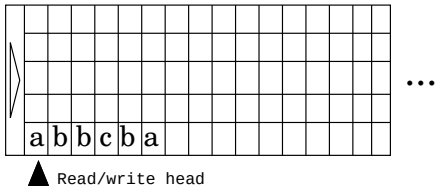
12-0: **Extending Turing Machines**

- When we added a stack to NFA to get a PDA, we increased computational power
- Can we do the same thing for Turing Machines?
  - That is, can we add some new "feature" to TMs that will increase their computational power?

12-1: **Multi-Track Tape**

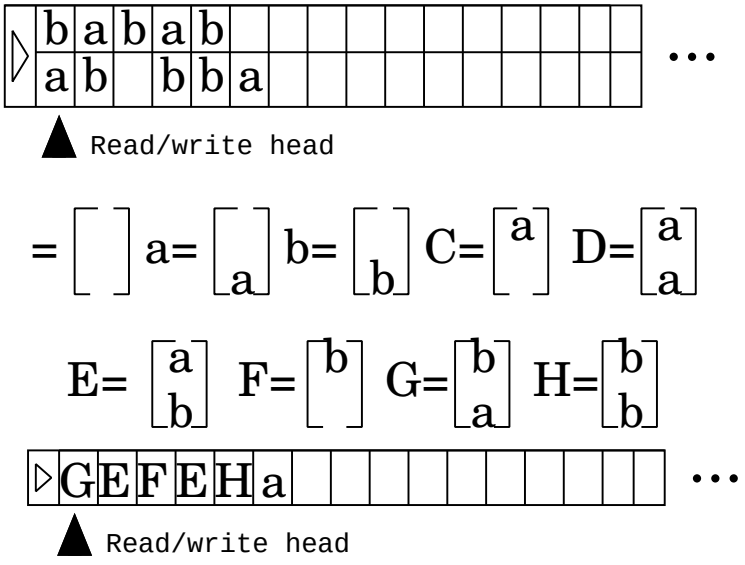
- Instead of each tape location holding a single symbol, we add several "tracks" to the tape
  - Based on contents of all tracks, either move head left, move head right, or write new values to any of the tracks



12-2: **Multi-Track Tape**

- Can simulate a mutli-track machine with a standard TM
  - Increase the size of the tape alphabet
  - k tracks, each with an alphabet of n symbols
  - New alphabet of size n^k

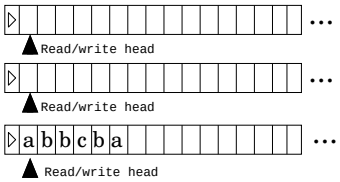
12-3: **Multi-Track Tape**



12-4: **Multiple Tapes**

- Several tapes, with independent read/write heads
- Reach symbol on each tape, and based on contents of all tapes:

- Write or move each tape independently
- Transition to new state



12-5: **Multiple Tapes**

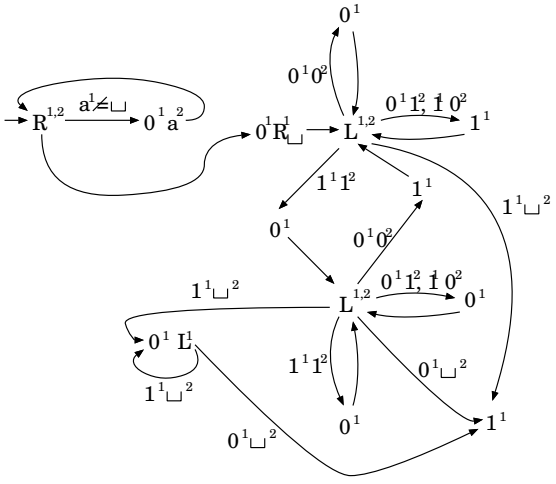
- Create a 2-Tape Machine that adds two numbers
  - Convert $\triangleright\underline{\sqcup}w; v$ to $\triangleright\underline{\sqcup}w + v$ (leading zeros OK)
- Assume that tape 1 holds input (and output), and tape 2 starts out with blanks

12-6: **Multiple Tapes**

- Create a 2-Tape Machine that adds two numbers
  - Convert $\triangleright\underline{\sqcup}w; v$ to $\triangleright\underline{\sqcup}w + v$ (leading zeros OK)
- Copy first # to second tape (zeroing out first # on first tape)
- Do "standard addition", keeping track of carries.

12-7: **Multiple Tapes**

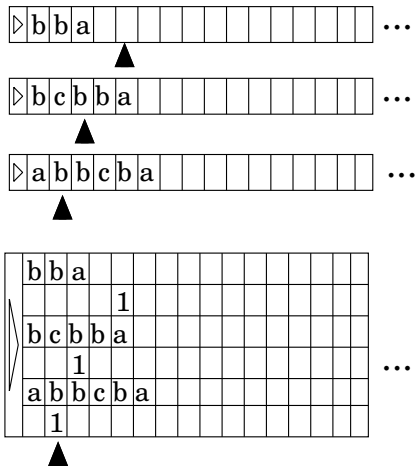
- Create a 2-Tape Machine that adds two numbers



12-8: **Multiple Tapes**

- Are k-tape machines more powerful than 1-tape machines?

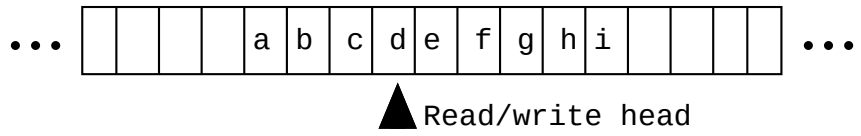
12-9: **Multiple Tapes**



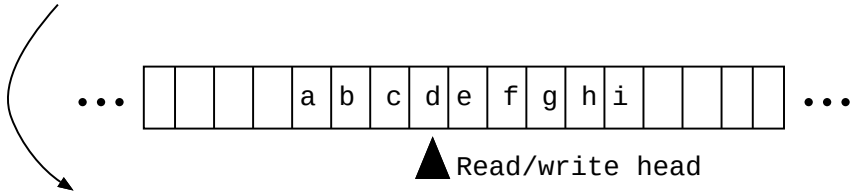
12-10: **Multiple Tapes**

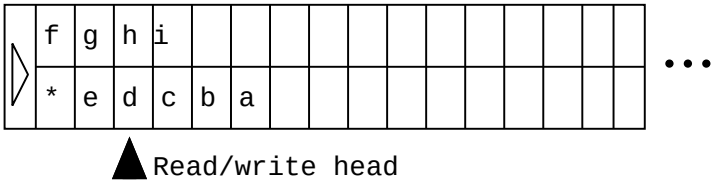
- Each transition from the original, multi-tape machine will require several transitions from the simulated machine – and each state in the multiple-tape machine will be represented by a set of states in the simulation machine
  - First, need to scan tape head to find all "virtual heads", and remember what symbol is stored at each head location
    - Use state to store this information
  - Next, scan tape to implement the action on each tape (moving head, rewriting symbols, etc)
  - Finally, transition to a new set of states

