CSE 102 Spring 2024 – Computer Programming Assignment 10

Due on May 17, 2024 at 23:59

For this assignment, you will be creating an assembly language execution simulator. The program will display the output of the assembly language code file to the user. The instruction set for this task are provided in the *"instructions.txt"* file, do not change the format of it. Each operation in the instruction set has an opcode, type, format, and explanation and the instruction set consists of 30 operation in total. In addition to this, explanations of each instruction are provided at the end of the homework.
You need to use struct operations for instruction handling and minimize the use of conditional statements by designing an algorithm to efficiently handle all given operations.

Part I. [20pts] Creating the Structure of the Simulation

Your program should begin by prompting the user to input the name of the assembly code file, which must have a ".asm" file extension. After processing the file, the program should display the output of the operation to the user. Ensure correctness of user input and maintain the program running until the user chooses to exit which is done by typing '0'.

The necessities of the assembly simulator are explained below:

- There are 2 registers in total, denoted as A and B. They should be implemented as char arrays capable of storing 8 bits each.
- A Program Counter (PC) which contains the line number of the assembly code and tracks the program's current position. It should be able to store 16 bits. It should be incremented to point the next instruction as the program runs.
- An Instruction Register (IR) is required to hold the current instruction, with each instruction being 24 bits long. Each instruction can be converted into binary form as given:
    - Place the opcode of the operation at the beginning of the line.
    - Convert each number into binary form.
    - Represent registers with its memory address.

3 - A	10000101

Current Instruction: ADDI A 2
Instruction Register: 00000010 10000101 00000010
(**Hint.** Spaces need to be ignored, they are placed for your convenience. Therefore, the actual value inside the IR is 000000101000010100000010)

- Memory should be structured as a stack, with each memory location (called word) being 8 bits long. Memory addresses can correspond to line numbers (1, 2, 3, 4, etc.). The upper portion of memory should be reserved for storing and reading an assembly program, while the lower portion can be allocated for variables and other operations. The size of the memory should be enough to store all this information.

Part II. [10pts] Convert Decimal Numbers into Binary and Binary into Decimal

Write a function named `int to_binary(int decimal_number)` to convert decimal numbers into a binary form. This function has to take an integer number as an input and return its binary representation. All the numbers must be represented as 8 bits.

Write a function named `int to_decimal(char binary_number[])` to convert binary numbers into a decimal form. This function has to take an 8 bits long char number array as an input and return its decimal representation.

Part III. [70pts] Run Simulation

The program flow cycle is explained and an example is given below:

1. **Fetching instruction:** Each instruction should be read line by line from the memory by using PC and putted into IR as a format that explained above.
2. **Decode instruction:** The instruction needs to be understood by the program and this cycle includes the determination of what operation needs to be performed such as identifying opcode and registers that used.
3. **Execute instruction:** The operations specified by the instruction needs to be done by the simulator.
4. **Store result:** The result needs to be stored into memory.

Input File:

```
1  LDAI A 100
2  LDA A M[A]
3  LDAI B 101
4  LDA B M[B]
5  SHR A 2
6  SHL B 2
7  ADD A B
8  PRI A
9  EXIT
```

Terminal:

```
Enter the filename: example_code.asm
Result of the execution: 25
Do you want to execute another assembly code: Press (1) for yes, Press (0) for exit: 0
```

The Memory in the Beginning and End:

	Memory - Beginning
1	00011101
2	01100110
3	01100100
4	00010101
5	01100110
6	00000101
7	00011101
8	01100111
9	01100101
10	00010101
11	01100111
12	00000110
13	00010000
14	01100110
15	00000010
16	00001111
17	01100111
18	00000010
19	00000001
20	01100110
21	01100111
22	00011010
23	01100110
24	00000000
25	00011101
26	00000000
27	00000000
…	…
100	00000101
101	00000110
102-A	00000000
103-B	00000000
104-PC	00000000
105-PC	00000001
106-IR	00010101
107-IR	01100110
108-IR	01100100
…	…

