1 Introduction

This document describes the internal structure of the pipeline stages of the TinyRISC processor. There are two versions of the TinyRISC processor: the full version and a prototype version, called Reduced TinyRISC (rTinyRISC). The pipeline stages are structurally identical in both implementations. Full TinyRISC and rTinyRISC differ only on the width of the external address buses and data bus.

2 The Fetch Stage

Figure 1 shows the internal datapath of the fetch stage.

![Figure 1: Datapath of the fetch stage.](image-url)
The components of the fetch stage are:

- **f_PC_REG**: program counter. This 32-bit register stores the address of the instruction being fetched in the current cycle;

- **f_NPC_REG**: next program counter. This 32-bit register stores the address of the instruction to be fetched in the next cycle. On the end of an instruction fetch, the program counter and the next program counter are updated in the following way:

  \[
  \text{f_PC_REG} \leftarrow \text{f_NPC_REG} \\
  \text{f_NPC_REG} \leftarrow \text{f_NPC_REG} + 1
  \]

- **f_MUX2**: a 2-input 32-bit multiplexer. This multiplexer selects the address which is loaded into the program counter. The possible inputs are:
  - input 0: the value of the next program counter. This input is selected during normal instruction fetch cycles.
  - input 1: the target instruction address. This input is selected when a taken control transfer instruction is in the decode stage;

- **f_INCR**: a 32-bit incrementer. It receives the address to be stored into PC and increments it by 1, producing the value to be stored into the NPC register;

- **f_TEMP_REG**: temporary register. This is a latch which stores the instruction on IB and the value in f_PC_REG when the IR signal is negated. This latch is necessary to preserve the last instruction fetched while the pipeline is stalled due to a data cache miss or a DMA-control instruction;

- **fPIPE_REG**: a positive edge-triggered pipeline register;

- **f_PIPE_LATCH**: a latch which is enabled during the low clock half-cycle. It is necessary to make the source register numbers available to the data register file before the beginning of the next clock cycle.

Figure 2 shows the input/output signals of the fetch stage’s control unit.

```
f_control

\[\begin{array}{ccc}
\text{clock} & \text{ir} \\
\text{reset} & \text{ready} \\
\text{insack} & \text{fstall} \\
\text{xstall} & \text{p stall}
\end{array}\]
```

Figure 2: Signals of the fetch stage’s control unit.
The signals to/from the control logic are:

- **CLOCK**: the external clock signal;
- **RESET**: the external reset signal;
- **INSACK**: the instruction acknowledgement signal from the instruction cache memory;
- **XSTALL**: when asserted, this signal indicates that the execute stage can not complete its operation on the current cycle;
- **IR**: this signal enables an instruction access in the instruction cache memory;
- **READY**: it indicates that a new instruction has been fetched;
- **FSTALL**: it indicates that the fetch stage has stalled, due to an instruction cache miss.

3 The Decode Stage

Figure 3 shows the internal datapath of the decode stage. The components of the decode stage are:

Figure 3: Internal structure of the decode stage.
- **DATA_REG_FILE**: the data register file. It contains 16 32-bit registers, with two read ports and one write port;

- **D_EXTENDER**: a 16-bit to 32-bit extender. Depending on the instruction, it zero-extends or signal-extends the value in the *imm* field of the instruction being decoded. For the LDUI instruction, it shifts the value in the *imm* field to the upper half of the extended word, zeroing the lower half. It provides the value 0x00000002 at its output when a JAL instruction is in the decode stage;

- **D_MUX3_SRC1**: a 3-input 32-bit multiplexer. This multiplexer selects the source of the first operand. The possible inputs are:
  
  - input 0: data forwarded from the write-back stage. This input is selected when there is a data dependency between the instructions in the write-back stage and in the decode stage;
  - input 1: data forwarded from the execute stage. This input is selected when there is a data dependency between the instructions in the execute stage and in the decode stage;
  - input 2: a value from the data register file.

- **D_MUX4_SRC2**: a 4-input 32-bit multiplexer. This multiplexer selects the source of the second operand. The possible inputs are:
  
  - input 0: a value from the data register file;
  - input 1: data forwarded from the write-back stage. This input is selected when there is a data dependency between the instructions in the write-back stage and in the decode stage;
  - input 2: data forwarded from the execute stage. This input is selected when there is a data dependency between the instructions in the execute stage and in the decode stage;
  - input 3: the extended 32-bit immediate data.

