TUNKU ABDUL RAHMAN UNIVERSITY COLLEGE

Faculty of Applied Sciences and Computing

Assignment

October 2017 Semester

Course code	:	AACS3064
Course Title	:	Computer Systems Architecture

Students' Name & ID No.	:	Name: ______________________________	ID. No.: ________________
		Name: ______________________________	ID. No.: ________________
		Name: ______________________________	ID. No.: ________________
Programme*	:	DCO2 / DST2	**[* Circle whichever is appropriate.]**
Tutorial Group	:	________________	
Tutor	:	Mr. Wong Hon Yoon / Ms. Choy Lai Fun / Mr. Loh Kian Nyak **[* Circle whichever is appropriate.]**	
Submission Date	:	**5/1/2018 - Week 12, Friday, before 12 noon**	

Members' Name	**Introduction**	**Coding & logic**	**I/O design**	**User guide**	**Total**
	(10marks)	**(25 marks)**		**(5 marks)**	**(40 marks)**
1)					
2)					
3)					

Comment:

Date of submission :

Date received :
(to-be filled by the tutor received)

Semester:________________ Course Code & Title: _______________________________

Declaration

I/We confirm that I/we have read and shall comply with all the terms and condition of TAR University College's plagiarism policy.

I/We declare that this assignment is free from all forms of plagiarism and for all intents and purposes is my/our own properly derived work.

I/We further confirm that the same work, where appropriate, has been verified by anti-plagiarism software ________________________ *(please insert).*

Signature(s):____
[By all members]

Name(s):____
[By all members]

Date: ___________________

Table of Contents

Introduction

As the trend of the decade inclines to the importance and development of cloud computing, IoT (Internet of Things), AI (Artificial Intelligence), etc. the focus of this report is however, revolves around a much lower-level of computing - Computer Systems Architecture, which largely consists of deep understanding of computer systems and their architectures, as well as a low-level programming language - the Assembly Language.

The importance of a deep understanding of the computer systems architecture is not to be underestimated, as it sculpts the foundations of the aforementioned trends of computing today.

In this report, a system would be proposed and later, developed, in assembly language, which shall include several arithmetic computation. The system would need to be of a given list of industries, namely:

- Agriculture industry
- Finance services industry
- Community system

Industry Selected

Hence, given the list of available industries, the one which we opt for is one of the branches of the **Finance Services Industry**, particularly, the **Loan Service Industry**.

While typically loan services are provided by banks, our company of choice is a standalone loan service agency.

Company Background

Wolfe’s Loan Corporation

Wolfe’s Loan Corporation is a newly registered company that aims to provide legal loan services of many sectors (including vehicle loans, business loans, housing loans, etc.) to individuals or businesses of any sort.

Being new in the industry, ***Wolfe’s Loan Corporation*** hired IT personnels to develop an 8086 microprocessor based system to measure and calculate several required functions.

Functions of the Program

The main function of the program being developed would be a loan calculator involving the formulas shown in the next section, which is used to calculate namely:

- Total Compounded Loan
  - Based on a desired amount of loan in USD
  - And the number of year the client wants to borrow the loan for
  - The annual interest rate is set based on the number of year (refer to the Assumptions' section)
- Monthly Loan Payment
  - The same input parameters as Total Compounded Loan

A receipt will be generated in a text file (receipt.txt), showing all the involved parameters and can be displayed from the 'Receipt' module, even after the program is quit.

Formulas Used

1) Monthly Loan Payment = $$\frac{\text{LoanAmount} \times \frac{R}{N}}{1 - (1 + \frac{R}{N})^{-\text{N} \text{x} \text{T}}}$$

R = Annual Interest Rate
N = Payments per year
T = Number of years

This formula is used to calculate the monthly amortization of a loan, the loan amount is the actual amount borrowed. Based on the formula, the shorter the payment period, the larger the amortization; the longer the payment period, the smaller the monthly payment due.

2) Total Compounded Loan = $$P(1 + \frac{R}{N})^{(\text{N} \times T)}$$

P = The Total Loan Amount
R = The Annual Interest Rate
N = The Number of Times the Interest is compounded each year(Usually 12)
T = The Number of Years the loan is borrowed for

This formula is used to calculate the Total Compounded Loan the customer pays after completing the consecutive payments for the loan.

Assumptions (Business Rules)

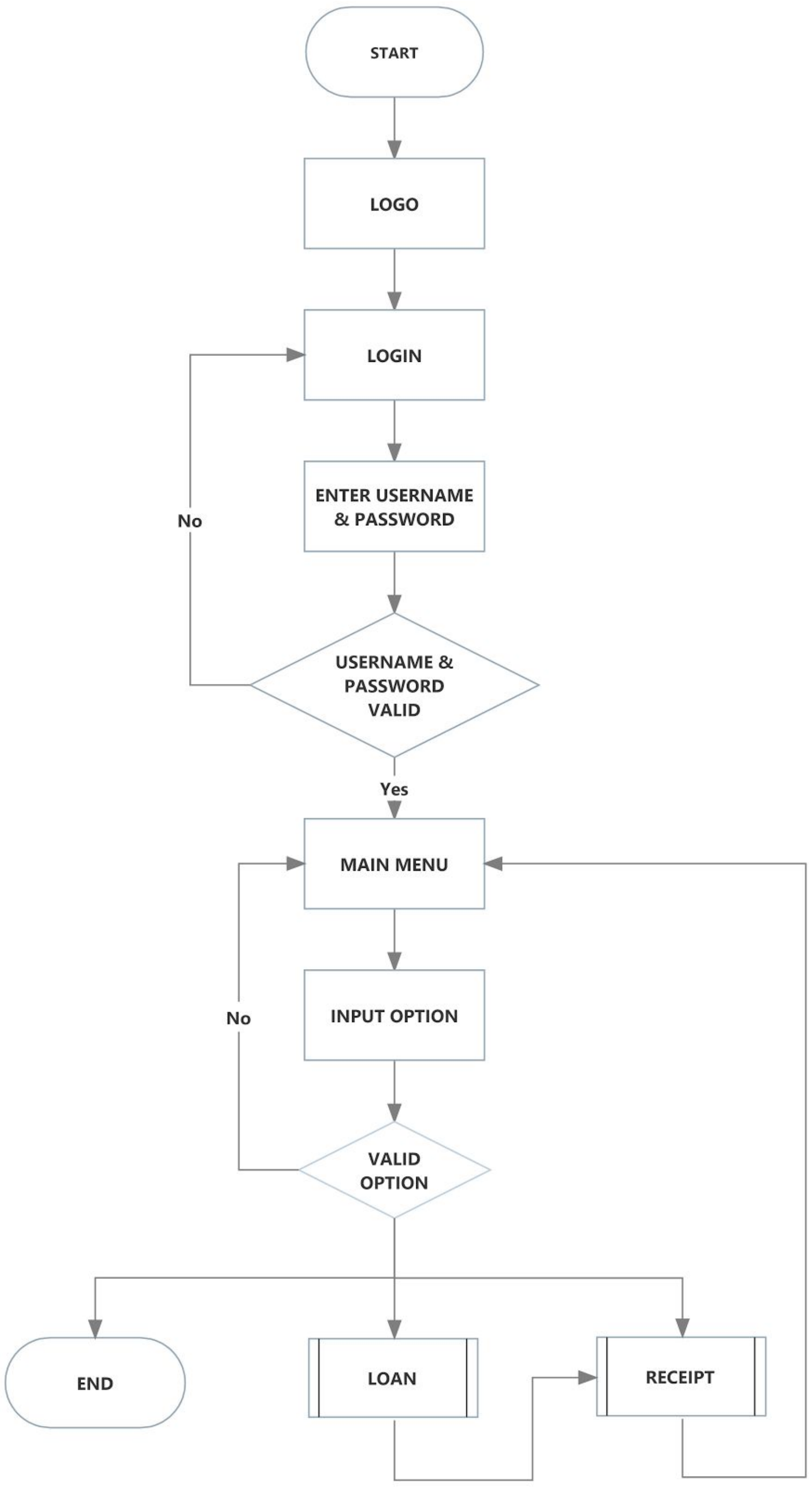
The business rules define the services offered and the policies of the company in providing the services. The functionalities of a program, like the company itself, must adhere to the business rules set, namely:

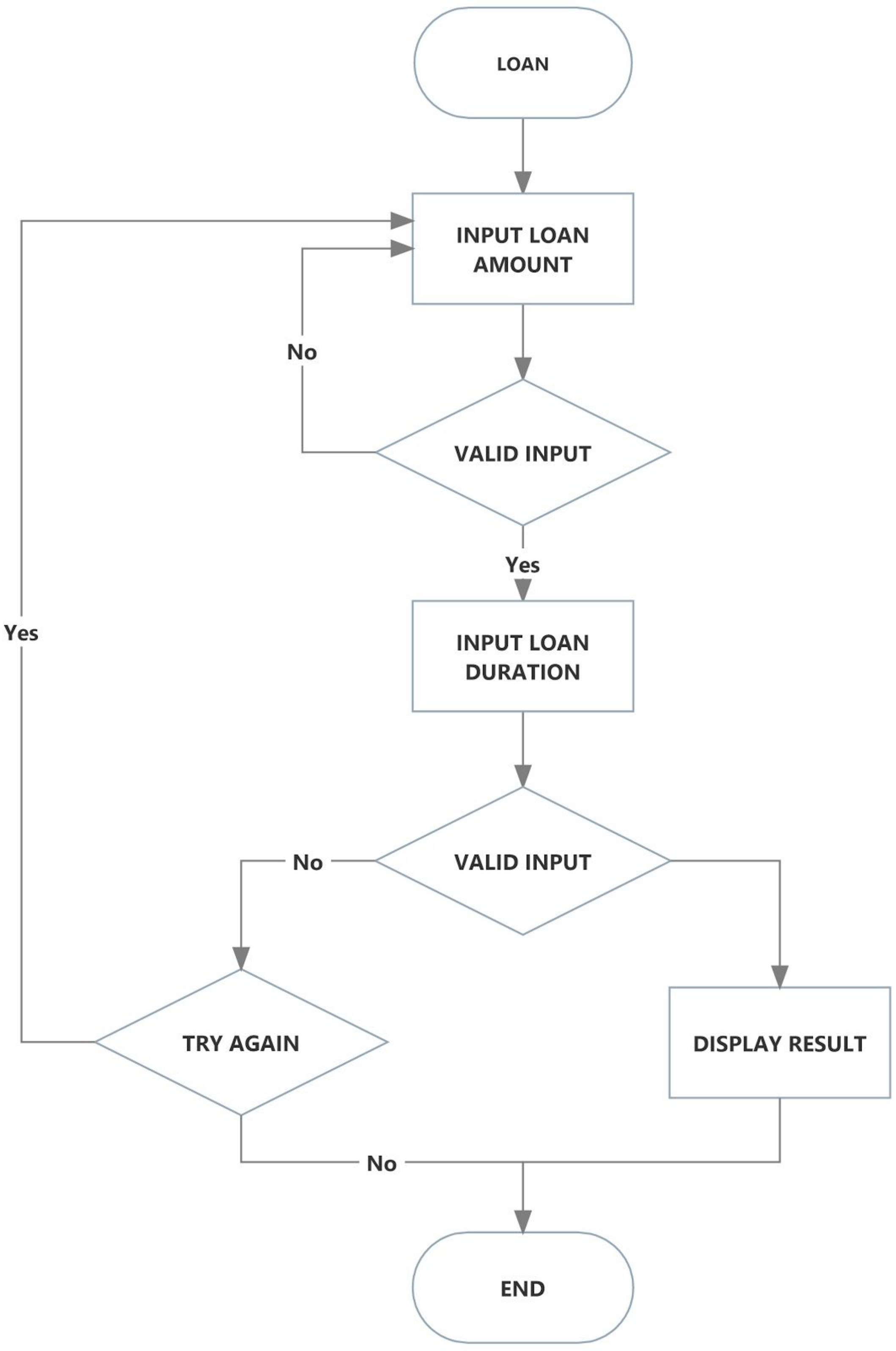
- ➔ All transactions must be in the currency of USD (US Dollar).
- ➔ Minimum loan amount in USD : 1000 (inclusive)
  Maximum loan amount in USD : 15000 (inclusive)
- ➔ Loan amount must be integers only (no decimals). But the results from the calculations will produce decimals rounded off to 2 decimal places.
- ➔ Loan durations are specified in years, not months. Clients have choices of 1 to 10 years inclusive.
- ➔ Loan Durations and their Interest Rates, respectively:

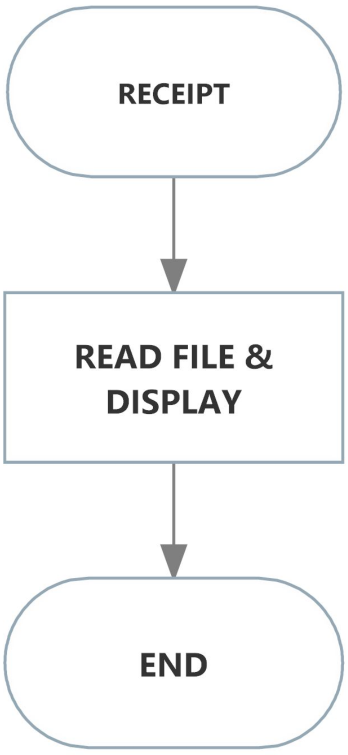
Loan Duration	1-3 years	4-6 years	7-10 years
Interest Rate	30%	18%	12%

- ➔ Due to limitations of the 8086 architecture, the accuracy of the results produced by the program would be ≈ ± 1 USD from the actual results.

Flow Chart


START
LOGO
LOGIN
ENTER USERNAME & PASSWORD
USERNAME & PASSWORD VALID
No
Yes
MAIN MENU
INPUT OPTION
VALID OPTION
No
END
LOAN
RECEIPT


LOAN
INPUT LOAN AMOUNT
No
VALID INPUT
Yes
INPUT LOAN DURATION
Yes
No
VALID INPUT
TRY AGAIN
DISPLAY RESULT
No
END


RECEIPT
READ FILE & DISPLAY
END

Coding & Logic

Low-level programming languages like the assembly language only include very basic functions for calculations, hence the calculations for the aforementioned formulas require heavy effort and an abundant amount of logical thinking to be completed.

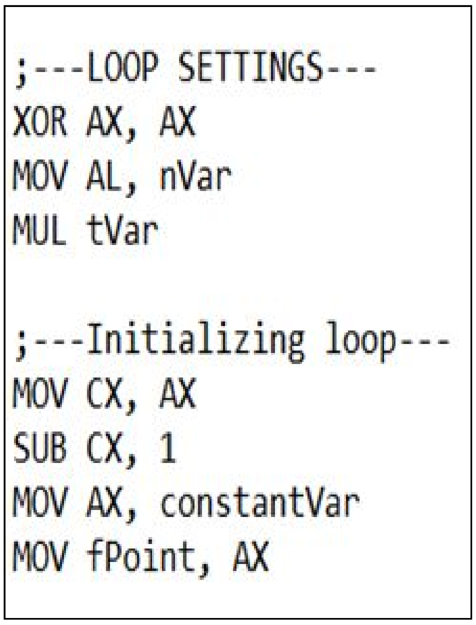
As such, below are some of the more notable features in the program, along with the detailed explanations of their logic:

The Power Function

ONG TUN YING

$$(1+\frac{R}{N})^{(N \times T)}$$

- Calculates power of a floating point number (3 decimal places) up to 4 decimal places and rounds it off to 3 decimal places.
- (1 + R/N) is a constant variable

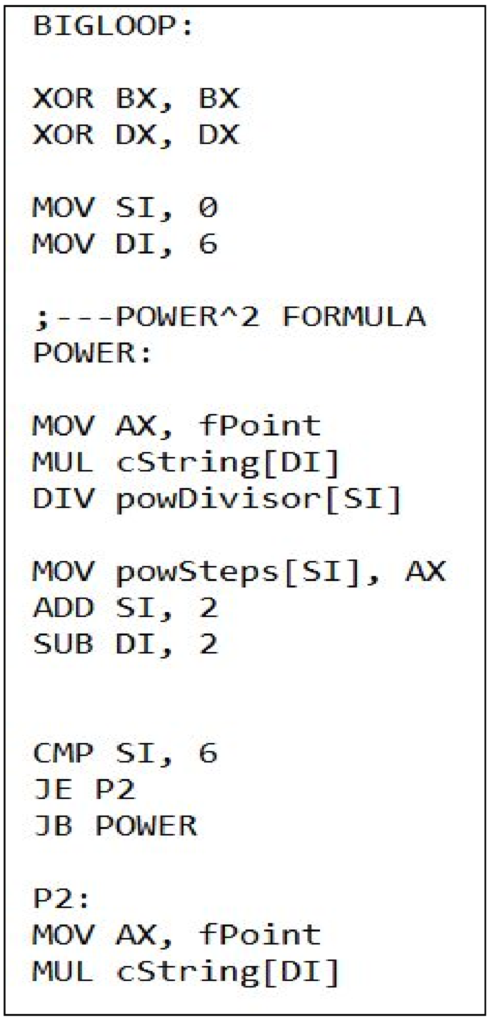


```
;---LOOP SETTINGS---
XOR AX, AX
MOV AL, nVar
MUL tVar

;---Initializing loop---
MOV CX, AX
SUB CX, 1
MOV AX, constantVar
MOV fPoint, AX
```

➢ Set (N x T) by multiplying N, the number of payments per year which is set to default as 12 with T, which is the number of years the loan is borrowed using the AL register.

➢ Initialize the loop with the result of (N x T) into the CX register and reducing it by 1 because power of 1 is negated.

