# CS 2614: Computer Organization

# Assembly Programming Project

**Spring 2024**

**<u>NOTE</u>:** This is an individual assignment. You must do it individually and **not in** Groups.

The UNIVERSITY of OKLAHOMA
Gallogly College of Engineering
School of Computer Science

# CS 2614 Computer Organization
## Assembly Programming Project
### Spring 2024

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 the class version of assembly language program to code and test the given problem.

**Problem Statement:**

Given an **octal number *n* as user input**, write an assembly language program to convert ***n* to decimal number** and **display it if the number is a multiple of 3 or 5**. The program should display "0" otherwise.

*n* (user input) will be exactly a **two-digit octal number**, not more and not fewer. An example, take the input *n* (user input) as $\mathbf{10_8}$,your program will first convert $\mathbf{10_8}$ to decimal equivalent "$\mathbf{8_{10}}$", and then your program should output "0" since it is not a multiple of 3 or 5. Another example, if *n* (user input) is $\mathbf{43_8}$, your program will first convert $\mathbf{43_8}$ to decimal equivalent "$\mathbf{35_{10}}$", and then your program should output "35" since it is a multiple of 5.

You need to take the input characters using INP in Assembler, convert the input characters into an octal number, convert the octal number into a decimal number, have loop(s) to check the factors. You **should not** hardcode the input/output numbers and **should not** give the direct formula to get the final result. For example, a code such as this: `if (n==43) then printf("35");` is not allowed.

Below is the ASCII table that shows the conversion from character to octal and decimal.

