“Always code as if the (person) who ends up maintaining your code will be a violent psychopath who knows where you live.”

– M. Golding

**Learning Objectives**

1. Introduction to memory-mapped I/O
2. Introduction to interrupt handlers
3. Basic understanding of SPIMbot simulator

**Work that needs to be handed in (via SVN)**

1. **part1.s**: Write a bot that is able to harvest at least 3 tiles via SPIMbot’s harvesting mechanism. **This is due by the first deadline.** Run with:
   ```bash
   QtSpimbot -part1 -file part1.s
   ```
2. **part2.s**: Write a bot that is able to put out fires on at least 3 tiles using SPIMbot’s watering mechanism. **This is due by the second deadline.** Run with:
   ```bash
   QtSpimbot -part2 -file part2.s
   ```

**Important!**

This is a solo lab. You may not work with anyone else on this lab.

We also cannot stress enough the importance of reading the entire lab handout.

**Guidelines**

- Same procedure as previous labs on MIPS. Use any MIPS instructions or pseudo-instructions you want. We may try to break your code.
- You will find it useful to refer to Appendix A.7 of the book, the example interrupt service routines (bonk.c and bonk.s) on the lecture notes page, and the SPIMbot documentation. Also, we provide an electronic version of the code from the discussion handout as a file in your SVN. See:
  - [https://wiki.cites.illinois.edu/wiki/display/cs233fa16/SPIMbot+documentation](https://wiki.cites.illinois.edu/wiki/display/cs233fa16/SPIMbot+documentation)
  - example.s
Desert World!

For this semester, we have deployed SPIMbot to a strange desert land, which is cohabitated by a species of distantly related bots. In order to survive in this rugged landscape, SPIMbot must grow and harvest plants for food and survival. There is little to see in this desolate land, but there are several abandoned crop and agricultural buildings that are suspected to have useful resources, including seeds, water, and fire. SPIMbot must forage for every scrap of seed and water in order to grow and harvest plants to survive.

The world is divided into $10 \times 10$ tiles (100 total). Since the desert land is arid and dry, one plant needs an entire tile to grow properly. SPIMbot decides to grow its crops in these tiles, using one tile for each plant. The other bots living in this land follow this tiling scheme as well, and the two robot races agree to grow their plants on different tiles, as there is enough land to go around for everyone.

The distantly related bots have been peaceful and have not caused any trouble for SPIMbot so far. These bots do not consume the same food as SPIMbot so there has been no trouble or interference between the survival of the two robot races. SPIMbot doesn’t anticipate having any trouble with their mysterious distant cousin.

When SPIMbot arrives in this mysterious land, they notice several growing plants in the land that they can harvest for food. Harvesting these plants will give SPIMbot a hopeful start for survival in this difficult land.

The entire SPIMbot race is counting on you to harvest these plants to survive, so good luck and may the force be with you!

---

1 Obviously a robot requires food. Why would you even question that?
Part 1: Harvesting Plants [40 points]

For this part of the lab you are required to write a bot that will harvest plants. In order to harvest plants, you first need to know where they are. You can find plants by using memory mapped I/O to request an array of TileInfo structs. This array will contain 100 TileInfo structs, and each struct will have information about if there is currently a plant growing on that tile, and if that plant is a plant that SPIMbot can harvest for consumption.

Don’t forget to use the -part1 flag!

Scanning for tiles

In order to get the TileInfo array, a memory mapped address, called TILE_SCAN, is provided. You should allocate a static memory space (array) of 1,600 bytes (4 bytes × 4 words × 100 tiles) in your .data section using the .space directive. It is also recommended that you use the .align directive to ensure word alignment. You can then write the address of that memory space to the TILE_SCAN memory I/O. This will tell SPIMbot to write the TileInfo array to the memory space you provided.

The TileInfo array will contain 100 TileInfo structs. The array will map to the world much like a standard 2D array would be mapped in memory. For example, the 13th element in the array would reference the tile that is in the 2nd row and 3rd column of the world.

