COMP1521 26T1

Week 2 Lecture 1

MIPS: Control and Data

Week 2 Revision Session TODAY

- Time: Week 2 Monday, straight after lecture, **5-7pm**
- Location: Online Moodle (BlackBoard Collaborate)
- If you are on campus feel free to sit in **Kora lab J17** for the online session

Content:

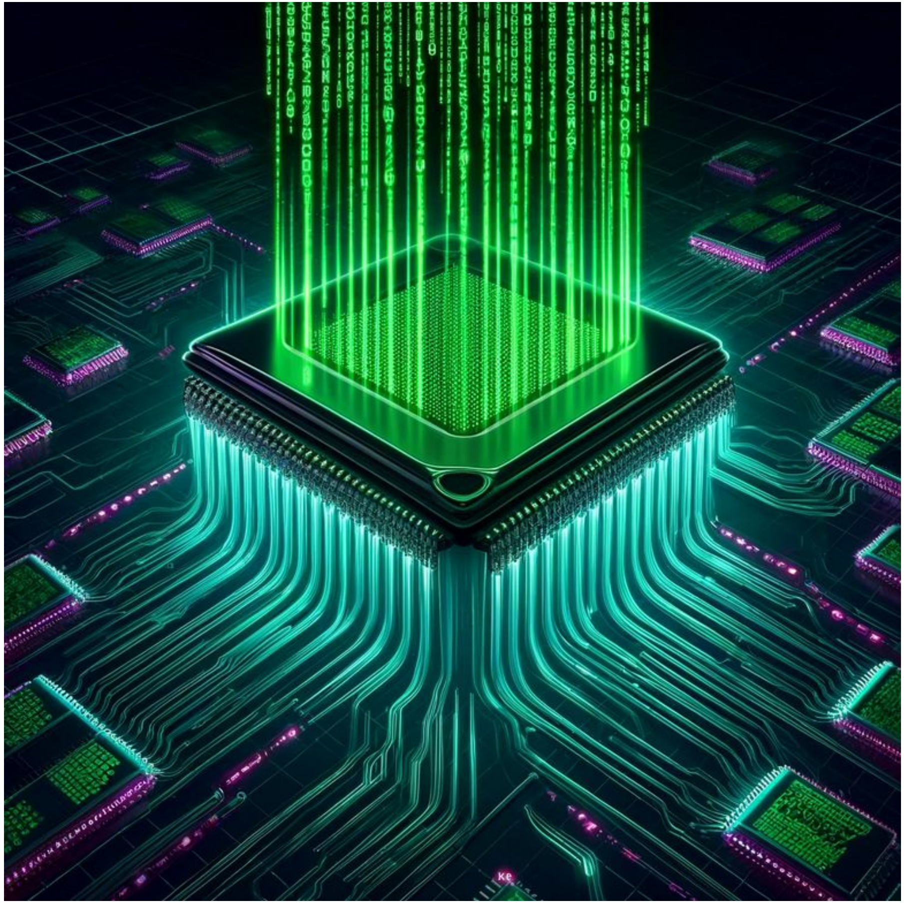
- C revision and recursion lab style based questions
- Can also get help with regular lab week 1

See forum post for more details:

https://discourse01.cse.unsw.edu.au/26T1/COMP1521/t/c-revision-session/22

Today’s Lecture

- Debugging with mipsy on the command line
- Recap Lecture 2
- Loops
  - needed for this week’s lab
  - for, while
  - break and continue
- Data and Memory
  - Global variables
  - Pointers



Command Line Mipsy Debugging

- You can run mipsy in interactive mode on the command line
- Type h for help to find out more

```
$ 1521 mipsy
[mipsy] h
COMMANDS:
load <files> -- {args}        - load a MIPS file to run
run                           - run the currently loaded program until it finishes
step [times] [subcommand]     - step forwards or execute a subcommand
Etc
Etc
Etc
```

