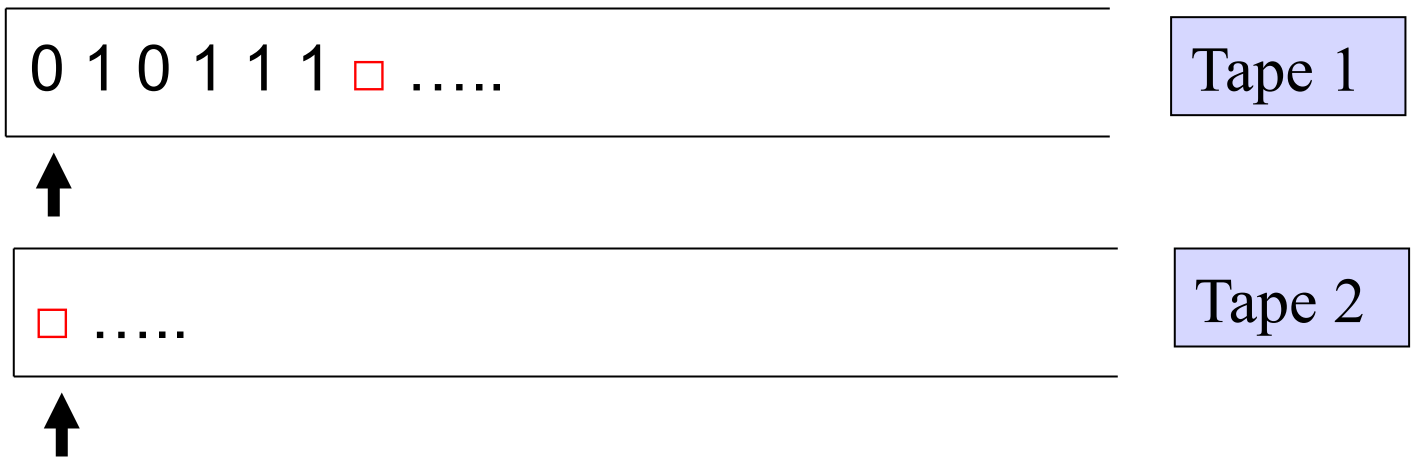
Are Turing machines too primitive?

- The answer is NO. It is believed that Turing machines are as powerful as "any" computational models.
- Today's lecture: Turing machines are powerful enough to "simulate" two other computation models.

