Von Neumann and MIPS

References:

1) MIPS_Vol2.pdf

2) Intro to MIPS Assembly Language Programming

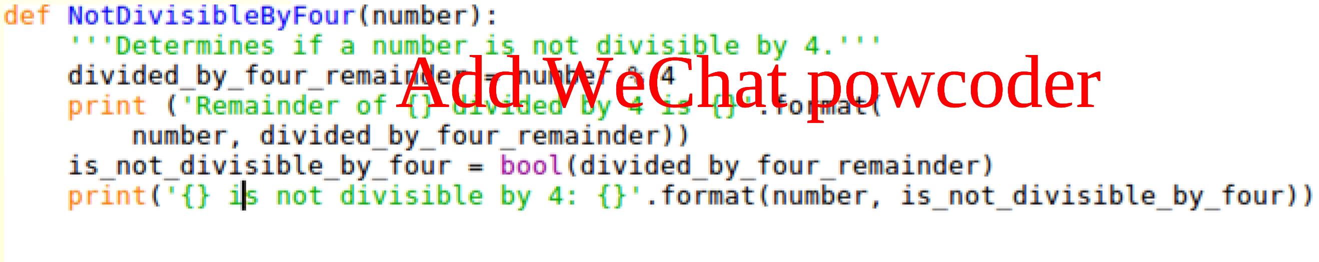
Reminder

- Any lab grading questions must be sent only to TA Kevin. You can CC me in the email

A Simple Program(Python) - Not Divisible by 4

Determines if a number is not divisible by 4

- Finds remainder of number divided by 4
- If remainder exists then number is not divisible by 4

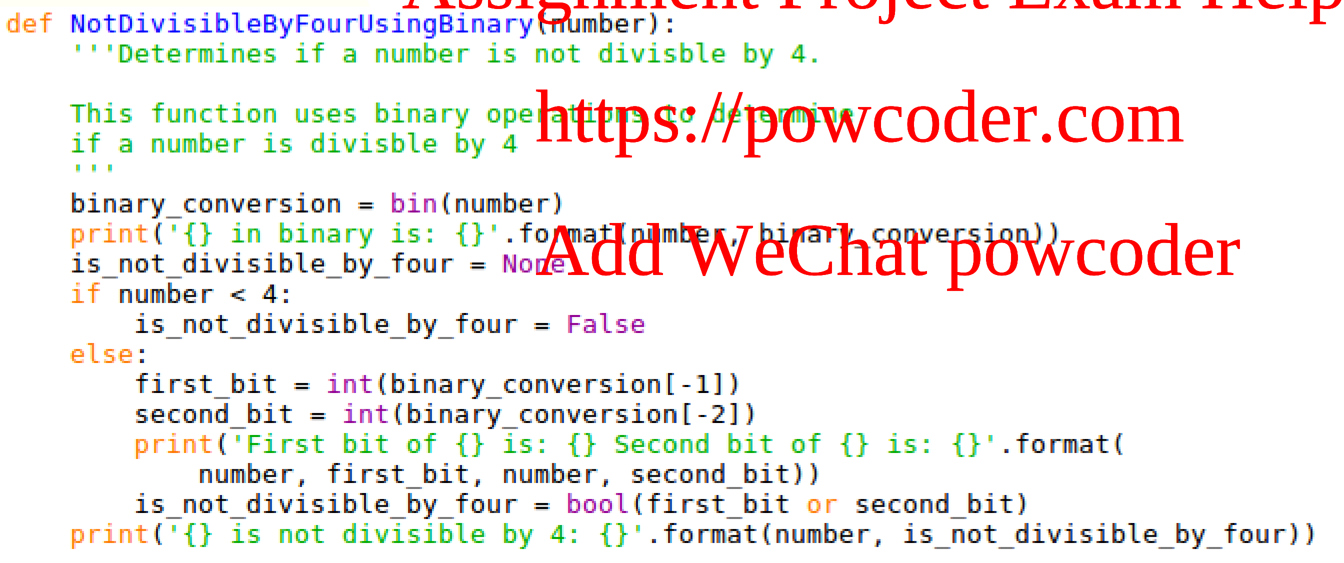


```
def NotDivisibleByFour(number):
    '''Determines if a number is not divisible by 4.'''
    divided_by_four_remainder = number % 4
    print ('Remainder of {} divided by 4 is {}'.format(
        number, divided_by_four_remainder))
    is_not_divisible_by_four = bool(divided_by_four_remainder)
    print('{} is not divisible by 4: {}'.format(number, is_not_divisible_by_four))
```

