TIGER21X Application-Specific Processor

Technical Documentation

Release: 2020

Contents

1 Introduction **3**

2 Functional Description **4**
2.1 Instruction Set Architecture (ISA) 4
2.2 Machine Cycle 6

3 Control Unit **8**
3.1 States 8
3.2 Control Word 9
3.3 IF stage control signals 10
3.4 ID stage control signals 10
3.5 OP stage control signals 10
3.6 EXE stage control signals 10
3.7 MEM1 stage control signals 11
3.8 MEM2 stage control signals 11
3.9 MEM3 stage control signals 11
3.10 WB stage control signals 11

4 Datapath **11**
4.1 Registers 11
4.2 Register File 12
4.3 ALU 13
4.4 16-to-32 Extender 13
4.5 Branch Logic 13

5 Memories **15**

1 Introduction

The aim of the following document is to present the **TIGER21X** processor architecture. TIGER21X is an Application-Specific Integrated Circuit (ASIC) designed for specific reduced functionalities.
TIGER21X is a **32-bit architecture** with a **load/store ISA (Instruction Set Architecture)**. This means that all arithmetic operations are performed once both operands are loaded in the register file from the main memory, and the result itself is written in the register file first, and only in a second time, with a dedicate instruction, it is stored in memory. TIGER21X has a set of **32-bit general-purpose registers**, from `r0` to `r31`, used for hosting intermediate processed data. `r0` always assume the value `0x00000000` and cannot be written.
TIGER21X is thought to work in a **Harvard** architecture environment: instructions and data are stored into two different and dedicate storage devices. All the instructions are encoded on 32-bit words.
In TIGER21Xm all stages of the instruction execution are perfomed **serially** and with no pipelining. Each single instruction execution starts when all the stages of the previous instructions are fully completed.
There are **8 stages** during the execution of an instruction, each one taking **one clock cycle**. In the **Instruction Fetch stage (IF)**, the instruction is loaded from the instruction memory and stored in a dedicate register. Once here, it is interpretated by the control unit of the processor that senses its **opcode** and computes the **control word** to be delivered to datapath. During the **Instruction Decode stage (ID)**, the fields of the instructions are divided into source and destination register addresses, immediate value (if any) and other options. In the **Read Operands stage (OP)**, register file is addressed with the fields obtained by the instruction encoding and register values are read. Then there is the actual **Execution stage (EXE)**, in which the Arithmetic and Logic Unit (ALU) of the processor works. The ALU of the TIGER21X is able to compute modular addition, modular multiplication, modular exponentiation and some types of comparisons (equal, greater, lower) for unsigned 32-bit numbers.
There are then 3 consecutive **Memory Operations stages (MEM)**, in which addresses for instruction and data memory are set, forwarded to the extern and results are awaited. For example, here a load instruction can be performed by sending to the memory the address just computed by the ALU, or a store operation can be done with a value coming from the register file. The **Write-Back stage (WB)** is the last stage and it is encharged of writing of the register file with data coming either from memory or from the ALU output registers.
The pinout of the TIGER21X chip and its connection with the two memories (program and data) is shown in Figure 1.

TIGER21X interfaces with the external world with the following ports:

- a 32-bit instruction address bus which carries out the content of the Program Counter (PCOUT)
- a 32-bit instruction bus which carries in the fetched instructions every new machine cycle (INSTR)
- a 32-bit address bus for addressing the data memory (ADDRESS)
- two 32-bit data bus for data input and data output from/to the data memory (DATAIN, DATAOUT)
- a control signal to enable the data memory (MEM_EN)
- a control signal to inform the memory that writing operation is intended (MEM_WR)

