

Digital Logic Design: a rigorous approach ©

Chapter 22: A Simplified DLX: Implementation

Guy Even Moti Medina

School of Electrical Engineering Tel-Aviv Univ.

June 16, 2020

Book Homepage:

<http://www.eng.tau.ac.il/~guy/Even-Medina>

Implementation of the Simplified DLX

- ① Goal: design a circuit that can execute any DLX program stored in memory.
- ② This circuit is a **stored program computer** also known as a computer with a **von Neumann architecture** based on von Neumann's paper from 1945.
- ③ A practical computer based on Turing's idea of a **universal Turing machine**.
- ④ First stored program computers built in 1948-1949 (SSEM, Manchester Mark 1, EDSAC)

Datapath and Control

The implementation consists of two parts:

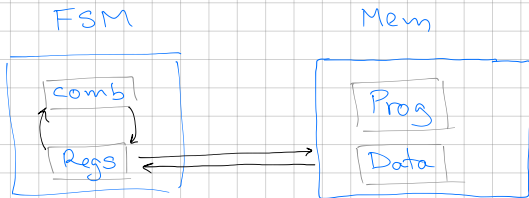
- a finite state machine, called the **control**, and
- a circuit containing registers and functional modules, called the **datapath**.

The separation of the design into a controller and a datapath greatly simplifies the task of designing the simplified DLX.

Design Principle

Execution of an instruction takes many clock cycles!

DLX implementation: 1st attempt



Problem:

$$|\text{states FSM}| = 2^{|\text{FF's}|}$$

we have 32 GPR's, MAR, MDR, IR, PC

⇒ too many states!

Goal: $|\text{states}| \approx 20$

be able to prove correctness

All the registers of the simplified DLX datapath are 32-bits wide, and are as follows.

- 1 There are 32 **General Purpose Registers (GPR)**: R0 to R31.
- 2 The **Instruction Register (IR)** is, also, a clock enabled parallel load register. This register is part of the **IR environment**.
- 3 The remaining registers: **Program Counter (PC)**, **Memory Address Register (MAR)**, **Memory Data Register (MDR)**, and registers *A*, *B* and *C* are all clock enabled parallel load registers. Each of these registers has a distinct clock enable signal that is computed by an FSM called the DLX control . The clock enable signals are called PCCE, MARCE, MDRCE, ACE, BCE, CCE.

The Outside World: The Memory Controller

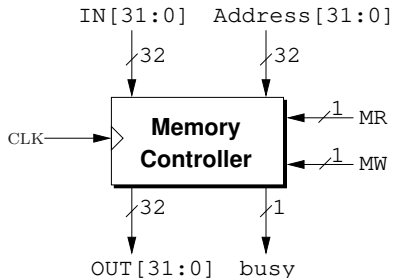
We begin with the “outside world”, that is the (external) memory. Recall that both the executed program and the data are stored in the memory.

- The **memory controller** is a circuit that is positioned between the DLX and the main memory.
- It is a synchronous circuit that receives memory access requests from the DLX.
- The main problem related to a memory access is that it requires an unpredictable number of cycles to complete.

Memory Controller - Discussion

- Accessing a register always takes a single clock cycle, however, loading or storing in the external memory typically requires several cycles.
- Why? Organization of the memory, also called the **memory hierarchy**. This organization involves caches, cache misses, page faults, and other issues that are beyond the scope of this course.
- The fact that the number of clock cycles required to complete a memory access is not fixed requires a special signal, called the **busy signal**.
- The busy signal is an output of the memory controller that tells the DLX whether the memory is still executing the previous memory access.
- The DLX may issue a new memory access request only if the busy signal is low.

Memory Controller - Schematic



The busses are connected to the memory controller as follows.

- The bus $AO[31 : 0]$ is connected to the $Address[31 : 0]$ input of the memory controller.
- The bus $DO[31 : 0]$ is connected to the $IN[31 : 0]$ input of the memory controller.
- The bus $DI[31 : 0]$ is connected to the $OUT[31 : 0]$ input of the memory controller.

The Memory Controller: Definition

Definition

The **Memory Controller** is a synchronous circuit specified as follows:

Input: $IN[31 : 0]$, $Address[31 : 0] \in \{0, 1\}^{32}$, $MR, MW \in \{0, 1\}$,
and a clock CLK .

Output: $OUT[31 : 0] \in \{0, 1\}^{32}$, $busy \in \{0, 1\}$.

Functionality:

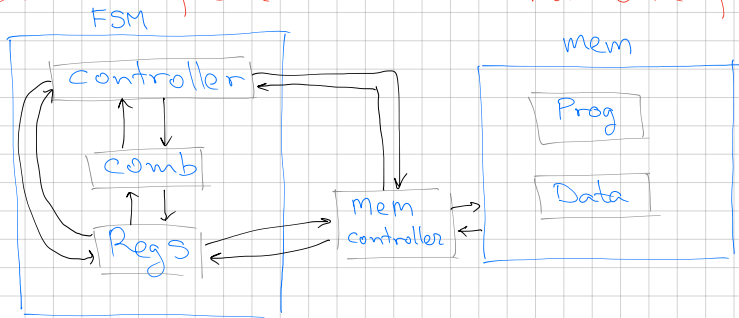
- 1 The input may change in cycle t only if $busy(t) = 0$.
- 2 If $busy(t) = 0$ and $busy(t - 1) = 1$, then:
 - 1 If $MR(t - 1) = 1$ then

