CS 2614: Computer Organization
Assembly Programming Project

Fall 2018

The UNIVERSITY of OKLAHOMA

Gallogly College of Engineering

School of Computer Science

CS 2614 Computer Organization
Assembly Programming Project
Fall-2018

This project is divided into two parts: 1) design document, and 2) assembly program. In the design document phase, you will submit a typed description of problem solving approaches and algorithm used to solve the given problem. In the assembly program phase, you will write an assembly language program to code and test the given problem.

Problem Statement:

Input a positive integer N, then compute the mean of 1 to N elements

The user input will ALWAYS be a **two-digit number** (between 01 and 99). You do NOT need to check the correctness of the input. For example, if the user input is "10", then your program will output "**5.5**".

$$\boldsymbol{Mean = \frac{1+2+3+4+5+6+7+8+9+10}{10} = 5.5}$$

Hint:

You can use the following algorithm to compute the mean.

```
def simple_mean(N):
   sum = 0
   for i = 1 to N
      sum += i
   return sum / N
```

Project Description:

1. *Part A: Design document*

The design document should describe how you are going to implement this program using flow charts or C-like pseudo-codes for each subroutine. Submit a typed description of the problem solving approaches and algorithm you will use to solve the given problem. Use words and be descriptive – what I'm looking for here is that you know how the assembler works and that you are comfortable working with it. This should be written in paragraph from.

Questions you should address apart from problem solving approaches and algorithm:

- How will you convert the two "digit" ASCII input into a two digit number?
- How will you deal with looping (since assembly gives no *while* or *for*)?
- What are your loop conditions and why (i.e. when will you exit the loop)?

Please note that these questions are only a general guideline. We just want to make sure you have started working on the project and can comment intelligently on the procedure you will implement.

Grading for design document (Worth 20% of the project)

Critical Elements	**Percentage Distribution**
Problem solving approaches	30%
Algorithm	20%
Type conversion such as ASCII to decimal, etc.	10%
Dealing with loops	10%
Loop conditions to exit the loop	10%
Articulation of Response such as free of errors, grammar, syntax, and organization	10%
Total	**100%**

2. *Part B: Assembly Program*

Submit an assembly program to solve the given problem. You can reuse the assembly codes in the textbook to solve the problem. To get started, you need to download and install the assembler simulator from the Canvas. The instructions to download, install the simulator and read, write, compile and run the code are specified in the Appendix section. Your program should be stored in a plain text file and should execute on the simulator. Following subroutines are mandatory in the programming.

- An input routine
- A routine to convert input characters to hexadecimal/binary number
- A routine to compute the sum
- A routine to convert a hexadecimal/binary number to characters
- An output routine

Grading for Assembly Program (Worth 80% of the project)

Critical Elements	**Percentage Distribution**
Documentation/Comments	10%
Routine to convert input characters to hexadecimal/binary number	20%
Routine to compute the sum	15%
Routine to convert a hexadecimal/binary number to characters	20%
Input/output subroutine	10%
Correctness of output	25%
Total	**100%**

Submission guidelines:

a) Design Document (**due Monday, Nov 16th by 11.59 PM on Canvas "project specification -Fall 2018" under assignment section**)
b) Assembly program (**due Friday, Nov 30th by 11:59 PM on Canvas "project – assembly program" under assignment section**)

Both submissions should be placed in the appropriate file uploads on Canvas by the deadline. You will demonstrate your programs to the TA from December 3rd to December 7th (Doodle poll link will be given later on Canvas to reserve your slot for the project demonstration).

Late penalty:

a. Late Design Documents will NOT be accepted
b. The penalty for late submissions of the assembly program will be **15% PER DAY**