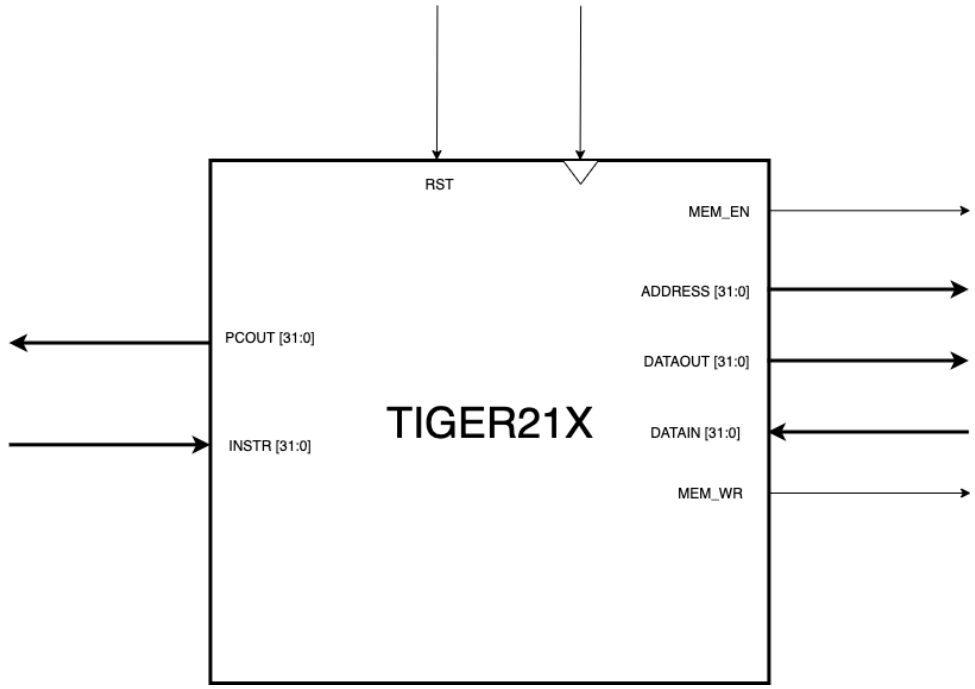

Figure 1: Pinout of TIGER21X.

- fundamental clock (CLK) and reset (RST) signals

The TIGER21X minimal environment is composed by the microprocessor and the two memories, for data and for instructions.
Both memories store words of 32-bit. No littler data type is allowed.
Clock and reset signals are common for all the environment: every component of the system is thus naturally synchronized. Reset signal is **active high** and **synchronous**.

2 Functional Description

The present section summarizes the functional features of the TIGER21X microprocessor.

2.1 Instruction Set Architecture (ISA)

TIGER21X supports in total **17 instructions**, all encoded on 32-bit words. The most significant 6 bits are reserved for the **OPCODE**, the unique identifier that classifies them. Instructions of the TIGER21X ISA belong to three different **types**:

1. **R-type instructions**: these instructions take two values from two data registers (RS1 and RS2) and use the ALU to compute one of the supported operation between them. The result is then stored in a destination register (RD). There are 32 registers in the register file, so they are specified using **5 bits**. The remaining 11 bits are unused.

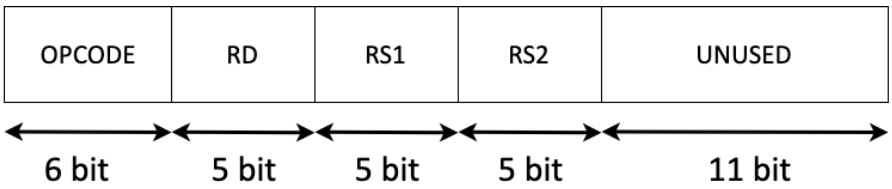

Figure 2: R-type instruction format.

2. **I-type instructions**: these instructions just specify two register operands (RD and RS1), while the third is an **immediate value** on 16 bits, that is then extended to 32 by the datapath. The meaning of RS and RD is decided by single opcodes.

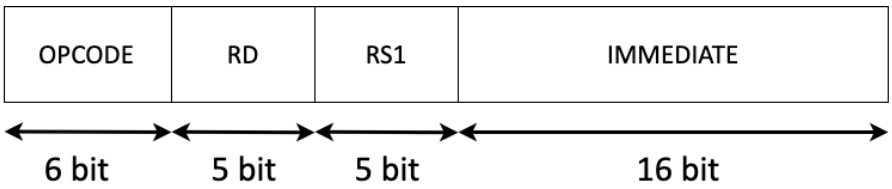

Figure 3: I-type instruction format.

The table below lists the entire instruction set for the TIGER21X application-specific processor. The following legenda is to be applied:

- `<rd>`: instruction bits [25:21]
- `<rs1>`: instruction bits [20:16]
- `<rs2>`: instruction bits [15:11], R-type only
- `<imm16>`: instruction bits [15:0], always extended to 32 bits before operation
- `DMEM[address]`: data memory location at address
- `+`: modular addition
- `*`: modular multiplication
- `^`: modular exponentiation
- `condition ? action1 : action2`: if the condition is verified, then take action1, otherwise take action2

Mnemonic	Type	Description
add <rd>, <rs1>, <rs2>	R	<rd> := <rs1> + <rs2>
addi <rd>, <rs1>, <imm16>	I	<rd> := <rs1> + <imm16>
beq <rs1>, <imm16>	I	if <rs1> == 0, pc := pc + 1 + <imm16>
bne <rs1>, <imm16>	I	if <rs1> != 0, pc := pc + 1 + <imm16>
bra <imm16>	I	pc := pc + 1 + <imm16>
exp <rd>, <rs1>, <rs2>	R	<rd> := <rs1> ^ <rs2>
expi <rd>, <rs1>, <imm16>	I	<rd> := <rs1> ^ <imm16>
ldr <rd>, <imm16>(<rs1>)	I	<rd> := DMEM(<rs1> + <imm16>)
mul <rd>, <rs1>, <rs2>	R	<rd> := <rs1> * <rs2>
muli <rd>, <rs1>, <imm16>	I	<rd> := <rs1> * <imm16>
seq <rd>, <rs1>, <rs2>	R	<rs1> == <rs2> ? <rd> := 1 : <rd> := 0
seqi <rd>, <rs1>, <imm16>	I	<rs1> == <imm16> ? <rd> := 1 : <rd> := 0
sgt <rd>, <rs1>, <rs2>	R	<rs1> > <rs2> ? <rd> := 1 : <rd> := 0
sgti <rd>, <rs1>, <imm16>	I	<rs1> > <imm16> ? <rd> := 1 : <rd> := 0
slt <rd>, <rs1>, <rs2>	R	<rs1> < <rs2> ? <rd> := 1 : <rd> := 0
slti <rd>, <rs1>, <imm16>	I	<rs1> < <imm16> ? <rd> := 1 : <rd> := 0
str <imm16>(<rs1>), <rd>	I	DMEM[<rs1> + <imm16>] := <rd>

