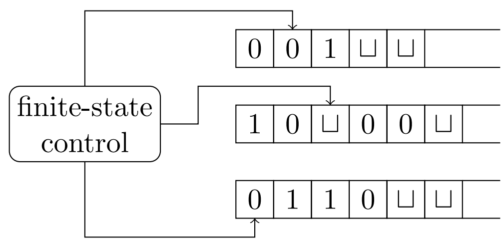
# 1 Variants of Turing Machines

## 1.1 Multi-Tape TM

### Multi-Tape Turing Machine




- Input on Tape 1
- Initially all heads scanning cell 1, and tapes 2 to $k$ blank
- In one step: Read symbols under each of the $k$-heads, and depending on the current control state, write new symbols on the tapes, move the each tape head (possibly in different directions), and change state.

---

### Multi-Tape Turing Machine

*Formal Definition*

A $k$-tape Turing Machine is $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$ where

- $Q$ is a finite set of control states
- $\Sigma$ is a finite set of input symbols
- $\Gamma \supseteq \Sigma$ is a finite set of tape symbols. Also, a blank symbol $\sqcup \in \Gamma \setminus \Sigma$
- $q_0 \in Q$ is the initial state
- $q_{\mathsf{acc}} \in Q$ is the accept state
- $q_{\mathsf{rej}} \in Q$ is the reject state, where $q_{\mathsf{rej}} \neq q_{\mathsf{acc}}$
- $\delta : (Q \setminus \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma^k \to Q \times (\Gamma \times \{\mathsf{L}, \mathsf{R}\})^k$ is the transition function.

---

### Computation, Acceptance and Language

- A configuration of a multi-tape TM must describe the state, contents of all $k$-tapes, and positions of all $k$-heads. Thus, $\textsc{c} \in Q \times (\Gamma^*\{*\}\Gamma\Gamma^*)^k$, where $*$ denotes the head position.

- Accepting configuration is one where the state is $q_{\mathsf{acc}}$, and starting configuration on input $w$ is $(q_0, *w, *\sqcup, \ldots, *\sqcup)$
- Formal definition of a single step is skipped.
- $w$ is *accepted* by $M$, if from the starting configuration with $w$ as input, $M$ reaches an accepting configuration.
- $\mathbf{L}(M) = \{w \mid w \text{ accepted by } M\}$

---

**Expressive Power of multi-tape TM**

**Theorem 1.** *For any $k$-tape Turing Machine $M$, there is a single tape TM single$(M)$ such that* $\mathbf{L}(\mathit{single}(M)) = \mathbf{L}(M)$.

**Challenges**

- How do we store $k$-tapes in one?
- How do we simulate the movement of $k$ independent heads?

---

**Storing Multiple Tapes**

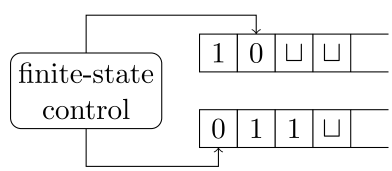



Figure 1: Multi-tape TM $M$

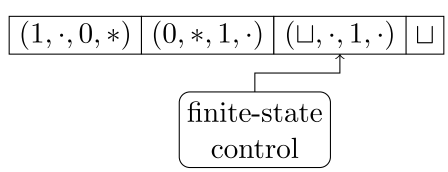



Figure 2: 1-tape equivalent single($M$)

Store in cell $i$ contents of cell $i$ of all tapes. "Mark" head position of tape with $*$.

---

**Simulating One Step**

Challenge 1: Head of 1-Tape TM is pointing to one cell. How do we find out all the $k$ symbols that are being read by the $k$ heads, which maybe in different cells?

- Read the tape from left to right, storing the contents of the cells being scanned in the *state*, as we encounter them.

Challenge 2: After this scan, 1-tape TM knows the next step of $k$-tape TM. How do we change the contents and move the heads?

- Once again, scan the tape, change all relevant contents, "move" heads (i.e., move $*$s), and change state.

