

9. Basic Processor Design: from Finite State Machines to Single-Cycle Datapaths

EECS 370 – Introduction to Computer Organization – Winter 2009

Prof. Todd Austin & Prof. Marios Papaefthymiou

EECS Department
University of Michigan, Ann Arbor, USA

© T. Austin & M. Papaefthymiou, 2009

The material in this presentation cannot be
copied in any form without our written permission

Lecture schedule

1. Combinational & Sequential Logic:
 - Basics of electronics; logic gates, muxes, decoders
 - clocks and data storage
2. ALU design
 - Building an adding circuit
3. **State Machines**
 - **Building a simple processor**

Processors

- ❑ General-purpose processors
 - Good for most tasks, maybe overkill for some, too slow for others
 - Programmable control (defined by machine instructions in memory)

- ❑ Special-purpose processors
 - Necessary for applications with non-standard requirements
 - Very high performance, regular tasks (graphics)
 - Very low power and/or cost (watch)
 - Special purpose may or may not be programmable

- ❑ This lecture: Special purpose microcontroller → General-purpose processor
 - Design a non-programmable controller
 - Implement it using the components from last lecture
 - Start designing a general-purpose processor (single cycle) for LC2 ISA

A simple device

Build a custom controller
for a vending machine.

We could use a general
purpose processor, but
we might save money
with a custom controller.

Take coins, give drinks.



Input and Output

Inputs:

- coin trigger
- refund button
- 10 drink selectors
- 10 pressure sensors

Outputs:

- 10 drink release latches
- Coin refund latch



Operation of Machine

- Accepts quarters only
- All drinks are \$0.75
- Once we get the money, they can select a drink.
- If they want a refund, release any coins inserted

No free drinks!

No stealing money!



Building the controller

- ❑ Finite State
 - Remember how many coins have been put in the machine and what inputs are acceptable

- ❑ Read-Only Memory (ROM)
 - Define the outputs and state transitions

- ❑ Custom combinatorial circuits
 - Reduce the size (and therefore cost) of the controller

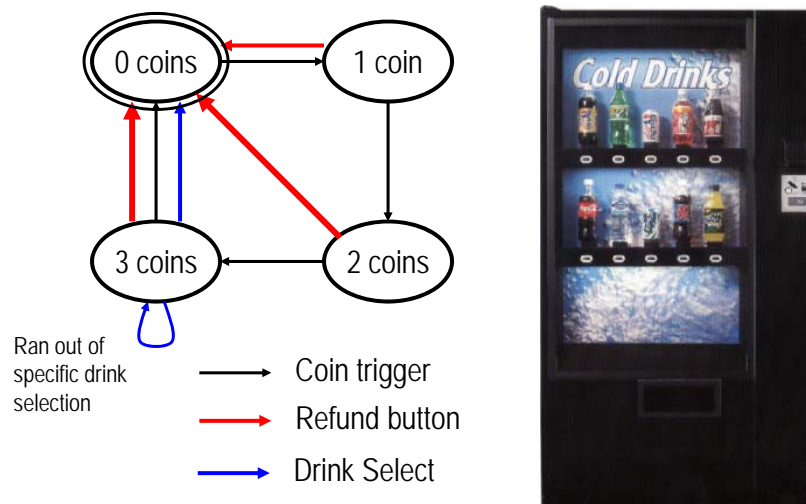
Finite State Machines

- ❑ A Finite State Machine (FSM) consists of:
 - K states: $S = \{s_1, s_2, \dots, s_k\}$, s_1 is initial state
 - N inputs: $I = \{i_1, i_2, \dots, i_n\}$
 - M outputs: $O = \{o_1, o_2, \dots, o_m\}$
 - Transition function $T(S, I)$ mapping each current state and input to next state
 - Output Function $P(S)$ [or $P(S, I)$] specifies output

