ECE-429

Computer Architecture

Project Deliverable 5 – Report

By Syed Saaem Raza Rizvi

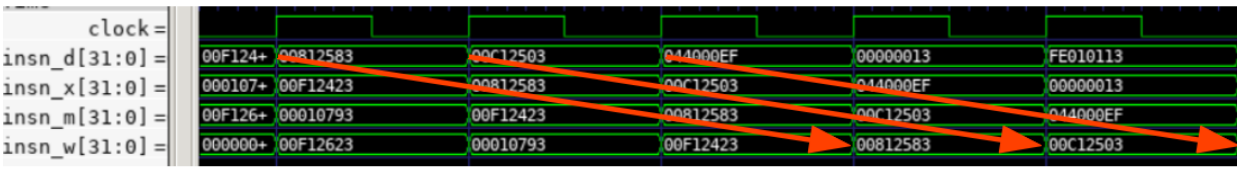
20650108

Introduction

Each major element of the pipelined processor is examined and demonstrated to work below using an appropriate waveform. At the bottom of this report are the outputs for each of the benchmarks.

Pipelining the Instruction

After the fetch state has been completed, it is necessary to pipeline the instruction fetched through the decode, execute, memory and writeback stages. The reason for this is that the immediate and the destination registers can then be determined within the same cycle they are needed, therefore a NOP instruction being inserted into the execute stage will not cause the wrong immediate to be used. Below is a waveform captured from the `Swap.x` benchmark. Note that the `insn_d`, `insn_x`, `insn_m`, `insn_w` signals represent the value in the instruction registers at the decode, execute, memory and writeback stages respectively. The instruction held at the first rising edge of the clock cycle at `insn_d` is at a program counter of `0x01000028`.

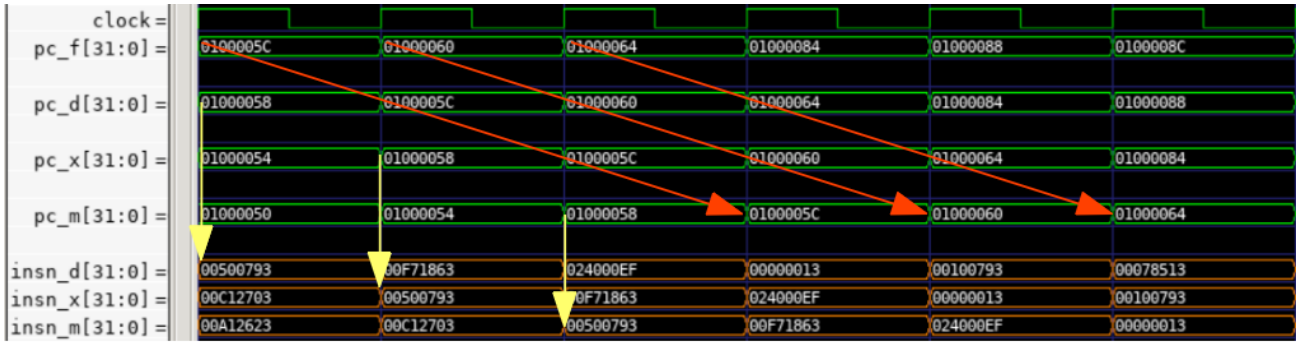


The red arrows show how the instruction gets pipelined through the decode, execute, memory and writeback stages at the rising edge of the clock cycles.

We can clearly see that instruction is being pipelined as in the next clock cycle `insn_x` = `insn_d`, `insn_m` = `insn_x` and `insn_w` = `insn_m`. This occurs at every rising edge of the clock cycle.

Pipelining the Program Counter

It is important to pipeline the program counter through fetch, decode, execute and memory stages. We need it to be pipelined to execute stage for correct ALU operation to be performed on the program counter, but we also need it to be pipelined into the memory stage so that the instruction such as `jalr` and `jal` can save the return address in the register file. Below is a waveform captured from `SimpleAdd.x`. Note that the `pc_f`, `pc_d`, `pc_x`, `pc_m` signals represent the value stored in the program counter registers at the fetch, decode, execute and memory stages respectively. The instruction registers at `insn_d`, `insn_x`, `insn_m` have also been provided so that the reader may see that the program counter register and instruction register do align at each stage (e.g. the instruction at program counter at `0x01000058` is indeed `0x00500793`)