12-11: **2-Way Infinite Tape**



12-12: **2-Way Infinite Tape**





12-13: **2-Way Infinite Tape**

- Make 2 copies of states in original machine: One set for top tape, one set for bottom tape
- Top Tape States
  - Use the top track
  - Execute as normal
  - When "Move Left" command, and beginning of tape symbol is on the bottom tape, move Right instead, switch to Bottom Tape States

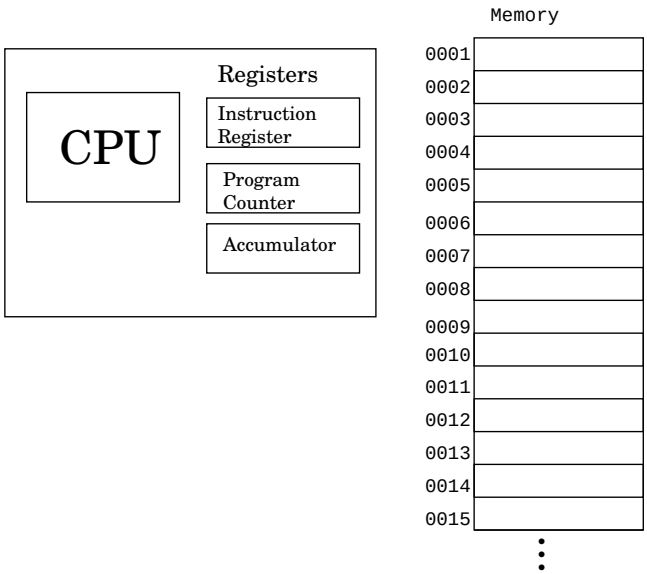
12-14: **2-Way Infinite Tape**

- Make 2 copies of states in original machine: One set for top tape, one set for bottom tape
- Bottom Tape States
  - Use the bottom track
  - Move left on a "Move Right" command, move right on a "Move Left" command
  - When the beginning of tape symbol is encountered, switch to Top Tape States

12-15: **Simple Computer**

- CPU
- 3 Registers (Instruction Register (IR), Program Counter (PC), Accumulator (ACC)
- Memory
- Operation:
  - Set IR $\rightarrow$ MEM[PC]
  - Increment PC
  - Execute instruction in IR
  - Repeat

12-16: **Simple Computer**



12-17: **Simple Computer**

	Instruction	Meaning
00	HALT	Stop Computation
01	LOAD x	ACC ← MEM[x]
02	LOADI x	ACC ← x
03	STORE x	MEM[x] ← AC
04	ADD x	ACC ← ACC + MEM[x]
05	ADDI x	ACC ← ACC + x
06	SUB x	ACC ← ACC - MEM[x]
07	SUBI x	ACC ← ACC - x
08	JUMP x	IP ← x
09	JZERO x	IP ← x if ACC = 0
10	JGT x	IP ← x if ACC $>$ 0

12-18: Write a program that multiplies two numbers (in locations 1000 & 1001), and stores the result in 1002

Simple Computer

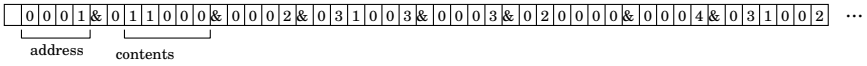
Memory	Machine Code	Assembly
0001	011000	LOAD 1000
0002	031003	STORE 1003
0003	020000	LOADI 0
0004	031002	STORE 1002
0005	021003	LOAD 1003
0006	090012	JZERO 0012
0007	070001	SUBI 1
0008	031003	STORE 1003
0009	011002	LOAD 1002
0010	041001	ADD 1001
0011	080004	STORE 1002
0012	000000	HALT

12-19: **Computers & TMs**

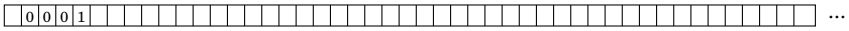
- We can simulate this computer with a multi-tape Turing machine:
  - One tape for each register (IR, IP, ACC)
  - One tape for the Memory
    - Memory tape will be entries of the form <address> <contents>

