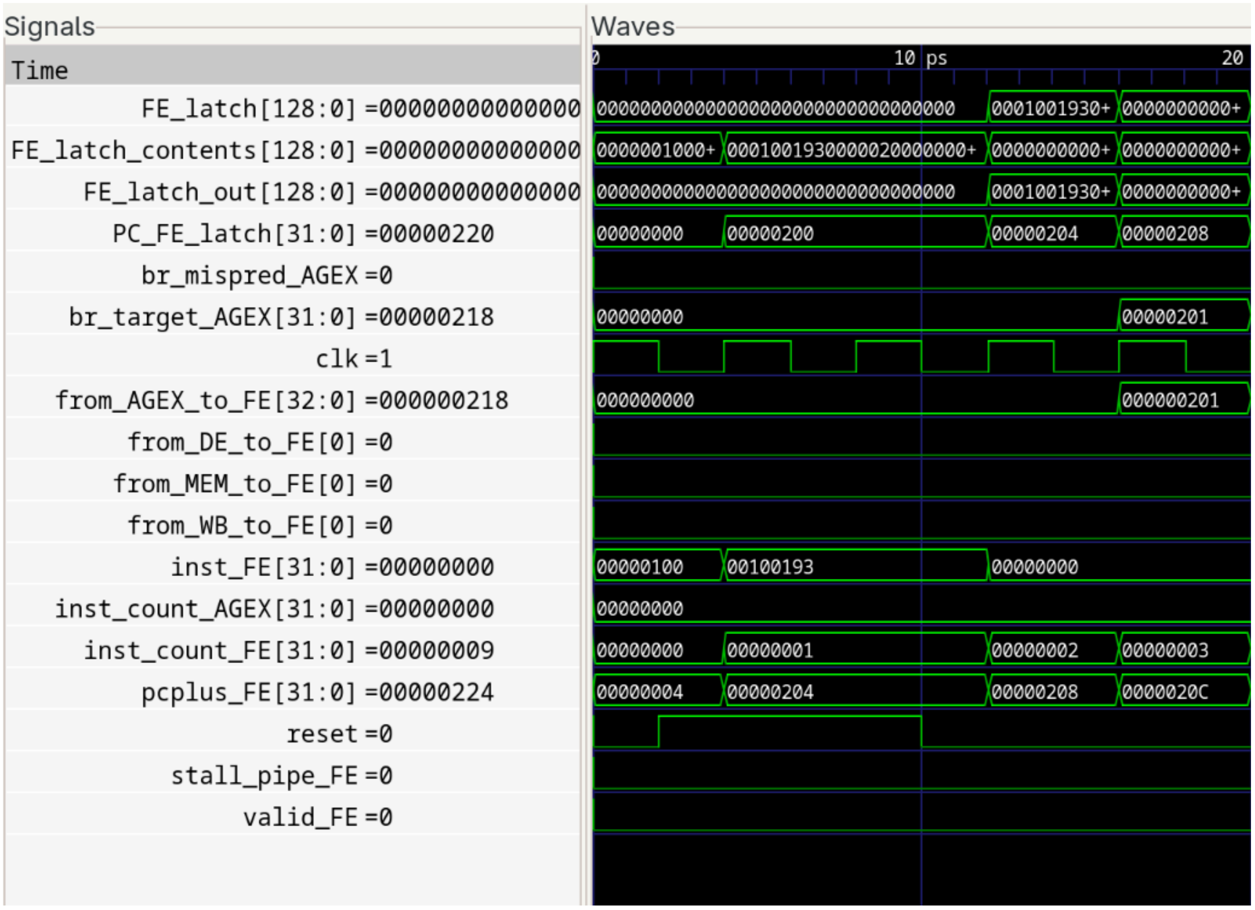
Part 1.2:
The test case for this question is simply executing the following instruction:

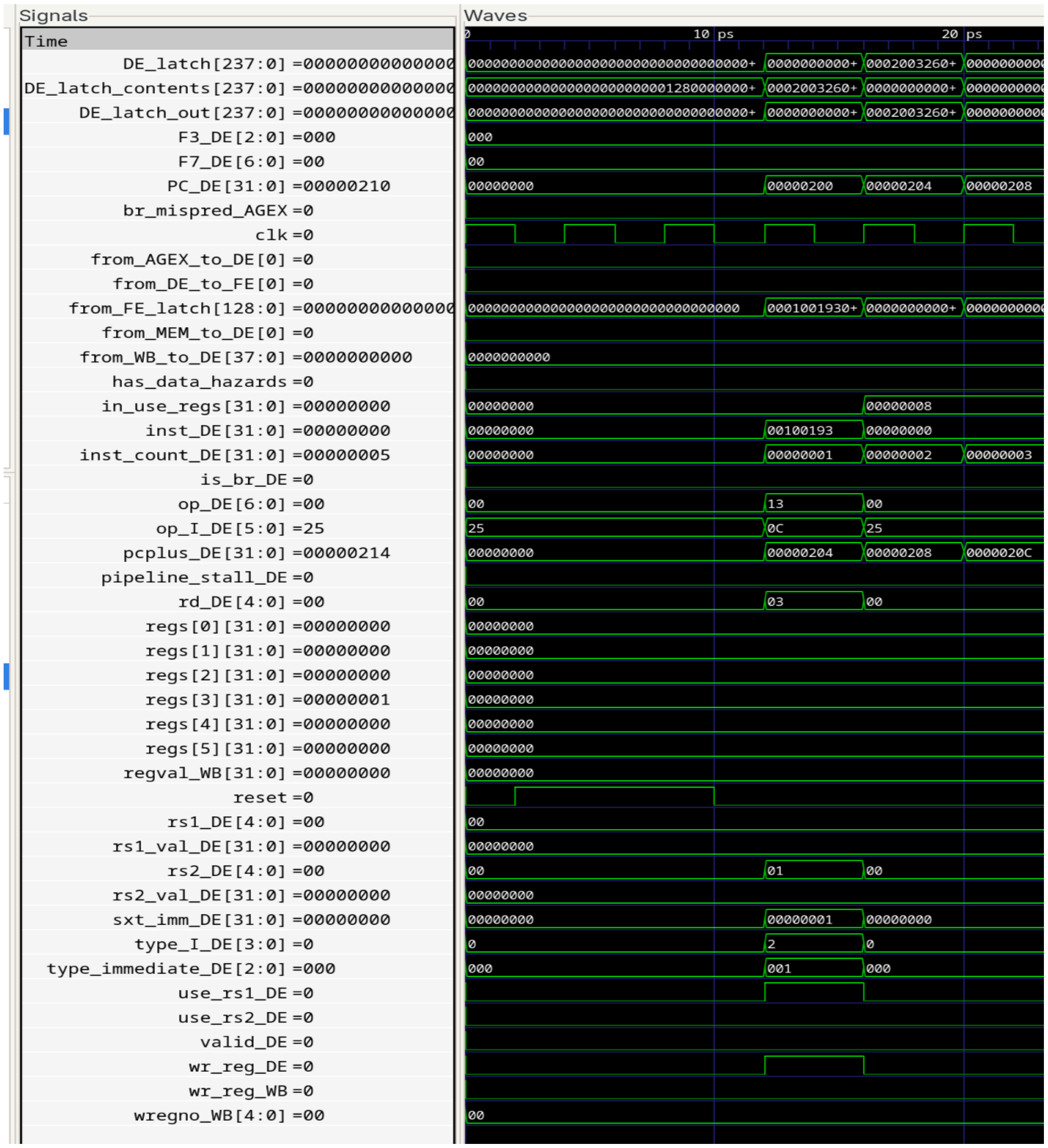
addi gp , x0,   1

This is adding an immediate value 1 to the zero register, and storing the result (1) in register gp.



