Task 1-3

.set noreorder # Avoid reordering instructions

.text # Start generating instructions

.globl start # The label should be globally known

.set noat

.ent start # The label marks an entry point

start: addi $1, $0, 0x2 # Load the value 2

addi $3, $0, 0x1 # Load the value 1

addi $6, $0, 0x3 # Load the value 3

addi $5, $0, 0x4 # Load the value 4

sub $2, $1, $3

and $12, $2, $5

or $13, $6, $2

add $14, $2, $2

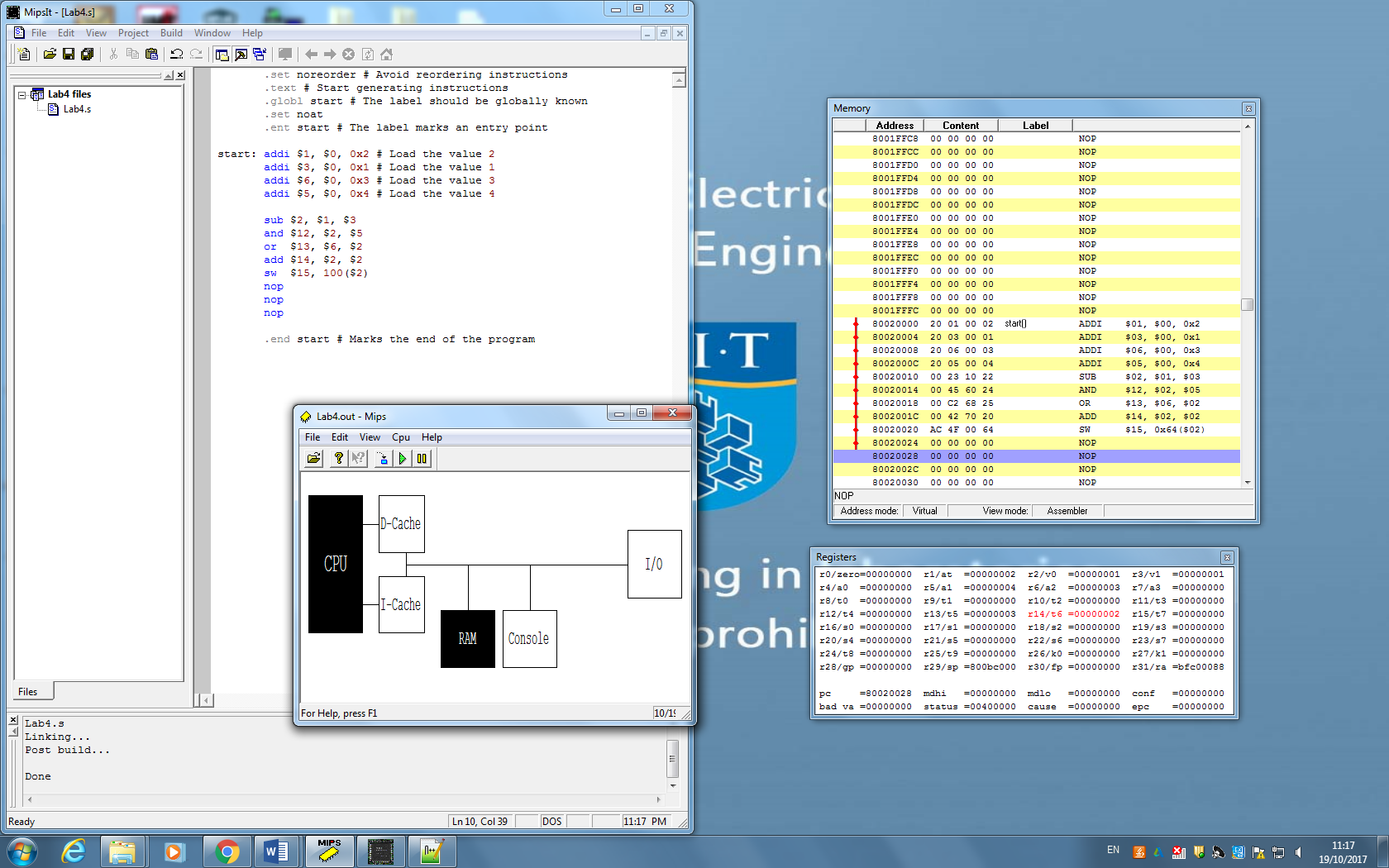
sw $15, 100($2)

