Name _____

Computer Architecture
EE 4720
Final Examination

10 May 1997, 12:30–14:30 CDT

Problem 1	_____	(25 pts)
Problem 2	_____	(25 pts)
Problem 3	_____	(25 pts)
Problem 4	_____	(25 pts)
Exam Total	_____	(100 pts)

Alias _____

Good Luck!

Problem 1: DLX's immediate instructions use 16-bit immediates. Because in many cases larger immediates are needed, new *larger-immediate* instructions are to be added to DLX. Larger-immediate instructions specify an integer arithmetic operation and an immediate, but no registers. The operation is performed using the immediate as one source and the most-recent destination register as both the other source and the destination. For example, consider larger-immediate instruction `addli` in the code fragment below:

```
lw r6, 0(r7)
sub r1, r2, r3
sw 0(r4), r5
addli #0x1ffff  ! Operation: r1 = r1 + 0x1ffff
```

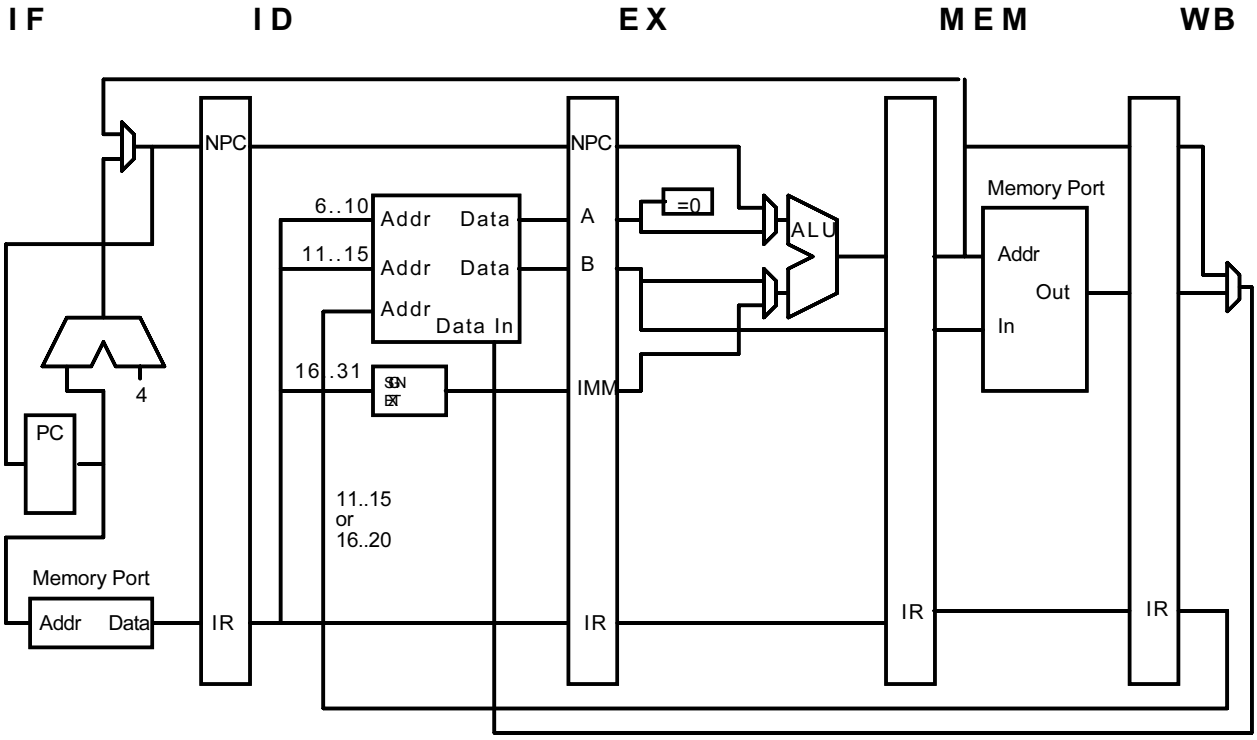
The `addli` instruction uses `r1` because it was the most-recent destination register used. (The `sw` does not modify `r4` or `r5`, so they aren't used. The `lw` does modify `r6`, but it is executed before the `sub`.) The larger-immediate instructions use the J-type format: a 6-bit opcode followed by a 26-bit immediate.

(*a*) Let register `r10` hold the memory address of an integer. The sum of that integer and 0x981234 is to be written to `r11`. Write two code fragments to perform this operation, one with and one without a larger-immediate instruction. (5 pts)

Problem 1 continued: (*b*) Show the modifications needed to implement larger-immediate instructions on the pipeline illustrated below. (Read the next part before solving.) (10 pts)

(*c*) Show how bypassing can be implemented for the instruction. *(Hint: This is easier than regular bypassing since one source is the most-recent destination.)* (10 pts)

- Addresses must be presented to the register file at the beginning of a cycle and the data won't be available until the end of the cycle.
- Be sure to label the function of each connection to registers and other devices. (*E.g.*, address, write, data.) Avoid magic boxes and clouds.
- Explain your modifications using an annotated timing diagram showing sample code executing. A detailed logic and timing diagram are preferred over a lengthy verbal description.



