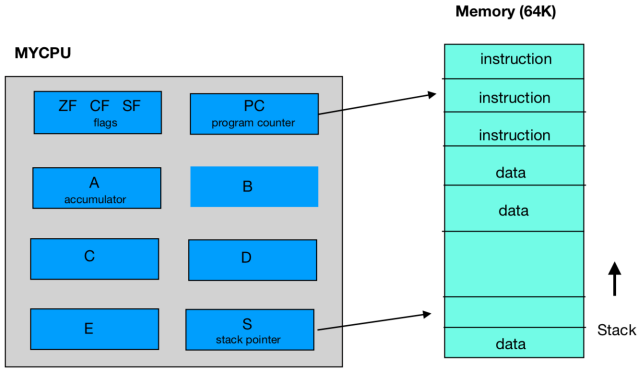
**CMPE 230 Systems Programming**

**Project** (due May 27th)

( This project can be implemented in groups of at most two students.)

This project is to be done with Python. In this project, you will implement (i) an assembler and (ii) an execution simulator for a hypothetical CPU called CPU230. CPU230 is illustrated in the following figure:




Each instruction has fixed length of 3 bytes with the following format:

| Opcode | Adressing mode | Operand |
|---|---|---|
| 6 bits | 2 bits | 16 bits |

Addressing mode bits are as follows:

| Bits(binary) | Addressing mode |
|---|---|
| 00 | operand is immediate data |
| 01 | operand is in given in the register |
| 10 | operand's memory address is given in the register |
| 11 | operand is a memory address |
| Note that registers are represented as bit patterns (here given in hex): PC=0000, A=0001, B=0002, C=0003, D=0004, E=0005, S=0006. | |

Instructions are as follows:

| Instruction | Instruction code (hex) | Operand | Meaning | Flags set |
|---|---|---|---|---|
| HALT | 1 | | Halts the CPU. | |
| LOAD | 2 | immediate<br>memory<br>register | Loads operand onto A . | |
| STORE | 3 | memory<br>register | Stores value in A to the operand. | |
| ADD | 4 | immediate<br>memory<br>register | adds operand to A. | CF,SF, ZF |
| SUB | 5 | immediate<br>memory<br>register | subtracts operand from A. | CF,SF, ZF |
| INC | 6 | immediate | increments operand (equivalent to add 1) | SF, ZF, CF |

| | | | | |
|---|---|---|---|---|
| | | memory<br>register | | |
| DEC | 7 | immediate<br>memory<br>register | decrements operand (equivalent to sub 1) | SF, ZF, CF |
| XOR | 8 | immediate<br>memory<br>register | Bitwise XOR operand with A and store result in A. | SF, ZF |
| AND | 9 | immediate<br>memory<br>register | Bitwise AND operand with A and store result in A. | SF, ZF |
| OR | A | immediate<br>memory<br>register | Bitwise OR operand with A and store result in A. | SF, ZF |
| NOT | B | immediate<br>memory<br>register | Take complement of the bits of the operand. | SF, ZF |
| SHL | C | register | Shift the bits of register one position to the left. | SF, ZF, CF |
| SHR | D | register | Shift the bits of register one position to the right. | SF, ZF |
| NOP | E | | No operation. | |
| PUSH | F | register | Push a word sized operand (two bytes) and update S by subtracting 2. | |
| POP | 10 | register | Pop a word sized data (two bytes) into the operand and update S by adding 2. | |
| CMP | 11 | immediate<br>memory<br>register | Perform comparison (AC-operand) and set flag accordingly. | SF, ZF, CF |
| JMP | 12 | immediate | Unconditional jump. Set PC to address. | |
| JZ<br>JE | 13 | immediate | Conditional jump. Jump to address (given as immediate operand) if zero flag is true. | |
| JNZ<br>JNE | 14 | immediate | Conditional jump. Jump to address (given as immediate operand) if zero flag is false. | |
| JC | 15 | immediate | Conditional jump. Jump if carry flag is true. | |
| JNC | 16 | immediate | Conditional jump. Jump if carry flag is false. | |
| JA | 17 | immediate | Conditional jump. Jump if carry flag is false. | |
| JAE | 18 | immediate | Conditional jump. Jump if above or equal. | |
| JB | 19 | immediate | Conditional jump. Jump if below. | |
| JBE | 1A | immediate | Conditional jump. Jump if below or equal. | |
| READ | 1B | memory<br>register | Reads a character into the operand. | |
| PRINT | 1C | immediate<br>memory<br>register | Prints the operand as a character. | |

Note that memory address can be given as **`[xxxx]`** or **`[r]`** where **`xxxx`** is a hexadecimal number or **`r`** where **`r`** is a register name.

Labels can also be used. A label: marks the address, **`xxxx`**, at the point it is defined. Wherever you use a label, you should substitute the marked address **`xxxx`** for the label.

The assembler you build will be called **`cpu230assemble`** and the execution simulator will be called **`cpu230exec`**. They will be used as follows. Suppose you are given a assembly program given in file prog.asm. The following command will assemble the program and produce the binary output prog.bin.

```
> cpu230assemble prog.asm
```

The following program will execute the binary

```
> cpu230exec prog.bin
```

The above process is illustrated in the example below:

| Assembly source code: `prog.asm` | Assemble | Assembled program : `prog.bin` | Execute | Output |
|---|---|---|---|---|
| `LOAD 'A'`<br>`STORE C`<br>`LOAD MYDATA`<br>`STORE B`<br>`LOAD 0004`<br>`STORE D`<br>`LOOP1:`<br>`PRINT C`<br>`LOAD C`<br>`STORE [B]`<br>`INC C`<br>`INC B`<br>`INC B`<br>`DEC D`<br>`JNZ LOOP1`<br>`HALT`<br>`MYDATA:` | `cpu230assemble prog.asm`<br>→ | `080041`<br>`0D0003`<br>`08002D`<br>`0D0002`<br>`080004`<br>`0D0004`<br>`710003`<br>`090003`<br>`0E0002`<br>`190003`<br>`190002`<br>`190002`<br>`1D0004`<br>`530012`<br>`040000` | `cpu230exec prog.bin`<br>→ | `A`<br>`B`<br>`C`<br>`D` |

Note also that in the above example, ascii codes of 'A', 'B', 'C', 'D' and 'E' are stored at the memory addresses `002D`, `002F`, `0031`, `0033`, `0035`.

**Grading**

Your project will be graded according to the following criteria:

| | |
|---|---|
| Documentation (written document describing how you implemented your project) | 12% |
| Comments in your code | 8% |
| Implementation and tests | 80% |

**Late Submission**

If the project is submitted late, the following penalties will be applied:

- 0 < hours late <= 24 : 25%
- 24 < hours late <= 48 : 50%
- hours late > 48 : 100%

**Timestamping**

Project file should include your names in it. Please timestamp your project file using https://opentimestamps.org/ before you submit it. Keep the project file and its corresponding timestamp .ots file.