RISC Design

Virendra Singh
Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering, and
Dept. of Computer Science & Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@{ee, cse}.iitb.ac.in

EE-739: Processor Design @ IITB

Lecture 14 (12 February 2022)

CADSL

RISC → Simple instructions
→ uniform & fixed encoding
→ Limited addressing modes.

$S \xrightarrow{T} S'$

↓ one go – Controller

Combinational logic.

in-efficiency

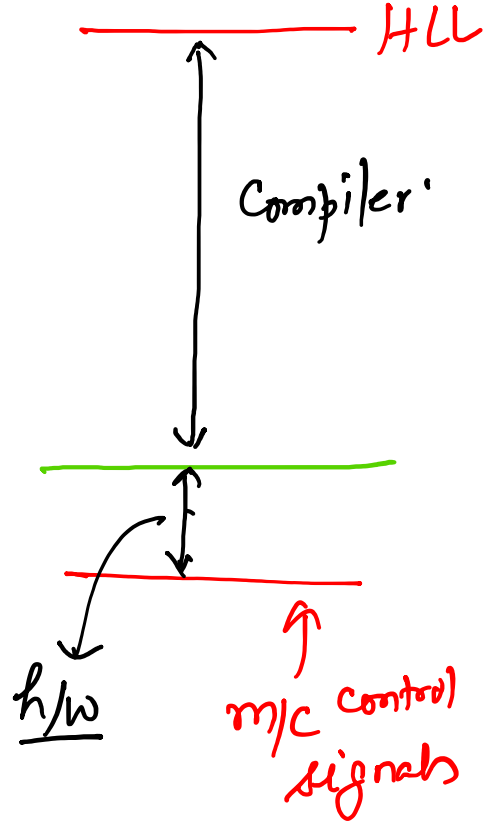
Time = τ_2 longest sequential operation

Load.

fetch → operand fetch → Generate add. → Read memory → Update the state

8085

A/L
Store
Branch



$S \xrightarrow{I} S'$

$S \rightarrow P \rightarrow P' \rightarrow P'' \rightarrow S'$

(. Multicycle Implement)

- Read & write – Memory
- Read & write – RF
- ALU operation

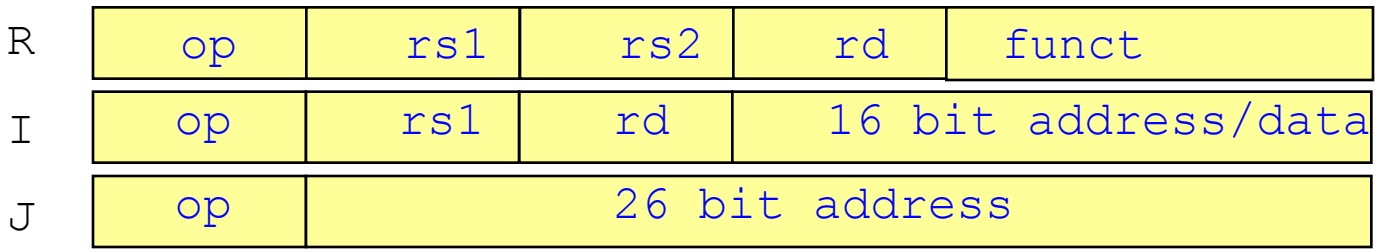
{Improve efficiency
{Better re-use of resources

↑

Cost

Overview of DLX ISA

- simple instructions, all 32 bits wide
- very structured, no unnecessary baggage
- only three instruction formats



- rely on compiler to achieve performance

Example Instruction Set: MIPS Subset

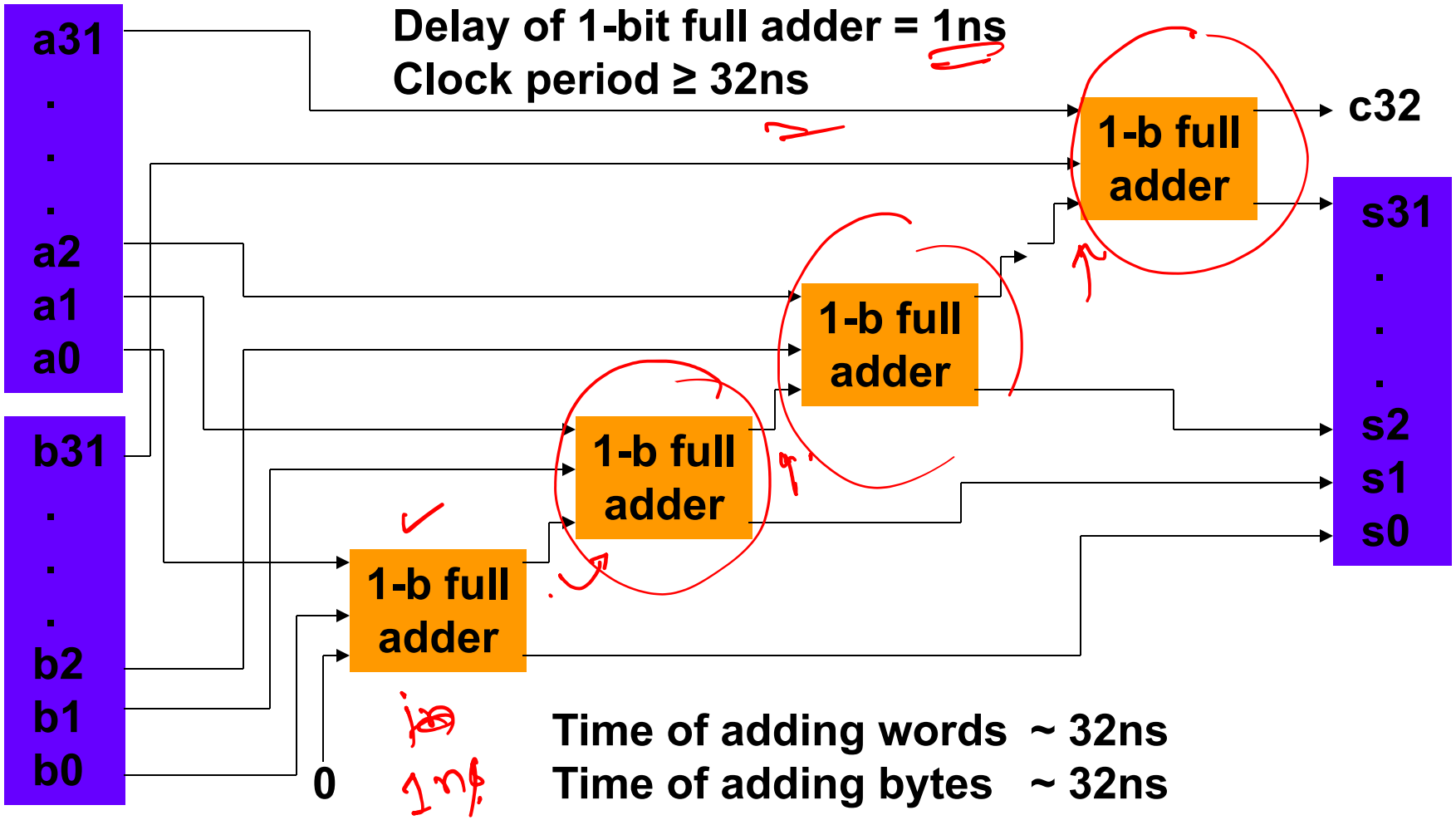
MIPS Instruction – Subset

- Arithmetic and Logical Instructions
  - add, sub, or, and, slt
- Memory reference Instructions
  - lw, sw
- Branch
  - beq, j