Each TileInfo struct will contain a state, an indication of the owning bot, the growth rate of the tile, and the amount of water held in the tile. Each of these elements of data will be ints, so the data in the TileInfo struct will be 16 bytes (4 bytes × 4 ints). The following struct represents how a TileInfo struct is arranged:

```c
struct TileInfo {
    int state; // Either 0 for EMPTY, 1 for GROWING
    int owning_bot; // 0 for owned by SPIMbot, 1 for owned by cohabitating bot
    int growth;
    int water;
};
```

For this lab, you will only have to worry about the first element in TileInfo structs - state - to determine if there is a plant growing in that tile. Note that although there will be no cohabitating bot for this lab, SPIMbot may encounter cohabitating bots in a future lab. Don’t worry about the growth rate and the amount of water at this point - these will be relevant for future labs.

In summary, you should perform the following procedure to request the TileInfo array:

1. Allocate static memory in the .data section
2. Load the address of this memory into a register
3. Write this address to the TILE_SCAN memory I/O to tell SPIMbot where the array of TileInfo structs should be stored
4. Read from the array to get the data of a plant
5. Repeat steps 2-4 every time you want to locate the plants again

Note that the TileInfo array will not be updated automatically and will only be updated whenever you write its address to TILE_SCAN.
The following snippet of MIPS code represents an example of how this could be implemented:

```
.data
TILE_SCAN = 0xffff0024

# This align directive will make sure the space is aligned to 2^2 bytes
.align 2
.tile_data: .space 1600

.text
main:
    ...
    la  $t0, tile_data
    sw  $t0, TILE_SCAN
    # tile_data has now been populated with the array of TileInfo structs
    ...
```

Harvesting Plants

Once you know where your plants are located, SPIMbot can drive above these plants, and harvest the plant for consumption.

You can control SPIMbot using the VELOCITY, ANGLE, and ANGLE_CONTROL memory mapped I/O addresses. You can get SPIMbot’s position using the BOT_X and BOT_Y memory mapped I/O addresses.

Note that once you find a tile that contains a plant, you will have to extract the coordinates of the tile. Using the index of the TileInfo struct in the TileInfo array, and your knowledge that the world is a 300 × 300 coordinate system, you can compute the coordinates of the chosen tile.

You can control SPIMbot’s harvesting mechanism using the HARVEST_TILE memory mapped I/O address. See the online SPIMbot documentation at the link from the beginning of this handout for more details.

Note that it is highly recommended that you write pseudo code before writing MIPS so you can consider various edge cases before diving into the nitty-gritty details of assembly. During office hours, we may ask to see your pseudo code before helping you.

Since this is your first experience navigating SPIMbot, we have outlined below one possible way to harvest one plant (you do not have to follow this suggestion).

1. Scan for tiles and pick a tile with a plant to harvest.
2. Move SPIMbot horizontally in order to match the chosen tile’s x coordinate.
3. Repeat step 2 until you reach the tile’s x coordinate.
4. Move SPIMbot vertically in order to match the chosen tile’s y coordinate.
5. Repeat step 4 until you reach the tile’s y coordinate.
6. Stop SPIMbot when it is directly above the tile.
7. Harvest the tile.

Grading

You will be graded based on if you have harvested 3 tiles within the time limit of 10,000,000 cycles. Do not blindly harvest a tile without checking if it has a growing plant or not. **You will lose points if you attempt to harvest a tile that does not have a growing plant.**

Be sure to check the terminal output with the `-debug` flag to make sure your bot is successfully harvesting...
plants. You can see the plant disappear when it is harvested and your score will increase by 1 on the map. **To receive full credit you must harvest at least 3 plants.**

The locations of the plants and your SPIMbot may not be the same when your SPIMbot is graded so be sure to make your code adaptable. You can use the `-randommap` or `-mapseed` flags to try out different scenarios.

**Important things to remember**

- The minimum space you need to allocate for the TileInfo array you give to `TILE_SCAN` is 1600 bytes (4 bytes × 4 words × 100 tiles).
- Due to inaccuracies with SPIMbot’s sensors, you should aim for the center of the tiles.
- There will be 10 growing tiles at the beginning. You will only need to harvest 3 of these tiles, but you are free to harvest all 10 if you choose to.
- You do not need to implement interrupts for this part (but you can if you want).
- You should make sure you don’t hit the boundary since if the bot hits the boundary it’s velocity will be set to 0. Alternatively you can handle the bonk interrupt and set your velocity back to normal upon hitting the boundary.
- Don’t forget that your SPIMbot has a radius of approximately 5 so you will hit the boundary before your \( x \) or \( y \) coordinate reaches 0 or 300.
- SpimBot automatically exits after 10,000,000 cycles, so make sure your SPIMbot is efficient enough to harvest at least three plants before it exits.
- You can find all the memory mapped addresses you need in the SPIMbot documentation link provided in the Guidelines section at the beginning. Additionally, the documentation also describes other flags that may be extremely helpful.
Part 2: Putting Out Fires [60 points]

In this part of the lab you will extend what you implemented in part one with interrupts in order to put out fires that have broken out on your plants.

