The other possible idea is to instead of construct the graph structure around the module, we can construct the graph structure around the instructions. Opposite to the other idea, I think this might make things easier when making inter module optimizations but make writing the internals of the modules a bit more difficult. I think this approach can prove to be more flexible.

The graph will describe directed edges for each wire.

If we take the ADD instruction as an example, what we will be after is representing this graph.

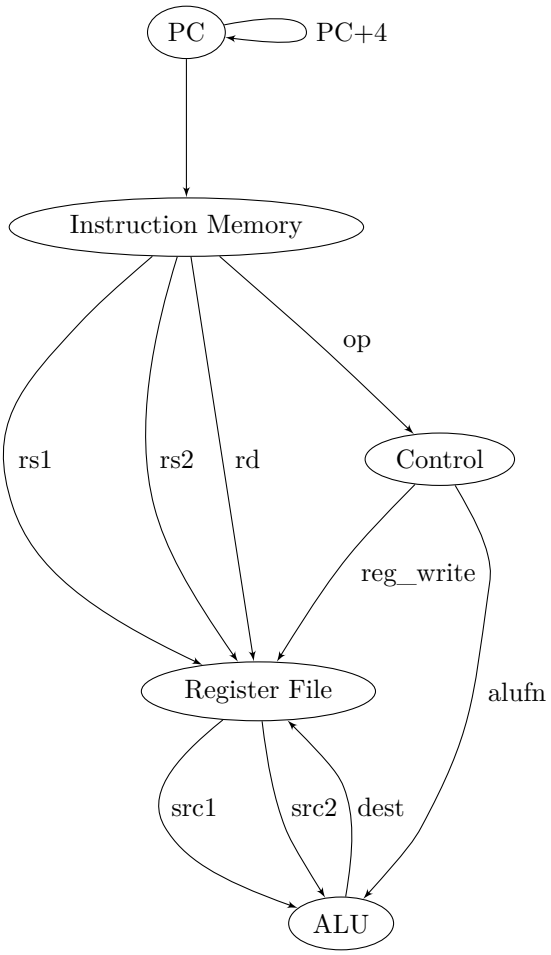

Figure 1:

Here is a possible way of presenting it in JSON

```
{
  "instruction": "ADD",
  "modules": [
    {
      "module": "PC",
      "input": [
        {
          "name": "in", "length": 32
        }
      ],
      "output": [
        {
          "name": "pc", "length": 32, "dest module": "instruction mem", "dest port": "in"
        },
        {
          "name": "new_pc", "length": 32, "dest module": "PC", "dest port": "in"
        }
      ]
```

```
    },
    {
      "module": "instruction memory",
      "input": [
        {
          "name": "address", "length": 32
        }
      "output": [
        {
          "name": "rs1", "length": 5, "dest module": "register file", "dest port": "src1"
        },
        {
          "name": "rs2", "length": 5, "dest module": "register file", "dest port": "src2"
        },
        {
          "name": "rd", "length": 5, "dest module": "register file", "dest port": "dest"
        }
      ]
      ]
    },
    {
      "module": "register file",
      "input": [
        {
          "name": "src1", "length": 5
        },
        {
          "name": "src2", "length": 5
        },
        {
          "name": "dest", "length": 5
        },
        {
          "name": "reg_write", "length": 1
        },
        {
          "name": "write_data", "length": 32
        }
      ],
      "output": [
        {
          "name": "read1", "length": 32, "dest module": "alu", "dest port": "a"
        },
        {
          "name": "read2", "length": 32, "dest module": "alu", "dest port": "b"
        }
      ]
    },
    {
      "module": "alu",
      "input": [
        {
          "name": "a", "length": 32
        },
        {
          "name": "b", "length": 32
```

```
          },
          {
            "name": "alufn", "length": 4
          }
        ],
        "output": [
          {
            "name": "out", "length": 32, "dest module": "reg file", "dest port":
    "write_data"
          }
        ]
      },
      {
        "module": "control",
        "input": [
          {
            "name": "op", "length": 5
          }
        ],
        "output": [
          {
            "name": "reg_write", "length": 1, "dest module": "register file", "
    dest port": "reg_write"
          },
          {
            "name": "alufn", "length": 4, "dest module": "alu", "dest port": "
    alufn"
          }
        ]
      }
    ]
  }

