California State University, Sacramento
College of Engineering and Computer Science

Computer Science 35: Introduction to Computer Architecture

Spring 2025 – Project – *Sacramento Trail*

Overview

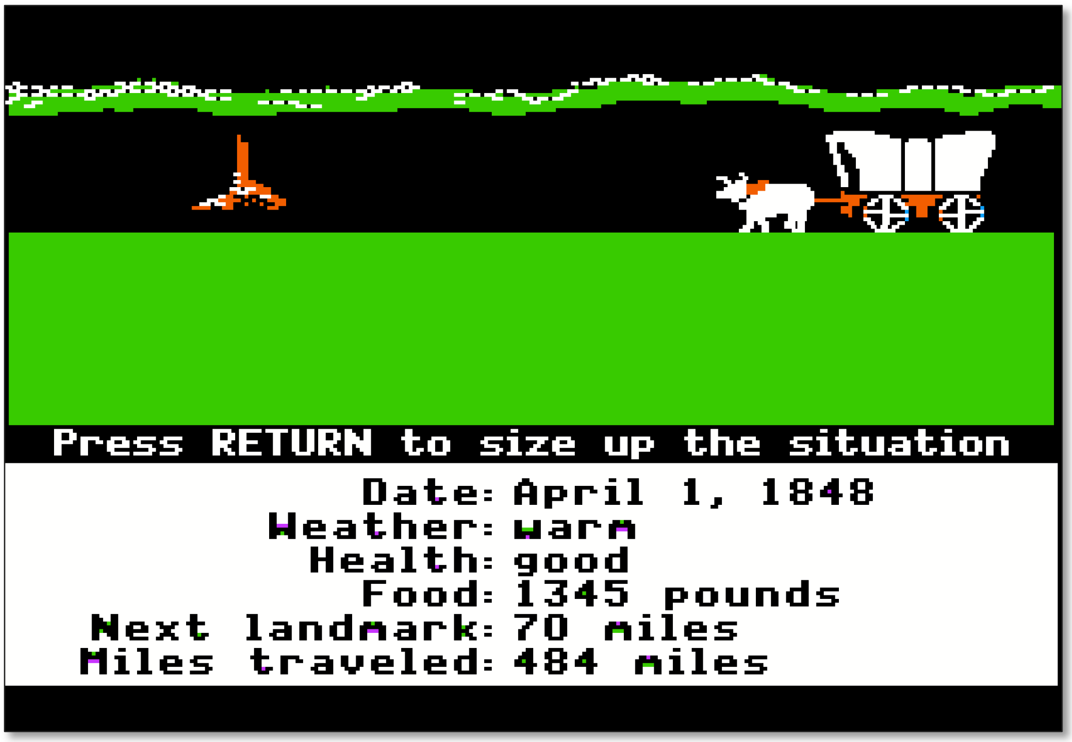
One of the greatest and most influential educational video games – ever - is Oregon Trail. This is a "resource management" game where the player attempts to keep resources (like health and food) high while the game keeps decreasing them. To make it fun, it is based on the historic Oregon Trail which settlers followed, in the mid nineteenth century, to Oregon and California.

The original version of the game was developed by Paul Dillenberger, Bill Heinemann, and Don Rawitsch in 1971. Yes, that's **54** years ago! The game was published a few years later, in 1975, by the Minnesota Educational Computing Consortium (MECC). Even though you might have never heard of MECC, this organization developed dozens of educational software titles and shaped the childhood of millions of children.

```
MONDAY MAY 10 1847

TOTAL MILEAGE IS 575
FOOD           BULLETS        CLOTHING       MISC. SUPP.    CASH
 46             1090           40             45             205
DO YOU WANT TO (1) STOP AT THE NEXT FORT, (2) HUNT, OR (3) CONTINUE
?2
TYPE BANG  BANG
RIGHT BETWEEN THE EYES---YOU GOT A BIG ONE!!!!
WATCH YOUR CALORIES TONIGHT!!!
DO YOU WANT TO EAT (1) POORLY  (2) MODERATELY
OR (3) WELL?2
HAIL STORM---SUPPLIES DAMAGED
```

Recreation of the original Oregon Trail (on green bar printer paper)

Over time, Oregon Trail has been ported dozens of times and implemented on dozens of computers. With each iteration, some details change, but the basic game holds true to history of the United States. One of the most popular versions started on the Apple II computer and was recreated, verbatim, on the IBM PC (Intel Processor).



```
Press RETURN to size up the situation
             Date: April 1, 1848
          Weather: warm
           Health: good
             Food: 1345 pounds
   Next landmark: 70 miles
  Miles traveled: 484 miles
```

Oregon Trail on the Apple II.

