∇ TRISC

TRISC-16 Datasheet

Features

- 16-bit RISC CPU with 8 general purpose registers.
- 28 simple and fast instructions that can perform various procedures when put together.
- Three-stage instruction execution cycle: fetch, decode and execute (except for stack operations).
- Can address up to 64 KB of program and data memory, as well as peripheral.

Applications

- Learning and teaching computer engineering.
- Simple computations.
- Minimalist embedded applications.

General Description

The TRISC (Training-RISC) was made to facilitate the learning process of computer engineering subjects, like Computer Architecture and Organization, Embedded Systems, Assembly Programming and Digital Systems Design, decreasing the learning curve of these subjects.

Design choices were made to maximize ease of learning, such as a RISC design philosophy, using pure Harvard architecture and port-mapped I/O (PMIO). Some common instructions have been removed to simplify the instruction set and CPU diagram as much as possible. This CPU was also made in VHDL to widen the learning capabilities.

Even so, this CPU is fully functional and can perform the most important operations that more complex processors can also perform, such as: logical, arithmetic, memory access, conditional jumps and I/O operations.

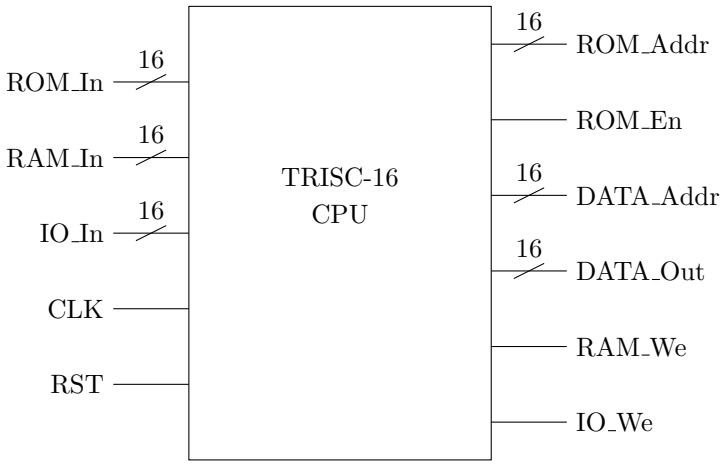

Figure 1: CPU Pinout

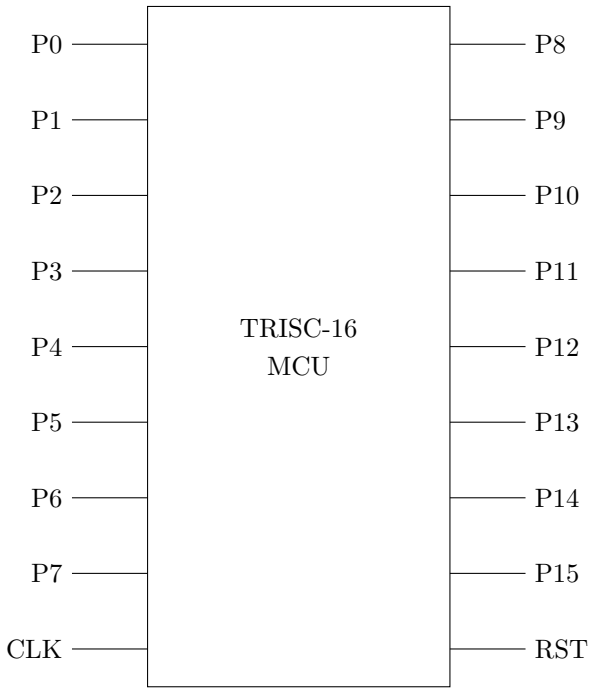

Figure 2: Microcontroller Pinout

TRISC Architecture

The TRISC 16-bit architecture, developed for the TRISC-16 processor, focus on an organization that facilitates the understanding of the processor, having 5 modulated and integrated units: **Control Unit** (controls the processor's other units), **Datapath** (perform the CPU operations), **ROM** (the [asynchronous] program memory), **RAM** (the [synchronous] data memory) and **I/O** (the I/O controller). The functional block diagram of the processor can be seen in the figure bellow.