A Simple Program(Python) - Not Divisible by 4

Determines if a number is not divisible by 4

- Converts number to binary
- Checks to see if bit 0 or bit 1 is 1



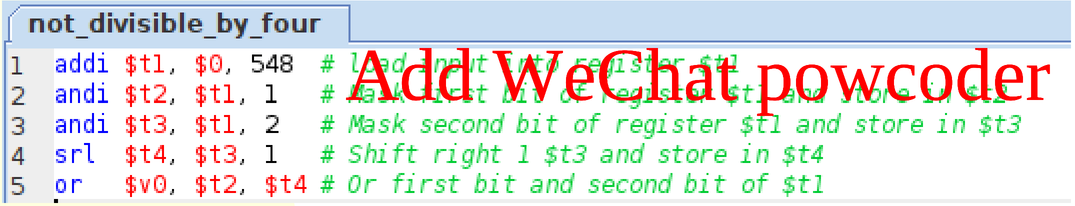
```
def NotDivisibleByFourUsingBinary(number):
    '''Determines if a number is not divisble by 4.

    This function uses binary operations to determine
    if a number is divisble by 4
    '''
    binary_conversion = bin(number)
    print('{} in binary is: {}'.format(number, binary_conversion))
    is_not_divisible_by_four = None
    if number < 4:
        is_not_divisible_by_four = False
    else:
        first_bit = int(binary_conversion[-1])
        second_bit = int(binary_conversion[-2])
        print('First bit of {} is: {} Second bit of {} is: {}'.format(
            number, first_bit, number, second_bit))
        is_not_divisible_by_four = bool(first_bit or second_bit)
    print('{} is not divisible by 4: {}'.format(number, is_not_divisible_by_four))
```

Decimal	Binary
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111

A Simple Program (Assembly) - Not Divisible by 4

Determines if a number is not divisible by 4

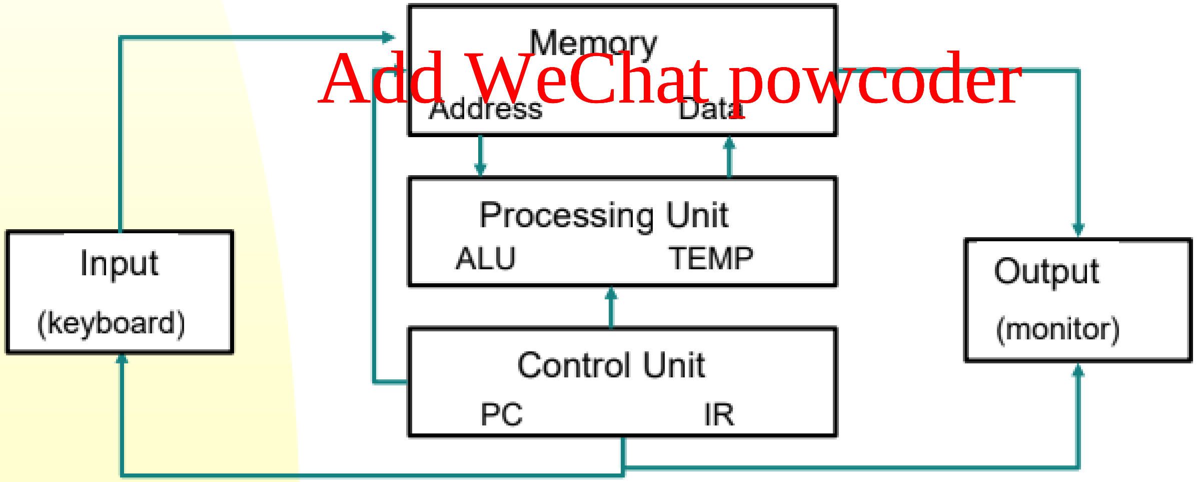
- Checks to see if bit 0 or bit 1 is 1



```
addi $t1, $0, 548  # load input into register $t1
andi $t2, $t1, 1   # Mask first bit of register $t1 and store in $t2
andi $t3, $t1, 2   # Mask second bit of register $t1 and store in $t3
srl  $t4, $t3, 1   # Shift right 1 $t3 and store in $t4
or   $v0, $t2, $t4 # Or first bit and second bit of $t1
```