$$OUT(t) \leftarrow M[\langle Address(t - 1) \rangle](t - 1).$$

- 2 If $MW(t - 1) = 1$ then

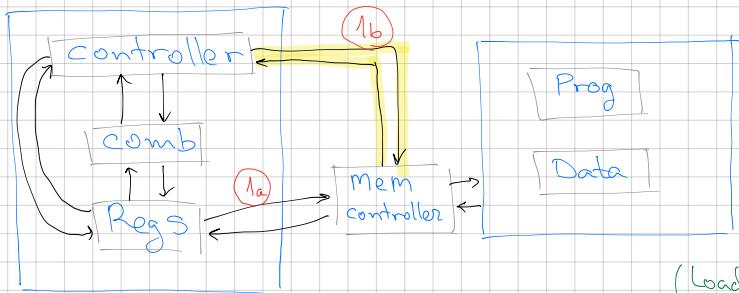
$$M[\langle Address(t - 1) \rangle](t) \leftarrow IN(t - 1).$$

DLX implementation: 2'nd attempt



- simplify FSM: break into 3 parts
- Controller is an FSM
- Mem controller is also an FSM
-

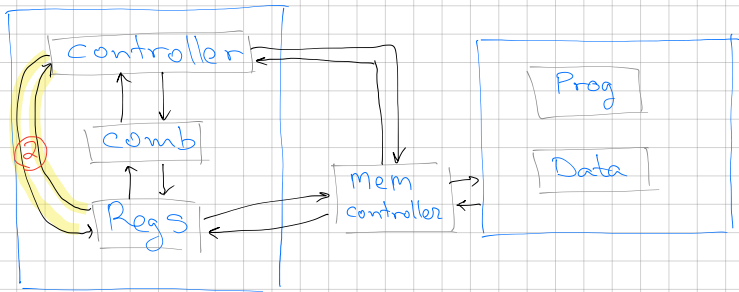
DLX implementation: 2'nd attempt



1a) address (MAR/PC) & data (MDR/IR) } (Load store instruction fetch)

1b) controller → mem. controller: read/write/idle
mem controller → controller: busy

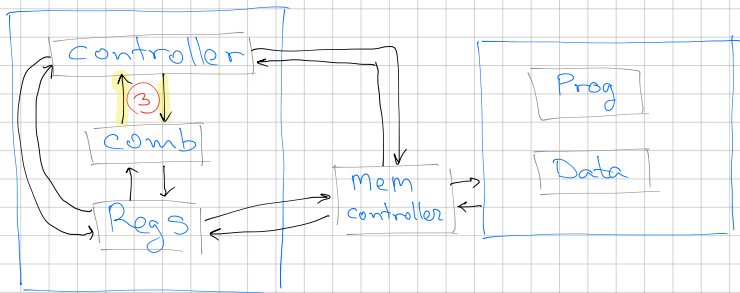
DLX implementation: 2'nd attempt



② controller \rightarrow Regs : clock enable signals

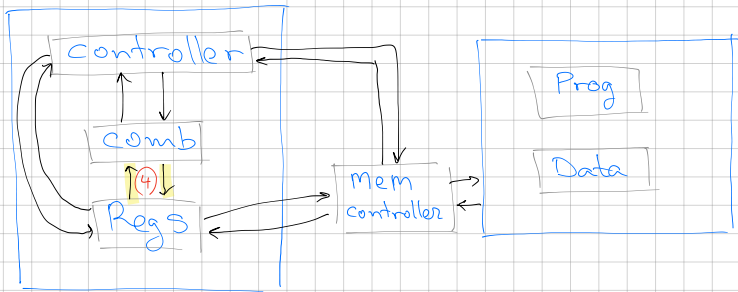
Regs \rightarrow controller: opcode of instruction

DLX implementation: 2'nd attempt



- ③ controller \rightarrow comb : ALU operation (add/sub/...)
comb \rightarrow controller : $A \stackrel{?}{=} 0$ (for branch inst.)

DLX implementation: 2'nd attempt

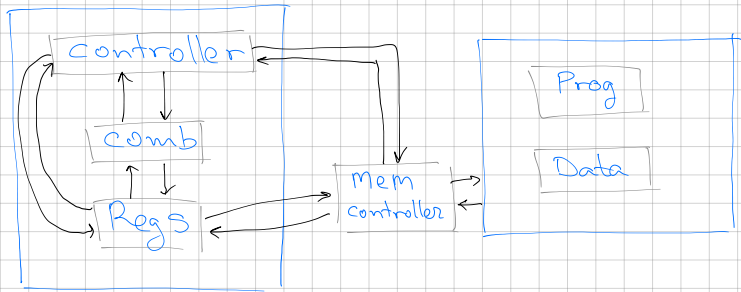


(4) Regs \rightarrow Comb : arguments
comb \rightarrow Regs : Result

examples : $R_j \leftarrow R_{i_1} + R_{i_2}$

$PC \leftarrow PC + 1$

DLX implementation: 2nd attempt



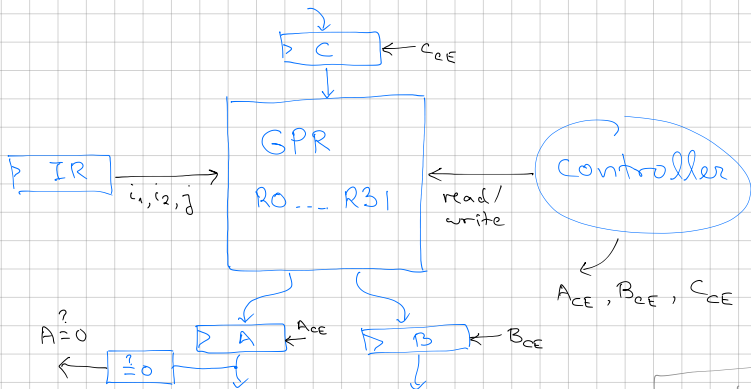
Problem:

mangement of $R0 \dots R31$ (GPR'S)

is complicated

(dual port RAM)