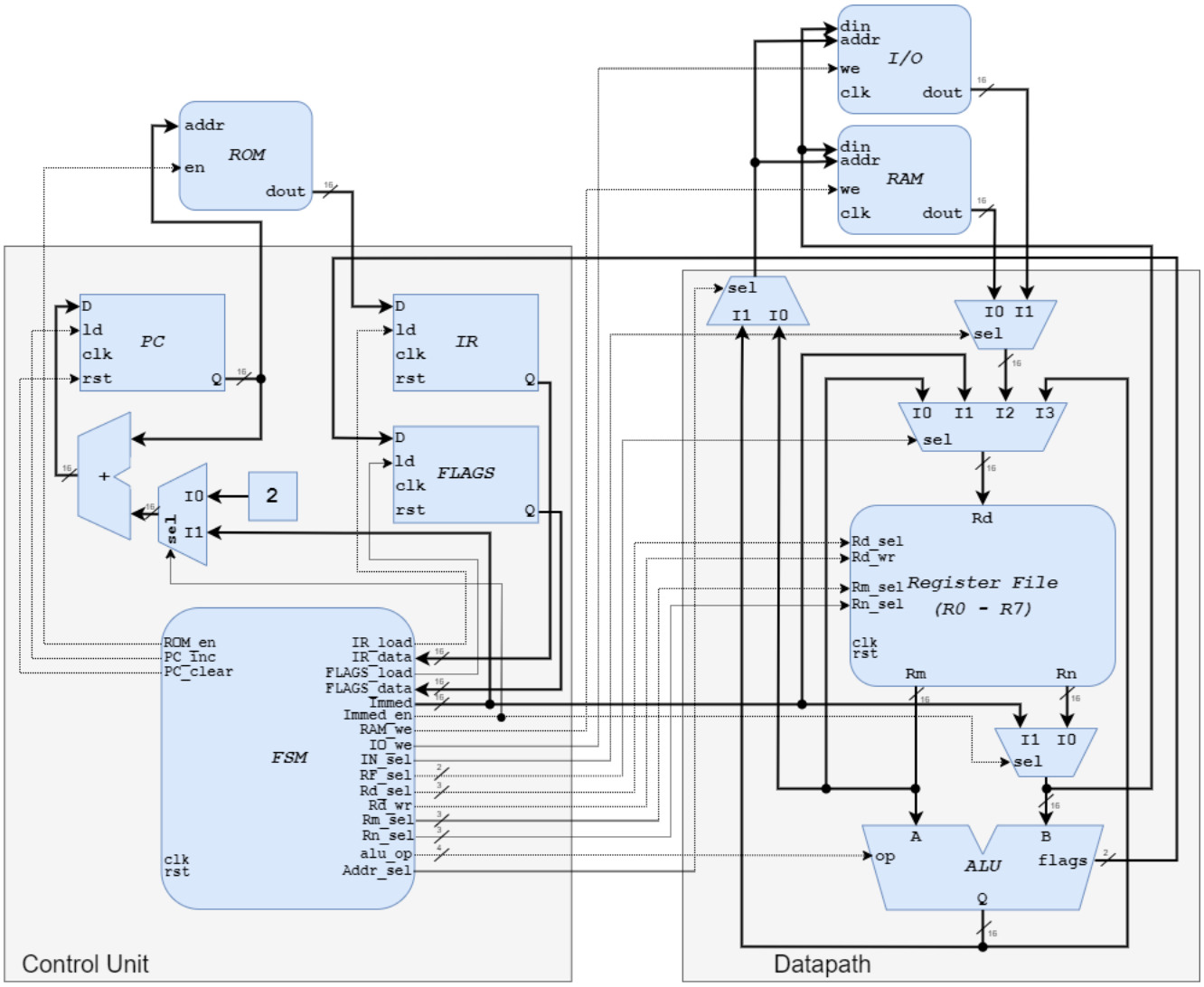

Figure 3: Processor's Block Diagram

Being a 16-bit CPU (not large and complicated as 32-bits, but not insuficient as 8-bits) it can work with 16-bit internal and external data and address buses. It has 8 general purpose 16-bit registers (R7 being treated as the **stack pointer**, or SP).

