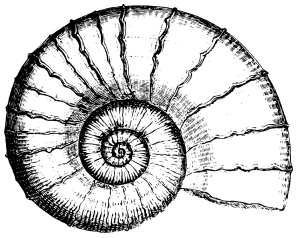
The Shell CPU

Datapath



Jonathan Rice Shelley II

September 30, 2020

Version 1.0

1 Contents

2 Shell Datapath Overview

The Shell datapath is shown in full in figure 4. The CPUs system memory is based on the Harvard architecture separating data and program memory (Figure 1 and 2). In figure 3 the CPU is shown without the surrounding memory. A high resolution picture of the full datapath can be found at this link.

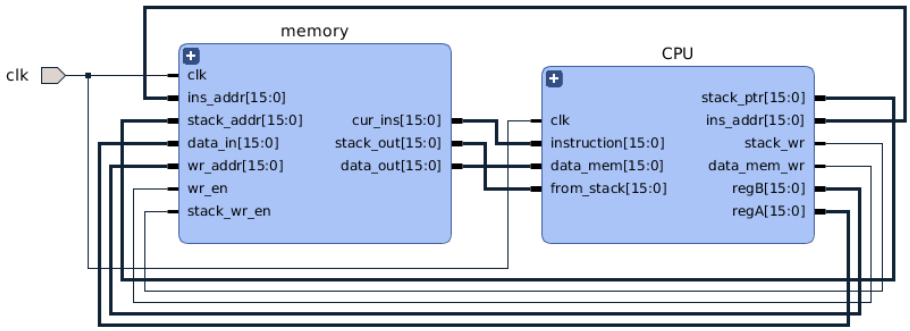


Figure 1: Top level separation of CPU and memory.

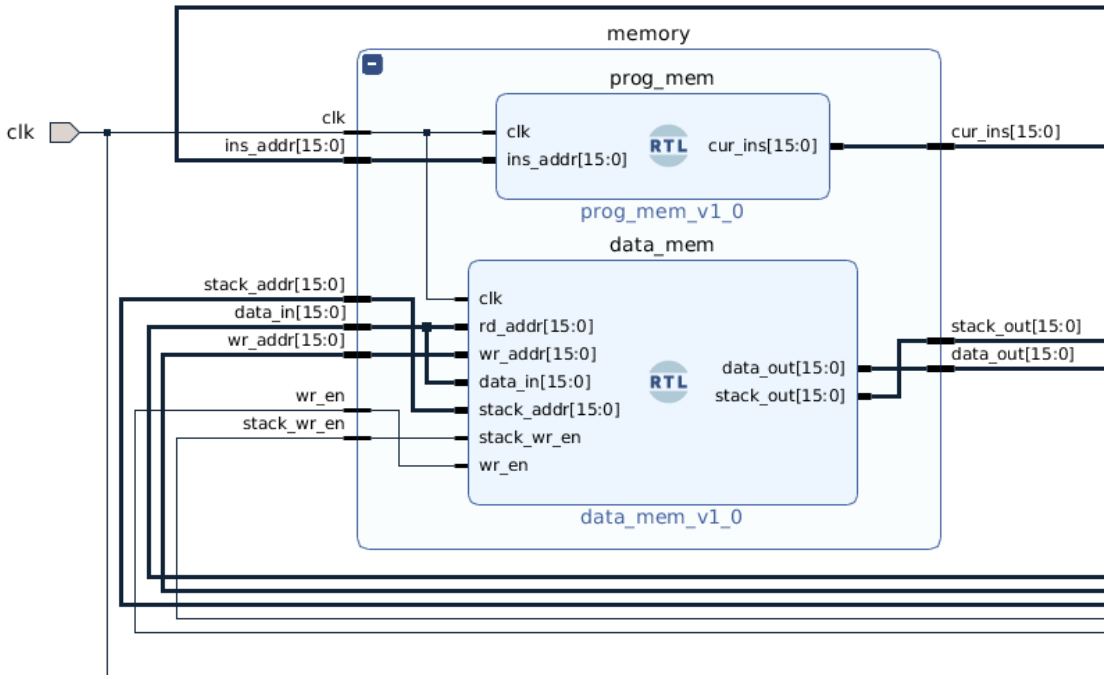


Figure 2: System memory.

