Name _____

Computer Architecture

EE 4720

Final Examination

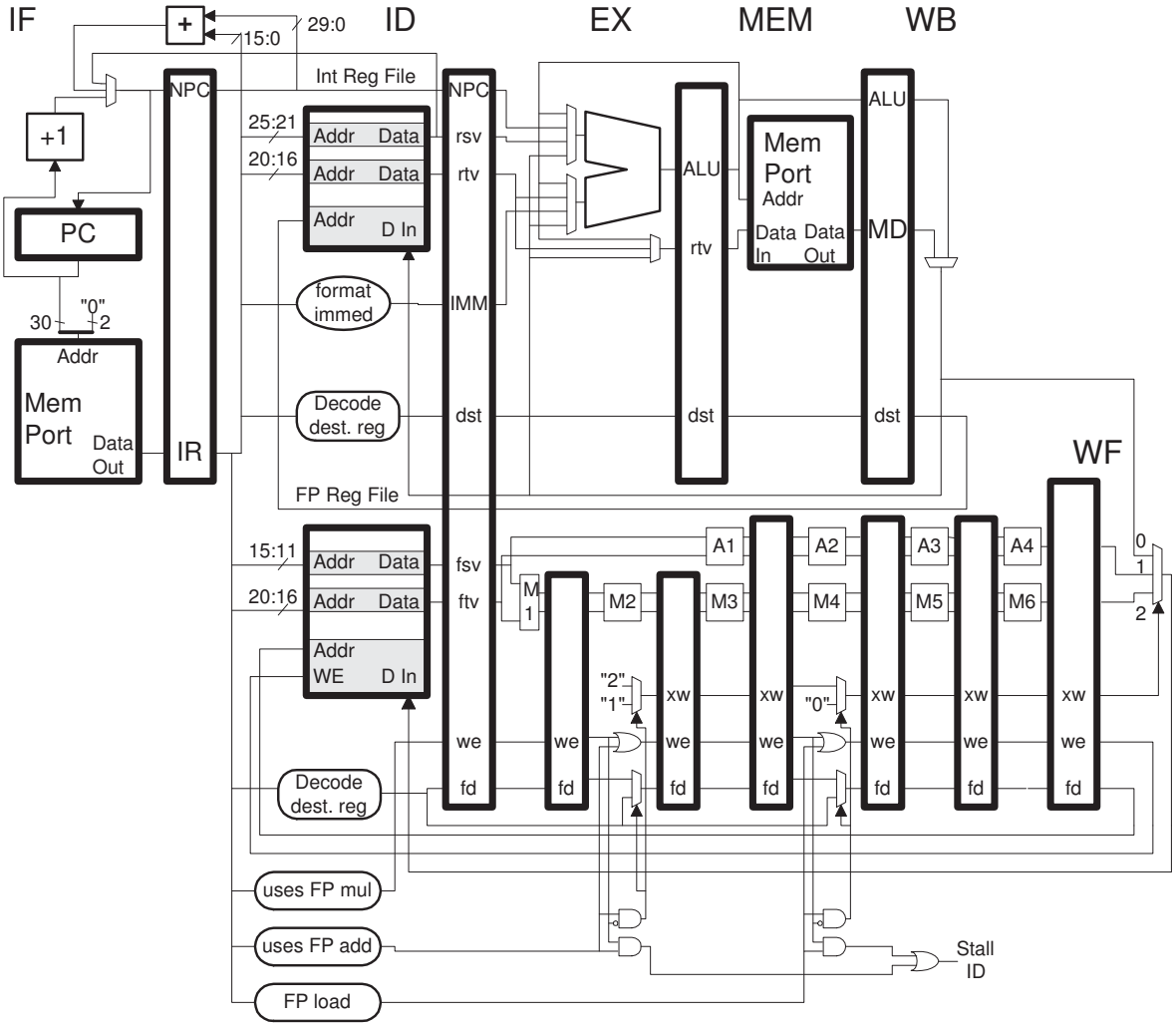
11 May 2010, 12:30–14:30 CDT

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 _____

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.

(*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.)