GPR Interface



read :

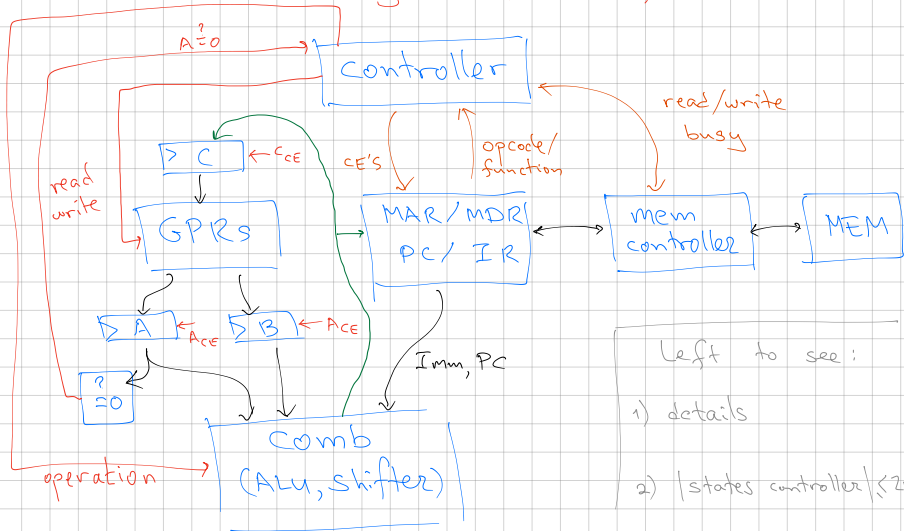
$$A \leftarrow R_{i_1} \quad \& \quad B \leftarrow R_{i_2}$$

write :

$$R_j \leftarrow C$$

Controller :
ignores details
 \Rightarrow fewer states !

DLX design (sketch)



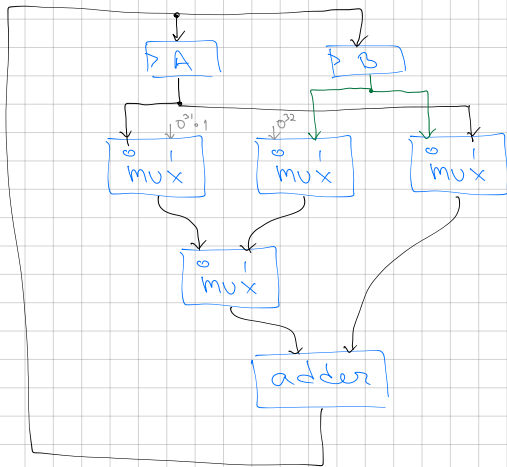
Left to see:

- 1) details
- 2) |states controller| ≤ 20

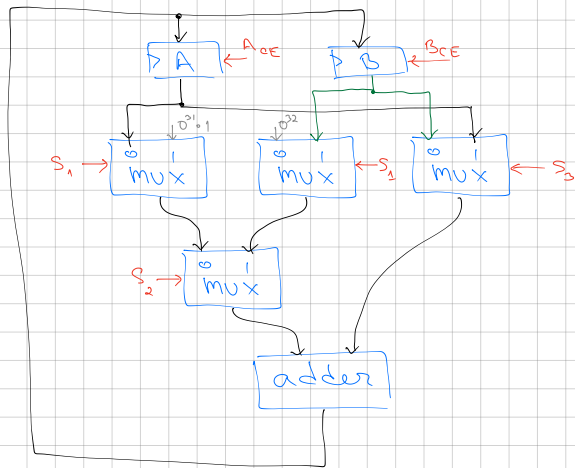
Register Transfer Language (RTL) Instructions

- The control governs the behavior of the datapath by its outputs called **control outputs**.
- The simplest control signal is a clock enable signal of a register in the datapath.
- In each state, the control tells which registers should store new values.
- We specify this action by a **Register Transfer Language**(RTL) instruction.
- The operands of an RTL instruction are the datapath registers, and the calculations are performed by the combinational circuits in the datapath.
- Every DLX instruction is executed by a specific sequence of RTL instructions.
- RTL instructions (except those that access the memory) require a single clock cycle.