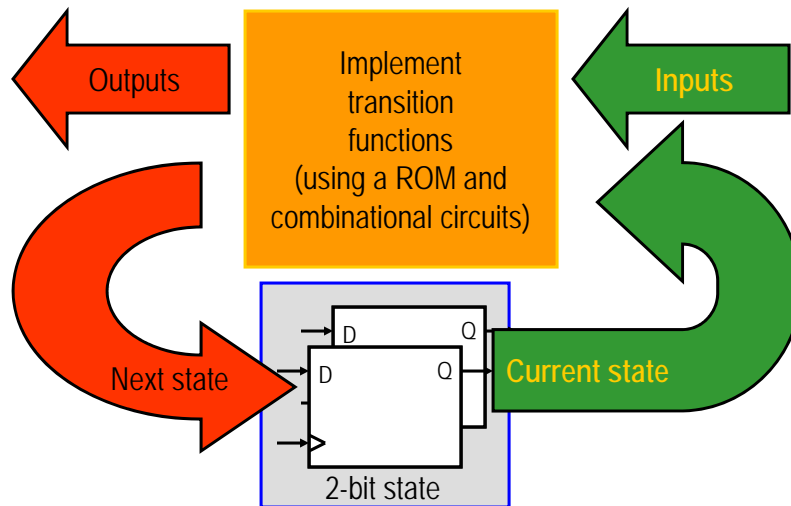
Two Common State Machines

- ❑ **Moore machine**
output function based on **current state** only
- ❑ **Mealy machine**
output function based on **current state** and **current input**

FSM for Vending Machine



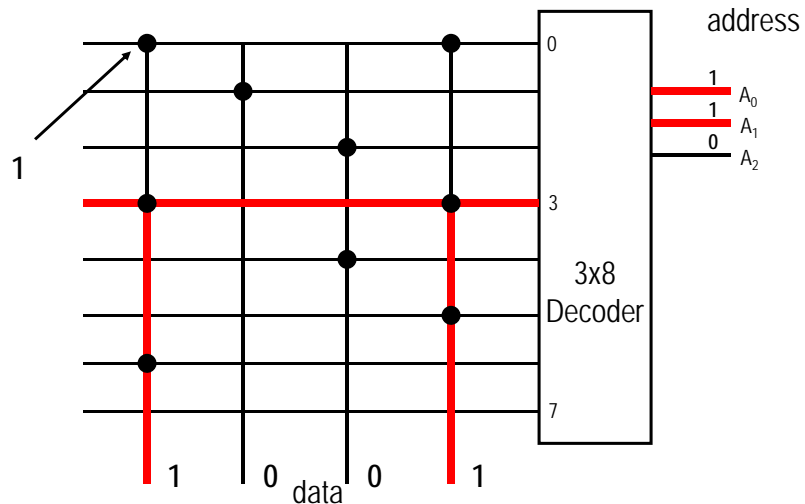
Implementing FSM



ROMs and PROMs

- ❑ Read Only Memory
 - Array of memory values that are constant
 - Non-volatile
- ❑ Programmable Read Only Memory
 - Array of memory values that can be written exactly once (destructive writes)
- ❑ You can use ROMs to implement FSM transition functions
 - ROM inputs (i.e., ROM address): current state, primary inputs
 - ROM outputs (i.e., ROM data): next state, primary outputs

8-entry 4-bit ROM



ROM for Vending Machine Controller

- ❑ Use current state and inputs as address
 - 2 state bits + 22 inputs = 24 bits (address)
- ❑ Read next state and outputs from ROM
 - 2 state bits + 11 outputs = 13 bit (memory)
- ❑ We need 2^{24} entry, 13 bit ROM memories
 - 218,103,808 bits! of ROM seems excessive for our cheap controller

Reducing the ROM needed

- Replace 10 selector inputs and 10 pressure inputs with a single bit input (drink selected)
 - Use drink selection input to specify which drink release latch to activate
 - Only allow trigger if pressure sensor indicates that there is a bottle in that selection. (10 2-bit ANDs)

- Now:
 - 2 current state bits + 3 input bits (5 bit ROM address)
 - 2 next state bits + 2 control trigger bits (4 bit memory)
 - $2^5 \times 4 = 128$ bit ROM (good!)

