🚀 ASM Chart Simulator

درس طراحی سیستم‌های دیجیتال
دکتر امین فصحتی

> *Amirreza Ghadyani* `99109206`
> *Alireza Habibzadeh* `99109393`

> *July 2022*

معرفی

شبیه‌ساز پیاده شده در در واقع فاز دوم پروژه‌ی اولیه یعنی Vmake ASM Chart Compiler است. این برنامه که باز هم با پایتون پیاده شده به طور ساده ASM Chart ورودی را شبیه‌سازی می‌کند.
ورودی تابع اصلی برنامه این بار هم یک گراف است که به صورت متنی پیاده می‌شود و باز هم برنامه رابط کاربری ندارد.

منطق کلی شبیه‌ساز به صورت یک تابع شبیه‌ساز تک سیکل (one cycle) با نام `simulate_one_cycle` و یک تابع driver با نام `simulate` است. که حال کارکرد آن‌ها را شرح می‌دهیم:

One Cycle Simulation

در این مرحله‌ی کار کافی است تنها یک سیکل از ماشین را شبیه‌سازی کنیم. ورودی‌ها و خروجی‌های ماژول به صورت یک dictionary مشخص گرفته شده‌اند. دلیل این که خروجی‌ها هم به صورت dictionary هستند و مقدار دارند این است که خروجی‌ها همگی رجیستر فرض شده‌اند و مقادیر پیشینی دارند که در روند برنامه موثر است. تنها در اولین state ماشین باید این رجیسترهای خروجی برابر با صفر شوند که آن کار قسمت بعدی است. فعلا تا اینجای کار با ورودی‌ها و خروجی‌ها می‌توان یکسان برخورد کرد.

ساختار تابع اصلی ما به این صورت است: البته از توابع دیگری نیز استفاده شده که در کد کامل ارسالی قابل مشاهده‌اند)

```
def simulate_one_cycle(node: Node, input_dict: dict = None, output_dict: dict = None):
    if output_dict is None:
        output_dict = {}
    output_list = check_input(output_dict.keys())

    if input_dict is None:
        input_dict = {}
    check_input(input_dict.keys())

```

```
    for input_name in input_dict:
        exec(f'{input_name} = {input_dict[input_name]}')
    for output_name in output_list:
        exec(f'{output_name} = {output_dict[output_name]}')

    if node.type != NodeType.STATE:
        raise ValueError('Root node must be a state')

    node.data = rename_verilog(node.data)
    exec(node.data)
    node = node.next

    while node.type != NodeType.STATE:
        if node.type == NodeType.CONDITION:
            if eval(node.data):
                node = node.next
            else:
                node = node.next_else
        else:
            node.data = rename_verilog(node.data)
            exec(node.data)
            node = node.next

    for output_name in output_list:
        try:
            output_dict[output_name] = eval(output_name)
        except NameError:
            pass
    return node, output_dict
```

Multicycle Simulation

برای شبیه‌سازی با تعدا سیکل دلخوای کافی است تابع شبیه‌سازی تک‌سیکل را به شکل مناسبی drive کنیم. باید خروجی‌های هر کلاک را تا کلاک بعدی نگه داشته و دوباره به تابع بدهیم تا بتواند از آن‌ها استفاده کند. در ابتدا هم مقادیر رجیسترها صفر در نظر گرفته شده.

```python
def simulate(root: Node, cycles=1, input_dict: dict = None, output_list: list = None):
    node = root
    if output_list is None:
        output_list = []
    output_dict = dict.fromkeys(output_list, 0)
    for i in range(cycles):
        node, output_dict = simulate_one_cycle(node, input_dict, output_dict)
        print(output_dict)
        input_dict = update_inputs(input_dict, output_dict, node)
```

برای آشنایی با نحوه‌ی ورودی دادن و کار با برنامه می‌توانید فایل `tests.py` که در پروژه قرار دارد را ببینید. در این فایل، نمونه‌هایی که در انتهای این گزارش می‌بینیم پیاده شده‌اند و به صورت unit test استاندارد پایتون قابل اجرا هستند.

