VLSI SYSTEMS AND ARCHITECTURE

2021-22

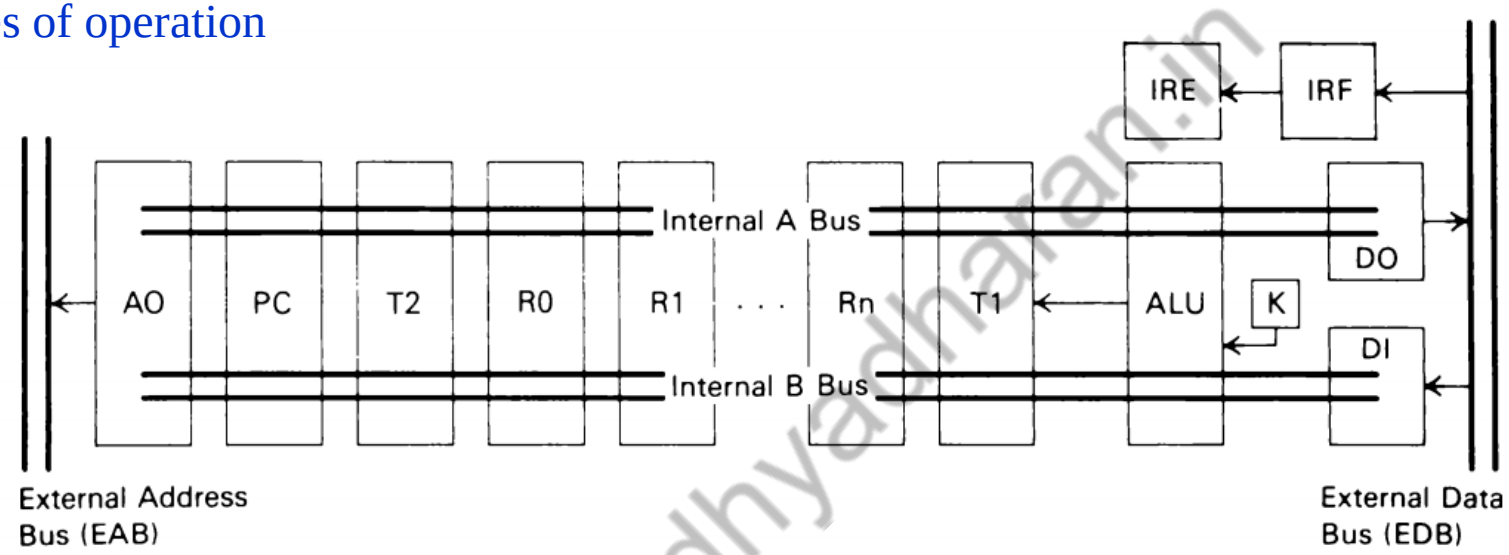
Lecture 7

Hardware Flow Chart-Part-2

By Dr. Sanjay Vidhyadharan

MIN execution unit block diagram

Rules of operation



ALU	Arithmetic and Logic Unit	DO	Data Out buffer	K	Constant generator
AO	Address Out buffer	IRF	Instruction Register for Fetch	PC	Program Counter
DI	Data Input register	IRE	Instruction Register for Execution	R0-Rn	Programmer's registers
				T1, T2	Temporary registers

Example Rules of Operation

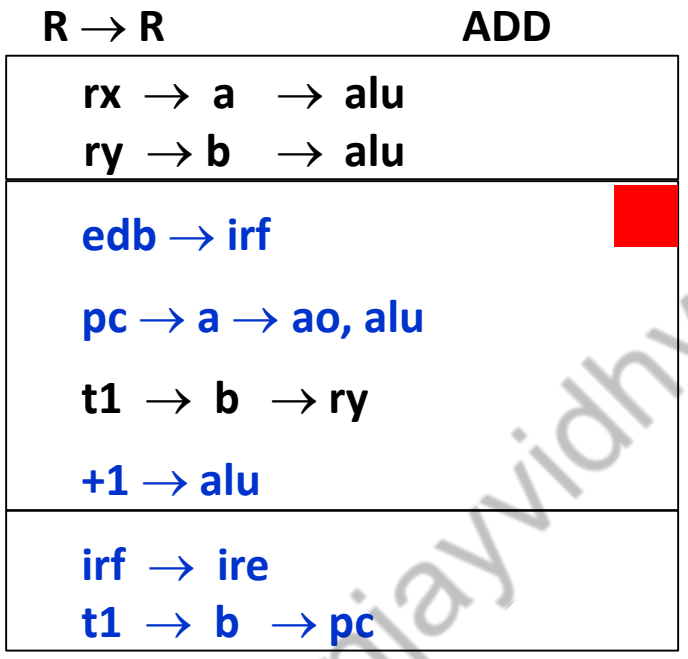
1. A transfer from source to bus to destination takes one state time.
2. A source can drive up to three destination loads.
3. Inputs to the ALU are from the A (internal) bus and either K (values 0, +1, −1) or the B (internal) bus.
4. When the ALU is a destination, T1 is automatically loaded from the ALU output.
5. A transfer to AO activates the on-chip external bus controller.

