CS418	**Advanced Computer Architecture**	**Spring 1995**

Homework 5 (NO CREDIT): Due 9 Feb

Note

- This homework is for your use in studying for the midterm (Thur 9 Feb).
- The assignment is not worth any points, and will not be graded.
- We will discuss the assignment in class on Tue 7 Feb, and the solutions will be handed out then.
- This is a relatively long assignment. You may not want to do all of the problems all of the way through. But, this assignment does cover the main concepts of the course so far, so you are expected to be able to do all of the problems.

General Information for This Homework

In this homework we will consider the following fragment of pseudo-code:

```
/* A, B : array [0 to 1023] of int (ints are 32 bits) */
for i = 0 to 1023 do
 A[i] = 3*A[i] + B[i]/A[i];
```

You will hand-compile the code for two different architectures and compare the execution times for an implementation of each architecture. The two architectures represent "Speed Demon" (*e.g.* DEC Alpha) and "Brainiac" (*e.g.* IBM Power) approaches to Risc architectures. The pipelines for the two microprocessors are shown in Figures 1 (speed demon) and **??** (brainiac).

Information Common to Both Microprocessors

Pipelines

- All "normal" arithmetic and logical instructions (*e.g.* ADD, SUB, XOR, MULT, *etc.*) use the Execute stage and then go directly to WriteBack.

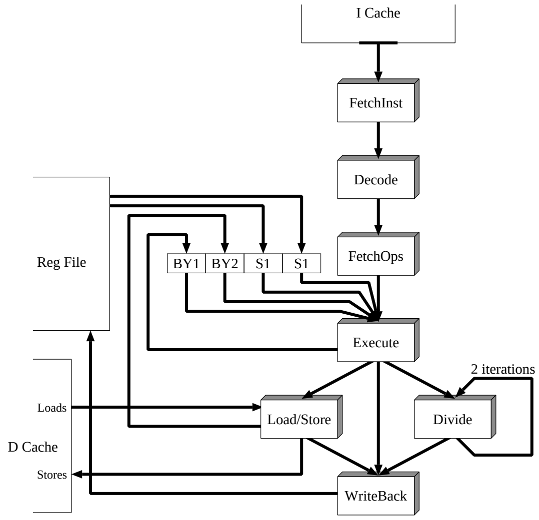

Figure 1: Speed Demon Pipeline

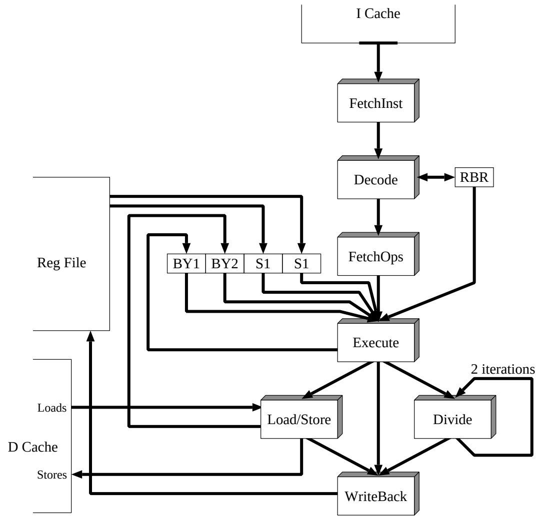

Figure 2: Brainiac Pipeline

- Divide (DIV) instructions use Execute, and then loop through the Divide stage *twice*, before going to WriteBack.
- Load instructions instructions use Execute and then the Load/Store stage before going to WriteBack.
- Store instructions use Execute and then the Load/Store stage. They do not use WriteBack.
- Each stage can contain at most one instruction.
- "BY1" is the register for the bypass from the output of Execute to its input.
- "BY2" is the register for the bypass from the output of Load/Store to the input of Execute.
- "S1" and "S2" are the registers where the register file puts the values currently stored in the registers corresponding to the operands in the FetchOps stage.

- There is no bypass path from the output of Divide to the input of Execute. Divide instructions *must* be written to the register file before they can be used.