امکانات

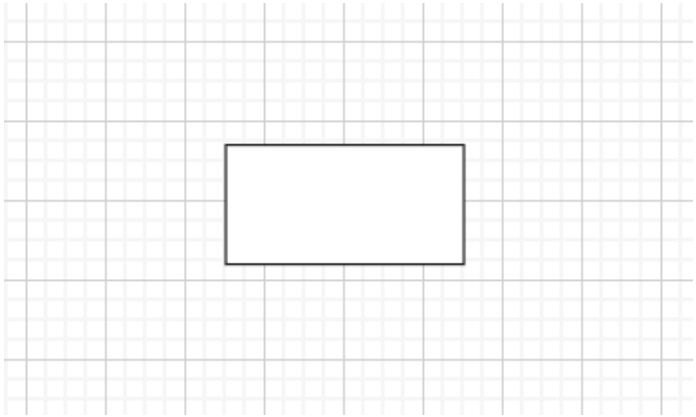
- تشخیص کاراکترهای غیرمجاز مانند “,” و “;” در نام ورودی/خروجی‌ها و اعلام آن
- تشخیص اسامی غیرمجاز مانند begin و end و if در نام ورودی/خروجی‌ها و اعلام آن
- امکان استفاده از شرط‌های تو در تو و هر نوع ترکیبی از و Decision boxها، Condition boxها و State boxها وجود دارد. تا وقتی که chartما از نظر قوانین طراحی ASM chart مجاز باشد شرط‌ها و نتایج آن‌ها به صورت تودرتو simulate می‌شوند.
-

تست‌ها و نمونه‌کار

برای کامپایلر در ابتدا چند تست ساده تعریف شده و در انتها به عنوان نمونه **ASM chart تمرین شماره 5 درس و یک ماژول محاسبه‌ی اعداد فیبوناچی** توسط برنامه simulate شده‌اند.

Do Nothing Test

تست خالی برای چک کردن اجرایی بودن ساختار کلی برنامه.

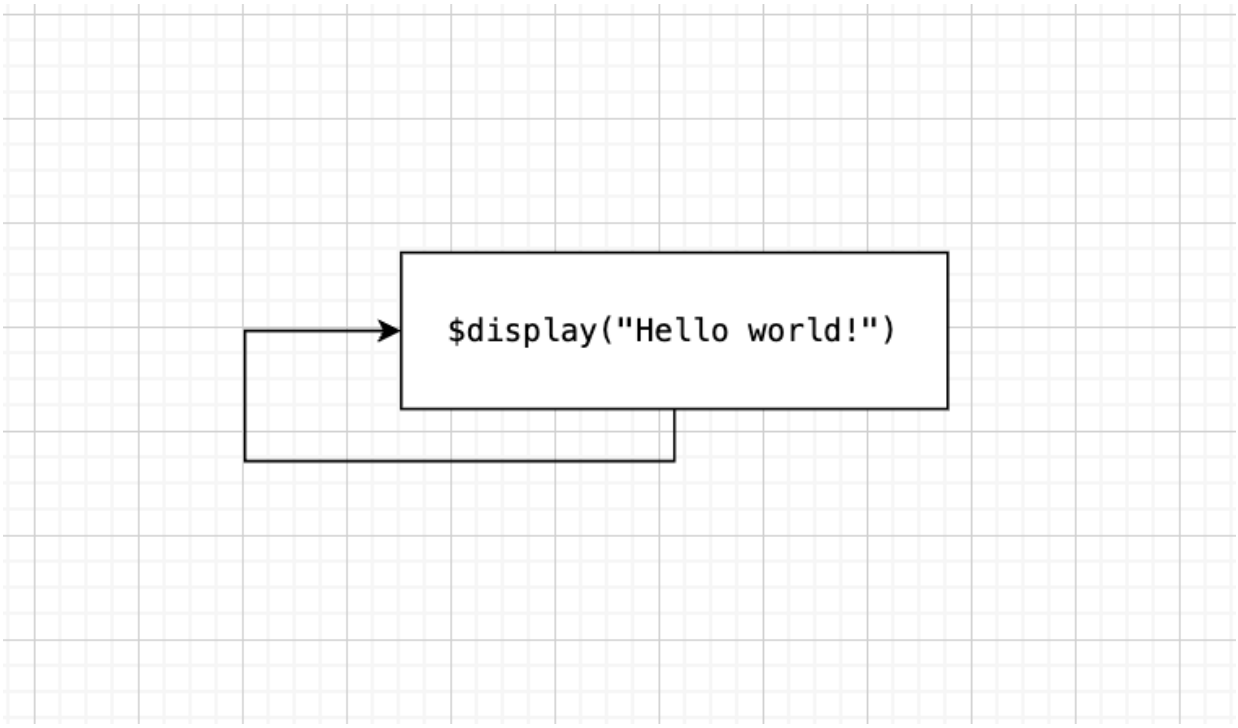


```
Ran 1 test in 0.001s

OK

Process finished with exit code 0
```

