COL216 Assignment 2

Stage 4 report

Aniruddha Deb
2020CS10869

Summary

This section expanded on the CPU: the predication logic was extended to include all 15 conditions, and the memory file structure was exported into a generic to make it easier to load tester code into memory for checking. Finally, in addition to the tests created, I wrote a utility program called `gentest` to generate VHDL testbenches directly from ARM files

File Structure

```
2020CS10869.zip
├── Makefile
├── alu.vhdl
├── commons.vhdl
├── commons_tb.vhdl
├── cpu.vhdl
├── cpu_multicycle.vhdl
├── cpu_synth.v
├── cpu_tb.vhdl
├── fact_tb.vhdl
├── mem.vhdl
├── mult_tb.vhdl
├── mytypes.vhdl
├── predicator.vhdl
├── regfile.vhdl
├── report.pdf
├── run.do
├── test_dp_tb.vhdl
├── test_progs
│   ├── fact.s
│   ├── gentest
│   ├── gentest.c
│   ├── mult.s
│   └── test_dp.s
├── wave_imgs
└── waves
    ├── test_dp.ghw
    ├── test_fact.ghw
    └── test_mult.ghw
```

Program Details

- `test_dp.s` - Testing all 16 Data Processing Instructions
- `mult.s` - A multiplier made using repetitive addition
- `fact.s` - A factorial program made using repetitive addition (again)
- `gentest.c` - A utility that reads in an assembly file and generates a VHDL testbench entity. The program is responsible for compiling, reading in the instructions from the compiled file and printing them to a VHDL file. Requires the ARM gcc compiler to work.
- Testbenches associated with the assembly files - `mult_tb.vhdl`, `fact_tb.vhdl`, `test_dp_tb.vhdl`

Testing

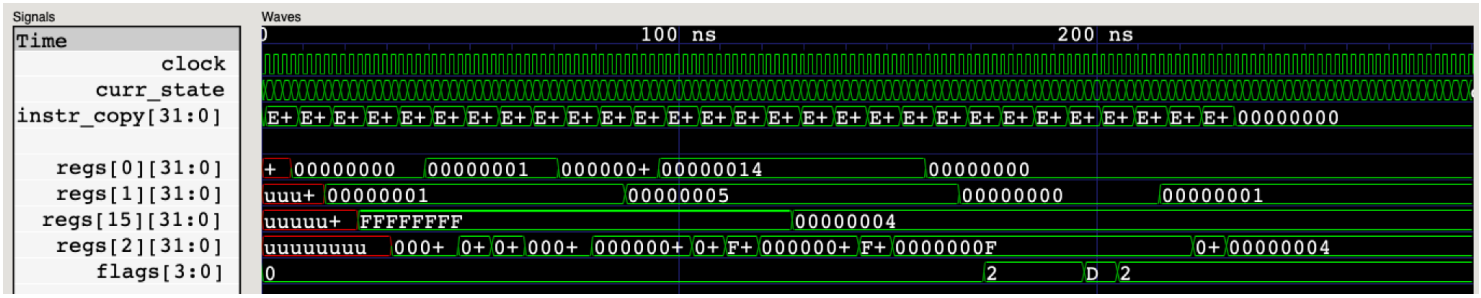
Associated testbenches are marked with `<filename>_tb.vhdl`. The CPU testbench only provides a clock to the CPU component.

The CPU is tested on the following three programs: each of whose code and instruction sets is shown side by side here, along with a graph of the relevant CPU signals. The graph highlights the internal working of the CPU and serves as the verification for the correctness of the CPU logic.

Test 1 - `test_dp_tb.vhdl`

```
    0 => X"E3A00000",
    1 => X"E3A01001",
    2 => X"E1E0F000",
    3 => X"E0012000",
    4 => X"E3A00001",
    5 => X"E0012000",
    6 => X"E20F20FD",
    7 => X"E0212000",
    8 => X"E3A00000",
    9 => X"E0212000",
    10 => X"E3A01005",
    11 => X"E3A00014",
    12 => X"E0402001",
    13 => X"E0602001",
    14 => X"E0802001",
    15 => X"E08FF001",
    16 => X"E0A12000",
    17 => X"E0C12000",
    18 => X"E0612000",
    19 => X"E3A00000",
    20 => X"E3A01000",
    21 => X"E1100001",
    22 => X"E1300001",
    23 => X"E3500000",
    24 => X"E3500001",
    25 => X"E1500001",
    26 => X"E3A01001",
    27 => X"E1812000",
    28 => X"E1CF2001",
    others => X"00000000"
```

```
    .text
    mov r0, #0
    mov r1, #1
    mvn r15, r0
    and r2, r1, r0
    mov r0, #1
    and r2, r1, r0
    and r2, r15, #0xFD
    eor r2, r1, r0
    mov r0, #0
    eor r2, r1, r0
    mov r1, #5
    mov r0, #20
    sub r2, r0, r1 @ 15
    rsb r2, r0, r1 @ -15
    add r2, r0, r1 @ 15
    add r15, r1 @ should overflow
    adc r2, r1, r0
    sbc r2, r1, r0
    rsb r2, r1, r0
    mov r0, #0
    mov r1, #0
    tst r0, r1
    teq r0, r1
    cmp r0, #0
    cmp r0, #1
    cmp r0, r1
    mov r1, #1
    orr r2, r1, r0
    bic r2, r15, r1
    .end
```