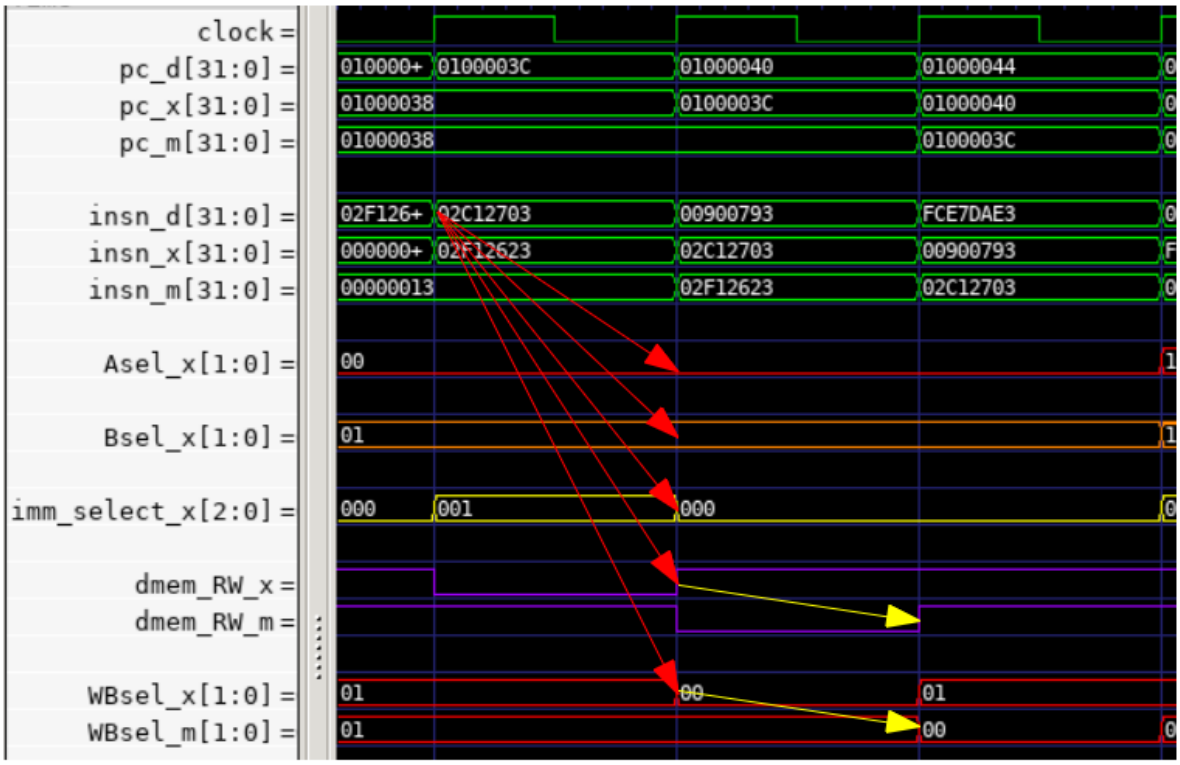


The red arrows show how the program counter is pipelined through the fetch, decode, execute and memory stages. Furthermore the yellow arrows indicate how the corresponding instruction gets pipelined along with its program counter

From this waveform we can see that the program counter is indeed pipelined up to the memory stage such that `pc_f` = `pc_d`, `pc_x` = `pc_d` and `pc_m` = `pc_x` at every rising edge of the clock cycle.

Pipelining the Control Signals

It is essential to pipeline the control signals of each instruction once it is decoded, so that each control signal arrives at the correct stage at the correct time. This time we will be looking at the `SumArray.x` binary file to show that the processor simulation is indeed pipelining the control signals. Note that we will be looking at a single instruction located at the program counter `0x0100003C`, which holds the instruction "`lw a4,44(sp)`" (RISCV: `0x02c12703`). This instruction does not need any bypassing to function correctly. Below is the waveform, where `Asel_x` is the mux select for first input into the ALU and `Bsel_x` is the mux select for the second input into the ALU. The `imm_select_x` signal is the immediate select type for the immediate generator. The `dmem_RW_x` and `dmem_RW_m` signals are the read/write signal for the data memory at the execute and memory stages respectively. Finally, `WBsel_x` and `WBsel_m` is the writeback select signals at the execute and memory stages respectively.



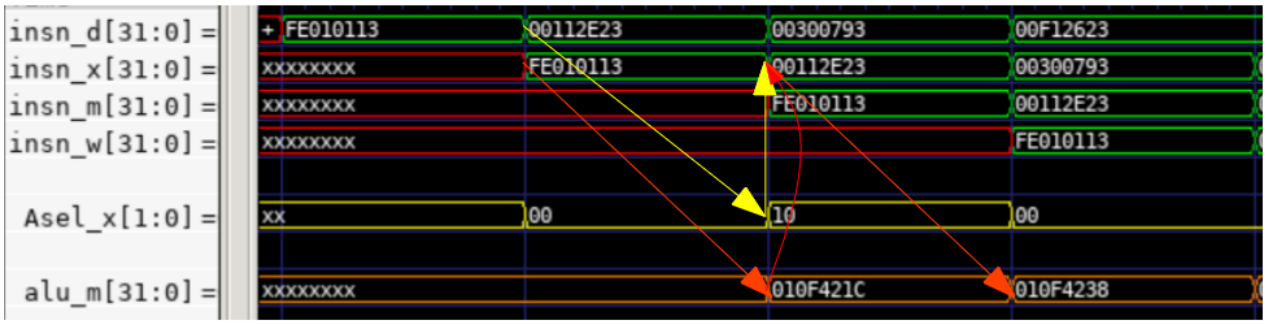
The red arrows show that the relevant control signal have been driven after the decode stage has been completed. The yellow arrows indicate that the control signals which are needed after the execute stage are pipelined.

The control signal will only be generated in the decode stage and stored in the respective registers. The control signal register values will only become visible until the execute stage starts. We can see that the control registers have the correct control signal values for a non-bypassed load instruction.

Furthermore, control signals which required in the memory stage are further pipelined to that stage. This ensures that instruction and the respective control signals arrive at the memory stage at the same time.

MX Bypassing

MX bypassing is required to reduce the number of stalls occurring due to a data hazard. It occurs when the next instruction is dependent on the current instruction. The source code detects whether an MX bypass is needed by checking the instruction registers at the decode and execute stage and sees whether the decode instruction register is dependent on the execute instruction register. It will drive the select signals for both the ALU inputs accordingly to get the correct values into the ALU. We will be looking at the `SimpleIf.x` binary file. We will be looking at the first two instructions in this file which are "`addi sp,sp,-32`" and "`sw ra,28(sp)`" (RISCV: `0xFE010113`, `0x00112e23`). We can see that the second instruction is dependent on the first instruction due to `sp`, therefore we need an MX bypass. The waveform below shows that this is achieved. Note that `alu_m` is the ALU output register at the memory stage and that the initial value of `sp` is `0x010F423C`.



