

# CSci 127: Introduction to Computer Science



[hunter.cuny.edu/csci](http://hunter.cuny.edu/csci)

# Announcements

- Please always read all Blackboard announcements



# Announcements

- Please always read all Blackboard announcements
- Online help is available in multiple forms when school is in session:



# Announcements



- Please always read all Blackboard announcements
- Online help is available in multiple forms when school is in session:
  - ▶ **Email:** csci127help@gmail.com

# Announcements



- Please always read all Blackboard announcements
- Online help is available in multiple forms when school is in session:
  - ▶ **Email:** csci127help@gmail.com
  - ▶ **Discussion Board:** on Blackboard, link on purple menu bar

# Announcements



- Please always read all Blackboard announcements
- Online help is available in multiple forms when school is in session:
  - ▶ **Email:** csci127help@gmail.com
  - ▶ **Discussion Board:** on Blackboard, link on purple menu bar
  - ▶ **Drop-in tutoring (12pm-5pm):**

# Announcements



- Please always read all Blackboard announcements
- Online help is available in multiple forms when school is in session:
  - ▶ **Email:** csci127help@gmail.com
  - ▶ **Discussion Board:** on Blackboard, link on purple menu bar
  - ▶ **Drop-in tutoring (12pm-5pm):**  
sign in here:  
<https://bit.ly/csci127Tutoring>  
then join the session here:  
<https://bit.ly/csci127TutoringSession>

# Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

# Today's Topics



- **Design Patterns: Searching**
  - Python Recap
  - Machine Language
  - Machine Language: Jumps & Loops
  - Binary & Hex Arithmetic
  - Final Exam: Format

## Predict what the code will do:

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')|
```

# Python Tutor

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

(Demo with pythonTutor)

# Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.

# Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.
- Start at the beginning of the list.

# Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.
- Start at the beginning of the list.
- Look at each item, one-by-one.

# Design Pattern: Linear Search

```
def search(nums, locate):
    found = False
    i = 0
    while not found and i < len(nums):
        print(nums[i])
        if locate == nums[i]:
            found = True
        else:
            i = i+1
    return(found)

nums= [1,4,10,6,5,42,9,8,12]
if search(nums,6):
    print('Found it! 6 is in the list!')
else:
    print('Did not find 6 in the list.')
```

- Example of **linear search**.
- Start at the beginning of the list.
- Look at each item, one-by-one.
- Stopping, when found, or the end of list is reached.

# Today's Topics



- Design Patterns: Searching
- **Python Recap**
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

# Python & Circuits Review: 10 Weeks in 10 Minutes



A whirlwind tour of the semester, so far...

# Week 1: print(), loops, comments, & turtles

# Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

```
#Date: September 1, 2017
```

← (for us, not computer to read)

```
#This program prints: Hello, World!
```

← (this one also)

```
print("Hello, World!")
```

← Prints the string "Hello, World!" to the screen

# Week 1: print(), loops, comments, & turtles

- Introduced comments & print():

```
#Name: Thomas Hunter
```

← These lines are comments

```
#Date: September 1, 2017
```

← (for us, not computer to read)

```
#This program prints: Hello, World!
```

← (this one also)

```
print("Hello, World!")
```

← Prints the string "Hello, World!" to the screen

- As well as definite loops & the turtle package:

The screenshot shows a Python code editor interface. On the left, the code file 'main.py' is open, containing the following code:

```
1 #A program that demonstrates turtles stamping
2
3 import turtle
4
5 taylor = turtle.Turtle()
6 taylor.color("purple")
7 taylor.shape("turtle")
8
9 for i in range(6):
10     taylor.forward(100)
11     taylor.stamp()
12     taylor.left(60)
```

On the right, the 'Result' tab displays the output of the program: a purple hexagon drawn on the screen with black star-shaped stamps at each vertex.

# Week 2: variables, data types, more on loops & range()

## Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.

## Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers

## Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers

## Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters

## Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters
  - ▶ **list**: a sequence of items

## Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters
  - ▶ **list**: a sequence of items
    - e.g. [3, 1, 4, 5, 9] or ['violet', 'purple', 'indigo']

## Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters
  - ▶ **list**: a sequence of items
    - e.g. [3, 1, 4, 5, 9] or ['violet', 'purple', 'indigo']
  - ▶ **class variables**: for complex objects, like turtles.

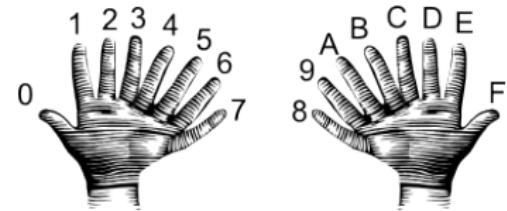
## Week 2: variables, data types, more on loops & range()

- A **variable** is a reserved memory location for storing a value.
- Different kinds, or **types**, of values need different amounts of space:
  - ▶ **int**: integer or whole numbers
  - ▶ **float**: floating point or real numbers
  - ▶ **string**: sequence of characters
  - ▶ **list**: a sequence of items
    - e.g. [3, 1, 4, 5, 9] or ['violet', 'purple', 'indigo']
  - ▶ **class variables**: for complex objects, like turtles.
- More on loops & ranges:

```
1 #Predict what will be printed:  
2  
3 for num in [2,4,6,8,10]:  
4     print(num)  
5  
6 sum = 0  
7 for x in range(0,12,2):  
8     print(x)  
9     sum = sum + x  
10  
11 print(sum)  
12  
13 for c in "ABCD":  
14     print(c)
```

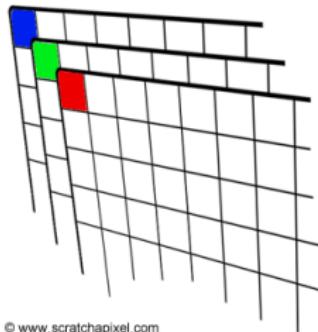
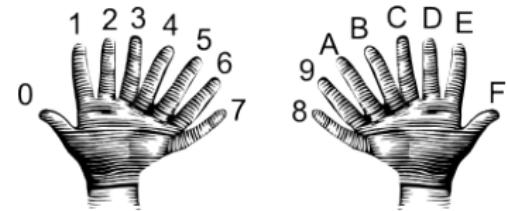
# Week 3: colors, hex, slices, numpy & images

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



# Week 3: colors, hex, slices, numpy & images

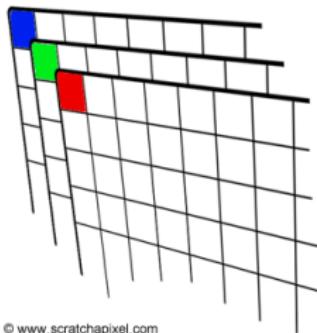
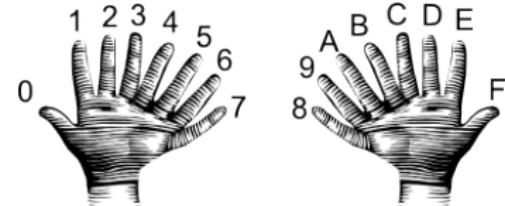
Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



© www.scratchapixel.com

# Week 3: colors, hex, slices, numpy & images

Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



```
>>> a[0:3:5]  
array([3,4])
```

```
>>> a[4:,4:]  
array([[44, 45],  
       [54, 55]])
```

```
>>> a[:,2]  
array([2,12,22,32,42,52])
```

```
>>> a[2::2,:,:2]  
array([[20,22,24],  
       [40,42,44]])
```

0	1	2	3	4	5
10	11	12	13	14	15
20	21	22	23	24	25
30	31	32	33	34	35
40	41	42	43	44	45
50	51	52	53	54	55

# Week 4: design problem (cropping images) & decisions



# Week 4: design problem (cropping images) & decisions



- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*

# Week 4: design problem (cropping images) & decisions



- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*
- Next: write pseudocode.
  - ① Import numpy and pyplot.
  - ② Ask user for file names and dimensions for cropping.
  - ③ Save input file to an array.
  - ④ Copy the cropped portion to a new array.
  - ⑤ Save the new array to the output file.

# Week 4: design problem (cropping images) & decisions



- First: specify inputs/outputs. *Input file name, output file name, upper, lower, left, right ("bounding box")*
- Next: write pseudocode.
  - ① Import numpy and pyplot.
  - ② Ask user for file names and dimensions for cropping.
  - ③ Save input file to an array.
  - ④ Copy the cropped portion to a new array.
  - ⑤ Save the new array to the output file.
- Next: translate to Python.

## Week 4: design problem (cropping images) & decisions

```
yearBorn = int(input('Enter year born: '))
if yearBorn < 1946:
    print("Greatest Generation")
elif yearBorn <= 1964:
    print("Baby Boomer")
elif yearBorn <= 1984:
    print("Generation X")
elif yearBorn <= 2004:
    print("Millennial")
else:
    print("TBD")

x = int(input('Enter number: '))
if x % 2 == 0:
    print('Even number')
else:
    print('Odd number')
```

# Week 5: logical operators, truth tables & logical circuits

```
origin = "Indian Ocean"
winds = 100
if (winds > 74):
    print("Major storm, called a ", end="")
    if origin == "Indian Ocean" or origin == "South Pacific":
        print("cyclone.")
    elif origin == "North Pacific":
        print("typhoon.")
    else:
        print("hurricane.")

visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
   (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

# Week 5: logical operators, truth tables & logical circuits

```
origin = "Indian Ocean"
winds = 100
if (winds > 74):
    print("Major storm, called a ", end="")
    if origin == "Indian Ocean" or origin == "South Pacific":
        print("cyclone.")
    elif origin == "North Pacific":
        print("typhoon.")
    else:
        print("hurricane.")

visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
   (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

in1	and	in2	returns:
False	and	False	False
False	and	True	False
True	and	False	False
True	and	True	True



# Week 6: structured data, pandas, & more design

Source: [https://en.wikipedia.org/wiki/Demographics\\_of\\_New\\_York\\_City](https://en.wikipedia.org/wiki/Demographics_of_New_York_City),  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....

.....  
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total  
1690,1,037,2017,,727,788,2017  
1771,21843,36232,,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,67541,6354,7042,1755,4543,75934  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,12013,14000,5348,10965,391114  
1850,35544,12800,18950,5348,10965,391115  
1860,813469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59940,5653,51980,33029,1911801  
1890,1367111,66582,51583,48128,31864,2145134  
1900,185093,116582,152999,200567,67921,2437202  
1910,2233142,1634351,284041,430980,8569,4766803  
1920,2211103,2018354,44603,44603,73201,11651,50048  
1930,1867111,1579128,1579128,1579128,1579128,4930446  
1940,1889924,2498285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7891957  
1960,1696101,2319319,1899049,1451277,191555,7891984  
1970,1539231,2465701,1899135,1471701,135443,7891984  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2229379,1332450,419782,8080879  
2010,1583873,2504705,2237722,1385108,474558,8175133  
2015,1444518,2436733,2339150,1459444,474558,8059405

nycHistPop.csv

In Lab 6

# Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt  
import pandas as pd
```