12-20: **Computers & TMs**

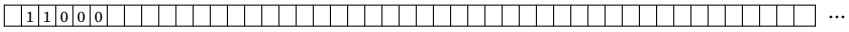
Memory



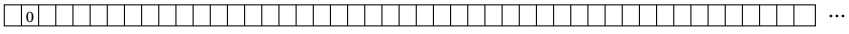
Instruction Pointer



Instruction Register



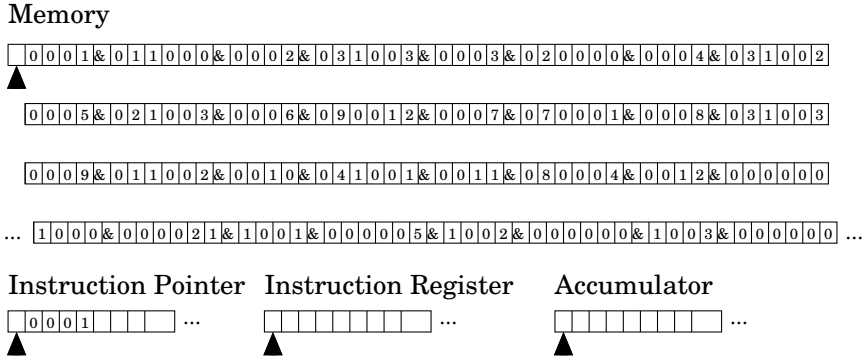
Accumulator



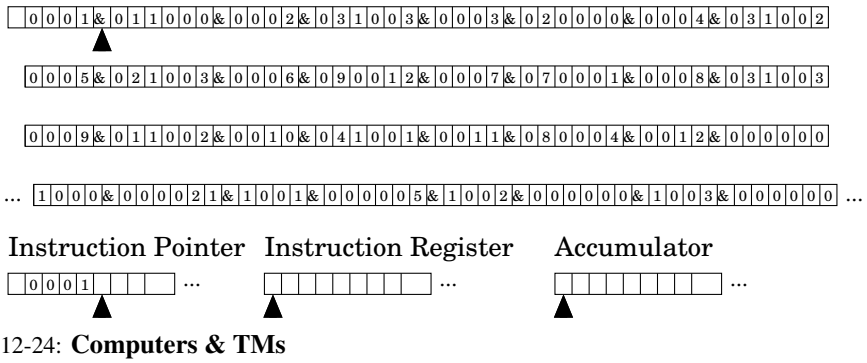
12-21: **Computers & TMs**

- Operation:
  - Scan through memory until reach an address that matches the IP
  - Copy contents of memory at that address to the IR
  - Increment IP
  - Based on the instruction code:
    - Copy value into IP
    - Copy a value into Memory
    - Copy a value into the ACC
    - Do addition/subtraction