| Dec | Hx | Oct | Char | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr | Dec | Hx | Oct | Html | Chr |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 000 | NUL (null) | 32 | 20 | 040 |   | Space | 64 | 40 | 100 | @ | @ | 96 | 60 | 140 | ` | ` |
| 1 | 1 | 001 | SOH (start of heading) | 33 | 21 | 041 | ! | ! | 65 | 41 | 101 | A | A | 97 | 61 | 141 | a | a |
| 2 | 2 | 002 | STX (start of text) | 34 | 22 | 042 | " | " | 66 | 42 | 102 | B | B | 98 | 62 | 142 | b | b |
| 3 | 3 | 003 | ETX (end of text) | 35 | 23 | 043 | # | # | 67 | 43 | 103 | C | C | 99 | 63 | 143 | c | c |
| 4 | 4 | 004 | EOT (end of transmission) | 36 | 24 | 044 | $ | $ | 68 | 44 | 104 | D | D | 100 | 64 | 144 | d | d |
| 5 | 5 | 005 | ENQ (enquiry) | 37 | 25 | 045 | % | % | 69 | 45 | 105 | E | E | 101 | 65 | 145 | e | e |
| 6 | 6 | 006 | ACK (acknowledge) | 38 | 26 | 046 | & | & | 70 | 46 | 106 | F | F | 102 | 66 | 146 | f | f |
| 7 | 7 | 007 | BEL (bell) | 39 | 27 | 047 | ' | ' | 71 | 47 | 107 | G | G | 103 | 67 | 147 | g | g |
| 8 | 8 | 010 | BS (backspace) | 40 | 28 | 050 | ( | ( | 72 | 48 | 110 | H | H | 104 | 68 | 150 | h | h |
| 9 | 9 | 011 | TAB (horizontal tab) | 41 | 29 | 051 | ) | ) | 73 | 49 | 111 | I | I | 105 | 69 | 151 | i | i |
| 10 | A | 012 | LF (NL line feed, new line) | 42 | 2A | 052 | * | * | 74 | 4A | 112 | J | J | 106 | 6A | 152 | j | j |
| 11 | B | 013 | VT (vertical tab) | 43 | 2B | 053 | + | + | 75 | 4B | 113 | K | K | 107 | 6B | 153 | k | k |
| 12 | C | 014 | FF (NP form feed, new page) | 44 | 2C | 054 | , | , | 76 | 4C | 114 | L | L | 108 | 6C | 154 | l | l |
| 13 | D | 015 | CR (carriage return) | 45 | 2D | 055 | - | - | 77 | 4D | 115 | M | M | 109 | 6D | 155 | m | m |
| 14 | E | 016 | SO (shift out) | 46 | 2E | 056 | . | . | 78 | 4E | 116 | N | N | 110 | 6E | 156 | n | n |
| 15 | F | 017 | SI (shift in) | 47 | 2F | 057 | / | / | 79 | 4F | 117 | O | O | 111 | 6F | 157 | o | o |
| 16 | 10 | 020 | DLE (data link escape) | 48 | 30 | 060 | 0 | 0 | 80 | 50 | 120 | P | P | 112 | 70 | 160 | p | p |
| 17 | 11 | 021 | DC1 (device control 1) | 49 | 31 | 061 | 1 | 1 | 81 | 51 | 121 | Q | Q | 113 | 71 | 161 | q | q |
| 18 | 12 | 022 | DC2 (device control 2) | 50 | 32 | 062 | 2 | 2 | 82 | 52 | 122 | R | R | 114 | 72 | 162 | r | r |
| 19 | 13 | 023 | DC3 (device control 3) | 51 | 33 | 063 | 3 | 3 | 83 | 53 | 123 | S | S | 115 | 73 | 163 | s | s |
| 20 | 14 | 024 | DC4 (device control 4) | 52 | 34 | 064 | 4 | 4 | 84 | 54 | 124 | T | T | 116 | 74 | 164 | t | t |
| 21 | 15 | 025 | NAK (negative acknowledge) | 53 | 35 | 065 | 5 | 5 | 85 | 55 | 125 | U | U | 117 | 75 | 165 | u | u |
| 22 | 16 | 026 | SYN (synchronous idle) | 54 | 36 | 066 | 6 | 6 | 86 | 56 | 126 | V | V | 118 | 76 | 166 | v | v |
| 23 | 17 | 027 | ETB (end of trans. block) | 55 | 37 | 067 | 7 | 7 | 87 | 57 | 127 | W | W | 119 | 77 | 167 | w | w |
| 24 | 18 | 030 | CAN (cancel) | 56 | 38 | 070 | 8 | 8 | 88 | 58 | 130 | X | X | 120 | 78 | 170 | x | x |
| 25 | 19 | 031 | EM (end of medium) | 57 | 39 | 071 | 9 | 9 | 89 | 59 | 131 | Y | Y | 121 | 79 | 171 | y | y |
| 26 | 1A | 032 | SUB (substitute) | 58 | 3A | 072 | : | : | 90 | 5A | 132 | Z | Z | 122 | 7A | 172 | z | z |
| 27 | 1B | 033 | ESC (escape) | 59 | 3B | 073 | ; | ; | 91 | 5B | 133 | [ | [ | 123 | 7B | 173 | { | { |
| 28 | 1C | 034 | FS (file separator) | 60 | 3C | 074 | < | < | 92 | 5C | 134 | \ | \ | 124 | 7C | 174 | | | \| |
| 29 | 1D | 035 | GS (group separator) | 61 | 3D | 075 | = | = | 93 | 5D | 135 | ] | ] | 125 | 7D | 175 | } | } |
| 30 | 1E | 036 | RS (record separator) | 62 | 3E | 076 | > | > | 94 | 5E | 136 | ^ | ^ | 126 | 7E | 176 | ~ | ~ |
| 31 | 1F | 037 | US (unit separator) | 63 | 3F | 077 | ? | ? | 95 | 5F | 137 | _ | _ | 127 | 7F | 177 | &#127; | DEL |

