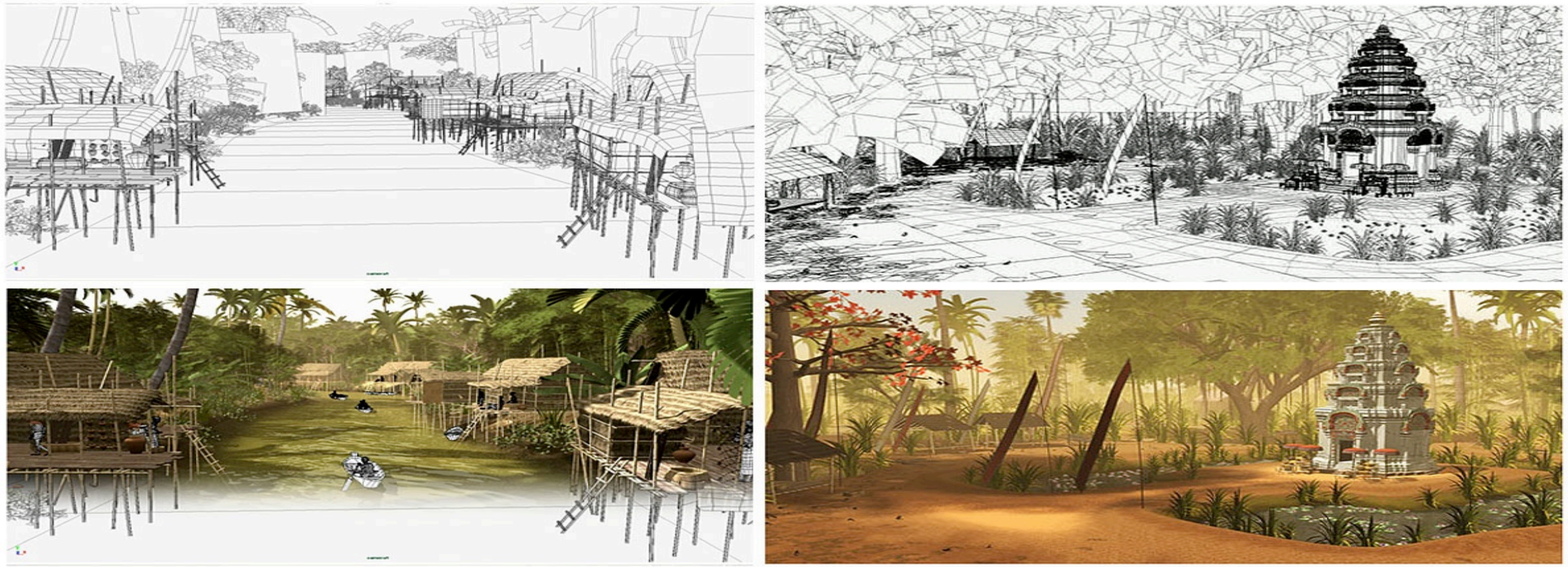
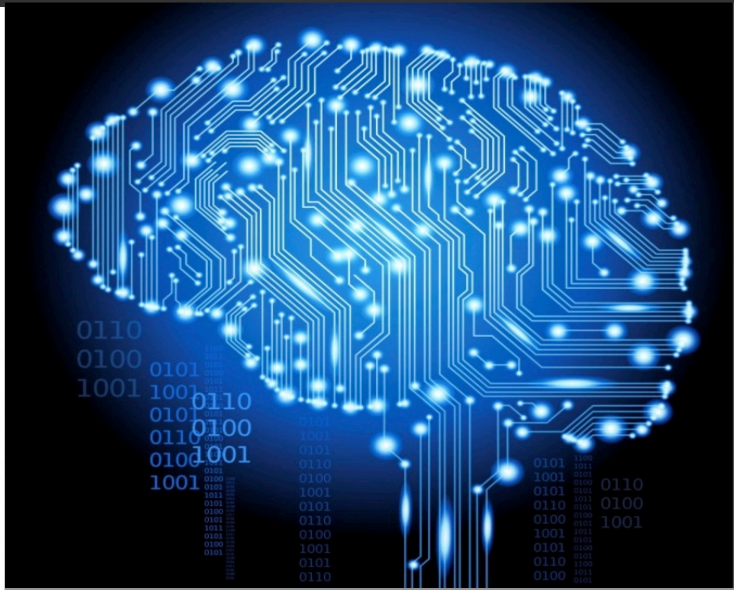
MONASH University

Information Technology

FIT1008/2085
MIPS – Selection

Prepared by:
Maria Garcia de la Banda
Revised by A. Aleti, D. Albrecht, G. Farr, J. Garcia and P. Abramson





Where are we up to?