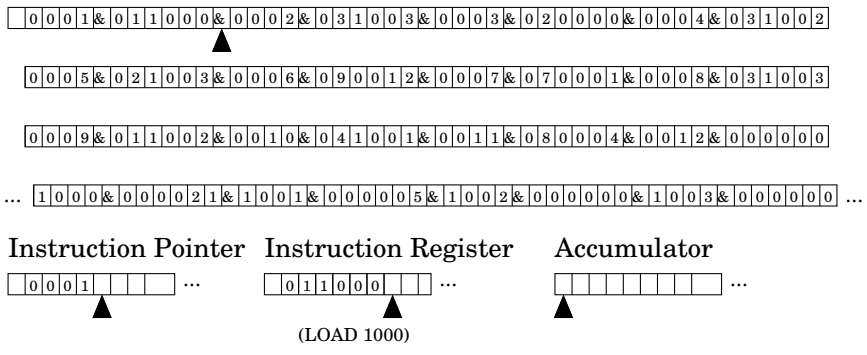
12-22: **Computers & TMs**



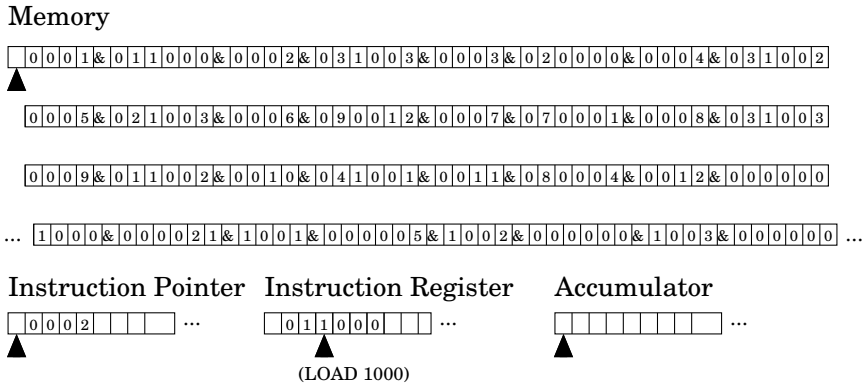
12-23: **Computers & TMs**



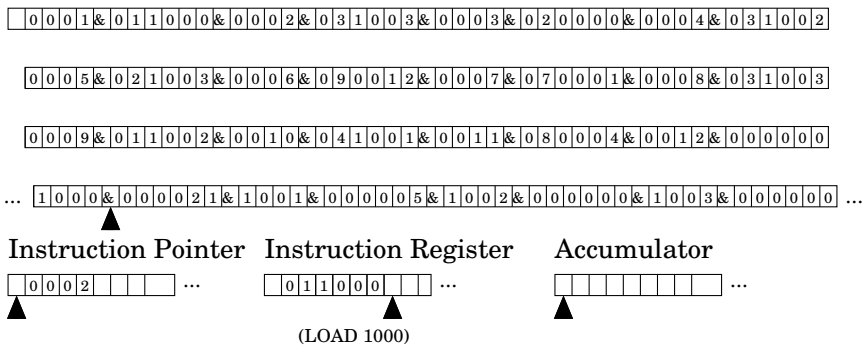
12-24: **Computers & TMs**



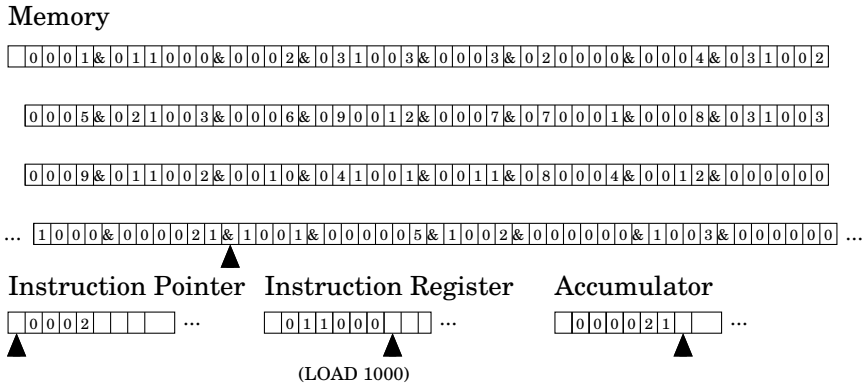
12-25: **Computers & TMs**



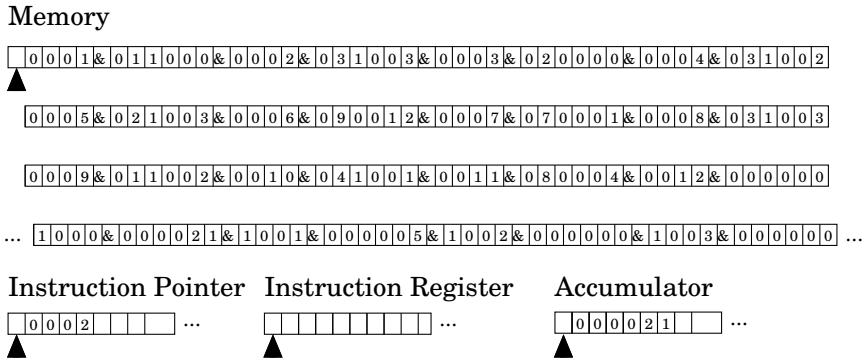
12-26: **Computers & TMs**



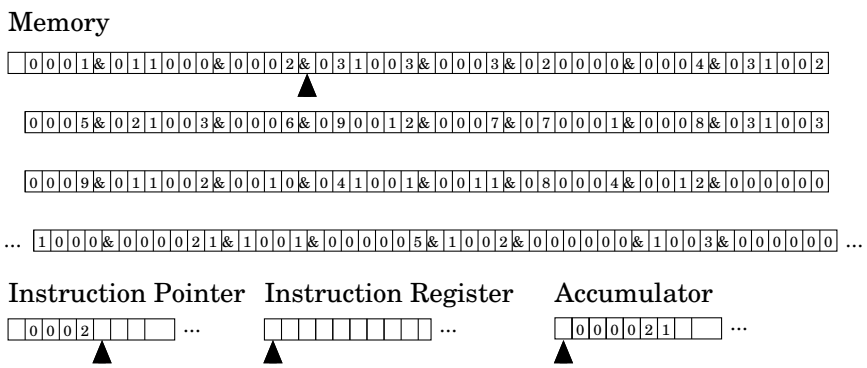
12-27: **Computers & TMs**



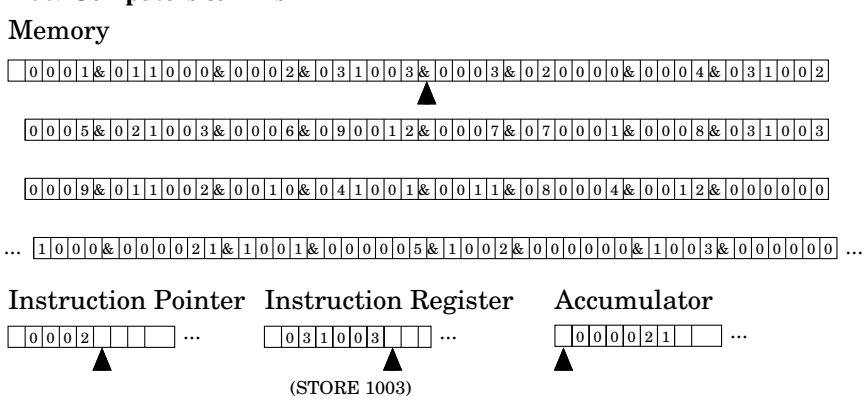
12-28: **Computers & TMs**



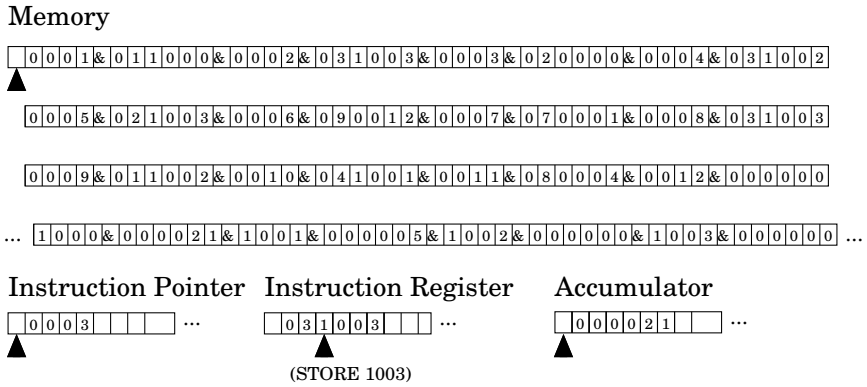
12-29: **Computers & TMs**



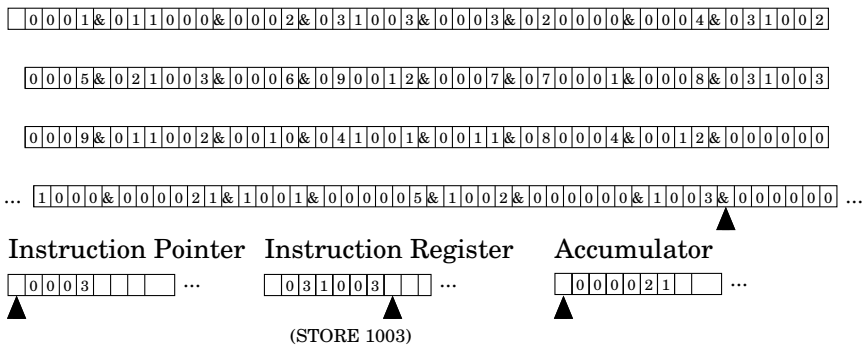
12-30: **Computers & TMs**



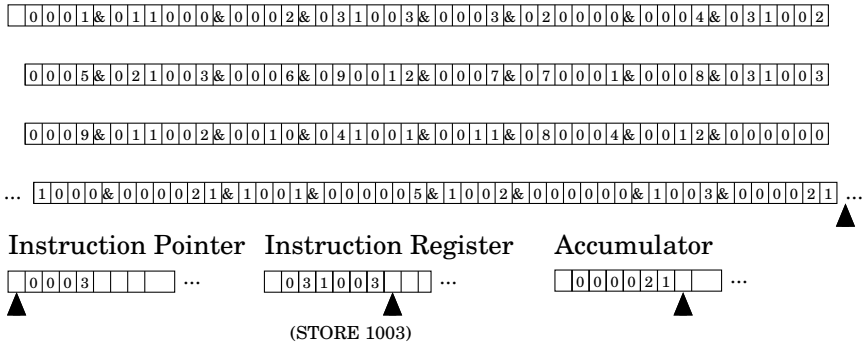
12-31: **Computers & TMs**



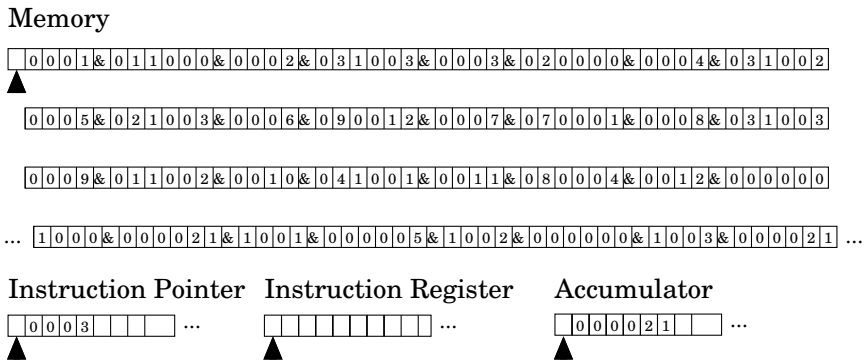
12-32: **Computers & TMs**



12-33: **Computers & TMs**



12-34: **Computers & TMs**



12-35: **Computers & TMs**

- "Simple Computer" can be modeled by a Turing Machine
- Any current machine can be modeled in the same way by a Turing Machine
- If there is an algorithm for it, a Turning Machine can do it
  - Note that at this point, we don't care *how long* it might take, just that it can be done

12-36: **Turing Complete**

- A computation formalism is "Turing Complete" if it can simulate a Turing Machine
- Turing Complete $\Rightarrow$ can compute anything
  - Of course it might not be convenient ...

12-37: **Non-Determinism**

- Final extension to Turing Machines: Non-Determinism
  - Just like non-determinism in NFAs, PDAs
  - String is accepted by a non-deterministic Turing Machine if there is at least one computational path that accepts

12-38: **Non-Determinism**

A Non-Deterministic Machine M *Decides* a language L if:

- All computational paths halt
- For each $w \in L$, at least one computational path for w accepts
- For all $w \notin L$, no computational path accepts

12-39: **Non-Determinism**

A Non-Deterministic Machine M *Semi-Decides* a language L if:

- For each $w \in L$, at least one computational path for w halts and accepts
- For all $w \notin L$, no computational path halts and accepts

12-40: **Non-Determinism**

A Non-Deterministic Machine M *Computes a Function* if:

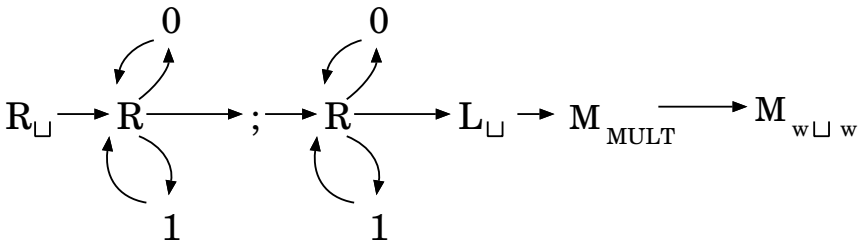
- All computational paths halt
- Every computational path produces the same result

12-41: **Non-Determinism**

- Non-Deterministic TM for $L = \{w \in \{0,1\} : w \text{ is composite }\}$
- (semi-decides is OK)

12-42: **Non-Determinism**

- Non-Deterministic TM for $L = \{w \in \{0,1\} : w \text{ is composite }\}$



How could we make this machine decide (instead of semi-decide) L? 12-43: **Non-Determinism**
How we can make this machine decide (instead of semi-decide) L

- First, transform w into $w \sqcup w; w$
- Non-deterministically modify the second 2 w's
- Multiply the second 2 w's
- Check to see if the resulting string is $w \sqcup w$

12-44: **Non-Determinism**

- Are Non-Deterministic Turing Machines more powerful than Deterministic Turing machines?
  - Is there some L which can be semi-decided by a non-deterministic Turing Machine, which cannot be semi-decided by a Deterministic Turing Machine?
- Non-determinism in Finite Automata didn't buy us anything
- Non-determinism in Push-Down Automata did

12-45: **Non-Determinism**

- How to Simulate a Non-Deterministic Turing Machine with a Deterministic Turing Machine

12-46: **Non-Determinism**

- How to Simulate a Non-Deterministic Turing Machine with a Deterministic Turing Machine
  - Try one computational path – if it says yes, halt and say yes. Otherwise, try a different computational path. Repeat until success

12-47: **Non-Determinism**

- How to Simulate a Non-Deterministic Turing Machine with a Deterministic Turing Machine
  - Try one computational path – if it says yes, halt and say yes. Otherwise, try a different computational path. Repeat until success
    - But what if the first computational path runs forever . . .

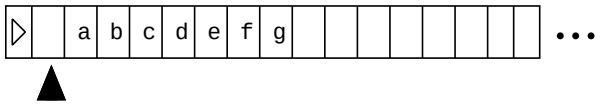
12-48: **Non-Determinism**

- How to Simulate a Non-Deterministic Turing Machine with a Deterministic Turing Machine
  - Try all computational paths of length 1
  - Try all computational paths of length 2
  - Try all computational paths of length 3

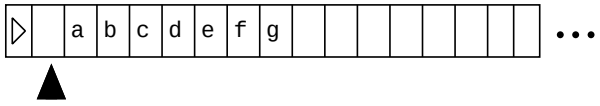
    . . .
- If there is a halting configuration, you will find it eventually. Otherwise, run forever.