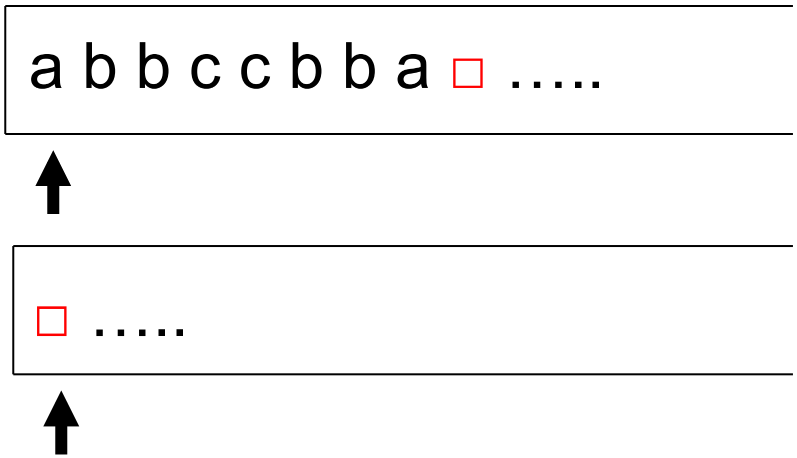
2-tape Turing machines



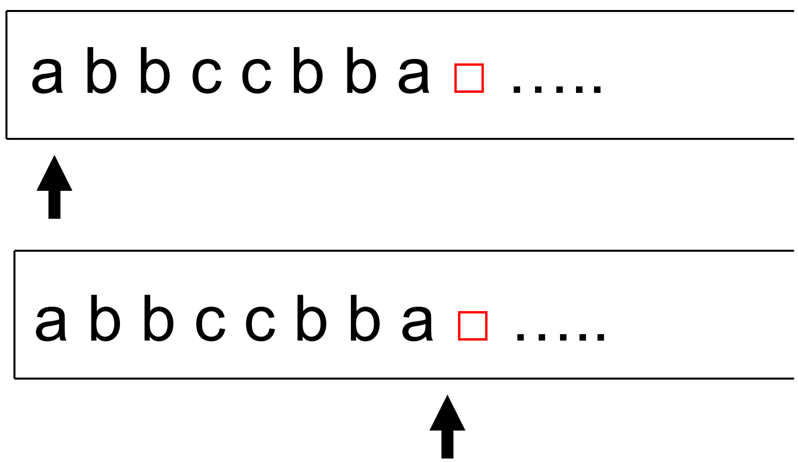
- Like an ordinary TM except that it is equipped with 2 tapes (each with its own tape head).
- In one step, the machine can read two symbols, one from each tape, and make a move.
- Initially, the input is in Tape 1.
- Formally, the transition function takes the form

$$\delta : Q \times \Gamma \times \Gamma \rightarrow Q \times \Gamma \times \Gamma \times \{L,R\} \times \{L,R\}$$

Palindromes revisited



A 2-tape Turing machine can determine palindromes more efficiently.



1-tape TM versus multi-tape TM

Is a multi-tape TM more "powerful" than a 1-tape TM ?

- A language L that can be decided by a 2-tape TM, but not by any 1-tape TM ?

Answer: NO (for k-tape TM for any constant k).

Proof: Step-by-step simulation.

Let **M** = $(Q, \Sigma, \Gamma, \delta, q_0, q_{ac}, q_{rj})$ be a 2-tape TM that decides L.

Suppose $\Gamma = \{a_1, a_2, \ldots, a_k\}$.

The 1-tape TM **M'** uses a larger tape alphabet,

- $\Gamma \cup \Gamma' \cup \Gamma'' \cup \{\#\}$, where $\Gamma' = \{a_1', a_2', \ldots, a_k'\}$ and $\Gamma'' = \{a_1'', a_2'', \ldots, a_k''\}$.

Simulation

Construct a 1-tape TM M' to simulate M as follows:

a b c b b a □ .....

d a a □ .....

a b c' b b a # d a" a # □ .....

The positions of the tape heads are represented by special symbols.

Details

To simulate one step of M, M' needs two phases:

Phase 1:

- M' reads the tape from left to right to find out the symbols under M's two tape heads;
- based on the state of M and these symbols, M' determines the next step of M.

Phase 2:

- update the tape according to the move of M.

M' accepts if M accepts; M' rejects if M rejects.

Questions

Suppose that for an input x of length n, M takes $f(n)$ steps before reaching a halting state. Assume $f(n) \geq n$.

Derive an upper bound on the number of steps used by **M'**.

Answer: $O(f(n)^2)$ steps.

Key observation: Each tape of M contains at most $f(n)$ non-blank symbols.

Simulation of TMs with $k > 1$ tapes? TMs with a matrix-like memory?

A **random access machine** (RAM) is a model for algorithm design & analysis.

- RAM is equipped with memory that can be accessed by addresses; support the concepts of instructions & programs.
- it models a simple but real computer.

