"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 catch 500 Zentners [1] of bunnies via SPIMbot's bunny catching mechanism. **This is due by the first deadline.** Run with:
   `QtSpimbot -part1 -file part1.s`
2. `part2.s`: Write a bot that is able to put 100 Zentners of bunnies in the playpen using SPIMbot's bunny placement mechanism. **This is due by the second deadline.** Run with:
   `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:
  - `http://pages.cs.wisc.edu/~larus/HP_AppA.pdf`
  - `https://wiki.cites.illinois.edu/wiki/display/cs233sp17/SPIMbot+documentation`
  - `example.s`

[1] The Zentner is an archaic unit of mass that was selected for no reason in particular. :) If you're curious, you can learn more here: `https://en.wikipedia.org/wiki/Zentner`

Bunny Land![2]

For this semester, we have deployed SPIMbot to a strange but extremely joyous land, which is inhabited by adorable bunnies. These bunnies are known to have mystical powers of cuteness that can fix almost any problem, and it is suspected that these powers are the key towards world peace. SPIMbot must catch bunnies so that their cuteness can be magnified to bring great joy and prosperity to the SPIMbot species. You can befriend a bunny by giving it a carrot, at which point the bunny will jump onto SPIMbot's back. Furthermore, the bunnies are very clever and will devise ways to escape from SPIMbot, so SPIMbot must work very hard in order to acquire the mystical bunny cuteness.

When SPIMbot arrives in this mysterious land, equipped with exactly 100 carrots, SPIMbot notices several roaming bunnies. Capturing these bunnies will prepare SPIMbot for their mission of maximizing cuteness and bringing about world peace.

The entire SPIMbot race is counting on you to capture these bunnies to save the world, so good luck and may the force be with you!

[2] For my fellow bunny lovers out there: `http://pin.it/2NABWQ3`

Part 1: Capturing Bunnies [40 points]

For this part of the lab you are required to write a bot that will capture bunnies.

Don't forget to use the `-part1` flag!

Searching for Bunnies

In order to capture bunnies, you first need to know where they are. You can find bunnies by using memory mapped I/O to request a BunniesInfo struct:

```
struct BunniesInfo {
    int num_bunnies; // number of bunnies on map
    Bunny info[30];  // array of Bunny structs containing info about each bunny
};
```

This struct will contain information about the bunnies currently roaming in the land, including the number of bunnies, and structs containing more specific information about each bunny (e.g. location, weight, etc.).

In order to get the BunniesInfo struct, a memory mapped address, called `SEARCH_BUNNIES`, is provided. You should allocate a static memory space for your BunniesInfo struct in your `.data` section using the `.space` directive. You should allocate 484 bytes (4 bytes (`num_bunnies` int) + 4 bytes × 4 words (in Bunny struct) × 30 Bunny structs). 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 `SEARCH_BUNNIES` memory I/O. This will tell SPIMbot to write the BunniesInfo struct to the memory space you provided.

Each Bunny struct will contain an x position, a y position, a weight, and the number of remaining cycles (before a bunny hop). Each of these elements of data will be ints, so the data in the Bunny struct will be 16 bytes (4 bytes × 4 ints). The following struct represents how a Bunny struct is arranged:

```
struct Bunny {
    int x;      // x location of bunny, in range [5, 295]
    int y;      // y location of bunny, in range [5, 295]
    int weight; // weight in Zentners, in range [5, 20]
    int remaining_cycles; // cycles remaning before bunny hops, in range [0, 1000000]
};
```

For this lab, you will only have to worry about the first three elements in the Bunny struct (`x`, `y`, and `weight`) to catch 500 Zentners worth of bunnies. Don't worry about the `remaining_cycles` at this point, this will be relevant for part 2.

In summary, you should perform the following procedure to request the BunniesInfo struct:

1. Allocate static memory in the `.data` section
2. Load the address of this memory into a register
3. Write this address to the `SEARCH_BUNNIES` memory I/O to tell SPIMbot where the BunniesInfo struct should be stored
4. Read from the struct to get the data of the bunnies

The following snippet of MIPS code represents an example of how this could be implemented:

```
.data
SEARCH_BUNNIES = 0xffff0054

# This align directive will make sure the space is aligned to 2^2 bytes
.align 2
bunnies_data: .space 484

.text
main:
        ...
        la      $t0, bunnies_data
        sw      $t0, SEARCH_BUNNIES
        # bunnies_data has now been populated with the BunniesInfo struct
        ...
```

Catching Bunnies

Once you know where bunnies are located, SPIMbot can drive above these bunnies, and catch them.

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. Keep in mind that the world is a 300 × 300 coordinate system.

You can control SPIMbot's bunny catching mechanism using the `CATCH_BUNNY` memory mapped I/O address. This will use 1 carrot to befriend and catch one bunny within 5 pixels of SPIMbot. 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 capture one bunny (you do not have to follow this suggestion).

1. Scan for bunnies and pick a bunny to catch.
2. Move SPIMbot horizontally in order to match the chosen bunny's x coordinate.
3. Repeat step 2 until you reach the bunny's x coordinate.
4. Move SPIMbot vertically in order to match the chosen bunny's y coordinate.
5. Repeat step 4 until you reach the bunny's y coordinate.
6. Stop SPIMbot when it is directly above the tile.
7. Catch the bunny.