Source: [https://en.wikipedia.org/wiki/Demographics\\_of\\_New\\_York\\_City](https://en.wikipedia.org/wiki/Demographics_of_New_York_City),  
All population figures are consistent with present-day boundaries.....  
Five census after the consolidation of the five boroughs.....

.....  
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total  
1890,4937,2017,,727,7181,10000  
1871,21843,3623,,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,63549,5540,6242,1755,4543,75934  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,312110,18013,14049,5346,10965,391114  
1850,35549,21891,18951,4895,10965,491115  
1860,513469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33029,1911803  
1890,1367111,70000,6534,5653,51980,2166134  
1900,185093,116582,152999,200567,67921,2437202  
1910,223342,1634351,284041,430980,8569,4766803  
1920,2210103,2018354,446071,446071,732013,116515,591048  
1930,1867111,70000,6534,5653,51980,2166134  
1940,1889924,2469285,1297634,1394711,1374441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690101,2738175,1550949,1451277,191555,7981984  
1970,1539231,2469285,1297634,1374441,7454995,7981984  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2229379,1332450,419728,8080879  
2010,1583873,2504705,2272722,1385108,419728,8175133  
2015,1444018,2436733,2339150,1459444,474558,8059405

nycHistPop.csv

In Lab 6

# Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt  
import pandas as pd
```

```
pop = pd.read_csv('nycHistPop.csv', skiprows=5)
```

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Borough,Brooklyn,Queens,Bronx,Staten Island,Totals  
1690,4937,2037,727,788,2422  
1770,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,62000,5850,6500,1800,4600,75350  
1820,123704,11187,8246,2792,6135,152056  
1830,20589,20535,9049,3023,7082,242278  
1840,31210,21013,14000,5348,10965,391114  
1850,35549,21880,18500,5800,11500,460015  
1860,613469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59940,5653,51980,33091,1911801  
1890,1367000,70000,60000,51800,33091,2131534  
1900,185093,116582,152999,200567,67921,2437202  
1910,233142,1634351,284041,430980,8569,476683  
1920,22101103,2018354,446000,720201,116500,500000  
1930,18671128,16671128,352543,158200,4930446  
1940,1889924,2690285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690000,2319319,1809000,1451277,191555,7981984  
1970,1539231,2465701,1874473,1472701,195443,7984640  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1320789,378977,7322564  
2000,1537195,2485326,2223379,1332450,419782,8080879  
2010,1583873,2504705,2271722,1385108,447558,8175133  
2015,1444518,2646733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6

# Week 6: structured data, pandas, & more design

```
import matplotlib.pyplot as plt  
import pandas as pd
```

```
pop = pd.read_csv('nycHistPop.csv', skiprows=5)
```

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
Five census after the consolidation of the five boroughs.....  
.....  
Year,Population  
1690,203,2037,...,727,7181  
1771,21843,36231,...,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,70000,6350,7000,1800,5300,93734  
1820,123704,11187,8246,2792,6135,152056  
1830,20589,20535,9049,3023,7082,242278  
1840,31510,21013,14000,5348,10965,391114  
1850,35549,21890,18895,7000,10000,500015  
1860,813469,279122,23903,23933,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33001,1911801  
1890,1370000,720000,68000,51000,35000,2100014  
1900,1850993,116582,152999,200567,67621,2437202  
1910,2233142,1634351,2841,430980,8569,476683  
1920,22101103,2018354,44600,720201,116000,500008  
1930,2667110,2000000,970128,115000,50000,4930446  
1940,1889924,2699285,1297634,1394711,1374441,7454995  
1950,1960101,2738175,1550949,1451277,191555,7991957  
1960,1690000,2319319,1809000,1400000,1200000,781984  
1970,1539231,2465700,1874731,1472701,135443,768460  
1980,1426285,2230936,1891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1302789,378977,7322564  
2000,1537195,2485326,2229379,1332650,419782,8080879  
2010,1583873,2504705,2277722,1385108,451750,8175133  
2015,1444518,2546733,2339150,1459446,474558,8056405
```