➢ Set floating point variable from constant (R/N) selected based on the years borrowed by the user using AX register.



```
BIGLOOP:

XOR BX, BX
XOR DX, DX

MOV SI, 0
MOV DI, 6

;---POWER^2 FORMULA
POWER:

MOV AX, fPoint
MUL cString[DI]
DIV powDivisor[SI]

MOV powSteps[SI], AX
ADD SI, 2
SUB DI, 2


CMP SI, 6
JE P2
JB POWER

P2:
MOV AX, fPoint
MUL cString[DI]
```

- ➢ Set the SI register with 0 and DI register with 6

- ➢ The floating point variable contains 4 digits. (E.g. 1.010 = 1010). Multiply the floating point variable starting with the last digit of the constant variable array (e.g. 1.010) using the AX register

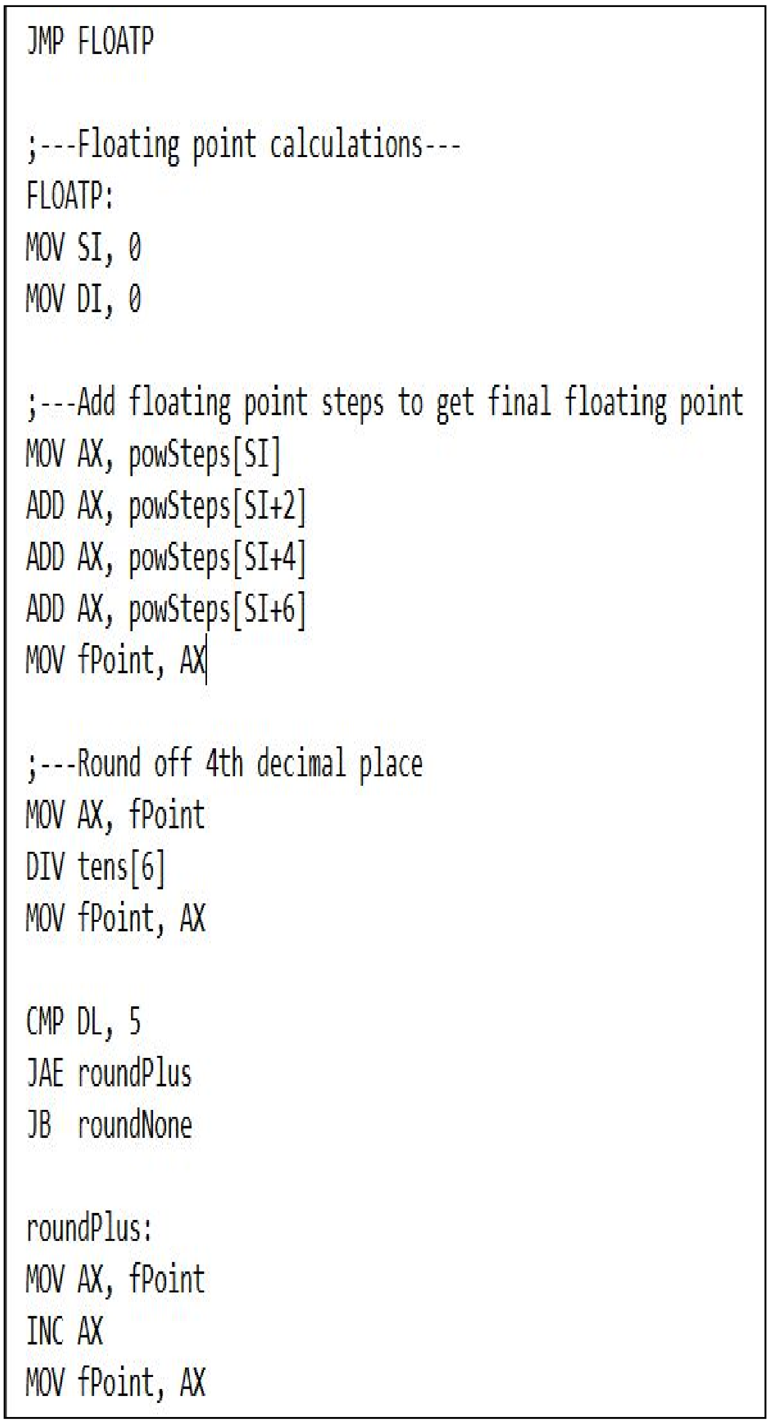
- ➢ Divide the AX register with powDivisor which contains multiples of 100 starting with 100. The next iteration will be 10, followed by 1.

- ➢ Store the result of division into the first position of an array variable called powSteps.

- ➢ Add SI register with 2 and subtract DI register with 2. (All variables with array above are WORD sized, 16-bit)

- ➢ Compares the SI register with 6, if equal jumps to the next label, P2, if below then jumps back to the power label. (Jumps back approx. 2 times)

- ➢ In label P2, the floating point variable is multiplied with the first digit in the constant variable string. Then jumps to label FLOATP



```
JMP FLOATP

;---Floating point calculations---
FLOATP:
MOV SI, 0
MOV DI, 0

;---Add floating point steps to get final floating point
MOV AX, powSteps[SI]
ADD AX, powSteps[SI+2]
ADD AX, powSteps[SI+4]
ADD AX, powSteps[SI+6]
MOV fPoint, AX

;---Round off 4th decimal place
MOV AX, fPoint
DIV tens[6]
MOV fPoint, AX

CMP DL, 5
JAE roundPlus
JB  roundNone

roundPlus:
MOV AX, fPoint
INC AX
MOV fPoint, AX
```

- ➢ Label FLOATP adds all the floating points inside the array powSteps into a 4 decimal floating point (5 digit variable now).

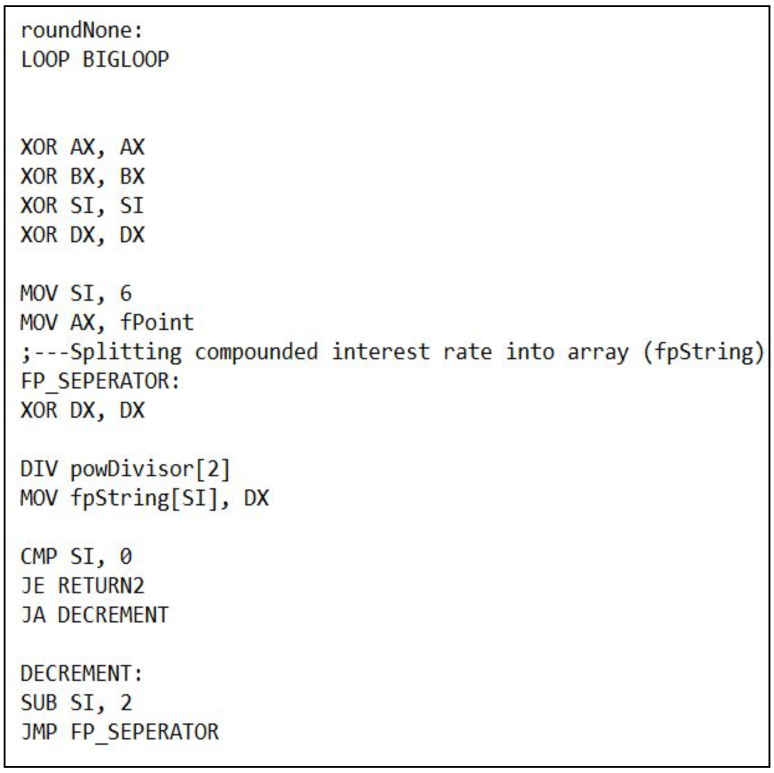
- ➢ After adding the new values of the floating point, it is moved back into the floating point variable as the new value. This is done using the AX register.

- ➢ After that, the new floating point value is rounded off to 3 decimal places.

- ➢ This is done by dividing the floating point value by 10 and moving the value back into the floating point variable (4 digit now). After that the DL register is compared with 5.

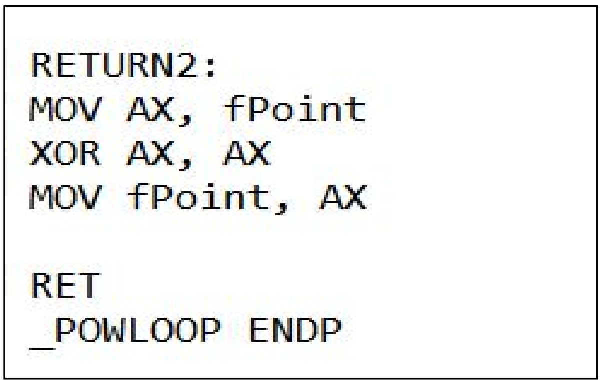
- ➢ If the DL register is equal or above the value 5 it will jump to the label roundPlus and increment the floating point variable by 1 using the AX register.

- ➢ However, if the DL register is below 5 it will remain the same value (4 digit, 3 decimal) and jump to label roundNone.



```
roundNone:
LOOP BIGLOOP


XOR AX, AX
XOR BX, BX
XOR SI, SI
XOR DX, DX

MOV SI, 6
MOV AX, fPoint
;---Splitting compounded interest rate into array (fpString)
FP_SEPERATOR:
XOR DX, DX

DIV powDivisor[2]
MOV fpString[SI], DX

CMP SI, 0
JE RETURN2
JA DECREMENT

DECREMENT:
SUB SI, 2
JMP FP_SEPERATOR
```

- ➢ After that, the label BIGLOOP will be looped and the CX register will decrement by 1.
- ➢ After the loop is done executing the SI register is set to 6
- ➢ In label FP_SEPERATOR the final value of the floating point variable is split into an array to be used by other functions as a variable.
- ➢ The floating point variable is divided by 10 and stored into the final floating point array from the last digit to the first digit by comparing the SI register with 0.



```
RETURN2:
MOV AX, fPoint
XOR AX, AX
MOV fPoint, AX

RET
_POWLOOP ENDP
```

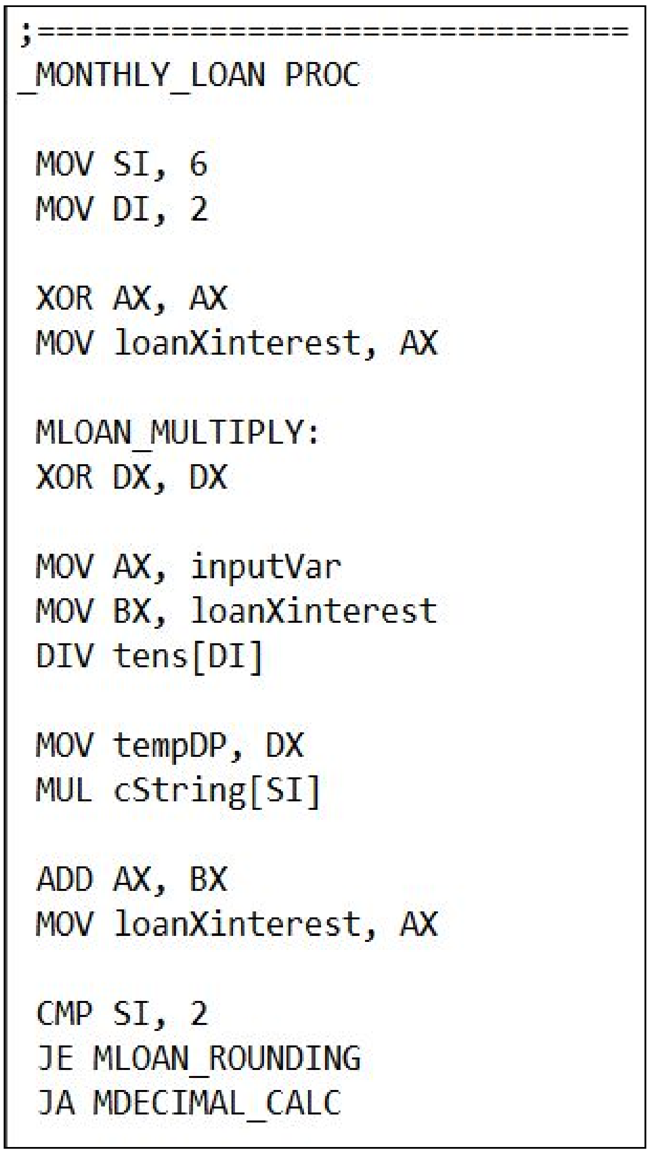
- ➢ The floating point variable is cleared using the AX register, then the function returns.

Monthly Loan Payment

ONG JON SHEN

$$\frac{\text{LoanAmount} \times \frac{R}{N}}{1 - (1 + \frac{R}{N})^{-\text{N}\times\text{T}}}$$

- Calls POWLOOP function to get (1 + R/N)^(N*T) floating point
- Performs division of 2 floating point numbers, 2 decimal places. (Rounded off 3 decimal place to 2 decimal)
- Split into 3 parts of calculation, numerator and denominator, then division of numerator and denominator



```
;==============================
_MONTHLY_LOAN PROC

 MOV SI, 6
 MOV DI, 2

 XOR AX, AX
 MOV loanXinterest, AX

 MLOAN_MULTIPLY:
 XOR DX, DX

 MOV AX, inputVar
 MOV BX, loanXinterest
 DIV tens[DI]

 MOV tempDP, DX
 MUL cString[SI]

 ADD AX, BX
 MOV loanXinterest, AX

 CMP SI, 2
 JE MLOAN_ROUNDING
 JA MDECIMAL_CALC
```

➢ MLOAN_MULTIPLY calculates the numerator multiplication of the formula.

➢ The variable loanXinterest is used to store the total value of the numerator after calculation is done.

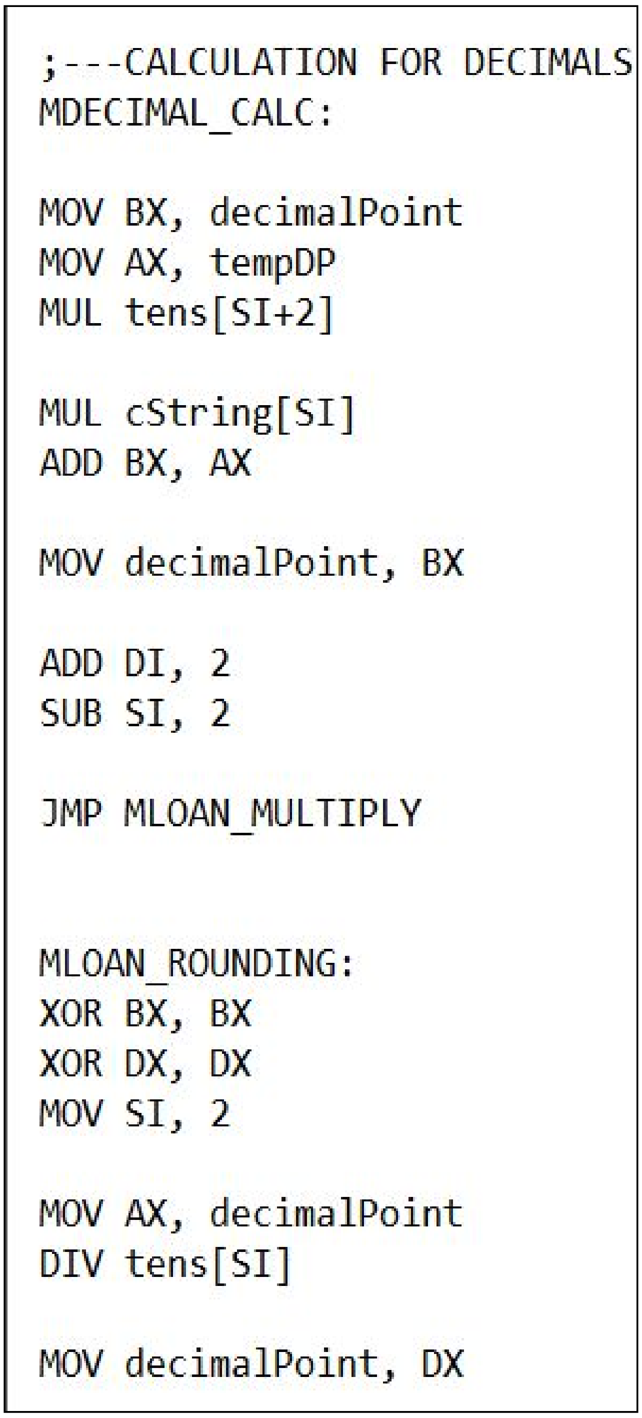
➢ loanXinterest is cleared, then the inputVar which contains the loan amount is divided by a 16-bit array variable which contains (10000, 1000, 100, 10, 1). It is first divided by the second place of the array starting at position 2 then lastly at position 8 using the DI register as the offset address.

➢ The remainder is stored into a temporary Decimal Point holder because the multiplication instruction clears the DX register.

➢ The AX register is multiplied with the last string constant variable starting from the last decimal place to the second using the SI register as the offset address.

➢ The result of the AX would be added with the current value of loanXinterest and hence, stored into it.

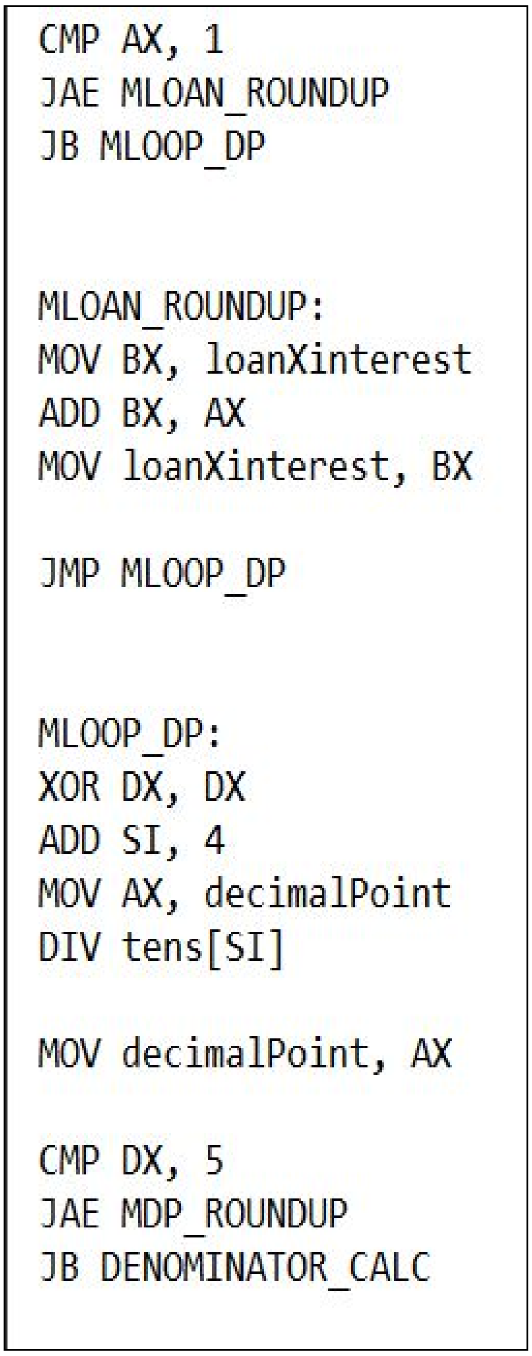
➢ The SI register is compared with 2.



```
;---CALCULATION FOR DECIMALS
MDECIMAL_CALC:

MOV BX, decimalPoint
MOV AX, tempDP
MUL tens[SI+2]

MUL cString[SI]
ADD BX, AX

MOV decimalPoint, BX

ADD DI, 2
SUB SI, 2

JMP MLOAN_MULTIPLY


MLOAN_ROUNDING:
XOR BX, BX
XOR DX, DX
MOV SI, 2

MOV AX, decimalPoint
DIV tens[SI]

MOV decimalPoint, DX
```

- ➢ If the SI register was above 2, it will jump to MDECIMAL_CALC. This calculates the multiplication of the remainder from the division.
- ➢ The variable decimalPoint is moved in to the BX register, while the tempDP is multiplied with tens array variable to get up to 3 digits, or 3 decimal places using AX register.
- ➢ After that, it is multipled with the constant string variable starting from the last place to the second place using the SI register as the offset address.
- ➢ The current value of the decimalPoint variable is added with the result and store into the decimalPoint variable using BX register
- ➢ DI register is added with 2 while the SI register is subtract with 2.
- ➢ The pointer returns to the MLOAN_MULTIPLY label.

- ➢ If the SI register is equal to 2 then it jumps to the MLOAN_ROUNDING label which rounds off any extra decimals and handles the carry.
- ➢ If there is any carry after adding up all the decimals together, the 1000th digit of the deicmalPoint var is divided and the remainder of the 3 digits is stored back into the decimalPoint var using the DX register.



```
CMP AX, 1
JAE MLOAN_ROUNDUP
JB MLOOP_DP


MLOAN_ROUNDUP:
MOV BX, loanXinterest
ADD BX, AX
MOV loanXinterest, BX

JMP MLOOP_DP


MLOOP_DP:
XOR DX, DX
ADD SI, 4
MOV AX, decimalPoint
DIV tens[SI]

MOV decimalPoint, AX

CMP DX, 5
JAE MDP_ROUNDUP
JB DENOMINATOR_CALC
```

- ➢ The carry digit is compared with 1 and if equal or above will jump to MLOAN_ROUNDUP which adds the carry back into the whole number loanXinterest using BX and AX register. It then continues to the MLOOP_DP label.

