

CS 573

HW 2

Sample Solution

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

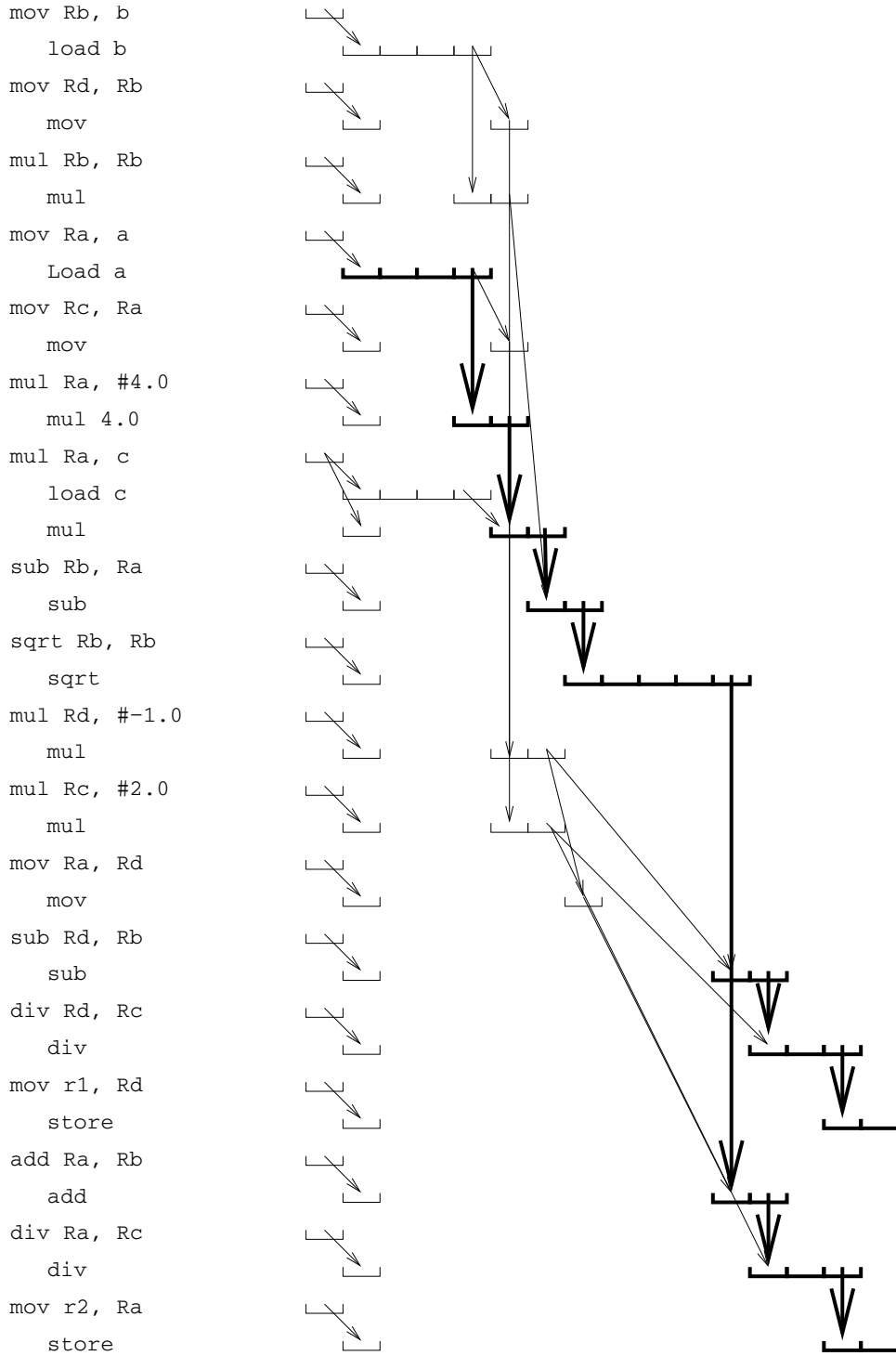
1. First, I wrote a solution that followed “good rules” for fast code - namely, avoidance of memory operations and repeating operations. This solution didn’t try to optimize the order of operations, though.

```
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 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
mul Rd, #-1.0
  mul
mul Rc, #2.0
  mul
mov Ra, Rd
  mov
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.