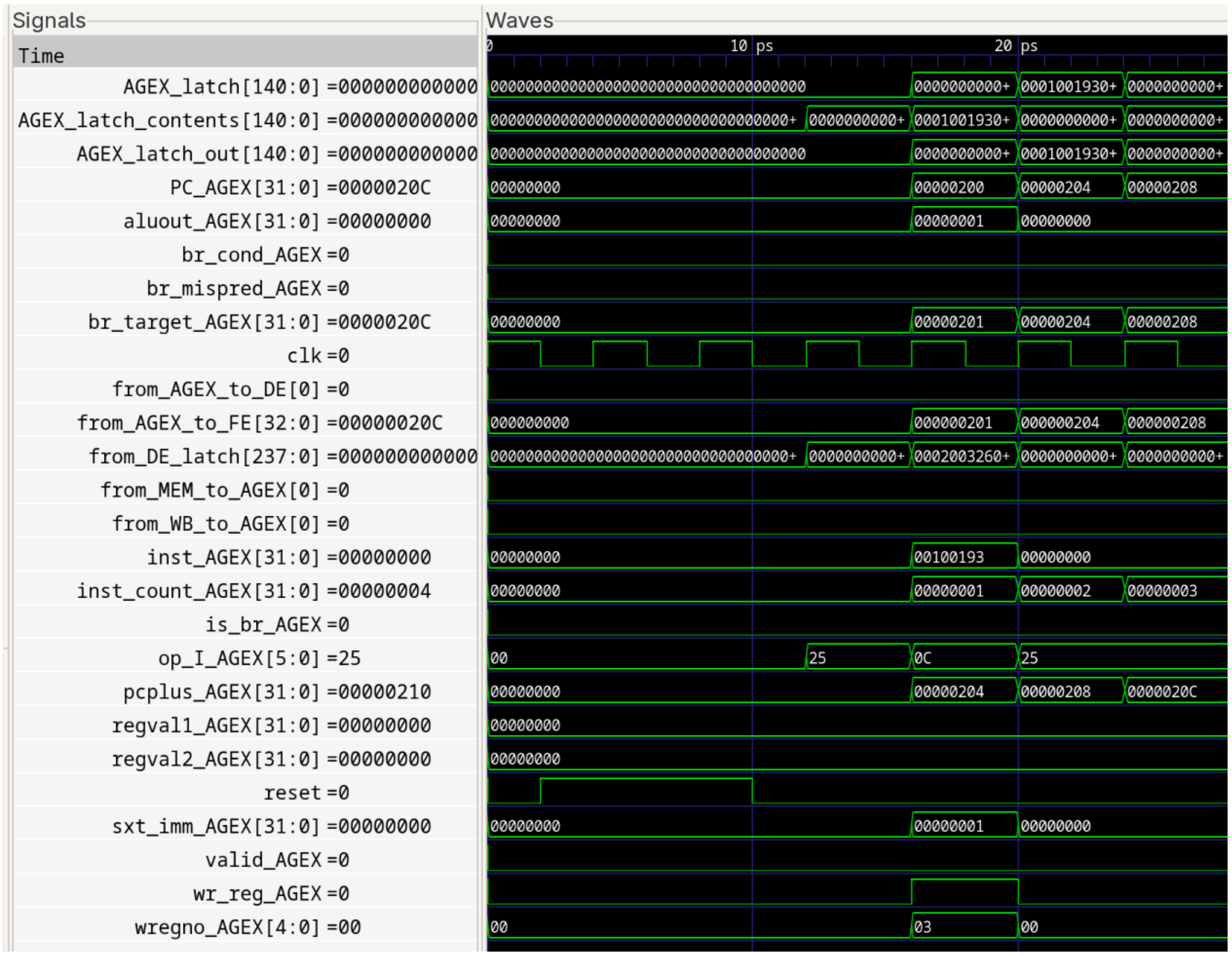
```
Signals
Time
FE_latch[128:0] =00000000000000
FE_latch_contents[128:0] =00000000000000
FE_latch_out[128:0] =00000000000000
PC_FE_latch[31:0] =00000220
br_mispred_AGEX =0
br_target_AGEX[31:0] =00000218
clk =1
from_AGEX_to_FE[32:0] =000000218
from_DE_to_FE[0] =0
from_MEM_to_FE[0] =0
from_WB_to_FE[0] =0
inst_FE[31:0] =00000000
inst_count_AGEX[31:0] =00000000
inst_count_FE[31:0] =00000009
pcplus_FE[31:0] =00000224
reset =0
stall_pipe_FE =0
valid_FE =0
```

