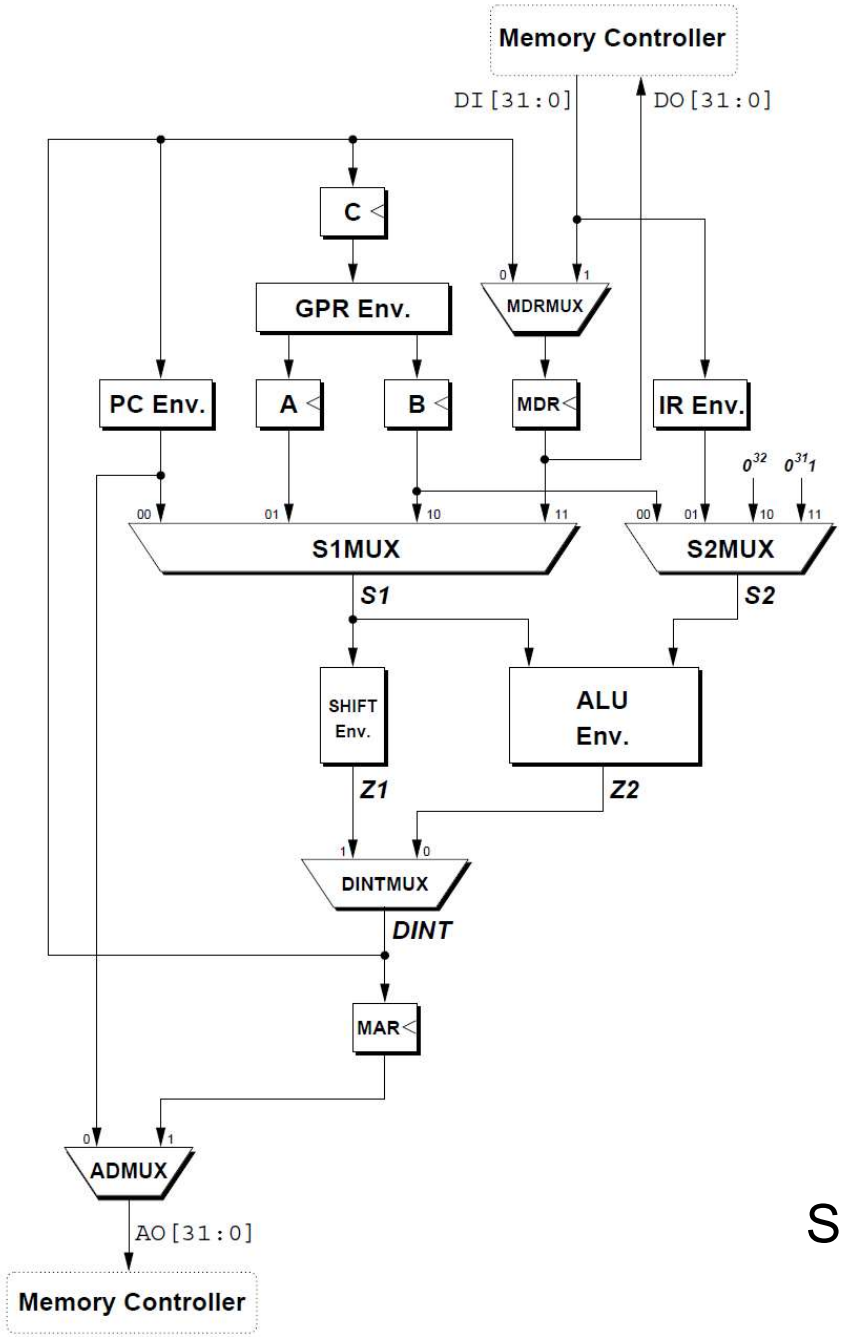


A Simplified DLX: Implementation

Guy Even and Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

June 11, 2019

The World of Computers

SW Programmer

Software

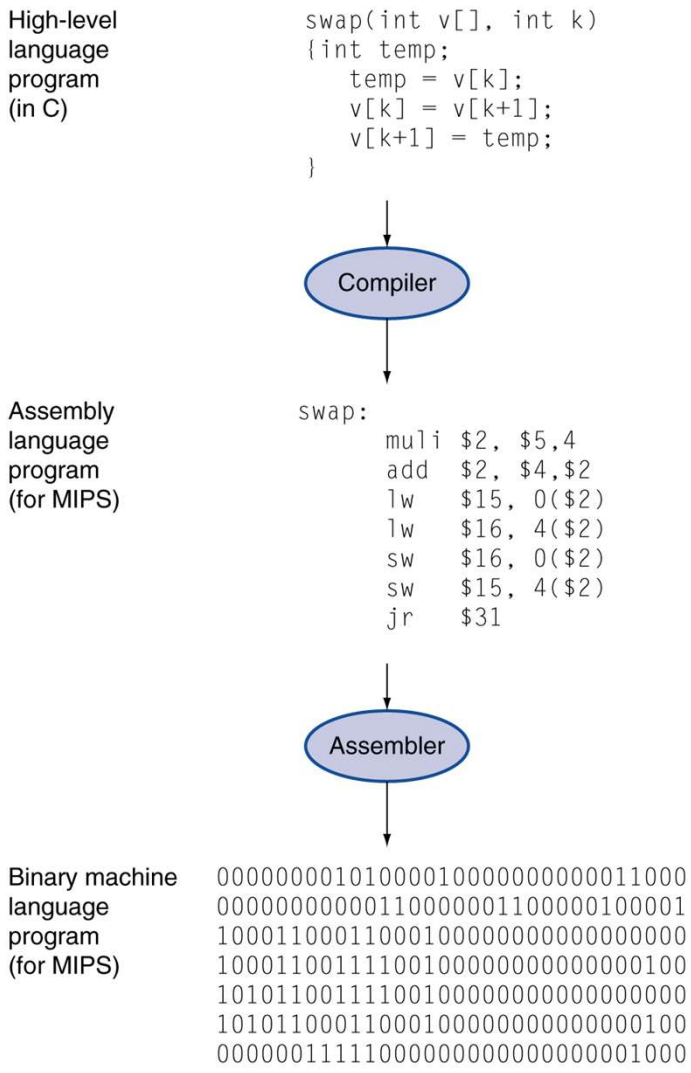
Architect

HW Designer

Hardware

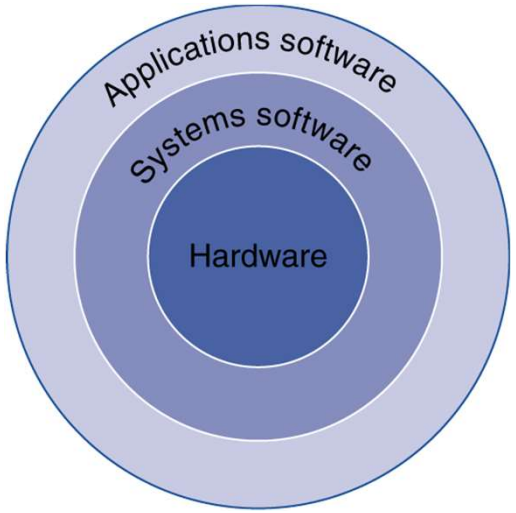
Reminder - Software

- High-level language
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- Assembly language
  - Textual representation of instructions
- Hardware representation
  - Binary digits (bits)
  - Encoded instructions and data



Below Your Program

- Application software
  - Written in high-level language
- System software
  - Compiler: translates HLL code to machine code
  - Operating System: service code
    - Handling input/output
    - Managing memory and storage
    - Scheduling tasks & sharing resources
- Hardware
  - Processor, memory, I/O controllers