12-49: **Non-Determinism**

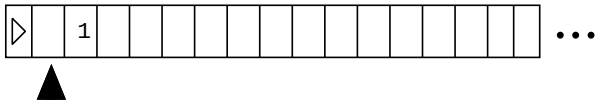
Original Tape



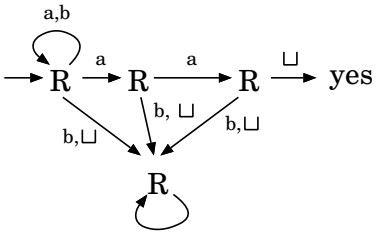
Work Tape



Control Tape

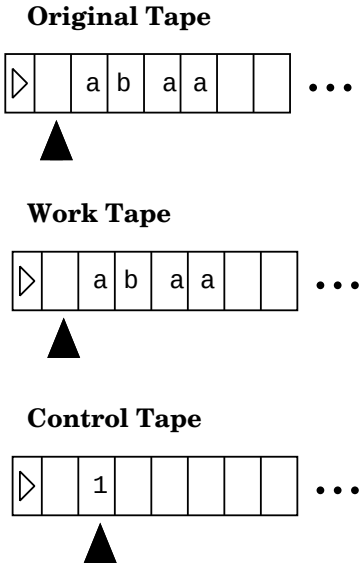


12-50: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

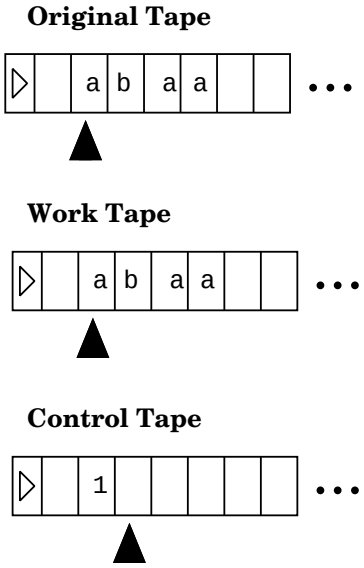
12-51: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q0

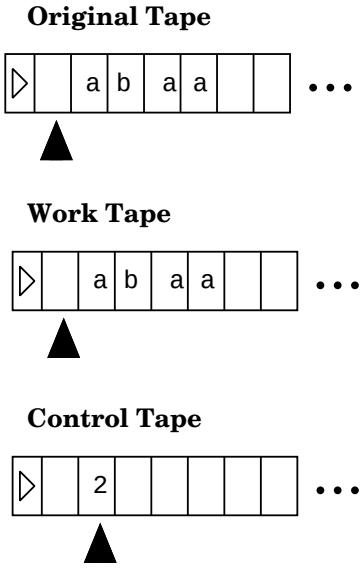
12-52: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q1

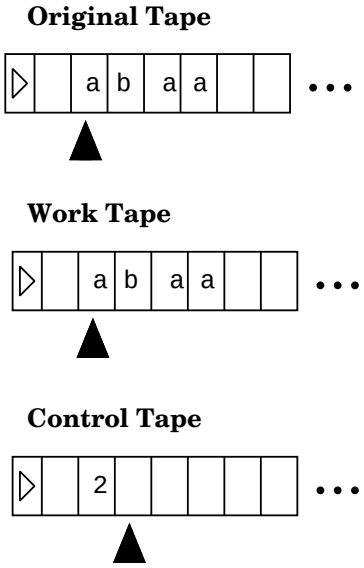
12-53: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q0

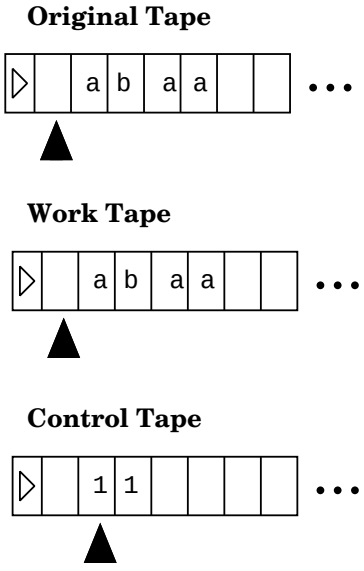
12-54: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q1

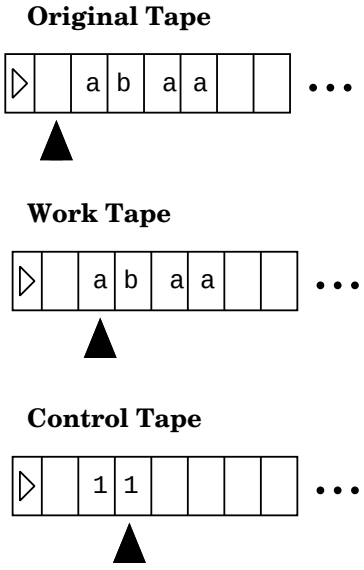
12-55: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q0

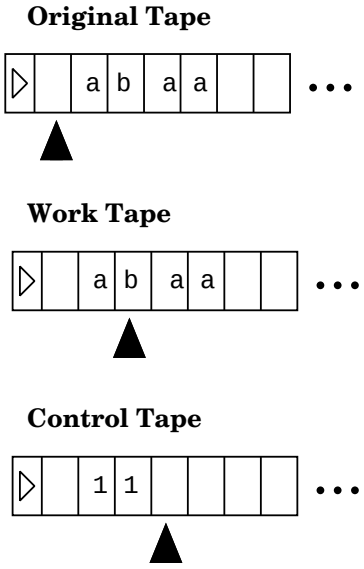
12-56: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q1

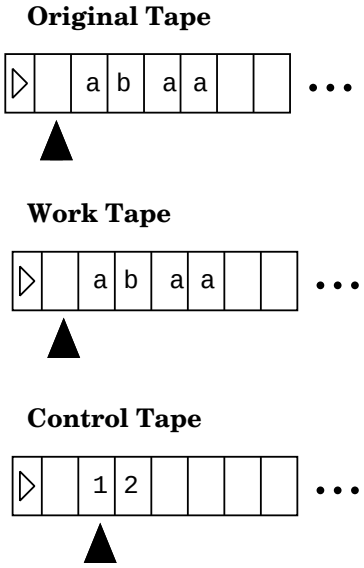
12-57: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q1

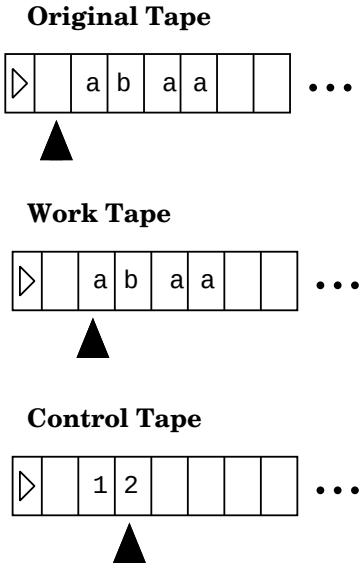
12-58: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q0