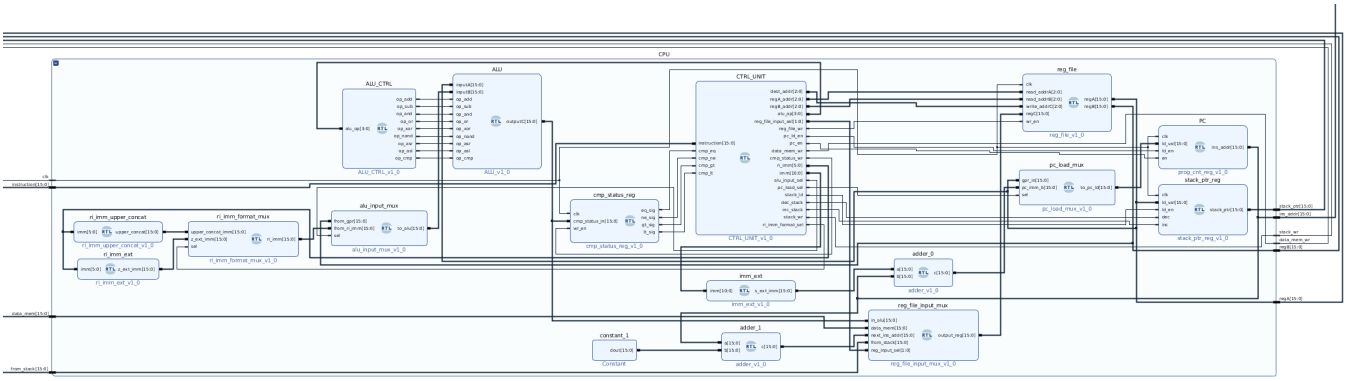


Figure 3: CPU structure.

