THE IMAGINATION UNIVERSITY PROGRAMME

RVfpga Lab 17

Superscalar Execution

© Copyright Imagination Technologies

1. INTRODUCTION

As we've already discussed, Western Digital's SweRV EH1 processor is a 9-stage pipelined 32-bit **2-way superscalar** core. In Labs 11-13, we analyzed the flow of the basic instructions through the SweRV processor and the details of each of the pipeline stages, and in Labs 14-16 we looked at how data, control, and structural dependencies are handled in this processor. Now, with that foundational understanding, you are ready to analyse superscalar execution!

NOTE: Before starting this lab, we recommend reading Section 7.7.4 of the book by S. Harris and D. Harris, "*Digital Design and Computer Architecture: RISC-V Edition*", Morgan Kaufmann [DDCARV]. Some of the content in this lab is inspired by that book.

Superscalar execution is a microarchitectural technique that improves the performance of a processor. A superscalar processor contains multiple copies of the datapath hardware to execute multiple instructions simultaneously. The latency of executing a single instruction remains unchanged, but the processor can execute and commit more instructions per cycle, thus improving its throughput. Figure 1 shows an example block diagram of a two-way superscalar processor extracted from [DDCARV].

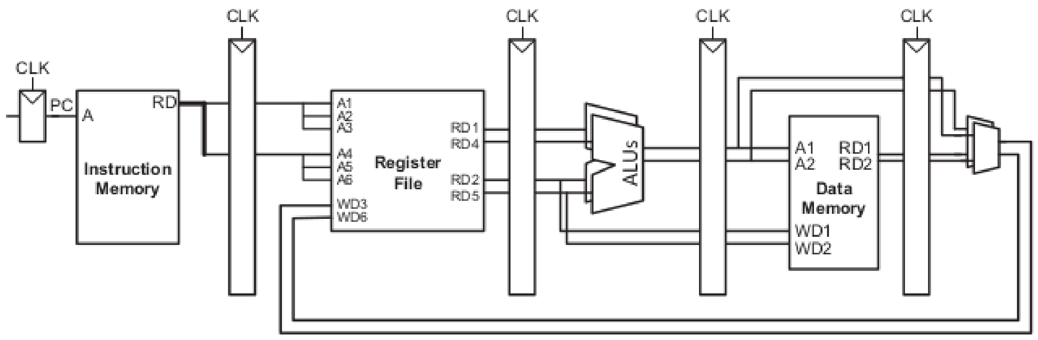


Figure 1. Figure 7.68 from [DDCARV]: Two-way superscalar processor block diagram

SweRV EH1 is a 2-way superscalar processor similar to the one shown in Figure 1 that can fetch, execute, and commit up to two instructions per cycle. The datapath aligns two instructions at a time that come from instruction memory. The multi-ported register file can read up to four source operands and write two values back in each cycle (plus one more value coming from a non-blocking load, as we analysed in Lab 15). The SweRV EH1 processor contains two Integer Pipes, one Multiply Pipe, one Load-Store Pipe, and one non-pipelined Divider. All these pipes are completely independent, so a pair of independent arithmetic-logic (AL) instructions or any pair of two independent different instructions can be executed simultaneously. However, as discussed in Lab 14, a pair of multiplications, divisions, or load/store instructions cannot be executed in the same cycle, because there is only one of each of these pipes in the processor, and thus two sequential equal non-AL instructions will lead to a structural hazard.

SweRV EH1 does not include support for dynamic instruction scheduling with out-of-order execution, except for in the case of non-blocking loads. However, it is possible to statically reorder the code in order to better exploit the resources, including the two ways of the pipeline. Ideally, in a 2-way superscalar processor such as SweRV EH1, throughput (IPC) would double when compared to a single-issue design. Unfortunately, actual programs don't typically achieve that ideal: in real programs, performance typically improves by 1.3x-1.5x when going from 1- to 2-way processors; however, adding the second way requires much more hardware.

In Section 2, we analyse two simple programs, comparing the behaviour when using single-issue and dual-issue configurations of SweRV EH1. Then, in Section 3, we propose several exercises related to superscalar execution.

2. SINGLE-ISSUE VS. DUAL-ISSUE

In this section, we work with two simple programs: the first one (Section 2.A) contains a loop with four AL instructions and the second one (Section 2.B) contains a loop with two multiplications interleaved with two AL instructions.

A. Four independent AL instructions

In this section, we compare performance of the program from Figure 2 running on the single- or dual-issue SweRV EH1 core. Recall that, in Appendix B of Lab 11, we described how the different core features (pipelined execution, branch prediction, superscalar, etc.) can be configured.