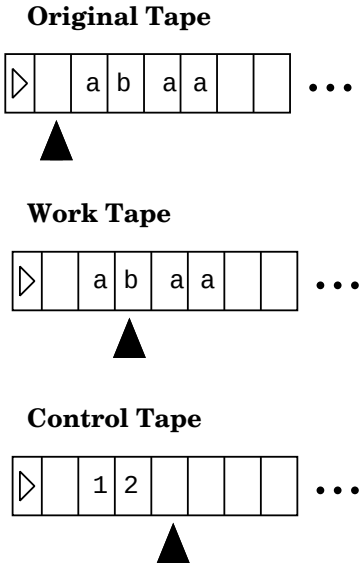
12-59: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q1

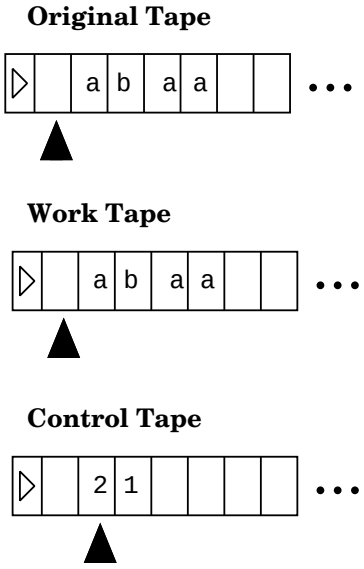
12-60: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q2

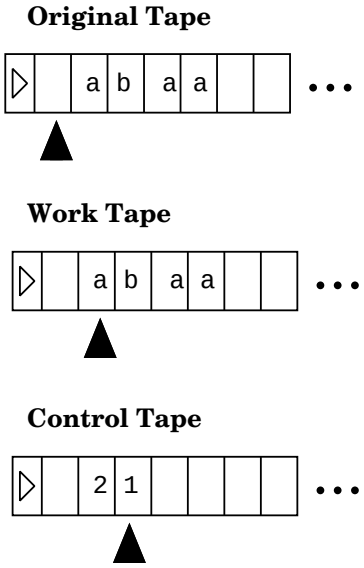
12-61: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q0

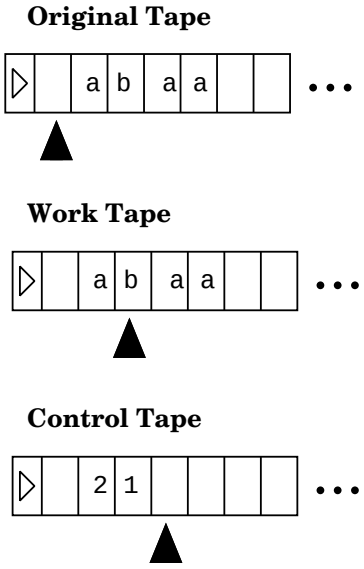
12-62: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q1

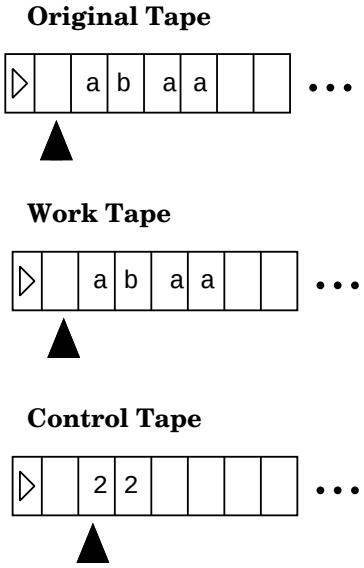
12-63: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q1

12-64: **Non-Determinism**

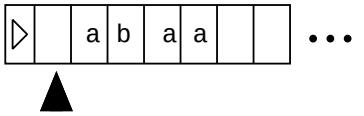


	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

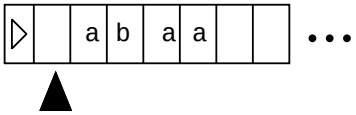
State: q0

12-65: **Non-Determinism**

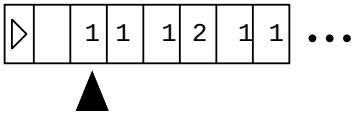
Original Tape



Work Tape



Control Tape

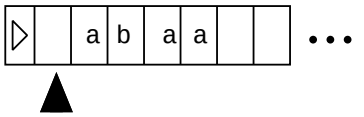


	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

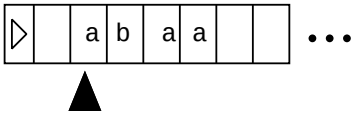
State: q0

12-66: **Non-Determinism**

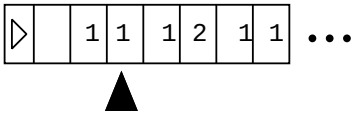
Original Tape



Work Tape



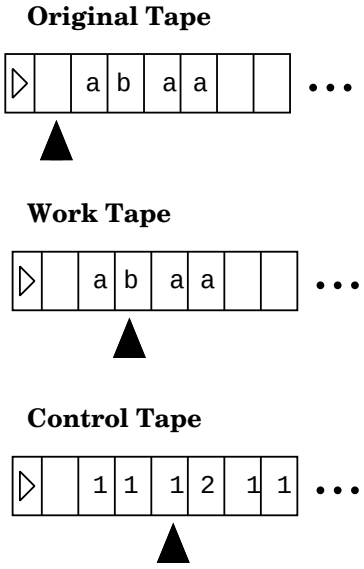
Control Tape



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q1

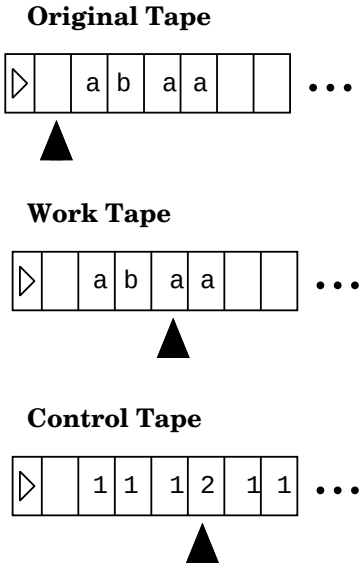
12-67: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q1