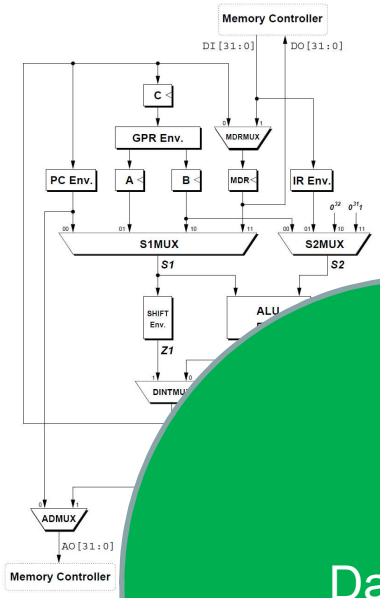


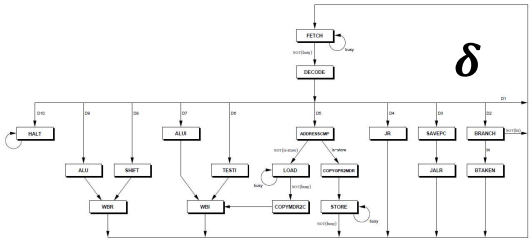
Today

Implementation

DLX

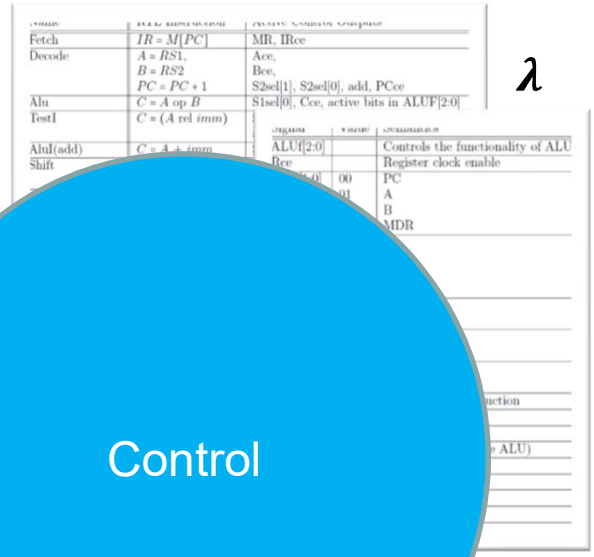
DLX Hardware





δ

λ



Control Signals
Selectors
CEs
ALUF
Add
Test
Shift, Right
Itype, JLINK

Datapath

Control

AEQZ
Inst

Data

Address

MW,MR

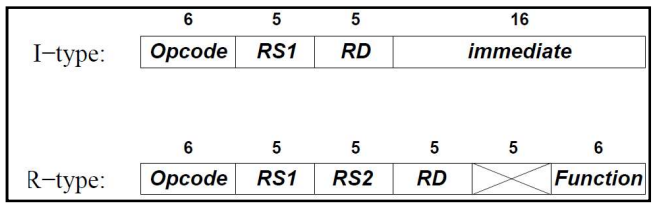
Busy

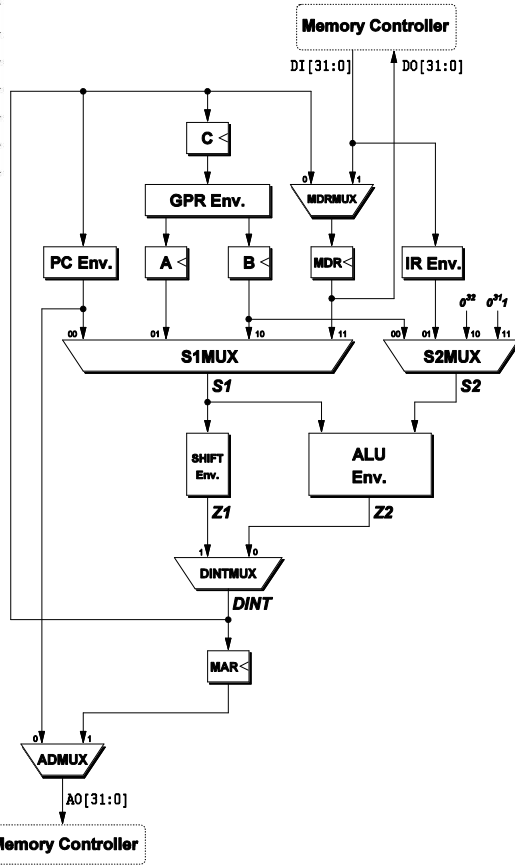
Memory

Name	RTL Instruction	Active Control Outputs
Fetch	$IR = M[PC]$	MR, IRce
Decode	$A = RS1$, $B = RS2$ $PC = PC + 1$	Ace, Bce, S2sel[1], S2sel[0], add, PCce
Alu	$C = A$ op B	S1sel[0], Cce, active bits in ALUF[2:0]
TestI	$C = (A$ rel $imm)$	S1sel[0], S2sel[0], Cce, test, Itype, active bits in ALUF[2:0]
AluI(add)	$C = A + imm$	S1sel[0], S2sel[0], Cce, add, Itype
Shift	$C = A$ shift sa $sa = 1, (-1)$	S1sel[0], Cce DINTsel, shift (,right)
Adr.Comp	$MAR = A + imm$	S1sel[0], S2sel[0], MARce, add
Load	MDR = M[MAR]	MDRce, ADsel, MR, MDRsel