Some of the ROM contents

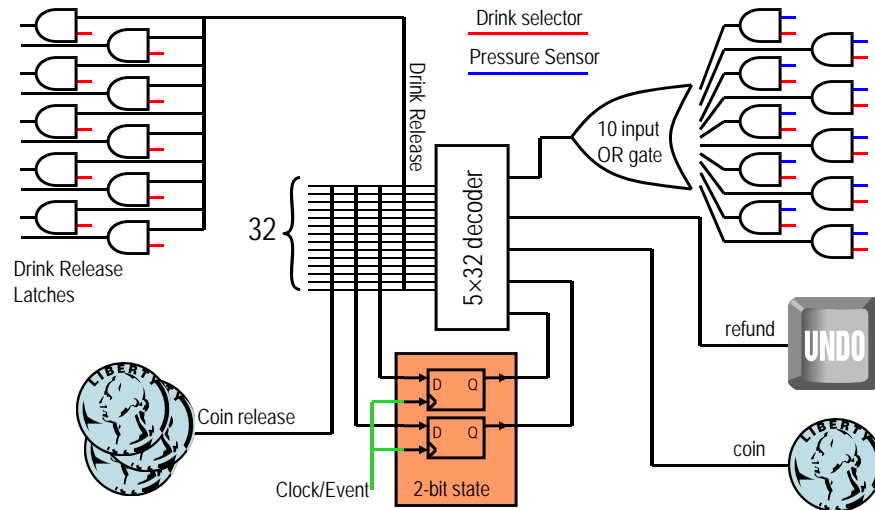
Current state	Coin trigger	Drink select	Refund button
0 0	0	0	0
0 0	0	0	1
0 0	0	1	0
0 0	1	0	0
0 1	1	0	0
1 0	1	0	0
1 1	0	1	0
1 1	1	0	0
... 24 more entries			

ROM address (current state, inputs)

Next state	Coin release	Drink release
... 24 more entries		

ROM contents (next state, outputs)

Putting it all together



EECS 370: Introduction to Computer Organization

The University of Michigan
© T. Austin & M. Papaefthymiou– 2009

17/33

Limitations of the controller

- ❑ What happens if we make the price \$1.00?, or what if we want to accept nickels, dimes and quarters?
 - Must redesign the controller (more state, different transitions)
 - A programmable processor only needs a software upgrade.
 - If you had written really good software anticipating a variable price, perhaps no change is even needed