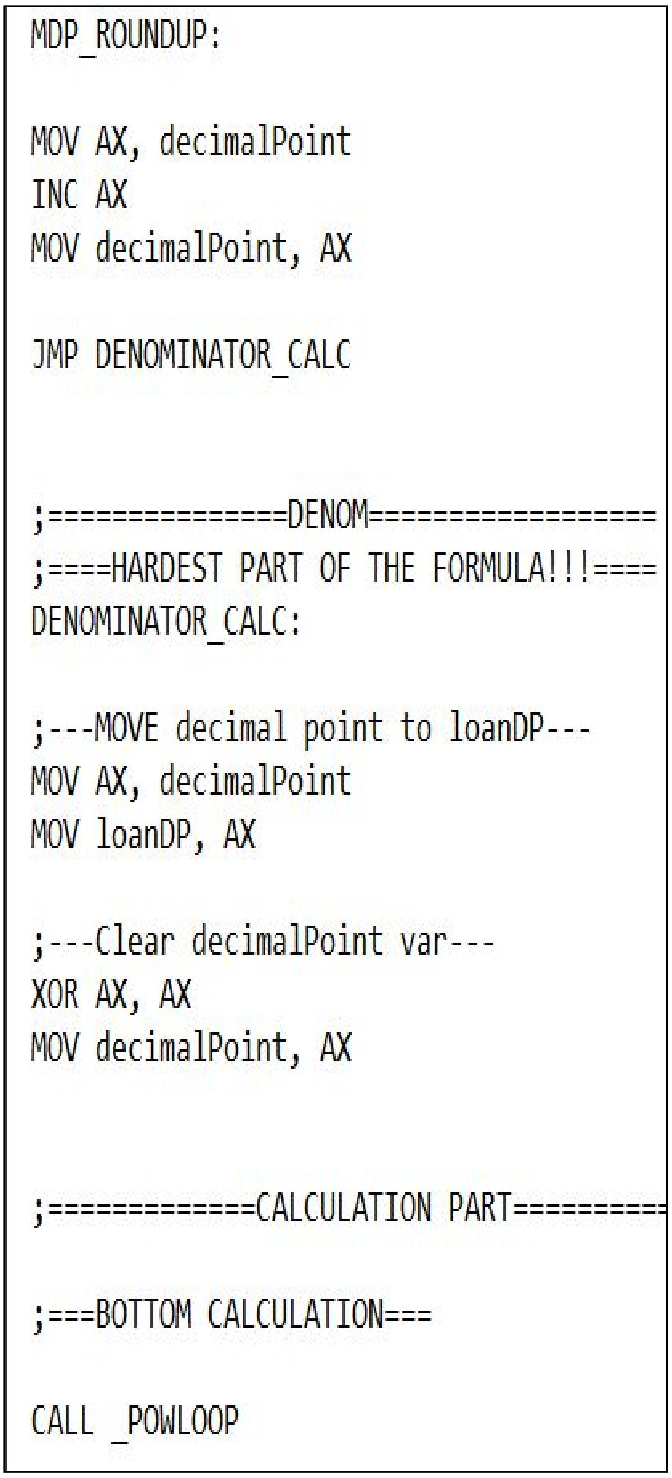
- ➢ Else if the carry digit is zero it will continue to MLOOP_DP label which rounds off the third digit of the decimal point into 2 decimal points.

- ➢ The MLOOP_DP label divides the last digit of the decimalPoint with 10.

- ➢ The result will be stored back into the decimalPoint var through the AX register. (2 digits now)

- ➢ The remainder rounded by comparing with 5, if it is above or equal to the value 5 using the DX register, it will proceed to MDP_ROUDUP which adds one to the 2 digit decimalPoint

- ➢ Else, if the remainder is below 5, it will move on to the DENOMINATOR_CALC which calculates the denominator portion of the formula. The value of the decimalPoint is kept at 2 digits.



```
MDP_ROUNDUP:

MOV AX, decimalPoint
INC AX
MOV decimalPoint, AX

JMP DENOMINATOR_CALC



;==============DENOM=================
;====HARDEST PART OF THE FORMULA!!!====
DENOMINATOR_CALC:

;---MOVE decimal point to loanDP---
MOV AX, decimalPoint
MOV loanDP, AX

;---Clear decimalPoint var---
XOR AX, AX
MOV decimalPoint, AX



;============CALCULATION PART=========

;===BOTTOM CALCULATION===

CALL _POWLOOP
```

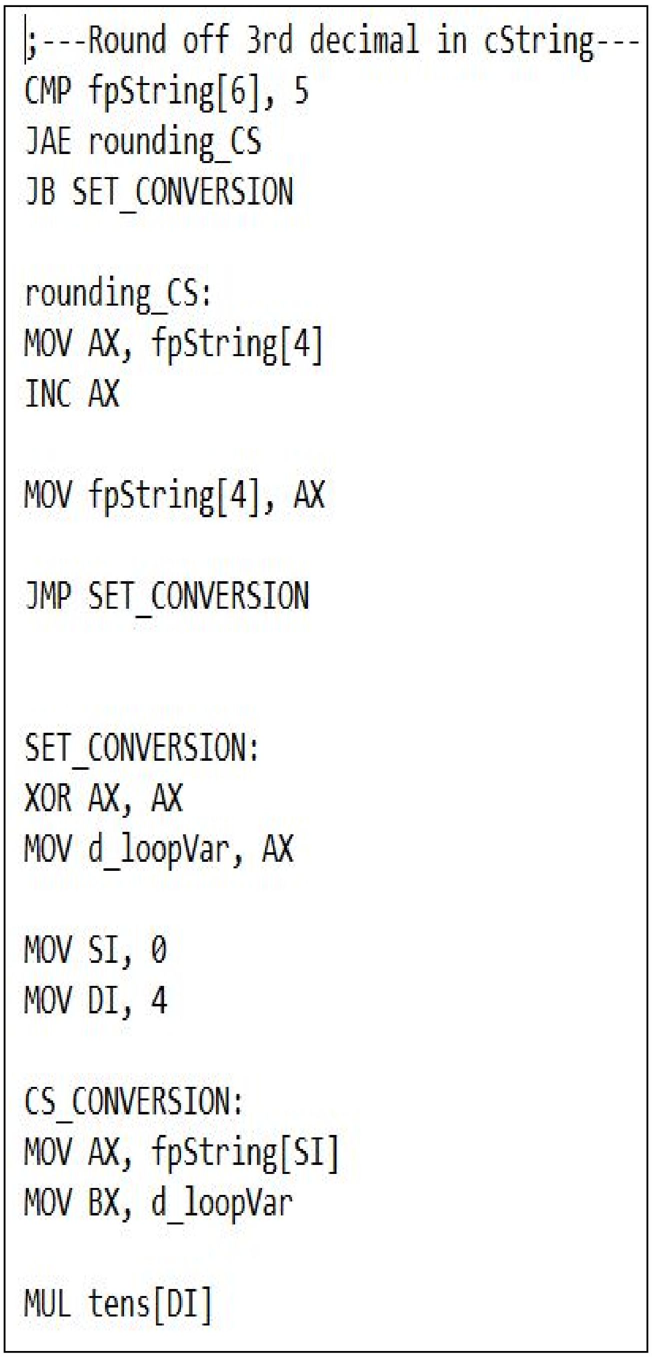
- ➢ In MDP_ROUNDUP, the decimalPoint is incremented by 1 using the AX register. Then the new value is stored back into decimalPoint.

- ➢ After that, it jumps to DENOMINATOR_CALC for the next part of the formula.

- ➢ In DENOMINATOR_CALC, the decimalPoint variable is moved into the loanDP (AX register) variable which will be the decimal point value of the numerator.

- ➢ The value of decimalPoint is cleared using the AX register for future use.

- ➢ In the CALCULATION PART, the function POWLOOP is called to get the value of (1 + R/N) ^ (T x N). This value is stored inside a string array called fpString.



```
;---Round off 3rd decimal in cString---
CMP fpString[6], 5
JAE rounding_CS
JB SET_CONVERSION

rounding_CS:
MOV AX, fpString[4]
INC AX

MOV fpString[4], AX

JMP SET_CONVERSION


SET_CONVERSION:
XOR AX, AX
MOV d_loopVar, AX

MOV SI, 0
MOV DI, 4

CS_CONVERSION:
MOV AX, fpString[SI]
MOV BX, d_loopVar

MUL tens[DI]
```

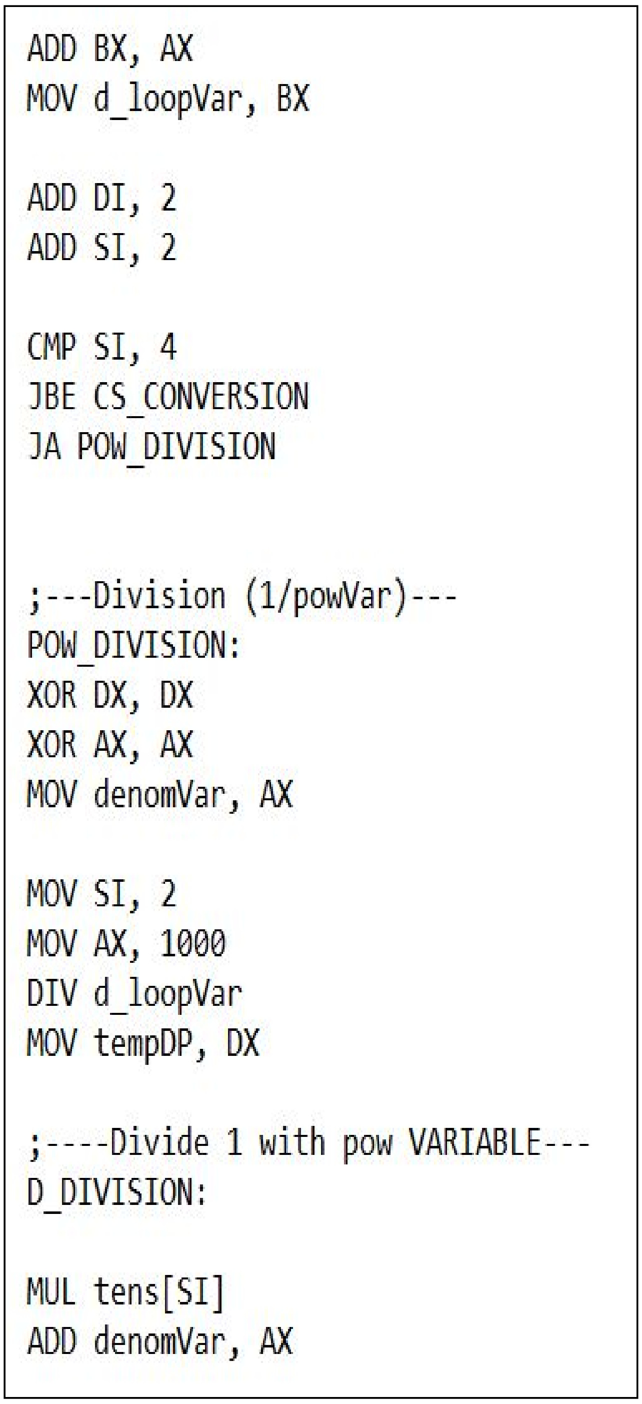
➢ The 3rd decimal of the constant array string is rounded off for division purposes by comparing the last digit with 5, if the value is above or equal it will increment the second last digit of the string by 1 (AX register).

➢ Else if the value is below it jump to SET_CONVERSION, which will move the new rounded off value of fpString into a variable called d_loopVar.

➢ In SET_CONVERSION, each value of the fpString array starting from the second last place will be multiplied with its respective decimal value. (E.g. 1,2,3,4 = 1000 + 200 + 30 + 4).

➢ This is done by using the SI and DI register as the offset address of fpString and tens.

➢ After that, CS_CONVERSION will add the new value of the fpString digit into d_loopVar. Using AX and BX registers.



```
ADD BX, AX
MOV d_loopVar, BX

ADD DI, 2
ADD SI, 2

CMP SI, 4
JBE CS_CONVERSION
JA POW_DIVISION


;---Division (1/powVar)---
POW_DIVISION:
XOR DX, DX
XOR AX, AX
MOV denomVar, AX

MOV SI, 2
MOV AX, 1000
DIV d_loopVar
MOV tempDP, DX

;----Divide 1 with pow VARIABLE---
D_DIVISION:

MUL tens[SI]
ADD denomVar, AX
```

➢ The DI and SI registers are added by 2, then the SI register is compared with 4. If the value is above it will jump to POW_DIVISION.

➢ Else it will jump back to CS_CONVERSION at the top.

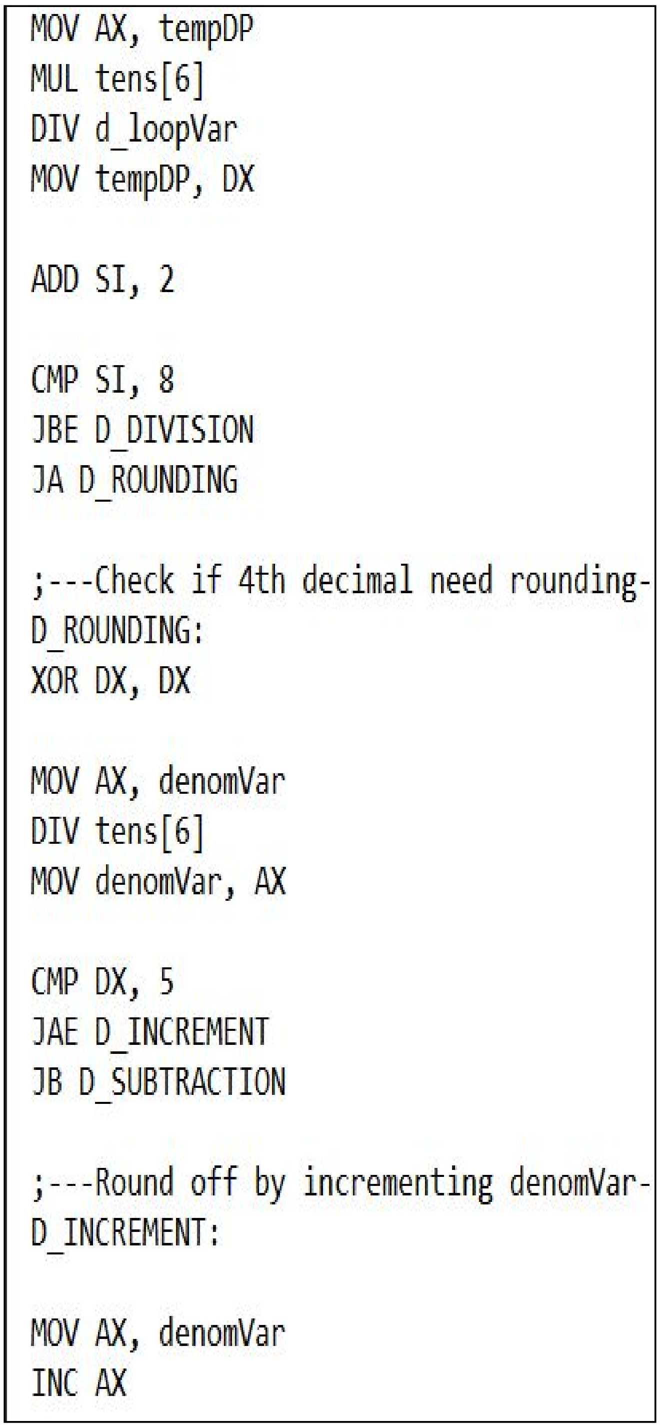
➢ POW_DIVISION handles the calculation of (1 + R/ N) ^ (- T x N) which becomes (1 / d_loopVar)

➢ A new variable is moved into the AX register to store the value of the denominator in whole number.

➢ By dividing 1000 with d_loopVar we willl get the first digit.

➢ The remainder will be moved into the tempDP using the DX register.

➢ After that, the D_DIVISION label will multiply the result with its new respective decimal values of (100, 10, 1) using the variable tens with the offset of the SI register.



```
MOV AX, tempDP
MUL tens[6]
DIV d_loopVar
MOV tempDP, DX

ADD SI, 2

CMP SI, 8
JBE D_DIVISION
JA D_ROUNDING

;---Check if 4th decimal need rounding-
D_ROUNDING:
XOR DX, DX

MOV AX, denomVar
DIV tens[6]
MOV denomVar, AX

CMP DX, 5
JAE D_INCREMENT
JB D_SUBTRACTION

;---Round off by incrementing denomVar-
D_INCREMENT:

MOV AX, denomVar
INC AX
```

➢ Then, the result will be added into the denomVar.

➢ Afterwards, the tempDP is multiplied with 10 (Make it 4 digit long) and divided with d_loopVar. (Using AX register)

➢ The remainder is stored back into the tempDP

➢ The SI registers are added by 2

➢ Set the condition to check if the division is taken place till the offset register SI is above 8. Else it will jump back to D_DIVISION to repeat the whole division process.

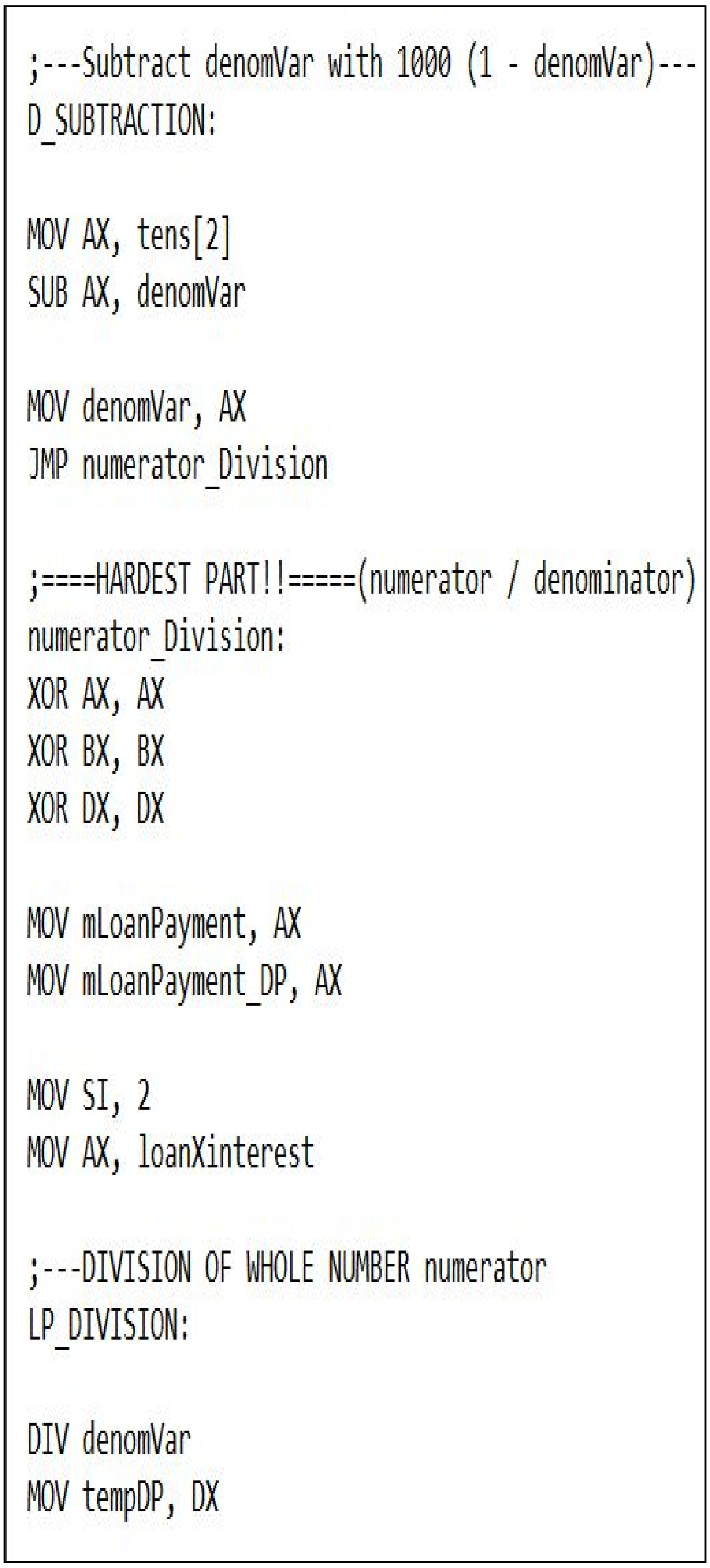
➢ After that, it will jump to D_ROUNDING. This label will check if the 4th decimal requires rounding up.