Level 2 flowcharts

Housekeeping tasks merged with the operation tasks to form level 2 flowcharts.

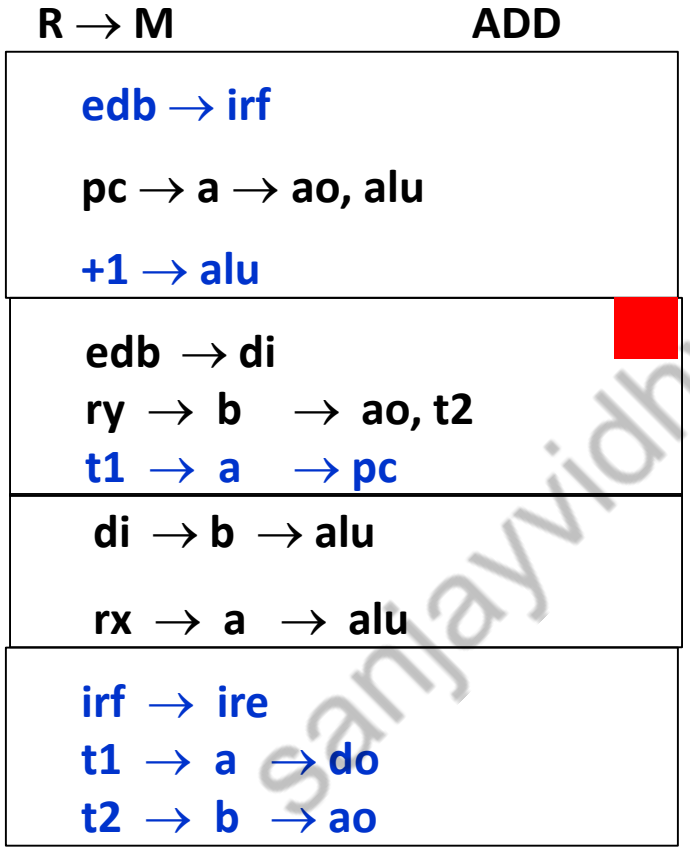
Level 2 flowcharts

Example 1: ADD RX AR RY



Level 2 flowcharts

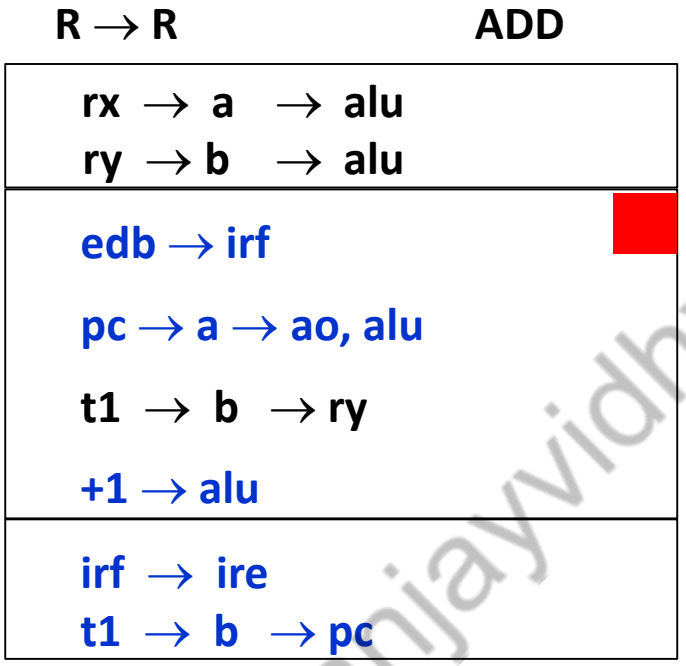
Example 2: ADD RX AR (RY)



Register T2 saves the operand address in the register-to memory Operation

Feedback on Execution Unit Design

Example 1: ADD RX AR RY



Less than full use of the A and B buses in the resulting sequence would signal the need to improve the execution unit.

