CSM CS61C Note #5: Finite State Machines (FSM), RISC-V Single-Cycle Datapath

Anthony Han

1 Boolean Algebra

Notation

- **Plus "+" for OR - "logical sum."**
- **Product "·" for AND - "logical product."**
- **Hat "‾" for NOT - complement.**

Truth Table for Combinational Logic

Exhaustive list of the output value generated for each combination of inputs. For a logic function with 3 inputs, we can do a truth table:

a	b	c	y
0	0	0	$F(0,0,0)$
0	0	1	$F(0,0,1)$
0	1	0	$F(0,1,0)$
0	1	1	$F(0,1,1)$
1	0	0	$F(1,0,0)$
1	0	1	$F(1,0,1)$
1	1	0	$F(1,1,0)$
1	1	1	$F(1,1,1)$

Given a truth table, we can derive its logic function as

$$F = \sum_{F(x_1,\ldots,x_n)==1} \prod_{i=1}^{n} \phi(x_i)$$

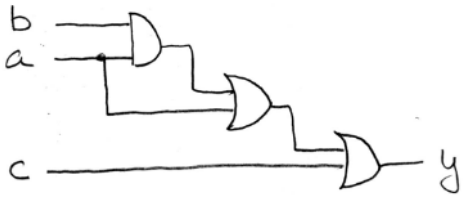
where $\phi(x_i) = \begin{cases} x_i & x_i == 1 \\ (\sim x_i) & x_i == 0 \end{cases}$.

Laws of Boolean Algebra

$X\overline{X} = 0$	$X + \overline{X} = 1$	Complementarity
$X0 = 0$	$X + 1 = 1$	Laws of 0's and 1's
$X1 = X$	$X + 0 = X$	Identities
$XX = X$	$X + X = X$	Idempotent Laws
$XY = YX$	$X + Y = Y + X$	Commutativity
$(XY)Z = X(YZ)$	$(X + Y) + Z = X + (Y + Z)$	Associativity
$X(Y + Z) = XY + XZ$	$X + YZ = (X + Y)(X + Z)$	Distribution
$XY + X = X$	$(X + Y)X = X$	Uniting Theorem
$\overline{X}Y + X = X + Y$	$(\overline{X} + Y)X = XY$	Uniting Theorem v2
$(\overline{XY}) = \overline{X} + \overline{Y}$	$(\overline{X + Y} = \overline{XY})$	DeMorgan's Law

Boolean Algebra Simplifies Circuits

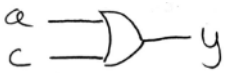
Given this circuit that implements $y = ab + a + c$, we could do gate minimization.



Using some boolean algebra, we get

$$\begin{aligned} y &= ab + a + c \\ &= a(b + 1) + c \\ &= a(1) + c \\ &= a + c. \end{aligned}$$

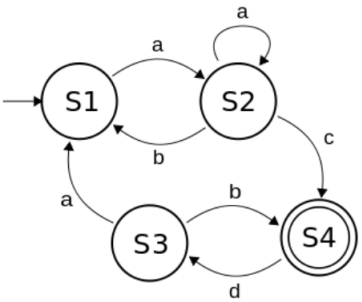
And we can simplify the gates.



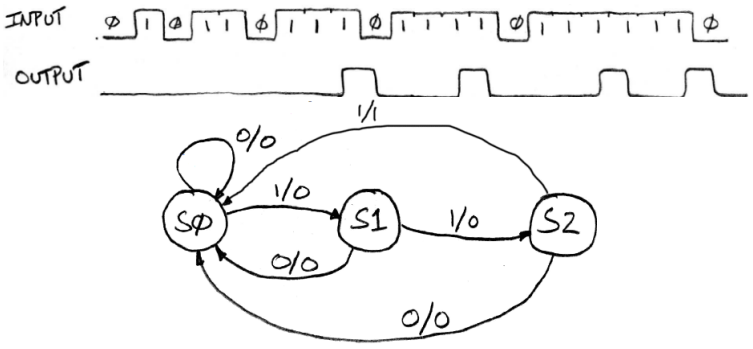
2 Finite State Machines (FSM)

Finite State Machines (FSM) Intro

- A convenient way to conceptualize computation over time.
- Start at a state and given an input, follow some edge to another (or the same) state.
- The function can be represented with a "state transition diagram."
- With combinational logic and registers, any FSM can be implemented in hardware.