Controlling the Data-Path (example)



Controlling the Data-Path (example)



RTL:

$$A \leftarrow A + B$$

$S_1 =$

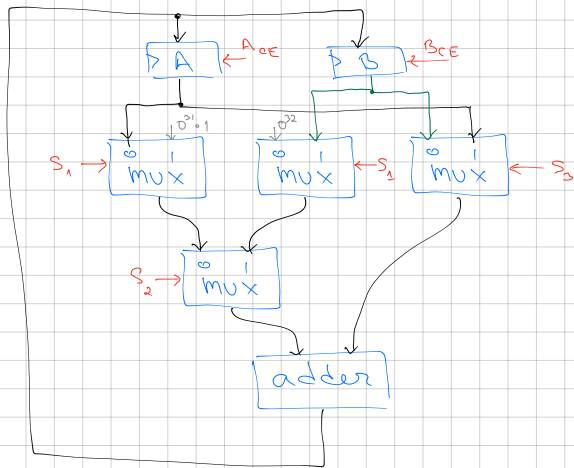
$S_2 =$

$S_3 =$

$A_{ce} =$

$B_{ce} =$

Controlling the Data-Path (example)



RTL:

$A \leftarrow A + 1$

$S_1 =$

$S_2 =$

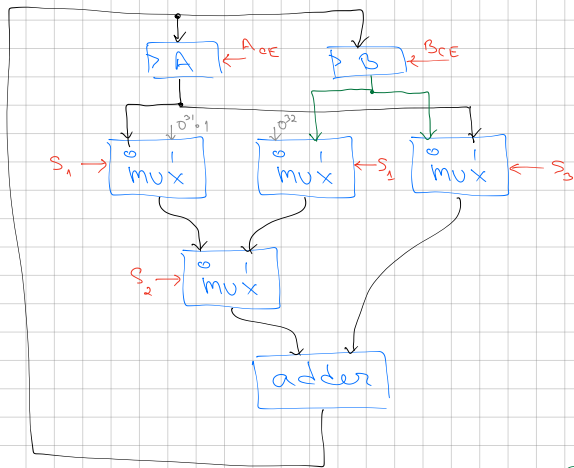
$S_3 =$

$A_{ce} =$

$B_{ce} =$

Q: $A \leftarrow A + B + 1$?

Controlling the Data-Path (example)



RTL:

$B \leftarrow A$

$S_1 =$

$S_2 =$

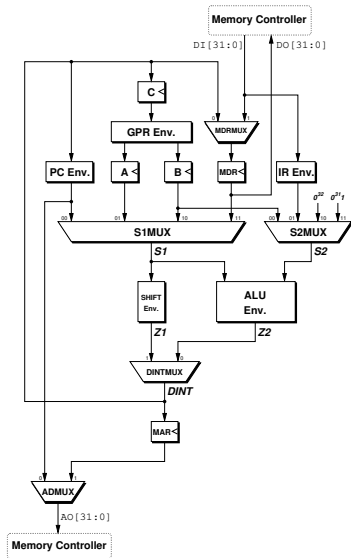
$S_3 =$

$A_{ce} =$

$B_{ce} =$

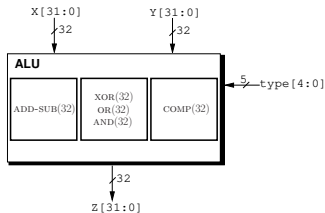
Q: $A \leftarrow B$?

The Datapath of the Simplified DLX Machine



ALU Environment

The ALU is a combinational circuit that supports: addition and subtraction, bitwise logical instructions, and comparison instructions.



The main three subcircuits of the ALU are: (1) 32-bit adder/subtractor, $ADD-SUB(32)$, (2) bitwise logical operations, XOR , OR , AND , and (3) a comparator, $COMP(32)$. Note that the comparator is fed by the outputs of the adder/subtractor circuit.

Definition

An **ALU environment** is a combinational circuit specified as follows:

Input: $x[31 : 0], y[31 : 0] \in \{0, 1\}^{32}$, $type \in \{0, 1\}^5$.

Output: $z[31 : 0] \in \{0, 1\}^{32}$.

Functionality:

$$\vec{z} \triangleq f_{type}(\vec{x}, \vec{y}),$$

We now need to describe how the ALU functions are encoded...

Encoding of ALU Functions

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

ALU - Functionality

- 1 The outcome of a comparison is one or zero depending on whether the expression is true.
- 2 The logical operations are bitwise.
- 3 The comparison operations return either 0^{32} or $0^{31} \circ 1$.
- 4 The input *type*[0] indicates if the function is addition. It is used, for example, to increment the program counter.
- 5 The input *type*[1] indicates if the function is comparison.

ALU - Connections in the Datapath

The datapath busses are connected to the ALU as follows.

- The bus $S1[31 : 0]$ is connected to the $x[31 : 0]$ input of the ALU.
- The bus $S2[31 : 0]$ is connected to the $y[31 : 0]$ input of the ALU.
- The bus $Z2[31 : 0]$ is connected to the $z[31 : 0]$ output of the ALU.

The signals $type[4 : 0]$ are outputs of the FSM called the DLX control.

Shifter Environment

- The shifter is a 32-bit bi-directional logical shifter by one position.
- Recall that $LLS(\vec{x}, i)$ denotes the logical left shift of \vec{x} by i positions, and that
- $LRS(\vec{x}, i)$ denotes the logical right shift of \vec{x} by i positions.

Shifter Environment: Definition

Definition

The **shifter environment** is a combinational circuit defined as follows:

Input:

- $x[31 : 0] \in \{0, 1\}^{32}$,
- $\text{shift} \in \{0, 1\}$, and
- $\text{right} \in \{0, 1\}$.

Output: $y[n - 1 : 0] \in \{0, 1\}^{32}$.

Functionality: The output \vec{y} satisfies

$$\vec{y} \triangleq \begin{cases} \vec{x}, & \text{if } \text{shift} = 0, \\ \text{LLS}(\vec{x}, 1), & \text{if } \text{shift} = 1, \text{right} = 0, \\ \text{LRS}(\vec{x}, 1), & \text{if } \text{shift} = 1, \text{right} = 1. \end{cases}$$

The shifter also implements the identity function: route a word through the shifter in the execution of some instructions.