Feedback on Execution Unit Design

Example 1: ADD RX AR RY

R → R	ADD
rx → a → alu ry → b → alu	
edb → irf pc → b → ao t1 → a → ry +1 → alu	
pc → b → alu +1 → alu	
irf → ire t1 → b → pc	

ADD sequence for an execution unit in which Output transferred to only one location

Additional One Cycle is required.

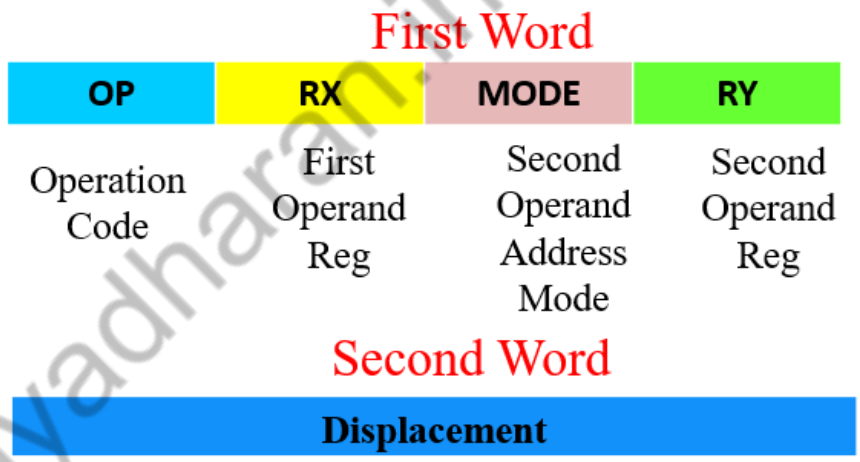
Feedback on Controller Design

Should we create level 1 flowcharts for the entire instruction set ?

Example 1: ADD RX AR RY

R → R ADD/SUB/AND/OR

rx → a → alu ry → b → alu
edb → irf pc → a → ao, alu t1 → b → ry +1 → alu
irf → ire t1 → b → pc