Store	M[MAR] = MDR	ADsel, MW
CopyMDR2C	C = MDR($\gg 0$)	S1sel[0], S1sel[1], S2sel[1], DINTsel, Cce
CopyGPR2MDR	MDR = B($\ll 0$)	S1sel[1], S2sel[1], DINTsel, MDRce
WBR	$RD = C$ (R-type)	GPR_WE
WBI	$RD = C$ (I-type)	GPR_WE, Itype
Branch	branch taken?	
Btaken	$PC = PC + imm$	S2sel[0], add, PCce
JR	$PC = A$	S1sel[0], S2sel[1], add, PCce
Save PC	$C = PC$	S2sel[1], add, Cce
JALR	$PC = A$ $R31 = C$	S1sel[0], S2sel[1], add, PCce, GPR_WE, jlink

Signal	Value	Semantics
ALUf[2:0]		Controls the functionality of ALU
Rce		Register clock enable
S1sel[1:0]	00	PC
	01	A
	10	B
	11	MDR
S2sel[1:0]	00	B
	01	IR
	10	0
	11	1
DINTsel	0	ALU
	1	Shifter
MDRsel	0	DINT
	1	DI
ADsel	0	PC
	1	MAR
shift		explicit Shift-Instruction
right		Shift to the right
add		Forces an addition
test		Forces a test (in the ALU)
MR		Memory Read
MW		Memory Write
GPR_WE		GPR write enable
itype		Itype-Instruction
jlink		jump and link

