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 12 (09 February 2022)

CADSL

CISC

1. Limited memory
2. Slow memory
3. Programmability

⇓

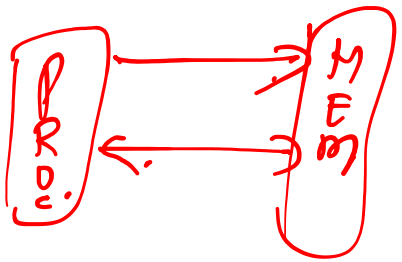
ASIP ✓

ML

Signal Processing (DSP) ✓

RISC
- Simple instructions
- Small no. of addressing modes
- Simple encoding.
  (Single instruction format)
  Fixed length encoding.

LS. ↑



1971 John Cocke: (IBM)

A/L

Memory

Register Addressing mode.

Constants: i++, i<1000

i==100

Immediate address mode.

- Base + Displacement
- Base + Index

Instructions — Simple, fixed length, 4B

Can facilitate to perform entire job one go

Processing of an instruction $S \xrightarrow{I} S'$

$S \xrightarrow{I} S'$

{ memory + Prog. Reg

S	S'
$a = 10$	$a = 10$
$b = 20$	$b = 20$
$r_1 = 100$	$r_1 = 100$
$r_2 = 50$	$r_2 = 50$
$r_3 = 70$	$r_3 = 150$
$r_3 = r_1 + r_2$	

One go

Single cycle

CPI = 1

Controller ↑ Combinational Logic

Time / Program = IC × CPI × τ =

IC × 1 × τ

↑ fix

3 GHz

τ = 330 ps .

Time / Prog ∝ IC .