Command Line Mipsy Debugging Example

```
$ 1521 mipsy
[mipsy] load calc.s
success: file loaded

[mipsy] step 6

start:
0x80000000 kernel [0x3c1a0040]    lui    $k0, 64
0x80000004 kernel [0x375a0000]    ori    $k0, $k0, 0
0x80000008 kernel [0x0340f809]    jalr   $ra, $k0

main:
0x00400000 7  [0x20080002]    addi   $t0, $zero, 2        #  li    $t0, 2
0x00400004 8  [0x20090003]    addi   $t1, $zero, 3        #  li    $t1, 3
0x00400008 9  [0x71095002]    mul    $t2, $t0, $t1        #  mul   $t2, $t0, $t1

[mipsy] print $t2
success: $t2 = 6
```

Recap of the Last Lecture

- We can write more fun assembly now!
- We can use `syscall` to ask the mipsy operating system to
    - read things in and print things out
- We can convert constructs like **loops** and **conditionals** into goto and branch

Recap: Putting data in registers

- **li** (load immediate) is loading a **fixed value** into a register
  - `li $t0, 7`
- **la** (load address) is for loading a **fixed address** into a register
  - remember, labels really just represent addresses!
  - `la $t0, my_label`
- **move** is for copying value from a **register** into another register
  - `move $t0, $t1`

Recap Exercise: Translate to MIPS

```c
// What does this code do?
int main(void) {
loop:
    printf("Forever and ever!\n");
    goto loop;
    return 0;
}
```

Recap Exercise:

```c
int main(void){
    int n;
    printf("Enter a number: ");
    scanf("%d",&n);
    if (n > MIN && n <= MAX){
        printf("In range\n");
    } else {
        printf("Out of range\n");
    }
    return 0;
}
```

Translate to Simplified C

Then to MIPS

Recap: Simplifying while loops

```c
int i = 0;
while (i < 10) {
    printf("%d\n", i);
    i++;
}
```

```c
int i;
loop_i_to_10__init:
    i = 0;
loop_i_to_10__cond:
    if (i >= 10) goto loop_i_to_10__end;

loop_i_to_10__body:
    printf("%d", i);
    putchar('\n');
loop_i_to_10__step:
    i++;
goto loop_i_to_10__cond;
loop_i_to_10__end:
```

Exercise: Debug my buggy MIPS implementation of this

Exercise: Sum 100 squares

Convert to simplified C
then to MIPS

```c
int sum = 0;
for (int i = 1; i <= 100; i++) {
    sum += i * i;
}
printf("%d\n", sum);
```

Sidenote: C break/continue

`break` can be used in a loop to completely exit the loop.
The loop condition here makes this look like an infinite loop:

```
while (1) {
    int c = getchar();
    if (c == ‘n’) break;
}
```

but `break` means it’s possible for the loop to be exited.

In simplified C/MIPS, a `break` is really just equivalent to going to the loop’s end label.

Sidenote: C break/continue

```
while (1) {
    int c = getchar();
    if (c == ‘n’) break;
}
```

```
    int c;
get_char_loop:
    c = getchar();
if_n:
    if (c == ‘n’)
goto get_char_loop_end;
end_if_n:
    goto get_char_loop;
get_char_loop_end:
```

Sidenote: C break/continue

`continue` can be used to proceed to the next iteration of a loop. This would be a (terrible) way to print even numbers:

```c
int i;
for (i = 0; i < 10; i++) {
    if (i % 2 != 0)
        continue;
    printf("%d\n", i);
}
```

In simplified C/MIPS, a `continue` is really just equivalent to going to the loop’s step label.

Sidenote: C break/continue

```c
int i;
for (i = 0; i < 10; i++) {
    if (i % 2 != 0)
        continue;
    printf("%d\n", i);
}
```

```c
int i = 0;
while (i < 10) {
    if (i % 2 != 0)
        continue;
    printf("%d\n", i);
    i++;
}
```

Is this while loop equivalent to the for loop?

