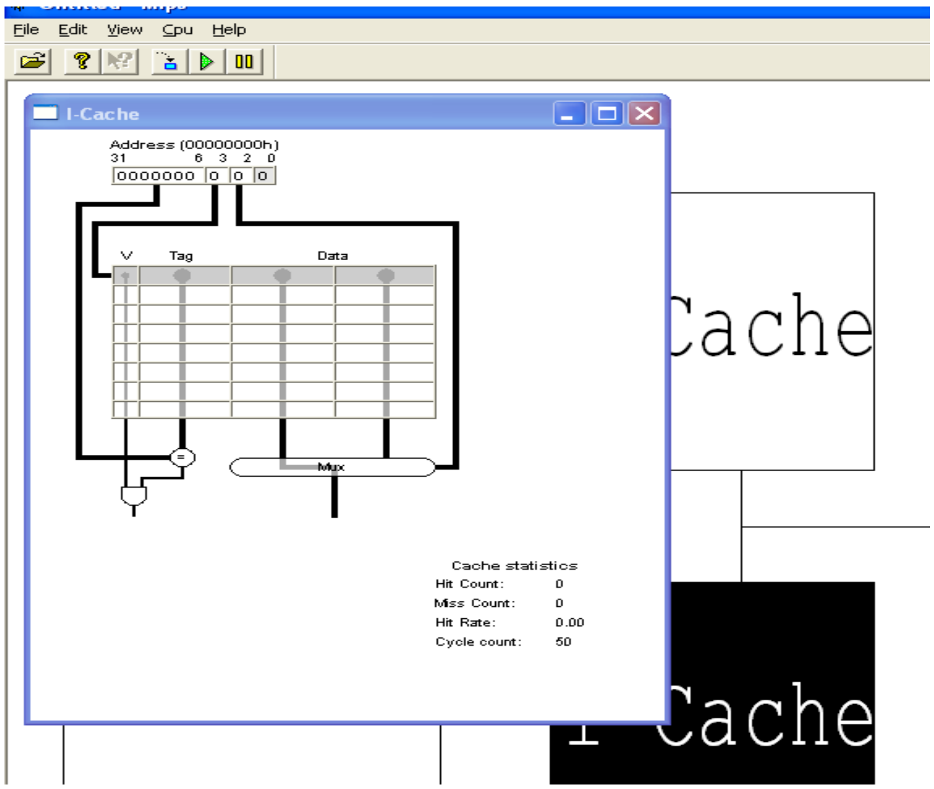
<u>DT081 Year 4 COMPUTER ARCHITECTURE 3</u>

<u>Lab 5 (Part 2)</u>
<u>Examining the Instruction and data caches on the MIPS simulator.</u>