	Memory - End
1	00011101
2	01100110
3	01100100
4	00010101
5	01100110
6	00000101
7	00011101
8	01100111
9	01100101
10	00010101
11	01100111
12	00000110
13	00010000
14	01100110
15	00000010
16	00001111
17	01100111
18	00000010
19	00000001
20	01100110
21	01100111
22	00011010
23	01100110
24	00000000
25	00011101
26	00000000
27	00000000
…	…
100	00000101
101	00000110
102-A	00011001
103-B	00011000
104-PC	00000000
105-PC	00011100
106-IR	00011101
107-IR	00000000
108-IR	00000000
…	…

Opcode	Type	Format	Explanation
00000001	G	ADD A B	It adds A and B register values then stores the result in A
00000010	T	ADDI A I	It adds register A and given number then stores the result in A.
00000011	U	ADDM A M[B]	It adds register A and a number that stored in memory address B then stores the result in A.
00000100	G	AND A B	It ands A and B register values then stores the result in A.
00000101	T	ANDI A I	It ands register A and given number then stores the result in A.
00000110	U	ANDM A M[B]	It ands register A and a number that stored in memory address B then stores the result in A.
00000111	G	OR A B	It does or operation between register A and B, then stores the result in A.
00001000	T	ORI A I	It does or operation between register A and given number, then stores the result in A.
00001001	U	ORM A M[B]	It does or operation between register A and the value inside the memory address B, then stores the result in A.
00001010	G	SUB A B	It does subtraction operation between A and B registers, then stores the result in A.
00001011	T	SUBI A I	It does subtraction operation between register A and given number, then stores the result in A.
00001100	U	SUBM A M[B]	It does subtraction operation between register A and the value inside the memory address B, then stores the result in A.
00001101	T	INC A	Increment A register by 1.
00001110	T	DEC A	Decrement B register by 1.
00001111	T	SHL A I	Shift register A value to the left I times.
00010000	T	SHR A I	Shift register A value to the rigth I times.
00010001	G	BRE A B	Branch on line A if A and B equal.
00010010	G	BRN A B	Branch on line A if A and B not equal.
00010011	G	J A B	Jump to specific line number which is A + B.
00010100	G	NOR A B	It does nor operation between register A and B, then stores the result in A.
00010101	U	LDA A M[B]	It loads the value from memory address B into register A.
00010110	U	STR A M[B]	It stores the value to memory address B from register A.
00010111	G	SWP A B	It swaps register A and B.
00011000	G	LESS A B	It checks whether A is less than B.
00011001	G	GRT A B	It check whether A is greater than B.
00011010	G	PRI A	It prints the value of register A. The value need to be converted to decimal number.
00011011	T	PRII I	It prints the value of I. The value need to be converted to decimal number.
00011100	U	PRIM M[A]	It prints the value from memory address A. The value need to be converted to decimal number.
00011101	T	LDAI A I	Load I value to A register
00011110	-	EXIT	It ends the program.

IMPORTANT NOTES:

- Submit your homework as a **zip file** named as your student id (StudentID.zip) and this file should include:
  - YourStudentID.c file
  - YourStudentID.pdf file that includes screenshots of the input files and terminal.
- Do not use any library other than stdio.h and string.h.
- The output format must be as given, do not change it.
- Compile your work with given command “gcc --ansi your_program.c -o your_program”.
- A demo session for this homework will be held after the deadline. The timetable for the demo sessions will be shared with you via Teams on **May 19th.** You will have **two days** to fill in your preferred time slots. **No excuses will be accepted after the deadline for selecting time slots.**
- For any questions and problems, you can always contact me **via email** (incikaramahmutoglu@gtu.edu.tr), or you can find me in Human Computer Interaction Lab. (No: 120) during scheduled office hours on May 7, 2024 and May 14, 2024 between 13:30 and 14:30.