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

Single Cycle implementation

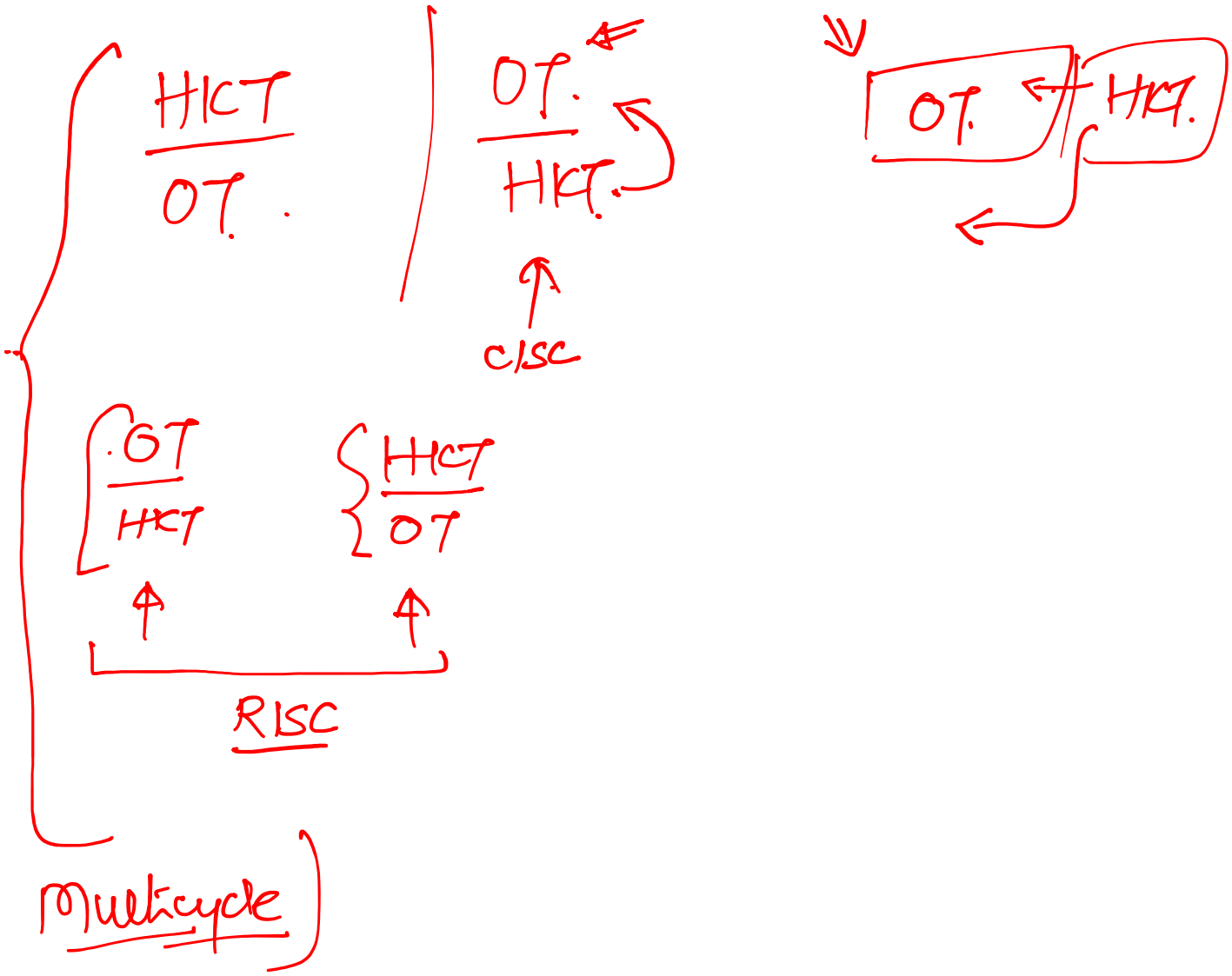
2 Memories → Program
→ Data

4. Executing

1. Fetch instruction
2. Update PC & understand instruction
3. Execute instruction
4. Update the System state
   - Memory
   - Programmer's register

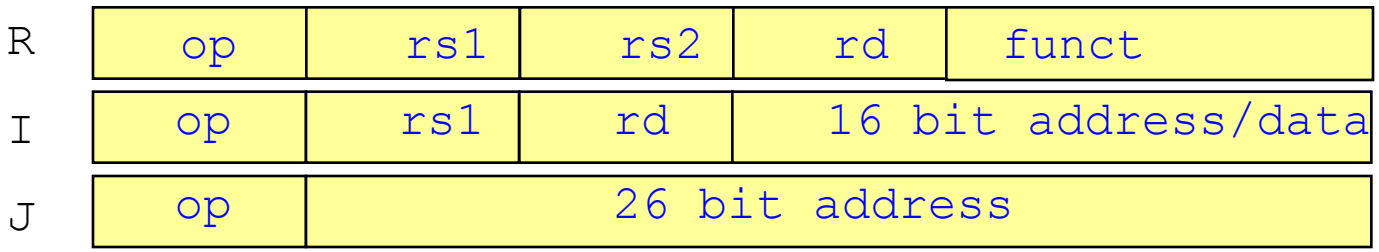
HKT

OT


HICT
OT.
OT.
HKT.
CISC
OT.
HKT.
OT
HKT
HICT
OT
RISC
Multicycle

Overview of DLX ISA

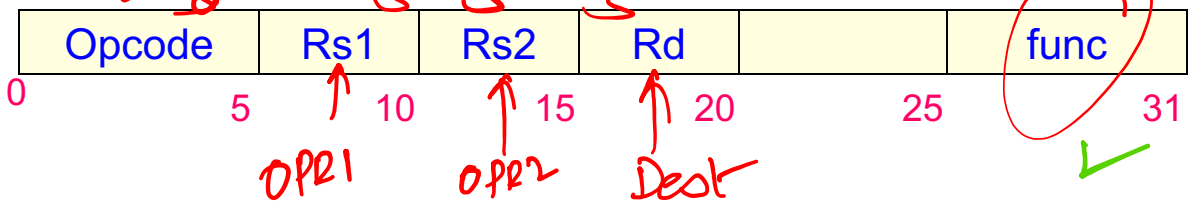
- simple instructions, all 32 bits wide
- very structured, no unnecessary baggage
- only three instruction formats



