Pipelined Microcontroller Processor

Micro Architectural Specifications

Table of contents

Reference to HAS

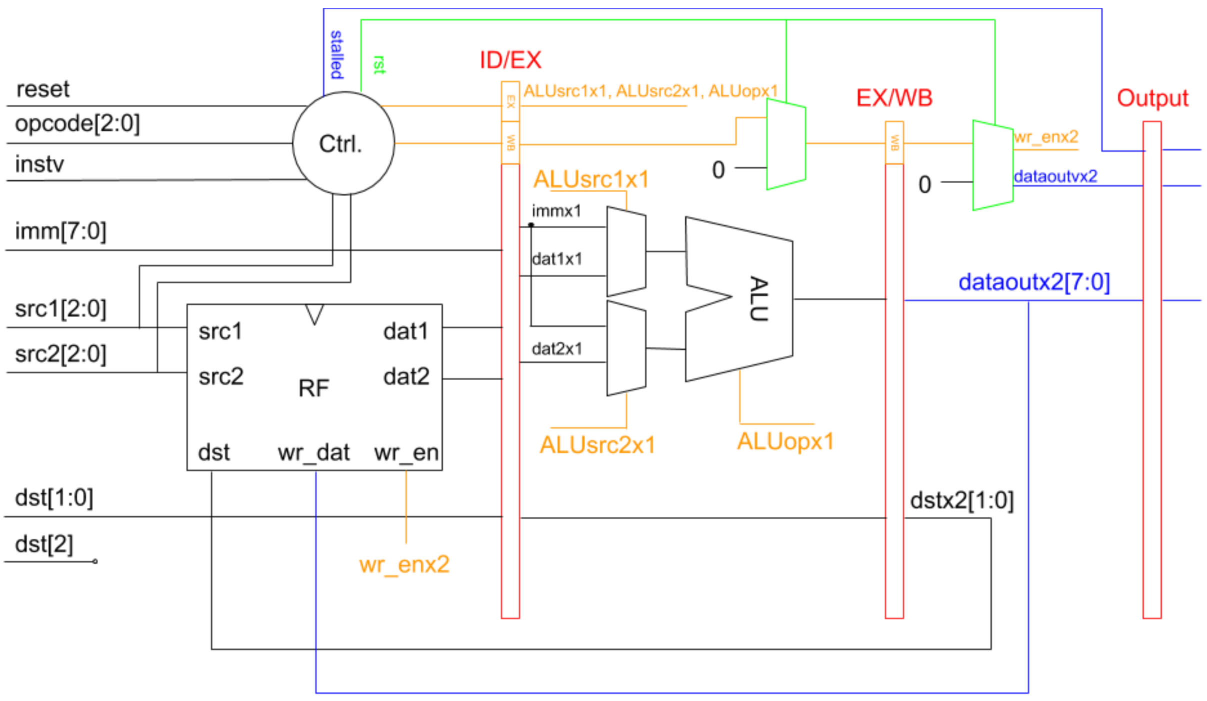
Available in the HAS document are:

- Instruction set
- Interface list

Reading them is crucial for the understanding of the rest of this document.

Top level

Diagram



Components

The system is built out of the following components:

- Memories: RF (register file) & pipeline registers
- ALU (arithmetic logic unit)
- Controller

Pipeline stages

To meet the requirement of a 3-cycle latency, there are 3 pipeline stages:

1. Decode (ID) - generating the appropriate control signals and extracting RF values
2. Execute (EX) - performing the calculation
3. Write Back (WB) - saving the calculation result to the RF

Data path components

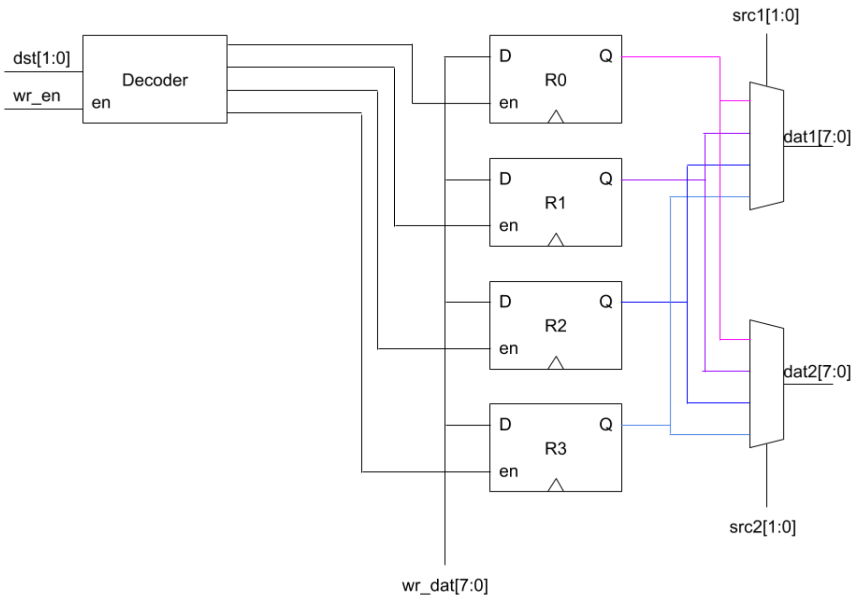
Register file

The RF has four 8-bit write-enabled registers, with 2 read ports and 1 write port.

