Digital Design and Computer Architecture LU

Lab Protocol

Exercise III

Group 14

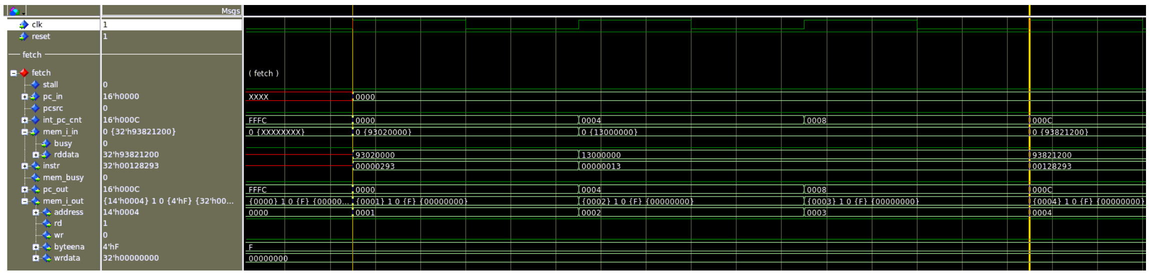
Anna Balla, Matr. Nr. 117709469

e11709469@student.tuwien.ac.at

Philipp Geisler, Matr. Nr. 11775812

e11775812@student.tuwien.ac.at

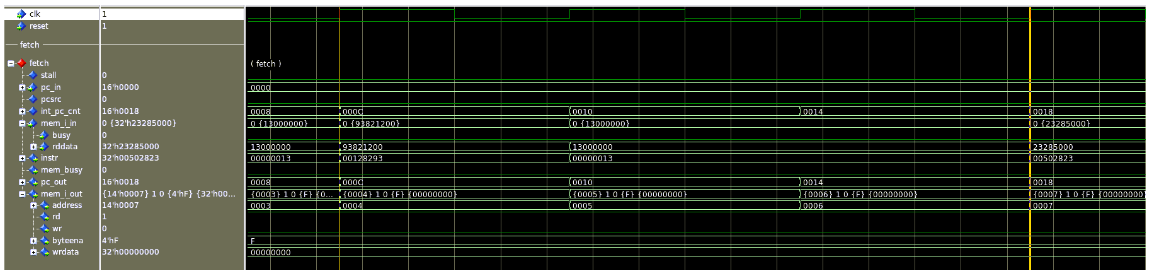
Vienna, June 29, 2020



```
clk              1
reset            1
fetch
fetch
  stall          0
  pc_in          16'h0000
  pcsrc          0
  int_pc_cnt     16'h000C
  mem_i_in       0 {32'h93821200}
    busy         0
    rddata       32'h93821200
  instr          32'h00128293
  mem_busy       0
  pc_out         16'h000C
  mem_i_out      {14'h0004} 1 0 {4'hF} {32'h00...
    address      14'h0004
    rd           1
    wr           0
    byteena      4'hF
    wrdata       32'h00000000
```

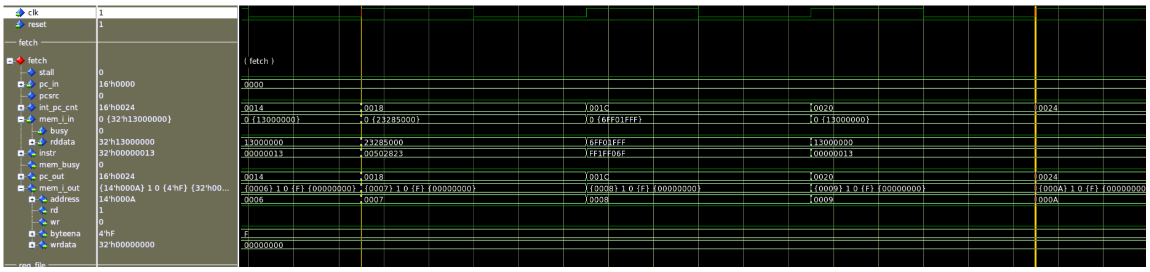
Figure 1: Simulation screenshot for Listing 1. 0.015 us to 0.045 us.

Make sure the following signals are visible in Figure 1 and the signal values are readable: the program counter in the fetch stage, the instruction being fetched, the content of register `x5` and the fields `address`, `rd`, `wr`, `byteena`, and `wrdata` in the `mem_out` signal coming out of the pipeline in the memory stage.