The program contains a loop that performs 1,000,000 (0xF4240) iterations; the loop body contains four AL independent instructions (`add`, `sub`, `or`, and `xor`), surrounded by nop instructions that allow us to see each iteration isolated from the others. Folder *[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions* provides the PlatformIO project that you can analyse, simulate, and modify as desired.

```
.globl Test_Assembly

.text

Test_Assembly:

li t2, 0x400            # Disable Dual-Issue Execution
csrrs t1, 0x7F9, t2

li t0, 0x0
li t1, 0x1
li t2, 0x1
li t3, 0x3
li t4, 0x4
li t5, 0x5
li t6, 0x6

lui t2, 0xF4
add t2, t2, 0x240

REPEAT:
  add t0, t0, 1
  INSERT_NOPS_10
  INSERT_NOPS_4
  add t3, t3, t1
  sub t4, t4, t1
  or  t5, t5, t1
  xor t6, t6, t1
  INSERT_NOPS_10
  INSERT_NOPS_3
  bne t0, t2, REPEAT # Repeat the loop
```

```
.end
```

Figure 2. Program with four AL instructions

In Section 2.A.i, we analyse the simulation in Verilator and the execution on the Nexys A7 board for the program from Figure 2 in a **single-issue** SweRV EH1 processor. For that purpose, we disable the dual-issue capability using the following two instructions at the beginning of the program:

```
li t2, 0x400
csrrs t1, 0x7F9, t2
```

In Section 2.A.ii, we analyse the simulation in Verilator and the execution on the Nexys A7 board for the program from Figure 2 in a **dual-issue** SweRV EH1 processor. To do so, we simply comment out the two previous instructions.

i. Execution in a single-issue SweRV EH1 processor

In its single-issue configuration, SweRV EH1 simply ignores the second way, executing only one instruction per cycle. Figure 3 illustrates the simulation of the program from Figure 2 in this SweRV EH1 configuration. We pick a random iteration of the *REPEAT* loop (except the first one), given that all iterations are the same.