I/Os

Name	**I/O**	**Top level connection**	**Description**
`clk`	Input	`clk`	Write clock
`src1[1:0]`	Input	`src1[1:0]`	Read address of port 1
`src2[1:0]`	Input	`src2[1:0]`	Read address of port 2
`dst[1:0]`	Input	`dstx2[1:0]`	Write address
`wr_dat[7:0]`	Input	`dataoutx2[7:0]`	Write data
`wr_en`	Input	`wr_enx2`	Write enable
`dat1[7:0]`	Output	`dat1x0[7:0]`	Read data of port 1
`dat2[7:0]`	Output	`dat2x0[7:0]`	Read data of port 2

Diagram



ALU

The ALU is capable of executing modular addition & subtraction, and bitwise NAND, NOR, XOR and shift left operations.

I/Os

Name	**I/O**	**Top level connection**	**Description**
`A[7:0]`	Input	`ALUdatin1x1[7:0]`	Operator 1
`B[7:0]`	Input	`ALUdatin2x1[7:0]`	Operator 2
`function[2:0]`	Input	`ALUfuncx1[2:0]`	Operation
`result[7:0]`	Output	`dataoutx1[7:0]`	Result of operation

Diagram

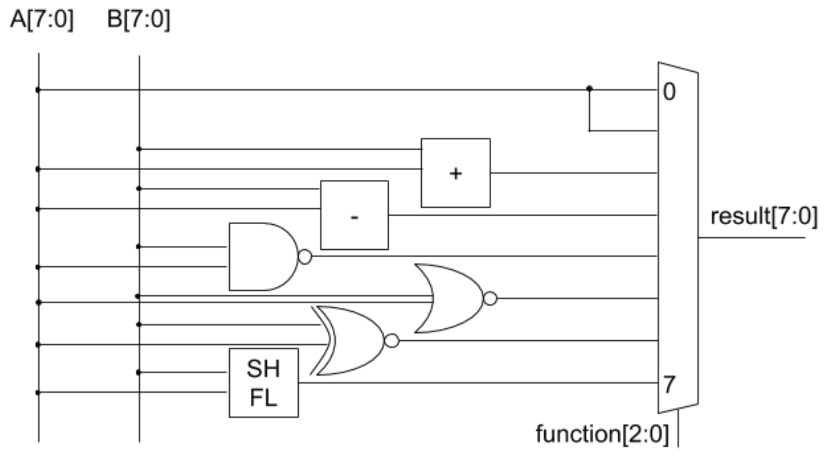


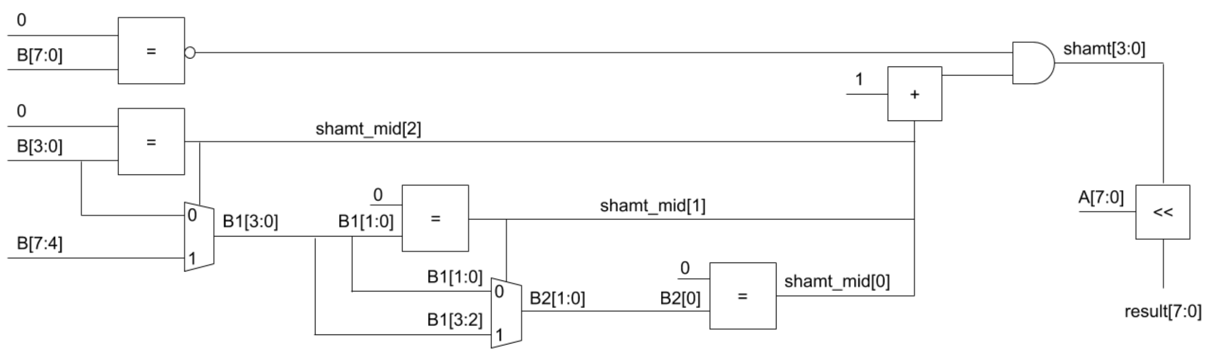
Diagram of SHFL logic

SHFL logic in this design is composed of a combination of:

- Count Leading Zeros - reversed (from the right side)
- Shift Left - can accept shift amount of 8

Notice that usually the shift amount for N bits varies between 0 and N-1, but here the design supports a shift range of 0 to N.