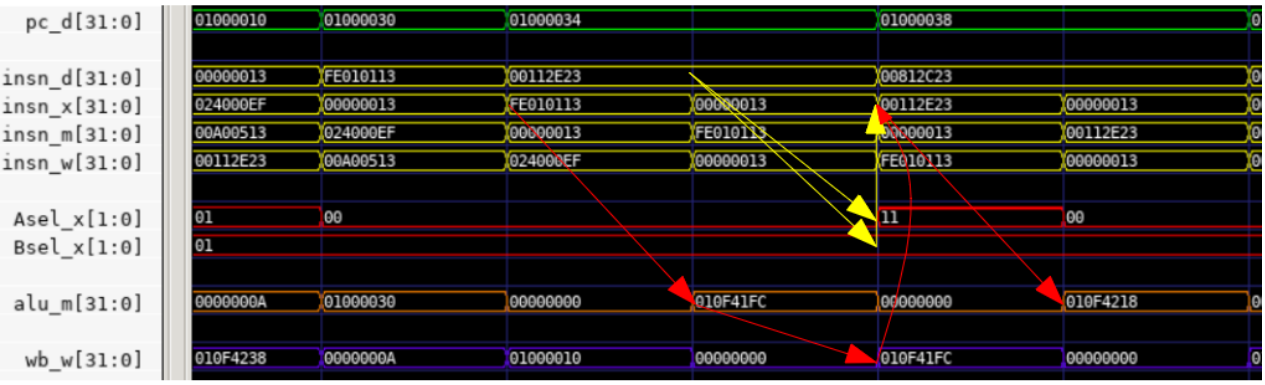
The red arrows show that the value in alu_m (from the load instruction) was bypassed into the execute stage and that the output of the execute stage for the store instruction is indeed correct. The yellow arrows indicate that a MX bypass usage will be needed, hence setting a value for the first input's mux select.

We can see that once the decode instruction register and execute instruction register are analyzed, `Asel_x` is set to a value of `2'b10` as the `rs1` in the second instruction is dependent. This means that the first input of the ALU will be the value stored in `alu_m`, which was set by the first instruction (`alu_m` = `sp` – 32 as `sp` = `sp` - 32). We can see the correct value of `alu_m` when the second instruction reaches the memory stage (`alu_m` = `sp` + 28), meaning that the MX bypass was successful. Of course, there are many other cases for which a bypass may be needed, and the source code is explicitly aware of it.

WX Bypassing

WX bypassing is required to reduce the number of data hazard stalls. It occurs when an instruction two cycles behind the current instruction is dependent on the current instruction. The source code detects whether an WX bypass is needed by checking the instruction registers at the decode and memory stages. It will drive the select signals for both the ALU inputs accordingly to get the correct values into the ALU. We will be looking at the `Fibonacci.x` binary file. We will be looking program counters `0x01000030` and `0x01000034` which hold the instructions "`addi sp,sp,-32`" and "`sw`

ra,28(sp)" respectively. Note that you might notice a NOP instruction being instructed due to a stall, but we will talk about this in the future. The stall allows us to do is to create a two cycle gap between the two instructions we are analyzing. We can see that the second instruction is dependent on the first due to sp. The initial value of before program counter 0x01000030 is 0x010F421C. The signal wb_w represents the value of the writeback register at the writeback stage.

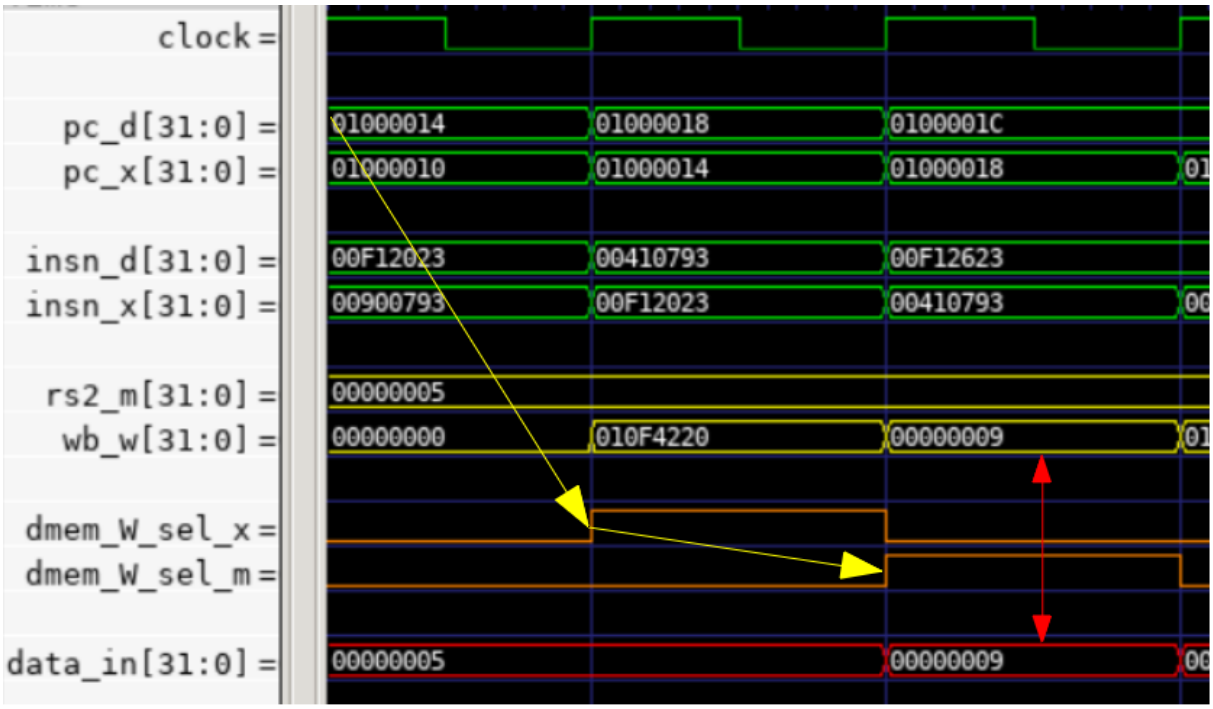


The red arrows show how the addi instruction produces a value in writeback register (wb_w) and bypasses it to the store instruction when it is in the execute stage, this allows it to get the correct value into alu_m. The yellow arrows indicate that WM bypass will be needed and that the mux select signals for the two inputs to the ALU have been driven.

