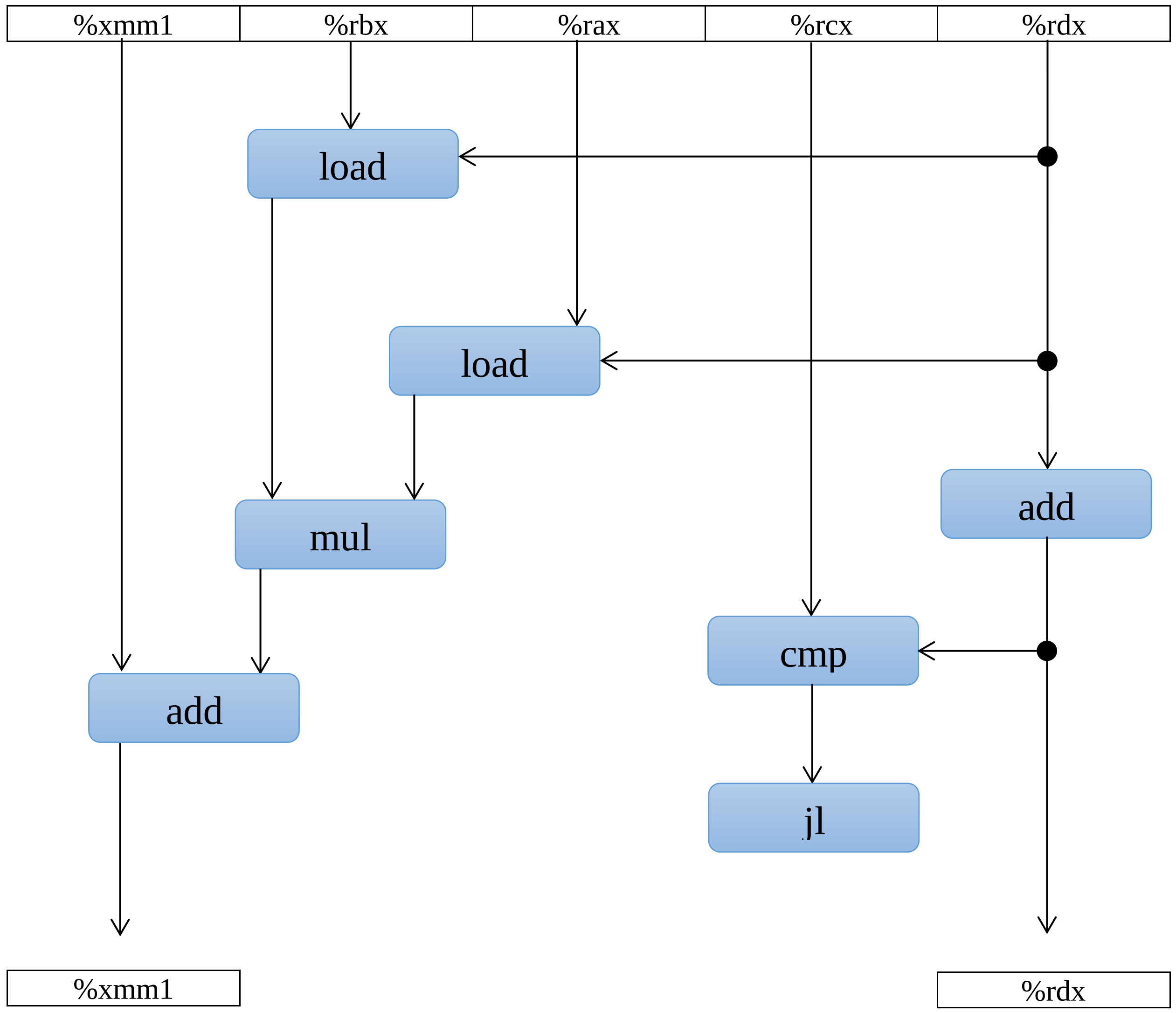
Henzi Kou

Project 6: Part 2

a)



b)