The two main units are connected, where the Control Unit contains a Finite State Machine-based controller, that issues diverse control signals to the system (such as RAM_we, that indicates whereas the operation is a read or write operation). It also has internal registers, such as the PC (program counter), IR (instruction register) and FLAGS (ALU status register), used for conditional branching.

Technical Details

Bellow are the summarized the technical details of the processor, being the project decisions that brought the TRISC-16 to a state of easier comprehension.

- 16-bit RISC processor made in VHDL, with 28 instructions.
- 8 general purpose 16-bit registers (R0-R7).
- Pure Harvard Architecture (separate buses for RAM and ROM).
- Port Mapped I/O (shared address and data bus with RAM).
- Up to 64 KB of program and data memory.
- Up to 2^{16} peripheral registers (6 used).
- GPIO that controls 16 pins and a 16-bit down Counter peripherals.

Revision Details

Some adjustments were made to the previous revision of the architecture, in order to add more meaningful operations, such as branching and I/O operations. Bellow are the additions to this revision.

- **I/O module** added as the Input/Output controller, supporting up to 2^{16} peripheral registers, and enabled to be written via the new signal **IO_we**. It shares the processor external output data bus and data address bus with the RAM module. The input data bus is multiplexed into the CPU using the new signal **IN_sel** control signal.
- The RAM input multiplexer was moved to inside the datapath, where it now selects whether the Rn register or a immediate will be used in the ALU or in the RAM memory. The FSM issues the **Immed_en** signal (replaced the RAM_sel signal) to select the immediate (when 1) or the register (when 0).
- To support jumping the PC now can be added to a immediate, so a new multiplexer was added. It selects whether the immediate or the standard 2 will be added to PC, via the Immed_en signal.
- To support conditional jumping a **FLAGS register** was added to the control unit to mantain the ALU flags after a logic or arithmetic operation. The ALU now exports two flags to the control unit, Zero and Carry flag, and the FSM module load it in the register via a FLAGS_load signal. The FSM can access it's data via FLAGS_data and performs the new JEQ, JGT and JLT instructions.
- In the register file input multiplexer the I1 and I2 inputs were swaped: now the immediate enters in I1 and the input from RAM and I/O enters in I2. Additionaly, another multiplexer that selects the RAM address was added to aid instructions that calculates it (such as POP), controller by **Addr_sel** signal.
- 14 new instructions were added: **PUSH**, **POP**, **CMP**, **JMP**, **JEQ**, **JLT**, **JGT**, **SHR**, **SHL**, **ROR**, **ROL**, **IN** and **OUT** (register and immediate iterations). The stack instructions (PUSH and POP) uses the R7 register as stack pointer (SP), and also require one additional step in the FSM, to update the stack pointer.

Instruction Set

The Instruction Set Architecture (ISA) of the TRISC-16 is shown bellow, having 28 simple, but fast and usable instructions. These instructions can do almost any operations a simple program require (even embedded, using the I/O ports).