Memory

- A level one cache hit has a latency of one cycle (instructions spend one cycle in Load/Store).
- A level two cache hit has a latency of five cycles (instructions spend five cycles in Load/Store. Remember only one instruction can be in a stage at a time).
- When there is a level one cache miss, 128 consecutive bytes of data are brought from the level two cache to the level one cache.
- All of the instructions are in the I cache.
- The array `A` starts at address 0x2000 and is currently in the level 1 cache.
- The array `B` starts at address 0x3000 and is currently in the level 2 cache.

Registers

For both architectures, assume that the code fragment for this homework can use seven registers. (Other registers could be already allocated for other variables, function arguments and return values, stack pointer, etc).

"Speed Demon" Architecture and Microprocessor

- Branch instructions test a general register (*e.g.* `BEQ R3, 300`)
- Branches are resolved in the Execute stage.
- Branch prediction occurs in the Decode stage of the pipeline. Therefore, if a branch is predicted to be taken, the instruction in the Fetch Instruction stage is nullified (killed) and the PC is updated to the branch destination. This is independent of nullifying instructions after a branch mispredict.
- Branches with negative offsets (earlier locations in code) are predicted to be taken.
- Branches with positive offsets (later locations in code) are predicted not taken.
- All Load/Store instructions use displacement addressing mode (*e.g.* `100(R3) = M[100+R3]`).
- Processor speed = 200MHz

"Brainiac" Architecture and Microprocessor

- Load/Store instructions use either displacement (*e.g.* `100(R3) = M[100+R3]`) or scaled (*e.g.* `100(R3)[R4] = M[100+R3+R4*d]`) addressing modes. Load/Stores using scaled addressing mode spend *two cycles* in Execute.
- RBR is a special decrement-and-test register, which is used with the `BCT` instruction. The instruction `BCT imm` decrements `RB` and branches to `PC+imm` if `RBR` is non-negative after decrementing. `RBR` is decremented and tested in the Decode stage and the branch is resolved in this stage. Your code fragment may use `RBR`, and this counts as one of the seven registers available.
- There is a special load-rbr-immediate instruction (`LDBR`), which is used to load `RBR` with a loop count at the beginning of a loop. This load is done in the Decode stage.
- `RBR` can be used as a source operand to Execute, but not as a destination.
- Processor speed = 100MHz

Problem 1 (0 Points)

Write the assembly code for both the Speed Demon and Brainiac processors. Be sure to use your registers effectively and schedule the code for optimum performance. Utilize the added features of the Brainiac when they will increase performance.

Problem 2 (0 Points)

(a) (0 Points) What are the static and dynamic code sizes for the two code fragments?

(b) (0 Points) How many static and dynamic memory references are there in each code fragment?

Problem 3 (0 Points)

Draw execution tables for the initiation of the loop (setting up register values), one iteration of the loop, and the conclusion of the loop (testing loop count when it is done with the array) for each of the two code fragments. Assume that all memory accesses in these iterations get level once cache hits.

Problem 4 (0 Points) Using the execution tables from Problem 3, calculate the average CPI for the code fragments for each microprocessor.

Problem 5 (0 Points) The average CPI calculated in Problem 4 is unrealistic, because it assumes that all memory accesses get level one cache hits. Using the memory-latency information given in the general information section, calculate the real CPI for the code fragments.

Problem 6 (0 Points) Calculate the total CPU time for each code fragment and the speedup of the faster machine vs. the slower machine.

Problem 7 (0 Points) What unrealistic assumptions were made in this assignment? What are realistic alternatives to these assumptions and how would it complicate the assignment if the realistic assumptions were used?

Problem 8 (0 Points) What performance features of Brainiac were most useful in this example?

Problem 9 (0 Points) What are the costs (*e.g.* where would extra hardware be needed, what would increase the cycle time and why) of each of the Brainiac's features. Be specific.

Problem 10 (0 Points) Is this example a good benchmark for comparing the Brainiac and Speed Demon? Justify your answer.