Don’t forget to use the -part2 flag!

It is an especially hot and dry day, and your plants have been spontaneously catching on fire. Luckily, SPIMbot is able to put out these fires by dumping water on the burning tile. SPIMbot can use the ON_FIRE interrupt to detect when a plant has caught on fire.

Fire interrupt

The fire interrupt will fire\(^2\) whenever a plant spontaneously catches on fire. You will have to use GET_FIRE_LOC memory mapped I/O address to find out exactly which tile triggered the interrupt. You must receive the ON_FIRE interrupt in order for GET_FIRE_LOC to work correctly. **Note that GET_FIRE_LOC only works inside the interrupt handler.**

GET_FIRE_LOC will return a 32-bit number where the upper 16 bits have the x_index and the lower 16 bits have the y_index. These values are encoded as two 16-bit numbers like the following struct:

```c
struct fire_loc {
    short x_index;
    short y_index;
};
```

It is important to remember, however, that this is a 32-bit number, *not* a struct. So, you will use logical and bitwise operations (probably) to extract the two numbers.

See the online SPIMbot documentation at the link from the beginning of this handout for more details and tips about the fire interrupt!

Putting Out Fires with Water

Resources are limited in desert world, and typically, it is near impossible to acquire water. Fortunately, SPIMbot has found that one of the abandoned buildings\(^3\), is unlocked and has a large store of water. At the beginning of this lab, SPIMbot’s water tank will be filled with this water.

Once you know where your burning plants are located and SPIMbot is armed with water, SPIMbot can drive above these burning plants, and dump water on them. You can control SPIMbot’s watering mechanism using the PUT_OUT_FIRE memory mapped I/O address. This mechanism will dump the exact amount of water needed to put out the fire at the tile that SPIMbot is above. See the online SPIMbot documentation at the link from the beginning of this handout for more details.

Again, it is highly recommended that you write pseudo code before writing MIPS so you can consider various edge cases before diving into the nitty-gritty details of assembly. During office hours, we may ask to see your pseudo code before helping you.

---

\(^2\)Pun intended :)

\(^3\)This building is mysteriously labeled “Crop Sciences Building”
Grading

You will be graded based on if you successfully put out 3 fires within the time limit of 10,000,000 cycles. You will also be graded for acknowledging the ON_FIRE interrupt so you must use the ON_FIRE interrupt. Do not blindly attempt to put out a fire on a tile without checking if it is on fire or not.

You will lose points if you:

- Attempt to put out a fire on a tile that is not on fire.\(^4\)
- Harvest growing tiles to avoid them from catching on fire by using the mechanism you wrote for part 1.

Be sure to check the terminal output with the -debug flag to make sure your bot is successfully putting out fires. You can see the fire go out when it is put out but your score on the map will not change. To receive full credit you must put out at least 3 fires.

The locations of the plants, the fires, and your SPIMbot may not be the same when your SPIMbot is graded so be sure to make your code adaptable. You can use the -randommap or -mapseed flags to try out different scenarios.

Don’t forget to refer to the interrupt handler MIPS code provided to you in examples.s and the SPIMbot documentation link provided in the Guidelines section in the beginning.

Important things to remember

- Most of the stuff from Important things to remember from Part 1.
- There will be 10 growing tiles at the beginning, and 5 of these will spontaneously catch on fire. You will only need to put out 3 of these tiles, but you are free to put out all 5 tiles if you choose to.
- Be careful to save/restore any registers you use in your interrupt handler so you don’t mess up your userspace code.
- You may need to find some way for your interrupt handler code to communicate to your userspace code. Registers won’t work since you can’t be sure that the register has something important already, so you’ll need to use the only other way you can save data in MIPS.
- Don’t forget about the -debug flag. It will printout extra useful information that could help you debug your SPIMbot.

Good Luck!

---

\(^4\) Besides, if you did this, you would be wasting SPIMbot’s limited supply of water.