States:

* Fetch:
  + Reg\_Src = r0
  + Src-out -> mem\_addr
  + Src\_incr = true
  + Mem\_out -> decode
  + Next: src\_index or src\_indirect or src\_incrment or (dst or exec)
* Src\_index
  + Fetch next word:
  + Reg\_Src = r0
  + Src-out -> mem\_addr
  + Src\_incr = true
  + Mem\_out -> src
* Src\_indirect
  + Fetch data at Rn
  + Reg\_Src -> mem\_addr
  + Src\_incr = false
  + Mem\_out -> src
* Src\_increment
  + Fetch data at Rn, Rn ++:
  + Reg\_Src Src -> mem\_addr
  + Src-out -> mem\_addr
  + Src\_incr = true
  + Mem\_out -> src
* Get\_Dst\_index
  + Fetch next word:
  + Reg\_Src = r0
  + Src-out -> mem\_addr
  + Src-out -> dst\_addr
  + Src\_incr = true
  + Mem\_out -> dst
* Execute
  + If a\_s == direct and a\_d == direct
  + Src = reg\_src
  + Dst = reg\_dst
  + Dst\_in = op(src, dst)
* Put\_dst\_index
  + dst\_addr -> mem\_addr
  + Src\_incr = false
  + Op(src, dst) -> Mem\_in
* Jump
  + R0 = r0 + offset
* Fetch reg->reg:
  + mem\_addr = PC
  + PC <- PC + 2
  + ---------------------------
  + instr <- mem\_out
  + typ, src\_reg, dst\_reg, as, ad = decode (instr)
  + If typ == two\_arg and as == reg and ad == reg:
    - src, dst =Regfile(src\_reg, dst\_reg)
    - dst\_in = op(src, dst)
* Fetch indirect->indirect:
  + mem\_addr = PC
  + PC <- PC + 2
  + instr <- mem\_out
  + typ, src\_reg, dst\_reg, as, ad = decode (instr)
  + If typ == two\_arg and as == indirect and ad == indirect:
    - mem\_addr, \_ = Regfile(src\_reg, dst\_reg)
    - ----------------------
    - src <- mem\_out
    - \_, mem\_addr = Regfile(src\_reg, dst\_reg)
    - ----------------------
    - dst <- mem\_out
    - mem\_in = op(src, dst)
    - ----------------------

Flipflops:

Registers

Current\_instr

src\_val – when dst is from memory

dst\_address – when dst is in memory

state

states:

fetch – fetch instr (and decode if reg-reg)

src\_index

src\_mem (also for indirect/autoincr)

dst\_index

dst\_mem

dst\_wr

push

reti1

reti2

jmp

Combinatorial resources:

Read registers

memory

alu

instruction decode