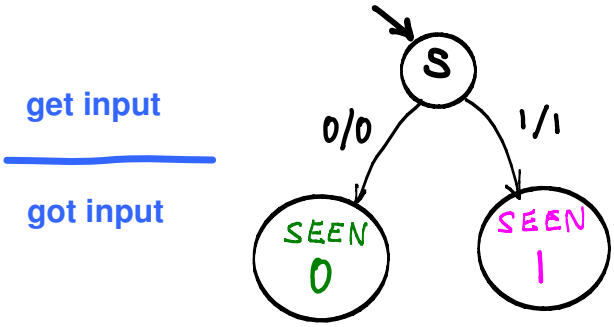
# Registers




"**SEEN 0**" **registers** that we've seen a **0**.

"**SEEN 1**" **registers** that we've seen a **1**.

State "registers" or "remembers"
what we have seen.

TM's **FSM** (the CONTROL part of a TM)
**No tape**

**one input** **IN**

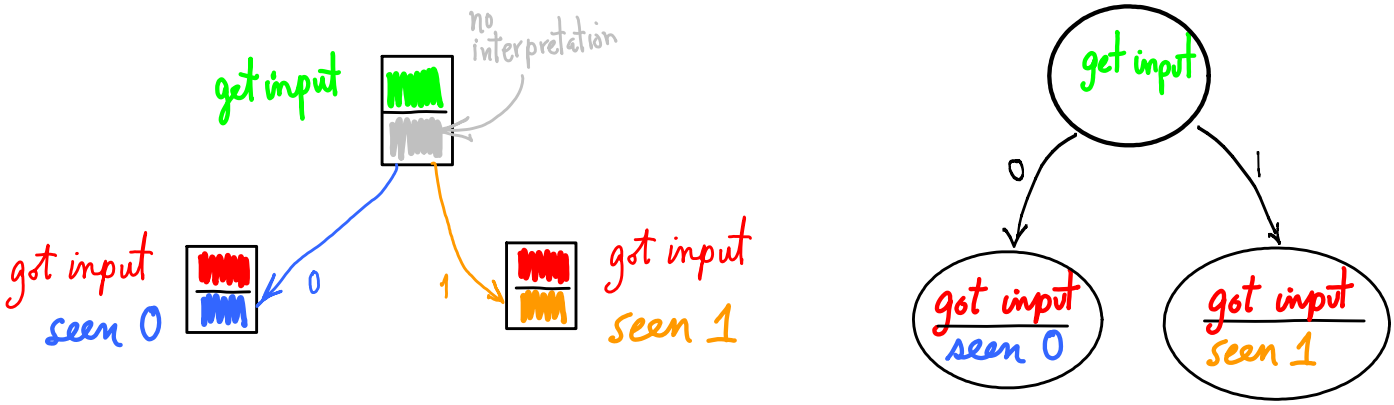
**two outputs** **OUT, MOVE**

**inputs/outputs** are **time series**:

| **time**: | 0 | 1 | 2 | 3 | 4 | 5 | 6 ... |
|---|---|---|---|---|---|---|---|
| **IN** : | 0 | 0 | 0 | 1 | 1 | 1 | 0 ... |
| **OUT** : | 0 | 1 | 1 | 0 | 1 | 0 | 1 ... |
| **MOVE** : | 0 | 1 | 1 | 0 | 1 | 0 | 1 ... |

Physical **state changes** in **time**.

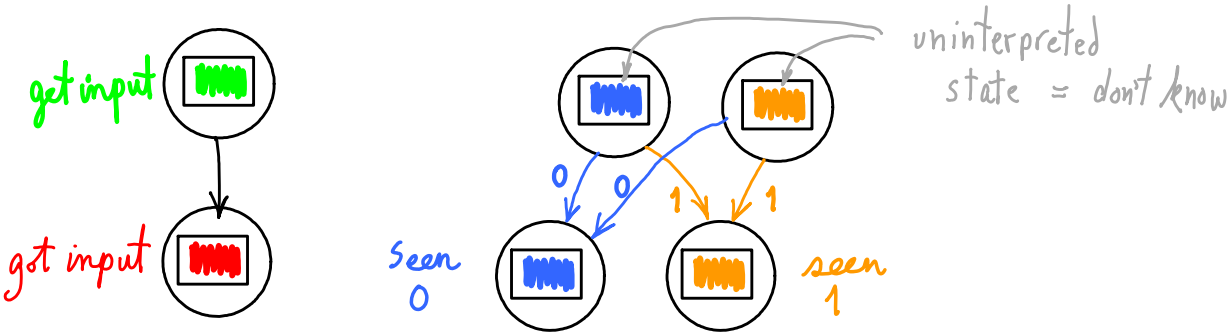
**Physical State** encodes **Logical State**

