6-10-16

Microprocessors EE308/9 — RISC multicycle Design (Extra class)

31 26	25 21	20 16	15 11	10 5	0
OP	Rd	RS1	RS2		Function
OP	Rd	RS1	Imm 16		
OP	Imm. 26				

MINI - DLX

R → ADD Rd, RS1, RS2
SUB
AND
XOR
SLL

I → LW Rd (RS1) Imm16
SW Rd (RS1) Imm16
BEQ Rd RS1 Imm16

J → J Imm26

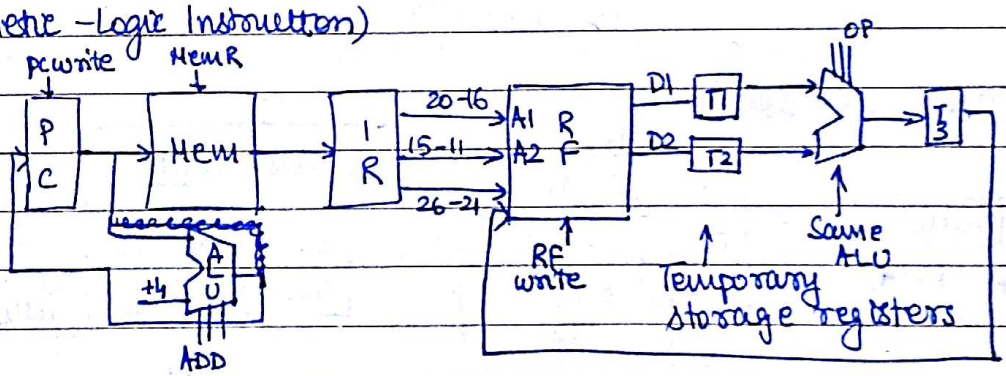
Multiple Cycles

→ Fetch
→ Decode & Reg Read
→ Execute/Computation of memory address
→ Memory Access (R/W)
→ RF Write

NOTE: Multi-cycle gives improved performance wrt time but the controller design becomes complicated

R-Type (Arithmetic - Logic Instruction)

Fetch:



State1 (add)

PC → mem
db → IR
PC → alu
+4 → alu
alu → PC

State2 (Register read)

I_{20-16} → A1-RF
I_{15-11} → A2-RF
D1 → T1
D2 → T2

State 3 (OP) (ALU operation)

t1 → alu
t2 → alu
alu → T3

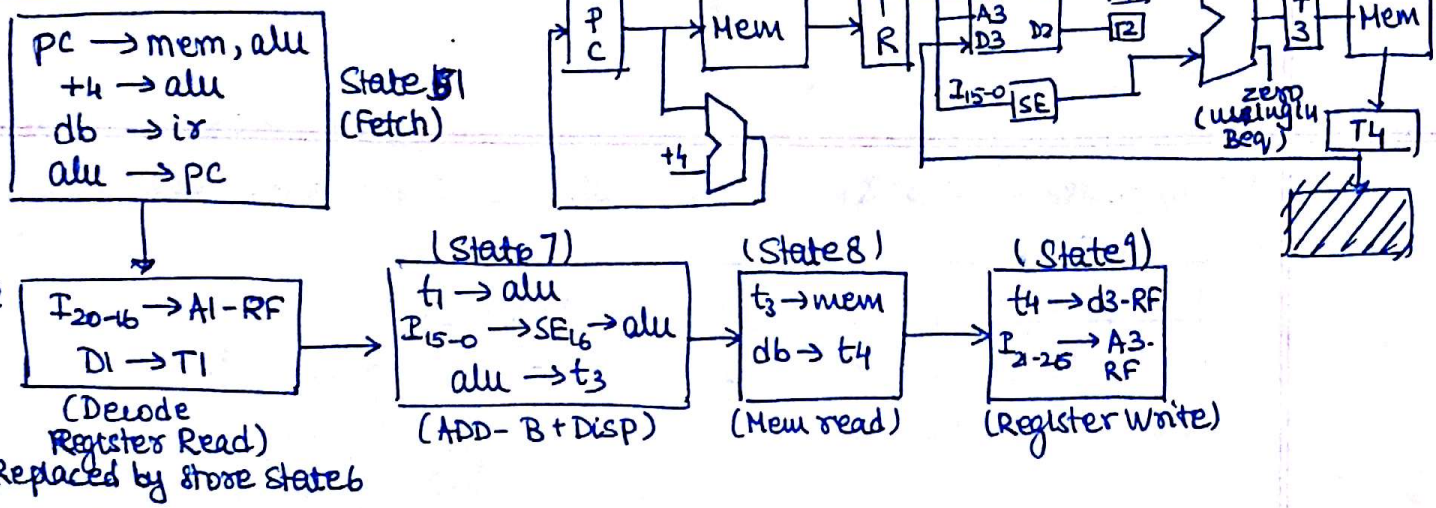
Only one operation per cycle. Here because alu is already in use we do not access register file/mem

State 4 (Register write)

