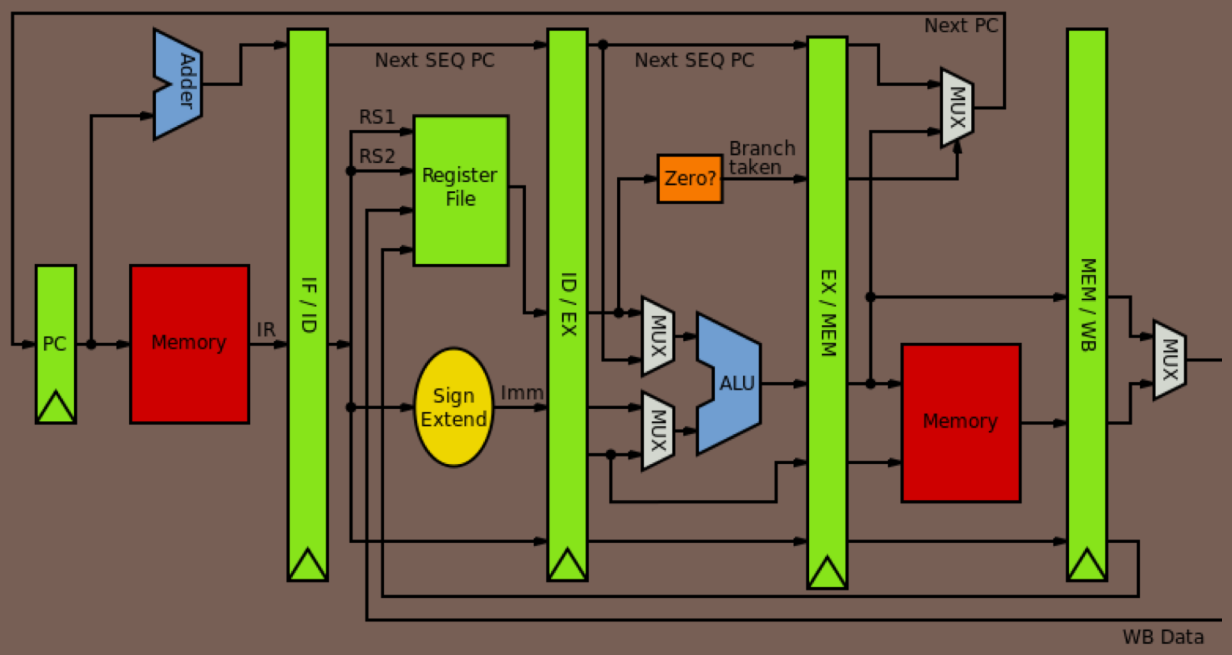


PROJECT 1 - 2

CSCE 3301

Mohamed Shalan, PhD

Time Line

- ~~MS1 (17/10): Names and task assignments~~
- MS2 (28/10): The datapath block diagram, as well as an alpha version of the CPU RTL model. Also, basic test cases to verify the CPU core RTL model. The RTL model must support all of the RV32I required instructions. At this stage, the 2nd constraint may not be implemented. Also, ebreak and ecall may not be implemented.
- MS3 (4/11): Add support for compressed instructions and the 2nd constraint.
- MS4 (11/11): Add simplified support for interrupts, ecall and ebreak. Also, add limited support for CSR instructions. More details will be given on how to implement those.

