Politecnico di Torino

Master degree in Electrical Engineering

Integrated systems architecture

Laboratory Session n°3
RISC-V-lite processor: Design & Development

Referents: Prof. Guido Masera, Maurizio Martina

Authors:

Tesser Andrea, Vianello Nicola

March 3, 2020

Abstract

The goal of this laboratory session is to design a RISC-V-lite processor with 5 pipeline stages. The lite version of the RISC-V processor supports only a limited subset of the whole RV32I instruction set, but in the design here reported all the standard instructions except ECALL, EBREAK, and FENCE are implemented.

The processor is tested by simulating the execution of a program that computes the minimum absolute value of an array of integers. After that, a custom instruction is added in order to compute the absolute value in a single clock cycle using some additional hardware. At this point the program is improved using the new instruction and simulated once again. Finally, a comparison between the two version is made in order to evaluate the advantages and the disadvantages of the two approaches.

Both the version of the processor are synthesized in a 45 nm standard-cell library and then place-and-routed in a physical design.

Features

32-bit RISC CPU:

- Five stages pipeline with forwarding;
- Automatic hazard detection;
- Up to 4GB of addressing space;
- Branch target buffer with 4-way 8-set associative cache, 8-bit TAG, LRU policy replacement;
- Ready for multi-cycle arithmetic operations.
- Absolute value custom instruction.

45 nm **CMOS technology**

- Max clock frequency: 500 MHz
- Supply voltage: 1.1 V
- Total power dissipation: 15.21 mW
- Silicon die: 226 µm x 226 µm

Supported instruction set
Register-to-register-operations: ADD, SUB, SLL, SLT, SLTU, XOR, SRL, SRA, OR, AND.
Immediate operations: ADDI, SLLI, SLTI, SLTIU, XORI, SRLI, SRAI, ORI, ANDI.
Load: LB, LH, LW, LBU, LHU.
Store: SB, SH, SW.
Branch: BEQ, BNE, BLT, BGE, BLTU, BGEU.
Jump: JAL, JALR.
Upper immediate operations: AUIPC, LUI.
Custom: ABS.

Contents

List of Figures

List of Tables

List of Listings

CHAPTER 1

RISC-V

The top level view of the processor is shown in figure 1.1, it has in addition to the clock and the reset ports, an interface for the IRAM (Instruction RAM) and one for the DRAM (Data RAM). Inside it we find the following blocks:

- The datapath: it is the block where both the memories are connected from the outside except for the DRAM write enable that is driven by the CU (Control Unit). Furthermore it receives the control from the CU, the FU (Forwarding Unit), and the HDU (Hazard Detection Unit), while the only output signal is the misprediction one, used by the BPU (Branch Prediction Unit) to inform the HDU that an hazard have occured.
- A set of registers used to keep track of all the instructions running inside the pipeline.
- The CU, which receives the instruction in the ID stage and according to this produces the appropriate control outputs.
- A set of registers that propagate the CU outputs, in order to send the signals to the datapath at the right time depending on the pipeline stage where they are used.
- The FU that reading all the instruction in execution from the second stage to the last stage can detect the data dependencies and, when it is possible, avoids the data hazard bypassing the data between the pipeline stages.
- The HDU, according on the instruction in the second and in the third stage and on the misprediction signal (received by the BPU inside the datapath) it has the ability to stop some pipeline stages and to discard an already fetched instruction, in order to avoid any kind of hazard.

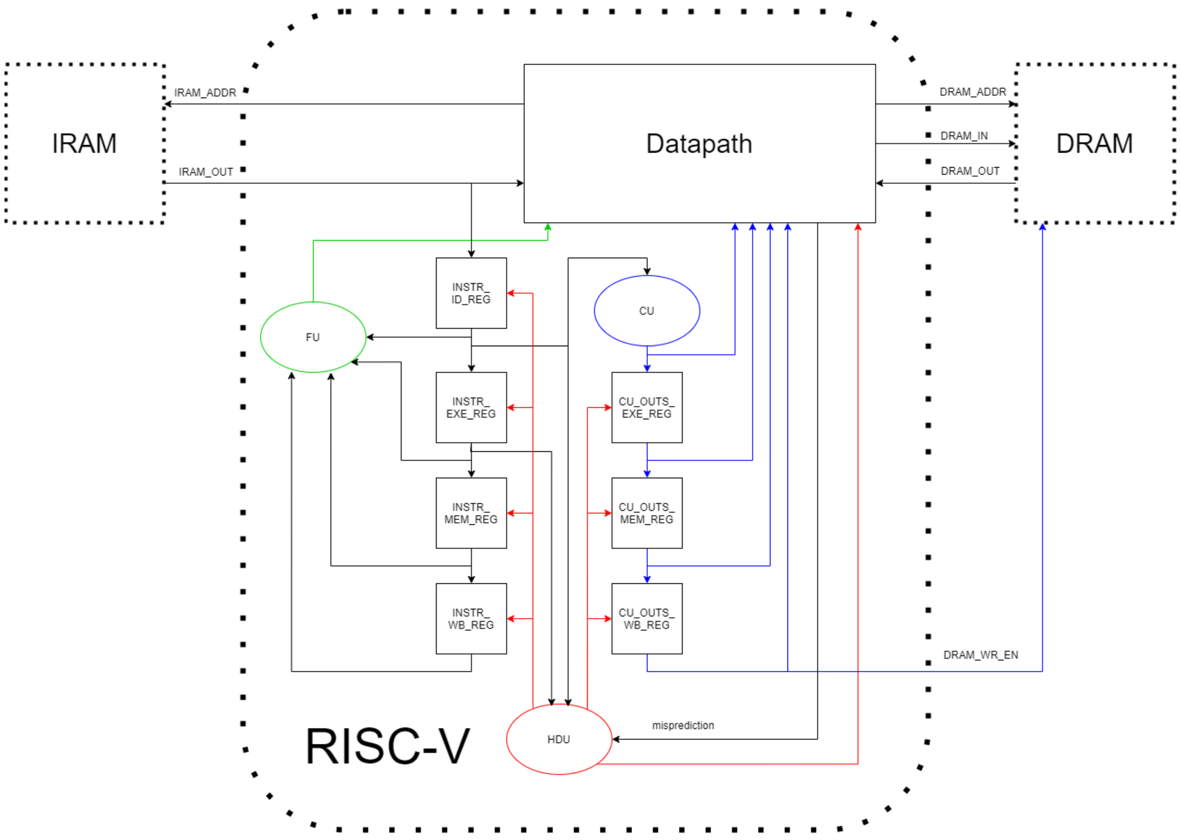


Figure 1.1: RISC-V top view diagram.

CHAPTER 2

Datapath

The schematic of the whole datapath is shown in figure 2.1. It is divided into five pipeline stages:

1. The first is the IF (Instruction Fetch) and has the task of fetching the instructions from the IRAM. Here the PC (Program Counter) register, the adder that sum four to the value of the PC, and the IRAM are collocated, remembering however that the last is connected from the outside. In the schematic the BPU is shown in this stage, because we can think of it as a MUX that drives the PC input and is able to send it the PC+4, the actual address computed in the next stage, and the output of the BTB. Anyway it is important to remember that the BPU has actually also a sequential control circuit and that this works both in the IF/ID step where it stores the information about the prediction, and in the ID/EXE step where it updates the cache according to the computed real address. For more accurate information on this component see section 2.3.

2. The second stage is the ID (Instruction Decode) and has the task of decoding the fetched instruction, reading the register file, and computing the real address if the instruction is a jump or a branch, in order to verify the prediction of the BPU. To do this last operation, the target address is computed by an adder which sums the value of the base address to the immediate offset, and a MUX selects as actual address this output or the address next to the jump/branch according to the result of the branch comparator. More details on this component are explained in section 2.2, but basically if the branch is tuned out to be taken or the jump is unconditional the MUX selects the target address computed by the adder, otherwise if the branch is turned out to be not taken the selected address is the one next to the branch. Furthermore, the base address from which the target address is computed can be taken either from the program counter or from a register, by means of another MUX.

   Finally, in this stage there are also two MUXes to forward the data already in the pipeline to the ALU inputs (see 3.2), a MUX to select from the instruction the correct bits that compose the immediate field according to the format (I, S, B, U, or J type), and a MUX used to propagate through the pipeline the value of the program counter or the next one (useful for JAL, JALR and AUIPC).

3. The third is the EXE (EXEcute) stage, where the arithmetic and logic operations are carried out thanks to the ALU (see section 2.1). A MUX determine whether the second operand is taken from the second register file output or from the immediate value. Again a last MUX is used for a particular case of forwarding that occurs when a store follows a load and it has to write the same data coming out from the memory (detail in chapter 3.2).

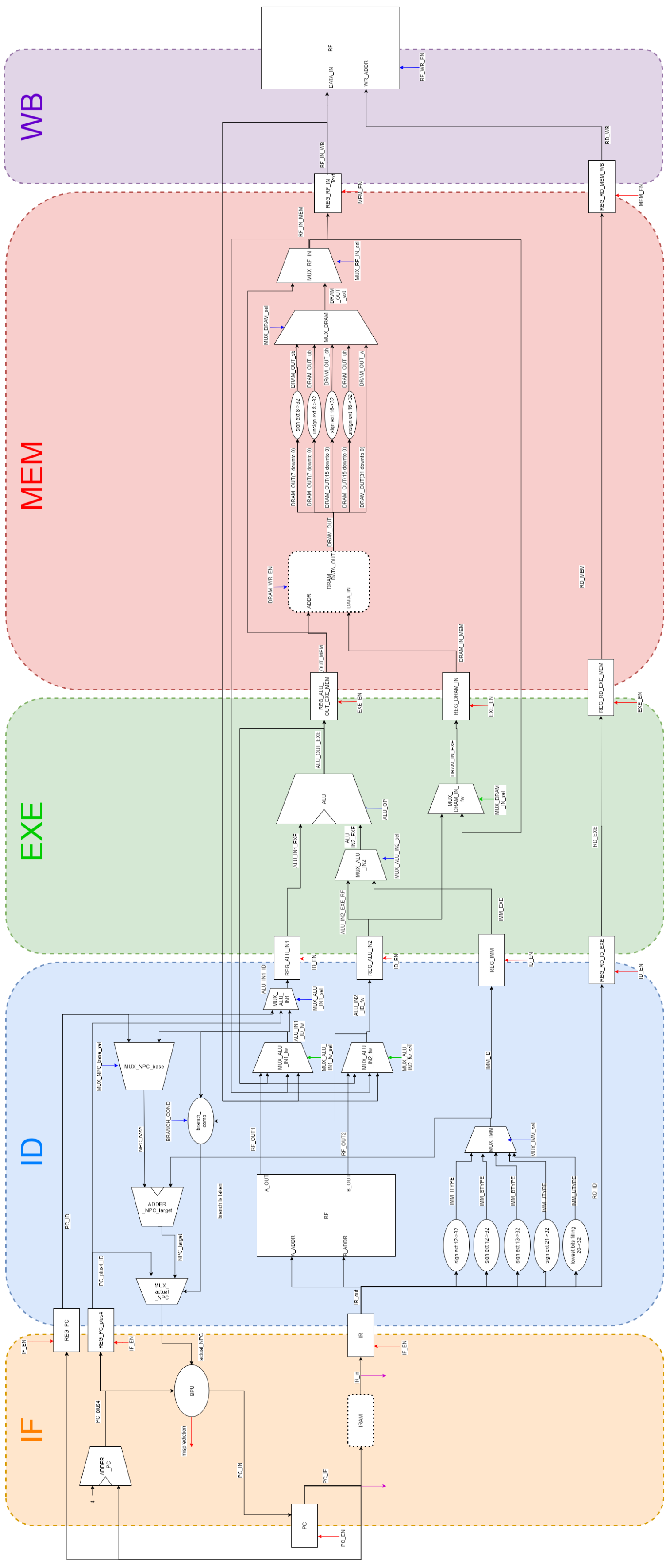


Figure 2.1: Datapath schematic. For clarity, the RF is represented twice, in the ID stage only for the reading-related ports, and in the WB stage for the write ports. The blue arrows represent the control from the CU, red arrows represent control from HDU, green arrows control from FU, purple arrows are signals read by the BPU.

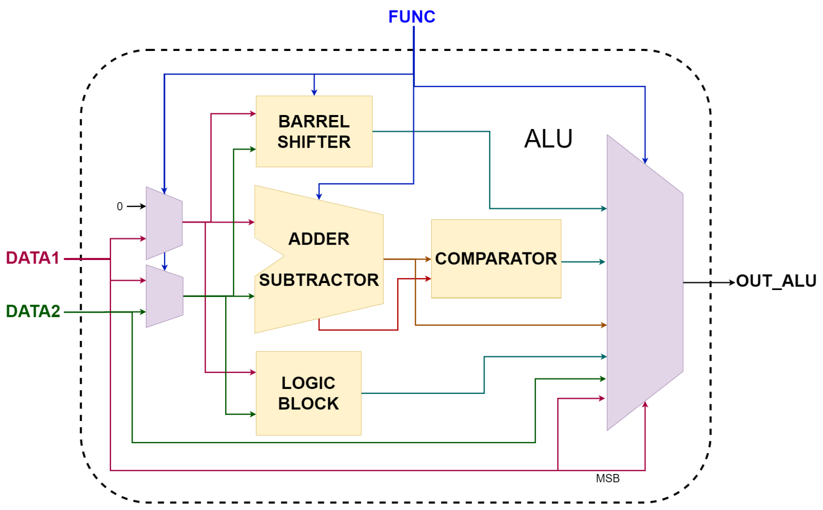


Figure 2.2: ALU internal diagram.

4. The fourth stage is aimed to read and write the DRAM. As for IRAM we can ideally place the DRAM here, without forgetting that it is not part of the datapath and will be connected from the outside. The address input and the data input of the memory are driven by the ALU and the second register file output respectively, obviously through the appropriate number of registers to assure the correct timing, while the data output has a slightly more complicated circuit: there are a set of bitwidth-extenders whose outputs are selected by a MUX controlled by the CU. Depending on the type of load required (byte/half-word/word, signed/unsigned) only one extender is read. One last MUX choose the data to send to the following stage beetween the extended DRAM output or the ALU output.

5. The fifth and last stage, WB (Write-Back), is where the RF is write.

2.1 ALU

The ALU structure is shown in figure 2.2. It has as inputs two operands and a signal that says what operation has to be carried out, and as output the result of the operation between these two operands. Inside it we find the following components:

- Adder/subtractor: it has the task to compute the addition or the subtraction between two inputs. The latter is needed not only when the corresponding instruction is executed but also when conditional-set instructions are performed. In fact this output is read by the comparator together with the carry out, and depending on these all the comparisons are made;
- Comparator: as just mentioned, it reads the result of the subtraction and the carry out and according to the type of comparison it set or not the first bit of the output;
- Barrel shifter: it allows to perform variable logical and arithmetical shifts in the two directions;
- Logic block: this is the block that computes the bitwise logic operation between the two inputs;
- Some MUXes: the biggest one selects the result of one of these blocks and send it to the ALU output, according on the FUNC signal. The other two are used to compute the absolute value, as will be explained below.

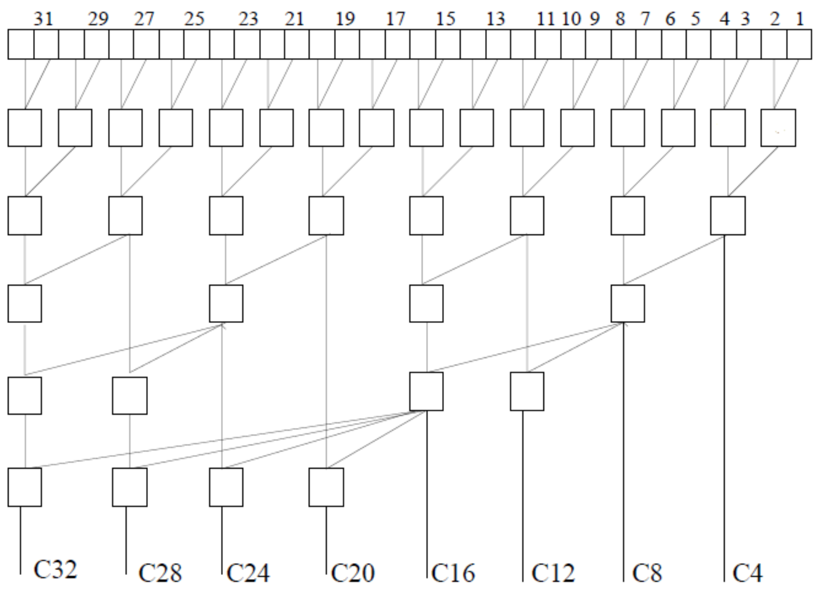


Figure 2.3: Parallel prefix network of the Intel Pentium 4.

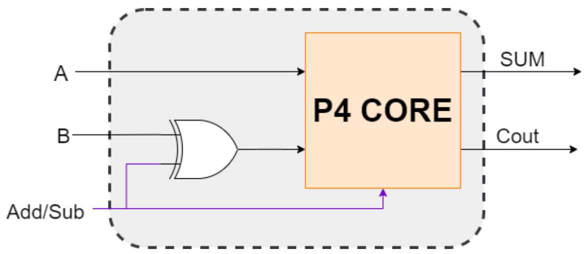


Figure 2.4: Circuit used to perform either an addition or a subtraction starting from an adder.

Since this is the only component, together with the control unit, that has additional hardware in order to support the absolute value operation, if we want to synthesis the processor without this feature, it is sufficient to comment the lines of VHDL aimed to instantiate the two MUXes (actually they are just an if/else branch inside a process). Another line can be saved, with a very risible area improvement, in the control unit VHDL but this will be explained in section 3.1.

2.1.1 Adder/subtractor

The chosen architecture is the one adopted in the Intel Pentium 4. It consists in a hybrid carry-lookahead/carry-select adder. In particular we can divide the architecture in a block dedicated to the sums generation and a block for the carries generation. The first use the same approach of a carry-select adder, but the carry inputs of each carry-select block are computed by a parallel prefix network very similar to a Ladner-Fischer tree (figure 2.3). The advantage of this type of approach is to exploit the speed of a parallel prefix adder but computing only a smaller number of carries thanks to the carry-select blocks, and so obtaining a smaller area compared to a traditional parallel prefix adder.

In order to also use this adder as subtractor, the circuit of figure 2.4 is adopted, ehich work in the following way:

- An Add/Sub signal is added, when it is set to 0 an addition is performed, when it is 1 it perform a subtraction.

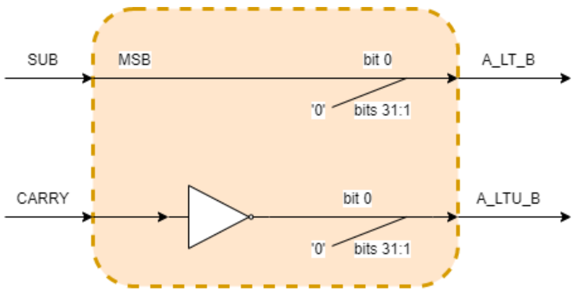


Figure 2.5: Comparator internal schematic.

- Each bit of the second input is sent in a xor gate together with the Add/Sub signal. So if this signal is low nothing happens and the operand pass unchanged, otherwise if it is high the xor gate acts as a not gate and inverts all the bits of the operand.
- The Add/Sub signal is connected to the carry in of the adder, so if it is high we are adding a 1 to the second operand that was previously inverted and this corresponds to get a 2' complement of the number.

2.1.2 Comparator

This component is needed for the conditional-set instructions. In particular it receives the result of the subtraction between the two operands and the carry out, and it has two 32-bit outputs. The first is equal to 1 (in decimal) if and only if the first operand is smaller than the second one and they are signed integers, otherwise it is equal to 0. The second output execute the same operation but in the case of unsigned operands.

The schematic is shown in figure 2.5. The signed comparison is made just observing the sign of the result, while for the unsigned comparison we have to look at the carry out: if this is low that means the second operand is greater than the first and vice-versa.

2.1.3 Barrel Shifter

This is the component able to execute a shift of an arbitrary amount of positions. The shift can be left or right, and in this last case logical or arithmetical. All these operations are completed in a single clock cycle thanks to the use of a cascade of MUXes. This architecture is used in the UltraSPARC T2 and is shown in figure 2.6a.

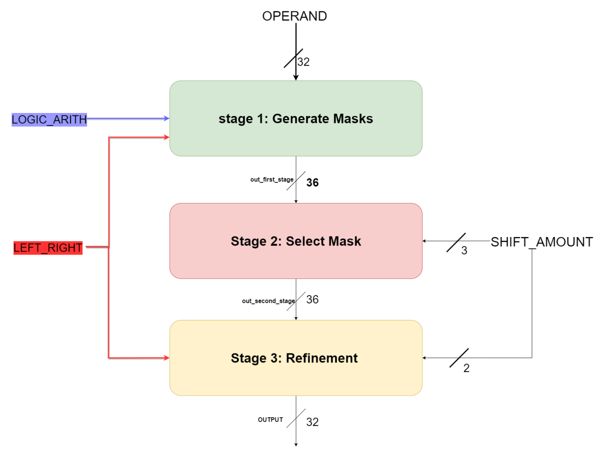
The first stage of the circuit is dedicated to the creation of eight "masks". Each mask has 36 bits and corresponds to the operand shifted by a number of position equal to a multiple of four (0, 4, 8, ...), left or right depending on the specific control signal. The incoming bits are filled with zeros or ones according on the logic/arith input, and on the shift direction. All these masks are generated with the same basic block, reported in figure 2.6b.

The second stage (figure 2.6c) takes in input all the masks generated on the previous stage and driven by the three most significant bits of the shift amount, selects the one closest to the requested shift. At this point the signals are still on 36 bits.

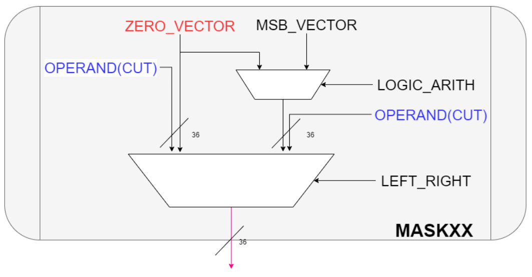
The final stage (figure 2.6d) read the shift direction and the last two bits of the shift amount, and according to these takes only 32 consecutive bits of the 36 incoming.

2.1.4 Logic Block

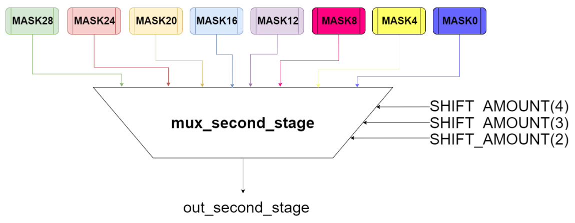
This block is designed to perform simple logic operations between the two operands. Inside it there are an AND, an OR, and a XOR gates, as shown in figure 2.7, each of 32 bits.



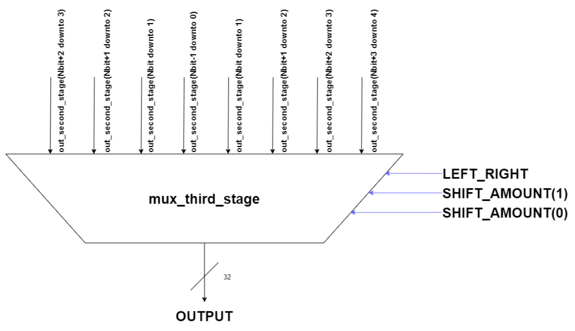
(a) Top view.



(b) Stage 1: masks generation.



(c) Stage 2: mask selection.


out_second_stage(Nbit+2 downto 3)
out_second_stage(Nbit+1 downto 2)
out_second_stage(Nbit downto 1)
out_second_stage(Nbit-1 downto 0)
out_second_stage(Nbit downto 1)
out_second_stage(Nbit+1 downto 2)
out_second_stage(Nbit+2 downto 3)
out_second_stage(Nbit+3 downto 4)
mux_third_stage
LEFT_RIGHT
SHIFT_AMOUNT(1)
SHIFT_AMOUNT(0)
32
OUTPUT

(d) Stage 3: fine-grain shift .

Figure 2.6: Barrel shifter diagrams.

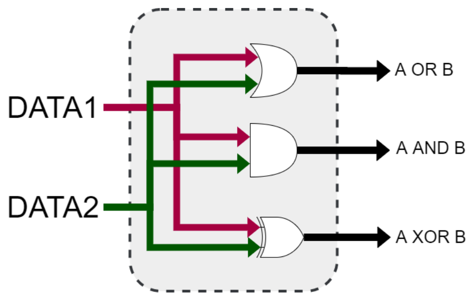


Figure 2.7: Logic block internal schematic.

2.1.5 Absolute Value

When an absolute value has to be computed, the MUXes on the inputs of the ALU (figure 2.2) send the number 0 to the non-inverting input of the adder/subtractor, and the first operand of the ALU to the inverting input of the ALU. The add/sub signal is driven in order to perform a subtraction and so this corresponds to obtain a 2's complement of the operand. The output MUX select this result or the original operand according on whether its sign is negative or positive.

2.2 Branch Comparator

As mentioned at the beginning of this chapter this component is used in the ID stage. It has the task to drive the multiplexer that send to the BPU the actual address after a branch/jump is fetched, in order to verify the correctness of the prediction. It receives a test condition from the CU and two words to be tested. The output, that is a single bit, is rised if the test condition is verified in the two words, otherwise it is lowered. The test conditions can be:

- never: the instruction is not a jump or a branch;
- always: the instruction is an unconditional jump, so it is always taken;
- equal: the branch is taken if the two operands are equal;
- not equal: the branch is taken if the two operands are not equal;
- less than: the branch is taken if the first operand is smaller than the second and they are signed;
- greater or equal: the branch is taken if the first operand is equal or bigger than the second and they are signed;
- less than unsigned: the branch is taken if the first operand is smaller than the second and they are unsigned;
- greater or equal unsigned: the branch is taken if the first operand is equal or bigger than the second and they are unsigned;

2.3 Branch Prediction Unit

The adopted BPU is a branch target buffer using a set-associative cache and a LRU (least recently used) policy replacement. Despite it is designed in a totally generic way, in this implementation it has been chosen to organize the cache in eight sets of eigth lines each, in which only eight bits of tag are stored. This choice leads to a small chance of collision (two branch/jump with a different address but the same set and tag fields). When it happens the prediction is based on the history of a wrong jump and very probably this will turn out to be a misprediction, but it allows to save a huge amount of area and the cache look-up is faster.

It is possible imagine the output of the whole BPU as a multiplexer that drives the program counter register and receives in input the address of the next sequential instruction, the branch/jump actual address computed in the ID stage, and the output of the cache. In normal conditions the next sequential address is reported to the output so as the program proceeds without any jump. As soon as a branch or a jump instruction is fetched, the value of the program counter is read and a cache look up is performed according on its set and tag fields. If there is a hit, the corresponding prediction is read from the cache and sent to the program counter instead of the next sequential address. This prediction is the actual address computed by the same branch/jump the previous time, so it can be

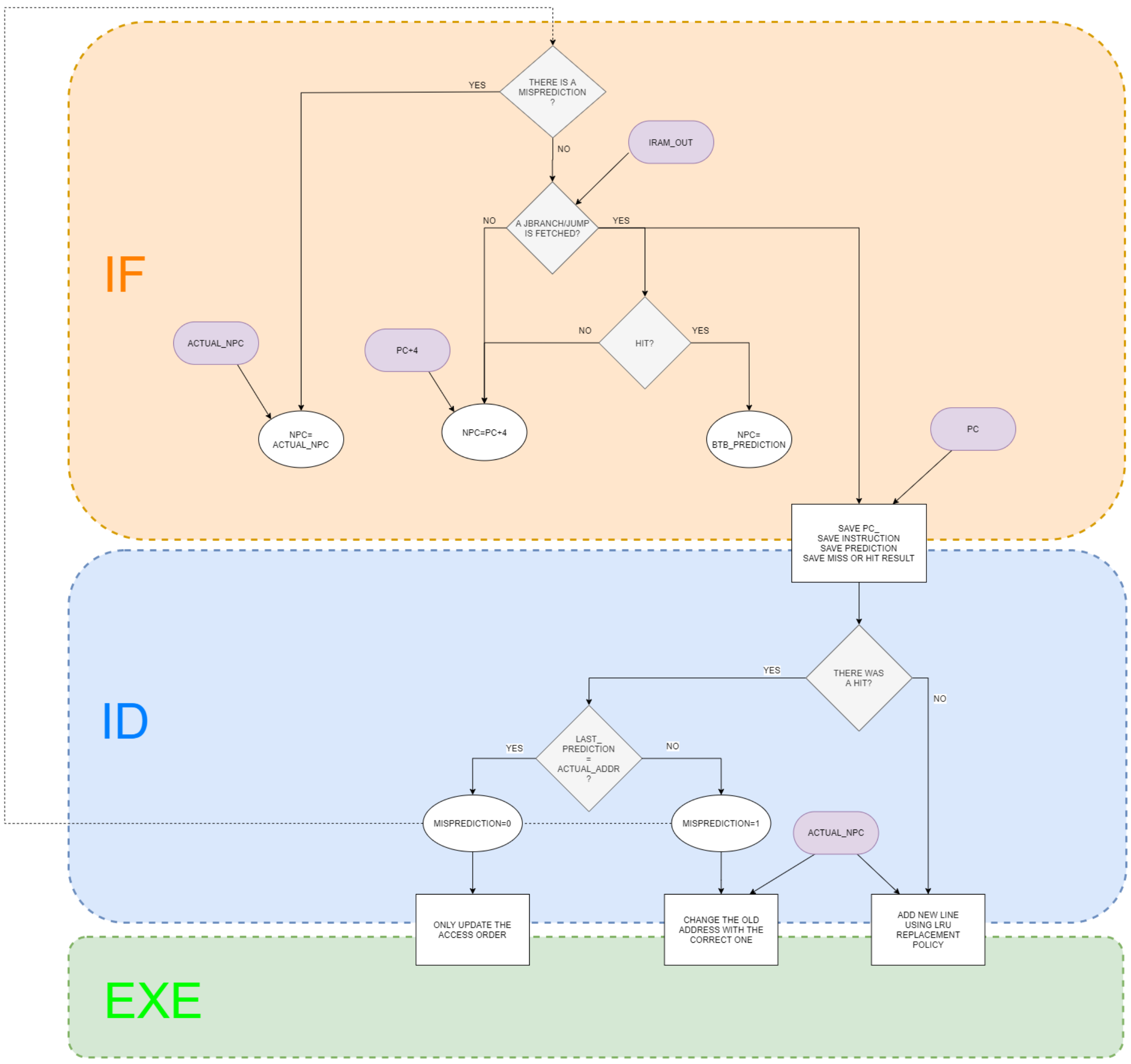


Figure 2.8: Flowchart of the BPU operation. The elliptical signals are combinational output, the squares content is perfomed at the rising clock edge, and the purple signal are the inputs from the outside.

either the next sequential address or the target address, according on whether the previous time the branch was taken or not. If instead there is a cache miss, the next sequential address is issued anyway. This corresponds to a not taken prediction in the case of a branch, or to a sure misprediction in the case of a jump. In both cases (hit or miss) at the rising edge of the clock, the program counter, the set and tag fields, the received address and the predicted one, and in the case of a hit also in which line it occurred, are temporarily saved. Moreover a flag is raised to say the BPU that in the next clock cycle it will have to check if the prediction is correct and, if not, to make all the necessary corrections. In the next clock cycle, once the flag is raised, the saved prediction is compared to the actual computed address and if they are equal nothing happens and only the access order inside the set is updated. Otherwise if the prediction is different from the actual address and so there is a misprediction, the HDU is informed that it will have to discard the fetched instruction, losing a clock cycle (see chapter 3.3), and the cache is updated. If the misprediction is due to a cache miss, a new data is added according to the LRU policy, using the saved set and tag and storing the just computed actual address. If otherwise the misprediction is due to a wrong prediction, only the address and the access order of the set are updated with the new ones.

In figure 2.8 the procedure just described is shown schematically.

2.4 Register File

The used RF has two output ports and one writing port, these numbers are sufficient to avoid any structural hazard due to this component.

The only special register is the one number 0. It is hardwired to zero, this ensure that it cannot be changed even after a program error, in this way there is always a register that can be used to store an immediate value in a register, to move a register into another, to use a direct addressing mode for a load/store, or to perform other types of instruction or pseudo-instruction that need a zero in one of the two registers.

CHAPTER 3

Sequential Units

As mentioned in chapter 1 and illustrate in figure 1.1, there are three totally independent units that drive the datapath:

- The Control Unit (CU), which send to the datapath the correct signals in order to carry out a specific operartion according on the decoded instruction;
- The Forwarding Unit (FU), to detect a data dependence and drive the forwarding MUXes in order to bypass the data between the pipeline stages;
- The Hazard Detection Unit (HDU), that enable or not the pipeline registers and can insert a bubble when it detect a hazard.

