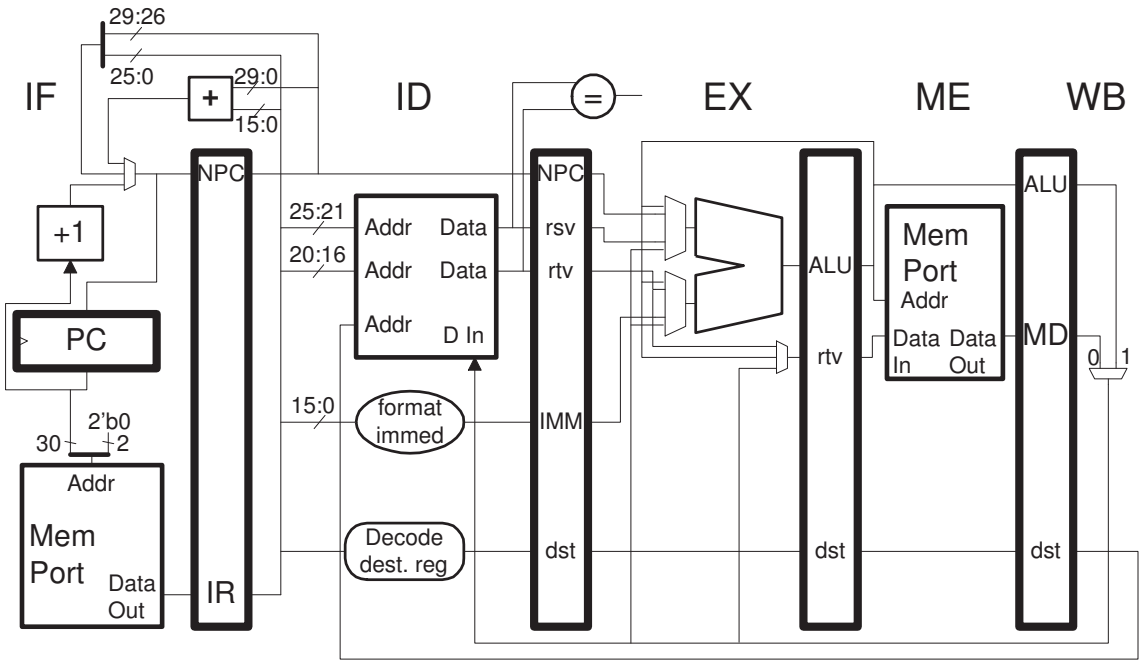
The line size of 2^8 characters is given, the size of an array element is $8 = 2^3$ characters, and so there are $2^{8-3} = 2^5$ elements per line. However, because the array index is multiplied by four only every fourth element is read, or $2^5/4 = 2^3$ elements per line. The first access, at `i=0`, will miss but bring in a line with 2^5 elements, the next $2^3 - 1$ accesses will be to data on the line, but eighth access after the miss will miss again. Therefore the hit ratio is $\frac{7}{8}$.

Problem 5: (15 pts) Several possible new MIPS instructions appear below. Show how each instruction can be encoded and show datapath changes needed to implement the instruction.

- The changes cannot break existing instructions.
- The changes can not have a large impact on clock frequency.
- A shift unit is present, but not shown.

Note: The original exam only asked for a summary of datapath changes, and did not mention the shift unit or clock frequency.

Also indicate the relative difficulty of implementing the instruction. If an instruction is deemed moderate or difficult indicate the most important reasons why.



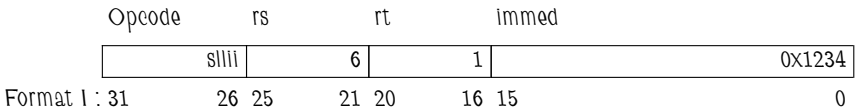
Continued on next page.

(*a*) The `sllii` instruction is like an `lui` except it can shift the immediate by any amount. For example, `sllii r1, 0x1234,16` would be equivalent to an `lui r1, 0x1234`.

```
sllii r1, 0x1234, 6   ! r1 = 0x1234 << 6
```

☑ Show possible encoding (instruction format (R, I, etc) and field (rs, rt, etc) usage):

Because it has an immediate and register it would have to be format I. The `rs` field will be used to hold the shift amount, leaving the role of the `rt` field unchanged.