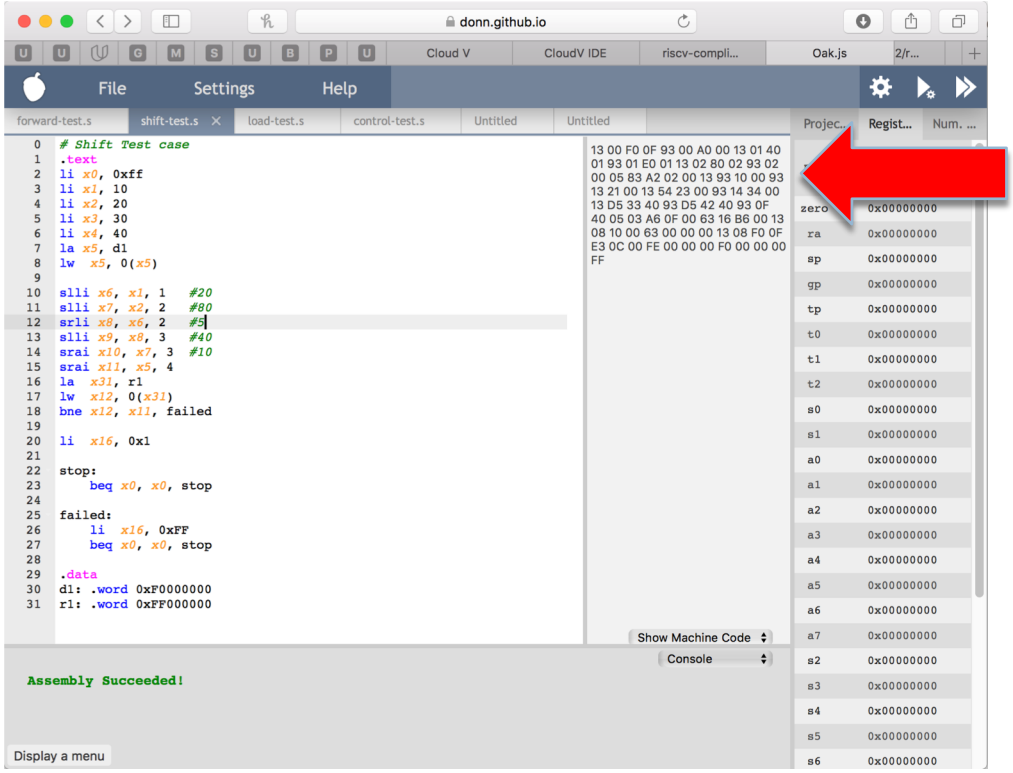
Constraints

1) A single ported memory is used for both data and instructions.
2) A memory transaction is done in 2 phases.
3) A dual ported memory is used for the register file.

Tips

- Developing “simple” testcases

  1) Use oak.js (https://donn.github.io/Oak.js/)



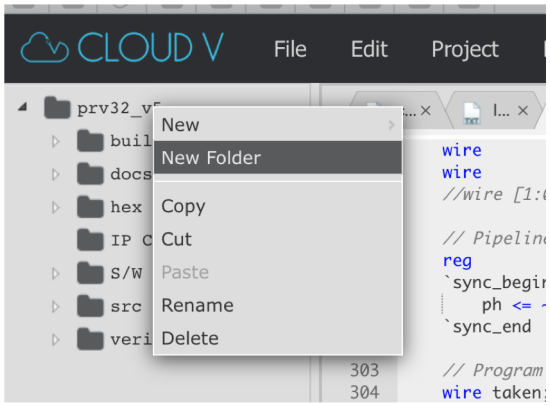
Tips

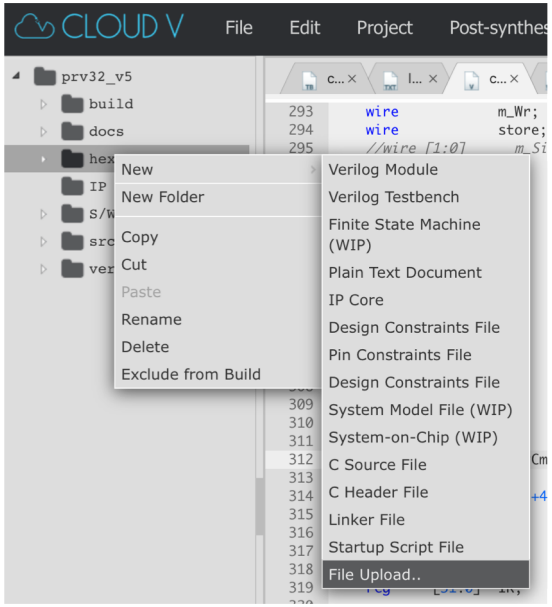
- Developing “simple” testcases

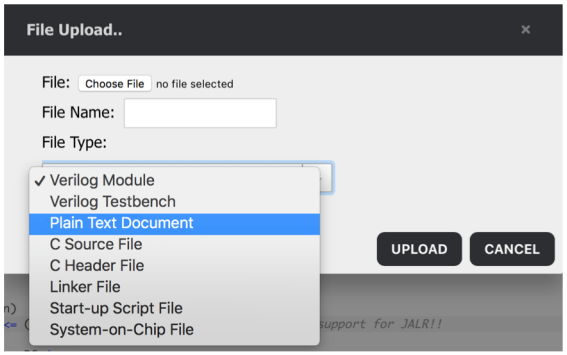
  2) Save the machine code into a file (.hex extension)