Instruction	Operation	Type	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NOP	nop	NOP	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
PUSH Rn	[SP] = Rn; SP = SP - 2	PUSH	0	0	0	0	0	-	-	-	-	-	-	Rn2	Rn1	Rn0	0	1
POP Rd	SP = SP + 2; Rd = [SP]	POP	0	0	0	0	0	Rd2	Rd1	Rd0	-	-	-	-	-	-	1	0
CMP Rm, Rn	Z = (Rm = Rn); C = (Rm < Rn)	ALU	0	0	0	0	0	-	-	-	Rm2	Rm1	Rm0	Rn2	Rn1	Rn0	1	1
JMP #Im	PC = PC + #Im	BRANCH	0	0	0	0	1	Im8	Im7	Im6	Im5	Im4	Im3	Im2	Im1	Im0	0	0
JEQ #Im	If Z and ¬C, then PC = PC + #Im	BRANCH	0	0	0	0	1	Im8	Im7	Im6	Im5	Im4	Im3	Im2	Im1	Im0	0	1
JLT #Im	If ¬Z and C, then PC = PC + #Im	BRANCH	0	0	0	0	1	Im8	Im7	Im6	Im5	Im4	Im3	Im2	Im1	Im0	1	0
JGT #Im	If ¬Z and ¬C, then PC = PC + #Im	BRANCH	0	0	0	0	1	Im8	Im7	Im6	Im5	Im4	Im3	Im2	Im1	Im0	1	1
MOV Rd, Rm	Rd = Rm	MOV	0	0	0	1	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	-	-	-	-	-
MOV Rd, #Im	Rd = #Im	MOV	0	0	0	1	1	Rd2	Rd1	Rd0	Im7	Im6	Im5	Im4	Im3	Im2	Im1	Im0
STR [Rm], Rn	[Rm] = Rn	STORE	0	0	1	0	0	-	-	-	Rm2	Rm1	Rm0	Rn2	Rn1	Rn0	-	-
STR [Rm], #Im	[Rm] = #Im	STORE	0	0	1	0	1	Im7	Im6	Im5	Rm2	Rm1	Rm0	Im4	Im3	Im2	Im1	Im0
LDR Rd, [Rm]	Rd = [Rm]	LOAD	0	0	1	1	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	-	-	-	-	-
ADD Rd, Rm, Rn	Rd = Rm + Rn	ALU	0	1	0	0	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	Rn2	Rn1	Rn0	-	-
SUB Rd, Rm, Rn	Rd = Rm Rn	ALU	0	1	0	1	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	Rn2	Rn1	Rn0	-	-
MUL Rd, Rm, Rn	Rd = Rm * Rn	ALU	0	1	1	0	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	Rn2	Rn1	Rn0	-	-
AND Rd, Rm, Rn	Rd = Rm AND Rn	ALU	0	1	1	1	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	Rn2	Rn1	Rn0	-	-
ORR Rd, Rm, Rn	Rd = Rm OR Rn	ALU	1	0	0	0	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	Rn2	Rn1	Rn0	-	-
NOT Rd, Rm	Rd = ¬Rm	ALU	1	0	0	1	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	-	-	-	-	-
XOR Rd, Rm, Rn	Rd = Rm XOR Rn	ALU	1	0	1	0	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	Rn2	Rn1	Rn0	-	-
SHR Rd, Rm, #Im	Rd = Rm #Im	ALU	1	0	1	1	1	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	Im4	Im3	Im2	Im1	Im0
SHL Rd, Rm, #Im	Rd = Rm #Im	ALU	1	1	0	0	1	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	Im4	Im3	Im2	Im1	Im0
ROR Rd, Rm	Rd = Rm 1; Rd(15) = Rm(0)	ALU	1	1	0	1	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	-	-	-	-	-
ROL Rd, Rm	Rd = Rm 1; Rd(0) = Rm(15)	ALU	1	1	1	0	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	-	-	-	-	-
IN Rd, Rm	Rd = IO(Rm)	I/O	1	1	1	1	0	Rd2	Rd1	Rd0	Rm2	Rm1	Rm0	-	-	-	0	1
OUT Rm, Rn	IO(Rm) = Rn	I/O	1	1	1	1	0	-	-	-	Rm2	Rm1	Rm0	Rn2	Rn1	Rn0	1	0
OUT Rm, #Im	IO(Rm) = #Im	I/O	1	1	1	1	1	Im5	Im4	Im3	Rm2	Rm1	Rm0	Im2	Im1	Im0	0	0
HALT	halt	HALT	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Table 1: Processor Instruction Set

The instruction set was made to teach the inner workings of a CPU, not to process any meaningful data. Even so, it's fully functional. There are simple instructions with unfilled "bit slots", which anyone using can implement more complex functionalities (in VHDL), add new instructions or replace existing ones.