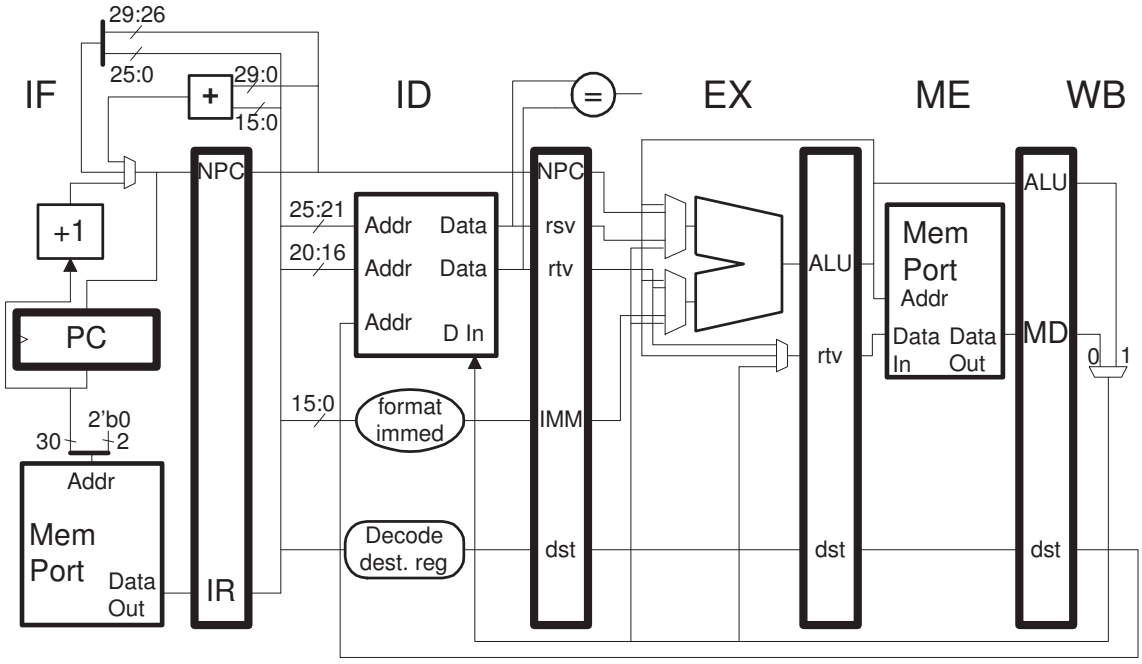


☑ Easy, moderate, or difficult to implement:

☑ Show datapath changes.

This should be easy because there already is a shift unit (for the `sll` and friends). For the new instruction the "shiftee" input to the shift unit will also have to connect to the `ID/EX.IMM` latch, and the shift amount input would have to connect to a new `ID/EX.rs` latch (not to be confused with `ID/EX.rsv`).

Problem 5, continued:

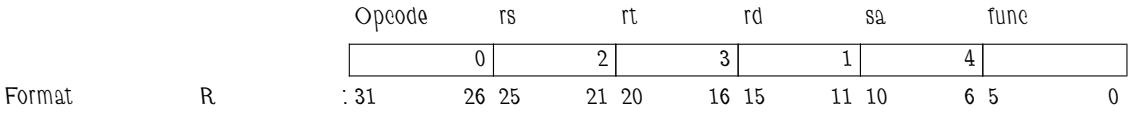


(*b*) The `adds` (add scaled) can shift the second operand left by any amount.

```
adds r1, r2, r3, 4   ! r1 = r2 + ( r3 << 4 )
```

☑ Show possible encoding:

The encoding is easy, just put the shift amount in the `sa` field. In fact, there would be no reason to choose a new `func` field value because `add` is ordinarily defined with an `sa` field value of 0.



☑ Easy, moderate, or difficult to implement:

☑ Show datapath changes.

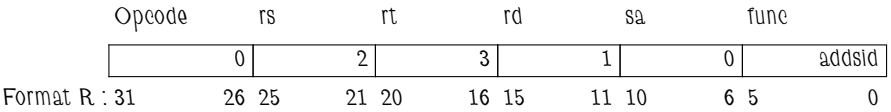
This would be moderately difficult to implement because the shift needs to be performed before the ALU operation. Since arbitrary shifts cannot be done quickly, a stage would have to be added between `ID` and `EX`.

(*c*) The `addsid` instruction produces a sum like `add` but one operand is obtained from memory.

```
addsid r1, r2, (r3)   ! r1 = r2 + Mem[r3]
```

☑ Show possible encoding:

The encoding here is also easy. Make the memory address register `rt`.



☑ Easy, moderate, or difficult to implement:

☑ Show datapath changes.

This is difficult because the memory port is in the stage after the ALU. There are no reasonable ways to get the value read from memory back to the ALU. Here are some unreasonable ways:

Add a new stage between ID and EX, and put a memory port there. This isn't a good idea because memory ports are expensive.

Add a new stage, call it MS (memory source). Unlike the solution above, the MS stage will share a memory port with ME. This will require a multiplexor on the memory port address input, as well as control logic to detect the structural hazard. Further, it will likely strain critical path since it's the memory port that is hardest to squeeze into one cycle.

