

CS3080-001 Spring 2024

HW02

PROBLEM:**Problem: 1**

ASSIGNMENT:

Filename: hw02_01

Write a program that opens and asks the user for the name of a Hack machine code file (WITHOUT the extension). Open the file and read the contents. Output a disassembled code listing to the console, in a format like the above (the original machine encoding, some space, and the disassembled instruction). Do NOT make this harder than it is! You do not need to know what any of these instructions do, you are merely translating patterns of ones and zeroes into corresponding strings of text. You may read the file a line at a time or all at once, it is up to you. Use a separate dictionary for the three fields of the C-type instruction.

WORK:

I created two dictionaries for C type instruction computation and one for jump mapping, using this I iterate through the file (which I ask for) and based off the key provided I assign assembly code to each line

OUTPUT:

Enter the name of a .hack file (without extension): hackman

0000000000000000 @0

1110101010001000 0=AAM;

1110111111000000 1=AAA;

1110111010000000 -1=AAA;

King 2 of 15

1110001100100001 D=MAA;JGT

1110110000010000 A=AMA;

1111110000000000 M=AAA;

1110001101000000 !D=AAA;

1110110001000010 !A=AAA;JEQ

0111111111111111 @32767

1111110001000000 !M=AAA;

1110001111000000 -D=AAA;

1110110011110000 -A=MMA;

1111110011000000 -M=AAA;

1110011111000000 D+1=AAA;

1110110111000000 A+1=AAA;

0000000011111111 @255

1111110111100011 M+1=MAA;JGE

1110001110000000 D-1=AAA;

1110110010000000 A-1=AAA;

1111110010101000 M-1=MAM;

1110000010000000 D+A=AAA;

1111000010000100 D+M=AAA;JLT

1110010011111000 D-A=MMM;

1111010011000000 ERROR=AAA;

1110000111000101 A-D=AAA;JNE

1111000111101110 D-M=MAM;JLE

1110000000000000 ERROR=AAA;

1111000000000000 D&A=AAA;

1110010101000000 D|A=AAA;

1111010101000111 ERROR=AAA;JMP

0100000000000001 @16385

@16385

King 3 of 15

CODE:

...

PROGRAMMER: Carson King

USERNAME: cking20

PROGRAM: hw02_01.py

DESCRIPTION: Dissassemble the assembly code given in the hackman file
and output it to the console and another file

(for future problems).

...

def disassemble_hack_code(filename):

 #Open the file

 with open(filename + ".hack", "r") as file:

 lines = file.readlines()

 #Dictionary for C-type instruction computation

 comp_dict = {

 "0101010": "0",

 "0111111": "1",

 "0111010": "-1",

 "0001100": "D",

 "0110000": "A",

 "0001101": "!D",

 "0110001": "!A",

 "0001111": "-D",

 "0110011": "-A",

 "0011111": "D+1",

```
"0110111": "A+1",  
"0001110": "D-1",
```

King 4 of 15

```
"0110010": "A-1",  
"0000010": "D+A",  
"0010011": "D-A",  
"0000111": "A-D",  
"1000000": "D&A",  
"0010101": "D|A",  
"1110000": "M",  
"1110001": "!M",  
"1110011": "-M",  
"1110111": "M+1",  
"1110010": "M-1",  
"1000010": "D+M",  
"1000111": "D-M",  
"1000001": "M-D",  
"1000011": "D&M",  
"1000110": "D|M"  
}
```

#Dictionary for jump instruction mapping

```
jump_dict = {  
    "000": "",  
    "001": "JGT",  
    "010": "JEQ",  
    "011": "JGE",  
    "100": "JLT",  
    "101": "JNE",
```

```

    "110": "JLE",
    "111": "JMP"
}

```

King 5 of 15

```

#List to store disassembled lines
disassembled_lines = []

#Iterate through lines and disassemble
for line in lines:
    line = line.strip() # Remove leading/trailing whitespace
    if line.startswith('0'):
        #A-type instruction
        disassembled_lines.append(line + " @" + str(int(line[1:],
2)) + "\n")
    elif line.startswith('111'):
        #C-type instruction
        comp = comp_dict.get(line[3:10], "ERROR")
        dest = ''.join('AMD'[int(bit)] for bit in line[10:13])
        jump = jump_dict.get(line[13:], "")
        disassembled_lines.append(line + " " + comp + "=" + dest +
";" + jump + "\n")

#Print the disassembled code
for line in disassembled_lines:
    print(line, end="")

#Write the disassembled code to a file
with open(filename + ".dis", "w") as output_file:
    output_file.writelines(disassembled_lines)