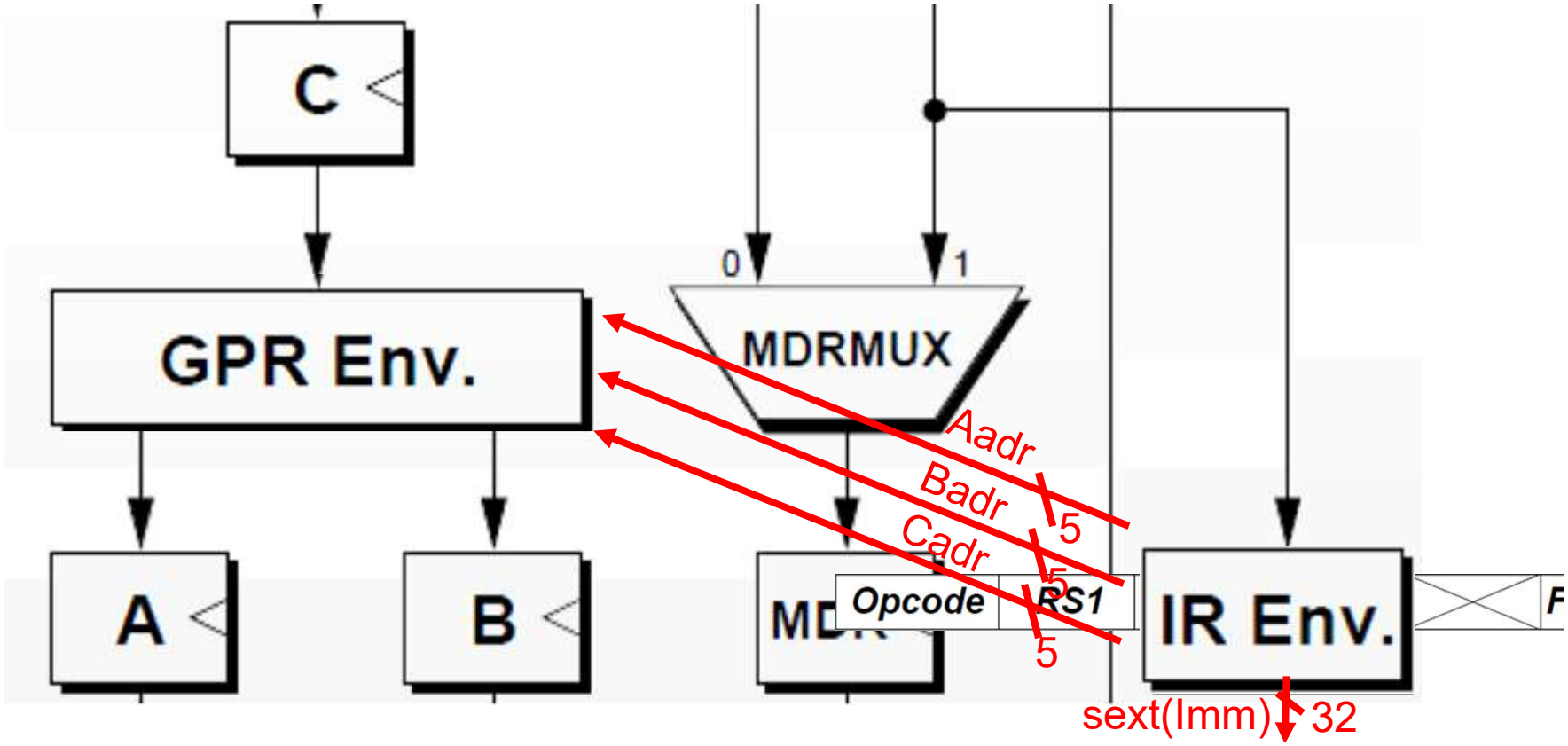
condition	when does it equal 1?
D1	special NOP
D2	beqz, bnez
D3	jalr
D4	jr
D5	lw, sw
D6	sgti, seqi, sgei, slti, snei, slei
D7	addi
D8	sll, srl
D9	add, sub, and, or, xor
D10	halt





GPR is addressed by IR

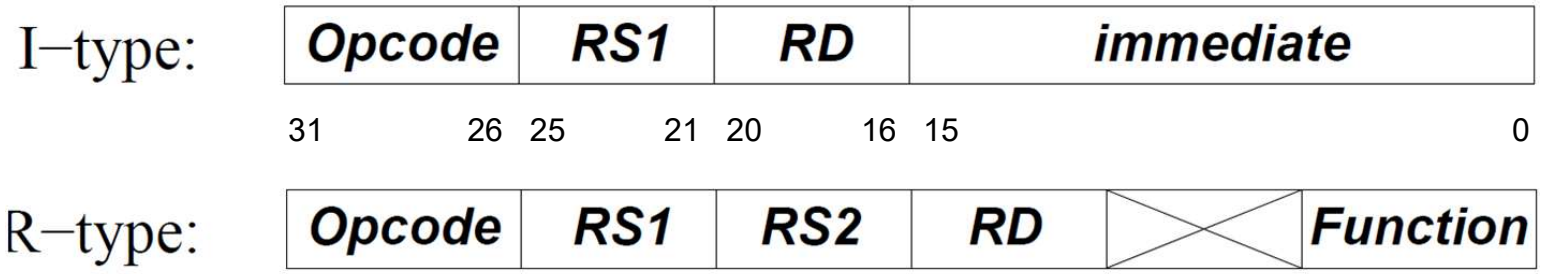
- The IR environment obtains the instruction and asserts the *Aadr*, *Badr*, *Cadr* bits to GPR.