<u>Task</u>

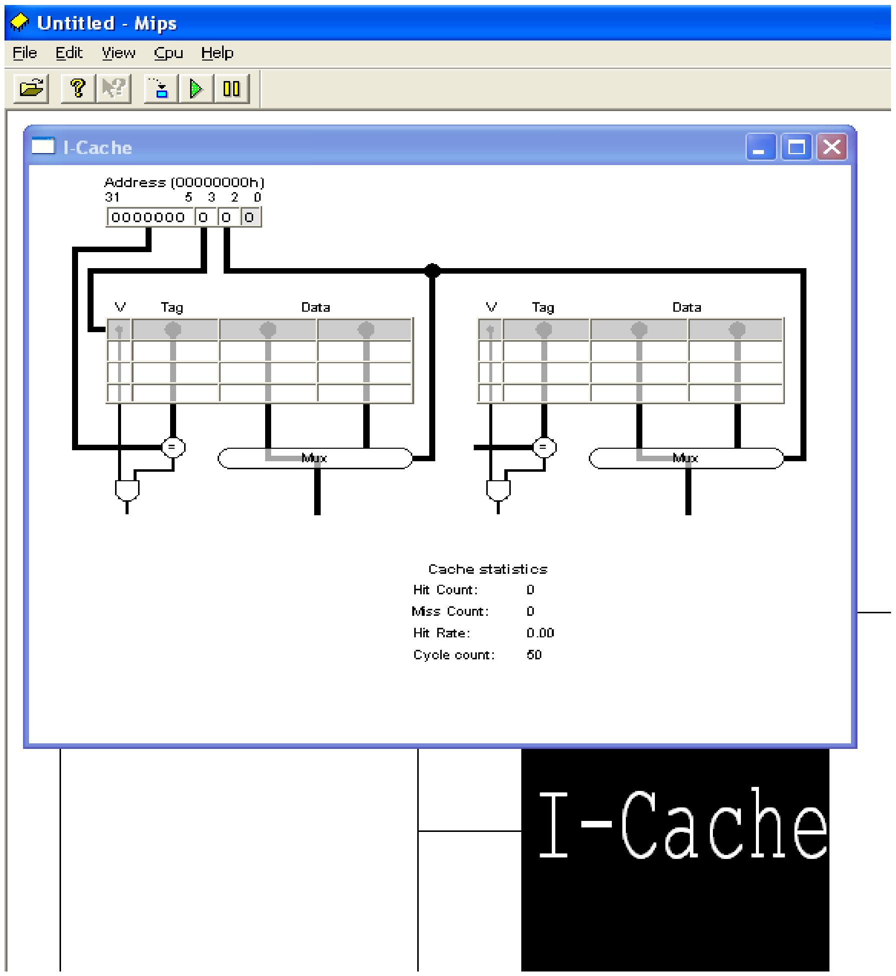
1. In MIPS simulator click on the I – Cache. The window will show the design of the instruction cache which defaults to a direct-mapped design with a 64-bit cache block.



Cache statistics

```
Hit Count:      0
Miss Count:     0
Hit Rate:       0.00
Cycle count:    50
```