nycHistPop.csv

In Lab 6

# Week 6: structured data, pandas, & more design

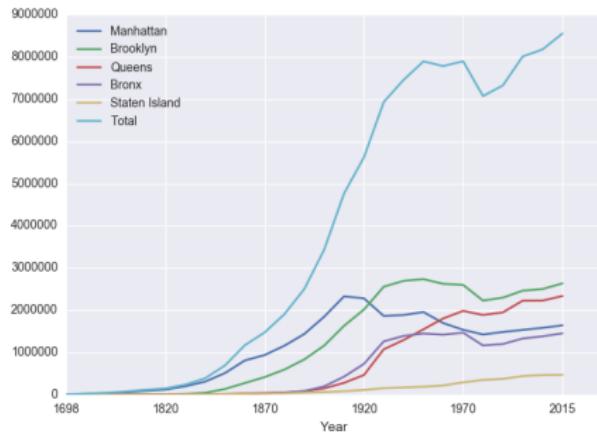
```
import matplotlib.pyplot as plt  
import pandas as pd
```

```
pop = pd.read_csv('nycHistPop.csv', skiprows=5)
```

```
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City.....  
All population figures are consistent with present-day boundaries.....  
First census after the consolidation of the five boroughs.....  
.....  
Year,Borough,Population  
1698,Manhattan,2037,727,7181  
1771,21843,36231,2847,28423  
1790,33131,4549,6159,1781,3827,49447  
1800,60515,5740,6442,1755,4543,75955  
1810,70531,6211,6881,1811,4971,89734  
1820,123704,11187,8246,2792,6135,152056  
1830,202589,20535,9049,3023,7082,242278  
1840,315110,11013,14041,5348,10965,391114  
1850,355441,12800,18951,5835,13041,45115  
1860,613469,279122,32903,23593,25492,174777  
1870,942292,419921,45468,37393,33029,1479103  
1880,1164473,59943,5653,51980,33051,1911801  
1890,1384473,72001,6381,57348,38501,1911814  
1900,1850993,116582,152999,200567,67921,2437202  
1910,2331842,1634351,2841,430980,8569,476683  
1920,2281103,2018354,44601,73201,11651,50048  
1930,2667103,2486128,57944,73545,15873,593446  
1940,1889924,2698285,1297634,1394711,174441,7454995  
1950,1960101,2738175,1550849,1451277,191555,7891957  
1960,1696101,2738175,1550849,1451277,191555,7891984  
1970,1539231,2467071,1472701,1247101,135443,7891946  
1980,1426285,2230936,11891325,1168972,352121,7071639  
1990,1487536,2300664,1951598,1203789,378977,7322564  
2000,1537195,2485326,2229379,1332450,413728,8080879  
2010,1583873,2504705,2216722,1385108,413728,8175133  
2015,1444518,2636733,2339150,1459446,474558,8059405
```

nycHistPop.csv

In Lab 6



# Week 7: functions

- Functions are a way to break code into pieces, that can be easily reused.

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

# Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.

# Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`

# Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program uses functions,
#     says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:

# Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:  
Example: `print("Hello", "World")`

# Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:  
Example: `print("Hello", "World")`
- Can write, or **define** your own functions,

# Week 7: functions

```
#Name: your name here
#Date: October 2017
#This program, uses functions,
#      says hello to the world!

def main():
    print("Hello, World!")

if __name__ == "__main__":
    main()
```

- Functions are a way to break code into pieces, that can be easily reused.
- Many languages require that all code must be organized with functions.
- The opening function is often called `main()`
- You **call** or **invoke** a function by typing its name, followed by any inputs, surrounded by parenthesis:  
Example: `print("Hello", "World")`
- Can write, or **define** your own functions, which are stored, until invoked or called.

# Week 8: function parameters, github

- Functions can have **input parameters**.

```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: '))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)  
  
dinner= float(input('Enter dinner total: '))  
dTip = float(input('Enter dinner tip: '))  
dTotal = totalWithTax(dinner, dTip)  
print('Dinner total is', dTotal)
```

# Week 8: function parameters, github

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).

# Week 8: function parameters, github

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.

# Week 8: function parameters, github

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**

# Week 8: function parameters, github

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**
- Functions can also **return values** to where it was called.

# Week 8: function parameters, github

```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)

lunch = float(input('Enter lunch total: '))
lTip = float(input('Enter lunch tip: '))
lTotal = totalWithTax(lunch, lTip)
print('Lunch total is', lTotal)
                                Actual Parameters

dinner= float(input('Enter dinner total: '))
dTip = float(input('Enter dinner tip: '))
dTotal = totalWithTax(dinner, dTip)
print('Dinner total is', dTotal)
```

- Functions can have **input parameters**.
- Surrounded by parenthesis, both in the function definition, and in the function call (invocation).
- The “placeholders” in the function definition: **formal parameters**.
- The ones in the function call: **actual parameters**.
- Functions can also **return values** to where it was called.

# Week 9: top-down design, folium, loops, and random()



```
def main():
    dataF = getData()
    latColName, lonColName = getColumnNames()
    lat, lon = getLocale()
    cityMap = folium.Map(location = [lat,lon], tiles = 'cartodbpositron',zoom_start=11)
    dotAllPoints(cityMap,dataF,latColName,lonColName)
    markAndFindClosest(cityMap,dataF,latColName,lonColName,lat,lon)
    writeMap(cityMap)
```

# Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```

# Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```

# Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.

# Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.
- To use, must include:  
`import random`.