- **D_MUX2_4**: a 2-input 4-bit multiplexer. This multiplexer selects the *sr2* or the *dr* field as the address of the destination register. The *sr2* field is selected only for instructions with an immediate operand.

- **DPIPE_REG**: a positive edge-triggered pipeline register;

- **BRANCH_UNIT**: the functional unit which handles control transfer instructions.

Figure 4 shows the control signals within the decode stage. The signals to/from the instruction decoder **D_INS_DECODER** are:

- **INS.OP** (7 bits): the instruction opcode;

- **IMM_BIT**: the instruction’s *imm* bit;

- **READY**: it indicates that the decode stage has a valid instruction;
Figure 4: Signals of the decode stage’s control unit.

- **ALU.OP**: it indicates that the instruction in the decode stage is an arithmetic/logical instruction;
- **SHT.OP**: it indicates that the instruction in the decode stage is a shift instruction;
- **CMP.OP**: it indicates that the instruction in the decode stage is a comparison instruction;
- **LDII.OP**: it indicates that the instruction in the decode stage is a LDIi instruction;
- **LDUI.OP**: it indicates that the instruction in the decode stage is a LDUI instruction;
- **MEM.OP**: it indicates that the instruction in the decode stage is a memory access instruction;
- **MEMW.OP**: it indicates that the instruction in the decode stage is a write data memory instruction;
- **BRC.OP**: it indicates that the instruction in the decode stage is a compare-and-branch instruction;
- **BRT.OP**: it indicates that the instruction in the decode stage is a BRT instruction;
- **BRF.OP**: it indicates that the instruction in the decode stage is a BRF instruction;
- **JAL.OP**: it indicates that the instruction in the decode stage is a JAL instruction;
- **SIGN.OP**: it indicates that the extender should perform a signed extension;
- **MORPH.OP**: it indicates that the instruction in the decode stage is a MorphoSys instruction;
• MORPH_DMA_OP: it indicates that the instruction in the decode stage is a DMA-control MorphoSys instruction;

• RCRISC_OP: it indicates that the instruction in the decode stage is a MorphoSys RCRISC instruction;

• SR1.VAL: it indicates that the instruction’s SR1 field is valid;

• SR2.VAL: it indicates that the instruction’s SR2 field is valid.

The signals to/from the control unit are (the signals not mentioned so far):

• CUR_SR1: the id of the first operand register;

• CUR_SR2: the id of the second operand register;

• CUR_DR: the id of the destination register;

• X_DR: the destination register of the instruction in the execute stage;

• W_DR: the destination register of the instruction in the write-back stage;

• X_DR_WR: it indicates that the instruction in the execute stage writes into a data register;

• W_DR_WREQ: it indicates that the instruction in the write-back stage writes into a data register;

• KILL: it indicates that the instruction currently in the decode stage should not be forwarded to the execute stage;

• STALL: it indicates that the decode stage should stall;

• SRC1_SEL (2 bits): this signal controls the _MUX3_SRC1 mux;

• SRC2_SEL (2 bits): this signal controls the _MUX4_SRC2 mux;

• DR_RD_WR: when negated, this signal enables a read from the data register file; when asserted, it enables a write into the data register file;

• DR_WREQ: it indicates that the current instruction writes into a destination register;

• READY: it indicates to the next stage that the decode stage has a new decoded instruction.

Figure 5 shows the data path of the branch unit. The components of the branch unit are:

• BU_ADDER: a 32-bit adder. It calculates the target address for conditional control transfer instructions, by adding an offset to the instruction’s address. For the JAL instruction, it adds the value 0x00000002 to the instruction’s address;

• BU_TARG_MUX: a 2-input, 32-bit multiplexer. It selects the source of the target address. The possible inputs are:
input 0: this input is selected in the case of conditional control transfer instructions, for which the target address is calculated by the internal adder;

input 1: this input is selected for the \texttt{JAL} instruction, for which the target address comes from the data register file;

- **BU\textsubscript{COMPARATOR}:** this component sign compares the values in the \texttt{SRC1} and \texttt{SRC2} inputs. Its output is 1 if the condition is true, being 0 otherwise;

