PRCO - Processor Documentation

PRCO304 - Processor Documentation

Ben Lancaster
March 6, 2018

Revision History

Table 1: Document revisions.

Date	Version	Changes
06/03/2018	1.20	Add sections for Status Register, Program Counter.
13/02/2018	1.10	Add Control and Pipeline section.
04/02/2018	1.00	Initial revision. Processor introduction. Initial ISA. Initial Register definitions.

Table of Contents

1 PRCO Processor

The PRCO processor is a soft-microprocessor design targeted for general purpose computing and co-processing.

1.1 Features

- Small, embeddable, Verilog core.
- 16-bit RISC instruction set.
- 16-bit register, ALU, and IO, bus widths.
- 12+12 general purpose IO inputs and outputs.
- 9 special IO pins.
  - 4 PWM pins.
  - 2 RS232 pins.
  - 3 SPI pins.

2 PRCO Architecture

2.1 Registers

PRCO has a total of 6 addressable, read and write, registers. These registers are identified by letters A through H.

2.1.1 General Purpose Registers

Registers A through E are designed for general purpose use and are safe to store user values over the run-time of the processor.

Table 2: General purpose registers.

Registers	Bits	Description
A through E	15:0	5 General purpose registers

Instructions that require a destination register, such as CMP, can reference any register (even special registers if that is your requirement). For the CMP instruction as an example, the processor will put the result of the comparison instruction in the destination register, overwriting any value present in that register.

2.1.2 Special Registers

Registers F through H are special registers within the processor. The processor cannot guarantee that a value written or read in these registers will persist over the run-time of the processor. Erroneously writing to these registers may severely affect program and processor behaviour.

Even though all registers can be used at the will of the programmer, it is recommended to isolate a few registers to provide special features, such as RAM stack management, interrupts, and IO multiplexing.

Table 3: Special registers.

Registers	Bits	Description
F	15:0	Status Register
G	15:0	RAM Stack pointer
H	15:0	RAM Base pointer

Status Register

2.2 Program Counter

2.3 Control and Pipelining

The PRCO processor employs a *feed-forward* pipeline strategy. This pipeline supports:

- Time-varying processes: Multi-clock cycle decoding; Memory access; ALU ops.
- Module re-ordering: Instruction dependencies; Module skipping; Output redirection.
- Interruption (see section 2.4: Interrupts and Exceptions).

As the pipeline is feed-forward, no information is sent back to previous modules to tell them of their status. This means that if a module is stalled (due to mutli-cycle processes or future modules are stalled), and the previous module is ready, the previous module will signal the next module that information is ready and it should take it, but the current module is unable to as it is busy. The pipeline resolves this issue by it's cyclic nature. This means that only 1 module at any time is processing data. Of-course, the downside to this approach is that instruction parallelism is reduced.

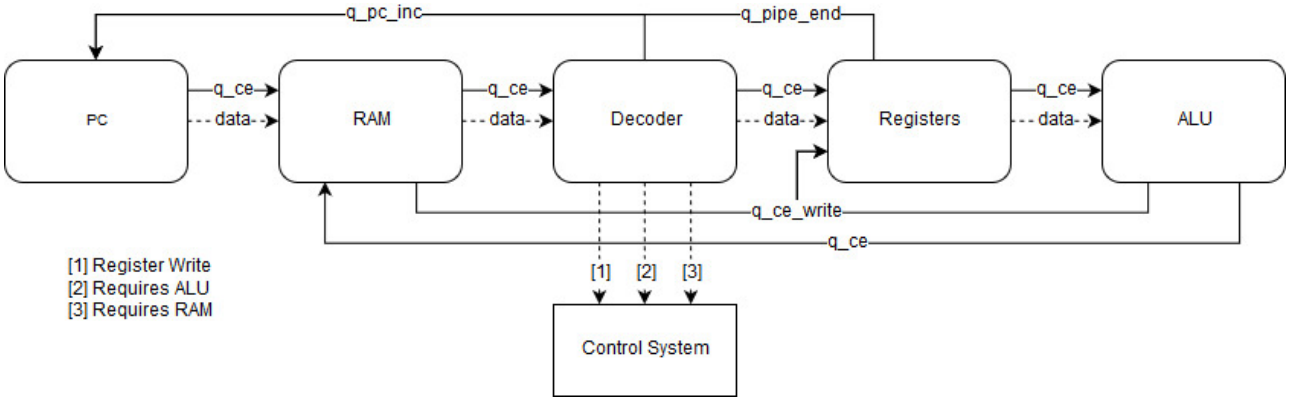

Figure 1: The feed-forward pipeline interconnect diagram used by the PRCO processor.

The pipeline structure is described in figure 1 (above). The general order of the modules is shown from left to right, but this can change due to the pipelines re-ordering functionality.