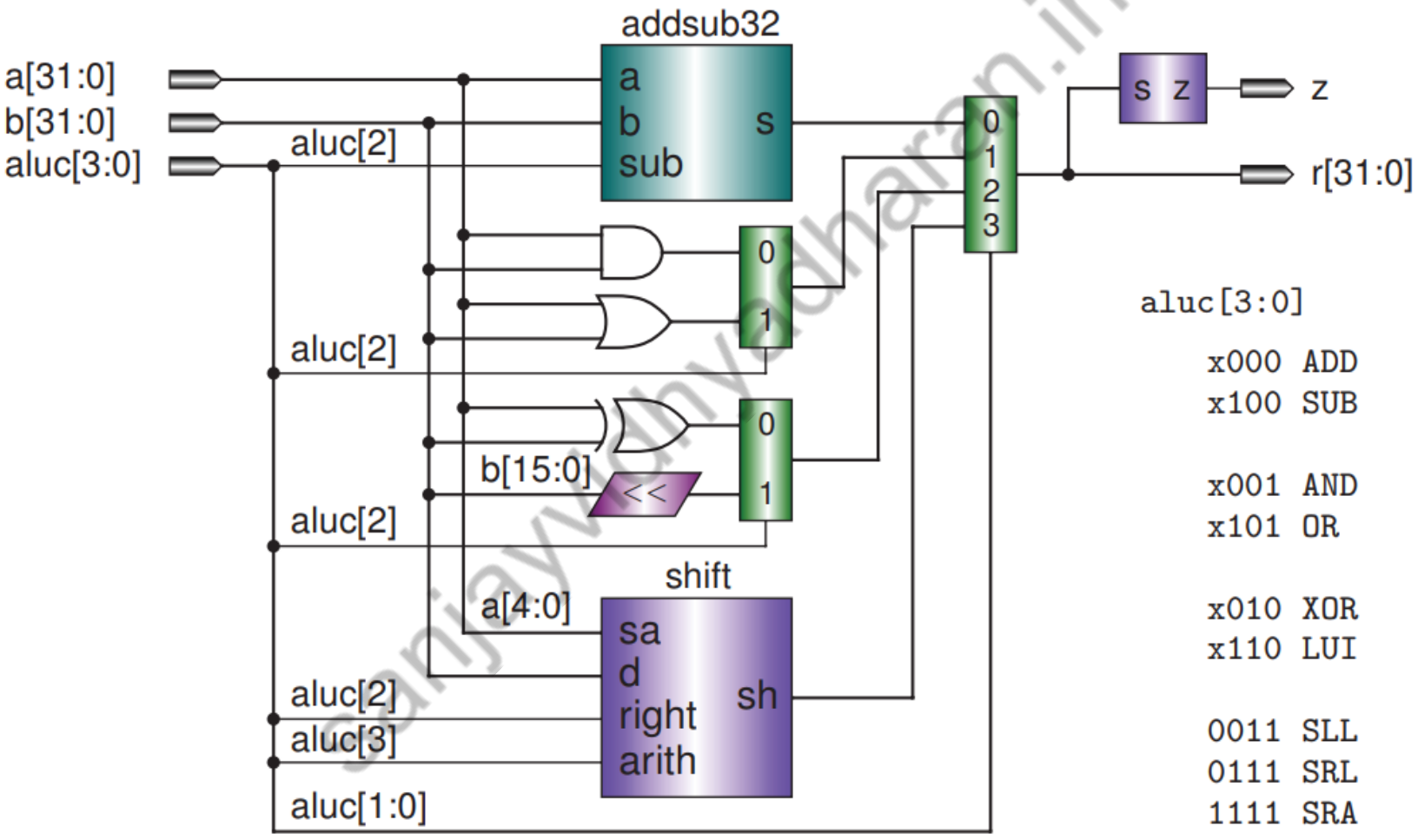


We write only one sequence for each instruction, independently of which registers are specified.

We need only decode the op code and the mode bits in the effective address field

The simple MIN CPU has *k* operations (ADD, AND, OR, SUB, and so on) and *a* address modes (Register Indirect, Base Plus Displacement, Indexed, and so on) . If any address mode is valid for any operation, I would need *k* a* instruction sequences.

ALU Design



Feedback on Controller Design

The combination of an address mode sequence and an execution sequence forms a control word sequence. If an instruction (such as a register-to-register ADD) does not need an address mode sequence, then the execution sequence and the control word sequence are the same.

Address Mode Sequences

Register Indirect

Example 2: ADD RX AR (RY)

edb → di
ry → b → ao, t2

Address Mode Sequences

Base Plus Displacement

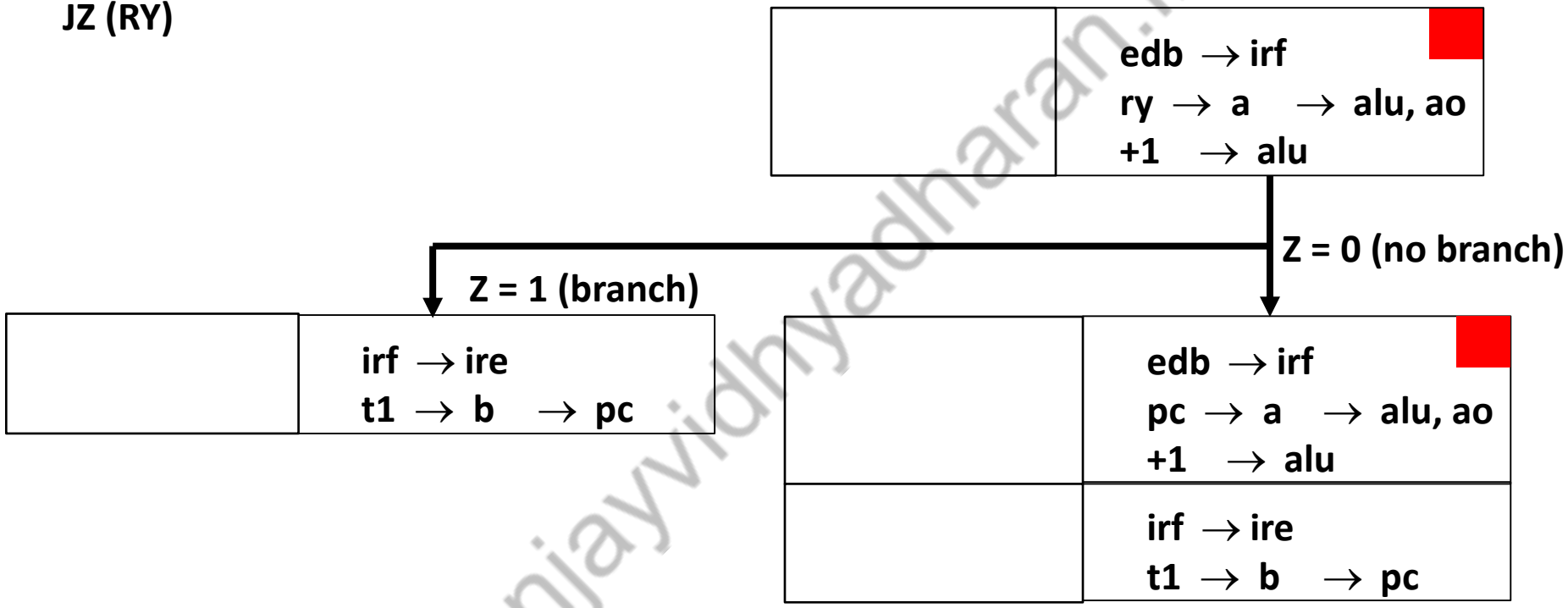
Example : ADD RX AR (RY+5H)