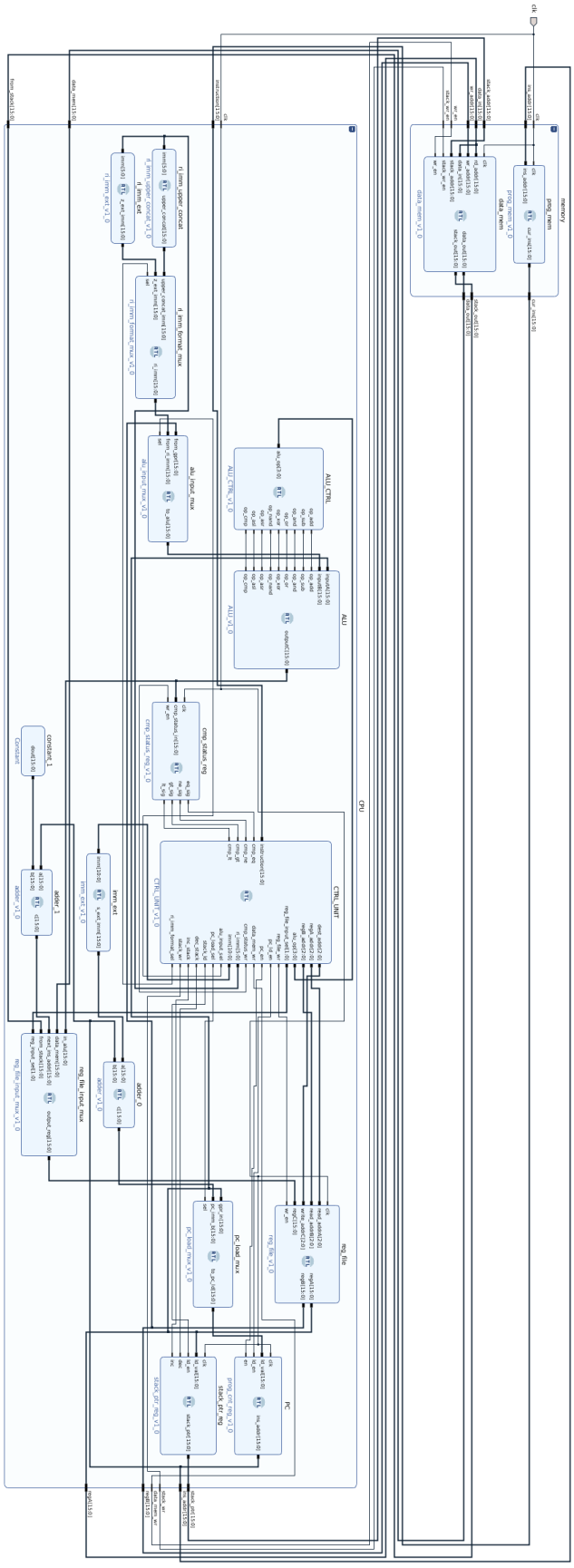


Figure 4: Full CPU structure (rotated for increased resolution).

3 Component Descriptions

Description of inputs, outputs, and function of each component in the datapath.

3.1 Arithmetic Logic Unit

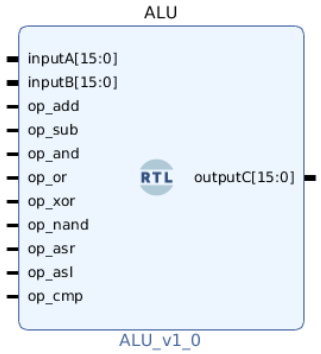


Figure 5: Arithmetic Logic Unit

Functional description

The arithmetic logic unit or ALU performs arithmetic operations on data in the CPU. The block is entirely combinational logic and controlled by its "op" signals. ALU op code signals are one hot. In the event that multiple signals are high priority will be taken to the op code signal that appears first in the IO table below.

IO table	
Inputs	**Description**
inputA	16 bit bus, ALU operand
inputB	16 bit bus, 2nd ALU operand
op_add	if signal high add operands
op_sub	if signal high subtract operands
op_and	if signal high bit-wise and operands
op_or	if signal high bit-wise or operands
op_xor	if signal high bit-wise xor operands
op_nand	if signal high bit-wise nand operands
op_asr	if signal high arithmetic shift 1 bit right input A
op_asl	if signal high arithmetic shift 1 bit left input A
op_cmp	if signal high compare operands.
Outputs	**Description**
outputC	the 16 bit result of some operation on input A and B

3.2 Simple Adder

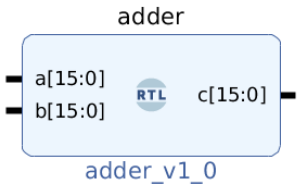


Figure 6: Simple adder unit

Functional description

The simple adder block is used to sum values together within the CPU datapath. The block is used once to calculate PC + 1. The block is used a second time to find pc + imm for B-Type instructions.

IO table	
Inputs	**Description**
a	16 bit bus
b	16 bit bus
Outputs	**Description**
c	16 bit bus, assigned to sum of a and b.

3.3 Arithmetic Logic Unit Controller

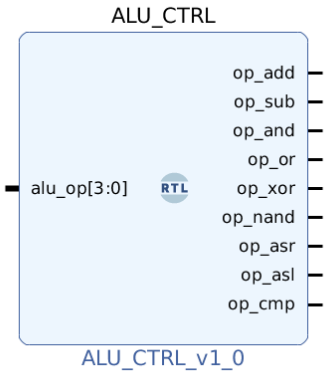


Figure 7: Arithmetic Logic Unit Controller

Functional description

The arithmetic logic unit controller is used to interpret the ALU op codes from the CPU controller. The outputs of the arithmetic logic unit controller directly control what operation the ALU performs.

IO table	
Inputs	**Description**
alu_op	ALU op code from CPU controller
Outputs	**Description**
op_add	add operands
op_sub	subtract operands
op_and	bit-wise and operands
op_or	bit-wise or operands
op_xor	bit-wise xor operands
op_nand	bit-wise nand operands
op_asr	arithmetic shift 1 bit right input A
op_asl	arithmetic shift 1 bit left input A
op_cmp	compare operands.