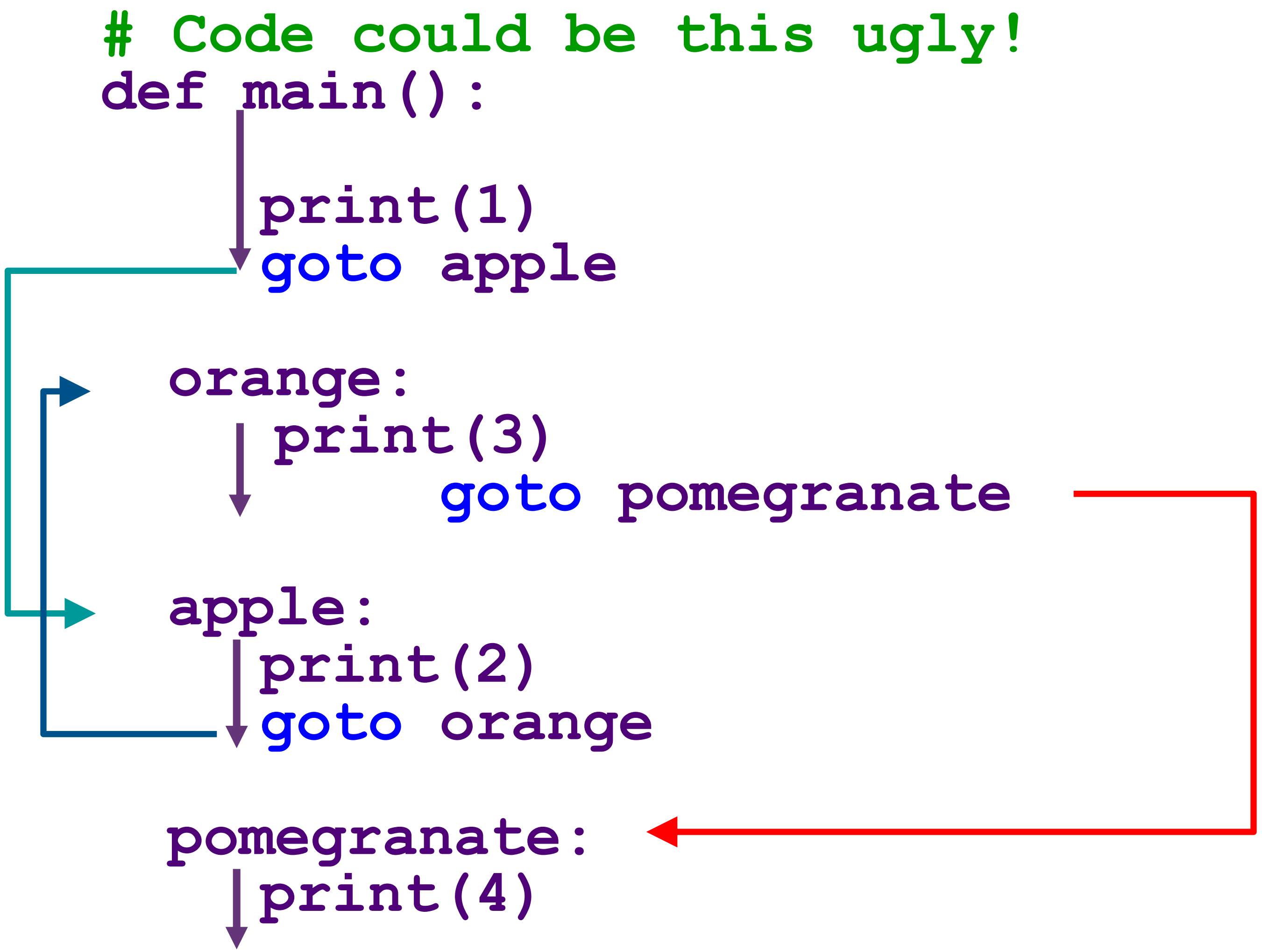
- **We now know the MIPS R2000 architecture**
  - 32 general purpose registers
  - Special purpose registers (HI, LO, PC, IR, etc)
  - ALU
  - Memory segments (text, data, heap, stack)
- **Understand the fetch-decode-execute cycle**
- **Able to use assembler directives**
- **Can program in assembly using many MIPS instruction set (maths, lw/sw, syscall, jumps, bitwise, shifts)**