Assertion: shift amount should never pass 8 bits.



Control components

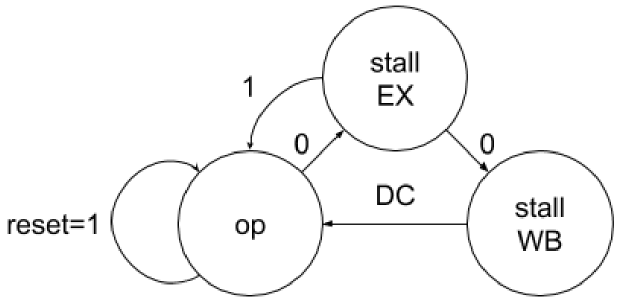
Main controller

The main controller is a Mealy finite state machine that counts stalling cycles.

I/Os

Name	**I/O**	**Top level connection**	**Description**
`reset`	Input	`reset`	Synchronous reset to the machine
`opcode[2:0]`	Input	`opcode[2:0]`	Instruction’s encoding
`instv`	Input	`instv`	Validity of instruction
`src1[2:0]`	Input	`src1[2:0]`	Register identity of exe operator 1
`src2[2:0]`	Input	`src2[2:0]`	Register identity of exe operator 2
`rst`	Output	`rst`	Invalidate output and prevent write
`ALUsrc1`	Output	`ALUsrc1x0`	0=>dat1 1=>imm
`ALUsrc2`	Output	`ALUsrc2x0`	0=>dat2 1=>imm
`ALUop`	Output	`ALUopx0`	Same format as described in HAS
`wr_en`	Output	`wr_enx0`	Enable writing to RF
`dataoutv`	Output	`dataoutvx0`	`dataout[7:0]` is valid
`stalled`	Output	`stalled`	Machine ignores future instructions

Diagram



All transitions in the diagram depend on the value of the `reset` signal.
The meaning of each state is:

- Op - instruction fetching (from inputs) and decoding
- Stall EX - execution stage at work, machine is stalling any other instructions
- Stall WB - write back stage at work, machine is stalling any other instructions

State table

Current state	Current inputs					Next state	Current outputs						
	reset	opcode	instv	src1	src2		rst	ALUsrc1	ALUsrc2	ALUop	wr_en	dataoutv	stalled
op	1	DC	DC	DC	DC	op	1	DC	DC	DC	0	0	0
op	0	DC	0	DC	DC	op	0	DC	DC	DC	0	0	0
op	0	LD	1	IMM/ERR	DC	stall EX/op	0	IMM	DC	LD	1/0	0	1/0
op	0	OUT	1	GPR/ERR	DC	stall EX/op	0	GPR	DC	OUT	0	1/0	1/0
op	0	other	1	IMM/GPR	IMM/GPR	stall EX	0	IMM/GPR	IMM/GPR	other	1	0	1
op	0	other	1	DC/ERR	ERR/DC	op	0	DC	DC	DC	0	0	1
stall EX	0	DC	DC	DC	DC	stall WB	0	DC	DC	DC	0	0	1
stall EX	1	DC	DC	DC	DC	op	1	DC	DC	DC	0	0	0
stall WB	0	DC	DC	DC	DC	Stall RF	0	DC	DC	DC	0	0	0
stall WB	1	DC	DC	DC	DC	op	1	DC	DC	DC	0	0	0

Definitions:

- DC - don't care
- ERR - any other value, which will be considered as an input error. Resulting state and outputs are in red.
- Other - ADD, SUB, NAND, NOR, XOR, SHFL

Assertion: dst[2] will never be ‘1’

Code information

Features

With the purpose of keeping the code clean, scalable and easy to debug, it features:

- Parameterized multiplexer
- Changeable data bus size
- Register file with flexible number of entries and R/W ports
- Use of assertions
- Use of generate blocks
- Use of packages
- Use of enumerated types
- Use of interfaces

How to run

Enter this link and press the Run button on the top-left corner. You will be asked to log in to EDA playground. Do so by clicking the orange “Log In (no save)” button. Choose your preferred login method - Google/Facebook account is good enough.
Once logged in, re-enter the link above and click Run again.
The log file at the bottom will display results of the testbench on the left.

Suggested enhancements

Ideas that came up during implementation, for future development:

- Using ADD hardware in ALU for SUB operation
- Dependency detector (avoid stalling without reason)
- Supersacalr (with dependency detector)
- PRF for WAW/WAR dependencies
- OOO machine