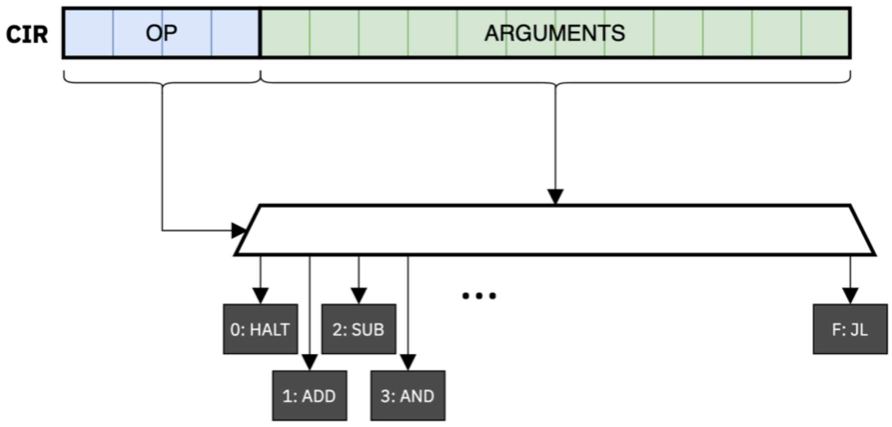
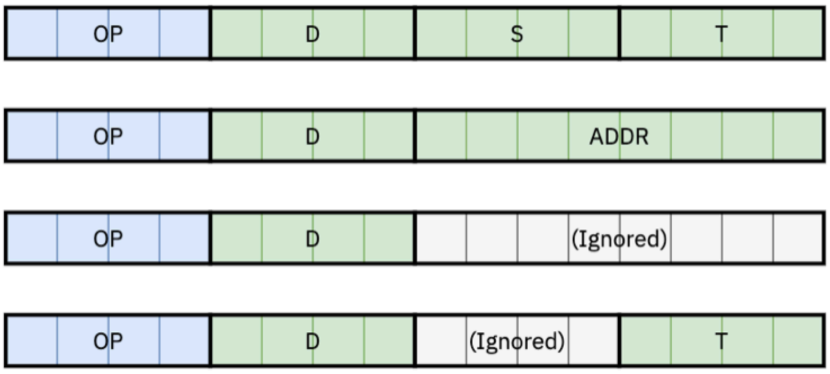
Fun with Toy Computer

Part 1: Programming in Machine Language

Toy only has 16 circuits in its ALU. A machine instruction consists of one word (two bytes or four nibbles). During the *decode* stage of the machine instruction cycle, the first nibble of the instruction is used as the *opcode* and the remaining three nibbles are used as the *arguments*. Toy uses a demultiplexor to determine which circuit to use: the opcode is fed to the selector and the arguments are fed to the input of the demultiplexor so that the arguments are sent to the appropriate circuit.



How the arguments are interpreted by the selected circuit depends on the circuit. Some circuits treat the arguments as three nibbles D (destination), S (source 1) and T (source 2), and some treat the arguments as a nibble D (destination) and a byte ADDR (address). Some circuits ignore parts of the arguments.



During the execute stage of the machine instruction cycle, the circuits have *side effects* (in other words, they change the state of the computer) that can be helpful for solving problems. (Indeed, computer programming is all about expressing problems in terms of side effects that create a solution.)

The side effects of the circuits in Toy can be summarized as follows:

```
0 Halt             -
1 Add              R[D] ← R[S] + R[T]
2 Subtract         R[D] ← R[S] - R[T]
3 Bitwise And      R[D] ← R[S] & R[T]
4 Bitwise Xor      R[D] ← R[S] ^ R[T]
5 Left Shift       R[D] ← R[S] << R[T]
6 Right Shift      R[D] ← R[S] >> R[T]
7 Load Address     R[D] ← ADDR
8 Load             R[D] ← M[ADDR]
9 Store            M[ADDR] ← R[D]
A Load Indirect    R[D] ← M[R[T]]
B Store Indirect   M[R[T]] ← R[D]
C Branch Zero      if R[D] = 0 PC ← ADDR
D Branch Positive  if R[D] > 0 PC ← ADDR
E Jump Register    PC ← R[D]
F Jump & Link      R[D] ← PC; PC ← ADDR
```

In the above pseudocode, R represents the array of the 16 general purpose registers in the CPU and M represents the array of the 256 registers in the RAM.