nop

nop

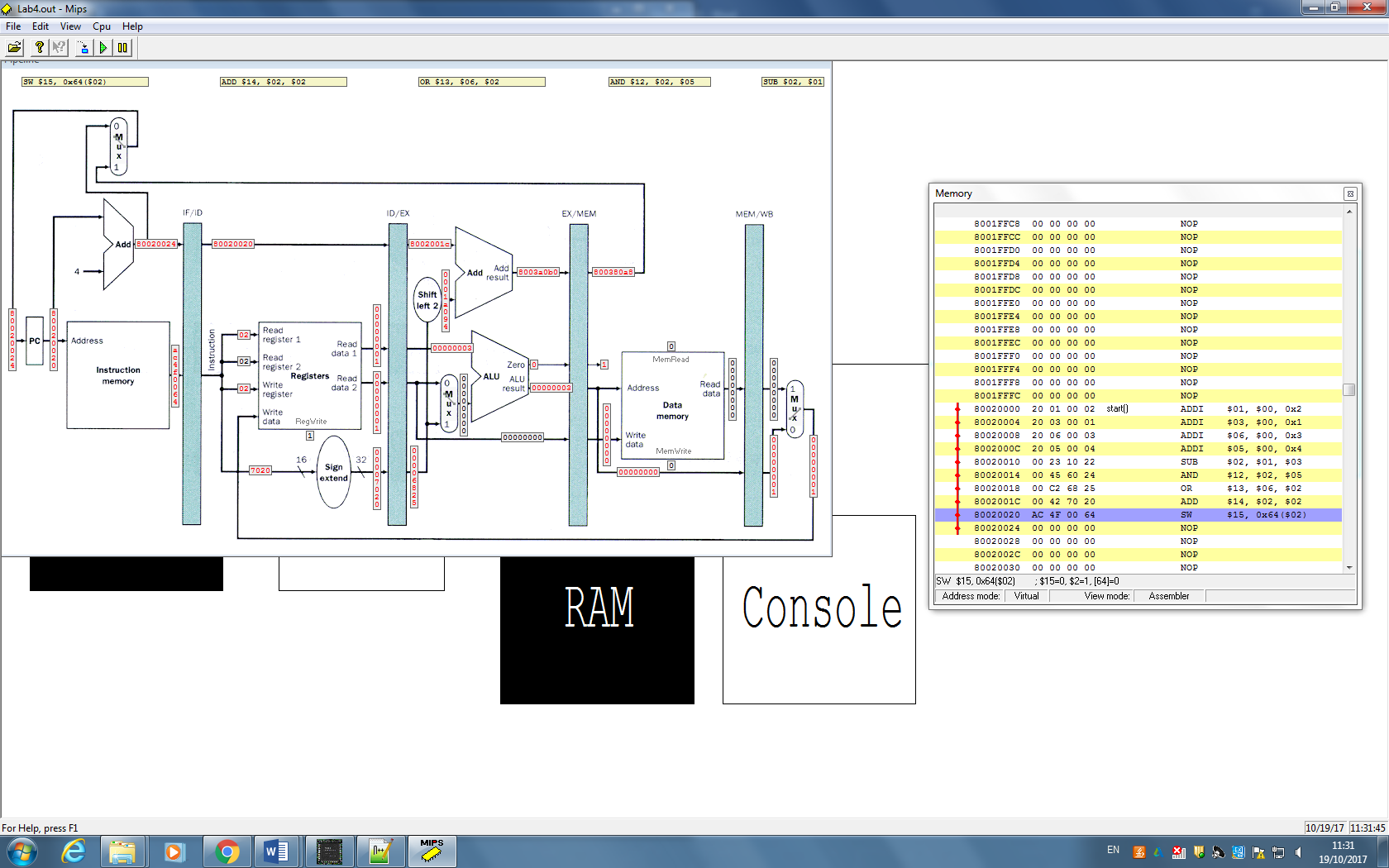
nop

.end start # Marks the end of the program



Task 4-6

Result: due to pipelining the results differ from the previous tasks.



Task 7

Hazards: sub $2, $1, $3

and $12, $2, $5

Hazard because register $2 result is writing in at 5th cycle of SUB instruction while and is trying to access register 2 at 3rd cycle of SUB instruction (2nd cycle of and instruction – decode)

and $12, $2, $5

or $13, $6, $2

Similarly for register $2 when OR instruction is trying to access the register before the AND instruction has been written by and instruction.

