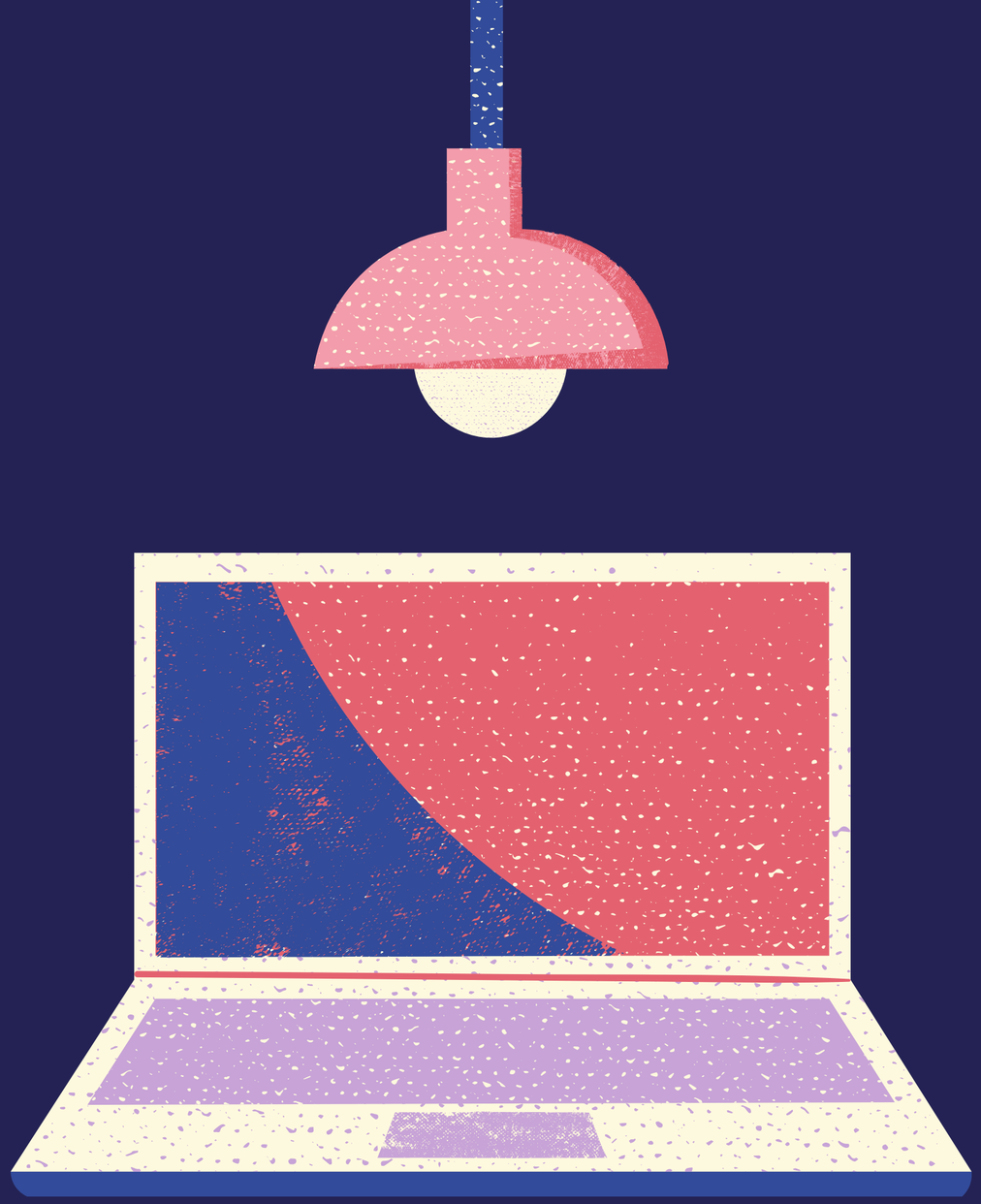


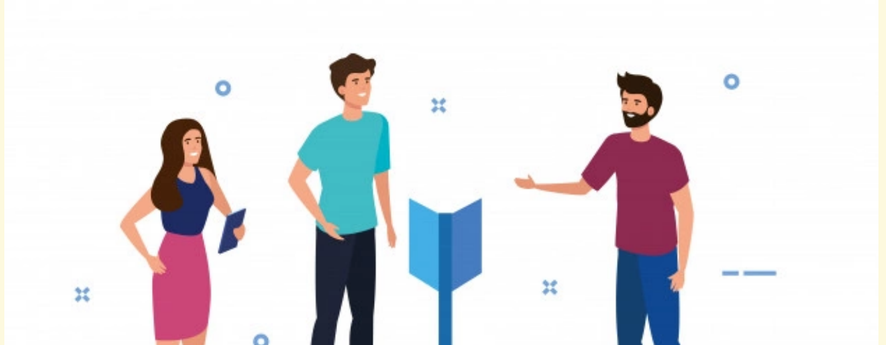
# ALPHAX compiler

Second delivery

# DEVELOPERS

ALPHAX Team

- Flores Constantino Diego
- Rojas Castañeda Karen Arleth



# Changes.

Tree unary operators were implemented

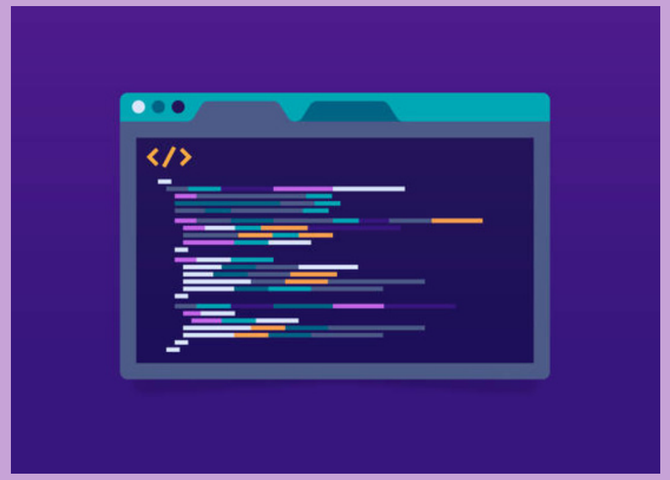
- Negation (-)

- Bitwise complement (~)

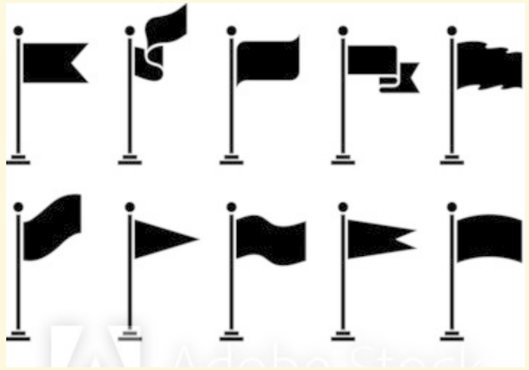
- Logical negation (!)

# AlphaX Compiler

alphaX
Digital Services



# Implementation



# Flags input

```
flore@LAPTOP-DLMCUKVT MINGW64 ~/Desktop/alphax1/Alphax/alphax_compiler (main)
$ ./Alphax -h
Available options:

      -c <filename.c>  Compile program (check the same folder for [filename].exe).
      -t <filename.c>  Show token list.
      -a <filename.c>  Show AST.
      -s <filename.c>  Show assembler code.
      -o <filename.c>  [newName] | Compile the program with a new name.
```

# Basic Compilation

```
flore@LAPTOP-DLMCUKVT MINGW64 ~/Desktop/alphax1/Alphax/alphax_compiler (main)
$ ./Alphax -c main.c
Compiling the file: main.c
Assembly code Generated : ./main.s
Exectutable generated: ./main
```

# Token List

Bitwise (~)

```
{:type, 1, [:intKeyWord]},
{:ident, 1, [:mainKeyWord]},
{:lParen, 1, []},
{:rParen, 1, []},
{:lBrace, 1, []},
{:ident, 2, [:returnKeyWord]},
{:operator, 2, [:bitW]},
{:num, 2, 12},
{:semicolon, 2, []},
{:rBrace, 3, []}
```

New tokens

Negation (-)

```
[
  {:type, 1, [:intKeyWord]},
  {:ident, 1, [:mainKeyWord]},
  {:lParen, 1, []},
  {:rParen, 1, []},
  {:lBrace, 1, []},
  {:ident, 2, [:returnKeyWord]},
  {:operator, 2, [:negation]},
  {:num, 2, 5},
  {:semicolon, 2, []},
  {:rBrace, 3, []}
]
```