Hello world! (10 clocks)




```
Ran 1 test in 0.001s

OK
Hello world!
```

```
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
Hello world!
None

Process finished with exit code 0
```

Illegal name 1

تست اسم غیرمجاز

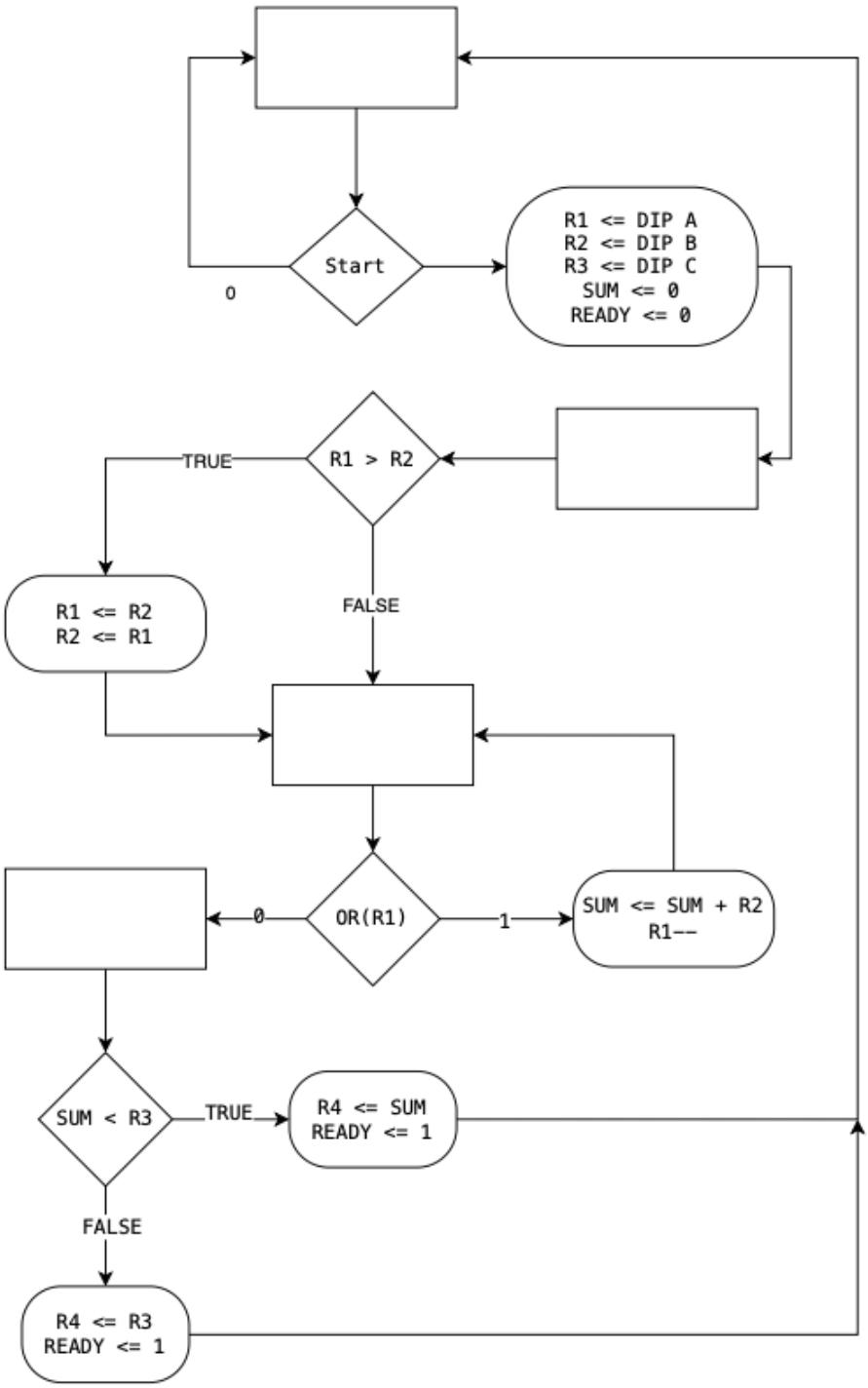
```
ValueError: p_state is a forbidden name
```

Illegal name 2

تست کاراکتر غیرمجاز در اسم

```
ValueError: my;input contains a forbidden character ";"
```

Homework 5



Input ASM Chart

تمرین 5 درس که در فایل tests.py توسط برنامه شبیه‌سازی شده.