3.1 Control Unit

The CU is developed using a mixture of the micro-programmed and the hardwired techniques. It is based on some look-up-table, one for each type of instruction: branch, load, store, immediate operations, register-to-registers operations, and others. The format of the different types of instruction and how to decode these is explained in the appendix A, but here the important things is that the type of instruction is distinguished looking the OPCODE field. According to this field the CU looks the appropriate LUT, which is addressed by the FUNCT3 field or directly by the OPCODE if the instruction is classify as "others".

This is very similar to the micro-programmed approach but it is important to notice that since there is no a "micro-program-counter", it can be synthesize as a combinational circuit by the synthesizer. In this way the advantages of the two types of implementation are exploited: on the one hand it is very simple to join new instructions, to modify the existing ones, or to add new outputs, on the other hand it keeps a lower area usage, the simplicity and speed of a combinational circuit, and it is well suited to be used in a RISC pipeline context.

For the correct timing of the signals within the pipeline, registers are used outside the control unit, as you can see in figure 1.1, in this way the CU is a completely combinational block.

If we want to synthesise the design without the custom instruction, in addition to avoid the components used for this purpose in the ALU, we can save more area simply commenting the line of the LUT where the control outputs of this instruction are stored. In this way its control signals will be substituted with "don't care" assignments, letting the synthesizer choose the better values in order to simplify as much as possible the LUT.

3.2 Forwarding Unit

The FU is designed as a standalone block and it is completely independent from the CU. It receives the instructions present in each stage of the pipeline from the ID stage to the WB stage (see figure 1.1) and according to these it controls the selection inputs of the forwarding MUXes (see figure 2.1). In particular there are two case of forwarding:

1. An instruction in the decode stage needs a data that has been previously computed by another instruction, or that has already been read from the memory, or that is a link address, but this data has yet to let the WB stage. In the case in which the needed data come from a load instruction that has not yet read the DRAM (this happens when the instruction that requires the data is the immediately following one) there are two possibilities: if the instruction that requires the data is a store that has to save the same data coming out from the memory, it is possible to exploit the forwarding in the next clock cycle (see the following case). If instead the instruction requiring the data is not a store or is a store but the data is needed to compute the DRAM address, it is not possible to exploit the forwarding and the HDU will intervene inserting a stall in the execute stage and blocking all the precedent stages of the pipeline (see chapter 3.3).

2. As just mentioned, if a store has to write immediately the same data read by a load, it is possible to exploit the forwarding when the store is in the execute stage, in fact at this point the value to be written is already come out from the memory and it can be reported at the data input.

The first case can occurs for both the source registers and so both have to be checked. Furthermore, if the required register is the number 0, the forwarding mustn't intervene because assuming that a hypothetical program tries to write without any consequence in this register, all its following readings must return zero independently on the instant when they are performed.

3.3 Hazard Detection Unit

The HDU is the block that has the task of controlling the progress of the pipeline stages, blocking some of them and inserting a stall if a hazard is detected. How and when it intervenes is explained below:

1. As already explained in section 3.2, if there is a data dependency between a load instruction and its following one, and the second is not a store that have to write the data just read, a data hazard occurs. This condition is verified in the decode stage, comparing the register sources of this stage with the register destination of the load. In the event that a hazard is detected, the program counter, the IF/ID registers, and the ID/EXE registers are disabled, and a bubble is introduced in the EXE stage. In this way an additional clock cycle is required but once the data will be extracted from the memory it will be possible to use the forwarding to bring it back to the ALU input.

2. The second case is a control hazard and it occurs when the BPU makes a misprediction. When this happens a wrong instruction is entered in the fetch stage, so the HDU intervene substituting the mispredicted instruction with a bubble.

3. The third and last possible hazard is a structural one, think for adding multi-cycle operation in future upgrades of the processor. When these types of operation (for example a multiplication or a division) comes in the execute stage, all the previous stages has to be stopped until the multi-cycle operation is completed. This is the only hazard that requires a sequential circuit to be detected, in fact a counter starts to count from zero when the instruction reach the EXE

stage, and as long as this count does not reach a value equal to the number of cycles required for the completion of the instruction, all the IF/ID, ID/EXE, and EXE/MEM registers and the program counter are disabled, and bubbles are introduced in the MEM stage.

CHAPTER 4

Memory and addressing space

Both the memories (IRAM and DRAM) are simulated with a VHDL file only for the testbench, because they are not synthesizable with the used library and anyway they are designed to be connected from the outside.

The two address spaces are disjointed, so a specific address corresponds to only one specific memory. Actually it would be possible to overlap the addresses for instructions and for data but this is not compliant with the RISC-V specifications.

In this design the IRAM is assumed to keep the space 0x0040XXXX, while the DRAM handle the space 0x1001XXXX. Both the memory has 16 bits of address and start from 0x0000 so they are activated by means of the chip select only when the other 16 bits correspond to their address space. The program counter is designed to start from 0x00400000 when reset, so there is no need to jump to this address at the power on.

The two memories are designed to have the same reading behavior and the same internal organization structure, but since the IRAM is only initilized and then it doesn't require to write new data inside it, the data input and the write enable are not present. The DRAM instead has a data input and a synchronous write enable to allow writing in runtime.

The internal structure is divided in cells of 8-bit in order to allow the writing or the reading of a single byte, an half-word (two bytes), or a whole word (four bytes). Once the address of a byte is sent to the memory, it will output not only the corresponding byte, but also the three consecutive bytes, concatenated to form a single word in which the byte with the highest address is the most significant. For the DRAM it will be up to the datapath controlled by the CU to choose whether to take only one, two or all four bits and in the first two cases also whether to extend with or without sign, based on the type of load required by the program.

The initialization of the memories is performed at the beginning of the simulation by means of two different files, one for the IRAM, and one for the DRAM. The program and the data has to be written in hexadecimal format, in lines of one word each, inserting a line break after each word. To obtain this format it is possible to process the binary with a bash script or to use a program like RARS. Remember that the files has to be put in the Modelsim directory.

Regarding the runtime writing of the DRAM, it has a two bits write enable that is read at the rising edge of the clock. With this signal it is possible to save in the memory a different number of bytes starting from the indicated address:

- Write enable equal to 0b00: nothing is done;
- Write enable equal to 0b01: only the least significant byte is written;
- Write enable equal to 0b10: the least significant half-word is written;

- Write enable equal to 0b11: a whole word is written.

Note that in the case of an half-word or a word writing the bytes that compose the data has to be sent in a little endian way, and in the case of a byte or an half-word writing the following three or two bytes in the memory respectively are not modified.

Finally, we underline that with this memory architecture, it is easily possible to add new data memories or new I/O devices simply connecting properly the chip select in order to give the wanted address space, and putting together the data and address ports with the ones of the DRAM (this works if the outputs are three-stated, otherwise a MUX is needed).

CHAPTER 5

Testbench

In order to test the design, the memories described in the previous chapter are connected to the processor using a verilog top-level testbench, driving properly the chip selects. In addition to the DRAM, the IRAM, and the DUT (device under test), there is also a clock and reset generator block needed to initialize the sequential components and to feed them with a clock. In figure 5.1 this organization is shown.

The design without the absolute value instruction is tested initializing the memories with the program in listing C.5, which computes the minimum absolute value of an array of integers. In this program the absolute value is computed via software using more instructions. The design with the custom instruction is instead initialized with the program of listing C.6, which substitutes all the instructions used to compute the absolute value with the new ABS instruction. In both cases, if the design is correct, at the end of the program we have to find the number 3 inside the DRAM cell of address 28.

Both the designs are simulated three times: pre-synthesis, post-synthesis, and post-place-and-route. The post-synthesis and post-place-and-route simulations are required in order to verify that the netlists generated by Synopsys Design Compiler and Cadence Innovous work properly and there are no non-synthesizable VHDLs. In figure 5.2 and 5.3 you can see the ending of the simulation of the program without and with the custom instruction respectively. The post-synthesis simulation is not shown in the figures because it is equal to the post-place-and-route one. As we can see from the figures, in the post-place-and-route simulations it is not possible to see the internal signals because the synthesizer collapsed the hierarchy and produced a flat design, renaming all the modules and the internal signals, so only the connections used in the testbench are visualized. Anyway we can observe that the addresses given to and the data coming from the IRAM are the same and in the same sequence

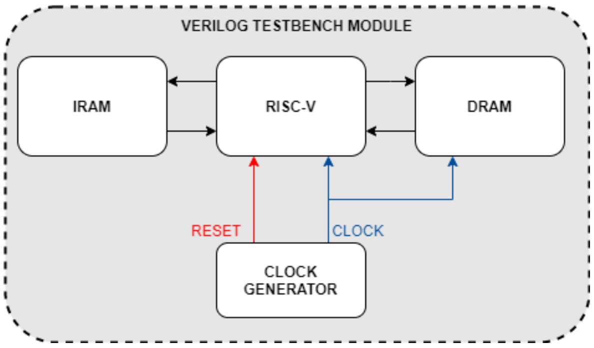
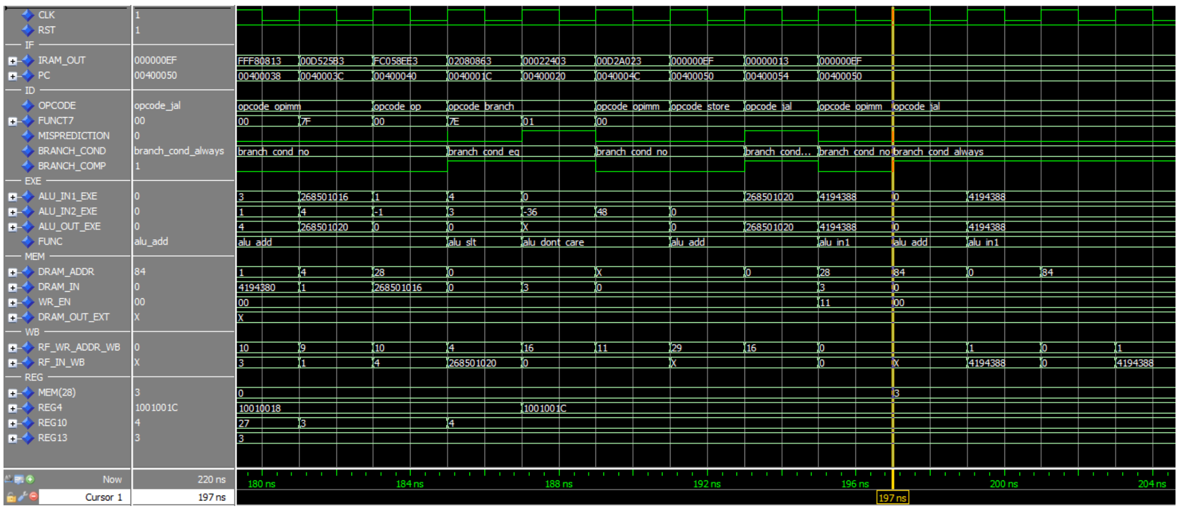
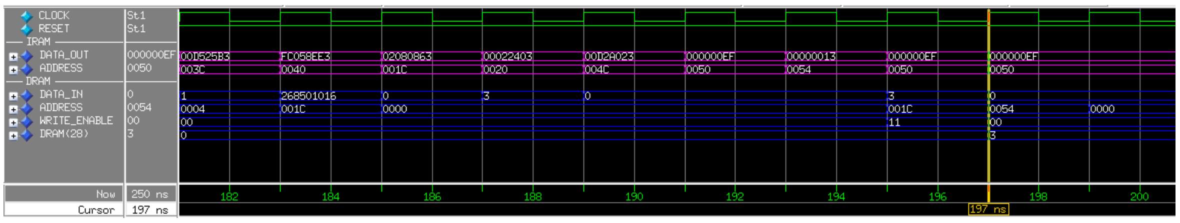


Figure 5.1: Testbench structure. White blocks are written in VHDL, grey one in Verilog.

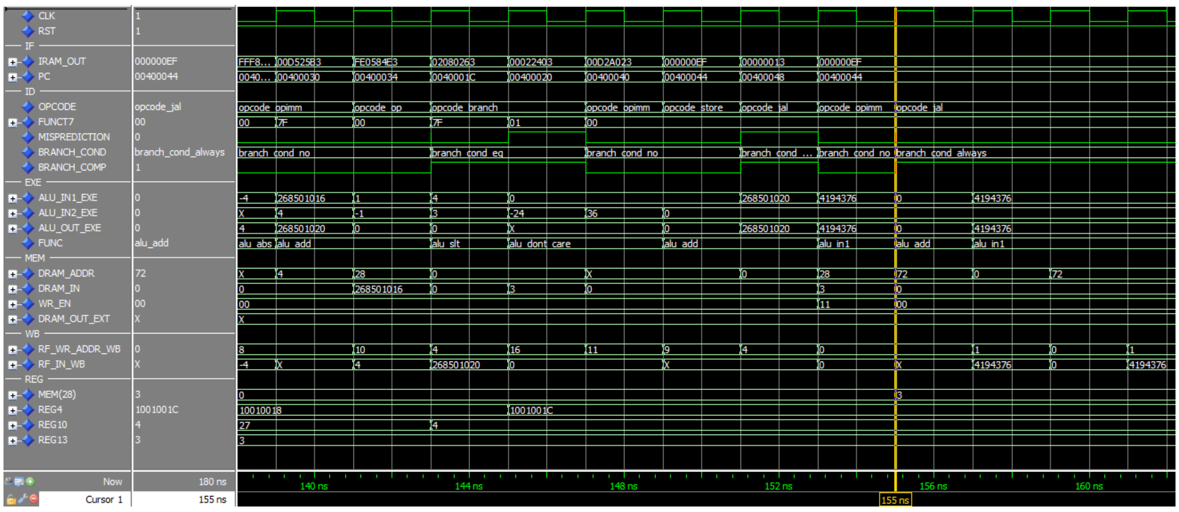


(a) Pre-synthesis.

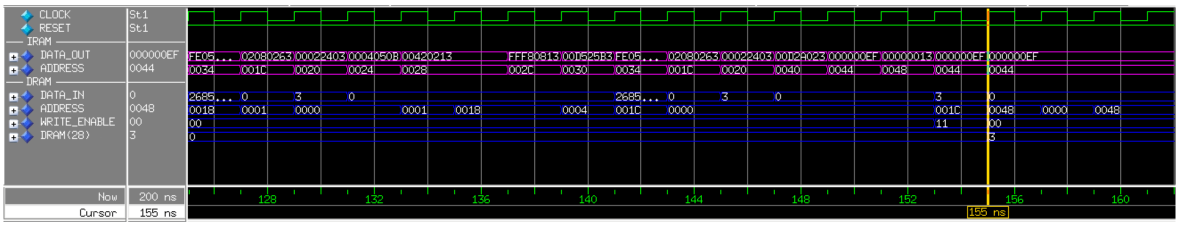


(b) Post-place-and-route.

Figure 5.2: End of the simulation of the program without the custom instruction.



(a) Pre-synthesis.



(b) Post-place-and-route.

Figure 5.3: End of the simulation of the program with the custom instruction.

as the pre-syntehsis simulation, and at the end of the simulation the number 3 is correctly written in the cell 28 of the DRAM.

Finally, a comparison between the two version of the program can be made. The one without in-hardware absolute value writes the result after 197 ns. Since the clock period is equal to 2 ns, it takes 98 clock cycles to complete the computation. The number of clock cycle is rounded down because of the initial reset pulse. On the other hand, the improved program lasts for 155 ns, so it takes 77 clock cycles. The overall speedup is therefor 21.43%.

CHAPTER 6

Synthesis

In order to make a comparison not only at the functional level on the number of clock cycles but also at the physical level on the power consumption, area, and maximum frequency, the design is synthesized twice. One time without modifying any VHDL files, and so including also the additional hardware that supports the absolute value instruction, and the second time commenting all the lines related to this custom instruction, and so avoiding this additional hardware. As a reminder, the only two components to modify are the ALU (section 2.1) and the CU (section 3.1).

Both the synthesis are performed using the TCL script of listing D.2, which execute the following tasks:

1. Loads all the needed files;
2. Elaborates the top-level entity setting the wanted parameters and performs a formal verification;
3. Gives a maximum critical path constraint of 2 ns (for both clock and combinational inputs/outputs) and an uncertainty of 0.07 ns to take into account the clock jitter, and set a maximum delay for inputs and outputs;
4. Synthesis the design in the ultra mode;
5. Performs retiming of sequential cells on the obtained netlist;
6. Generate SDC, SDF and the synthesised netlist in verilog format;
7. Generate timing, area and power reports.

The parameters of the top level entity are three, all related to the BPU:

- BPU_TAG_FIELD_SIZE set to eight. It specify the size in bits of the TAG field;
- BPU_SET_FIELD_SIZE set to three. This configure the bit width of the SET field and consequently the number of sets that make up the set-associative cache. In this case eight setS are generated;
- BPU_LINES_PER_SET set to four. That is the number of lines for each set of the cache.

Before running this script, a simpler synthesis without any type of optimization was performed, setting a maximum clock period of 0 ns in order to find an estimation of the maximum reachable frequency. With this synthesis, which doesn't flat the hierarchy, it was also possible to find the bottleneck of the design, that is the path shown in red in figure 6.1. The fault is the forwarding that

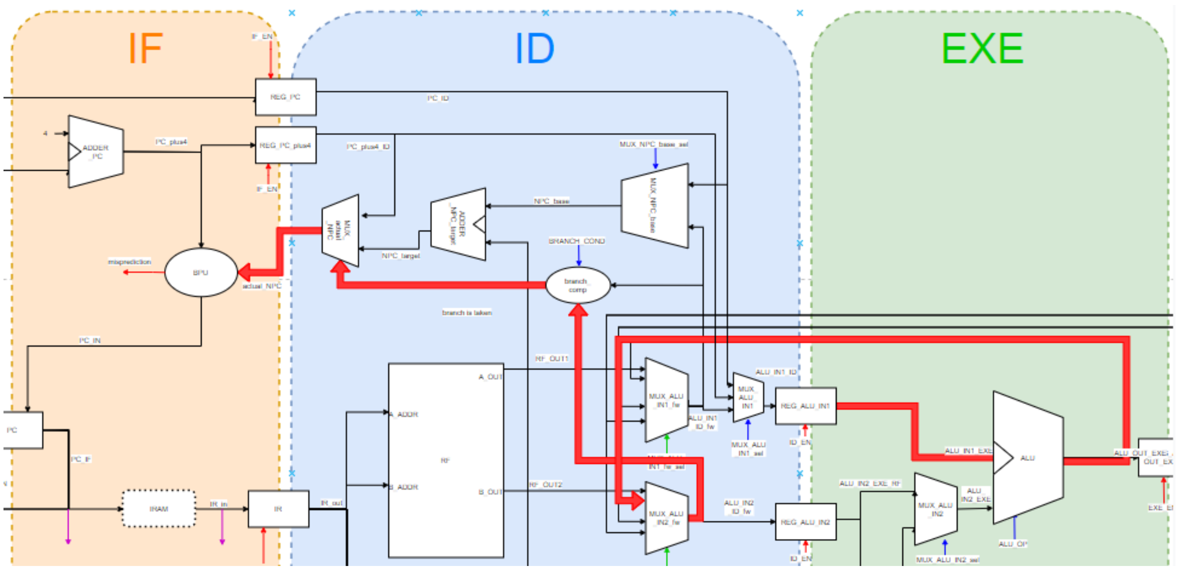


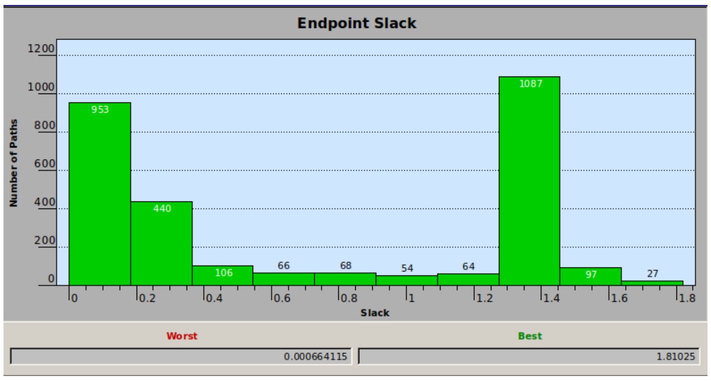
Figure 6.1: Critical path.

collapses the critical path of the ID and EXE stages together in a single path. A solution to avoid this is to not allow forwarding to conditional branches. Anyway, this doesn't mean necessarily a better performance because even if the clock frequency could be higher, the number of clock cycles would be increased.

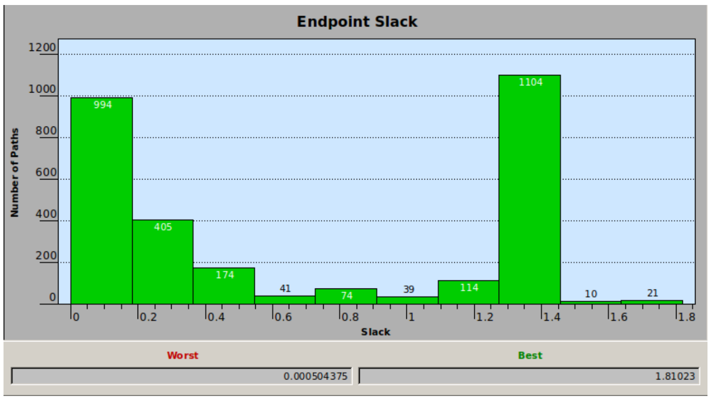
Found an estimation of the maximum reachable frequency that is 578 MHz for the design without the custom instruction, and 571 MHz for the design with the custom instruction, it has been chosen to constraint the final synthesis to 500 MHz for both the designs, in order to leave room for power improvements and to allow a better comparison.

In figure 6.2 it is possible to see the slack distribution of the two final synthesis. We can notice two groups of paths, one on the left so with a small slack (0 ns to 0.4 ns approximately), and one on the right so with a higher slack (about 1.3 ns - 1.4 ns). The left group is a sign that the syntehsizer did a good job: it has constrained all the longest path to a null slack using as many fast gates as necessary but without exceeding in terms of power consumption, and at the same time it has shift to the left as much as possible all the shortest paths using low-power gates. The right group of paths probably is due to the many brief paths that are already using only low-power gates and so have no more room for improvements.

The other synthesis reports are summarized in the tables, 6.1 and 6.2. In particular for the power consumption are reported two estimation: the one obtained immediately after the synthesis, and the one obtained using jointly Mentor Modelsim and Synopsys Design Compiler in order to get a power consumption based on the switching activity of the testing program. Considering that it is slightly variable from a program to another, we can say that the power consumption of the two designs is approximately the same. Regarding the area, we can confirm that as expected, it is a bit higher in the design with the in-hardware absolute value.



(a) Design without the custom instruction.



(b) Design with the custom instruction.

Figure 6.2: Post-synthesis slack distribution.

Custom instruction	**Combinational Area** [µm^2]	**Non-combinational Area** [µm^2]	**Total Area** [µm^2]
No	12260.20	15232.49	27492.69
Yes	12497.21	15307.50	27804.71

Table 6.1: Area reports.

Custom instruction	**Switching-activity estimation**	**Internal Power** [mW]	**Switching Power** [µW]	**Leakage Power** [µW]	**Total Power** [mW]
No	No	10.5	583	492.5	11.6
	Yes	11.1	1650	508.5	13.3
Yes	No	10.6	636	501.1	11.8
	Yes	11.1	1580	518.4	13.2

Table 6.2: Power consumption reports before and after the switching-activity estimation

CHAPTER 7

Place-and-route

The place and route is executed by Cadence Innovus using the script of listing D.3. The sequence of operations is the following:

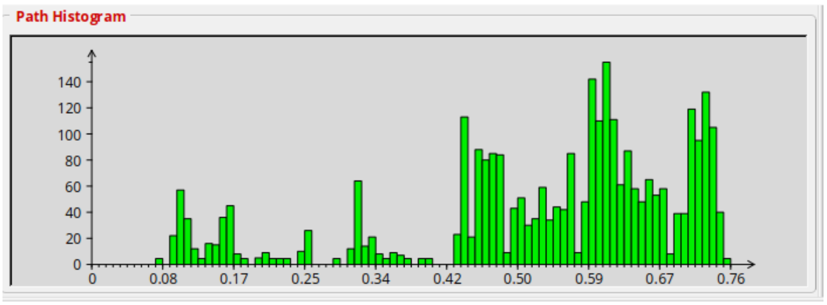
1. Load the synthesized netlist, the cell library, and the configuration files;
2. Specify the size of the silicon die and its core;
3. Add a power ring for VDD and one for VSS;
4. Prepare the horizontal wires for the cells power supply;
5. Place the cells on the core;
6. Optimize the design to achieve the required timing constraints;
7. Place fillers;
8. Route all the connections;
9. Post-routing optimization;
10. Save the project and the layout picture;
11. Extract the parasitic impedences;
12. Verify the connectivity and the design rules;
13. Write reports, generate post-place-and-route netlist, and annotate the delay.

In table 7.1 some measurements on the layout area are reported and it is possible to compare this result with the one obtained during the synthesis phase: the results are quite similar.

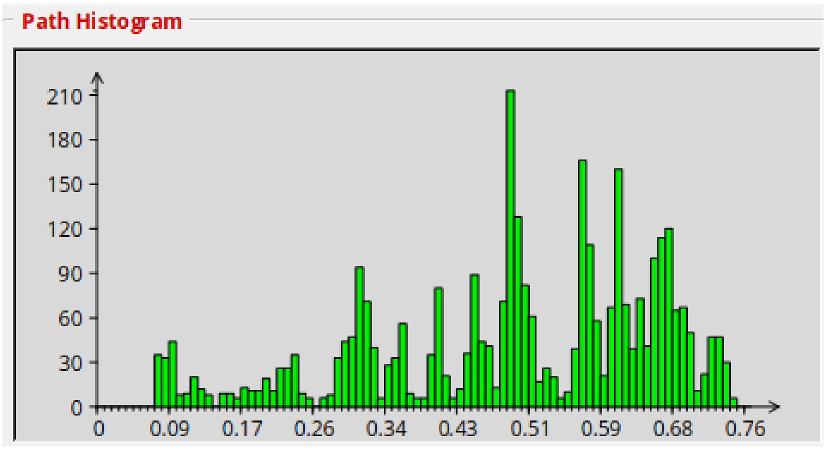
In figure 7.1 the post-place-and-route slack distribution is shown, now the two groups of paths are vanished and there is a more uniform distribution, with slightly less paths in the left half than in the

Custom instruction	**Gates**	**Cells**	**Total Area** [μm^2]
Yes	34574	12845	27590.1
No	35124	13353	28029.2

Table 7.1: Area reports.



(a) Without the custom instruction.



(b) With the custom instruction.

Figure 7.1: Post-place-and-route slack distribution.

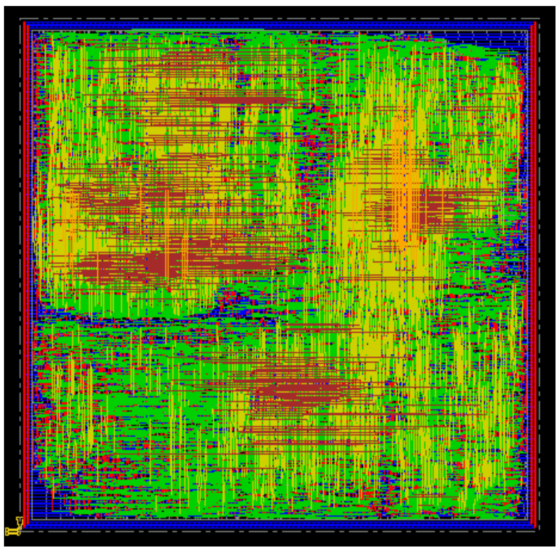
Custom instruction	**Internal Power** [mW]	**Switching Power** [mW]	**Leakage Power** [µW]	**Total Power** [mW]
No	10.27	4.23	541.9	15.04
Yes	10.32	4.335	554.7	15.21

Table 7.2: Post-place-and-route switching-activity-based power consumption estimation.

right one. There are no more paths with a slack between 0.76 nm and 2 nm, and the critical paths are moved toward right letting no 0 nm slack paths. This is due to a more accurate estimation of Cadence Innovus than Synopsys Design Compiler, which has no information about the layout level, and thanks to the final physical optimizations.

At this point the switching-activity-based power consumption is again evaluated using Modelsim and Innovus together. The outcomes are reported in table 7.2, where you can see that now the switching power is more than doubled because of the interconnections.

Finally, in figures 7.2a and 7.2b an overview of the two floorplans is shown.


(a) Without the custom instruction.

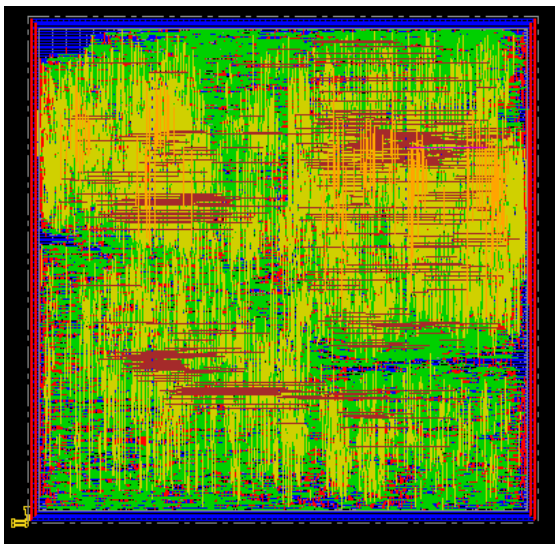

(b) With the custom instruction.

Figure 7.2: Floorplans overview.

CHAPTER 8

Possible improvements and future works

Despite this project work correctly at present, there are a lot of possible improvements that can be accomplished in order to get a better performance, more functions, and a smaller power consumption. Here only the most important are sketched:

- All the functional units inside the datapath work even if there is no reason to do it and only some of these are used. For example when two new data arrive in the ALU inputs and we have to compute an addition, not only the adder is working but also the logic block is computing all the logic functions between these two data, and the same for the barrel shifter and the comparator. But actually the output of these units will never be read and so there is a huge waste of dynamic power. A solution to avoid this is the power gating that consists in using some latches in order to shield the unused blocks and to not waste dynamic power when it is not necessary.
- Another idea to save dynamic power is like the previous one but for the sequential units. The clock makes them switch every clock cycle even if the output doesn't change. The solution for this is the clock gating, that is using a gate to mask the clock when the sequential unit is not used.
- One way to add functionality is to add also the FENCE, ECALL, and EBREAK instructions, in order to support completely the RV32I base instruction set.
- The performance can be improved using a 2-bit counter in the BPU, which would not change prediction following only a single misprediction.

APPENDIX A

Instruction Set

In the RV32I base ISA, there are six types of instruction formats:

- R-type for register-to-register operations;
- I-type for immediate operations, loads, and jump-and-link-register (JALR);
- S-type for stores;
- B-type for conditional branches;
- U-type for upper immediate instructions;
- J-type for jump-and-link (JAL).

In figure A.1 these formats are shown together with their bit fields organization.

The first operation to do in order to decode the instruction is looking at the OPCODE field, from this we can distinguish some instructions or groups of these:

- OPCODE=0b0000011: the instruction is a load;
- OPCODE=0b0001011: custom opcode, in this case the instruction is always an ABS instruction;
- OPCODE=0b0010011: the instruction is an immediate one;
- OPCODE=0b0010111: AUIPC instruction;
- OPCODE=0b0100011: the instruction is a store;
- OPCODE=0b0110011: the instruction is a register-to-register operation;
- OPCODE=0b0110111: LUI instruction;
- OPCODE=0b1100011: the instruction is a branch;
- OPCODE=0b1100111: JALR instruction;
- OPCODE=0b1101111: JAL instruction;

In the case of an ABS, AUIPC, LUI, JALR, or JAL opcode, at this point we already know the instruction and we can decode it using the appropriate format among those previously seen. Otherwise if the opcode belongs to a group of instructions (like for example a branch), we have to also look at the FUNCT3 field. This field will reveal the specific instruction among those belonging to that group

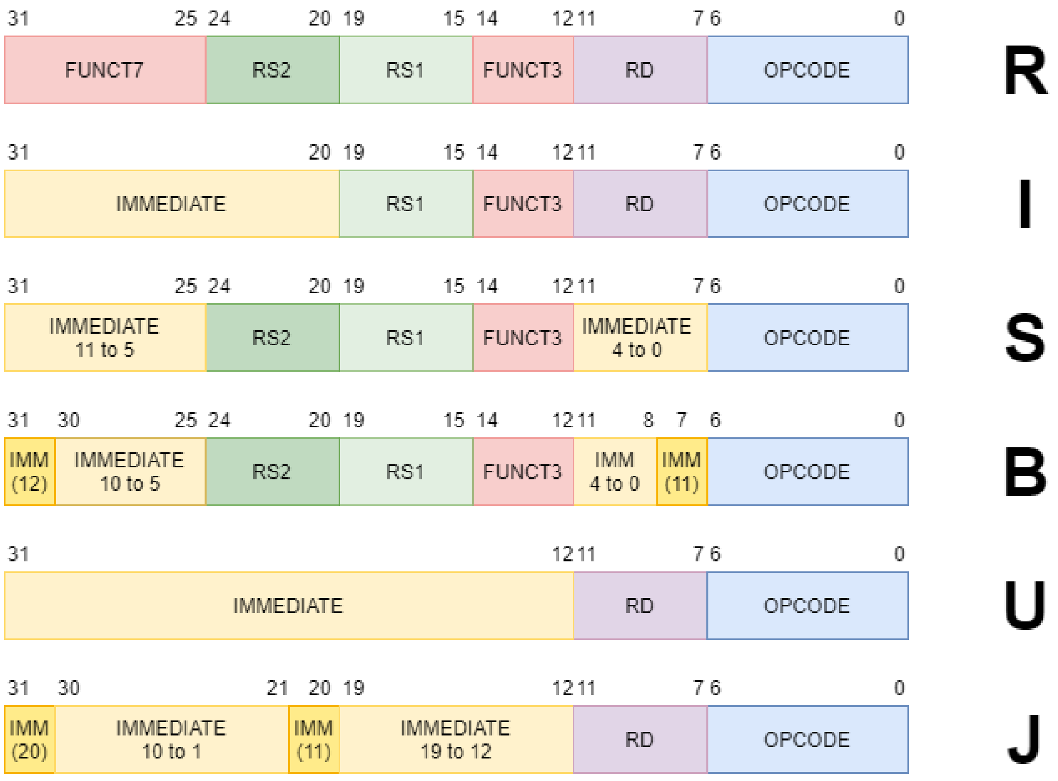


Figure A.1: Formats and relative fields of the different types of instruction.