- rely on compiler to achieve performance

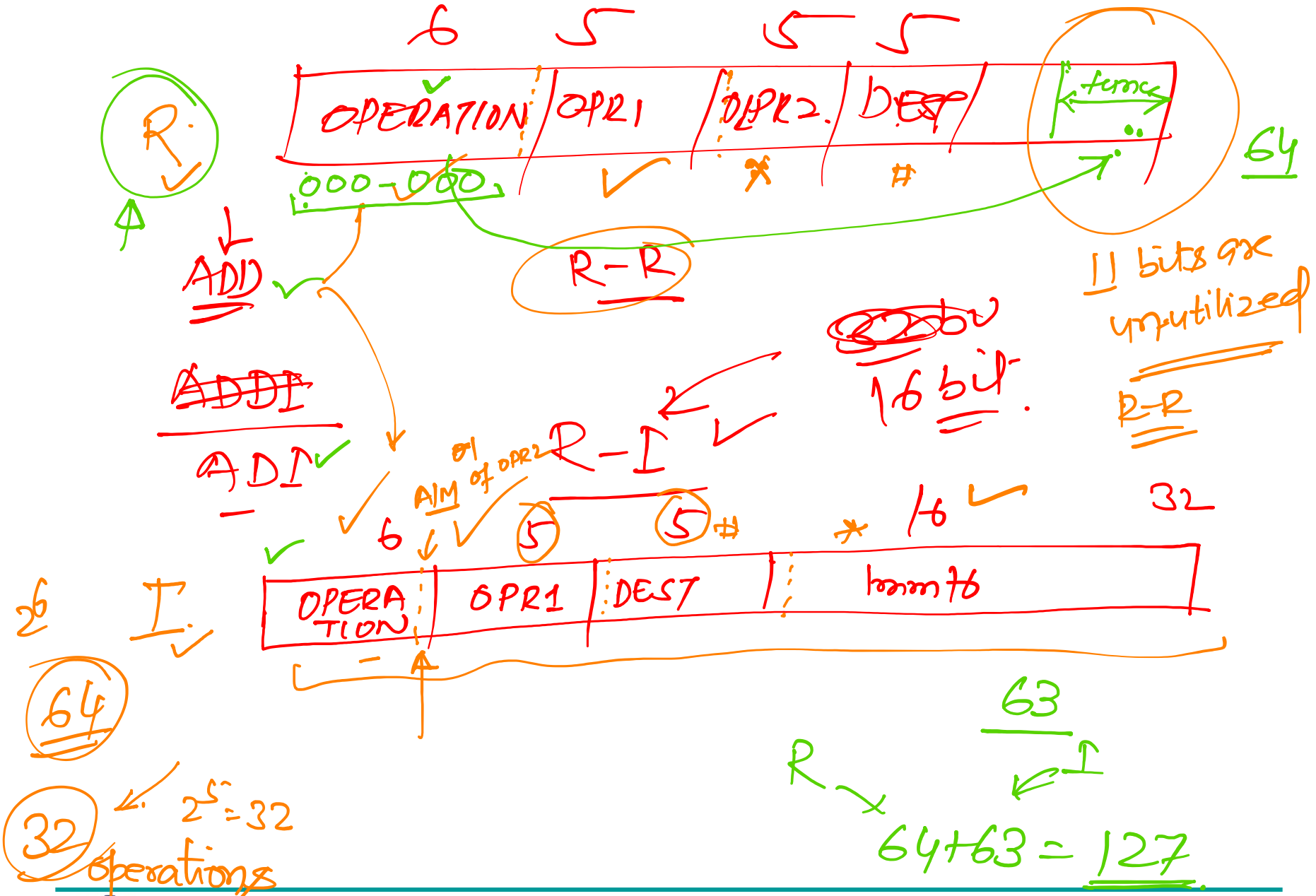
Instruction Set

Register-Register Instructions



Arithmetic and Logical Instruction

- ADD Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] + Reg[Rs2]
- SUB Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] - Reg[Rs2]
- AND Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] and Reg[Rs2]
- OR Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] or Reg[Rs2]
- XOR Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1] xor Reg[Rs2]
- SUB Rd, Rs1, Rs2 Regs[Rd] <= Reg[Rs1]-Reg[Rs2]


6
5
5
5
OPERATION
OPR1
OPR2
DEST
func
64
R
000-000
ADD
R-R
11 bits are unutilized
R-R
16 bit
ADDI
ADI
R-I
6
5
5
16
32
OPERATION
OPR1
DEST
immediate
26
I
64
32
operations
$2^5 = 32$
63
64+63 = 127

DLX Instruction Set

ADD Rd, Rs1, Rs2	Rd ← Rs1 + Rs2 (overflow – exception)	R	000_000 000_100
SUB Rd, Rs1, Rs2	Rd ← Rs1 - Rs2 (overflow – exception)	R	000_000 000_110
AND Rd, Rs1, Rs2	Rd ← Rs1 and Rs2	R	000_000/ 001_000
OR Rd, Rs1, Rs2	Rd ← Rs1 or Rs2	R	000_000/ 001_001
XOR Rd, Rs1, Rs2	Rd ← Rs1 xor Rs2	R	000_000/ 001_010
