Computer Organization and Design

PROJECT REPORT

“Luhn’s Algorithm on assembly language RISC-V”

Submitted by

Mallikarjun Yeshlur
PES1UG22EC148

For the partial fulfillment of the course COD

(Computer Organization and Design)

UE22EC352A

Electronics and Communication Engineering

Submitted to

Prof. Pavitra Y J

Faculty of Engineering

(Program: B.Tech)

Introduction

What is Luhn's Algorithm?

The Luhn algorithm, also known as the **modulus 10 or mod 10 algorithm**, is a simple checksum formula used to validate a variety of identification numbers, such as credit card numbers, IMEI numbers etc.

Most credit cards and many government identification numbers use the algorithm as a simple method of distinguishing valid numbers from mistyped or otherwise incorrect numbers. It was designed to protect against accidental errors, not malicious attacks.

Luhn's makes it possible to check numbers such as credit card, gift card, IMEI (International Mobile Equipment Identity), etc.

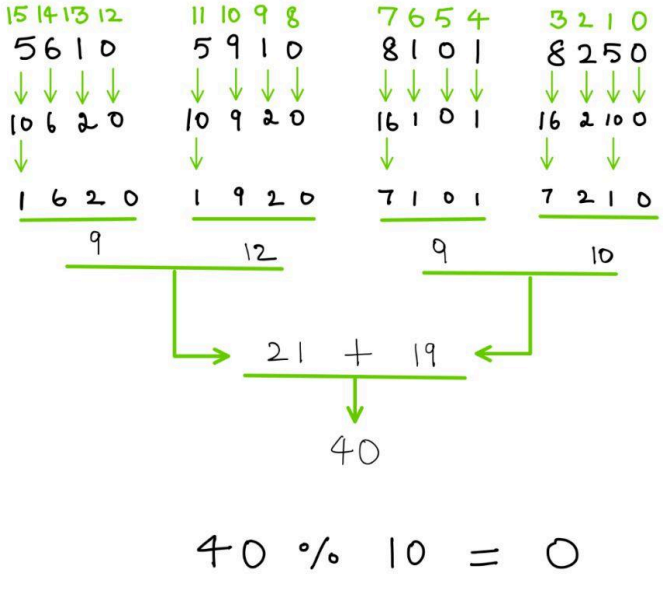
Luhn's Algorithm is known because MasterCard,American Express (AMEX), Visa and other credit card companies use it to validate credit card numbers.