The Decoder module will decode instruction words from memory and will output appropriate signals containing the requirements of the instruction, such as requiring register write access, any ALU operation, and whether the instructions requires access to internal/external memory.

To improve instruction performance, the decoder can also choose what modules are required and when they are called. For example, for the MOVI (move immediate) instruction the Decoder will assign the following modules in the following order: ALU and Register write, resulting in a total of 5 stages (including PC, Fetch, and Decode). The last module in this pipeline, the Register write, will raise the *q_pipe_end* signal indicating that the pipeline has finished and to start fetching the next instruction.

For the NOP instruction, the decoder identifies that the instruction requires no dependencies and will hence signal the *q_pc_inc* signal resulting in only 3 pipeline stages.

For instructions that require RAM access, a typical pipeline order might look like: PC, Fetch, Decoder, Register Read, ALU, RAM, resulting in 6 stages being used.

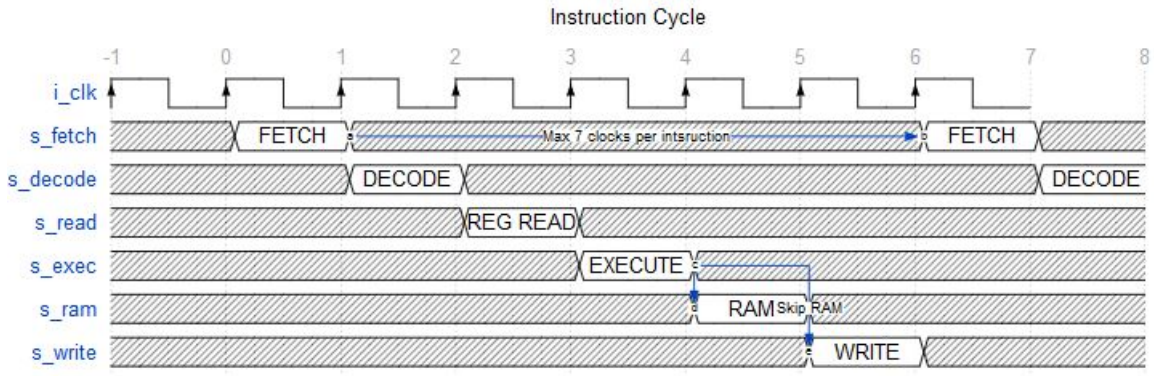

Figure 2: PRCO processor instruction cycle time diagram.

2.4 Interrupts and Exceptions

3 PRCO Instruction Set Architecture

This section describes instructions available on the PRCO processor.

The following instruction definitions use the following letters to describe values: X for any value; 0 for all zeros; 1 for all ones; Imm8 for unsigned 8-bit immediate; Simm5 for signed 5-bit immediate.

3.1 Timings

As the processor does not employ instruction pipelining techniques, but instead uses control signals to individually turn on sub-processes on the CPU. These sub-processes do not happen in parallel.

For instructions that do not require a RAM read/write request, the RAM stage of the control sequence is skipped reducing the instruction cycle by 1 CPU clock for that instruction.

The fastest instruction in terms of CPU cycles is the NOP instruction. The greatest number of cycles for an instruction includes all RAM read/write request operations, such as the LW and SW instructions (see section 3.2).

3.2 General Instructions

The term, general instruction, is given to instructions that are common to primitive operations such as arithmetic and comparison instructions.

3.2.1 Instruction List

Table 4: Number of respondants for treatment

Type 1	15-11	10-8	7-5	4-0	Semantics
Type 2	15-11	10-8	7-0		Semantics
Type 3	15-11	10-8	7-5	4-2	Semantics
NOP	00000	X	X	X	PC <= PC + 1
HALT	X0000	X	X	X	
LW	00001	Rd	Ra	Simm5	Rd <= RAM[Ra + Simm5]
LW.L	X0001	Rd	Ra	Simm5	Rd <= (RAM[Ra + Simm5] & 0xFF)
SW	00010	Rd	Ra	Simm5	RAM[Ra + Simm5] <= Rd
SW.L	X0010	Rd	Ra	Simm5	RAM[Ra + Simm5] <= Rd & 0xFF
MOV	00011	Rd	Ra	X	Rd <= Ra
MOVI	00100	Rd	Simm8		Rd <= Simm8
ADD	01000	Rd	Ra	X	Rd <= Rd + Ra
ADDI	01001	Rd	Simm8		Rd <= Rd + Simm8
SUB	01010	Rd	Ra	X	Rd <= Rd - Ra
SUBI	01011	Rd	Simm8		Rd <= Rd - Simm8
JMP	01100	Rd	X		See JMP.
CMP	01101	Rd	Ra	Rb	Set SR flags

3.2.2 NOP

Description The NOP instruction performs no action for 1 instruction cycle (see section 3.1).