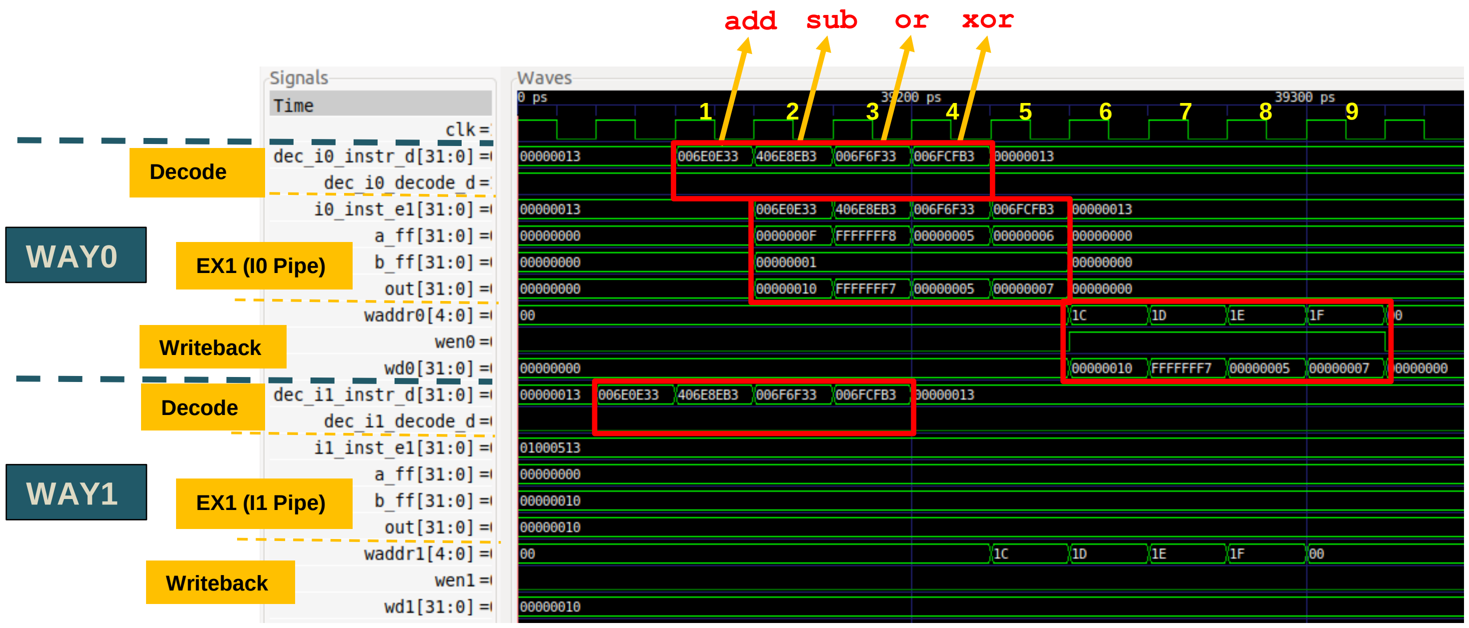


Figure 3. Simulation of the program from Figure 2 in a single-Issue SweRV EH1

The instructions are received in both ways at decode time (see signals `dec_i0_instr_d[31:0]` and `dec_i1_instr_d[31:0]`), but they are only sent to execution in Way 0, because Way 1 is disabled. Signals `dec_i0_decode_d` and `dec_i1_decode_d` determine if the instruction is propagated from the Decode Stage to the EX1 Stage, and signals `i0_inst_e1[31:0]` and `i1_inst_e1[31:0]` contain the instruction in Way 0 and Way 1, respectively, in the E1 Stage.

- **Way 0**:
  - Signal `dec_i0_decode_d` is always 1 in our example; specifically, it is 1 for the four AL instructions under analysis.

© Copyright Imagination Technologies

- The instruction in the Decode Stage (`dec_i0_instr_d[31:0]`) **is propagated** to the I0 Pipe (`i0_inst_e1[31:0]`)

- **Way 1**:
  - Signal `dec_i1_decode_d` is always 0 in our example; specifically, it is 0 for the four AL instructions under analysis.
  - The instruction at the Decode Stage (`dec_i1_instr_d[31:0]`) **is NOT propagated** (`i1_inst_e1[31:0]`) to the Execution Stage.

Accordingly, only the ALU from the I0 Pipe is used (see signals `aff`, `bff` and `out` in both ways) and only write port 0 of the Register File is used (see signals `waddr`, `wen` and `wd` in both ways).

Figure 4 illustrates the flow of the four AL instructions through the I0 Pipe, from the Decode to the Writeback (EX5) stages. It shows the nine cycles (1 to 9) specified in Figure 3. The empty holes correspond to the nop instructions that surround the four AL instructions, which are not shown in the figure for the sake of simplicity.

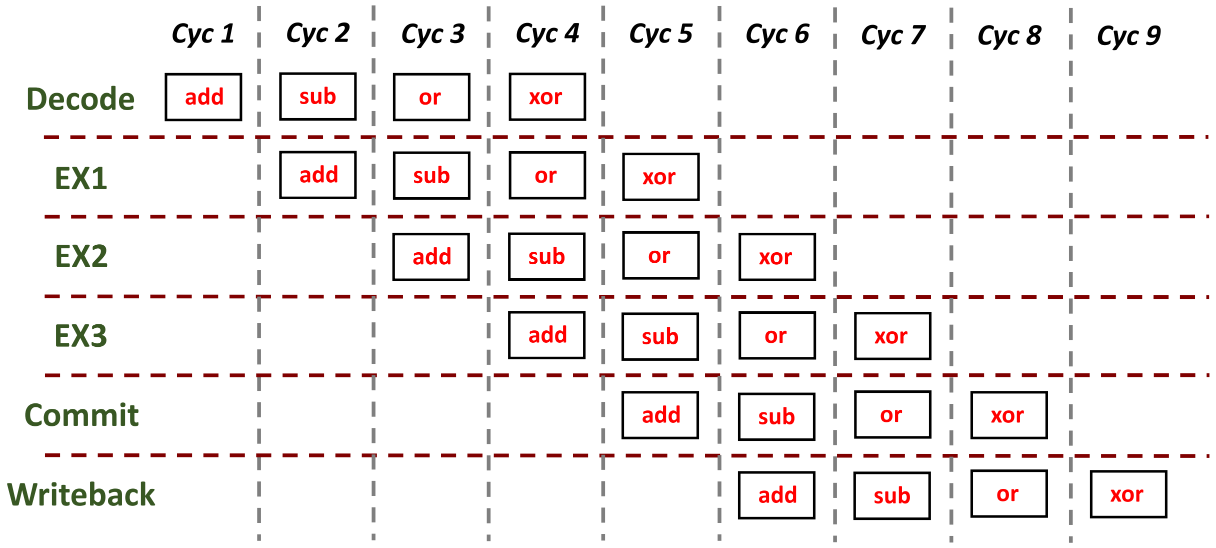


Figure 4. Flow of the 4 AL instructions through I0 Pipe

<u>TASK</u>: In the simulation from Figure 3, include the trace signals and highlight the instructions as they go through the pipeline from the Decode stage to the Writeback stage, similarly to Figure 4. You can use the *.tcl* file provided at: *[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_task1.tcl*.