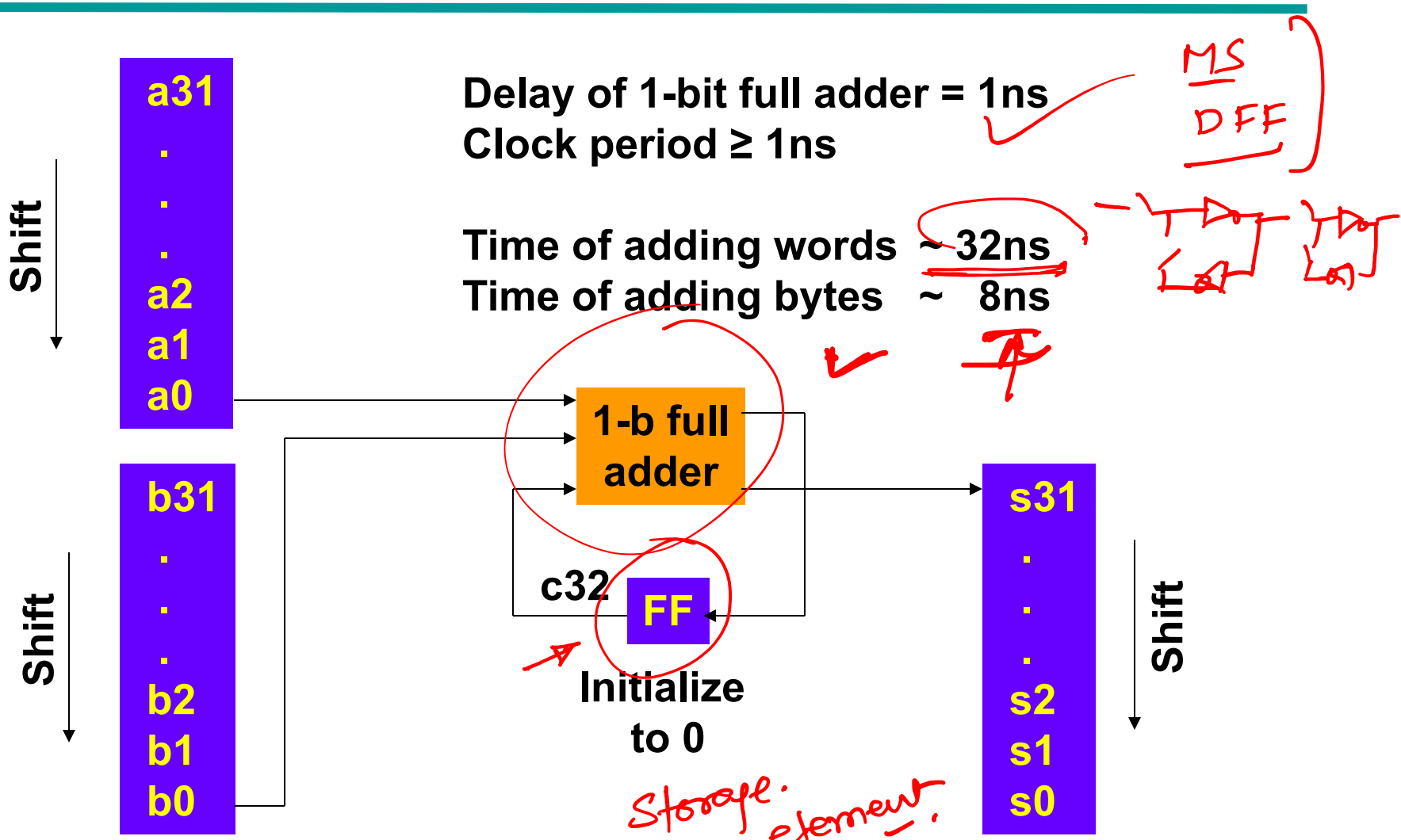
How Fast Can the Clock Be?

- If every instruction is executed in one clock cycle, then:
  - Clock period must be at least 8ns to perform the longest instruction, i.e., *lw*.
  - This is a single cycle machine.
  - It is slower because many instructions take less than 8ns but are still allowed that much time.
- Method of speeding up: Use multicycle datapath.

A Single Cycle Example



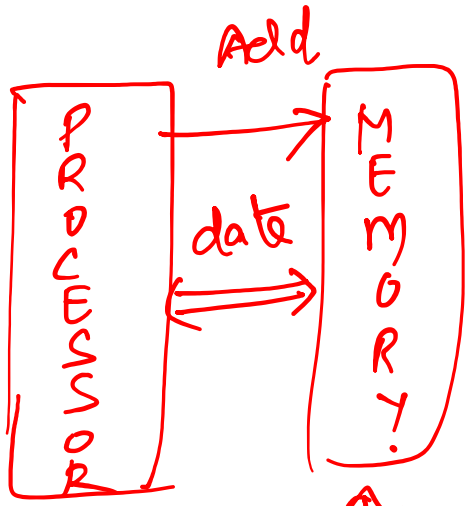
A Multicycle Implementation



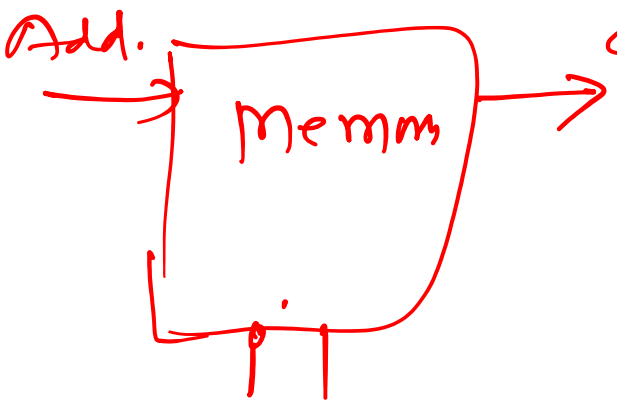
Major tasks:
- Read & Write of Memory — 2 ns
- Read & Write of RF — 1 ns
- ALU operation — 2 ns

(do not want to cascade)

Parallel ~~tasks~~ major tasks are fine



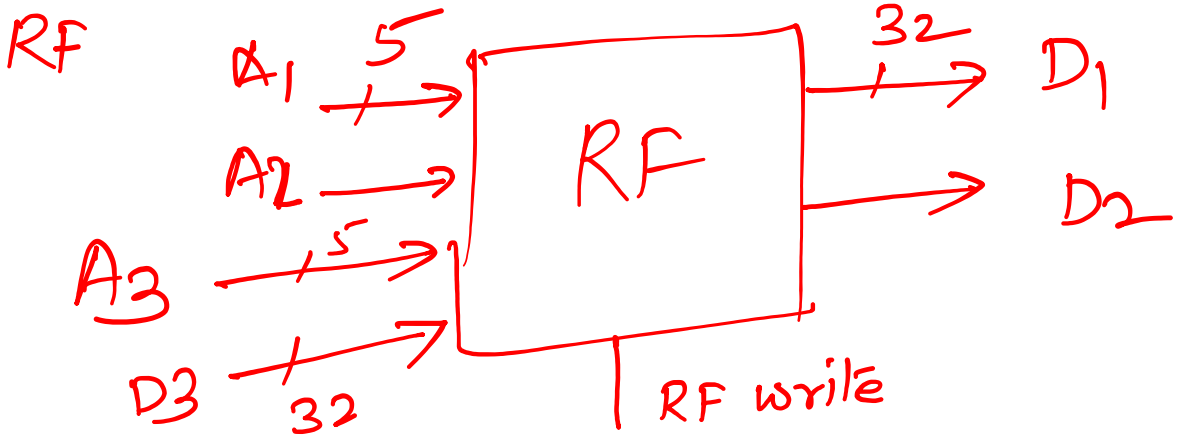
⇒

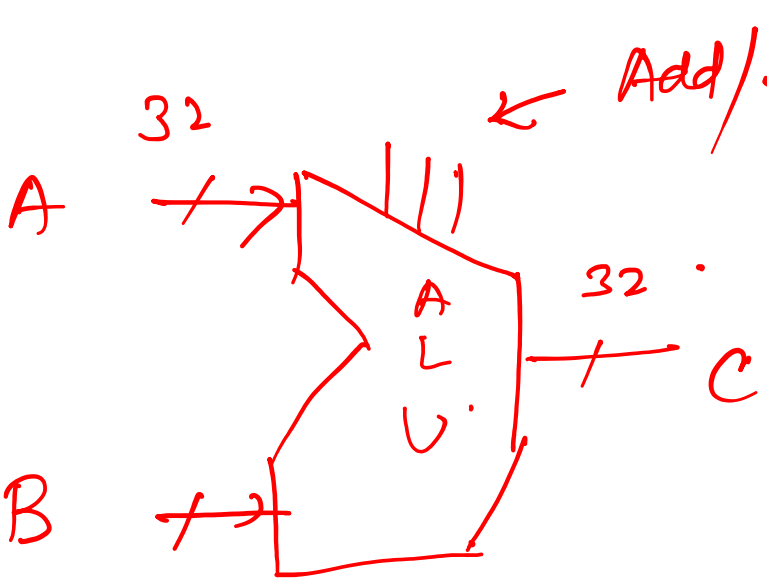


{ ROM / RAM }

one memory (Instruction / data)

ROM





MUX based.

