Language 414

Introduction

This specialized language aims to the demands of EE414 and EE514 courses at Northern Arizona University (NAU), alleviating the cumbersome process of manually crafting assembly instructions. Initially, I explored using 'gcc' for C language, but that is not a solution. Notably, the incomplete alignment between the course's instruction set and the full RISC-V set posed challenges. Additionally, 'gcc' lacked the capability to bind variables and registers for direct register access, further complicating matters.

Subsequently, I conceived Language 414 as a tailored solution for the class. Although Language 414 is extremely simple, dropping features such as functions, `for` loops, and string support, it simplifies the process of writing assembly instructions considerably.

Given the short timeline of this project (completed over a single weekend), it's conceivable that bugs may have eluded detection. Any assistance in identifying or rectifying these potential issues would be immensely appreciated.

Variables

No declaration is needed for variables. Variables are stored in the registers, typically `x1-x31`, `t0-t6`, `s0-s11`, and `a0-a7`. You can also modify it by editing the list at the top of the file 'Excuteable.py'.

Examples:

```
i = 0
a = (i + 12 + 4)
```

The name of a variable must start with a letter, '_' (underline) is not supported.

Assign

Usually, the compiler will allocate a register to the variable, but you can also specify which register for the variable by using `assign`. Three parts are included in an `assign` statement. The first part is the assign keyword used to indicate that this command is an `assign` statement. The second part is the register you want to allocate. The third part is the variable name. The location of the variable name and the register are interchangeable. Here are some examples.

```
assign SrcArray = reg(x8)
```

```
assign reg(x9) = target
```

While Loop

With the `while` loop we can execute a set of statements as long as a condition is `true`. For instance,

```
i = 0
while i < 12:
        a = (i + 12 + 4)
        i += 1
target = a
```

The `break` statement can stop a `while` loop manually.

```
i = 0
target = 1
while i < 6:
        target = 1 * 2
        if i == 3:
                break
        i += 1
```

With the `continue` statement, we can stop the current iteration, and continue with the next.

```
i = 0
target = 1
while i < 6:
        i += 1
        if i == 3:
                continue
        target = 1 * 2
```

Unlike some other language, like Python, The ‘while’ loop in language 414 don’t support `else` statement.

If…Else

Language 414 supports the usual logic conditions from mathematics:

- equals: `a == b`
- Not equals: `a != b`
- Less than: `a < b`
- Less than or equal to: `a <= b`
- Greater than: `a > b`
- Greater than or equal to: `a >= b`

The `or` and `and` keywords are also supported to combine conditional statements.

An 'if statement' is written by using the if keyword. The `else` keyword catches anything that isn't caught by the preceding conditions.

```
a = 5
b = 6
if b >= a:
        target = 0
else:
        target = 1
```

Language 414 doesn't support the `else if` keyword.

Indentation

Just like Python, Language 414 relies on indentation (whitespace at the beginning of the line) to determine the scope of the code. The only thing that the compiler examines is the count of the characters in the indentation, it doesn't distinguish whether the whitespace is a 'tab' or a 'space'. So, don't mix using 'tab' and 'space', that will cause chaos.

Array

An array is something like `A[234]`. Before using an array, you need to manually specify the first address of the array. When using an array, the program simply adds two variables together and loads the word in that address. The example below will store the '5' to the memory address 1012.

```
A = 1000
A[12] = 5
```

The codes below have the same effect.

```
A = 1000
12[A] = 5
```

```
12[1000] = 5
```

```
A = 1000
a = 12
A[a] = 5
```

Comment

Comments start with a #. Unlike most languages, the comment in Language 414 will not be ignored by the compiler, instead, the compiler will keep the comment and output it to the assemble instructions.

```
a = 5
b = 6
# Beginning of the If.
if b >= a and a < b:
        target = 0
else:
        target = 1
```

```
addi x1, x0, 5
addi x2, x0, 6
// Beginning of the If.
addi a7, x0, 0
blt x2, x1, BTMP0
addi a7, x0, 1
BTMP0:
```

There are some limitations of comments. You must write the comment in a separate line, a line only has comments, you can't write a comment at the end of a statement.

Multiline comments are not supported in Language 414.

Functions

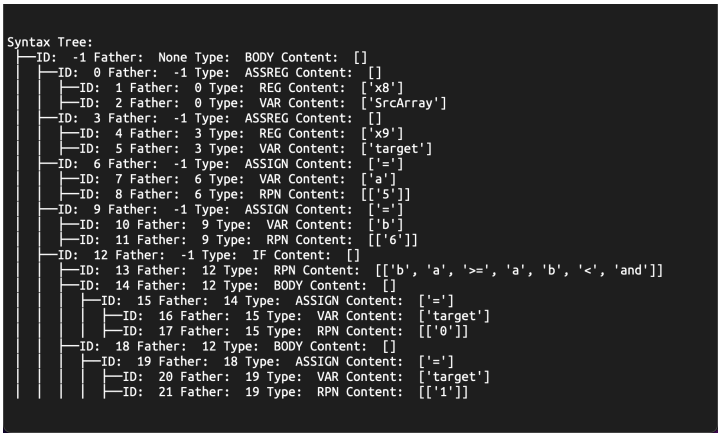
Language 414 doesn't support functions.

