COL216 Assignment 2

Stage 3 report

Aniruddha Deb
2020CS10869

Summary

This project started with a full rewrite of most major components, including the ALU, Memory file, Predication logic, as well as glue components such as Registers, Multiplexers and Sign Extenders. Following this, the CPU implementation was changed into a CPU dataflow and a CPU controller, both of which were linked with each other in CPU. Testing was done in GHDL/GtkWave (easy to use on Mac, Synthesis to Verilog + Simulation, better waveform viewer), DigitalJS (for viewing the CPU and debugging), and on EDAPlayground (class-mandated, used for synthesis). The code uses the IEEE libraries `numeric_std` and `std_logic_1164`, as well as the `mytypes` library (again heavily rewritten). The decoder has been integrated into the CPU, and is self written.

File Structure

```
2020CS10869.zip
├── Makefile
├── alu.vhdl
├── commons.vhdl
├── commons_tb.vhdl
├── cpu.vhdl
├── cpu_multicycle.vhdl
├── cpu_synth.v
├── cpu_tb.vhdl
├── design_plan.jpeg
├── mem.vhdl
├── mytypes.vhdl
├── predicator.vhdl
├── regfile.vhdl
├── report.pdf
├── run.do
├── logs
│   └── cpu_synth_log.txt
├── synth_imgs
│   └── cpu_synth_digitaljs.png
├── test_progs
│   ├── test_prog_1.s
│   ├── test_prog_2.s
│   ├── test_prog_3.s
│   └── test_prog_4.s
```

```
├── wave_imgs
│   ├── cpu_prog_1.png
│   ├── cpu_prog_2.png
│   ├── cpu_prog_3.png
│   └── cpu_prog_4.png
├── waves
│   ├── cpu_prog_1.ghw
│   ├── cpu_prog_2.ghw
│   ├── cpu_prog_3.ghw
│   ├── cpu_prog_4.ghw
│   └── mux_tb.ghw
└── work-obj93.cf
```

Program Details

- `commons.vhdl` - common components (registers, multiplexers)
- `commons_tb.vhdl` - testbench for commons
- `cpu_multicycle.vhdl` - contains the datapath and controller entities
- `cpu.vhdl` - the integration of the datapath and controller
- `cpu_synth.v` - A synthesis of the CPU, generated by GHDL in Verilog, for the purposes of visualization in DigitalJS. Generated using `ghdl --synth --out=verilog cpu > cpu_synth.v`
- `mem.vhdl` – A rewritten unified memory module with byte addressing, word-indexed little endian memory.
- `predicator.vhdl` - Predication logic. Only supports eq, ne and al at the moment, but can easily be extended.
- `design_plan.jpeg` - The planned schematic that was finally implemented. Includes minor deviations from the multicycle architecture specified in class, because of using the same adder for PC

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 same four programs as before: 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.

Note that the programs are slightly altered, since the program and instruction memories are at the same place and a collision might occur if we write to an instruction.

Test 1

Taken From Assignment Description

```
    0 => X"E3A00028",
    1 => X"E3A01005",
    2 => X"E5801000",
    3 => X"E2811002",
    4 => X"E5801004",
    5 => X"E5902000",
    6 => X"E5903004",
    7 => X"E0434002",
    others => X"00000000"
```