Sidenote: C break/continue

```c
int i;
for (i = 0; i < 10; i++) {
    if (i % 2 != 0)
        continue;
    printf("%d\n", i);
}
```

```c
int i;
loop_i_to_10__init:
    i = 0;
loop_i_to_10__cond:
    if (i >= 10) goto loop_i_to_10__end;
loop_i_to_10__body:
    if (i % 2 != 0) goto loop_i_to_10__step;
    printf("%d", i);
    putchar('\n');
loop_i_to_10__step:
    i++;
    goto loop_i_to_10__cond;
loop_i_to_10__end:
```

MIPS: Data and Memory

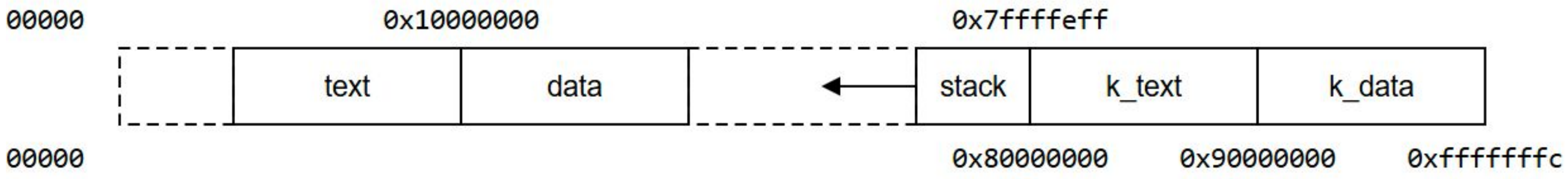
How do we store/use interesting data?

How does the data segment really work?

How do we:

- Store simple types like chars and ints?
- Store and increment a global variable?
- Work with pointers?
- Work with 1D arrays?
- Work with 2D arrays??
- C Structs !?

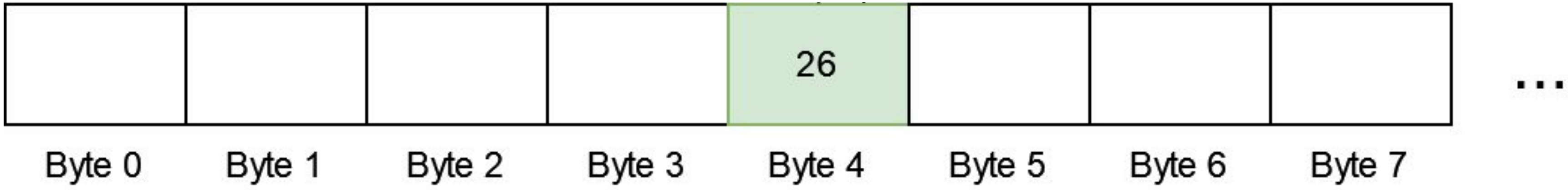
MIPS Memory Layout



- MIPS addresses are 32 bits (4 bytes)
- Notes:
  - There is no heap like in C, but the data segment can expand (not needed in this course except maybe challenge exercises)
  - The text segment is the only segment that is executable
  - The text segment is writable, unlike a real system

Memory Addresses

- Data will live at an address in memory
- We can think of it like a large 1D array
- Each byte (usually 8 bits) has a unique **address**
  - So memory can be thought of as one large array of bytes
  - Address = index into the array
  - Eg. The byte at address 4 below has the value 26



				26				…
Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	

Recall Common Data Types in C

- What are the sizes in bytes of data we commonly used in C on our system?
  - char = ? bytes
  - int = ? bytes
  - double = ? bytes
  - pointer = ? bytes
- We can find out using sizeof!
- These are the same as in MIPS except pointers since MIPS has 4 bytes.
  - MIPS is a 32 bit platform instead of 64 bit platform.