Source: www.LookupTables.com

*Hint:*

The following pseudo-code demonstrates the logic for checking the multiple of 3 or 5 from number n, where n is the given input. Please note that this program utilizes low-level functions not readily available in the assembly language, such as: conversion of ASCII-hexadecimal value to numeric-octal value, integer division, and modulus operation.

```
int octal;             //the input number in octal
int decimal = 0;   //the octal input number converted into decimal
int i = 0;            //iterator variable
int remainder;     //remainder number after modulus operation

// Input octal number
printf("Enter an octal number: ");
scanf("%d", &octal);

// Convert octal to decimal
while (octal != 0) {
   remainder = octal % 10;
   decimal += remainder * pow(8, i);
   ++i;
   octal /= 10;
}

// Check if decimal is a multiple of 3 or 5
if (decimal % 3 == 0 || decimal % 5 == 0) {
   printf("%d\n", decimal);
}
else {printf("0");};
```

The integer division ('/') and modulus operation ('%') can be implemented using the following pseudo-code.

```
while (dividend >= divisor) {
   dividend -= divisor;
   quotient++;
}
printf("Quotient: %d\n", quotient);
printf("Remainder: %d\n", dividend);
```

Please also note that these are just hints. You are free to use any algorithms as long as they produce the correct results, and the results are not hard-coded.

**Project Description:**

1. *Part A: Design document* ***(at least one page)***

The design document should describe, using flow charts or C-like pseudo-codes for each subroutine,how you will implement the program using the assembly language. Submit a typed description of the problem-solving approaches you will use to solve the given problem. Use words and be descriptive!
– What I am looking for here is that you know how the assembler works and that you are comfortable working with it. **It would be best if you wrote an explanation for each item in the grading criteria below**.

Questions you should address:

- How to achieve looping in assembly language? (Hints: assembly language has no built-in *while* loop and *for* loop)
- Specifically, what are the loop conditions (initial values, end values, increments in each step, stopping conditions)?

These questions are only a general guideline. The objective is to help you get started working on the project and be able to **dissect the problem** into smaller procedures (like all other programming problems). The reasonable page length is 2.

***Grading criteria for the Design Document*** *(Worth 20% of the project)*

| Critical Elements | Percentage Distribution |
|---|---|
| The use of the assembly instructions in the problem-solving approaches | 35% |
| How to convert input characters to decimal | 20% |
| Loop conditions for checking the multiples | 20% |
| Articulation of response such as free of errors, grammar, syntax, and organization | 25% |
| **Total** | **100%** |

2. *Part B: Assembly Program*

Submit an assembly program to solve the given problem. To get started, you need to download and run the assembler simulator (Assembler.jar) from Canvas. The instructions to download, run the simulator then compile and run the code are specified in the "*Helpful Resource*" section below. Your program should be stored **in a plain text file** and able to be executed on the simulator.

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

| Critical Elements | Percentage Distribution |
|---|---|
| Well Commented Code | 20% |
| Variable initialization | 20% |
| Get input *N* from user | 20% |
| Converting N to equivalent decimal | 20% |
| Correctness of output in showing the multiple of 3 or 5 | 20% |
| **Total** | **100%** |

**Submission guidelines:**

a) Design Document: **.pdf** or **.docx** (**due Saturday, April 6, 2024, on Canvas**)
b) Assembly program: **.txt** only (**due Friday, April 19, 2024, on Canvas**)

Both submissions should be placed in the appropriate file uploads on Canvas by the deadline. **Please check your submission**. If we cannot open it, then you will get a penalty for any resubmission lateness. You will demonstrate your programs to the TAs from April 29 to May 3, 2024 (Appointment slots will be posted on Canvas for you to reserve your project demonstration schedule).