<u>TASK</u>: Remove all the `nop` instructions within the body of the loop from Figure 2. Repeat the simulation from Figure 3. What is the expected IPC for this program? Execute the program on the board and verify that the IPC obtained is the one that you expected.

ii. Execution in 2-Way Superscalar SweRV EH1

Comment out the two instructions from Figure 2 to enable the dual-issue capability of SweRV EH1. Now, SweRV EH1 will send to execution two instructions per cycle whenever possible. Figure 5 illustrates the simulation of the program. As before, we pick a random iteration of the *REPEAT* loop (except the first one), given that all iterations are the same.

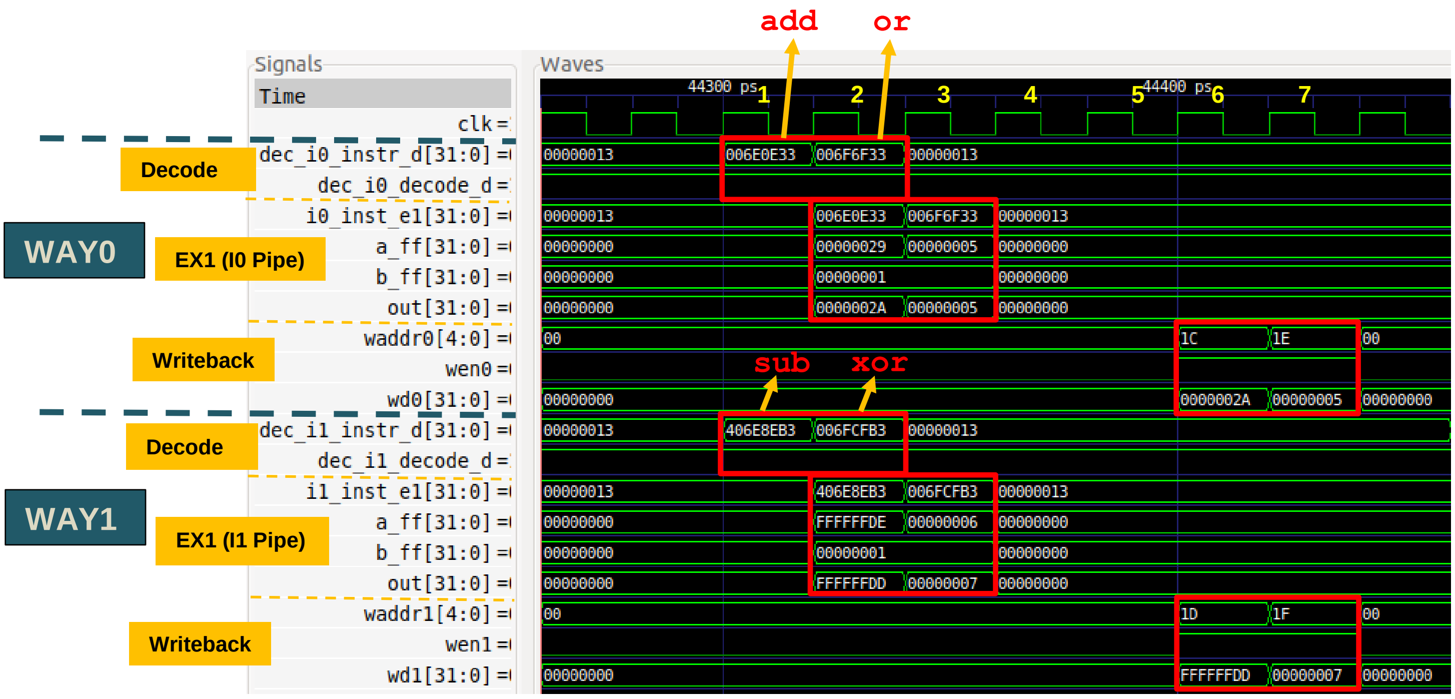


Figure 5. Simulation of the program from Figure 2 in a dual-issue SweRV EH1

Two instructions are received, one in each way, at decode time (see signals `dec_i0_instr_d[31:0]` and `dec_i1_instr_d[31:0]`), and two instructions per cycle are sent to the Execute stages, one through the I0 Pipe and the other through the I1 Pipe.