# Week 10: more on loops, max design pattern, random()

```
dist = int(input('Enter distance: '))
while dist < 0:
    print('Distances cannot be negative.')
    dist = int(input('Enter distance: '))

print('The distance entered is', dist)
```

```
import turtle
import random

trey = turtle.Turtle()
trey.speed(10)

for i in range(100):
    trey.forward(10)
    a = random.randrange(0,360,90)
    trey.right(a)
```

- Indefinite (while) loops allow you to repeat a block of code as long as a condition holds.
- Very useful for checking user input for correctness.
- Python's built-in random package has useful methods for generating random whole numbers and real numbers.
- To use, must include:  
`import random`.
- The max design pattern provides a template for finding maximum value from a list.

# Python & Circuits Review: 10 Weeks in 10 Minutes



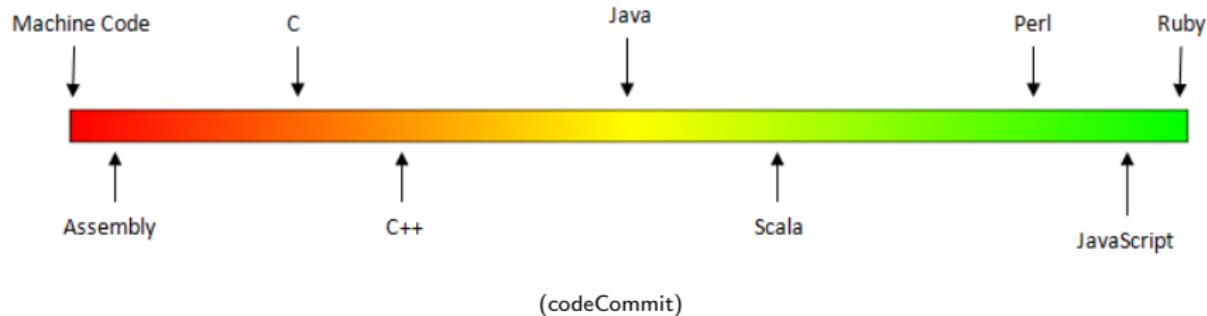
- Input/Output (I/O): `input()` and `print()`; pandas for CSV files
- Types:
  - ▶ Primitive: `int`, `float`, `bool`, `string`;
  - ▶ Container: lists (but not dictionaries/hashes or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: if-elif-else
- Logical Expressions & Circuits
- Functions: parameters & returns
- Packages:
  - ▶ Built-in: `turtle`, `math`, `random`
  - ▶ Popular: `numpy`, `matplotlib`, `pandas`, `folium`

# Today's Topics



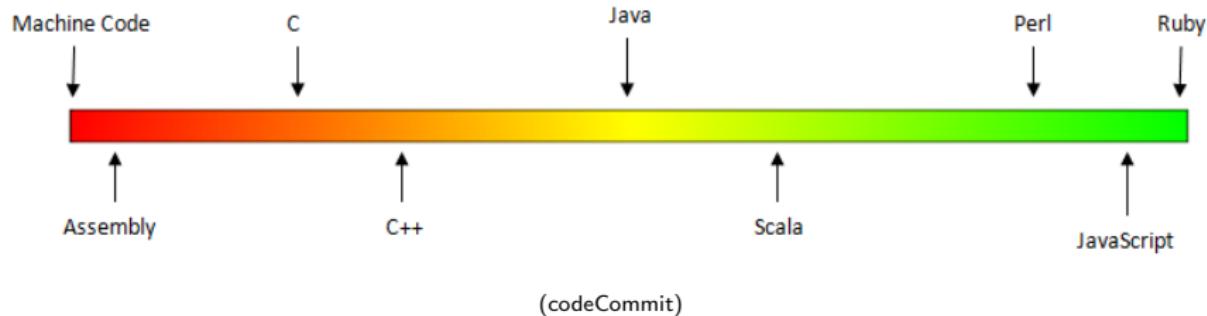
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

# Low-Level vs. High-Level Languages



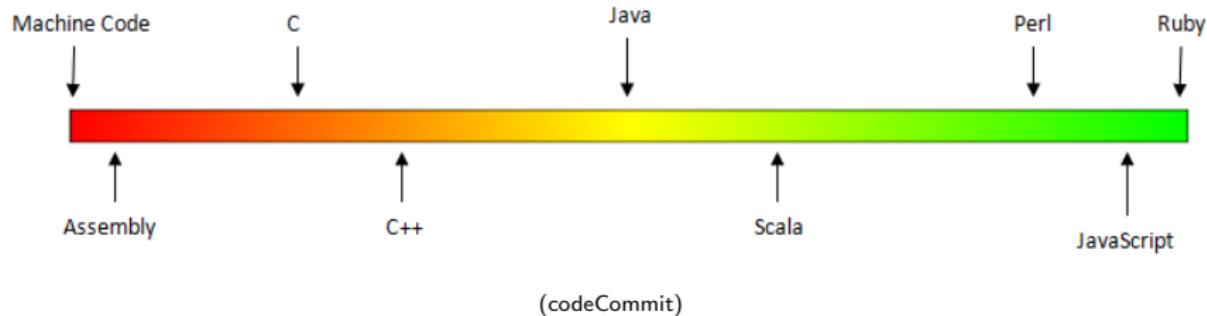
- Can view programming languages on a continuum.

