



POLITECHNIKA WARSZAWSKA

WYDZIAŁ ELEKTRYCZNY

---

Konwersja liczb naturalnych z systemu  
decymalnego na szesnastkowy za pomocą  
Maszyny Turinga

AUTOMATYKA I ROBOTYKA STOSOWANA

23 czerwca 2022

---

# *Spis treści*

---

<b>1</b>	<b>Wstęp</b>	<b>1</b>
1.1	Maszyna Turinga . . . . .	1
1.1.1	Definicje . . . . .	1
1.1.2	Założenia . . . . .	2
<b>2</b>	<b>Translator</b>	<b>3</b>
2.1	Podstawowe wytyczne i działanie . . . . .	3
2.1.1	Budowa programu . . . . .	3
<b>3</b>	<b>Definicja algorytmu</b>	<b>5</b>
3.1	Idea . . . . .	5
3.1.1	Oznaczenia w schematach . . . . .	5
3.1.2	Schemat ideowy . . . . .	5
3.2	Skrótowy opis działania . . . . .	6
3.3	Funkcja <i>dec to bin</i> . . . . .	7
3.4	Funkcja <i>bin to hex</i> . . . . .	8
<b>4</b>	<b>Podsumowanie</b>	<b>9</b>

---

# 1 Wstęp

---

Sprawozdanie opisuje działanie Maszyny Turinga składającej się z **translatora** i **pliku definiującego**. Plik definiujący **DecToHex.txt** zawiera definicje stanów, alfabetu i taśmy właściwe dla zrealizowania algorytmu konwersji dec do hex. Translator **turing.py** został napisany w Pythonie (3.9.7), jego funkcją jest obsługa powyższego pliku tekstowego i interpretacja definicji stanów symulująca działanie Maszyny Turinga. Jednostka translatora jest w stanie zrealizować dowolny algorytm, który ma odpowiednią składnię definicji.

## 1.1 Maszyna Turinga

Jest abstrakcyjną maszyną składającą się z ruchomej głowicy i taśmy o nieskończonej ilości komórek wypełnionych pojedynczymi znakami (dopuszcza się też maszyny operujące na większej ilości par). Uproszczony model działania:

1. Odczytanie znaku z taśmy przez głowicę
2. Zapisanie nowego znaku w tej samej komórce
3. Przejście maszyny w następny stan
4. Przesunięcie głowicy o jedną komórkę w lewo/prawo

Formalną definicję można zapisać jako:

$$M = \langle Q, q_0, F, \Gamma, \Sigma, B \rangle \quad (1.1)$$

$Q$  - skończony zbiór stanów  
 $q_0$  - stan początkowy  
 $F$  - zbiór stanów końcowych  
 $\Gamma$  - alfabet, czyli zbiór wszystkich akceptowanych znaków  
 $\Sigma \in \Gamma - \{B\}$  - zbiór dopuszczalnych znaków inputu  
 $B$  - znak pusty

### 1.1.1 Definicje

**Stan** opisuje etap podczas pracy maszyny, zbiór wszystkich definicji zawierających dany stan jest równoważny do przebiegu funkcji/metody. Na przykład stan o nazwie CHANGE mógłby być wykorzystany do inwersji binarnej.

**Stan wejściowy** ten, na którym obecnie operuje maszyna.

**Stan wyjściowy** ten, do którego przejdzie maszyna po odczytaniu danego znaku

**Stan początkowy** jest stanem wejściowym po uruchomieniu maszyny, w dalszej części oznaczany jako QINIT

**Stan końcowy** po odczytaniu go jako stan wejściowy maszyna przestaje pracować, zdefiniowany jako FIN

**Definicja stanu** to paczka danych definiująca zachowanie maszyny, zapis przyjęty w pracy to:

$$\langle \text{Stan wejściowy} \rangle, \quad \langle \text{Input} \rangle, \quad \langle \text{Stan wyjściowy} \rangle, \quad \langle \text{Output} \rangle, \quad \langle \text{Kierunek ruchu} \rangle \quad (1.2)$$

### 1.1.2 Założenia

Zasady, które muszą być spełnione by maszyna działała:

- Taśma
  - Taśma musi zawierać znaki należące do alfabetu
- Stany
  - Nie mogą istnieć dwie definicje stanu wejściowego o tej samej nazwie i tym samym inpucie
  - Każdy stan wyjściowy musi być zdefiniowany jako stan wejściowy (oprócz FIN)
  - Musi istnieć chociaż jedna definicja stanu, w której stan wyjściowy to FIN

### Losowe słowa generator

Lecimy freestyle B), ej ej ej Ej wchodzę sobie na tych dźwiękach, jak nawijam te freestyle to nigdy nie pękam, a moją nawijkę weź sobie mordo zapamiętaj, bo w obiektywie wyglądam lepiej niż *modelka*, **p a t r z** wchodzę sobie na majkach jak nawija tutaj edzio zawsze płynie zajawka, mówią że moje freestyle to jest bajka no bo wjechałem wam na głowę bardziej niż *czapka*, **p a t r z** to jest dobra rozkmina, słucha mnie każdy chłopak i każda dziewczyna, a jak kiedyś skończyłem to od nowa zaczynam, *l o s o w e   s ł o w a   g e n e r a t o r . . .*

---

## 2 *Translator*

---

### 2.1 Podstawowe wytyczne i działanie

Translator ma trzy główne funkcjonalności:

- Kontrolna - weryfikacja składni pliku z inputem
- Sprawcza - symulacja działania Maszyny Turinga i zwrócenie końcowego stanu taśmy
- Informacyjna - zawiera plik **readMe.txt** zawierający kluczowe informacji o działaniu programu

Oprócz założeń wynikających bezpośrednio z definicji Maszyny, istnieją te dotyczące samej jednostki translacyjnej a dokładniej jej składni:

- Alfabet - poniższe znaki nie mogą być zadeklarowane jako część alfabetu:
  - # - zarezerwowany jako pusty znak
  - / - zarezerwowany jako znak początku komentarza
- Taśma - nie może zawierać znaku spacji
- Stany:
  - Kolejne dane definicji stanu muszą być rozdzielone sekwencją ", " (przecinek, spacja)
  - Kierunek jest określany jako: < (lewo), > (prawo)
  - Dopuszczalne są linijki zawierające wyłącznie znak nowej linii \n
- Input - musi zostać podany w kolejności:
  1. Alfabet
  2. Taśma
  3. Definicja stanów + komentarze

#### 2.1.1 Budowa programu

Program operuje na klasach: `TuringMachine` i `State`. `State` pełni funkcję "kontenera" zawierającego paczkę 5 danych formułujących definicję stanu:

```
class State:
    def __init__(self, name, input, state_to, output, dir):
        self.__name__ = name
        self.__in__ = input
        self.__nstate__ = state_to
        self.__out__ = output
        self.__d__ = dir
```

Cała reszta funkcjonalności programu znajduje się wewnątrz klasy **TuringMachine**. Obiekt po utworzeniu wykonuje operacje zdefiniowane w pliku tekstowym, którego ścieżka/nazwa została podana jako argument `file`.

```
class TuringMachine:
    def __init__(self, file):
        f = open(file, "r")
        self.__alphabet__ = self.init_alphabet(f)
        self.__tape__ = self.init_tape(f)
        self.__states__ = self.init_states(f)
        f.close()
        self.__tape__ = self.execute()
```

Metody klasy **TuringMachine** wraz z ich skrótowym działaniem:

Inicjalizacja	
<code>init_alphabet(self, f)</code>	Zczytuje podany w inpucie alfabet, tworzy z niego zbiór i weryfikuje czy użytkownik nie podał zarezerwowanych znaków; Zwraca zbiór
<code>init_tape(self, f)</code>	Zczytuje podaną taśmę i sprawdza czy podane znaki są w alfabecie; Zwraca listę znaków
<code>init_states(self, f)</code>	Zczytuje definicje stanów i zapisuje je do napotkania EOF; weryfikuje składnie według wytycznych; Zwraca listę obiektów
Symulacja	
<code>execute(self)</code>	Metoda symulująca Maszynę, odczytuje dane z tablicy powstałej na podstawie taśmy, odszukuje stany wyjścia, zapisuje output i porusza "głowicą"
<code>show_output(self, clear_sym = None)</code>	Wyświetla stan taśmy w terminalu, możliwe jest pominięcie znaków zadeklarowanych pod <code>clear_sym</code>
Pomocnicze	
<code>readMe(self, file = "readMe.txt")</code>	Wypisuje zawartość <code>readMe</code> w terminalu
<code>create_dic(self)</code>	Tworzy słownik rodzin stanów; pozwala na szybsze znajdowanie definicji stanów wyjściowych