The operations of addition cannot be pipelined because each time the addition occurs (float and integer), it is being compared in the for loop to ensure that the addition can indeed take place. Therefore, the addition instructions depend on the result from the previous loop iteration. First, we see that we have a mulss which is float multiplication, thus it has a latency of 5 cycles. Given that we have adds (float addition) this means that this has a latency of 3 cycles. Next, we also have addq (integer addition) which has a latency of 1 cycle. But from part (a) as we can see that the program pipelines the two load values into the multiplication instruction. Additionally, the add integer instruction gets stacked upon itself which is used in the two load instructions. Therefore, the whole procedure has only a latency bound of 3 cycles (float addition), where the instructions and executions are all bottleneck into that instruction.

d)

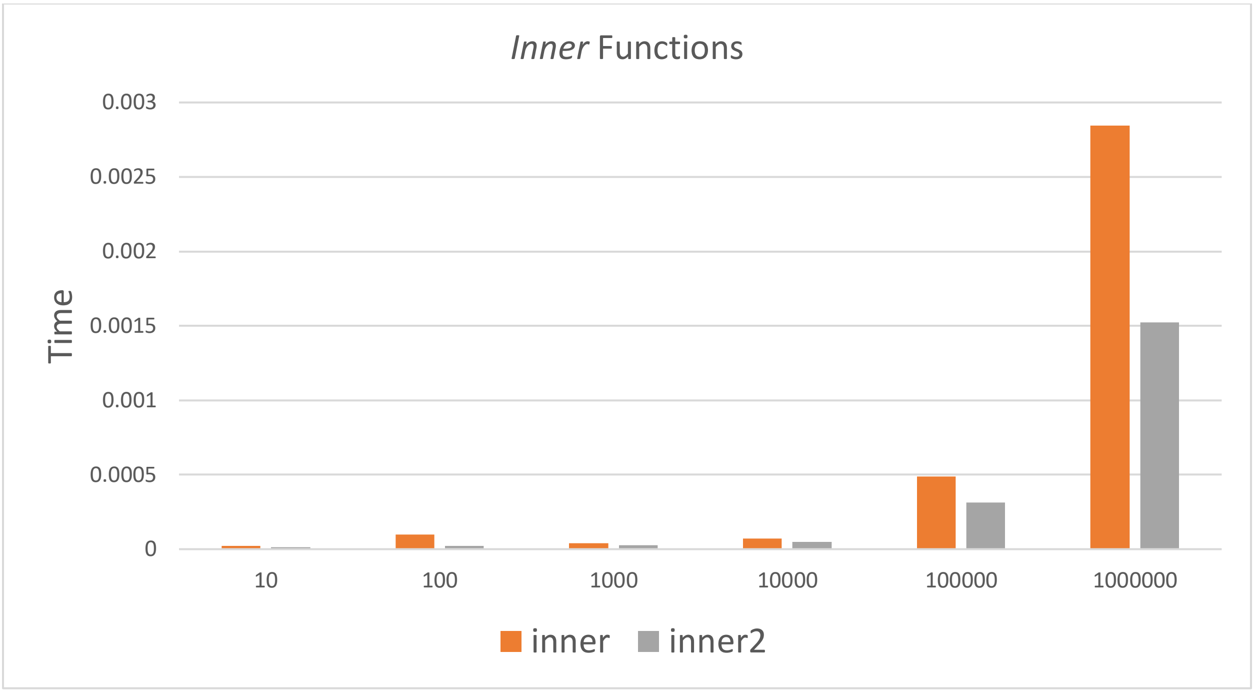
The run times for the unrolled function, *inner()* are graphed below:

10	0.000022
100	0.000099
1000	0.000039
10000	0.000073
100000	0.000489
1000000	0.002844

The run times for the 4-way loop unrolling function *inner2()* are graphed below:

10	0.000015
100	0.000021
1000	0.000026
10000	0.000049
100000	0.000312
1000000	0.001521

As we can clearly see that the four-way loop unrolling function, *inner2()* has a 2x increase in processing speed as the number of items to process rapidly increases to 1,000,000 and above. However there are some discrepancies that occur when we test the *inner()* function with 100 and 1,000 iterations but this is expected because each time that I run the program the run time values varies.


Inner Functions
Time
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
10
100
1000
10000
100000
1000000
inner
inner2