EECS 370: Introduction to Computer Organization

The University of Michigan
© T. Austin & M. Papaefthymiou– 2009

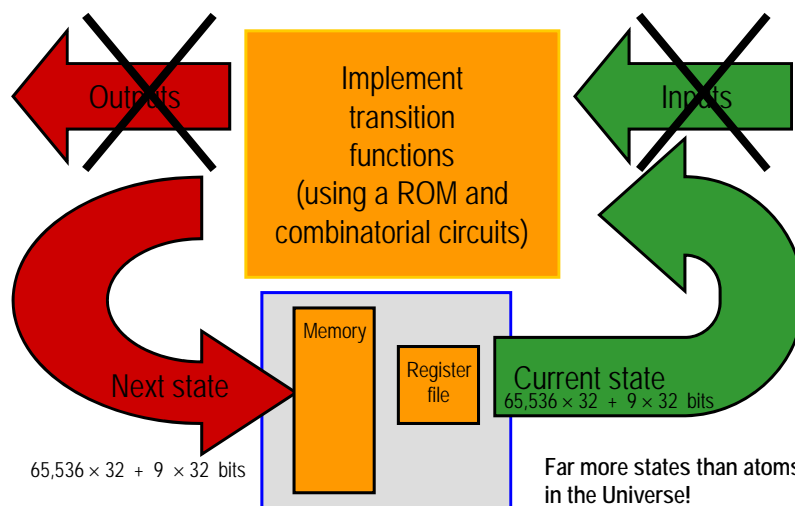
18/33

Single-Cycle Processor Design

□ General-Purpose Processor Design

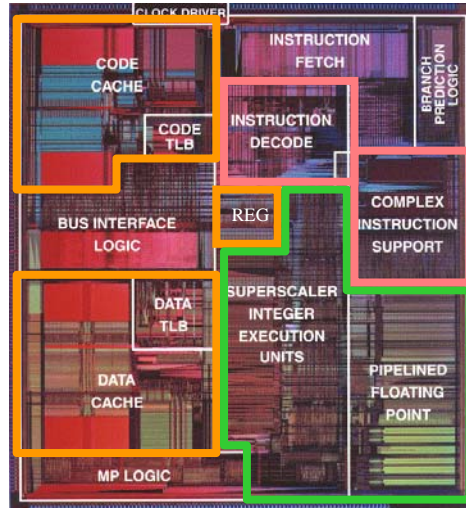
- Fetch Instructions
- Decode Instructions
 - Instructions are input to control ROM
- ROM data controls movement of data
 - Incrementing PC, reading registers, ALU control
- Clock drives it all
- Single-cycle datapath: Each instruction completes in one clock cycle

LC2Kx Processor as FSM



Pentium Processor Die

- ❑ State
 - Registers
 - Memory
- ❑ Control ROM
- ❑ Combinational logic (Compute)

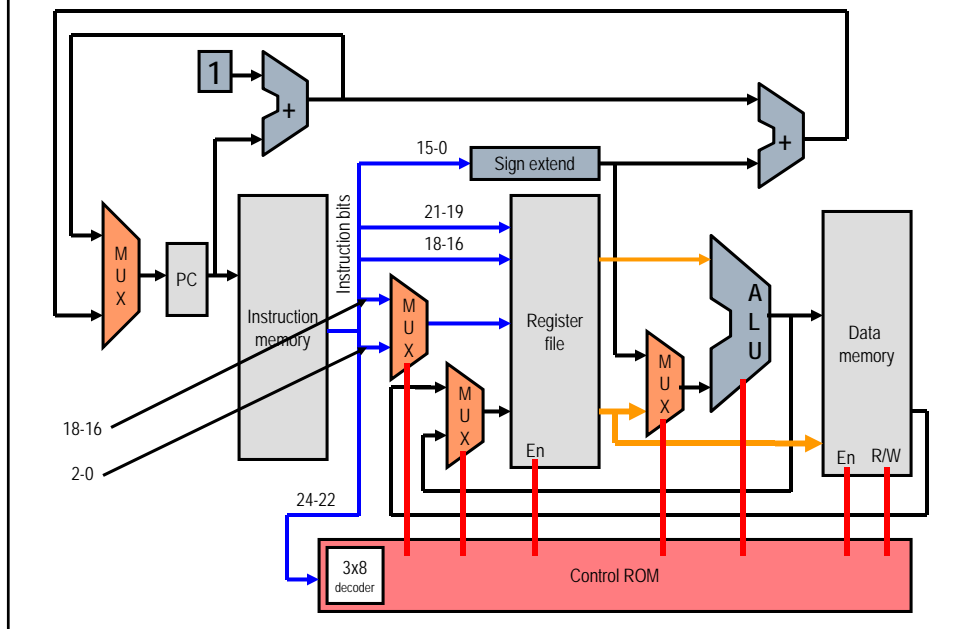


EECS 370: Introduction to Computer Organization

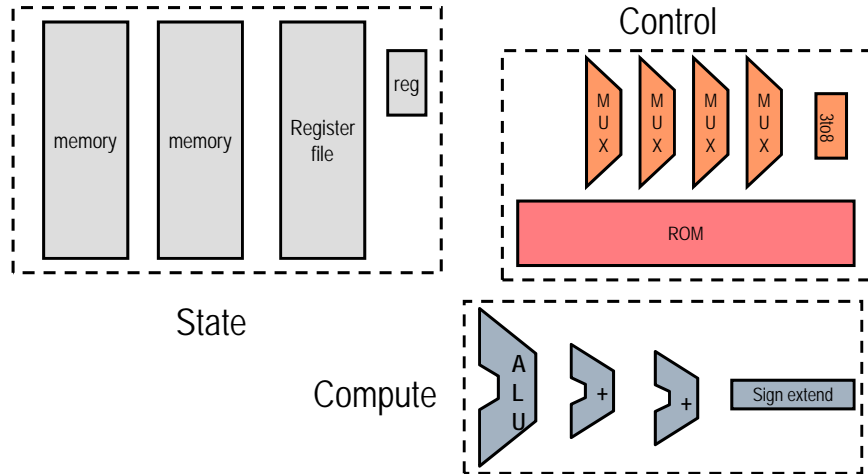
The University of Michigan
© T. Austin & M. Papaefthymiou– 2009

