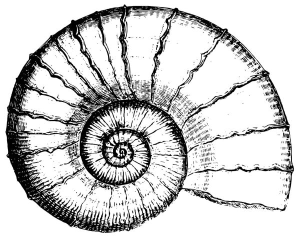
The Shell CPU

Memory Modules and Final Testing



Jonathan Rice Shelley II

November 18, 2020

Version 1.0

Contents

1 Overview

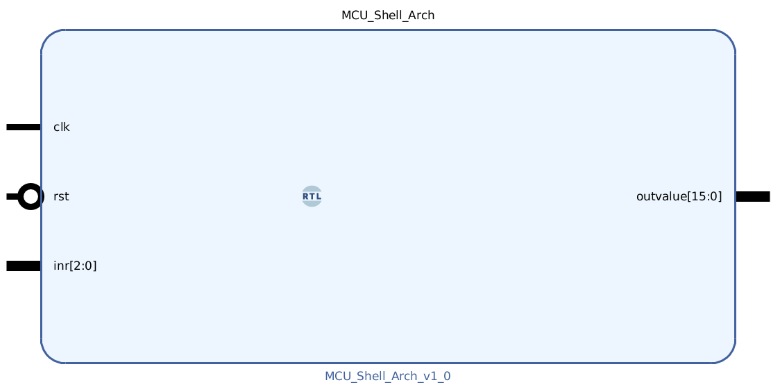
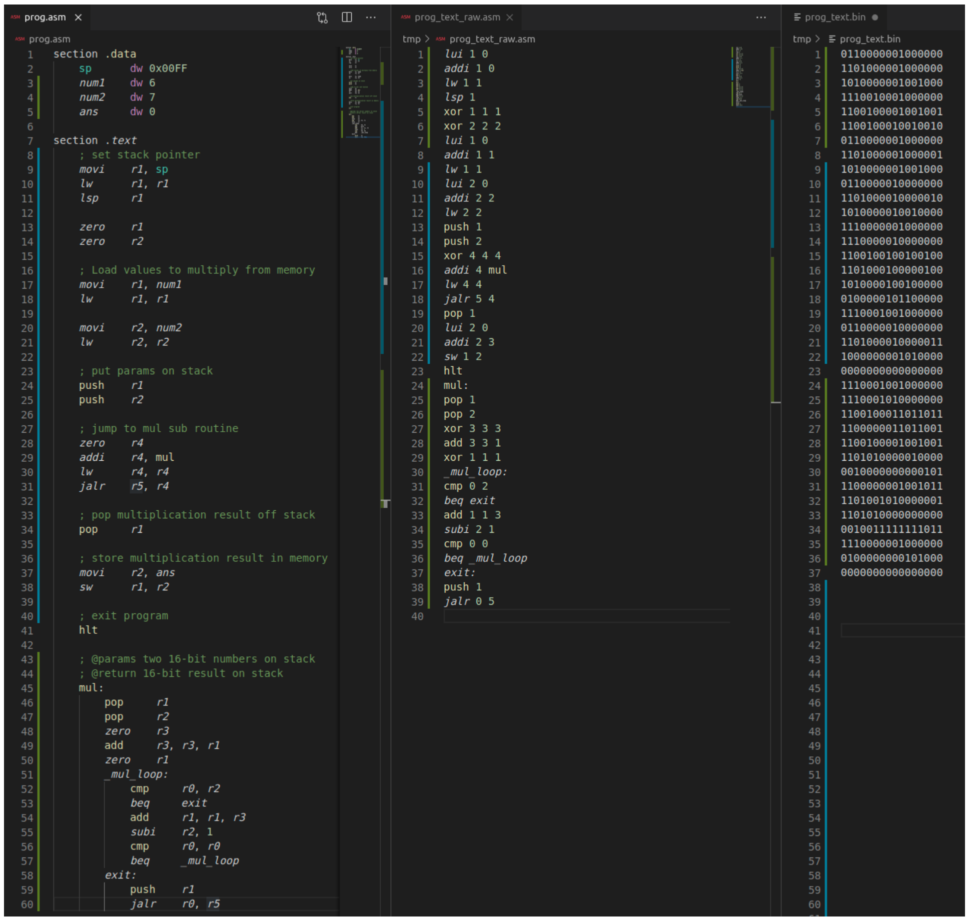


Figure 1: MCU Shell Architecture.

This document details the final implementation of the Shell CPU into a MCU top level module (Figure 1). The Design has three inputs and one 16-bit output port. Embedded program memory and data memory have been added to the design using block RAMs. The three bit inr line is used to select which of the eight internal CPU registers is displayed on the 16-bit bus outvalue. The VHDL for each module is to long to be listed and can be found with other documentation here.

2 Test Program



prog.asm

```
section .data
    sp      dw 0x00FF
    num1    dw 6
    num2    dw 7
    ans     dw 0

section .text
    ; set stack pointer
    movi    r1, sp
    lw      r1, r1
    lsp     r1

    zero    r1
    zero    r2

    ; Load values to multiply from memory
    movi    r1, num1
    lw      r1, r1

    movi    r2, num2
    lw      r2, r2

    ; put params on stack
    push    r1
    push    r2

    ; jump to mul sub routine
    zero    r4
    addi    r4, mul
    lw      r4, r4
    jalr    r5, r4

    ; pop multiplication result off stack
    pop     r1

    ; store multiplication result in memory
    movi    r2, ans
    sw      r1, r2

    ; exit program
    hlt

    ; @params two 16-bit numbers on stack
    ; @return 16-bit result on stack
    mul:
        pop     r1
        pop     r2
        zero    r3
        add     r3, r3, r1
        zero    r1
        _mul_loop:
            cmp     r0, r2
            beq     exit
            add     r1, r1, r3
            subi    r2, 1
            cmp     r0, r0
            beq     _mul_loop
        exit:
            push    r1
            jalr    r0, r5
```