2. Change the cache design to a 2-way set-associative cache using menu Edit/cache-mem config.



Cache statistics

Hit Count: 0

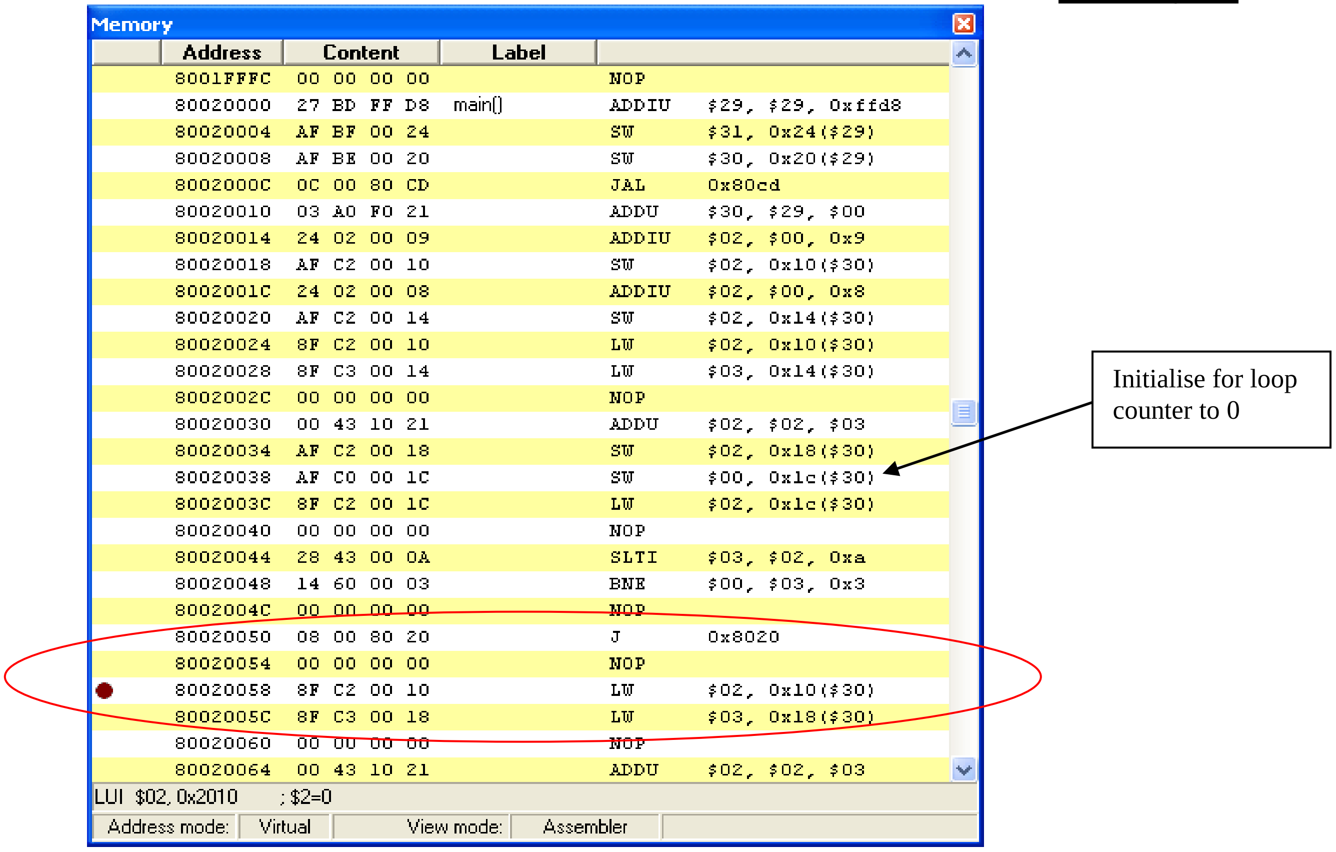
Miss Count: 0

Hit Rate: 0.00

Cycle count: 50

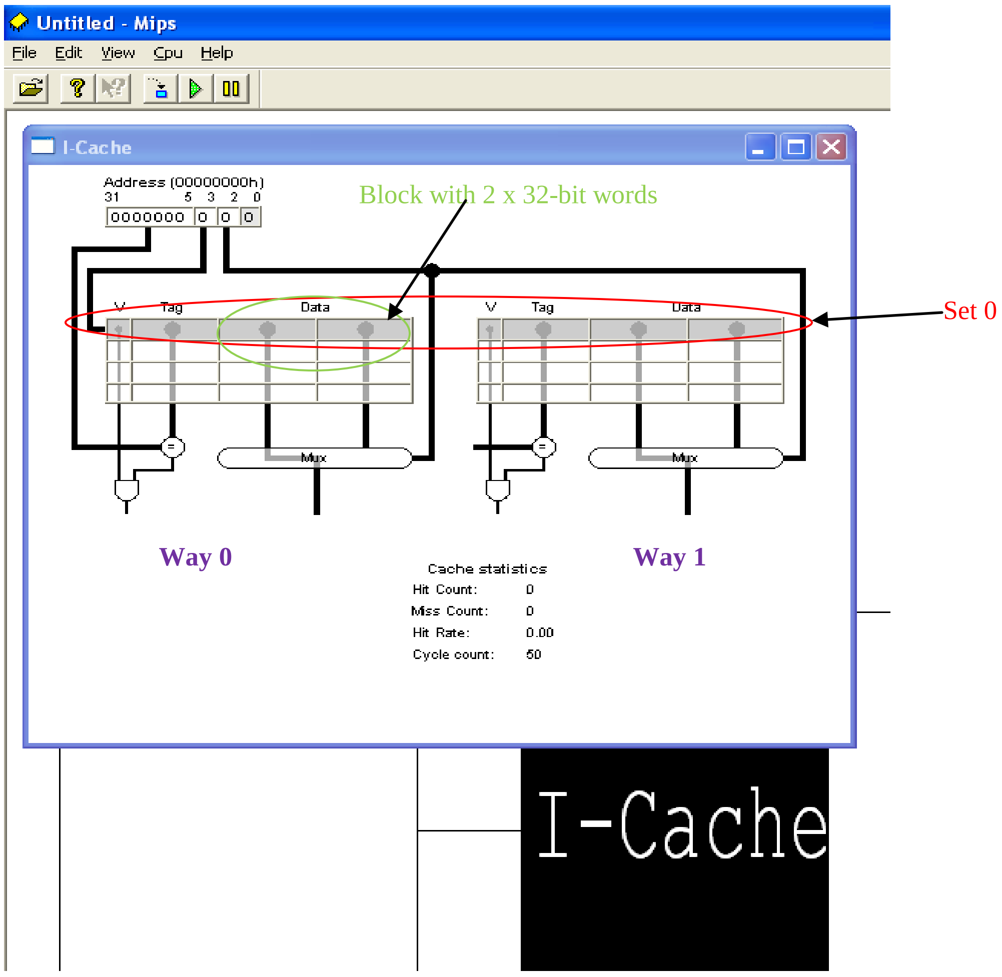
3. Repeat this for the D-Cache.
4. Modify the program you used in part 1 of this lab by adding a *for loop* after the *z=x+y* instruction. The *for loop* should consist of a single instruction *z = x+z;* executed 10 times.
5. While still in MipsIt, view the assembly version of the C program by selecting Build/View Assembler. See below.
6. Upload to the simulator.
7. Again identify the core assembly instructions in the RAM of the simulator that are the compiled version of your C program.
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.