➢ The last digit of denomVar is separated by dividing it by 10.

➢ The new value of denomVar is stored back into it using AX register. (3 decimal now)

➢ The last digit is compared with 5, if above or equal will jump to D_INCREMENT else will jump to D_SUBTRACTION

➢ In D_INCREMENT, the denomVar is incremented by 1 and moved back into the denomVar. Then jumps to numerator_division.



```
;---Subtract denomVar with 1000 (1 - denomVar)---
D_SUBTRACTION:

MOV AX, tens[2]
SUB AX, denomVar

MOV denomVar, AX
JMP numerator_Division

;====HARDEST PART!!=====(numerator / denominator)
numerator_Division:
XOR AX, AX
XOR BX, BX
XOR DX, DX

MOV mLoanPayment, AX
MOV mLoanPayment_DP, AX

MOV SI, 2
MOV AX, loanXinterest

;---DIVISION OF WHOLE NUMBER numerator
LP_DIVISION:

DIV denomVar
MOV tempDP, DX
```

- ➢ In D_SUBTRACTION, the formula (1 - denomVar) will take place.
- ➢ 1000 is subtracted from denomVar (Contains 3 digits) to get a final value of 3 decimal places. (Using AX register)

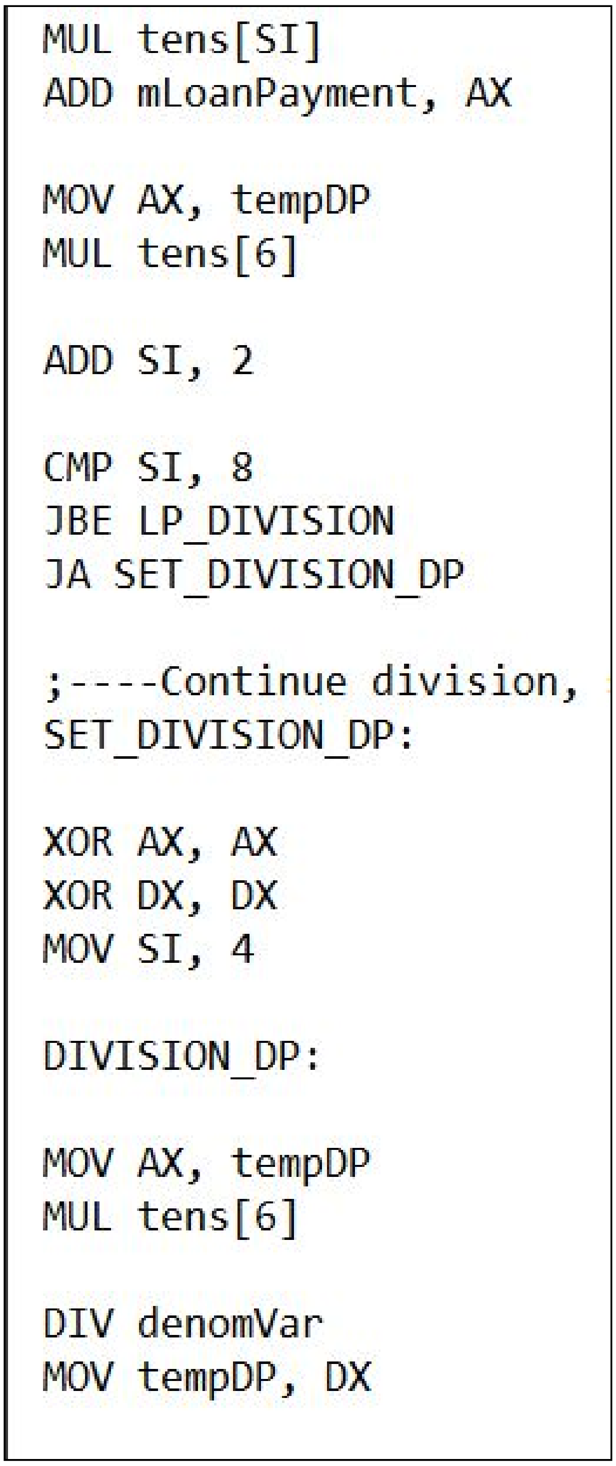
- ➢ Then the result is moved back into denomVar and it jumps to numerator_Division.

- ➢ numerator_Division handles the calculation of the (numerator / denominator).

- ➢ New variables are introduced, mLoanPayment (Stores monthly loan payment whole number) and mLoanPayment_DP (Stores monthly loan payment decimal points)

- ➢ Move loanXinterest (numerator result whole number)

- ➢ LP_DIVISION handles the division of the whole number numerator (loanXinterest)
- ➢ Divide loanXinterest with denomVar (AX register)
- ➢ Move the remainder into tempDP(DX register)



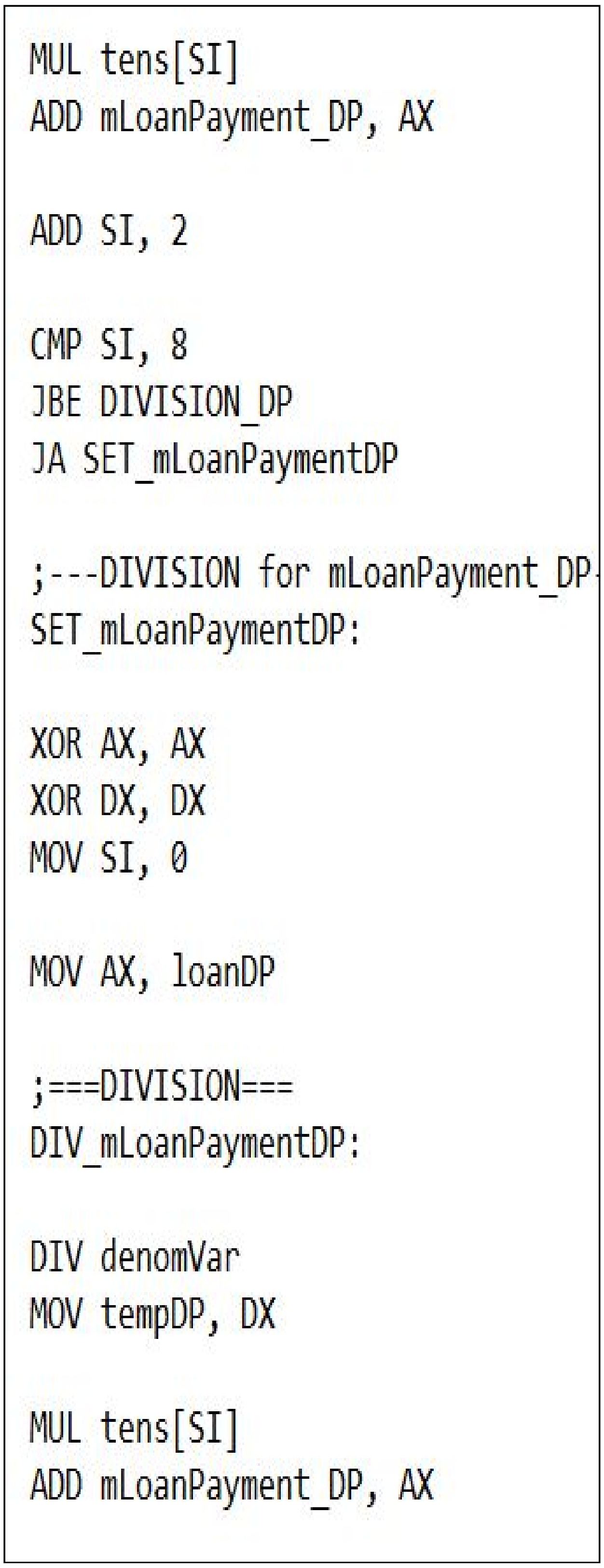
```
MUL tens[SI]
ADD mLoanPayment, AX

MOV AX, tempDP
MUL tens[6]

ADD SI, 2

CMP SI, 8
JBE LP_DIVISION
JA SET_DIVISION_DP

;----Continue division,
SET_DIVISION_DP:

XOR AX, AX
XOR DX, DX
MOV SI, 4

DIVISION_DP:

MOV AX, tempDP
MUL tens[6]

DIV denomVar
MOV tempDP, DX
```

- ➢ Multiply with the respective decimal value (1000, 100, 10, 1)
- ➢ Add the result into mLoanPayment.
- ➢ Multiply back the remainder(tempDP) with 10 in the AX register
- ➢ Compare the SI register with 8 to get the complete division of the whole number.
- ➢ If SI is above 8 then jump to SET_DIVISION_DP

- ➢ SET_DIVISION_DP handles the division of the decimals from the loanXinterest whole number.

- ➢ Multiply the reaminder(tempDP) with 10 and divide it with denomVar. (AX register)

- ➢ Store ramainder into tempDP (DX register)



```
MUL tens[SI]
ADD mLoanPayment_DP, AX

ADD SI, 2

CMP SI, 8
JBE DIVISION_DP
JA SET_mLoanPaymentDP

;---DIVISION for mLoanPayment_DP
SET_mLoanPaymentDP:

XOR AX, AX
XOR DX, DX
MOV SI, 0

MOV AX, loanDP

;===DIVISION===
DIV_mLoanPaymentDP:

DIV denomVar
MOV tempDP, DX

MUL tens[SI]
ADD mLoanPayment_DP, AX
```

➢ Multiply the result with the respective decimal places (100, 10, 10) AX register

➢ Check condition if SI register is below or equal to 8, if true then jump to DIVISION_DP above.

➢ If false then jump to SET_mLoanPaymentDP

➢ SET_mLoanPaymentDP calculates the division for the decimal place of loanDP (numerator 2 decimal point)

➢ Move the loanDP inside the AX register for arithmetic usage.

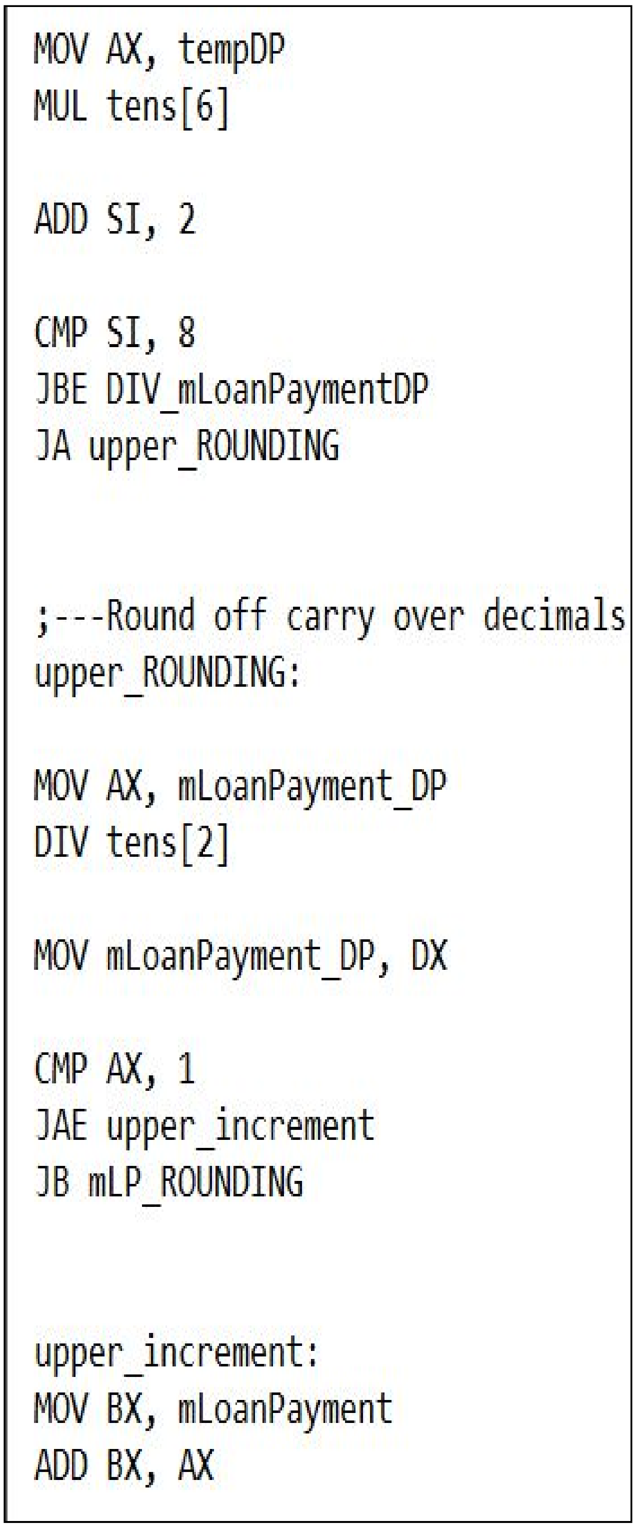
➢ DIV_mLoanPaymentDP, handles division process.

➢ Divide the AX register with with denomVar.

➢ Store remainder in tempDP

➢ Multiply the AX register with respective decimals (10000, 1000, 100, 10, 1) If the result of division is zero then then multiplication will also be zero,

➢ Add the result of AX into DIV_mLoanPaymentDP



```
MOV AX, tempDP
MUL tens[6]

ADD SI, 2

CMP SI, 8
JBE DIV_mLoanPaymentDP
JA upper_ROUNDING


;---Round off carry over decimals
upper_ROUNDING:

MOV AX, mLoanPayment_DP
DIV tens[2]

MOV mLoanPayment_DP, DX

CMP AX, 1
JAE upper_increment
JB mLP_ROUNDING


upper_increment:
MOV BX, mLoanPayment
ADD BX, AX
```

➢ Multiply the remainder(tempDP) with 10

➢ Check condition if SI register is below or equal to 8, if true then jump to DIV_mLoanPaymentDP above.

➢ If above 8 then jump to upper_ROUDNING

➢ upper_ROUNDING rounds off the carry over decimals into the whole number.

➢ Divide the 1000th digit of the decimal point mLoanPayment_DP (e.g. 1234, 1 is the carry over)

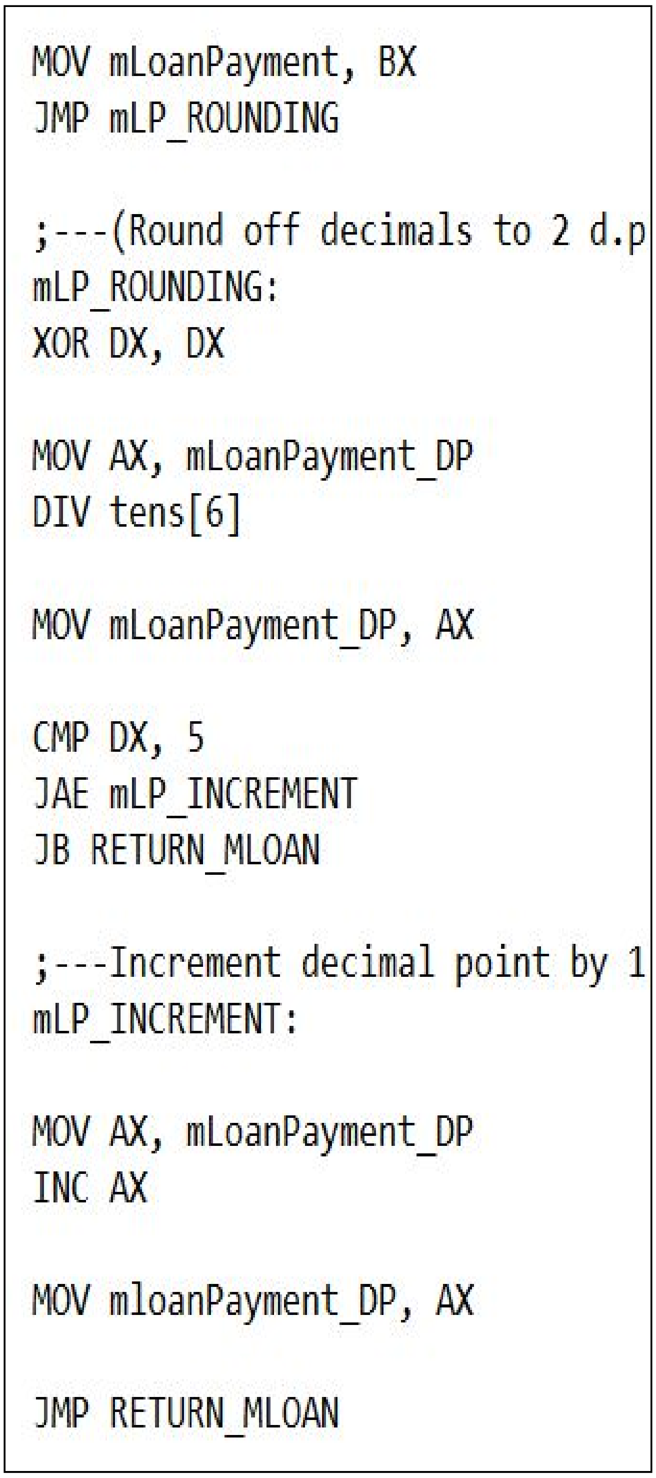
➢ Move back the value of the remainder into mLoanPayment_DP

➢ Compare AX, 1

➢ If above or equal jump to upper_incremnt

➢ If below then continue to mLP_ROUDNING

➢ Add the carry decimal from AX to mLoanPayment through BX.



```
MOV mLoanPayment, BX
JMP mLP_ROUNDING

;---(Round off decimals to 2 d.p
mLP_ROUNDING:
XOR DX, DX

MOV AX, mLoanPayment_DP
DIV tens[6]

MOV mLoanPayment_DP, AX

CMP DX, 5
JAE mLP_INCREMENT
JB RETURN_MLOAN

;---Increment decimal point by 1
mLP_INCREMENT:

MOV AX, mLoanPayment_DP
INC AX

MOV mloanPayment_DP, AX

JMP RETURN_MLOAN
```

➢ Store back the new value of mLoanPayment. (Whole number)

➢ mLP_ROUNDING roundes of the decimals to 2 decimal places (Currently has 3 dp)

➢ Get the last digit of the decimal point by dividing 10. (AX register)

➢ Move back the new value of mLoanPayment_DP using AX register. (2 dp now)

