# Project description

* 1. **Topic**

A program to simulate a specific task of a MIPS compiler, which is opcode syntax checking (basic instructions only).

* 1. **Requirements**
* User enters a mips instruction.

Ex : beq $t1, $t2, label

* Check if input opcode exists.
* If input opcode exists, continue to check its operands’ validation behind.
* If input mips instruction is valid, print out the cycles in which it is executed.

# Algorithms

* 1. **Main**
     1. **Registers used**
* $a0: input string, opcode cycles
* $a1: classified group of opcodes
* $a2: opcode
* $a3: string of operands
* $v0: flag -> 0/1
  1. **If-else check**

*Note: Opcode and the number of cycles is found by using “check\_substring\_appearance” defined in Utils below.*

|  |  |
| --- | --- |
| **Step 1** | Read input string from user |
| **Step 2** | Split string into 2 parts: *Opcode* & *Operands* |
| **Step 3** | Use branch instructions to find a possible *Opcode* in every classified group |
| **Step 4** | If *Opcode* is not found, then branch to the next label to find it.  (brute force search)  If *Opcode* is not found in any groups, then print an invalid message to console |
| **Step 5** | If *Opcode* is found, start checking the *Operands* part by using the global variables and validation-checking functions defined in resource file |
| **Step 6** | If operands part is valid, print valid valid message console, followed by the its number of cycles, which could be found by using the global variables and functions defined in resource and utils file (*reg\_char\_first\_pos*) |

* 1. **Utils**
  2. **Purpose**

To define subprograms, which will be supportively used for the miniprojects.

*Note: In this miniproject, I use macros to define every subprogram for better understanding.*

* 1. **Important functions**

1. *check\_in\_reg\_range(%int\_register, %lower\_bound, %upper\_bound)*

* Purpose: to check if an integer stored in a register is in range [lower\_bound, upper\_bound]
* Return: $v0 = 1 if true, 0 if false

1. *check\_label(%string\_reg)*

* Purpose: to check if a label stored in a register is valid or not
* Return: $v0 = 1 if true, 0 if false

1. *check\_substring\_appearance(%string\_reg, %substring\_reg)*

* Purpose: to check if a substring belongs to a given string in which words are split by spaces

Ex: substring “c” in string “a b c d e”

* Return: $v0 = 1 if found, 0 if not found

1. *convert\_num\_hex(%register\_string)*

* Purpose: to convert a valid string of 16-bit hexadecimal to decimal
* Return: $v0 = num

1. *is\_hexa(%register\_string)*

* Purpose: to check if a string stored in a register is a valid 16-bit hexadecimal or not

Ex: 0xAF14, 0x0032E6, 0x12345678

* Return: $v1 = 1, $v0 = convert\_num\_hex(%register\_string) if valid

$v1 = 0, $v0 = 0 if invalid

1. *is\_num(%register\_string)*

* Purpose: to check if a string stored in a register is a valid integer or not
* Return: $v1 = 0, $v0 = 0 if invalid

$v1 = 1, $v0 = num(decimal) if valid

1. *reg\_char\_first\_pos(%char\_reg, %string\_reg)*

* Purpose: to find the first position of a character in a string, both stored in registers
* Return: $v0 = position if found, -1 if not found

1. *split\_by\_literal\_separator(%string\_reg, %char)*

* Purpose: to split a string stored in a register into 2 substrings, using separator character
* Return: if %char is found:

$a2 = the first half of the string split by this function

$a3 = the address of the second half of the string

(remove redundant spaces as many as possible)

else: $a2 = %string\_reg

$a3 = null

Ex: $t0 = “add $t1 , $t2, $t3 “

split\_by\_literal\_separator($t0, “ “) -> $a2 = “add”

$a3 = “$t1 , $t2, $t3”

**3. Resource**

* 1. **Global variables**

*Note: In this miniproject, I classify:*

*+ every opcode stored in a specific string named based on operands and number of cycles*

*+ every register based on type*

*Ex: cee : condition flag, even-float-register, even-float-register*

*es16\_i : even-float-register, signed\_16\_bit\_integer(integer\_register)*

* Abbreviation of operands list:

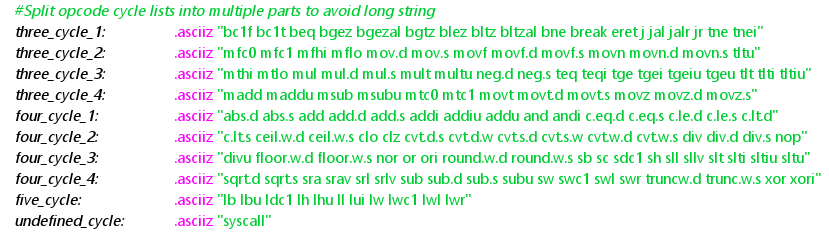
|  |  |
| --- | --- |
| **Abbreviation** | **Meaning** |
| c | Condition flag (0-7) |
| e | Even float register ($f0, $f2, …, $f30) |
| f | Float register ($f0, $f1, …, $f31) |
| i | Integer Register ($zero, $at, …, $ra, $0, $1, …, $31) |
| 8 | Coprocessor 0 ($8, $12, $13, $14) |
| l | Label |
| s16 | Signed 16-bit integer |
| u16 | Unsigned 16-bit integer |
| u5 | Unsigned 5-bit integer |
| none | No operands |