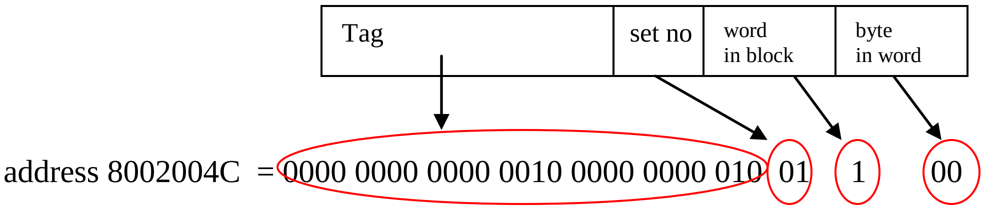


```
Memory
Address    Content       Label    
8001FFFC   00 00 00 00            NOP
80020000   27 BD FF D8   main()   ADDIU   $29, $29, 0xffd8
80020004   AF BF 00 24            SW      $31, 0x24($29)
80020008   AF BE 00 20            SW      $30, 0x20($29)
8002000C   0C 00 80 CD            JAL     0x80cd
80020010   03 A0 F0 21            ADDU    $30, $29, $00
80020014   24 02 00 09            ADDIU   $02, $00, 0x9
80020018   AF C2 00 10            SW      $02, 0x10($30)
8002001C   24 02 00 08            ADDIU   $02, $00, 0x8
80020020   AF C2 00 14            SW      $02, 0x14($30)
80020024   8F C2 00 10            LW      $02, 0x10($30)
80020028   8F C3 00 14            LW      $03, 0x14($30)
8002002C   00 00 00 00            NOP
80020030   00 43 10 21            ADDU    $02, $02, $03
80020034   AF C2 00 18            SW      $02, 0x18($30)
80020038   AF C0 00 1C            SW      $00, 0x1c($30)
8002003C   8F C2 00 1C            LW      $02, 0x1c($30)
80020040   00 00 00 00            NOP
80020044   28 43 00 0A            SLTI    $03, $02, 0xa
80020048   14 60 00 03            BNE     $00, $03, 0x3
8002004C   00 00 00 00            NOP
80020050   08 00 80 20            J       0x8020
80020054   00 00 00 00            NOP
● 80020058 8F C2 00 10            LW      $02, 0x10($30)
8002005C   8F C3 00 18            LW      $03, 0x18($30)
80020060   00 00 00 00            NOP
80020064   00 43 10 21            ADDU    $02, $02, $03
LUI  $02, 0x2010       ; $2=0
Address mode:  Virtual      View mode:  Assembler
```