# Low-Level vs. High-Level Languages



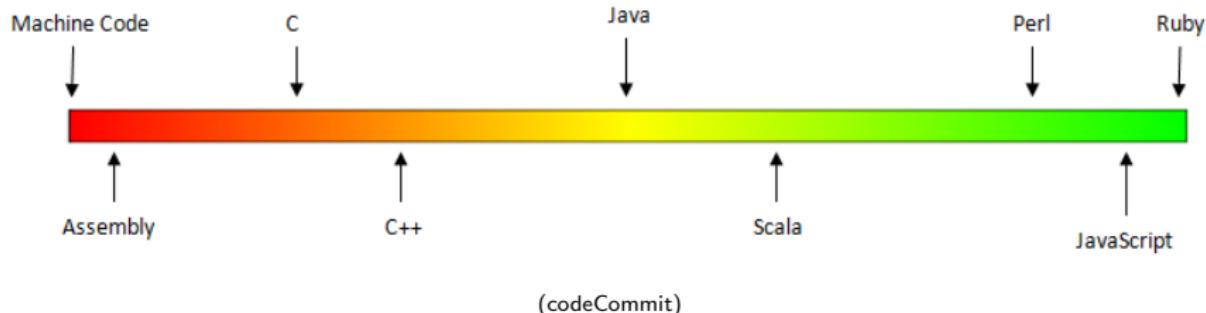
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

# Low-Level vs. High-Level Languages



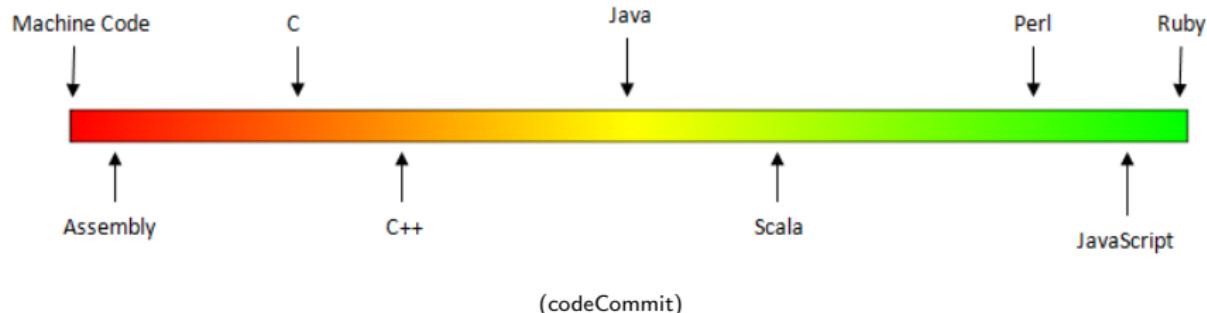
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

# Low-Level vs. High-Level Languages



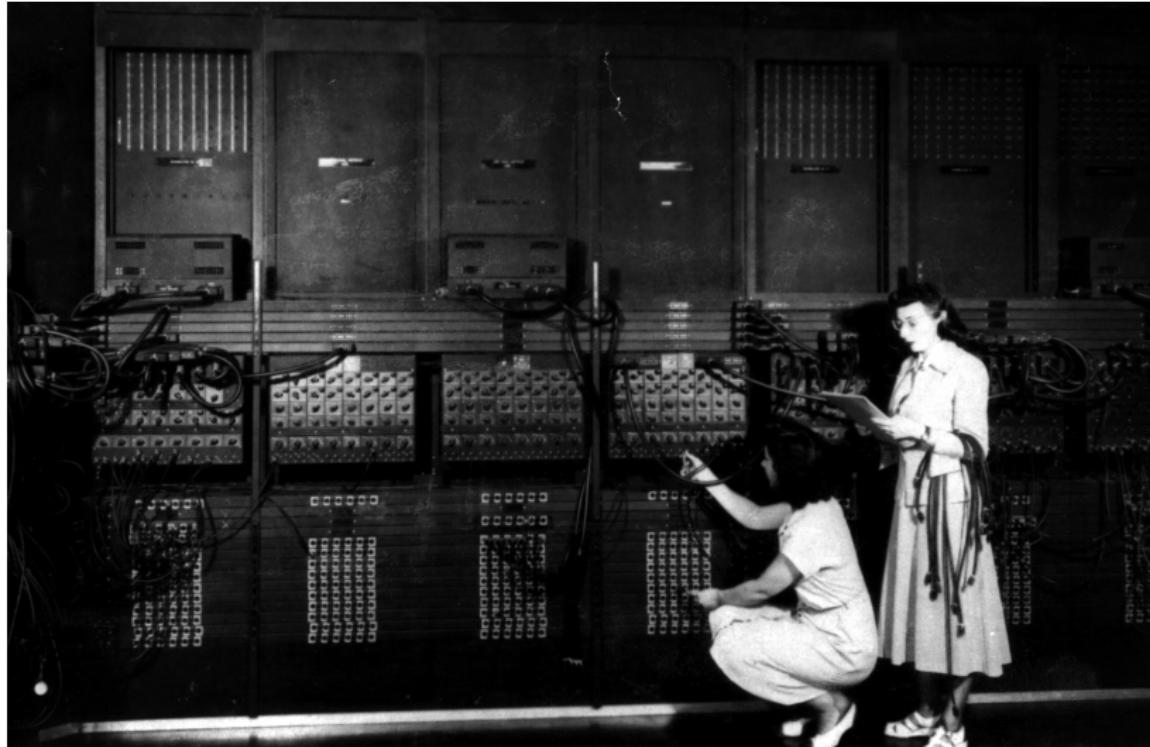
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine details, such as complex variables, functions and looping that do not translate directly into machine code) are called **high-level languages**.