* Classified groups of opcodes based on operands:

A screenshot of a cell phone

Description automatically generated

* Classified groups of opcodes based on the number of cycles:



* Classified groups of registers based on type:

A picture containing table

Description automatically generated

* 1. **Operands validation-checking functions**
* **Function Format: *valid\_[group](%string\_of\_operands)***
* **Return: $v0 = 1 if valid, 0 if invalid**

*Note: For each classified groups of opcodes based on operands, I define a specific operand validation-checking function.*

As almost all operand validation-checking function share similarities, I only make a general description of algorithm based on the number of operands.

1. *3 operands (Ex: add $t0, $t1, $t2)*

|  |  |
| --- | --- |
| **Step 1** | Load the suitable group of opcodes depending on its name to $t0  Ex: Function “valid\_cee” needs Group “cee” |
| **Step 2** | * Split the string of operands into 2 parts with separator ‘,’ (comma), using function split\_by\_literal\_separator defined in Utils, then we have   $a2 = the first operand, $a3 = the rest of $t0  Ex: “$t0, $t1, $t2” -> split\_by\_literal\_separator($t0, ‘,’)   * $a2 = “$t0” * $a3 = “$t1, $t2” * Based on the operand’s type, check if that operand exists, by using:   + “check\_substring\_appearance” if operand is a register  + “is\_num”/”is\_hexa” && ”check\_int\_reg\_range” if operand is a number/offset  + “check\_label” if operand if a label   * If valid, go to the next step, else invalid |
| **Step 3** | Similar to Step 2, split the string of operands in $a3 to get the second operand by split\_by\_literal\_separator($a3, ‘,’), then check the validation of it  Ex: “$t1, $t2” -> split\_by\_literal\_separator($a3, ‘,’)   * $a2 = “$t1” * $a3 = “$t2”   If valid, go to the next step, else invalid |
| **Step 4** | Repeat Step 3 for the third operand  Ex: “$t2” -> split\_by\_literal\_separator($a3, ‘,’)   * $a2 = “$t2” * $a3 = null |
| **Step 5** | Check if the fourth operand exists by getting the string length of $a3.  If length > 0 return invalid, else valid |

1. *2 operands (Ex: bc1f 1, label)*

|  |  |
| --- | --- |
| **Step 1** | Load the suitable group of opcodes depending on its name to $t0  Ex: Function “valid\_ee” needs Group “ee” |
| **Step 2** | * Split the string of operands into 2 parts with separator ‘,’ (comma), using function split\_by\_literal\_separator defined in Utils, then we have   $a2 = the first operand, $a3 = the rest of $t0  Ex: “1, label” -> split\_by\_literal\_separator($t0, ‘,’)   * $a2 = “1” * $a3 = “label” * Based on the operand’s type, check if that operand exists, by using:   + “check\_substring\_appearance” if operand is a register  + “is\_num”/”is\_hexa” && ”check\_int\_reg\_range” if operand is a number/offset  + “check\_label” if operand if a label   * If valid, go to the next step , else invalid |
| **Step 3** | Similar to Step 2, split the string of operands in $a3 to get the second operand by split\_by\_literal\_separator($a3, ‘,’), then check the validation of it  Ex: “label” -> split\_by\_literal\_separator($t0, ‘,’)   * $a2 = “$t1” * $a3 = “$t2”   If valid, go to next step, else invalid |
| **Step 4** | Check if the third operand exists by getting the string length of $a3.  If length > 0 return invalid, else valid |

1. *1 operand (Ex: mfhi $t1)*

|  |  |
| --- | --- |
| **Step 1** | Load the suitable group of opcodes depending on its name to $t0  Ex: Function “valid\_i” needs Group “i” |
| **Step 2** | * Split the string of operands into 2 parts with separator ‘,’ (comma), using function split\_by\_literal\_separator defined in Utils, then we have   $a2 = the first operand, $a3 = the rest of $t0  Ex: “$t1” -> split\_by\_literal\_separator($t0, ‘,’)   * $a2 = “$t1” * $a3 = null * Based on the operand’s type, check if that operand exists, by using:   + “check\_substring\_appearance” if operand is a register  + “is\_num”/”is\_hexa” && ”check\_int\_reg\_range” if operand is a number/offset  + “check\_label” if operand if a label   * If valid, go to the next step , else invalid |
| **Step 3** | Check if the second operand exists by getting the string length of $a3.  If length > 0 return invalid, else valid |

1. *0 operand (Ex: syscall)*

|  |  |
| --- | --- |
| **Step 1** | Load the suitable group of opcodes depending on its name to $t0  Ex: Function “valid\_none” needs Group “none” |
| **Step 2** | Check if the first operand exists by getting the string length of $t0.  If length > 0 return invalid, else valid |

# Simulation Sample Result

A screenshot of a cell phone

Description automatically generated

A screenshot of a social media post

Description automatically generated