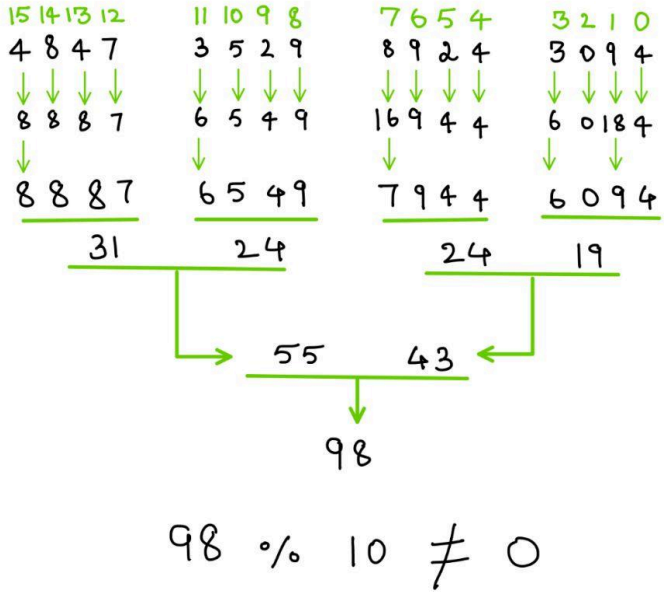
Working of Luhn's Algorithm

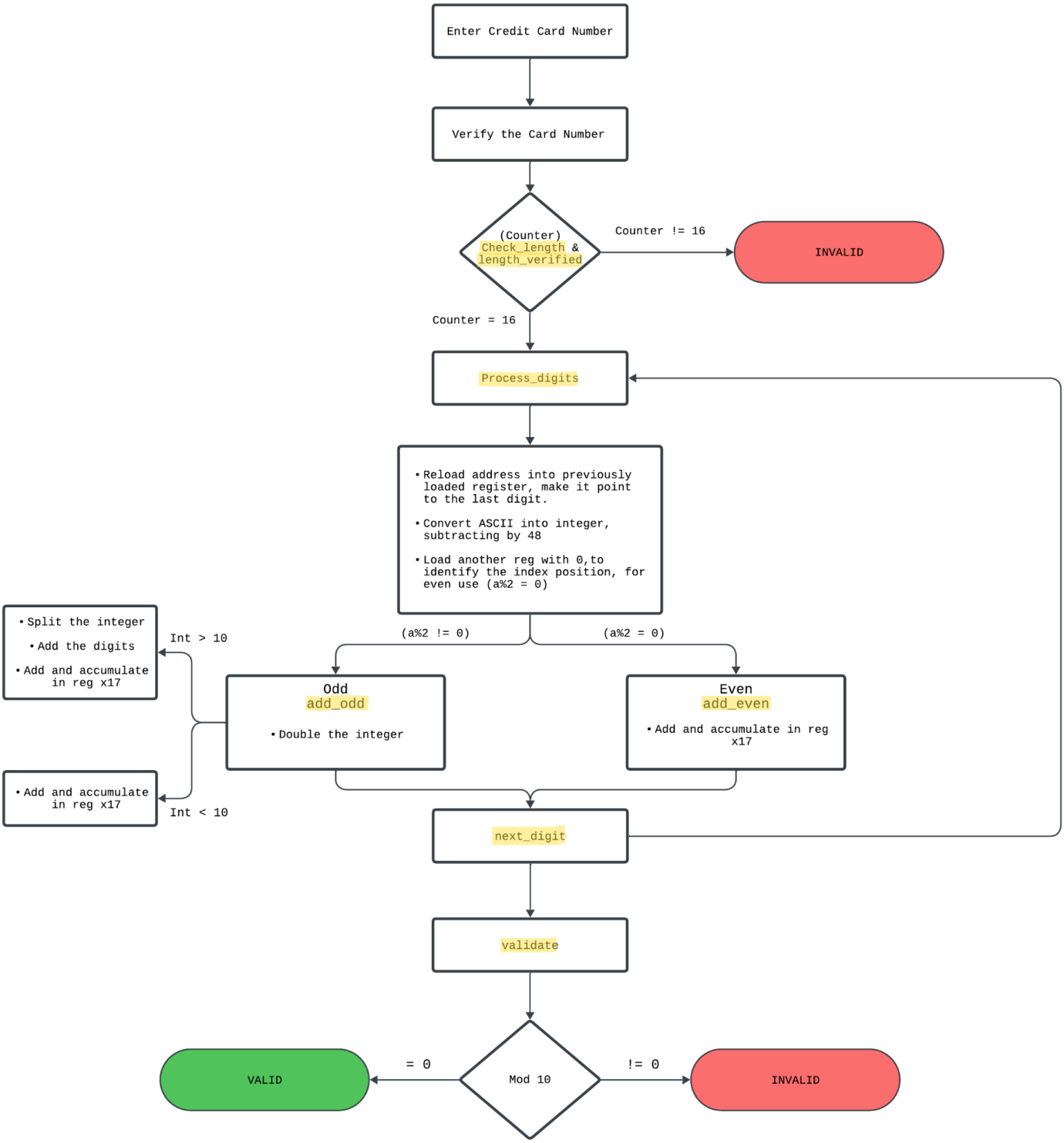
Luhn's algorithm for **credit card** has multiple steps:

1. Validate the number of digits, that must be 16.
2. Starting from the rightmost digit, double the value of every second digit.
3. To access every second digit, identify the odd places using the condition (a%2 !=0) .
4. If doubling a number results in a two digit number i.e greater than 9 (e.g., 9 × 2 = 18), then add the digits of the product (e.g., 18: 1 + 8= 9) to get a single digit number.
5. Digits in even place remains same.
6. Take the sum of all the digits.
7. If the Sum modulo 10 is equal to 0 then the number is valid according to the Luhn formula; else it is not valid.

Example:






Enter Credit Card Number
Verify the Card Number
(Counter)
Check_length &
length_verified
Counter != 16
INVALID
Counter = 16
Process_digits
• Reload address into previously loaded register, make it point to the last digit.
• Convert ASCII into integer, subtracting by 48
• Load another reg with 0,to identify the index position, for even use (a%2 = 0)
(a%2 != 0)
(a%2 = 0)
Odd
add_odd
• Double the integer
Even
add_even
• Add and accumulate in reg x17
Int > 10
• Split the integer
• Add the digits
• Add and accumulate in reg x17
Int < 10
• Add and accumulate in reg x17
next_digit
validate
Mod 10
= 0
VALID
!= 0
INVALID

Valid case

GPR

Name	Alias	Value
x0	zero	0
x1	ra	0
x2	sp	2147483632
x3	gp	268435456
x4	tp	0
x5	t0	0
x6	t1	0
x7	t2	0
x8	s0	0
x9	s1	0
x10	a0	0
x11	a1	0
x12	a2	0
x13	a3	0
x14	a4	0
x15	a5	0
x16	a6	0
x17	a7	0
x18	s2	0
x19	s3	0
x20	s4	0
x21	s5	0
x22	s6	0