Data path ... Steering Logic ⇒ Point-to-point

- HKT ✓ OT
  OT HKT.

⇓

fetch instruction (PC / IP) ✓

⇒ update the PC

⇒ understand instruction (Decoder)

→ Fetch the operand. ✓ (RF / Immediate)

→ Execute the instruction. → Effective add. of memn.
↕
→ Access memory

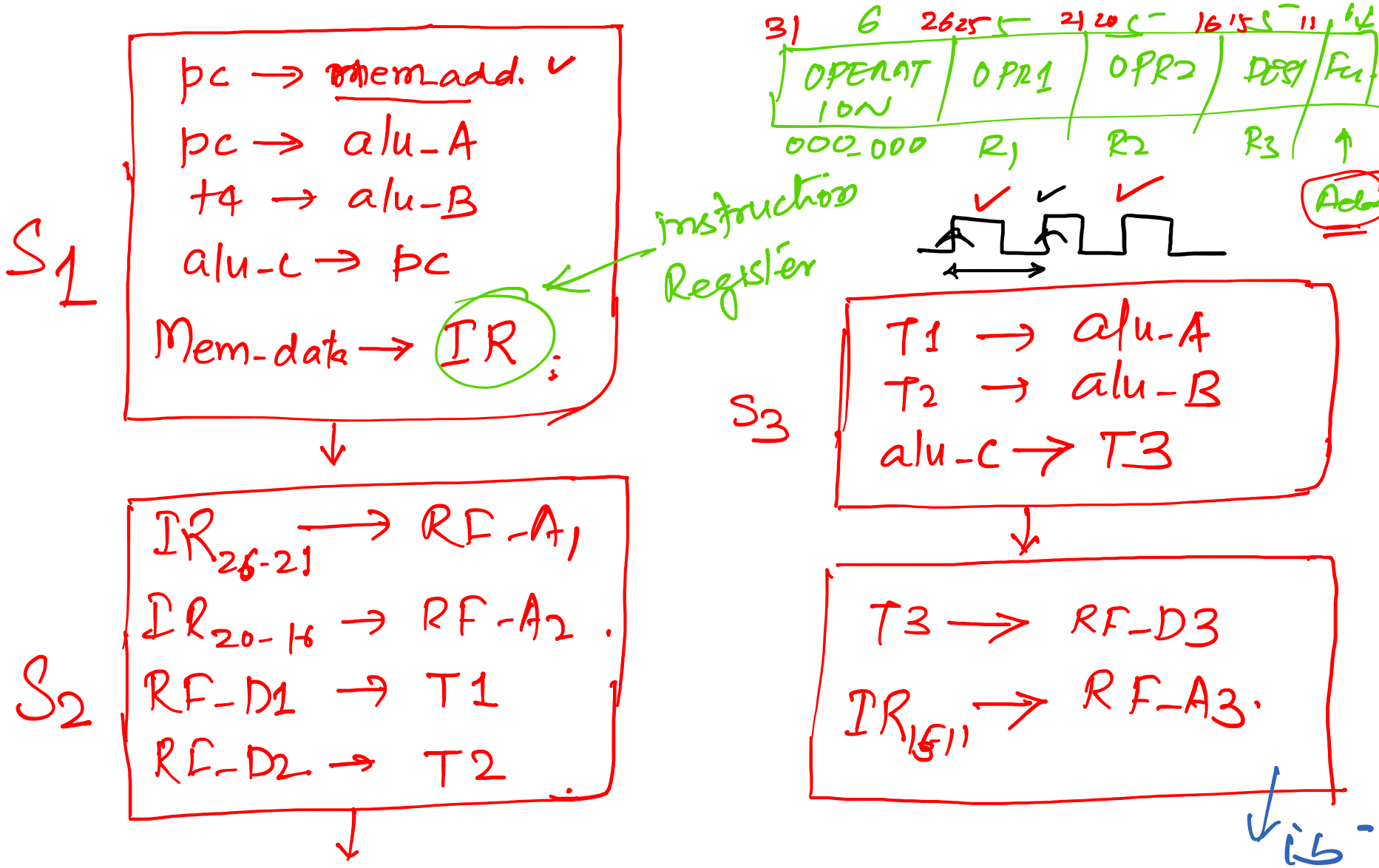
→ Update the system state (RF)

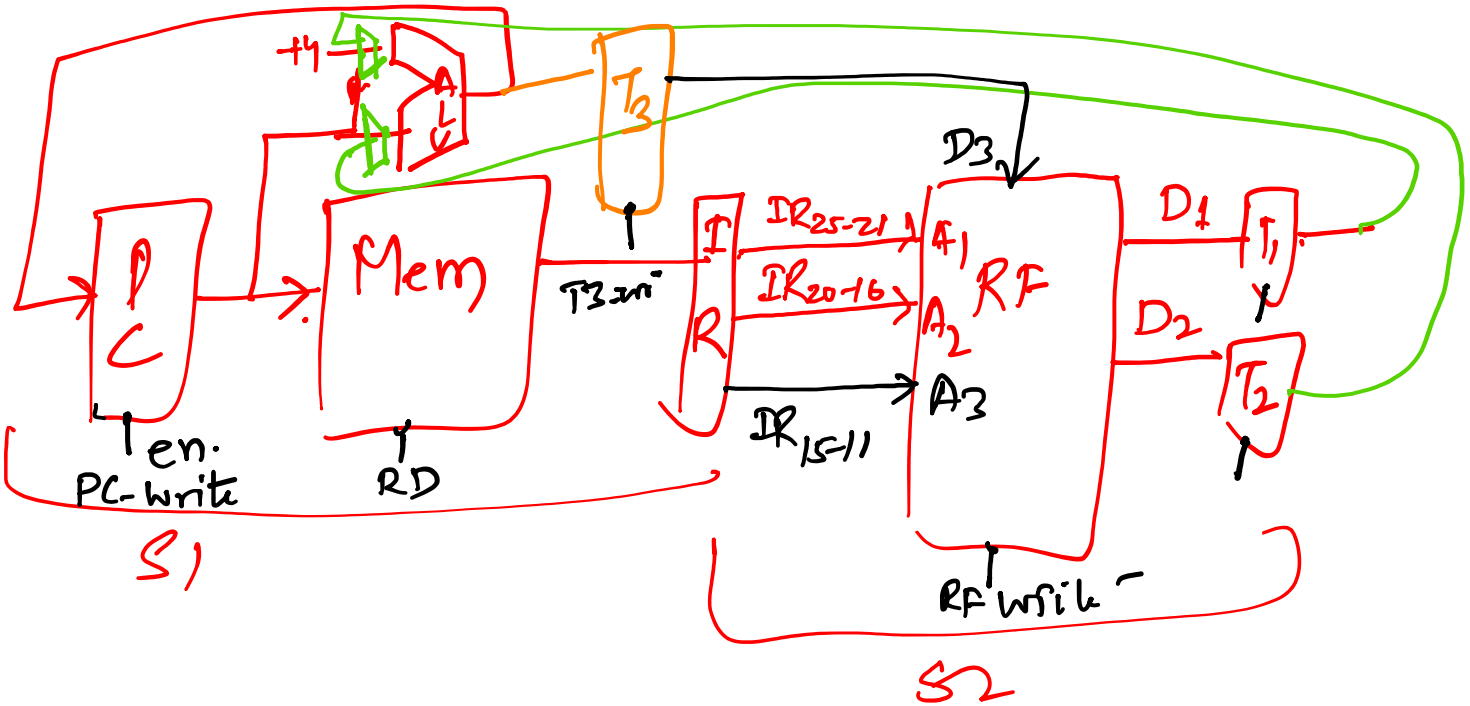
R-Type Instructions

- Also known as arithmetic-logical instructions
- add, sub, slt
- Example: add $t0, $s1, $s2
  - Machine instruction word

    000000 10001 10010 01000 00000 100000

    opcode $s1 $s2 $t0 function
  - Read two registers
  - Write one register
  - Opcode and function code go to control unit that generates RegWrite and ALU operation code.