tmp > prog_text_raw.asm

```
lui 1 0
addi 1 0
lw 1 1
lsp 1
xor 1 1 1
xor 2 2 2
lui 1 0
addi 1 1
lw 1 1
lui 2 0
addi 2 2
lw 2 2
push 1
push 2
xor 4 4 4
addi 4 mul
lw 4 4
jalr 5 4
pop 1
lui 2 0
addi 2 3
sw 1 2
hlt
mul:
pop 1
pop 2
xor 3 3 3
add 3 3 1
xor 1 1 1
_mul_loop:
cmp 0 2
beq exit
add 1 1 3
subi 2 1
cmp 0 0
beq _mul_loop
exit:
push 1
jalr 0 5
```

tmp > prog_text.bin

```
0110000001000000
1101000001000000
1010000001001000
1110010001000000
1100100001001001
1100100010010010
0110000001000000
1101000001000001
1010000001001000
0110000010000000
1101000010000010
1010000010010000
1110000001000000
1110000010000000
1100100100100100
1101000100000100
1010000100100000
0100000101100000
1110001001000000
0110000010000000
1101000010000011
1000000001010000
0000000000000000
1110001001000000
1110001010000000
1100100011011011
1100000011011001
1100100001001001
1101010000010000
0010000000000101
1100000001001011
1101001010000001
1101010000000000
0010011111111011
1110000001000000
0100000000101000
0000000000000000
```

Figure 2: Test Program

A test program is written to demonstrate the functionality of system memory (Figure 2). The test program computes seven times six. The values of seven and six are loaded from memory multiplied and the result is written back to memory. The stack is used to pass data to and from the multiplication subroutine. The stack pointer is set to 255 at the beginning of the program. This test program adequately test all CPU memory functionality.

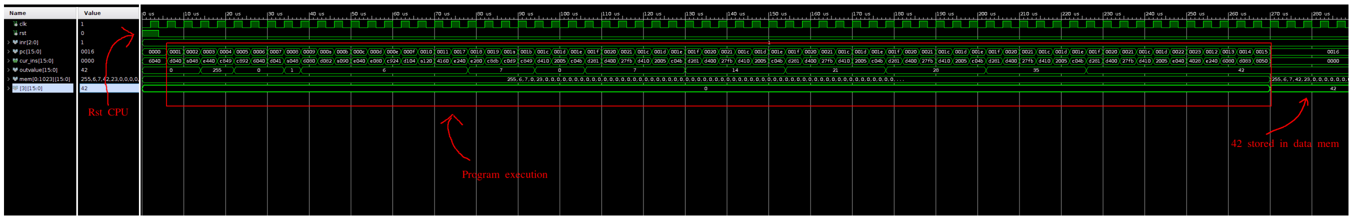
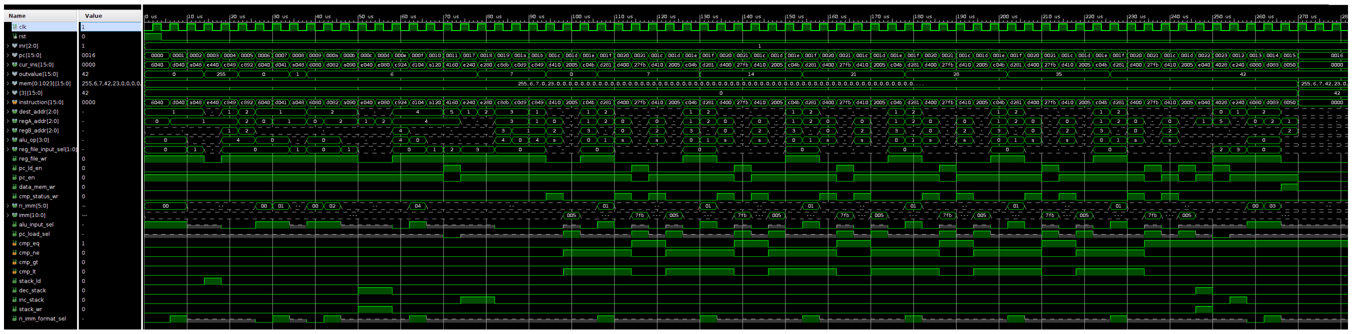


Figure 3: Test Program Simulation

The test program shown in Figure 2 is simulated in Figure 3. As shown by the Figure 3 annotations the ultimate answer 42 is computed and stored in memory at the end of the program. This final result 42 verifys that all the stack operations, loads, stores, and conditional / non-conditional jumps worked as intended. Figure 4 shows the same content as Figure 3 but also displays all CPU control signals.



Name	Value
clk	1
rst	0
inr[2:0]	1
pc[15:0]	0016
cur_ins[15:0]	0000
outvalue[15:0]	42
mem[0:1023][15:0]	255,6,7,42,23,0,0,0,0,
[3][15:0]	42
instruction[15:0]	0000
dest_addr[2:0]	-
regA_addr[2:0]	-
regB_addr[2:0]	-
alu_op[3:0]	-
reg_file_input_sel[1:0]	-
reg_file_wr	0
pc_ld_en	0
pc_en	0
data_mem_wr	0
cmp_status_wr	0
ri_imm[5:0]	--
imm[10:0]	---
alu_input_sel	-
pc_load_sel	-
cmp_eq	1
cmp_ne	0
cmp_gt	0
cmp_lt	0
stack_ld	0
dec_stack	0
inc_stack	0
stack_wr	0
ri_imm_format_sel	-

Figure 4: Test Program Simulation With Control Signals