3.4 Arithmetic Logic Unit Input Multiplexer

Figure 8: Arithmetic Logic Unit Input Multiplexer

Functional description

The arithmetic logic unit input multiplexer is used to mux the input to the 16-bit B operand of the ALU. The input to the ALU can be switched between the output of the GPR file or a ri_imm.

IO table	
Inputs	**Description**
from_gpr	16-bit input from GPR file
from_ri_imm	16-bit input from ri_imm
sel	select line of multiplexer
Outputs	**Description**
to_alu	16-bit output of mux

3.5 Comparator Status Register

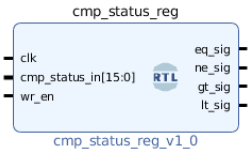


Figure 9: Comparator Status Register

Functional description

The comparator status register stores the results of a comparison.

IO table	
Inputs	**Description**
clk	CPU clk signal
cmp_status_in	16-bit result of comparison from ALU
wr_en	write enable line
Outputs	**Description**
eq_sig	High when comparison shows equality
ne_sig	High when comparison shows inequality
gt_sig	High when comparison is greater than
lt_sig	High when comparison is less than

3.6 CPU Control Unit

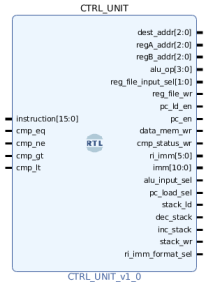


Figure 10: CPU Control Unit

Functional description

The CPU control unit decodes instructions and configures the CPU to execute the instructions.

IO table	
Inputs	**Description**
instruction	16-bit instruction
cmp_eq	status bit from last comparison
cmp_ne	status bit from last comparison
cmp_gt	status bit from last comparison
cmp_lt	status bit from last comparison
Outputs	**Description**
dest_addr	Address of destination register
regA_addr	Address of register A
regB_addr	Address of register B
alu_op	ALU op code
reg_file_input_sel	select line for reg file input mux
reg_file_wr	register file write line
pc_ld_en	load PC reg enable line
pc_en	enable PC increment signal
data_mem_wr	write to data memory signal
cmp_status_wr	write to comparator register signal
ri_imm[5:0]	ri-type 6-bit immediate
imm[10:0]	B-type 11 bit immediate
alu_input_sel	ALU input mux select line
pc_load_sel	program counter load input mux select line
stack_ld	stack load input signal
dec_stack	decrement stack input signal
inc_stack	increment stack input signal
stack_wr	enable writing to memory at stack pointer
ri_imm_format_sel	ri-type format mux select line

3.7 Data Memory

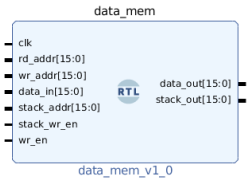


Figure 11: Data Memory

Functional description

Memory for stack and program data.

IO table	
Inputs	**Description**
clk	CPU clk signal
rd_addr	16-bit read address
wr_addr	16-bit write address
data_in	16-bit word to write
stack_addr	16-bit stack pointer
stack_wr_en	stack write enable
wr_en	memory write enable
Outputs	**Description**
data_out	16-bit word from memory at wr_addr
stack_out	16-bit word from memory at stack pointer

3.8 11-bit Immediate Sign Extend

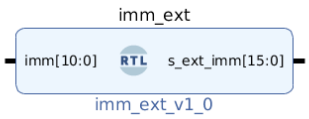


Figure 12: 11-bit Immediate Sign Extend

Functional description

Used to sign extend 11-bit immediates from B-type instructions.

IO table	
Inputs	**Description**
imm	immediate from b-type instruction
Outputs	**Description**
s_ext_imm	sign extended immediate from b-type instruction

3.9 Program Counter Load Multiplexer

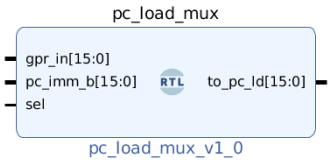


Figure 13: Program Counter Load Multiplexer

Functional description

Used to multiplex inputs to PC load bus.