edb → di pc → a → ao, alu +1 → alu
t1 → a → ao
di → b → alu ry → a → alu
edb → di t1 → b → ao, t2

Branch Instruction

Z : Zero Flag

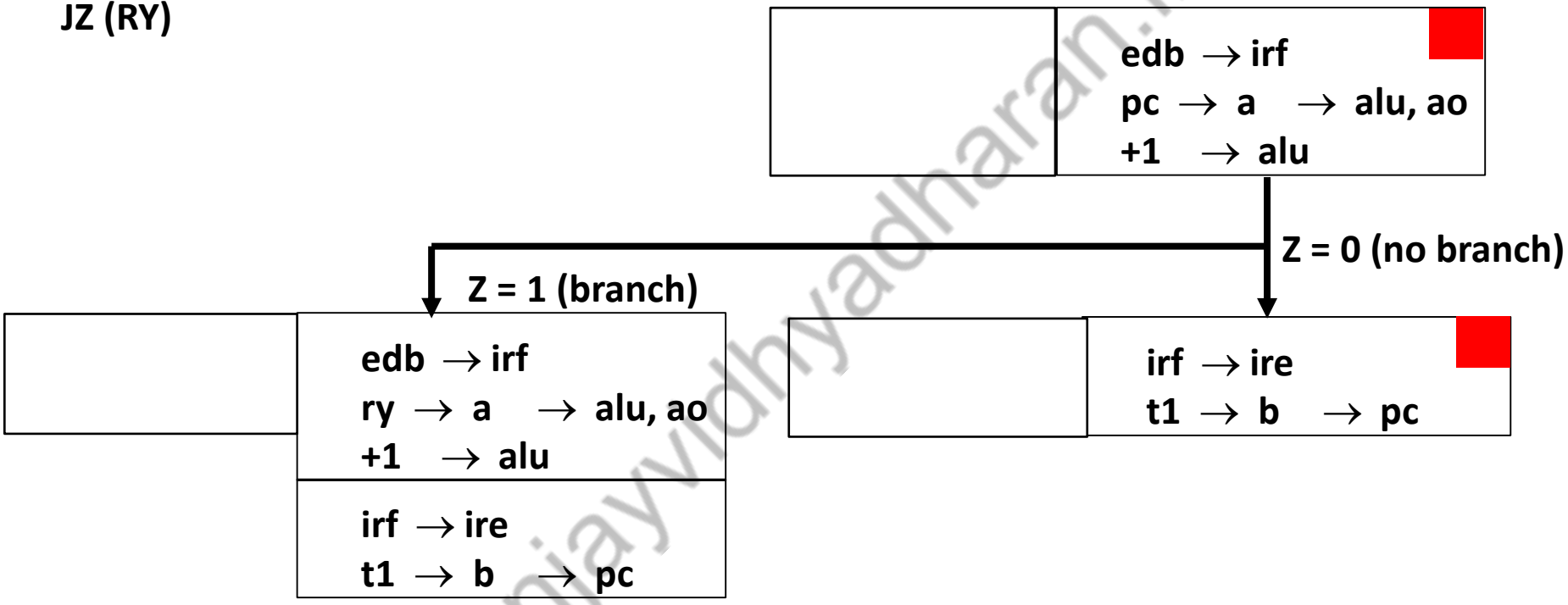
JZ (RY)



Branch Instruction

Z : Zero Flag

JZ (RY)



Rules for Doing Level 1 Flowcharts

1. At the beginning of instruction execution, IRE is assumed to contain the current instruction. It must be loaded by the previous instruction. Each instruction's control word sequence will, therefore, have to fetch the next instruction and load it into IRE.