```

```
#Prompt user for the filename
```

```
filename = input("Enter the name of a .hack file (without extension):  
")
```

King 6 of 15

```
# Call the function to disassemble the code
```

```
disassemble_hack_code(filename)
```

PROBLEM

2.

ASSIGNMENT

Filename: hw02_02.py

Allowed modules: none

Write a program that asks the user for a loan amount, an APR, and a term (in years) and produces a

report (to the console) stating the loan terms, including the monthly payment and the amount of the

final payment. It should also provide the total amount that the borrow will pay over the life of the loan

and the cost of credit (i.e., the total amount of interest paid).

Run your program for a 30-year loan of \$388,000 at an APR of 2.5% and also at an APR of 7%.

WORK

I used the same equations i used last time and formatted it to the specifications (I already used a monthly payment calculator in hw1,), then I used tuples and method to collect and format my data, and slightly changed my print summary to fit the new additions.

OUTPUT

1 of 2

DATA ENTRY

Enter loan amount (\$): 388000

Enter loan APR (%): 2.5

Enter loan term (yr): 30

MORTGAGE TERMS

Loan amount: \$388000.0

Loan rate: 2.5%

Loan term: 30.0 years

Monthly payment: \$ 1533.07

King 8 of 15

Final Payment: \$ 1532.58

Total paid \$ 551904.71

Cost of Credit \$ 163904.71

OUTPUT

2 of 2

DATA ENTRY

Enter loan amount (\$): 388000

Enter loan APR (%): 7

Enter loan term (yr): 30

MORTGAGE TERMS

Loan amount: \$388000.0

Loan rate: 7.0%

Loan term: 30.0 years

Monthly payment: \$ 2581.37

Final Payment: \$ 2585.86

Total paid \$ 929297.69

Cost of Credit \$ 541297.69

CODE

...

PROGRAMMER: Carson L. King

USERNAME: cking20

PROGRAM: HW02_02.py

DESCRIPTION: asks user to enter a loan amount (in dollars), an APR (in percent), and a term (in years). Run a payment simulation over the term of the loan, tracking the

remaining balance as on-time payments are made and interest is collected.

King 9 of 15

...

#Gets the mortgage payment

def mortgage_payment(term, APR, loan_amount):

#gets the term amount in months

term_months = 12.0 * term;

#Monthly interest rate

monthly_interest_rate = (APR /100) / 12;

#calcs the necessary minimum monthly payment

monthly_payment = round((loan_amount * monthly_interest_rate) / (1 - (1 + monthly_interest_rate)**-term_months),2);

return monthly_payment

#Gets the mortgage residuals

def mortgage_residual(term,APR,loan_amount):

#calcs the total payment

```
#gets the term amount in months
```

```
term_months = 12.0 * term;
```

```
#Monthly interest rate
```

```
monthly_interest_rate = (APR /100) / 12;
```

```
#Sets up some holder vars
```

```
remaining_bal = loan_amount;
```

```
minimum_remaining_bal = loan_amount;
```

King 10 of 15

```
total_interest_paid = 0.0;
```

```
#gets the monthly payment
```

```
monthly_payment = mortgage_payment(term,APR,loan_amount);
```

```
for month in range (1,int(term_months) + 1):
```

```
    #minimum monthly interest
```

```
    minimum_monthly_interest = minimum_remaining_bal *  
monthly_interest_rate;
```

```
    #calc the amount paid to the principle (minimum loan)
```

```
    minimum_principal_payment = monthly_payment -  
minimum_monthly_interest;
```

```
    #update remaining balance(minimum loan)
```

```
    minimum_remaining_bal -= minimum_principal_payment;
```

```
    #total interest paid
```

```

        total_interest_paid += minimum_monthly_interest;

#Gets the final totals and rounds them correctly
final_total = total_interest_paid + loan_amount;
minimum_rounded_bal = round(minimum_remaining_bal,2);
minimum_final_payment = round(monthly_payment +
minimum_remaining_bal,2);

return monthly_payment,minimum_final_payment, final_total,
total_interest_paid