- **Way 0**:
  - Signal `dec_i0_decode_d` is always 1 – being true for two of the four AL instructions of our example (the other two AL instructions are decoded in Way 1).
  - The instruction in the Decode stage (`dec_i0_instr_d[31:0]`) is propagated to the I0 Pipe (`i0_inst_e1[31:0]`).

- **Way 1**:
  - Signal `dec_i1_decode_d` is always 1 – being true for two of the four AL instructions of our example (the other two AL instructions are decoded in Way 0).
  - The instruction in the Decode stage (`dec_i1_instr_d[31:0]`) is propagated to the I1 Pipe (`i1_inst_e1[31:0]`).

Thus, the ALUs in both pipes (I0 and I1) are used (see signals `aff`, `bff`, and `out` in both ways), and both Register File write ports are used (see signals `waddr`, `wen` and `wd` in both ways).

© Copyright Imagination Technologies

Figure 6 illustrates the flow of the four AL instructions through the I0 Pipe and the I1 Pipe, from the Decode Stage to the EX5 Stage. It shows the seven cycles (1 to 7) specified in Figure 5. The empty holes correspond to the nop instructions that surround the four AL instructions, which are not shown in the figure for the sake of simplicity.

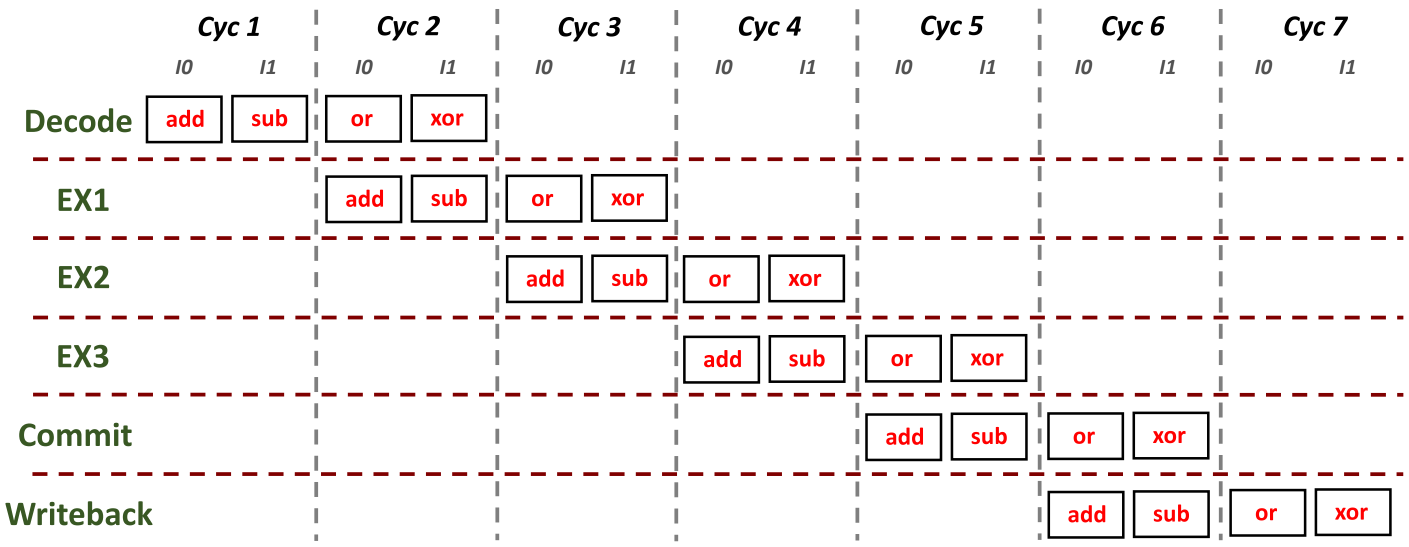


Figure 6. Flow of the 4 instructions through both pipes (I0 and I1)

<u>TASK</u>: In the simulation from Figure 5, add trace signals and highlight the instructions as they go through the pipeline from the Decode to Writeback stages, similar to what's shown in Figure 6. You can use the *.tcl* file provided at: *[RVfpgaPath]/RVfpga/Labs/Lab17/Four_AL_Instructions/test_task2.tcl*.