31 6 26 25 21 20 16 15 11 10
OPERATION
OPR1
OPR2
DES
FU
000_000
R1
R2
R3
Add
S1
PC → mem_add.
PC → alu_A
+4 → alu_B
alu_C → PC
Mem_data → IR
instruction Register
S2
IR25-21 → RF_A1
IR20-16 → RF_A2
RF_D1 → T1
RF_D2 → T2
S3
T1 → alu_A
T2 → alu_B
alu_C → T3
T3 → RF_D3
IR15-11 → RF_A3


+4
ALU
T3
D3
PC
Mem
T3 out
IR
IR25-21
IR20-16
IR15-11
A1
A2
A3
RF
D1
D2
T1
T2
en.
PC-write
RD
RF write
S1
S2

Steering.

Load and Store Instructions

- I-type instructions
- lw $t0, 1200 ($t1) # incr. in bytes

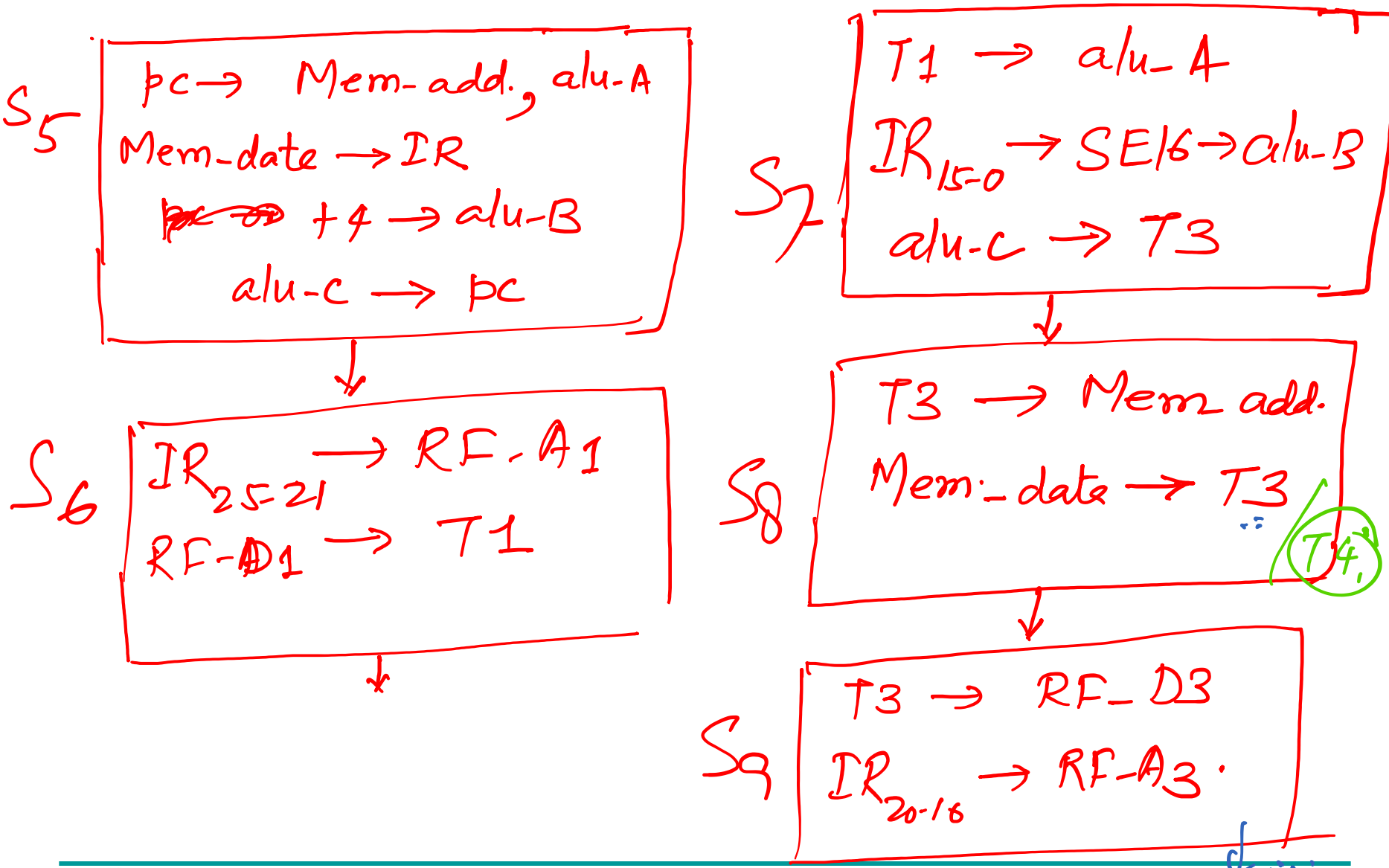
  100011 01001 01000 0000 0100 1011 0000