2.2 Machine Cycle

This subsection is intended to present the basic functioning cycle of the TIGER21X, going through the behavioral presentation of each of the stages of the datapath.
After the assertion of the reset signal, the TIGER21X enters in a preliminary **two-cycle stage** in which:

- all internal registers and flip-flops are reset;
- the register PC (Program Counter) is reset and forwarded to the instruction memory;
- the first instruction of the memory is fetched inside the processor.

At that point, the basic execution cycle is repeated as long as the processor detects a **fault** (see later). The following 8 stages are performed one after the other, taking one clock cycle each.

1. **Fetch (IF)**: Register IR (Instruction Register) is loaded with the input coming from the instruction memory. The value of the register feeds a considerable amount of blocks:
   - The most significant 6 bits (IR[31:26]) feed the **control unit**, as they represent the **opcode** of the instruction. Internally, the control unit processes this value to compose the **control word** containing all the control signal for the datapath for all the following stages;
   - The intermediate bits (IR[25:11]) are used to feed registers used as address registers for the **register file**, indicating destination operand (RD), first source register operand (RS1) and second source register operand (RS2). All these are **5-bit** registers, as they can address from r0 to r31;
   - For those opcode requiring an immediate operand, the last 16 bits (IR[15:0]) are used to feed the 16-bit immediate value register (RIMM). The content of this register is then transformed into a 32-bit value by the datapath (padding with 0 the higher part);
2. **Decode (ID)**: RS1, RS2, RD2 and RIMM are loaded. Read signals for the register file are sent by the control unit. The register file is thus activated and within the next clock tick reads the required locations possibly indicated by the values of RS1 and RS2. The value of RIMM

is **extended** to 32 bits to correctly fit the next stages. For this purpose, it feeds a **16-to-32 extender** combinational block.

3. **Read Operands (OP)**: This stage is just allocated for the register file operations. By the end of the cycle, the register file writes its output ports, OUT1 and OUT2.

4. **Execute (EXE)**: Registers REGA (Register A), REGB (Register B) and REGIMM (Register of operand IMMediate), which are used as input registers of the ALU, are loaded. Register NPC (Next Program Counter) is loaded with value of register PC + 1, to indicate the next instruction pointer. The first multiplexer MUXA (MUltipleXer of operand A) decides between the value of REGA, containing the value of the register whose address was written in RS1, and the value of NPC. This last value is only chosen in case the instruction is a *branch*, to be combined with an immediate 16-bit offset. The second multiplexer MUXB (MUltipleXer of operand B) decides between the value of REGB, containing the value of the register whose address was written in RS2, and the extended immediate value contained in REGIMM. The two selectors signals for the two muxes come from the control unit. The multiplexers' outputs feed the ALU as operands. The ALU behavior is controlled by the control unit through a signal that give commands about the arithmetic or comparison operation performed by the ALU (ALU_OPCODE). Value of REGA is also analyzed by a **comparator to 0** that outputs an high signal if the register REGA is 0. This is an important information to detect if a branch is to be taken or not, in case the instruction is a conditional branch. This signal, along with other three signals coming from the control unit, are combined to compute a boolean value which indicates if a branch is to be taken or not (BRANCHREGD) (see subsection 4.5). BRANCHREGD will be sampled by the flip-flop BRANCHREG (BRANCH REGister) at the end of the stage.

5. **Memory Operations 1 (MEM1)**: Register ALU_OUTREG is loaded. Control signals for reading and writing the data memory are set by the control unit. ALU_OUTREG value is used as address, since in the *load/store* operations, the ALU is used to compute the address, while REGB value is possibly used as input data for the memory.
   The filp-flop BRANCHREG is loaded and works as selector for BRANCH_MUX (BRANCH MUltipleXer) that selects between ALU_OUTREG output (if set) and the value of NPC (if reset).

6. **Memory Operations 2 (MEM2)**: This stage is just allocated for the data memory operations. By the end of the cycle, the data memory writes its output port, which is DATAIN for the TIGER21X, or writes its internal content according to the commands.

7. **Memory Operations 3 (MEM3)**: Register LMD (Load Memory Data) is loaded with data present on DATAIN input data bus. PC is loaded according to the value decided by BRANCH_MUX and exit the processor through the PCOUT port, which feeds the instruction memory. The control unit sends command for the multiplexer WB_MUX (Write-Back MUltipleXer) to select between two possible values: LMD or ALU_OUTREG.