(for example if the branch is a BEQ, or a BNE, or a BLT, and so on...). Only for the case of register-to-register operations, we have to look at the FUNCT7 field too. This is needed just to distinguish an addition from a subtraction, and a logical shift right from an arithmetical one. For completeness, in table A.1 all the supported instructions are summarized showing these fields. The meaning of each instruction is explained below. To additional information refer to the official RISC-V Instruction Set Manual.

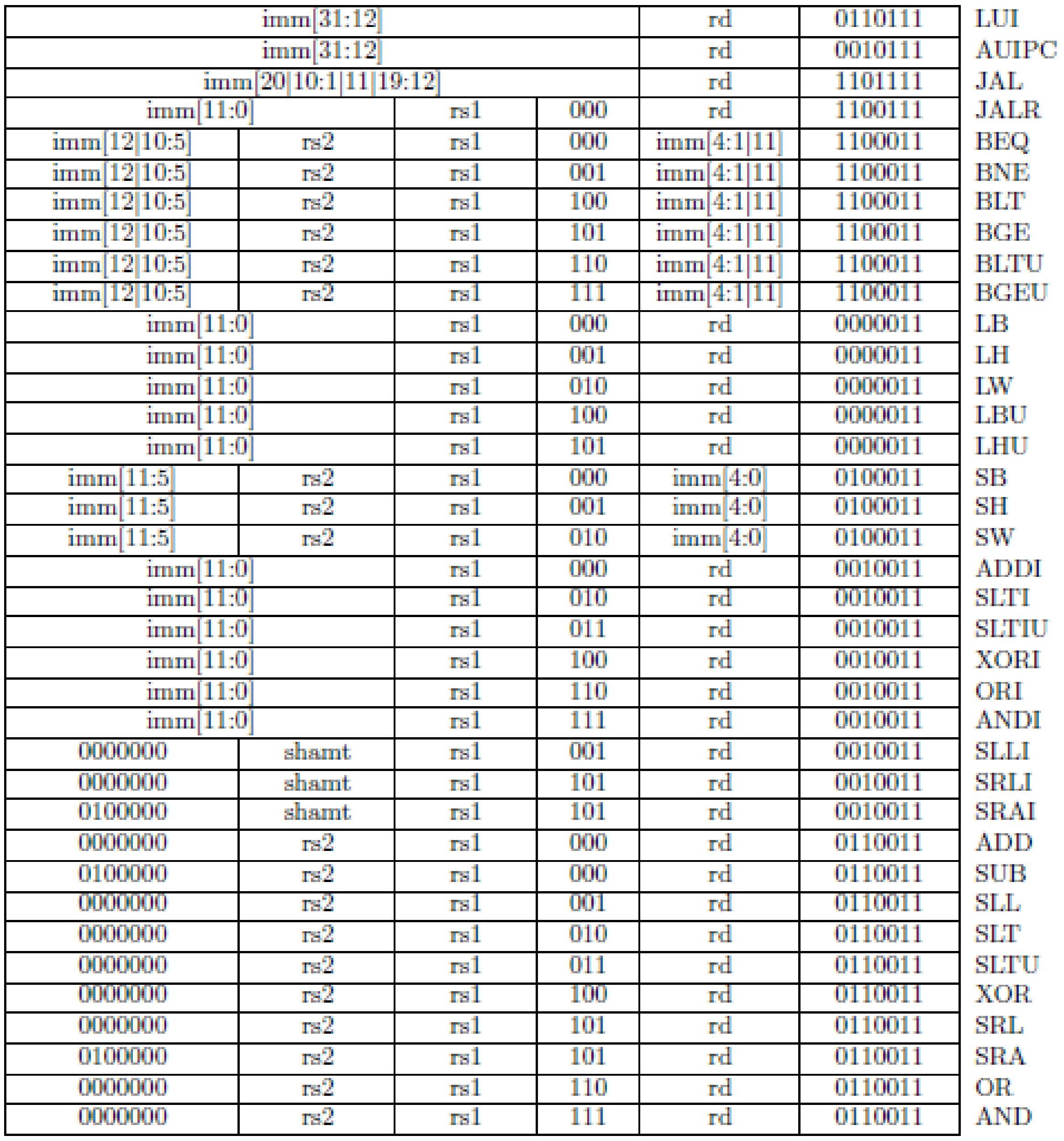
Register-to-register operations

- **ADD rd, rs1, rs2**
  $rd \leftarrow rs1 + rs2$

- **SUB rd, rs1, rs2**
  $rd \leftarrow rs1 - rs2$

- **SLL rd, rs1, rs2**
  $rd \leftarrow rs1 << rs2$

- **SLT rd, rs1, rs2**
  $if\ (rs1 < rs2)\ then\ rd \leftarrow 1;\ else\ rd \leftarrow 0$
  $rs1$ and $rs2$ are signed.



imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
0000000	shamt	rs1	001	rd	0010011	SLLI
0000000	shamt	rs1	101	rd	0010011	SRLI
0100000	shamt	rs1	101	rd	0010011	SRAI
0000000	rs2	rs1	000	rd	0110011	ADD
0100000	rs2	rs1	000	rd	0110011	SUB
0000000	rs2	rs1	001	rd	0110011	SLL
0000000	rs2	rs1	010	rd	0110011	SLT
0000000	rs2	rs1	011	rd	0110011	SLTU
0000000	rs2	rs1	100	rd	0110011	XOR
0000000	rs2	rs1	101	rd	0110011	SRL
0100000	rs2	rs1	101	rd	0110011	SRA
0000000	rs2	rs1	110	rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND

Table A.1: Supported instruction set and relative decoding.

- **SLTU rd, rs1, rs2**
  $if\ (rs1 < rs2)\ then\ rd \leftarrow 1;\ else\ rd \leftarrow 0$
  $rs1$ and $rs2$ are unsigned.

- **XOR rd, rs1, rs2**
  $rd \leftarrow rs1 \oplus rs2$

- **SRL rd, rs1, rs2**
  $rd \leftarrow rs1 >> rs2$
  The incoming bits are filled with zeros.

- **SRA rd, rs1, rs2**
  $rd \leftarrow rs1 >> rs2$
  The incoming bits are filled with the MSB of $rs1$.

- **OR rd, rs1, rs2**
  $rd \leftarrow rs1 \vee rs2$

- **AND rd, rs1, rs2**
  $rd \leftarrow rs1 \wedge rs2$

Immediate operations

- **ADDI rd, rs, imm**
  $rd \leftarrow rs + imm$

- **SLLI rd, rs, imm**
  $rd \leftarrow rs << imm$

- **SLTI rd, rs, imm**
  $if\ (rs < imm)\ then\ rd \leftarrow 1;\ else\ rd \leftarrow 0$
  $rs1$ and $rs2$ are signed.

- **SLTIU rd, rs, imm**
  $if\ (rs < imm)\ then\ rd \leftarrow 1;\ else\ rd \leftarrow 0$
  $rs1$ and $rs2$ are unsigned.

- **XORI rd, rs, imm**
  $rd \leftarrow rs \oplus imm$

- **SRLI rd, rs, imm**
  $rd \leftarrow rs >> imm$

The incoming bits are filled with zeros.

- **SRAI rd, rs, imm**
  $rd \leftarrow rs >> imm$
  The incoming bits are filled with the MSB of $rs1$.

- **ORI rd, rs, imm**
  $rd \leftarrow rs \vee imm$

- **ANDI rd, rs, imm**
  $rd \leftarrow rs \wedge imm$

Load

- **LB rd, imm(rs)**
  $rd \leftarrow M[rs + imm]$
  The three leftmost bytes of rd are filled with the MSB of $M[rs + imm]$.

- **LH rd, imm(rs)**
  $rd \leftarrow M[rs + imm] + M[rs + imm + 1] << 8$
  The two leftmost bytes of rd are filled with the MSB of $M[rs + imm + 1]$.

- **LW rd, imm(rs)**
  $rd \leftarrow M[rs+imm]+(M[rs+imm+1] << 8)+(M[rs+imm+2] << 16)+(M[rs+imm+3] << 24)$

- **LBU rd, imm(rs)**
  $rd \leftarrow M[rs + imm]$
  The three leftmost bytes of rd are filled with zeros.

- **LHU rd, imm(rs)**
  $rd \leftarrow M[rs + imm] \vee M[rs + imm + 1] << 8$
  The two leftmost bytes of rd are filled with zeros.

Store

- **SB rs2, imm(rs1)**
  $M[rs1 + imm] \leftarrow\ rs2(7 : 0)$

- **SH rs2, imm(rs1)**
  $M[rs1 + imm] \leftarrow\ rs2(7 : 0)$
  $M[rs1 + imm + 1] \leftarrow\ rs2(15 : 8)$

- **SW rs2, imm(rs1)**
  $M[rs1 + imm] \leftarrow\ rs2(7:0)$
  $M[rs1 + imm + 1] \leftarrow\ rs2(15:8)$
  $M[rs1 + imm + 2] \leftarrow\ rs2(23:16)$
  $M[rs1 + imm + 3] \leftarrow\ rs2(31:24)$

Branch

- **BEQ rs1, rs2, imm**
  $if\ (rs1 = rs2)\ then\ pc \leftarrow pc + imm$

- **BNE rs1, rs2, imm**
  $if\ (rs1 \neq rs2)\ then\ pc \leftarrow pc + imm$

- **BLT rs1, rs2, imm**
  $if\ (rs1 < rs2)\ then\ pc \leftarrow pc + imm$
  $rs1$ and $rs2$ are signed.

- **BGE rs1, rs2, imm**
  $if\ (rs1 \geq rs2)\ then\ pc \leftarrow pc + imm$
  $rs1$ and $rs2$ are signed.

- **BLTU rs1, rs2, imm**
  $if\ (rs1 < rs2)\ then\ pc \leftarrow pc + imm$
  $rs1$ and $rs2$ are unsigned.

- **BGEU rs1, rs2, imm**
  $if\ (rs1 \geq rs2)\ then\ pc \leftarrow pc + imm$
  $rs1$ and $rs2$ are unsigned.

Jump

- **JAL rd, imm**
  $rd \leftarrow pc + 4$
  $pc \leftarrow pc + imm$

- **JALR rd, rs, imm**
  $rd \leftarrow pc + 4$
  $pc \leftarrow rs + imm$

Upper immediate operations

- **LUI rd, imm**
  $rd \leftarrow imm << 12$

- **AUIPC rd, imm**
  $rd \leftarrow pc + (imm << 12)$

APPENDIX B

Sources

Listing B.1: my_package.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use ieee.math_real.all;

package my_package is

    type std_logic_vector_intrange is array (integer range <>) of std_logic;    --standard logic vector with integer range

    --function that receives an integer greater or equal to 1 and returns its base-2 logarithm rounded up to the nearest greater (or
         ↪ equal) integer
    function log2_ceiling (N: positive) return natural;

    --operator xor between a std_logic_vector and a signle std_logic
    function "xor" (vector: std_logic_vector; one_bit: std_logic) return std_logic_vector;

    --nor all bits of a generic width std_logic_vector
    function nor_vector (vector: std_logic_vector) return std_logic;

end package my_package;

package body my_package is

    function log2_ceiling (N: positive) return natural is
    begin
        return natural(ceil(log2(real(N))));
    end function log2_ceiling;

    function "xor" (vector: std_logic_vector; one_bit: std_logic) return std_logic_vector is
        variable temp: std_logic_vector(vector'range);
    begin
        for i in vector'range loop
            temp(i):=vector(i) xor one_bit;
        end loop;
        return temp;
    end function "xor";

    function nor_vector (vector: std_logic_vector) return std_logic is
        variable temp: std_logic:='0';
    begin
        for i in vector'range loop
            temp:= temp or vector(i);
        end loop;
        return not temp;
    end function nor_vector;

end package body my_package;
```

Listing B.2: RISCV_package.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

package RISCV_package is

    constant BYTE_SIZE : integer := 8;
    constant HALFWORD_SIZE : integer := 2*BYTE_SIZE;
    constant WORD_SIZE : integer := 4*BYTE_SIZE;
    constant DOUBLEWORD_SIZE : integer := 8*BYTE_SIZE;
    subtype BYTE_range is integer range BYTE_SIZE-1 downto 0;
    subtype BYTE1_range is integer range 2*BYTE_SIZE-1 downto BYTE_SIZE;
    subtype BYTE2_range is integer range 3*BYTE_SIZE-1 downto 2*BYTE_SIZE;
    subtype BYTE3_range is integer range 4*BYTE_SIZE-1 downto 3*BYTE_SIZE;
    subtype HALFWORD_range is integer range HALFWORD_SIZE-1 downto 0;
    subtype HALFWORD1_range is integer range 2*HALFWORD_SIZE-1 downto HALFWORD_SIZE;
    subtype WORD_range is integer range WORD_SIZE-1 downto 0;
    subtype DOUBLEWORD_range is integer range DOUBLEWORD_SIZE-1 downto 0;
    subtype byte is std_logic_vector(BYTE_range);
    subtype halfword is std_logic_vector(HALFWORD_range);
    subtype word is std_logic_vector(WORD_range);
    subtype doubleword is std_logic_vector(DOUBLEWORD_range);

    constant IF_STAGE   : integer := 0;
    constant ID_STAGE   : integer := 1;
    constant EXE_STAGE  : integer := 2;
    constant MEM_STAGE  : integer := 3;
    constant WB_STAGE   : integer := 4;
    constant N_STAGE    : integer := 5;

    subtype OPCODE_range    is integer range 6 downto 0;
    subtype RD_range        is integer range 11 downto 7;
    subtype FUNCT3_range    is integer range 14 downto 12;
    subtype RS1_range       is integer range 19 downto 15;
    subtype RS2_range       is integer range 24 downto 20;
    subtype FUNCT7_range    is integer range 31 downto 25;
    subtype IMM_LONG_range  is integer range 31 downto 12;

    constant OPCODE_SIZE    : integer := OPCODE_range'high-OPCODE_range'low+1;          --OPCODE field size
    constant REG_ADDR_SIZE  : integer := RD_range'high-RD_range'low+1;                  --register file address field size
    constant FUNCT3_SIZE    : integer := FUNCT3_range'high-FUNCT3_range'low+1;          --FUNC field size
    constant FUNCT7_SIZE    : integer := FUNCT7_range'high-FUNCT7_range'low+1;          --OFFSET field size
    constant IMM_SHORT_SIZE : integer := FUNCT7_SIZE+REG_ADDR_SIZE;                     --IMMEDIATE size in I-type, S-type, SB-type
            ↪ instructions
    constant IMM_LONG_SIZE  : integer := FUNCT7_SIZE+2*REG_ADDR_SIZE+FUNCT3_SIZE;       --IMMEDIATE size in UJ-type, U-type,
            ↪ instructions

    constant N_REG      : integer := 2**REG_ADDR_SIZE;  --number of registers
    constant N_FUNCT3   : integer := 2**FUNCT3_SIZE;    --number of possible FUNCT3s
    constant N_FUNCT7   : integer := 2**FUNCT7_SIZE;    --number of possible FUNCT7s
    constant N_OPCODE   : integer := 2**OPCODE_SIZE;    --number of possible OPCODEs

    subtype REG_addr is std_logic_vector(REG_ADDR_SIZE-1 downto 0);
    subtype FUNCT7_type is std_logic_vector(FUNCT7_SIZE-1 downto 0);

    --all possible OPCODES in order
    type OPCODE_type is (
        OPCODE_LOAD,
        OPCODE_LOADFP,
        OPCODE_CUSTOM0,
        OPCODE_MISCMEM,
        OPCODE_OPIMM,
        OPCODE_AUIPC,
        OPCODE_OPIMM32,
        OPCODE_WASTE0,
        OPCODE_STORE,
        OPCODE_STOREFP,
        OPCODE_CUSTOM1,
        OPCODE_AMO,
        OPCODE_OP,
        OPCODE_LUI,
        OPCODE_OP32,
        OPCODE_WASTE1,
        OPCODE_MADD,
        OPCODE_MSUB,
        OPCODE_NMSUB,
        OPCODE_NMADD,
        OPCODE_OPFP,
        OPCODE_RESERVED0,
        OPCODE_CUSTOM2,
        OPCODE_WASTE2,
        OPCODE_BRANCH,
        OPCODE_JALR,
        OPCODE_RESERVED1,
        OPCODE_JAL,
        OPCODE_SYSTEM,
        OPCODE_RESERVED2,
        OPCODE_CUSTOM3,
        OPCODE_WASTE3
    );

    --all possible FUNCT3 when OPCODE=OPCODE_BRANCH in order
    type FUNCT3_BRANCH_type is (
        FUNCT3_BEQ,
        FUNCT3_BNE,
        FUNCT3_WASTE0,
        FUNCT3_WASTE1,
        FUNCT3_BLT,
        FUNCT3_BGE,
        FUNCT3_BLTU,
        FUNCT3_BGEU
    );

    --all possible FUNCT3 when OPCODE=OPCODE_LOAD in order
```

```
type FUNCT3_LOAD_type is (
    FUNCT3_LB,
    FUNCT3_LH,
    FUNCT3_LW,
    FUNCT3_WASTE0,
    FUNCT3_LBU,
    FUNCT3_LHU,
    FUNCT3_WASTE1,
    FUNCT3_WASTE2
);

--all possible FUNCT3 when OPCODE=OPCODE_STORE in order
type FUNCT3_STORE_type is (
    FUNCT3_SB,
    FUNCT3_SH,
    FUNCT3_SW,
    FUNCT3_WASTE0,
    FUNCT3_WASTE1,
    FUNCT3_WASTE2,
    FUNCT3_WASTE3,
    FUNCT3_WASTE4
);

--all possible FUNCT3 when OPCODE=OPCODE_OPIMM in order
type FUNCT3_OPIMM_type is (
    FUNCT3_ADDI,
    FUNCT3_SLI,
    FUNCT3_SLTI,
    FUNCT3_SLTIU,
    FUNCT3_XORI,
    FUNCT3_SRI,     --logical or arithmetical according to bit 10 of immediate field (bit 5 of funct7 field)
    FUNCT3_ORI,
    FUNCT3_ANDI
);

constant FUNCT7_STD0    : FUNCT7_type := "0000000";
constant FUNCT7_STD1    : FUNCT7_type := "0100000";
constant FUNCT7_RV32M   : FUNCT7_type := "0000001";

--all possible FUNCT3 when OPCODE=OPCODE_OP and FUNCT7=FUNCT7_STD0 in order
type FUNCT3_OP_STD0_type is (
    FUNCT3_ADD,
    FUNCT3_SL,
    FUNCT3_SLT,
    FUNCT3_SLTU,
    FUNCT3_XOR,
    FUNCT3_SRL,
    FUNCT3_OR,
    FUNCT3_AND
);

--all possible FUNCT3 when OPCODE=OPCODE_OP and FUNCT7=FUNCT7_STD1 in order
type FUNCT3_OP_STD1_type is (
    FUNCT3_SUB,
    FUNCT3_WASTE0,
    FUNCT3_WASTE1,
    FUNCT3_WASTE2,
    FUNCT3_WASTE3,
    FUNCT3_SRA,
    FUNCT3_WASTE4,
    FUNCT3_WASTE5
);

--all possible FUNCT3 when OPCODE=OPCODE_OP and FUNCT7=FUNCT7_RV32M in order
type FUNCT3_OP_RV32M_type is (
    FUNCT3_MUL,
    FUNCT3_MULH,
    FUNCT3_MULHSU,
    FUNCT3_MULHU,
    FUNCT3_DIV,
    FUNCT3_DIVU,
    FUNCT3_REM,
    FUNCT3_REMU
);

--type conversion functions for FUNCT3_BRANCH_type
function to_integer (FUNCT3: FUNCT3_BRANCH_type) return natural;
function to_std_logic_vector (FUNCT3: FUNCT3_BRANCH_type) return std_logic_vector;
function to_FUNCT3_BRANCH_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_BRANCH_type;
function to_FUNCT3_BRANCH_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_BRANCH_type;

--type conversion functions for FUNCT3_LOAD_type
function to_integer (FUNCT3: FUNCT3_LOAD_type) return natural;
function to_std_logic_vector (FUNCT3: FUNCT3_LOAD_type) return std_logic_vector;
function to_FUNCT3_LOAD_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_LOAD_type;
function to_FUNCT3_LOAD_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_LOAD_type;

--type conversion functions for FUNCT3_STORE_type
function to_integer (FUNCT3: FUNCT3_STORE_type) return natural;
function to_std_logic_vector (FUNCT3: FUNCT3_STORE_type) return std_logic_vector;
function to_FUNCT3_STORE_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_STORE_type;
function to_FUNCT3_STORE_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_STORE_type;

--type conversion functions for FUNCT3_OPIMM_type
function to_integer (FUNCT3: FUNCT3_OPIMM_type) return natural;
function to_std_logic_vector (FUNCT3: FUNCT3_OPIMM_type) return std_logic_vector;
```

```
    function to_FUNCT3_OPIMM_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_OPIMM_type;
    function to_FUNCT3_OPIMM_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_OPIMM_type;

    --type conversion functions for FUNCT3_OP_STD0_type
    function to_integer (FUNCT3: FUNCT3_OP_STD0_type) return natural;
    function to_std_logic_vector (FUNCT3: FUNCT3_OP_STD0_type) return std_logic_vector;
    function to_FUNCT3_OP_STD0_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_OP_STD0_type;
    function to_FUNCT3_OP_STD0_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_OP_STD0_type;

    --type conversion functions for FUNCT3_OP_STD1_type
    function to_integer (FUNCT3: FUNCT3_OP_STD1_type) return natural;
    function to_std_logic_vector (FUNCT3: FUNCT3_OP_STD1_type) return std_logic_vector;
    function to_FUNCT3_OP_STD1_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_OP_STD1_type;
    function to_FUNCT3_OP_STD1_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_OP_STD1_type;

    --type conversion functions for FUNCT3_OP_RV32M_type
    function to_integer (FUNCT3: FUNCT3_OP_RV32M_type) return natural;
    function to_std_logic_vector (FUNCT3: FUNCT3_OP_RV32M_type) return std_logic_vector;
    function to_FUNCT3_OP_RV32M_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_OP_RV32M_type;
    function to_FUNCT3_OP_RV32M_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_OP_RV32M_type;

    --type conversion functions for OPCODE_type
    function to_integer (OPCODE: OPCODE_type) return natural;
    function to_std_logic_vector (OPCODE: OPCODE_type) return std_logic_vector;
    function to_OPCODE_type (OPCODE_N: integer range 0 to N_OPCODE-1) return OPCODE_type;
    function to_OPCODE_type (OPCODE_FIELD: std_logic_vector(OPCODE_SIZE-1 downto 0)) return OPCODE_type;

    --functions to get directly opcode, funct3 or registers number from an instruction
    function get_OPCODE (INSTR: word) return OPCODE_type;
    function get_FUNCT3_BRANCH (INSTR: word) return FUNCT3_BRANCH_type;
    function get_FUNCT3_LOAD (INSTR: word) return FUNCT3_LOAD_type;
    function get_FUNCT3_STORE (INSTR: word) return FUNCT3_STORE_type;
    function get_FUNCT3_OPIMM (INSTR: word) return FUNCT3_OPIMM_type;
    function get_FUNCT3_OP_STD0 (INSTR: word) return FUNCT3_OP_STD0_type;
    function get_FUNCT3_OP_STD1 (INSTR: word) return FUNCT3_OP_STD1_type;
    function get_FUNCT3_OP_RV32M (INSTR: word) return FUNCT3_OP_RV32M_type;
    function get_FUNCT7 (INSTR: word) return FUNCT7_type;
    function get_RD (INSTR: word) return natural;
    function get_RS1 (INSTR: word) return natural;
    function get_RS2 (INSTR: word) return natural;
    function get_IMM_ITYPE (INSTR: word) return integer;
    function get_IMM_STYPE (INSTR: word) return integer;
    function get_IMM_BTYPE (INSTR: word) return integer;
    function get_IMM_JTYPE (INSTR: word) return integer;
    function get_IMM_UTYPE (INSTR: word) return integer;

    --instruction encoding functions
    function INSTR_BRANCH (FUNCT3: FUNCT3_BRANCH_type; RS1,RS2: integer range 0 to N_REG; IMM: integer range -2**IMM_SHORT_SIZE to 2**
        ↪ IMM_SHORT_SIZE-1) return word;
    function INSTR_LOAD (FUNCT3: FUNCT3_LOAD_type; RD,RS: integer range 0 to N_REG; IMM: integer range -2**(IMM_SHORT_SIZE-1) to 2**(
        ↪ IMM_SHORT_SIZE-1)-1) return word;
    function INSTR_STORE (FUNCT3: FUNCT3_STORE_type; RS1,RS2: integer range 0 to N_REG; IMM: integer range -2**(IMM_SHORT_SIZE-1) to
        ↪ 2**(IMM_SHORT_SIZE-1)-1) return word;
    function INSTR_OPIMM (FUNCT3: FUNCT3_OPIMM_type; RD,RS: integer range 0 to N_REG; IMM: integer range -2**(IMM_SHORT_SIZE-1) to 2**(
        ↪ IMM_SHORT_SIZE-1)-1) return word;
    function INSTR_OP_STD0 (FUNCT3: FUNCT3_OP_STD0_type; RD,RS1,RS2: integer range 0 to N_REG) return word;
    function INSTR_OP_STD1 (FUNCT3: FUNCT3_OP_STD1_type; RD,RS1,RS2: integer range 0 to N_REG) return word;
    function INSTR_OP_RV32M (FUNCT3: FUNCT3_OP_RV32M_type; RD,RS1,RS2: integer range 0 to N_REG) return word;
    function INSTR_JAL (RD: integer range 0 to N_REG; IMM: integer range -2**IMM_LONG_SIZE to 2**IMM_LONG_SIZE-1) return word;
    function INSTR_JALR (RD,RS: integer range 0 to N_REG; IMM: integer range -2**(IMM_SHORT_SIZE-1) to 2**(IMM_SHORT_SIZE-1)-1) return
        ↪ word;
    function INSTR_UTYPE (OPCODE: OPCODE_type; RD: integer range 0 to N_REG; IMM: integer range -2**(IMM_LONG_SIZE-1) to 2**(
        ↪ IMM_LONG_SIZE-1)-1) return word;

    --instruction type recognition functions
    function is_a_jump (INSTR: word) return boolean;
    function is_a_branch (INSTR: word) return boolean;
    function is_a_store (INSTR: word) return boolean;
    function is_a_load (INSTR: word) return boolean;
    function is_a_mult (INSTR: word) return boolean;
    function is_a_op (INSTR: word) return boolean;
    function is_a_upperimm (INSTR: word) return boolean;

    constant INSTR_NOP: word;

end package RISCV_package;

package body RISCV_package is

    function to_integer (FUNCT3: FUNCT3_BRANCH_type) return natural is
    begin
        return FUNCT3_BRANCH_type'pos(FUNCT3);
    end function to_integer;

    function to_std_logic_vector (FUNCT3: FUNCT3_BRANCH_type) return std_logic_vector is
    begin
        return std_logic_vector(to_unsigned(to_integer(FUNCT3), FUNCT3_SIZE));
    end function to_std_logic_vector;

    function to_FUNCT3_BRANCH_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_BRANCH_type is
    begin
        return FUNCT3_BRANCH_type'val(FUNCT3_N);
    end function to_FUNCT3_BRANCH_type;

    function to_FUNCT3_BRANCH_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_BRANCH_type is
    begin
```

```
        return to_FUNCT3_BRANCH_type(to_integer(unsigned(FUNCT3_FIELD)));
    end function to_FUNCT3_BRANCH_type;

    function to_integer (FUNCT3: FUNCT3_LOAD_type) return natural is
    begin
        return FUNCT3_LOAD_type'pos(FUNCT3);
    end function to_integer;

    function to_std_logic_vector (FUNCT3: FUNCT3_LOAD_type) return std_logic_vector is
    begin
        return std_logic_vector(to_unsigned(to_integer(FUNCT3), FUNCT3_SIZE));
    end function to_std_logic_vector;

    function to_FUNCT3_LOAD_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_LOAD_type is
    begin
        return FUNCT3_LOAD_type'val(FUNCT3_N);
    end function to_FUNCT3_LOAD_type;

    function to_FUNCT3_LOAD_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_LOAD_type is
    begin
        return to_FUNCT3_LOAD_type(to_integer(unsigned(FUNCT3_FIELD)));
    end function to_FUNCT3_LOAD_type;

    function to_integer (FUNCT3: FUNCT3_STORE_type) return natural is
    begin
        return FUNCT3_STORE_type'pos(FUNCT3);
    end function to_integer;

    function to_std_logic_vector (FUNCT3: FUNCT3_STORE_type) return std_logic_vector is
    begin
        return std_logic_vector(to_unsigned(to_integer(FUNCT3), FUNCT3_SIZE));
    end function to_std_logic_vector;

    function to_FUNCT3_STORE_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_STORE_type is
    begin
        return FUNCT3_STORE_type'val(FUNCT3_N);
    end function to_FUNCT3_STORE_type;

    function to_FUNCT3_STORE_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_STORE_type is
    begin
        return to_FUNCT3_STORE_type(to_integer(unsigned(FUNCT3_FIELD)));
    end function to_FUNCT3_STORE_type;

    function to_integer (FUNCT3: FUNCT3_OPIMM_type) return natural is
    begin
        return FUNCT3_OPIMM_type'pos(FUNCT3);
    end function to_integer;

    function to_std_logic_vector (FUNCT3: FUNCT3_OPIMM_type) return std_logic_vector is
    begin
        return std_logic_vector(to_unsigned(to_integer(FUNCT3), FUNCT3_SIZE));
    end function to_std_logic_vector;

    function to_FUNCT3_OPIMM_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_OPIMM_type is
    begin
        return FUNCT3_OPIMM_type'val(FUNCT3_N);
    end function to_FUNCT3_OPIMM_type;

    function to_FUNCT3_OPIMM_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_OPIMM_type is
    begin
        return to_FUNCT3_OPIMM_type(to_integer(unsigned(FUNCT3_FIELD)));
    end function to_FUNCT3_OPIMM_type;

    function to_integer (FUNCT3: FUNCT3_OP_STD0_type) return natural is
    begin
        return FUNCT3_OP_STD0_type'pos(FUNCT3);
    end function to_integer;

    function to_std_logic_vector (FUNCT3: FUNCT3_OP_STD0_type) return std_logic_vector is
    begin
        return std_logic_vector(to_unsigned(to_integer(FUNCT3), FUNCT3_SIZE));
    end function to_std_logic_vector;

    function to_FUNCT3_OP_STD0_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_OP_STD0_type is
    begin
        return FUNCT3_OP_STD0_type'val(FUNCT3_N);
    end function to_FUNCT3_OP_STD0_type;

    function to_FUNCT3_OP_STD0_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_OP_STD0_type is
    begin
        return to_FUNCT3_OP_STD0_type(to_integer(unsigned(FUNCT3_FIELD)));
    end function to_FUNCT3_OP_STD0_type;

    function to_integer (FUNCT3: FUNCT3_OP_STD1_type) return natural is
    begin
        return FUNCT3_OP_STD1_type'pos(FUNCT3);
    end function to_integer;

    function to_std_logic_vector (FUNCT3: FUNCT3_OP_STD1_type) return std_logic_vector is
    begin
        return std_logic_vector(to_unsigned(to_integer(FUNCT3), FUNCT3_SIZE));
    end function to_std_logic_vector;

    function to_FUNCT3_OP_STD1_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_OP_STD1_type is
    begin
        return FUNCT3_OP_STD1_type'val(FUNCT3_N);
```

```
end function to_FUNCT3_OP_STD1_type;

function to_FUNCT3_OP_STD1_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_OP_STD1_type is
begin
    return to_FUNCT3_OP_STD1_type(to_integer(unsigned(FUNCT3_FIELD)));
end function to_FUNCT3_OP_STD1_type;

function to_integer (FUNCT3: FUNCT3_OP_RV32M_type) return natural is
begin
    return FUNCT3_OP_RV32M_type'pos(FUNCT3);
end function to_integer;

function to_std_logic_vector (FUNCT3: FUNCT3_OP_RV32M_type) return std_logic_vector is
begin
    return std_logic_vector(to_unsigned(to_integer(FUNCT3), FUNCT3_SIZE));
end function to_std_logic_vector;

function to_FUNCT3_OP_RV32M_type (FUNCT3_N: integer range 0 to N_FUNCT3-1) return FUNCT3_OP_RV32M_type is
begin
    return FUNCT3_OP_RV32M_type'val(FUNCT3_N);
end function to_FUNCT3_OP_RV32M_type;

function to_FUNCT3_OP_RV32M_type (FUNCT3_FIELD: std_logic_vector(FUNCT3_SIZE-1 downto 0)) return FUNCT3_OP_RV32M_type is
begin
    return to_FUNCT3_OP_RV32M_type(to_integer(unsigned(FUNCT3_FIELD)));
end function to_FUNCT3_OP_RV32M_type;

function to_integer (OPCODE: OPCODE_type) return natural is
begin
    return OPCODE_type'pos(OPCODE)*4+3;
end function to_integer;

function to_std_logic_vector (OPCODE: OPCODE_type) return std_logic_vector is
begin
    return std_logic_vector(to_unsigned(to_integer(OPCODE), OPCODE_SIZE));
end function to_std_logic_vector;

function to_opcode_type (OPCODE_N: integer range 0 to N_OPCODE-1) return OPCODE_type is
begin
        return OPCODE_type'val(OPCODE_N/4);
end function to_opcode_type;

function to_opcode_type (OPCODE_FIELD: std_logic_vector(OPCODE_SIZE-1 downto 0)) return OPCODE_type is
begin
    return to_opcode_type(to_integer(unsigned(OPCODE_FIELD)));
end function to_opcode_type;

function get_opcode (INSTR: word) return OPCODE_type is
begin
    return to_opcode_type(INSTR(OPCODE_range));
end function get_opcode;

function get_FUNCT3_BRANCH (INSTR: word) return FUNCT3_BRANCH_type is
begin
    return to_FUNCT3_BRANCH_type(INSTR(FUNCT3_range));
end function get_FUNCT3_BRANCH;

function get_FUNCT3_LOAD (INSTR: word) return FUNCT3_LOAD_type is
begin
    return to_FUNCT3_LOAD_type(INSTR(FUNCT3_range));
end function get_FUNCT3_LOAD;

function get_FUNCT3_STORE (INSTR: word) return FUNCT3_STORE_type is
begin
    return to_FUNCT3_STORE_type(INSTR(FUNCT3_range));
end function get_FUNCT3_STORE;

function get_FUNCT3_OPIMM (INSTR: word) return FUNCT3_OPIMM_type is
begin
    return to_FUNCT3_OPIMM_type(INSTR(FUNCT3_range));
end function get_FUNCT3_OPIMM;

function get_FUNCT3_OP_STD0 (INSTR: word) return FUNCT3_OP_STD0_type is
begin
    return to_FUNCT3_OP_STD0_type(INSTR(FUNCT3_range));
end function get_FUNCT3_OP_STD0;

function get_FUNCT3_OP_STD1 (INSTR: word) return FUNCT3_OP_STD1_type is
begin
    return to_FUNCT3_OP_STD1_type(INSTR(FUNCT3_range));
end function get_FUNCT3_OP_STD1;

function get_FUNCT3_OP_RV32M (INSTR: word) return FUNCT3_OP_RV32M_type is
begin
    return to_FUNCT3_OP_RV32M_type(INSTR(FUNCT3_range));
end function get_FUNCT3_OP_RV32M;

function get_FUNCT7 (INSTR: word) return FUNCT7_type is
variable temp: FUNCT7_type;
begin
    temp:=INSTR(FUNCT7_range);
    return temp;
end function get_FUNCT7;

function get_RS1 (INSTR: word) return natural is
begin
```

```
    return to_integer(unsigned(INSTR(RS1_range)));
end function get_RS1;

function get_RS2 (INSTR: word) return natural is
begin
    return to_integer(unsigned(INSTR(RS2_range)));
end function get_RS2;

function get_RD (INSTR: word) return natural is
begin
    return to_integer(unsigned(INSTR(RD_range)));
end function get_RD;

function get_IMM_ITYPE (INSTR: word) return integer is
begin
    return to_integer(signed(INSTR(FUNCT7_range) & INSTR(RS2_range)));
end function get_IMM_ITYPE;

function get_IMM_STYPE (INSTR: word) return integer is
begin
    return to_integer(signed(INSTR(FUNCT7_range) & INSTR(RD_range)));
end function get_IMM_STYPE;

function get_IMM_BTYPE (INSTR: word) return integer is
    variable FUNCT7_field : FUNCT7_type := INSTR(FUNCT7_range);
    variable RD_field : REG_addr := INSTR(RD_range);
begin
    return to_integer(signed(FUNCT7_field(6) & RD_field(0) & FUNCT7_field(5 downto 0) & RD_field(4 downto 1) & '0'));
end function get_IMM_BTYPE;

function get_IMM_JTYPE (INSTR: word) return integer is
    variable IMM_field : std_logic_vector(IMM_LONG_SIZE-1 downto 0) := INSTR(IMM_LONG_range);
begin
    return to_integer(signed(IMM_field(19) & IMM_field(7 downto 0) & IMM_field(8) & IMM_field(18 downto 9) & '0'));
end function get_IMM_JTYPE;

function get_IMM_UTYPE (INSTR: word) return integer is
begin
    return to_integer(signed(INSTR(IMM_LONG_range)) & to_signed(0, 12));
end function get_IMM_UTYPE;

function INSTR_BRANCH (FUNCT3: FUNCT3_BRANCH_type; RS1,RS2: integer range 0 to N_REG; IMM: integer range -2**IMM_SHORT_SIZE to 2**
      ↪ IMM_SHORT_SIZE-1) return word is
    constant IMM_std_logic_vector : std_logic_vector(IMM_SHORT_SIZE downto 0) := std_logic_vector(to_signed(IMM, IMM_SHORT_SIZE+1));
    constant temp: word:= IMM_std_logic_vector(12) & IMM_std_logic_vector(10 downto 5) & std_logic_vector(to_unsigned(RS2,
          ↪ REG_ADDR_SIZE)) & std_logic_vector(to_unsigned(RS1,REG_ADDR_SIZE)) & to_std_logic_vector(FUNCT3) &
          ↪ IMM_std_logic_vector(4 downto 1) & IMM_std_logic_vector(11) & to_std_logic_vector(OPCODE_BRANCH);
begin
    return temp;
end function INSTR_BRANCH;

function INSTR_LOAD (FUNCT3: FUNCT3_LOAD_type; RD,RS: integer range 0 to N_REG; IMM: integer range -2**(IMM_SHORT_SIZE-1) to 2**(
      ↪ IMM_SHORT_SIZE-1)-1) return word is
    constant temp: word:= std_logic_vector(to_signed(IMM, IMM_SHORT_SIZE)) & std_logic_vector(to_unsigned(RS,REG_ADDR_SIZE)) &
          ↪ to_std_logic_vector(FUNCT3) & std_logic_vector(to_unsigned(RD,REG_ADDR_SIZE)) & to_std_logic_vector(OPCODE_LOAD);
begin
    return temp;
end function INSTR_LOAD;

function INSTR_STORE (FUNCT3: FUNCT3_STORE_type; RS1,RS2: integer range 0 to N_REG; IMM: integer range -2**(IMM_SHORT_SIZE-1) to
      ↪ 2**(IMM_SHORT_SIZE-1)-1) return word is
    constant IMM_std_logic_vector : std_logic_vector(IMM_SHORT_SIZE-1 downto 0) := std_logic_vector(to_signed(IMM, IMM_SHORT_SIZE));
    constant temp: word:= IMM_std_logic_vector(11 downto 5) & std_logic_vector(to_unsigned(RS2,REG_ADDR_SIZE)) & std_logic_vector(
          ↪ to_unsigned(RS1,REG_ADDR_SIZE)) & to_std_logic_vector(FUNCT3) & IMM_std_logic_vector(4 downto 0) & to_std_logic_vector
          ↪ (OPCODE_STORE);
begin
    return temp;
end function INSTR_STORE;

function INSTR_OPIMM (FUNCT3: FUNCT3_OPIMM_type; RD,RS: integer range 0 to N_REG; IMM: integer range -2**(IMM_SHORT_SIZE-1) to 2**(
      ↪ IMM_SHORT_SIZE-1)-1) return word is
    constant temp: word:= std_logic_vector(to_signed(IMM, IMM_SHORT_SIZE)) & std_logic_vector(to_unsigned(RS,REG_ADDR_SIZE)) &
          ↪ to_std_logic_vector(FUNCT3) & std_logic_vector(to_unsigned(RD,REG_ADDR_SIZE)) & to_std_logic_vector(OPCODE_OPIMM);
begin
    return temp;
end function INSTR_OPIMM;

function INSTR_OP_STD0 (FUNCT3: FUNCT3_OP_STD0_type; RD,RS1,RS2: integer range 0 to N_REG) return word is
    constant temp: word := FUNCT7_STD0 & std_logic_vector(to_unsigned(RS2,REG_ADDR_SIZE)) & std_logic_vector(to_unsigned(RS1,
          ↪ REG_ADDR_SIZE)) & to_std_logic_vector(FUNCT3) & std_logic_vector(to_unsigned(RD,REG_ADDR_SIZE)) & to_std_logic_vector(
          ↪ OPCODE_OP);
begin
    return temp;
end function INSTR_OP_STD0;

function INSTR_OP_STD1 (FUNCT3: FUNCT3_OP_STD1_type; RD,RS1,RS2: integer range 0 to N_REG) return word is
    constant temp: word := FUNCT7_STD1 & std_logic_vector(to_unsigned(RS2,REG_ADDR_SIZE)) & std_logic_vector(to_unsigned(RS1,
          ↪ REG_ADDR_SIZE)) & to_std_logic_vector(FUNCT3) & std_logic_vector(to_unsigned(RD,REG_ADDR_SIZE)) & to_std_logic_vector(
          ↪ OPCODE_OP);
begin
    return temp;
end function INSTR_OP_STD1;

function INSTR_OP_RV32M (FUNCT3: FUNCT3_OP_RV32M_type; RD,RS1,RS2: integer range 0 to N_REG) return word is
begin
    return FUNCT7_RV32M & std_logic_vector(to_unsigned(RS2,REG_ADDR_SIZE)) & std_logic_vector(to_unsigned(RS1,REG_ADDR_SIZE)) &
```

```
            ↪ to_std_logic_vector(FUNCT3) & std_logic_vector(to_unsigned(RD,REG_ADDR_SIZE)) & to_std_logic_vector(OPCODE_OP);
    end function INSTR_OP_RV32M;

    function INSTR_JAL (RD: integer range 0 to N_REG; IMM: integer range -2**IMM_LONG_SIZE to 2**IMM_LONG_SIZE-1) return word is
        constant IMM_std_logic_vector : std_logic_vector(IMM_LONG_SIZE downto 0) := std_logic_vector(to_unsigned(IMM, IMM_LONG_SIZE+1));
        constant temp: word :=  IMM_std_logic_vector(20) & IMM_std_logic_vector(10 downto 1) & IMM_std_logic_vector(11) &
            ↪ IMM_std_logic_vector(19 downto 12) & std_logic_vector(to_unsigned(RD,REG_ADDR_SIZE)) & to_std_logic_vector(OPCODE_JAL)
            ↪ ;
    begin
        return temp;
    end function INSTR_JAL;

    function INSTR_JALR (RD,RS: integer range 0 to N_REG; IMM: integer range -2**(IMM_SHORT_SIZE-1) to 2**(IMM_SHORT_SIZE-1)-1) return
        ↪ word is
        constant temp: word:= std_logic_vector(to_signed(IMM, IMM_SHORT_SIZE)) & std_logic_vector(to_unsigned(RS,REG_ADDR_SIZE)) & "000"
            ↪ & std_logic_vector(to_unsigned(RD,REG_ADDR_SIZE)) & to_std_logic_vector(OPCODE_JALR);
    begin
        return temp;
    end function INSTR_JALR;

    function INSTR_UTYPE (OPCODE: OPCODE_type; RD: integer range 0 to N_REG; IMM: integer range -2**(IMM_LONG_SIZE-1) to 2**(
        ↪ IMM_LONG_SIZE-1)-1) return word is
        constant temp: word:= std_logic_vector(to_signed(IMM, IMM_LONG_SIZE)) & std_logic_vector(to_unsigned(RD,REG_ADDR_SIZE)) &
            ↪ to_std_logic_vector(OPCODE);
    begin
        return temp;
    end function INSTR_UTYPE;

    function is_a_jump(instr: word) return boolean is
        variable opcode: OPCODE_type;
    begin
        opcode:=get_opcode(instr);
        return opcode=OPCODE_JAL or opcode=OPCODE_JALR;
    end function is_a_jump;

    function is_a_branch(instr: word) return boolean is
        variable opcode: OPCODE_type;
    begin
        opcode:=get_opcode(instr);
        return opcode=OPCODE_BRANCH;
    end function is_a_branch;

    function is_a_store (INSTR: word) return boolean is
        variable opcode: OPCODE_type;
    begin
        opcode:=get_opcode(INSTR);
        return opcode=OPCODE_STORE;
    end function is_a_store;

    function is_a_load (INSTR: word) return boolean is
        variable opcode: OPCODE_type;
    begin
        opcode:=get_opcode(INSTR);
        return opcode=OPCODE_LOAD;
    end function is_a_load;

    function is_a_mult (INSTR: word) return boolean is
        variable opcode: OPCODE_type;
        variable funct7: FUNCT7_type;
        variable funct3: FUNCT3_OP_RV32M_type;
    begin
        opcode:=get_OPCODE(INSTR);
        funct7:=get_FUNCT7(INSTR);
        funct3:=get_FUNCT3_OP_RV32M(INSTR);
        return opcode=OPCODE_OP and funct7=FUNCT7_RV32M and (funct3=FUNCT3_MUL or funct3=FUNCT3_MULH or funct3=FUNCT3_MULHSU or funct3=
            ↪ FUNCT3_MULHU);
    end function is_a_mult;

    function is_a_op (INSTR: word) return boolean is
        variable opcode: OPCODE_type;
    begin
        opcode:=get_opcode(INSTR);
        return opcode=OPCODE_OP;
    end function is_a_op;

    function is_a_upperimm (INSTR: word) return boolean is
        variable opcode: OPCODE_type;
    begin
        opcode:=get_opcode(INSTR);
        return opcode=OPCODE_LUI or opcode=OPCODE_AUIPC;
    end function is_a_upperimm;

    constant INSTR_NOP: word:= INSTR_OPIMM(FUNCT3_ADDI, 0, 0, 0);

end package body RISCV_package;
```