SLL Rd, Rs1, Rs2	Rd ← Rs1 << Rs2 (logical) (5 lsb of Rs2 are significant)	R	000_000 001_100
SRL Rd, Rs1, Rs2	Rd ← Rs1 >> Rs2 (logical) (5 lsb of Rs2 are significant)	R	000_000 001_110
SRA Rd, Rs1, Rs2	Rd ← Rs1 >> Rs2 (arithmetic) (5 lsb of Rs2 are significant)	R	000_000 001_111

DLX Instruction Set

			OPERATION
ADDI Rd, Rs1, Imm	Rd ← Rs1 + Imm (sign extended) (overflow – exception)	I	010_100
SUBI Rd, Rs1, Imm	Rd ← Rs1 – Imm (sign extended) (overflow – exception)	I	010_110
ANDI Rd, Rs1, Imm	Rd ← Rs1 and Imm (zero extended)	I	011_000
ORI Rd, Rs1, Imm	Rd ← Rs1 or Imm(zero extended)	I	011_001
XORI Rd, Rs1, Imm	Rd ← Rs1 xor Imm(zero extended)	I	011_010
SLLI Rd, Rs1, Imm	Rd ← Rs1 << Imm (logical) (5 lsb of Imm are significant)	I	011_100
SRLI Rd, Rs1, Imm	Rd ← Rs1 >> Imm (logical) (5 lsb of Imm are significant)	I	011_110
SRAI Rd, Rs1, Imm	Rd ← Rs1 >> Imm (arithmetic) (5 lsb of Imm are significant)	I	011_111

DLX Instruction Set

LHI Rd, Imm	Rd(0:15) ← Imm Rd(16:32) ← hex0000 (Imm: 16 bit immediate)	I	011_011
NOP	Do nothing	R	000_000 000_000