Your Game: Sacramento Trail

Overview

You are going to create a ***greatly simplified*** version of this game. The resources will also be simplified but will still create very basic "skeleton" of a game. You will have to travel a total 1,000 miles in 2 months. This is considerably shortened period from the original game: 3,000 miles in 6 months. We'll call this version "Sacramento Trail" for obvious reasons.

This will be a long journey, and your decisions will have an impact on the game – and whether you win or lose. You will need to rest, gather food, and do maintenance work on the wagon.

Your Goal

- You have to travel 1000 miles.
- You must do this in 60 days.

Your Resources

- Food. You will eat between 5 and 10 pounds of food each day. You will start with 30 pounds.

- Health. If it reaches zero, you die. Your health decreases between 5% and 10% each day you are traveling the trail. However, if you are starving (i.e., no food) it decreases between 10% and 20%.

- Wagon Condition. If it reaches zero, your wagon is broken and you cannot continue. You will have to repair it to continue.

Uh oh.

Your choices:

1. Rest. This increases your health between 30% and 60%. However, this might cost you anywhere from 1 to 5 days.

2. Repair Wagon. This increases the wagon's condition between 10% and 50%. However, this might cost you anywhere from 1 to 3 days.

3. Hunt for food. You can gain 0 to 100 pounds. Yes, it can be zero! And, this will take a day.

4. Continue on. You advance between 5 and 80 miles. This is grueling work on both the settlers and the wagon itself. Your health will drop between 5% and 10% per day. The wagon's condition will drop between 5% and 15% per day. If your wagon condition drops to zero (or below), it is broken, and you cannot proceed.

Sample Output

Your solution doesn't have to look exactly like this. However, this should show you the basic gameplay. For readability, the user's input is displayed in **blue**, resources are in **green**, and random values are in **orange**. You don't have to use color (unless you are going for extra credit). *As always, please feel free to change the wording of the text.*

```
============================
Sacramento Trail
============================

* You have 60 days to travel 1000 miles.
* Your resources
   * If your wagon's condition drops 5-15% each day. If it reaches zero,
     the wagon is broken and you can't move.
   * You will eat 5-10 pounds of food each day.
   * If you are starving, your health drops 10-20% each day.

* Your choices:
   1. Rest.  This increases your health (30-60%), but it takes between 1-5 days.
   2. Repair Wagon.  This improves the wagon (10-50%), but it takes between 1-3 days.
   2. Hunt for food. You can find between 20-200 pounds of food. It takes 1 day.
   3. Keep Traveling. You advance 5-80 miles, but your wagon's condition drops 5-15%.
      Your health also drops 5-10% each day.

*  If time runs out or your health drops to 0%, you lose.

JOURNEY DAY 1

Distance left    : 1000
Wagon condition : 100%
Your health      : 100%
Food left        : 30 pounds

Do you want to 1. Rest, 2. Repair wagon, 3. Hunt for food, or 4. Continue on?
4

You advanced 24 miles.
Your wagon condition dropped 5%
You lost 5 health.
On day 1, you ate 9 pounds of food.
```

If you are low on food, you can hunt.

```
JOURNEY DAY 7

Distance left   : 819
Wagon condition : 56%
Your health     : 63%
Food left       : 10 pounds

Do you want to 1. Rest, 2. Repair wagon, 3. Hunt for food, or 4. Continue on?
3

BANG! You gained 89 pounds of food.

On day 7, you ate 9 pounds of food.
```