<u>TASK</u>: Remove all the `nop` instructions within the body of the loop from Figure 2. Repeat the simulation from Figure 5. What is the expected IPC for this program? Execute the program on the board and verify that the IPC obtained is the one that you expected.

B. Two `mul` instructions interleaved with two AL instructions

In this section we analyse the program from Figure 7 on a dual-issue SweRV EH1 core. The program executes a loop that performs 1,000,000 iterations; the loop body contains two `mul` instructions interleaved with two AL instructions (`mul`, `add`, `mul`, and `sub`), surrounded by nop instructions that allow us to see each iteration isolated from the others. The four instructions are independent. We analyse the simulation in Verilator, highlighting the main signals and providing a brief explanation of the program behaviour in each case. Folder *[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL_TwoMUL_Instructions* provides the PlatformIO project that you can analyse, simulate, and modify as desired.

```
.globl Test_Assembly

.text

Test_Assembly:
```

© Copyright Imagination Technologies

```
# li t2, 0x400              # Disable Dual-Issue Execution
# csrrs t1, 0x7F9, t2

li t3, 0x3
li t4, 0x4
li t5, 0x5
li t6, 0x6
li t0, 0x0
lui t1, 0xF4
add t1, t1, 0x240

REPEAT:
  add t0, t0, 1
  INSERT_NOPS_10
  INSERT_NOPS_4
  mul t3, t3, t1
  add t4, t4, t1
  mul t5, t5, t1
  sub t6, t6, t1
  INSERT_NOPS_10
  INSERT_NOPS_3
  bne t0, t1, REPEAT # Repeat the loop

.end
```

Figure 7. Program with 2 `mul` instructions and 2 AL instructions

In this program, the processor will send to execution one `mul` instruction and one AL instruction per cycle, so the Multiply Pipe as well as the I0 or the I1 Pipe are used. Figure 8 illustrates the simulation of the program from Figure 7 on the 2-way superscalar SweRV EH1 processor. We pick a random iteration of the *REPEAT* loop (except the first one), given that all iterations are the same.