---

**Overall Construction**
First we outline the high-level algorithm for the 1-tape TM. On input $w$

1. First the machine will rewrite input $w$ to be in "new" format.
2. To simulate one step
   - Read from left-to-right remembering symbols read on each tape, and move all the way back to leftmost position.
   - Read from left-to-right, changing symbols, and moving those heads that need to be moved right.
   - Scan back from right-to-left moving the heads that need to be moved left.

Formally, let $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$. To define the machine single($M$) it is useful to identify modes that single($M$) could be in while simulating $M$. The modes are

$$\text{mode} = \{\text{init}, \text{back-init}, \text{read}, \text{back-read}, \text{fix-right}, \text{fix-left}\}$$

where

- init means that the machine is rewriting input in new format
- back-init means the machine is just going back all the way to the leftmost cell after "initiallizing" the tape
- read means the machine is scanning from left to right to read all symbols being read by $k$-tape machine
- back-read means the machine is going back to leftmost cell after the "read" phase
- fix-right means the machine is scanning from left to right and is going to make all tape changes and move those heads that need to be moved right
- fix-left means the machine is scanning right to left and moving all heads that need to be moved left

Now single($M$) $= (Q', \Sigma', \Gamma', \delta', q'_0, q'_{\mathsf{acc}}, q'_{\mathsf{rej}})$ where

- Recall, based on the high-level description, single($M$) needs to remember a few things in its state. It needs to keep track of the current "mode"; $M$s state; during the read phase the symbols being scanned by each head of $M$; at the end of the read phase, the new symbols to be written and direction to move the heads. Thus,

  $$Q' = \{q'_0, q'_{\mathsf{acc}}, q'_{\mathsf{rej}}\} \cup (\text{modes} \times Q \times (\Gamma \times \{\mathsf{L}, \mathsf{R}, *\})^k)$$

  where $q'_0, q'_{\mathsf{acc}}, q'_{\mathsf{rej}}$ are new initial, accept and reject states, respectively. "$*$" is new special symbol that we will use to when placing new head positions, and can be ignored for now. Intuitively, when the mode is "read" the directions don't mean anything, and symbols in $\Gamma$ will be the symbols that $M$ is scanning. During the "fix" phases the directions are the directions each head needs to be moved, and the symbols are the new symbols to be written.

- $\Sigma' = \Sigma$; the input alphabet does not change

- On the tape, we write the contents of one cell of each of the $k$-tapes and whether the head scans that position or not. Thus, $\Gamma' = \{\rhd, \sqcup\} \cup (\Gamma \times \{\cdot, *\})^k$, where $\rhd$ will be a new left-end-marker, as it will be useful for single($M$) to know when it has finished scanning all the way to the left. $\sqcup$ as always is the blank symbol of the machine.

- The initial state, accept state and reject states are the new states $q'_0, q'_{\mathsf{acc}}$, and $q'_{\mathsf{rej}}$.

We will now formally define the transition function $\delta$. We will describe $\delta$ for various cases below; for a case not covered below, we will assume our usual convention that the machine single($M$) goes to the reject state $q'_{\mathsf{rej}}$ and moves the head left.

**Initial State** In the first step, single($M$) will move to the "initialization phase", which will write a (new) left endmarker, and rewrite the tape in the correct format for the future. Thus, from initial state $q'_0$ you go to a state whose "mode" is init. Since we are going to insert a new left end-marker, we need to "shift" all symbols of the input one-space to right, which can be accomplished by remembering the next symbol to be written in the state. So $\delta'(q'_0, a) = (\langle \text{init}, q_0, a, *, 0, \mathsf{L}, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, \rhd)$; the symbols 0 and $\mathsf{L}$ don't mean anything (and so can be changed to whatever you please), and the $*$ remembers that when we write the next symbol all heads must be in that position.