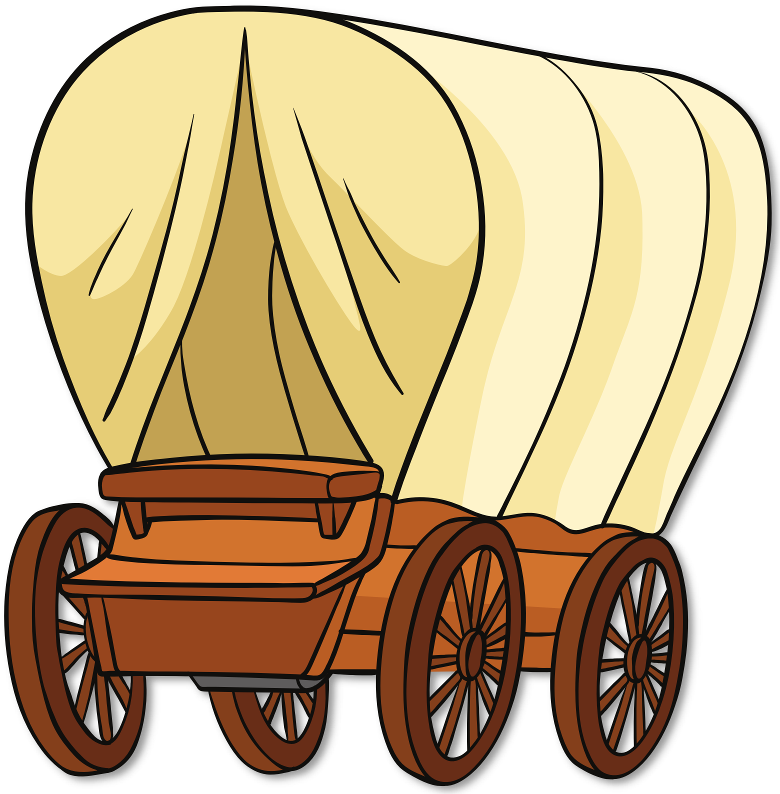
If the wagon is broken, you will have to perform some repairs before you can continue. In this case, repairs took three days. Notice that you had to eat every day.

```
JOURNEY DAY 16

Distance left   : 438
Wagon condition : -10%
Your health     : 4%
Food left       : 35 pounds

Do you want to 1. Rest, 2. Repair wagon, 3. Hunt for food, or 4. Continue on?
2

** The repairs took 3 days. **

Wagon condition gained 22%

On day 16, you ate 10 pounds of food.
On day 17, you ate 6 pounds of food.
On day 18, you ate 7 pounds of food.
```

If your health is getting very low, it is important to rest. Resting can take a few days.

```
JOURNEY DAY 23

Distance left    : 579
Wagon condition : -2%
Your health      : 9%
Food left        : 16 pounds

Do you want to 1. Rest, 2. Repair wagon, 3. Hunt for food, or 4. Continue on?
1

** You rest for 3 days. **

You gained 32% health.

On day 23, you ate 10 pounds of food.
On day 24, you ate 9 pounds of food.
YOU ARE STARVING ON DAY 25! You lost 20 health.
```

Oh no! You ran out of food on Day 25 and lost 20 health. You need to make decisions carefully.



Getting the Demo Program

You can download a copy of the game (in its most basic form). **Type the following verbatim:**

```
curl devincook.com/csc/35/trail > trail
```

Again, UNIX will display information during the download. You can ignore this. It means it worked correctly.

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100 19584  100 19584    0     0  35737      0 --:--:-- --:--:-- --:--:-- 35737
```

You can check that the file downloaded by typing `ll` (that's two lowercase L's) and pressing the Enter Key. You should see the library file listed. It should have **19,584 bytes**. **If it doesn't, you typed the command above incorrectly.**

Once you have downloaded the file, you need to change is access rights so you can execute it. Type the following:

```
chmod 500 trail
```

You'll learn more about this command in CSC 60. Now, to run the program type the following:

```
./trail
```

Project Pseudocode

The single-letter variables, like `i`, are temporary and can just be stored in registers. I've colored each of the important variables. You should store these in a specifically chosen register or a memory location using `.quad`.