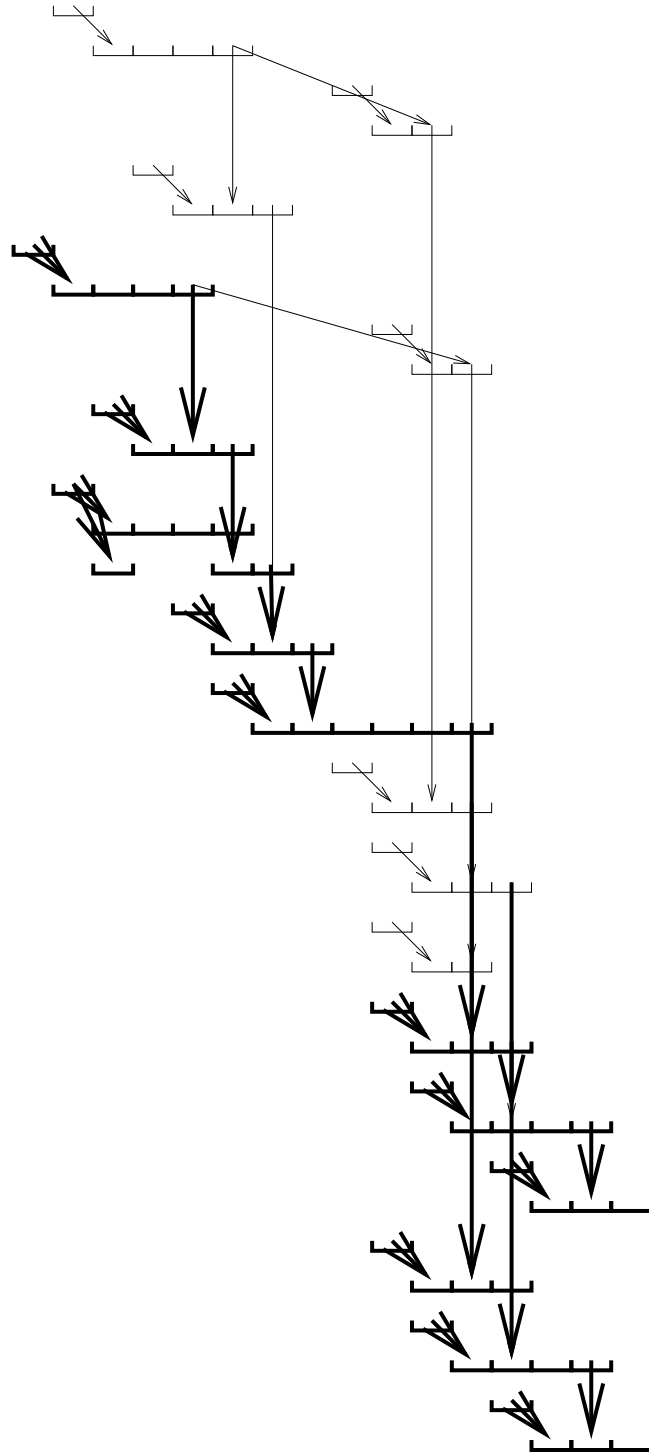


4. See how late each instruction could be started without disturbing the critical path.

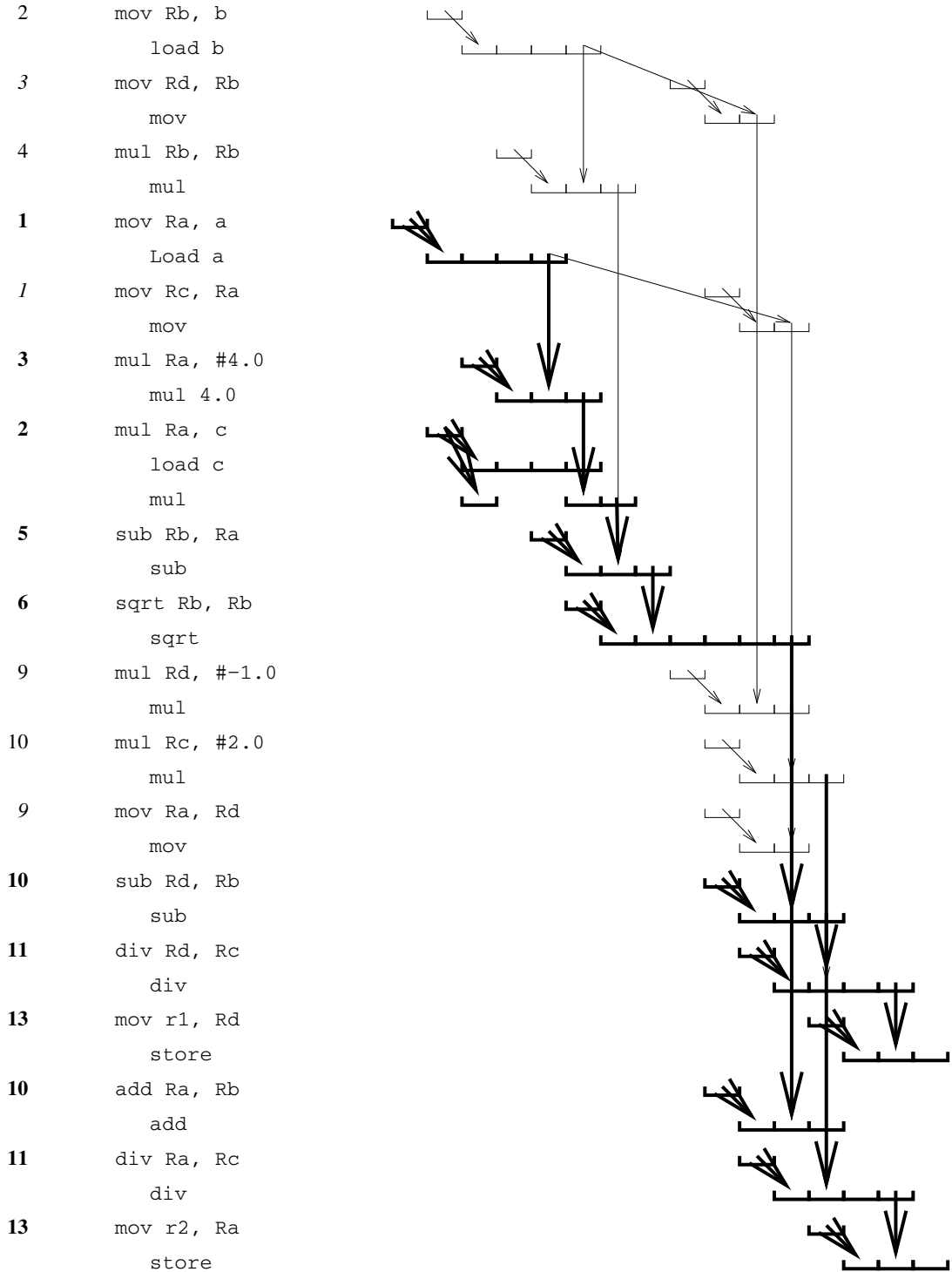
```

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
mul Rd, #-1.0
  mul
mul Rc, #2.0
  mul
mov Ra, Rd
  mov
sub Rd, Rb
  sub
div Rd, Rc
  div
mov r1, Rd
  store
add Ra, Rb
  add
div Ra, Rc
  div
mov r2, Ra
  store

```