**Initialization** In the initialization phase, we just read a symbol and write it in the "new format", which means writing blank symbols for all the other tape cells, and moving right. When we scan the entire input to go back left, i.e., change mode to back-init. There are two caveats to this. First we are shifting symbols of the input one position to the right because of the left endmarker $\rhd$; so we actually write what we remembered in our state, and remember what we read in the state. Also, in the first position, we need to "place" all the heads; this is remembered because of $*$. So we have

$$\delta'(\langle \text{init}, q_0, a, *, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, b) = (\langle \text{init}, q_0, b, \mathsf{L}, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, (a, *, \sqcup, *, \ldots, \sqcup, *), \mathsf{R})$$
$$\delta'(\langle \text{init}, q_0, a, \mathsf{L}, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, b) = (\langle \text{init}, q_0, b, \mathsf{L}, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, (a, \cdot, \sqcup, \cdot, \ldots, \sqcup, \cdot), \mathsf{R})$$
$$\delta'(\langle \text{init}, q_0, a, \mathsf{L}, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, \sqcup) = (\langle \text{back-init}, q_0, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, (a, \cdot, \sqcup, \cdot, \ldots, \sqcup, \cdot), \mathsf{L})$$

**Ending Initialization** After we have rewritten the tape, we move the head all the way back, and move to the next phase which is "reading". Here, the fact that we wrote a left end-marker will be useful in realizing, when we have gone all the way back. So formally,

$$\delta'(\langle \text{back-init}, q_0, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, b) = (\langle \text{back-init}, q_0, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, b, \mathsf{L})$$
$$\delta'(\langle \text{back-init}, q_0, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, \rhd) = (\langle \text{read}, q_0, 0, \mathsf{L}, \ldots, 0, \mathsf{L}\rangle, \rhd, \mathsf{R})$$

where $b \neq \rhd$

**Reading** Here we scan the tape to the right, and whenever we encounter a position, where there is a tape head (i.e., a $*$ in the appropriate position), we will remember that symbol in the state. When we reach the right end (i.e., read a $\sqcup$), we know all the information to determine the next step of $M$. We will remember the new symbols to right and directions of the head in the state, and move to the next phase back-read where we just go back all the way to the left end. These two cases are formally given as

- Suppose the current state is $P = \langle \text{read}, q, a_1, d_1, \ldots, a_i, d_i, \ldots, a_k, d_k\rangle$, and we read a symbol $X = (b_1, h_1, \ldots, b_i, h_i, \ldots, b_k, h_k)$, where if $h_i = *$ that means that the $i$th tape head is read this position, and if $h_i = \cdot$ then the $i$th tape head is not reading this position. Thus,

  $$\delta'(P, X) = (\langle q, a'_1, d_1, \ldots, a'_i, d_i, \ldots, a'_k, d_k\rangle, X, \mathsf{R})$$

  where $a'_i = a_i$ if $h_i = \cdot$ and $a'_i = b_i$ if $h_i = *$.
- Suppose the current state is $P = \langle \text{read}, q, a_1, d_1, \ldots, a_i, d_i, \ldots, a_k, d_k\rangle$, and we read $\sqcup$. This means we have finished scanning and all the symbols $a_i$ are the symbols that are being read by $M$, and its state is $q$. So suppose $M$'s transition function $\delta$ says

  $$\delta(q, a_1, a_2, \ldots, a_k) = (q', b_1, d'_1, \ldots b_k, d'_k)$$

  That is, it says "replace symbol $a_i$ on tape $i$ by $b_i$ and move its head in direction $d'_i$". Then

  $$\delta'(P, \sqcup) = (\langle \text{back-read}, q', b_1, d'_1, \ldots b_k, d'_k\rangle, \sqcup, \mathsf{L})$$

So the new state of $M$, symbols to be written and direction of heads in stored in the state and we go back.