21/33

LC2Kx Datapath Implementation



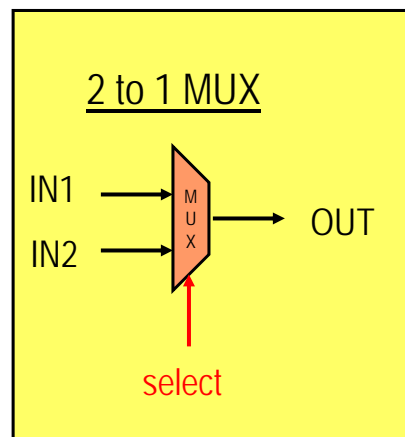
Building Blocks for the LC2



Here are the pieces, go build yourself a processor!

23/33

Control Building Blocks (1)

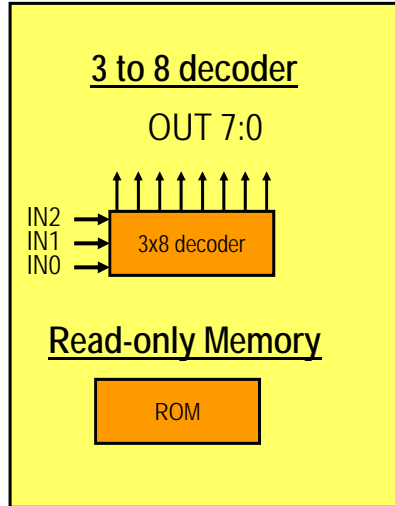


Connect one of the inputs to OUT based on the value of select

If (select == 0)
OUT = IN1
Else
OUT = IN2

24/33

Control Building Blocks (2)



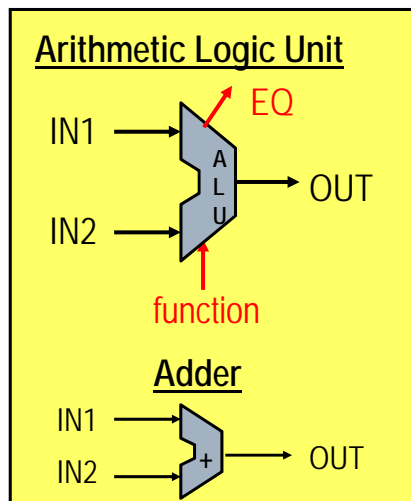
Decoder activates one of the output lines based on the input

IN	OUT
<u>210</u>	<u>76543210</u>
000	00000001
001	00000010
010	00000100
011	00001000
etc.	

ROM just stores preset data in each location.
Give address to access data.

25/33

Compute Building Blocks (1)



Perform basic arithmetic functions

$$\text{OUT} = f(\text{IN1}, \text{IN2})$$

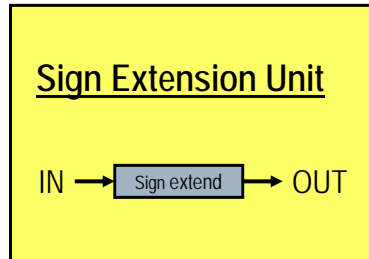
$$\text{EQ} = (\text{IN1} == \text{IN2})$$

For LC2, $f = \text{add, nand}$.
For other processors, there are many more functions.