How to Use

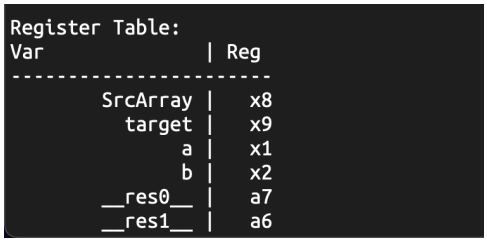
Put your source code in the same path as the program files and run ‘main.py’. Then input your source file name.

```
python3 main.py
Source File Name: test.414
```

The program will generate a ‘.out’ file, which includes the assembly instructions. Also, some information will be printed to the terminal window. The first part is the syntax tree, and the second part is a table that records which register is allocated to variables.



```
Syntax Tree:
├──ID:  -1 Father:  None Type:  BODY Content:  []
|  ├──ID:  0 Father:  -1 Type:  ASSREG Content:  []
|  |  ├──ID:  1 Father:  0 Type:  REG Content:  ['x8']
|  |  ├──ID:  2 Father:  0 Type:  VAR Content:  ['SrcArray']
|  ├──ID:  3 Father:  -1 Type:  ASSREG Content:  []
|  |  ├──ID:  4 Father:  3 Type:  REG Content:  ['x9']
|  |  ├──ID:  5 Father:  3 Type:  VAR Content:  ['target']
|  ├──ID:  6 Father:  -1 Type:  ASSIGN Content:  ['=']
|  |  ├──ID:  7 Father:  6 Type:  VAR Content:  ['a']
|  |  ├──ID:  8 Father:  6 Type:  RPN Content:  [['5']]
|  ├──ID:  9 Father:  -1 Type:  ASSIGN Content:  ['=']
|  |  ├──ID:  10 Father:  9 Type:  VAR Content:  ['b']
|  |  ├──ID:  11 Father:  9 Type:  RPN Content:  [['6']]
|  ├──ID:  12 Father:  -1 Type:  IF Content:  []
|  |  ├──ID:  13 Father:  12 Type:  RPN Content:  [['b', 'a', '>=', 'a', 'b', '<', 'and']]
|  |  ├──ID:  14 Father:  12 Type:  BODY Content:  []
|  |  |  ├──ID:  15 Father:  14 Type:  ASSIGN Content:  ['=']
|  |  |  |  ├──ID:  16 Father:  15 Type:  VAR Content:  ['target']
|  |  |  |  ├──ID:  17 Father:  15 Type:  RPN Content:  [['0']]
|  |  ├──ID:  18 Father:  12 Type:  BODY Content:  []
|  |  |  ├──ID:  19 Father:  18 Type:  ASSIGN Content:  ['=']
|  |  |  |  ├──ID:  20 Father:  19 Type:  VAR Content:  ['target']
|  |  |  |  ├──ID:  21 Father:  19 Type:  RPN Content:  [['1']]
```

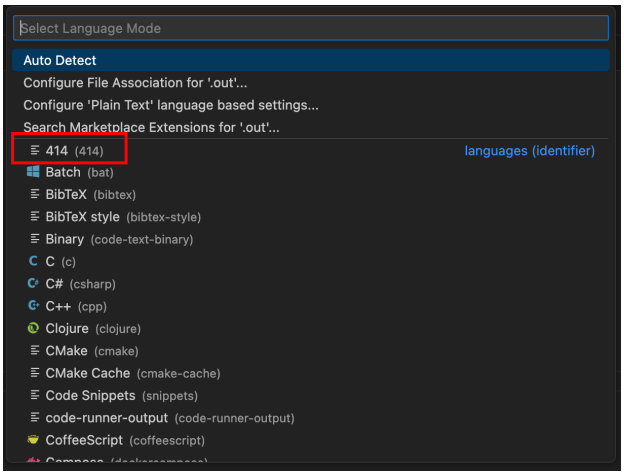


```
Register Table:
Var              | Reg
-----------------------
        SrcArray |   x8
          target |   x9
               a |   x1
               b |   x2
        __res0__ |   a7
        __res1__ |   a6
```

Visual Studio Code Syntax Highlight

You can manually install an extension to highlight syntax, like keywords, brackets, and numbers. The extension only highlights the syntax and will not check your grammar issues.

To automatically enable highlight syntax, your source file name should end with ‘.414’, so that VS Code can know this is a text file written in Language 414. You can also manually choose the 414 in the VS Code.



To install the extension, just copy the ‘language-414’ folder to the ‘~/.vscode/extensions’ and relaunch the VS Code.