2. Instruction execution begins with the address mode sequence (if the instruction has one) and implicitly branches to the appropriate execution sequence for completion.

3. The execution sequences for register-to-register instructions cannot be shared with execution sequences for memory reference instructions.

4. The execution sequences for standard dual operand instructions (ADD, AND, SUB, and so on) are identical except for the(implied) ALU function . They can use a common execution sequence if the op code directly specifies the ALU operation (the same way register fields select the registers).

Execution Sequences for Register-to-Register

LOAD

ry → a → alu, rx 0 → alu	edb → irf pc → a → alu, ao + 1 → alu
	irf → ire t1 → a → pc

STORE

rx → a → alu, ry 0 → alu	edb → irf pc → a → alu, ao + 1 → alu
	irf → ire t1 → a → pc

ADD

rx → a → alu ry → b → alu	edb → irf pc → a → alu, ao + 1 → alu
t1 → a → ry	irf → ire t1 → a → pc

SUB

rx → a → alu ry → b → alu	edb → irf pc → a → alu, ao + 1 → alu
t1 → a → ry	irf → ire t1 → a → pc

Execution Sequences with a Memory Operand Reference

ADD

di → b → alu rx → a → alu	edb → irf pc → a → alu, ao + 1 → alu
t1 → a → do t2 → b → ao	irf → ire t1 → b → pc

AND

di → b → alu rx → a → alu	edb → irf pc → a → alu, ao + 1 → alu
t1 → a → do t2 → b → ao	irf → ire t1 → b → pc

SUB

di → b → alu rx → a → alu	edb → irf pc → a → alu, ao + 1 → alu
t1 → a → do t2 → b → ao	irf → ire t1 → b → pc

Execution Sequences with a Memory Operand Reference

LOAD

di → b → rx, t2	edb → irf pc → a → alu, ao +1 → alu
t2 → a → alu 0 → alu	irf → ire t1 → b → p c

STORE

rx → a → alu, do t2 → b → a0 0 → alu	edb → irf pc → a → alu, ao +1 → alu
	irf → ire t1 → b → p c

Register T2 saves the operand address in the register to memory Operation

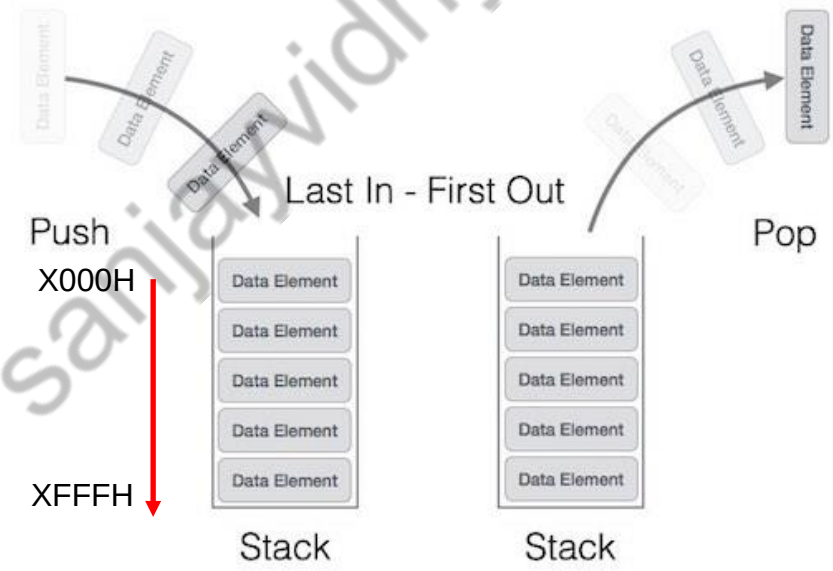
Execution Sequences for Special Instructions

POP

edb → di ry → a → alu, ao + 1 → alu	edb → irf pc → a → alu, ao + 1 → alu
di → b → rx t1 → a → ry	irf → ire t1 → a → pc

PUSH

ry → a → alu − 1 → alu	edb → irf pc → a → alu, ao + 1 → alu
rx → a → do t1 → b → ao, ry	irf → ire t1 → a → pc



Thank you