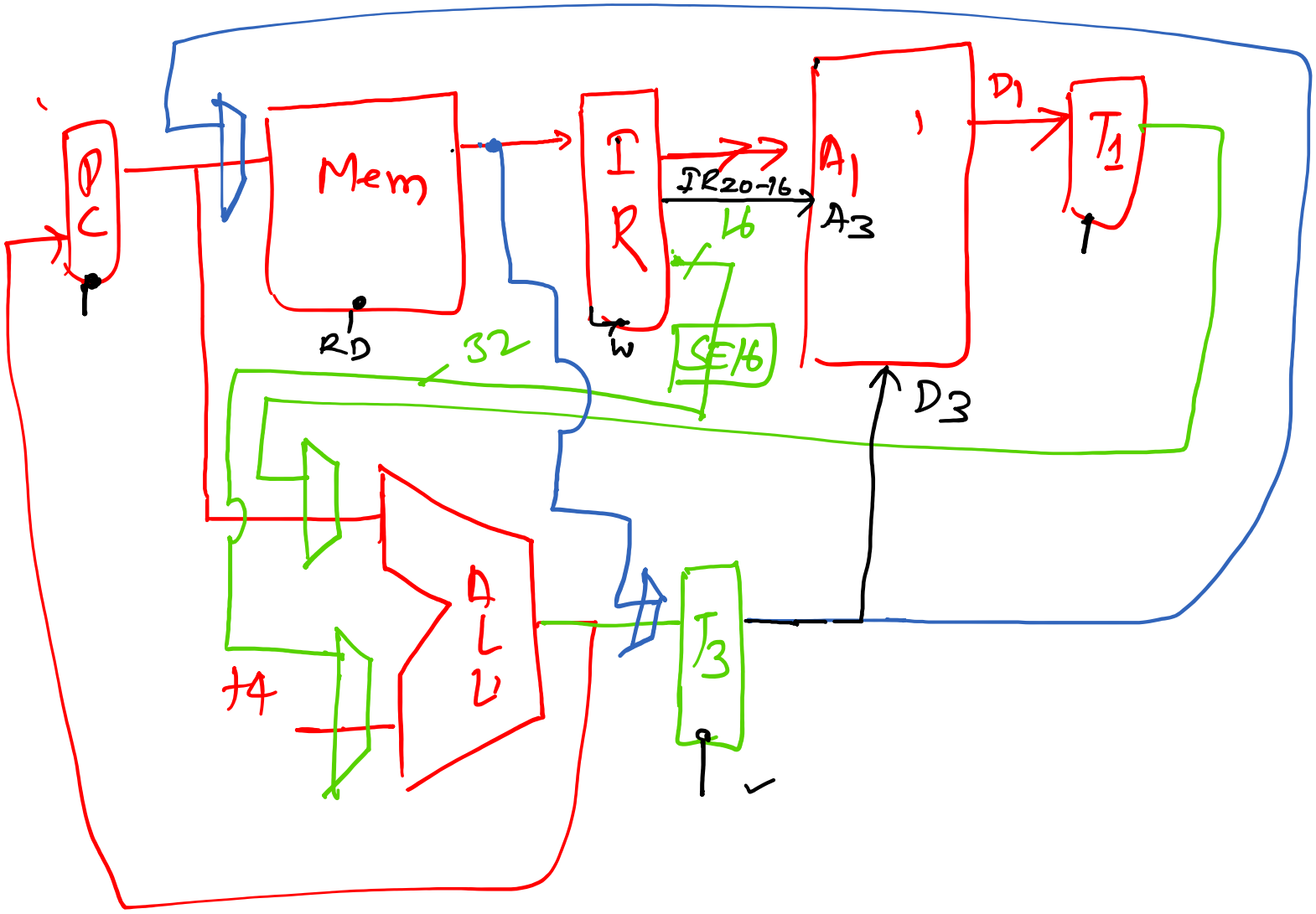
  opcode $t1 $t0 1200

- sw $t0, 1200 ($t1) # incr. in bytes

  101011 01001 01000 0000 0100 1011 0000

  opcode $t1 $t0 1200


S5
pc → Mem-add., alu-A
Mem-date → IR
pc → +4 → alu-B
alu-C → pc
S6
IR25-21 → RF-A1
RF-D1 → T1
S7
T1 → alu-A
IR15-0 → SE16 → alu-B
alu-C → T3
S8
T3 → Mem add.
Mem-data → T3
T4
S9
T3 → RF-D3
IR20-16 → RF-A3.
IF


PC
Mem
RD
IR
W
IR20-16
16
A1
A3
D1
T1
SE16
32
D3
ALU
T4
T3

Load and Store Instructions

- I-type instructions
- lw $t0, 1200 ($t1) # incr. in bytes

  100011 01001 01000 0000 0100 1011 0000

  opcode $t1 $t0 1200

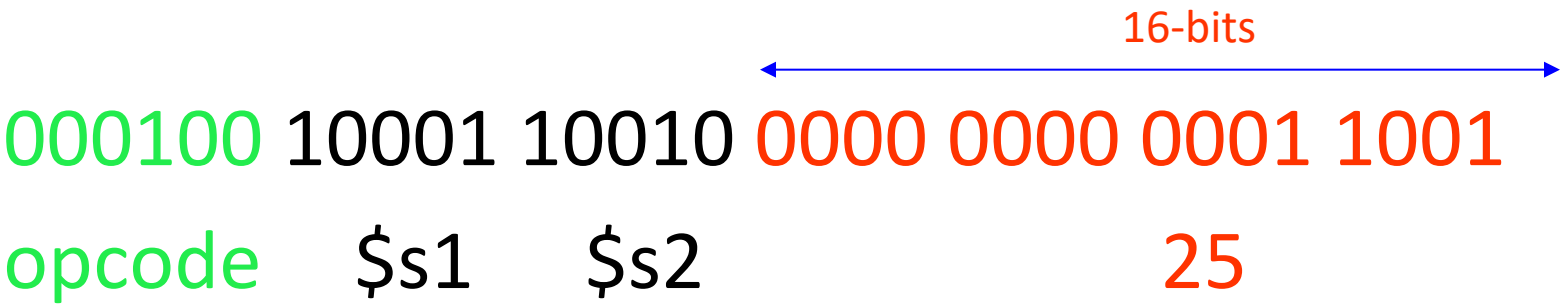
- sw $t0, 1200 ($t1) # incr. in bytes

  101011 01001 01000 0000 0100 1011 0000

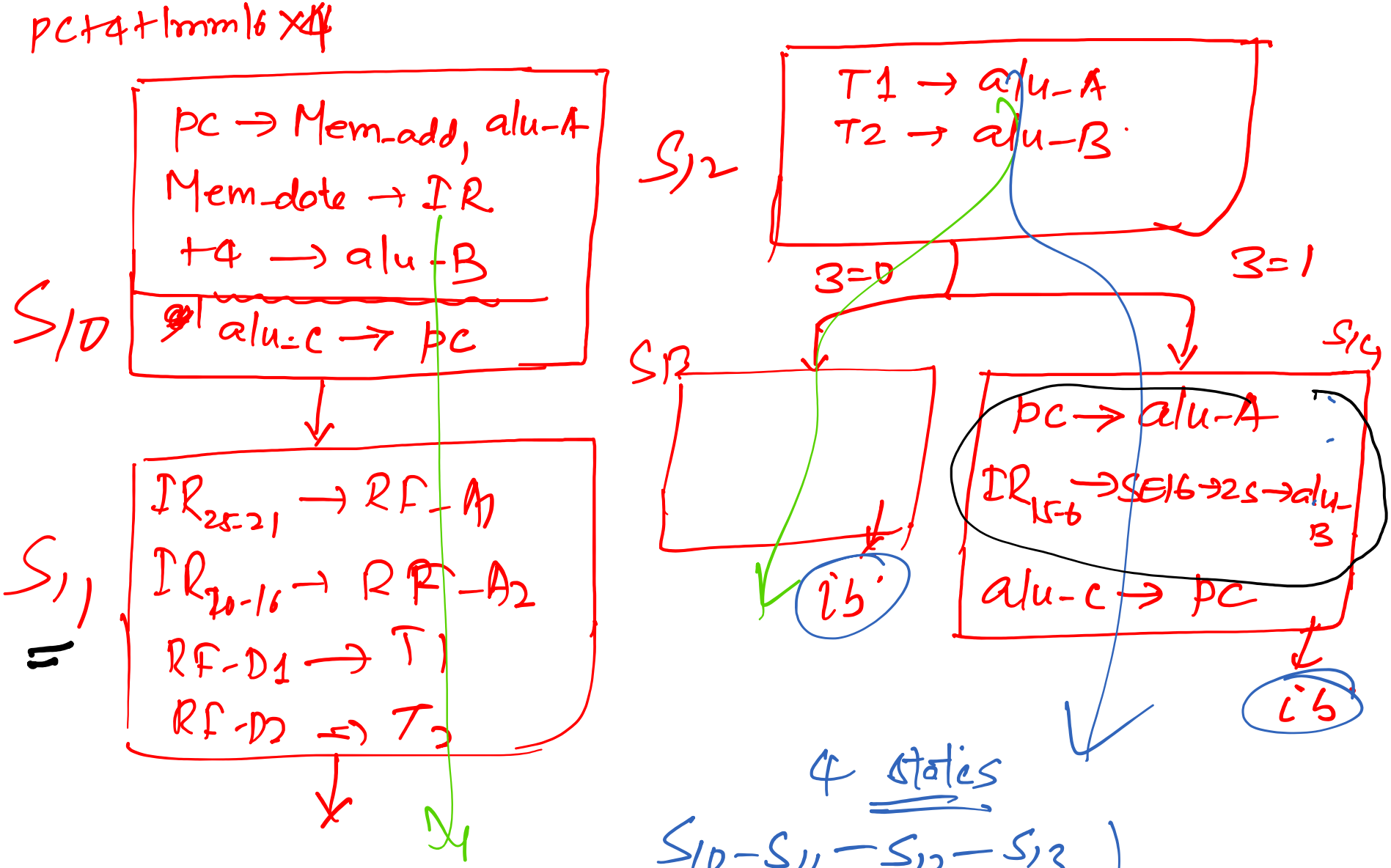
  opcode $t1 $t0 1200

Branch Instruction (I-Type)

- beq $s1, $s2, 25 # if $s1 = $s2, advance PC through 25 instructions



Note: Can branch within ± 2^{15} words from the current instruction address in PC.