Simple ALU – Does only adds

26/33

Compute Building Blocks (2)



Sign extend input by replicating the MSB to width of output

$$\text{OUT}(31:0) = \text{SE}(\text{IN}(15:0))$$

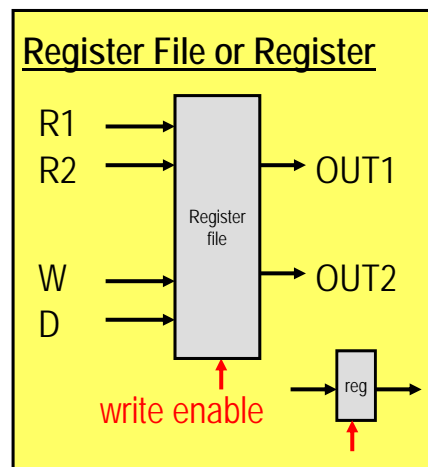
$$\text{OUT}(31:16) = \text{IN}(15)$$

$$\text{OUT}(15:0) = \text{IN}(15:0)$$

Useful when compute unit is wider than data

27/33

State Building Blocks (1)



Small/fast memory to store temporary values

n entries (LC2 = 8)

r read ports (LC2 = 2)

w write ports (LC2 = 1)

* Ri specifies register number to read

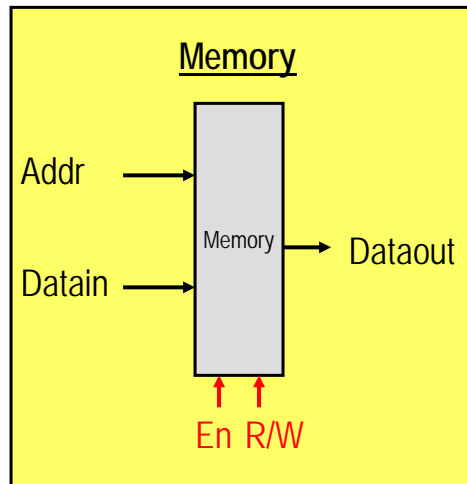
* W specifies register number to write

* D specifies data to write

How many bits are Ri and Wi in LC2?

28/33

State Building Blocks (2)



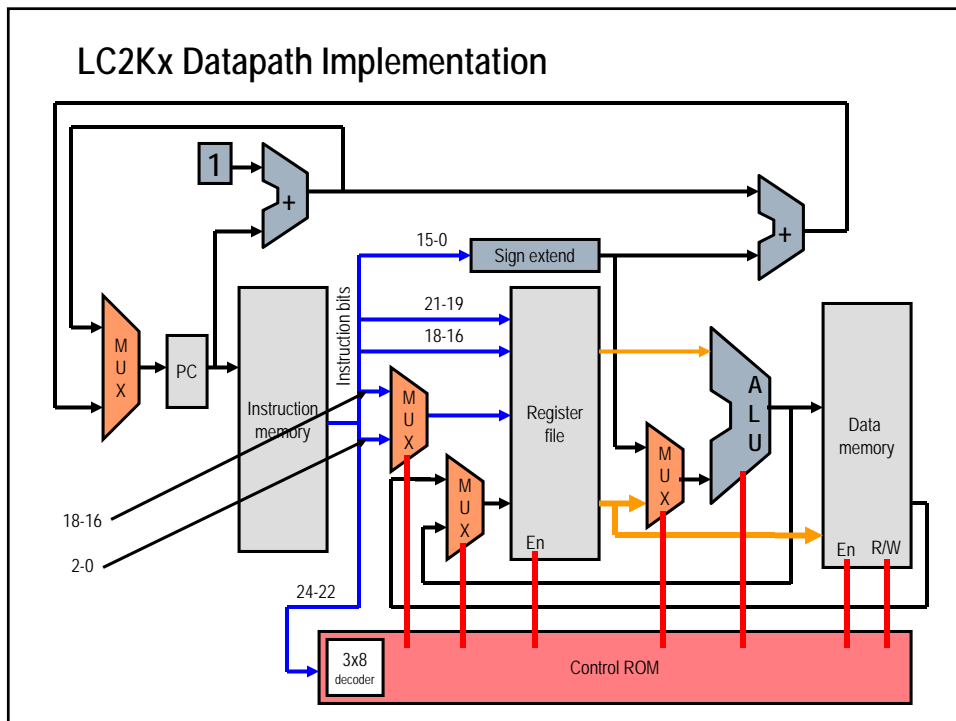
Slower storage structure to hold large amounts of stuff.