8. **Write Back (WB)**: Register WB_REG (Write-Back REGister) is loaded. Its value is the data input for the register file. The control unit sends the write command for the register file if needed. The address register indicating the destination register has been already loaded during the ID stage. By the end of the cycle, the register file will be written according to these values. The stage is also used for the operations of the instruction memory, which samples the value of the PCOUT address port. By the end of the cycle, instruction bus INSTR will be written with the encoding of the next instruction.

3 Control Unit

The control unit of TIGER21X is the main block of the processor, the one able to understand which is the just-fetched instruction stored in the register IR, and, according to it, to generate the correct **control word** for the datapath. The control word is the set of all signals that are required along the datapath to commit the instruction: **clock enable signals for the registers**, **multiplexer selectors** and any other type of command for the datapath combinational blocks that process data. Such signals are multi-bit signals, i.e. they express a **code** to give the right command to the corresponding block.
The control word of TIGER21X has **26 control signals** in total. One of them, ALU_OPCODE, is on 3 bits, so the total amount of bits is **28**.
When IR loads a new instructions, the opcode (IR[31:26]) field is read by the control unit to compute the control word. All control words for all instructions are saved inside the control unit in the form of **microcode ROM**, accessed by the opcode value.
Once read, the control word is split in 8 parts and saved in 8 different internal registers. Each registers groups the signal of each one of the 8 stages. The output of the control unit to the datapath are driven by the bits of each of these registers.
The microcode ROM is read during the IF stage, therefore the command signals for that stage are **set by default to active**, as they only relate enable signals for register file addresses. In the microcode ROM, the segment relative to the IF stage is thus not saved.
The 7 internal registers are released to the datapath one after the other, while the machine advances from cycle to cycle.
The control unit is a completely **synchronous** block, so it is programmed to react to rising edge of the clock signal only.

3.1 States

Internally, the control unit is designed as a FSM (Finite-State Machine) which at each clock cycle, can be in one of the following states:

- **OFF**: Unknown state, entered when powered on. When here, the unit only waits for a reset.
- **START1**: The unit migrates to this state when the reset is sensed. Here, the PC loads its first value.
- **START2**: This state is left for the first instruction to be read from instruction memory.
- **FETCH**: Control signals for the IF stage are sent to the datapath.
- **DECODE**: Control signals for the ID stage are sent to the datapath.
- **READ_OPERANDS**: Control signals for the OP stage are sent to the datapath.
- **EXECUTE**: Control signals for the EXE stage are sent to the datapath.
- **MEMORY1**: Control signals for the MEM1 stage are sent to the datapath.
- **MEMORY2**: Control signals for the MEM2 stage are sent to the datapath.
- **MEMORY3**: Control signals for the MEM3 stage are sent to the datapath.
- **WRITEBACK**: Control signals for the WB stage are sent to the datapath. At next clock cycle, the unit migrates again to FETCH.
- **FAULT**: This state is entered when an illegal instruction or the `trap` instruction is fetched. It can be only left through a reset.

Figure 4 shows the schematic of the TIGER21X control unit, listing the ports over which the control word is mapped and highlighting the stages in which each signal is delivered to the datapath.