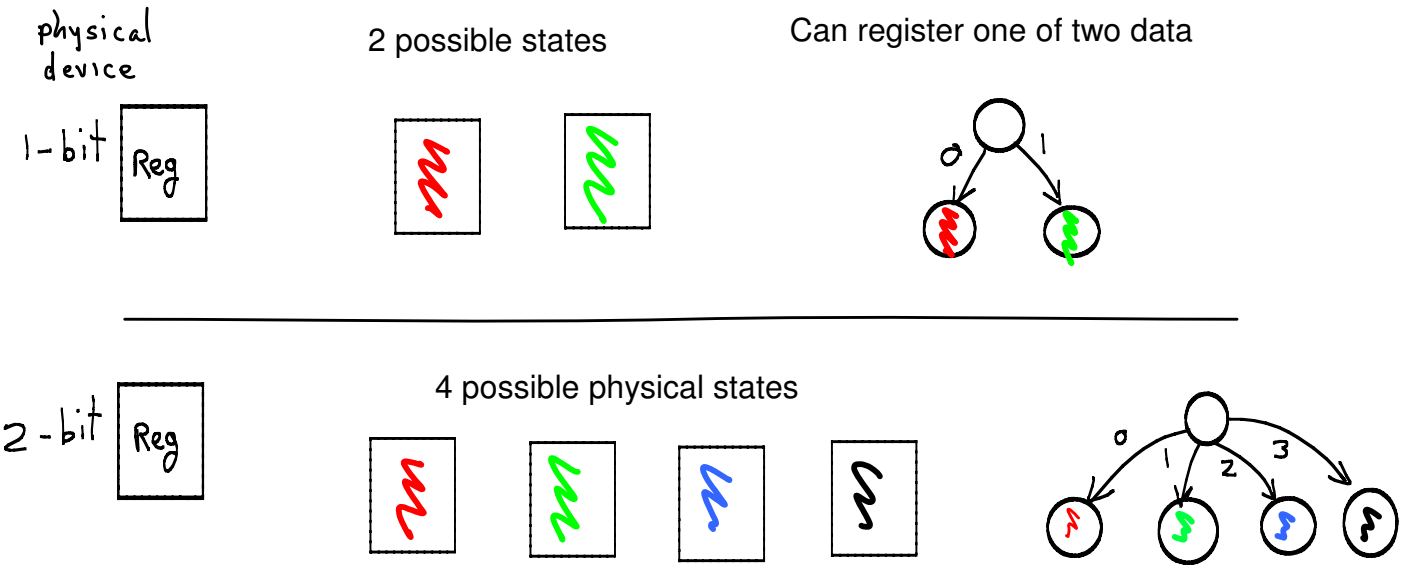



STATE can be thought of as consisting of **two parts**:

**step** of operation

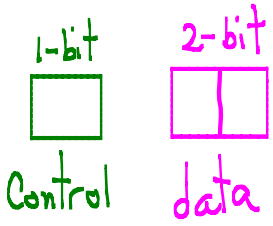
**symbol** seen





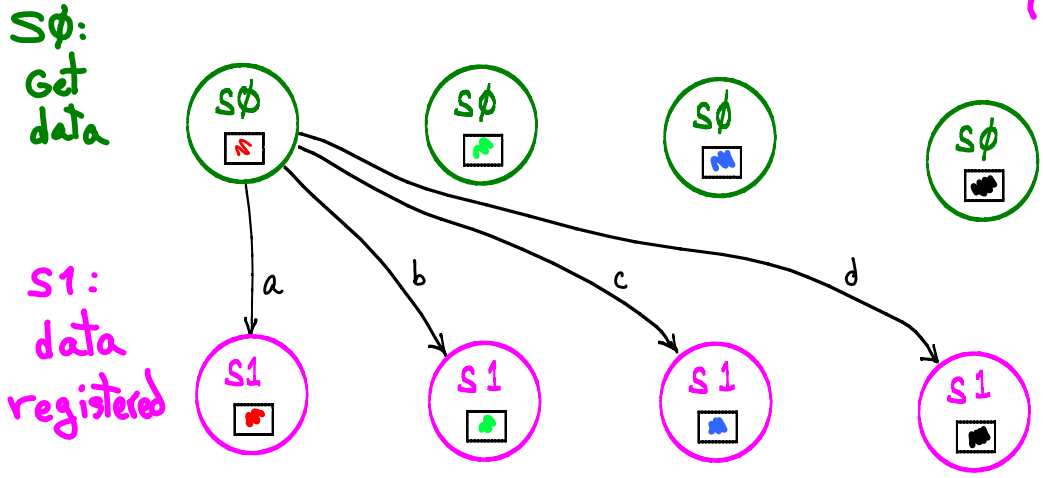

**1**-bit register ==> 2^**1** == 2 states
**k**-bit register ==> 2^**k** states

2 **1**-bit registers ==> (2^**1**)(2^**1**) == 4 states
2 **k**-bit registers ==> (2^**k**)(2^**k**) == 2^(**k**+**k**) states




