ECE 593: Fundamentals of PreSilicon Validation
Architectural Description: Pipelined Picoblaze

Group 3: Yashodhan Wagle, Ramaa Potnis, Supreet Gulavani

Architecture Overview

The programmer visible architecture of the Rojoblaze is derived from the Picoblaze microcontroller and is therefore identical architecturally and differs from the latter in microarchitecture. Thus, the architecture of the Picoblaze also applies to the Rojoblaze.
It is an 8-bit microcontroller that originally (the Picoblaze) is designed for Spartan-3 FPGAs and the modified Rojoblaze has a more general RTL model that is not optimized for any specific platform. The general architecture is described as:

- 8-bit data path
- 16 addressable registers
- 18-bit instruction word
- Maximum of 1024 instructions supported in a single program
- The architecture features CISC like flags.

Program Support:
The intent of the Picoblaze is to implement state machines and thus has limitations on the program size of the controller. More complex programs that require significantly more memory (for eg. C programs) are handled by the Microblaze. The microcontroller supports a maximum of 1024 instructions that it reads off of a ROM created by that assembler. Thus, the program is restricted to a 10-bit address. This comes in the form of a .vhdl file for actual implementation. For verification purposes, the testbench creates a variable number of fixed length (1024 to be precise) instruction programs to run on the processor.

General Purpose Registers:
All the 16 8-bit registers and general purpose, which means that all of them are addressable, none have a specified purpose and all have the same level of priority. These are named from s0 to sF.

ALU:
The ALU performs arithmetic functions on operands provided by the registers only. The primary operand (sX) is where the value of the operation is returned and if a secondary operand is needed, it can be provided from another register (sY) or as a constant in the instruction itself.

Flags and Control Flow:
The ZERO and CARRY flags are set when the result is all 0s and when the result overflows from the register respectively. The CARRY flag can be set because of an arithmetic operation that overflows or because of a logical operation like shift whose significant bits move beyond the scope of the 8-bit register. These flags can further be used to control the flow of instructions for jumps, loops and subroutine calls.

Reset:
Resets the state of the processor apart from the contents of the registers.

Input/Output:
During an Input operation value at the input port is transferred to one of the 16 registers. Indicated by the signalREAD_STROBE. At an output operation, contents of any of the 16 registers are transferred to the output port. Indicated by the signal WRITE_SIGNAL.

Scratchpad Memory:
This is an internal 64 byte general purpose memory. During any STORE instruction, the contents of any of the 16 registers can be written to the 64 locations.During a FETCH instruction contents of any of the 64 memory locations can be written to any of the 16 registers.

Interrupt:
There is a single interrupt input signal.The interrupts, by default , are masked. An active interrupt

Stack:
There is a hardware stack maintained to track the return address for the subroutine calls and the interrupt handler mechanism. If at all there is an overflow detected, an internal reset gets generated which defaults the program from address 0x0000.

Instruction Set:
Below are the instructions supported by the microcontroller:

Program Control:

- JUMP
- JUMP Z
- JUMP NZ
- JUMP C
- JUMP NC
- CALL
- CALL Z
- CALL NZ
- CALL C
- CALL NZ
- RETURN
- RETURN Z
- RETURN NZ
- RETURN C
- RETURN NC

Arithmetic:

- ADD
- ADDCY
- SUB
- SUBCY
- COMPARE

Interrupts:

- RETURNI ENABLE
- RETURNI DISABLE
- ENABLE INTERRUPT
- DISABLE INTERRUPT

Logical:

- LOAD
- AND
- OR
- XOR
- TEST

Storage:

- STORE
- FETCH

Shift and Rotate:

- SR0
- SR1
- SRX
- SRA
- RR
- SL0
- SL1
- SLX
- SLA
- RL

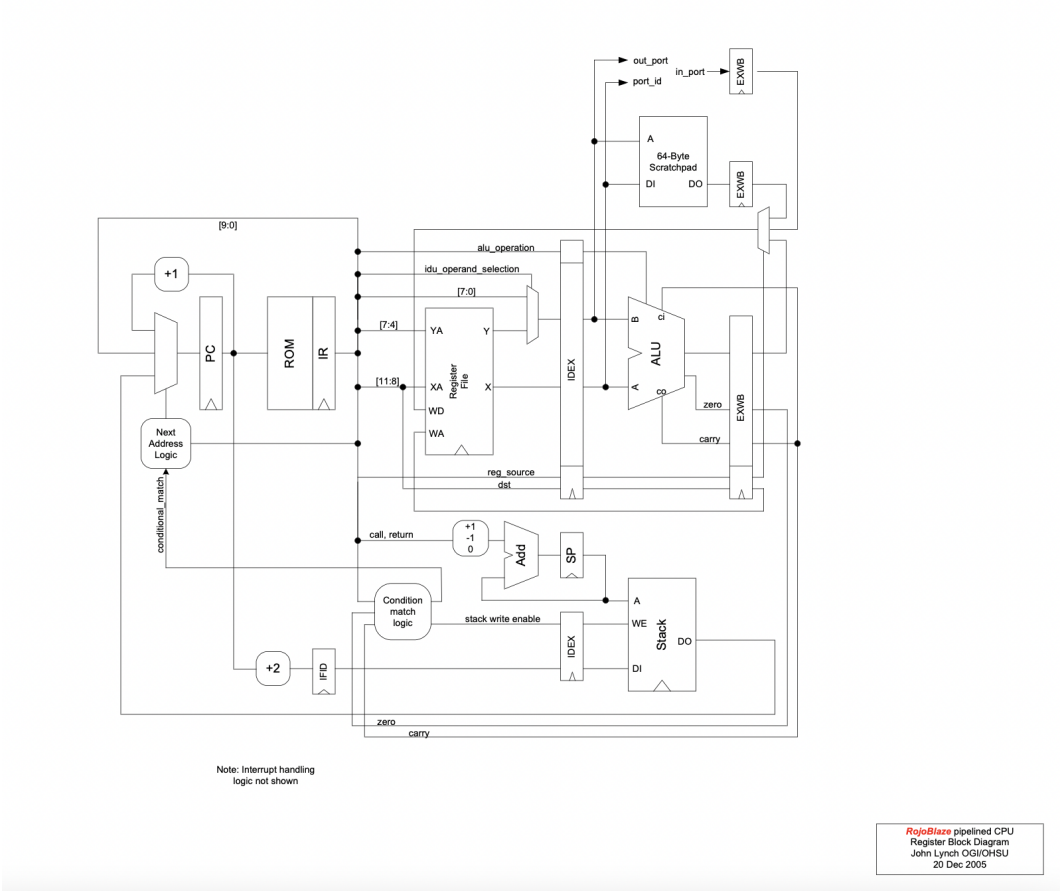
Input/Output:

- INPUT
- OUTPUT

The instructions are pretty self explanatory. We decided that a detailed description of each instruction along with the decoding is out of the scope of this document. A more detailed description of each instruction can be found in the user guide.

MicroArchitecture Overview

The Rojoblaze differs from the Picoblaze in the microarchitectural implementation of the processor. While the Picoblaze guarantees that any operation will be completed in 2 clock cycles, the Rojoblaze, being pipelined in nature, may take more than 2 cycles but runs at much higher clock cycles and thus has better performance while giving up on the 2 cycle instruction completion feature.
The pipeline featured in the Rojoblaze is a 4 stage pipeline with its stages being Instruction Fetch (IF), Instruction Decode (ID), Execute (EX) and Writeback (WB). Since the microcontroller is pipelined, it has 2 versions each with pipeline hazard detection supported and unsupported. Each stage is discussed in detail.



Instruction Fetch Stage:

The Instruction Fetch Stage includes the program counter increment logic along with the ROM containing the program itself that is created by the assembler. The next address logic is responsible to choose the PC increment operation between the increment to the next instruction, jumps or branches and its outcome with the help of instruction opcode and condition outcome for branches.

Instruction Decode Stage:
The Instruction Decode stage breaks apart the instruction into its required sub fields like register code, opcode code and constants if any. Since the Picoblaze and such a small instruction set, it does not require a dedicated decoder and the instruction word can be broken down into the alu operation and idu operation opcode. Apart from the register access and selection of operands for the execute stage takes place. This particular microarchitecture also implements the outcome of condition for branching and subroutine calls in this stage.

Execute Stage:
The Execution stage contains the ALU that performs the logical and arithmetic operations. The operands are preselected and the output along with the flags are saved to that pipeline registers. The stage also contains access to internal (scratchpad) memory and the stack for return address from subroutines.

Writeback Stage:
The last stage is responsible for writing the computed results to the registers and the address of the register that is to be written to.

Data forwarding:
As mentioned above the design also implements data forwarding in the pipeline for better pipeline performance.