در ورودی به برنامه در پایین، تمام non-blocking statementها با متغییرهای موقت به blocking statement تبدیل شده‌اند و متغییر `SUM` به دلیل تداخل داشتن با تابع `sum` پایتون به `TOTAL` تغییر نام داده شده است.

```
def test_homework_5(self):
```

```
    state1 = Node('', NodeType.STATE)
    cond1 = Node('START == 1', NodeType.CONDITION)
    dec1 = Node('R1 = DIP_A\n'
                'R2 = DIP_B\n'
                'R3 = DIP_C\n'
                'TOTAL = 0\n'
                'READY = 0', NodeType.DECISION)

    state2 = Node('', NodeType.STATE)
    cond2 = Node('R1 > R2', NodeType.CONDITION)
    dec2 = Node('temp = R1\n'
                'R1 = R2\n'
                'R2 = temp', NodeType.DECISION)

    state3 = Node('', NodeType.STATE)
    cond3 = Node('R1', NodeType.CONDITION)
    dec3 = Node('TOTAL = TOTAL + R2\n'
                'R1 = R1 - 1', NodeType.DECISION)

    state4 = Node('', NodeType.STATE)
    cond4 = Node('TOTAL < R3', NodeType.CONDITION)
    dec4 = Node('R4 = TOTAL\n'
                'READY = 1', NodeType.DECISION)
    dec5 = Node('R4 = R3\n'
                'READY = 1', NodeType.DECISION)

    state1.next = cond1
    cond1.next = dec1
    cond1.next_else = state1
    dec1.next = state2
    state2.next = cond2
    cond2.next = dec2
    cond2.next_else = state3
    dec2.next = state3
    state3.next = cond3
    cond3.next = dec3
```

```
    dec3.next = state3
    cond3.next_else = state4
    state4.next = cond4
    cond4.next = dec4
    cond4.next_else = dec5
    dec4.next = state1
    dec5.next = state1

    simulate(state1, 20, {'DIP_A': 54, 'DIP_B': 12, 'DIP_C': 37, 'START': 1}, ['R1', 'R2', 'R3', 'TOTAL', 'READY', 'R4'])
```

در تابع شبیه‌ساز چندسیکلی یک print اضافه شده تا پس از هر کلاک خروجی‌های مدار نشان داده شود. نتیجه مطابق زیر است:

```
{'R1': 54, 'R2': 12, 'R3': 37, 'TOTAL': 0, 'READY': 0, 'R4': 0}
{'R1': 12, 'R2': 54, 'R3': 37, 'TOTAL': 0, 'READY': 0, 'R4': 0}
{'R1': 11, 'R2': 54, 'R3': 37, 'TOTAL': 54, 'READY': 0, 'R4': 0}
{'R1': 10, 'R2': 54, 'R3': 37, 'TOTAL': 108, 'READY': 0, 'R4': 0}
{'R1': 9, 'R2': 54, 'R3': 37, 'TOTAL': 162, 'READY': 0, 'R4': 0}
{'R1': 8, 'R2': 54, 'R3': 37, 'TOTAL': 216, 'READY': 0, 'R4': 0}
{'R1': 7, 'R2': 54, 'R3': 37, 'TOTAL': 270, 'READY': 0, 'R4': 0}
{'R1': 6, 'R2': 54, 'R3': 37, 'TOTAL': 324, 'READY': 0, 'R4': 0}
{'R1': 5, 'R2': 54, 'R3': 37, 'TOTAL': 378, 'READY': 0, 'R4': 0}
{'R1': 4, 'R2': 54, 'R3': 37, 'TOTAL': 432, 'READY': 0, 'R4': 0}
{'R1': 3, 'R2': 54, 'R3': 37, 'TOTAL': 486, 'READY': 0, 'R4': 0}
{'R1': 2, 'R2': 54, 'R3': 37, 'TOTAL': 540, 'READY': 0, 'R4': 0}
{'R1': 1, 'R2': 54, 'R3': 37, 'TOTAL': 594, 'READY': 0, 'R4': 0}
{'R1': 0, 'R2': 54, 'R3': 37, 'TOTAL': 648, 'READY': 0, 'R4': 0}
{'R1': 0, 'R2': 54, 'R3': 37, 'TOTAL': 648, 'READY': 0, 'R4': 0}
{'R1': 0, 'R2': 54, 'R3': 37, 'TOTAL': 648, 'READY': 1, 'R4': 37}
{'R1': 54, 'R2': 12, 'R3': 37, 'TOTAL': 0, 'READY': 0, 'R4': 37}
{'R1': 12, 'R2': 54, 'R3': 37, 'TOTAL': 0, 'READY': 0, 'R4': 37}
{'R1': 11, 'R2': 54, 'R3': 37, 'TOTAL': 54, 'READY': 0, 'R4': 37}
{'R1': 10, 'R2': 54, 'R3': 37, 'TOTAL': 108, 'READY': 0, 'R4': 37}


Ran 1 test in 0.004s
```

```
OK

Process finished with exit code 0
```

