Fun with Toy Computer

Part 3: Project

Now that we have some experience with machine language and assembly programming for *Toy*, we're going to explore some programs in terms of the machine instructions they employ.

Select one of the programs presented in this lab and write a report that contains:

- A suitable heading.
- An introduction that explains what the program does, what the problem being explored is, and what the report contains.
- The full assembly language program.
- The lookup table created by the compiler that maps address labels to addresses.
- The data in the RAM when the program is first compiled (before the program is run).
- The output from an example run.
- A response to the problem that refers to excerpts from the assembly code and resulting compiled instructions / data as needed.

Your report will be peer assessed according to agreement with the following assertions.

- The report has a suitable heading.
- The report has an introduction that:
  - Introduces the program.
  - Explains what the problem being explored is.
  - Outlines the structure of the rest of the report.
- The whole assembly program is included in the report.
- The state of the computer, including all values in the RAM, after the program has been compiled (before the program has been run) is included in the report.
- The lookup table created during compilation, which maps assembly labels to memory addresses, is shown.
- An example run of the program is included in the report to support the description of the program.
- The problem has been effectively addressed using:
  - Relevant excerpts of the assembly script.
  - Relevant machine code instructions.
  - Identification of the machine instructions involved.
- The report has a good structure including consistent formatting of headings, subheadings, code and text.

Problem 1: Comparing Two Numbers

Guess is a game in which we try to guess the number the computer is thinking of. After each guess, the computer tells us whether we have guessed too high or too low.

Example Run

```
Welcome to guess!
I'm thinking of a number from 0 to 255.
Try to guess it!
Guess number: 1
What is your guess? 100
Too high!
Guess number: 2
What is your guess? 50
Too high!
Guess number: 3
What is your guess? 25
Too high!
Guess number: 4
What is your guess? 10
Too high!
Guess number: 5
What is your guess? 5
Too high!
Guess number: 6
What is your guess? 2
Too low!
Guess number: 7
What is your guess? 3
Too low!
Guess number: 8
What is your guess? 4
You got it in 8 moves! Great job!
```

Assembly

```
    intro_1: .ascii "Welcome to guess!"
    intro_2: .ascii "I'm thinking of a number from 0 to 255."
    intro_3: .ascii "Try to guess it!"
      guess: .ascii "Guess number: "
     prompt: .ascii "What is your guess? "
   too_high: .ascii "Too high!"
    too_low: .ascii "Too low!"
  correct_1: .ascii "You got it in "
  correct_2: .ascii " moves! Great job!"

      print: ld %c [%a]
             jz %c done_print
             .char %c
             add %a 1
             jmp print

 done_print: ret %b

             .main
             ld %a intro_1
             call %b print
             .line
             ld %a intro_2
             call %b print
             .line
             ld %a intro_3
             call %b print
             .line
             .rand %0
             and %0 0x00FF
             ld %1 0

       loop: add %1 1
             ld %a guess
             call %b print
             .den %1
             .line
             ld %a prompt
             call %b print
             .input %2
             xor %3 %0 %2
             jz %3 if_correct
             mv %4 %0

    compare: sub %2 1
             jz %2 if_too_low
             sub %4 1
             jz %4 if_too_high
             jmp compare

 if_too_low: ld %a too_low
             call %b print
             .line
             jmp loop
```

```
if_too_high: ld %a too_high
             call %b print
             .line
             jmp loop

 if_correct: ld %a correct_1
             call %b print
             .den %1
             ld %a correct_2
             call %b print
             .line
             halt
```

Problem

Notice that this program needs to be able to compare two values to determine which clue to give and when to end the game. However, *Toy* does not have any circuits that are explicitly designed to compare two values.

- Identify the circuits used by *Toy* to compare two numbers in this program and explain how the comparison is accomplished at the machine level.

- Is this a very efficient way of comparing two numbers? Can you implement a more efficient method for comparing two numbers?

Problem 2: Reversing an Input String

Hello is a silly little program that demonstrates handling string input from the user. It asks the user for a name, gets the name backwards, asks the user if the backwards name is preferred and then says goodbye.