The von Neumann Model

- Memory: holds both data and instructions
- Processing Unit: carries out the instructions
- Control Unit: sequences and interprets instructions
- Input: external information into the memory
- Output: produces results for the user



von Neumann Model - Memory

- Each location has an address and contents
  - Address: bit pattern that uniquely identifies a memory location
  - Contents: bit pattern stored at a given address.
  - Analogy: P.O. boxes have fixed numbers, but changing contents.
- Address Space:
  - The total number of memory locations (“boxes”) available.
  - E.g., a 28-bit address provides an address space of 2^28 locations.
  - MIPS has an address space of 2^32 locations - i.e. it uses a 32-bit address.

von Neumann Model – Memory (2)

- Addressability (Byte vs. Word):
  - a **word** is the basic unit of data used by the processing unit, often multiple bytes;
  - frequently, an instruction must store or retrieve an entire word with a single memory access.
  - Addressability refers to the number of bytes of memory referenced by a given address.
  - **Byte Adressable**

von Neumann Model – Processing Unit

- Does the actual work!
  - Can consist of many sub-units, each specializing in one complex function.
  - At a minimum, has Arithmetic & Logic Unit (ALU) and General Purpose Registers (GPRs)

von Neumann Model – Processing Unit (2)

- ALU
  - Performs basic operations: add, subtract, and, not, etc.
  - Generally operates on whole words of data.

  - Some can also operate on subsets of words

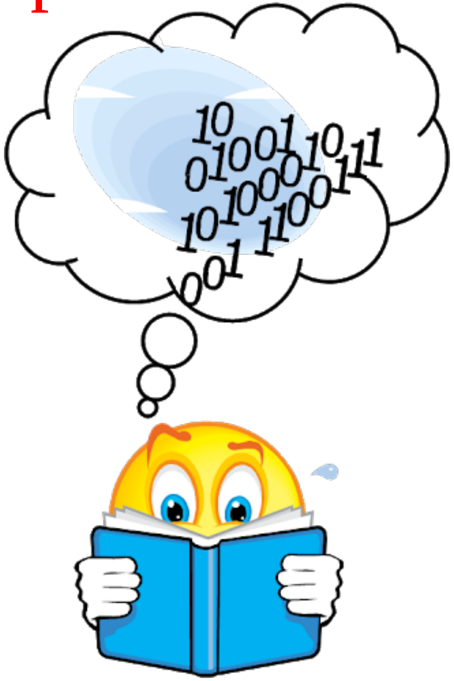
- Registers

  - Fast “on-board” memory for a small number of words.

  - Invaluable for intermediate data storage while processing
  - Close to the ALU (much faster access than RAM).

von Neumann Model – Control Unit

- The control unit coordinates all actions needed to execute the instruction
  - It fetches & **decodes** each instruction, and sets up the appropriate inputs for the Memory, Processing, and I/O units as required.
  - It communicates with memory via the Program Counter (PC) and Instruction Register (IR)
- PC (aka Instruction Pointer)
  - Holds the address of the next instruction to be fetched.
- IR (Instruction Register)
  - Holds the instruction currently being executed.
  - This can be a single word, or multiple words.



von Neumann Model – Input/Output

- Generically known as peripherals - not in the sense that they matter less, but because they are external to the CPU.
- This means we will have to develop mechanisms for autonomous devices to communicate with each other - more on this later.