COSC 420- Computer Architecture
Spring semester 2024
Project 2 – CPU Simulator – can be submitted at any time until 04/26/2024.

The CPU has 8-bit op codes and 16-bit addresses. Instructions are either 8 bits, 16 bits, or 24 bits in length.

Registers

There are 3 registers directly controlled by the programmer:

AC: 8-bit accumulator. AC receives the result of arithmetic and logic instructions. It provides one of the operands for two-operand arithmetic and logic instructions. Data is loaded to/from AC from/to memory.
R: 8-bit general purpose register which supplies the 2nd operand of all two-operand arithmetic and logic instructions. It can also be used as a temporary storage area.
Flag: 4-bit register containing the following status flags (set/reset as a result of the current operation):

Z: Zero Flag: This bit is set/reset as a result of arithmetic or logic operations. If the result of an arithmetic/logic operation is zero, then Z is set to . If the result of an arithmetic/logic operation is not zero, then Z is set to 0.
The Zero Flag is set/reset as a result of ADD, SUB, INAC, AND,OR, XOR, and NOT.

C: Carry Flag: This bit reflects the result of the carry out of the most significant bit of the result of the last executed arithmetic operation. In other words, if data is treated as unsigned, it represents the result too large in the case of add or a borrow in the case of subtract.
The Carry Flag is set/reset as a result of ADD, INAC and SUB.

V: Overflow Flag: This bit is set if the carry into the most significant bit of the result does not equal the carry out of the most significant bit of the result of the last executed arithmetic operation. It is reset if the carry into of the most significant bit equals the carry out of the most significant bit of the result of the last arithmetic operation. This is like the carry except for signed data, the addition of two positive numbers results in a negative number.
The Overflow flag is set/reset as a result of ADD, INAC, and SUB.

N: Negative Flag: This bit is set if the result of the operation is negative - the msb of the result is set.
The Negative Flag is set/reset as a result of AND, OR, XOR, NOT, ADD, INAC, and SUB.

You **Must** use the following definitions (or equivalent constant definitions) for the bits in the flag register:

```
#define Z 1 (least significant bit of the register)
#define C 2
#define V 4
#define N 8 (most significant bit of the register)
```

As the result of operations, you will be “oring” the appropriate flag bits on and/or “anding” the appropriate flag bits off.

The CPU also contains several registers in addition to those specified in the instruction set. The following registers are used control the fetch/execute of the instructions:

- **AR**16-bit address register which supplies an address to memory
- **PC**16-bit Program Counter which contains the address of the next instruction to be executed or the address of the next operand of the current instruction.
- **DR**8-bit data register which receives instructions and data from memory and transfers data to memory.
- **IR**8-bit instruction register which contains the opcode fetched from memory to be executed.
- **TR**8-bit temporary register which holds data during instruction execution.

Instruction Set

An Op Code followed by the letter A means that an address follows the Opcode in the program. M[A] means memory addressed by A.

Instruction Name	**Op Code**	**Operation**
NOP	0000 0000	No Operation
LDAC	0000 0001 A	AC<-M[A]
STAC	0000 0010 A	M[A]<-AC
MVAC	0000 0011	R<-AC
MOVR	0000 0100	AC<-R
ADD	0000 1000	AC<-AC + R, ZCVN set/reset
SUB	0000 1001	AC<-AC - R, ZCVN set/reset
INAC	0000 1010	AC<-AC + 1, ZCVN set/reset
CLAC	0000 1011	AC<-0, Z<-1 and CNV reset
AND	0000 1100	AC<-AC bitwise AND R, ZN set/reset
OR	0000 1101	AC<-AC bitwise OR R, ZN set/reset
XOR	0000 1110	AC<-AC bitwise XOR R, ZN set/reset
NOT	0000 1111	AC<bitwise NOT (AC), ZN set/reset
MVI	0001 0110 D	AC<-D AC loaded with 8 bits following instruction

To Do

Using appropriate data structures to represent memory and **ALL** registers, write a simulator to simulate the CPU. **Take Note,** the 16- bit address register AR implies memory size of 65536 bytes.

Your program must:

1. Output your name and a description of the program.
2. Prompt the user for the name of a file containing the program.
3. Echo the name of the file.
4. Read the program into memory and then execute it from memory. Always load the program at M[0]. Initialize PC to 0 and begin executing the program.

The data in the file will be characters making up a hexadecimal representation of the program. Each line will contain a byte. A file containing:

```
00
0b
0a
02
20
00
00
ff
```

is a 6 instruction program :

```
NOP
Clear Accumulator
Increment Accumulator
Store value in Accumulator at address 2000
NOP
Halt
```

Your program must print out an initial value of :

```
Flag
AR
PC
DR
IR
TR
AC
R
```

Fetch/Decode

The fetch cycle of the CPU consists of 3 sub-phases:

- Fetch1: AR<-PC
- Fetch2: DR<-M[AR], PC<-PC+1
- Fetch3: IR<-DR, AR<-PC

Fetch1, Fetch2, and Fetch3 must be done consecutively.

You must provide a fetch function.

At the completion of Fetch1, print out the label Fetch1 and the values of AR and PC. At the completion of Fetch2, print out a label Fetch2 and the values in DR and PC. At the completion of Fetch3, print out a label Fetch3 and the value of IR and AR. Be sure to identify the data printed.

In the Decode Cycle, determine the instruction contained in IR and call the appropriate function to execute the instruction. **You must provide a function for each instruction.**

Execute Cycle

Execution of the function of the instruction decoded.

Prior to returning, each instruction function must print the following:

```
AC
R
Flag
AR
PC
DR
```

Based on the instruction, you may also be required to print other information.

NOP Instruction
Return, no action

LDAC Instruction
Multiword instruction: Op Code, high-order half of address, and low-order half of the address. The execution function must first get the address from memory then the data from the memory location and load it into the accumulator.

Based on fetch, PC and AR point to the address - the high-order half of the address A.

The execution of the LDAC instruction is as follows:

- LDAC1: Read the high-order half of the address A into DR: DR<-M[AR]

  Increment PC: PC<-PC+1

  Increment AR: AR<-AR+1

  Print out values of DR, PC, and AR labeled with LDAC1

- LDAC2: obtain the low-order half of the address A, first you must save contents of DR as you need it:
  TR<-DR
  DR<-M[AR]
  PC<-PC+1

  With the label LDAC2, print out TR, DR, AR, and PC.

- LDAC3: Form the address AR<-TR,DR

  With the label LDAC3, print out AR

- LDAC4: DR<-M[AR]

  With the label LDAC4, print out DR

- LDAC5: AC<-DR

  With the label LDAC5, print out AC

Remaining Instructions:

The LDAC is one of the more complex instructions. I have provided you with the details of the execution sub-phases of LDAC and exactly what to print out. Using LDAC as a guide, use the definition of the instruction given in the table above to determine the execution sub-phases for each instruction. At the completion of each sub-phase you must print out each register that is modified during the sub-phase.

For the **STAC Instruction**, prior to storing the data in memory, you must print the address where data is to be stored, the contents of memory location prior to the store, and print the same data after memory has been updated.

What to submit:

Your program must be written in C++. **Included are some test files along with their outputs. You may use these to verify that your program functions correctly by matching the provided outputs.** Programs that do not compile are not worth any points. **You must test your code to ensure that every instruction produces the correct result.**

This project was originally created by Dr. Barbara Bracken, modified by Dr. Mohamed Aturban.