**Physical system, 8 states:**

| control | data |
|---|---|
| 0 | 00 |
| 0 | 01 |
| 0 | 10 |
| 0 | 11 |
| 1 | 00 |
| 1 | 01 |
| 1 | 10 |
| 1 | 11 |

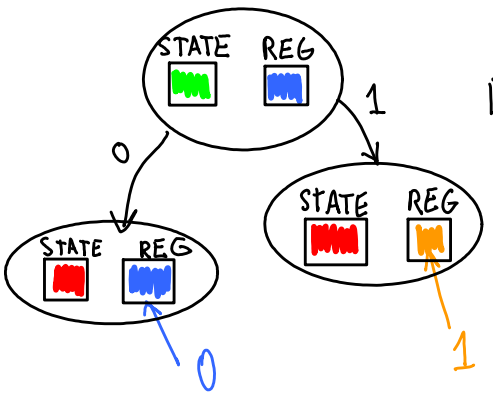



**Before reading** input ( **"get-data"** control state): we **don't care** which of upper 4 states we are in.

**After reading**, (**"data-registered"** control state) we are in **one of 4 states, data is recorded.**

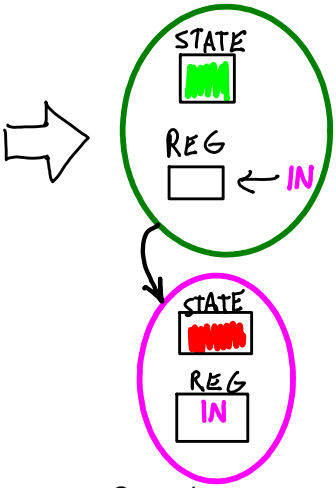
**32-bit data** ==> **4G branches**. We'd like to **ignore data state**, **concentrate on control state**.)

## "Total State"




Complete state:
(1) ready-for-data-and-reg-is-zero,
(2) got-data-and-reg-is-zero
(3) got-data-and-reg-is-one

## "Control State" + "Data Register"




Control state:
(1) get-data
(2) got-data

**BIG IDEA:**

**SPLIT TOTAL STATE** into **two parts**:

--- 1. **OPERATIONAL STATE**
**Where we are** in doing things

--- 2. **DATA REGISTER STATE**
**What we know** at this point

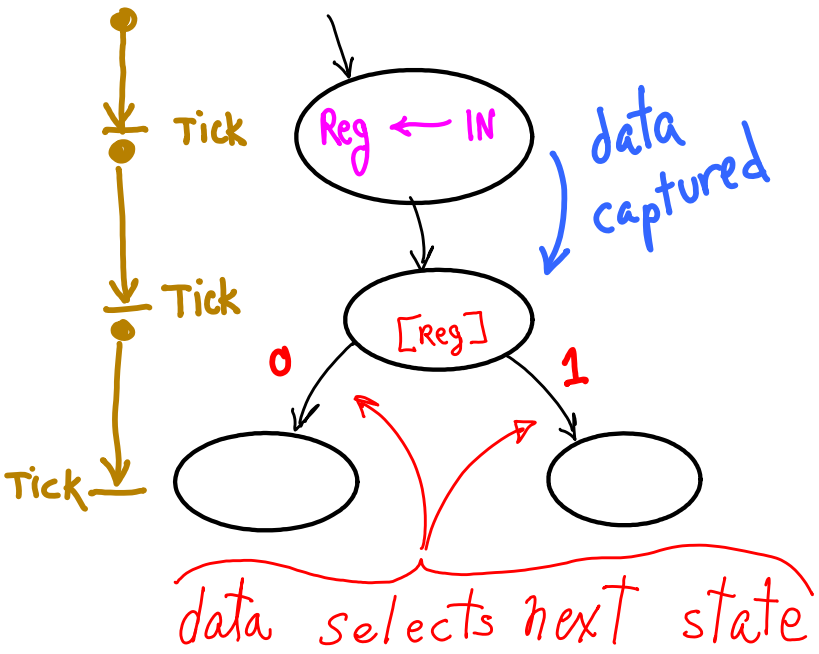
**registering a 32-bit input symbol** :

(**2 CONTROL** states) X (**4G data** states)

versus

-- **2 CONTROL** states (+ content of reg.)

# Control Branching




**Next CONTROL state**
depends on **register content**.

-- **States** === **Register Transfers**

-- **Branches** labeled
w/ **register content**.