We can see that once the decode instruction register and execute instruction register are analyzed, Asel_x is set to a value of 2'b11 as the rs1 in the second instruction is dependent. This means that first input of the ALU will be the value stored at the wb_w (wb_w = sp – 32 as sp = sp - 32). We can see that the correct value of wb_w reaches the ALU as the value of alu_m is what we were expecting (alu_m = sp + 28) meaning that the WX bypass was successful. Of course, there are many other cases for which a bypass may be needed, and the source code is explicitly aware of it.

WM Bypassing

WM bypassing is required to reduce the number of data hazard stalls. It occurs when rs2 (the value to write into data memory) of a store instruction is dependent on the previous instruction. The source code detects the WM bypass by analyzing the instruction registers at the decode and execute stage. It will drive a select signal for the data write input port. If the select signal is 1'b0 then rs2 will be written into data memory. If the select signal is 1'b1 the value to be written into data memory will be the value in the writeback register in the writeback stage (wb_w). We will analyze this bypass using the Swap.x binary file. The waveform below shows how this is achieved. The two instruction which will be analyzed are "li a5,9" (RISCV: 0x00900793 at program counter: 0x01000010) and "sw a5,0(sp)" (RISCV: 0x00F12023 at program counter: 0x01000014). We can see that the store instruction rs2 is dependent on the first instruction. Note that dmem_W_sel_x and dmem_W_sel_m are the data write select signals at the execute and memory stages respectively. The data_in signal shows the value which will be written into data memory. Finally, rs2_m is the value of rs2 gotten from the register file and pipelined to the memory stage

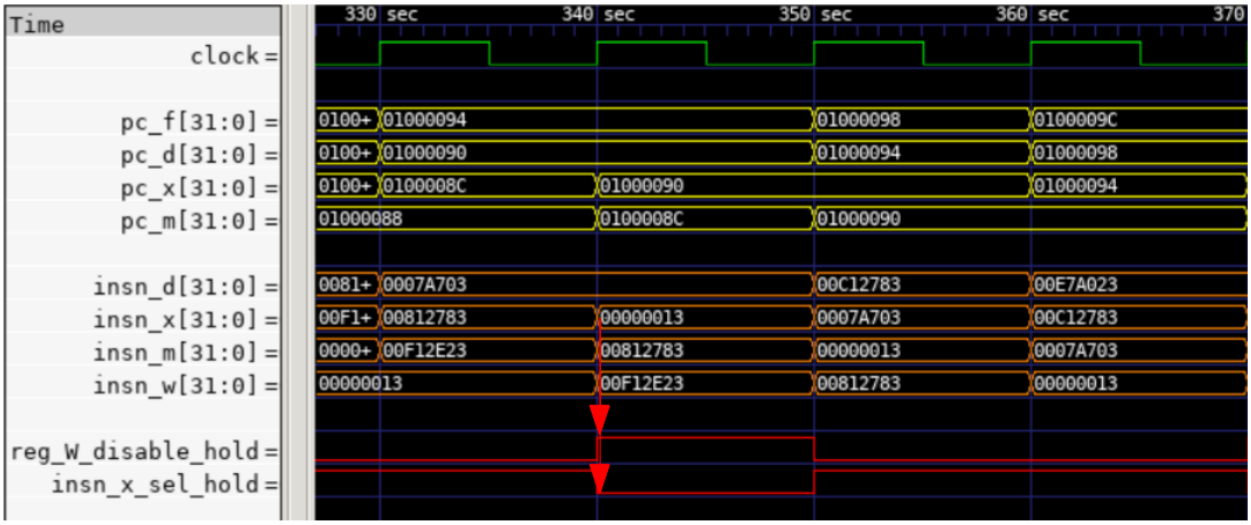


The red arrow shows that the value of data_in of the data memory is the same as the writeback register (wb_w). The yellow arrows show that a WM bypass needs to be implemented, therefore the data memory select signal is driven and pipelined to the memory stage

We can see that when the store instruction is in the decode instruction register and the load immediate instruction is in the execute instruction register, the source code tries to detect a WM bypass. It does so successfully as `dmem_W_sel_x` is set `1'b1` and this value is then bypassed to `dmem_W_sel_m` for the data write mux to utilize. Furthermore, we can see that the value of `data_in` is that of `wb_w` meaning that the WM bypass was successful.

Load-Use Stall

When a load instruction is followed up by a dependent instruction after it, a single cycle stall is required for the dependent instruction as there is no way to bypass the value which will be written into the destination register (determined by the load instruction). The stall will occur at the decode stage if load instruction is detected in the execution stage. A NOP instruction will be inserted into the execute stage in the next cycle to allow the pipeline to still function correctly. On top of this it is essential to hold the values inside the program counters at the fetch and decode stage and decode instruction register for one extra cycle for the stall to work properly. We will be looking at the waveform produced by `Swap.x` and in particular we will be looking at two instruction "`lw a5,8(sp)`" (RISCV: `0x00812783` at program counter: `0x0100008C`) and "`lw a4,0(a5)`" (RISCV: `0x0007A703` at program counter: `0x01000090`). Note that `insn_x_sel_hold` signal is the select signal for `insn_x`, as we either want it to hold `insn_d` or a NOP instruction if we need a stall (if select is `1'b0` then `insn_x` = NOP). Moreover, setting the `reg_W_disable_hold` signal makes sure that the three previously mentioned registers (`pc_f`, `pc_d`, `insn_f`) are not overwritten in the next clock cycle.