IO table	
Inputs	**Description**
gpr_in	GPR file 16-bit output
pc_imm_b	value of pc + immediate
sel	multiplexer select signal
Outputs	**Description**
to_pc_ld	output value to PC load word port

3.10 Program Counter Register

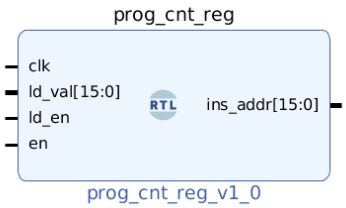


Figure 14: Program Counter Register

Functional description

Program counter used to keep the address of the next instruction in memory.

IO table	
Inputs	**Description**
clk	CPU clock
ld_val	16-bit word to load PC with
ld_en	PC load enable
en	enable PC increment.
Outputs	**Description**
ins_addr	address of the next instruction in memory

3.11 Program Memory

Figure 15: Program Memory

Functional description

Program memory used to store program instructions.

IO table	
Inputs	**Description**
clk	CPU clock
ins_addr	instruction address
Outputs	**Description**
cur_ins	current instruction

3.12 General Purpose Register File

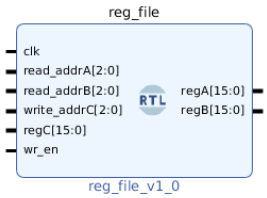


Figure 16: General Purpose Register File

Functional description

GPR (General Purpose Register) file is used to store temporary values during computation.

IO table	
Inputs	**Description**
clk	CPU clock
read_addrA	GPR address
read_addrB	GPR address
write_addrC	GPR address
regC	data word to be written to GPR
wr_en	write enable
Outputs	**Description**
regA	contents of GPR at addrA
regB	contents of GPR at addrB

3.13 Register File Input Multiplexer

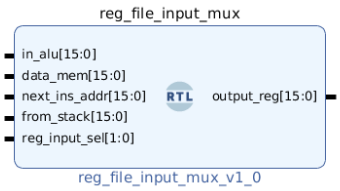


Figure 17: Register File Input Multiplexer

Functional description

Used to multiplex the data input to the GPR file.

IO table	
Inputs	**Description**
in_alu	16-bit output from ALU
data_mem	16-bit output from data memory
next_ins_addr	16-bit PC + 1
from_stack	16-bit word from stack
reg_input_sel	multiplexer select line
Outputs	**Description**
output_reg	16-bit output of multiplexer

3.14 RI-Type Zero Extend Immediate

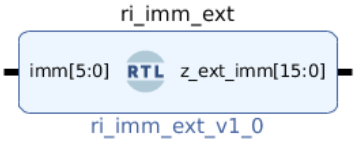


Figure 18: RI-Type Zero Extend Immediate

Functional description

Used to zero extend the ri-type immediate field.

IO table	
Inputs	**Description**
imm	6-bit immediate
Outputs	**Description**
z_ext_imm	16-bit zero extended 6-bit immediate

3.15 RI-Type Immediate Format Multiplexer

Figure 19: RI-Type Immediate Format Multiplexer

Functional description

Multiplex RI-Type immediate between zero extended and upper 6-bit concatenation format.

IO table	
Inputs	**Description**
upper_concat_imm	16-bit concat immediate
z_ext_imm	16-bit zero extended immediate
sel	multiplexer select line
Outputs	**Description**
ri_imm	16-bit formatted ri-type immediate

3.16 RI-Type Immediate Upper 6-bit Concatenation

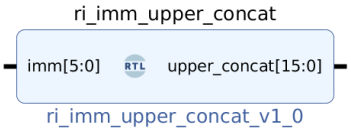


Figure 20: RI-Type Immediate Upper 6-bit Concatenation

Functional description

Concatenate 6-bit immediate with upper part of 16-bit register zero all lower bits.

