

# CS 573

## Homework 2

### Due Wednesday, March 3, 2004

24th February 2004

For this assignment, we will consider a two-operand computer architecture which will be very, very loosely based on the Intel IA-32 architecture.

### The Instruction Set Architecture

The ISA instruction set has four architected registers, named  $R_a$ ,  $R_b$ ,  $R_c$ , and  $R_d$ .

Every arithmetic instruction in this machine takes two floating point operands, which we will refer to as  $Src$  and  $Dst$ . These operands can be registers, immediate values, or memory locations, but at least one of them has to be a register or an immediate value. In other words

- $Src$  can be a register or an immediate value, in which case  $Dst$  can be either a register or a memory location.
- $Src$  can be a memory location, in which case  $Dst$  must be a register.

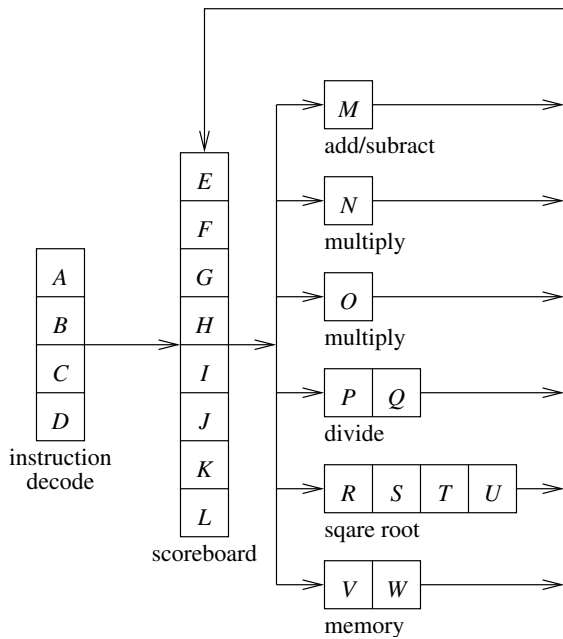
(it wouldn't make any sense for  $Dst$  to be an immediate value, of course).

For this assignment, the important instructions are in the following table.

Instruction	Example	Description
mov $Dst, Src$	mov $R_a, A$	Copy a value from $Src$ to $Dst$
add $Dst, Src$	add $R_c, R_2$	Add $Src$ to $Dst$ , placing the result in $Dst$
sub $Dst, Src$	sub $val, R_b$	Subtract from in $Dst$ , placing the result in $Dst$
mul $Dst, Src$	mul $loc, \#2.0$	Multiply $Src$ by $Dst$ , placing the result in $Dst$
div $Dst, Src$	div $R_d, R_b$	Divide $Dst$ by $Src$ , placing the result in $Dst$
sqrt $Dst, Src$	sqrt $R_a, 8.0$	Compute the square root of $Src$ , placing the result in $Dst$

### The Microarchitecture

Let's take a look at a block diagram of the machine.



All of the components of the machine are labelled for clarity later (and in honor of Rube Goldberg, who ought to be the patron saint of modern microarchitectures).

### Instruction Decode Unit (A-D)

The instruction decode unit is able to obtain up to four ISA instructions per cycle. Its purpose is to translate ISA instructions into  $\mu$ ops, and translate architected registers into physical register numbers. Instructions are read in-order from the instruction cache, and  $\mu$ ops are transferred in-order to the scoreboard.

Every ISA instruction is translated into one, two, or three  $\mu$ ops, as follows:

- An instruction whose *Src* and *Dst* are both registers or immediate values is translated into a single  $\mu$ op. For simplicity, we'll refer to the resulting  $\mu$ op by the same name as the ISA instruction.
- An instruction whose *Src* is a memory location and whose *Dst* is a register is translated into two  $\mu$ ops: a load  $\mu$ op which fetches the value from memory and places it in a register, and a register-register arithmetic  $\mu$ op.
- An instruction whose *Src* is a register or an immediate value and whose *Dst* is a memory location is translated into three  $\mu$ ops: a load  $\mu$ op, which fetches the value from memory and places it in a register, a register-register arithmetic  $\mu$ op, and a store  $\mu$ op to place the result in memory.

load, store, and register-register *mov*  $\mu$ ops are two-operand, just like the ISA instructions. Arithmetic  $\mu$ ops are three-operand, with a destination and two sources specified.

There are sixteen physical registers (R0 through R15). Whenever an architected register is used as the destination of an instruction, the smallest-numbered available physical register is allocated for it. A physical register remains allocated until (1) there are no instructions in the scoreboard waiting for it, and (2) there is no architected register assigned to it.

### Scoreboard (E-L)

The scoreboard is able to accept as many  $\mu$ ops as are available from the decoder on each cycle, subject to a maximum of eight  $\mu$ ops in the scoreboard on any cycle.  $\mu$ ops are held in the scoreboard until their operands are ready and a functional unit is ready to accept them.

An arbitrary number of  $\mu$ ops can be dispatched to the functional units per cycle, subject to the constraints that their operands are available and there have to be functional units for them to go to. A  $\mu$ op remains in the scoreboard until (1) it has finished executing, and (2) no older  $\mu$ op is still in the scoreboard (so  $\mu$ ops are retired in-order, though multiple  $\mu$ ops can be retired on a cycle).

A register-to-register `mov` takes place entirely within the scoreboard, and takes one cycle. An arbitrary number of register-to-register `movs` can take place simultaneously.

## Functional Units (*M-W*)

The six functional units are specialized as shown, take as many cycles as shown (one for add/sub and multiply, two for division and memory load/store, and four for square root), and are fully pipelined (so it's possible to start one memory operation per cycle, for instance).

We are assuming separate instruction and data caches (so there is no conflict between instruction fetches and data reads and writes), and assuming all memory accesses are satisfied from cache.

On the cycle after a  $\mu$ op finishes executing in a functional unit, its result is available to the scoreboard and any instructions waiting for it can enter the pipelines.

## An Example

Let's see how this works, for a two-instruction sequence.

```
mov Ra, addr1
add addr2, Ra
```

### Cycle 1

The two instructions are fetched from the instruction cache into the instruction decode stage (*A* and *B*) and translated into four  $\mu$ ops (note that if there were more instructions in the sequence, up to four could have been fetched and decoded here). The physical registers have been allocated in the order used.

```
load R0, addr1
load R1, addr2
add R2, R0, R1
store addr2, R2
```

### Cycle 2

The four  $\mu$ ops are transferred to the scoreboard (*E*, *F*, *G* and *H*) (note that if there were more  $\mu$ ops, up to eight of them could have been transferred here).

### Cycle 3

The two `load`  $\mu$ ops have no dependencies, and can be transferred to the memory pipe. Unfortunately, the memory pipe is only one  $\mu$ op wide, so only the `load R0, addr1`  $\mu$ op can start down the pipe (*V*).



## Your Problem

The Quadratic Formula, used to find the roots of quadratic equations, is  $r = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$ .

1. Write an optimized (in the sense of running as quickly as possible on this machine) program to compute both roots of a quadratic equation. The coefficients should be in memory locations named `a`, `b`, and `c`; the results should be stored in memory locations `r1` and `r2` (I'll bet you were wondering why this architecture had a square root instruction, weren't you?). You may assume the roots will be real (in other words, you may assume  $b^2 - 4ac$  is non-negative).
2. Show how your code is translated into  $\mu$ ops.
3. Draw a timing diagram similar to the one above showing how your code is executed by this machine.