Reward! Extra credit to the first person to find a mistake!

1. First, I wrote a solution that followed “good rules” for fast code - namely, avoidance of memory operations and repeating operations. This solution didn’t try to optimize the order of operations, though.

```
mov Rb, b  Rb = b
mov Rd, Rb Rd = b
mul Rb, Rb Rb = b^2
mov Ra, a Ra = a
mov Rc, Ra Rc = a
mul Ra, #4.0 Ra = 4a
mul Ra, c Ra = 4ac
sub Rb, Ra Rb = b^2 - 4ac
sqrt Rb, Rb Rb = sqrt(b^2 - 4ac)
mul Rd, #-1.0 Rd = -b
mul Rc, #2.0 Rc = 2a
mov Ra, Rd Ra = -b
sub Rd, Rb Rd = -b - sqrt(b^2 - 4ac)
div Rd, Rc Rd = (-b - sqrt(b^2 - 4ac))/2a
mov r1, Rd
add Ra, Rb Ra = -b + sqrt(b^2 - 4ac)
div Ra, Rc Ra = (-b + sqrt(b^2 - 4ac))/2a
mov r2, Ra
```
2. Second, I did a partial decode of all the instructions. Assignment of registers for renaming will come later, along with decode and scoreboard slots.

```
mov Rd, b
load b
mov Rd, Rb
mov
mul Rd, Rd
mul
mov Ra, a
load a
mov Rc, Ra
mov
mul Ra, #4.0
mul
mul Ra, c
load c
mul
sub Rb, Ra
sub
sqrt Rb, Rb
sqrt
sub Rd, Rb
sub
div Rd, Rc
div
mov r1, Rd
store
add Ra, Rb
add
div Ra, Rc
div
mov r2, Ra
store
```
3. Next I ran all the instructions, assuming infinite hardware resources, so I could find the dependencies and critical path. Dependencies are shown with arrows; critical path is bold.

```assembly
mov Rd, b
load b
mov Rd, Rb
load c
mul Rb, Rb
sqrt Rb, Rb
sub Rd, Rb
div Rd, Rc
mov r1, Rd
store
add Ra, Rb
div Ra, Rc
div
mov r2, Ra
store
```
4. See how late each instruction could be started without disturbing the critical path.

```
mov Rb, b
load b
mov Rd, Rb
mov
mul Rb, Rb
mul
mov Ra, a
Load a
mov Rc, Ra
mov
mul Ra, #4.0
mul 4.0
mul Ra, c
load c
mul
sub Rb, Ra
sub
sqrt Rb, Rb
sqrt
sub Rd, Rb
sub
div Rd, Rc
div
mov r1, Rd
store
add Ra, Rb
add
div Ra, Rc
div
mov r2, Ra
store
```
5. Calculate the latest issue time for each instruction that won’t interfere with the critical path. If there is an antidependency between an instruction and a later instruction, place the earlier one a cycle earlier than the critical path analysis would indicate. In this figure, the last starting time for each instruction is shown on the left; instructions on the critical path are bold and instructions whose start time have been adjusted for antidependencies are in italics.

```
2     mov Rb, b
     load b
3     mov Rd, Rb
     mov
4     mul Rb, Rb
     mul
1     mov Ra, a
     Load a
1     mov Rc, Ra
     mov
3     mul Ra, #4.0
     mul 4.0
2     mul Ra, c
     load c
     mul
5     sub Rb, Ra
     sub
6     sqrt Rb, Rb
     sqrt
9     mul Rd, #-1.0
     mul
10    mul Rc, #2.0
     mul
9     mov Ra, Rd
     mov
10    sub Rd, Rb
     sub
11    div Rd, Rc
     div
13    mov r1, Rd
     store
10    add Ra, Rb
     add
11    div Ra, Rc
     div
13    mov r2, Ra
     store
```
6. Sort the instructions according to start cycle. If multiple instructions show the same start cycle, put the instructions on the critical path first.

1. mov Ra, a  
   Load a
2. mov Rc, Ra  
   mov
3. mul Ra, c  
   load c  
   mul
4. mov Rb, b  
   load b
5. mul Ra, #4.0  
   mul 4.0
6. mov Rd, Rb  
   mov
7. mul Rb, Rb  
   mul
8. sub Rb, Ra  
   sub
9. sqrt Rb, Rb  
   sqrt
10. mul Rd, #−1.0  
    mul
11. mov Ra, Rd  
    mov
12. sub Rd, Rb  
    sub
13. add Ra, Rb  
    add
14. mul Rc, #2.0  
    mul
15. div Rd, Rc  
    div
16. div Ra, Rc  
    div
17. mov r1, Rd  
    store
18. mov r2, Ra  
    store
7. Run the sorted code, assigning to hardware resources appropriately. I took a slight risk here in assuming I’d have enough physical registers to work out the timing first and the registers second. As it turned out, I had lots of registers left over.
8. Assign the physical registers. The columns on the left are a scratch pad I used to keep track of which registers were allocated and what cycle they’d be available to be reallocated.

|   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Ra | Rb | Rc | Rd |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R0(7)** |   |   | mov Ra, a |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| Load R0, a | mov Rc, Ra | mov R1, R0 |   |   |   |   | sqrt R0, R8 |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R1(11)** |   |   | mov R1, R0 |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R2(7)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R3(8)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R4(9)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R5(9)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R6(9)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R7(9)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R8(10)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R0(14)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R2(11)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R3(11)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R4(16)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R5(12)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R6(17)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |
| **R8(18)** |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |   |

```
mov Ra, a
Load R0, a
mov Rc, Ra
mov R1, R0
mul Ra, c
load R2, c
mul R3, R0, Ra
mov Rb, b
mul Ra, #4.0
mul R5, R3, 4.0
mov Rd, Rb
mul R6, R4
mul R7, R4, R4
sub Rb, Ra
mul R8, R7, R2
sqrt Rb, Rb
mul Ra, Rd
mov R9, R2, R3
add Ra, Rb
add R4, R3, R0
mul Rc, #2.0
div Rd, Rc
div R6, R9, R5
div Ra, Rc
div R8, R4, R5
mov r1, Rd
store R6, r1
mov r2, Ra
store R8, r2
```