Instruction Register (IR) Environment

- The **IR environment** holds the 32 bits of the current instruction.
- When executing an I-type instruction, the IR environment outputs the sign extension of the immediate field, and the indices of *RS1* and *RD*.
- When executing an R-type instruction, the IR environment outputs the indices of *RS1*, *RS2* and *RD*.
- The *RD* field is positioned in a different “places”.
- Selecting the right bits requires a circuit that computes whether the instruction is an I-type instruction. We delegate this computation to the DLX control, and denote the outcome of this computation as the *Itype* signal.

Definition

The **IR environment** is a synchronous circuit defined as follows:

Input: $DI[31 : 0] \in \{0, 1\}^{32}$, $IRce$, $JLINK$, $Itype \in \{0, 1\}$
and a clock signal CLK .

Output: An instruction $Inst[31 : 0]$, sign extension of the immediate constant $Imm[31 : 0]$, and the GPR addresses

$Aadr[4 : 0], Badr[4 : 0], Cadr[4 : 0] \in \{0, 1\}^5$.

Definition

[Functionality:]

$$\text{Inst}(t + 1) = \begin{cases} \text{Inst}(t) & \text{if IRce}(t) = 0, \\ DI(t) & \text{if IRce}(t) = 1. \end{cases}$$

$\text{Imm}[31 : 0](t) =$ sign extension of $\text{Inst}[15 : 0](t)$ to 32 bits.

$\text{Aadr}[4 : 0](t) = \text{Inst}[25 : 21](t),$

$\text{Badr}[4 : 0](t) = \text{Inst}[20 : 16](t),$

$$\text{Cadr}[4 : 0](t) = \begin{cases} 11111 & \text{if JLINK}(t) = 1, \\ \text{Inst}[20 : 16](t), & \text{if Itype}(t) = 1 \text{ and JLINK}(t) = 0, \\ \text{Inst}[15 : 11](t), & \text{otherwise.} \end{cases}$$

The IR environment is implemented by a parallel load clock enabled register and a 3 : 1-mux to select the value of Cadr.

Inputs and outputs of IR environment are connected as follows.

- The datapath bus $DI[31 : 0]$ is connected to the $DI[31 : 0]$ input of the IR environment.
- The $Imm[31 : 0]$ output of the IR environment is connected to the S2MUX.
- The outputs $Aadr$, $Badr$ and $Cadr$ are input to the GPR environment.
- The output $Inst[31 : 0]$ is in input to the FSM called the DLX control.
- The inputs $Itype$, $JLINK$ and $IRce$ are outputs of the DLX control.

Program Counter (PC) Environment

The PC environment is simply a 32-bit clock enabled parallel load register.

There are 32 registers in the GPR Environment, called R_0, R_1, \dots, R_{31} . The GPR Environment (or GPR, for short) can support one of two operations in each clock cycle.

- 1 Write the value of input C in R_i , where $i = \langle C_{\text{adr}} \rangle$.
- 2 Read the contents of the registers R_i and R_j , where $i = \langle A_{\text{adr}} \rangle$ and $j = \langle B_{\text{adr}} \rangle$.

Definition

A GPR is a synchronous circuit specified as follows.

Inputs: GPR addresses (output by the IR environment) $A_{\text{adr}}[4 : 0], B_{\text{adr}}[4 : 0], C_{\text{adr}}[4 : 0] \in \{0, 1\}^5$, a data input $C[31 : 0] \in \{0, 1\}^{32}$, a write-enable signal $\text{GPR_WE} \in \{0, 1\}$ and a clock signal CLK .

Output: $A[31 : 0], B[31 : 0] \in \{0, 1\}^{32}$, and a flag $\text{AEQZ} \in \{0, 1\}$.

Functionality: Let $R[i]$ denote the i th register in the GPR. The functionality of a GPR is specified by the following program:

Definition (Cont.)

- 1 data: array $R[31 : 0]$ of 32-bit wide registers.
- 2 initialize: $\forall i : R[i] \leftarrow 0^{32}$.
- 3 For $t = 0$ to ∞ do
 - 1 If $GPR_WE = 1$ and $\langle Cadr \rangle \neq 0$, then

$$R[\langle Cadr \rangle](t + 1) \leftarrow \vec{C}(t).$$

- 2 If $GPR_WE = 0$ then

$$\vec{A}(t + 1) \leftarrow R[\langle Aadr \rangle](t),$$

$$\vec{B}(t + 1) \leftarrow R[\langle Badr \rangle](t)$$

- 3 $AEQZ(t) \leftarrow \text{is-zero}(\vec{A}(t))$

1. only registers with active CEs are mentioned above.
2. sloppy: AEQZ depends on A which is not part of GPR.

- The control is an FSM that interprets the DLX instructions.
- For every DLX instruction I_j , it executes a sequence of RTL-instructions whose execution constitutes the execution of I_j .
- Each RTL-instruction corresponds to a state of the control and is implemented via the outputs of the control that are sent to the datapath and the memory controller.

A High Level View of the Execution Cycle

An execution of a DLX instruction requires multiple clock cycles. It is common to consider the following steps in the execution of an instruction:

- 1 Instruction Fetch: copy the instruction to be executed from the main memory to the Instruction Register (IR).