Problem 2: The code fragment repeated below is to be executed on several machines similar to DLX. All machines have an integer execution unit, a multiply FP unit, an add FP unit, and a divide FP unit. The integer unit takes one cycle, the add unit takes two cycles, the multiply unit takes 4 cycles, and the divide unit takes nine cycles. The multiply unit has an initiation interval of 2, the divide unit has an initiation interval of 9, all other functional units have an initiation of 1. In all cases make sure distinct pipeline segments have distinct names. All registers and functional units are available when the fragment starts. State any assumptions. (25 pts)

(a) Show how the sequence would execute on a single-issue pipeline without register renaming and which is fully bypassed.

```
div f0,f10,f11

mul f0,f3,f2

add f8,f9,f0

add f0,f1,f2

mul f3,f0,f4

mul f5,f0,f6
```

(b) Show how the sequence would execute on a single-issue pipeline using Tomasulo's approach as described in class, with two reservation stations per functional unit.

```
div f0,f10,f11

mul f0,f3,f2

add f8,f9,f0

add f0,f1,f2

mul f3,f0,f4

mul f5,f0,f6
```

Problem 2 continued:

(*c*) Show how the sequence would execute on a dynamic issue two-way superscalar machine using Tomasulo's approach with two reservation stations per functional unit. The superscalar machine has the same functional units as the others. (*I.e.*, there is not two of every functional unit.) Of course, there are two common data busses; every functional unit can write either one.

```
div f0,f10,f11

mul f0,f3,f2

add f8,f9,f0

add f0,f1,f2

mul f3,f0,f4

mul f5,f0,f6
```

(*d*) Show how the sequence would execute on a static issue two-way superscalar machine. The superscalar machine has the same functional units as the others. (*I.e.*, there is not two of every functional unit.) The pipelines are fully bypassed.

```
div f0,f10,f11

mul f0,f3,f2

add f8,f9,f0

add f0,f1,f2

mul f3,f0,f4

mul f5,f0,f6
```

Problem 3: Consider a 3-way, set-associative cache with 4096 sets and 256-byte blocks using LRU replacement to be used with a CPU reading data in aligned 8-byte units. The size of the address space is 64 bits, the cache uses virtual addresses.

(*a*) (1) Show the bit positions used for the tag, index, and offset. (2) How many bytes can the cache cache? (3) How many bytes of memory does it take to implement the cache? (5 pts)

(*b*) Modify the program below (or re-write completely) so that it moves as many blocks as possible (without replacing other blocks) into each of 2048 sets and moves exactly one block into each of the other 2048 sets. The program should accomplish this using the **minimum** number of array accesses. The cache is empty before the program is run; assume that only array accesses generate memory references. (5 pts)

```
double *a = 0x10000000;  /* Assume the entire address space is ours. */
for( i=0; i<5; i++ ) total += a[i];
```

The size of a double is 8 bytes. The address of `a[0]` is 0x10000000.

Problem 3 continued:

(*c*) Because of the 64-bit address space, a large amount of memory is used by the cache for storing tags. If the number of distinct tags present in the cache is much less than the number of blocks, that memory is wasted. To reduce the storage needed, tags can be converted to smaller *htags* using a hash function. A hash function is given[1] which hashes a tag to an 8-bit htag. (If you're hopelessly confused, see the last point below.) Design a cache with the above specifications (3-way ...) that uses the hash function to reduce the amount of storage needed for tags. (15 pts)

- Show hardware to convert addresses from the CPU to addresses for the cache's memory devices, to detect cache hits, to detect hash-function collisions, and to connect data read from the cache to the CPU. (The hash function can map two different tags to the same htag, that's called a collision.) *Do not* show hardware needed for writes, hardware to handle misses, and connections to main memory.
- Show the hash function itself as a box with a single input and output. Assume memory devices of any size are available (any number of address bits and any number of data bits). Where appropriate, specify the bits used in a connection (*e.g.*, 1..3). Omit controller details.
- Briefly describe what happens when (1) an address with a new tag is presented; (2) an address with an already-cached tag is presented (Note that this can be a miss or a hit.); and (3) an address with a tag colliding with a tag in the cache is presented.
- **For reduced credit:** (1) design just the cache described on the previous page (showing the memory devices, how the memory address is converted to addresses for the devices, etc.) and (2) assuming the cache is write-back, describe the sequence of events during a cache hit and cache miss requiring replacement. *This part can be skipped if the tag-hashing design is presented.*

[1] Read this footnote after the exam. The hash function would have to be more elaborate than the simple bit mask used in the BHT. The hash function properties and design are not a part of the problem, so don't think them until the test is over.

Problem 4: Answer each question below and on the next page.

(*a*) As presented in class, a system using Tomasulo's approach attempts to write the common data bus in the cycle *after* execution is complete. The DLX pipeline using bypassing was able to transmit a result in the last cycle of execution. Why would it be more difficult (but far from impossible) to write the common data bus in the last cycle of execution than it is to do bypassing? (5 pts)

(*b*) Computer engineers are hard at work on the next implementation of an architecture. Consider two benchmarks, one is a synthetic benchmark written by the engineers, the other is an application benchmark suite used by *P.O. Signers*, a computer magazine the engineers' customers read and respect. Describe a situation in which the benchmark suite would be a better choice for the engineers and another situation where the synthetic benchmark would be the better choice. Briefly justify your answers. (5 pts)

Problem 4 continued:

(*c*) During program execution on the Chapter 3 DLX implementation, an exception is raised in the IF stage while fetching an `add` instruction. No other exceptions occur on this or previous cycles. Show a timing diagram in which this occurs and in which the trap handler for a *different* instruction ends up being called. Explain why a different handler was called. Point out important steps in the timing diagram. The timing diagram should end with the fetch of the first instruction of the handler. (5 pts)

(*d*) How is the code scheduling done for static-issue superscalar machines similar to, and different from, the re-compiling needed for VLIW machines? (5 pts)

(*e*) Suppose the value of m were to be increased in an (m, n) correlating branch predictor while keeping the size of the branch history table constant. Give a reason why branch prediction might improve, and a reason why branch prediction might get worse. (Both are possible, depending upon how large m was before the increase.) (5 pts)

Yes, it's finally over.