Grading

You will be graded based on if you have caught 500 Zentners of bunnies within the time limit of 10,000,000 cycles (the weight displayed in the bottom left of the map should be $\geq$ 500). Do not blindly use the `CATCH_BUNNY` memory mapped I/O without being sure that there is a bunny beneath SPIMbot. **You will waste 1 carrot if you use `CATCH_BUNNY` when there is no bunny within 5 pixels of SPIMbot's current location.**

Be sure to check the terminal output with the `-debug` flag to make sure your bot is successfully catching bunnies. You can see the bunny disappear when it is caught and your weight will increase by the weight of the bunny on the map. **To receive full credit you must capture at least 500 Zentners of bunnies.**

The locations of the bunnies 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 BunniesInfo array you give to `SEARCH_BUNNIES` is 484 bytes (4 bytes (`num_bunnies` int) + 4 bytes × 4 words (in Bunny struct) × 30 Bunny structs).
- Due to inaccuracies with SPIMbot's sensors, you should aim for the center of the bunny.
- There will be 10 bunnies at random locations with different weights at the beginning, and SPIMbot will begin with 100 carrots. The bunnies will not move, and more bunnies will spawn throughout the game.
- You will need to catch 500 Zentners of bunnies.
- `num_bunnies` in the BunniesInfo struct will have a range of [0, 10] for this part.
- 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, its 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 catch 500 Zentners of bunnies 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 Hopping Bunnies in Playpen [60 points]

In this part of the lab you will extend what you implemented in part one with interrupts in order to put caught bunnies into a playpen. Furthermore, this time, the bunnies have become smarter, and have started to move in order to hide from SPIMbot.

Don't forget to use the `-part2` flag!

Playpen

It can be heavy carrying so many bunnies around on your back![3] Luckily, a mysterious playpen has appeared in this land, and SPIMbot can drop off bunnies in this playpen.

After you catch a bunny using your logic from part 1, you will have to use `PLAYPEN_LOCATION` to find the playpen, and `PUT_BUNNIES_IN_PLAYPEN` to put some number of bunnies into the playpen.

`PLAYPEN_LOCATION` will return a 32-bit number where the upper 16 bits have the x_location and the lower 16 bits have the y_location. These values are encoded as two 16-bit numbers like the following:

31 16	15 0
0 1 0 1 1 1 0 0 0 0 1 1 1 0 1 1	1 1 0 1 0 0 0 1 0 0 1 1 0 1 0 0
x_location	y_location

You will use logical and bitwise operations (probably) to extract the two numbers.

Once you know where your playpen is located and SPIMbot is armed with bunnies, SPIMbot can drive above the playpen, and place bunnies into the playpen. You can control SPIMbot's bunny placement mechanism by writing to the `PUT_BUNNIES_IN_PLAYPEN` memory mapped I/O address with a non-negative integer. This mechanism will drop off the given number of bunnies into the playpen, based on the order that they were caught. **Note that if the provided number of bunnies is larger than the number of bunnies carried, or if the bot location is not within 5 pixels of the playpen location, this action will fail entirely (neither bunnies carried, nor bunnies in the playpen will change).**

See the online SPIMbot documentation at the link from the beginning of this handout for more details on using `PLAYPEN_LOCATION` and `PUT_BUNNIES_IN_PLAYPEN` memory mapped I/O addresses.

Bunny Move interrupt

SPIMbot can use the `BUNNY_MOVE` interrupt to detect when a bunny has hopped to a different location.

The bunny move interrupt will fire whenever a bunny hops to a new location. Note that the BunniesInfo struct will not be updated automatically when a bunny moves, and will only be updated whenever you write its address to `SEARCH_BUNNIES`. Therefore, after a bunny move interrupt has fired, you will have to use `SEARCH_BUNNIES` memory mapped I/O address to find out exactly which bunny triggered the interrupt.

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

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.

[3] Obviously a robot can get weary. Why would you even question that?

Grading

You will be graded based on if you have put 100 Zentners of bunnies into the playpen within the time limit of 10,000,000 cycles (the points displayed in the bottom left of the map should be ≥ 100). You will also be graded for acknowledging the `BUNNY_MOVE` interrupt so you **must use the `BUNNY_MOVE` interrupt**.

Be sure to check the terminal output with the `-debug` flag to make sure your bot is successfully putting bunnies in the playpen. You can see your score will increase by the weight of the bunnies placed in the playpen. **To receive full credit you must put at least 100 Zentners of bunnies into the playpen.**

The location of the playpen 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 15 bunnies at random locations with different weights at the beginning, and SPIMbot will begin with 100 carrots. The bunnies *will* move, and more bunnies will spawn throughout the game.
- You will need to put 100 Zentners of bunnies into the playpen.
- `num_bunnies` in the BunniesInfo struct will have a range of [0, 15] for this part.
- The `remaining_cycles` int in the Bunny struct simply indicates how many cycles are remaining before a bunny hops. This may come in handy if you don't want to waste time driving to a bunny that is about to move!
- 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!