A. Suppose for the sake of contradiction that A is context-free. Then, by the pumping lemma for context-free languages, there exists a pumping length p. Let $w = 1^p01^p\#1^p10^p$. This w is a string in A of length at least p, so there exist strings u, v, x, y, z such that $w = uvxyz$, $|vxy| \leq p$, $|vy| > 0$, and $uv^ixy^iz \in A$ for all $i \in \{0, 1, 2, \ldots\}$. In order to pump to other strings in the language, it cannot be that (A) v or y contains $\#$, (B) v and y are both substrings of 1^p01^p, or (C) v and y are both substrings of 1^p10^p. Because $|vxy| \leq p$, the only possibility is that vxy is a substring of $1^p\#1^p$, with x containing $\#$ and v and y consisting of 1s only. But then pumping v and y again leaves the language A. Thus A cannot be context-free. (Fill in the details.)

B. It is obvious that a two-dimensional Turing machine (2DTM) can simulate an ordinary Turing machine (TM). The former simply moves its head right one cell, to the beginning of the input, and then runs the TM's transitions verbatim. The 2DTM never uses any cells other than the ones directly to the right of its origin.

In order to simulate a 2DTM on a TM, we need to store the 2DTM tape on an ordinary TM tape. Any computable bijection $f : \mathbb{Z}^2 \to \mathbb{N}$ suffices. That is, f is a one-to-one correspondence between pairs of integers (which index tape cells in the 2DTM) and natural numbers (which index tape cells in the TM). For example, f may enumerate 2DTM tape cells by numbering them in a spiral that emanates from the origin.

We will simulate the 2DTM using a three-tape TM M (which can be simulated by a one-tape TM, as we have seen). The first tape will store the two-dimensional tape, in the order defined by f. The second tape will store a pair (x, y) of signed integers, indicating the location of the 2DTM's tape head. The third tape will be used for computing f. This new M will have all of the states of the 2DTM, plus additional states for implementing certain subroutines.

Here is a subroutine of M for moving the 2DTM tape head to cell (x, y). Assume that these coordinates are already on the second tape. Copy them to the third tape. Run the computable function f using the third tape, to produce $f(x, y)$ on the third tape. Return the first tape head to the left end of its tape. Then repeatedly move this tape head right, each time decrementing the third tape, until the third tape is zero. At that point, the first tape head is over cell $f(x, y)$, which corresponds to cell (x, y) in the 2DTM.

When M is given input $w = w_1 \cdots w_n$ on its first tape, it first pre-processes its input into the form expected by a 2DTM. It writes $(x, y) = (n, 0)$ to the second tape, blanks w_n from the first tape, remembers w_n in its state, uses the subroutine described above to move to cell (x, y), and writes w_n to cell (x, y). It repeats these steps to move w_i to cell $(i, 0)$, for $i = n - 1, \ldots, 1$. It also enters the symbol $\vdash$ in cell $(0, 0)$.

To execute a 2DTM transition $\delta(q, a) = (r, b, d)$, M does these steps:

1. On the first tape, where the tape head is currently seeing a, replace the a with a b.

2. On the second tape, either increment or decrement either x or y, depending on d.

3. Use the subroutine described above, to move to cell (x, y).

4. Transition to state r.

The three-tape TM accepts or rejects whenever it enters the accept or reject state of the 2DTM.

C. First we reduce $\overline{HALT_{TM}}$ to $\overline{DEC}$. For any Turing machine M and input w, define N to be the Turing machine that, on any input x, runs M on w and outputs whatever M outputs. If M halts on w, then N halts on all inputs. If M does not halt on w, then N halts on no inputs, and in particular does not halt on all inputs. Thus the function that produces $\langle N \rangle$ from $\langle M, w \rangle$ is a computable reduction of $\overline{HALT_{TM}}$ to $\overline{DEC}$. Because $\overline{HALT_{TM}}$ is not recognizable, it follows that $\overline{DEC}$ is not recognizable.

In a sense, reducing a problem to $\overline{DEC}$ is inherently easier than reducing a problem to DEC. A hypothetical recognizer for $\overline{DEC}$ is very useful in solving other problems, because TMs in $\overline{DEC}$ are hard to deal with, because they don't always halt. In contrast, a hypothetical recognizer for DEC is not so useful, because the TMs that it recognizes are precisely the TMs that are already easy to use.

We now reduce $\overline{A_{TM}}$ to DEC. For any Turing machine M and input w, define a Turing machine N that, on input x, does the following. N runs M on w, but only for $|x|$ steps. If M accepts, then N enters into a loop. Otherwise, N accepts. In analyzing N, consider four cases:

1. If M accepts w in no more than $|x|$ steps, then N loops on x.

2. If M accepts w in more than $|x|$ steps, then N accepts x.

3. If M rejects w, then N accepts x.

4. If M loops on w, then N accepts x.

If $\langle M, w \rangle \in \overline{A_{TM}}$, then $\langle M, w \rangle$ fits into one of the two latter cases. In these cases, N accepts all inputs x, and thus $N \in DEC$. On the other hand, if $\langle M, w \rangle \in A_{TM}$, then $\langle M, w \rangle$ fits into one of the two former cases. Let t be the number of steps in which M accepts w. Then N loops on inputs x such that $|x| \geq t$, so $N \in \overline{DEC}$. Thus the function that produces $\langle N \rangle$ from $\langle M, w \rangle$ is a computable reduction of $\overline{A_{TM}}$ to DEC. Because $\overline{A_{TM}}$ is not recognizable, it follows that DEC is not recognizable.