**CLOCK** causes:
---- **register transfers**
---- **control state changes**

**REGISTERS** for
---- **CONTROL STATE**
---- **DATA**

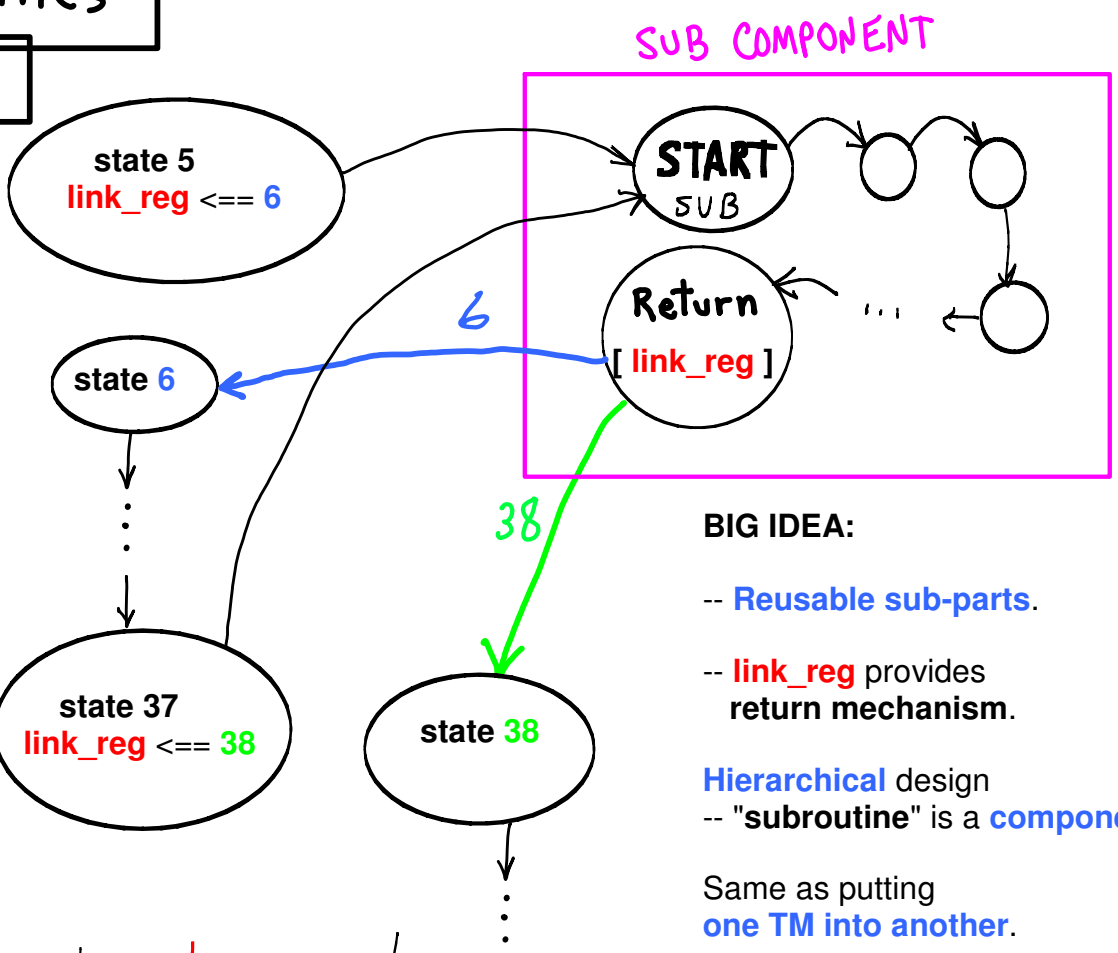
---- BOTH types change w/ CLOCK

We choose which part of data state is relevant, ignoring the rest.

# Subroutines

## sub-units

## HW/SW




**BIG IDEA:**

-- **Reusable sub-parts**.

-- **link_reg** provides **return mechanism**.

**Hierarchical** design
-- "**subroutine**" is a **component**

Same as putting **one TM into another**.

**HW = SW**
-- **Machines/descriptions** have **hierarchical design**.

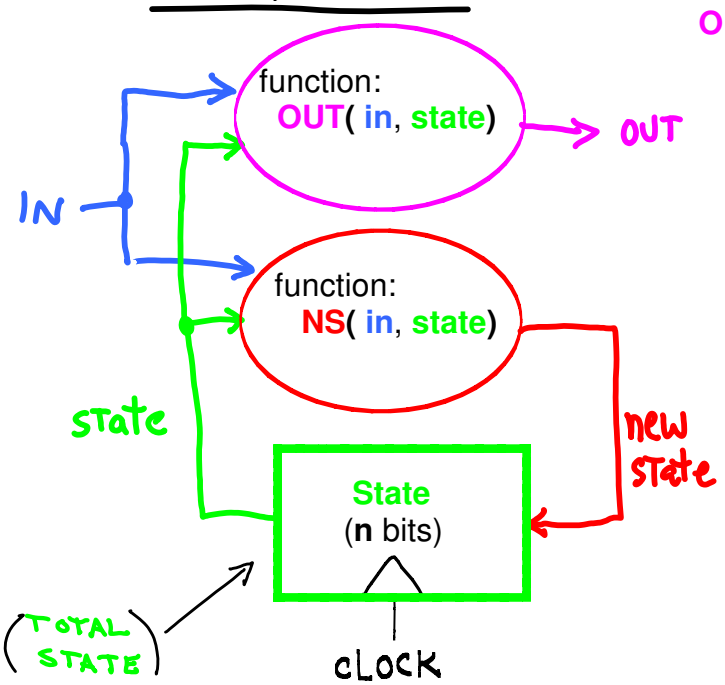
- 2 different "entries" into START state