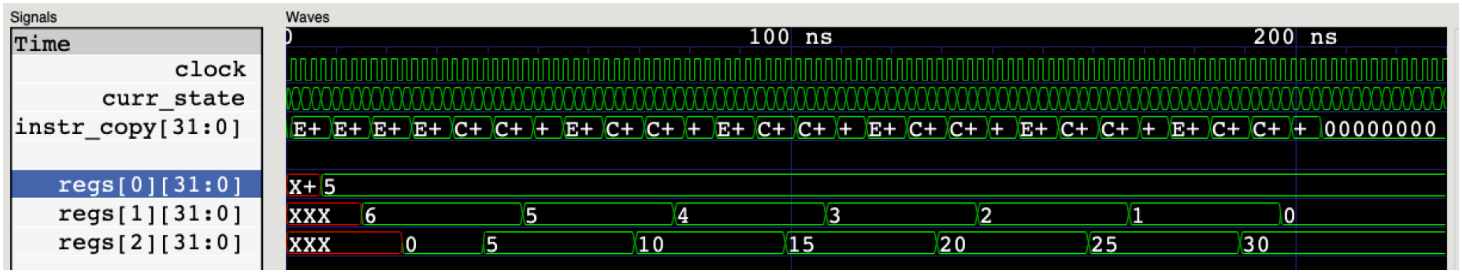


Test 2 - `mult_tb.vhdl`

```
    0 => X"E3A00005",
    1 => X"E3A01006",
    2 => X"E3A02000",

    3 => X"E3510001",
    4 => X"C0822000",
    5 => X"C2411001",
    6 => X"CAFFFFFB",
    others => X"00000000"
```

```
    .text
    mov r0, #5
    mov r1, #6
    mov r2, #0
mult:
l:  cmp r1, #1
    addgt r2, r0
    subgt r1, #1
    bgt l
    .end
```



(waveform output for registers is formatted in decimal instead of hexadecimal for easier reading)

Test 3 - `fact_tb.vhdl`

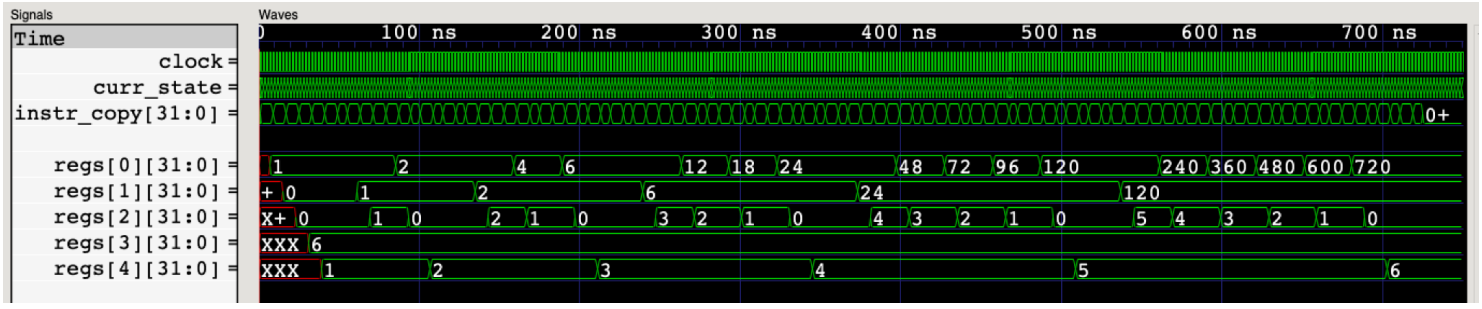
```
    0 => X"E3A00001",
    1 => X"E3A01000",
    2 => X"E3A02000",
    3 => X"E3A03006",
    4 => X"E3A04001",

    5 => X"E1530004",
    6 => X"0A000007",
    7 => X"E1A01000",
    8 => X"E1A02004",

    9 => X"E3520001",
   10 => X"C0800001",
   11 => X"C2422001",
   12 => X"CAFFFFFB",
   13 => X"E2844001",
   14 => X"EAFFFFF5",

    others => X"00000000"
```

```
    .text
    mov r0, #1 @ factorial stored here
    mov r1, #0 @ multiplicand
    mov r2, #0 @ multiplication counter
    mov r3, #6 @ factorial num to be found
    mov r4, #1 @ factorial counter
fact:
    cmp r3, r4
    beq end
    mov r1, r0
    mov r2, r4
    mult:
        cmp r2, #1
        addgt r0, r1
        subgt r2, #1
        bgt mult
    add r4, #1
    b fact
end:
    .end
```



(waveform output for registers is formatted in decimal instead of hexadecimal for easier reading)