Instruction Cycles

There are 3 main states for the CPU execution: **fetch** (instruction brought from the ROM to the IR), **decode** (find out which instruction will be executed, that is, the group, and which operands will be used) and **execution** (executes the specific instruction). There are 11 instructions groups with similar instructions. Additionally, there is a initial state, **init**, to setup the correct signals in the reset event.

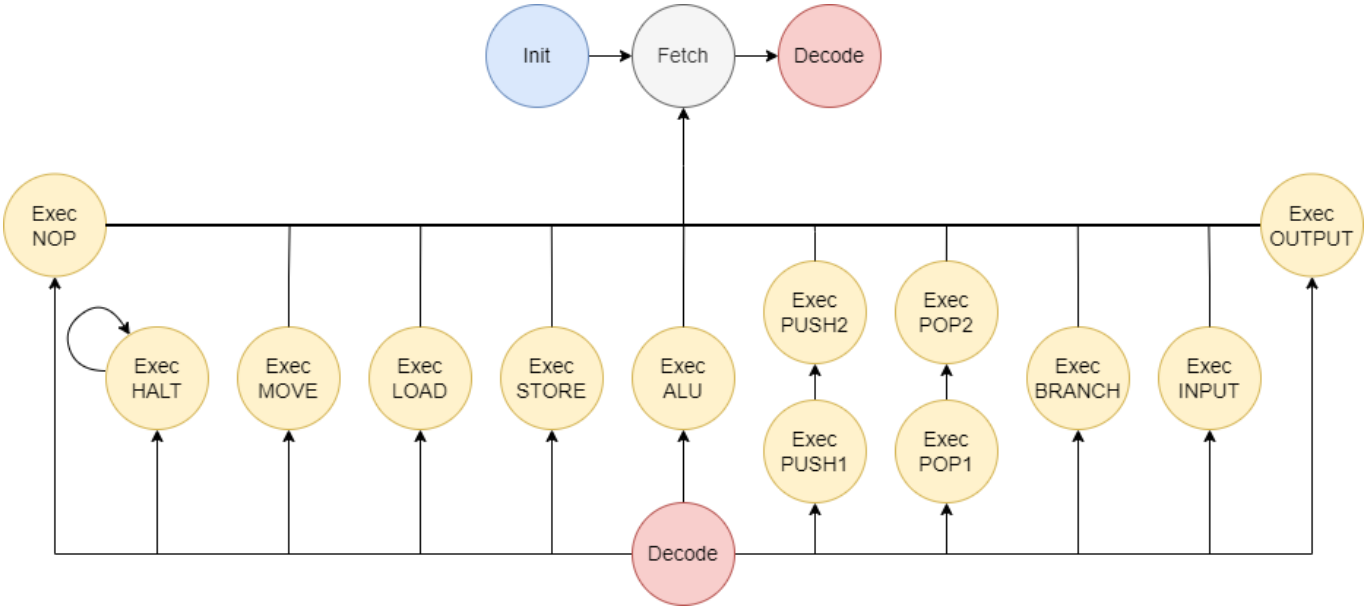

Figure 4: Control Unit Execution Cycles

The stack instructions, PUSH and POP, have an aditional state to update the stack pointer (SP). The state **PUSH1** stores the data in memory, and **PUSH2** increment the stack pointer by 2. In other hand, the state **POP1** first decrement the stack pointer by 2 the the **POP2** loads the data from memory. Therefore the TRISC stack is **empty and decrescent**.