Add a new stage, EX2, after ME, the new stage will have an ALU to do any operations that could not be done in EX. Besides the cost, there will be bypass issues with this solution.

Problem 6: (10 pts) Consider a single four-way superscalar implementation and a chip with four scalar implementations (like our five-stage MIPS). Both are statically scheduled and have similar features.

☑ Why might the clock frequency of the superscalar system be lower than the scalar systems?

Bypass paths are the main culprit in several ways. First, physical distances are larger and so there will be higher propagation delays. Bypass multiplexors will also have more inputs, $2n$ inputs each for an n-way superscalar. If control logic is placed in ID then that should not have an impact on clock frequency for something as small as a 4-way system.

☑ Why might the chip area (or cost) of the four scalar systems be less than the superscalar system?

An n-way, 5-stage superscalar might have about $2n^2$ bypass paths, whereas n scalar systems would just have $2n$ paths. For that reason the scalar systems would cost less. There might be some savings with the superscalar systems, such as fewer memory ports or floating-point units.

☑ How does the average program run less efficiently on the superscalar system than on one of the scalar systems? (Less efficiently means more stalls and squashes.)

There are only a few instruction pairs that can cause the scalar system to stall, such as a load followed by an instruction that uses the loaded value. There are no stalls for consecutive dependent arithmetic instructions, and no stalls or squashes for taken branches (assuming the needed bypasses are present). In contrast, dependent instructions in the same group (fetched together) on the superscalar system will cause a stall. For that reason, execution will be less efficient.

☑ Given the answers above, why does a typical chip have two four-way superscalar implementations rather than eight scalar implementations? (Please do not confuse *eight scalar implementations* with *one eight-way superscalar system.*)

Lets consider the simpler tradeoff of one 4-way superscalar vs. four scalar processors. If a program could be perfectly parallelized (achieve linear speedup) then it would execute faster with four scalar processors because their clock frequency would be higher and because there would be fewer stalls. However, writing parallel code is often difficult, for some classes of programs very very difficult. The 4-way superscalar is faster for the programs you already have. (Beyond 4-way efficiency drops sharply, which is why chips have several 4-way cores.)

Problem 7: (15 pts) Answer each question below.

(*a*) One use for exceptions is to implement rare or specialized instructions in software (called *emulation*). One example is the SPARC quad-precision arithmetic instructions, such as faddq f4, f8, f12 (floating-point add quad). No existing SPARC implementation can execute these instructions in hardware.

☑ Why would `faddq` have to raise a precise exception in order to be emulated?

The emulation code will reproduce what the `faddq` was supposed to do, meaning it will read the source registers, compute a 128-bit floating-point sum, and write that sum to the destination registers. It will then resume execution at the instruction following `faddq`. To do that the exception handler (which calls the emulation code) must be reached as though execution jumped to the handler just before the `faddq` (the precise exception definition). If the exception were not precise then execution might have reached several instructions after `faddq`, perhaps overwriting one of its source registers, making it impossible to emulate, and anyway making it impossible to return to the instruction after `faddq` (which would then execute a second time).

(*b*) Consider an implementation similar to our pipelined MIPS in which a floating-point overflow on a `fmuld` did not raise a precise exception (because the hardware to do so would not be worth the trouble).

☑ Does that mean it would not be possible for the `faddq` to raise the precise exception needed for emulation? Explain.

No, because the `faddq` would raise an illegal instruction exception, which is detected early enough to be precise.

(*c*) Answer the following questions about the SPECcpu benchmarks.

☑ Why are branches in the FP suite easier to predict than branches in the integer suite?

Because many programs in the FP suite perform scientific calculations, which typically operate on large sets of uniform data. Loops have many iterations, making them easier to predict.

☑ Why is it important that the source code is available for the SPECcpu benchmarks?

SPECcpu is intended to measure the potential of new ISAs and new implementations. The tester compiles the source code for the particular implementation being tested, presumably in such a way that brings out the tested system's full potential. If source code were not included the suite would have to provide pre-built executables. These could not be used to test new ISAs since SPEC wouldn't have the tools to build code for them (because they are new). New implementation might benefit from new compilers, which SPEC could also not be expected to have.

(*d*) Answer each ISA question below:

☑ Describe a feature of VLIW ISAs that distinguishes them from RISC and CISC ISAs.

Instructions are handled in groups called bundles. A bundle includes several (usually 3) instructions plus a field (sometimes called a template) providing dependency and other information.

☑ Describe a feature of CISC that distinguishes it from RISC.

Instructions can vary in length. Memory access is not restricted to load and store instructions. For example, an arithmetic instruction can load operands from memory and write a result to memory.