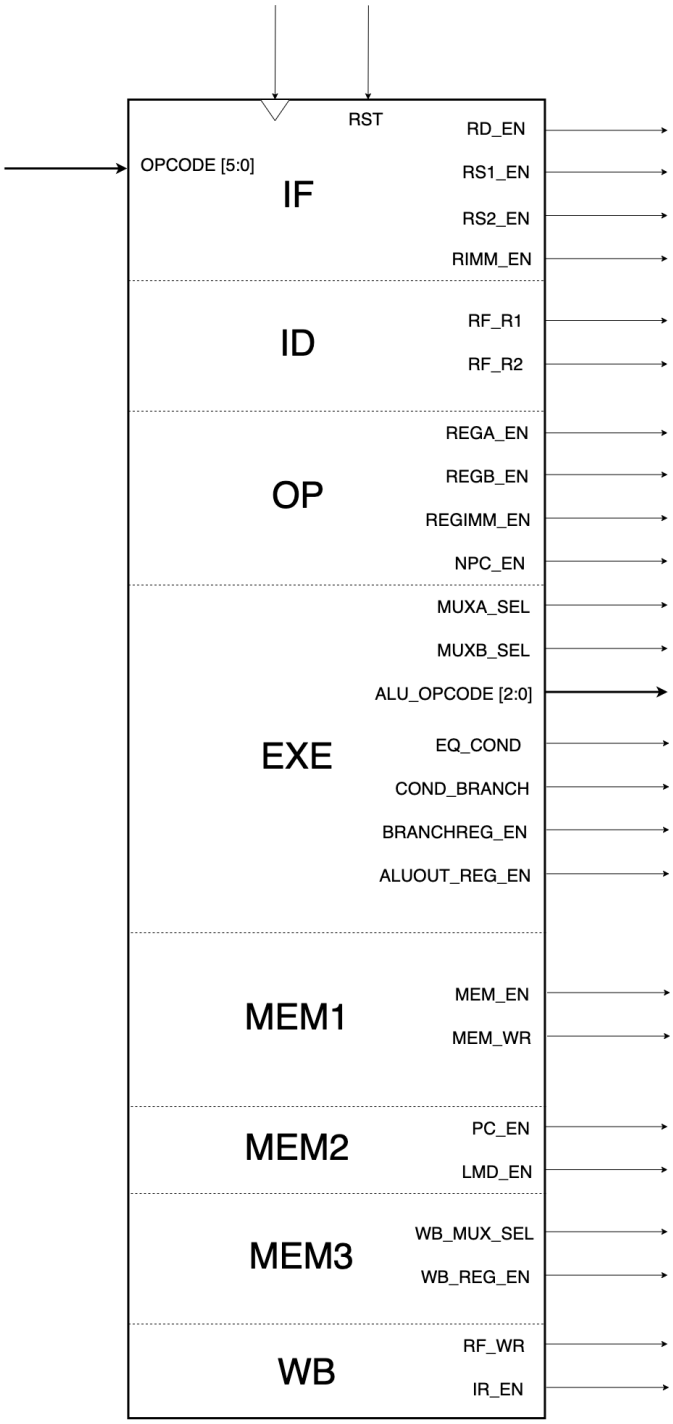

Figure 4: TIGER21X control unit.

3.2 Control Word

The present subsection is intended to list all the signals delivered by the control unit to the datapath stage by stage. The most part of them is constituted by **register enable signals**. To ensure a correct loading, the enable signals are to be activated in the clock cycle *before* the cycle of their

update. This is the reason why they are delivered during the (i-1)-th stage for the registers working in the i-th stage.

3.3 IF stage control signals

- **RD_EN** : enables the update of register RD at the next stage
- **RS1_EN** : enables the update of register RS1 at the next stage
- **RS2_EN** : enables the update of register RS2 at the next stage
- **RIMM_EN** : enables the update of register RIMM at the next stage

3.4 ID stage control signals

- **RF_R1** : enables the reading of the register file at the location indicated by the value of RS1 on port OUT1
- **RF_R2** : enables the reading of the register file at the location indicated by the value of RS2 on port OUT2

3.5 OP stage control signals

- **REGA_EN** : enables the update of register REGA at the next stage
- **REGB_EN** : enables the update of register REGB at the next stage
- **REGIMM_EN** : enables the update of register REGIMM at the next stage
- **NPC_EN** : enables the update of register NPC at the next stage

3.6 EXE stage control signals

- **MUXA_SEL** : if `0`, selects content of register REGA; if `1`, selects content of register NPC
- **MUXB_SEL** : if `0`, selects content of register REGB; if `1`, selects output of register REGIMM
- **ALU_OPCODE[2:0]** : feeds the ALU to indicate which is the arithmetic or comparison operation to be performed. Allowed values are:
  - **ALU_ADD** (`000`) for the modular addition
  - **ALU_MUL** (`001`) for the modular multiplication
  - **ALU_EXP** (`010`) for the modular exponentiation
  - **ALU_SEQ** (`011`) for the EQ (equal) comparison
  - **ALU_SLT** (`100`) for the LT (lower-than) comparison
  - **ALU_SGT** (`101`) for the GT (greater-than) comparison
- **EQ_COND** : assumes value `1` when the instruction is a `beq`, `0` if it is a `bne`
- **COND_BRANCH** : assumes value `1` only in case of conditional branch instruction (`bne`, `beq`)
- **NCOND_BRANCH** : assumes value `1` only in case of non-conditional branch instruction (`bra`)
- **BRANCHREG_EN** : enables the update of register BRANCHREG at the next stage
- **ALUOUT_REG_EN** : enables the update of register ALUOUT_REG at the next stage

3.7 MEM1 stage control signals

- **MEM_EN** : activates read and write operations on the data memory. If no write signal is activated along with it, operation is intended to be a read
- **MEM_WR** : indicates that a write operation must be performed and the entire word stored in REGB is to be written in memory

3.8 MEM2 stage control signals

- **LMD_EN** : enables the update of register LMD at the next stage
- **PC_EN** : enables the update of register PC at the next stage

3.9 MEM3 stage control signals

- **WB_MUX_SEL** : selects the register content to be written back in register file. It assumes value 0 when LMD is selected, 1 when ALU_OUTREG is selected
- **WB_REG_EN** : enables the update of register WB_REG at the next stage

3.10 WB stage control signals

- **RF_WR** : enables the writing of the register file
- **IR_EN** : enables the update of register IR at the next stage

4 Datapath

This section is intended to present the details about the datapath of TIGER21X.

4.1 Registers

TIGER21X has in total **14** datapath registers, excluded the 32 general-purpose registers in the register file. Their update is controlled by the control unit through **clock enable (CE) signals**.