# Token List

Logical negation (!)

```
[
  {:type, 1, [:intKeyWord]},
  {:ident, 1, [:mainKeyWord]},
  {:lParen, 1, []},
  {:rParen, 1, []},
  {:lBrace, 1, []},
  {:ident, 2, [:returnKeyWord]},
  {:operator, 2, [:logicalN]},
  {:num, 2, 5},
  {:semicolon, 2, []},
  {:rBrace, 3, []}
]
```

New token

# Abstract Syntax Tree

## Bitwise (~)

```
%AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 12
        },
        node_name: :unary,
        right_node: nil,
        value: :bitW
      },
      node_name: :return,
      right_node: nil,
      value: :return
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

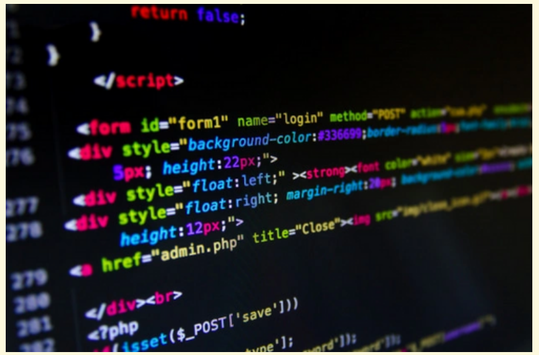
## Negation (-)

```
%AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 5
        },
        node_name: :unary,
        right_node: nil,
        value: :negation
      },
      node_name: :return,
      right_node: nil,
      value: :return
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

## Logical negation (!)

```
%AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 5
        },
        node_name: :unary,
        right_node: nil,
        value: :logicalN
      },
      node_name: :return,
      right_node: nil,
      value: :return
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

# Assembler Code



Bitwise (~)

```
$ ./Alphax -s bit_wise.c
Assembly code



    .section        __TEXT,__text,regular,pure_instructions
    .p2align        4, 0x90
    .globl  _main           ## -- Begin function main
_main:                      ## @main
    movl    12, %rax
    not     %rax
    ret
```

# Assembler Code

Negation (-)

```
$ ./Alphax -s neg.c
Assembly code

    .section        __TEXT,__text,regular,pure_instructions
    .p2align        4, 0x90
    .globl  _main           ## -- Begin function main
_main:                      ## @main
    movl    $5, %eax
    neg     %eax
    ret
```

# Assembler Code

Logical Negation (!)

```
$ ./Alphax -s not_five.c
Assembly code



    .section        __TEXT,__text,regular,pure_instructions
    .p2align        4, 0x90
    .globl  _main           ## -- Begin function main
_main:                      ## @main
    movl    5, %rax
    cmpl    $0, %rax
    movl    $0, %rax
    sete    %al
    ret
```

# Test Plan

```
Arleth@DESKTOP-KRHDMLS MINGW64 ~/desktop/alphax_compiler/alphax_compiler3E (mast
er)
$ mix test
.....................................

Finished in 0.7 seconds
39 tests, 0 failures

Randomized with seed 284000
```

# Test plan

To pass

```
int main() {
    return !5;
}
```

```
int main() {
    return ~12;
}
```

```
int main() {
    return !-3;
}
```

```
int main() {
    return !0;
}
```

# Test plan

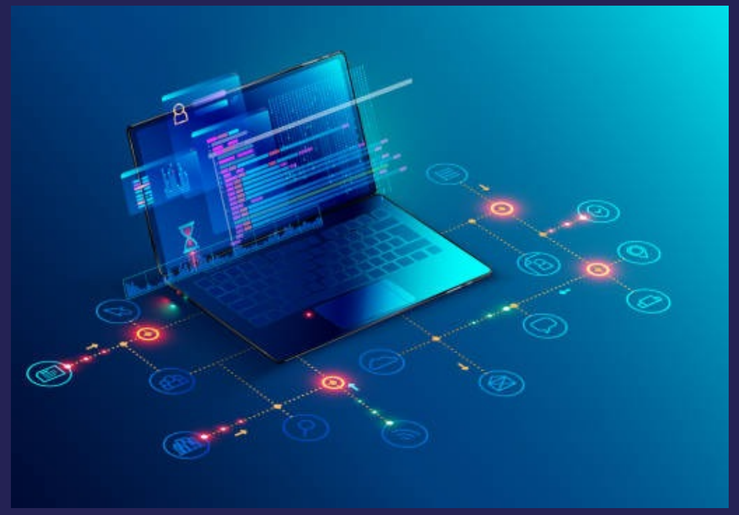
To fail