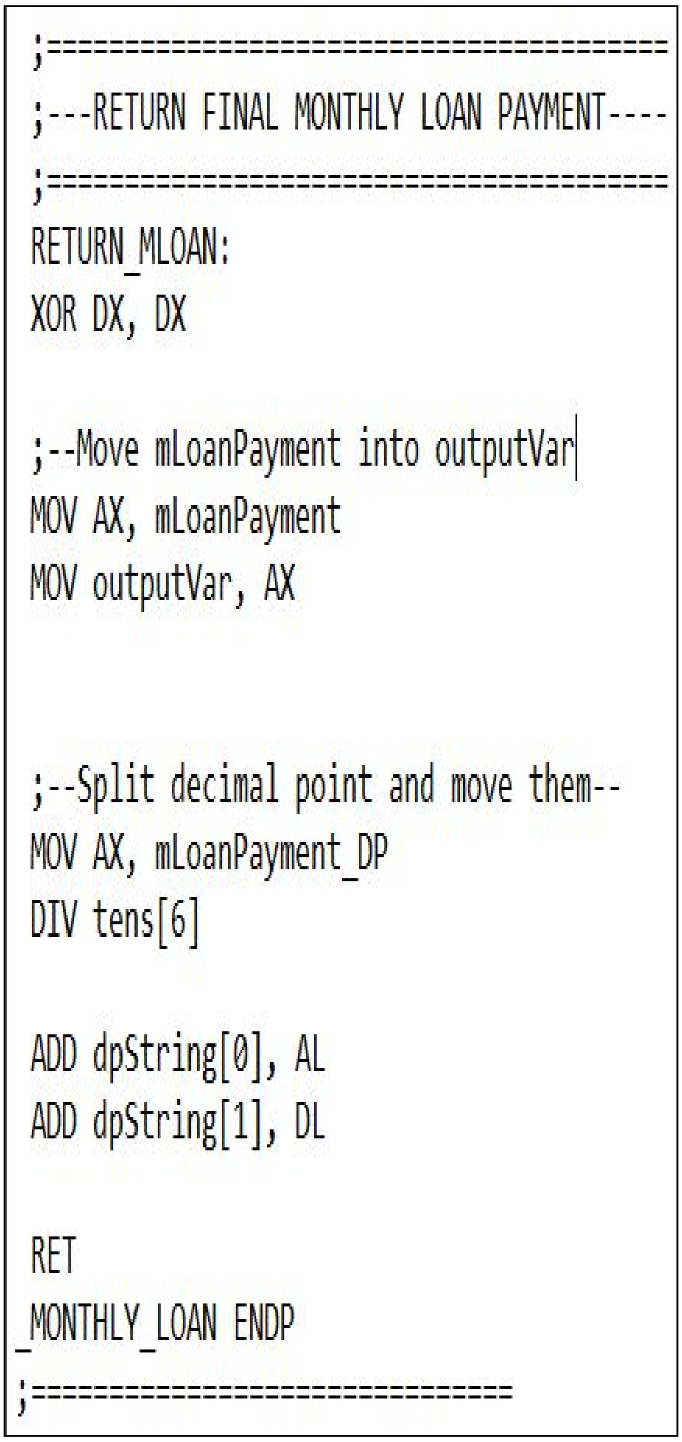
➢ Compare the remainder with 5 (DX register)

➢ If above or equal, jump to mLP_INCREMENT

➢ Else if below, jump of RETURN_MLOAN

➢ mLP_INCREMENT increments the new value of mLoanPayment_DP by 1 (Using AX register)

➢ Store the new value back to mLoanPayment_DP and jump to RETURN_MLOAN



```
;========================================
;---RETURN FINAL MONTHLY LOAN PAYMENT----
;========================================
RETURN_MLOAN:
XOR DX, DX

;--Move mLoanPayment into outputVar
MOV AX, mLoanPayment
MOV outputVar, AX



;--Split decimal point and move them--
MOV AX, mLoanPayment_DP
DIV tens[6]

ADD dpString[0], AL
ADD dpString[1], DL

RET
_MONTHLY_LOAN ENDP
;==============================
```

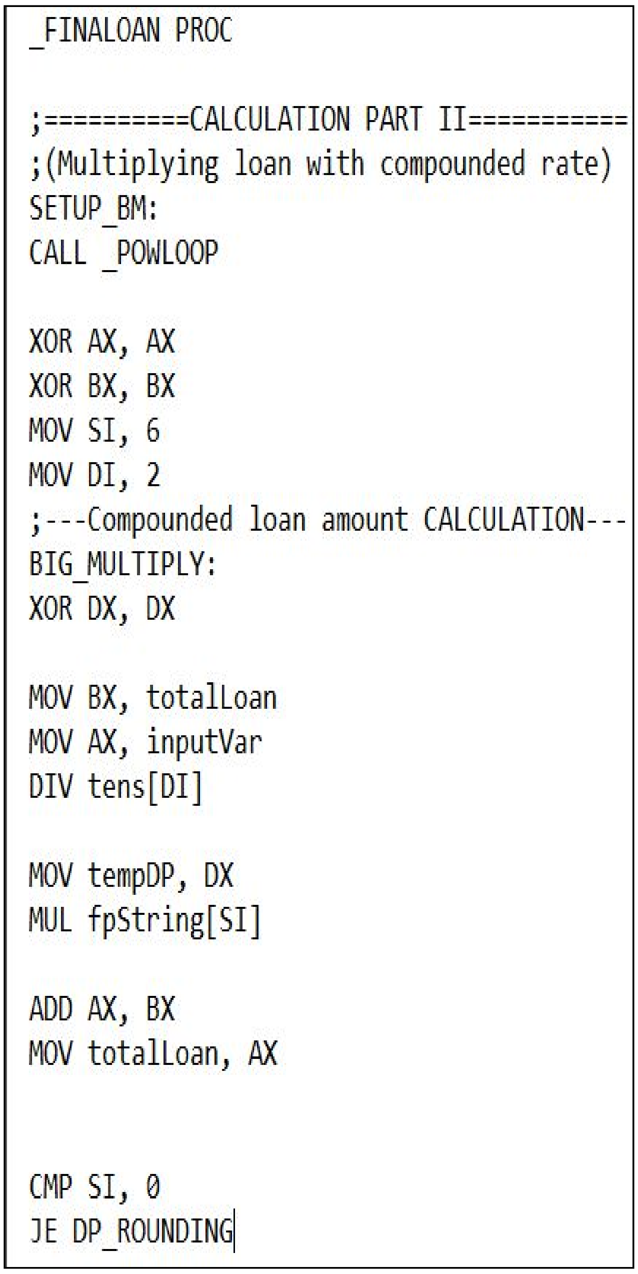
- RETURN_MLOAN moves back the result into the outputVar and splits the decimal points into an array for output later.
- Return function

Total Compounded Loan Function

OOI KWOK QUEN

$$P(1+\frac{R}{N})^{(N\times T)}$$

- Multiply a 5 digit variable with floating point (3 decimal places)
- (1 + R / N) ^ (N x T) is already set by the POWLOOP function
- P is the Loan Amount



```
_FINALOAN PROC

;==========CALCULATION PART II===========
;(Multiplying loan with compounded rate)
SETUP_BM:
CALL _POWLOOP

XOR AX, AX
XOR BX, BX
MOV SI, 6
MOV DI, 2
;---Compounded loan amount CALCULATION---
BIG_MULTIPLY:
XOR DX, DX

MOV BX, totalLoan
MOV AX, inputVar
DIV tens[DI]

MOV tempDP, DX
MUL fpString[SI]

ADD AX, BX
MOV totalLoan, AX


CMP SI, 0
JE DP_ROUNDING
```

➢ CALL _POWLOOP to get the value of (1 + R/N) ^ (T x N)

➢ BIG_MULTIPLY divides the inputVar with tens array from (1000, 100, 10, 1)

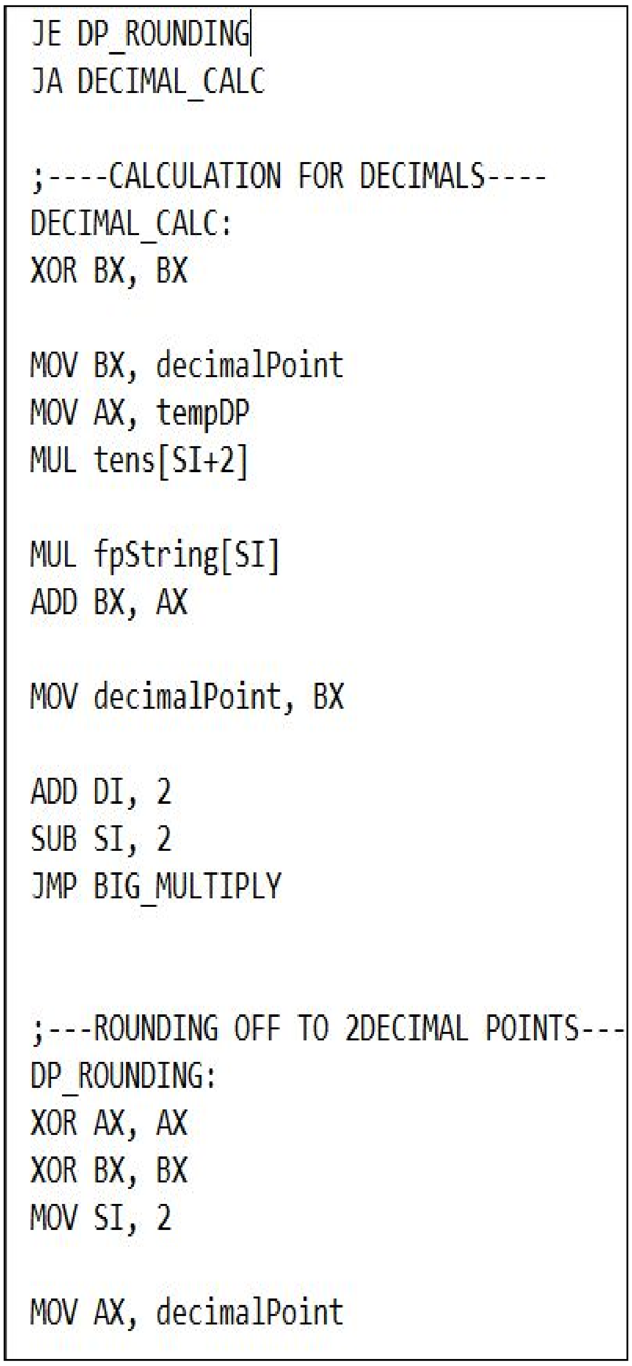
➢ Move the remainder in a temporary variable called tempDP

➢ Multiply the AX register with the string array of the power variable (Starting from the last digit to the first using SI register as offset address)

➢ Add the AX with the totalLoan (BX)

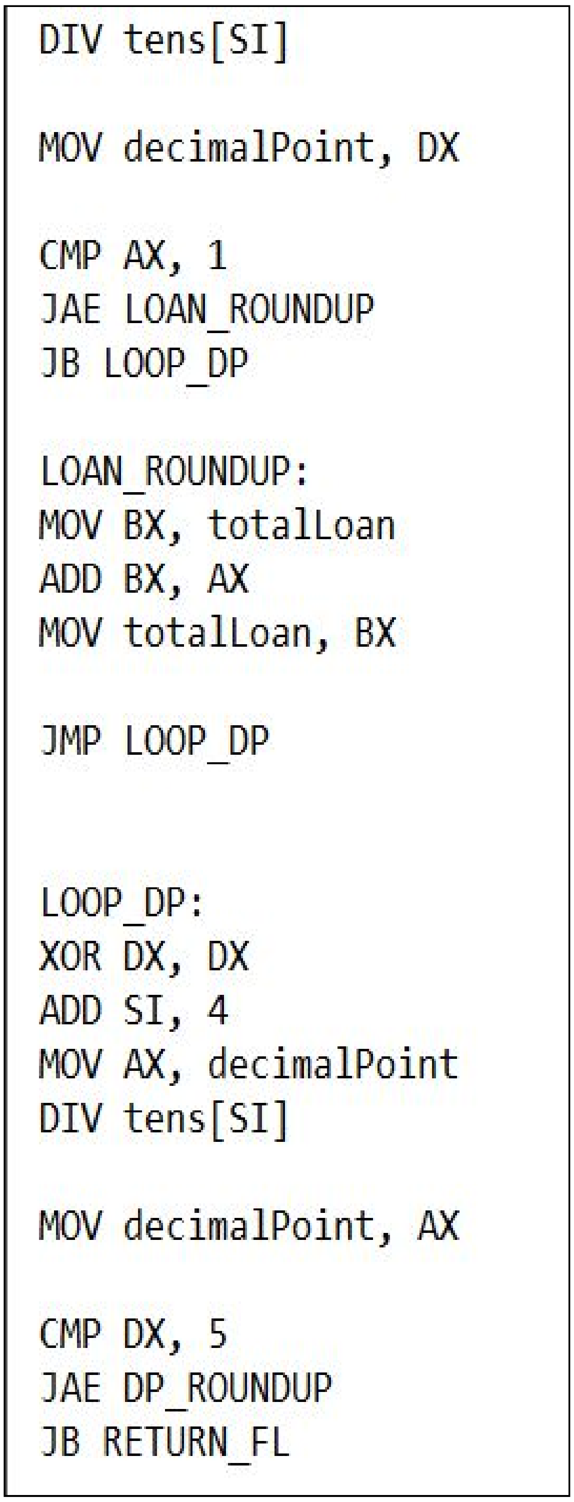
➢ Store the value into totalLoan (AX)

- Compare SI register with 0.



```
JE DP_ROUNDING
JA DECIMAL_CALC

;----CALCULATION FOR DECIMALS----
DECIMAL_CALC:
XOR BX, BX

MOV BX, decimalPoint
MOV AX, tempDP
MUL tens[SI+2]

MUL fpString[SI]
ADD BX, AX

MOV decimalPoint, BX

ADD DI, 2
SUB SI, 2
JMP BIG_MULTIPLY


;---ROUNDING OFF TO 2DECIMAL POINTS---
DP_ROUNDING:
XOR AX, AX
XOR BX, BX
MOV SI, 2

MOV AX, decimalPoint
```

- ➢ If equal to 0, jump to DP_ROUNDING
- ➢ If above 0, jump to DECIMAL_CALC
- ➢ DECIMAL_CALC, does the calculation for all the decimals.
- ➢ Multiply the tempDP to get 3 digits
- ➢ Multiply the result with the fpString array starting with the last digit (SI register as offset address)
- ➢ Add the AX register into the decimalPoint variable
- ➢ Store the new value of decimalPoint (BX)
- ➢ ADD DI with 2, and subtract SI with 2 to change the offset address the the array varaibles. (16 - bit)
- ➢ Jump to BIG_MULTIPLY label
- ➢ DP_ROUNDING, Round off the 3 decimal points into 2 decimal points.
- ➢ Move in current value of decimalPoint (AX)



```
DIV tens[SI]

MOV decimalPoint, DX

CMP AX, 1
JAE LOAN_ROUNDUP
JB LOOP_DP

LOAN_ROUNDUP:
MOV BX, totalLoan
ADD BX, AX
MOV totalLoan, BX

JMP LOOP_DP


LOOP_DP:
XOR DX, DX
ADD SI, 4
MOV AX, decimalPoint
DIV tens[SI]

MOV decimalPoint, AX

CMP DX, 5
JAE DP_ROUNDUP
JB RETURN_FL
```

➢ (Handle carry over)Divide the AX register with 1000 to get the carry over decimal (SI)
➢ Store the remainder into decimalPoint (3 digit now)

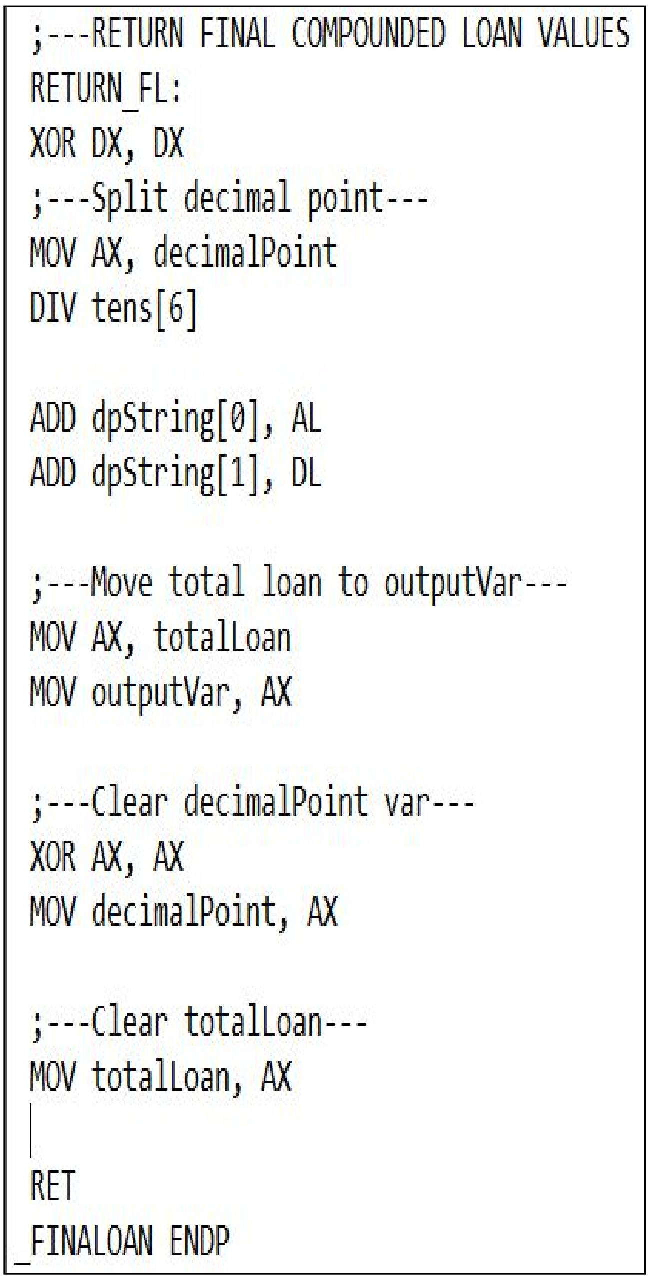
➢ Compare the AX register with 1, if above or equal then jump to LOAN_ROUNDUP
➢ Else if below then jump to LOOP_DP

➢ LOOP_ROUNDUP - Adds the carry over to the whole number totalLoan (AX, BX)
➢ Store the new value of totalLoan (BX)
➢ Jump to LOOP_DP

➢ LOOP_DP - Round off the 3rd digit/ decimal into 2 digits / decimal places.
➢ Divide decimalPoint with 10 to get the last digit value.

➢ Move back the new value of decimalPoint (AX, now has 2 digit)

➢ Compare the remainder with 5, if more than or equal jump to DP_ROUNDUP (DX)
➢ Else if below jump to RETURN_FL



```
;---RETURN FINAL COMPOUNDED LOAN VALUES
RETURN_FL:
XOR DX, DX
;---Split decimal point---
MOV AX, decimalPoint
DIV tens[6]

ADD dpString[0], AL
ADD dpString[1], DL

;---Move total loan to outputVar---
MOV AX, totalLoan
MOV outputVar, AX

;---Clear decimalPoint var---
XOR AX, AX
MOV decimalPoint, AX

;---Clear totalLoan---
MOV totalLoan, AX

RET
_FINALOAN ENDP
```

➢ RETURN_FL - Returns final compounded loan values and clears non-constant variables for reusability.

➢ Split the decimal points by dividing decimalPoint with 10 and store the answer into an output array called dpString.

➢ Move totalLoan to the outputVar for output printing.