Listing B.3: ALU_package.vhd

```
use work.my_package.all;

package ALU_package is
```

```
    --all possible ALU operations
    type ALU_OP_type is (
        ALU_DONT_CARE,  --the operation don't care
        ALU_IN1,        --out = first operand
        ALU_IN2,        --out = second operand
        ALU_ADD,        --addition
        ALU_SUB,        --subtraction
        ALU_XOR,        --bitwise xor
        ALU_AND,        --bitwise and
        ALU_OR,         --bitwise or
        ALU_SL,         --shift left
        ALU_SRL,        --shift right logical
        ALU_SRA,        --shift right airthmetical
        ALU_SLT,        --set if less than (signed)
        ALU_SLTU,       --set if less than (unsigned)
        ALU_ABS         --absolute value
        );

    constant N_ALU_OP : integer := ALU_OP_type'pos(ALU_OP_type'high)+1;     --number of possible ALU operations
    constant ALU_OP_SEL_SIZE : integer := log2_ceiling(N_ALU_OP);           --ALU operation selector size

end package ALU_package;
```

Listing B.4: CU_package.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.ALU_package.all;
use work.RISCV_package.all;

package CU_package is

    type BRANCH_COND_type is (
        BRANCH_COND_NO,     --don't jump
        BRANCH_COND_ALWAYS, --jump always
        BRANCH_COND_EQ,     --branch if equal
        BRANCH_COND_NE,     --branch if not equal
        BRANCH_COND_LT,     --branch if less than
        BRANCH_COND_GE,     --branch if greater than or equal to
        BRANCH_COND_LTU,    --branch if less than (unsigned)
        BRANCH_COND_GEU     --branch if greater than or equal to (unsigned)
    );

    type CU_OUTS_ID_type is record
        -- ID outputs
        MUX_NPC_base_sel    : std_logic;                    --select whether the jump/branch target is computed starting from a register
              ↪  or from the PC
        MUX_ALU_IN1_sel     : std_logic_vector(1 downto 0); --select wether the first operand is taken from the register file or if it
              ↪ is the PC+4
        MUX_IMM_sel         : std_logic_vector(2 downto 0); --type of immediate extension
        BRANCH_COND         : BRANCH_COND_type;             --branch condition
    end record CU_OUTS_ID_type;

    type CU_OUTS_EXE_type is record
        -- EXE outputs
        MUX_ALU_IN2_sel     : std_logic;        --select whether the second operand is B or IMM
        ALU_OP              : ALU_OP_type;      --ALU operation selector
    end record CU_OUTS_EXE_type;

    type CU_OUTS_MEM_type is record
        -- MEM outputs
        DRAM_WR_EN          : std_logic_vector(1 downto 0); --data RAM write enable
        MUX_DRAM_sel        : std_logic_vector(2 downto 0); --select size and type of extension of data out from DRAM
        MUX_RF_IN_sel       : std_logic;                    --select wether the data to write in the RF is taken from the DRAM or from
              ↪ the ALU
    end record CU_OUTS_MEM_type;

    type CU_OUTS_WB_type is record
        -- WB outputs
        RF_WR_EN            : std_logic;    --register file write enable
    end record CU_OUTS_WB_type;

    type CU_OUTS_type is record
        ID  : CU_OUTS_ID_type;
        EXE : CU_OUTS_EXE_type;
        MEM : CU_OUTS_MEM_type;
        WB  : CU_OUTS_WB_type;
    end record CU_OUTS_type;

    type LUT_FUNCT3_type is array (0 to N_FUNCT3-1) of CU_OUTS_type;
    type LUT_OPCODE_type is array (0 to N_OPCODE-1) of CU_OUTS_type;

    --ID outputs constants
    constant NPC_BASE_REG       : std_logic:= '1';
    constant NPC_BASE_PC        : std_logic:= '0';
    constant ALU_IN1_PC         : std_logic_vector:= "00";
    constant ALU_IN1_PC_plus4   : std_logic_vector:= "01";
    constant ALU_IN1_RF         : std_logic_vector:= "10";
    constant IMM_ITYPE          : std_logic_vector:= "000";
    constant IMM_STYPE          : std_logic_vector:= "001";
    constant IMM_BTYPE          : std_logic_vector:= "010";
    constant IMM_JTYPE          : std_logic_vector:= "011";
    constant IMM_UTYPE          : std_logic_vector:= "100";
    --EXE outputs constants
    constant OUT_ALU        : std_logic:= '0';
    constant OUT_MULT       : std_logic:= '1';
    constant ALU_IN2_RF     : std_logic:= '0';
    constant ALU_IN2_IMM    : std_logic:= '1';
    --MEM outputs
    constant DRAM_WR_OFF    : std_logic_vector:="00";
    constant DRAM_WR_B      : std_logic_vector:="01";
    constant DRAM_WR_H      : std_logic_vector:="10";
    constant DRAM_WR_W      : std_logic_vector:="11";
    constant MUX_DRAM_SB    : std_logic_vector:= "000";
    constant MUX_DRAM_UB    : std_logic_vector:= "001";
    constant MUX_DRAM_SH    : std_logic_vector:= "010";
    constant MUX_DRAM_UH    : std_logic_vector:= "011";
    constant MUX_DRAM_W     : std_logic_vector:= "100";
    constant RF_IN_ALU      : std_logic:= '0';
    constant RF_IN_DRAM     : std_logic:= '1';
    --WB outputs
    constant RF_WR_OFF      : std_logic:= '0';
    constant RF_WR_ON       : std_logic:= '1';

    --                                      ID                                                                        EXE
           ↪              MEM                                              WB
    --                                      MUX_NPC_base_sel     MUX_ALU_IN1_sel     MUX_IMM_sel  BRANCH_COND         MUX_ALU_IN2_sel
           ↪  ALU_OP           DRAM_WR_EN      MUX_DRAM_sel    MUX_RF_IN_sel    RF_WR_EN
    constant NOP_outs   :CU_OUTS_type:=(    ('-',                "--",               "---",       BRANCH_COND_NO),    ('-',
           ↪  ALU_DONT_CARE), (DRAM_WR_OFF,   "---",          '-'),            (others=>RF_WR_OFF  ));
    --OPIMM instructions outs
    constant ADDI_outs  :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_ADD),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SLI_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_SL),         (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SLTI_outs  :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_SLT),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SLTIU_outs :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_SLTU),       (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant XORI_outs  :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_XOR),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SRLI_outs  :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_SRL),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SRAI_outs  :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_SRA),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant ORI_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_OR),         (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant ANDI_outs  :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_AND),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    --OP instructions outs
    constant ADD_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         "---",       BRANCH_COND_NO),    (ALU_IN2_RF,
           ↪  ALU_ADD),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SL_outs    :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         "---",       BRANCH_COND_NO),    (ALU_IN2_RF,
           ↪  ALU_SL),         (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SLT_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         "---",       BRANCH_COND_NO),    (ALU_IN2_RF,
           ↪  ALU_SLT),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SLTU_outs  :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         "---",       BRANCH_COND_NO),    (ALU_IN2_RF,
           ↪  ALU_SLTU),       (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant XOR_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         "---",       BRANCH_COND_NO),    (ALU_IN2_RF,
           ↪  ALU_XOR),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SRL_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         "---",       BRANCH_COND_NO),    (ALU_IN2_RF,
           ↪  ALU_SRL),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant OR_outs    :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         "---",       BRANCH_COND_NO),    (ALU_IN2_RF,
           ↪  ALU_OR),         (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant AND_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         "---",       BRANCH_COND_NO),    (ALU_IN2_RF,
           ↪  ALU_AND),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SUB_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         "---",       BRANCH_COND_NO),    (ALU_IN2_RF,
           ↪  ALU_SUB),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    constant SRA_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         "---",       BRANCH_COND_NO),    (ALU_IN2_RF,
           ↪  ALU_SRA),        (DRAM_WR_OFF,   "---",          RF_IN_ALU),      (others=>RF_WR_ON   ));
    --STORE instructions outs
    constant SB_outs    :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_STYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_ADD),        (DRAM_WR_B,     "---",          '-'),            (others=>RF_WR_OFF  ));
    constant SH_outs    :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_STYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_ADD),        (DRAM_WR_H,     "---",          '-'),            (others=>RF_WR_OFF  ));
    constant SW_outs    :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_STYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_ADD),        (DRAM_WR_W,     "---",          '-'),            (others=>RF_WR_OFF  ));
    --LOAD instructions outs
    constant LB_outs    :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_ADD),        (DRAM_WR_OFF,   MUX_DRAM_SB,    RF_IN_DRAM),     (others=>RF_WR_ON   ));
    constant LH_outs    :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_ADD),        (DRAM_WR_OFF,   MUX_DRAM_SH,    RF_IN_DRAM),     (others=>RF_WR_ON   ));
    constant LW_outs    :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_ADD),        (DRAM_WR_OFF,   MUX_DRAM_W,     RF_IN_DRAM),     (others=>RF_WR_ON   ));
    constant LBU_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_ADD),        (DRAM_WR_OFF,   MUX_DRAM_UB,    RF_IN_DRAM),     (others=>RF_WR_ON   ));
    constant LHU_outs   :CU_OUTS_type:=(    ('-',                ALU_IN1_RF,         IMM_ITYPE,   BRANCH_COND_NO),    (ALU_IN2_IMM,
           ↪  ALU_ADD),        (DRAM_WR_OFF,   MUX_DRAM_UH,    RF_IN_DRAM),     (others=>RF_WR_ON   ));
    --BRANCH instructions outs
    constant BEQ_outs   :CU_OUTS_type:=(    (NPC_BASE_PC,        ALU_IN1_RF,         IMM_BTYPE,   BRANCH_COND_EQ),    ('-',
           ↪  ALU_DONT_CARE), (DRAM_WR_OFF,   "---",          '-'),            (others=>RF_WR_OFF  ));
    constant BNE_outs   :CU_OUTS_type:=(    (NPC_BASE_PC,        ALU_IN1_RF,         IMM_BTYPE,   BRANCH_COND_NE),    ('-',
           ↪  ALU_DONT_CARE), (DRAM_WR_OFF,   "---",          '-'),            (others=>RF_WR_OFF  ));
    constant BLT_outs   :CU_OUTS_type:=(    (NPC_BASE_PC,        ALU_IN1_RF,         IMM_BTYPE,   BRANCH_COND_LT),    ('-',
           ↪  ALU_DONT_CARE), (DRAM_WR_OFF,   "---",          '-'),            (others=>RF_WR_OFF  ));
    constant BGE_outs   :CU_OUTS_type:=(    (NPC_BASE_PC,        ALU_IN1_RF,         IMM_BTYPE,   BRANCH_COND_GE),    ('-',
        ↪ ALU_DONT_CARE), (DRAM_WR_OFF,   "---",          '-'),           (others=>RF_WR_OFF  ));
    constant BLTU_outs  :CU_OUTS_type:=(    (NPC_BASE_PC,       ALU_IN1_RF,         IMM_BTYPE,  BRANCH_COND_LTU),   ('-',
        ↪ ALU_DONT_CARE), (DRAM_WR_OFF,   "---",          '-'),           (others=>RF_WR_OFF  ));
    constant BGEU_outs  :CU_OUTS_type:=(    (NPC_BASE_PC,       ALU_IN1_RF,         IMM_BTYPE,  BRANCH_COND_GEU),   ('-',
        ↪ ALU_DONT_CARE), (DRAM_WR_OFF,   "---",          '-'),           (others=>RF_WR_OFF  ));
    --J-TYPE instructions outs
    constant JAL_outs   :CU_OUTS_type:=(    (NPC_BASE_PC,       ALU_IN1_PC_plus4,   IMM_JTYPE,  BRANCH_COND_ALWAYS),('-',
        ↪ ALU_IN1),       (DRAM_WR_OFF,   "---",          RF_IN_ALU),     (others=>RF_WR_ON   ));
    constant JALR_outs  :CU_OUTS_type:=(    (NPC_BASE_REG,      ALU_IN1_PC_plus4,   IMM_ITYPE,  BRANCH_COND_ALWAYS),('-',
        ↪ ALU_IN1),       (DRAM_WR_OFF,   "---",          RF_IN_ALU),     (others=>RF_WR_ON   ));
    --U-TYPE instruction outs
    constant LUI_outs   :CU_OUTS_type:=(    ('-',               "--",               IMM_UTYPE,  BRANCH_COND_NO),    (ALU_IN2_IMM,
        ↪ ALU_IN2),       (DRAM_WR_OFF,   "---",          RF_IN_ALU),     (others=>RF_WR_ON   ));
    constant AUIPC_outs :CU_OUTS_type:=(    ('-',               ALU_IN1_PC,         IMM_UTYPE,  BRANCH_COND_NO),    (ALU_IN2_IMM,
        ↪ ALU_ADD),       (DRAM_WR_OFF,   "---",          RF_IN_ALU),     (others=>RF_WR_ON   ));
    --custom instructions
    constant ABS_outs   :CU_OUTS_type:=(    ('-',               ALU_IN1_RF,         "---",      BRANCH_COND_NO),    ('-',
        ↪ ALU_ABS),       (DRAM_WR_OFF,   "---",          RF_IN_ALU),     (others=>RF_WR_ON   ));

end package CU_package;
```

Listing B.5: FU_package.vhd

```
library ieee;
use ieee.std_logic_1164.all;

package FU_package is

    --FU package
    type FU_OUTS_type is record
        MUX_ALU_IN1_sel             : std_logic_vector(1 downto 0);
        MUX_ALU_IN2_sel             : std_logic_vector(1 downto 0);
        MUX_DRAM_IN_sel             : std_logic;
    end record FU_OUTS_type;

    constant ALU_forward_NO     : std_logic_vector(1 downto 0):= "00";
    constant ALU_forward_EXE    : std_logic_vector(1 downto 0):= "01";
    constant ALU_forward_MEM    : std_logic_vector(1 downto 0):= "10";
    constant ALU_forward_WB     : std_logic_vector(1 downto 0):= "11";
    constant DRAM_forward_NO    : std_logic:= '0';
    constant DRAM_forward_WB    : std_logic:= '1';

end package FU_package;
```

Listing B.6: HDU_package.vhd

```
library ieee;
use ieee.std_logic_1164.all;

package HDU_package is

    type HDU_OUTS_type is record
        PC_EN       : std_logic;
        IF_ID_EN    : std_logic;
        ID_EXE_EN   : std_logic;
        EXE_MEM_EN  : std_logic;
        MEM_WB_EN   : std_logic;
        ID_bubble   : std_logic;
        EXE_bubble  : std_logic;
        MEM_bubble  : std_logic;
        WB_bubble   : std_logic;
    end record HDU_OUTS_type;