```
assign distance = 1000, wagon = 100, health = 100, food = 100, day = 1

while distance >= 1 and day <= 60 and health >= 1
    ... Part 1: Summary
    print "JOURNEY DAY ", day

    print "Distance left    : ", distance
    print "Wagon condition : ", wagon
    print "Your health      : ", health
    print "Food left        : ", food

    ... Part 2: Decision
    print "Do you want to 1. Rest, 2. Repair wagon, 3. Hunt for food, 4. Continue on?"
    read choice

    switch choice
        case 1:          ...Rest
            duration = random number between 1 and 5
            print "You rest for ", duration, " days."

            i = random number between 30 and 60
            add i to health
            print "You gained ", i, "% health."

        case 2:          ...Repair
            duration = random number between 1 and 3
            print "The repairs took ", duration,  " days"

            i = random number between 10 and 50
            add i to wagon
            print "Wagon condition gained ", i, "%."

        case 3:          ...Hunt for food
            duration = 1

            i = random number between 20 and 100
            add i to food
            print "You gained ", i, " pounds of food."
        case 4:          ...Travel
            duration = 1

            if wagon >= 1
                i = random number between 5 and 80
                add i to distance
                print "You advanced ", i, " miles."

                w = random number between 5 and 15
                subtract w from wagon
                print " Your wagon condition dropped ", w, "%."
            else
                print " YOUR WAGON IS BROKEN!\n\0"
            end if

            ... You also lose some heath
            h = random number between 5 and 10
            subtract h from health
            print "You lost ", h, " health."
    end switch

    ... Part 4: Duration in days. Decrement food (and maybe health)
    loop duration number of times
        if food > 0
            i = random number between 5 and 10
            subtract i from food
            print "On day", day, " you ate ", i, " pounds of food."
        else
            h = random number between 10 and 30
            subtract h from health
            print "YOU ARE STARVING! ON DAY ", day, "! You lost ", h, " health."
        end if

        add 1 to day
    next loop
end while
```

Extra Credit

1. *Color – 5 points*

Make use of color to enhance your game. The color must be meaningful – don't just set the color at the beginning of the program.

2. *ASCII Art – 5 points each for a max of 10*

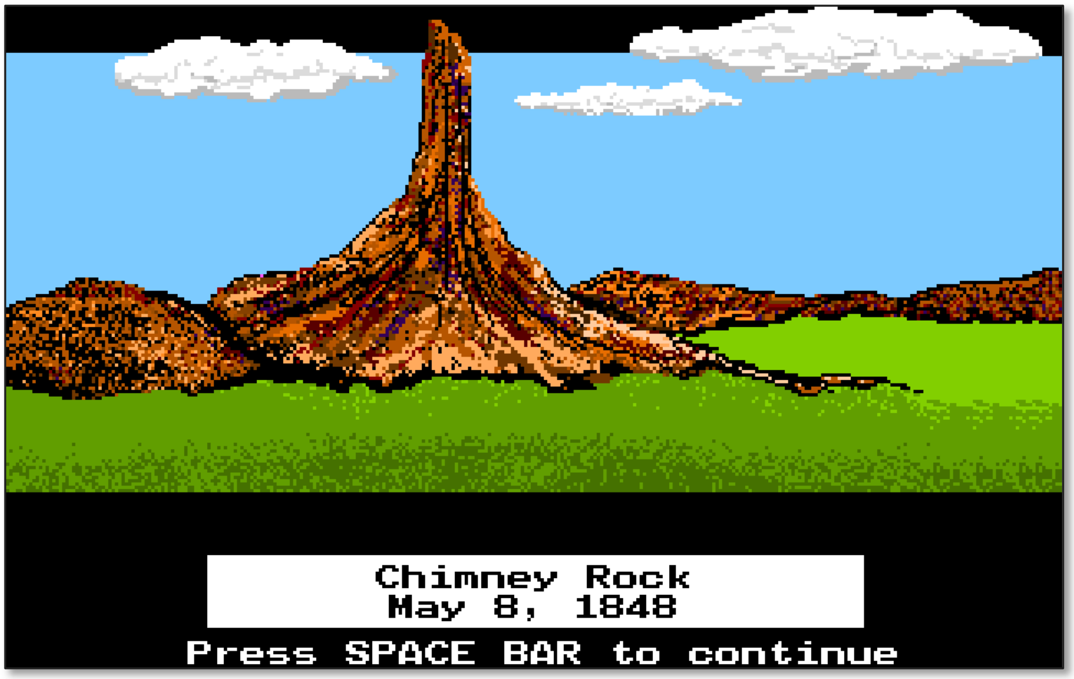
Use ASCII-art to make your program exciting. The ASCII-art must be meaningful and not something overly simple like:

```
==========:)
```

It's a happy worm!

You can use multiple `.ascii` directives under the same label. Only place the `\0` at the end of the final one. You will also have to put a backslash `\` before any other backslashes or double-quotes `"`. For example, the following creates a square.

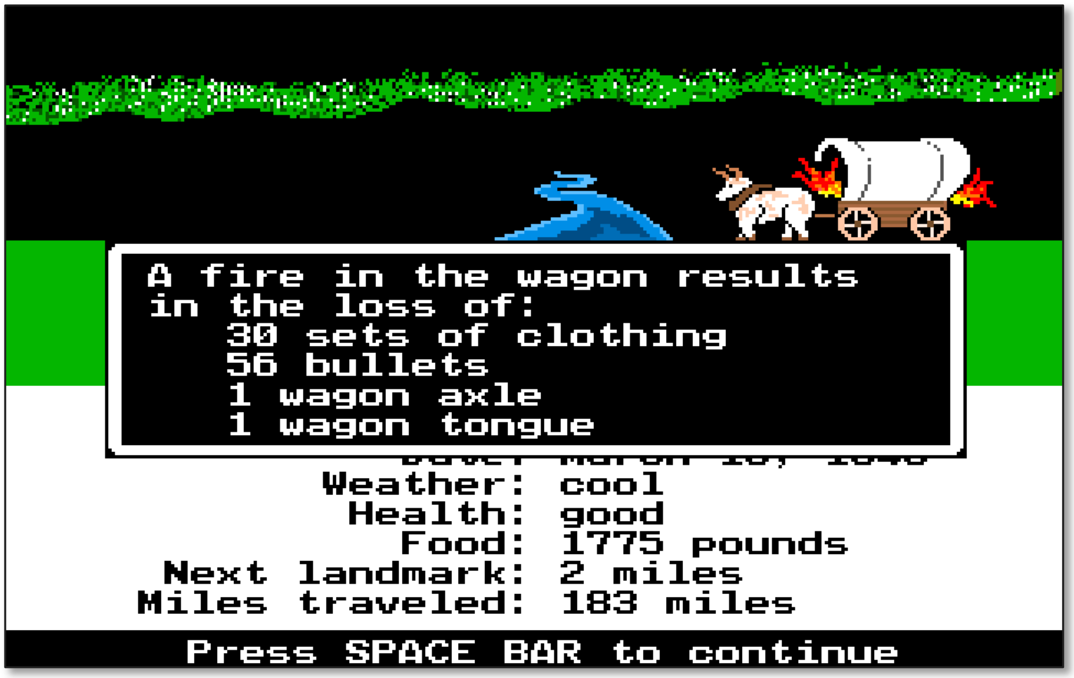
```
Square:
    .ascii "****\n"
    .ascii "*  *\n"
    .ascii "****\n\0"
```



Images, even ASCII art, can be immersive.

3. Random events – 5 each for a max of 20 points.

What other types of events, good and bad, can occur. The more you add, the more immersive your game will become.



```
A fire in the wagon results
in the loss of:
   30 sets of clothing
   56 bullets
   1 wagon axle
   1 wagon tongue

       Weather: cool
        Health: good
          Food: 1775 pounds
 Next landmark: 2 miles
Miles traveled: 183 miles