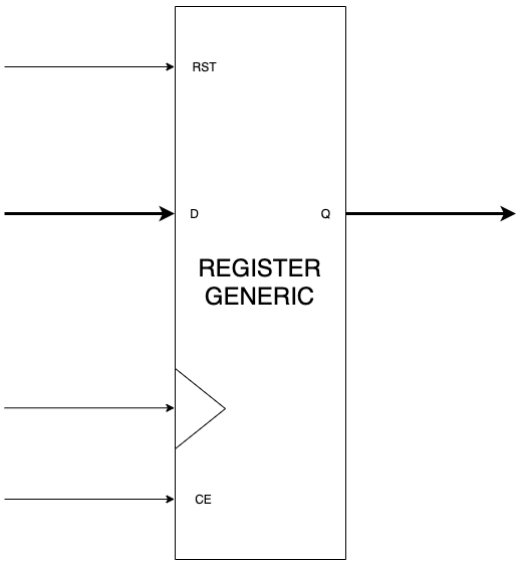

Figure 5: A TIGER21X register.

The Figure 5 shows the ports of the generic register used in the datapath. Each register has its own **reset** signal, as well. Following, a table reporting all the registers of the DLX_BASIC is reported, listing the name of each register, its dimension, at what stage they are updated and the description.

Name	Dimension (in bits)	Stage	Description
IR	32	IF	Contains the just-fetched instruction to be executed
RS1	5	ID	Contains the code of the first source register of the instruction
RS2	5	ID	Contains the code of the second source register of the instruction
RIMM	16	ID	Contains the immediate value encoded in the instruction
RD	5	ID	Contains the code of the destination register of the instruction
REGA	32	EXE	Contains the value of the first source register of the instruction
REGB	32	EXE	Contains the value of the second source register of the instruction
REGIMM	32	EXE	Contains the immediate value to be used in the instruction (extended to 32 bits)
NPC	32	EXE	Contains the PC value of the next instruction that is to be executed
BRANCHREG	1	MEM1	Stores a logic 1 when the next instruction address has been computed by the ALU
ALU_OUTREG	32	MEM1	Contains the result of the ALU, can be used either to address the memory or to carry on the result along the pipeline
LMD	32	MEM3	Contains the word read from the memory
PC	32	MEM3	Contains the address of the next instruction to be loaded
WB_REG	32	WB	Contains the result of the instruction, to be written back in the register file

4.2 Register File

This register file is a synchronous device, hosting the 32 general-purpose registers of the processor. Operations are triggered by the three commands RD1_EN, RD2_EN and WR_EN. At clock rising edge, these three signals are analyzed to detect if an operation is needed. If one of these is active, the register file is waken up and reads the address corresponding to the given command (ADDR_RD1, ADDR_RD2 or ADDR_WR) and possibly also the input data (DATAIN). The device is thought to complete its operation in 1 clock cycle time: within the next rising edge, ports OUT1 and OUT2, that respectively feed REGA and REGB, are supposed to be correctly written so the values can be correctly sampled, or DATAIN is supposed to be correcly stored.

A reset pin (RST) is also present. The reset signal is synchronous, and sets all registers from `r0` to `r31` to value `0x00000000`.
Figure 6 shows the ports of the TIGER21X register file.

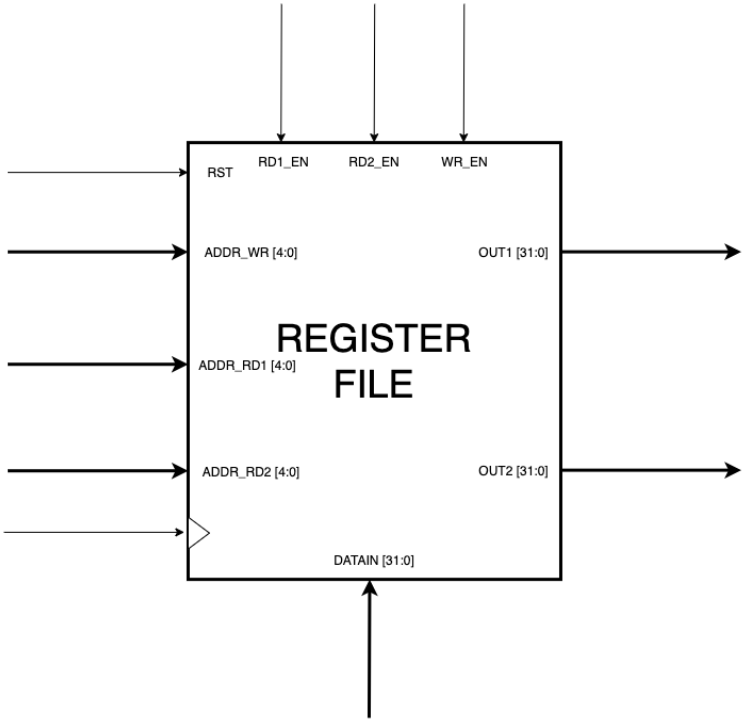

Figure 6: The register file of TIGER21X.

4.3 ALU

The Arithmetic and Logic Unit (ALU) of TIGER21X is able to perform **6** different operations: **modular addition, modular multiplication, modular exponentiation** and **EQ, GT and LT comparisons**. Comparison is implemented through a custom combinational logic that immediately produces the value 0 or 1 depending on the result of the comparison between two operands.
All supported ALU opcodes are listed in 3.6.
Figure 7 shows the schematic of the TIGER21X ALU.