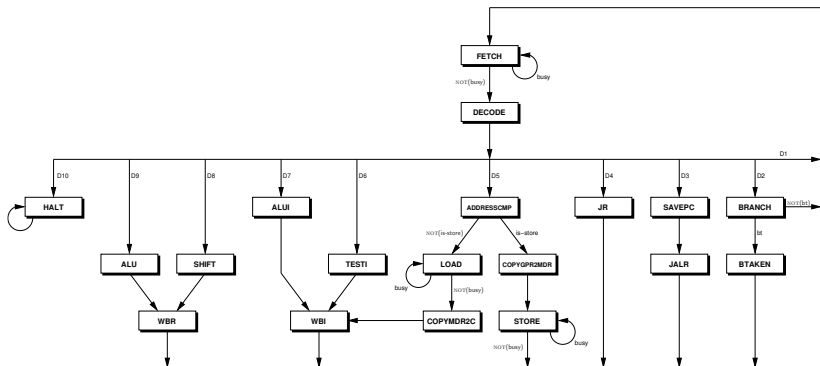
$$IR \leftarrow M[\langle PC \rangle].$$

- 2 Instruction Decode. Decode the instruction stored in the IR. Decoding means that the control decides what actions should take place.
- 3 Execute. Execute the instruction, for example, in an add instruction, the addition takes place in this step.
- 4 Memory Access. In this step load and store instructions access the main memory.
- 5 Write-back. Store the result of an instruction, if needed, in the GPR.

DLX Control - a finite state machine

- States
- Input alphabet: each bit is called a control input
- Output alphabet: each bit is called a control output
- State transition function
- Output function

DLX Controller State Diagram



The FSM has 19 states. We first list the states that correspond to steps in the execution cycle:

- ➊ Instruction Fetch. The Fetch state is the only state that deals with instruction fetch.
- ➋ Instruction Decode. The Decode state is the only state that deals with instruction decode.
- ➌ Execute. The states: Alu, Testl, Alul, and Shift deal with the execute step.
- ➍ Memory Access. The states Load and Store deal with memory access.
- ➎ Write-back. The states WBR and WBI deal with writing back the result in the GPR.

There are additional states that do not belong to the standard execution steps. These include the following states:

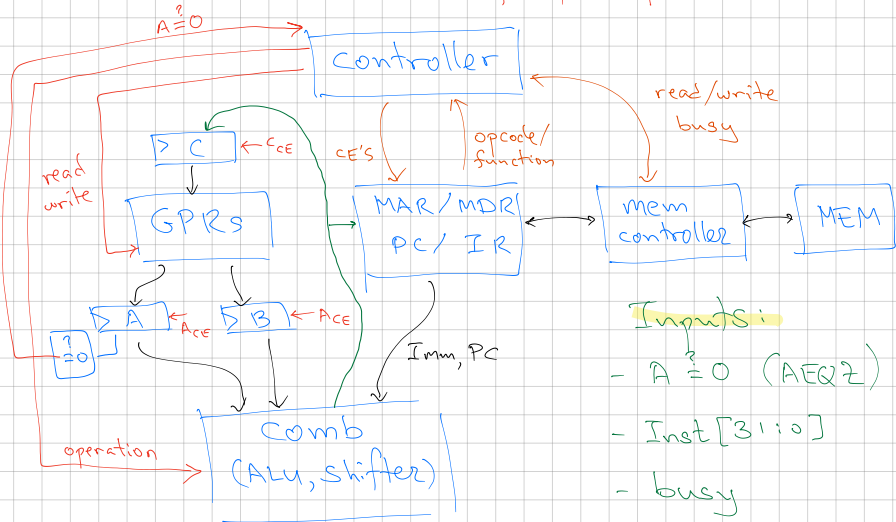
- 1 States that deal with the execution of branch and jump instructions. These are the states: Branch, Btaken, JR, Save PC, and JALR.
- 2 States that deal with load and store instructions. These are the states: Address-Computation, CopyMDR2C, and CopyGPR2MDR.
- 3 A sink state, called Halt, for stopping the execution.

Each bit of the input alphabet of the FSM is called a **control input**.

List of the control inputs:

- 1 The current instruction `Inst[31 : 0]` that is an output of the IR environment.
- 2 The `AEQZ` flag that indicates if the output of register `A` equals zero. (We view this flag as an output of the GPR environment.)
- 3 The busy flag that is output by the memory controller.

DLX-Controller Inputs/Outputs



Inputs:

- $A \neq 0$ (AEQZ)
- $Inst[31:0]$
- busy

FSM Outputs

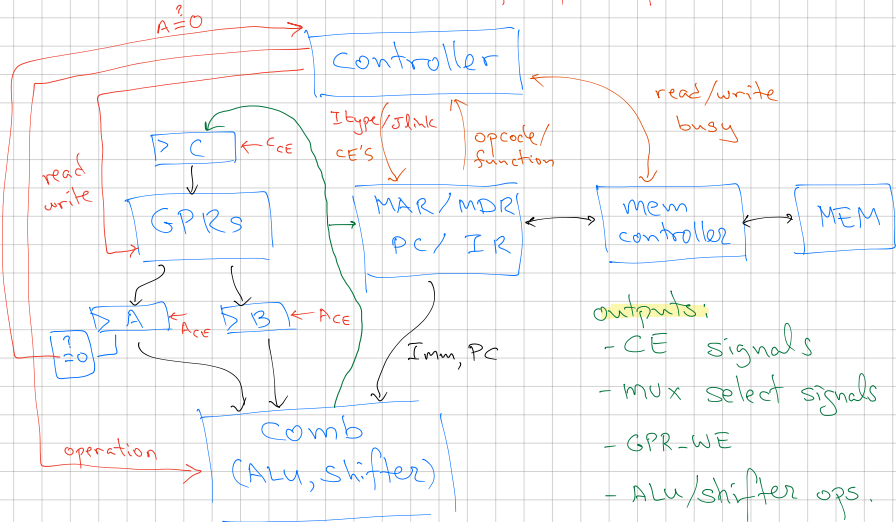
Each bit of the output alphabet of the FSM is called an **control output**.