- 2 different "exits" from HALT state

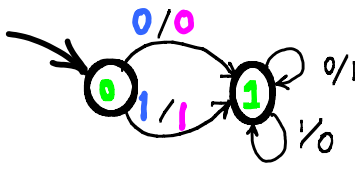
⟹ Reusable sub-"Routine"

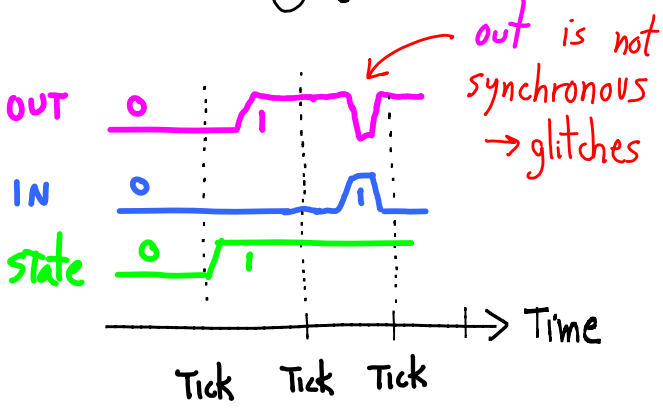
# Implementation of FSM

## Mealy Machine




**OUT** is **continous** function of **IN**+**STATE**.
**STATE changes** w/ clock
**OUT changes** w/ **STATE** and/or **IN**

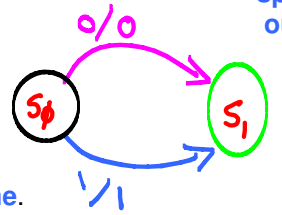


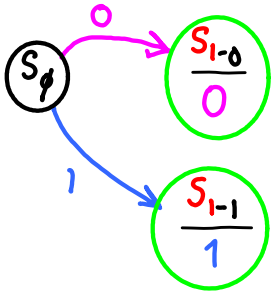

## synchronous output Moore Machine

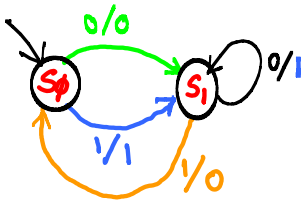

**Output** in **state** is **always the same**.

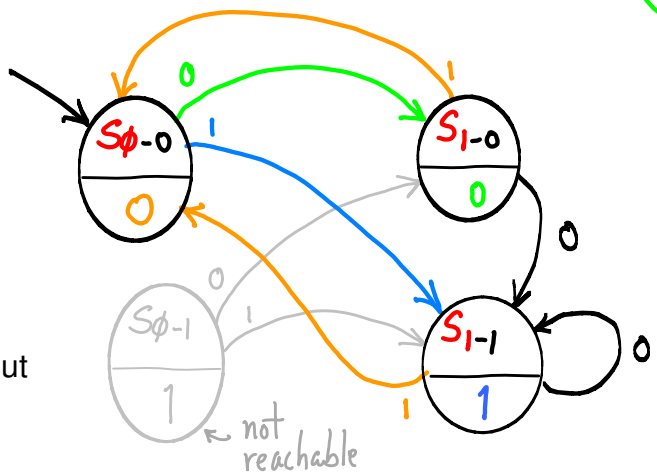
**Output only depends on current state**.