end package HDU_package;
```

Listing B.7: RISCV.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.RISCV_package.all;
use work.CU_package.all;
use work.my_package.all;
use work.FU_package.all;
use work.HDU_package.all;

entity RISCV is
    generic(
        BPU_TAG_FIELD_SIZE  : integer :=8;
        BPU_SET_FIELD_SIZE  : integer :=3;
        BPU_LINES_PER_SET   : integer :=4
    );
    port(
        IRAM_ADDR   : out   word;
        IRAM_OUT    : in    word;
        DRAM_ADDR   : out   word;
        DRAM_IN     : out   word;
        DRAM_OUT    : in    word;
        DRAM_WR_EN  : out   std_logic_vector(1 downto 0);
        CLK         : in    std_logic;
        RST         : in    std_logic
    );
end entity RISCV;

architecture structural of RISCV is

    component reg is
        Generic (N : positive:= 1 );                                --number of bits
        Port(   D       : In    std_logic_vector(N-1 downto 0);      --data input
                Q       : Out   std_logic_vector(N-1 downto 0);      --data output
                EN      : In    std_logic;                           --enable active high
                CLK     : In    std_logic;                           --clock
                RST     : In    std_logic);                          --asynchronous reset active low
    end component reg;

    component CU is
        port(
            INSTR_ID    : in word;                  --instruction register input
            CU_OUTS     : out CU_OUTS_type          --CU outs to datapath
        );
    end component CU;

    component HDU is
        port(
            INSTR_ID        : in word;
            INSTR_EXE       : in word;
            misprediction   : in std_logic;
            HDU_OUTS        : out HDU_OUTS_type;
            clk             : in std_logic;
            rst             : in std_logic
        );
    end component HDU;

    component FU is
        port(
            INSTR_ID    : in word;
            INSTR_EXE   : in word;
            INSTR_MEM   : in word;
            INSTR_WB    : in word;
            FU_OUTS     : out FU_OUTS_type
        );
    end component FU;

    component datapath is
        generic(
            BPU_TAG_FIELD_SIZE  : integer :=8;
            BPU_SET_FIELD_SIZE  : integer :=3;
            BPU_LINES_PER_SET   : integer :=4
        );
        port(
            IRAM_ADDR       : out   word;
            IRAM_OUT        : in    word;
            DRAM_ADDR       : out   word;
            DRAM_IN         : out   word;
            DRAM_OUT        : in    word;
            control_from_CU : in    CU_OUTS_type;
            control_from_FU : in    FU_OUTS_type;
            control_from_HDU: in    HDU_OUTS_type;
            misprediction   : out   std_logic;
            CLK             : in    std_logic;
            RST             : in    std_logic
        );
    end component datapath;

    --signals to pipelined the CU outputs
    signal CU_OUTS, CU_OUTS_pipelined                           : CU_OUTS_type;
    signal CU_OUTS_EXE_atEXE                                    : CU_OUTS_EXE_type;
    signal CU_OUTS_MEM_atEXE, CU_OUTS_MEM_atMEM                 : CU_OUTS_MEM_type;
    signal CU_OUTS_WB_atEXE, CU_OUTS_WB_atMEM, CU_OUTS_WB_atWB  : CU_OUTS_WB_type;

    --signals to connect components
    signal HDU_OUTS : HDU_OUTS_type;
    signal FU_OUTS  : FU_OUTS_type;
    signal misprediction    : std_logic;

    --signals to record the instruction for each stage
    signal INSTR_ID, INSTR_EXE, INSTR_MEM, INSTR_WB : word;

begin

    instr_prop: process(CLK, RST) is
    begin
        if RST='0' then
            INSTR_ID<=(others=>'0');
            INSTR_EXE<=(others=>'0');
            INSTR_MEM<=(others=>'0');
            INSTR_WB<=(others=>'0');
        elsif rising_edge(CLK) then
```

```vhdl
            if HDU_OUTS.ID_bubble='1' then
                INSTR_ID<=INSTR_NOP;
            elsif HDU_OUTS.IF_ID_EN='1' then
                INSTR_ID<=IRAM_OUT;
            end if;
            if HDU_OUTS.EXE_bubble='1' then
                INSTR_EXE<=INSTR_NOP;
            elsif HDU_OUTS.ID_EXE_EN='1' then
                INSTR_EXE<=INSTR_ID;
            end if;
            if HDU_OUTS.MEM_bubble='1' then
                INSTR_MEM<=INSTR_NOP;
            elsif HDU_OUTS.EXE_MEM_EN='1' then
                INSTR_MEM<=INSTR_EXE;
            end if;
            if HDU_OUTS.WB_bubble='1' then
                INSTR_WB<=INSTR_NOP;
            elsif HDU_OUTS.MEM_WB_EN='1' then
                INSTR_WB<=INSTR_MEM;
            end if;
        end if;
    end process instr_prop;

    HDU_instance: HDU
        port map(
            HDU_OUTS=>HDU_OUTS,
            INSTR_ID=>INSTR_ID,
            INSTR_EXE=>INSTR_EXE,
            misprediction=>misprediction,
            clk=>clk,
            rst=>rst
        );

    FU_instance: FU
        port map(
            INSTR_ID=>INSTR_ID,
            INSTR_EXE=>INSTR_EXE,
            INSTR_MEM=>INSTR_MEM,
            INSTR_WB=>INSTR_WB,
            FU_OUTS=>FU_OUTS
        );

    CU_instance: CU
        port map(
            INSTR_ID=>INSTR_ID,
            CU_OUTS=>CU_OUTS
        );

    REGS_CU: process(CLK, RST)
    begin
        if RST='0' then
            CU_OUTS_EXE_atEXE<=NOP_outs.EXE;
            CU_OUTS_MEM_atEXE<=NOP_outs.MEM;
            CU_OUTS_WB_atEXE<=NOP_outs.WB;
            CU_OUTS_MEM_atMEM<=NOP_outs.MEM;
            CU_OUTS_WB_atMEM<=NOP_outs.WB;
            CU_OUTS_WB_atWB<=NOP_outs.WB;
        elsif rising_edge(CLK) then
            if HDU_OUTS.EXE_bubble='1' then
                CU_OUTS_EXE_atEXE<=NOP_outs.EXE;
                CU_OUTS_MEM_atEXE<=NOP_outs.MEM;
                CU_OUTS_WB_atEXE<=NOP_outs.WB;
            elsif HDU_OUTS.ID_EXE_EN='1' then
                CU_OUTS_EXE_atEXE<=CU_OUTS.EXE;
                CU_OUTS_MEM_atEXE<=CU_OUTS.MEM;
                CU_OUTS_WB_atEXE<=CU_OUTS.WB;
            end if;
            if HDU_OUTS.MEM_bubble='1' then
                CU_OUTS_MEM_atMEM<=NOP_outs.MEM;
                CU_OUTS_WB_atMEM<=NOP_outs.WB;
            elsif HDU_OUTS.EXE_MEM_EN='1' then
                CU_OUTS_MEM_atMEM<=CU_OUTS_MEM_atEXE;
                CU_OUTS_WB_atMEM<=CU_OUTS_WB_atEXE;
            end if;
            if HDU_OUTS.WB_bubble='1' then
                CU_OUTS_WB_atWB<=NOP_outs.WB;
            elsif HDU_OUTS.MEM_WB_EN='1' then
                CU_OUTS_WB_atWB<=CU_OUTS_WB_atMEM;
            end if;
        end if;
    end process REGS_CU;

    CU_OUTS_pipelined<=(
        ID=>CU_OUTS.ID,
        EXE=>CU_OUTS_EXE_atEXE,
        MEM=>CU_OUTS_MEM_atMEM,
        WB=>CU_OUTS_WB_atWB
    );

    datapath_instance: datapath
        generic map(
            BPU_TAG_FIELD_SIZE=>BPU_TAG_FIELD_SIZE,
            BPU_SET_FIELD_SIZE=>BPU_SET_FIELD_SIZE,
            BPU_LINES_PER_SET=>BPU_LINES_PER_SET
        )
        port map(
            IRAM_ADDR=>IRAM_ADDR,
            IRAM_OUT=>IRAM_OUT,
            DRAM_ADDR=>DRAM_ADDR,
            DRAM_IN=>DRAM_IN,
            DRAM_OUT=>DRAM_OUT,
            control_from_CU=>CU_OUTS_pipelined,
            control_from_FU=>FU_OUTS,
            control_from_HDU=>HDU_OUTS,
            misprediction=>misprediction,
            CLK=>CLK,
            RST=>RST
        );

    DRAM_WR_EN<=CU_OUTS_pipelined.MEM.DRAM_WR_EN;

end architecture structural;
```

Listing B.8: CU.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.RISCV_package.all;
use work.ALU_package.all;
use work.CU_package.all;

entity CU is
    port(
        INSTR_ID    : in word;                  --instruction register output
        CU_OUTS     : out CU_outs_type          --CU outs to datapath
    );
end entity CU;

architecture behavioral of CU is

    constant LUT_BRANCH: LUT_FUNCT3_type:=(
        FUNCT3_BRANCH_type'pos(FUNCT3_BEQ)  => BEQ_outs,
        FUNCT3_BRANCH_type'pos(FUNCT3_BNE)  => BNE_outs,
        FUNCT3_BRANCH_type'pos(FUNCT3_BLT)  => BLT_outs,
        FUNCT3_BRANCH_type'pos(FUNCT3_BGE)  => BGE_outs,
        FUNCT3_BRANCH_type'pos(FUNCT3_BLTU) => BLTU_outs,
        FUNCT3_BRANCH_type'pos(FUNCT3_BGEU) => BGEU_outs,
        others  => NOP_outs
    );

    constant LUT_LOAD: LUT_FUNCT3_type:=(
        FUNCT3_LOAD_type'pos(FUNCT3_LB)     => LB_outs,
        FUNCT3_LOAD_type'pos(FUNCT3_LH)     => LH_outs,
        FUNCT3_LOAD_type'pos(FUNCT3_LW)     => LW_outs,
        FUNCT3_LOAD_type'pos(FUNCT3_LBU)    => LBU_outs,
        FUNCT3_LOAD_type'pos(FUNCT3_LHU)    => LHU_outs,
        others  => NOP_outs
    );

    constant LUT_STORE: LUT_FUNCT3_type:=(
        FUNCT3_STORE_type'pos(FUNCT3_SB)    => SB_outs,
        FUNCT3_STORE_type'pos(FUNCT3_SH)    => SH_outs,
        FUNCT3_STORE_type'pos(FUNCT3_SW)    => SW_outs,
        others  => NOP_outs
    );

    signal LUT_OPIMM: LUT_FUNCT3_type;  --computed in runtime as one of the two following LUT (they differ only for shift right)

    constant LUT_OPIMM_0: LUT_FUNCT3_type:=(
        FUNCT3_OPIMM_type'pos(FUNCT3_ADDI)  => ADDI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_SLI)   => SLI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_SLTI)  => SLTI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_SLTIU) => SLTIU_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_XORI)  => XORI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_SRI)   => SRLI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_ORI)   => ORI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_ANDI)  => ANDI_outs,
        others  => NOP_outs
    );

    constant LUT_OPIMM_1: LUT_FUNCT3_type:=(
        FUNCT3_OPIMM_type'pos(FUNCT3_ADDI)  => ADDI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_SLI)   => SLI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_SLTI)  => SLTI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_SLTIU) => SLTIU_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_XORI)  => XORI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_SRI)   => SRAI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_ORI)   => ORI_outs,
        FUNCT3_OPIMM_type'pos(FUNCT3_ANDI)  => ANDI_outs,
        others  => NOP_outs
    );

    constant LUT_OP_STD0: LUT_FUNCT3_type:=(
        FUNCT3_OP_STD0_type'pos(FUNCT3_ADD)     => ADD_outs,
        FUNCT3_OP_STD0_type'pos(FUNCT3_SL)      => SL_outs,
        FUNCT3_OP_STD0_type'pos(FUNCT3_SLT)     => SLT_outs,
        FUNCT3_OP_STD0_type'pos(FUNCT3_SLTU)    => SLTU_outs,
        FUNCT3_OP_STD0_type'pos(FUNCT3_XOR)     => XOR_outs,
        FUNCT3_OP_STD0_type'pos(FUNCT3_SRL)     => SRL_outs,
        FUNCT3_OP_STD0_type'pos(FUNCT3_OR)      => OR_outs,
        FUNCT3_OP_STD0_type'pos(FUNCT3_AND)     => AND_outs,
        others  => NOP_outs
    );

    constant LUT_OP_STD1: LUT_FUNCT3_type:=(
        FUNCT3_OP_STD1_type'pos(FUNCT3_SUB)     => SUB_outs,
        FUNCT3_OP_STD1_type'pos(FUNCT3_SRA)     => SRA_outs,
        others  => NOP_outs
    );

    constant LUT_OP_RV32M: LUT_FUNCT3_type:=(
        -- FUNCT3_OP_RV32M_type'pos(FUNCT3_MUL) => MUL_outs,
        -- FUNCT3_OP_RV32M_type'pos(FUNCT3_MULH)    => MULH_outs,
        -- FUNCT3_OP_RV32M_type'pos(FUNCT3_MULHSU)  => MULHSU_outs,
        -- FUNCT3_OP_RV32M_type'pos(FUNCT3_MULHU)   => MULHU_outs,
        -- FUNCT3_OP_RV32M_type'pos(FUNCT3_DIV) => DIV_outs,
        -- FUNCT3_OP_RV32M_type'pos(FUNCT3_DIVU)    => DIVU_outs,
        -- FUNCT3_OP_RV32M_type'pos(FUNCT3_REM) => REM_outs,
        -- FUNCT3_OP_RV32M_type'pos(FUNCT3_REMU)    => REMU_outs,
        others  => NOP_outs
    );

    constant LUT_OTHERS: LUT_OPCODE_type :=(
        OPCODE_type'pos(OPCODE_JALR)    => JALR_outs,
        OPCODE_type'pos(OPCODE_JAL)     => JAL_outs,
        OPCODE_type'pos(OPCODE_LUI)     => LUI_outs,
        OPCODE_type'pos(OPCODE_AUIPC)   => AUIPC_outs,
        OPCODE_type'pos(OPCODE_CUSTOM0) => ABS_outs,
        others  => NOP_outs
    );

    signal OPCODE: OPCODE_type;
    signal FUNCT7: FUNCT7_type;

begin

    OPCODE<=get_OPCODE(INSTR_ID);
    FUNCT7<=get_FUNCT7(INSTR_ID);

    LUT_OPIMM<= LUT_OPIMM_0 when FUNCT7(5)='0' else
                LUT_OPIMM_1 when FUNCT7(5)='1' else
                (others=>NOP_outs);


    CU_OUTS <=  LUT_BRANCH(to_integer(get_FUNCT3_BRANCH(INSTR_ID)))     when OPCODE=OPCODE_BRANCH                       else
                LUT_LOAD(to_integer(get_FUNCT3_LOAD(INSTR_ID)))         when OPCODE=OPCODE_LOAD                         else
                LUT_STORE(to_integer(get_FUNCT3_STORE(INSTR_ID)))       when OPCODE=OPCODE_STORE                        else
                LUT_OPIMM(to_integer(get_FUNCT3_OPIMM(INSTR_ID)))       when OPCODE=OPCODE_OPIMM                        else
                LUT_OP_STD0(to_integer(get_FUNCT3_OP_STD0(INSTR_ID)))   when OPCODE=OPCODE_OP and FUNCT7=FUNCT7_STD0    else
                LUT_OP_STD1(to_integer(get_FUNCT3_OP_STD1(INSTR_ID)))   when OPCODE=OPCODE_OP and FUNCT7=FUNCT7_STD1    else
                LUT_OP_RV32M(to_integer(get_FUNCT3_OP_RV32M(INSTR_ID))) when OPCODE=OPCODE_OP and FUNCT7=FUNCT7_RV32M   else
                LUT_OTHERS(to_integer(OPCODE)/4);

end architecture behavioral;
```

Listing B.9: FU.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.RISCV_package.all;
use work.FU_package.all;

entity FU is
    port(
        INSTR_ID    : in word;
        INSTR_EXE   : in word;
        INSTR_MEM   : in word;
        INSTR_WB    : in word;
        FU_OUTS     : out FU_OUTS_type
    );
end entity FU;

architecture behavioral of FU is

begin

    FU_process: process(INSTR_ID, INSTR_EXE, INSTR_MEM, INSTR_WB) is

        variable RS1_ID, RS2_ID, RS2_EXE, RD_EXE, RD_MEM, RD_WB : integer range 0 to N_REG;

    begin

        --compute usefull constants
        RS1_ID  :=  get_RS1(INSTR_ID);
        RS2_ID  :=  get_RS2(INSTR_ID);
        RS2_EXE :=  get_RS2(INSTR_EXE);
        RD_EXE  :=  get_RD(INSTR_EXE);
        RD_MEM  :=  get_RD(INSTR_MEM);
        RD_WB   :=  get_RD(INSTR_WB);

        --default assignment
        FU_OUTS.MUX_ALU_IN1_sel<=ALU_forward_NO;
        FU_OUTS.MUX_ALU_IN2_sel<=ALU_forward_NO;
        FU_OUTS.MUX_DRAM_IN_sel<=DRAM_forward_NO;

        --ALU_IN1 forwarding
        if RS1_ID/=0 and RS1_ID=RD_EXE and not is_a_branch(INSTR_EXE) and not is_a_store(INSTR_EXE) then    --if INSTR_EXE is a load,
              ↪ the HDU intervenes
            FU_OUTS.MUX_ALU_IN1_sel<=ALU_forward_EXE;
        elsif RS1_ID/=0 and RS1_ID=RD_MEM and not is_a_branch(INSTR_MEM) and not is_a_store(INSTR_MEM) then
            FU_OUTS.MUX_ALU_IN1_sel<=ALU_forward_MEM;
        elsif RS1_ID/=0 and RS1_ID=RD_WB and not is_a_branch(INSTR_WB) and not is_a_store(INSTR_WB) then
            FU_OUTS.MUX_ALU_IN1_sel<=ALU_forward_WB;
        end if;

        --ALU_IN2 forwarding
        if RS2_ID/=0 and RS2_ID=RD_EXE and not is_a_branch(INSTR_EXE) and not is_a_store(INSTR_EXE) then    --if INSTR_EXE is a load,
              ↪ the HDU intervenes
            FU_OUTS.MUX_ALU_IN2_sel<=ALU_forward_EXE;
        elsif RS2_ID/=0 and RS2_ID=RD_MEM and not is_a_branch(INSTR_MEM) and not is_a_store(INSTR_MEM) then
            FU_OUTS.MUX_ALU_IN2_sel<=ALU_forward_MEM;
        elsif RS2_ID/=0 and RS2_ID=RD_WB and not is_a_branch(INSTR_WB) and not is_a_store(INSTR_WB) then
            FU_OUTS.MUX_ALU_IN2_sel<=ALU_forward_WB;
        end if;

        --DRAM IN forwarding
        if RS2_EXE/=0 and RS2_EXE=RD_MEM then
            FU_OUTS.MUX_DRAM_IN_sel<=DRAM_forward_WB;
        end if;

    end process FU_process;

end architecture behavioral;
```

Listing B.10: HDU.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use work.RISCV_package.all;
use work.HDU_package.all;

entity HDU is   --hazard detection unit
    port(
        INSTR_ID        : in word;
        INSTR_EXE       : in word;
        misprediction   : in std_logic;
        HDU_OUTS        : out HDU_OUTS_type;
        clk             : in std_logic;
        rst             : in std_logic
    );
end entity HDU;

architecture behavioral of HDU is

    signal count: integer range 0 to 3;

begin

    HDU_proc: process(INSTR_ID, INSTR_EXE, misprediction, count) is

        variable RS1_ID, RS2_ID, RD_EXE : integer range 0 to N_REG;

    begin

        --compute useful constants
        RS1_ID  :=  get_RS1(INSTR_ID);
        RS2_ID  :=  get_RS2(INSTR_ID);
        RD_EXE  :=  get_Rd(INSTR_EXE);

        --default assignment
        HDU_OUTS.PC_EN      <='1';
        HDU_OUTS.IF_ID_EN   <='1';
        HDU_OUTS.ID_EXE_EN  <='1';
        HDU_OUTS.EXE_MEM_EN <='1';
        HDU_OUTS.MEM_WB_EN  <='1';
        HDU_OUTS.ID_bubble  <='0';
        HDU_OUTS.EXE_bubble <='0';
        HDU_OUTS.MEM_bubble <='0';
        HDU_OUTS.WB_bubble  <='0';

        --multicycle operations structural hazard
        if is_a_mult(INSTR_EXE) and count/=3 then
            HDU_OUTS.PC_EN<='0';
            HDU_OUTS.IF_ID_EN<='0';
            HDU_OUTS.ID_EXE_EN<='0';
            HDU_OUTS.EXE_MEM_EN<='0';
            HDU_OUTS.MEM_bubble<='1';
        --load from DRAM data hazard
        elsif (RS1_ID/=0 and RS1_ID=RD_EXE and is_a_load(INSTR_EXE) and not is_a_jump(INSTR_ID) and not is_a_upperimm(INSTR_ID)) or --
            ↪ every instruction following a load that requires its data in RS1 can trigger a hazard
        (RS2_ID/=0 and RS1_ID=RD_EXE and is_a_load(INSTR_EXE) and (is_a_op(INSTR_ID) or is_a_branch(INSTR_ID))) then                --in
            ↪  the case the data is required in RS2, if the instruction is a store, it can forward the data in the next clock cycle
            HDU_OUTS.PC_EN<='0';
            HDU_OUTS.IF_ID_EN<='0';
            HDU_OUTS.ID_EXE_EN<='0';
            HDU_OUTS.EXE_bubble<='1';
        --branch misprediction control hazard
        elsif misprediction='1' then
            HDU_OUTS.IF_ID_EN<='0';
            HDU_OUTS.ID_bubble<='1';
        end if;

    end process HDU_proc;

    count_proc: process(clk, rst)
    begin
        if rst='0' then
            count<=0;
        elsif rising_edge(clk) then
            if is_a_mult(INSTR_EXE) then
                if count=3 then
                    count<=0;
                else
                    count<=count+1;
                end if;
            end if;
        end if;
    end process count_proc;

end architecture behavioral;
```

Listing B.11: BPU.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.RISCV_package.all;

entity BPU is
    generic(
        TAG_FIELD_SIZE  : integer := 8;
        SET_FIELD_SIZE  : integer := 3;
        LINES_PER_SET   : integer := 4
    );
    port(
        clk             : in std_logic;
        rst             : in std_logic;
        IRAM_out        : in word;
        pc              : in word;
        pc_plus4        : in word;
        npc             : out word;
        misprediction   : out std_logic;
        actual_NPC      : in word;
        IF_ID_EN        : in std_logic;     --fetch-decode registers enable
        ID_EXE_EN       : in std_logic      --decode-execute registers enable
    );
end entity BPU;

architecture behavioral of BPU is

    constant N_SET: integer := 2**SET_FIELD_SIZE;
    constant DATA_SIZE: integer:= WORD_SIZE-2;

    subtype SET_range is integer range SET_FIELD_SIZE+1 downto 2;
    subtype TAG_range is integer range SET_FIELD_SIZE+TAG_FIELD_SIZE+1 downto SET_FIELD_SIZE+2;
    subtype DATA_range is integer range WORD_SIZE-1 downto 2;

    subtype SET_index_type is integer range 0 to N_SET-1;
    subtype LINE_index_type is integer range 0 to LINES_PER_SET-1;
    subtype LINE_index_type_ext is integer range -1 to LINES_PER_SET-1;
    subtype TAG_type is std_logic_vector(TAG_FIELD_SIZE-1 downto 0);
    subtype DATA_type is std_logic_vector(DATA_SIZE-1 downto 0);

    type CACHE_LINE_type is record
        TAG: TAG_type;
        DATA: DATA_type;
        YOUTH: LINE_index_type_ext;     --the higher this value, the more recently the data has been accessed, if equal to -1 this line
            ↪ is free
    end record CACHE_LINE_type;

    type CACHE_SET_type is array (LINE_index_type) of CACHE_LINE_type;

    type CACHE_type is array (SET_index_type) of CACHE_SET_type;

    signal cache : CACHE_type;

    --asyncronous signal used by read_proc
    signal prediction : word;
    signal hit_index : LINE_index_type_ext;
    --asynchronous signal used to read misprediction port
    signal misprediction_sig: std_logic;

    --synchronous signal used by write_proc
    signal last_pc_plus4 : word;
    signal last_prediction : word;
    signal verify: std_logic;

begin

    misprediction<=misprediction_sig;
    npc<=prediction;

    read_proc: process(cache, pc_plus4, IRAM_out, pc, misprediction_sig, actual_NPC)
        variable set: SET_index_type;
        variable tag: TAG_type;
    begin
        --useful constants
        set:=to_integer(unsigned(pc(SET_range)));
        tag:=pc(TAG_range);
        --default assignments
        prediction<=pc_plus4;
        hit_index<= -1;
        if misprediction_sig='1' then
            prediction<=actual_NPC;
        elsif is_a_branch(IRAM_out) or is_a_jump(IRAM_out) then
            for i in 0 to LINES_PER_SET-1 loop
                if cache(set)(i).TAG=tag and cache(set)(i).YOUTH/=-1 then
                    hit_index<=i;
                    prediction<=cache(set)(i).DATA & "00";
                    exit;
                end if;
            end loop;
        end if;
    end process read_proc;

    write_proc: process(clk, rst)
        variable last_set: SET_index_type;
        variable last_tag: TAG_type;
        variable last_hit_index: LINE_index_type_ext;
        variable oldest_line: LINE_index_type;
        procedure update_youth(set: SET_index_type; index_to_update: LINE_index_type) is
            variable youth_to_update: LINE_index_type_ext;
        begin
            youth_to_update:=cache(set)(index_to_update).YOUTH;
            for i in 0 to LINES_PER_SET-1 loop
                if cache(set)(i).YOUTH>youth_to_update and cache(set)(i).YOUTH/=0 then
                    cache(set)(i).YOUTH<=cache(set)(i).YOUTH-1;
                end if;
            end loop;
            cache(set)(index_to_update).YOUTH<=LINES_PER_SET-1;
        end procedure update_youth;
        procedure get_oldest_line(set: SET_index_type) is
            variable smaller_youth: LINE_index_type_ext;
        begin
            smaller_youth:=LINES_PER_SET-1;
            oldest_line:=0;
            for i in 0 to LINES_PER_SET-1 loop
                if cache(set)(i).YOUTH<smaller_youth then
                    smaller_youth:=cache(set)(i).YOUTH;
                    oldest_line:=i;
                end if;
            end loop;
        end procedure get_oldest_line;
    begin
        if rst='0' then
            cache<=(others=>(others=>(TAG=>(others=>'0'), DATA=>(others=>'0'), YOUTH=>-1)));
            verify<='0';
            last_prediction<=(others=>'0');
            last_pc_plus4<=(others=>'0');
            last_set:=0;
            last_tag:=(others=>'0');
            last_hit_index:= -1;
        elsif rising_edge(clk) then
            if ID_EXE_EN='1' then
                if verify='1' then  --if there was a branch/jump fetch in the previous clock cycle
                    if last_hit_index=-1 then   --if there was a cache miss, save the new address (both if the prediction was correct or
                        ↪  if it was not)
                        get_oldest_line(last_set);
                        cache(last_set)(oldest_line).TAG<=last_tag;
                        cache(last_set)(oldest_line).DATA<=actual_NPC(DATA_range);
                        update_youth(last_set, oldest_line);
                    else    --if there was a cache hit
                        if misprediction_sig='1' then   --if there was a cache hit but a wrong prediction update the correct address
                            cache(last_set)(last_hit_index).DATA<=actual_NPC(DATA_range);
                            update_youth(last_set, last_hit_index);
                        else --if there was a cache hit and a correct prediction
                            update_youth(last_set, last_hit_index);
                        end if;
                    end if;
                    verify<='0';
                end if;
            end if;
            if IF_ID_EN='1' then
                if (is_a_branch(IRAM_out) or is_a_jump(IRAM_out)) and misprediction_sig='0' then
```

```
                    last_pc_plus4<= pc_plus4;
                    last_prediction<= prediction;
                    last_set:= to_integer(unsigned(pc(SET_range)));
                    last_tag:= pc(TAG_range);
                    last_hit_index:= hit_index;
                    verify<='1';
                end if;
            end if;
        end if;
    end process write_proc;

    misp_proc: process(verify, last_prediction, actual_NPC)
    begin
        misprediction_sig<='0'; --default assignment
        if verify='1' and actual_NPC/=last_prediction then
            misprediction_sig<='1';
        end if;
    end process misp_proc;

end architecture behavioral;
```

Listing B.12: datapath.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.CU_package.all;
use work.ALU_package.all;
use work.RISCV_package.all;
use work.my_package.all;
use work.FU_package.all;
use work.HDU_package.all;

entity datapath is
    generic(
        BPU_TAG_FIELD_SIZE  : integer :=8;
        BPU_SET_FIELD_SIZE  : integer :=3;
        BPU_LINES_PER_SET   : integer :=4
    );
    port(
        IRAM_ADDR       : out   word;
        IRAM_OUT        : in    word;
        DRAM_ADDR       : out   word;
        DRAM_IN         : out   word;
        DRAM_OUT        : in    word;
        control_from_CU : in    CU_OUTS_type;
        control_from_FU : in    FU_OUTS_type;
        control_from_HDU: in    HDU_OUTS_type;
        misprediction   : out   std_logic;
        CLK             : in    std_logic;
        RST             : in    std_logic
    );
end entity datapath;

architecture structural of datapath is

    component reg is
        Generic (N : positive:= 1 );                                    --number of bits
        Port(   D       : In    std_logic_vector(N-1 downto 0);      --data input
                Q       : Out   std_logic_vector(N-1 downto 0);      --data output
                EN      : In    std_logic;                           --enable active high
                CLK     : In    std_logic;                           --clock
                RST     : In    std_logic);                          --asynchronous reset active low
    end component reg;

    component PC is
        port(
            PC_IN   : In    word;           --new program count
            PC_OUT  : Out   word;           --current program count
            EN      : In    std_logic;      --enable active high
            CLK     : In    std_logic;      --clock
            RST     : In    std_logic       --asynchronous reset active low
        );
    end component PC;

    component IR is
        port(
            IR_IN   : In    word;           --instruction register input
            IR_OUT  : Out   word;           --instruction register output
            EN      : In    std_logic;      --enable active high
            BUBBLE  : in    std_logic;      --if high a bubble is introduced
            CLK     : In    std_logic;      --clock
            RST     : In    std_logic       --asynchronous reset active low
        );
    end component IR;

    component RCA is
        generic (N:  integer := 8);                             --number of bits
        port (  A:  In  std_logic_vector(N-1 downto 0);      --data input 1
                B:  In  std_logic_vector(N-1 downto 0);      --data input 2
                Ci: In  std_logic;                           --carry in
                S:  Out std_logic_vector(N-1 downto 0);     --data output
                Co: Out std_logic);                         --carry out
    end component RCA;

    component  ADDER_P4 is
        GENERIC (N_BIT  : integer := 32);   --number of bits. Must be a number from 4 to 32 and a multiple of 4
        PORT   (A       : in  std_logic_vector(N_BIT - 1 downto 0);     -- input operand 1
                B       : in  std_logic_vector(N_BIT - 1 downto 0);     -- input operand 2
                add_sub : in  std_logic;                                -- carry-in
                Cout    : out std_logic;                                -- carry-out
                SUM     : out std_logic_vector(N_BIT -1 downto 0));     -- ouput sum
    end component ADDER_P4;

    component RF is
        generic(N_bit:      positive := 64;     --bitwidth
                N_reg:      positive := 32);    --number of address bits, the number of registers is equal to 2**N_address
        port(   CLK:        IN std_logic;       --clock
                RST:        IN std_logic;       --asynchronous reset, active low
                WR_EN:      IN std_logic;       --synchronous write, active high
                ADDR_WR:    IN std_logic_vector(log2_ceiling(N_reg)-1 downto 0);    --writing register address
                ADDR_RD1:   IN std_logic_vector(log2_ceiling(N_reg)-1 downto 0);    --reading register address 1
                ADDR_RD2:   IN std_logic_vector(log2_ceiling(N_reg)-1 downto 0);    --reading register address 2
                DATA_IN:    IN std_logic_vector(N_bit-1 downto 0);      --data to write
                OUT1:       OUT std_logic_vector(N_bit-1 downto 0);     --data to read 1
                OUT2:       OUT std_logic_vector(N_bit-1 downto 0));    --data to read 2
    end component RF;

    component MUX_2to1 is
        Generic (N: integer:= 1);                               --number of bits
        Port (  IN0:    In  std_logic_vector(N-1 downto 0);     --data input 1
                IN1:    In  std_logic_vector(N-1 downto 0);     --data input 2
                SEL:    In  std_logic;                          --selection input
                Y:      Out std_logic_vector(N-1 downto 0));    --data output
    end component MUX_2to1;

    component MUX_4to1 is
    Generic (N: integer:= 1);   --number of bits
    Port (  IN0, IN1, IN2, IN3  : In    std_logic_vector(N-1 downto 0);     --data inputs
            SEL                 : In    std_logic_vector(1 downto 0);       --selection input
            Y                   : Out   std_logic_vector(N-1 downto 0));    --data output
    end component MUX_4to1;

    component MUX_8to1 is
        Generic (N: integer:= 1);   --number of bits
        Port (  IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7  : In    std_logic_vector(N-1 downto 0);     --data inputs
                SEL                 : In    std_logic_vector(2 downto 0);       --selection input
                Y                   : Out   std_logic_vector(N-1 downto 0));    --data output
    end component MUX_8to1;

    component ALU is
        generic (N : integer := 32);                                        -- number of bit
        port(   FUNC            : IN ALU_OP_type;                           -- operation to do
                DATA1, DATA2    : IN std_logic_vector(N-1 downto 0);        -- data inputs
                OUT_ALU         : OUT std_logic_vector(N-1 downto 0));      -- data output
    end component ALU;

    component branch_comp
        generic(N: integer:= 32);   --number of data-in bits
        port(
            BRANCH_COND     : in BRANCH_COND_type;                  --condition to take branch
            DATA_IN1        : in std_logic_vector(N-1 downto 0);    --data to test
            DATA_IN2        : in std_logic_vector(N-1 downto 0);    --data to test
            BRANCH_IS_TAKEN : out std_logic);                       --high if the branch is taken
    end component branch_comp;

    component BPU is
        generic(
            TAG_FIELD_SIZE  : integer := 8;
            SET_FIELD_SIZE  : integer := 3;
            LINES_PER_SET   : integer := 4
        );
        port(
            clk             : in std_logic;
            rst             : in std_logic;
            IRAM_out        : in word;
            pc              : in word;
            pc_plus4        : in word;
            npc             : out word;
            misprediction   : out std_logic;
            actual_NPC      : in word;
            IF_ID_EN        : in std_logic;     --fetch-decode registers enable
            ID_EXE_EN       : in std_logic      --decode-execute registers enable
        );
    end component BPU;

    signal PC_IN, PC_IF, PC_ID, NPC_base, NPC_target, PC_plus4, PC_plus4_ID, actual_NPC                                 : word;
    signal IR_in, IR_out                                                                                                : word;
    signal IMM_ITYPE, IMM_STYPE, IMM_BTYPE, IMM_JTYPE, IMM_UTYPE, IMM_ID, IMM_EXE                                       : word;
    signal RD_ID, RD_EXE, RD_MEM, RD_WB                                                                                 : REG_addr;
    signal RF_OUT1, RF_OUT2                                                                                             : word;
    signal ALU_IN1_ID, ALU_IN1_ID_fw, ALU_IN2_ID_fw, ALU_IN2_EXE_RF, ALU_IN1_EXE, ALU_IN2_EXE, ALU_OUT_EXE, ALU_OUT_MEM : word;
    signal MULT_OUT                                                                                                     : doubleword;
    signal branch_is_taken                                                                                              : std_logic;
    signal DRAM_IN_EXE, DRAM_IN_MEM, DRAM_OUT_sb, DRAM_OUT_ub, DRAM_OUT_sh, DRAM_OUT_uh, DRAM_OUT_w, DRAM_OUT_ext       : word;
    signal RF_IN_MEM, RF_IN_WB                                                                                          : word;
```