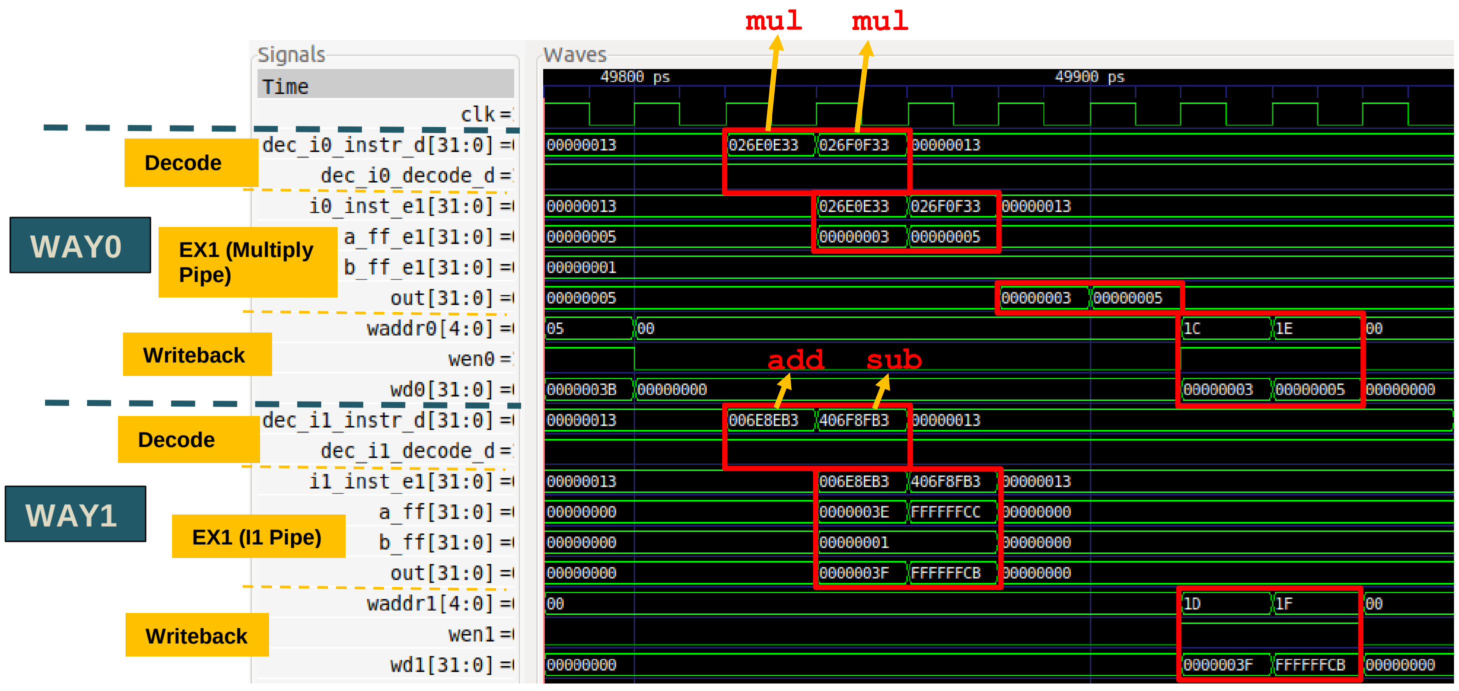


Figure 8. Simulation of the program from Figure 7

© Copyright Imagination Technologies

The instructions are received in both ways at decode time (see signals `dec_i0_instr_d[31:0]` and `dec_i1_instr_d[31:0]`) and are sent to the execution stages in both ways.

- **Way 0**:
  - Signal `dec_i0_decode_d` is always 1 – for two of the four instructions analysed in our example (the other two instructions are decoded in Way 1).
  - The instruction at the Decode Stage (`dec_i0_instr_d[31:0]`) is propagated to the Multiply Pipe (`i0_inst_e1[31:0]`)

- **Way 1**:
  - Signal `dec_i1_decode_d` is always 1 – for two of the four instructions analysed in our example (the other two instructions are decoded in Way 1).
  - The instruction at DECO (`dec_i1_instr_d[31:0]`) is propagated to the I1 Pipe (`i1_inst_e1[31:0]`)

Thus, the ALU from the I1 Pipe and the Multiplier are used (see signals `a_ff_e1`, `b_ff_e1`, and `out` and signals `a_ff`, `b_ff`, and `out`), and both Register File write ports are used (see signals `waddr`, `wen`, and `wd` in both ways).

TASK: In the simulation from Figure 8, add trace signals and highlight the instructions as they go through the pipeline from the Decode to Writeback stages. You can use the *.tcl* file provided at: *[RVfpgaPath]/RVfpga/Labs/Lab17/TwoAL_TwoMUL_Instructions /test_taskMuls.tcl*.

TASK: Remove all the `nop` instructions within the body of the loop from Figure 7. Repeat the simulation from Figure 8. What is the expected IPC for this program? Execute the program on the board and verify that the IPC obtained is the one that you expected.

Repeat the same experiments for the single-issue configuration and compare the results.

3. EXERCISES

1) Create programs similar to those in Figure 2 and Figure 7 using combinations of instructions that show new situations related to dual-issue execution.

2) Analyse the differences between the (dual-issue) SweRV EH1 processor and the example superscalar processor proposed in Section 7.7.4 of the textbook by S. Harris and D. Harris, “Digital Design and Computer Architecture: RISC-V Edition” [DDCARV] (shown in Figure 1 for convenience).

3) Analyse the program from Figure 7.70 in Section 7.7.4 of DDCARV, which is provided in a PlatformIO project in folder *[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_SuperscalarExample*. Run the program on SweRV EH1, both in simulation and on the board (for the latter remove the nop instructions). Explain the results. If necessary, reorder the program trying to obtain the optimal IPC.

© Copyright Imagination Technologies

Next, disable the dual-issue execution as explained in this lab – and in SweRVref.docx (Section 2). Compare the simulation and the results obtained on the board when compared to when the dual-issue feature is enabled.

4) Modify the program from Exercise 3 substituting instruction `add s9, s8, t1` for instruction `add t2, s8, t1`. Explain the results. If necessary, reorder the program to try to obtain the optimal IPC.

   Then disable the dual-issue execution as explained in this lab and in SweRVref.docx (Section 2). Compare the simulation and the results obtained on the board when compared to when the dual-issue feature is enabled.