For interactivity, *Toy* has been adapted so that:

- If we try to load a value from `M[f0]` to a register (i.e. using circuit 7, 8 or A), Toy will wait for the user to input a value and then load that value to the register. If we try to load from M[fa], a random value will be loaded.

- If we try to store a value from a register to `M[f1]` to `M[f7]` (i.e. using circuit 9 or B), then the computer will output the value in binary (`f1`), octal (`f2`), hexadecimal (`f3`), denary (`f4`), as an ascii character (`f5`) as a new line (`f6`) or as a binary pattern (`f7`).

When we program in machine language, we write instructions as values that are then stored directly in the RAM of the computer.

Example

Say we would like to write a program that takes in two values from the user and then outputs the sum of the two values. Remember that memory address `f0` is a special address we can use to load user input, f4 is a special address we can use to store a value that will be output in denary, and `f5` is a special address we can use to output a new line.

We could break up the logic of the program into the following steps.

1. Get a value from the user and store it. Specifically, load a value from memory address `f0` and store the result to a register, say register `0`.
2. Get a second value from the user and store it to another register, say register 1.
3. Calculate the sum of the two values and store the result to another register, say register 2.
4. Output the value of the sum (in denary).
5. Output a new line.
6. End the program.

For steps 1 and 2, we could use circuit 8 (load), for step 3, we could use circuit 1 (add), for steps 4 and 5 we could use circuit 9 (store), and for step 6, we use the circuit `0` (halt).

Putting all this together, we get the program:

example.mc

```
PC: 00   ; set the address for the first instruction
00: 80f0 ; load user input to register 0
01: 81f0 ; load user input to register 1
02: 1201 ; load the sum of R[0] and R[1] to register 2
03: 92f4 ; output the value in register 2
04: 90f5 ; output a new line
05: 0000 ; halt the program
```

When we compile and run the program, an example run gives us the following (user input shown in bold).

```
10
15
25
```

Exercises

Problem 1

Write a program in Toy Machine Language that inputs a value from the user and then outputs the value in binary, octal, denary and hexadecimal, each on a new line.

Problem 2

Notice that Toy Computer does not have a machine instruction for multiplication. Write a machine code program that inputs two numbers from the user and outputs the product of the two numbers. (Hint: treat multiplication as repeated addition.)

Problem 3

What does this machine language program do?

```
PC: 00
00: 7001
01: 8af0
02: 7b00
03: 7c01
04: 9bf4
05: 9bf6
06: 1dbc
07: 7b00
08: 1bbc
09: 7c00
0A: 1ccd
0B: 2aa0
0C: da04
0D: 0000
```

Problem 4

(a) Write a program in machine language that inputs three values from the user. The first number represents a first term, the second number represents a common difference and the third number represents how many terms are desired. The program should then output that many terms of the arithmetic sequence thus described.

(b) Repeat part (a), but this time, instead of outputting the terms in the sequence, the program should store the terms in the computer's memory starting at address B0.

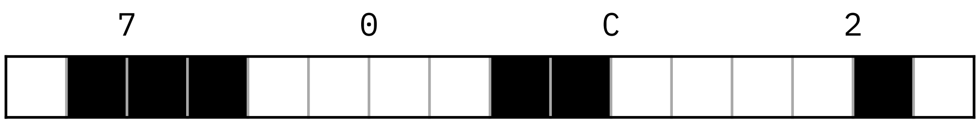
Problem 5

The following machine language program consists of data meant to be interpreted as ascii characters in addresses 00 to 0d, and data meant to be interpreted as program instructions in addresses 0e to 18. What does this program do?