- 1 IRCE, PCCE, ACE, BCE, CCE, MARCE, MDRCE: clock enable signals of the corresponding registers.
- 2 S1SEL[1:0], S2SEL[1:0], DINTSEL, MDRSEL, ADSEL: select signals of the S1MUX, S2MUX, DINTMUX, MDRMUX, and ADMUX selectors in the datapath.
- 3 ALUF[2:0], ADD, TEST: signals that are input to the ALU environment, as follows: $type[4:2] \leftarrow ALUF[2:0]$, $type[1] \leftarrow test$, and $type[0] \leftarrow add$. The value of ALUF[2:0] is computed by

$$ALUF[2:0] \leftarrow \begin{cases} opcode[2:0] & \text{if Inst is an I-type instruction} \\ function[2:0] & \text{if Inst is an R-type instruction.} \end{cases} \quad (1)$$

- ① SHIFT, RIGHT: signals that are input to the Shifter environment.
- ② Itype: indicates whether the current instruction is an I-type instruction. The Itype signal is input to the IR environment.
- ③ JLINK: This signal is input to the IR environment. The signal equals one if and only if the current instruction is a jalr instruction.

DLX-Controller Inputs/Outputs



outputs:

- CE signals
- mux select signals
- GPR-WE
- ALU/shifter ops.
- Mem. Control R/W
- JLINK, ITYPE

Summary of the control outputs

Signal	Value	Semantics
ALUf[2:0]		Controls the functionality of ALU
Rce		Register clock enable
S1sel[1:0]	00 01 10 11	PC A B MDR
S2sel[1:0]	00 01 10 11	B IR 0 1
DINTsel	0 1	ALU Shifter
MDRsel	0 1	DINT DI
ADsel	0 1	PC MAR

Summary of the Control Outputs - cont

Signal	Value	Semantics
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

$$IR = M[PC],$$

- Means: copy the contents of $M[PC]$ to the IR .
- Reading from the value stored in $M[PC]$ is performed by setting a control signal MR to be high.
- Once the result of the read is ready, the value is stored in the IR register since the clock enable of the IR register is set to high.
- We denote this clock enable signal by $IRCE$.

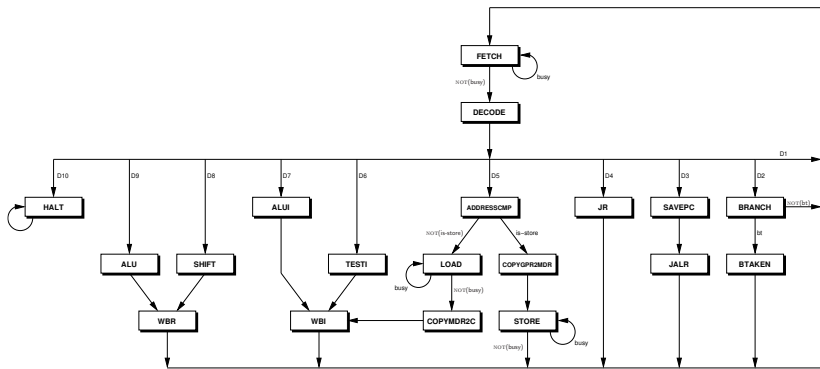
Output Function

State	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	$C = A \text{ op } B$	S1sel[0], Cce, active bits in ALUF[2:0]
Testl	$C = (A \text{ rel } imm)$	S1sel[0], S2sel[0], Cce, test, ltype, active bits in ALUF[2:0]
Alul(add)	$C = A + imm$	S1sel[0], S2sel[0], Cce, add, ltype
Shift	$C = A \text{ shift } sa$ $sa = 1, (-1)$	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

Output Function - cont

State	RTL Instruction	Active Control Outputs
CopyMDR2C	$C = \text{MDR}(\gg 0)$	S1sel[0], S1sel[1], S2sel[1], DINTsel, Cce
CopyGPR2MDR	$\text{MDR} = B(\ll 0)$	S1sel[1], S2sel[1], DINTsel, MDRce
WBR	$RD = C$ (R-type)	GPR_WE
WBI	$RD = C$ (I-type)	GPR_WE, ltype
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

DLX Controller State Diagram



Transition Function

The out-degree of most the control states is one. This means that the FSM transitions to the only “next state” independent of the input to the FSM. Only six states have an out-degree greater than one. We elaborate on the transitions from these six states.