```
begin

    -------------------------------------IF stage

    IRAM_ADDR<=PC_IF;
    IR_in<=IRAM_OUT;

    PC_instance: PC
        port map(
            PC_IN=>PC_IN,
            PC_OUT=>PC_IF,
            EN=>control_from_HDU.PC_EN,
            CLK=>clk,
            RST=>rst
        );

    PC_plus4<=std_logic_vector(unsigned(PC_IF)+4);

    BPU_instance: BPU
        generic map(
            TAG_FIELD_SIZE=>BPU_TAG_FIELD_SIZE,
            SET_FIELD_SIZE=>BPU_SET_FIELD_SIZE,
            LINES_PER_SET=>BPU_LINES_PER_SET
        )
        port map(
            clk=>clk,
            rst=>rst,
            IRAM_out=>IRAM_out,
            pc=>PC_IF,
            pc_plus4=>PC_plus4,
            npc=>PC_IN,
            misprediction=>misprediction,
            actual_NPC=>actual_NPC,
            IF_ID_EN=>control_from_HDU.IF_ID_EN,
            ID_EXE_EN=>control_from_HDU.ID_EXE_EN
        );

    REG_PC_plus4: reg
        generic map(N=>WORD_SIZE)
        port map(
            D=>PC_plus4,
            Q=>PC_plus4_ID,
            EN=>control_from_HDU.IF_ID_EN,
            CLK=>clk,
            RST=>rst
        );

    REG_PC: reg
        generic map(N=>WORD_SIZE)
        port map(
            D=>PC_IF,
            Q=>PC_ID,
            EN=>control_from_HDU.IF_ID_EN,
            CLK=>clk,
            RST=>rst
        );

    IR_instance: IR
        port map(
            IR_IN=>IR_in,
            IR_OUT=>IR_out,
            EN=>control_from_HDU.IF_ID_EN,
            BUBBLE=>control_from_HDU.ID_bubble,
            CLK=>clk,
            RST=>rst
        );

    ------------------------------------------------ID stage

    MUX_NPC_base: MUX_2to1
        generic map(N=>WORD_SIZE)
        port map(
            IN0=>PC_ID,
            IN1=>ALU_IN1_ID_fw,
            SEL=>control_from_CU.ID.MUX_NPC_base_sel,
            Y=>NPC_base
        );

    NPC_target<=std_logic_vector(unsigned(NPC_base)+unsigned(IMM_ID));

    branch_comp_instance: branch_comp
        generic map(N=>WORD_SIZE)
        port map(
            BRANCH_COND=>control_from_CU.ID.BRANCH_COND,
            DATA_IN1=>ALU_IN1_ID_fw,
            DATA_IN2=>ALU_IN2_ID_fw,
            BRANCH_IS_TAKEN=>branch_is_taken
        );

    MUX_actual_NPC: MUX_2to1
        generic map(N=>WORD_SIZE)
        port map(
            IN0=>PC_plus4_ID,
            IN1=>NPC_target,
            SEL=>branch_is_taken,
            Y=>actual_NPC
```

```vhdl
        );

    MUX_ALU_IN1: MUX_4to1
        generic map(N=>WORD_SIZE)
        port map(
            IN0=>PC_ID,
            IN1=>PC_plus4_ID,
            IN2=>ALU_IN1_ID_fw,
            IN3=>(others=>'-'),
            SEL=>control_from_CU.ID.MUX_ALU_IN1_sel,
            Y=>ALU_IN1_ID
        );

    MUX_ALU_IN1_fw: MUX_4to1
        generic map(N=>WORD_SIZE)
        port map(
            IN0=>RF_OUT1,
            IN1=>ALU_OUT_EXE,
            IN2=>RF_IN_MEM,
            IN3=>RF_IN_WB,
            SEL=>control_from_FU.MUX_ALU_IN1_sel,
            Y=>ALU_IN1_ID_fw
        );

    MUX_ALU_IN2_fw: MUX_4to1
        generic map(N=>WORD_SIZE)
        port map(
            IN0=>RF_OUT2,
            IN1=>ALU_OUT_EXE,
            IN2=>RF_IN_MEM,
            IN3=>RF_IN_WB,
            SEL=>control_from_FU.MUX_ALU_IN2_sel,
            Y=>ALU_IN2_ID_fw
        );

    RD_ID<=IR_out(RD_range);

    REG_RD_ID_EXE: reg
        generic map(N=>REG_ADDR_SIZE)
        port map(
            D=>RD_ID,
            Q=>RD_EXE,
            EN=>control_from_HDU.ID_EXE_EN,
            CLK=>clk,
            RST=>rst
        );

    RF_instance: RF
        generic map(
            N_bit=>WORD_SIZE,
            N_reg=>N_REG
        )
        port map(
            CLK=>clk,
            RST=>rst,
            WR_EN=>control_from_CU.WB.RF_WR_EN,
            ADDR_WR=>RD_WB,
            ADDR_RD2=>IR_out(RS2_range),
            ADDR_RD1=>IR_out(RS1_range),
            DATA_IN=>RF_IN_WB,
            OUT1=>RF_OUT1,
            OUT2=>RF_OUT2
        );

    REG_ALU_IN1: reg
        generic map(N=>WORD_SIZE)
        port map(
            D=>ALU_IN1_ID,
            Q=>ALU_IN1_EXE,
            EN=>control_from_HDU.ID_EXE_EN,
            CLK=>clk,
            RST=>rst
        );

    REG_ALU_IN2: reg
        generic map(N=>WORD_SIZE)
        port map(
            D=>ALU_IN2_ID_fw,
            Q=>ALU_IN2_EXE_RF,
            EN=>control_from_HDU.ID_EXE_EN,
            CLK=>clk,
            RST=>rst
        );

    IMM_ITYPE<=std_logic_vector(to_signed(get_IMM_ITYPE(IR_out), WORD_SIZE));
    IMM_STYPE<=std_logic_vector(to_signed(get_IMM_STYPE(IR_out), WORD_SIZE));
    IMM_BTYPE<=std_logic_vector(to_signed(get_IMM_BTYPE(IR_out), WORD_SIZE));
    IMM_JTYPE<=std_logic_vector(to_signed(get_IMM_JTYPE(IR_out), WORD_SIZE));
    IMM_UTYPE<=std_logic_vector(to_signed(get_IMM_UTYPE(IR_out), WORD_SIZE));

    MUX_IMM: MUX_8to1
        generic map(N=>WORD_SIZE)
        port map(
            IN0=>IMM_ITYPE,
            IN1=>IMM_STYPE,
            IN2=>IMM_BTYPE,
            IN3=>IMM_JTYPE,
            IN4=>IMM_UTYPE,
            IN5=>(others=>'-'),
            IN6=>(others=>'-'),
            IN7=>(others=>'-'),
            SEL=>control_from_CU.ID.MUX_IMM_sel,
            Y=>IMM_ID
        );

    REG_IMM: reg
        generic map(N=>WORD_SIZE)
        port map(
            D=>IMM_ID,
            Q=>IMM_EXE,
            EN=>control_from_HDU.ID_EXE_EN,
            CLK=>clk,
            RST=>rst
        );

    ---------------------------------------EXE stage

    MUX_DRAM_IN_fw: MUX_2to1
        generic map(N=>WORD_SIZE)
        port map(
            IN0=>ALU_IN2_EXE_RF,
            IN1=>RF_IN_MEM,
            SEL=>control_from_FU.MUX_DRAM_IN_sel,
            Y=>DRAM_IN_EXE
        );

    MUX_ALU_IN2: MUX_2to1
        generic map(N=>WORD_SIZE)
        port map(
            IN0=>ALU_IN2_EXE_RF,
            IN1=>IMM_EXE,
            SEL=>control_from_CU.EXE.MUX_ALU_IN2_sel,
            Y=>ALU_IN2_EXE
        );

    ALU_instance: ALU
        generic map(N=>WORD_SIZE)
        port map(
            FUNC=>control_from_CU.EXE.ALU_OP,
            DATA1=>ALU_IN1_EXE,
            DATA2=>ALU_IN2_EXE,
            OUT_ALU=>ALU_OUT_EXE
        );


    REG_ALU_OUT_EXE_MEM: reg
        generic map(N=>WORD_SIZE)
        port map(
            D=>ALU_OUT_EXE,
            Q=>ALU_OUT_MEM,
            EN=>control_from_HDU.EXE_MEM_EN,
            CLK=>clk,
            RST=>rst
        );

    REG_DRAM_IN: reg
        generic map(N=>WORD_SIZE)
        port map(
            D=>DRAM_IN_EXE,
            Q=>DRAM_IN_MEM,
            EN=>control_from_HDU.EXE_MEM_EN,
            CLK=>clk,
            RST=>rst
        );

    REG_RD_EXE_MEM: reg
        generic map(N=>REG_ADDR_SIZE)
        port map(
            D=>RD_EXE,
            Q=>RD_MEM,
            EN=>control_from_HDU.EXE_MEM_EN,
            CLK=>clk,
            RST=>rst
        );

    ----------------------------------------------------------------MEM stage

    DRAM_ADDR<=ALU_OUT_MEM;
    DRAM_IN<=DRAM_IN_MEM;

    DRAM_OUT_sb<=std_logic_vector(resize(signed(DRAM_OUT(byte_range)), WORD_SIZE));
    DRAM_OUT_ub<=std_logic_vector(resize(unsigned(DRAM_OUT(byte_range)), WORD_SIZE));
    DRAM_OUT_sh<=std_logic_vector(resize(signed(DRAM_OUT(halfword_range)), WORD_SIZE));
    DRAM_OUT_uh<=std_logic_vector(resize(unsigned(DRAM_OUT(halfword_range)), WORD_SIZE));
    DRAM_OUT_w<=DRAM_OUT;

    MUX_DRAM: MUX_8to1
        generic map(N=>WORD_SIZE)
        port map(
            IN0=>DRAM_OUT_sb,
            IN1=>DRAM_OUT_ub,
            IN2=>DRAM_OUT_sh,
            IN3=>DRAM_OUT_uh,
            IN4=>DRAM_OUT_w,
            IN5=>(others=>'-'),
            IN6=>(others=>'-'),
            IN7=>(others=>'-'),
            SEL=>control_from_CU.MEM.MUX_DRAM_sel,
            Y=>DRAM_OUT_ext
        );

    MUX_RF_IN: MUX_2to1
        generic map(N=>WORD_SIZE)
        port map(
            IN0=>ALU_OUT_MEM,
            IN1=>DRAM_OUT_ext,
            SEL=>control_from_CU.MEM.MUX_RF_IN_sel,
            Y=>RF_IN_MEM
        );

    REG_RF_IN: reg
        generic map(N=>WORD_SIZE)
        port map(
            D=>RF_IN_MEM,
            Q=>RF_IN_WB,
            EN=>control_from_HDU.MEM_WB_EN,
            CLK=>clk,
            RST=>rst
        );

    REG_RD_MEM_WB: reg
        generic map(N=>REG_ADDR_SIZE)
        port map(
            D=>RD_MEM,
            Q=>RD_WB,
            EN=>control_from_HDU.MEM_WB_EN,
            CLK=>clk,
            RST=>rst
        );

    ------------------------------------------------------------WB stage


end architecture structural;
```

Listing B.13: RF.vhd

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use work.my_package.all;    --for log2_ceiling

entity RF is
    generic(N_bit:      positive := 64;     --bitwidth
            N_reg:      positive := 32);    --number of registers
    port(   CLK:        IN std_logic;       --clock
            RST:        IN std_logic;       --asynchronous reset, active low
            WR_EN:      IN std_logic;       --synchronous write, active high
            ADDR_WR:    IN std_logic_vector(log2_ceiling(N_reg)-1 downto 0);    --writing register address
            ADDR_RD1:   IN std_logic_vector(log2_ceiling(N_reg)-1 downto 0);    --reading register address 1
            ADDR_RD2:   IN std_logic_vector(log2_ceiling(N_reg)-1 downto 0);    --reading register address 2
            DATA_IN:    IN std_logic_vector(N_bit-1 downto 0);      --data to write
            OUT1:       OUT std_logic_vector(N_bit-1 downto 0);     --data to read 1
            OUT2:       OUT std_logic_vector(N_bit-1 downto 0));    --data to read 2
end entity RF;

architecture behavioral of RF is

    type    REG_ARRAY_TYPE is array(1 to N_reg-1) of std_logic_vector(N_bit-1 downto 0);    --array of registers type
    signal  REGISTERS : REG_ARRAY_TYPE;                                                    --registers instantiation

    signal ADDR_RD1_int, ADDR_RD2_int, ADDR_WR_int : natural;

begin

    ADDR_RD1_int<=to_integer(unsigned(ADDR_RD1));
    ADDR_RD2_int<=to_integer(unsigned(ADDR_RD2));
    ADDR_WR_int<=to_integer(unsigned(ADDR_WR));

    OUT1 <= REGISTERS(ADDR_RD1_int) when ADDR_RD1_int<=N_reg-1 and ADDR_RD1_int>=1 else
            (others=>'0');

    OUT2 <= REGISTERS(ADDR_RD2_int) when ADDR_RD2_int<=N_reg-1 and ADDR_RD2_int>=1 else
            (others=>'0');

    register_file_proc: process(CLK, RST)
    begin
        if RST='0' then
            REGISTERS <= (others=>(others=>'0'));
        elsif rising_edge(CLK) then
            if WR_EN = '1' then     --if write signal is active
                if (ADDR_WR_int<=N_reg-1 and ADDR_WR_int>=1) then   --and the write address is valid
                    REGISTERS(ADDR_WR_int) <= DATA_IN;  --write register pointed by ADDR_WR with the value contained in DATAIN
                end if;
            end if;
        end if;
    end process;

end architecture behavioral;
```

Listing B.14: branch_comp.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.CU_package.all;

entity branch_comp is
    generic(N: integer:= 32);   --number of data-in bits
    port(
        BRANCH_COND     : in BRANCH_COND_type;                  --condition to take branch
        DATA_IN1        : in std_logic_vector(N-1 downto 0);    --data to test
        DATA_IN2        : in std_logic_vector(N-1 downto 0);    --data to test
        BRANCH_IS_TAKEN : out std_logic);                       --high if the branch is taken
end entity branch_comp;

architecture behavioral of branch_comp is
begin

    branch_comparator_process: process(BRANCH_COND, DATA_IN1, DATA_IN2)
    begin
        case BRANCH_COND is
            when BRANCH_COND_NO =>
                BRANCH_IS_TAKEN<='0';
            when BRANCH_COND_ALWAYS =>
                BRANCH_IS_TAKEN<='1';
            when BRANCH_COND_EQ =>
                if DATA_IN1=DATA_IN2 then
                    BRANCH_IS_TAKEN<='1';
                else
                    BRANCH_IS_TAKEN<='0';
                end if;
            when BRANCH_COND_NE =>
                if DATA_IN1/=DATA_IN2 then
                    BRANCH_IS_TAKEN<='1';
                else
                    BRANCH_IS_TAKEN<='0';
                end if;
            when BRANCH_COND_LT =>
                if signed(DATA_IN1)<signed(DATA_IN2) then
                    BRANCH_IS_TAKEN<='1';
                else
                    BRANCH_IS_TAKEN<='0';
                end if;
            when BRANCH_COND_GE =>
                if signed(DATA_IN1)>=signed(DATA_IN2) then
                    BRANCH_IS_TAKEN<='1';
                else
                    BRANCH_IS_TAKEN<='0';
                end if;
            when BRANCH_COND_LTU =>
                if unsigned(DATA_IN1)<unsigned(DATA_IN2) then
                    BRANCH_IS_TAKEN<='1';
                else
                    BRANCH_IS_TAKEN<='0';
                end if;
            when BRANCH_COND_GEU =>
                if unsigned(DATA_IN1)>=unsigned(DATA_IN2) then
                    BRANCH_IS_TAKEN<='1';
                else
                    BRANCH_IS_TAKEN<='0';
                end if;
        end case;
    end process branch_comparator_process;

end architecture behavioral;
```

Listing B.15: IR.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.RISCV_package.all;

entity IR is
    port(
        IR_IN   : In    word;           --instruction register input
        IR_OUT  : Out   word;           --instruction register output
        EN      : In    std_logic;      --enable active high
        BUBBLE  : in    std_logic;      --if high a bubble is introduced
        CLK     : In    std_logic;      --clock
        RST     : In    std_logic       --asynchronous reset active low
    );
end entity IR;

architecture behavioral of IR is

begin

    IR_proc: process(CLK, RST)
    begin
        if RST='0' then
            IR_OUT<=(others=>'0');
        elsif rising_edge(CLK) then
            if BUBBLE='1' then
                IR_OUT<=INSTR_NOP;
            elsif EN='1' then
                IR_OUT<=IR_IN;
            end if;
        end if;
    end process IR_proc;

end behavioral;
```

Listing B.16: PC.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use work.RISCV_package.all;

entity PC is
    port(
        PC_IN   : In    word;           --new program count
        PC_OUT  : Out   word;           --current program count
        EN      : In    std_logic;      --enable active high
        CLK     : In    std_logic;      --clock
        RST     : In    std_logic       --asynchronous reset active low
    );
end entity PC;

architecture behavioral of PC is

begin
    PC_proc: process(CLK, RST)          --asynchronous reset
    begin
        if RST='0' then                 --if reset is active
            PC_OUT<=X"00400000";        --initialize de PC to the first instruction
        elsif rising_edge(CLK) then     --otherwise if there is a positive clock edge
            if EN='1' then              --and enable is active
                PC_OUT <= PC_IN;        --writes the input on the output
            end if;
        end if;
    end process;

end behavioral;
```

Listing B.17: ALU.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
use work.ALU_package.all;
use work.my_package.all;

entity ALU is
    generic (N : integer := 32);                                --number of bits
    port(   FUNC            : in ALU_OP_type;                   --operation to do
            DATA1, DATA2    : in std_logic_vector(N-1 downto 0);        --data inputs
            OUT_ALU         : out std_logic_vector(N-1 downto 0));      --data output
end entity ALU;

architecture behavioral of ALU is

    component barrel_shifter is
        port(
            OPERAND     :   IN std_logic_vector(31 downto 0);       -- operand that we want shift
            SHIFT_AMOUNT:   IN std_logic_vector(4 downto 0);        -- number of position to shift
            LOGIC_ARITH :   IN std_logic;                           -- logic 0 arith 1
            LEFT_RIGHT  :   IN std_logic;                           -- left 0 right 1
            OUTPUT      :   OUT std_logic_vector(31 downto 0)       -- result
        );
    end component barrel_shifter;

    component adder_subtractor is
        generic(N_BIT   : integer := 32);                           --number of bits
        port(
            A       : in  std_logic_vector(N_BIT - 1 downto 0);     --input operand 1
            B       : in  std_logic_vector(N_BIT - 1 downto 0);     --input operand 2
            add_sub : in  std_logic;                                --addition 0 subtraction 1
            Cout    : out std_logic;                                --carry-out
            sum     : out std_logic_vector(N_BIT -1 downto 0)       --sum
        );
    end component adder_subtractor;

    component logic_block is
        generic (N: integer:= 32);                                  --number of bits
        port(
            A           :   In  std_logic_vector(N-1 downto 0);     --operand 1
            B           :   In  std_logic_vector(N-1 downto 0);     --operand 2
            A_AND_B     :   Out std_logic_vector(N-1 downto 0);     --and
            A_XOR_B     :   Out std_logic_vector(N-1 downto 0);     --xor
            A_OR_B      :   Out std_logic_vector(N-1 downto 0)      --or
        );
    end component logic_block;

    component comparator is
        generic (N: integer:= 32);                                  --number of bits
        port(
            SUB         :   In  std_logic_vector(N-1 downto 0);     --result of the subtraction between the two operands
            CARRY       :   In  std_logic;                          --carry out of the subtraction between the two operands
            A_LT_B      :   Out std_logic_vector(N-1 downto 0);     --less than (signed)
            A_LTU_B     :   Out std_logic_vector(N-1 downto 0)      --less than (unsigned)
        );
    end component comparator;

    --first five bits of second operand
    signal SHIFT_AMOUNT_sig: std_logic_vector(4 downto 0);

    --signals to select the shift type
    signal logic_arith, left_right  : std_logic;

    --signal to select addition or subtraction
    signal add_sub      : std_logic;

    --adder_subtractor carry_out
    signal carry_out                        : std_logic;
    --functional units outputs
    signal out_adder_subtractor             : std_logic_vector(N-1 downto 0);
    signal out_barrel_shifter               : std_logic_vector(N-1 downto 0);
    signal out_and,out_or,out_xor           : std_logic_vector(N-1 downto 0);
    signal out_slt,out_sltu                 : std_logic_vector(N-1 downto 0);

    --inputs of the adder/subtractor
    signal in1_addsub, in2_addsub           : std_logic_vector(N-1 downto 0);

    begin

        SHIFT_AMOUNT_sig<=DATA2(4 downto 0);

        barrel_shifter_instance : barrel_shifter
            port map(
                LOGIC_ARITH=>logic_arith,
                LEFT_RIGHT=>left_right,
                OPERAND=> DATA1,
                SHIFT_AMOUNT=>SHIFT_AMOUNT_sig,
                OUTPUT=> out_barrel_shifter
            );

        adder_subtractor_instance: adder_subtractor
            generic map(N)
            port map(
                A=>in1_addsub,
                B=>in2_addsub,
                add_sub=>add_sub,
                Cout=>carry_out,
                SUM=>out_adder_subtractor
            );

        logic_block_instance: logic_block
            generic map(N)
            port map(
                A=>DATA1,
                B=>DATA2,
                A_AND_B=>out_and,
                A_XOR_B=>out_xor,
                A_OR_B=>out_or
            );

        comparator_instance: comparator
            generic map(N)
            port map(
                sub=>out_adder_subtractor,
                carry=>carry_out,
                A_LT_B=>out_slt,
                A_LTU_B=>out_sltu
            );

        func_process: process(FUNC, out_adder_subtractor, out_barrel_shifter, out_and, out_or, out_xor, out_slt, out_sltu, DATA1, DATA2)
        begin
            --default assignments
            OUT_ALU<=(others=>'-');
            add_sub<='-';
            left_right<='-';
            logic_arith<='-';
            in1_addsub<=DATA1;
            in2_addsub<=DATA2;
            if FUNC=ALU_IN1 then
                OUT_ALU<=DATA1;
            elsif FUNC=ALU_IN2 then
                OUT_ALU<=DATA2;
            elsif FUNC=ALU_ADD then
                OUT_ALU<=out_adder_subtractor;
                add_sub<='0';
            elsif FUNC=ALU_SUB then
                OUT_ALU<=out_adder_subtractor;
                add_sub<='1';
            elsif FUNC=ALU_XOR then
                OUT_ALU<=out_xor;
            elsif FUNC=ALU_AND then
                OUT_ALU<=out_and;
            elsif FUNC=ALU_OR then
                OUT_ALU<=out_or;
            elsif FUNC=ALU_SL then
                OUT_ALU<=out_barrel_shifter;
                left_right<='0';
            elsif FUNC=ALU_SRL then
                OUT_ALU<=out_barrel_shifter;
                left_right<='1';
                logic_arith<='0';
            elsif FUNC=ALU_SRA then
                OUT_ALU<=out_barrel_shifter;
                left_right<='1';
                logic_arith<='1';
            elsif FUNC=ALU_SLT then
                OUT_ALU<=out_slt;
                add_sub<='1';
            elsif FUNC=ALU_SLTU then
                OUT_ALU<=out_sltu;
                add_sub<='1';
            elsif FUNC=ALU_ABS then
                if DATA1(N-1)='0' then
                    OUT_ALU<=DATA1;
                else
                    OUT_ALU<=out_adder_subtractor;
                    add_sub<='1';
                    in1_addsub<=(others=>'0');
                    in2_addsub<=DATA1;
                end if;
            end if;
        end process;

end architecture behavioral;
```

Listing B.18: barrel_shifter.vhd

```
library IEEE;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity barrel_shifter is
port    (   OPERAND      :   IN std_logic_vector(31 downto 0);       -- operand that we want shift
            SHIFT_AMOUNT:   IN std_logic_vector(4 downto 0);        -- number of position to shift
            LOGIC_ARITH :   IN std_logic;                            -- logic 0 arith 1
            LEFT_RIGHT  :   IN std_logic;                            -- left 0 right 1
            OUTPUT      :   OUT std_logic_vector(31 downto 0));      -- result
end  barrel_shifter;


architecture structural of  barrel_shifter is

    component MUX_2to1 is
        Generic (N: integer:= 1);                               --number of bits
        Port (  IN0:    In  std_logic_vector(N-1 downto 0);     --data input 1
                IN1:    In  std_logic_vector(N-1 downto 0);     --data input 2
                SEL:    In  std_logic;                          --selection input
                Y:      Out std_logic_vector(N-1 downto 0));    --data output
    end component MUX_2to1;

    component MUX_8to1 is
        Generic (N: integer:= 1);   --number of bits
        Port (  IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7  : In    std_logic_vector(N-1 downto 0);     --data inputs
                SEL                 : In    std_logic_vector(2 downto 0);                           --selection input
                Y                   : Out   std_logic_vector(N-1 downto 0));                        --data output
    end component MUX_8to1;

    constant Nbit       : positive  := 32;
    type matrix is array (Nbit/4 downto 1) of std_logic_vector(Nbit+3 downto 0);
    signal out_left, out_right, out_mux_right, out_first_stage: matrix ;

    -- oversized signal to manage the mask creation
    signal zero_vector, MSB_vector : std_logic_vector(Nbit+3 downto 0);

    -- signal used to propagate result from stage two to stage three
    signal out_second_stage : std_logic_vector(Nbit+3 downto 0);

    -- signal used to manage the mux of the third stage
    signal sel_mux_out              : std_logic;
    signal select_mux_3s            :std_logic_vector(2 downto 0);

    -- it holds only five bits of the SHIFT_AMOUNT input signal because on 32 bit, it is possible has a max shift of 32 bit (log2(32) =
          ↪ 5)
    signal shift_pos                : std_logic_vector ( 4 downto 0);
BEGIN
    -- create a vector of all 0
    zero_vector <= (others => '0');

    -- simple assignement to manage the shift position
    shift_pos <= SHIFT_AMOUNT;
    -- it generates a signal with all the bits equal to the MSB. It is used in the arithmetical shift
    MSB_fill: for x in 0 to Nbit+3 generate
    MSB_vector (x)<= OPERAND(31);
    end generate MSB_fill;

    -- FIRST STAGE -------------------------------------------------------------------------------------------------------
    -- create the eight possible general mask for left shift
    -- mask 0 : OPERAND(31 downto 0) & "0000"
    -- mask 4 : OPERAND(27 downto 0) & "00000000"
    -- ...other mask...
    -- mask 28: OPERAND(3 downto 0) & "00000000000000000000000000000000"
    FIRST_STAGE_1: for x in 1 to Nbit/4 generate
        out_left(x) <= (OPERAND(35-4*x downto 0) & zero_vector(4*x -1 downto 0));
    end generate FIRST_STAGE_1;

    -- it select if fill with the MSB or with the 0 based on if it is a arithmetic or a logical shift
    FIRST_STAGE_2: for x in 1 to Nbit/4 generate
        right_MUX2to1 : MUX_2to1
                        Generic map(N => 4*x)
                        Port map(IN0=> (others=>'0'),IN1=> MSB_vector(4*x-1 downto 0),SEL=> LOGIC_ARITH ,Y=> out_mux_right(x)(4*x-1
                              ↪ downto 0));
    end generate FIRST_STAGE_2;

    --  create the eight possible general mask for right shift
    -- if LOGIC shift:
    -- mask 0 : "0000" & OPERAND(31 downto 0)
    -- mask 4 : "00000000" & OPERAND(31 downto 4)
    -- ...other mask...
    -- mask 28: "00000000000000000000000000000000" & OPERAND(31 downto 28)
    -- if ARITHMETIC shift:
    -- mask 0 : "1111" & OPERAND(31 downto 0)
    -- mask 4 : "11111111" & OPERAND(31 downto 4)
    -- ...other mask...
    -- mask 28: "11111111111111111111111111111111" & OPERAND(31 downto 28)
    FIRST_STAGE_3: for x in 1 to Nbit/4 generate
        out_right(x) <= (out_mux_right(x)(4*x-1 downto 0) & OPERAND(Nbit-1 downto 4*x-4));
    end generate FIRST_STAGE_3;

    -- it select if it is a mask for right shift or a mask for left shift
    FIRST_STAGE_4: for x in 1 to Nbit/4 generate
        MASK_MUX2to1 : MUX_2to1
                    Generic map(N => Nbit + 4)
                    Port map(IN0=> out_left(x) ,IN1=> out_right(x) ,SEL=> LEFT_RIGHT ,Y=>out_first_stage(x));
    end generate FIRST_STAGE_4;

    -- SECOND STAGE -------------------------------------------------------------------------------------
    -- select the best mask approximation based on the 3 upper bits of shift_pos
    mux_second_stage: MUX_8to1
                    Generic map(N => Nbit + 4)
                    Port map(IN0=>out_first_stage(1),IN1=>out_first_stage(2),IN2=>out_first_stage(3),
                    IN3=>out_first_stage(4),IN4=>out_first_stage(5),IN5=>out_first_stage(6),
                    IN6=>out_first_stage(7),IN7=>out_first_stage(8),SEL=>shift_pos(4 downto 2),Y=> out_second_stage);

    -- THIRD STAGE --------------------------------------------------------------------------------------
    -- It select the right mask based on the last two bit of shift_pos and on the LEFT_RIGHT signal.
    -- if LEFT_RIGHT = LEFT it select one mask between input OPERAND and D
    -- if LEFT_RIGHT = RIGHT it select one mask between input E and H
    -- The different entries cut the input signal in order to provide the correct shifting
    select_mux_3s <= (LEFT_RIGHT & shift_pos(1) & shift_pos(0));
    mux_third_stage: MUX_8to1
                    Generic map(N => Nbit)
                    Port map(IN0=>out_second_stage(Nbit+3 downto 4),IN1=>out_second_stage(Nbit+2 downto 3),
                    IN2=>out_second_stage(Nbit+1 downto 2),IN3=>out_second_stage(Nbit downto 1),
                    IN4=>out_second_stage(Nbit -1 downto 0),IN5=>out_second_stage(Nbit downto 1),
                    IN6=>out_second_stage(Nbit+1 downto 2),IN7=>out_second_stage(Nbit+2 downto 3),
                    SEL=>select_mux_3s,Y=> OUTPUT);

end structural;
```

Listing B.19: logic_block.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;
use work.my_package.all;
```

```
entity logic_block is
    generic (N: integer:= 32);                              --number of bits
    port(
        A           :   In  std_logic_vector(N-1 downto 0);      --operand 1
        B           :   In  std_logic_vector(N-1 downto 0);      --operand 2
        A_AND_B     :   Out std_logic_vector(N-1 downto 0);      --and
        A_XOR_B     :   Out std_logic_vector(N-1 downto 0);      --xor
        A_OR_B      :   Out std_logic_vector(N-1 downto 0)       --or
    );
end entity logic_block;

architecture structural of logic_block is
begin

    A_AND_B <= A and B;
    A_XOR_B <= A xor B;
    A_OR_B  <= A or  B;

end architecture structural;
```

Listing B.20: comparator.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use ieee.numeric_std.all;

entity comparator is
    generic (N: integer:= 32);                                  --number of bits
    port(
        SUB         : in    std_logic_vector(N-1 downto 0);     --result of the subtraction between the two operands
        CARRY       : in    std_logic;                          --carry out of the subtraction between the two operands
        A_LT_B      : out   std_logic_vector(N-1 downto 0);     --less than (signed)
        A_LTU_B     : out   std_logic_vector(N-1 downto 0)      --less than (unsigned)
    );
end comparator;

architecture structural of comparator is

    signal lt, ltu  :std_logic;

begin

    lt<=SUB(N-1);
    ltu<=not CARRY;

    A_LT_B(0)<=lt;
    A_LT_B(N-1 downto 1)<=(others=>'0');

    A_LTU_B(0)<=ltu;
    A_LTU_B(N-1 downto 1)<=(others=>'0');

end architecture structural;
```