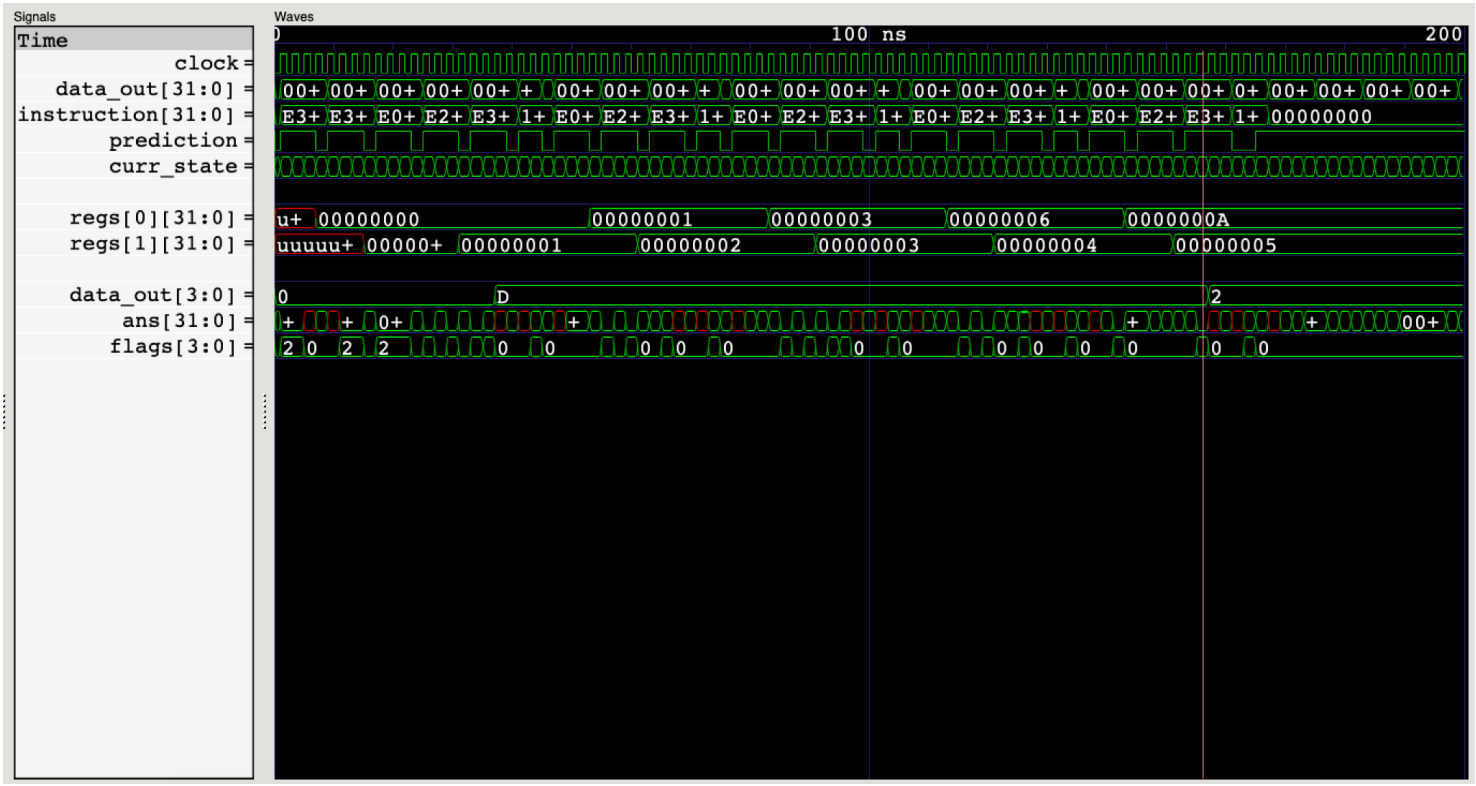
```
    .text
    mov r0, #40
    mov r1, #5
    str r1, [r0]
    add r1, r1, #2
    str r1, [r0, #4]
    ldr r2, [r0]
    ldr r3, [r0, #4]
    sub r4, r3, r2
    .end
```

Test 2

Taken From Assignment Description

```
    0 => X"E3A00000",
    1 => X"E3A01000",
    2 => X"E0800001",
    3 => X"E2811001",
    4 => X"E3510005",
    5 => X"1AFFFFFB",
    others => X"00000000"
```

```
    .text
    mov r0, #0
    mov r1, #0
Loop: add r0, r0, r1
    add r1, r1, #1
    cmp r1, #5
    bne Loop
    .end
```