IO table	
Inputs	**Description**
imm	6-bit immediate
Outputs	**Description**
upper_concat	16-bit formatted ri-type immediate

3.17 Stack Pointer Register

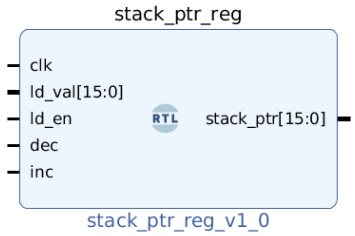


Figure 21: Stack Pointer Register

Functional description

Register holds stack pointer.

IO table	
Inputs	**Description**
clk	CPU clk
ld_val	word to be loaded into stack pointer
ld_en	enable write to stack pointer register
dec	decrement stack pointer
inc	increment stack pointer
Outputs	**Description**
stack_ptr	16-bit stack pointer

4 Datapath Register Transfers

The datapath register transfers section details the register transfers that take place for each instruction. In the below table R_n corresponds to registers in the GPR file.

No.	Instruction	Register Transfers
1	hlt	PC => PC
2	add R_1, R_2, R_3	R_2, R_3 => ALU => R_3
3	sub R_1, R_2, R_3	R_2, R_3 => ALU => R_3
4	and R_1, R_2, R_3	R_2, R_3 => ALU => R_3
5	or R_1, R_2, R_3	R_2, R_3 => ALU => R_3
6	xor R_1, R_2, R_3	R_2, R_3 => ALU => R_3
7	nand R_1, R_2, R_3	R_2, R_3 => ALU => R_3
8	lw R_1, R_2	R_2 => Data Memory => R_1
9	sw R_1, R_2	R_1, R_2 => Data Memory
10	asr R_1, R_2	R_2 => ALU => R_1
11	asl R_1, R_2	R_2 => ALU => R_1
12	cmp R_1, R_2	R_1, R_2 => ALU => compare register
13	jalr R_1, R_2	PC => + 1 adder => R_1 | R_2 => PC
14	push R_1	R_1 => data memory at addr SP
15	pop R_1	data memory at addr SP => R_1
16	lsp R_1	R_1 => Stack Pointer Register
17	addi R_1, *imm*	R_1, *imm* => zero extend => ALU => R_1
18	subi R_1, *imm*	R_1, *imm* => zero extend => ALU => R_1
19	lui R_1, *imm*	R_1, *imm* => zero extend => ALU => R_1
20	beq *imm*	*imm* => sign extend => + current PC adder => PC
21	bne *imm*	*imm* => sign extend => + current PC adder => PC
22	bgt *imm*	*imm* => sign extend => + current PC adder => PC
23	blt *imm*	*imm* => sign extend => + current PC adder => PC

5 CPU Control and Operation

This section list all control signals and their values needed to perform each instruction in the Shell ISA.

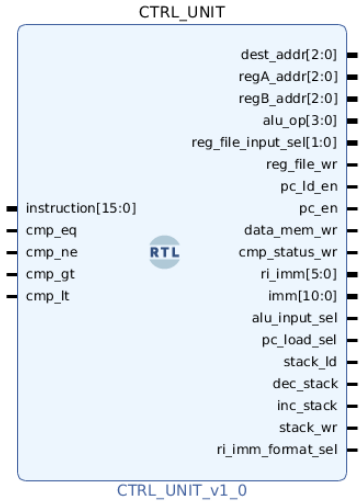


Figure 22: Shell CPU Control Unit

To perform each instruction the combinational logic control unit shown above must decode the 16-bit instruction and configure its outputs to set the CPU into the desired state. Each instruction will configure the outputs of this block in a unique format. The outputs for proper control unit configuration for **every** instruction is listed below.

The key below describes the symbols used in the tables that describe the control unit output for each instruction. Each symbol will always correspond to 1 bit.

Control Truth Table Symbol Key	
Symbol	**Description**
0	0 bit / low signal
1	1 bit / high signal
x	don't care bit
n	value defined by field in instruction. Ex: (imm or GPR addresses)
{ ctrl_unit_input_sig_name }	{} denotes the use of a ctrl unit input signal

5.1 Halt S-Type Instruction

The halt instruction stops the CPU from executing instructions.