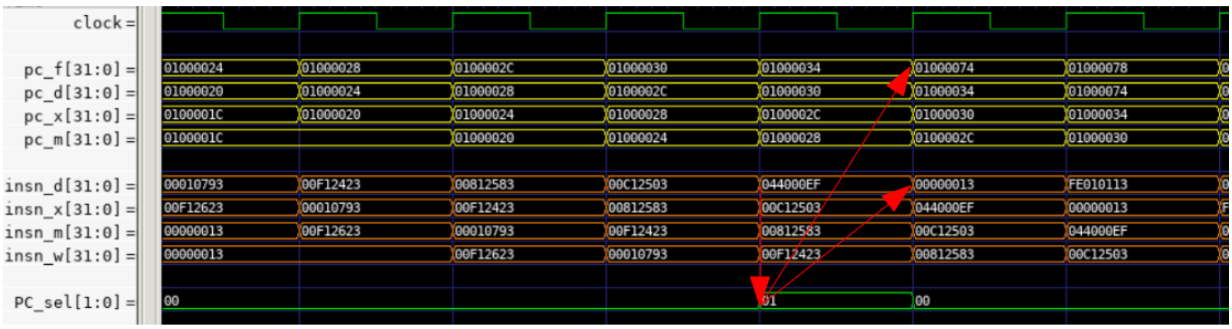
The red arrows show that a load-use situation has been detected and therefore inserts a NOP into the execute stage, relevant control signal have also been set. The values in insn_d, pc_f and pc_d remain unchanged.

Looking at the waveform, when the second load instruction is the decode stage and the first load instruction is in the execute, the source code detects that a load-use situation is going to occur and sets the `reg_W_disable_hold` to `1'b1` and `insn_x_sel_hold` to `1'b0` within the same clock cycle. Due to this, we can see that the three registers indeed retain their values for an extra clock cycle and that a NOP instruction was inserted into the execute stage. This NOP instruction will allow the WX bypass to occur to make sure another stall cycle is not needed.

It is worth mentioning that the load-use stall will also handle the structural stalls. This means that the instruction after the 'use' instruction is also stalled in the fetch stage.

Unconditional Jumps

Unconditional jumps by their name do not require a condition to branch to a target address. In order to reduce the number of stalls in the pipeline, if a jump instruction (`jal` or `jalr`) is detected in the decode stage it will set the PC select to `2'b01` and will place NOP instruction in the decode instruction register in the next cycle. Furthermore, it will change the value of the program counter register at the fetch stage, so that the processor can fetch the instruction at the target address. The calculation for the target address will take place within the decode stage. We will have a look at the `Swap.x` waveform. We will be focusing our attention on the instruction "`jal ra,0x01000074`" (RISCV: `0x044000EF` at program counter: `0x01000030`). Note that the signal `PC_sel` selects the value of the program counter depending on the jump instruction. As mentioned previously unconditional jumps have a `PC_sel` value of `2'b01`. `2'b00` is the value for the select signal such that the program counter increments by 4.

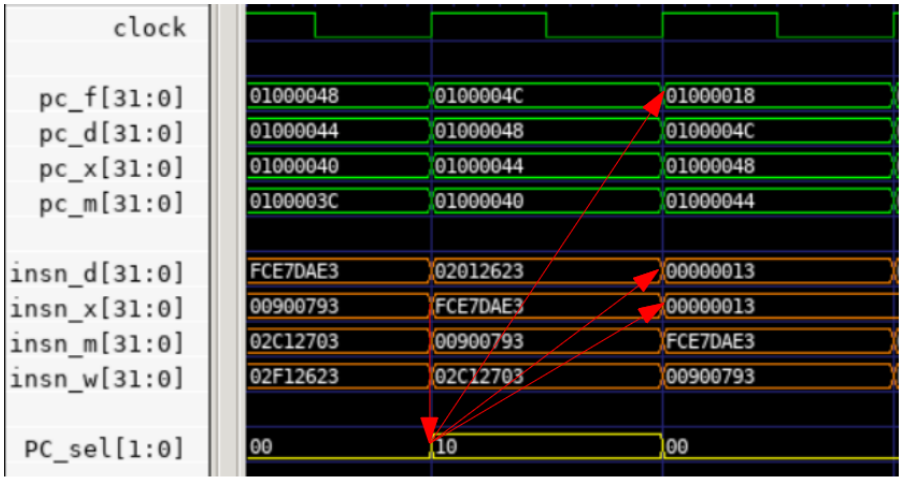


The red arrows show that a unconditional jump instruction has been detected. It inserts a NOP instruction in the next cycle, whilst also changing the value of pc_f to the target address.

From the waveform diagram we can see that when the `PC_sel` signal is set for an unconditional jump as soon as it detects the jump instruction in the decode stage. In the next cycle a NOP instruction is placed in the decode instruction register instead of the instruction at program counter `0x01000034`. At the same time the program counter in the fetch stage has the value of the target address specified in the jump instruction. This means that only one instruction had to be bubbled which was our intention therefore unconditional jumps are working as intended.

Conditional Jumps

Conditional jumps require a condition to be met before jumping. To reduce the number of bubbled instructions, it is important to figure out whether the branch will be taken in the execute stage. If it is not, the processor will continue as normal. In the case that the branch must be taken, both the decode and execute instruction registers need to be replace with NOP instructions. Furthermore, the `PC_sel` has to be set to `2'b10` so that the correct instruction can be fetched. We will be looking at the `SumArray.x` waveform. The instruction of interest is "`bge a5,a4,0x01000018`" (RISCV: `0xFCE7DAE3` at program counter `0x01000044`).