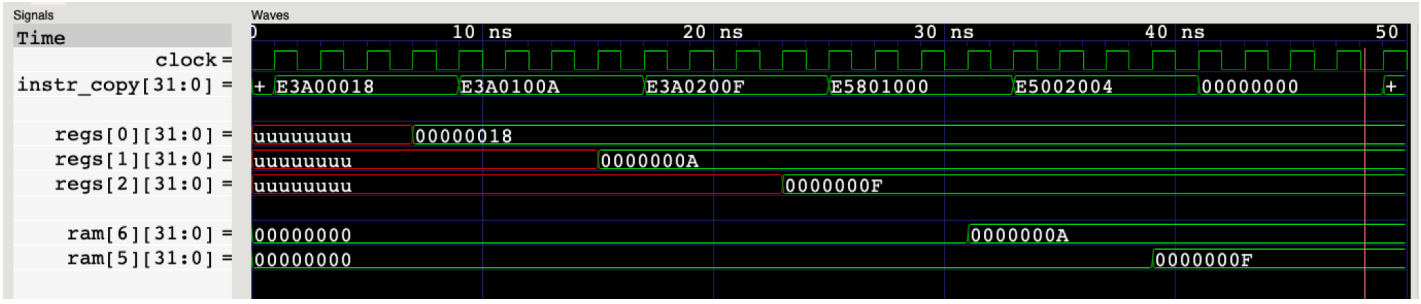


Test 3

Created by Self: Testing negative indexes for load/store operations

```
    0 => x"E3A00005",
    1 => x"E3A0100A",
    2 => x"E3A0200F",
    3 => x"E5801000",
    4 => x"E5002004",
    others => x"00000000"
```

```
    .text
    mov r0, #5
    mov r1, #10
    mov r2, #15
    str r1, [r0]
    str r2, [r0, #-4]
    .end
```



Test 4

Created by Self: Testing all three different branches

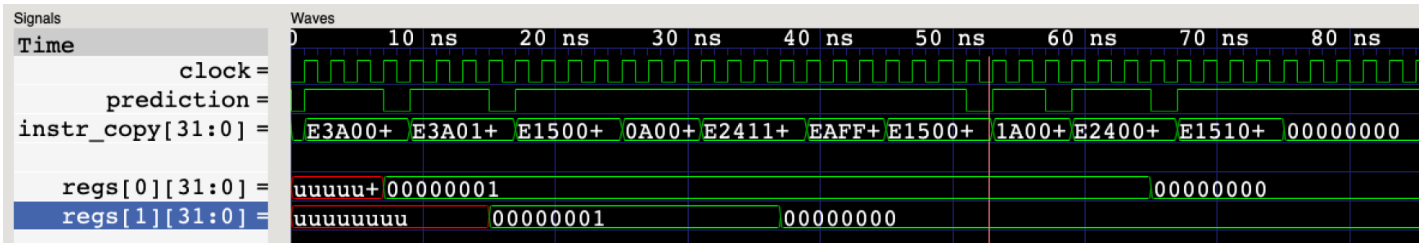
```
    0 => x"E3A00001",
    1 => x"E3A01001",
    2 => x"E1500001",
    3 => x"0A000001",

    4 => x"E1500001",
    5 => x"1A000001",

    6 => x"E2411001",
    7 => x"EAFFFFFB",

    8 => x"E2400001",
    9 => x"E1510000",
    others => x"00000000"
```

```
    .text
    mov r0, #1
    mov r1, #1
    cmp r0, r1
    beq equal
back:
    cmp r0, r1
    bne end
equal:
    sub r1, #1
    b back
end:
    sub r0, #1
    cmp r1, r0
    .end
```



Synthesis

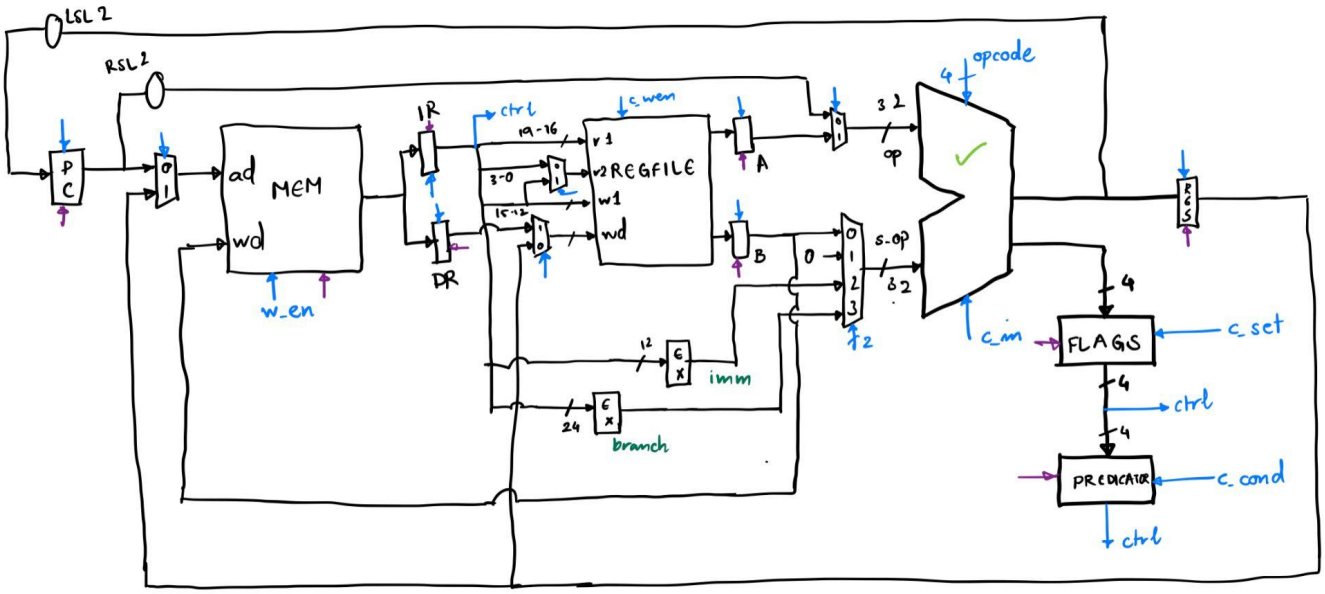
The CPU entity successfully synthesized using Mentor Precision on EDAPlayground. The logs are in cpu_synth_log.txt in logs. **Note that the CPU had only one IO pin, and that was synthesized. Mentor did not synthesize any of the other entities declared and port mapped inside the CPU (ALU, register file, program and data memory etc).**