Learning objectives for this lecture

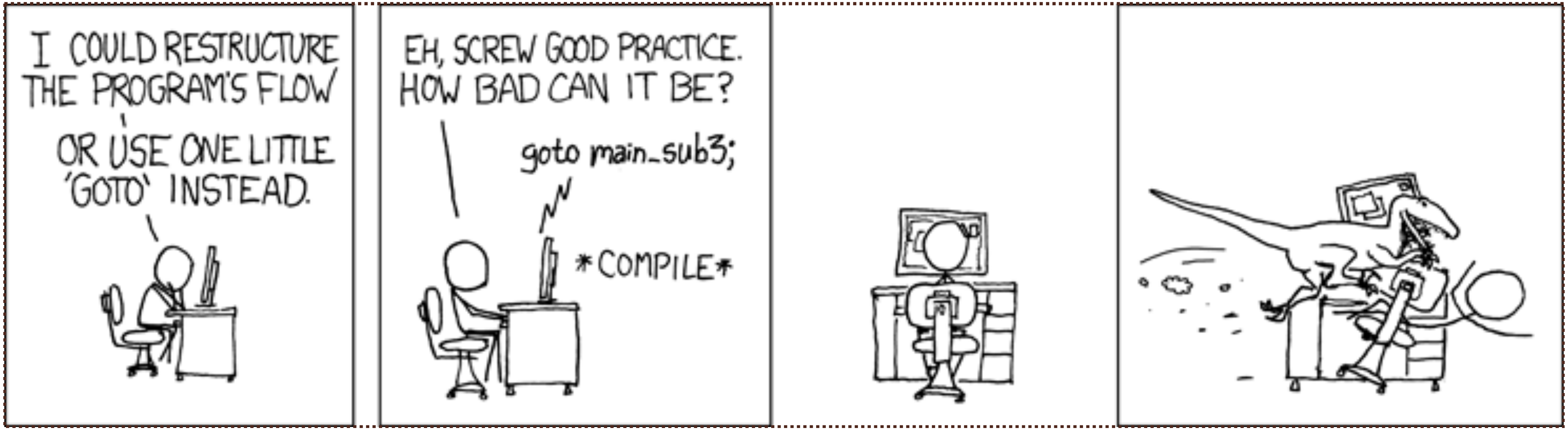
- To learn about MIPS conditional control transfer instructions
- To learn about MIPS comparison instructions
- To be able to use them to translate simple selection: `if-then`
- To learn what to do when I ask you to perform a faithful translation