   Press SPACE BAR to continue
```

This random event was negative. They can be good.

4. Bound the percentages – 5 points

The most basic version of the program allows the student to have knowledge, stress, and endurance outside the "normal" range of 0% to 100%. For this extra credit, make sure these values stay in the normal range.

5. At least 2 more decisions – 5 points

There is more to do than just hunting, resting, and making repairs. Give the program more decisions that can help or hinder the game.

6. Awesomeness – 5 to 10 points

If you do something incredibly clever, or add a really impressive feature, I can give you up to 10 points.

Requirements

⚠ **YOU MUST DO YOUR OWN WORK. DO NOT ASK OTHER STUDENTS FOR HELP.**

If you ask for help, both you and the student who helped you will receive a 0. Based on the severity, I might have to go to the University.

⚠ **This activity may only be submitted using the assembly format used in class:**

1. **x64**
2. **Intel-syntax**
3. **Uses the CSC 35 library.**

Using AT&T syntax will result in a zero. Any work from a prior semester will receive a zero.

1. **Print the title of your program.**

2. **Print the game rules. Let the player know how the game works!**

3. **Loop until distance ≥ 1000.**
   If you change the project theme, please feel free to change this value (e.g. 12 for months)

4. **Exit if your health drops to zero or the day exceeds 60.**
   If you do the extra credit below, the game won't exit, but something else will happen.

5. **Decision.**
   Input the player's choice. There needs to be at least 4 choices. The program must do different things based on the input.

6. **Broken wagon.**
   If Wagon's endurance is at zero, you cannot advance.

7. **Loop for the duration (in days).**
   Some actions take several days. Your food needs to decrease each day... and if you starving, it drops rapidly.

8. **Comment your code!**

9. **Proper formatting:**
   Labels are never indented. Instructions are always indented the same number of spaces. Add blank lines for readability.

Due Date

The assignment is due **May 4th by midnight**.. I strongly suggest that you get to work on this assignment as early as possible. If you did well on your labs, it shouldn't take more than a few hours.

Have Fun

Use your imagination. Your game doesn't have to be based on Oregon Trail. You can base your game on any fun theme that you want. But, only if you keep the same gameplay. For example, here are some possible scenarios:

- Animals
- Cartoons: Spongebob Squarepants, Rick and Morty, Archer, etc….
- Movies: comedy, sci-fi, horror, etc…
- Video games
- Television programs
- Characters from a book
- etc…

Tips

- **DON'T** attempt to write the entire program at one time. If you do, you won't be able to debug it. Experienced programmers use incremental design. Make a basic program and, very slowly, add the features you need.
- So, first get the main loop working… then, bit by bit, add the rest of the functionality.
- If you get stuck in an infinite loop, you can press **Control + C** to exit any UNIX program.

Random Numbers

The library has a built-in subroutine called "Random" that you must use to make your project work.

You will pass the **range** of numbers into `rsi`. After you call Random, rsi will contain a random number between 0 to n - 1 into `rsi`. For example, if you store 100 into `rsi` and call the subroutine, `rsi` will contain 0 to 99.

So, for example, do you create a random number between 10 and 30? Well, let's look at the actual range (or, in other words, how many numbers there are). Here, the range is 21 which is computed using (30 - 10 + 1). If we call Random with `rsi` set to 21, it will return a number between 0 and 20. So, how do we make it a value between 10 and 30? Just add 10!

In general, this is the mathematical equation that is used for getting a random number with a *minimum* value and a given *range*:

```
minimum + Random(range)
```

The actual assembly, for the example above, will look like the following:

```
mov   rsi, 21
call  GetRandom          #rsi =  0 .. 20
add   rsi, 10            #rsi = 10 .. 30
```

Connecting to the Server

Please use **ONE** of the following. I recommend using MobaXTerm, but PuTTY is still an excellent application. You don't have to open both of them. You have many options to "Telnet" to a remote server.

MobaXTerm Instructions

1. Open MobaXTerm.

2. Click on the "Session" button on the top-left corner of the screen.

3. Click on SSH (it stands for Secure Shell).

4. Enter the following in the Remote Host box:

```
coding1.ecs.csus.edu
```

5. Click the "Ok" button.

6. Enter your Saclink username and password. Please note: when you type a password in UNIX, it doesn't display any text to the screen. Don't worry, UNIX is listening.

PuTTY Instructions

1. Open PuTTY.

2. Enter the following in the Host Name box:

```
coding1.ecs.csus.edu
```

3. Make sure the SSH checkbox is selected. (SSH stands for Secure Shell).

4. Click the "Open" button.

5. Enter your Saclink username and password. Please note: when you type a password in UNIX, it doesn't display any text to the screen. Don't worry, UNIX is listening.

Submitting Your Lab

This activity may only be submitted using the assembly format used in class:

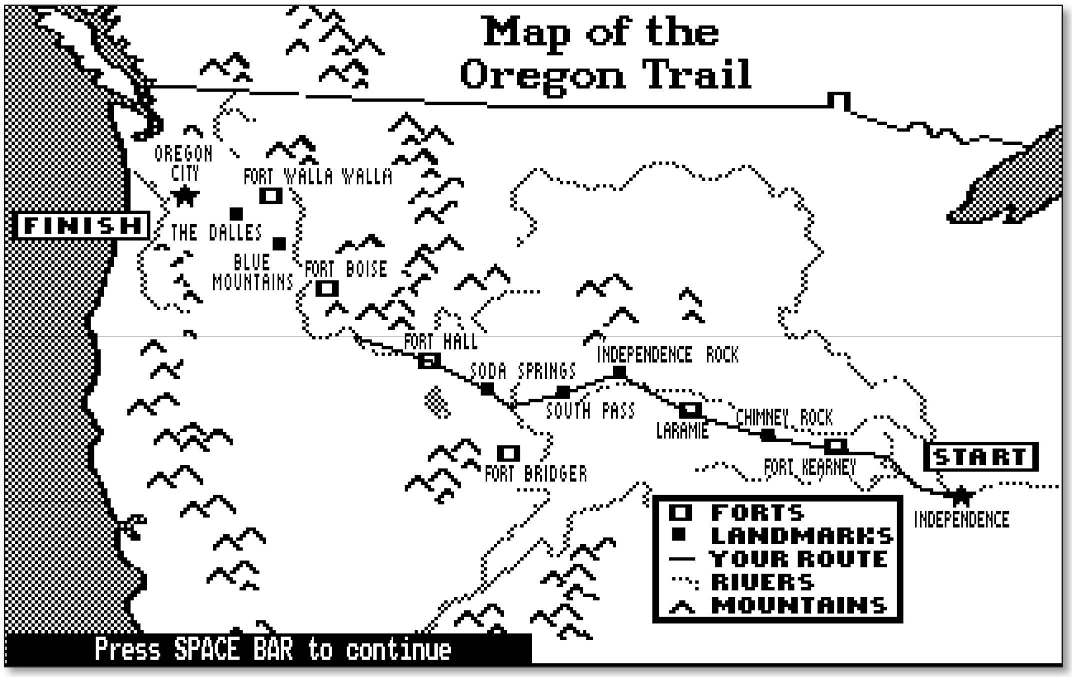
4. **x64**
5. **Intel-syntax**
6. **Uses the CSC 35 library.**

Using AT&T syntax will result in a zero. Any work from a prior semester will receive a zero.

Type the following to submit your lab. This will transfer a copy of your lab to me. In the example below, the project file is called "project.asm". You can call it anything you want.

```
./submit project.asm
```

NOTE: You can only submit your assignment once. Make sure you got it working.



UNIX Commands

Editing

Action	Command	Notes
Edit File	`nano` *filename*	"Nano" is an easy-to-use text editor.
E-Mail	`alpine`	"Alpine" is text-based e-mail application. You will e-mail your assignments it.
Assemble File	`as -o` *object source*	Don't mix up the *objectfile* and *asmfile* fields. It will destroy your program!
Link File	`ld -o` *exe object(s)*	Link and create an executable file from one (or more) object files

Folder Navigation

Action	Command	Description
Change current folder	`cd` *foldername*	"Changes Directory"
Go to parent folder	`cd ..`	Think of it as the "back button".
Show current folder	`pwd`	Gives the current a file path
List files	`ls`	Lists the files in current directory.

File Organization

Action	Command	Description
Create folder	`mkdir` *foldername*	Folders are called directories in UNIX.
Copy file	`cp` *oldfile newfile*	Make a copy of an existing file
Move file	`mv` *filename foldername*	Moves a file to a destination folder
Rename file	`mv` *oldname newname*	Note: same command as "move".
Delete file	`rm` *filename*	Remove (delete) a file. There is **no** undo.