**Note: Again, in the project you must use loop functionality to achieve the output. Pre-defined values or answers in the code are NOT accepted. Also, any plagiarism WILL BE REPORTED to OU Academic Integrity office.**

**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.**

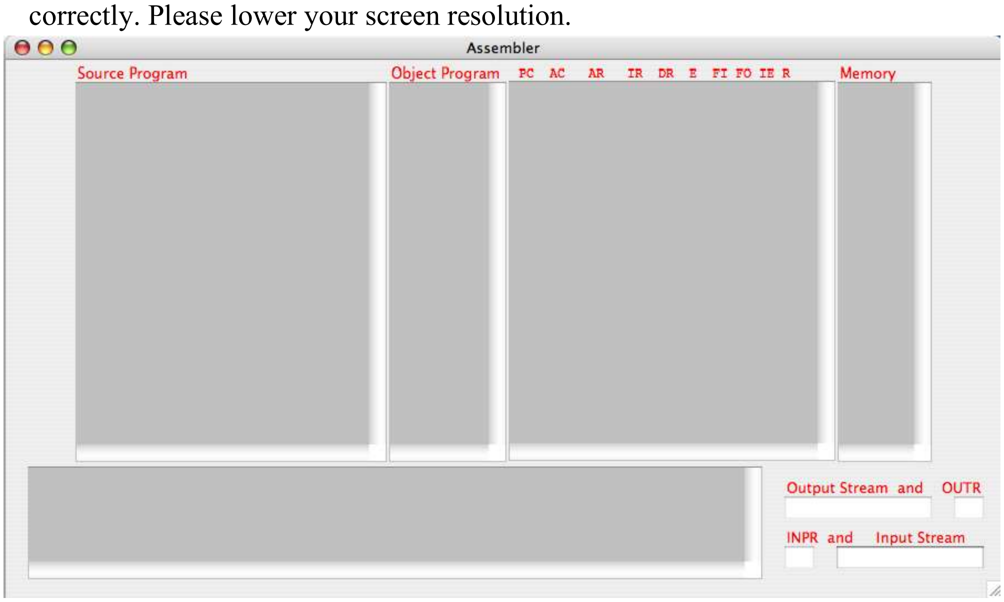
# Helpful Resources

### 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**. Follow thegiven 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.
- If your screen resolution is too high, the simulator's interface will not show all views correctly. Please lower your screen resolution.



## *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"
  3. One demo program is available with the simulator (File-> Demo Source File)

- **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 INP to read inputs, you can type the inputs in the"input Stream" text box. Outputs from OUT will be shown in Output Stream.

**<u>NOTE:</u> ALWAYS TRY TO COMPILE THE CODE, GIVE INP TO READ INPUTS AND THEN RUN YOUR PROGRAM.**

## *A3. Instructions Review:*

There are three groups of instructions in this assembler:

- Memory Reference Instructions
- Non memory Reference Instructions
- Pseudo Instructions (i.e., Assembler directive instructions)

***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, "addr" denotes effective address.)**

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

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

Load effective memory word to AC.
i.e., AC = M[addr];

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

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

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[addr] = PC; PC = addr + 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 instruction).
i.e., M[addr] = M[addr] + 1; if (M[addr] == 0) PC = PC + 1;
Note: ISZ is used to count iterative loops.

***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(15) = 0) PC = PC + 1;

SNA  Skip next instruction, if AC is Negative, i.e., if (AC(15) = 1) 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 output stream

SKI Skip on Input flag

***Pseudo Instructions***

| | |
|---|---|
| ORG hhh | Instruction listed in the following line will be placed at address 'hhh' (Hex) DECn |
| | 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 |

***Instructions Table***

| Symbol | Hexadecimal code |
|---|---|
| AND | 0 or 8 |
| ADD | 1 or 9 |
| LDA | 2 or A |
| STA | 3 or B |
| BUN | 4 or C |
| BSA | 5 or D |
| ISZ | 6 or E |
| CLA | 7800 |
| CLE | 7400 |
| CMA | 7200 |
| CME | 7100 |
| CIR | 7080 |
| CIL | 7040 |
| INC | 7020 |
| SPA | 7010 |
| SNA | 7008 |
| SZA | 7004 |
| SZE | 7002 |
| HLT | 7001 |
| INP | F800 |
| OUT | F400 |
| SKI | F200 |
| SKO | F100 |
| ION | F080 |
| IOF | F040 |

**Book Reference**

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