GPR is addressed by IR

- The output products of the IR:

$$\texttt{Imm}[31:0](t) = \textit{sign extension of } \texttt{Inst}[15:0](t) \textit{ to 32 bits.}$$
$$\texttt{Aadr}[4:0](t) = \texttt{Inst}[25:21](t),$$
$$\texttt{Badr}[4:0](t) = \texttt{Inst}[20:16](t),$$
$$\texttt{Cadr}[4:0](t) = \begin{cases} 11111 & \textit{if } \texttt{JLINK}(t) = 1, \\ \texttt{Inst}[20:16](t), & \textit{if } \texttt{Itype}(t) = 1 \textit{ and } \texttt{JLINK}(t) = 0, \\ \texttt{Inst}[15:11](t), & \textit{otherwise.} \end{cases}$$



Practice the instruction execution!

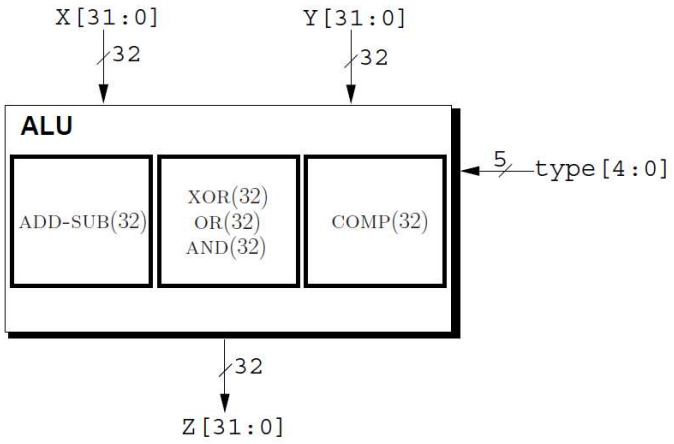
For every instruction in the ISA go over its control state path. For each control state – draw on the Datapath the flow of the relevant information.

Tips:

1. Take the Datapath diagram, put it in a transparent nylon bag – easier to draw and erase!
2. no need to simulate all the s*rel*i, just take one representative.
3. no need to simulate all the add,sub,xor,or,and – take one representative.

Executing Instructions - add

Instruction	Semantics
add RD RS1 RS2	RD := RS1 + RS2



$type[4:2]$	$type[1]$	$type[0]$	$f_{type}(\vec{x},\vec{y})$
001	1	0	$[\vec{x}] > [\vec{y}]$
010	0	0	$[\vec{x}] - [\vec{y}] \pmod{2^{32}}$
010	1	0	$[\vec{x}] = [\vec{y}]$
011	0	0	$[\vec{x}] + [\vec{y}] \pmod{2^{32}}$
011	1	0	$[\vec{x}] \geq [\vec{y}]$
100	0	0	$\mathrm{XOR}(\vec{x},\vec{y})$
100	1	0	$[\vec{x}] < [\vec{y}]$
101	0	0	$\mathrm{OR}(\vec{x},\vec{y})$
101	1	0	$[\vec{x}] \neq [\vec{y}]$
110	0	0	$\mathrm{AND}(\vec{x},\vec{y})$
110	1	0	$[\vec{x}] \leq [\vec{y}]$
***	*	1	$[\vec{x}] + [\vec{y}] \pmod{2^{32}}$

ALUF[2:0]