**Ending Reading** After reading and determining the next step, we move the head all the way back, and move to the next phase which is fixing the tape to reflect the new situation. Here, the fact that we wrote a left end-marker will be useful in realizing when we have gone all the way back. So formally,

$$\delta'(\langle \text{back-read}, q, a_1, d_1, \ldots, a_k, d_k\rangle, b) = (\langle \text{back-read}, q, a_1, d_1, \ldots, a_k, d_k\rangle, b, \mathsf{L})$$
$$\delta'(\langle \text{back-read}, q, a_1, d_1, \ldots, a_k, d_k\rangle, \rhd) = (\langle \text{fix-right}, q, a_1, d_1, \ldots, a_k, d_k\rangle, \rhd, \mathsf{R})$$

where $b \neq \rhd$.

**Right Scan of Fixing** In this phase we move all the way to the right. Along the way, we change the symbols to new symbols, wherever the old heads were, and move all heads that need to be moved right. To move a head right, we will use "$*$" in the state to remember that the old head position of the tape is in current cell, and it needs to be moved to the next cell. Finally, there is the boundary case of moving the head on some tape to right of the rightmost non-blank symbol on any tape. We will capture these cases formally.

- Let the current state of single($M$) be $P = \langle \text{fix-right}, q, a_1, d_1, \ldots, a_k, d_k \rangle$ and let the symbol being read be $X = (b_1, h_1, \ldots b_k, h_k)$. In such a situation, single($M$) will move to state $P' = \langle \text{fix-right}, q, a_1, d'_1, \ldots, a_k, d'_k \rangle$, move $\mathsf{R}$ and write $X' = (b'_1, h'_1, \ldots b'_k, h'_k)$. $P'$ and $X'$ are determined as follows. If $h_i = *$ (that is, tape $i$'s head was here) and $d_i = \mathsf{R}$ then $b'_i = a_i$ (write new symbol), $h'_i = \cdot$ (new head is not here), and $d'_i = *$ (remember to put head in next cell). If $h_i = *$ and $d_i = \mathsf{L}$ then $b'_i = a_i$ (write new symbol), $h'_i = h_i$ (defer head movement to next phase), and $d'_i = d_i$. If $h_i = \cdot$ and $d_i = *$ (i.e., we remember new head position is here) then $b'_i = b_i$ (don't change symbol), $h'_i = *$ (new head is here), and $d'_i = \mathsf{R}$ (this was the original direction). If $h_i = \cdot$ and $d_i \neq *$ then $b'_i = b_i$ and $d'_i = d_i$.
- Consider the case when the state is $P = \langle \text{fix-right}, q, a_1, d_1, \ldots, a_k, d_k \rangle$, and $\sqcup$ is on the tape. There are two possibilities. If $d_i \neq *$ for every $i$ then we move to state $P' = \langle \text{fix-left}, q, a_1, d_1, \ldots q_k, d_k \rangle$, write $\sqcup$ and move $\mathsf{L}$. On the other hand, suppose there is some $i$ such that $d_i = *$. Then we move to state $P' = \langle \text{fix-right}, q, a_1, d'_1, \ldots a_k, d'_k \rangle$, move $\mathsf{R}$ and write $X' = (b'_1, h'_1, \ldots b'_k, h'_k)$, where $X'$ and $P'$ are given as follows. First $b'_i = \sqcup$ for all $i$. Next, if $d_i = *$ then $h_i = *$ and $d'_i = \mathsf{R}$. On the other hand if $d_i \neq *$ then $h'_i = \cdot$ and $d'_i = d_i$.

**Left Scan of Fixing** In this phase we move all the way to the left, and along the way we move all the head positions that needed to be moved left. These changes are similar to the case of moving heads to the right, except for the case when moving a head left from the leftmost position.