FETCH
During the fetch stage, the instruction is pulled from memory, and stored into the PC. The computer is configured to start user programs at address 0x200, which you can see gets initialized at wave **PC_FE_latch[31:0]**. Simultaneously, the instruction itself in hexadecimal form gets populated in wave **inst_FE[31:0]**. After these are prepared in the FE_latch_contents, it gets shipped to the DECODE module on the next cycle, shown by **FE_latch_contents[128:0].**


```
Signals                                          Waves
Time                                             0                            10 ps                        20 ps
DE_latch[237:0] =00000000000000                  0000000000000000000000000000000000+  0000000000+  0002003260+  000000000
DE_latch_contents[237:0] =00000000000000         0000000000000000000000001280000000+  0002003260+  0000000000+  000000000
DE_latch_out[237:0] =00000000000000              0000000000000000000000000000000000+  0000000000+  0002003260+  000000000
F3_DE[2:0] =000                                  000
F7_DE[6:0] =00                                   00
PC_DE[31:0] =00000210                            00000000                             00000200     00000204     00000208
br_mispred_AGEX =0
clk =0
from_AGEX_to_DE[0] =0
from_DE_to_FE[0] =0
from_FE_latch[128:0] =00000000000000             0000000000000000000000000000000000   0001001930+  0000000000+  000000000
from_MEM_to_DE[0] =0
from_WB_to_DE[37:0] =0000000000                  0000000000
has_data_hazards =0
in_use_regs[31:0] =00000000                      00000000                                          00000008
inst_DE[31:0] =00000000                          00000000                             00100193     00000000
inst_count_DE[31:0] =00000005                    00000000                             00000001     00000002     00000003
is_br_DE =0
op_DE[6:0] =00                                   00                                   13           00
op_I_DE[5:0] =25                                 25                                   0C           25
pcplus_DE[31:0] =00000214                        00000000                             00000204     00000208     0000020C
pipeline_stall_DE =0
rd_DE[4:0] =00                                   00                                   03           00
regs[0][31:0] =00000000                          00000000
regs[1][31:0] =00000000                          00000000
regs[2][31:0] =00000000                          00000000
regs[3][31:0] =00000001                          00000000
regs[4][31:0] =00000000                          00000000
regs[5][31:0] =00000000                          00000000
regval_WB[31:0] =00000000                        00000000
reset =0
rs1_DE[4:0] =00                                  00
rs1_val_DE[31:0] =00000000                       00000000
rs2_DE[4:0] =00                                  00                                   01           00
rs2_val_DE[31:0] =00000000                       00000000
sxt_imm_DE[31:0] =00000000                       00000000                             00000001     00000000
type_I_DE[3:0] =0                                0                                    2            0
type_immediate_DE[2:0] =000                      000                                  001          000
use_rs1_DE =0
use_rs2_DE =0
valid_DE =0
wr_reg_DE =0
wr_reg_WB =0
wregno_WB[4:0] =00                               00
```