```

If you noticed, this specific implementation also involves a alight design shift in the datapath. Here, we do longer do operations on inter module wires. All module to module connections happen directly from one output to another input. For instance, this means that how the rs1, rs2 and rd are connected are to be changed into an independent output coming out of the instruction memory module rather than having the rs1, rs2 and rd lines selected at the inputs of the register memory. This is to keep the graph more simplified and manageable. I don't see an elegant way of dealing with otherwise. I'd really want other ideas in this.

As for a hard part, handling multiplexers. What I'm thinking is that we can leave multiplexers as the very last stage of creating the final graph that will be printed into verilog. Basically, if we try to combine the graphs of multiple instructions, we'll end up with multiple outputs from multiple modules directed into a single input in a single module. This means a multiplexer is meant to go there. Then using a look up table that defines the control signal needed for the mux to handle each of these outputs received at the single input. This would allows us to size the mux to exactly what is needed.

On second thought, Perhaps that we can avoid needing a 1 to 1 alignment between module input and output and keep the inter module logic through instead of defining the where each line leads to, we can define where each line comes from and their appropriate modifiers. To illustrate this I modified the JSON to reflect this idea. This can also be made easier with the use of regular expressions.

```
  {
    "instruction": "ADD",
    "modules": [
      {
        "module": "PC",
        "input": [
          {
            "name": "in", "length": 32, "from module": "pc", "from port": "pc"
  , "modifier": "%c+4" #(here %c is the regular expression where %c is to be
  replaced with the wire name)
          }
```

```
        ],
        "output": [
          {
            "name": "pc", "length": 32
          },
          {
            "name": "new_pc", "length": 32
          }
        ]
      },
      {
        "module": "instruction memory",
        "input": [
          {
            "name": "address", "length": 32, "from module": "PC", "from port": "pc"
          }
        "output": [
          {
            "name": "inst_out", "length": 32
          }
        ]
        ]
      },
      {
        "module": "register file",
        "input": [
          {
            "name": "src1", "length": 5, "from module": "instruction memory", "from port": "inst_out", "modifier": "%c[`IR_rs1]"
          },
          {
            "name": "src2", "length": 5, "from module": "instruction memory", "from port": "inst_out", "modifier": "%c[`IR_rs2]"
          },
          {
            "name": "dest", "length": 5, "from module": "instruction memory", "from port": "inst_out", "modifier": "%c[`IR_rd]"
          },
          {
            "name": "reg_write", "length": 1, "from module": "control", "from port": "reg_write", "modifier": null
          },
          {
            "name": "write_data", "length": 32, "from module": "alu", "from port": "out", "modifier": null
          }
        ],
        "output": [
          {
            "name": "read1", "length": 32
          },
          {
            "name": "read2", "length": 32
          }
        ]
      },
      {
        "module": "alu",
        "input": [
          {
```

```
            "name": "a", "length": 32, "form module": "register file", "from
    port": "read1", "modifier": null
          },
          {
            "name": "b", "length": 32, "from module": "register file", "from
    port": "read1", "modifier": null
          },
          {
            "name": "alufn", "length": 4, "from module": "control", "from port
    ": "read2", "modifier": null
          }
        ],
        "output": [
          {
            "name": "out", "length": 32
          }
        ]
      },
      {
        "module": "control",
        "input": [
          {
            "name": "op", "length": 5, "from module": "instruction memory", "
    from port": "inst_out", "modifier": "%c[`IR_opcode]"
          }
        ],
        "output": [
          {
            "name": "reg_write"
          },
          {
            "name": "alufn"
          }
        ]
      }
    ]
  }

```

The reason I want to leave as regular expressions is because I think it might prove easier to procedurally name inter module wires rather than them being fixed. would it?