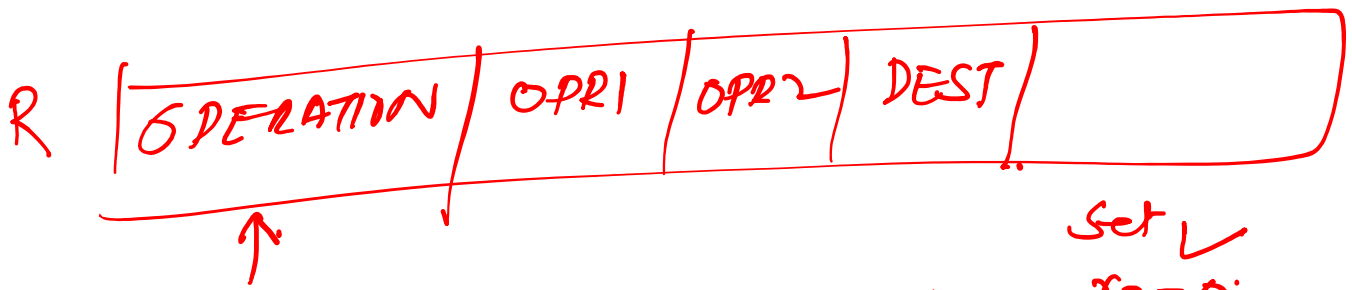
Condition

CC Register

Condition code / Status flag

==
<
>
<=
>=
≠

Without CC



$(r_1 == r_2)$ then $r_3 = 0$ else $r_3 = 1$.

DLX Instruction Set

SEQ R3, R1, R2 (R1 == R2) then R3 = 1 else R3 = 0

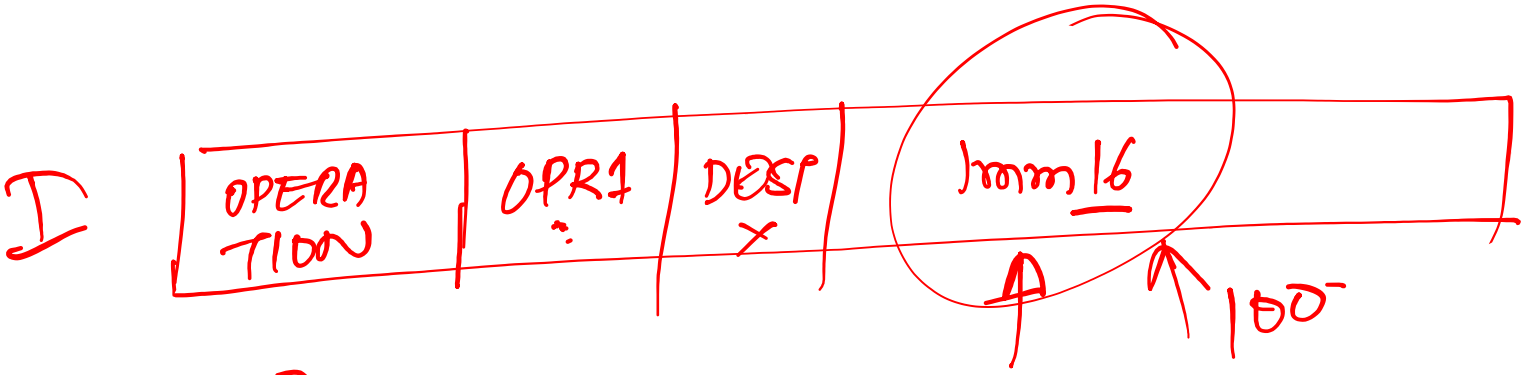
SEQ Rd, Rs1, Rs2	Rs1 = Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_000
SNE Rd, Rs1, Rs2	Rs1 /= Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_010
SLT Rd, Rs1, Rs2	Rs1 < Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_100
SLE Rd, Rs1, Rs2	Rs1 <= Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 010_110
SGT Rd, Rs1, Rs2	Rs1 > Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 011_000
SGE Rd, Rs1, Rs2	Rs1 >= Rs2: Rd ← hex0000_0001 else: Rd ← hex0000_0000	R	000_000 011_010