Representing Common Data types in MIPS

Data Type	Size in Bytes	Location
`char`	1	Memory, Register
`int`	4	Memory, Register
`pointer`	4 (32 bit architectures)	Memory, Register
`array`	sequence of bytes, elements accessed by calculated index	Memory
`struct`	sequence of bytes, elements accessed by calculated offset	Memory

Local vs Global variables

How do we represent local variables in MIPs?
How do we represent global variables in MIPs?

```c
int global_var = 0;
int main(int argc, char *argv[]) {
  int local_var = 0;
  return 0;
}
```

Local vs Global variables in MIPS

Local Variables:

- Stored in **registers** for speed when possible
  - Single values that fit in a single register such as
    - int, char, pointers
  - Not stored in registers if their address is needed
  - Limited number of registers
- Otherwise stored on **stack**
  - We'll revisit this next week

Global Variables:

- Stored in the in **data segment**

Initialising Global Data

We can use directives to initialise memory in the .data section

```
    .word 42         # initialises a 4 byte value to 42
    .half  7         # initialises a 2 byte value to 7
    .byte 'a'        # initialises a 1 byte value to ‘a’
    .asciiz “hello” # initialises a string
```

We can also just ask for some memory without initialising it (typically we prefer to initialise it)

```
    .space 8         # sets aside 8 uninitialised bytes
```

But where does this live and how can we access it?

Global Data: C

```c
int a = 42;          // .word 42
short b = 7;         // .half 7
char c = 'a';        // .byte ‘a’
char d[6] = "hello"; // .asciiz "hello"
char space[8];       // .space 8
int main (void) {
    // This program does nothing :)
    return 0;
}
```

Accessing Memory

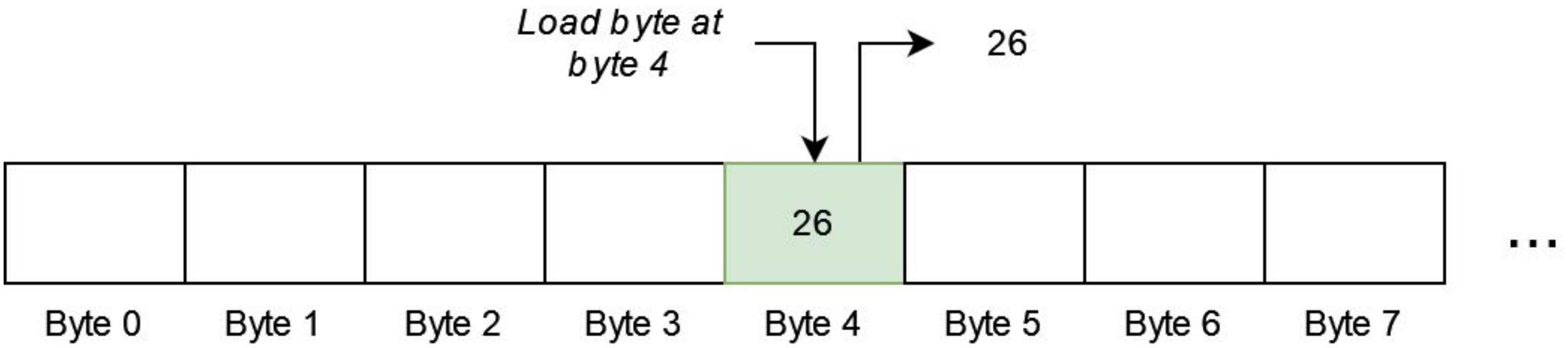
- Loading data:
  - To perform computations, data must be transferred from memory into the CPU registers

- Storing data:
  - Modified data must be written back from the CPU registers to memory

- We can load and store whole bytes (not bits)