The red arrows indicate that a conditional jump has been detected and that the condition was met. It inserts 2 NOP instruction in the next cycle, one each at insn_d and insn_x. The value of pc_f is also changed to the target address.

We can see from the waveform that as soon as the conditional jump instruction is in the execute instruct register, it has been detected that the branch will indeed be taken. Due to this both `insn_d` and `insn_x` registers are place with NOP instruction in the next cycle, whilst the program counter in the fetch stage store the value of the target address specified in the branch instruction. Due to this, the two instruction after the unconditional jump are bubbled, which means the processor is handling the branch correctly. The conditional jump instruction continues to be pipelined.

BubbleSort.x Output

```
WARNING: ./include/memory_module.v:21: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/memory_module.v:21: $readmemh(test/BubbleSort.x): Not enough words in the file for the requested range [0:249999].
-----------------------------------------
Initial Register File State
-----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: xxxxxxxx
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: xxxxxxxx
x         15: xxxxxxxx
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
-----------------------------------------

WARNING: ./include/data_memory_module.v:29: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/data_memory_module.v:29: $readmemh(test/BubbleSort.x): Not enough words in the file for the requested range [0:249999].
VCD info: dumpfile BubbleSort.vcd opened for output.
VCD warning: ignoring signals in previously scanned scope dut.insn_memory.
VCD warning: ignoring signals in previously scanned scope dut.decoder.
VCD warning: ignoring signals in previously scanned scope dut.register_file.
VCD warning: ignoring signals in previously scanned scope dut.imm_generator.
VCD warning: ignoring signals in previously scanned scope dut.alu.
VCD warning: ignoring signals in previously scanned scope dut.writeback.
VCD warning: ignoring signals in previously scanned scope dut.data_memory.
VCD warning: ignoring signals in previously scanned scope dut.forwarding_control.
VCD warning: ignoring signals in previously scanned scope dut.stall_detect.
VCD warning: ignoring signals in previously scanned scope dut.jump_detect.
-----------------------------------------
After Execution Registers
-----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: 00000001
x         11: 00000008
x         12: 00000078
x         13: 010f420c
x         14: 00000007
x         15: 00000001
x         16: 00000009
x         17: 0000000c
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
-----------------------------------------
```

We can see that the execution of this program was successful as `x15` hold a return value of `1` and `x14` holds the correct result value which can be found in `BubbleSort.c`.

`CheckVowel.x` Output

```
WARNING: ./include/memory_module.v:21: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/memory_module.v:21: $readmemh(test/CheckVowel.x): Not enough words in the file for the requested range [0:249999].
-----------------------------------------
Initial Register File State
-----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: xxxxxxxx
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: xxxxxxxx
x         15: xxxxxxxx
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
-----------------------------------------

WARNING: ./include/data_memory_module.v:29: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/data_memory_module.v:29: $readmemh(test/CheckVowel.x): Not enough words in the file for the requested range [0:249999].
VCD info: dumpfile CheckVowel.vcd opened for output.
VCD warning: ignoring signals in previously scanned scope dut.insn_memory.
VCD warning: ignoring signals in previously scanned scope dut.decoder.
VCD warning: ignoring signals in previously scanned scope dut.register_file.
VCD warning: ignoring signals in previously scanned scope dut.imm_generator.
VCD warning: ignoring signals in previously scanned scope dut.alu.
VCD warning: ignoring signals in previously scanned scope dut.writeback.
VCD warning: ignoring signals in previously scanned scope dut.data_memory.
VCD warning: ignoring signals in previously scanned scope dut.forwarding_control.
VCD warning: ignoring signals in previously scanned scope dut.stall_detect.
VCD warning: ignoring signals in previously scanned scope dut.jump_detect.
-----------------------------------------
After Execution Registers
-----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: 00000001
x         11: xxxxxxxx
x         12: 63656843
x         13: 776f566b
x         14: 00000003
x         15: 00000001
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
-----------------------------------------
```

We can see that the execution of this program was successful as `x15` hold a return value of `1` and `x14` holds the correct result value which can be found in `CheckVowel.c`.

`Fact.x` Output

```
WARNING: ./include/memory_module.v:21: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/memory_module.v:21: $readmemh(test/fact.x): Not enough words in the file for the requested range [0:249999].
-----------------------------------------
Initial Register File State
-----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: xxxxxxxx
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: xxxxxxxx
x         15: xxxxxxxx
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
-----------------------------------------

WARNING: ./include/data_memory_module.v:29: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/data_memory_module.v:29: $readmemh(test/fact.x): Not enough words in the file for the requested range [0:249999].
VCD info: dumpfile fact.vcd opened for output.
VCD warning: ignoring signals in previously scanned scope dut.insn_memory.
VCD warning: ignoring signals in previously scanned scope dut.decoder.
VCD warning: ignoring signals in previously scanned scope dut.register_file.
VCD warning: ignoring signals in previously scanned scope dut.imm_generator.
VCD warning: ignoring signals in previously scanned scope dut.alu.
VCD warning: ignoring signals in previously scanned scope dut.writeback.
VCD warning: ignoring signals in previously scanned scope dut.data_memory.
VCD warning: ignoring signals in previously scanned scope dut.forwarding_control.
VCD warning: ignoring signals in previously scanned scope dut.stall_detect.
VCD warning: ignoring signals in previously scanned scope dut.jump_detect.
-----------------------------------------
After Execution Registers
-----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: 00000001
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: 00058980
x         15: 00000001
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
-----------------------------------------
```

We can see that the execution of this program was successful as `x15` hold a return value of `1` and `x14` holds the correct result value which can be found in `fact.c`.