Assembly NOP

Pseudocode

Registers altered

Clock cycles 2 (FETCH, DECODE)

15:11	10:0
00000	X

3.2.3 LW - Load Word

Description Copies a 16-bit word from RAM to a register.

Assembly LW Rd, +4(Ra)

Pseudocode Rd <= RAM[Ra + Simm5]

Registers altered Rd

Clock cycles 6 (FETCH, DECODE, READ, EXECUTE, RAM, WRITE)

15:11	10:8	7:5	4:0
00001	Rd	Ra	Simm5

3.2.4 SW - Store Word

Description Copies a 16-bit from a register to RAM.

Assembly SW Rd, +4(Ra)

Pseudocode RAM[Ra+Simm5] <= Rd

Registers altered None

Clock cycles 6 (FETCH, DECODE, READ, EXECUTE, RAM, WRITE)

15:11	10:8	7:5	4:0
00001	Rd	Ra	Simm5

3.2.5 MOVR

Description The MOVR instruction copies a 16-bit register value to another register.

Assembly MOVR %Ra, %Rd

Pseudocode Rd <= Ra

Registers altered Rd

Clock cycles 5 (FETCH, DECODE, READ, EXECUTE, WRITE)

15:11	10:8	7:5	4:0
00011	Rd	Ra	X

3.2.6 MOVI

Description The MOVR instruction copies a 16-bit register value to another register.

Assembly MOVR %Ra, %Rd

Pseudocode Rd <= Ra

Registers altered Rd

Clock cycles 5 (FETCH, DECODE, READ, EXECUTE, WRITE)

15:11	10:8	7:0
00100	Rd	Imm8

3.2.7 ADD

Description The ADD instruction adds an immediate value to a destination register, Rd.

Assembly ADDI $255, %Rd

Pseudocode Rd <= Rd + Imm8

Registers altered Rd

Clock cycles 5 (FETCH, DECODE, READ, EXEC, WRITE)

15:11	10:8	7:5	4:0
01000	Rd	Ra	X

3.2.8 ADDI

Description The ADD instruction adds an immediate value to a destination register, Rd.

Assembly ADDI $255, %Rd

Pseudocode Rd <= Rd + Imm8

Registers altered Rd

Clock cycles 5 (FETCH, DECODE, READ, EXEC, WRITE)

15:11	10:8	7:0
01001	Rd	Imm8

3.2.9 SUBI

Description The SUB instruction subtracts an immediate value from a destination register, Rd.

Assembly SUBI $255, %Rd

Pseudocode Rd <= Rd - Imm8

Registers altered Rd

Clock cycles 5 (FETCH, DECODE, READ, EXEC, WRITE)

15:11	10:8	7:0
01001	Rd	Imm8

3.2.10 CMP

Description Sets register, Rd, to the value of Ra - Rb.

Assembly CMP Rd, Ra, Rb

Pseudocode Rd <= CMP(Ra, Rb)

Registers altered Rd

Clock cycles 5 (FETCH, DECODE, READ, EXEC, WRITE)

Note Rd should be set to SR (Status Register) as the JMP instruction operates on the SR register.

15:12	11:9	8:6	5:3	2:0
0003	Rd	Ra	Rb	X

3.2.11 JMP

Description Jumps the Program Counter if the condition is met within the Status Register register.

Assembly JMP Rd, Imm8

Pseudocode Program Counter <= Rd if (Status Register & Imm8).

Registers altered None

Clock cycles 5 (FETCH, DECODE, READ, EXEC, BRANCH)

15:11	10:8	7:0
01100	Rd	Imm8

An 8 bit immediate (7-0) can be set in the JMP instruction to create conditional jumps.

Table 5: Conditional jump immediate bits

	15-11	10-8	7-0	Semantics	Status Register
JMP	01100	Rd	0000 0000	Unconditional Jump	Any
JE	01100	Rd	0000 0001	Jump Equal	ZF=1
JNE	01100	Rd	0000 0010	Jump Not Equal	ZF=0
JG	01100	Rd	0000 0011	Jump Greater Than	ZF=0 and SF=OF
JGE	01100	Rd	0000 0100	Jump Greater Than or Equal	SF=OF
JL	01100	Rd	0000 0101	Jump Less Than	SF<>OF
JLE	01100	Rd	0000 0110	Jump Less Than or Equal	ZF=1 or SF<>OF
JS	01100	Rd	0000 0111	Jump Signed	SF=1
JNS	01100	Rd	0000 1000	Jump Not Signed	SF=0

3.3 Special Instructions

4 Compiler

4.1 Features

4.2 Usage

The compiler is invoked using the libprco/cli executable.

4.3 Grammar

4.4 Code Generation