# Low-Level vs. High-Level Languages



- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).
- Those that have strong abstraction (allow programming paradigms independent of the machine details, such as complex variables, functions and looping that do not translate directly into machine code) are called **high-level languages**.
- Some languages, like C, are in between— allowing both low level access and high level data structures.

# Machine Language



(Ruth Gordon & Ester Gerston programming the ENIAC, UPenn)

# Machine Language

```
I FDX 12:01a 23- 1
A 002000 C2 30      REP #$30
A 002002 18          CLC
A 002003 F8          SED
A 002004 A9 34 12    LDA #$1234
A 002007 69 21 43    ADC #$4321
A 00200A 8F 03 7F 01 STA $017F03
A 00200E D8          CLD
A 00200F E2 30      SEP #$30
A 002011 00          BRK
A 2012

r
PB PC  NUMxDIZC .A .X .Y SP DP DB
; 00 E012 00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

PB PC  NUMxDIZC .A .X .Y SP DP DB
; 00 2013 00110000 5555 0000 0002 CFFF 0000 00
m 7f03 7f03
>007F03 55 55 00 00 00 00 00 00 00 00 00 00 00 00:UU .....
```

(wiki)

# Machine Language

- We will be writing programs in a simplified machine language, WeMIPS.

(wiki)

# Machine Language

- We will be writing programs in a simplified machine language, WeMIPS.
  - It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.

(wiki)

# Machine Language

```

A 002300 C2 3B REP #3B
A 002302 7F CLC
A 002303 FB SED
A 002304 34 12 ADD #1224
A 002307 69 21 43 ADC #4321
A 002308 8F B3 7F B1 STA $0017F9
A 00230E D0 CLD
A 00230F E2 3B SEP #3B
A 002311 90 BPK
A 002312

F
PB PC Min:012C A X Y SP DP IR
; 00 2013 00110800 0550 0000 0002 CFFF 0000 00
$ 2000

BREAK

PB PC Min:012C A X Y SP DP IR
; 00 2013 00110800 0550 0000 0002 CFFF 0000 00
$ 7793 7793

$0017F9 55 55 00 00 00 00 00 00 00 00 00 00 00 00 00 00

```

(wiki)

- We will be writing programs in a simplified machine language, WeMIPS.
  - It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.
  - Due to its small set of commands, processors can be designed to run those commands very efficiently.

# Machine Language

(wiki)

- We will be writing programs in a simplified machine language, WeMIPS.
  - It is based on a reduced instruction set computer (RISC) design, originally developed by the MIPS Computer Systems.
  - Due to its small set of commands, processors can be designed to run those commands very efficiently.
  - More in future architecture classes....

# "Hello World!" in Simplified Machine Language

Line: 3 Go! Show/Hide Demos

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

```
1 # Store 'Hello world!' at the top of the stack
2 ADDI $sp, $sp, -13
3 ADDI $t0, $zero, 72 # H
4 SB $t0, 0($sp)
5 ADDI $t0, $zero, 101 # e
6 SB $t0, 1($sp)
7 ADDI $t0, $zero, 108 # l
8 SB $t0, 2($sp)
9 ADDI $t0, $zero, 108 # i
10 SB $t0, 3($sp)
11 ADDI $t0, $zero, 111 # o
12 SB $t0, 4($sp)
13 ADDI $t0, $zero, 32 # (space)
14 SB $t0, 5($sp)
15 ADDI $t0, $zero, 119 # w
16 SB $t0, 6($sp)
17 ADDI $t0, $zero, 111 # o
18 SB $t0, 7($sp)
19 ADDI $t0, $zero, 114 # r
20 SB $t0, 8($sp)
21 ADDI $t0, $zero, 108 # l
22 SB $t0, 9($sp)
23 ADDI $t0, $zero, 100 # d
24 SB $t0, 10($sp)
25 ADDI $t0, $zero, 33 # !
26 SB $t0, 11($sp)
27 ADDI $t0, $zero, 0 # (null)
28 SB $t0, 12($sp)
29
30 ADDI $v0, $zero, 4 # 4 is for print string
31 ADDI $a0, $sp, 0
32 syscall          # print to the log
```

User Guide | Unit Tests | Docs

Step Run  Enable auto switching

S	T	A	V	Stack	Log
s0:				10	
s1:				9	
s2:				9	
s3:				22	
s4:				696	
s5:				976	
s6:				927	
s7:				418	

(WeMIPS)

# WeMIPS

The screenshot shows the WeMIPS debugger interface. At the top, there are tabs for 'Live', '3', 'Data', and 'ShowWhile Device'. Below these are navigation links: 'Addition Doubler', 'Stax', 'Looper', 'Stack Test', and 'Hello World'. There are also buttons for 'Code Gen Save String', 'Interactive', 'Binary2 Decimal', and 'Decimal2 Binary'. A 'Debug' button is also present.