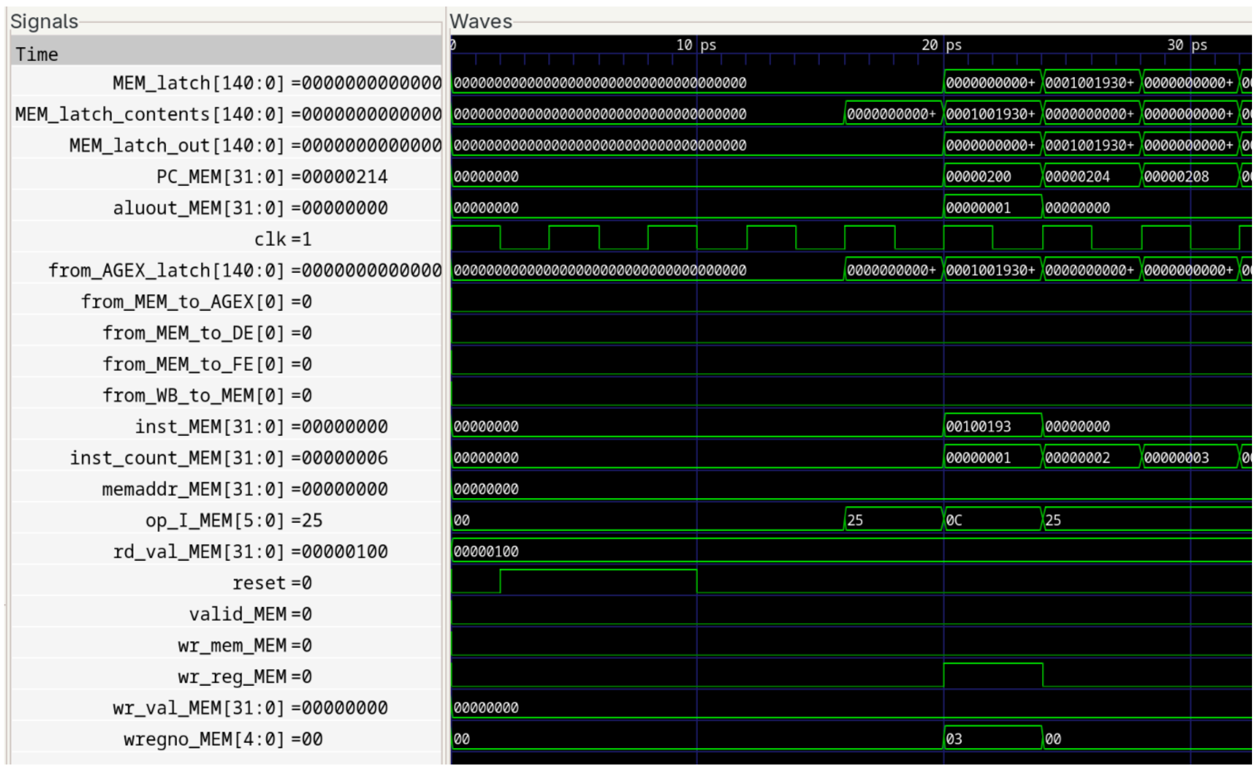
DECODE

During the decode stage, the instruction is decoded, control signals are set, and source registers are read. After the decode module receives the **from_FE_latch[128:0]**, it extracts an opcode (**op_DE[6:0]**) of 0x13, a destination register (**rd_DE[4:0]**) of 3, a source register of 0x0 (**rs1_DE[0:4]**), and an immediate value (**sxt_imm_DE[31:0]**) of 1. All of this is bundled up in **DE_latch_contents[237:0]** and sent off to the AGEX module on the next clock cycle.



```
AGEX_latch[140:0] =000000000000
AGEX_latch_contents[140:0] =000000000000
AGEX_latch_out[140:0] =000000000000
PC_AGEX[31:0] =0000020C
aluout_AGEX[31:0] =00000000
br_cond_AGEX =0
br_mispred_AGEX =0
br_target_AGEX[31:0] =0000020C
clk =0
from_AGEX_to_DE[0] =0
from_AGEX_to_FE[32:0] =00000020C
from_DE_latch[237:0] =000000000000
from_MEM_to_AGEX[0] =0
from_WB_to_AGEX[0] =0
inst_AGEX[31:0] =00000000
inst_count_AGEX[31:0] =00000004
is_br_AGEX =0
op_I_AGEX[5:0] =25
pcplus_AGEX[31:0] =00000210
regval1_AGEX[31:0] =00000000
regval2_AGEX[31:0] =00000000
reset =0
sxt_imm_AGEX[31:0] =00000000
valid_AGEX =0
wr_reg_AGEX =0
wregno_AGEX[4:0] =00
```