Signal	Binary value
dest_addr[2:0]	xxx
regA_addr[2:0]	xxx
regB_addr[2:0]	xxx
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	xx
reg_file_wr	0
pc_ld_en	0
pc_en	0
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	x
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.2 Add RRR-Type Instruction

The add instruction uses the ALU to add 2 registers from the GPR file and stores the result back in the GPR file.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	nnn
alu_op[3:0]	0000
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	0
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.3 Sub RRR-Type Instruction

The sub instruction uses the ALU to subtract 2 registers from the GPR file and stores the result back in the GPR file.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	nnn
alu_op[3:0]	0001
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	0
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.4 And RRR-Type Instruction

The and instruction uses the ALU to bit-wise and 2 registers from the GPR file and stores the result back in the GPR file.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	nnn
alu_op[3:0]	0010
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	0
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.5 Or RRR-Type Instruction

The or instruction uses the ALU to bit-wise or 2 registers from the GPR file and stores the result back in the GPR file.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	nnn
alu_op[3:0]	0011
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	0
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.6 Xor RRR-Type Instruction

The xor instruction uses the ALU to bit-wise xor 2 registers from the GPR file and stores the result back in the GPR file.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	nnn
alu_op[3:0]	0100
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	0
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.7 Nand RRR-Type Instruction

The nand instruction uses the ALU to bit-wise nand 2 registers from the GPR file and stores the result back in the GPR file.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	nnn
alu_op[3:0]	0101
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	0
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.8 Lw RR-Type Instruction

The lw instruction loads a 16-bit word from memory and stores it in the GPR file.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	xxx
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	01
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	x
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.9 Sw RR-Type Instruction

The sw instruction stores a 16-bit word at an address in data memory. The address and value stored are from the GPR file.

Signal	Binary value
dest_addr[2:0]	xxx
regA_addr[2:0]	nnn
regB_addr[2:0]	nnn
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	xx
reg_file_wr	0
pc_ld_en	0
pc_en	1
data_mem_wr	1
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	x
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.10 Asr RR-Type Instruction

The asr instruction performs an arithmetic shift right using the ALU.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	xxx
alu_op[3:0]	0110
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	x
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.11 Asl RR-Type Instruction

The asl instruction performs an arithmetic shift left using the ALU.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	xxx
alu_op[3:0]	0111
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	x
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.12 Cmp RR-Type Instruction

Compare two registers using ALU store result in compare register.

Signal	Binary value
dest_addr[2:0]	xxx
regA_addr[2:0]	nnn
regB_addr[2:0]	nnn
alu_op[3:0]	1010
reg_file_input_sel[1:0]	xx
reg_file_wr	0
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	1
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	0
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.13 Jalr RR-Type Instruction

The jalr or (jump and link register) instruction jumps to an address and writes the old PC value + 1 into a specified register.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	xxx
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	10
reg_file_wr	1
pc_ld_en	1
pc_en	0
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	x
pc_load_sel	0
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.14 Push R-Type Instruction

The push instruction puts a register value onto the stack and decrement the stack pointer.

Signal	Binary value
dest_addr[2:0]	xxx
regA_addr[2:0]	nnn
regB_addr[2:0]	xxx
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	xx
reg_file_wr	0
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	x
pc_load_sel	x
stack_ld	0
dec_stack	1
inc_stack	0
stack_wr	1
ri_imm_format_sel	x

5.15 Pop R-Type Instruction

The pop instruction gets a value off the stack to store in GPR file and increments the stack pointer.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	xxx
regB_addr[2:0]	xxx
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	11
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	x
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	1
stack_wr	0
ri_imm_format_sel	x

5.16 Lsp R-Type Instruction

The lsp (load stack pointer) instruction is used to set the value of the stack pointer from a register in the GPR file.