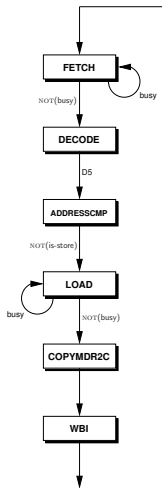
- 1 The Fetch, Load and Store states have a self-loop labeled by `busy`. This means, that if the input `busy` equals one, then the FSM stays in the same state.
- 2 The Branch state has two possible transitions. The transition to state `BTaken` is labeled `bt`, and the transition back to Fetch is labeled `NOT(bt)`. The value of `bt` is computed by the control and equals one if the condition of a conditional branch is satisfied. It is computed by

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

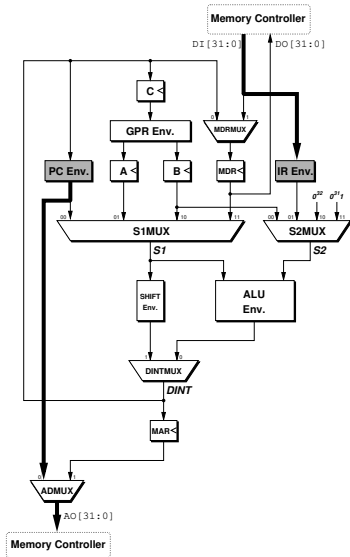
- 1 The Address-Computation has two possible transitions. The transition to CopyGPR2MDR is labeled `is-store`, and transition to Load is labeled `NOT(is-store)`. The value of `is-store` is computed by the control and equals 1 if the current instruction is a store-word (`sw`) instruction.
- 2 The Decode state has 10 possible transitions. These transitions are labeled D1 – D10. Exactly one of these signals equals 1, so that the transition is well defined.

Step-by-Step Execution of load-word

Sequence of control states:

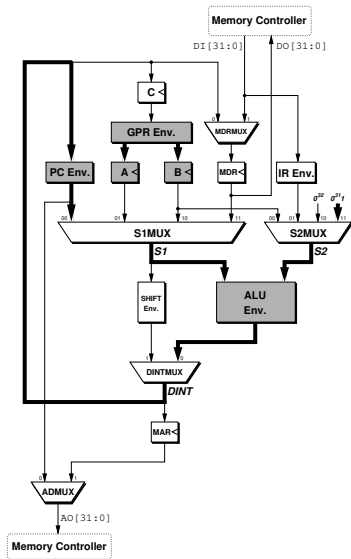


What happens in each state? $IR = M[PC]$



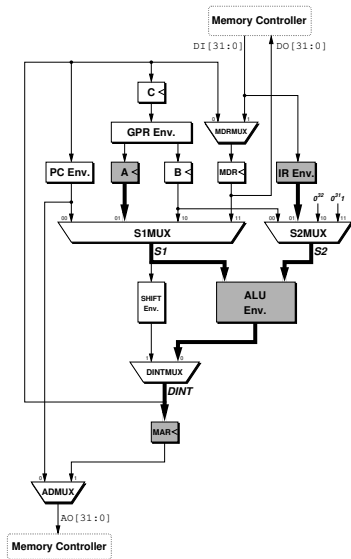
What happens in each state? DECODE:

$A = RS1, B = RS2, PC = PC + 1$

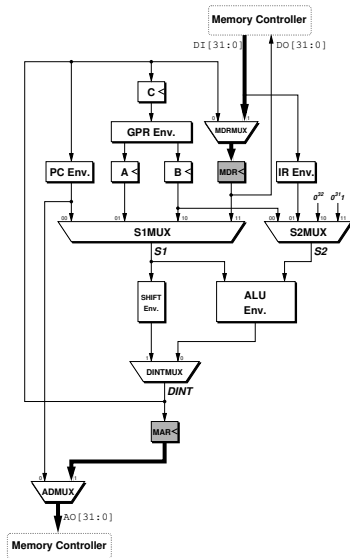


What happens in each state? ADDRESSCMP:

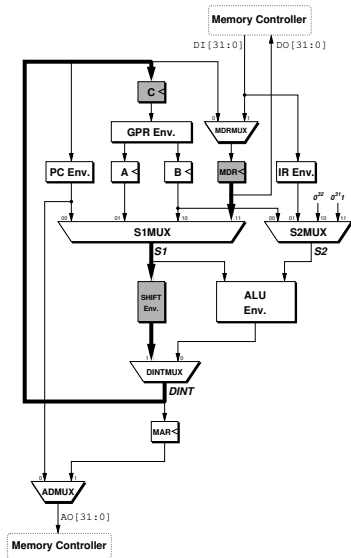
$$MAR = A + imm$$



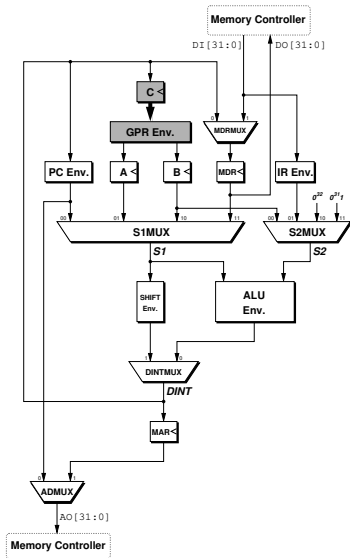
What happens in each state? LOAD: $MDR = M[MAR]$



What happens in each state? COPYMDR2C: $C = MDR$



What happens in each state? WBI: $RD = C$



- We described every module in the datapath by specifying its inputs, outputs and functionality.
- We described the control of the DLX by its state machine.
- We “glued” all these components by describing which RTL instruction is executed in every step.
- We executed a DLX instruction step by step.
- Full details of an implementation of the simplified DLX.
- There is no need to learn this implementation by heart. It is merely a suggestion for an implementation.
- Try to understand the underlying principles. The best way to see how the design works is by executing all the instructions step by step.