4. Modify the program you used in part 1 of this lab by adding a *for loop* after the *z=x+y* instruction. The *for loop* should consist of a single instruction *z = x+z;* executed 10 times.

**main()**

**{**

**int x,y,z,c;**

**x = 9;**

**y = 8;**

**z = x+y;**

**for(c=0; c<10; c++)**

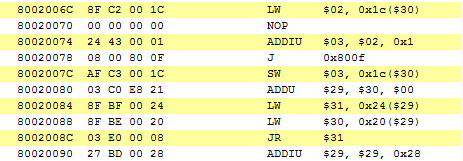
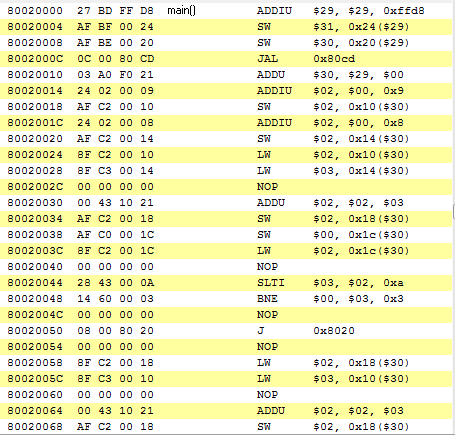
**{**

**z = x+z;**

**}**

**}**

7. Again identify the core assembly instructions in the RAM of the simulator that are the compiled version of your C program.



Start of for loop

ADDU R2 and R3 and store in R2

SW $2, 0x18($30) store word from R2 to mem are pointed to by R30 offset 18 i.e. loc of z

SW $0, 0x1c($30) store word 0 to memory pointed to by R30 offset 1c

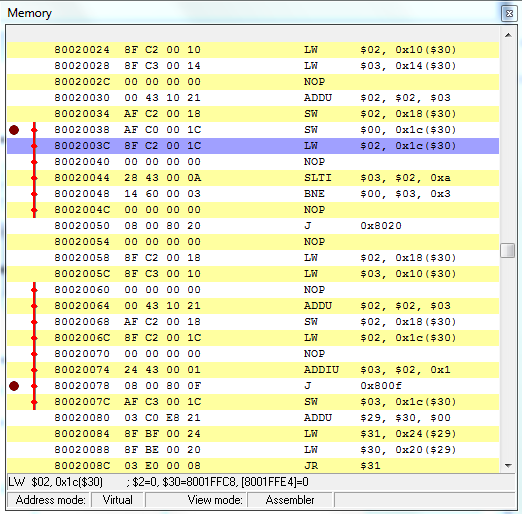
LW $2, 0x1c($30) load word from memory location R30 offset 1c

NOP

SLTI

8. Set a breakpoint at the beginning of this block of code. (see below)

9. Run the program up to the breakpoint.



10. How many instruction cache hits and misses do you predict will occur by the time your loop has fully completed execution. (see below)

11. Open the edit window and look at the instruction cache at this point and select OK to clear the cache before you start to execute your core instructions.

12. Single step though 1 instruction and watch the instruction c line being loaded after the first miss.

13. Interpret the layout of the cache.

14. Single step through 1 iteration of the loop and make sure you understand the hit and miss count.

15. Now estimate how many cache hits and misses should occur when you execute the rest of the iterations of the loop.

16. Single step through your program from this point and confirm that your estimate of hits and misses is correct. If they don’t match, explain what errors you made in your estimate.

17. Now modify your program by replacing the *for loop* with a block of code that sums an array of 5 integers.

18. Build the executable version.

19. While still in MipsIt, view the assembly version of the C program by selecting Build/View Assembler. See below.

20. Upload to the simulator.

21. Again identify the core assembly instructions in the RAM of the simulator that are the

compiled version of your C program.

22. Set a breakpoint at the beginning of this block of code.

23. Run the program up to the breakpoint.

24. Examine the contents of the data cache at this point before you start to execute your core instructions.

25. Now single step through your program from this point and explain the behaviour of the data cache.

26. Modify the characteristics of the data cache so as to reduce the number of misses and confirm that you have achieved this by again single stepping through the program.

Blahh [1] etc [1]