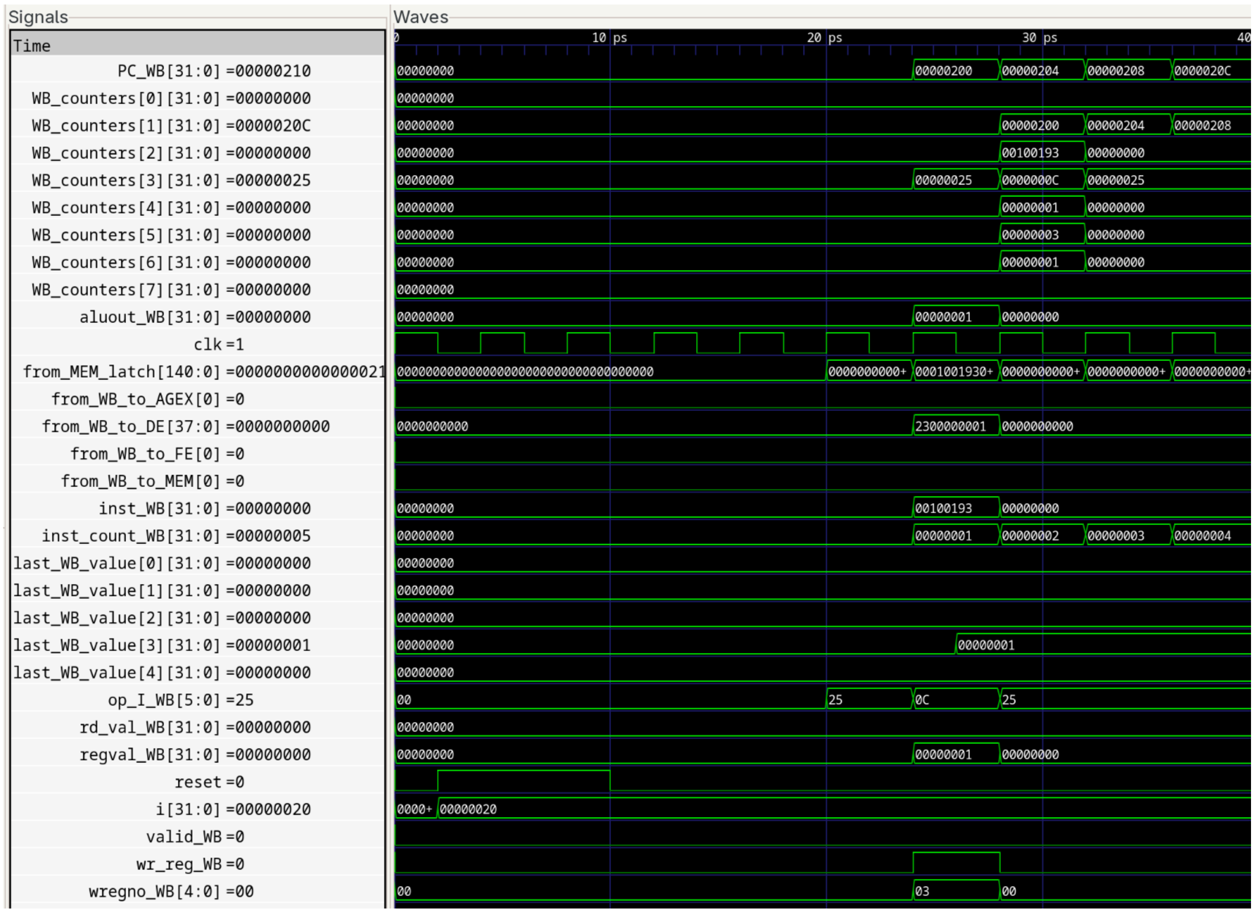
AGEX
In this module, the actual math is completed. After receiving the **from_DE_latch[237:0]**, this uses the alu to combinationally compute the sum, which is 1 and held in **aluout_AGEX[31:0].** It is then shipped off to the **MEMACCESS** stage, via **AGEX_latch_out[140:0].**



```
Signals
Time
MEM_latch[140:0] =0000000000000
MEM_latch_contents[140:0] =0000000000000
MEM_latch_out[140:0] =0000000000000
PC_MEM[31:0] =00000214
aluout_MEM[31:0] =00000000
clk =1
from_AGEX_latch[140:0] =0000000000000
from_MEM_to_AGEX[0] =0
from_MEM_to_DE[0] =0
from_MEM_to_FE[0] =0
from_WB_to_MEM[0] =0
inst_MEM[31:0] =00000000
inst_count_MEM[31:0] =00000006
memaddr_MEM[31:0] =00000000
op_I_MEM[5:0] =25
rd_val_MEM[31:0] =00000100
reset =0
valid_MEM =0
wr_mem_MEM =0
wr_reg_MEM =0
wr_val_MEM[31:0] =00000000
wregno_MEM[4:0] =00
```

MEMACCESS

Nothing notable happens in this portion of the pipeline because ADDI does not access memory. Information is simply passed through to the final stage, **WRITEBACK**.



```
Signals
Time
PC_WB[31:0] =00000210
WB_counters[0][31:0] =00000000
WB_counters[1][31:0] =0000020C
WB_counters[2][31:0] =00000000
WB_counters[3][31:0] =00000025
WB_counters[4][31:0] =00000000
WB_counters[5][31:0] =00000000
WB_counters[6][31:0] =00000000
WB_counters[7][31:0] =00000000
aluout_WB[31:0] =00000000
clk =1
from_MEM_latch[140:0] =0000000000000021
from_WB_to_AGEX[0] =0
from_WB_to_DE[37:0] =0000000000
from_WB_to_FE[0] =0
from_WB_to_MEM[0] =0
inst_WB[31:0] =00000000
inst_count_WB[31:0] =00000005
last_WB_value[0][31:0] =00000000
last_WB_value[1][31:0] =00000000
last_WB_value[2][31:0] =00000000
last_WB_value[3][31:0] =00000001
last_WB_value[4][31:0] =00000000
op_I_WB[5:0] =25
rd_val_WB[31:0] =00000000
regval_WB[31:0] =00000000
reset =0
i[31:0] =00000020
valid_WB =0
wr_reg_WB =0
wregno_WB[4:0] =00
```