IR[5 : 0]	Mnemonic	Semantics
100 011	add	RD = RS1 + RS2

Executing Instructions – beqz, jalr

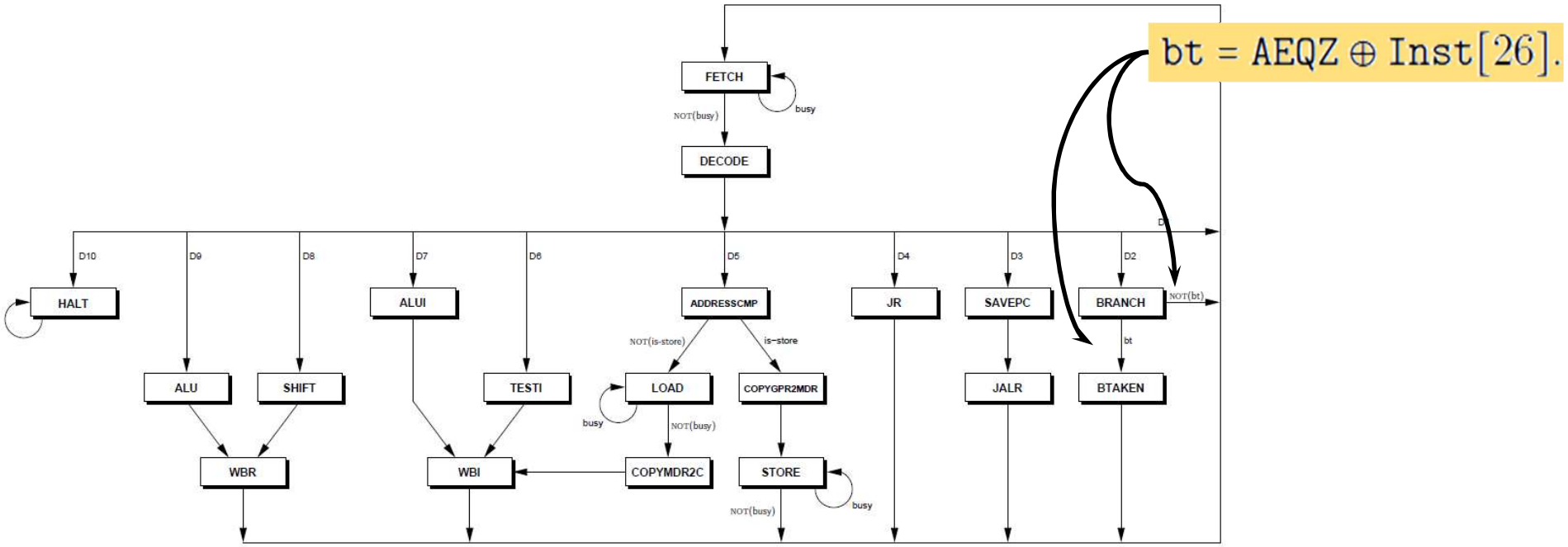
Instruction	Semantics
beqz RS1 imm	PC = PC + 1 + sext(imm), if RS1 = 0 PC = PC + 1, if RS1 ≠ 0
bnez RS1 imm	PC = PC + 1, if RS1 = 0 PC = PC + 1 + sext(imm), if RS1 ≠ 0
jr RS1	PC = RS1
jalr RS1	R31 = PC+1; PC = RS1

Inst[26]

000 100 || beqz

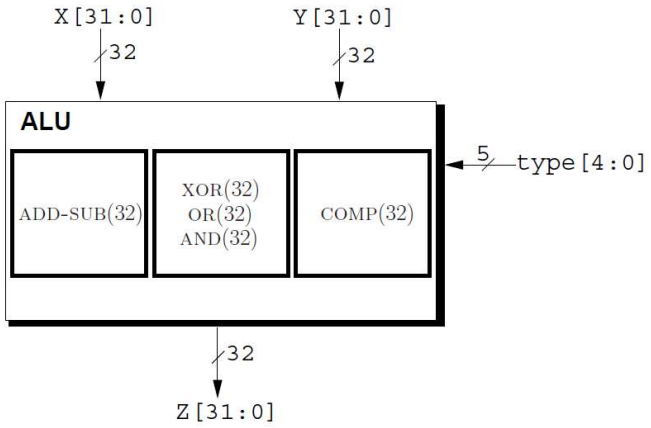
000 101 || bnez

$bt = AEQZ \oplus Inst[26].$



Executing Instructions - slti

Instruction	Semantics
`s`*rel*`i RD RS1 imm`	RD := 1, if condition is satisfied,
	RD := 0 otherwise
if *rel* =`lt`	test if RS1 $<$ sext(imm)