FSM Example: Detecting Three Consecutive 1's in the Input



3 RISC-V Single-Cycle Datapath

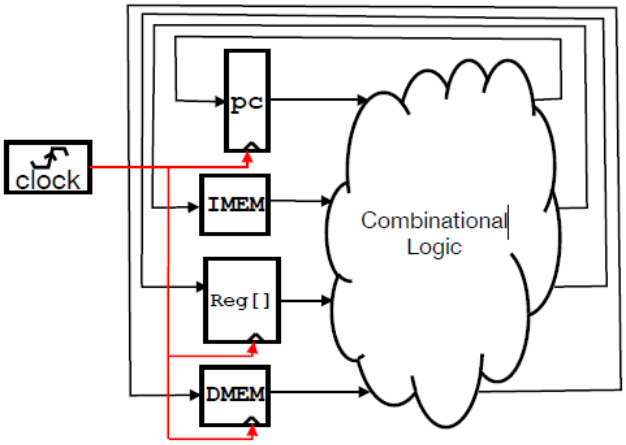
"State" Required by RV32I ISA

If we treat a RISC-V processor as a FSM, then each instruction reads and updates this state during execution:

- **Registers** (x0, ..., x31)
  - **Register file (or *regfile*)** Reg holds 32 registers × 32 bits/register: Reg[0], ..., Reg[31].
  - **First register read** specified by *rs1* field in instruction.
  - **Second register read** specified by *rs2* field in instruction.
  - **Write register (destination)** specified by *rd* field in instruction.
  - x0 is always 0 (**writes to Reg[0] are ignored**).
- **Program Counter** (PC)
  - Holds address of current instruction.
- **Memory** (MEM)
  - Holds both instructions & data, in one 32b byte-addressed memory space.
  - We'll use **separate memories for instructions (IMEM) and data (DMEM).** Later, these will be replaced with instruction and data caches.
  - Instructions are fetched from instruction memory (assume IMEM read-only).
  - Load/store instructions access data memory.

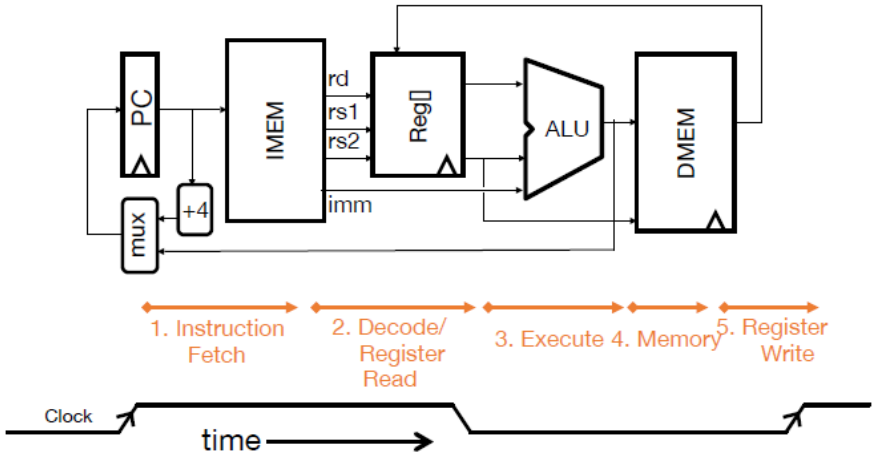
One-Instruction-Per-Cycle RISC-V Machine

First let's see a single cycle model: on every tick of the clock, the computer executes one instruction.

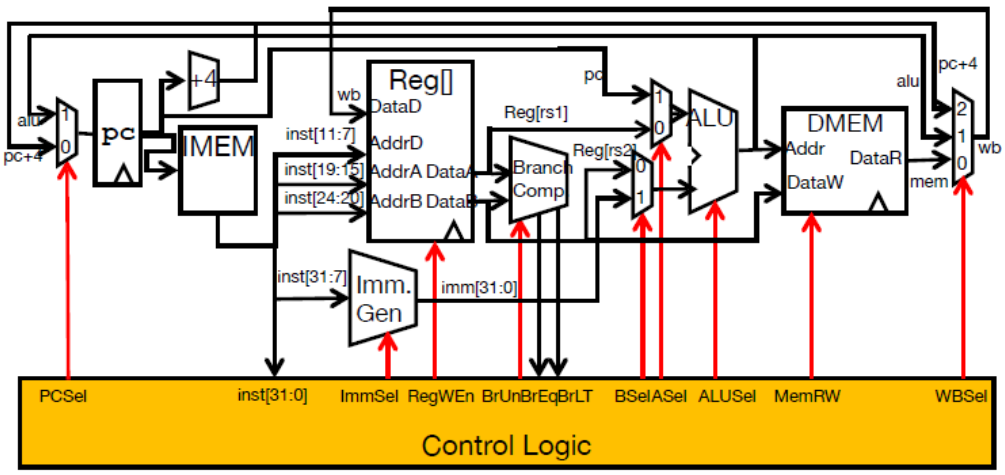


1. **Current state outputs drive the inputs to the combinational logic, whose outputs settles at the values of the state before the next clock edge.**
2. **At the rising clock edge, all the state elements are updated** with the combinational logic outputs, and execution moves to the next clock cycle.
3. **Separate instruction/data memory:** For simplification, memory is **asynchronous read** (not clocked), but **synchronous write** (is clocked).