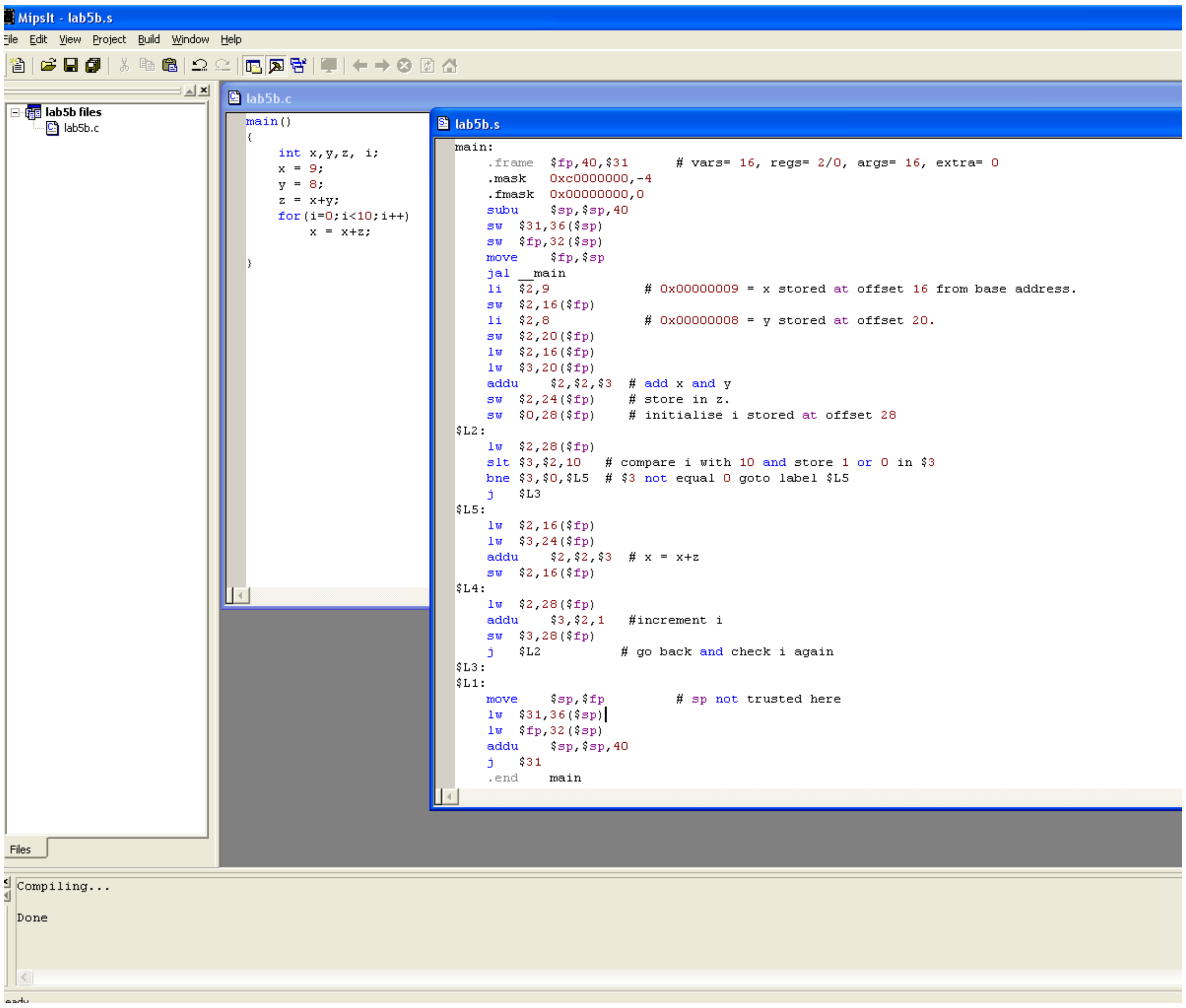
Initialise for loop counter to 0



Cache statistics
Hit Count: 0
Miss Count: 0
Hit Rate: 0.00
Cycle count: 50


Tag
set no
word in block
byte in word
address 8002004C = 0000 0000 0000 0010 0000 0000 010 01 1 00