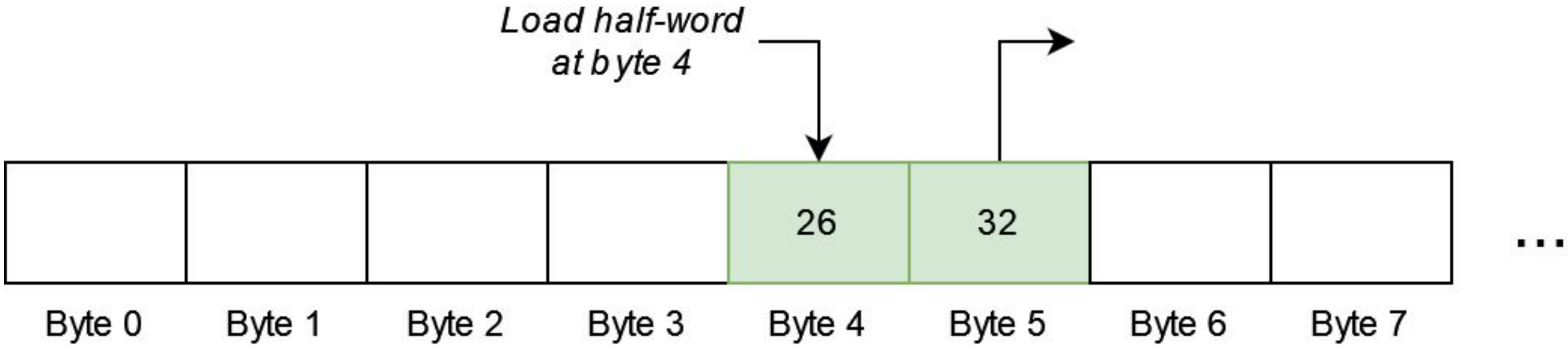
Loading from Memory

- E.g Loading the byte from address 4 would load the byte containing 26 in the specified register



Bytes, half-words, words

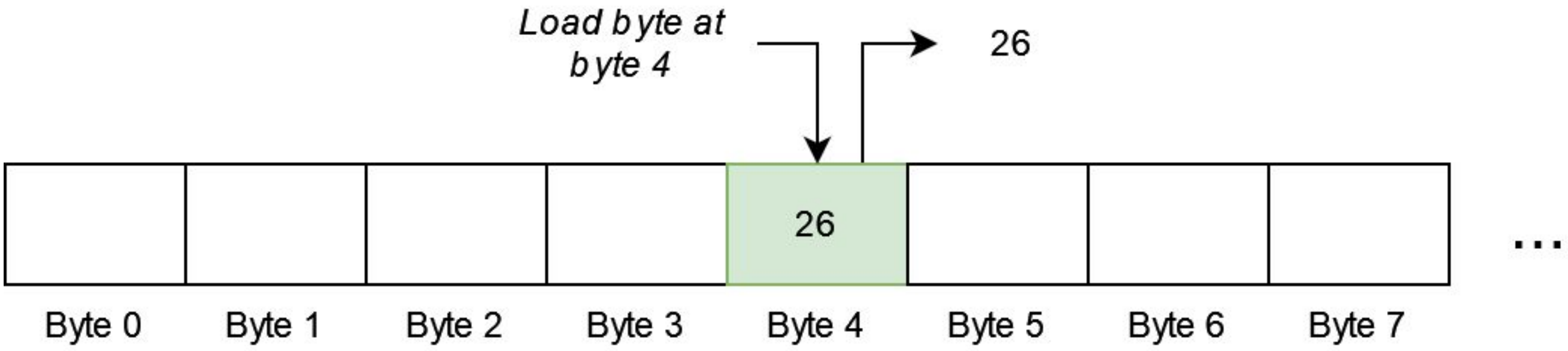
- Typically, small groups of bytes can be loaded/stored at once
- E.g. in MIPS:
  - 1-byte (a byte) loaded/stored with ……………………………. **`lb/sb`**
  - 2-bytes (a half-word) loaded/stored with…………………… **`lh/sh`**
  - 4-bytes (a word) loaded/stored with……………………….. **`lw/sw`**



Working with Memory Addresses in MIPS

- Memory addresses in load/store instructions are the sum of:
  - Value in a specific register
  - And a 16-bit constant (often 0)

    -
      ```
      la      $t0, 4
      lb      $t1, 0($t0)
      ```



Load byte at byte 4

26

26

Byte 0 Byte 1 Byte 2 Byte 3 Byte 4 Byte 5 Byte 6 Byte 7 ...