Listing B.21: adder_subtractor.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use work.my_package.all;

entity  adder_subtractor IS
    generic (N_BIT  : integer := 32);   --number of bits, must be a number from 4 to 32 and a multiple of 4
    port(
        A       : in  std_logic_vector(N_BIT - 1 downto 0);     --input operand 1
        B       : in  std_logic_vector(N_BIT - 1 downto 0);     --input operand 2
        add_sub : in  std_logic;                                --addition 1 subtraction 0
        Cout    : out std_logic;                                --carry-out
        sum     : out std_logic_vector(N_BIT -1 downto 0)       --ouput sum
    );
end entity adder_subtractor;

architecture structural of adder_subtractor is

    component sum_generator is
        generic(
            Nblocks         : positive := 8;    --number of carry select block
            bits_per_block  : positive := 4     --number of bit per each block
        );                                      --the number of input bits is equal to Nblocks*bits_per_block
        port(
            A               : in  std_logic_vector(bits_per_block*Nblocks - 1 downto 0);    --data input 1
            B               : in  std_logic_vector(bits_per_block*Nblocks - 1 downto 0);    --data input 2
            carry_select    : in  std_logic_vector(Nblocks-1 downto 0);                     --block carry-in
            sum             : out std_logic_vector(bits_per_block*Nblocks - 1 downto 0)     --block sum
        );
    end component;
    component carry_generator is
        generic(Nbit        : positive := 32);                  --number of bits of the sparse tree, must be between 4 and 32 and a
               ↪ multiple of 4
        port(
            A       : in  std_logic_vector(Nbit - 1 downto 0);  --input operand 1
            B       : in  std_logic_vector(Nbit - 1 downto 0);  --input operand 2
            Cin     : in  std_logic;                            --carry-in
            Cout    : out std_logic_vector(Nbit/4 downto 0)     --carry-out
        );
    end component;

    signal carries  : std_logic_vector(N_BIT/4  downto 0);  --carries between the two modules
    signal B_xor    : std_logic_vector(N_BIT - 1 downto 0); --B xor add_sub
begin

    carry_generator_instance: carry_generator
        generic map (Nbit => N_BIT)
        port map  (A => A,  B => B_xor, Cin => add_sub, Cout => carries);

    sum_generator_instance: sum_generator
        generic map(Nblocks=> N_BIT/4, bits_per_block => 4)
        port map(A => A, B => B_xor , carry_select => carries (N_BIT/4-1 downto 0), sum => sum);

    --the last carry is the carry-out of the entire structure
    Cout <= carries(N_BIT/4);

    --if add_sub=1, B is complemented and the carry-in of the adder becomes 1
    B_xor <= B xor add_sub;

end structural;
```

Listing B.22: sum_generator.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity  sum_generator is
    generic(Nblocks         : positive := 8;    --number of carry select block
            bits_per_block  : positive := 4);   --number of bit per each block
                                                --the number of input bits is equal to Nblocks*bits_per_block
    port (
      A             : in  std_logic_vector(bits_per_block*Nblocks - 1 downto 0);    --data input 1
      B             : in  std_logic_vector(bits_per_block*Nblocks - 1 downto 0);    --data input 2
      carry_select  : in  std_logic_vector(Nblocks-1 downto 0);                     --carries from sparse tree
      SUM           : out std_logic_vector(bits_per_block*Nblocks - 1 downto 0));   --data output
end entity sum_generator;

architecture structural of sum_generator is

    component carry_select_block is
        generic(N: positive := 4);                                --number of bits of the block
        port(   A, B    :   in std_logic_vector(N-1 downto 0);    --data inputs
            S       :   out std_logic_vector(N-1 downto 0);       --data output
            Ci      :   in std_logic                              --block carry-in
        );
    end component;

begin

    gen_block: for i in 1 to Nblocks generate   --generate "Nblocks" carry select blocks
    block_n : carry_select_block
        generic map (N =>bits_per_block)        --each with "bits_per_block" bits
        port map(   A=> A((bits_per_block*i)-1 downto (i-1)*bits_per_block),        --split input A in "Nblocks" sub-signal each of "
               ↪ bits_per_block" bits and connect each sub-signal to each block input A
                    B=> B((bits_per_block*i)-1 downto (i-1)*bits_per_block),        --idem for input B
                    S=> SUM((bits_per_block*i)-1 downto (i-1)*bits_per_block),      --put together all the outputs of the singles blocks
                           ↪  to the output of the sum_generator
                    Ci => CARRY_SELECT(i-1));                                       --connect all the input carries from the carry
                           ↪ generator to the carry select of each blocks
    end generate gen_block;

end structural;
```

Listing B.23: carry_generator.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;
use ieee.math_real.all;
use ieee.numeric_std.all;

-- CLA sparse tree module used in Pentium 4
entity  carry_generator is
    generic(Nbit     : positive := 32);                         --number of bits of the sparse tree, must be between 4 and 32 and a
           ↪ multiple of 4
    port(
        A            : IN  std_logic_vector(Nbit - 1 downto 0);     --input operand 1
        B            : IN  std_logic_vector(Nbit - 1 downto 0);     --input operand 2
        Cin          : IN  std_logic;                                --carry-in
        Cout         : OUT std_logic_vector(Nbit/4 downto 0)        --carry-out
    );
end entity carry_generator;

architecture structural of carry_generator is
    --propagate and generate network
    component PG_NETWORK IS
        PORT (
            op1              : in  std_logic;       --input bit 1
            op2              : in  std_logic;       --input bit 2
            g,p              : out std_logic);  --output propagate and generate
    end component;

    --general propagate block
    component PG_BLOCK IS
        PORT (
            P_ik , G_ik              : in  std_logic;
            P_k1j,G_k1j              : in  std_logic;
            P_ij, G_ij           : out std_logic);
    end component;

    --general generate block
    component G_BLOCK IS
        PORT (
            P_ik , G_ik              : in  std_logic;
            G_k1j                    : in  std_logic;
            G_ij                     : out std_logic);
    end component;

    -- We create two matrix, one for general propagate(p) and one for general generate(g)
    -- in this way, we have used the column to select the stage, and the row to select the product
    -- for example: g(0)(1) identify the general generate of the first element(1) provided by pg network(0)
    type matrix is array (4 downto 0) of std_logic_vector(Nbit downto 0);
    signal g, p : matrix ;

    -- temporal signal (tmp_p and tmp_g) are used only to semplify the generation of the last stage
    signal tmp_p,tmp_g : std_logic_vector(Nbit/4 -1 downto 0);

    -- This signal make easier the last stage implementation
    signal reference_gij : std_logic_vector((Nbit/4)-4 downto 0);

    -- co_tmp it is used to keep the temporal carry of the system
    signal co_tmp         : std_logic_vector(Nbit/4 - 1 downto 0);

    -- G_10 it is used only in case of carry in = 1. It modify the value provide to G block on first stage, after the pg network
    signal G_10: std_logic;


BEGIN
    -- particular case assignement
    p(0)(0) <= '0';
    g(0)(0) <= Cin;

    -- Create the pg network for the required number of bit. It is consider as stage 0, so the index of the column is 0.
    -- Stage 0 ---------------------------------------------------------------------------------------------------------
    generation_PG_Network: for x in 1 to Nbit generate
        Block_PG_NET : PG_NETWORK
        PORT MAP(op1 => A(x-1), op2=>B(x-1), g=> g(0)(x), p=>p(0)(x));
    end generate generation_PG_Network;

    -- in case of carry in, g(0)(1) must be different (in lecture it is the value identify by G [1:0]), so:
    G_10 <= g(0)(1) or (p(0)(1) and g(0)(0));

    -- First stage -----------------------------------------------------------------------------------------------------
    g_1 : G_BLOCK PORT MAP(P_ik => p(0)(2), G_ik =>g(0)(2) ,G_k1j => G_10, G_ij => g(1)(0));

    First_stage: for x in 1 to (Nbit-2)/2 generate
        Block_Stage_ONE : PG_BLOCK
        PORT MAP(P_ik => p(0)(x*2+2), G_ik =>g(0)(x*2+2) , P_k1j=>p(0)(x*2+1) , G_k1j =>g(0)(x*2+1) , P_ij => p(1)(x), G_ij =>g(1)(x) );
    end generate First_stage;

    -- Second stage ----------------------------------------------------------------------------------------------------
    g_2 : G_BLOCK PORT MAP(P_ik => p(1)(1), G_ik =>g(1)(1) ,G_k1j => g(1)(0), G_ij => co_tmp(0));

    Second_stage: for x in 1 to Nbit/4 -1 generate
        Block_Stage_TWO : PG_BLOCK
        PORT MAP( P_ik =>p(1)(x*2+1) , G_ik => g(1)(x*2+1), P_k1j=> p(1)(x*2), G_k1j =>g(1)(x*2) , P_ij =>p(2)(x), G_ij => g(2)(x));
    end generate Second_stage;

    -- Third Stage -----------------------------------------------------------------------------------------------------
    if4: if Nbit>4 generate
        g_3 : G_BLOCK PORT MAP(P_ik =>p(2)(1), G_ik =>g(2)(1) ,G_k1j =>co_tmp(0), G_ij =>co_tmp(1));

        Third_stage: for x in 1 to (Nbit/8 -1 )generate
            Block_Stage_THREE : PG_BLOCK
            PORT MAP( P_ik => p(2)(x*2+1), G_ik =>  g(2)(x*2+1), P_k1j=>  p(2)(x*2), G_k1j =>  g(2)(x*2), P_ij =>p(3)(x), G_ij =>g(3)(x)
                   ↪  );
        end generate Third_stage;

    end generate if4;

    -- Fourth Stage ----------------------------------------------------------------------------------------------------
    if8: if Nbit> 8 generate
        stage_c12_c16: for x in 0 to 1 generate
            g_4_c12_c16: G_BLOCK PORT MAP(P_ik =>p(2+x)(2-x), G_ik =>g(2+x)(2-x) ,G_k1j =>co_tmp(1), G_ij =>co_tmp(2+x));
        end generate stage_c12_c16;
    end generate if8;

    if16_1: if Nbit>= 16 generate
        loopK: for k in 2 to Nbit/16 generate
            Fourth_stage : for x in 0 to 1 generate
                Block_stage_FOUR : PG_BLOCK PORT MAP( P_ik =>p(x+2)((-2+4*k)/(x+1)), G_ik =>g(x+2)((-2+4*k)/(x+1)) , P_k1j=>p(3)(2*(k-1)
                        ↪ ), G_k1j => g(3)(2*(k-1)), P_ij =>p(4)(x+2*k-2), G_ij =>g(4)(x+2*k-2) );
            end generate Fourth_stage;
        end generate loopK;
    end generate if16_1;
    ----------------------------------------------------------------------------------------------------------------------------------
    -- Fifth stage.
    -- This are used to create a order sequence of p and g operand, so the last stage is simpler
    if16_2: if (Nbit>16) generate
        create_vector: for x in 1 to integer(log2(real(Nbit)))-4 generate
            tmp_p (4*x-1 downto 4*x-4) <=  ( p(4)(2*x+1) & p(4)(2*x) & p(3)(2*x) & p(2)(4*x));
            tmp_g (4*x-1 downto 4*x-4) <=  ( g(4)(2*x+1) & g(4)(2*x) & g(3)(2*x) & g(2)(4*x));
        end generate create_vector;
    end generate if16_2;

    -- The two following if, are used to create the PG block that preceding the G block that generate the carry52,carry56,carry60 and
        ↪ carry64
    if64_1: if Nbit>32 generate
        Stage_52_56 : for x in 0 to 1 generate
            Block_carry52_carry56 : PG_BLOCK PORT MAP( P_ik =>p(2+x)(12/(x+1)), G_ik =>g(2+x)(12/(x+1)) , P_k1j=>p(4)(5), G_k1j => g(4)
                ↪ (5), P_ij =>tmp_p(8+x), G_ij =>tmp_g(8+x) );
        end generate Stage_52_56;
    end generate if64_1;

    if64_2: if Nbit>32 generate
        Stage_60_64 : for x in 0 to 1 generate
            Block_carry60_carry64 : PG_BLOCK PORT MAP( P_ik =>p(4)(6+x), G_ik =>g(4)(6+x) , P_k1j=>p(4)(5), G_k1j => g(4)(5), P_ij =>
                ↪ tmp_p(10+x), G_ij =>tmp_g(10+x) );
        end generate Stage_60_64;
    end generate if64_2;

    -- i value is:
    -- 0 for 32 bit
    -- 1 for 64 bit
    -- This is used to create a vector with some value repeated.
    ----------------------------------- Example: if Nbit = 64 it creates:
    -- (co_tmp(7) & co_tmp(7) & co_tmp(7) & co_tmp(7) & co_tmp(7) & co_tmp(7) & co_tmp(7) & co_tmp(7) & co_tmp(3) & co_tmp(3) & co_tmp
        ↪ (3) & co_tmp(3))
    -- co_tmp(7) is used by all 8 block of the last stage between 32 and 62, while co_tmp(3) is used by the 4 block between 16 and 32
    -- k starts from 1 and not from 0
    if16_3: if Nbit > 16 generate
        selection_i : for i in 0 to integer(log2(real(Nbit)))-5 generate
            range_selection: for k in 4*(2**(i+1)-1)-(2**(i+2)-1) to 4*(2**(i+1)-1) generate
                reference_gij(k) <= co_tmp(-1+2**(i+2));
            end generate range_selection;
        end generate selection_i;
    end generate if16_3;

     -- This is used to generate the last stage of G block
    if16_4: if (Nbit>16) generate
        Fifth_stage : for x in 4 to Nbit/4 -1 generate
            Block_stage_FIVE : G_BLOCK PORT MAP(P_ik =>tmp_p(x-4), G_ik =>tmp_g(x-4) ,G_k1j => reference_gij(x-3), G_ij => co_tmp(x));
        end generate Fifth_stage;
    end generate if16_4;
    -- END OF STAGES ------------------------------------------------------------------------------------------------------------------

    -- Now, the carries obtained from the various stages are assigned, with also the concatenation of the carry in, to the output of the
        ↪ tree
    Cout <= co_tmp & Cin;

end STRUCTURAL;
```

Listing B.24: carry_select_block.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity carry_select_block is
    generic(N: positive := 4);                              --number of bits of the block
    port(   A, B:   in std_logic_vector(N-1 downto 0);      --data inputs
            S:      out std_logic_vector(N-1 downto 0);     --data output
            Ci:     in std_logic);                          --block carry-in
end entity carry_select_block;

architecture structural of carry_select_block is

    component MUX_2to1 is
        Generic (N: integer:= 1);                               --number of bits
        Port (  IN0:    In  std_logic_vector(N-1 downto 0);     --data input 1
                IN1:    In  std_logic_vector(N-1 downto 0);     --data input 2
                SEL:    In  std_logic;                          --selection input
                Y:      Out std_logic_vector(N-1 downto 0));    --data output
    end component MUX_2to1;

    component RCA is
        generic (N:  integer := 8);                         --number of bits
        Port (  A:  In  std_logic_vector(N-1 downto 0);     --data input 1
                B:  In  std_logic_vector(N-1 downto 0);     --data input 2
                Ci: In  std_logic;                          --carry-in
                S:  Out std_logic_vector(N-1 downto 0);     --data output
                Co: Out std_logic);                         --carry-out
    end component RCA;

    signal S_0: std_logic_vector(N-1 downto 0);             --sum if carry-in is 0
    signal S_1: std_logic_vector(N-1 downto 0);             --sum if carry-in is 1

begin

    --sum with carry-in=0
    RCA_0: RCA  generic map(N)
                port map(A=>A, B=>B, Ci=>'0', S=>S_0, Co=>open);

    --sum with carry-in=1
    RCA_1: RCA  generic map(N)
                port map(A=>A, B=>B, Ci=>'1', S=>S_1, Co=>open);

    --select the actual sum depending on the effective carry-in
    SUM_SELECT_MUX:  MUX_2to1   generic map(N)
                                port map(IN0=>S_0, IN1=>S_1, SEL=>Ci, Y=>S);

end architecture structural;
```

Listing B.25: RCA.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity RCA is
    generic (N:  integer := 8);                         --number of bits
    Port (  A:  In  std_logic_vector(N-1 downto 0);     --data input 1
            B:  In  std_logic_vector(N-1 downto 0);     --data input 2
            Ci: In  std_logic;                          --carry in
            S:  Out std_logic_vector(N-1 downto 0);     --data output
            Co: Out std_logic);                         --carry out
end RCA;

architecture STRUCTURAL of RCA is

    signal STMP : std_logic_vector(N-1 downto 0);   --sum
    signal CTMP : std_logic_vector(N downto 0);     --carries

    component FA
        Port(   A:  In  std_logic;
                B:  In  std_logic;
                Ci: In  std_logic;
                S:  Out std_logic;
                Co: Out std_logic);
    end component;

begin

  CTMP(0) <= Ci;        --carry in
  S <= STMP;            --sum
  Co <= CTMP(N);        --carry out

  ADDER_GEN: for I in 1 to N generate   --generates a fulladder for each bit of the adder
    FULLADDER : FA
        port map (A => A(I-1), B=> B(I-1), Ci => CTMP(I-1), S=> STMP(I-1), Co=> CTMP(I));
    end generate ADDER_GEN;

end STRUCTURAL;
```

Listing B.26: PG_network.vhd

```
library ieee;
use ieee.std_logic_1164.all;

--Create the two starting signal for each pair of bit

ENTITY PG_NETWORK IS
    PORT (
        op1 : in  std_logic;    --input bit 1
        op2 : in  std_logic;    --input bit 2
        g, p: out std_logic);   --output propagate and generate
END ENTITY PG_NETWORK;

architecture STRUCTURAL of PG_NETWORK is
```

```
BEGIN

    g <= op1 and op2;   --generate
    p <= op1 xor op2;   --propagate

end STRUCTURAL;
```

Listing B.27: G_block.vhd

```
library ieee;
use ieee.std_logic_1164.all;

-- General generate block. It takes three inputs and give one output.
-- P_ik is the general propagate with index i:k which arrives from the same column
-- G_ik is the general generate with index i:k which arrives from the same column
-- G_k1j is the general generate with index k-1:j which arrives from different column
-- G_ij is the result of the block with index i:j

ENTITY  G_BLOCK IS
    PORT (
      P_ik , G_ik   : in  std_logic;
      G_k1j         : in  std_logic;
      G_ij          : out std_logic);
END ENTITY G_BLOCK;

architecture STRUCTURAL of G_BLOCK is

BEGIN

    --definition of a general generate block
    G_ij <=  G_ik or (P_ik and G_k1j);

end STRUCTURAL;
```

Listing B.28: PG_block.vhd

```
library ieee;
use ieee.std_logic_1164.all;

-- General propagate block. It takes four inputs and give two outputs.
-- It as general generate block with the adding of the computation of propagate signal
-- For G:
-- P_ik is the general propagate with index i:k which arrives from the same column
-- G_ik is the general generate with index i:k which arrives from the same column
-- G_k1j is the general generate with index k-1:j which arrives from different column
-- G_ij is the result of the block with index i:j
-- For P:
-- P_ik is the general propagate with index i:k which arrives from the same column
-- P_k1j is the general propagate with index k-1:j which arrives from different column
-- P_ij is the result of the block with index i:j

ENTITY  PG_BLOCK IS
    PORT (
        P_ik , G_ik : in  std_logic;
        P_k1j, G_k1j: in  std_logic;
        P_ij, G_ij  : out std_logic);
END ENTITY PG_BLOCK;

architecture STRUCTURAL of PG_BLOCK is
BEGIN

    --definition of a general propagate block
    P_ij <=  P_ik and P_k1j;
    G_ij <=  G_ik or (P_ik and G_k1j);

end STRUCTURAL;
```

Listing B.29: FA.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity FA is
    Port (  A:  In  std_logic;          --input bit 1
            B:  In  std_logic;          --input bit 2
            Ci: In  std_logic;          --carry in
            S:  Out std_logic;          --sum
            Co: Out std_logic);         --carry out
end FA;

architecture structural of FA is
```

```
begin

  S <= (A xor B xor Ci);                                --sum
  Co <= ((A and B) or (B and Ci) or (A and Ci));        --carry out

end structural;
```

Listing B.30: MUX_2to1.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MUX_2to1 is
    Generic (N: integer:= 1);                               --number of bits
    Port (  IN0:    In  std_logic_vector(N-1 downto 0);     --data input 1
            IN1:    In  std_logic_vector(N-1 downto 0);     --data input 2
            SEL:    In  std_logic;                          --selection input
            Y:      Out std_logic_vector(N-1 downto 0));    --data output
end entity MUX_2to1;

architecture behavioral of MUX_2to1 is
begin

    with SEL select Y<=
        IN0 when '0',
        IN1 when others;

end architecture behavioral;
```

Listing B.31: MUX_4to1.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MUX_4to1 is
    Generic (N: integer:= 1);   --number of bits
    Port (  IN0, IN1, IN2, IN3  : In    std_logic_vector(N-1 downto 0);     --data inputs
            SEL                 : In    std_logic_vector(1 downto 0);       --selection input
            Y                   : Out   std_logic_vector(N-1 downto 0));    --data output
end entity MUX_4to1;

architecture behavioral of MUX_4to1 is
begin

    with SEL select Y<=
        IN0 when "00",
        IN1 when "01",
        IN2 when "10",
        IN3 when others;

end architecture behavioral;
```

Listing B.32: MUX_8to1.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity MUX_8to1 is
    Generic (N: integer:= 1);   --number of bits
    Port (  IN0, IN1, IN2, IN3, IN4, IN5, IN6, IN7  : In    std_logic_vector(N-1 downto 0);     --data inputs
            SEL                 : In    std_logic_vector(2 downto 0);       --selection input
            Y                   : Out   std_logic_vector(N-1 downto 0));    --data output
end entity MUX_8to1;

architecture behavioral of MUX_8to1 is
begin

    with SEL select Y<=
        IN0 when "000",
        IN1 when "001",
        IN2 when "010",
        IN3 when "011",
        IN4 when "100",
        IN5 when "101",
        IN6 when "110",
        IN7 when others;

end architecture behavioral;
```

Listing B.33: reg.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;

entity reg is
    generic (N: positive:= 1 );                                     --number of bits
    port(   D       : In    std_logic_vector(N-1 downto 0);         --data input
            Q       : Out   std_logic_vector(N-1 downto 0);         --data output
            EN      : In    std_logic;                              --enable active high
            CLK     : In    std_logic;                              --clock
            RST     : In    std_logic                               --asynchronous reset active low
    );
end entity reg;

architecture behavioral of reg is   --register with asyncronous reset and enable

begin
    reg_proc: process(CLK, RST)     --asynchronous reset
    begin
        if RST='0' then             --if reset is active
            Q<=(others=>'0');       --clear the output
        elsif rising_edge(CLK) then     --otherwise if there is a positive clock edge
            if EN='1' then          --and enable is active
                Q <= D;             --writes the input on the output
            end if;
        end if;
    end process;

end behavioral;
```

APPENDIX C

Testbench

Listing C.1: TB_RISCV.v

```
module TB_RISCV ();

    wire [31:0] IRAM_ADDR, DRAM_ADDR, IRAM_OUT, DRAM_OUT, DRAM_IN;
    wire [1:0] DRAM_WR_EN;
    reg IRAM_CS, DRAM_CS;
    wire CLK, RST;

    IRAM IRAM_instance(
        .DATA_OUT(IRAM_OUT),
        .ADDR(IRAM_ADDR[15:0]),
        .CS(IRAM_CS)
    );

    always @ (IRAM_ADDR) begin
        if (IRAM_ADDR[31:16]=='H0040)
            IRAM_CS=1;
        else
            IRAM_CS=0;
    end

    DRAM DRAM_instance(
        .DATA_IN(DRAM_IN),
        .DATA_OUT(DRAM_OUT),
        .ADDR(DRAM_ADDR[15:0]),
        .WR_EN(DRAM_WR_EN),
        .CS(DRAM_CS),
        .CLK(CLK)
    );

    always @ (DRAM_ADDR) begin
        if (DRAM_ADDR[31:16]=='H1001)
            DRAM_CS=1;
        else
            DRAM_CS=0;
    end

    clk_gen clk_gen_instance(
        .CLK(CLK),
        .RST(RST)
    );

    RISCV DUT(
        .IRAM_ADDR(IRAM_ADDR),
        .IRAM_OUT(IRAM_OUT),
        .DRAM_ADDR(DRAM_ADDR),
        .DRAM_OUT(DRAM_OUT),
        .DRAM_IN(DRAM_IN),
        .DRAM_WR_EN(DRAM_WR_EN),
        .CLK(CLK),
        .RST(RST)
    );

endmodule
```

Listing C.2: DRAM.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;
use work.RISCV_package.all;
use work.CU_package.all;

entity DRAM is
    port(
        DATA_IN     : in word;
        DATA_OUT    : out word;
        ADDR        : in halfword;
        WR_EN       : in std_logic_vector(1 downto 0);  --00 off, 01 byte, 10 halfword, 11 word
        CS          : in std_logic;                     --chip select
        CLK         : in std_logic
    );
end entity DRAM;

architecture behavioral of DRAM is

    constant MEM_SIZE: positive:= 2**HALFWORD_size;
    type MEM_type is array (0 to MEM_SIZE-1) of byte;

    signal MEM: MEM_type;
    signal ADDR_to_int: natural;
    signal byte0_out, byte1_out, byte2_out, byte3_out: byte;

begin

    ADDR_to_int<=to_integer(unsigned(ADDR));

    byte0_out <= MEM(ADDR_to_int) when CS='1' else
                 (others=>'0');

    byte1_out <= MEM(ADDR_to_int+1) when ADDR_to_int+1<=MEM_SIZE-1 and CS='1'  else
                 (others=>'0');

    byte2_out <= MEM(ADDR_to_int+2) when ADDR_to_int+2<=MEM_SIZE-1 and CS='1'  else
                 (others=>'0');

    byte3_out <= MEM(ADDR_to_int+3) when ADDR_to_int+3<=MEM_SIZE-1 and CS='1'  else
                 (others=>'0');

    DATA_OUT <= byte3_out & byte2_out & byte1_out & byte0_out;

    file_proc: process(CLK)
        file file_to_read: text;
        variable file_line : line;
        variable n_byte : integer := 0;
        variable instruction : word;
        variable init : boolean:= true;
    begin
        if rising_edge(CLK) then
            if init=true then
                init:=false;
                file_open(file_to_read,"data.hex",READ_MODE);
                while (not endfile(file_to_read)) loop
                    readline(file_to_read,file_line);
                    hread(file_line,instruction);
                    MEM(n_byte) <= instruction(byte_range);
                    MEM(n_byte+1) <= instruction(byte1_range);
                    MEM(n_byte+2) <= instruction(byte2_range);
                    MEM(n_byte+3) <= instruction(byte3_range);
                    n_byte := n_byte + 4;
                end loop;
            end if;
            if CS='1' then
                if WR_EN=DRAM_WR_W then
                    MEM(ADDR_to_int)<=DATA_IN(byte_range);
                    if ADDR_to_int+1<=MEM_SIZE-1 then
                        MEM(ADDR_to_int+1)<=DATA_IN(byte1_range);
                    end if;
                    if ADDR_to_int+2<=MEM_SIZE-1 then
                        MEM(ADDR_to_int+2)<=DATA_IN(byte2_range);
                    end if;
                    if ADDR_to_int+3<=MEM_SIZE-1 then
                        MEM(ADDR_to_int+3)<=DATA_IN(byte3_range);
                    end if;
                elsif WR_EN=DRAM_WR_H then
                    MEM(ADDR_to_int)<=DATA_IN(byte_range);
                    if ADDR_to_int+1<=MEM_SIZE-1 then
                        MEM(ADDR_to_int+1)<=DATA_IN(byte1_range);
                    end if;
                elsif WR_EN=DRAM_WR_B then
                    MEM(ADDR_to_int)<=DATA_IN(byte_range);
                end if;
            end if;
        end if;
    end process file_proc;

end architecture behavioral;
```

Listing C.3: IRAM.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use std.textio.all;
use ieee.std_logic_textio.all;
use work.RISCV_package.all;

entity IRAM is
    port(
        DATA_OUT    : out word;
        ADDR        : in halfword;
        CS          : in std_logic  --chip select
    );
end entity IRAM;

architecture behavioral of IRAM is

    constant MEM_SIZE: positive:= 2**HALFWORD_size;
    type MEM_type is array (0 to MEM_SIZE-1) of byte;

    signal MEM: MEM_type;
    signal ADDR_to_int: natural;
    signal byte0_out, byte1_out, byte2_out, byte3_out: byte;

begin

    ADDR_to_int<=to_integer(unsigned(ADDR));

    byte0_out <= MEM(ADDR_to_int) when CS='1' else
                 (others=>'0');

    byte1_out <= MEM(ADDR_to_int+1) when ADDR_to_int+1<=MEM_SIZE-1 and CS='1' else
                 (others=>'0');

    byte2_out <= MEM(ADDR_to_int+2) when ADDR_to_int+2<=MEM_SIZE-1 and CS='1'  else
                 (others=>'0');

    byte3_out <= MEM(ADDR_to_int+3) when ADDR_to_int+3<=MEM_SIZE-1 and CS='1'  else
                 (others=>'0');

    DATA_OUT <= byte3_out & byte2_out & byte1_out & byte0_out;

    file_read_proc: process
        file file_to_read: text;
        variable file_line : line;
        variable n_byte : integer := 0;
        variable instruction : word;
    begin
        file_open(file_to_read,"test.hex",READ_MODE);
        while (not endfile(file_to_read)) loop
            readline(file_to_read,file_line);
            hread(file_line,instruction);
            MEM(n_byte) <= instruction(byte_range);
            MEM(n_byte+1) <= instruction(byte1_range);
            MEM(n_byte+2) <= instruction(byte2_range);
            MEM(n_byte+3) <= instruction(byte3_range);
            n_byte := n_byte + 4;
        end loop;
        wait;
    end process file_read_proc;

end behavioral;
```

Listing C.4: clk_gen.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity clk_gen is
    port(
        clk     : out std_logic;
        rst     : out std_logic
    );
end entity clk_gen;

architecture behavioural of clk_gen is

    constant clk_period : time := 2 ns;

begin

    clk_proc: process
    begin
        CLK<='0';
        wait for clk_period/2;
        CLK<='1';
        wait for clk_period/2;
    end process clk_proc;
    RST<='0', '1' after clk_period;

end architecture behavioural;
```

Listing C.5: test_standard.asm

```
################
# Basic VERSION
# This program takes an array v and computes
# min{|v[i]|}, the minimum of the absolute value,
# where v[i] is the i-th element in the array
    .data
    .align  2
v:
    .word   10
    .word   -47
    .word   22
    .word   -3
    .word   15
    .word   27
    .word   -4
m:
    .word   0

    .text
    .align  2
    .globl  __start
__start:
    li x16,7          # put 7 in x16
    la x4,v           # put in x4 the address of v
    la x5,m           # put in x5 the address of m
    li x13,0x3fffffff # init x13 with max pos
loop:
    beq x16,x0,done   # check all elements have been tested
    lw x8,0(x4)       # load new element in x8
    add x10,x10,x9    # x10 += x9 (add the carry in)
    addi x4,x4,0x4    # point to next element
    addi x16,x16,-1   # decrease x16 by 1
    slt x11,x10,x13   # x11 = (x10 < x13) ? 1 : 0
    beq x11,x0,loop   # next element
    add x13,x10,x0    # update min
    jal loop          # next element
done:
    sw x13,0(x5)      # store the result
endc:
    jal endc      # infinite loop
    addi x0,x0,0
```

Listing C.6: test_custom.asm

```
################
# Improved VERSION
# This program takes an array v and computes
# min{|v[i]|}, the minimum of the absolute value,
# where v[i] is the i-th element in the array
    .data
    .align  2
v:
    .word   10
    .word   -47
    .word   22
    .word   -3
    .word   15
    .word   27
    .word   -4
m:
    .word   0

    .text
    .align  2
    .globl  __start
__start:
    li x16,7          # put 7 in x16
    la x4,v           # put in x4 the address of v
    la x5,m           # put in x5 the address of m
    li x13,0x3fffffff # init x13 with max pos
loop:
    beq x16,x0,done   # check all elements have been tested
    lw x8,0(x4)       # load new element in x8
    abs x10, x8       # compute the absolute value
    addi x4,x4,0x4    # point to next element
    addi x16,x16,-1   # decrease x16 by 1
    slt x11,x10,x13   # x11 = (x10 < x13) ? 1 : 0
    beq x11,x0,loop   # next element
    add x13,x10,x0    # update min
    jal loop          # next element
done:
    sw x13,0(x5)      # store the result
endc:
    jal endc      # infinite loop
    addi x0,x0,0
```

Listing C.7: test_standard.hex

```
00700813
0fc10217
ffc20213
0fc10297
01028293
400006b7
fff68693
02080863
00022403
41f45493
00944533
0014f493
00950533
00420213
fff80813
00d525b3
fc058ee3
000506b3
fd5ff0ef
00d2a023
000000ef
00000013
```

Listing C.8: test_custom.hex

```
00700813
0fc10217
ffc20213
0fc10297
01028293
400006b7
fff68693
02080263
00022403
0004050b
00420213
fff80813
00d525b3
fe0584e3
000506b3
fe1ff0ef
00d2a023
000000ef
00000013
```

Listing C.9: data.hex