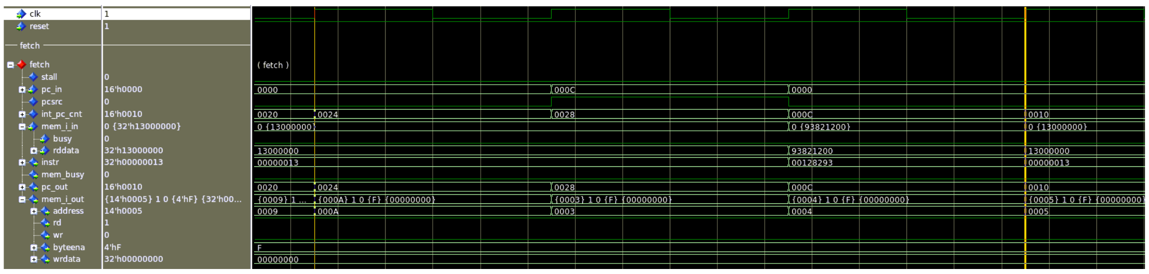
```
clk              1
reset            1
fetch
fetch
  stall          0
  pc_in          16'h0000
  pcsrc          0
  int_pc_cnt     16'h0018
  mem_i_in       0 {32'h23285000}
    busy         0
    rddata       32'h23285000
  instr          32'h00502823
  mem_busy       0
  pc_out         16'h0018
  mem_i_out      {14'h0007} 1 0 {4'hF} {32'h00...
    address      14'h0007
    rd           1
    wr           0
    byteena      4'hF
    wrdata       32'h00000000
```

Figure 2: Simulation screenshot from 0.045 us after start-up until 0.075 us.



```
clk              1
reset            1
fetch
fetch
  stall          0
  pc_in          16'h0000
  pcsrc          0
  int_pc_cnt     16'h0024
  mem_i_in       0 {32'h13000000}
    busy         0
    rddata       32'h13000000
  instr          32'h00000013
  mem_busy       0
  pc_out         16'h0024
  mem_i_out      {14'h000A} 1 0 {4'hF} {32'h00...
    address      14'h000A
    rd           1
    wr           0
    byteena      4'hF
    wrdata       32'h00000000
reg_file
```

Figure 3: Simulation screenshot from 0.075 us after start-up until 0.105 us.



```
clk              1
reset            1
fetch
fetch
  stall          0
  pc_in          16'h0000
  pcsrc          0
  int_pc_cnt     16'h0010
  mem_i_in       0 {32'h13000000}
    busy         0
    rddata       32'h13000000
  instr          32'h00000013
  mem_busy       0
  pc_out         16'h0010
  mem_i_out      {14'h0005} 1 0 {4'hF} {32'h00...
    address      14'h0005
    rd           1
    wr           0
    byteena      4'hF
    wrdata       32'h00000000
```

Figure 4: Simulation screenshot from 0.105 us after start-up until 0.135 us.

The figures 1, 2, 3 and 4 show one iteration of the whole code presented in *submission.S* and the behavior the code triggers in the fetch stage.

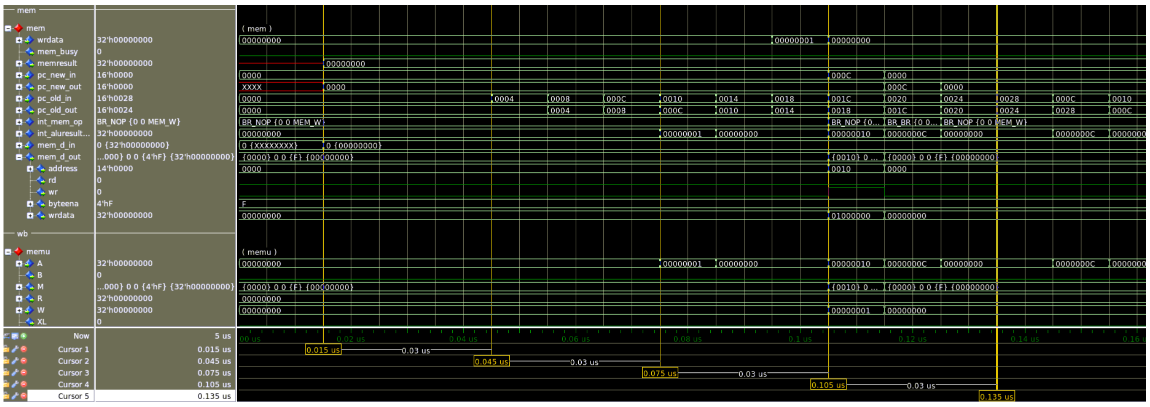


Figure 5: Simulation screenshot showing the behavior of the mem stage during one iteration.