- Let the current state of single($M$) be $P = \langle \text{fix-left}, q, a_1, d_1, \ldots, a_k, d_k \rangle$ and let the symbol being read be $X = (b_1, h_1, \ldots b_k, h_k)$. In such a situation, single($M$) will move to state $P' = \langle \text{fix-left}, q, a_1, d'_1, \ldots, a_k, d'_k \rangle$, move $\mathsf{L}$ and write $X' = (b_1, h'_1, \ldots b_k, h'_k)$. $P'$ and $X'$ are determined as follows. If $h_i = *$ and $d_i = \mathsf{L}$ (that is this tape's head needs to be moved left) then $h'_i = \cdot$ (new head is not here), and $d'_i = *$ (remember to put head in next cell). If $h_i = *$ and $d_i = \mathsf{R}$ then $h'_i = h_i$ and $d'_i = d_i$ (don't do anything since we already handled the right moves). If $h_i = \cdot$ and $d_i = *$ (i.e., we remember new head position is here) then $h'_i = *$ (new head is here), and $d'_i = \mathsf{L}$ (this was the original direction). If $h_i = \cdot$ and $d_i \neq *$ then $h'_i = h_i$ and $d'_i = d_i$.
- Now we consider the case when we finish the left scan, i.e., the symbol being read is $\triangleright$. Let the state be $P = \langle \text{fix-left}, q, a_1, d_1, \ldots, a_k, d_k \rangle$. Now there are two possibilities. Either there are no pending left moves, i.e., $d_i \neq *$ for all $i$, or there is a pending left move $d_i = *$ for some $i$. In the first case, we fixed all the moves correctly, and so we move to state $P' = \langle \text{read}, q, a_1, d_1, \ldots, a_k, d_k \rangle$, write $\triangleright$, and move $\mathsf{R}$, and start simulating the next step. In the second case, we need to re-mark some head positions (since on left moves from the end of the tape, we stay there). We will do this in two steps. First single($M$) will remain in state $P$, write $\triangleright$, and move $\mathsf{R}$. In the next step, the transitions already defined, will place the heads correctly, move left in a state without pending head moves to read $\triangleright$ and will be handled by the previous case.

**Acceptance/Rejection** If $M$ accepts/rejects (i.e., its state is $q_{\mathsf{acc}}$ or $q_{\mathsf{rej}}$) then single($M$) will move to its accept/reject state.

$$\delta'(\langle \text{read}, q_{\mathsf{acc}}, a_1, d_1, \ldots, a_k, d_k \rangle, b) = (q'_{\mathsf{acc}}, b, \mathsf{L})$$
$$\delta'(\langle \text{read}, q_{\mathsf{rej}}, a_1, d_1, \ldots, a_k, d_k \rangle, b) = (q'_{\mathsf{rej}}, b, \mathsf{L})$$

---

## 1.2 Nondeterministic TM

### Nondeterministic Turing Machine

Deterministic TM: At each step, there is one possible next state, symbols to be written and direction to move the head, or the TM may halt.

Nondeterministic TM: At each step, there are finitely many possibilities. So formally, $M = (Q, \Sigma, \Gamma, \delta, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}})$, where

- $Q, \Sigma, \Gamma, q_0, q_{\mathsf{acc}}, q_{\mathsf{rej}}$ are as before for 1-tape machine
- $\delta : (Q \setminus \{q_{\mathsf{acc}}, q_{\mathsf{rej}}\}) \times \Gamma \to \mathcal{P}(Q \times \Gamma \times \{\mathsf{L}, \mathsf{R}\})$

---

### Computation, Acceptance and Language

- A configuration of a nondeterministic TM is exactly the same as that of a 1-tape TM. So are notions of starting configuration and accepting configuration.
- A single step $\vdash$ is defined similarly. $X_1 X_2 \cdots X_{i-1} q X_i \cdots X_n \vdash X_1 X_2 \cdots p X_{i-1} Y \cdots X_n$, if $(p, Y, L) \in \delta(q, X_i)$; case for right moves is analogous.
- $w$ is *accepted* by $M$, if from the starting configuration with $w$ as input, $M$ reaches an accepting configuration, for some sequence of choices at each step.
- $\mathbf{L}(M) = \{w \mid w \text{ accepted by } M\}$

---

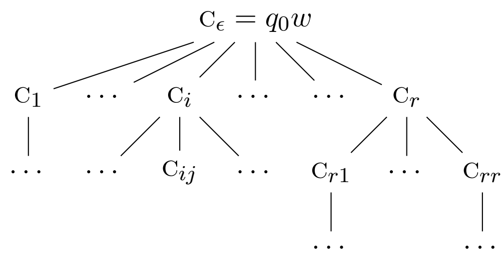
### Expressive Power of Nondeterministic TM