PC+4+Imm16 X4
PC → Mem_add, alu-A
Mem_data → IR
+4 → alu-B
alu-C → PC
S10
IR25-21 → RF-A1
IR20-16 → RF-A2
RF-D1 → T1
RF-D2 → T2
S11
T1 → alu-A
T2 → alu-B
S12
Z=0
Z=1
S13
i5
PC → alu-A
IR15-0 → SE16→2S → alu-B
alu-C → PC
S14
i5
4 states
S10-S11-S12-S13
S10-S11→S12⇒S14

S_{10}

PC → Mem add, alu-A
t4 → alu-B
Mem_data → IR
alu_c → PC

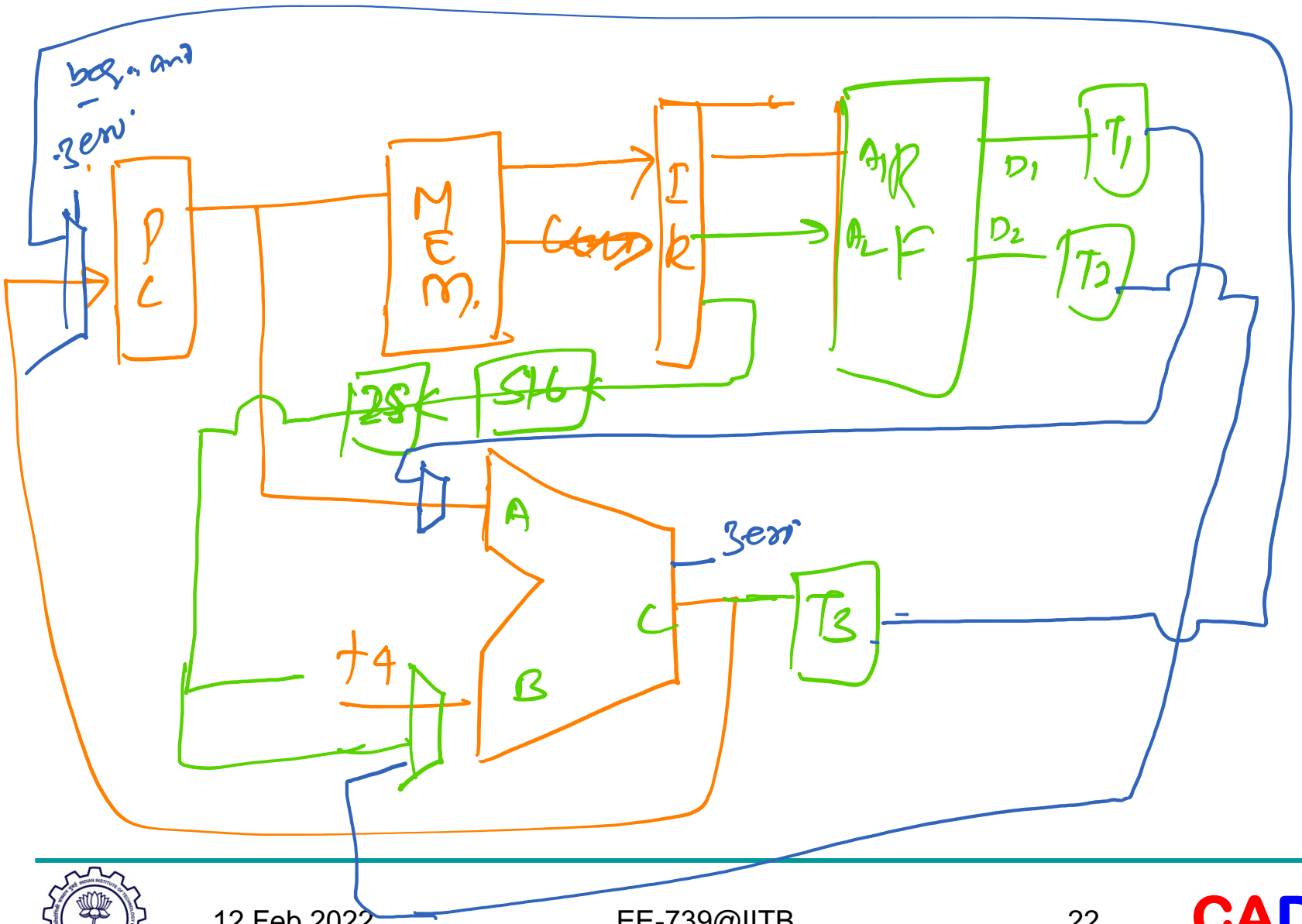
S_{12}'

T1 → alu-A
T2 → alu-B
if (Zero == 1)
then: T3 → PC

S_{11}'

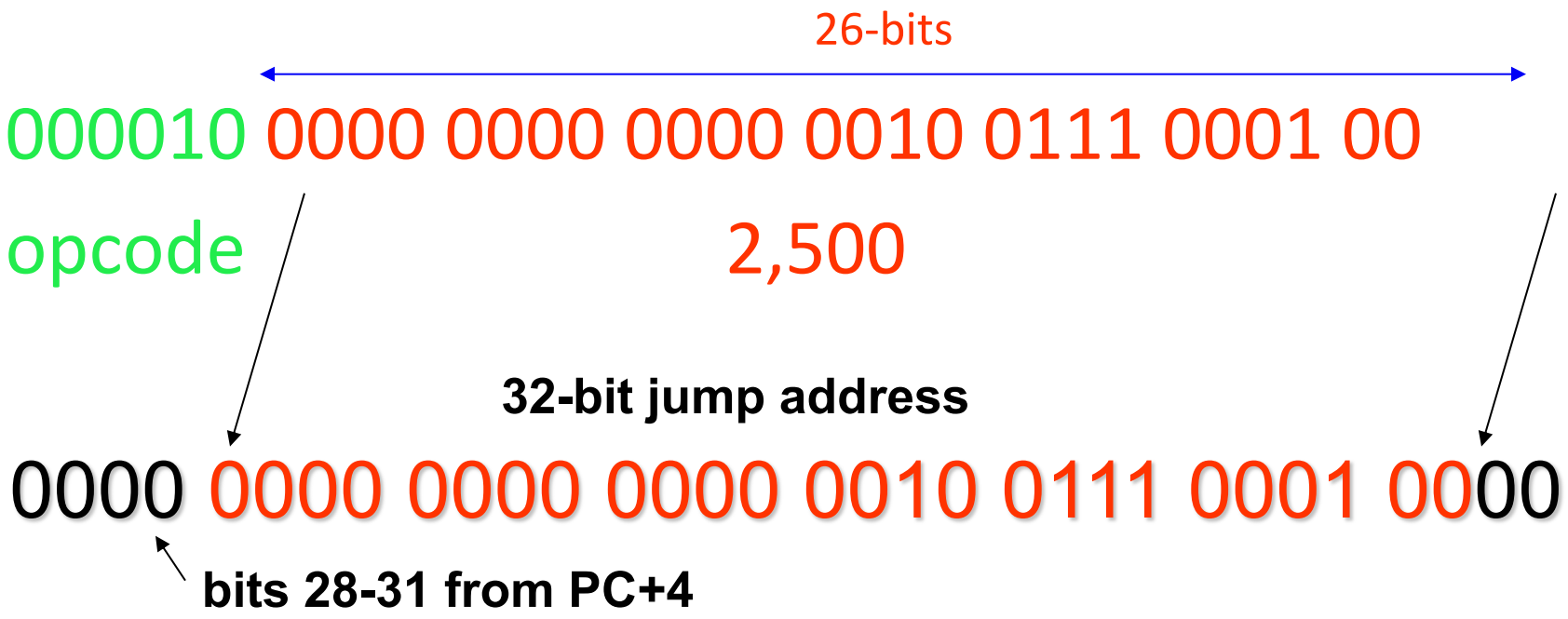
IR_{25-21} → RF-A1
IR_{20-16} → RF-A2
RF-D1 → T1
RF-D2 → T2
PC → alu-A
IR_{15-0} → SE16→25 → alu-B
alu-C → T3