4.4 16-to-32 Extender

The 16-to-32 Extender is a little piece of combinational logic that stand between the two registers RIMM and REGIMM. It has the purpose to **extend** to 32 bits the immediate value that is reported in the instruction, padding the higher part with 0s.
Figure 8 shows the schematic of the TIGER21X Immediate Interpreter.

4.5 Branch Logic

Figure 9 shows the part of the datapath of TIGER21X deciding the next value of the Program Counter. This value is controlled by the BRANCH_MUX, which selects between two values:

- the value of the register NPC (Next Program Counter), which is equal to the previous value of PC plus 1, selected when the instruction **is not** a branch instruction;

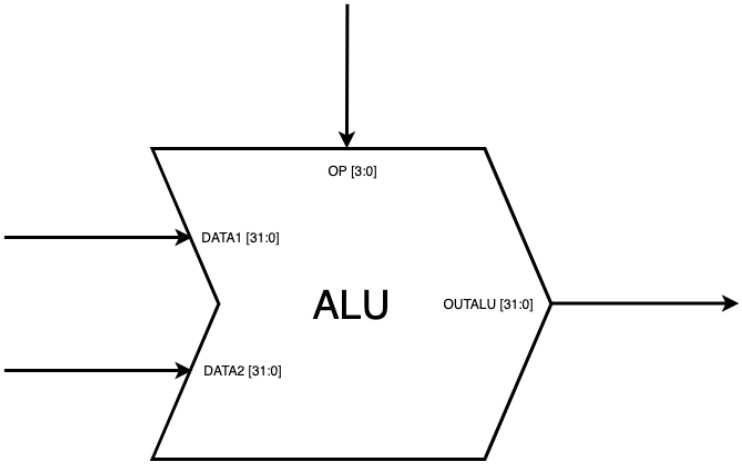

Figure 7: Schematic of TIGER21X ALU.

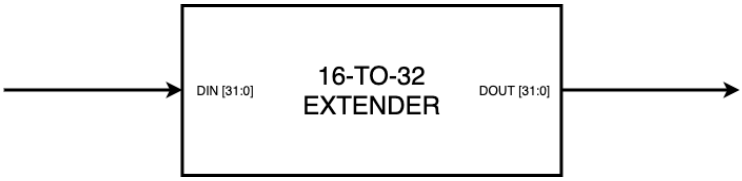

Figure 8: Schematic of the 16-to-32 Extender.

- the value of the register ALUOUT_REG, which contains the addition between the value of NPC and of an immediate encoded in the instruction, selected when the instruction **is** a branch instruction.

The selector of the multiplexer is the output of the BRANCHREG flip flop, which registers a boolean value whose meaning is ”take a branch”. As any other register of the datapath, BRANCHREG is updated each machine cycle at the MEM1 stage, after the ALU operations. The previous logic actually decides if it contains a `0` or a `1`.
At EXE stage, the control unit delivers 3 important signals:

- **EQ_COND**: in case of conditional branch instruction, assumes value `1` when the instruction is a `beq`, `0` if it is a `bne`. It has no meaning when any other instruction is executed;
- **COND_BRANCH**: indicates whether the instruction is a conditional branch instruction (`bne`, `beq`);
- **NCOND_BRANCH**: indicates whether the instruction is a non-conditional branch instruction (`bra`).

These three signals are processed by a custom logic to determine the presence and the type of the branch. To take the decision, the custom logic needs a fourth signal, **REGA_ZERO**, which is the output signal of a **0-comparator** which processes the value of register REGA. REGA contains the value of the register depending on which the decision is to be taken.
The net shown in Figure 9 is a **sum-of-products** logic net, that computes the next value of BRANCHREG as following. Starting from the upper AND gate going to the lower,

- The first AND gate combines **NCOND_BRANCH** and the complement of **COND_BRANCH**. The result is then `1` if that instruction is a branch instruction with no conditions to be tested (`bra`).