Loading/Storing a byte from/to Memory

Loading a byte (no labels)

```
        .text
main:
        la      $t1, 0x10010000
        lb      $t0, 0($t1)

        .data
        .byte 'Q'
```

Storing a byte (no labels)

```
        .text
main:
        li      $t0, 'y'
        la      $t1, 0x10010000
        sb      $t0, 0($t1)
```

Note: To save space, I have omitted return 0 equivalent code.

Labels

- We do NOT want to keep track of the memory locations and hard code them ourselves.
- What if we add/remove variables as we develop our code?
- We use **labels** which are used by mipsy to represent the **memory locations**.

Loading/Storing a byte from/to Memory

loading a byte with labels

```
        .text
main:
        la      $t1, my_letter
        lb      $t0, 0($t1)

        .data
my_letter:
        .byte 'Q'
```

storing a byte with labels

```
        .text
main:
        li      $t0, 'y'
        la      $t1, my_letter
        sb      $t0, 0($t1)

        .data
my_letter:
        .space 1
```

Loading/Storing a word from/to Memory

loading a word

```
        .text
main:
        la      $t1, my_word
        lw      $t0, 0($t1)

        .data
my_word:
        .word 10
```

storing a word

```
        .text
main:
        li      $t0, 9
        la      $t1, my_word
        sw      $t0, 0($t1)

        .data
my_word:
        .space 4
```

Mipsy short cuts

- We can just write the constant memory address locations
- We don’t need to load them to another register

```
        .text
main:
        li $t0, 42
        la $t1, my_label
        sw $t0, 0($t1)
        .data
my_label:
        .word 0
```

=

```
        .text
main:
        li $t0, 42
        sw $t0, my_label

        .data
my_label:
        .word 0
```

Addressing in assembly

```
sb $t0, 4($t1)  # store $t0 4 bytes past address in $t1
sb $t0, 0($t1)  # store $t0 0 bytes past address in $t1
sb $t0, ($t1)   # store $t0 at address in $t1 (same as above)

sb $t0, x       # store $t0 in byte at address labelled x
sb $t1, x+15    # store $t1 15 bytes past address labelled x
sb $t2, x($t3)  # store $t2 $t3 bytes past address labelled x
```

Demo program - global_increment.c

- Let’s write a program with a global variable and increment it

```c
#include <stdio.h>

int global_counter = 0;

int main(void) {
    // Increment the global counter.
    global_counter++;
    printf("%d", global_counter);
    putchar('\n');
}
```

Exercise - global_increment_char.c

- Modify our last MIPS program to implement this

```
#include <stdio.h>

char global_letter = ‘A’;

int main(void) {
    // Increment the global letter.
    global_letter++;
    printf("%c", global_letter);
    putchar('\n');
}
```

Alignment

- C standard and MIPS requires simple types of **size N** bytes to be stored only at addresses which are **divisible by N**
  - A 4 byte int , must be stored at address divisible by 4
  - An 8 byte double, must be stored at address divisible by 8
  - Compound types (arrays, structs) must be aligned so their components are aligned
- Example:
  - If you are using lw, or sw, you must be loading/storing the 4 bytes from/to an address divisible by 4
- Tip: hex numbers ending in 0, 4, 8, C are divisible by 4

Alignment problem demo - align_broken.s

```
        .text
main:
        li      $t0,    99
        sw      $t0,    g       # g = 99

        li      $v0, 0          # return 0
        jr      $ra

        .data
f:      .asciiz "hello"         # char f[] = "hello";
g:      .space 4                # int g;
```

Alignment Solutions

Reorder variables

```
    .data
g:  .space 4                # int g;
                            # Not always practical or possible
                            # Works fine in this situation
f:  .asciiz "hello"         # char f[] = "hello";
```