Fibonacci.x Output

```
----------------------------------------
Initial Register File State
----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: xxxxxxxx
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: xxxxxxxx
x         15: xxxxxxxx
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
----------------------------------------

WARNING: ./include/data_memory_module.v:29: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/data_memory_module.v:29: $readmemh(test/Fibonacci.x): Not enough words in the file for the requested range [0:249999].
VCD info: dumpfile Fibonacci.vcd opened for output.
VCD warning: ignoring signals in previously scanned scope dut.insn_memory.
VCD warning: ignoring signals in previously scanned scope dut.decoder.
VCD warning: ignoring signals in previously scanned scope dut.register_file.
VCD warning: ignoring signals in previously scanned scope dut.imm_generator.
VCD warning: ignoring signals in previously scanned scope dut.alu.
VCD warning: ignoring signals in previously scanned scope dut.writeback.
VCD warning: ignoring signals in previously scanned scope dut.data_memory.
VCD warning: ignoring signals in previously scanned scope dut.forwarding_control.
VCD warning: ignoring signals in previously scanned scope dut.stall_detect.
VCD warning: ignoring signals in previously scanned scope dut.jump_detect.
----------------------------------------
After Execution Registers
----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: 00000001
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: 00000037
x         15: 00000001
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
----------------------------------------
```

We can see that the execution of this program was successful as `x15` hold a return value of `1` and `x14` holds the correct result value which can be found in `Fibonacci.c`.

gcd.x Output

```
----------------------------------------
Initial Register File State
----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: xxxxxxxx
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: xxxxxxxx
x         15: xxxxxxxx
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
----------------------------------------

WARNING: ./include/data_memory_module.v:29: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/data_memory_module.v:29: $readmemh(test/gcd.x): Not enough words in the file for the requested range [0:249999].
VCD info: dumpfile gcd.vcd opened for output.
VCD warning: ignoring signals in previously scanned scope dut.insn_memory.
VCD warning: ignoring signals in previously scanned scope dut.decoder.
VCD warning: ignoring signals in previously scanned scope dut.register_file.
VCD warning: ignoring signals in previously scanned scope dut.imm_generator.
VCD warning: ignoring signals in previously scanned scope dut.alu.
VCD warning: ignoring signals in previously scanned scope dut.writeback.
VCD warning: ignoring signals in previously scanned scope dut.data_memory.
VCD warning: ignoring signals in previously scanned scope dut.forwarding_control.
VCD warning: ignoring signals in previously scanned scope dut.stall_detect.
VCD warning: ignoring signals in previously scanned scope dut.jump_detect.
----------------------------------------
After Execution Registers
----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: 00000001
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: 00000004
x         15: 00000001
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
----------------------------------------
```

We can see that the execution of this program was successful as `x15` hold a return value of `1` and `x14` holds the correct result value which can be found in `gcd.c`.

`SimpleAdd.x` Output

```
----------------------------------------
Initial Register File State
----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: xxxxxxxx
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: xxxxxxxx
x         15: xxxxxxxx
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
----------------------------------------

WARNING: ./include/data_memory_module.v:29: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/data_memory_module.v:29: $readmemh(test/SimpleAdd.x): Not enough words in the file for the requested range [0:249999].
VCD info: dumpfile SimpleAdd.vcd opened for output.
VCD warning: ignoring signals in previously scanned scope dut.insn_memory.
VCD warning: ignoring signals in previously scanned scope dut.decoder.
VCD warning: ignoring signals in previously scanned scope dut.register_file.
VCD warning: ignoring signals in previously scanned scope dut.imm_generator.
VCD warning: ignoring signals in previously scanned scope dut.alu.
VCD warning: ignoring signals in previously scanned scope dut.writeback.
VCD warning: ignoring signals in previously scanned scope dut.data_memory.
VCD warning: ignoring signals in previously scanned scope dut.forwarding_control.
VCD warning: ignoring signals in previously scanned scope dut.stall_detect.
VCD warning: ignoring signals in previously scanned scope dut.jump_detect.
----------------------------------------
After Execution Registers
----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: 00000001
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: 00000005
x         15: 00000001
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
----------------------------------------
```

We can see that the execution of this program was successful as `x15` hold a return value of `1` and `x14` holds the correct result value which can be found in `SimpleAdd.c`.

`SimpleIf.x` Output

```
----------------------------------------
Initial Register File State
----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: xxxxxxxx
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: xxxxxxxx
x         15: xxxxxxxx
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
----------------------------------------

WARNING: ./include/data_memory_module.v:29: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/data_memory_module.v:29: $readmemh(test/SimpleIf.x): Not enough words in the file for the requested range [0:249999].
VCD info: dumpfile SimpleIf.vcd opened for output.
VCD warning: ignoring signals in previously scanned scope dut.insn_memory.
VCD warning: ignoring signals in previously scanned scope dut.decoder.
VCD warning: ignoring signals in previously scanned scope dut.register_file.
VCD warning: ignoring signals in previously scanned scope dut.imm_generator.
VCD warning: ignoring signals in previously scanned scope dut.alu.
VCD warning: ignoring signals in previously scanned scope dut.writeback.
VCD warning: ignoring signals in previously scanned scope dut.data_memory.
VCD warning: ignoring signals in previously scanned scope dut.forwarding_control.
VCD warning: ignoring signals in previously scanned scope dut.stall_detect.
VCD warning: ignoring signals in previously scanned scope dut.jump_detect.
----------------------------------------
After Execution Registers
----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: 00000001
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: 00000007
x         15: 00000001
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
----------------------------------------
```

We can see that the execution of this program was successful as `x15` hold a return value of `1` and `x14` holds the correct result value which can be found in `SimpleIf.c`.