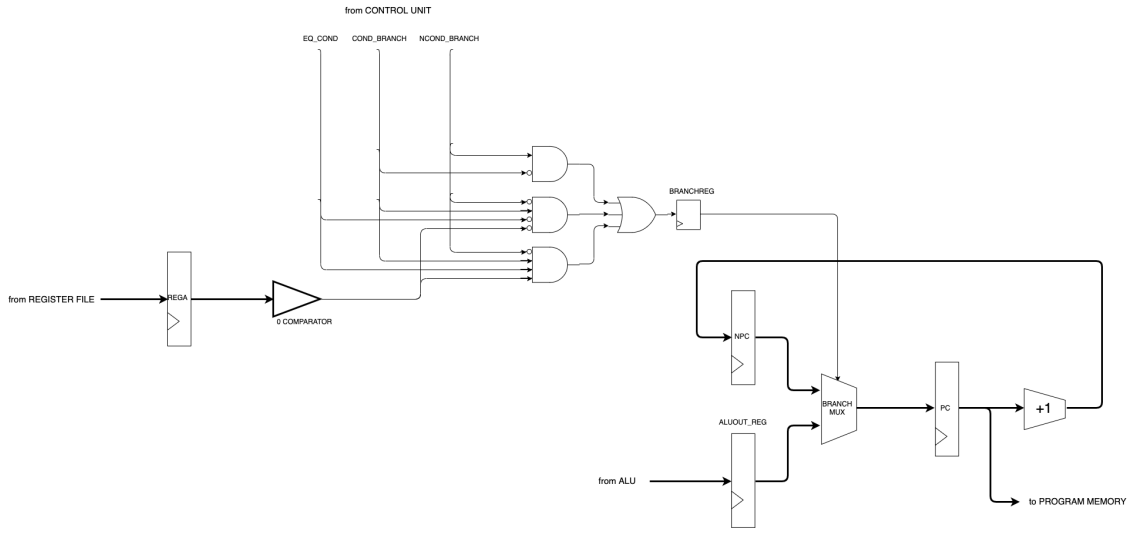

Figure 9: The parts of the datapath that deals with the update of a new value for the Program Counter.

- The second AND gate combines the complement of **NCOND_BRANCH**, **COND_BRANCH** and the complement of both **EQ_COND** and **REGA_ZERO**. The result is then 1 if that instruction is a bne instruction with a non-zero register argument.
- The third AND gate combines the complement of **NCOND_BRANCH**, **COND_BRANCH**, **EQ_COND** and **REGA_ZERO**. The result is then 1 if that instruction is a beq instruction with a zero register argument.

All these three conditions are equally sufficient to determine if the branch is to be taken or not, i.e., if the PC is to be loaded with a value computed by the ALU. If one of these three conditions is true, a branch is to be taken. This is the reason of the final OR gate producing the content of register BRANCHREG.

5 Memories

This section focuses on the memory devices operating with TIGER21X.
TIGER21X has a 32-bit parallelism, which means that the space that can be addressed by the processor is **4 Giga-words** wide. Moreover, TIGER21X is thought to work in a **Harvard** architecture environment, so this space is doubled: 4 Giga for instructions and 4 Giga for data.
Both program and data memory are **32-bit** memories, i.e., with each row and address corresponding to a 4-byte data. Addresses are then referred to words, and not to bytes, when expressed. Therefore, the total memory space available for each block is **16 GB**.
The Figure 10 shows the interconnections between TIGER21X and the essential memory devices, the **program memory** and the **data memory**. Clock and reset signals are common for all the three entities.
The program memory is a **read-only** memory which allows an external entity just to express an address and receive a 32-bit data. The data memory is instead a **read-write** memory, offering all the interface signals to correctly manage reads and writes. If the enable signal is activated alone, then the operation is to be intended as a read, while writes, a dedicate write signal is available on the interface.
Both the memories are clocked entity which respond in **one-clock-cycle time** to the processor

request. At every clock rising edge, the memories respond on the basis of the signals which are stable on their input ports. This means that a previous clock cycle is needed for warranty to stabilize the signals coming from the processor.
As well, another clock cycle is to be taken at the end of the memory operation to wait for the stabilization of the output signal and their correct sampling by internal registers of the processor. This is why, in total, every memory operation takes **3 clock cycles**. This is the reason of the three stages of memory inside the machine cycle, and this is the reason of updating PC register at stage MEM3 to correctly sample IR at stage IF.

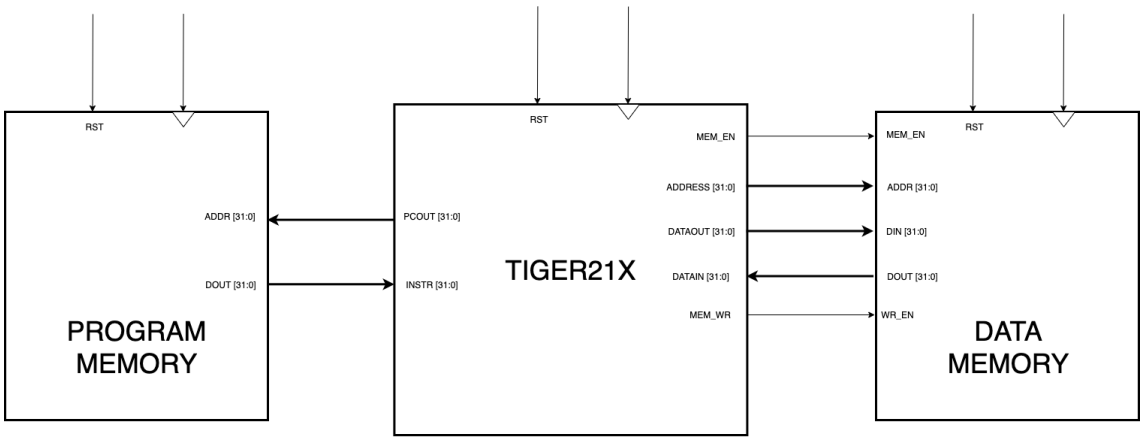

Figure 10: Connections between memories present in the project and the TIGER21X.