DLX Instruction Set

SEQI Rd, Rs1, Imm	Rs1 = Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000 (Imm: Sign extended 16 bit immediate)	I	100_000
SNEI Rd, Rs1, Imm	Rs1 /= Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	100_010
SLTI Rd, Rs1, Imm	Rs1 < Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	100_100
SLEI Rd, Rs1, Imm	Rs1 <= Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	100_110
SGTI Rd, Rs1, Imm	Rs1 > Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	101_000
SGEI Rd, Rs1, Imm	Rs1 >= Imm : Rd ← hex0000_0001 else: Rd ← hex0000_0000	I	101_010

DLX Instruction Set

BEQZ Rs, Label	Rs = 0: PC ← PC+4+Label Rs /= 0: PC ← PC+4 (Label: Sign extended16 bit immediate)	I	010_000
BNEZ Rs, Label	Rs /= 0: PC ← PC+4+Label Rs = 0: PC ← PC+4	I	010_001
J Label	PC ← PC + 4 + sign_extd(imm26)	J	001_100
JAL Label	R31 ← PC + 4 PC ← PC+ 4 + sign_extd(imm26)	J	001_100
JAL Label	R31 ← PC + 4 PC ← PC+ 4 + sign_extd(imm26)	J	001_101
JR Rs	PC ← Rs	I	001_110
JALR Rs	R31 ← PC + 4 PC ← Rs	I	001_111