Example Run

```
Hello! What is your name? Mr Ambler
Nice to meet you, relbmA rM!
Whoops! I guess I got it backwards!
Do you prefer it that way (y/n)? y
I knew you would!
Good bye!
```

Assembly

```
      greet: .ascii "Hello! What is your name? "
        say: .ascii "Nice to meet you, "
    exclaim: .ascii "!"
    mistake: .ascii "Whoops! I guess I got it backwards!"
        ask: .ascii "Do you prefer it that way (y/n)? "
        yes: .ascii "I knew you would!"
         no: .ascii "Sorry to hear that!"
        bye: .ascii "Good bye!"

      print: ld %1 [%0]
             jz %1 done_print
             .char %1
             add %0 1
             jmp print

 done_print: ret %a

             .main
             ld %0 greet
             call %a print
             ld %0 user_input
             .string %0
             ld %0 say
             call %a print
             ld %0 user_input
             mv %1 %0

   find_end: ld %2 [%1]
             jz %2 found_end
             add %1 1
             jmp find_end
```

```
   found_end: ld %2 [%1]
              .char %2
              xor %2 %1 %0
              jz %2 backwards
              sub %1 1
              jmp found_end

   backwards: ld %0 exclaim
              call %a print
              .line
              ld %0 mistake
              call %a print
              .line
              ld %0 ask
              call %a print
              ld %0 user_input
              .string %0
              ld %0 [user_input]
              ld %1 0x79
              xor %2 %1 %0
              jz %2 user_prefers
              ld %0 no
              call %a print
              jmp end

user_prefers: ld %0 yes
              call %a print

         end: .line
              ld %0 bye
              call %a print
              .line
              halt

  user_input: .word
```

Problem

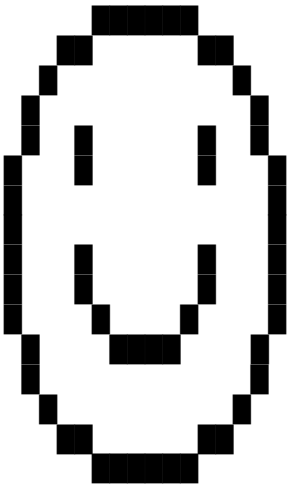
Notice that this program stores the user's name and later outputs the name backwards. Identify the *Toy* machine instructions involved and explain how this is accomplished at the level of the machine.

Problem 3: Selecting an Image to Output

Smiley is a silly little program that asks how we're feeling and then outputs a graphic to reflect this.

Example Run

```
How are you feeling today (1: happy, 2: neutral, 3: sad)? 1
```



Assembly

```
     prompt: .ascii "How are you feeling today (1: happy, 2: neutral, 3: sad)? "

             .main
             ld %0 prompt

loop_prompt: ld %1 [%0]
             jz %1 done_prompt
             .char %1
             add %0 1
             jmp loop_prompt

done_prompt: .input %0
             and %0 0x0003
             jp %0 okay
             halt

       okay: .line
             sub %0 1
             ld %a happy
             ld %b 16

  find_face: jz %0 draw_face
             add %a 16
             sub %0 1
             jmp find_face
```

```
draw_face: jz %b end
           ld %c [%a]
           .pattern %c
           add %a 1
           sub %b 1
           jmp draw_face

      end: .line
           halt

    happy: .data 0x07E0, 0x1818, 0x2004, 0x4002
           .data 0x4812, 0x8811, 0x8001, 0x8001
           .data 0x8811, 0x8811, 0x8421, 0x43C2
           .data 0x4002, 0x2004, 0x1818, 0x07E0

  neutral: .data 0x07E0, 0x1818, 0x2004, 0x4002
           .data 0x4002, 0x8811, 0x8811, 0x8001
           .data 0x8001, 0x8001, 0x8001, 0x4FF2
           .data 0x4002, 0x2004, 0x1818, 0x07E0

      sad: .data 0x07E0, 0x1818, 0x2004, 0x4002
           .data 0x4422, 0x8C31, 0x8001, 0x9009
           .data 0x83C1, 0x97E9, 0x87E1, 0x57EA
           .data 0x47E2, 0x2004, 0x1818, 0x07E0
```

Problem

Notice that the program stores the data used to draw the respective faces and then reads a subset of the data to output as an image. Identify the *Toy* machine instructions used to select the appropriate subset of data, and explain how the selection process is accomplished.

Problem 4: Writing an Assembly Program (Challenge)

Hurkle is a game in which a hurkle is hiding in a 16x16 grid. For each turn, the user guesses where the hurkle is by saying how far north and how far east to look. The computer then gives the user clues as to which direction the user should search next. The game continues until the user finds the hurkle, in which case the number of guesses taken is output.

An example run (with user input in bold) follows.

```
Hurkle!
Guess where it is!

Guess 1
 How far east? 7
 How far north? 7
Go northeast

Guess 2
 How far east? 10
 How far north? 10
Go northeast

Guess 3
 How far east? 14
 How far north? 14
Go south

Guess 4
 How far east? 14
 How far north? 13
Go south

Guess 5
 How far east? 14
 How far north? 12

You found it in 5 guesses!
```

Write an assembly program for this game and explain how the program awards clues and keeps track of the number of guesses the user has used. (Hint: this is fundamentally a similar problem to problem 1.)