MSB of address not used.
Tag = 0001002
Set = 1
Word = 1
Byte = 0 not used in instruction cache (greyed out)



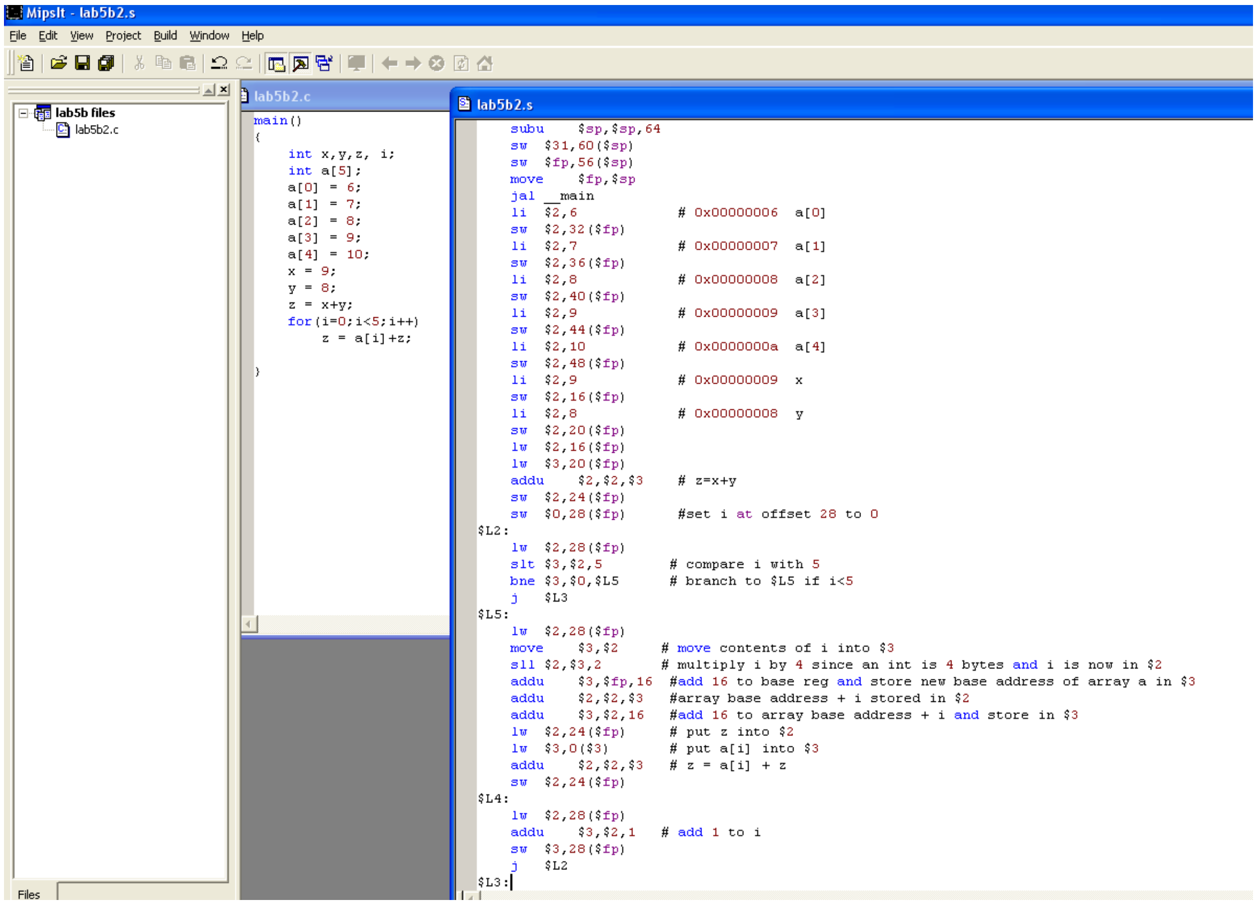
```
main()
{
     int x,y,z, i;
     x = 9;
     y = 8;
     z = x+y;
     for(i=0;i<10;i++)
          x = x+z;

}
```

```
main:
     .frame  $fp,40,$31       # vars= 16, regs= 2/0, args= 16, extra= 0
     .mask   0xc0000000,-4
     .fmask  0x00000000,0
     subu    $sp,$sp,40
     sw  $31,36($sp)
     sw  $fp,32($sp)
     move    $fp,$sp
     jal __main
     li  $2,9              # 0x00000009 = x stored at offset 16 from base address.
     sw  $2,16($fp)
     li  $2,8              # 0x00000008 = y stored at offset 20.
     sw  $2,20($fp)
     lw  $2,16($fp)
     lw  $3,20($fp)
     addu    $2,$2,$3  # add x and y
     sw  $2,24($fp)    # store in z.
     sw  $0,28($fp)    # initialise i stored at offset 28
$L2:
     lw  $2,28($fp)
     slt $3,$2,10   # compare i with 10 and store 1 or 0 in $3
     bne $3,$0,$L5  # $3 not equal 0 goto label $L5
     j   $L3
$L5:
     lw  $2,16($fp)
     lw  $3,24($fp)
     addu    $2,$2,$3  # x = x+z
     sw  $2,16($fp)
$L4:
     lw  $2,28($fp)
     addu    $3,$2,1   #increment i
     sw  $3,28($fp)
     j   $L2          # go back and check i again
$L3:
$L1:
     move    $sp,$fp       # sp not trusted here
     lw  $31,36($sp)
     lw  $fp,32($sp)
     addu    $sp,$sp,40
     j   $31
     .end    main
```

```
Compiling...

Done
```



```c
main()
{
    int x,y,z, i;
    int a[5];
    a[0] = 6;
    a[1] = 7;
    a[2] = 8;