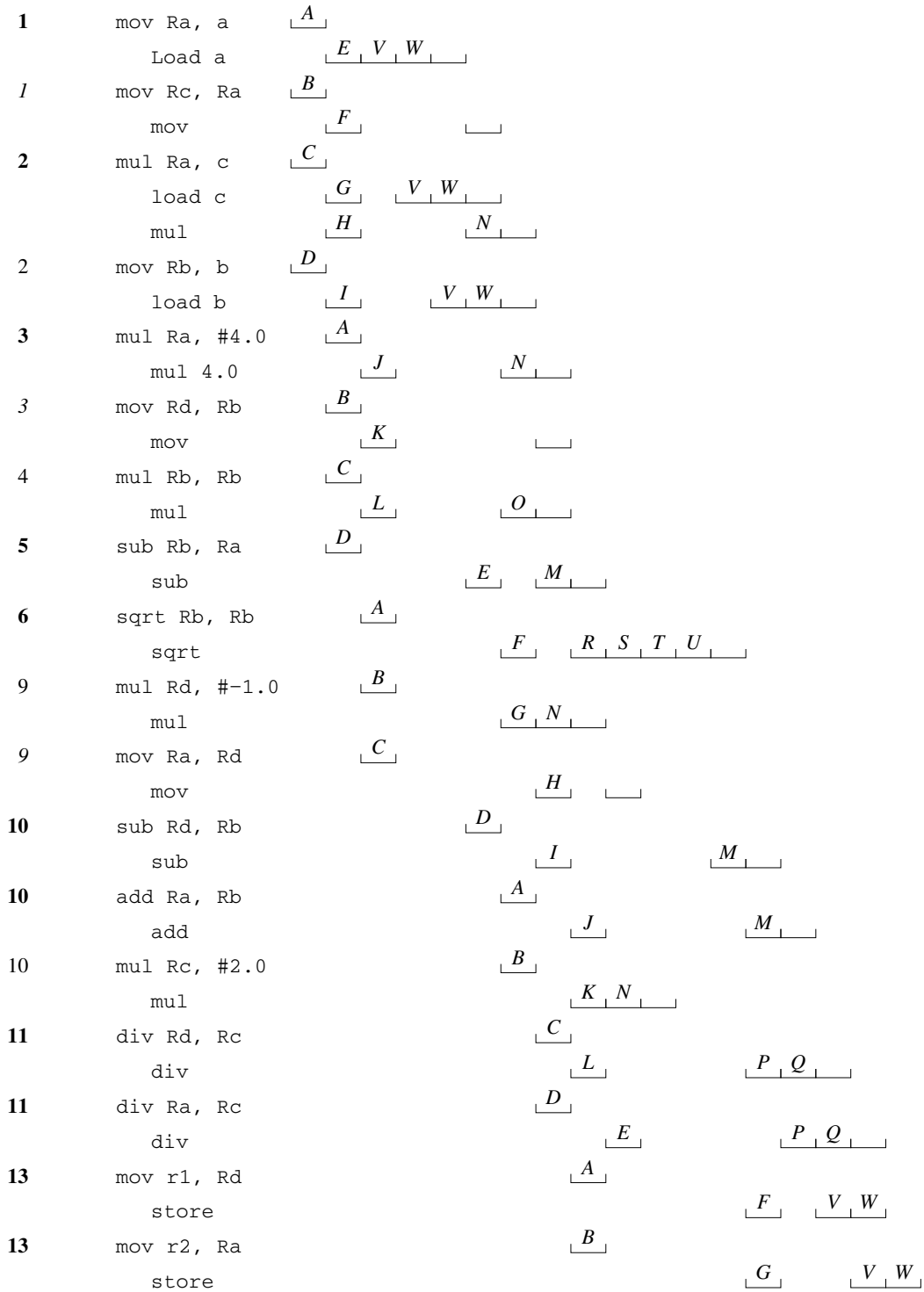
5. Calculate the latest issue time for each instruction that won't interfere with the critical path. If there is an antidependency between an instruction and a later instruction, place the earlier one a cycle earlier than the critical path analysis would indicate. In this figure, the last starting time 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.