Basic Phases of Instruction Execution



Single-Cycle RISC-V RV32I Datapath



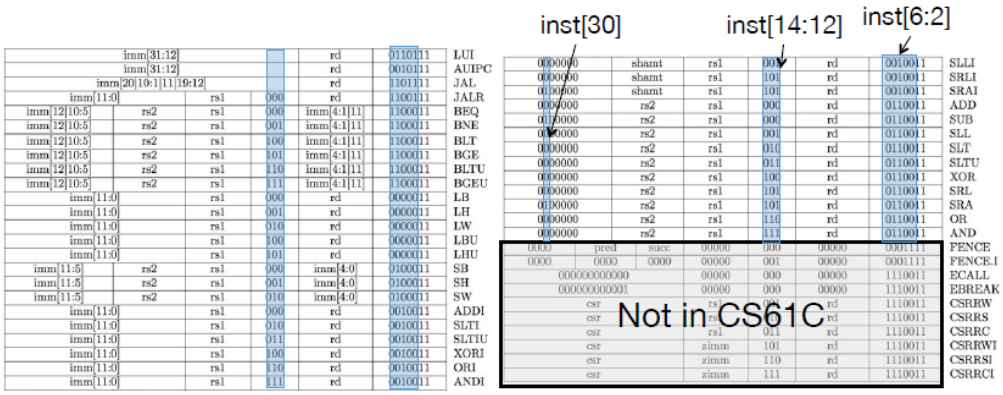
- **Universal datapath** - capable of executing all RISC-V instructions in one cycle each.
- 5 phases of execution:
  - **IF** - instruction fetch;
  - **ID** - instruction decode;
  - **EX** - execute;
  - **MEM** - memory access;
  - **WB** - write back.
- **Not all units and phases are used in every instruction.**
- **Controller specifies how to execute instructions.**

RISC-V Single-Cycle Control

A simple way to do the control logic is to implement a **truth table**:

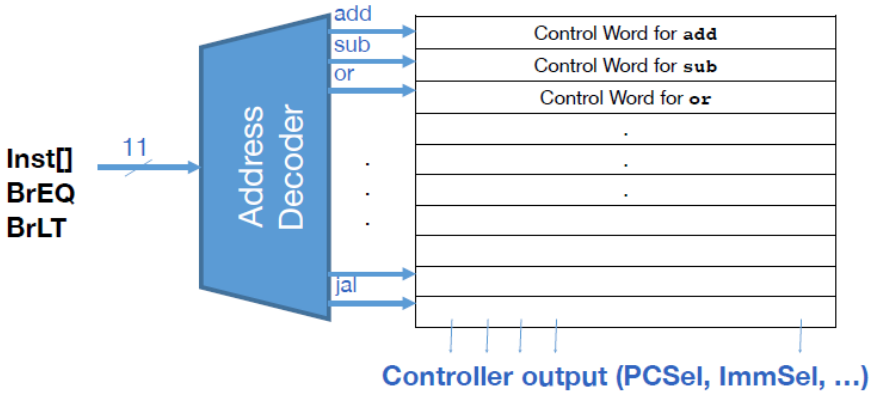
Inst[31:0]	BrEq	BrLT	PCSel	ImmSel	BrUn	ASel	BSel	ALUSel	MemRW	RegWEn	WBSel
`add`	*	*	+4	*	*	Reg	Reg	Add	Read	1	ALU
`sub`	*	*	+4	*	*	Reg	Reg	Sub	Read	1	ALU
(R-R Op)	*	*	+4	*	*	Reg	Reg	*(Op)*	Read	1	ALU
`addi`	*	*	+4	I	*	Reg	Imm	Add	Read	1	ALU
`lw`	*	*	+4	I	*	Reg	Imm	Add	Read	1	Mem
`sw`	*	*	+4	S	*	Reg	Imm	Add	Write	0	*
`beq`	0	*	+4	B	*	PC	Imm	Add	Read	0	*
`beq`	1	*	ALU	B	*	PC	Imm	Add	Read	0	*
`bne`	0	*	ALU	B	*	PC	Imm	Add	Read	0	*
`bne`	1	*	+4	B	*	PC	Imm	Add	Read	0	*
`blt`	*	1	ALU	B	0	PC	Imm	Add	Read	0	*
`bltu`	*	1	ALU	B	1	PC	Imm	Add	Read	0	*
`jalr`	*	*	ALU	I	*	Reg	Imm	Add	Read	1	PC+4
`jal`	*	*	ALU	J	*	PC	Imm	Add	Read	1	PC+4
`auipc`	*	*	+4	U	*	PC	Imm	Add	Read	1	ALU

In fact, we note that **instruction types can be encoded using only 9 bits: inst[30], inst[14:12], inst[6:2]:**



There are primarily two control realization options:

- **ROM - Read-Only Memory**
  - Regular structure.
  - Can be easily reprogrammed: fix errors, add instructions.
  - Popular when designing control logic manually.



- Nowadays, chip designers use logic synthesis tools to convert truth tables to **networks of gates.**