Przykładowa definicja pliku txt wraz z outputem:

```
//binaryNegation.txt

1 0
1000100101
qinit, 1, qinit, 1, >
qinit, 0, qinit, 0, >
qinit, #, pass, #, <
pass, 1, pass, 1, <
pass, 0, pass, 0, <
pass, #, neg, #, >
neg, 1, neg, 0, >
neg, 0, neg, 1, >
neg, #, fin, #, >
```

Output: 0111011010

---

## 3 Definicja algorytmu

---

### 3.1 Idea

Celem algorytmu jest przekształcenie naturalnej liczby dziesiętnej na liczbę w systemie szesnastkowym. Alfabet, z którego korzysta Maszyna:

```
//alfabet
0 1 2 3 4 5 6 7 8 9 A B C D E F !
```

W celu skrócenia zapisu przejść między stanami w dalszej części sprawozdania wykorzystano oznaczenia zbiorów, do których należy rozważany znak:

- *bin* 0 lub 1
- *dec\_num* cyfra dziesiętna
- *even* parzysta cyfra lub 0
- *odd* nieparzysta cyfra

#### 3.1.1 Oznaczenia w schematach

**Sześciokąt** funkcja zbiorcza o znaczeniu ideowym

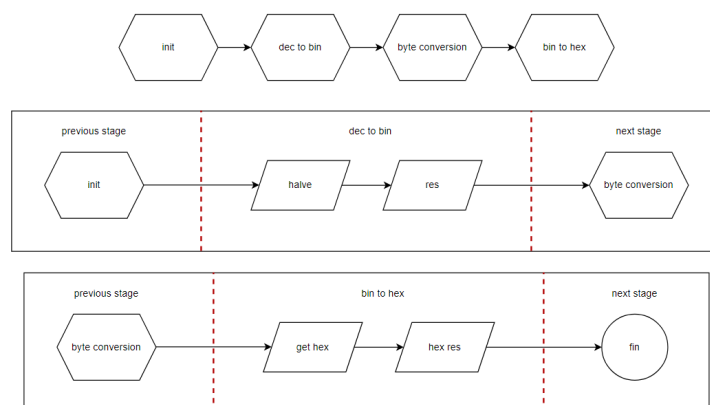
**Równoległobok** podfunkcja o niższym poziomie abstrakcji

**Kwadrat** rodzina definicji stanów pełniących tą samą funkcję

**Koło** oznaczenie stanu; Niebieski kolor stanu oznacza stan "początkowy" danego segmentu

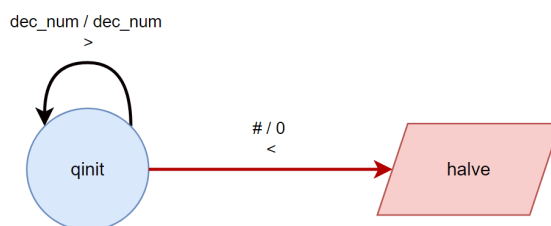
**Kolor** niebieski - stan początkowy segmentu; czerwony - przejście do innego segmentu; zielony - przejście do stanu końcowego