```
0000000a
ffffffd1
00000016
fffffffd
0000000f
0000001b
fffffffc
```

APPENDIX D

Scripts

Listing D.1: sim.tcl

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate -format Logic -label CLOCK /TB_RISCV/CLK
add wave -noupdate -format Logic -label RESET /TB_RISCV/RST
add wave -noupdate -divider IRAM
add wave -noupdate -color Magenta -format Literal -label DATA_OUT -radix hexadecimal /TB_RISCV/IRAM_instance/data_out
add wave -noupdate -color Magenta -format Literal -label ADDRESS -radix hexadecimal /TB_RISCV/IRAM_instance/addr
add wave -noupdate -divider DRAM
add wave -noupdate -color Blue -format Literal -label DATA_IN -radix hexadecimal /TB_RISCV/DRAM_instance/data_in
add wave -noupdate -color Blue -format Literal -label ADDRESS -radix hexadecimal /TB_RISCV/DRAM_instance/addr
add wave -noupdate -color Blue -format Literal -label WRITE_ENABLE /TB_RISCV/DRAM_instance/wr_en
add wave -noupdate -color Blue -format Literal -label DRAM(28) -radix decimal /TB_RISCV/DRAM_instance/mem(28)
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {0 ns} 0}
configure wave -namecolwidth 281
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 0
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
update
WaveRestoreZoom {0 ns} {200 ns}
```

Listing D.2: syn.tcl

```
#*****************************************************************************
# This script is used to synthesize the RISC-V
#*****************************************************************************

#to preserve rtl names in the netlist for power consumption estimation.
set power_preserve_rtl_hier_names true

#analyze all possible file contained in the work folder
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/my_package.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/ALU_package.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/FU_package.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/HDU_package.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/RISCV_package.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/CU_package.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/FA.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/MUX_2to1.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/RCA.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/MUX_4to1.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/G_block.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/PG_block.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/PG_network.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/carry_select_block.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/MUX_8to1.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/barrel_shifter.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/HDU.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/FU.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/RF.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/BPU.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/adder_subtractor.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/carry_generator.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/sum_generator.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/PC.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/reg.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/branch_comp.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/IR.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/CU.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/comparator.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/logic_block.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/ALU.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/datapath.vhd}
analyze -library WORK -format vhdl {/home/isa25/Desktop/RISCV/src/RISCV.vhd}

#elaborate top entity, set the variables
elaborate RISCV -architecture structural -library WORK -parameters "BPU_TAG_FIELD_SIZE = 8 , BPU_SET_FIELD_SIZE = 3, BPU_LINES_PER_SET =
       ↪  4"

#**************** CONSTRAINT THE SYNTHESIS ****************#
#timing constraint. WCP holds the minimum value of the delay
set WCP 2
#forces a combinational max delay from any input to any output
set_max_delay $WCP -from [all_inputs] -to [all_outputs]
#create a clock signal with a period equal to the worst critical path
create_clock -name "CLK" -period $WCP CLK
#clock uncertainty (jitter)
set_clock_uncertainty 0.07 [get_clocks CLK]
#max input delay
set_input_delay 0.5 -max -clock CLK [remove_from_collection [all_inputs] CLK]
#max output delay
set_output_delay 0.5 -max -clock CLK [all_outputs]
#output load equal to the buffer input capacitance of the library
set OUT_LOAD [load_of NangateOpenCellLibrary/BUF_X4/A]
set_load $OUT_LOAD [all_outputs]
#verify the correct creation of the clock
report_clock > clock_test.rpt
set_dont_touch_network CLK

#compile ultra command, in order to perform an advanced synthesis
compile_ultra

#perform retiming on the compiled netlist
optimize_register

ungroup -all -flatten
change_names -hierarchy -rules verilog
#delay description
write_sdf ../netlist/RISCV.sdf
#write verilog netlist
write -f verilog -hierarchy -output ../netlist/RISCV.v
#input/output constraints
write_sdc ../netlist/RISCV.sdc

report_timing > ../report/timing.rpt
report_area > ../report/area.rpt
report_power > ../report/power.rpt
```

Listing D.3: pr.tcl

```
set_global _enable_mmmc_by_default_flow      $CTE::mmmc_default
suppressMessage ENCEXT-2799
getDrawView
loadWorkspace -name Physical
win
set init_design_netlisttype verilog
set init_design_settop 1
set init_top_cell RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
set init_verilog ../netlist/RISCV.v
set init_lef_file /software/dk/nangate45/lef/NangateOpenCellLibrary.lef
set init_gnd_net VSS
set init_pwr_net VDD
set init_mmmc_file mmm_design.tcl
init_design
getIoFlowFlag
setIoFlowFlag 0
floorPlan -coreMarginsBy die -site FreePDK45_38x28_10R_NP_162NW_34O -r 1 0.6 5 5 5 5
uiSetTool select
getIoFlowFlag
fit
set sprCreateIeRingOffset 1.0
set sprCreateIeRingThreshold 1.0
set sprCreateIeRingJogDistance 1.0
set sprCreateIeRingLayers {}
set sprCreateIeRingOffset 1.0
set sprCreateIeRingThreshold 1.0
set sprCreateIeRingJogDistance 1.0
set sprCreateIeRingLayers {}
set sprCreateIeStripeWidth 10.0
set sprCreateIeStripeThreshold 1.0
set sprCreateIeStripeWidth 10.0
set sprCreateIeStripeThreshold 1.0
set sprCreateIeRingOffset 1.0
set sprCreateIeRingThreshold 1.0
set sprCreateIeRingJogDistance 1.0
set sprCreateIeRingLayers {}
set sprCreateIeStripeWidth 10.0
set sprCreateIeStripeThreshold 1.0
setAddRingMode -ring_target default -extend_over_row 0 -ignore_rows 0 -avoid_short 0 -skip_crossing_trunks none -stacked_via_top_layer
      ↪ metal10 -stacked_via_bottom_layer metal1 -via_using_exact_crossover_size 1 -orthogonal_only true -skip_via_on_pin {
      ↪ standardcell } -skip_via_on_wire_shape {  noshape }
addRing -nets {VDD VSS} -type core_rings -follow core -layer {top metal1 bottom metal1 left metal2 right metal2} -width {top 0.8 bottom
      ↪ 0.8 left 0.8 right 0.8} -spacing {top 0.8 bottom 0.8 left 0.8 right 0.8} -offset {top 1.8 bottom 1.8 left 1.8 right 1.8} -
      ↪ center 1 -extend_corner {} -threshold 0 -jog_distance 0 -snap_wire_center_to_grid None
clearGlobalNets
globalNetConnect VDD -type pgpin -pin VDD -inst * -module {}
setSrouteMode -viaConnectToShape { noshape }
sroute -connect { blockPin padPin padRing corePin floatingStripe } -layerChangeRange { metal1(1) metal10(10) } -blockPinTarget {
      ↪ nearestTarget } -padPinPortConnect { allPort oneGeom } -padPinTarget { nearestTarget } -corePinTarget { firstAfterRowEnd } -
      ↪ floatingStripeTarget { blockring padring ring stripe ringpin blockpin followpin } -allowJogging 1 -crossoverViaLayerRange {
      ↪ metal1(1) metal10(10) } -nets { VDD VSS } -allowLayerChange 1 -blockPin useLef -targetViaLayerRange { metal1(1) metal10(10) }
setPlaceMode -prerouteAsObs {1 2 3 4 5 6 7 8}
setPlaceMode -fp false
placeDesign
setOptMode -fixCap true -fixTran true -fixFanoutLoad false
optDesign -postCTS
optDesign -postCTS -hold
getFillerMode -quiet
addFiller -cell FILLCELL_X8 FILLCELL_X4 FILLCELL_X32 FILLCELL_X2 FILLCELL_X16 FILLCELL_X1 -prefix FILLER
setNanoRouteMode -quiet -timingEngine {}
setNanoRouteMode -quiet -routeWithSiPostRouteFix 0
setNanoRouteMode -quiet -drouteStartIteration default
setNanoRouteMode -quiet -routeTopRoutingLayer default
setNanoRouteMode -quiet -routeBottomRoutingLayer default
setNanoRouteMode -quiet -drouteEndIteration default
setNanoRouteMode -quiet -routeWithTimingDriven false
setNanoRouteMode -quiet -routeWithSiDriven false
routeDesign -globalDetail
setAnalysisMode -analysisType onChipVariation
setOptMode -fixCap true -fixTran true -fixFanoutLoad false
optDesign -postRoute
optDesign -postRoute -hold
saveDesign RISCV_Basic_ppr.enc
reset_parasitics
extractRC
rcOut -setload RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4.setload -rc_corner my_rc
rcOut -setres RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4.setres -rc_corner my_rc
rcOut -spf RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4.spf -rc_corner my_rc
rcOut -spef RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4.spef -rc_corner my_rc
redirect -quiet {set honorDomain [getAnalysisMode -honorClockDomains]} > /dev/null
timeDesign -preCTS -pathReports -drvReports -slackReports -numPaths 50 -prefix
      ↪ RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4_preCTS -outDir timingReports
redirect -quiet {set honorDomain [getAnalysisMode -honorClockDomains]} > /dev/null
timeDesign -postRoute -pathReports -drvReports -slackReports -numPaths 50 -prefix
      ↪ RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4_postRoute -outDir timingReports
redirect -quiet {set honorDomain [getAnalysisMode -honorClockDomains]} > /dev/null
timeDesign -postRoute -hold -pathReports -slackReports -numPaths 50 -prefix
      ↪ RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4_postRoute -outDir timingReports
get_time_unit
report_timing -machine_readable -max_paths 10000 -max_slack 0.75 -path_exceptions all > top.mtarpt
load_timing_debug_report -name default_report top.mtarpt
verifyConnectivity -type all -error 1000 -warning 50
setVerifyGeometryMode -area { 0 0 0 0 } -minWidth true -minSpacing true -minArea true -sameNet true -short true -overlap true -offRGrid
      ↪ false -offMGrid true -mergedMGridCheck true -minHole true -implantCheck true -minimumCut true -minStep true -viaEnclosure true
      ↪  -antenna false -insuffMetalOverlap true -pinInBlkg false -diffCellViol true -sameCellViol false -padFillerCellsOverlap true -
      ↪ routingBlkgPinOverlap true -routingCellBlkgOverlap true -regRoutingOnly false -stackedViasOnRegNet false -wireExt true -
      ↪ useNonDefaultSpacing false -maxWidth true -maxNonPrefLength -1 -error 1000
verifyGeometry
setVerifyGeometryMode -area { 0 0 0 0 }
reportGateCount -level 5 -limit 100 -outfile RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4.gateCount
saveNetlist RISCV_ppr.v
all_hold_analysis_views
all_setup_analysis_views
write_sdf  -ideal_clock_network RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4.sdf
set_power_analysis_mode -reset
set_power_analysis_mode -method static -corner max -create_binary_db true -write_static_currents true -honor_negative_energy true -
      ↪ ignore_control_signals true
set_power_output_dir -reset
set_power_output_dir ./
set_default_switching_activity -reset
set_default_switching_activity -input_activity 0.2 -period 10.0
read_activity_file -reset
read_activity_file -format VCD -scope /TB_RISCV/DUT -start {} -end {} -block {} ../vcd/design.vcd
set_power -reset
set_powerup_analysis -reset
set_dynamic_power_simulation -reset
report_power -rail_analysis_format VS -outfile .//RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4.rpt
report_power -sort { total }
report_power -outfile POWER_AFTER_SWITCH -sort { total }
```

APPENDIX E

Reports

Listing E.1: area_syn_standard.rpt

```
****************************************
Report : area
Design : RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
Version: O-2018.06-SP4
Date   : Fri Feb 21 21:49:34 2020
****************************************

Library(s) Used:

    NangateOpenCellLibrary (File: /software/dk/nangate45/synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)

Number of ports:                          164
Number of nets:                         13738
Number of cells:                        12775
Number of combinational cells:           9909
Number of sequential cells:              2864
Number of macros/black boxes:               0
Number of buf/inv:                       1059
Number of references:                      54

Combinational area:              12260.206064
Buf/Inv area:                      642.922000
Noncombinational area:           15232.490488
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                 27492.696552
Total area:                 undefined
1
```

Listing E.2: power_syn_standard.rpt

```
Information: Propagating switching activity (low effort zero delay simulation). (PWR-6)
Warning: Design has unannotated primary inputs. (PWR-414)
Warning: Design has unannotated sequential cell outputs. (PWR-415)

****************************************
Report : power
        -analysis_effort low
Design : RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
Version: O-2018.06-SP4
Date   : Fri Feb 21 21:49:36 2020
****************************************


Library(s) Used:

    NangateOpenCellLibrary (File: /software/dk/nangate45/synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)


Operating Conditions: typical   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
                       5K_hvratio_1_1    NangateOpenCellLibrary
```

```
Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =  10.5216 mW   (95%)
  Net Switching Power  = 582.9406 uW    (5%)
                         ---------
Total Dynamic Power    =  11.1045 mW  (100%)

Cell Leakage Power     = 492.5637 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000            0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000            0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000            0.0000            0.0000            0.0000  (   0.00%)
clock_network      0.0000            0.0000            0.0000            0.0000  (   0.00%)
register       1.0288e+04          184.3803        2.2945e+05        1.0702e+04  (  92.28%)
sequential         0.0000            0.0000            0.0000            0.0000  (   0.00%)
combinational    233.2560          398.5603        2.6311e+05          894.9277  (   7.72%)
--------------------------------------------------------------------------------------------------
Total          1.0522e+04 uW       582.9406 uW     4.9256e+05 nW     1.1597e+04 uW
1
```

Listing E.3: timing_syn_standard.rpt

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
Version: O-2018.06-SP4
Date   : Fri Feb 21 21:49:34 2020
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: typical   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

  Startpoint: CLK_r_REG2758_S4
              (rising edge-triggered flip-flop clocked by CLK)
  Endpoint: CLK_r_REG2671_S2
            (rising edge-triggered flip-flop clocked by CLK)
  Path Group: CLK
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
                     5K_hvratio_1_1        NangateOpenCellLibrary

  Point                                                    Incr       Path
  --------------------------------------------------------------------------
  clock CLK (rise edge)                                    0.00       0.00
  clock network delay (ideal)                              0.00       0.00
  CLK_r_REG2758_S4/CK (DFFS_X1)                            0.00 #     0.00 r
  CLK_r_REG2758_S4/Q (DFFS_X1)                             0.11       0.11 f
  U3762/ZN (OR2_X1)                                        0.07       0.18 f
  U3760/ZN (NAND2_X2)                                      0.06       0.24 r
  datapath_instance_RF_instance_U150/ZN (INV_X1)           0.05       0.29 f
  datapath_instance_RF_instance_U209/ZN (NAND2_X1)         0.06       0.35 r
  datapath_instance_RF_instance_U219/ZN (OR2_X1)           0.06       0.41 r
  datapath_instance_RF_instance_U220/ZN (INV_X1)           0.06       0.47 r
  datapath_instance_RF_instance_U344/ZN (AOI22_X1)         0.07       0.54 r
  datapath_instance_RF_instance_U345/ZN (NAND2_X1)         0.03       0.57 f
  datapath_instance_RF_instance_U346/ZN (AOI21_X1)         0.05       0.62 r
  datapath_instance_RF_instance_U348/ZN (NAND4_X1)         0.04       0.66 f
  datapath_instance_RF_instance_U27/ZN (OR3_X1)            0.07       0.74 f
  U5096/ZN (AOI22_X1)                                      0.05       0.78 r
  U4318/ZN (AND3_X2)                                       0.07       0.85 r
  U5836/ZN (NOR2_X1)                                       0.03       0.88 f
  U3875/ZN (NOR2_X1)                                       0.05       0.93 r
  U2766/ZN (AND2_X1)                                       0.06       0.99 r
  U3823/ZN (INV_X1)                                        0.03       1.02 f
  U4086/ZN (OAI211_X1)                                     0.05       1.06 r
  U3824/ZN (AOI21_X1)                                      0.03       1.09 f
  U3712/ZN (OAI21_X1)                                      0.04       1.13 r
  U3822/ZN (NAND2_X1)                                      0.03       1.16 f
  U5710/ZN (NAND3_X1)                                      0.03       1.19 r
  U5641/ZN (NAND3_X1)                                      0.03       1.22 f
  U2935/ZN (NAND3_X1)                                      0.03       1.25 r
  U2934/ZN (NAND2_X1)                                      0.03       1.27 f
  U2674/ZN (NAND2_X1)                                      0.03       1.30 r
  U2920/ZN (NAND2_X1)                                      0.03       1.33 f
  U2672/ZN (NAND3_X1)                                      0.03       1.35 r
  U2671/ZN (NAND2_X1)                                      0.03       1.38 f
  U5907/ZN (NAND2_X1)                                      0.04       1.42 r
  U5914/Z (BUF_X2)                                         0.07       1.48 r
  U5915/Z (MUX2_X1)                                        0.09       1.57 f
  datapath_instance_BPU_instance_U288/ZN (XNOR2_X1)        0.05       1.62 r
  datapath_instance_BPU_instance_U289/ZN (NAND4_X1)        0.04       1.66 f
  datapath_instance_BPU_instance_U290/ZN (NOR2_X1)         0.04       1.71 r
  datapath_instance_BPU_instance_U1162/ZN (NAND4_X1)       0.05       1.76 f
  datapath_instance_BPU_instance_U1149/ZN (AND2_X1)        0.04       1.80 f
  U4461/ZN (NOR2_X1)                                       0.04       1.85 r
  U4462/ZN (INV_X1)                                        0.03       1.88 f
  CLK_r_REG2671_S2/D (DFFR_X1)                             0.01       1.89 f
  data arrival time                                                   1.89

  clock CLK (rise edge)                                    2.00       2.00
  clock network delay (ideal)                              0.00       2.00
  clock uncertainty                                       -0.07       1.93
  CLK_r_REG2671_S2/CK (DFFR_X1)                            0.00       1.93 r
  library setup time                                      -0.04       1.89
  data required time                                                  1.89
  --------------------------------------------------------------------------
  data required time                                                  1.89
  data arrival time                                                  -1.89
  --------------------------------------------------------------------------
  slack (MET)                                                         0.00


1
```

Listing E.4: sa_syn_standard.rpt

```
****************************************
Report : power
        -analysis_effort low
Design : RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
Version: O-2018.06-SP4
Date   : Sat Feb 22 12:43:04 2020
****************************************


Library(s) Used:

    NangateOpenCellLibrary (File: /software/dk/nangate45/synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)


Operating Conditions: typical   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
                       5K_hvratio_1_1    NangateOpenCellLibrary


Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =  11.1311 mW   (87%)
  Net Switching Power  =   1.6440 mW   (13%)
                         ---------
Total Dynamic Power    =  12.7751 mW  (100%)

Cell Leakage Power     = 508.4825 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000            0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000            0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000            0.0000            0.0000            0.0000  (   0.00%)
clock_network      0.0000            0.0000            0.0000            0.0000  (   0.00%)
register       1.0317e+04          338.3600        2.4813e+05        1.0904e+04  (  82.09%)
sequential         0.0000            0.0000            0.0000            0.0000  (   0.00%)
combinational    813.5981        1.3057e+03        2.6035e+05        2.3796e+03  (  17.91%)
--------------------------------------------------------------------------------------------------
Total          1.1131e+04 uW     1.6440e+03 uW     5.0848e+05 nW     1.3283e+04 uW
```

Listing E.5: area_syn_custom.rpt

```
****************************************
Report : area
Design : RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
Version: O-2018.06-SP4
Date   : Fri Feb 21 16:21:17 2020
****************************************

Library(s) Used:

    NangateOpenCellLibrary (File: /software/dk/nangate45/synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)

Number of ports:                          164
Number of nets:                         14122
Number of cells:                        13158
Number of combinational cells:          10278
Number of sequential cells:              2878
Number of macros/black boxes:               0
Number of buf/inv:                       1108
Number of references:                      48

Combinational area:              12497.212059
Buf/Inv area:                      659.680000
Noncombinational area:           15307.502491
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (Wire load has zero net area)

Total cell area:                 27804.714550
Total area:                 undefined
1
```

Listing E.6: power_syn_custom.rpt

```
Information: Propagating switching activity (low effort zero delay simulation). (PWR-6)
Warning: Design has unannotated primary inputs. (PWR-414)
Warning: Design has unannotated sequential cell outputs. (PWR-415)

****************************************
Report : power
        -analysis_effort low
Design : RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
Version: O-2018.06-SP4
Date   : Fri Feb 21 16:21:19 2020
****************************************


Library(s) Used:

    NangateOpenCellLibrary (File: /software/dk/nangate45/synopsys/NangateOpenCellLibrary_typical_ecsm_nowlm.db)


Operating Conditions: typical   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

Design        Wire Load Model            Library
------------------------------------------------
RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
                       5K_hvratio_1_1    NangateOpenCellLibrary


Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =  10.5982 mW   (94%)
  Net Switching Power  = 635.8781 uW    (6%)
                         ---------
Total Dynamic Power    =  11.2341 mW  (100%)

Cell Leakage Power     = 501.0743 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000            0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000            0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000            0.0000            0.0000            0.0000  (   0.00%)
clock_network      0.0000            0.0000            0.0000            0.0000  (   0.00%)
register       1.0345e+04          198.8457        2.3050e+05        1.0774e+04  (  91.81%)
sequential         0.0000            0.0000            0.0000            0.0000  (   0.00%)
combinational    253.2335          437.0318        2.7058e+05          960.8422  (   8.19%)
--------------------------------------------------------------------------------------------------
```

```
Total          1.0598e+04 uW       635.8775 uW     5.0107e+05 nW     1.1735e+04 uW
1
```

Listing E.7: timing_syn_custom.rpt

```
****************************************
Report : timing
        -path full
        -delay max
        -max_paths 1
Design : RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
Version: O-2018.06-SP4
Date   : Fri Feb 21 16:21:17 2020
****************************************

 # A fanout number of 1000 was used for high fanout net computations.

Operating Conditions: typical   Library: NangateOpenCellLibrary
Wire Load Model Mode: top

  Startpoint: datapath_instance_BPU_instance_CLK_r_REG1888_S6
              (rising edge-triggered flip-flop clocked by CLK)
  Endpoint: IRAM_ADDR[23]
            (output port clocked by CLK)
  Path Group: CLK
  Path Type: max

  Des/Clust/Port     Wire Load Model       Library
  ------------------------------------------------
  RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
                     5K_hvratio_1_1        NangateOpenCellLibrary

  Point                                                    Incr       Path
  --------------------------------------------------------------------------
  clock CLK (rise edge)                                    0.00       0.00
  clock network delay (ideal)                              0.00       0.00
  datapath_instance_BPU_instance_CLK_r_REG1888_S6/CK (DFFR_X2)
                                                           0.00 #     0.00 r
  datapath_instance_BPU_instance_CLK_r_REG1888_S6/Q (DFFR_X2)
                                                           0.23       0.23 r
  datapath_instance_BPU_instance_U635/ZN (OAI22_X1)       0.07       0.31 f
  datapath_instance_BPU_instance_U636/ZN (NOR4_X1)        0.11       0.42 r
  datapath_instance_BPU_instance_U415/Z (XOR2_X1)         0.07       0.48 r
  datapath_instance_BPU_instance_U103/ZN (AND2_X1)        0.04       0.53 r
  datapath_instance_BPU_instance_U451/ZN (NAND3_X1)       0.03       0.56 f
  datapath_instance_BPU_instance_U431/ZN (OR2_X4)         0.10       0.66 f
  datapath_instance_BPU_instance_U708/ZN (NAND3_X1)       0.07       0.72 r
  datapath_instance_BPU_instance_U709/ZN (INV_X1)         0.03       0.75 f
  datapath_instance_BPU_instance_U96/ZN (NAND2_X2)        0.07       0.82 r
  datapath_instance_BPU_instance_U101/ZN (OR2_X1)         0.17       1.00 r
  datapath_instance_BPU_instance_U1661/ZN (OAI211_X1)     0.06       1.06 f
  datapath_instance_BPU_instance_U1662/ZN (INV_X1)        0.03       1.09 r
  datapath_instance_BPU_instance_U356/ZN (NAND3_X1)       0.03       1.12 f
  datapath_instance_BPU_instance_U94/ZN (OR2_X1)          0.06       1.18 f
  datapath_instance_BPU_instance_U1664/ZN (NAND2_X1)      0.03       1.21 r
  datapath_instance_BPU_instance_U1666/ZN (OAI211_X1)     0.04       1.25 f
  datapath_instance_BPU_instance_U1667/ZN (INV_X1)        0.03       1.28 r
  datapath_instance_BPU_instance_U1668/ZN (NAND4_X1)      0.05       1.33 f
  U6202/ZN (NAND2_X1)                                      0.03       1.37 r
  U6204/ZN (NAND2_X1)                                      0.04       1.40 f
  IRAM_ADDR[23] (out)                                      0.03       1.43 f
  data arrival time                                                   1.43

  clock CLK (rise edge)                                    2.00       2.00
  clock network delay (ideal)                              0.00       2.00
  clock uncertainty                                       -0.07       1.93
  output external delay                                   -0.50       1.43
  data required time                                                  1.43
  --------------------------------------------------------------------------
  data required time                                                  1.43
  data arrival time                                                  -1.43
  --------------------------------------------------------------------------
  slack (MET)                                                         0.00


1
```

Listing E.8: sa_syn_custom.rpt

```
Design        Wire Load Model            Library
------------------------------------------------
RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
                       5K_hvratio_1_1    NangateOpenCellLibrary
```

```
Global Operating Voltage = 1.1
Power-specific unit information :
    Voltage Units = 1V
    Capacitance Units = 1.000000ff
    Time Units = 1ns
    Dynamic Power Units = 1uW    (derived from V,C,T units)
    Leakage Power Units = 1nW


  Cell Internal Power  =  11.1056 mW   (88%)
  Net Switching Power  =   1.5750 mW   (12%)
                         ---------
Total Dynamic Power    =  12.6806 mW  (100%)

Cell Leakage Power     = 518.3163 uW


                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000            0.0000            0.0000            0.0000  (   0.00%)
memory             0.0000            0.0000            0.0000            0.0000  (   0.00%)
black_box          0.0000            0.0000            0.0000            0.0000  (   0.00%)
clock_network      0.0000            0.0000            0.0000            0.0000  (   0.00%)
register       1.0324e+04          313.6996        2.4932e+05        1.0887e+04  (  82.48%)
sequential         0.0000            0.0000            0.0000            0.0000  (   0.00%)
combinational    781.7789        1.2613e+03        2.6899e+05        2.3121e+03  (  17.52%)
--------------------------------------------------------------------------------------------------
Total          1.1106e+04 uW     1.5750e+03 uW     5.1832e+05 nW     1.3199e+04 uW
```

Listing E.9: gatecount_pr_standard.rpt

```
Gate area 0.7980 um^2
Level 0 Module RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4 Gates=     34574 Cells=   12845 Area=   27590.1 um^2
```

Listing E.10: sa_pr_standard.rpt

```
*---------------------------------------------------------------------------------------
*   Innovus 17.11-s080_1 (64bit) 08/04/2017 11:13 (Linux 2.6.18-194.el5)
*
*
*   Date & Time:    2020-Feb-22 19:13:58 (2020-Feb-22 18:13:58 GMT)
*
*---------------------------------------------------------------------------------------
*
*   Design: RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
*
*   Liberty Libraries used:
*           MyAnView: /software/dk/nangate45/liberty/NangateOpenCellLibrary_typical_ecsm_nowlm.lib
*
*   Power Domain used:
*       Rail:        VDD    Voltage:        1.1
*
*       Power View : MyAnView
*
*       User-Defined Activity : N.A.
*
*   Switching Activity File used:
*           ../vcd/design.vcd
*           Vcd Window used(Start Time, Stop Time):(-4.95821e-17, -4.95838e-17)
*                     Vcd Scale Factor: 1
**                    Design annotation coverage: 0/13812 = 0%
*
*       Hierarchical Global Activity: N.A.
*
*       Global Activity: N.A.
*
*       Sequential Element Activity: N.A.
*
*       Primary Input Activity: 0.200000
*
*       Default icg ratio: N.A.
*
*       Global Comb ClockGate Ratio: N.A.
*
*   Power Units = 1mW
*
*   Time Units = 1e-09 secs
*
*       report_power -outfile POWER_AFTER_SWITCH -sort total
*
-----------------------------------------------------------------------------------------


Total Power
-----------------------------------------------------------------------------------------
```

```
Total Internal Power:       10.26803181        68.2712%
Total Switching Power:       4.23007558        28.1254%
Total Leakage Power:         0.54194468         3.6033%
Total Power:                15.04005212
-----------------------------------------------------------------------------------------


Group                           Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Sequential                         7.908      0.9923       0.246       9.146       60.81
Macro                                  0           0           0           0           0
IO                                     0           0           0           0           0
Combinational                       2.36       3.238      0.2959       5.894       39.19
Clock (Combinational)                  0           0           0           0           0
Clock (Sequential)                     0           0           0           0           0
-----------------------------------------------------------------------------------------
Total                              10.27        4.23      0.5419       15.04         100
-----------------------------------------------------------------------------------------


Rail                  Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
VDD                       1.1      10.27        4.23      0.5419       15.04         100


-----------------------------------------------------------------------------------------
*   Power Distribution Summary:
*       Highest Average Power:         CLK_r_REG1956_S26 (DFFS_X1):        0.01455
*       Highest Leakage Power:          CLK_r_REG2817_S4 (DFFR_X2):      0.0001324
*       Total Cap:  6.97523e-11 F
*       Total instances in design: 12845
*       Total instances in design with no power:     0
*          Total instances in design with no activity:     0
*       Total Fillers and Decap:     0
-----------------------------------------------------------------------------------------
```

Listing E.11: gatecount_pr_custom.rpt

```
Gate area 0.7980 um^2
Level 0 Module RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4 Gates=     35124 Cells=   13353 Area=   28029.2 um^2
```

Listing E.12: sa_pr_custom.rpt

```
*-----------------------------------------------------------------------------------------
*   Innovus 17.11-s080_1 (64bit) 08/04/2017 11:13 (Linux 2.6.18-194.el5)
*
*
*   Date & Time:    2020-Feb-21 18:48:34 (2020-Feb-21 17:48:34 GMT)
*
*-----------------------------------------------------------------------------------------
*
*   Design: RISCV_BPU_TAG_FIELD_SIZE8_BPU_SET_FIELD_SIZE3_BPU_LINES_PER_SET4
*
*   Liberty Libraries used:
*           MyAnView: /software/dk/nangate45/liberty/NangateOpenCellLibrary_typical_ecsm_nowlm.lib
*
*   Power Domain used:
*       Rail:        VDD    Voltage:        1.1
*
*       Power View : MyAnView
*
*       User-Defined Activity : N.A.
*
*   Switching Activity File used:
*           ../vcd/design.vcd
*           Vcd Window used(Start Time, Stop Time):(1.68509e-28, 1.68509e-28)
*                     Vcd Scale Factor: 1
**                    Design annotation coverage: 0/14321 = 0%
*
*       Hierarchical Global Activity: N.A.
*
*       Global Activity: N.A.
*
*       Sequential Element Activity: N.A.
*
*       Primary Input Activity: 0.200000
*
*       Default icg ratio: N.A.
*
*       Global Comb ClockGate Ratio: N.A.
*
*   Power Units = 1mW
*
*   Time Units = 1e-09 secs
```

```
*
*       report_power -outfile Power_switching_activity_pr -sort total
*
-----------------------------------------------------------------------------------------


Total Power
-----------------------------------------------------------------------------------------
Total Internal Power:       10.31512987        67.8396%
Total Switching Power:       4.33535706        28.5124%
Total Leakage Power:         0.55469462         3.6481%
Total Power:                15.20518158
-----------------------------------------------------------------------------------------


Group                           Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
Sequential                         7.919       0.992      0.2473       9.159       60.23
Macro                                  0           0           0           0           0
IO                                     0           0           0           0           0
Combinational                      2.396       3.343      0.3074       6.047       39.77
Clock (Combinational)                  0           0           0           0           0
Clock (Sequential)                     0           0           0           0           0
-----------------------------------------------------------------------------------------
Total                              10.32       4.335      0.5547       15.21         100
-----------------------------------------------------------------------------------------


Rail                  Voltage   Internal   Switching     Leakage       Total  Percentage
                                Power      Power         Power         Power  (%)
-----------------------------------------------------------------------------------------
VDD                       1.1      10.32       4.335      0.5547       15.21         100


-----------------------------------------------------------------------------------------
*   Power Distribution Summary:
*       Highest Average Power: datapath_instance_RF_instance_U2326 (INV_X2):       0.02393
*       Highest Leakage Power:                 FE_RC_1_0 (INV_X16):      0.0002309
*       Total Cap:  7.16402e-11 F
*       Total instances in design: 13353
*       Total instances in design with no power:     0
*          Total instances in design with no activity:     0
*       Total Fillers and Decap:     0
-----------------------------------------------------------------------------------------
```