Signal	Binary value
dest_addr[2:0]	xxx
regA_addr[2:0]	nnn
regB_addr[2:0]	xxx
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	xx
reg_file_wr	0
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	xxxxxxxxxxx
alu_input_sel	x
pc_load_sel	x
stack_ld	1
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.17 Addi RI-Type Instruction

The addi instruction uses the ALU to add a 6 bit unsigned immediate to a register. The result is stored into a specified destination register.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	xxx
alu_op[3:0]	0000
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	nnnnnn
imm[10:0]	xxxxxxxxxxx
alu_input_sel	1
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	1

5.18 Subi RI-Type Instruction

The subi instruction uses the ALU to subtract a 6 bit unsigned immediate to a register. The result is stored into a specified destination register.

Signal	Binary value
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	xxx
alu_op[3:0]	0001
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	nnnnnn
imm[10:0]	xxxxxxxxxxx
alu_input_sel	1
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	1

5.19 Lui RI-Type Instruction

The lui (load upper immediate) instruction sets the upper 6-bits of a specified register in the GPR file.

Signal	**Binary value**
dest_addr[2:0]	nnn
regA_addr[2:0]	nnn
regB_addr[2:0]	000
alu_op[3:0]	0000
reg_file_input_sel[1:0]	00
reg_file_wr	1
pc_ld_en	0
pc_en	1
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	nnnnnn
imm[10:0]	xxxxxxxxxxx
alu_input_sel	1
pc_load_sel	x
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	0

5.20 Beq B-Type Instruction

The beq (branch equal) instruction performs a jump with an 11 bit signed immediate when the eq_sig flag of the compare register is set high.

Signal	Binary value
dest_addr[2:0]	xxx
regA_addr[2:0]	xxx
regB_addr[2:0]	xxx
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	xx
reg_file_wr	0
pc_ld_en	{ cmp_eq }
pc_en	{ ¬ cmp_eq }
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	nnnnnnnnnnn
alu_input_sel	xx
pc_load_sel	1
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.21 Bne B-Type Instruction

The bne (branch not equal) instruction performs a jump with an 11 bit signed immediate when the ne_sig flag of the compare register is set high.

Signal	Binary value
dest_addr[2:0]	xxx
regA_addr[2:0]	xxx
regB_addr[2:0]	xxx
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	xx
reg_file_wr	0
pc_ld_en	{ cmp_ne }
pc_en	{ ¬ cmp_ne }
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	nnnnnnnnnnn
alu_input_sel	xx
pc_load_sel	1
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.22 Bgt B-Type Instruction

The bgt (branch greater than) instruction performs a jump with an 11 bit signed immediate when the gt_sig flag of the compare register is set high.

Signal	**Binary value**
dest_addr[2:0]	xxx
regA_addr[2:0]	xxx
regB_addr[2:0]	xxx
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	xx
reg_file_wr	0
pc_ld_en	{ cmp_gt }
pc_en	{ ¬ cmp_gt }
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	nnnnnnnnnnn
alu_input_sel	xx
pc_load_sel	1
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

5.23 Blt B-Type Instruction

The blt (branch less than) instruction performs a jump with an 11 bit signed immediate when the lt_sig flag of the compare register is set high.

Signal	Binary value
dest_addr[2:0]	xxx
regA_addr[2:0]	xxx
regB_addr[2:0]	xxx
alu_op[3:0]	xxxx
reg_file_input_sel[1:0]	xx
reg_file_wr	0
pc_ld_en	{ cmp_lt }
pc_en	{ ¬ cmp_lt }
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	xxxxxx
imm[10:0]	nnnnnnnnnnn
alu_input_sel	xx
pc_load_sel	1
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	x

6 Design Tradeoffs

The Shell datapath design favors simplistic implementation of the instruction set. A single clock cycle design was used. On positive clock edges values are stored with edge triggered flip-flops. During the remaining clock cycle the combinational logic decodes the instruction and performs necessary operations. Dedicated components were used to cut down on unnecessary complexity in the datapath. Due to the Shell CPUs single clock cycle design the clock must have a period greater than the slowest instruction. A stack was included in the datapath so that recursive subroutine calls can be easily implemented in software.