SumArray.x Output

```
----------------------------------------
Initial Register File State
----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: xxxxxxxx
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: xxxxxxxx
x         15: xxxxxxxx
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
----------------------------------------

WARNING: ./include/data_memory_module.v:29: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/data_memory_module.v:29: $readmemh(test/SumArray.x): Not enough words in the file for the requested range [0:249999].
VCD info: dumpfile SumArray.vcd opened for output.
VCD warning: ignoring signals in previously scanned scope dut.insn_memory.
VCD warning: ignoring signals in previously scanned scope dut.decoder.
VCD warning: ignoring signals in previously scanned scope dut.register_file.
VCD warning: ignoring signals in previously scanned scope dut.imm_generator.
VCD warning: ignoring signals in previously scanned scope dut.alu.
VCD warning: ignoring signals in previously scanned scope dut.writeback.
VCD warning: ignoring signals in previously scanned scope dut.data_memory.
VCD warning: ignoring signals in previously scanned scope dut.forwarding_control.
VCD warning: ignoring signals in previously scanned scope dut.stall_detect.
VCD warning: ignoring signals in previously scanned scope dut.jump_detect.
----------------------------------------
After Execution Registers
----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: 00000001
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: 0000002d
x         15: 00000001
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
----------------------------------------
```

We can see that the execution of this program was successful as `x15` hold a return value of `1` and `x14` holds the correct result value which can be found in `SumArray.c`.

Swap.x Output

```
WARNING: ./include/memory_module.v:21: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/memory_module.v:21: $readmemh(test/Swap.x): Not enough words in the file for the requested range [0:249999].
-----------------------------------------
Initial Register File State
-----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: xxxxxxxx
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: xxxxxxxx
x         15: xxxxxxxx
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
-----------------------------------------

WARNING: ./include/data_memory_module.v:29: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/data_memory_module.v:29: $readmemh(test/Swap.x): Not enough words in the file for the requested range [0:249999].
VCD info: dumpfile Swap.vcd opened for output.
VCD warning: ignoring signals in previously scanned scope dut.insn_memory.
VCD warning: ignoring signals in previously scanned scope dut.decoder.
VCD warning: ignoring signals in previously scanned scope dut.register_file.
VCD warning: ignoring signals in previously scanned scope dut.imm_generator.
VCD warning: ignoring signals in previously scanned scope dut.alu.
VCD warning: ignoring signals in previously scanned scope dut.writeback.
VCD warning: ignoring signals in previously scanned scope dut.data_memory.
VCD warning: ignoring signals in previously scanned scope dut.forwarding_control.
VCD warning: ignoring signals in previously scanned scope dut.stall_detect.
VCD warning: ignoring signals in previously scanned scope dut.jump_detect.
-----------------------------------------
After Execution Registers
-----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: 00000001
x         11: 010f421c
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: 0000000e
x         15: 00000001
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
-----------------------------------------
```

We can see that the execution of this program was successful as `x15` hold a return value of `1` and `x14` holds the correct result value which can be found in `Swap.c`.

`SwapShift.x` Output

```
WARNING: ./include/memory_module.v:21: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/memory_module.v:21: $readmemh(test/SwapShift.x): Not enough words in the file for the requested range [0:249999].
-----------------------------------------
Initial Register File State
-----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: xxxxxxxx
x         11: xxxxxxxx
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: xxxxxxxx
x         15: xxxxxxxx
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
-----------------------------------------

WARNING: ./include/data_memory_module.v:29: $readmemh: Standard inconsistency, following 1364-2005.
WARNING: ./include/data_memory_module.v:29: $readmemh(test/SwapShift.x): Not enough words in the file for the requested range [0:249999].
VCD info: dumpfile SwapShift.vcd opened for output.
VCD warning: ignoring signals in previously scanned scope dut.insn_memory.
VCD warning: ignoring signals in previously scanned scope dut.decoder.
VCD warning: ignoring signals in previously scanned scope dut.register_file.
VCD warning: ignoring signals in previously scanned scope dut.imm_generator.
VCD warning: ignoring signals in previously scanned scope dut.alu.
VCD warning: ignoring signals in previously scanned scope dut.writeback.
VCD warning: ignoring signals in previously scanned scope dut.data_memory.
VCD warning: ignoring signals in previously scanned scope dut.forwarding_control.
VCD warning: ignoring signals in previously scanned scope dut.stall_detect.
VCD warning: ignoring signals in previously scanned scope dut.jump_detect.
-----------------------------------------
After Execution Registers
-----------------------------------------
x          0: 00000000
x          1: 00000000
x          2: 010f423c
x          3: xxxxxxxx
x          4: xxxxxxxx
x          5: xxxxxxxx
x          6: xxxxxxxx
x          7: xxxxxxxx
x          8: xxxxxxxx
x          9: xxxxxxxx
x         10: 00000001
x         11: 010f421c
x         12: xxxxxxxx
x         13: xxxxxxxx
x         14: 00000026
x         15: 00000001
x         16: xxxxxxxx
x         17: xxxxxxxx
x         18: xxxxxxxx
x         19: xxxxxxxx
x         20: xxxxxxxx
x         21: xxxxxxxx
x         22: xxxxxxxx
x         23: xxxxxxxx
x         24: xxxxxxxx
x         25: xxxxxxxx
x         26: xxxxxxxx
x         27: xxxxxxxx
x         28: xxxxxxxx
x         29: xxxxxxxx
x         30: xxxxxxxx
x         31: xxxxxxxx
-----------------------------------------
```

We can see that the execution of this program was successful as `x15` hold a return value of `1` and `x14` holds the correct result value which can be found in `SwapShift.c`.