#### 3.1.2 Schemat ideowy



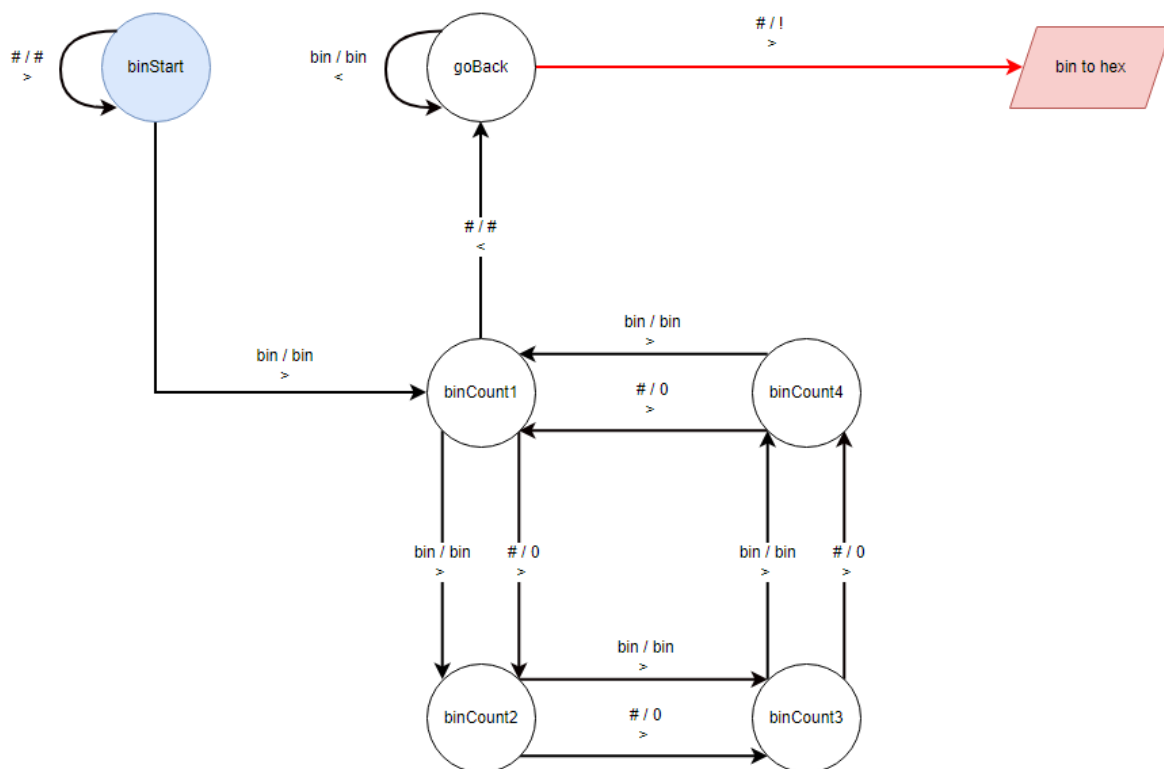
## 3.2 Skrótowy opis działania

**init** weryfikuje czy taśma składa się wyłącznie z cyfr.



**dec to bin** konwersja z systemu dziesiętnego do binarnego. Składa się z dwóch podfunkcji: *halve* dzielącej liczbę na pół i przenoszącej resztę z dzielenia; *res* przenoszącej wynik (0 lub 1) w odpowiednie miejsce taśmy.