**Theorem 2.** *For any nondeterministic Turing Machine $M$, there is a (deterministic) TM $det(M)$ such that $\mathbf{L}(det(M)) = \mathbf{L}(M)$.*

**Proof Idea**
$\det(M)$ will simulate $M$ on the input.

- Idea 1: $\det(M)$ tries to keep track of all possible "configurations" that $M$ could possibly be after each step. Works for DFA simulation of NFA but not convenient here.
- Idea 2: $\det(M)$ will simulate $M$ on each possible sequence of computation steps that $M$ may try in each step.

## Nondeterministic Computation




- If $r = \max_{q,X} |\delta(q, X)|$ then the runs of $M$ can be organized as an $r$-branching tree.
- $\mathrm{c}_{i_1 i_2 \cdots i_n}$ is the configuration of $M$ after $n$-steps, where choice $i_1$ is taken in step 1, $i_2$ in step 2, and so on.
- Input $w$ is accepted iff $\exists$ accepting configuration in tree.

---

## Deterministic Simulation
## Proof Idea

The machine $\det(M)$ will search for an accepting configuration in computation tree

- The configuration at any vertex can be obtained by simulating $M$ on the appropriate sequence of nondeterministic choices
- $\det(M)$ will perform a BFS on the tree. Why not a DFS?

Observe that $\det(M)$ may not terminate if $w$ is not accepted.

---

## Proof Details

$\det(M)$ will use 3 tapes to simulate $M$ (note, multitape TMs are equivalent to 1-tape TMs)

- Tape 1, called *input tape*, will always hold input $w$
- Tape 2, called *simulation tape*, will be used as $M$'s tape when simulating $M$ on a sequence of nondeterministic choices
- Tape 3, called *choice tape*, will store the current sequence of nondeterministic choices

---

## Execution of det($M$)

1. Initially: Input tape contains $w$, simulation tape and choice tape are blank
2. Copy contents of input tape onto simulation tape
3. Simulate $M$ using simulation tape as its (only) tape
   (a) At the next step of $M$, if state is $q$, simulation tape head reads $X$, and choice tape head reads $i$, then simulate the $i$th possibility in $\delta(q, X)$; if $i$ is not a valid choice, then goto step 4
   (b) After changing state, simulation tape contents, and head position on simulation tape, move choice tape's head to the right. If Tape 3 is now scanning $\sqcup$, then goto step 4
   (c) If $M$ accepts then accept and halt, else goto step 3(1) to simulate the next step of $M$.
4. Write the lexicographically next choice sequence on choice tape, erase everything on simulation tape and goto step 2.

---

**Deterministic Simulation**
*In a nutshell*

- det($M$) simulates $M$ over and over again, for different sequences, and for different number of steps.
- If $M$ accepts $w$ then there is a sequence of choices that will lead to acceptance. det($M$) will eventually have this sequence on choice tape, and then its simulation $M$ will accept.
- If $M$ does not accept $w$ then no sequence of choices leads to acceptance. det($M$) will therefore never halt!

---

## 1.3 Random Access Machine

**Random Access Machines**

This is an idealized model of modern computers. Have a finite number of "registers", an infinite number of available memory locations, and store a sequence of instructions or "program" in memory.

- Initially, the program instructions are stored in a contiguous block of memory locations starting at location 1. All registers and memory locations, other than those storing the program, are set to 0.

---

**Instruction Set**

- `add X, Y`: Add the contents of registers $X$ and $Y$ and store the result in $X$.
- `loadc X, I`: Place the constant $I$ in register $X$.