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

☐ Datapath for `swc1`.

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
```

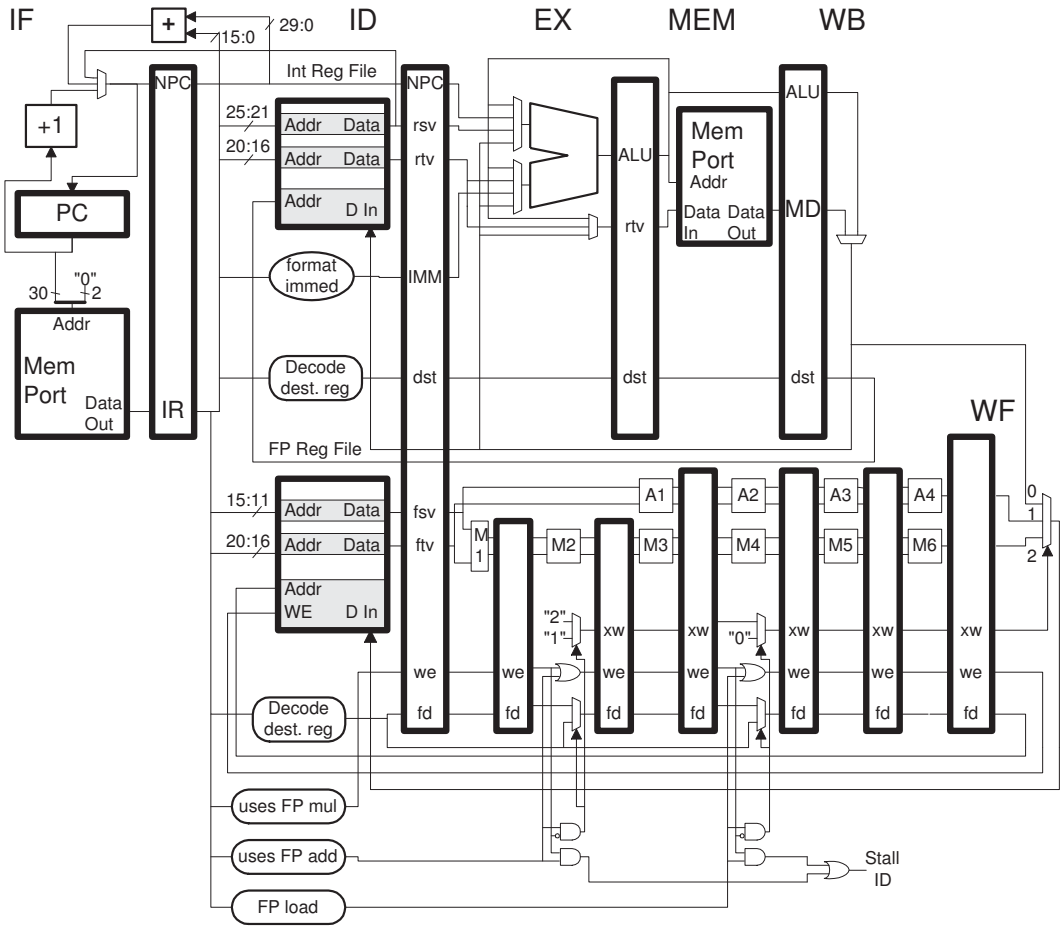
(*d*) The code fragment execution (pipeline diagram) above could not occur on the pipeline above because certain bypass paths are needed.

☐ Add those bypass paths for the code above.

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

(*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.



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`?

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

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

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

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

☐ What is the accuracy of the global predictor on `B3`?

☐ How small can the global history size be made without affecting the accuracy of branch `B3`? Explain.