Reminder: unconditional control transfer

Remember: If Python had goto …



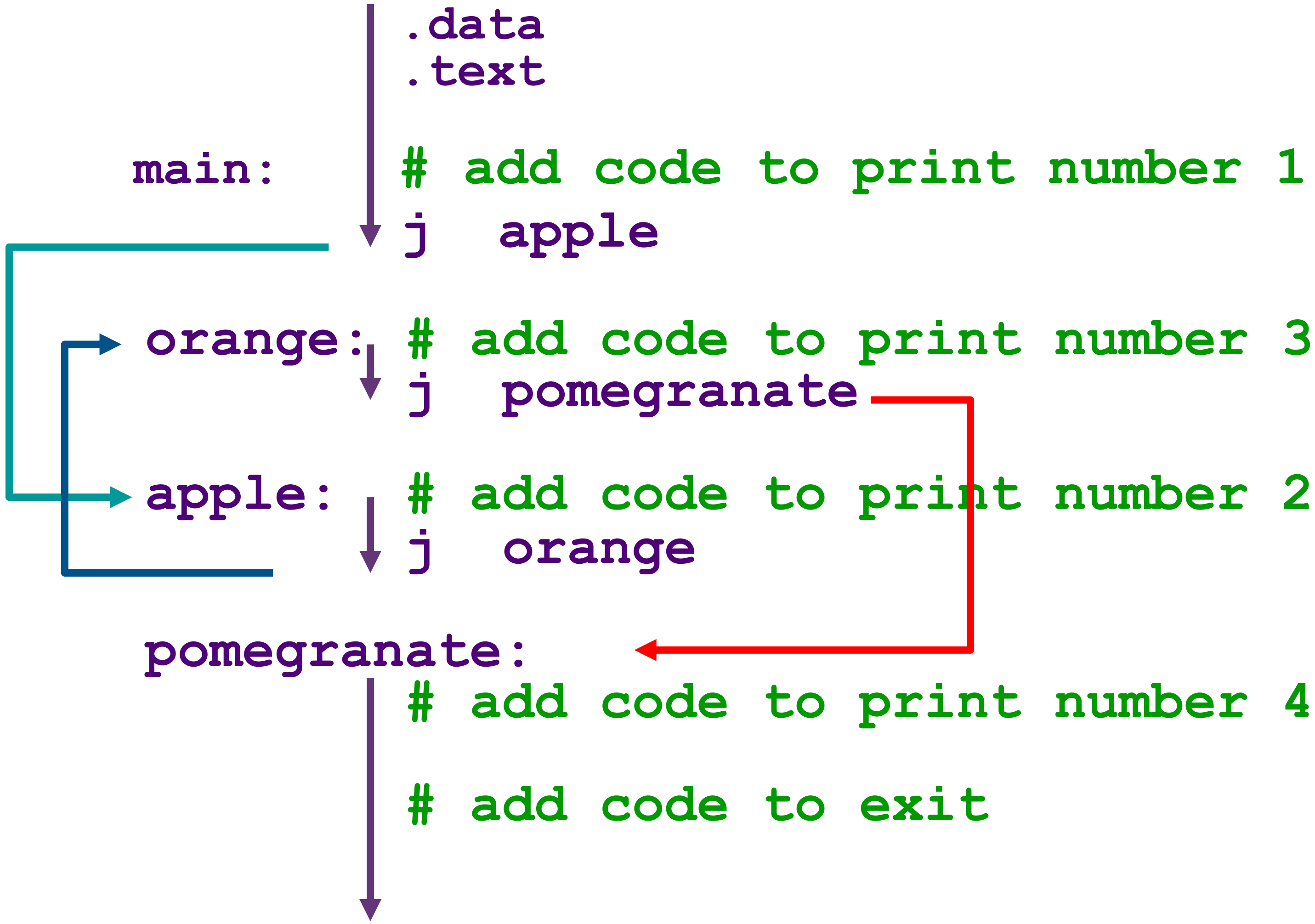
```
# Code could be this ugly!
def main():
    print(1)
    goto apple

  orange:
     print(3)
          goto pomegranate

  apple:
    print(2)
    goto orange

  pomegranate:
    print(4)
```



I COULD RESTRUCTURE THE PROGRAM'S FLOW
OR USE ONE LITTLE 'GOTO' INSTEAD.
EH, SCREW GOOD PRACTICE. HOW BAD CAN IT BE?
goto main_sub3;
COMPILE