```
int main() {
    return !;
}
```

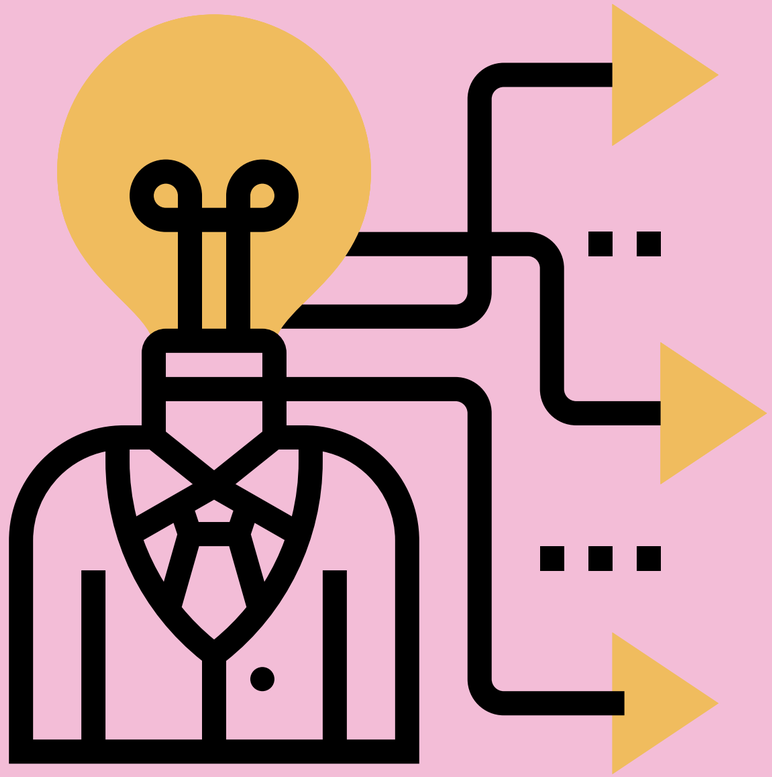
```
int main() {
    return !5
}
```

```
int main() {
    return !~;
}
```

```
int main() {
    return 4-;
}
```



# Third delivery



# Changes.

Four binary operators were implemented

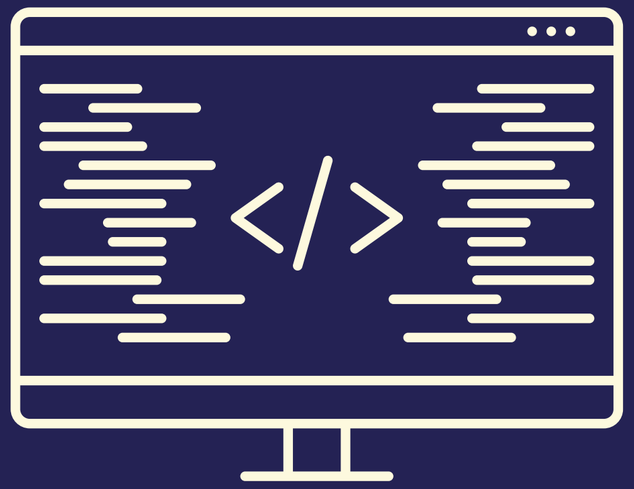
- Addition (+)

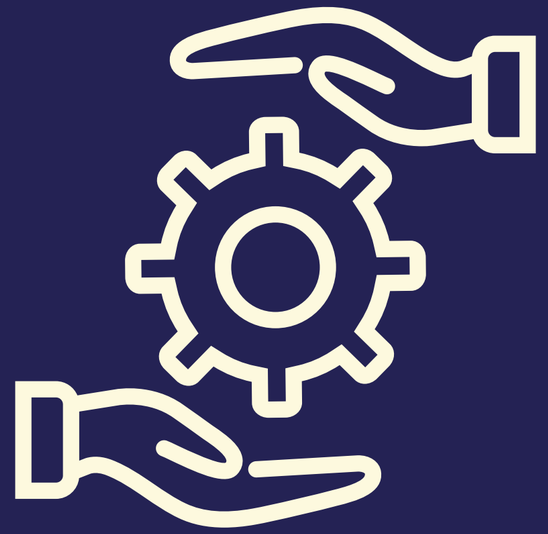
- Subtraction (-)

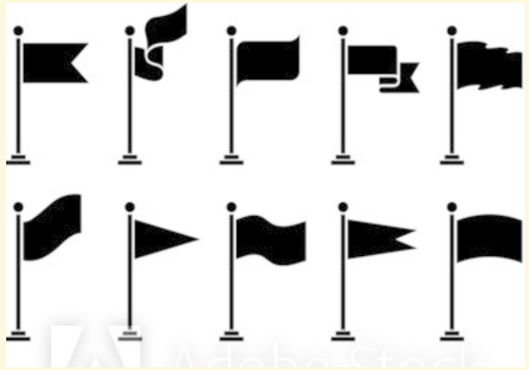
- Multiplication (*)

- Division (/)

# Implementation







# Flags input