Given: RAM with a particular program P

construct a (multi-tape) Turing machine doing the same job

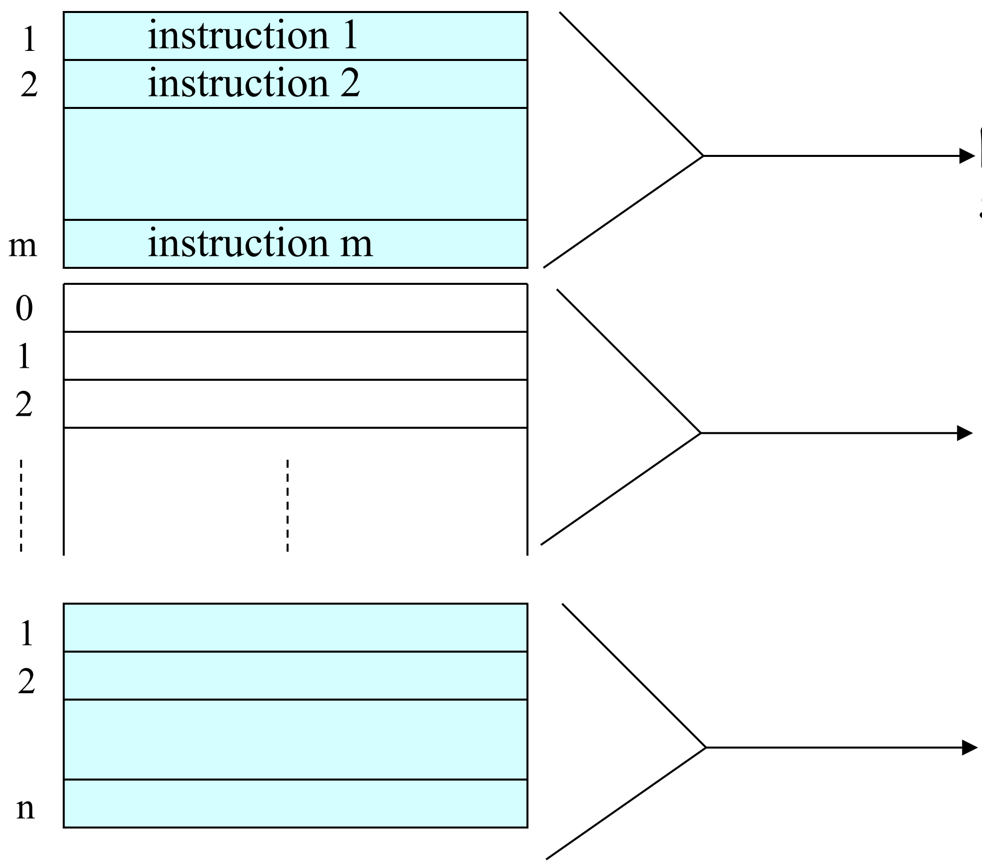
Implication

Let L be a language.

- If L can be decided by a RAM program (or any *C* program for your PC), then L can also be decided by a Turing machine.
- If L is not Turing decidable, no RAM program can decide L.

Random access machine (RAM)

5 program counter



program, a fixed-length sequence of instructions

registers, each capable of holding an arbitrarily long binary integer

input registers

Notations: r_j – the j-th register
i_j – the j-th input register

RAM instruction set (simplified)

Read j	$(r_0 = i_j)$
Read ↑j	$(r_0 = i_{r_j})$
Store j	$(r_j = r_0)$
Store ↑j	
Load =j, Load j, Load ↑j	$(r_0 = j, r_0 = r_j, r_0 = r_{r_j})$
Add =j, Add j, Add ↑j	$(r_0 = r_0 + j, r_0 = r_0 + r_j, r_0 = r_0 + r_{r_j})$
Sub =j, Sub j, Sub ↑j	
Half =j, Half j, Half ↑j	
Jump j	(program counter = j)
Jzero j	(if $r_0 = 0$, program counter = j)
Jneg j, Jpos j	
Halt	

NB. j is a binary integer.

INPUT	
i_1	a
i_2	b
i_3	c

Suppose that we want to compute a+b-c.