http://xkcd.com/292/

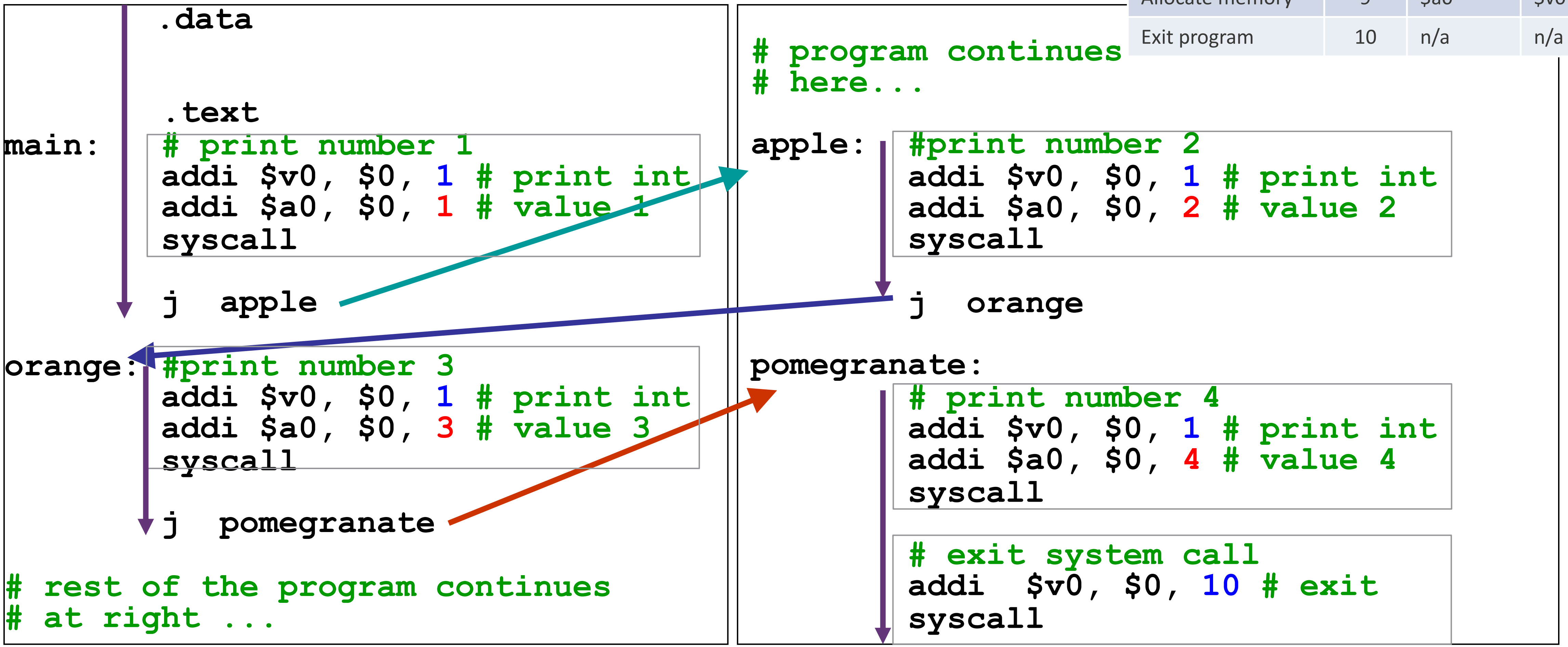
Using MIPS j instruction



```
        .data
        .text

main:   # add code to print number 1
        j   apple

orange: # add code to print number 3
        j   pomegranate

apple:  # add code to print number 2
        j   orange

pomegranate:
        # add code to print number 4

        # add code to exit
```

Service	Code	Arg	Res
Print integer	1	$a0	n/a
Print string	4	$a0	n/a
Read integer	5	n/a	$v0
Read string	8	$a0 $a1	n/a
Allocate memory	9	$a0	$v0
Exit program	10	n/a	n/a

Using MIPS j instruction



```
        .data

        .text
main:   # print number 1
        addi $v0, $0, 1 # print int
        addi $a0, $0, 1 # value 1
        syscall

        j  apple

orange: #print number 3
        addi $v0, $0, 1 # print int
        addi $a0, $0, 3 # value 3
        syscall

        j  pomegranate

# rest of the program continues
# at right ...
```

```
# program continues
# here...

apple:  #print number 2
        addi $v0, $0, 1 # print int
        addi $a0, $0, 2 # value 2
        syscall

        j  orange

pomegranate:
        # print number 4
        addi $v0, $0, 1 # print int
        addi $a0, $0, 4 # value 4
        syscall

        # exit system call
        addi  $v0, $0, 10 # exit
        syscall
```

Do I need to set `$v0` to 1 again?