  3) Upload into CloudV workspace

  - Create a folder named hex
  - Upload







Tips

- Developing “simple” testcases

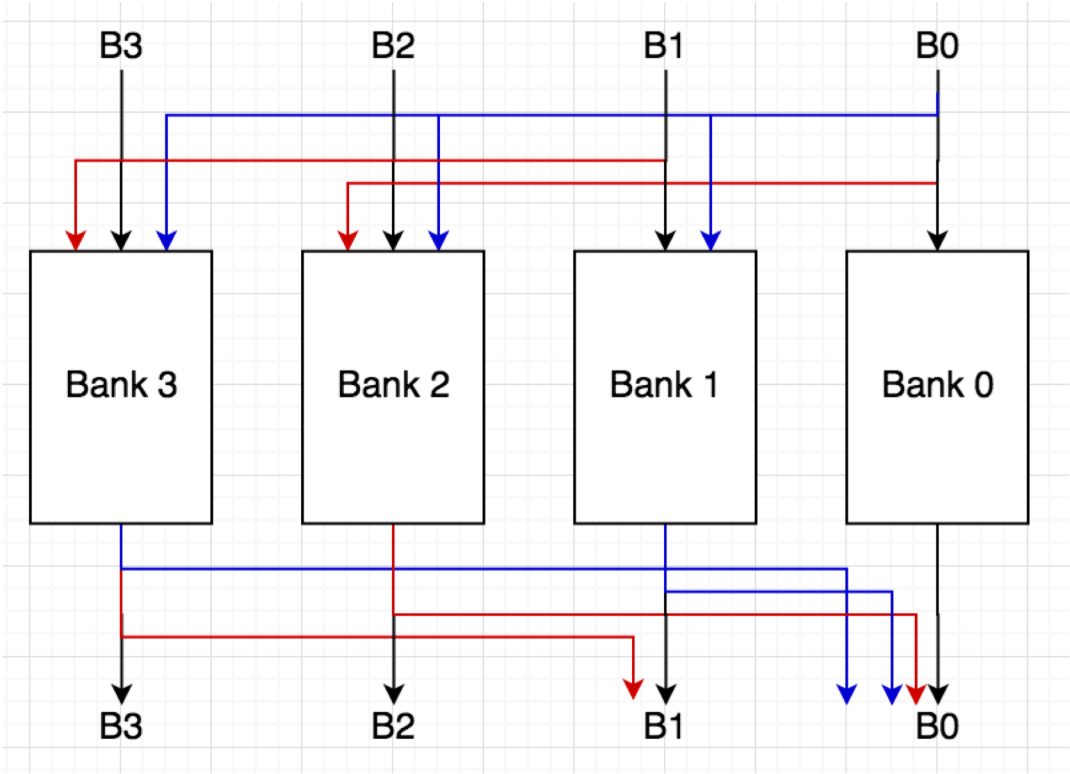
  4) Load the hex file into your memory (Verilog array) using the system task $readmemh()

```
reg[7:0] mem[(4*1024-1):0]; // 4KB memory

initial begin
    $readmemh("./hex/test2.hex", mem);
end
```

Tips

- Memory support for different data sizes



Tips

- Instruction Fetch and Compression
  - Because of mixing 16-bit and 32-bit instruction words, you may end up with fetching a 32-bit word from a location which is divisible by 2 (not 4). Each half word comes from different words!
  - If it is because the instruction that preceded this instruction is 16-bit and located @ a location which divisible by 4, you may always fetch one 32-bit instruction word every fetch cycle and save the unused half word to be used to construct the 32-bit word with the next fetch.
  - If it is because of branching/jumping (transfer the control to a location which is not divisible by 4, you may fetch twice (stall the pipeline) to get the word. Other solutions are too expensive and degrade the performance!

Tips

- Watch for this RAW Hazard

```
LW   x1, 0(x2)
SW   x1, 0(x4)
```

  - The forwarding is from the pipeline register to the memory

How to implement GT, LE, GTU, LEU

- Subtract and look at the flags: Z, C, V and S
  - EQ = Z
  - NE = ~Z;
  - LT = (S != V)
  - GE = (S == V)
  - LTU = ~C
  - GEU = C