A excerpt of the synthesis log is shown here

```
# Info: ************************************************************
# Info: Device Utilization for 7A100TCSG324
# Info: ************************************************************
# Info: Resource                          Used    Avail   Utilization
# Info: ------------------------------------------------------------
# Info: IOs                               1       210       0.48%
# Info: Global Buffers                    0       32        0.00%
# Info: LUTs                              0       63400     0.00%
# Info: CLB Slices                        0       15850     0.00%
# Info: Dffs or Latches                   0       126800    0.00%
# Info: Block RAMs                        0       135       0.00%
# Info: DSP48E1s                          0       240       0.00%
# Info: ------------------------------------------------------------
# Info: ******************************************************
# Info: Library: work    Cell: cpu    View: cpu_arch
# Info: ******************************************************
# Info:  Number of ports :                            1
# Info:  Number of nets :                             0
# Info:  Number of instances :                        0
# Info:  Number of references to this view :          0
# Info: Total accumulated area :
# Info:  Number of gates :                            0
# Info:  Number of accumulated instances :            0
# Info: ****************************
# Info:  IO Register Mapping Report
# Info: ****************************
# Info: Design: work.cpu.cpu_arch
# Info: +----------+-----------+----------+----------+----------+
# Info: | Port     | Direction |   INFF   |  OUTFF   |  TRIFF   |
# Info: +----------+-----------+----------+----------+----------+
# Info: | clock    | Input     |          |          |          |
# Info: +----------+-----------+----------+----------+----------+
# Info: Total registers mapped: 0
```

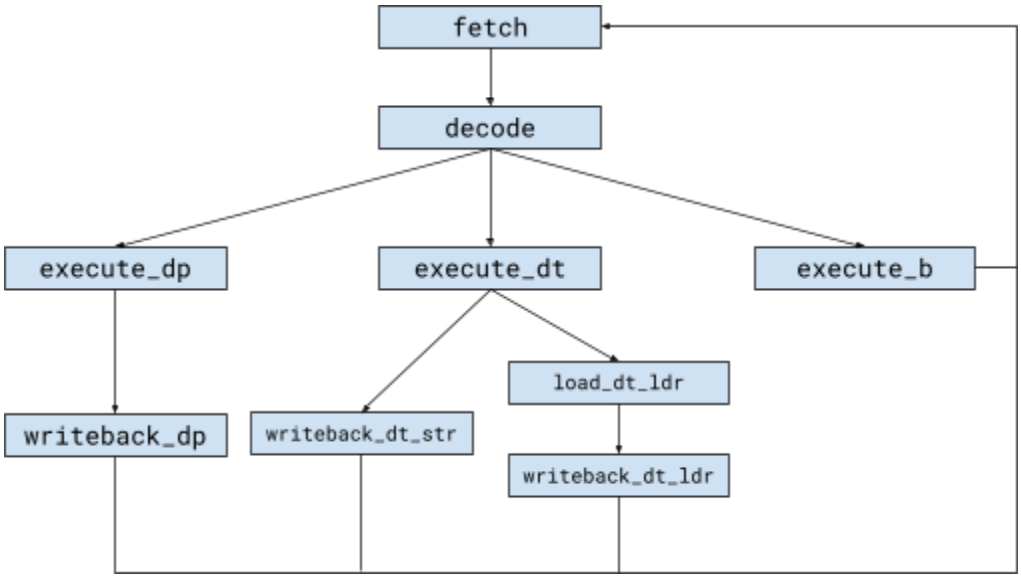
Design and Verification

The design to be implemented was as follows:

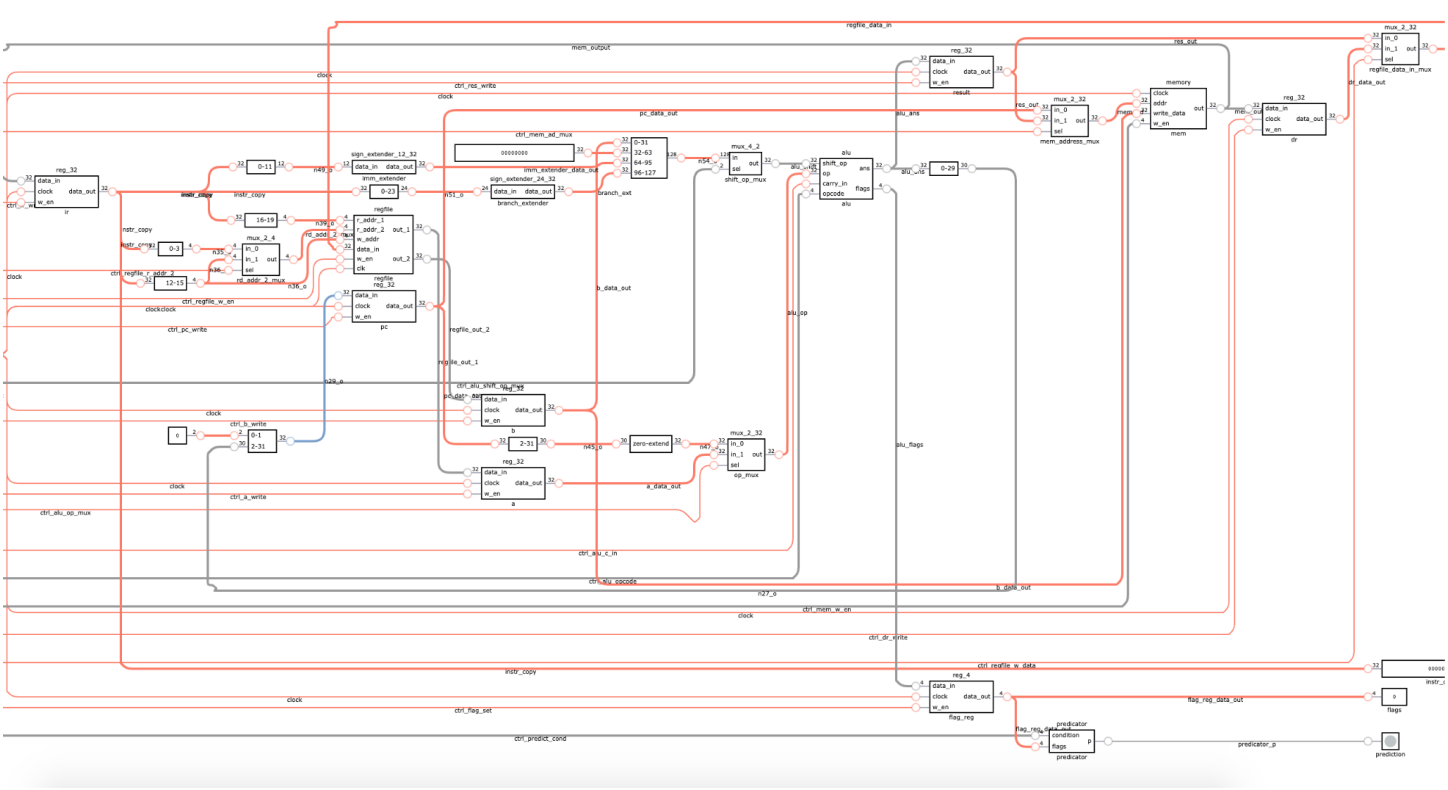


Notice the minor modifications made to allow for incrementing PC+8+(offset<<2) using just the ALU and two left/right shifters.

The control FSM implemented was the same as shown in the lecture: the state-name mapping is shown here



The final implementation was synthesized using the GHDL synthesizer and the verilog code generated was drawn out using DigitalJS. The final result is as follows:



(a high-res image can be found at `synth_imgs/cpu_synth_digitaljs.png`)