برای اجرای این تست کافی است تست با همین نام در فایل `tests.py` را اجرا کنید. همچنین می‌توان اعداد ورودی را تغییر داد. اینجا اعداد طوری داده شده‌اند که هم آن تعویض مورد نیاز برای بهینه‌شدن کلاک‌ها صورت بگیرد و هم نتیجه از `DIP_C` بیشتر شود و خود `DIP_C` در خروجی (`R4`) بشیند. همانطور که می‌بینیم مقدار `TOTAL` در نهایت برابر با 648 شده که همان 54×12 است.

آن خطوطی که پس از `READY = 1` برنامه همچنان به کار خود ادامه داده به این دلیل است که سیگنال `START` ما یک باقی مانده و برنامه دوباره دارد همان کار قبلی را می‌کند.

اجرا با یک ورودی دیگر:

Test

```
simulate(state1, 23, {'DIP_A': 15, 'DIP_B': 103, 'DIP_C': 2000, 'STAR
T': 1},
         ['R1', 'R2', 'R3', 'TOTAL', 'READY', 'R4'])
```

Output

```
{'R1': 15, 'R2': 103, 'R3': 2000, 'TOTAL': 0, 'READY': 0, 'R4': 0}
{'R1': 15, 'R2': 103, 'R3': 2000, 'TOTAL': 0, 'READY': 0, 'R4': 0}
{'R1': 14, 'R2': 103, 'R3': 2000, 'TOTAL': 103, 'READY': 0, 'R4': 0}
{'R1': 13, 'R2': 103, 'R3': 2000, 'TOTAL': 206, 'READY': 0, 'R4': 0}
{'R1': 12, 'R2': 103, 'R3': 2000, 'TOTAL': 309, 'READY': 0, 'R4': 0}
{'R1': 11, 'R2': 103, 'R3': 2000, 'TOTAL': 412, 'READY': 0, 'R4': 0}
{'R1': 10, 'R2': 103, 'R3': 2000, 'TOTAL': 515, 'READY': 0, 'R4': 0}
{'R1': 9, 'R2': 103, 'R3': 2000, 'TOTAL': 618, 'READY': 0, 'R4': 0}
{'R1': 8, 'R2': 103, 'R3': 2000, 'TOTAL': 721, 'READY': 0, 'R4': 0}
{'R1': 7, 'R2': 103, 'R3': 2000, 'TOTAL': 824, 'READY': 0, 'R4': 0}
{'R1': 6, 'R2': 103, 'R3': 2000, 'TOTAL': 927, 'READY': 0, 'R4': 0}
{'R1': 5, 'R2': 103, 'R3': 2000, 'TOTAL': 1030, 'READY': 0, 'R4': 0}
{'R1': 4, 'R2': 103, 'R3': 2000, 'TOTAL': 1133, 'READY': 0, 'R4': 0}
{'R1': 3, 'R2': 103, 'R3': 2000, 'TOTAL': 1236, 'READY': 0, 'R4': 0}
{'R1': 2, 'R2': 103, 'R3': 2000, 'TOTAL': 1339, 'READY': 0, 'R4': 0}
{'R1': 1, 'R2': 103, 'R3': 2000, 'TOTAL': 1442, 'READY': 0, 'R4': 0}
{'R1': 0, 'R2': 103, 'R3': 2000, 'TOTAL': 1545, 'READY': 0, 'R4': 0}
{'R1': 0, 'R2': 103, 'R3': 2000, 'TOTAL': 1545, 'READY': 0, 'R4': 0}
```