$type[4:2]$	$type[1]$	$type[0]$	$f_{type}(\vec{x},\vec{y})$
001	1	0	$[\vec{x}] > [\vec{y}]$
010	0	0	$[\vec{x}] - [\vec{y}] \pmod{2^{32}}$
010	1	0	$[\vec{x}] = [\vec{y}]$
011	0	0	$[\vec{x}] + [\vec{y}] \pmod{2^{32}}$
011	1	0	$[\vec{x}] \geq [\vec{y}]$
100	0	0	$\mathrm{XOR}(\vec{x},\vec{y})$
100	1	0	$[\vec{x}] < [\vec{y}]$
101	0	0	$\mathrm{OR}(\vec{x},\vec{y})$
101	1	0	$[\vec{x}] \neq [\vec{y}]$
110	0	0	$\mathrm{AND}(\vec{x},\vec{y})$
110	1	0	$[\vec{x}] \leq [\vec{y}]$
***	*	1	$[\vec{x}] + [\vec{y}] \pmod{2^{32}}$

IR[31 : 26]	Mnemonic	Semantics
011 100	slti	RD = (RS1 $<$ sext(imm))

Executing Instructions - sw

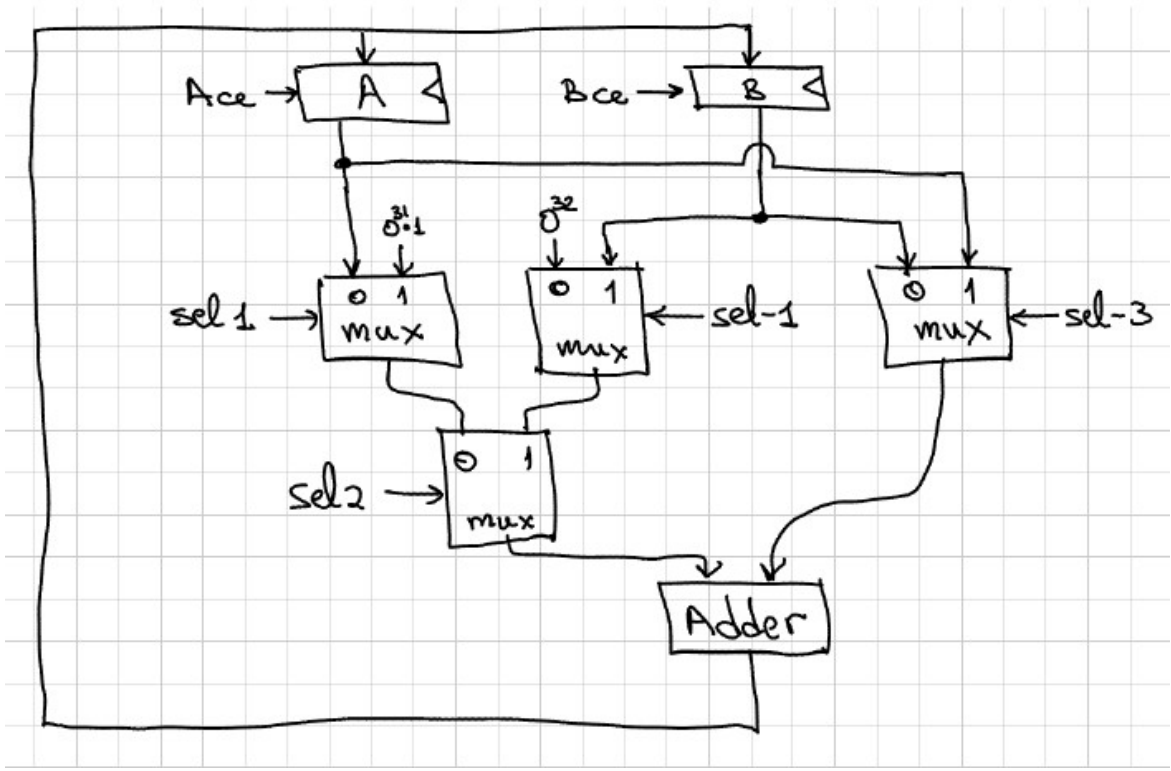
Load/Store	Semantics
lw RD RS1 imm	RD := M[sext(imm)+RS1]
sw RD RS1 imm	M[sext(imm)+RS1] := RD

Using a simplified datapath

EXECUTING RTL INSTRUCTIONS

Executing RTL Instructions 1

- $A \leftarrow B + A$
- $A \leftarrow B + A + 1$



Executing RTL Instructions 2

- $B \leftarrow A$
- $A \leftarrow 1$
- $A \leftarrow 2(A + B)$