Faithful translations from high-level to MIPS

- **Each line is translated independently of each other:**
  - Load/store variable values from memory for every Python instruction
  - No reuse of registers across instructions (as we did for `$v0` in previous example)
- **Each line is translated in the order given by the Python program:**
  - For example, if i+=1 is at the end of a loop, put it at the end
- **Global variables are encoded as globals, locals as locals (we will see how)**
- **Strings are encoded as given (don't add or subtract characters)**
- **Conditions in if-then-else and loops encoded exactly as given (see later)**
- **Aside from this: you are free to optimise translation of each Python line**

Main **aims** of faithfulness: for you to not get confused while coding; for us to be able to test properly and mark consistently

Service	Code	Arg	Res
Print integer	1	$a0	n/a
Print string	4	$a0	n/a
Read integer	5	n/a	$v0
Read string	8	$a0 $a1	n/a
Allocate memory	9	$a0	$v0
Exit program	10	n/a	n/a

Using MIPS j instruction

```
        .data


        .text
main:   # print number 1
        addi $v0, $0, 1 # print int
        addi $a0, $0, 1 # value 1
        syscall

        j  apple

orange: #print number 3
        addi $v0, $0, 1 # print int
        addi $a0, $0, 3 # value 3
        syscall

        j  pomegranate

# rest of the program continues
# at right ...
```

```
# program continues
# here...

apple:  #print number 2
        addi $v0, $0, 1 # print int
        addi $a0, $0, 2 # value 2
        syscall

        j  orange

pomegranate:
        # print number 4
        addi $v0, $0, 1 # print int
        addi $a0, $0, 4 # value 4
        syscall

        # exit system call
        addi  $v0, $0, 10 # exit
        syscall
```

Do I need to set $v0 to 1 again?

Only if I ask you to be faithful!

MONASH University

Conditional control transfer instructions

Conditional control transfer instructions

- **Branch to a label if one value is equal/not equal another**
  - branch if equal to
    ```
    beq $t1, $t2, foo  # if $t1==$t2 goto foo
    ```
  - branch if not equal to
    ```
    bne $t1, $t2, foo  # if $t1!=$t2 goto foo
    ```
- **If MIPS condition false, do normal PC update (PC = PC + 4)**
- **If true, alter PC to equal label (PC = address `foo`)**
- **Interesting: `foo` is encoded as a signed offset (not as an address)**
  - Counts in words and, when added to PC, points to address `foo`
  - Branches effectively says “jump forward/backward *N* places”
  - PC + 4 + (sign extended immediate field <<2)

What is the shift doing?

Valid instruction addresses are multiples of four, so their last two bits are always 0

Conditional control transfer instructions

- **Is that it? Not really, MIPS also has:**
  - branch if less than

    `blt $t1, $t2, foo  # if $t1<$t2 goto foo`
  - branch if less than or equal to

    `ble $t1, $t2, foo  # if $t1<=$t2 goto foo`
  - branch if greater than

    `bgt $t1, $t2, foo  # if $t1>$t2 goto foo`
  - branch if greater or equal to

    `bge $t1, $t2, foo  # if $t1>=$t2 goto foo`
- **These are pseudoinstructions. And remember:**
  - They transform into several instructions
  - We will not use them (you must practice with the basics)
  - The only pseudoinstruction we will use in FIT1008/2085 is `la`