Chart showing hazards:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sub | f | d | e | m | **w** |  |  |  |  |
| and |  | f | **d** | e | m | w |  |  |  |
| or |  |  | f | **d** | e | m | **w** |  |  |
| add |  |  |  | f | **d** | e | m | w |  |
| sw |  |  |  |  | f | **d** | e | m | w |

Task 8-11

.set noreorder # Avoid reordering instructions

.text # Start generating instructions

.globl start # The label should be globally known

.set noat

.ent start # The label marks an entry point

start: addi $1, $0, 0x2 # Load the value 2

addi $3, $0, 0x1 # Load the value 1

addi $6, $0, 0x3 # Load the value 3

addi $5, $0, 0x4 # Load the value 4

sub $2, $1, $3

nop

nop

nop

and $12, $2, $5

or $13, $6, $2

nop

nop

nop

add $14, $2, $2

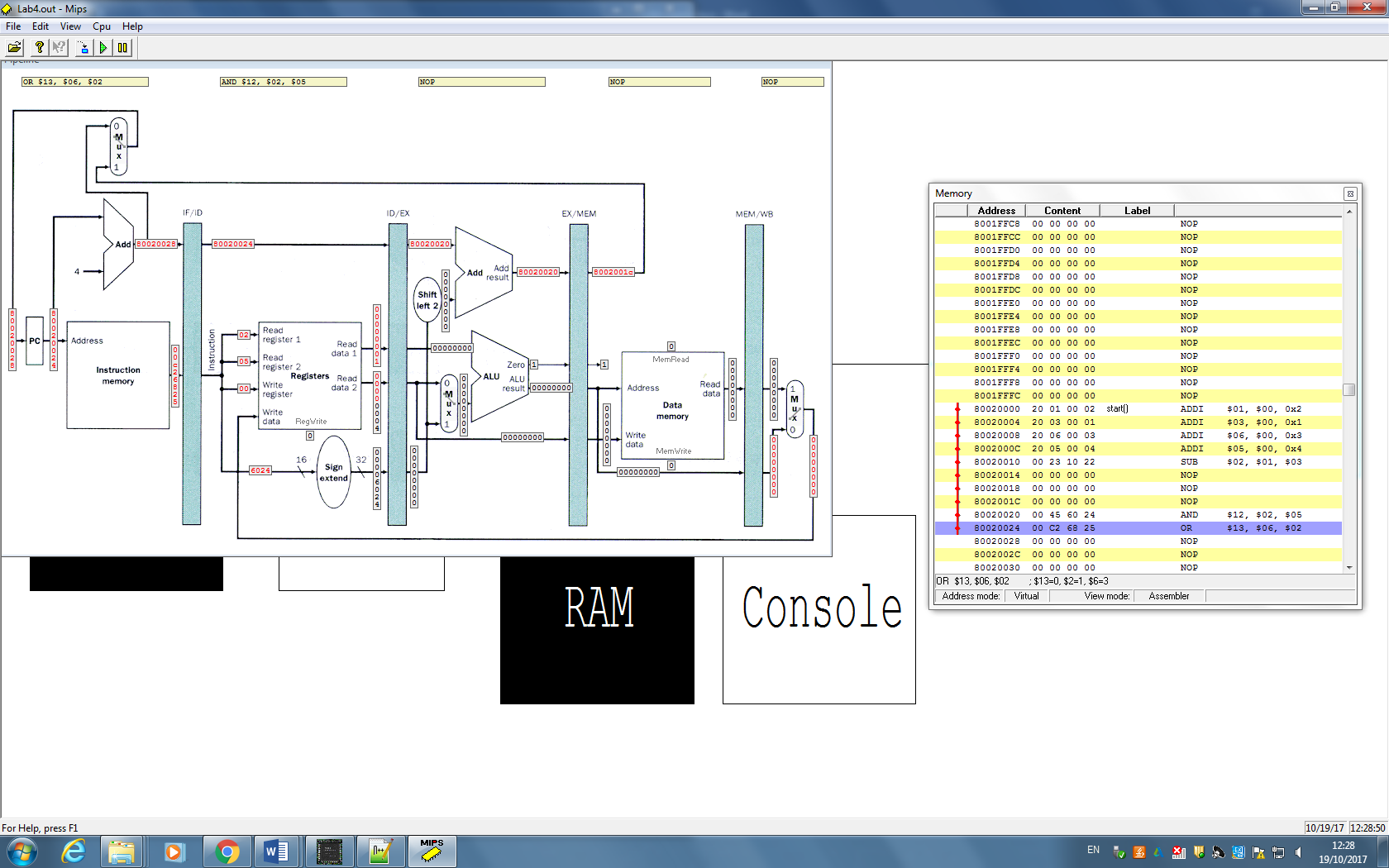
sw $15, 100($2)

nop

nop

nop

.end start # Marks the end of the program



As seen from the diagram, after adding the no operation instruction at the correct places (and for the correct number of times – 3 in our case) the correct values are forwarded. E.g: After subtracting reg1 from reg3 we save this result in reg2 (2-1=1=reg2), then we have 3 nops to avoid data hazards when AND is trying to access reg2 (AND inst waits until reg2 is been written to and only then it decodes AND inst.)

In similar fashion we eliminate other data hazards.