```
flore@LAPTOP-DLMCUKVT MINGW64 ~/Desktop/alphax1/Alphax/alphax_compiler (main)
$ ./Alphax -h
Available options:

      -c <filename.c>  Compile program (check the same folder for [filename].exe).
      -t <filename.c>  Show token list.
      -a <filename.c>  Show AST.
      -s <filename.c>  Show assembler code.
      -o <filename.c>  [newName] | Compile the program with a new name.
```

# Basic Compilation

```
flore@LAPTOP-DLMCUKVT MINGW64 ~/Desktop/alphax1/Alphax/alphax_compiler (main)
$ ./Alphax -c main.c
Compiling the file: main.c
Assembly code Generated : ./main.s
Exectutable generated: ./main
```

# Token List

```
[
  {:type, 1, [:intKeyWord]},
  {:ident, 1, [:mainKeyWord]},
  {:lParen, 1, []},
  {:rParen, 1, []},
  {:lBrace, 1, []},
  {:ident, 2, [:returnKeyWord]},
  {:num, 2, 3},
  {:operator, 2, [:negation]},
  {:num, 2, 4},
  {:operator, 2, [:addition]},
  {:num, 2, 10},
  {:operator, 2, [:division]},
  {:num, 2, 5},
  {:operator, 2, [:addition]},
  {:num, 2, 1},
  {:operator, 2, [:multiplication]},
  {:num, 2, 7},
  {:semicolon, 2, []},
  {:rBrace, 3, []}
]
```

New tokens

Addition (+)

```
[
  {:type, 1, [:intKeyWord]},
  {:ident, 1, [:mainKeyWord]},
  {:lParen, 1, []},
  {:rParen, 1, []},
  {:lBrace, 1, []},
  {:ident, 2, [:returnKeyWord]},
  {:num, 2, 1},
  {:operator, 2, [:addition]},
  {:num, 2, 2},
  {:semicolon, 2, []},
  {:rBrace, 3, []}
]
```

# Token List

Multiplication (*)

```
[
  {:type, 1, [:intKeyWord]},
  {:ident, 1, [:mainKeyWord]},
  {:lParen, 1, []},
  {:rParen, 1, []},
  {:lBrace, 1, []},
  {:ident, 2, [:returnKeyWord]},
  {:num, 2, 2},
  {:operator, 2, [:multiplication]},
  {:num, 2, 3},
  {:semicolon, 2, []},
  {:rBrace, 3, []}
]
```

New tokens

Division (/)

```
[
  {:type, 1, [:intKeyWord]},
  {:ident, 1, [:mainKeyWord]},
  {:lParen, 1, []},
  {:rParen, 1, []},
  {:lBrace, 1, []},
  {:ident, 2, [:returnKeyWord]},
  {:num, 2, 4},
  {:operator, 2, [:division]},
  {:num, 2, 2},
  {:semicolon, 2, []},
  {:rBrace, 3, []}
]
```

# Abstract Syntax Tree

## Addition (+)

```
%AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 1
        },
        node_name: :binary,
        right_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 2
        },
        value: :addition
      },
      node_name: :return,
      right_node: nil,
      value: :return
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