Display type: Unsigned

GPR

Name	Alias	Value
x0	zero	0
x1	ra	268435456
x2	sp	16
x3	gp	0
x4	tp	16
x5	t0	1
x6	t1	0
x7	t2	10
x8	s0	1
x9	s1	0
x10	a0	268435473
x11	a1	0
x12	a2	0
x13	a3	0
x14	a4	0
x15	a5	0
x16	a6	0
x17	a7	10
x18	s2	2
x19	s3	0
x20	s4	0
x21	s5	0
x22	s6	0

Display type: Unsigned

Console

```
valid
Program exited with code: 0
```

Execution info

Cycles:	339
Instrs.retired:	339
CPI:	1
IPC:	1
Clock rate:	10.64 Hz

Invalid case

GPR

Name	Alias	Value
x0	zero	0
x1	ra	0
x2	sp	2147483632
x3	gp	268435456
x4	tp	0
x5	t0	0
x6	t1	0
x7	t2	0
x8	s0	0
x9	s1	0
x10	a0	0
x11	a1	0
x12	a2	0
x13	a3	0
x14	a4	0
x15	a5	0
x16	a6	0
x17	a7	0
x18	s2	0
x19	s3	0
x20	s4	0
x21	s5	0
x22	s6	0

Display type: Unsigned

GPR

Name	Alias	Value
x0	zero	0
x1	ra	268435456
x2	sp	16
x3	gp	0
x4	tp	16
x5	t0	8
x6	t1	8
x7	t2	10
x8	s0	1
x9	s1	6
x10	a0	268435479
x11	a1	0
x12	a2	0
x13	a3	0
x14	a4	0
x15	a5	0
x16	a6	0
x17	a7	10
x18	s2	2
x19	s3	0
x20	s4	0
x21	s5	0
x22	s6	0

Console

```
invalid
Program exited with code: 0
```

Execution info

Cycles:	331
Instrs.retired:	331
CPI:	1
IPC:	1
Clock rate:	10.64 Hz

CODE

```
.data
#number_str:.string "4847352989243094"   # Example 16-digit number string
number_str:.string "5610591081018250"   # Example 16-digit number string
valid_msg:.string "valid"
invalid_msg:.string "invalid"
.text
#----------Step 1: Checking the length-----------------------------------
la x1, number_str         # x1 points to the start of number_str
li x2, 0                # counter
check_length:
lb x3, 0(x1)              # load each byte from the string
beq x3, x0, VERIFY_LENGTH # If null char 0 found, exit loop
addi x2, x2, 1            # increment counter
addi x1, x1, 1            # move to the next char in number_str
j check_length
VERIFY_LENGTH:
li x4, 16                 # expected length of credit card number
bne x2, x4, invalid       # If length not 16, go to invalid
#----------STAGE 2: Convert characters to integers and check positions----------

la x1, number_str         # reload address of number_str back to start
add x1,x1,x2              # move pointer to last char
li x2, 0                # Initialize position counter to 0
li x17, 0                 # x17 will accumulate the final sum
process_digits:
addi x1,x1,-1             # move towards first char from last char
lb x5, 0(x1)              # Load character at x1 into x5
beq x5,x0, validate       # If null terminator is found, exit loop and go to VALIDATE
addi x5, x5, -48          # Convert ASCII to integer
li x31,2
# Check if position is odd or even
rem x6, x2, x31           # Remainder to check if position is odd  (a%2!=0)
beq x6, x0, add_even      # If even, add directly to sum
# Odd position - double the digit
slli x5, x5, 1            # Double the digit
li x7, 10
blt x5, x7, add_odd       # If doubled digit is less than 10, add directly
# If doubled digit >= 10, separate digits and sum
div x8, x5, x7            # x8 = doubled digit / 10 (tens place)
rem x9, x5, x7            # x9 = doubled digit % 10 (ones place)
add x5, x8, x9            # Sum of separated digits
```

```
add_odd:
add x17, x17, x5          # Add the processed odd-position digit to x17
j next_digit
add_even:
add x17, x17, x5          # Add the even-position digit directly to x17
next_digit:
addi x2, x2, 1            # Increment position counter
li x10, 16                # Loop should exit after 16 digits
bge x2, x10, validate     # If processed 16 digits, go to validation
j process_digits
validate:

#-------------- Check if the accumulated sum in x17 is divisible by 10------------------
#---------------------------- the main part of algorithm -------------------------------
li x7, 10
rem x6, x17, x7           # Check if x17 % 10 == 0
beq x6, x0, valid
#----------- STAGE 3: PRINT OUTPUT MESSAGE NON THE CONSOLE ----------------------
invalid:
# Print "invalid"
la a0, invalid_msg
li a7, 4                  # Print string syscall
ecall
j end_program
valid:
# Print "valid"
la a0, valid_msg
li a7, 4                  # Print string syscall
ecall
end_program:
li a7, 10                 # Exit program
ecall
```