- `load X, M`: Load the contents of memory location $M$ into register $X$.
- `loadI X, M`: Load the contents of the location "pointed to" by the contents of $M$ into register $X$.
- `store X, M`: store the contents of register $X$ in memory location $M$.
- `jmp M`: The next instruction to be executed is in location $M$.
- `jmz X, M`: If register $X$ is 0, then jump to instruction $M$.
- `halt`: Halt execution.

---

**Expressive Power of RAMs**

**Theorem 3.** *Anything computed on a RAM can be computed on a Turing machine.*

*Proof.* We outline a proof sketch in next. □

---

**Capturing State of RAM**

In order to simulate the RAM, the TM stores contents of registers, memory etc., in different tapes as follows.

- Instruction Counter Tape: Stores the memory location where the next instrcution is stored; initially it is 1.
- Memory Address Tape: Stores the address of memeory location where a load/store operation is to be performed.
- Register Tape: Stores the contents of each of the registers.
  - Has register index followed by contents of each register as follows: $\#\langle\text{RegisterNumber}\rangle * \langle\text{RegisterValue}\rangle\#\cdots$. For example, if register 1 has 11, register 2 has 100, register 3 has 11011, etc, then tape contains $\#1 * 11\#10 * 100\#11 * 11011\#\cdots$
- Memory Tape: Like register tape, store $\#\langle\text{Address}\rangle * \langle\text{Contents}\rangle\#$
  - To store an instruction, have opcode, $\langle\text{arguments}\rangle$
- Work Tapes: Have 3 additional work tapes to simulate steps of the RAM

---

**Simulating a RAM**

- TM starts with the program stored in memory, and the instruction location tape initalized to 1.

- Each step of the RAM is simulated using many steps.
  - Read the instruction counter tape
  - Search for the relevant instruction in memory
  - Store the opcode of instruction and register address (of argument) in the finite control. Store the memory addres (of argument) in memory address tape.
  - Retrieve the values from register tape and/or memory tape and store them in work tape
  - Perform the operation using work tapes
  - Update instruction counter tape, register tape, and memory tape.

---

**Example: ADD instruction**

- Suppose instruction counter tape holds 101.
- TM searches memory tape for the pattern $\#101*$.
- Suppose the memory tape contains $\cdots \#101 * \langle \texttt{add} \rangle, 11, 110\# \cdots$
- TM stores "add", 11 and 110 in its finite control. In other words, it moves to a state $q_{\texttt{add}\ 11,110}$ whose job it is to add the contents of register 11 and 110 and put the result in 11.
- Search the register tape for the pattern $\#11*$. Suppose the register tape contains $\cdots \#11 * 10110\# \cdots$; in other words, the contents of register 11 is 10110. Copy 10110 to one of the work-tapes.
- Search the register tape for pattern $\#101*$, and copy the contents of register onto work tape 2.
- Compute the sum of the contents of the work tapes
- Search the register tape for $\#11*$ and replace the string 10110 by the answer computed on the work tape. This may involve shifting contents of the register tape to the right (or left).
- Add 1 to the instruction counter tape.

---

# 2 Church-Turing Thesis

## 2.1 Universality of the Model

**Robustness of the Class of TM Languages**

Various efforts to capture mechanical computation have the same expressive power.

- Non-Turing Machine models: random access machines, $\lambda$-calculus, type 0 grammars, first-order reasoning, $\pi$-calculus, ...

- Enhanced Turing Machine models: TM with 2-way infinite tape, multi-tape TM, nondeterministic TM, probabilistic Turing Machines, quantum Turing Machines ...
- Restricted Turing Machine models: queue machines, 2-stack machines, 2-counter machines, ...

---

## 2.2 Church-Turing Thesis

### Church-Turing Thesis

"Anything solvable via a mechanical procedure can be solved on a Turing Machine."

- Not a mathematical statement that can be proved or disproved!
- Strong evidence based on the fact that many attempts to define computation yield the same expressive power

---

### Consequences

- In the course, we will use an informal pseudo-code to argue that a problem/language can be solved on Turing machines
- To show that something can be solved on Turing machines, you can use any programming language of choice, *unless the problem specifically asks you to design a Turing Machine*

---