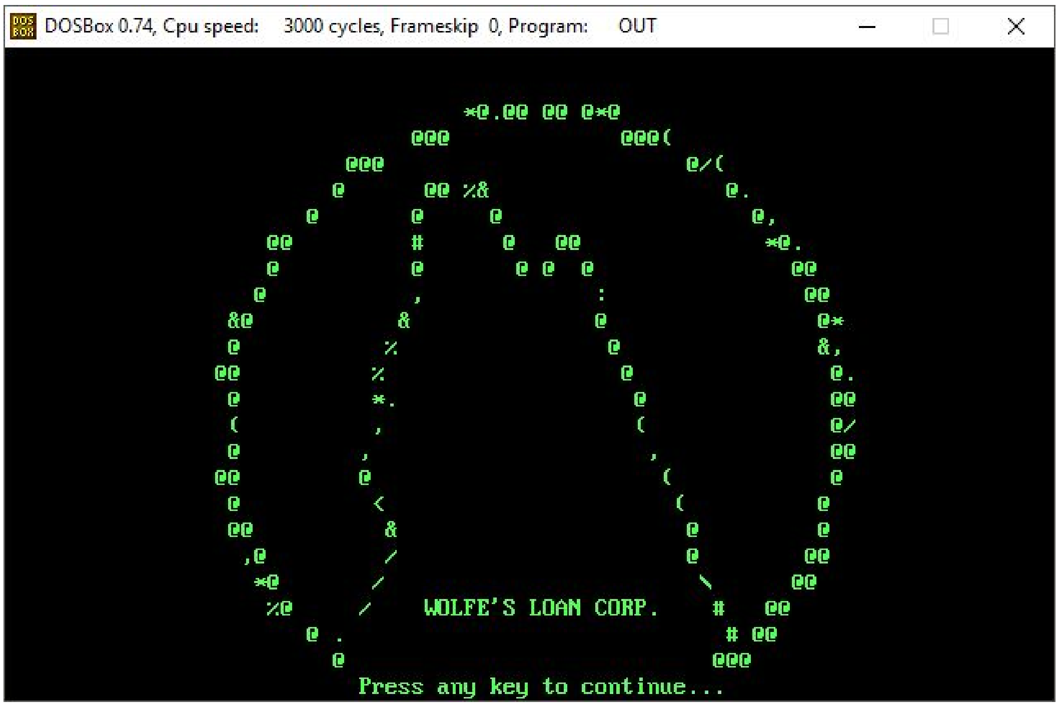
➢ Clear variables such as decimalPoint and totalLoan.

➢ Return function

I/O Design

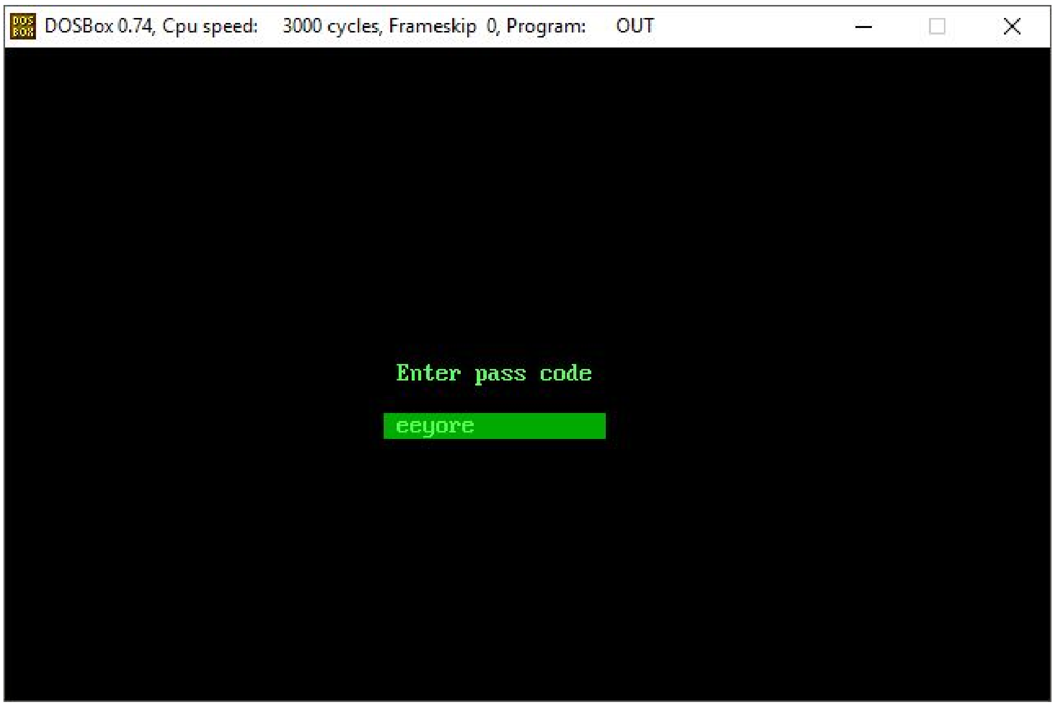
The I/O designs of the program shall be presented via print screen outputs:

Logo (No previous action)



```
DOSBox 0.74, Cpu speed:   3000 cycles, Frameskip  0, Program:   OUT

WOLFE'S LOAN CORP.

Press any key to continue...
```

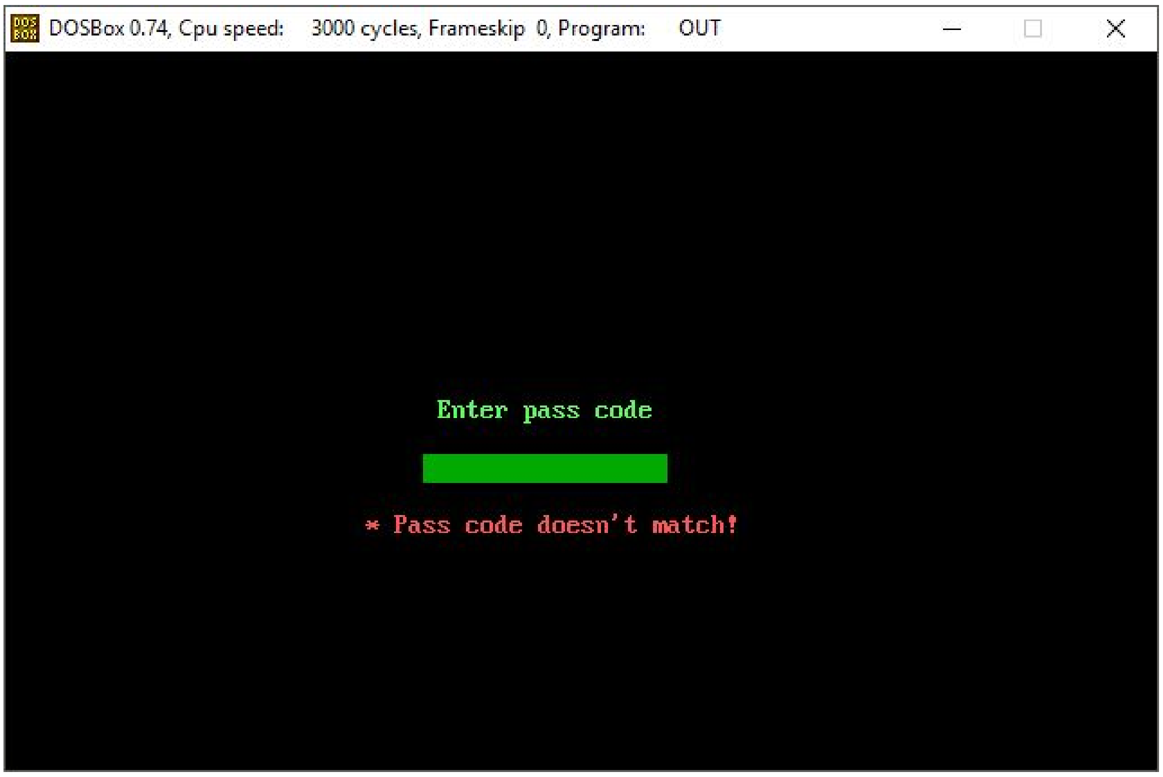
Login screen (Pressed a key)



```
DOSBox 0.74, Cpu speed:   3000 cycles, Frameskip  0, Program:   OUT

Enter pass code

eeyore
```

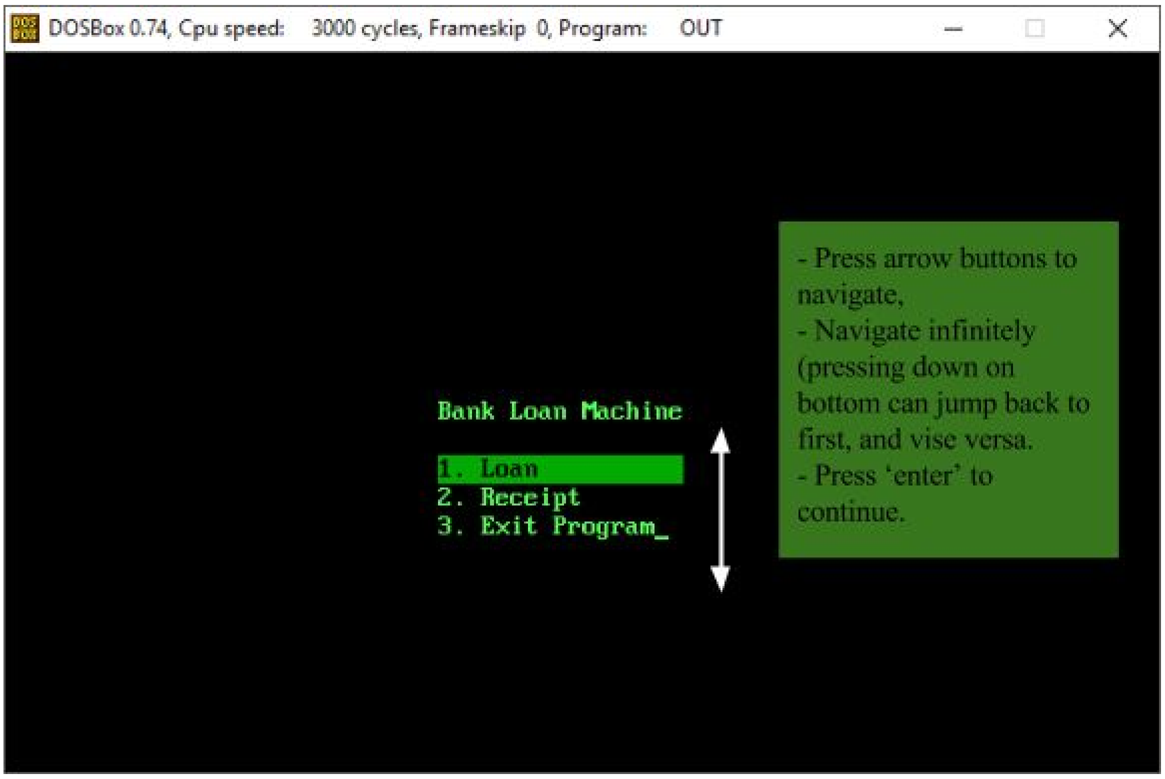
Pass code : eeyore

Login Screen (Pressed ‘Enter’, Incorrect Pass Code)


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

```
Enter pass code

* Pass code doesn't match!
```

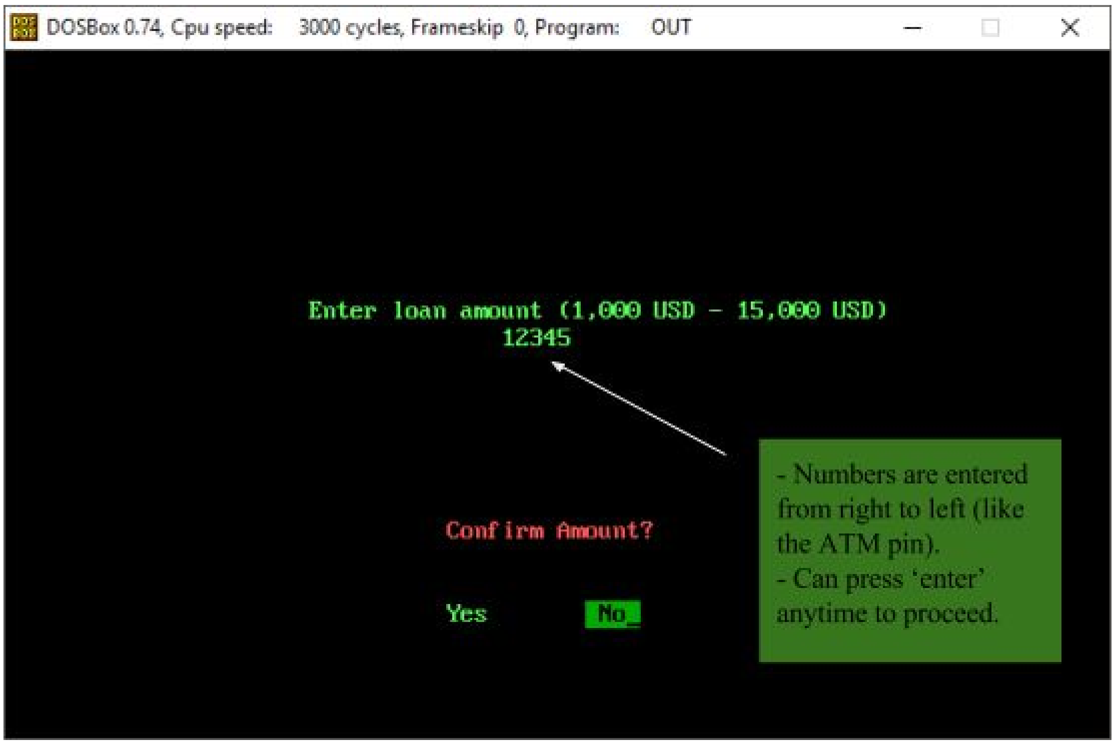
Main Menu (Pressed ‘Enter’, Correct Pass Code)


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

```
Bank Loan Machine

1. Loan
2. Receipt
3. Exit Program_
```

- Press arrow buttons to navigate,
- Navigate infinitely (pressing down on bottom can jump back to first, and vise versa.
- Press ‘enter’ to continue.

Loan Module Screen 1 - Main (Chose ‘Loan’, pressed ‘Enter’)

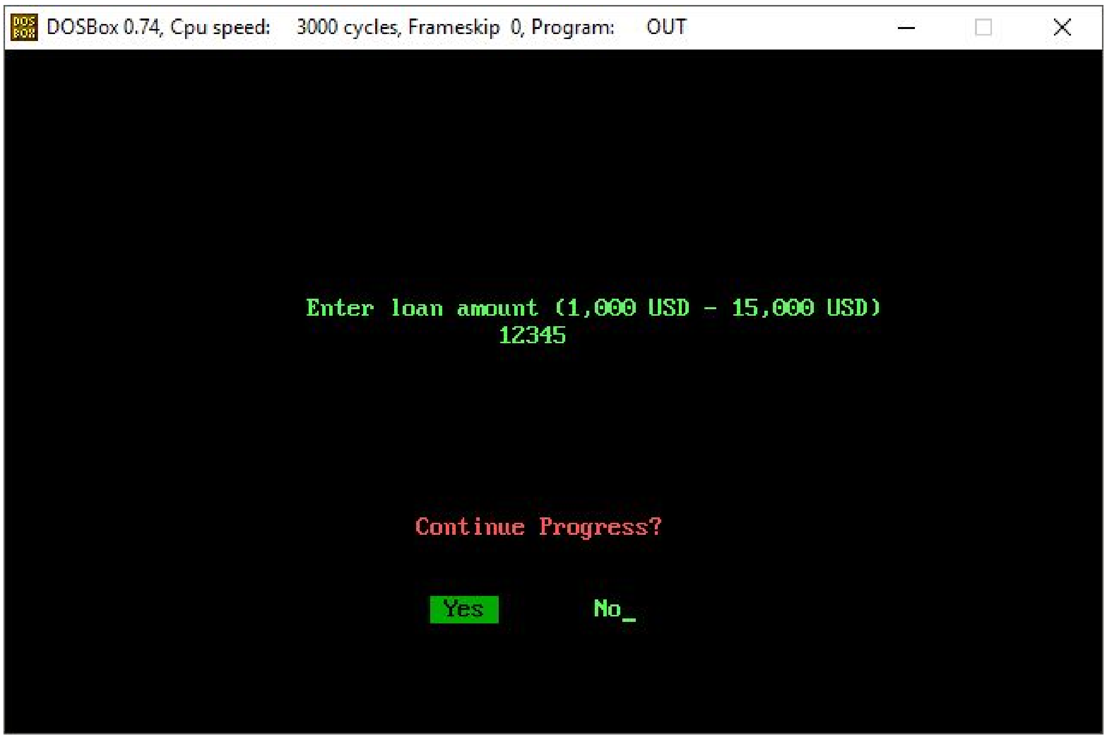

DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

```
Enter loan amount (1,000 USD - 15,000 USD)
                12345


           Confirm Amount?

           Yes          No_
```

- Numbers are entered from right to left (like the ATM pin).
- Can press ‘enter’ anytime to proceed.

Sample Data for Loan Amount : 12345

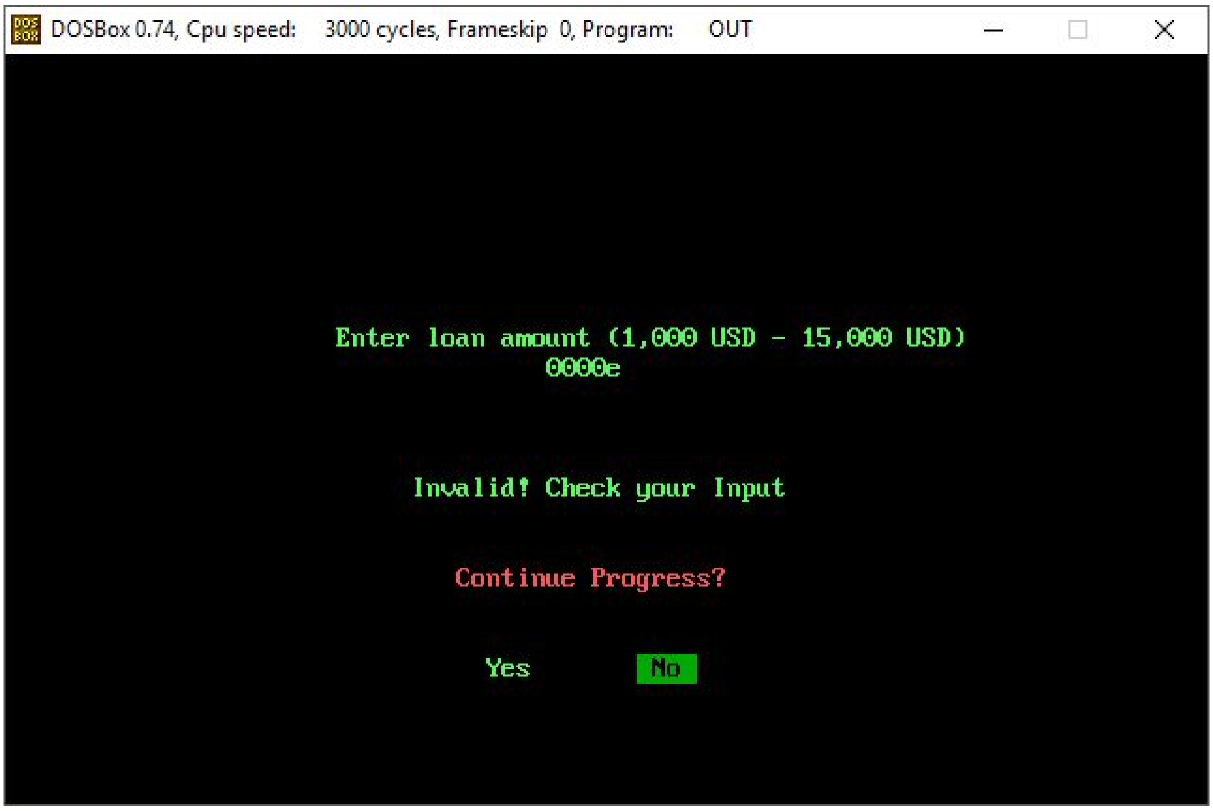
Loan Module Screen 1 - Prompt Back to Main Menu (Confirm Amount : No)


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

```
Enter loan amount (1,000 USD - 15,000 USD)
                12345


          Continue Progress?

           Yes          No_
```