I_{26-21} → A3-RF
t3 → D3-RF

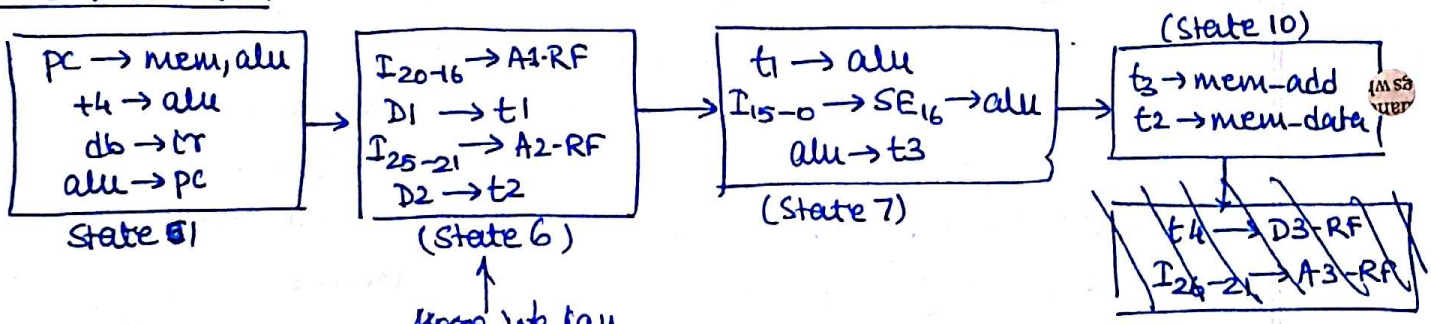
NOTE: There are no duplicates. They are made so as to simplify the flow diagram

Load Instruction



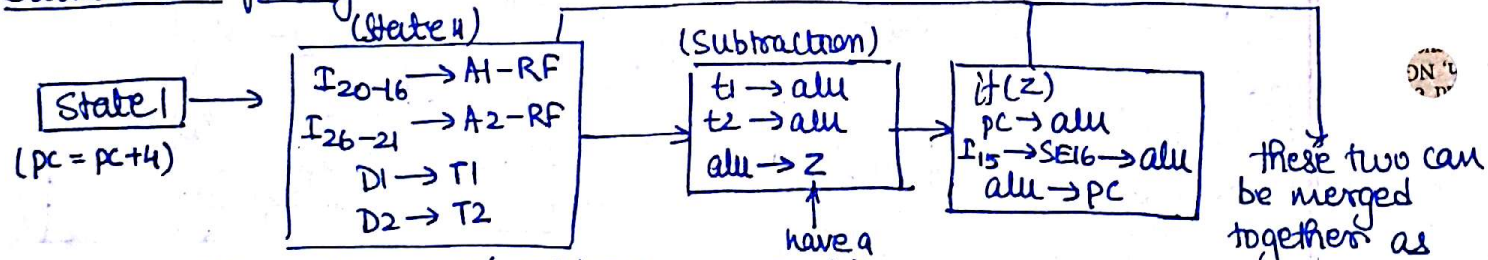
—Replaced by store state6

Store Instruction



These we can use the same state for load as well because doing the extra op of reading one more register does not hurt the time

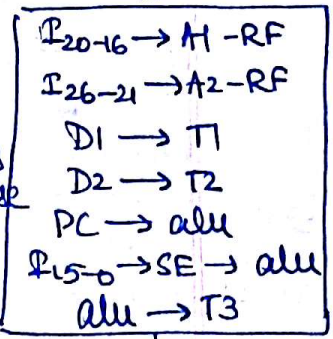
Branch on Equality



(All registers need not have an enable because it does not hurt the operation. This reduces the no. of control signals) So we do not control T1 & T2 They are always written.

have a one bit register

This merger was possible because both the states occur in '1' and not in sequence



This merge need not be done if the state 11 is to be used as a generic state for other instructions as well.

↔ Read next page

No two major operations must be combined sequentially

- ALU
- Memory R/W
- Register R/W

- The flowchart for Jump can be written similarly (Beq like)
- Now we look at all our states and all the possible inputs to an unit and add necessary multiplexors.

  Eg: Multiplexor at the input of A_2 because it can take I_{15-11} or I_{26-21}. A larger number of multiplexors will be required before
- We require two different sign extends if the jump instruction also gives an offset.
- Store at all the states and all possible inputs to put multiplexors
- Merge all flowcharts to minimise the total number of states

Redundancies can be introduced to minimise states. This is done only if # of states is more important than power consumption.
- This will then lead to a state-flow diagram with branches depending on the op code.

$I_{15-11} \rightarrow A_2$-RF and $I_{26-21} \rightarrow A_2$-RF occurs on states which only differ in this one line. Hence we can add an control signal for this purpose which can be generated by the decoder itself and not the FSM

- Something about VHDL implementation of FSMs
  - → 3 processes — one for instantiating flip-flops
    - — next-state logic
    - — control signal logic
  - → specify all signal everytime to prevent latches
  - → always have default case to prevent errors, any errors will cause a reset.