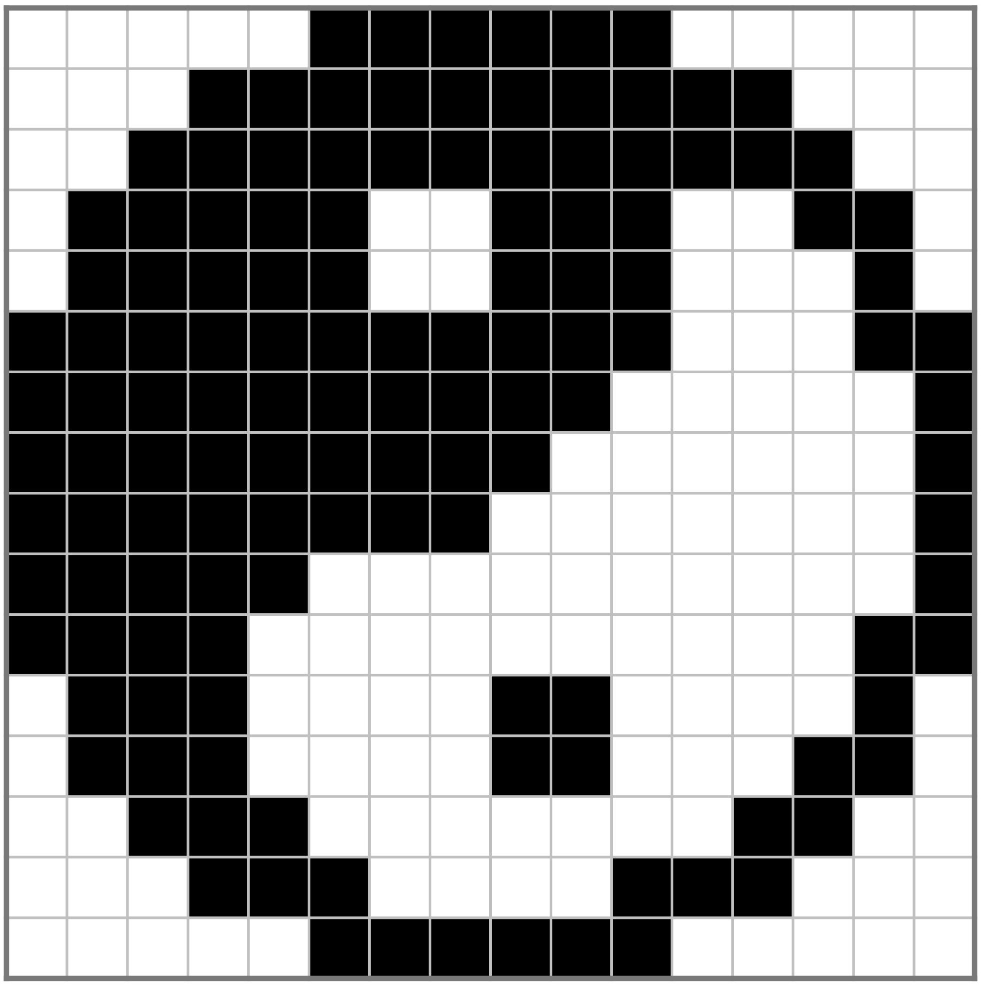
```
PC: 0e
00: 0048
01: 0065
02: 006c
03: 006c
04: 006f
05: 002c
06: 0020
07: 0077
08: 006f
09: 0072
0a: 006c
0b: 0064
0c: 0021
0d: 0000
0e: 7a00
0f: ab0a
10: cb16
11: 9bf5
12: 7f01
13: 1aaf
14: 7f0f
15: ef00
16: 90f6
17: 0000
```

Problem 6

If we store a value *v* to memory address 0xF7 of our toy computer, a pixel pattern based on the binary representation of v is output to the terminal screen. For example, if we store the value 0x70C2 to address 0xF7, then the following pattern will be output to the terminal:



(a) Write a program in machine language that outputs the following yin yang symbol to the terminal.



(b) Write a program in machine language that inputs values from the user until the user inputs zero, and then generates an image from the values input by the user similarly to how this was done in part (a).

(c) Use the program you wrote in part (b) to create the following images.