2.1 - Week 2 - Applied - Theory

Objectives of this applied session

- To understand better the MIPS architecture.
- To be able to write simple MIPS programs and understand what happens when they are executed.
- To be able to read simple MIPS programs and understand what they do, and what is wrong with them (if anything).

Important

As we said for Applied 1 (we will repeat it once again) the purpose of Applied Sessions is not to give you the solutions to the exercises. Solutions will be released after all applied sessions have finished, so if all you want is the solutions, you can simply wait. The purpose of applied sessions is to make you think further about the material covered during the workshops and, in particular, to generate discussion about some of the more interesting and challenging concepts.

Since the applied sessions are only 2 hours you would get much more from the session if you prepare solutions to the exercises before you come to the class. The solutions do not need to be correct and you might still have many questions about how to do them. These (possibly incorrect) solutions and questions you bring are very valuable, as they will help generate lots of discussions and learning for you and for the other students.

Lastly, applied session sheets often contain too many exercises for the length of the session. This is done to provide you with more solutions to common exercises. Since there will not be enough time in the class to discuss all of them, coming prepared with questions for your TAs is recommended to get the most out of these classes.

Introduction to MIPS

Think about the particular decisions taken while designing the MIPS Architecture to answer the following questions:

Question 1

Why do you think is the stack segment designed, seemingly backwards, to extend into lower addresses as it grows? In particular, why do the heap and stack segments grow towards each other, rather than in the same direction?

No response

Question 2

MIPS provides 32 general-purpose registers for your programs to use, presumably because that’s the number that its designers thought best to supply. What decisions and tradeoffs do you think they made in arriving at this number? What is good/bad about a bigger number? And about a smaller one?

No response

Basic Instructions in MIPS

We will now test your familiarity with some basic instructions in MIPS and how they translate from Python.

Question 1 *Submitted Aug 1st 2022 at 9:15:31 pm*

As you may (should) know, `$v0` is the register that holds the system call code. Each call code has a specific function. These will be very useful through the semester

Use your MIPS reference sheet and match the call code with the function of that particular call code:

`$v0: 1`

`$v0: 4`

`$v0: 5`

`$v0: 9`

`$v0: 10`

Put the following functions in the order of the above system call codes

Print Integer

Print String

Take Input Integer

Allocate space

Exit program

Question 2 *Submitted Aug 1st 2022 at 9:15:41 pm*

What command is used to load an integer from the data section?

[✓ `lw`]

`sw`

- ○ `la`
- ○ `addi`

Question 3 *Submitted Aug 1st 2022 at 9:15:47 pm*

What command is used to store an integer back into a variable in the data section?

- ○ `lw`
- [✓ `sw`]
- ○ `la`
- ○ `addi`

Question 4 *Submitted Aug 1st 2022 at 9:19:21 pm*

What is the difference between `la` and `lw`?

`la` means to load address to register where `lw` means to load word stored in an address. `la` is usually used to print a string stored in a specific address in the register where `lw` is used to replace the data stored in the word in memory.

Question 5 *Submitted Aug 1st 2022 at 9:19:51 pm*

What instruction is used to add a register to an immediate value, and store this result in a register?

- ○ `disc`
- ○ `sum`
- [✓ `addi`]
- ○ `plus`

- ○ `add`

Question 6 *Submitted Aug 1st 2022 at 9:20:12 pm*

What register holds what information after a multiply (`mult`) and a divide (`div`) operation?

- ○ `mult` : hi - product, lo - overflow | `div` : hi - remainder, lo - quotient
- [✓ `mult` : hi - overflow, lo - product | `div` : hi - remainder, lo - quotient]
- ○ `mult` : hi - product, lo - overflow | `div` : hi - quotient, lo - remainder
- ○ `mult` : hi - overflow, lo - product | `div` : hi - quotient, lo - remainder

Faithful Translation - Sum of Two Numbers

Consider the following Python code:

```
print("Enter two integers: " )
integer1 = int(input( ))
integer2 = int(input( ))

integerSum = integer1 + integer2

print("Sum is",integerSum)
```

A translation from high-level code (say in Python) into MIPS is faithful if it is done by translating each line of code independently of the others (i.e., without optimizing the code by reusing the value of registers computed in previous instructions). While this can introduce considerable space/speed costs, it does make the translation easier, thus reducing the chances of committing mistakes. Since we do not want you to commit mistakes (and we want to be able to understand your code easily), we will often (but not always) ask you to do your translations faithfully.

Convert the program above into MIPS assembly code. We have already given you some comments to point out blocks which will be very useful for you.

i You can test your program by clicking "Run", or by running the following command in the console: `java -jar Mars4_5.jar IntegerSum.asm`

Understanding how registers change

Consider the following (glaringly uncommented) MIPS code:

```
        .text

main:   addi $t0 , $0 , 62
        addi $t1 , $0 , -28
        addi $t2 , $0 , 20
        addi $t3 , $0 , 3
        add $t4 , $t1 , $t0
        sub $t4 , $t4 , $t2
        mult $t3 , $t4
        mflo $t5
        div $t5 , $t4
        mflo $t6
        div $t2 , $t3
        mfhi $t6
```

1. Prepare a table (feel free to use the excel file from the scaffold) where each column corresponds to one of the following registers: PC, HI, LO, $0, $t0, $t1, $t2, $t3, $t4, $t5 and $t6. For each instruction in the MIPS code, write in the appropriate table entry:

- The value of the program counter, assuming that for line 2, PC has the value 0x00400000.
- The value of the registers that participate in the instruction, and
- A mark (e.g., *), to indicate a register's content has changed.

