4.3.4 RANDOM ACCESS MEMORY TURING MACHINES

1. Random Access:
   Random Access is an indexed memory (tape) in which a particular location (index) can be reached in one step.

We can define a Turing Machine (TM) with Random Access Memory (RAM) as follows:

2. Instruction of RAM TM: (page 142)
   (Assembly Operation of Micro Code)
<table>
<thead>
<tr>
<th>Instruction</th>
<th>Operand</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>read</td>
<td>j</td>
<td>AC $\leftarrow$ T[R]</td>
</tr>
<tr>
<td>write</td>
<td>j</td>
<td>T[Rj] $\leftarrow$ AC</td>
</tr>
<tr>
<td>store</td>
<td>j</td>
<td>Rj $\leftarrow$ AC</td>
</tr>
<tr>
<td>load</td>
<td>j</td>
<td>AC $\leftarrow$ Rj</td>
</tr>
<tr>
<td>load c</td>
<td></td>
<td>AC $\leftarrow$ c</td>
</tr>
<tr>
<td>add</td>
<td>j</td>
<td>AC $\leftarrow$ AC+ Rj</td>
</tr>
<tr>
<td>add c</td>
<td>c</td>
<td>AC $\leftarrow$ AC+c</td>
</tr>
<tr>
<td>sub</td>
<td>j</td>
<td>AC $\leftarrow$ max{AC-Rj,0}</td>
</tr>
<tr>
<td>sub c</td>
<td>c</td>
<td>AC $\leftarrow$ max{AC-c,0}</td>
</tr>
<tr>
<td>jump</td>
<td>s</td>
<td>PrC $\leftarrow$ s</td>
</tr>
<tr>
<td>jpos</td>
<td>s</td>
<td>If AC &gt; 0 then PrC $\leftarrow$ s</td>
</tr>
<tr>
<td>jzero</td>
<td>s</td>
<td>If AC = 0 then PrC $\leftarrow$ s</td>
</tr>
</tbody>
</table>

PrC start from 0, and is increased by 1 after each instruction, unless otherwise started (i.e. when the instruction executed is a jump instruction)

**Question:**
Is RAM TM more powerful than a one-tape TM?

**Answer:**
NO

3. **Theorem:**
Given a rAM TM $M_{RAM}$ we can construct a one-tape TM $M$ such that they both compute the same language.

$L(M_{RAM}) = L(M)$

It suffices to construct a multiple tape TM to simulate $M_{RAM}$.

Assume $M_{RAM}$ has k register. Assume that the program of $M_{RAM}$ is $\alpha_1 \alpha_2 \alpha_3 \ldots$

$\alpha_n$ each of these $\alpha$’s is one instruction.

We will construct a (k+3) tape TM M to simulate this $M_{RAM}$ as follow:

Input

Working tape
Create n finite states \( Q_1, Q_2, Q_3, \ldots, Q_n \) each \( Q_i \) corresponding to \( \alpha_i \). Within each \( Q_i \) we’ll have some other states which are used to simulate \( \alpha_i \).

We now only need to see how to simulate \( \alpha_i \) with \((k+3)\)-tape.

In the tape that simulate the addressable memory, we’re going to place a pair \((i, x)\) in each cell, where \( i \) represents the cell in the addressable memory, and \( x = T[i] \).

### 4.4 NONDETERMINISTIC TURING MACHINES

A Nondeterministic Turing Machine (NTM) is a five-tuple

\[ M = (S, \Sigma, \delta, s, H) \]

Where

\[ S, \Sigma, s, H \text{ are the same DTM} \]

And \( \delta = (S-H) \times \Sigma \rightarrow 2^{S \times (\Sigma \cup \{ \leftarrow, \rightarrow \})} \)

1. **Definition of NTM:**

   We say that an input is accepted by an NTM \( M \) is there is a computation path that on input \( x \) halts.

   **Picture**

   ![Diagram](image)

   Let \( L(M) = \{ x \mid x \text{ is accepted by } M \} \)
**Question:**
Is NTM TM more powerful than DTM?

**Answer:**
NO

We can simulate an NTM $M_N$ using $M_D$ such that both accept the same language.

**Idea:**
Use breadth-first search to traverse the computation tree of $M_N$ on a given input $x$, as soon as a halt state is reached the $M_D$ halts.

**Question:**
How do we construct $M_D$ based on this idea?