**byte conversion** uzupełnia liczbę binarną zerami tak by składała się z pełnych bajtów.



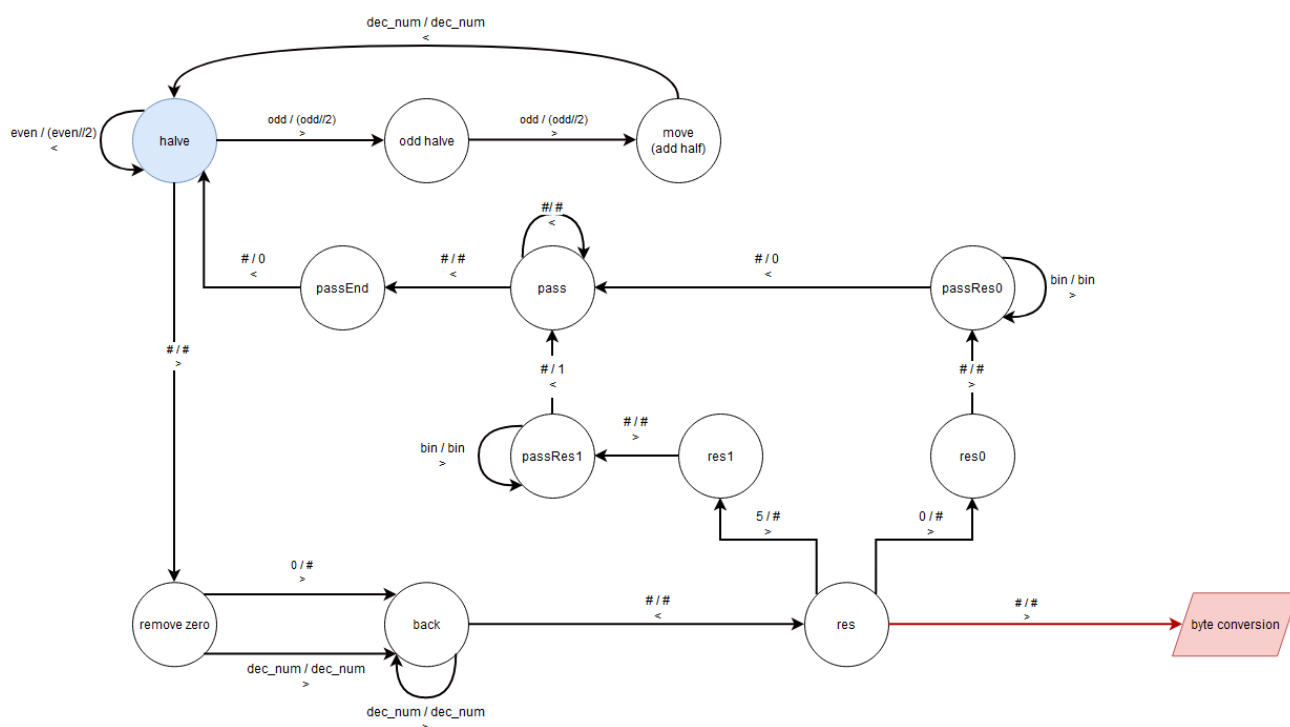
**bin to hex** przeprowadza konwersję z binarnego zapisu do szesnastkowego. Podfunkcje działają analogicznie do *dec to bin*



### 3.3 Funkcja *dec to bin*

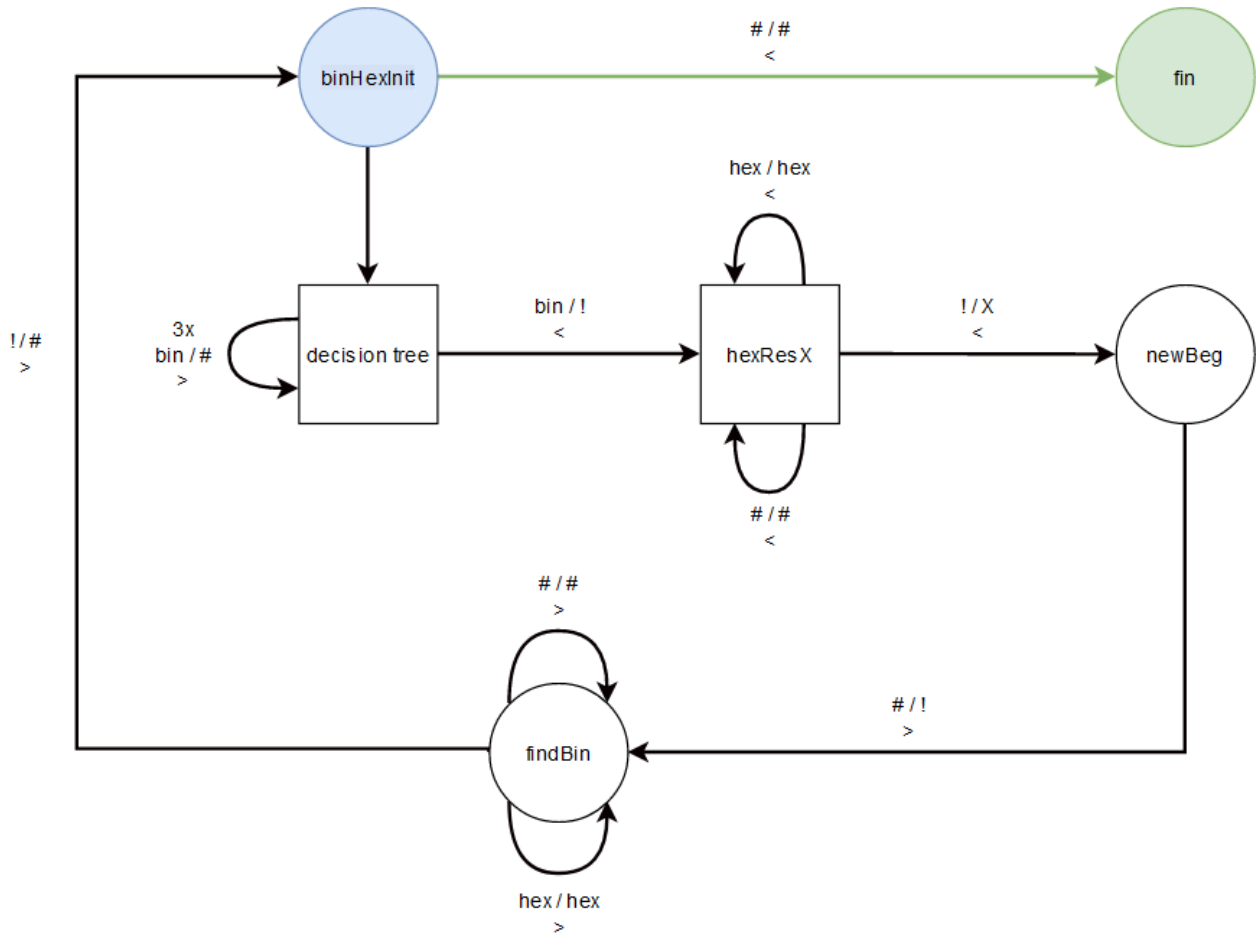
Algorytm konwersji został stworzony przez Maxa von Buelowa oraz kd3x i zaimplementowano go zgodnie z licencją CC BY-NC-SA 3.0. Przybliżona praca algorytmu:

1. Dzielenie liczby.
2. Zaprzeszanie dzielenie po odczytaniu hasha
3. Przesuwanie się w prawo aż do znalezienia hasha
4. Wejście w jedną z dwóch rodzin stanów przenoszącą wynik (na podstawie odczytanej reszty z dzielenia) lub przejście do kolejnej funkcji
5. Przesuwanie się do znalezienia hasha
6. Zapisanie wyniku (0 lub 1)
7. Szukanie hasha przesuwając się w lewo
8. Zapisanie 0
9. Powrót do punktu 1



### 3.4 Funkcja *bin to hex*

Konwersja bin to hex jest oparta na drzewie decyzyjnym, które pełni rolę "pamięci". Każdy bit odczytywanego bajta może mieć wartość 1 lub 0, w związku z czym tworząc przejścia stanów można kontrolować sekwencje odczytanych znaków.



Kwadrat oznacza zbiór rodzin z licznymi stanami, ale które mają tę samą zasadę działania. *hexResX* oznacza przenoszenie wyniku podobne do *dec to bin* z zastrzeżeniem, że *X* to znak określający wartość otrzymaną w drzewie decyzyjnym.

Symbol ! służy do zaznaczania "końca" hexów oraz początku ciągu bajtów binarnych. Został użyty by uniknąć przechodzenia Maszyny w nieskończoną pętlę

---

## 4 Podsumowanie

---

Translator działa zgodnie z założeniami i jest w stanie zrealizować dowolny program (zapisany poprawną składnią). Możliwe udoskonalenia do wprowadzenia: zabezpieczenia w przypadku nieistnienia plików, których ścieżkę podaje użytkownik; zabezpieczenie zmiennej `clear_sym`; dodanie testów. Pamięć może być potencjalnym problemem, niektóre algorytmy mogą generować dużo pustych znaków na taśmie, by uniknąć problemu można zaimplementować coś w rodzaju pamięci cache

Algorytm działa poprawnie również dla dużych liczb. Plik zawiera ponad 500 definicji stanów, szczególnie nieoptymalną metodą (pod względem objętościowym) wydaje się konwersja bin to hex.