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
<i>1</i>	mov Rc, Ra mov
2	mul Ra, c load c mul
<i>2</i>	mov Rb, b load b
3	mul Ra, #4.0 mul 4.0
<i>3</i>	mov Rd, Rb mov
<i>4</i>	mul Rb, Rb mul
5	sub Rb, Ra sub
6	sqrt Rb, Rb sqrt
<i>9</i>	mul Rd, #-1.0 mul
<i>9</i>	mov Ra, Rd mov
10	sub Rd, Rb sub
10	add Ra, Rb add
<i>10</i>	mul Rc, #2.0 mul
11	div Rd, Rc div
11	div Ra, Rc div
13	mov r1, Rd store
13	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		1	2	3	4	5	6	7	8	9	10	11	12	12	14	15	16	17	18
				mov Ra, a	<u>A</u>																	
R0(7)				Load R0, a	<u>E</u>	<u>V</u>	<u>W</u>															
				mov Rc, Ra	<u>B</u>																	
		R1(11)		mov R1, R0	<u>F</u>																	
				mul Ra, c	<u>C</u>																	
R2(7)				load R2, c	<u>G</u>	<u>V</u>	<u>W</u>															
R3(8)				mul R3, R0, R2	<u>H</u>					<u>N</u>												
				mov Rb, b	<u>D</u>																	
	R4(9)			load R4, b	<u>I</u>			<u>V</u>	<u>W</u>													
				mul Ra, #4.0	<u>A</u>																	
R5(9)				mul R5, R3, 4.0	<u>J</u>						<u>N</u>											
				mov Rd, Rb	<u>B</u>																	
		R6(9)		mov R6, R4	<u>K</u>																	
				mul Rb, Rb	<u>C</u>																	
R7(9)				mul R7, R4, R4	<u>L</u>						<u>O</u>											
				sub Rb, Ra	<u>D</u>																	
R8(10)				sub R8, R7, R2						<u>E</u>	<u>M</u>											
				sqrt Rb, Rb	<u>A</u>																	
R0(14)				sqrt R0, R8							<u>F</u>	<u>R</u>	<u>S</u>	<u>T</u>	<u>U</u>							
				mul Rd, #-1.0	<u>B</u>																	
		R2(11)		mul R2, R6, -1.0							<u>G</u>	<u>N</u>										
				mov Ra, Rd	<u>C</u>																	
R3(11)				mov R3, R2								<u>H</u>										
				sub Rd, Rb						<u>D</u>												
		R9(15)		sub R9, R2, R3							<u>I</u>					<u>M</u>						
				add Ra, Rb						<u>A</u>												
R4(16)				add R4, R3, R0								<u>J</u>				<u>M</u>						
				mul Rc, #2.0						<u>B</u>												
		R5(12)		mul R5, R1, 2									<u>K</u>	<u>N</u>								
				div Rd, Rc								<u>C</u>										
		R6(17)		div R6, R9, R5									<u>L</u>			<u>P</u>	<u>Q</u>					
				div Ra, Rc								<u>D</u>										
R8(18)				div R8, R4, R5									<u>E</u>			<u>P</u>	<u>Q</u>					
				mov r1, Rd								<u>A</u>										
				store R6, r1												<u>F</u>	<u>V</u>	<u>W</u>				
				mov r2, Ra								<u>B</u>										
				store R8, r2												<u>G</u>		<u>V</u>	<u>W</u>			