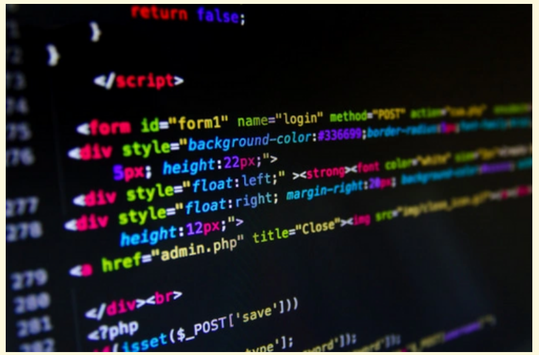
## Multiplication (*)

```
%AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 2
        },
        node_name: :binary,
        right_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 3
        },
        value: :multiplication
      },
      node_name: :return,
      right_node: nil,
      value: :return
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

## Division (/)

```
%AST{
  left_node: %AST{
    left_node: %AST{
      left_node: %AST{
        left_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 4
        },
        node_name: :binary,
        right_node: %AST{
          left_node: nil,
          node_name: :constant,
          right_node: nil,
          value: 2
        },
        value: :division
      },
      node_name: :return,
      right_node: nil,
      value: :return
    },
    node_name: :function,
    right_node: nil,
    value: :main
  },
  node_name: :program,
  right_node: nil,
  value: nil
}
```

# Assembler Code



Addition (+)

```
    .section        __TEXT,__text,regular,pure_instructions
    .p2align        4, 0x90
    .globl  _main           ## -- Begin function main
_main:                      ## @main
    movl    1, %rax
    push    %rax
    movl    2, %rax
    pop    %rbx
    pop     %rax
    add     %rax, %rcx
    push    %rax
    pop    %rbx
    ret
    push    %rax
    pop    %rbx
    push    %rax
    pop    %rbx
```

# Assembler Code

Multiplication (*)

```
    .section        __TEXT,__text,regular,pure_instructions
    .p2align        4, 0x90
    .globl  _main           ## -- Begin function main
_main:                      ## @main
    movl    2, %rax
    push    %rax
    movl    3, %rax
    pop    %rbx
    imul    %rbx, %rax
    push    %rax
    pop    %rbx
    ret
    push    %rax
    pop    %rbx
    push    %rax
    pop    %rbx
```

# Assembler Code

Division (/)

```
    .section        __TEXT,__text,regular,pure_instructions
    .p2align        4, 0x90
    .globl  _main           ## -- Begin function main
_main:                      ## @main
    movl    4, %rax
    push    %rax
    movl    2, %rax
    pop    %rbx
    push    %rax
    mov     %rbx, %rax
    pop     %rbx
    cqo
    idivq   %rbx
    push    %rax
    pop    %rbx
    ret
    push    %rax
    pop    %rbx
    push    %rax
    pop    %rbx
```

# Test Plan

```
Arleth@DESKTOP-KRHDML5 MINGW64 ~/desktop/alphax_compiler/alphax_compiler (master)
$ mix test
.....................................................................

Finished in 0.5 seconds
71 tests, 0 failures

Randomized with seed 496000
```

# Test plan

To pass

```
int main() {
    return 6 / 3 / 2;
}
```

```
int main() {
    return 1 + 2;
}
```

```
int main() {
    return 2 + 3 * 4;
}
```

```
int main() {
    return 2 * 3;
}
```

# Test plan

To fail

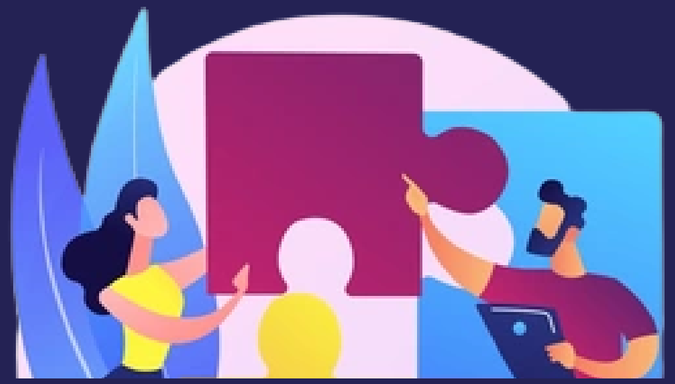
```
int main() {
    return /3;
}
```

```
int main() {
    return 1 + ;
}
```

```
int main() {
    return 2*2
}
```

# Use of github

For the version control we used a github repository



# CONCLUSIONS & LEARNED LESSONS