```

King 11 of 15

```

def main():
    #Variable lists
    str_loan_amount = "";
    str_APR = "";
    str_term = "";
    valid = False;

    #Creates a loop to make sure information added is correct
    while not valid:

        #Asks the user to enter their loan amount
        print("DATA ENTRY");
        str_loan_amount = input("Enter loan amount ($): ..... ");
        str_APR = input("Enter loan APR (%): ..... ");
        str_term = input("Enter loan term (yr): ..... ");
        print("");

        #converts inputs into floating points, if not displays an error
        and repeats the loop

```

```

    try:
        loan_amount = float(str_loan_amount);
        APR = float(str_APR);
        term = float(str_term);
        valid = True;
    except ValueError:
        print("Entered amount is not a number");
        valid = False;

#End of while loop
mortgage_all = mortgage_residual(term, APR, loan_amount);

King 12 of 15

#Prints the totals with the calculated minimum payment
print("MORTGAGE TERMS")
print("Loan amount: ..... $" + str(loan_amount));
print("Loan rate: ..... " + str(APR) + "%");
print("Loan term: ..... " + str(term) + " years");
print("Monthly payment: ..... $ " + str(mortgage_all[0]));
print("Final Payment: ..... $ " + str(mortgage_all[1]));
print("Total paid ..... $ " +
str(round(mortgage_all[2],2)));
print("Cost of Credit ..... $ " +
str(round(mortgage_all[3],2)));

#runs main
if __name__ == "__main__":
    main()

```

PROBLEM:

3

ASSIGNMENT:

Filename: hw02_03.py

Allowed modules: re

Capture the output from your Problem #2 code (or have your script for that problem write it to a file, your choice) named 'hackman.dis'.

Write a program that asks the user for a disassembler output file (w/o extension), reads the file (.dis

extension), and produces an assembler file (.asm extension) with just the assembly code.

The code

should also be echoed to the console.

Run your program on your hackman.dis file.

WORK:

Edited problem 1's work to adapt it to only display the assembly code, and made it output to a file.

OUTPUT:

@0

0=AAM;

1=AAA;

**-1=AAA;
D=MAA;JGT
A=AMA;
M=AAA;
!D=AAA;
!A=AAA;JEQ
@32767
!M=AAA;
-D=AAA;
-A=MMA;**

King 14 of 15

**-M=AAA;
D+1=AAA;
A+1=AAA;
@255
M+1=MAA;JGE
D-1=AAA;
A-1=AAA;
M-1=MAM;
D+A=AAA;
D+M=AAA;JLT
D-A=MMM;
ERROR=AAA;
A-D=AAA;JNE
D-M=MAM;JLE
ERROR=AAA;
D&A=AAA;
D|A=AAA;
ERROR=AAA;JMP
@16385**

CODE:

'''

PROGRAMMER: Carson King

USERNAME: cking20

PROGRAM: hw02_03.py

DESCRIPTION: Takes the output for question one and outputs only the assembly code

'''

def main():

#Input the filename from the user

filename = input("Enter a disassembler output file name (w/o extension): ")

King 15 of 15

try:

#Open the disassembler output file

with open(filename + ".dis", "rt") as dis_file:

#Read the contents of the file

disassembly_code = dis_file.readlines()

#Process each line to extract the assembly instructions

assembly_code = ""

for line in disassembly_code:

Splits the line by whitespace and get the assembly instruction part

parts = line.split()

if len(parts) > 1:

assembly_code += parts[1] + "\n" #Adds the assembly instruction to the code

#Write the assembly code to a new file

with open(filename + ".asm", "wt") as asm_file:

```
print("ASSEMBLY CODE")  
print(assembly_code) #Prints to console  
asm_file.write(assembly_code)
```

```
except FileNotFoundError:
```

```
    print("File not found. Please make sure the file exists.")
```

```
#runs main
```

```
if __name__ == "__main__":
```

```
    main()
```