- **BU\textsubscript{CONTROL}:** this component controls the modification of the value in the program counter. The \texttt{KILL} signal is asserted for all control transfer instructions except for \texttt{JAL} and indicates that the control transfer instruction should not be forwarded to the next pipeline stage. The \texttt{CHG\_PC} signal controls the modification of the program counter register and it is asserted in the following cases:
  - for a \texttt{BRT} instruction, when the least significant bit of the value in \texttt{SRC1} is 1;
  - for a \texttt{BRF} instruction, when the least significant bit of the value in \texttt{SRC1} is 0;
  - for compare-and-branch instructions, when the output of the comparator is 1;
  - for a \texttt{JAL} instruction.

4 The Execute Stage

Figure 6 shows the datapath of the execute stage. The components of the execute stage are:

- **ARIT\_LOG\_UNIT:** the arithmetic/logical unit. The alu is constituted by four sub-units:
  - **ALU\_ADDSUB:** performs the 32-bit signed add and subtract operations;
Figure 6: Internal structure of the execute stage.

- **alu_logical**: performs the 32-bitwise logical operations;
- **alu_compare**: performs the 32-bit unsigned and signed comparison operations;
- **alu_bypass**: bypasses one of the inputs to the output.

- **shift_unit**: executes logical and arithmetic shifts;
- **mem_unit**: executes the memory access instructions;
- **morph_unit**: the interface logic with the external MorphoSys decoder;
- **x_mux4**: this multiplexer selects the result to be forwarded to the write-back stage. The possible inputs are:
  - input 0: the result comes from the alu. This input is selected for arithmetic, logical, comparison, load-immediate and JAL instructions;
  - input 1: the result comes from the shift unit;
  - input 2: the result comes from the memory unit;
  - input 3: the result comes from the external MorphoSys decoder.

Figure 7 shows the internal structure of the memory unit. It is constituted by the following components:

- **maddr_latch**: it latches the memory address produced in the decode stage;
- **byte_mux**: this multiplexer selects the source of the word offset in a cache line. The possible inputs are:
Figure 7: Organization of the memory unit.

- input 0: word offset from the 32-bit memory address. This is the input normally selected;
- input 1: the word offset produced by the memory unit controller. This input is selected only during a cache line load or cache line write-back operation.

- **BYTE_MUX**: this multiplexer selects the 25 most significant bits for the data address bus. The possible inputs are:
  - input 0: the most significant bits come from the 32-bit memory address. This is the input normally selected;
  - input 1: the most significant bits are the tag bits for the selected line. This input is selected only during a cache line write-back operation.

- **MDIN_LATCH**: it latches the memory data produced in the decode stage;

- **DB_BUFF1**: this 3-state buffer connects the data input from the cache core to the main memory data bus. It is enabled only during a cache line write-back operation, if the internal data cache is enabled;

- **DB_BUFF2**: this 3-state buffer connects the data input from the decode stage to the main memory data bus. It is enabled only when the internal data cache is disabled;

- **MEMU_CONTROL**: it contains a finite state machine and combinational logic, and generates all the control signals for the internal components of the memory unit, for the data cache core and for the external main memory.
The execute stage’s control unit just selects one of the inputs of the $x_{-\text{MUX4}}$ multiplexer according to the instruction currently in the execute stage. The $x_{-\text{STALL}}$ signal is asserted in two conditions: (1) a data cache miss or (2) a DMA-control MorphoSys instruction finds the DMA controller busy.

![Figure 8: Signals of the execute stage’s control unit.](image)

5 The Write-Back Stage

Figure 9 shows the internal structure of the execute stage. The control logic simply asserts the $\text{dr}_{-}\text{wren}$ signal when the $\text{dr}_{-}\text{wreq}$ and $\text{ins}_{-}\text{ready}$ signals are asserted.

![Figure 9: Internal structure of the write-back stage.](image)

6 The Data Cache

The TinyRISC processor includes an internal data cache memory. The data cache controller is embedded in the memory unit’s controller, see Section 4. The data cache core is custom-designed, and its signal interface with the TinyRISC core is shown in Figure 10.
Figure 10: Signal interface between TinyRISC and data cache core.