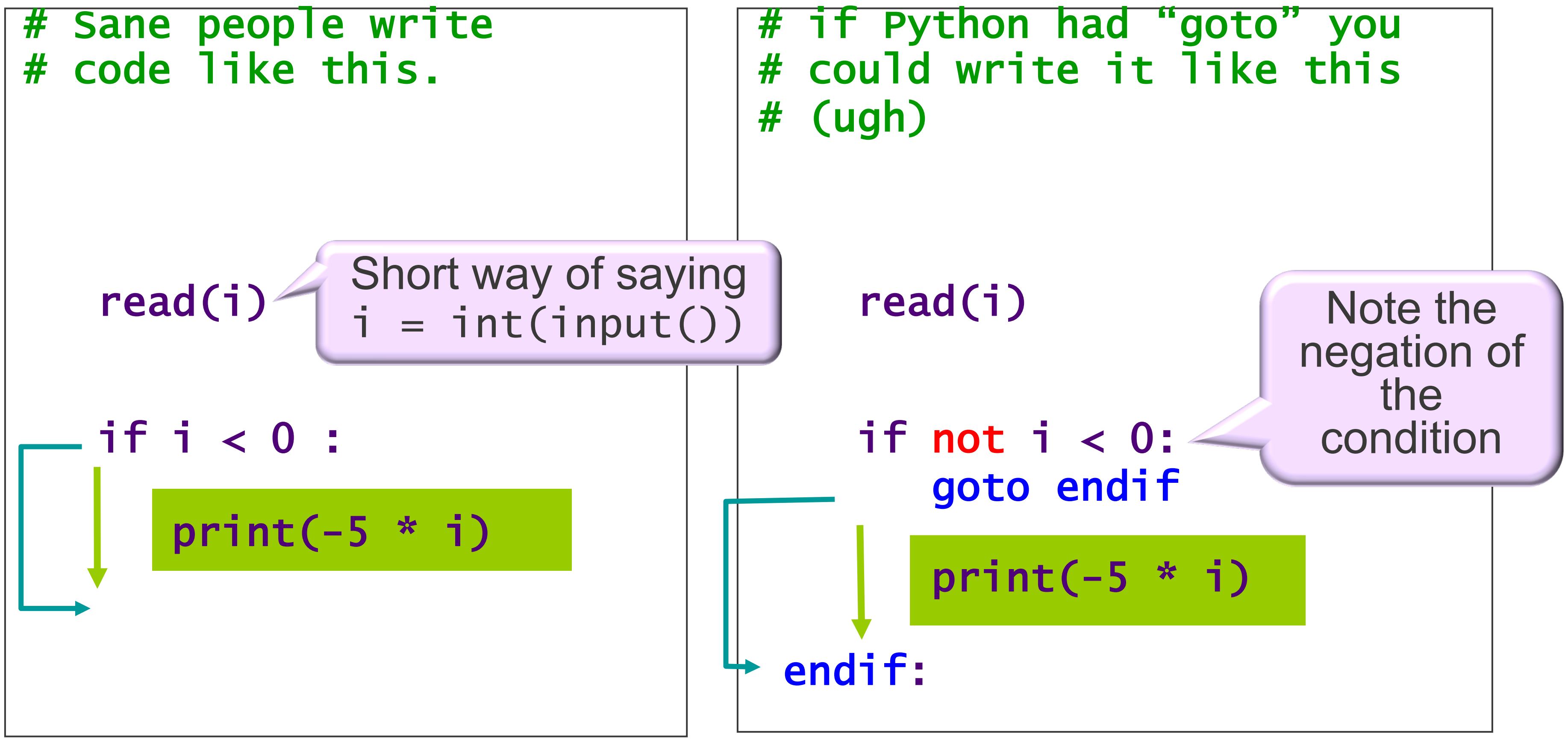
To be crystal clear

YOU ARE NEVER ALLOWED TO USE THEM IN FIT1008/FIT2085

Control transfer is useful for selection

- **Selection is how programs make choices**
  - In Python: if-then, if-then-else, if-then-elif-…like switch cases)
- **Achieved by selectively not executing some lines of code**

Selection: `if-then`



```
# Sane people write
# code like this.


read(i)

if i < 0 :
    print(-5 * i)
```

Short way of saying
`i = int(input())`

```
# if Python had “goto” you
# could write it like this
# (ugh)


read(i)

if not i < 0:
   goto endif
   print(-5 * i)
endif:
```

Note the negation of the condition

Comparison instructions

Comparison Instructions

- **Control transfer is not enough, you also need to decide what to select: need to compare (`i < 0`)**

- **set less than**

  - `slt $t0,$t1,$t2 # if $t1<$t2 then $t0=1`
    ```
    slt $t0,$t1,$t2 # if $t1<$t2 then $t0=1
                    # else $t0 = 0
    ```
  - Use this in conjunction with branch instructions to translate IF statements in high-level languages

- **set less than immediate**

  -
    ```
    slti $t0,$t1,1  # if $t1<1 then $t0=1
                    # else $t0 = 0
    ```

- **Note: comparisons are performed by the ALU**

  - So comparison instructions are really arithmetic ones

Set Less Than – Example