**split states by output** on a transition.

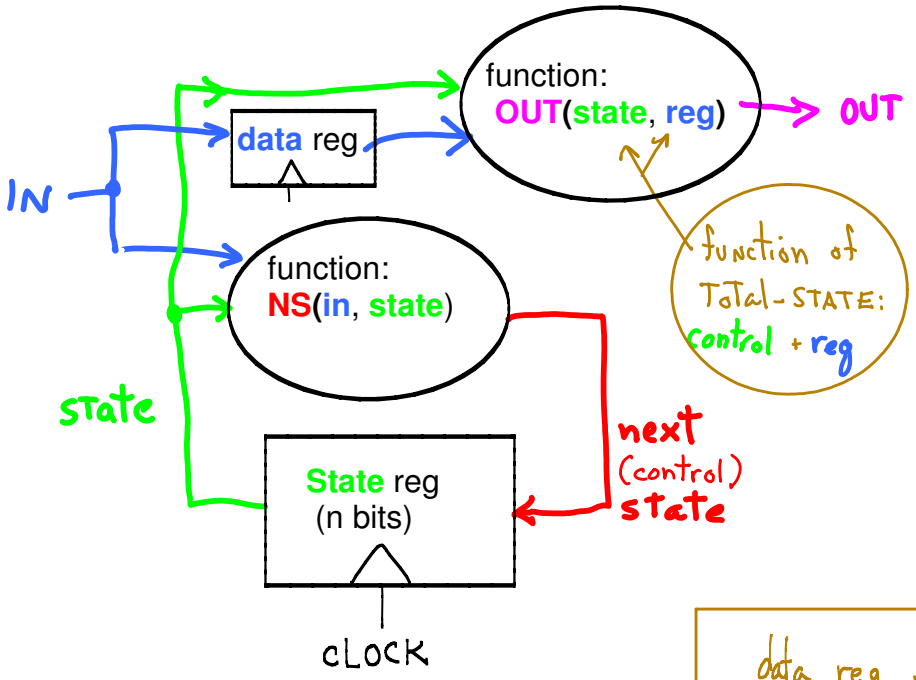



Are they equivalent? Check that same input streams give same output streams.

# Moore Machine




**AT CLOCK**:

-- **data_reg** output **changes**:

**data_reg**.out <== **in**

-- **State_reg** output **changes**:

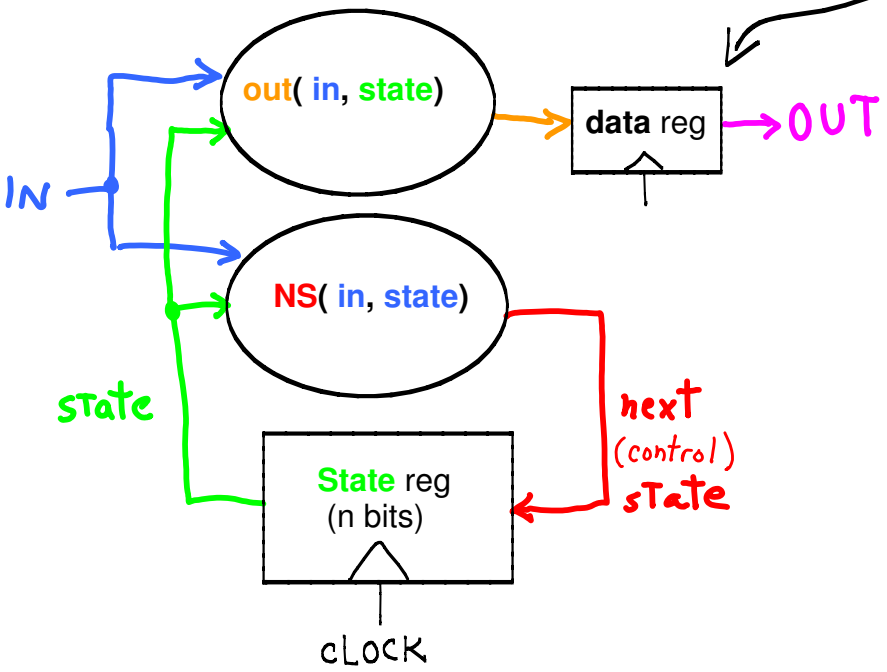
**State_reg**.out <== **NS**(**in**, **state**)

**No changes** until next tick,
**even if input changes**.

data reg + state reg = total state

# Shortcut Mealy-to-Moore Conversion

Add output reg




--- **OUT changes w/ clock** when **total STATE changes**.

--- **out** is a function that **changes w/ in**

--- **OUT** is a function **only** of **total STATE** == (**control + data registers**)

**In** changes, **OUT is steady**.

Note: **OUT** is **determined** by **previous** (**STATE + in**).