2. Comment each line of the code in a way that makes it more meaningful to you.

Multiplication and Division

Assume a student is translating into MIPS the following part of a Python program (where x,y,z are global variables already defined):

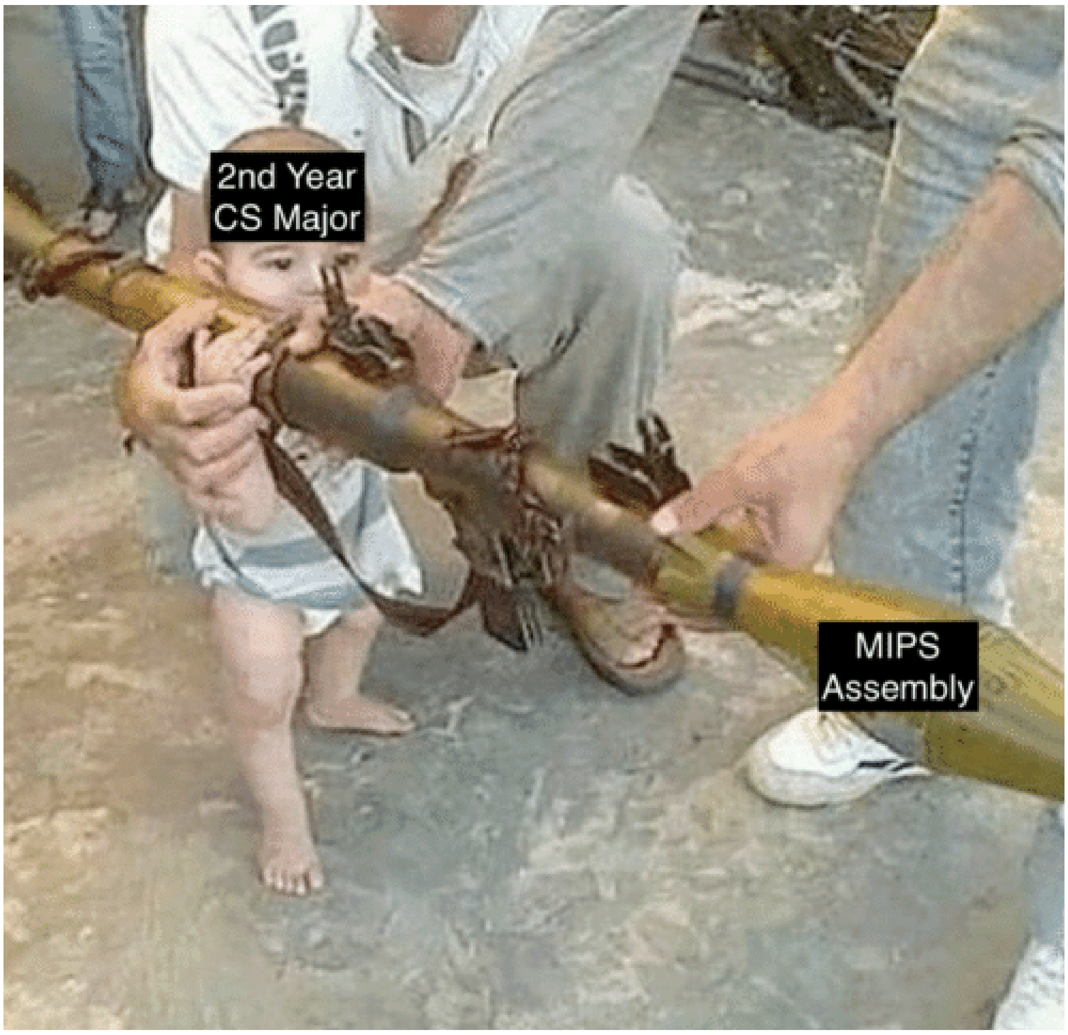
```
x = y*z
x = x%4

print(x)
```

and the student obtains the given (uncommented and incorrect) MIPS code. Identify the incorrect lines of code, explain why they are wrong and provide the corrected ones.

Hooray (v2.0 - Green tick enabled)

We've reached the end of the introduction to MIPS and you have a powerful weapon in your arsenal.



We hope you understand a little bit about how MIPS architecture works and how basic mathematical operations can be translated in MIPS.

Make sure you attend the optional applied session for some more practice on these commands so you can effectively do your assignment well.

Additionally, try the following exercises post-class (if you dare!)

Take a string input? What?

Your friend is trying to faithfully translate the following python code into MIPS:

```python
str_label = "Enter your name"
a_label = "Hello there "
b_label = " , my friend , "
name_str = input(str_label )

print( a_label , name_str )
```

However, they are struggling to understand why the given translation is wrong and ask you to look into their MIPS code.

Expanding your knowledge of MIPS

Consider the code that was given to you in the last exercise, and try to answer these questions below

Question 1 *Submitted Aug 1st 2022 at 9:20:52 pm*

1. The MIPS code translation was faithful to the python code.

True [✓ False]

Question 2 *Submitted Aug 1st 2022 at 9:21:06 pm*

The MIPS code produces the same output as the given python code.

True [✓ False]

Question 3

What is the output of the MIPS version?

No response

Question 4

What happens if you change the value at line 14 from 10 to 5 and why?

No response

Question 5

How can you fix your friend's MIPS code and comment it for better code readability?

No response

Extracting the MSB

Assume you have an integer already stored in global variable bits. Using only bitwise and shift MIPS operations, show at least one way of:

1. Extracting and printing the most significant bit from bits. That is if the MSB indicates the number is positive it prints 0, and if it is negative it prints 1.
2. Extracting and printing the bit in position N, where N=0 for the least significant bit, and 31 for the most significant one.

You can test your program by running the following commands in the terminal:

```
java -jar Mars4_5.jar Ex6_1.asm
```

```
java -jar Mars4_5.jar Ex6_2.asm
```