5) (*The following exercise is based on exercise 4.31 from the book "Computer Organization and Design – RISC-V Edition", by Patterson & Hennessy ([HePa]).*)

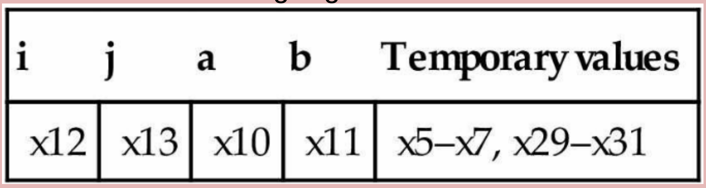
In this exercise we compare the performance of single- and dual-issue processors, taking into account program transformations that can be made to optimize for dual-issue execution. Problems in this exercise refer to the following loop (written in C):

```
   for(i=0;i!=j;i+=2) b[i]=a[i]-a[i+1];
```

A compiler doing little or no optimization might produce the following RISC-V assembly code:

```
   li x12, 0
   li x13, 8000
   li x14, 0
   TOP:
     slli x5, x12, 2
     add x6, x10, x5
     lw x7, 0(x6)
     lw x29, 4(x6)
     sub x30, x7, x29
     add x31, x11, x5
     sw x30, 0(x31)
     addi x12, x12, 2
     ENT: bne x12, x13, TOP
```

This code uses the following registers:



i	j	a	b	Temporary values
x12	x13	x10	x11	x5–x7, x29–x31

This code is provided in *[RVfpgaPath]/RVfpga/Labs/Lab17/PaHe_SuperscalarExample* with a few minor modifications compared with the code provided by the exercise from the book that do not affect to the behaviour of the program:

- Register `x13` is initialized to 8000, so that the loop will perform 4000 iterations.
- The `jal` instruction is removed.
- The `ld` and `sd` instructions are substituted for `lw` and `sw` instructions. This implies changing the accesses from 4- to 8-bytes wide.

Assume a dual-issue, statically scheduled processor that has the following properties:

© Copyright Imagination Technologies

1. One instruction must be a memory operation; the other must be an arithmetic/logic instruction or a branch.
2. The processor has all possible forwarding paths between stages.
3. The processor has perfect branch prediction.
4. Two instruction may not issue together if one depends on the other.
5. If a stall is necessary, both instructions in a stage must stall.

a) Compare the properties of this example processor and the properties of the SweRV EH1 processor.
b) Draw a pipeline diagram and a simulation showing how a random iteration of the loop (except the first one) of the RISC-V code given above executes on the dual-issue SweRV EH1 processor. Assume that the loop exits after four thousand iterations (this is the case in the above code).
c) What is the speedup of going from a single-issue to a dual-issue SweRV EH1 processor? Explain the results. Test the program on the board and enable/disable dual-issue execution.
d) Rearrange/rewrite the RISC-V code given above to achieve better performance on the two-issue SweRV EH1 processor. (However, do not unroll the loop.)
e) Now, unroll the RISC-V code so that each iteration of the unrolled loop handles two iterations of the original loop. Then, rearrange/rewrite your unrolled code to achieve better performance on the two-issue SweRV EH1 processor.

6) (*The following exercise is based on exercises 7.30, 7.32 and 7.34 from Chapter 7 of DDCARV.*)

Suppose the SweRV EH1 processor is running the following code snippet. Recall that SweRV EH1 has a Hazard Unit. You may assume a memory system that returns the result within one cycle (for that purpose we use the DCCM, and we insert the code snippet into a loop and avoid the first iteration so that there are no I$ misses).

```
addi s1, t0, 11    # t0 contains the base address of the DCCM
lw   s2, 25(s1)
lw   s5, 16(s2)
add  s3, s2, s5
or   s4, s3, t4
and  s2, s3, s4
```

a) Simulate the program with Verilator and GTKWave. Analyse the results and for each cycle, specify:
   * Which instructions are decoded, issued to execution and commited?
   * Which registers are being written and which are being read?
   * What forwarding and stalls occur?
b) What is the CPI of the processor on this program? First answer theoretically and then confirm your answer by executing the program on the board.
c) Perform the same analysis on the single-issue processor and compare the results with the results from the dual-issue processor.

A PlatformIO project is provided at: *[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_Exercises-30-32-34*. The program under analysis is inserted in a loop so that it is easier to understand in the simulation (any iteration but the first one can be used for analysis) and it can be measured using performance counters.

© Copyright Imagination Technologies

7) (*The following exercise is based on exercises 7.31, 7.33 and 7.35 from Chapter 7 of DDCARV.*)

Repeat exercise 7 for the following code snippet.

```
addi s1, t0, 52
addi s0, s1, -4
lw   s3, 16(s0)
sw   s3, 20(s0)
xor  s2, s0, s3
or   s2, s2, s3
```

A PlatformIO project is provided at: *[RVfpgaPath]/RVfpga/Labs/Lab17/DDCARV_Exercises-31-33-35*.

© Copyright Imagination Technologies