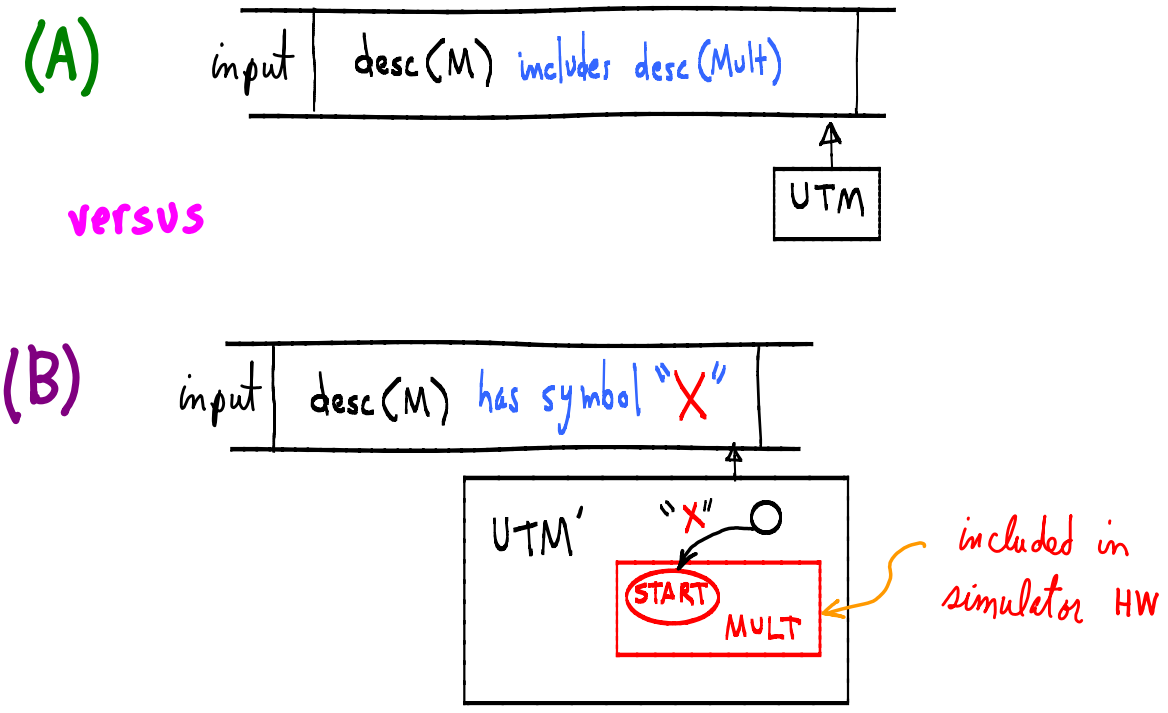
**BIG IDEA:** **Extend simulator** with **additional hardware** (hardware subroutines).

--- **(A)** simulated **M**'s **description** has **sub-routine, desc( MULTIPLY )**

--- versus **(B) add a symbol "X"** to **M**'s **description**

**branch to a hardware MULTIPLY** subroutine:
===> **New Simulator is FASTER**,
===> **Desc(M) is SMALLER**: desc( **MULTIPLY** ) **is gone**

--- **Software == Hardware**




---

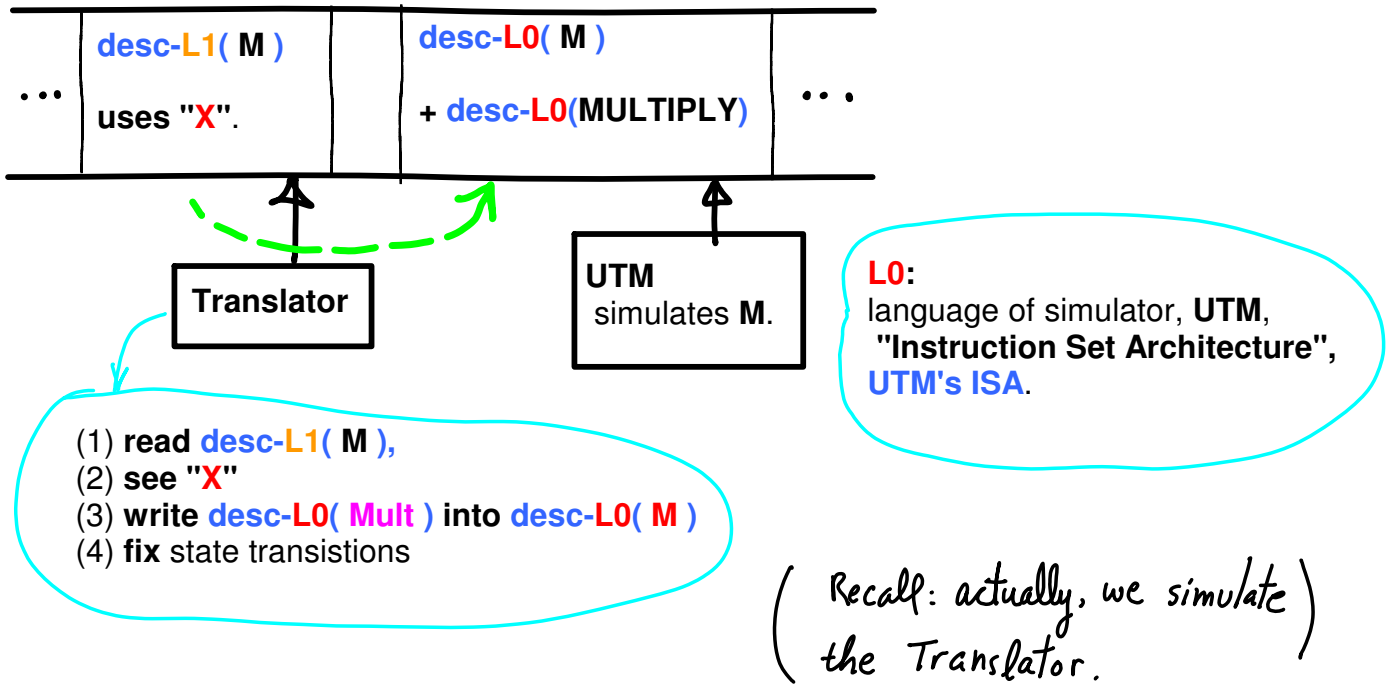
## Hierarchical Translation variant of (A)