Control Unit Signals

The control signals emitted by the FSM are shown bellow:

Signals	Init	Fetch	Decode	MOV	LOAD	STORE	ALU	BRANCH	INPUT	OUTPUT	PUSH1	PUSH2	POP1	POP2
PC_clr	1	0	-	-	-	-	-	-	-	-	-	-	-	-
PC_inc	0	1	0	-	-	-	-	1 / 0	-	-	-	-	-	-
ROM_en	0	1	0	-	-	-	-	-	-	-	-	-	-	-
IR_load	0	1	0	-	-	-	-	-	-	-	-	-	-	-
FLAGS_load	0	0	-	-	-	-	1	-	-	-	-	-	-	-
Immed	0x0000	-	-	-	-	-	-	-	-	-	-	0x0002	0x0002	-
Immed_en	0	0	1 / 0	-	-	-	-	-	-	-	0	1	1	0
RAM_we	0	0	-	-	-	1	-	-	-	-	1	-	-	-
IO_we	0	0	-	-	-	-	-	-	-	1	-	-	-	-
IN_sel	0	-	-	-	0	-	-	-	1	-	-	-	-	0
RF_sel	00	-	-	01 / 00	10	-	11	-	10	-	-	-	-	10
Rd_sel	000	-	-	-	-	-	-	-	-	-	-	111	111	-
Rd_wr	0	0	-	1	1	-	1 / 0	-	1	-	-	1	1	1
Rm_sel	000	-	-	-	-	-	-	-	-	-	111	111	111	111
Rn_sel	000	-	-	-	-	-	-	-	-	-	-	-	-	-
alu_op	0000	-	-	-	-	-	-	-	-	-	-	0101	0100	-
Addr_sel	0	0	-	-	-	-	-	-	-	-	-	-	1	1

Table 2: Processor Control Signals

These signals are emitted from the control unit to the datapath, ROM, RAM and I/O modules.

There are some control signals issued conditionally to execute a instruction.

- If the bit 11 of the instruction is 1, than it works with immediates, like MOV, STR, SHR, SHL, OUT and branch instructions. If so, the **Immed_en** must be 1 to select the immediate, or 0 otherwise.
- The PUSH and POP instructions work only with registers, but the update states uses the immediate (Immed_en is 1 then) to increment or decrement the stack pointer.
- For the MOV instruction to use the immediate, additionally to Immed_en, the **RF_sel** must be 0b10 to select the immediate to enter the register file. If it has to use the register version, then RF_sel is 0b00 to select Rm.
- Two clock signals are needed to read from RAM to a register (one to read the RAM and another to write to the register), and the POP2 state attempts to do a load in one cycle. To minimize the delay, the POP1 state issued a **Addr_sel** signal to select the address to issue to RAM directly from the ALU (the result of decreascing the SP), rather than waiting for it to be saved to the register and used in POP2. Therefore, the POP1 state issues a signal that will take help the next state.

Peripherals

The TRISC-16 microcontroller has two simple peripherals: a GPIO to control up to 16 pins, and a 16-bit down counter. Simple registers can be used to control the peripherals via the IN and OUT instructions, as is shown bellow:

Peripheral	Address	Register	Offset	Description
GPIO	0x0000	DATADIR	0x00	Sets the pin data direction: 1 to set the pin to OUTPUT, and 0 for INPUT.
		DATAOUT	0x02	Sets the pin data: 1 to drive the pin HIGH, and 0 for LOW.
		DATAIN	0x04	Read the pin data: 1 if the pin is HIGH, and 0 if is LOW.
Counter	0x0006	RELOAD	0x00	Value to be loaded in the counter (0 to 65535)
		CONTROL	0x02	Starts the count with bit 0 (START) and flags the completion with bit 1 (COUNTFLAG).
		COUNT	0x04	Value currently being counted down.

- **GPIO**: Controls pins P0 through P15. Provides basic input and output capabilities, where each bit of it's register associates with a pin.
- **Counter**: Counts downwards, from the value in RELOAD to 0 (upon start by writing 1 to the bit 0 of CONTROL), flaggin in the bit 1 of CONTROL it's completion.