The output is to be stored in r_0

READ 1	$r_0 \leftarrow i_1$
STORE 12	$r_{12} \leftarrow r_0$
READ 2	$r_0 \leftarrow i_2$
ADD 12	$r_0 \leftarrow \overset{a}{r_0} + \overset{b}{r_{12}}$
STORE 12	$r_{12} \leftarrow r_0$
READ 3	$r_0 \leftarrow i_3$
STORE 4	$r_4 \leftarrow r_0$
LOAD 12	$r_0 \leftarrow r_{12}$
SUB 4	$r_0 \leftarrow r_0 - r_4$
HALT	

- Let Φ be a function mapping any finite sequence of integers to an integer.

  E.g., $\Phi((4,5,6)) = 15$.

- "A RAM with a program $P = (\pi_1 , \pi_2 ,.. \pi_m)$ computes a function Φ." What does it mean?

- For any input sequence of integers $I = (i_1, i_2, \dots i_l)$, the RAM (using P) will eventually execute the "**HALT**" instruction, and by that time, the value of $\Phi(I)$ is stored in register 0, i.e., r_0.

Input size

- Note that the input I is a sequence of integers in binary. The <u>input size</u> is their total <u>length</u>, i.e. $\lceil \log i_1 \rceil + \lceil \log i_2 \rceil + \cdots$, instead of their values.

- Notation : size(I) denotes the size of I.

- A RAM operates within $f(n)$ time if,
  for any input I of size n, it executes at most $f(n)$ instructions to compute the output (**unit-cost model**).

N.B. ***log-cost model*** : an instruction is charged according to the size of the operands.

e.g.

ADD 5 $\quad (r_0 \longleftarrow r_0 + r_5)$

$$r_0 \quad \boxed{111111100}$$
$$r_5 \quad \boxed{1111110001}$$

This instruction is charged a cost of 9 + 10.

Observations

Consider running a RAM program P with input I of size n.

- After P has executed t steps, what is the <u>length</u> of the biggest integer stored in the memory?

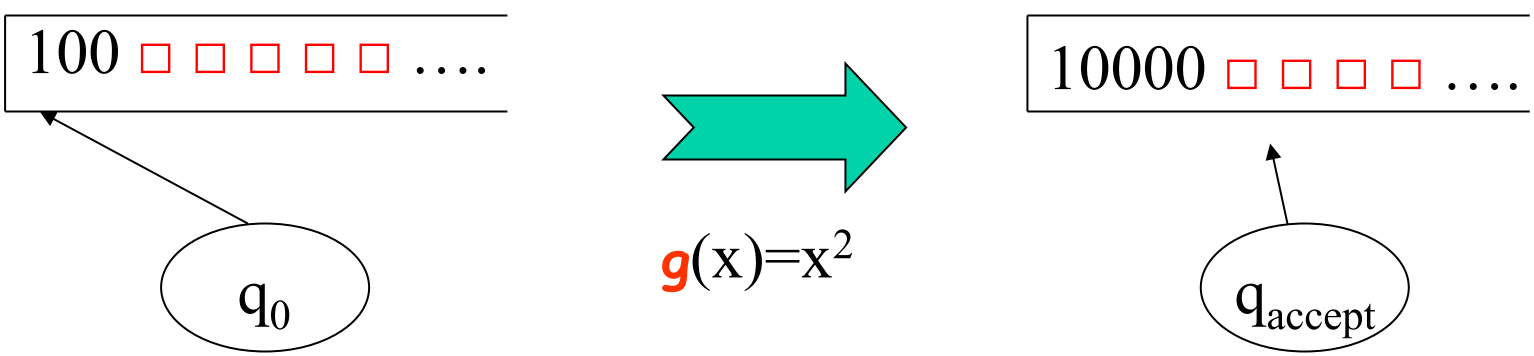
- Answer: max(n, max-constant(P)) + t

  Notation: max-constant(P) denotes the size of the largest possible integer or address in the program P.

Turing machines compute functions

A Turing machine M is said to compute a function
$g : \Sigma^* \rightarrow \Sigma^*$ if for all $x \in \Sigma^*$,

- M starting with x as input halts with g (x) left on its tape.