Could we **extend desc(M)** in the same way?

**"X" in desc( M )**

Translator **adds desc(MULTIPY)** to **desc(M)**?

===> **Libraries**, code inserted where referenced.




--- **Add layers** of translation

**scripting language** ==> C++ ==> C ==> asm ==> ISA

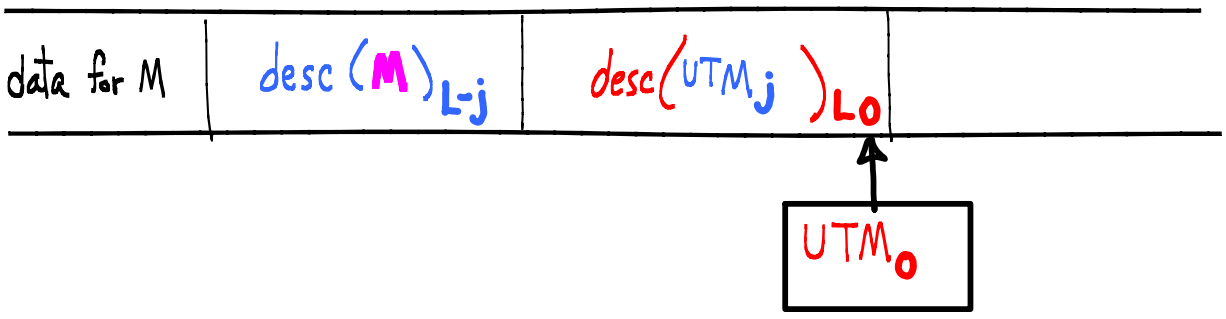
--- **Migrate subroutines down** (maybe **into UTM's hardware**)

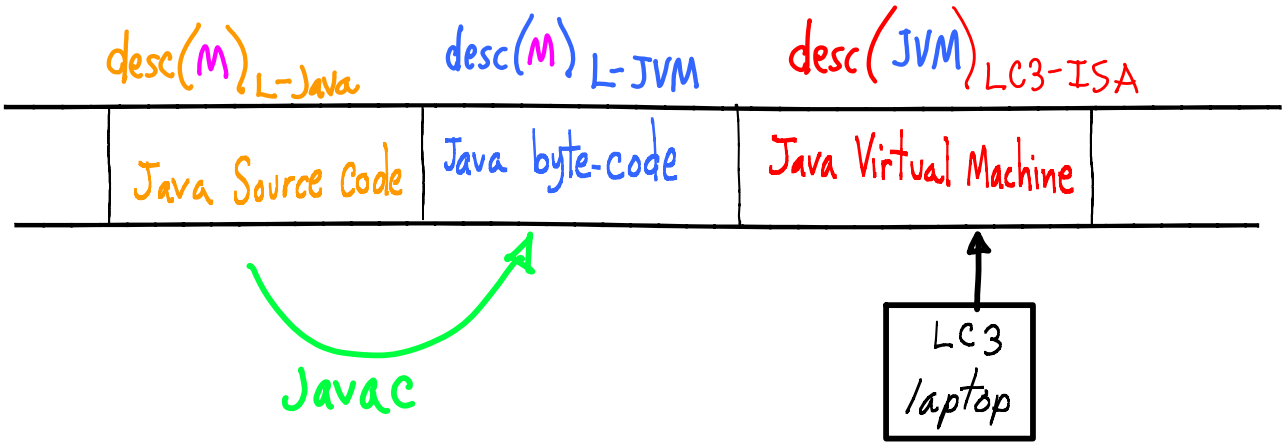
--- **Simulate a different UTM** ==> **interpreted languages** (JAVA bytecode, e.g.)

**simulate desc-L0( UTM-j )**

**UTM-j has its own ISA, L-j.**

**Simulate UTM-j, which simulates M.**






**Let's make things even more exciting, add Heirarchy!**

**symbol "X" in L-Java,**
**===> symbol "X" in L-JVM,**
**===> desc-LC3( TM-X ) ===> executed directly, not via UTM-jvm simulation**
**OR**
**(hardware sub-routine)**