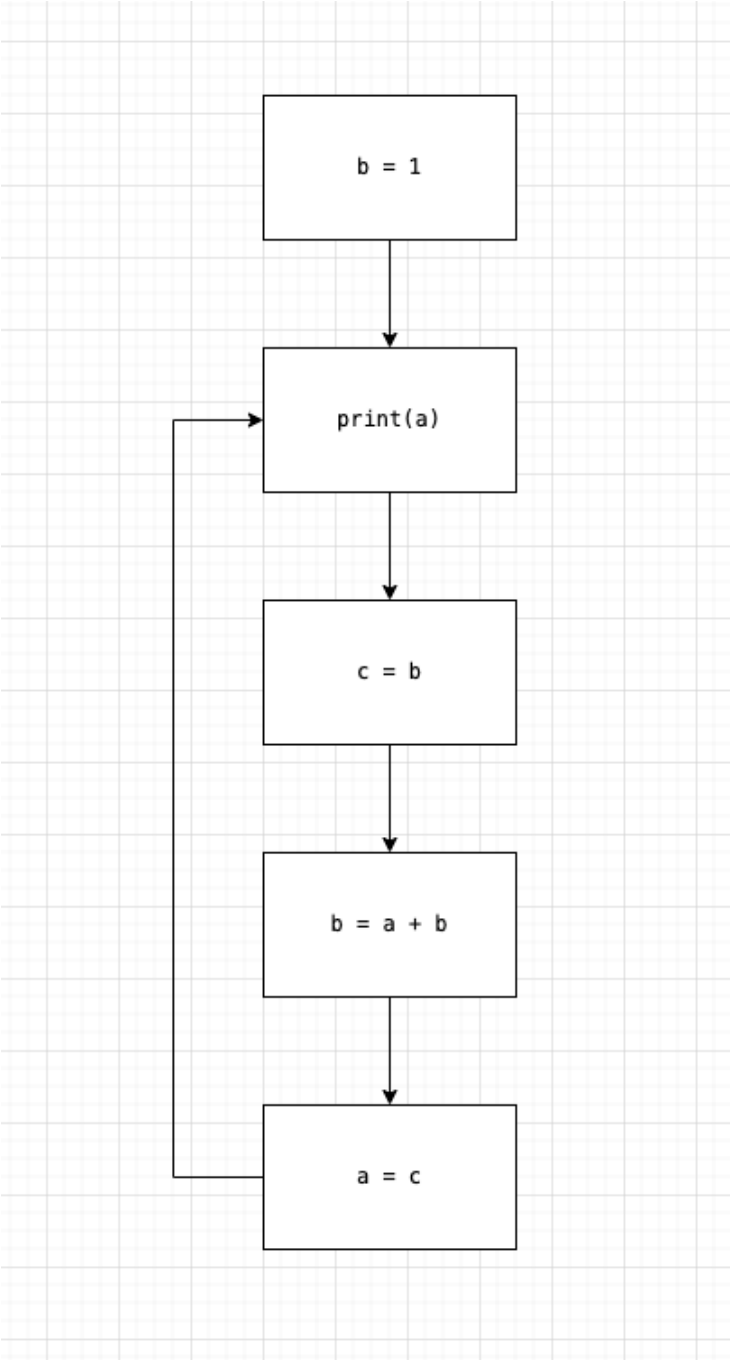
```
{'R1': 0, 'R2': 103, 'R3': 2000, 'TOTAL': 1545, 'READY': 1, 'R4': 1545}
{'R1': 15, 'R2': 103, 'R3': 2000, 'TOTAL': 0, 'READY': 0, 'R4': 1545}
{'R1': 15, 'R2': 103, 'R3': 2000, 'TOTAL': 0, 'READY': 0, 'R4': 1545}
{'R1': 14, 'R2': 103, 'R3': 2000, 'TOTAL': 103, 'READY': 0, 'R4': 1545}
{'R1': 13, 'R2': 103, 'R3': 2000, 'TOTAL': 206, 'READY': 0, 'R4': 1545}


Ran 1 test in 0.007s

OK

Process finished with exit code 0
```