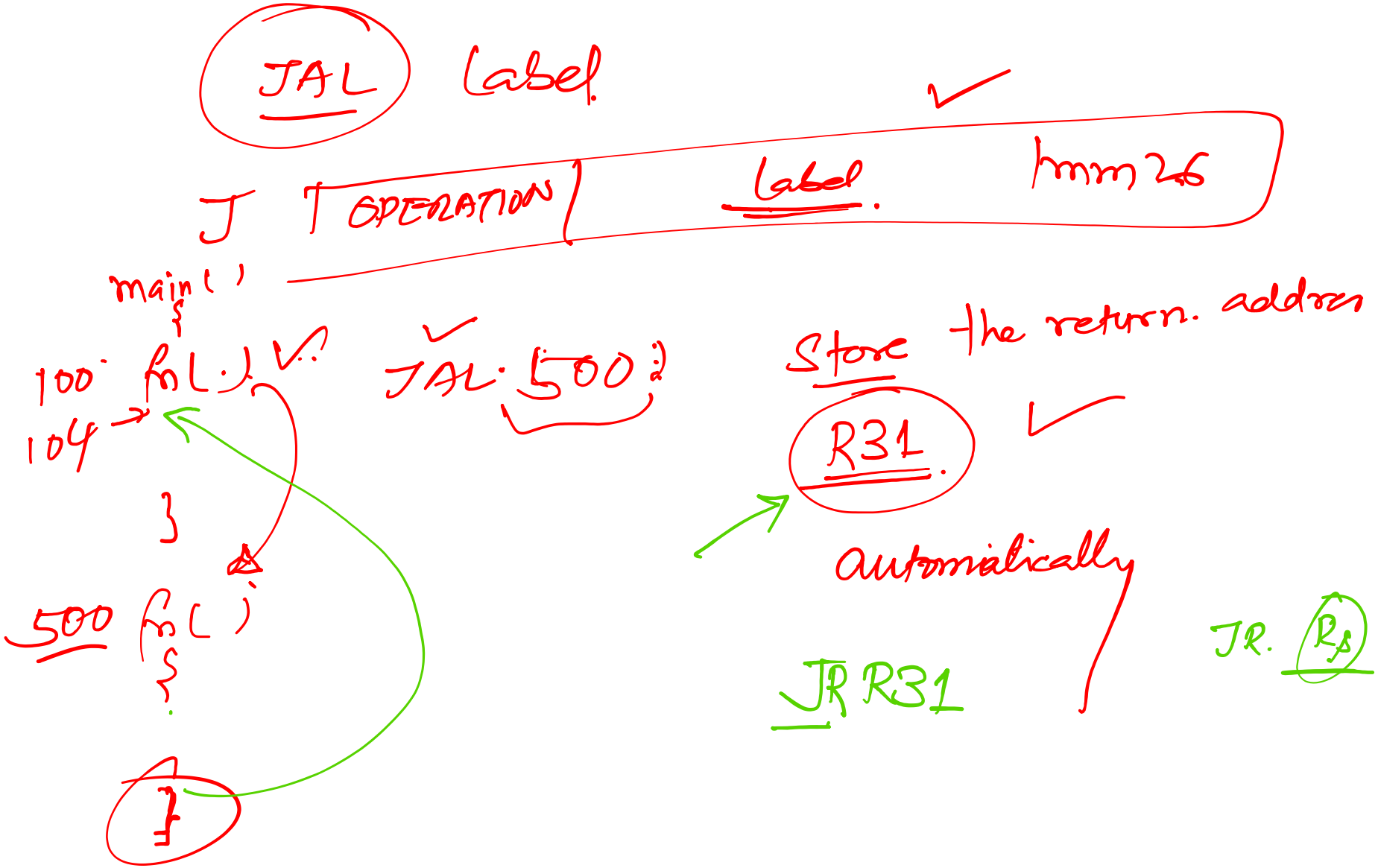


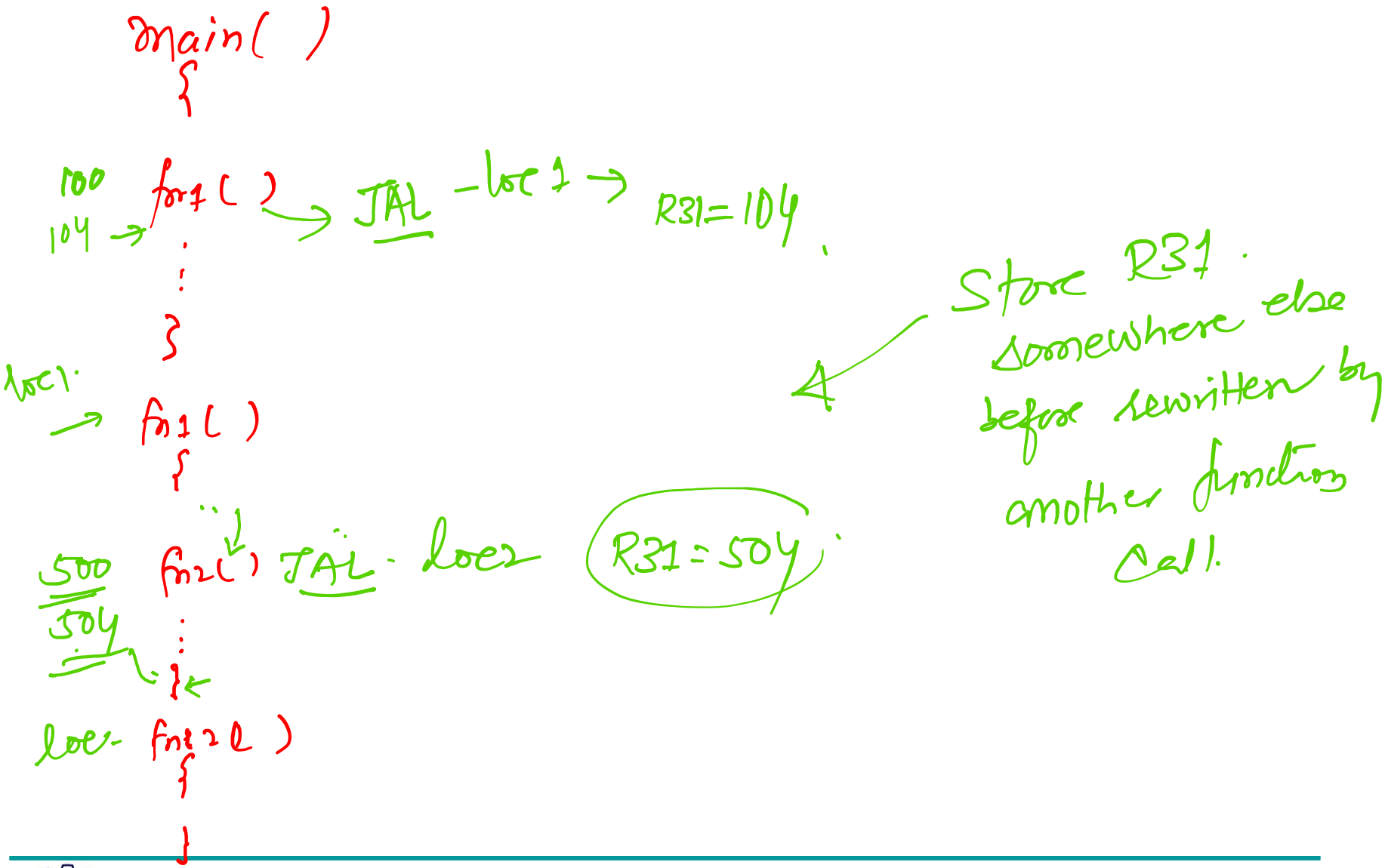
if OPR == 0 → PC = PC+4 + 100 X 4

w.r.t to next instruction

else

PC = PC+4 .


JAL
label
J / OPERATION
label
imm26
main()
{
100 fn();
104
}
500 fn()
{
}
JAL 500;
Store the return address
R31
automatically
JR R31
JR Rs


main( )
{
100
104
fn1( ) → JAL - loc1 → R31 = 104.
}
loc1.
fn1( )
{
fn2( ) JAL - loc2 R31 = 504.
500
504
loc2 - fn2( )
{
}
Store R31 somewhere else before rewritten by another function call.

DLX Instruction Set

addressing mode → B+D ← I
B+D ← R

LW Rd, Rs2 (Rs1)	Rd ← M(Rs1 + Rs2) (word aligned address)	R	000_000 100_000
SW Rs2(Rs1), Rd	M(Rs1 + Rs2) ← Rd	R	000_000 101_000
LH Rd, Rs2 (Rs1)	Rd (16:31)← M(Rs1 + Rs2) (Rd sign extended to 32 bit)	R	000_000 100_001
SH Rs2(Rs1), Rd	M(Rs1 + Rs2) ← Rd(16:31)	R	000_000 101_001