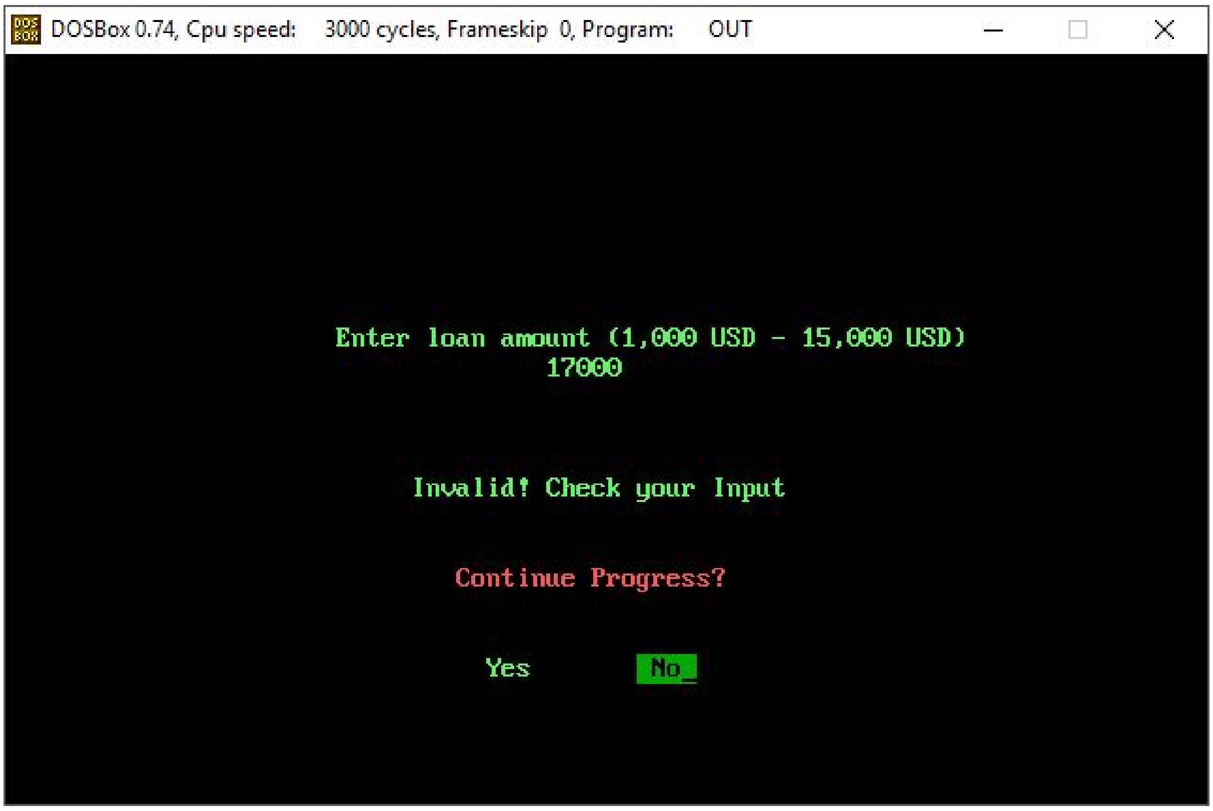
Continue Progress : Yes (Loop Back to Loan Module Screen 1, Loan Amount resetted)

Loan Module Screen 1 - Error (Invalid Character)


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

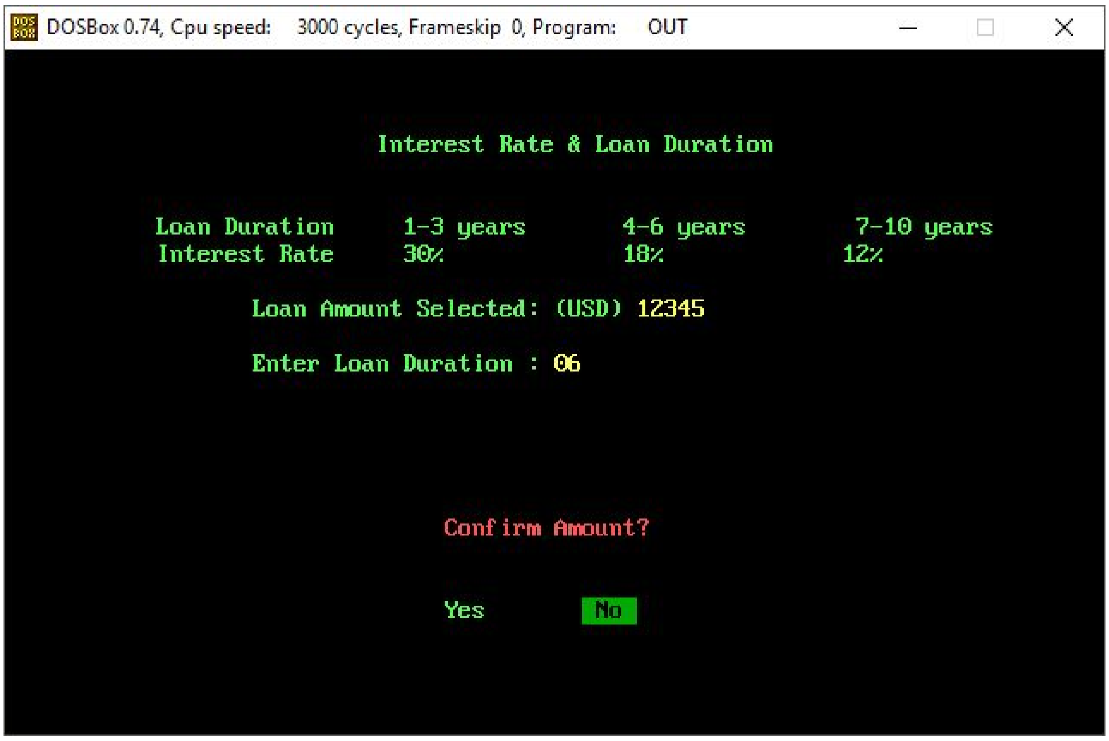
```
Enter loan amount (1,000 USD - 15,000 USD)
           0000e


     Invalid! Check your Input

        Continue Progress?

        Yes        No
```

Loan Module Screen 1 - Error (Input out of range)


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

```
Enter loan amount (1,000 USD - 15,000 USD)
           17000


     Invalid! Check your Input

        Continue Progress?

        Yes        No_
```

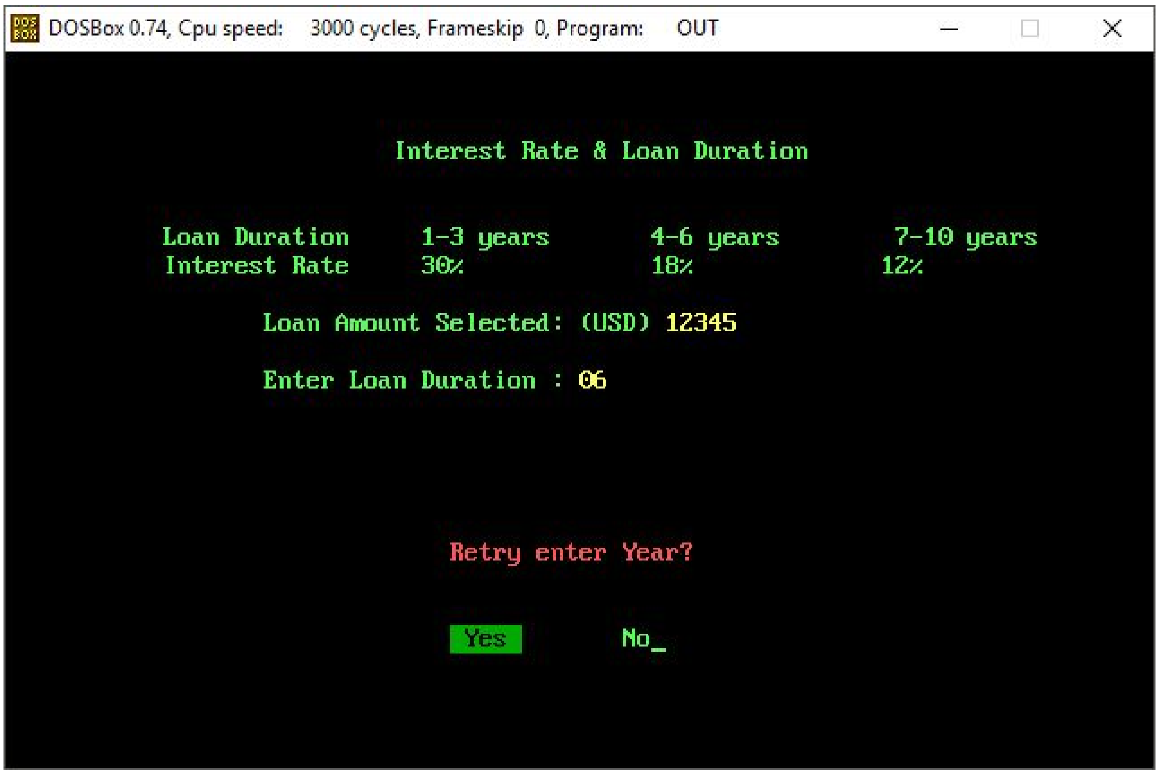
Loan Module Screen 2 - Main (Confirm Amount : Yes)



```
DOSBox 0.74, Cpu speed:    3000 cycles, Frameskip  0, Program:    OUT

                     Interest Rate & Loan Duration

         Loan Duration      1-3 years      4-6 years       7-10 years
         Interest Rate      30%            18%             12%

               Loan Amount Selected: (USD) 12345

               Enter Loan Duration : 06



                          Confirm Amount?

                          Yes       No
```

Sample Data for Loan Duration : 06

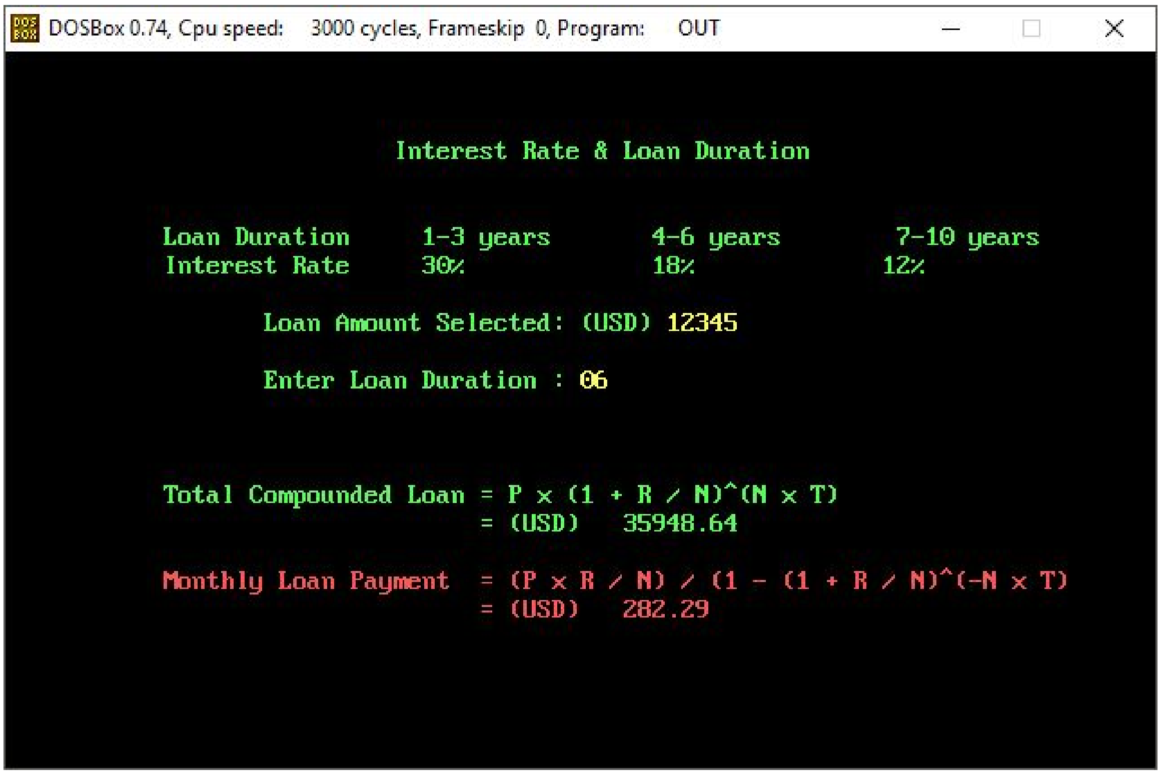
Loan Module Screen 2 - Retry Enter Year (Confirm Amount : No)



```
DOSBox 0.74, Cpu speed:    3000 cycles, Frameskip  0, Program:    OUT

                     Interest Rate & Loan Duration

         Loan Duration      1-3 years      4-6 years       7-10 years
         Interest Rate      30%            18%             12%

               Loan Amount Selected: (USD) 12345

               Enter Loan Duration : 06



                          Retry enter Year?

                          Yes       No_
```

Retry enter Year : Yes (Loop Back to Loan Module Screen 2, Loan Duration Resetted)

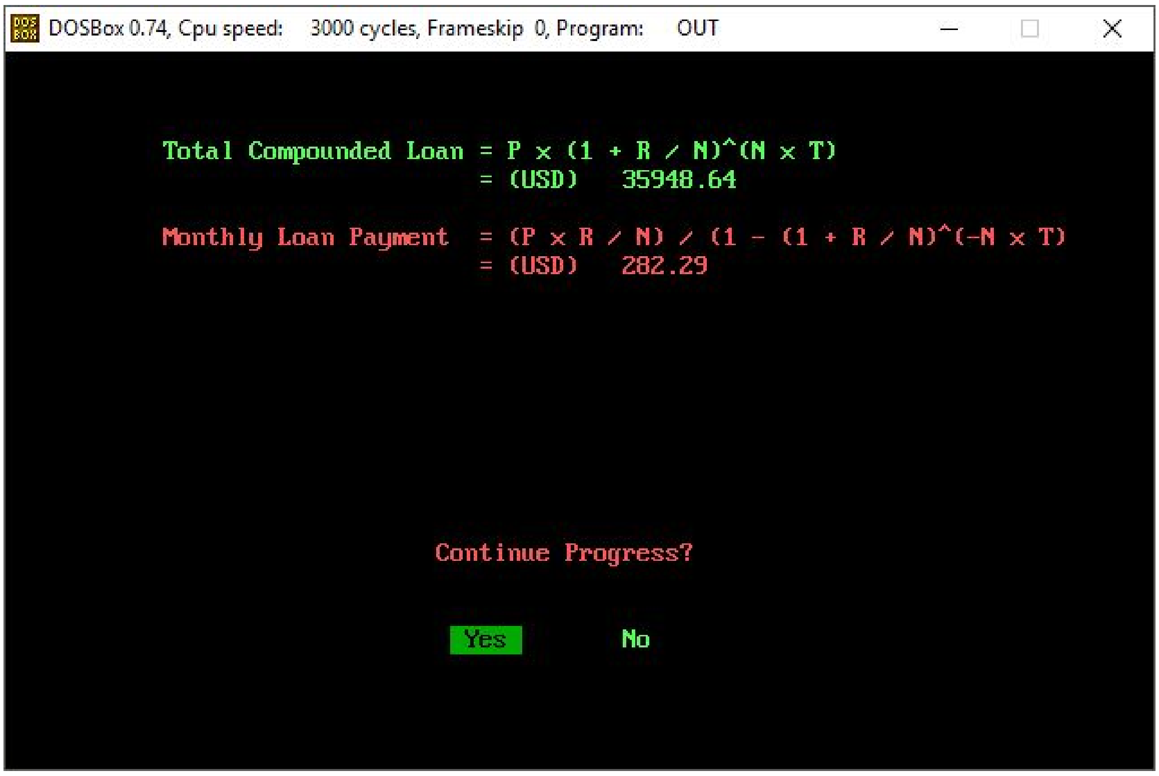
Loan Module Screen 2 - Result (Confirm Amount : Yes)



DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

```
                          Interest Rate & Loan Duration

          Loan Duration      1-3 years       4-6 years        7-10 years
          Interest Rate      30%             18%             12%

                 Loan Amount Selected: (USD) 12345

                 Enter Loan Duration : 06


          Total Compounded Loan = P x (1 + R / N)^(N x T)
                                = (USD)    35948.64

          Monthly Loan Payment  = (P x R / N) / (1 - (1 + R / N)^(-N x T)
                                = (USD)    282.29
```

Load Module Screen 2 - Continue? (Pressed a key)

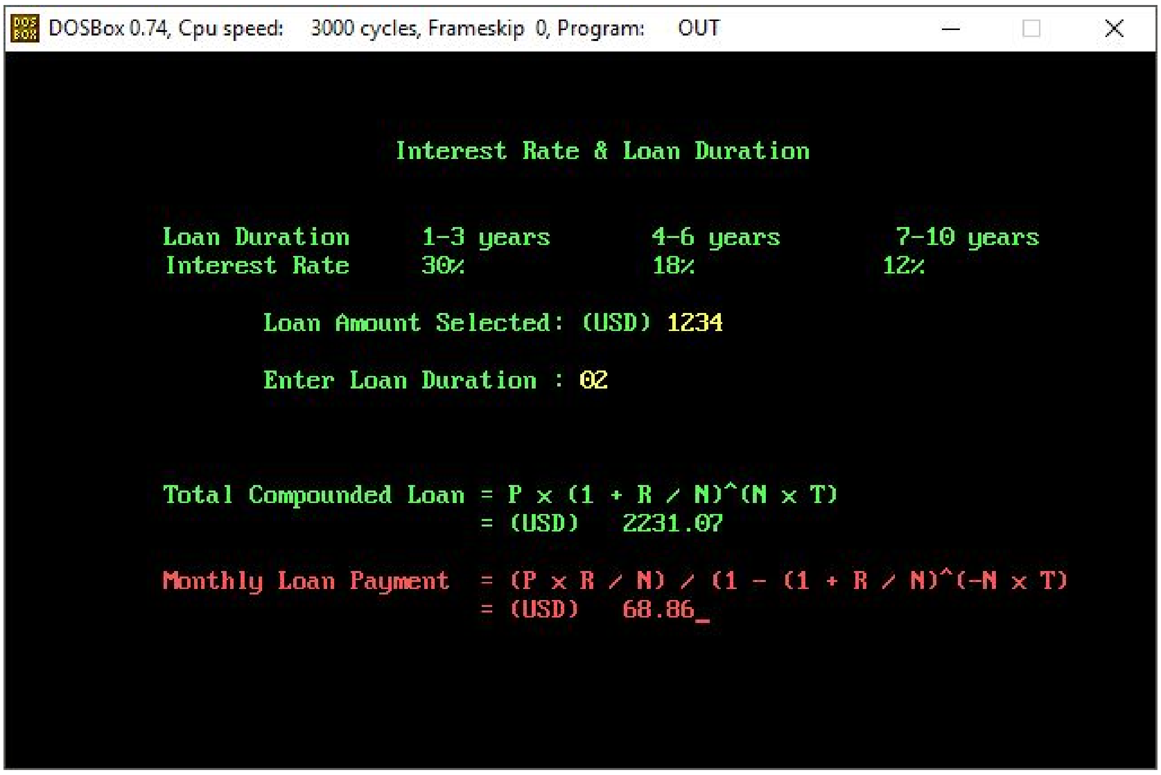


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

```
          Total Compounded Loan = P x (1 + R / N)^(N x T)
                                = (USD)    35948.64

          Monthly Loan Payment  = (P x R / N) / (1 - (1 + R / N)^(-N x T)
                                = (USD)    282.29



                              Continue Progress?

                               Yes       No
```