A function g is said to be (Turing) computable if there exists a Turing machine computing g .

Theorem. Let Φ be any function that can be computed by a RAM with a program P in time $f(n) \geq n$. Then there exists a Turing machine M computing Φ in time $O(f^3(n))$.

Proof (by simulation).

Input :

tape 1 $\triangleright\ i_1\ ;\ i_2\ ;\ i_3\ ;\ \ldots\ ;\ i_l\ \sqcup$

in binary

Registers :

tape 2 $\triangleright\ \mathbf{0}\ ,\ r_0\ ;\ \mathbf{4}\ ,\ r_4\ ;\ 12\ ,\ r_{12}\ ;\ \ldots$

(index,value)

Initially, the tape is empty

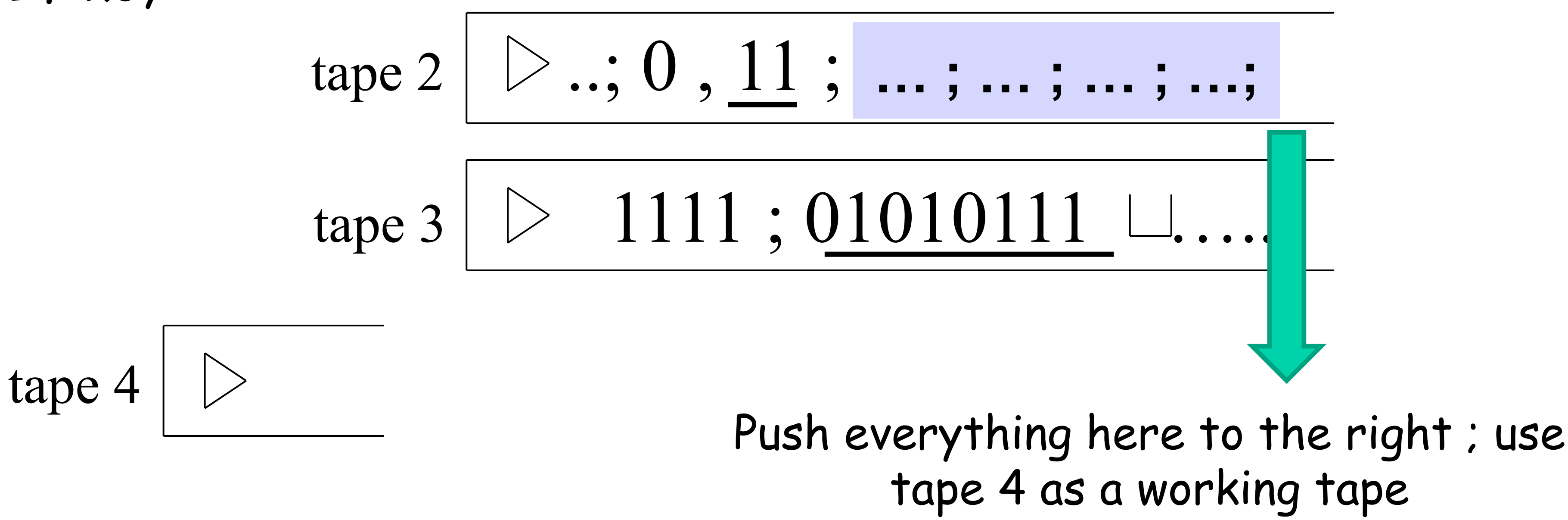
We only keep track of those registers that have been referenced. If a register is missed here, it contains a zero.

- Consider the instruction **LOAD** 15, i.e. $r_0 \leftarrow r_{15}$
- Let us design a TM to simulate this instruction.
- Write 15 in binary on tape 3. Scan tape 2 from left to right to match the <u>index</u> 15.

tape 2	$\triangleright$ 0 , 1111 ; 111 , 01111 ; …
tape 3	$\triangleright$ 1111 $\sqcup$

- If found, copy the value to tape 3 ; otherwise, append a zero on tape 3.
- Scan tape 2 again to find index 0.