LB Rd, Rs2 (Rs1)	Rd (24:31)← M(Rs1 + Rs2) (Rd sign extended to 32 bit)	R	000_000 101_010
SB Rs2(Rs1), Rd	M(Rs1 + Rs2) ← Rd(24:31)	R	000_000 101_010

DLX Instruction Set

LWI Rd, Imm (Rs)	Rd ← M(Rs + Imm) (Imm: sign extended 16 bit) (word aligned address)	I	000_100
SWI Imm(Rs), Rd	M(Rs + Imm) ← Rd	I	001_000
LHI Rd, Imm (Rs)	Rd (16:31)← M(Rs + Imm) (Rd sign extended to 32 bit)	I	000_101
SHI Imm(Rs), Rd	M(Rs1 + Rs2) ← Rd(16:31)	I	001_001
LBI Rd, Imm (Rs)	Rd (24:31)← M(Rs + Imm) (Rd sign extended to 32 bit)	I	000_110
SBI Imm(Rs), Rd	M(Rs + Imm) ← Rd(24:31)	I	001_010

R0 — R31 ← return address (Link register)

JAL ϕ —

i > 0

for (i = n; i > 0; i--)

i == 0

→ [Check against zero]

R0 = 0 — Read only

$R_2 \leftarrow R_2$

Add R_2, R_1, R_0

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

Thank You