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

☐ What is the warmup time of the global predictor on branch `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:

(*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?

(*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.

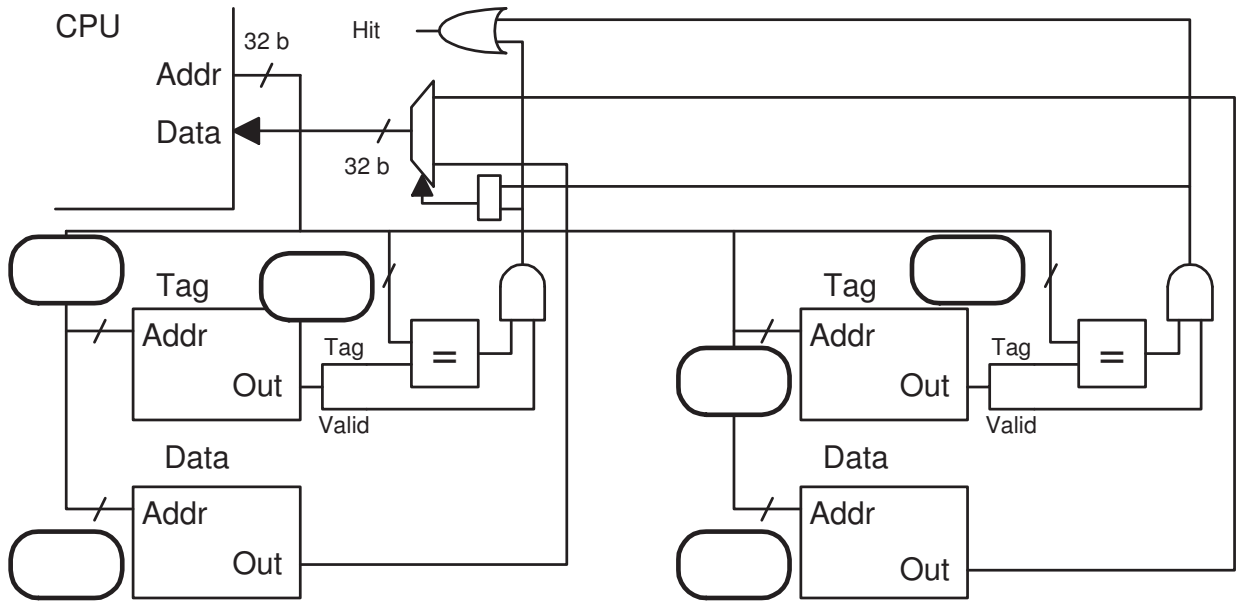
(*d*) What is the benefit of detecting BHT collisions on a four-way superscalar statically scheduled MIPS implementation?

☐ Benefit for 4-way MIPS:

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.)

Address: | | | |

☐ Cache Capacity (Indicate Unit!!):

☐ Associativity:

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

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

Address: | | | |

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

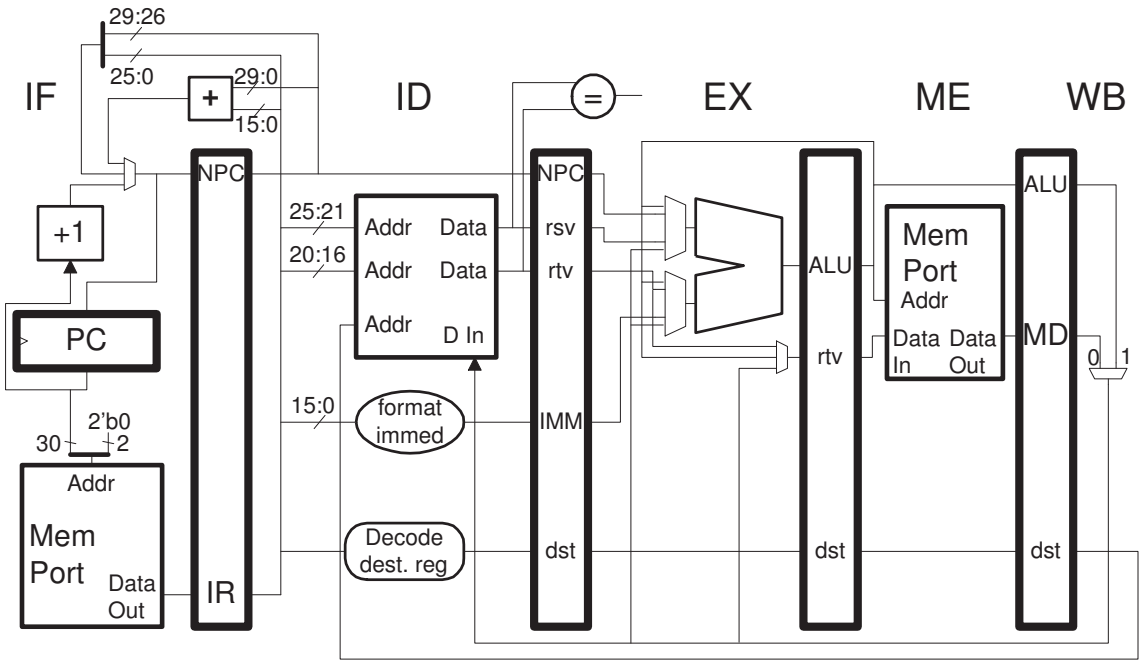
```
double sum = 0.0;
double *a = 0x2000000; // sizeof(double) == 8
int i;
int ILIMIT = 1 << 11;    // = 2^11