    a[3] = 9;
    a[4] = 10;
    x = 9;
    y = 8;
    z = x+y;
    for(i=0;i<5;i++)
        z = a[i]+z;

}
```

```
    subu    $sp,$sp,64
    sw  $31,60($sp)
    sw  $fp,56($sp)
    move    $fp,$sp
    jal __main
    li  $2,6            # 0x00000006  a[0]
    sw  $2,32($fp)
    li  $2,7            # 0x00000007  a[1]
    sw  $2,36($fp)
    li  $2,8            # 0x00000008  a[2]
    sw  $2,40($fp)
    li  $2,9            # 0x00000009  a[3]
    sw  $2,44($fp)
    li  $2,10           # 0x0000000a  a[4]
    sw  $2,48($fp)
    li  $2,9            # 0x00000009  x
    sw  $2,16($fp)
    li  $2,8            # 0x00000008  y
    sw  $2,20($fp)
    lw  $2,16($fp)
    lw  $3,20($fp)
    addu    $2,$2,$3    # z=x+y
    sw  $2,24($fp)
    sw  $0,28($fp)      #set i at offset 28 to 0
$L2:
    lw  $2,28($fp)
    slt $3,$2,5        # compare i with 5
    bne $3,$0,$L5      # branch to $L5 if i<5
    j   $L3
$L5:
    lw  $2,28($fp)
    move    $3,$2     # move contents of i into $3
    sll $2,$3,2       # multiply i by 4 since an int is 4 bytes and i is now in $2
    addu    $3,$fp,16  #add 16 to base reg and store new base address of array a in $3
    addu    $2,$2,$3   #array base address + i stored in $2
    addu    $3,$2,16   #add 16 to array base address + i and store in $3
    lw  $2,24($fp)     # put z into $2
    lw  $3,0($3)       # put a[i] into $3
    addu    $2,$2,$3   # z = a[i] + z
    sw  $2,24($fp)
$L4:
    lw  $2,28($fp)
    addu    $3,$2,1   # add 1 to i
    sw  $3,28($fp)
    j   $L2
$L3:
```