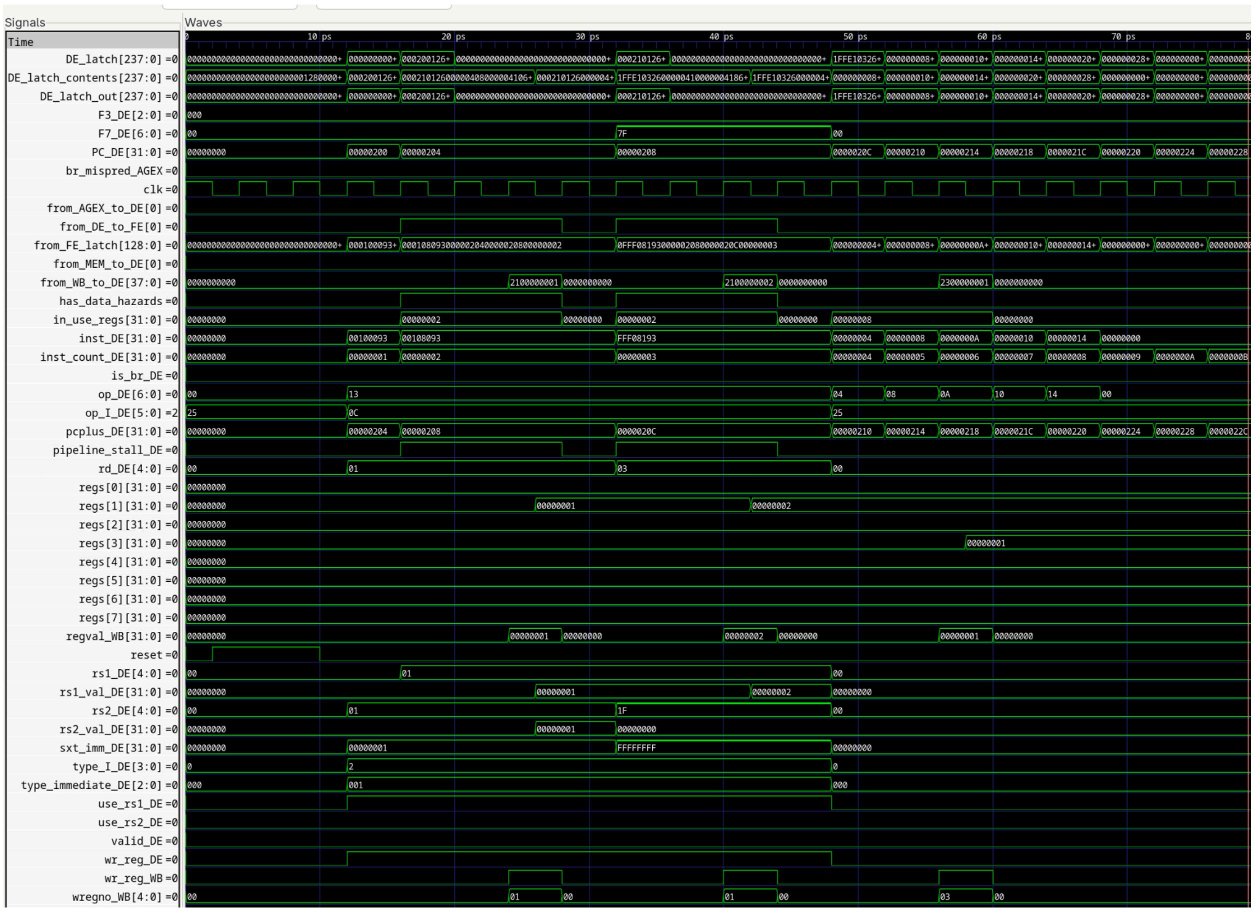
WB

In this stage, the result of the addition is written back to the destination register. **wregno_WB[4:0]** holds the destination register (0x3), and **regval_WB[31:0]** holds the value that we must write back (0x1).

Part 1.3:
This test case executes the following instructions:

```
addi x1 , x0,   1
addi x1 , x1,   1
addi gp , x1,   -1
```

There are two read after write data hazards in this program. To treat the hazards, our processor stalls until the hazard is resolved. Examine the waves below from the decode module:

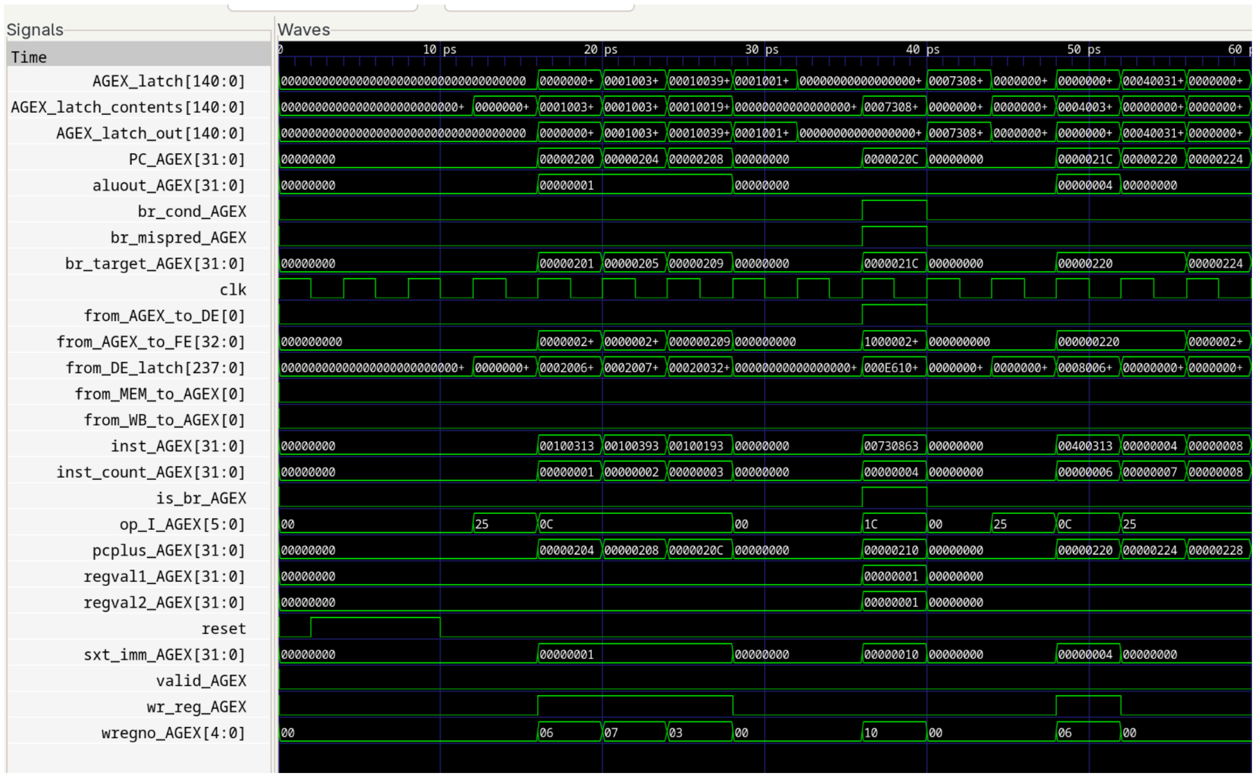


The key waves here are **in_use_regs[31:0], pipeline_stall_DE** and **from_DE_to_FE[0]**. There is logic in the decode module to detect for data hazards. It looks like the following:

```
assign has_data_hazards = (use_rs1_DE && in_use_regs[rs1_DE])
                        || (use_rs2_DE && in_use_regs[rs2_DE]);
```

The Decode stage maintains a bitmask called **in_use_regs** . When an instruction leaves Decode it marks its destination register as "in use." When the next instruction arrives at Decode, it checks if its source registers (rs1 or rs2) are marked as "in use." The FE and DE Stages are stalled until the producer instruction reaches the WRITEBACK stage and writes to the register file. Then, the in_use_regs flag for that register is cleared. The stall signal goes LOW, and the dependent instruction is allowed to proceed.

Part 1.4:



In this lab, the processor uses a static "Never Taken" prediction strategy and always assumes the next instruction is at PC + 4. This branch decision, however, is not resolved until the Execute (AGEX) stage. The ALU compares the operands (regval1 == regval2).
If the branch condition is taken (True), the "Not-Taken" prediction was wrong, and the signal **br_mispred_AGEX** goes high.

In order to recover, the br_mispred_AGEX signal is sent to the FE and DE stages via **from_AGEX_toFE[32:0] and from_AGEX_to_DE[0]**. It acts as a synchronous reset for the pipeline latches, turning the valid instructions currently in Fetch and Decode into NOPs. This effectively kills the wrongly fetched instructions.

The **br_mispred_AGEX** signal also tells the Fetch stage to disregard its current PC logic and load the correct target address (**br_target_AGEX**) into the PC latch for the next cycle.