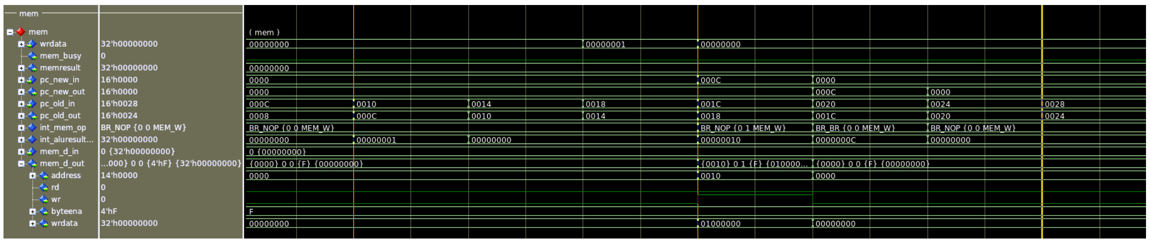


Figure 6: Simulation screenshot from 0.075 us to 0.135 us of the mem stage.

The figures 5 and 6 show whats going on in the mem stage.

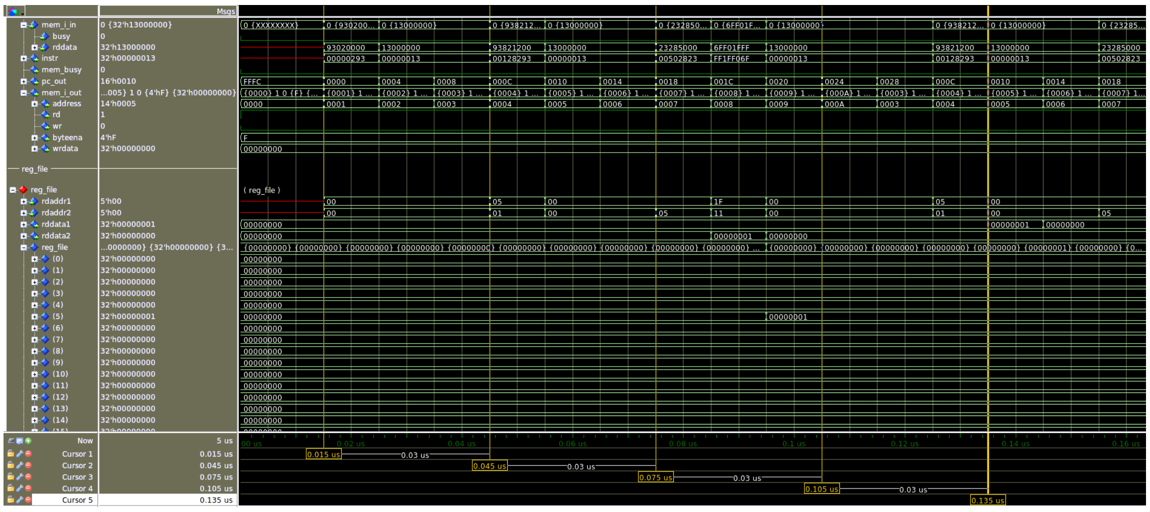


Figure 7: Simulation screenshot from 0 us after start-up until 0.135 us.

Figure 7 shows what's going on in the regfile during the iteration. One can clearly see that the output of the addi instruction is written back to the regfile. If you simulate longer than the first iteration one can see that the assembly-code essential implements a counter that is being incremented.

Listing 1: Assembler example without forwarding

```
        addi x5, x0, 0
        nop
        nop
loop:
        addi x5, x5, 1
        nop
        nop
        sw x5, 16(x0)
        jal x0, loop
        nop
        nop
        nop
```