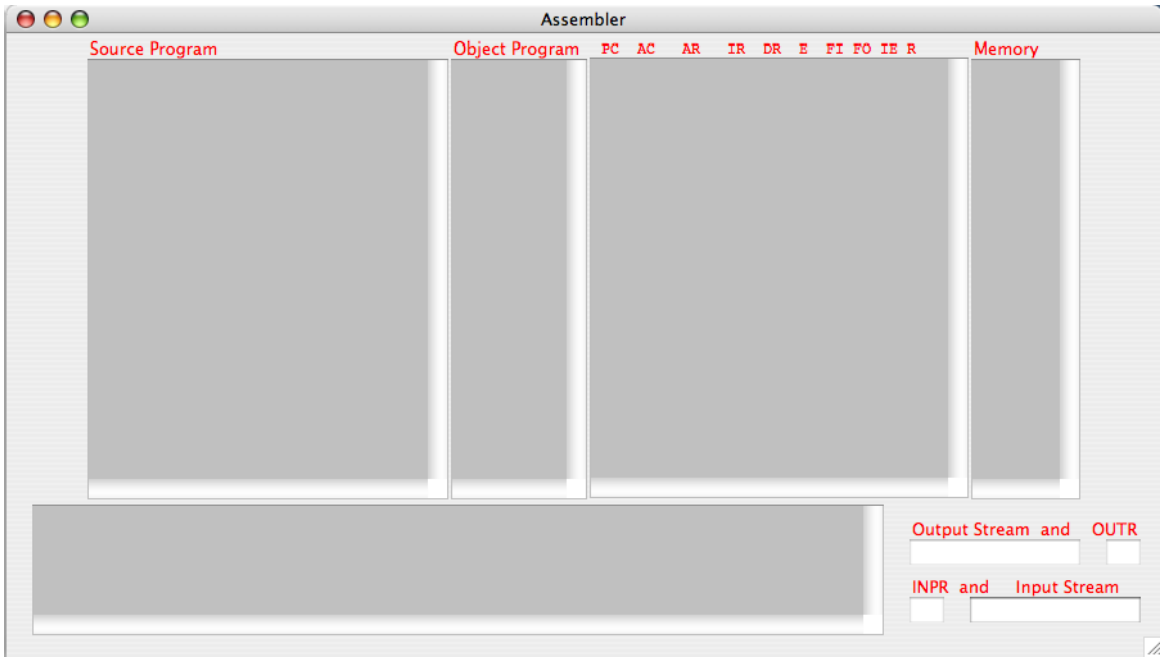
Appendix

Assembler Simulator for the Programming Assignment

Download the assembler simulator from the Canvas. The assembler simulator is written in Java, you need to have Java Runtime Environment (JRE) installed to run this program irrespective of the operating system. Follow the given instructions to run and use the simulator.

A1. To run the Simulator:

- Double click on the program (Assembler.jar).
- You will get the following interface to read, compile and run your code.



A2. To use the Simulator:

- **Read your source program:**
  1. From the menu: File -> Open -> Select your Program (*must be .txt file*)

     or
  2. Simply copy and paste your program to the text box under 'Source Program'

- **Compile your program:**

  From the menu: Tools -> Compile

- **Run your program:**

  From the menu: Tools -> Execute -> Run or Walk (step by step). Values of registers and memory will be shown.

- **Inputs/Outputs:**

  If your program uses INPR to read inputs, you can type the inputs in the "input Stream" text box. Outputs from OUTR will be shown in Output Stream.

A3. Review of Instructions:

There are three groups of instructions in this assembler:

1. Memory Reference Instructions
2. Non memory Reference Instructions
3. Pseudo Instructions (i.e., Assembler directive instructions)

A3.1 Memory Reference Instructions (MRI)

Direct Addressing: opcode operand e.g., ADD num
Memory word at location 'num' is added to accumulator AC. i.e.,
AC = AC + M[num];
Here, effective address of the operand is 'num'

Indirect Addressing: opcode operand I e.g., ADD num I
Memory word of memory word at location 'num' is added to AC. i.e.,
AC = AC + [M[num]]
Here, effective address of the operand is M[num].

MRI Instructions: (In the following, 'eee' denotes effective address.) AND

xxx AND xxx I
Logical AND of effective memory word to AC i.e.,
AC = AC and M[eee];

ADD xxx ADD xxx I
Add effective memory word to AC.
i.e., AC = AC + M[eee];

LDA xxx LDA xxx I
Load effective memory word to AC.
i.e., AC = M[eee];

STA xxx STA xxx I
Store content of AC to effective memory word. i.e.,
M[eee] = AC;

BUN xxx BUN xxx I
Branch, unconditionally, to effective address. i.e.,
PC = eee;

BSA xxx BSA xxx I
Address of next instruction (i.e., PC) is stored in effective memory word. Then, execute the instruction following the effective address.

i.e., M[eee] = PC; PC = eee + 1;
Note: BSA is useful to save the return address and to branch to a procedure.

ISZ xxx ISZ xxx I
Increment memory word. If incremented value is 0, increment PC (i.e., skip next instr). i.e., M[eee] = M[eee] + 1; if (M[eee] == 0) PC = PC + 1;
Note: ISZ is used to count iterative loops.

A3.2 Non Memory Reference Instructions

These instructions do not have the operand part or the addressing mode. CLA

Clear AC

CLE Clear E, the extended bit of AC

CMA Complement AC

CME Complement E

CIR Circular shift to the Right on AC and E

CIL Circular shift to the Left on AC and E INC

Increment AC

SPA Skip next instruction, if AC is Positive, i.e., if (AC > 0) PC = PC + 1; SNA

Skip next instruction, if AC is Negative, i.e., if (AC < 0) PC = PC + 1;

SZA Skip next instruction, if AC is Zero, i.e., if (AC == 0) PC = PC + 1;
(Note: SPA, SNA, and SZA are used in conditional branching.)

SZE Skip next instruction, if E is Zero, i.e., if (E == 0) PC = PC + 1; HLT

Halt the execution

INP Input a character from INPR to low-order bits of AC OUT

Output a character from low-order bits of AC to OUTR SKI

Skip on Input flag

A3.3 Pseudo Instructions

ORG hhh	Instruction listed in the following line wll be placed at address 'hhh' (Hex) DEC
n	Decimal number 'n' will be placed in the memory word
HEX n	Hexadecimal number 'n' will be placed in the memory word
END	Denotes the end of assembly language source program

Reference
Sections 5.3, 5.5, 5.6, 5.7, 6.3 of
Computer System Architecture (3e) by M. Morr