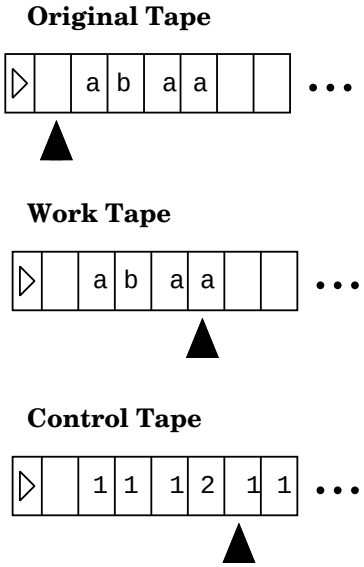
12-68: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q1

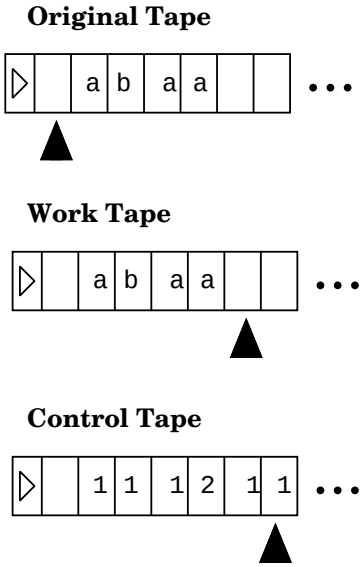
12-69: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q2

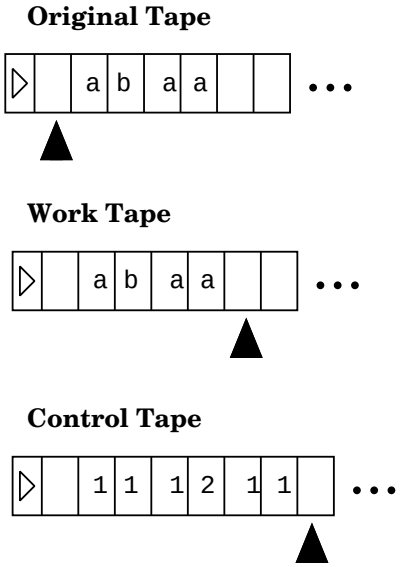
12-70: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

State: q3

12-71: **Non-Determinism**



	a	b	␣
q0	(q1,R)	(q1,R)	(q1,R)
q1	(q1,R) (q2,R)	(q1,R)	(q4,R)
q2	(q3,R)	(q4,R)	(q4,R)
q3	(q4,R)	(q4,R)	yes
q4	(q4,R)	(q4,R)	(q4,R)

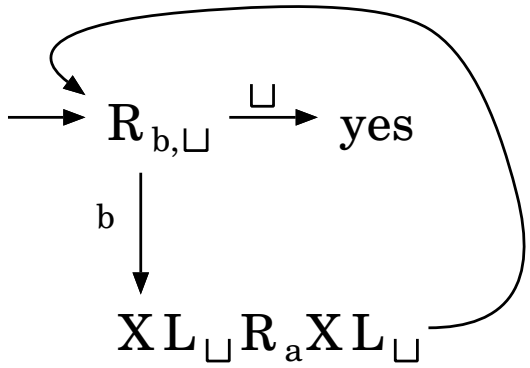
State: yes

12-72: **Turing Machines**

- Some Turing Machine review problems:
  - Create a Turing Machine that *semi-decides* the language L = all strings over {a, b} with at least as many a's as b's

12-73: **Turing Machines**

- Create a Turing Machine that *semi-decides* the language L = all strings over {a, b} with at least as many a's as b's



12-74: **Turing Machines**

- Some Turing Machine review problems:
  - Create a Turing Machine that computes the function $\lceil \lg x \rceil$, where x is a binary number

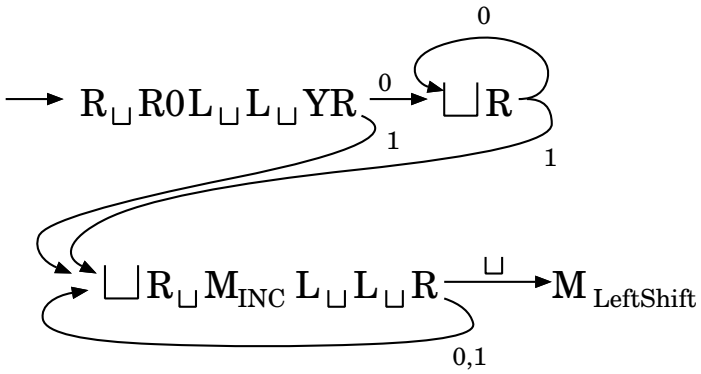
12-75: **Turing Machines**

- Some Turing Machine review problems:
  - Create a Turing Machine that computes the function $\lceil \lg x \rceil$, where x is a binary number

- Set result to 0
- While $x \leq 2$, divide x by 2, and add one to the result

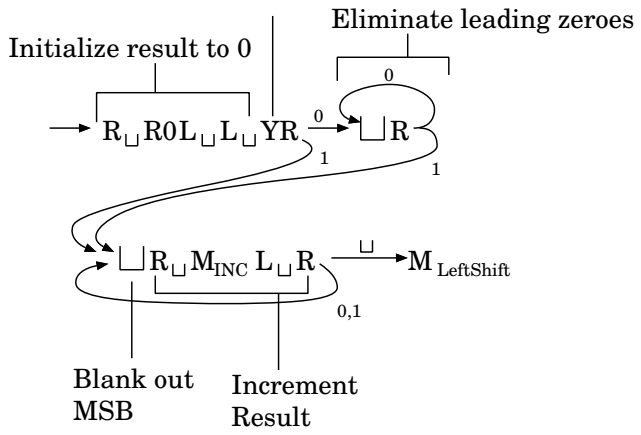
12-76: **Turing Machines**

- Create a Turing Machine that computes the function $\lceil \lg x \rceil$, where x is a binary number



12-77: **Turing Machines**

- Create a Turing Machine that computes the function $\lceil \lg x \rceil$, where x is a binary number



12-78: **Turing Machines**