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

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.



(*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`.

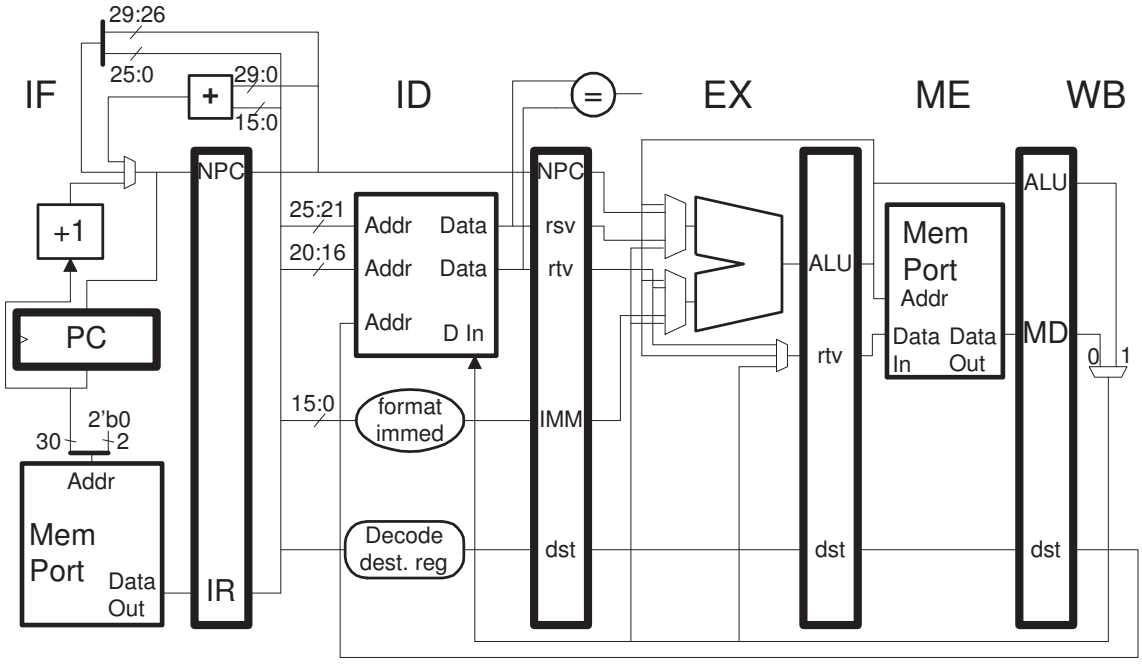
```
sllii r1, 0x1234, 6   ! r1 = 0x1234 << 6
```

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

☐ Easy, moderate, or difficult to implement:

☐ Show datapath changes.

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:

☐ Easy, moderate, or difficult to implement:

☐ Show datapath changes.

(*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:

☐ Easy, moderate, or difficult to implement:

☐ Show datapath changes.

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?

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

☐ 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.)

☐ 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.*)

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?

(*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.

(*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?

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

(*d*) Answer each ISA question below:

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

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