```
.text
 addi $t1,$0,4        # $t1 ← 4
 addi $t2,$0,2        # $t2 ← 2
 slt $t0,$t1,$t2      # $t0 ← 0
 slti $t3,$t2,3       # $t3 ← 1
```

Practicing MIPS branching with `slt`

$t0	$t1	$t2	
X	Y	$t0< $t1	$t1<$t0
10	15	1	0
15	15	0	0
15	10	0	1

X<Y

Y<X

not X<Y

same as X>=Y

not Y< X

same as X <=Y

When translating: always draw this table

Putting it all together: if-then

- Example: assume `X` is in `$t0` and `Y` in `$t1`

  - `if X == Y:` ⇨

    ```
    bne $t0, $t1, endif
    ```

    Same as saying if not X==Y go to endif

  - `if X < Y:` ⇨

    ```
    slt $t2, $t0, $t1
    beq $t2, $0, endif
    ```

    Same as saying if not X<Y go to endif

  - `if X <= Y:` ⇨

    ```
    slt $t2, $t1, $t0
    bne $t2, $0, endif
    ```

    Same as saying
    if Y<X go to endif
    which is equivalent:
    if not X<=Y go to endif

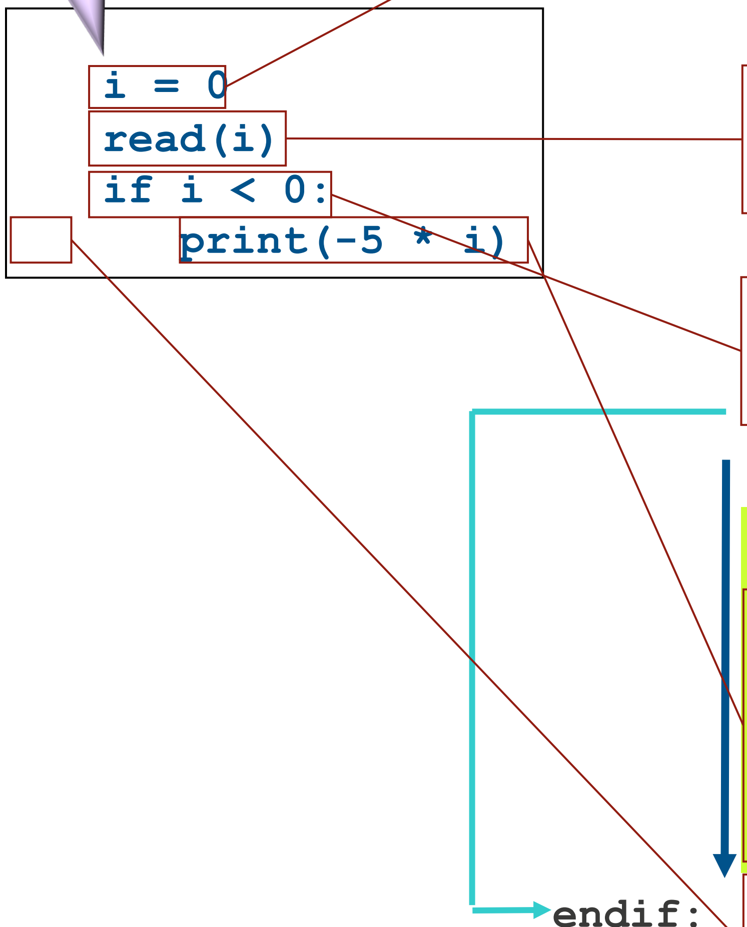
- We use comparison to evaluate the condition (if needed)
- We use branch instructions to jump over the "then"
- We use jump instructions to jump over the "else"

MONASH University

Translating if-then

If-then in MIPS

We will treat i as a global variable



```
i = 0
read(i)
if i < 0:
    print(-5 * i)
```

```
          .data
         i:   .word 0

          .text
          # Read integer “i” from input
          addi  $v0, $0, 5         # system call code to
          syscall                  # read an int
          sw  $v0, i               # store result in I

          # Comparison part: if not i < 0: goto endif
          lw  $t0, i               # $t0=i
          slt $t1, $t0, $0         # $t1 = 0 if not i<0
          beq  $t1, $0, endif      # if $t1 = 0 go to endif

          # ... else fall through to here

          # and print out -5*i
          lw    $t0, i             # $t0=i
          addi $t1, $0, -5         # store the 5 into a register
          mult $t0, $t1            # -5*i
          mflo $a0                 # $a0 = -5*i
          addi $v0, $0, 1          # call code to print an integer
          syscall

  endif:  # exit program
          addi  $v0, $0, 10        # call code to exit
          syscall                  # exit
```

Summary

- **Learned about MIPS:**
  - Conditional control transfer instructions
  - Comparison instructions
- **Are able to use them to translate `if-then`**
- **Know how to perform a faithful translation**