Padding with `.space`

```
    .data
f:  .asciiz "hello"         # char f[] = "hello";
    .space 2                # padding - we have to calculate the space
                            # ourselves. Error prone. May break if we modify
                            # our string “hello”
g:  .space 4                # int g;
```

Alignment Solutions

Fix with **.align**

```
   .data
f: .asciiz "hello"      # char f[] = "hello";
   .align 2             # align next object on 4 byte address (2 pow 2)
                        # (2 to the power of 2) less error prone
g: .space 4             # int g;
```

Use **.word**

```
   .data
f: .asciiz "hello"      # char f[] = "hello";
g: .word 0              # int g; MIPS will align .word automatically
```

Pointer Example

```c
int answer = 42;


int main(void) {
    int i;
    int *p;
    p = &answer;
    i = *p;
    printf("%d\n", i);
    *p = 27;
    printf("%d\n", answer);
    return 0;
}
```

What would this print?

How could we write this in MIPS?

What did we learn today?

- MIPS Control
  - recap of if statements
  - loops
- MIPS Data
  - Defining global data
  - Loading and storing global data
    - ints, chars, pointers
  - Data alignment
- Next lecture:
  - 1d arrays, 2d arrays, structs

Additional Resources

- MIPS Control Notes MIPS Control Code
- MIPS Data Notes MIPS Data Code

Feedback Please!

Your feedback is valuable!

If you have any feedback from today's lecture, please follow the link below or use the QR Code.

Please remember to keep your feedback constructive, so I can action it and improve your learning experience.

https://forms.office.com/r/VbzTZmHCyL

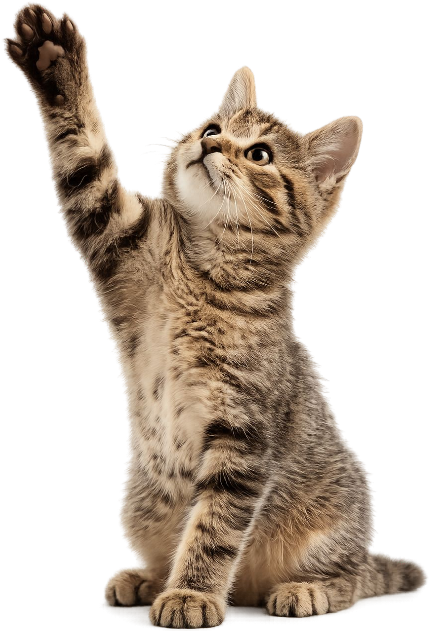
Reach Out

Content Related Questions:

Forum

Admin related Questions email:

cs1521@cse.unsw.edu.au



Student Support | I Need Help With...

My Feelings and Mental Health Managing Low Mood, Unusual Feelings & Depression	**Mental Health Connect**	student.unsw.edu.au/**counselling** Telehealth	**In Australia Call Afterhours UNSW Mental Health Support Line** — 1300 787 026 5pm-9am
	Mind HUB	student.unsw.edu.au/**mind-hub** Online Self-Help Resources	**Outside Australia Afterhours 24-hour Medibank Hotline** — +61 (2) 8905 0307
Uni and Life Pressures Stress, Financial, Visas, Accommodation & More	**Student Support Indigenous Student Support**	student.unsw.edu.au/**advisors**	
Reporting Sexual Assault/Harassment	**Equity Diversity and Inclusion (EDI)**	edi.unsw.edu.au/**sexual-misconduct**	
Educational Adjustments To Manage my Studies and Disability / Health Condition	**Equitable Learning Service (ELS)**	student.unsw.edu.au/**els**	
Academic and Study Skills	**Academic Language Skills**	student.unsw.edu.au/**skills**	
Special Consideration Because Life Impacts our Studies and Exams	**Special Consideration**	student.unsw.edu.au/**special-consideration**	