- If the index 0 is not found (very unlikely), make a new entry at the end of tape 2.
- Otherwise, check if the previous content of r_0 has the <u>same length</u> as that of r_{15}.
- If yes, copy the value of r_{15} from tape 3 to tape 2
- If no,



- **LOAD** $\uparrow 15$ ($r_0 \leftarrow r_{r_{15}}$)
- Put 15 on tape 3 and search tape 2 for 15.
  - If no :   done
  - If yes : put the value of r_{15} on tape 3

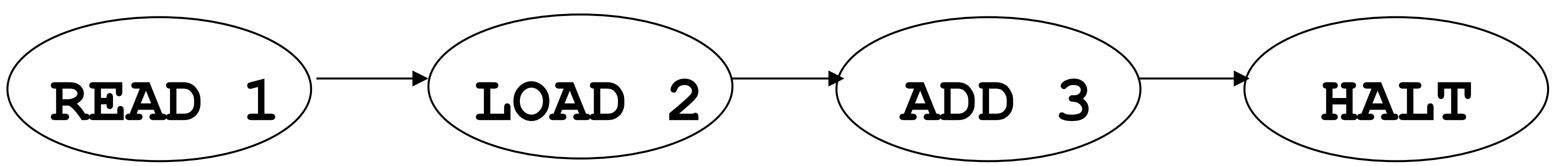
tape 3 $\triangleright$ 1111 ; $\underline{1101111}$ $\sqcup$.....

- Search tape 2 again for an index matching what we have just put down on tape 3.
- Repeat the steps before.

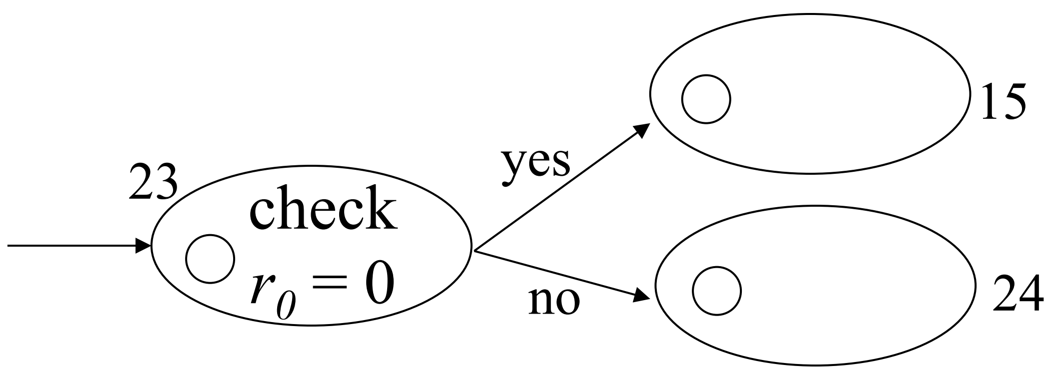
- Exercise: construct a TM for **ADD** 16 ($r_0 \longleftarrow r_0+r_{16}$)
- How to simulate a sequence of instructions?

```
READ 1

LOAD 2

ADD 3

HALT
```

- Use a Turing machine with 4 groups of states.



- Note that the size of the RAM program P is fixed in advance (i.e., a constant not depending on the input).
- Branching instructions are simulated by the transition among the states.
- E.g., instruction 23: **JZERO** 15   (if r_0 = 0, goto instruction 15)



- Assume the RAM operates within $f(n)$ time.
- Consider an input of size *n*.
- At any time, a tape contains at most $O(f^2(n))$ non-blank symbols. Why?
- Simulating a step of the RAM requires $O(f^2(n))$ time.
- In other words, the whole simulation requires $O(f^3(n))$ time.