Continue Progress : Yes (Goes back to Loan Module Screen 1)
Continue Progress : No (Goes back to Main Menu)

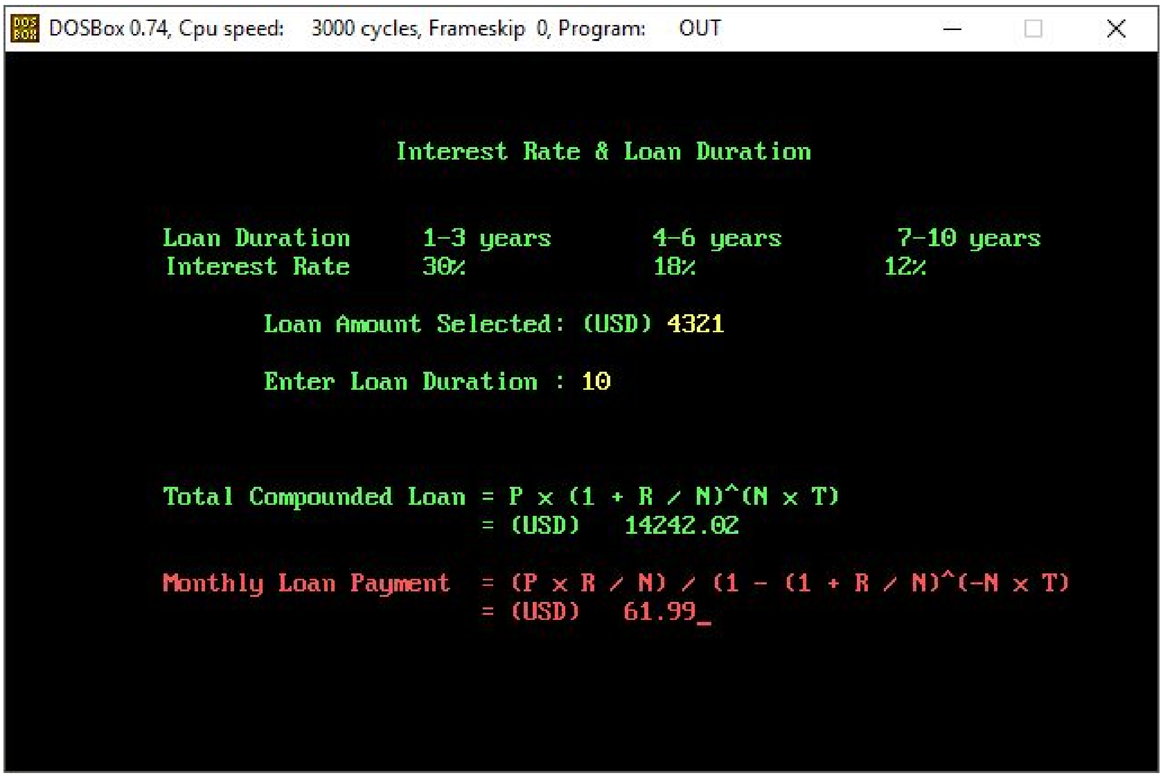
Loan Module Screen 2 - Result (More Samples)



DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

```
                    Interest Rate & Loan Duration

  Loan Duration      1-3 years       4-6 years       7-10 years
  Interest Rate      30%             18%             12%

         Loan Amount Selected: (USD) 1234

         Enter Loan Duration : 02


  Total Compounded Loan = P x (1 + R / N)^(N x T)
                        = (USD)   2231.07

  Monthly Loan Payment  = (P x R / N) / (1 - (1 + R / N)^(-N x T)
                        = (USD)   68.86_
```

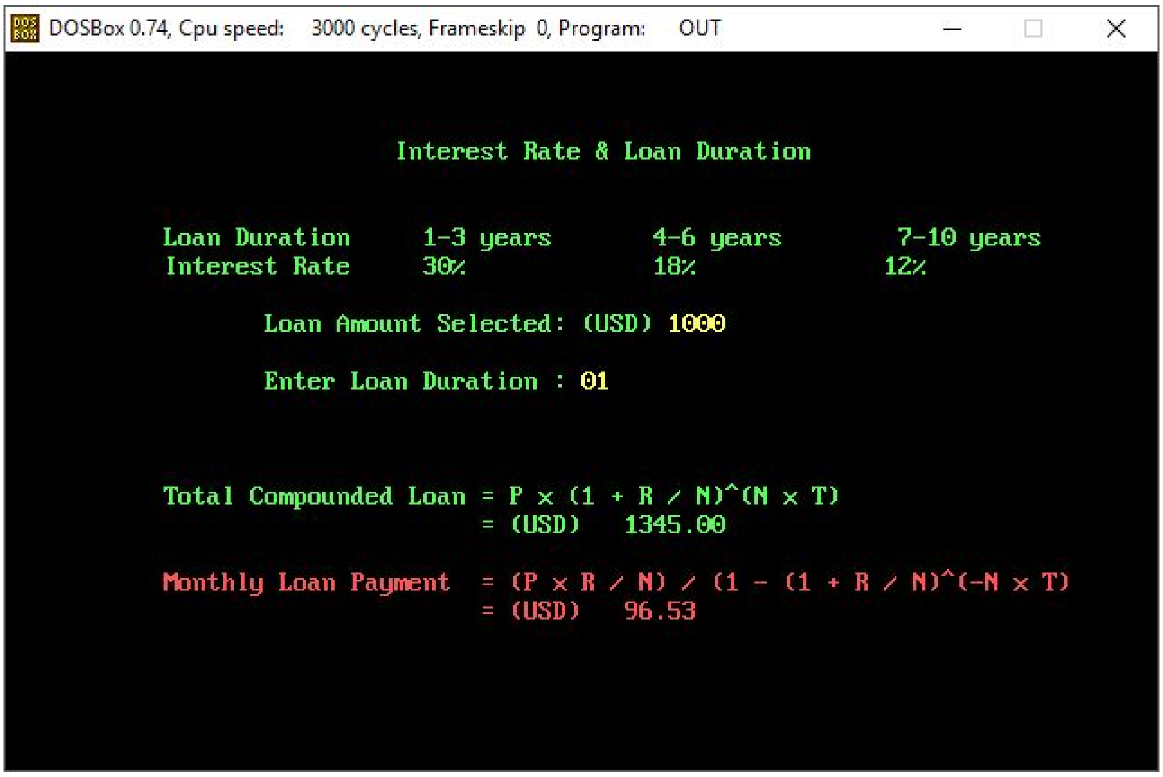
Load Module Screen 2 - Result (More Samples)



DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

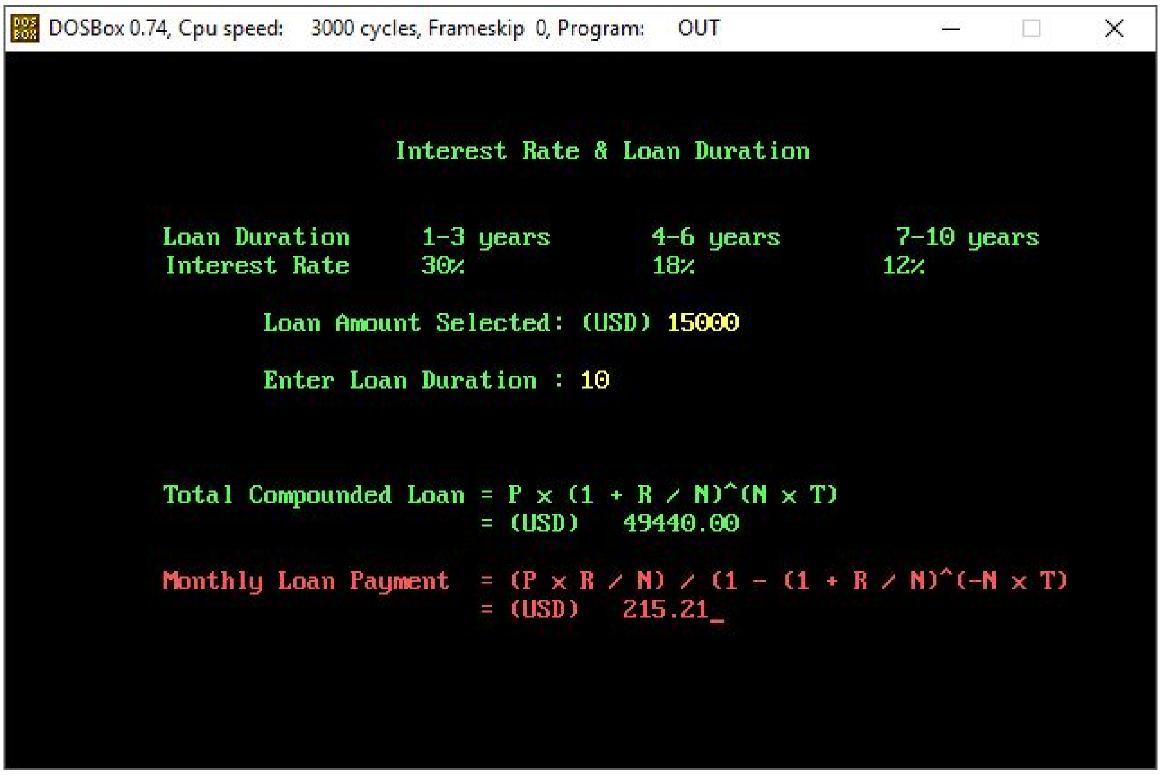
```
                    Interest Rate & Loan Duration

  Loan Duration      1-3 years       4-6 years       7-10 years
  Interest Rate      30%             18%             12%

         Loan Amount Selected: (USD) 4321

         Enter Loan Duration : 10


  Total Compounded Loan = P x (1 + R / N)^(N x T)
                        = (USD)   14242.02

  Monthly Loan Payment  = (P x R / N) / (1 - (1 + R / N)^(-N x T)
                        = (USD)   61.99_
```

Loan Module Screen 2 - Result (Min. Result)


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

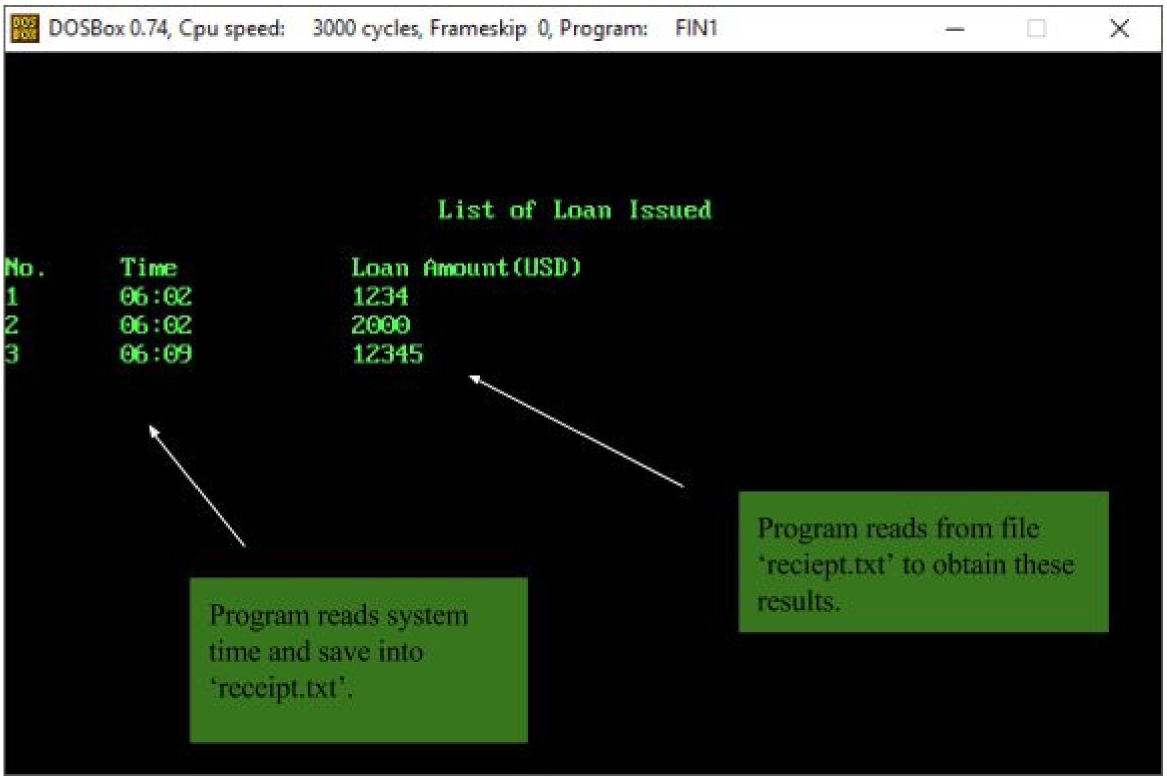
```
                 Interest Rate & Loan Duration

Loan Duration      1-3 years       4-6 years        7-10 years
Interest Rate      30%             18%             12%

       Loan Amount Selected: (USD) 1000

       Enter Loan Duration : 01


Total Compounded Loan = P x (1 + R / N)^(N x T)
                      = (USD)   1345.00

Monthly Loan Payment  = (P x R / N) / (1 - (1 + R / N)^(-N x T)
                      = (USD)   96.53
```

Loan Module Screen 2 - Result (Max. Result)


DOSBox 0.74, Cpu speed: 3000 cycles, Frameskip 0, Program: OUT

```
                 Interest Rate & Loan Duration

Loan Duration      1-3 years       4-6 years        7-10 years
Interest Rate      30%             18%             12%

       Loan Amount Selected: (USD) 15000

       Enter Loan Duration : 10


Total Compounded Loan = P x (1 + R / N)^(N x T)
                      = (USD)   49440.00

Monthly Loan Payment  = (P x R / N) / (1 - (1 + R / N)^(-N x T)
                      = (USD)   215.21_
```

Receipt Module - Main (No Previous Action)



```
DOSBox 0.74, Cpu speed:    3000 cycles, Frameskip  0, Program:    FIN1

                               List of Loan Issued

No.      Time             Loan Amount(USD)
1        06:02            1234
2        06:02            2000
3        06:09            12345
```

Program reads system time and save into 'receipt.txt'.

Program reads from file 'reciept.txt' to obtain these results.

Sample Data : 12345 USD as Loan Amount

User Guide

Wolfe's Loan Corporation also provides a guideline for its employee on how to use the program.

Software Used

The completion of the program relies on:

- Microsoft Macro Assembler 8.0 (MASM) Package (x86)
- DOSBox 0.74 on Windows 10 v1709
- DOSBox 0.74 on Point Linux 3.2 running Wine 1.6.2-20
- emu8086 v4.0.8 on Windows 10 v1709
- emu8086 v4.0.8 on Point Linux 3.2 running Wine 1.6.2-20

The completion of this document relies on:

- Google Docs
- Google Drive
- Adobe Acrobat DC
- SmartDraw 2013 Enterprise Edition

The requirements on installing and using the software:

- Windows XP and later (only tested on Windows 7 and Windows 10)
  OR
  A Linux machine capable of running Wine (only tested on Debian (Jessie) based Point Linux 3.2 running latest Wine version)
- Microsoft Macro Assembler 8.0 (MASM) Package (x86)
- DOSBox 0.74

Installation

1. Check software requirements.
2. Obtain a copy of the program.
3. Install DOSBox 0.74 .
4. Copy ‘8086’ folder to root directory of any drive (drive C in this example).
5. Copy the [filename].asm and ‘reciept.txt’ files to the ‘8086’ folder.
6. Place
7. Launch DOSBox 0.74 .
8. The following steps will mount the ‘8086’ folder onto the emulator and assemble the source code.
   On the MS-DOS screen (not the emulator console), type:
   ```
   mount C C:\8086\
   C:
   masm [filename].asm;
   link [filename].obj;
   ```
9. The executable (.exe) file should appear in the ‘8086’ folder.
10. To make sure the program runs properly, clear the screen before executing.
    On the MS-DOS screen, type:
    ```
    cls
    ```
11. Execute the program.
    On the MS-DOS screen, type:
    ```
    [filename].exe
    ```

Login Password

The login password is ‘eeyore’ without quotes.

Usage Guidelines

To calculate loan:

1. Execute the program as per guided in the ‘Installation’ section.
2. Enter the correct password.
3. Choose the ‘Loan’ module.
4. Enter the amount of loan desired.
5. Enter the number of year on borrowing the loan.
6. Result is displayed.

To display receipt:

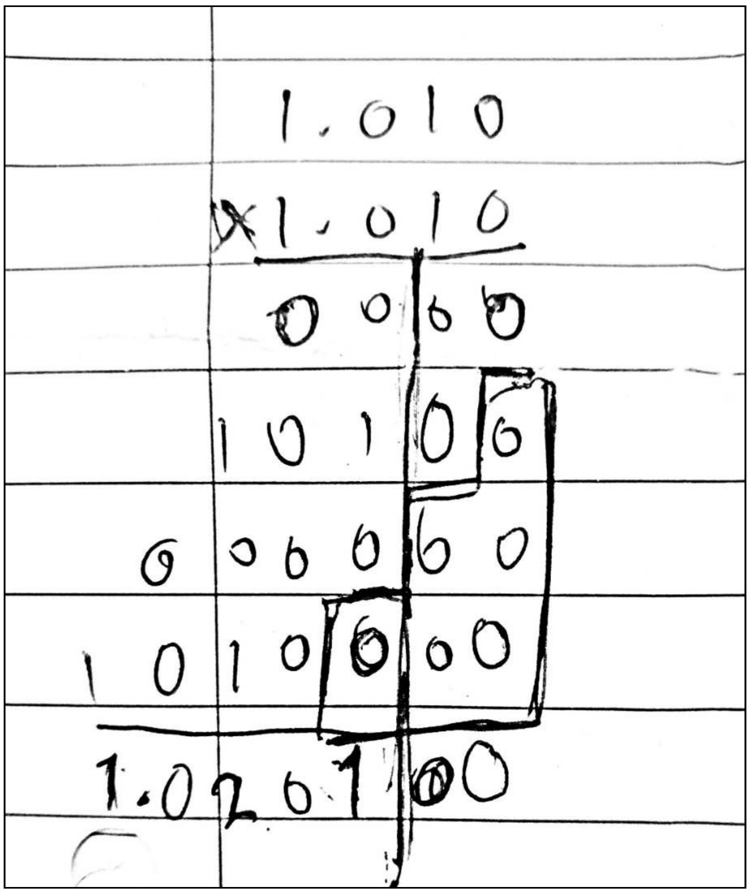
1. Execute the program as per guided in the ‘Installation’ section.
2. Enter the correct password.
3. Choose the ‘Receipt’ module.
4. Result is displayed.

References

1. Lecture Notes on AACS3064 Computer Systems Architecture (2017/2018).
2. documentation for emu8086 - assembler and microprocessor emulator. 2005. Complete 8086 instruction set. [ONLINE] Available at: http://www.emu8086.com/. [Accessed 4 January 2018].
3. Daniel B. Sedory. 2013. A P P E N D I X E S for A Guide to DEBUG (The Microsoft® DEBUG.EXE Program). [ONLINE] Available at: http://thestarman.pcministry.com/asm/debug/8086REGs.htm#REGS. [Accessed 4 January 2018].
4. Ralf Brown. 2017. The x86 Interrupt List aka "Ralf Brown's Interrupt List", "RBIL". [ONLINE] Available at: http://www.cs.cmu.edu/afs/cs.cmu.edu/user/ralf/pub/WWW/files.html. [Accessed 4 January 2018].

Appendices

The Ladder Diagram (Hand Sketch)



Used for calculation of a decimal number multiplying with another decimal number.