$S_{10} \rightarrow S_{11}' \rightarrow S_{12}'$

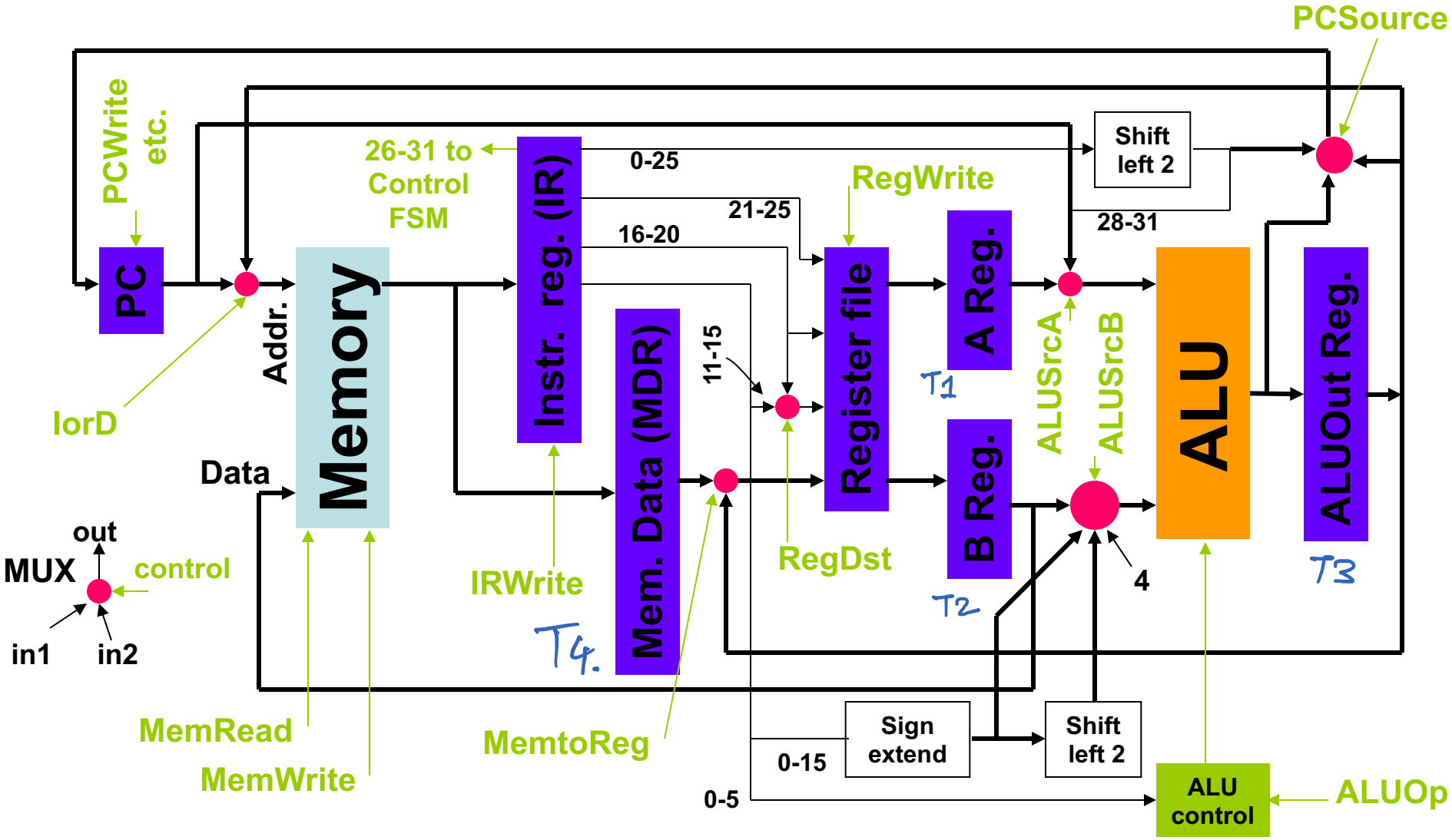

PC
MEM.
I
R
AR
ALF
D1
D2
T1
T2
T3
A
B
C
zero
+4

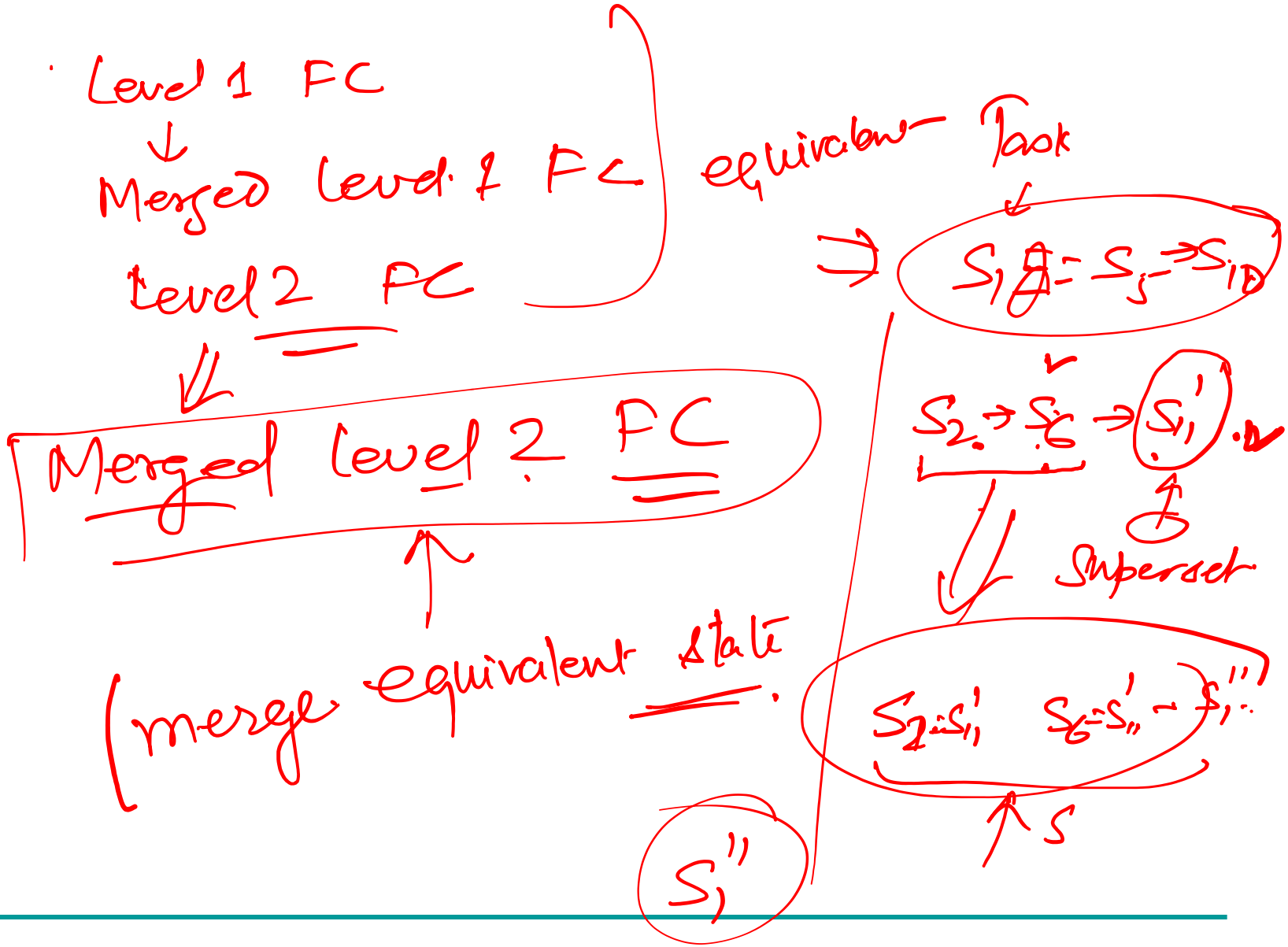
J-Type Instruction

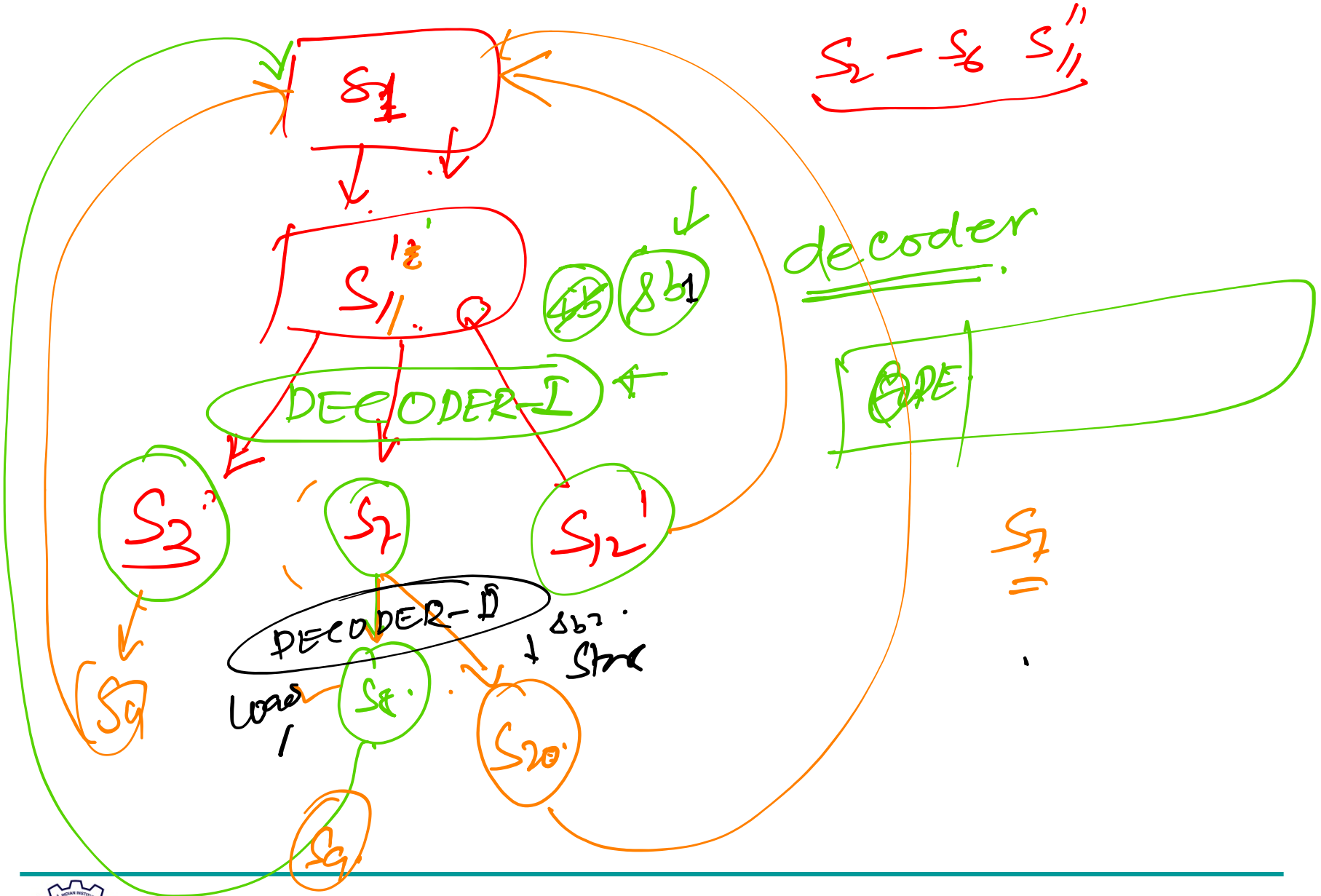
- j 2500 # jump to instruction 2,500

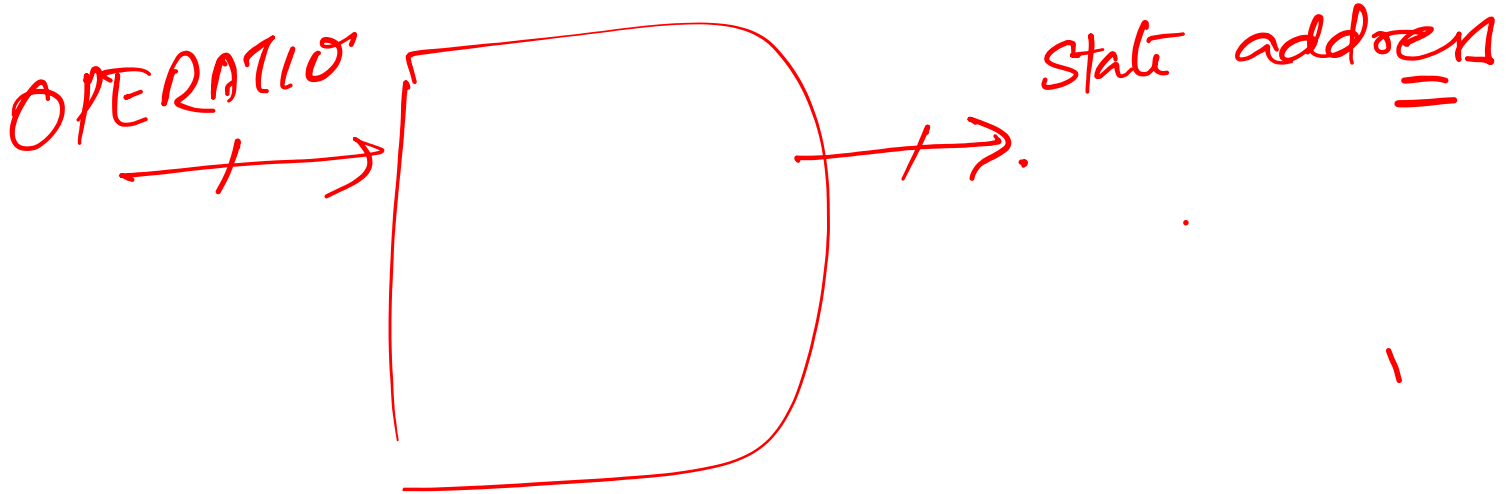


Multicycle Datapath




Level 1 FC
Merged Level 1 FC
Level 2 FC
Merged Level 2 FC
merge equivalent state
equivalent Task
$S_1 \rightarrow S_5 \rightarrow S_{10}$
$S_2 \rightarrow S_6 \rightarrow S_1'$
Supersede
$S_2 = S_1'$ $S_6 = S_{11}' = S_1''$
S
S_1''


S1
S11'
DECODER-I
S2'
S7
S12'
DECODER-D
Load
Store
S9
S8
S10
S6
S11
decoder
CODE
S7
=


OPERATIO
Stati addres

3 to 5 Cycles for an Instruction

Step	R-type (4 cycles)	Mem. Ref. (4 or 5 cycles)	Branch type (3 cycles)	J-type (3 cycles)
Instruction fetch	IR ← Memory[PC]; PC ← PC+4			
Instr. decode/ Reg. fetch	A ← Reg(IR[21-25]); B ← Reg(IR[16-20]) ALUOut ← PC + (sign extend IR[0-15]) << 2			
Execution, addr. Comp., branch & jump completion	**ALUOut ← A op B**	**ALUOut ← A+sign extend (IR[0-15])**	**If (A= =B) then PC←ALUOut**	**PC←PC[28-31] || (IR[0-25]<<2)**
Mem. Access or R-type completion	**Reg(IR[11-15]) ← ALUOut**	**MDR←M[ALUout] or M[ALUOut]←B**		
Memory read completion		**Reg(IR[16-20]) ← MDR**		

Thank You