Fibonacci



Fibonacci ASM Chart

Test

```
def test_fibonacci(self):
    state0 = Node('b = 1', NodeType.STATE)
    state1 = Node('print(a)', NodeType.STATE)
    state2 = Node('c = b', NodeType.STATE)
    state3 = Node('b = a + b', NodeType.STATE)
    state4 = Node('a = c', NodeType.STATE)

    state0.next = state1
```

```
    state1.next = state2
    state2.next = state3
    state3.next = state4
    state4.next = state1

    simulate(state0, 50, None, ['a', 'b', 'c'])
```

Output

```
Ran 1 test in 0.005s

OK
{'a': 0, 'b': 1, 'c': 0}
0
{'a': 0, 'b': 1, 'c': 0}
{'a': 0, 'b': 1, 'c': 1}
{'a': 0, 'b': 1, 'c': 1}
{'a': 1, 'b': 1, 'c': 1}
1
{'a': 1, 'b': 1, 'c': 1}
{'a': 1, 'b': 1, 'c': 1}
{'a': 1, 'b': 2, 'c': 1}
{'a': 1, 'b': 2, 'c': 1}
1
{'a': 1, 'b': 2, 'c': 1}
{'a': 1, 'b': 2, 'c': 2}
{'a': 1, 'b': 3, 'c': 2}
{'a': 2, 'b': 3, 'c': 2}
2
{'a': 2, 'b': 3, 'c': 2}
{'a': 2, 'b': 3, 'c': 3}
{'a': 2, 'b': 5, 'c': 3}
{'a': 3, 'b': 5, 'c': 3}
3
{'a': 3, 'b': 5, 'c': 3}
{'a': 3, 'b': 5, 'c': 5}
{'a': 3, 'b': 8, 'c': 5}
```

```
{'a': 5, 'b': 8, 'c': 5}
5
{'a': 5, 'b': 8, 'c': 5}
{'a': 5, 'b': 8, 'c': 8}
{'a': 5, 'b': 13, 'c': 8}
{'a': 8, 'b': 13, 'c': 8}
8
{'a': 8, 'b': 13, 'c': 8}
{'a': 8, 'b': 13, 'c': 13}
{'a': 8, 'b': 21, 'c': 13}
{'a': 13, 'b': 21, 'c': 13}
13
{'a': 13, 'b': 21, 'c': 13}
{'a': 13, 'b': 21, 'c': 21}
{'a': 13, 'b': 34, 'c': 21}
{'a': 21, 'b': 34, 'c': 21}
21
{'a': 21, 'b': 34, 'c': 21}
{'a': 21, 'b': 34, 'c': 34}
{'a': 21, 'b': 55, 'c': 34}
{'a': 34, 'b': 55, 'c': 34}
34
{'a': 34, 'b': 55, 'c': 34}
{'a': 34, 'b': 55, 'c': 55}
{'a': 34, 'b': 89, 'c': 55}
{'a': 55, 'b': 89, 'c': 55}
55
{'a': 55, 'b': 89, 'c': 55}
{'a': 55, 'b': 89, 'c': 89}
{'a': 55, 'b': 144, 'c': 89}
{'a': 89, 'b': 144, 'c': 89}
89
{'a': 89, 'b': 144, 'c': 89}
{'a': 89, 'b': 144, 'c': 144}
{'a': 89, 'b': 233, 'c': 144}
{'a': 144, 'b': 233, 'c': 144}
```

```
144
{'a': 144, 'b': 233, 'c': 144}

Process finished with exit code 0
```

Roadmap

رابطه‌ی گرافیکی (GUI)
بهتر است در آینده یک رابط گرافیکی درست کنیم تا با کامپایلر ما ارتباط برقرار کند و کاربر مستقیما به شکل گرافیکی ASM chart را در نرم‌افزار بکشد. البته یک راه ساده‌تر و اتفاقا بهتر این است که برنامه خروجی‌های استاندارد ASM chart را بتواند به عنوان ورودی دریافت و simulate کند. برای مثال کاربر در سایت https://app.diagrams.net یا نرم‌افزار VIsio چارت را می‌کشد و سپس خروجی آن را در برنامه‌ی ما import می‌کند.

Non-blocking statements

فعلا شبیه‌ساز ما همه‌ی statementها را blocking در نظر می‌گیرد. در آینده می‌توان این امکان را به برنامه اضافه کرد.