Use 2 memories for LC2

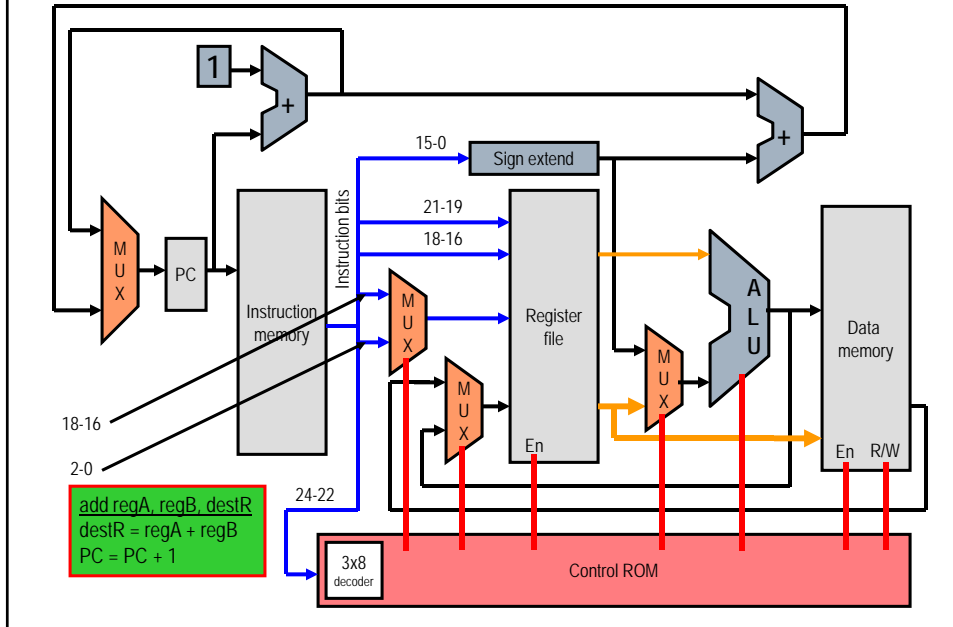
- * Instructions
- * Data
- * 65,536 total words

29/33

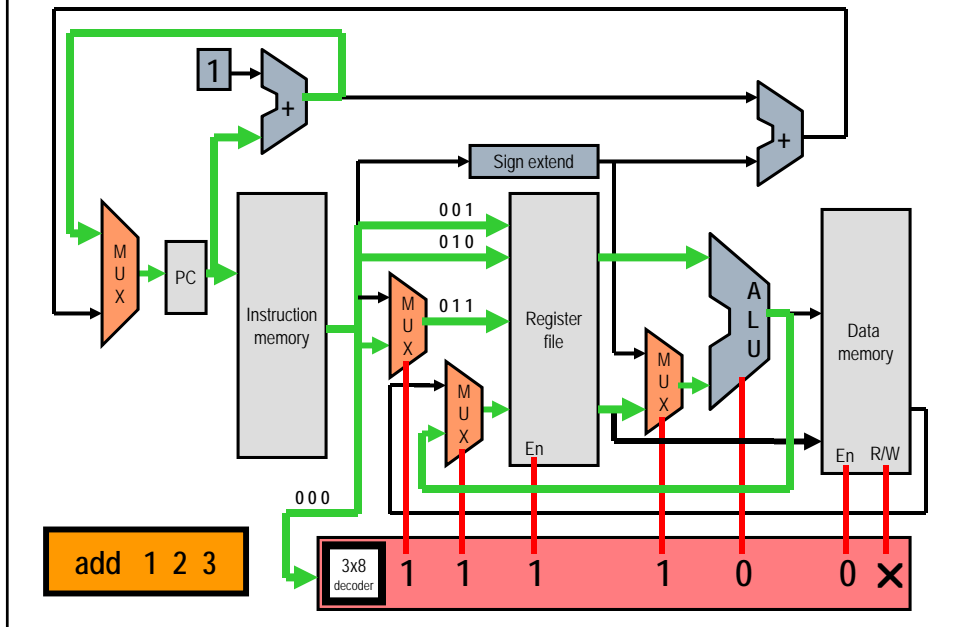
LC2Kx Datapath Implementation



Executing an ADD Instruction



Executing an ADD Instruction on LC2Kx Datapath



LC2Kx Processor - To Be Continued...

33/33