The main area displays assembly code:

```
# Store 'Hello world!' at the top of the stack
    .text
    .globl _start
_start:
    li $t0, 0x484f4c4c        # 'Hello'
    li $t1, 0x6f6f6f6f        # ', '
    li $t2, 0x6e6e6e6e        # ' '
    li $t3, 0x65656565        # ' '
    li $t4, 0x64646464        # ' '
    li $t5, 0x63636363        # ' '
    li $t6, 0x62626262        # ' '
    li $t7, 0x61616161        # ' '
    li $t8, 0x60606060        # ' '
    li $t9, 0x65726f6d646564 # 'Hello world!\n'
    li $t10, 0x0               # null
    li $t11, 0x1               # r
    li $t12, 0x2               # s
    li $t13, 0x3               # t
    li $t14, 0x4               # u
    li $t15, 0x5               # v
    li $t16, 0x6               # w
    li $t17, 0x7               # x
    li $t18, 0x8               # y
    li $t19, 0x9               # z
    li $t20, 0xa              # !
    li $t21, 0xb              # %
    li $t22, 0xc              # *
    li $t23, 0xd              # /
    li $t24, 0xe              # .
    li $t25, 0xf              # ,
    li $t26, 0x10             # ;
    li $t27, 0x11             # :
    li $t28, 0x12             # .
    li $t29, 0x13             # ?
    li $t30, 0x14             # =
    li $t31, 0x15             # +
    li $t32, 0x16             # -
    li $t33, 0x17             # *
    li $t34, 0x18             # /
    li $t35, 0x19             # %
    li $t36, 0x1a             # ^
    li $t37, 0x1b             # |
    li $t38, 0x1c             # ~
    li $t39, 0x1d             # [
    li $t40, 0x1e             # ]
    li $t41, 0x1f             # {
    li $t42, 0x100            # }
    li $t43, 0x101            # ;
    li $t44, 0x102            # :
    li $t45, 0x103            # .
    li $t46, 0x104            # ?
    li $t47, 0x105            # =
    li $t48, 0x106            # +
    li $t49, 0x107            # -
    li $t50, 0x108            # *
    li $t51, 0x109            # /
    li $t52, 0x10a            # %
    li $t53, 0x10b            # ^
    li $t54, 0x10c            # |
    li $t55, 0x10d            # ~
    li $t56, 0x10e            # [
    li $t57, 0x10f            # ]
    li $t58, 0x100            # {
    li $t59, 0x101            # }
    li $t60, 0x102            # ;
    li $t61, 0x103            # :
    li $t62, 0x104            # .
    li $t63, 0x105            # ?
    li $t64, 0x106            # =
    li $t65, 0x107            # +
    li $t66, 0x108            # -
    li $t67, 0x109            # *
    li $t68, 0x10a            # /
    li $t69, 0x10b            # %
    li $t70, 0x10c            # ^
    li $t71, 0x10d            # |
    li $t72, 0x10e            # ~
    li $t73, 0x10f            # [
    li $t74, 0x100            # ]
    li $t75, 0x101            # {
    li $t76, 0x102            # }
    li $t77, 0x103            # ;
    li $t78, 0x104            # :
    li $t79, 0x105            # .
    li $t80, 0x106            # ?
    li $t81, 0x107            # =
    li $t82, 0x108            # +
    li $t83, 0x109            # -
    li $t84, 0x10a            # *
    li $t85, 0x10b            # /
    li $t86, 0x10c            # %
    li $t87, 0x10d            # ^
    li $t88, 0x10e            # |
    li $t89, 0x10f            # ~
    li $t90, 0x100            # [
    li $t91, 0x101            # ]
    li $t92, 0x102            # {
    li $t93, 0x103            # }
    li $t94, 0x104            # ;
    li $t95, 0x105            # :
    li $t96, 0x106            # .
    li $t97, 0x107            # ?
    li $t98, 0x108            # =
    li $t99, 0x109            # +
    li $t100, 0x10a           # -
    li $t101, 0x10b           # *
    li $t102, 0x10c           # /
    li $t103, 0x10d           # %
    li $t104, 0x10e           # ^
    li $t105, 0x10f           # |
    li $t106, 0x100           # ~
    li $t107, 0x101           # [
    li $t108, 0x102           # ]
    li $t109, 0x103           # {
    li $t110, 0x104           # }
    li $t111, 0x105           # ;
    li $t112, 0x106           # :
    li $t113, 0x107           # .
    li $t114, 0x108           # ?
    li $t115, 0x109           # =
    li $t116, 0x10a           # +
    li $t117, 0x10b           # -
    li $t118, 0x10c           # *
    li $t119, 0x10d           # /
    li $t120, 0x10e           # %
    li $t121, 0x10f           # ^
    li $t122, 0x100           # |
    li $t123, 0x101           # ~
    li $t124, 0x102           # [
    li $t125, 0x103           # ]
    li $t126, 0x104           # {
    li $t127, 0x105           # }
    li $t128, 0x106           # ;
    li $t129, 0x107           # :
    li $t130, 0x108           # .
    li $t131, 0x109           # ?
    li $t132, 0x10a           # =
    li $t133, 0x10b           # +
    li $t134, 0x10c           # -
    li $t135, 0x10d           # *
    li $t136, 0x10e           # /
    li $t137, 0x10f           # %
    li $t138, 0x100           # ^
    li $t139, 0x101           # |
    li $t140, 0x102           # ~
    li $t141, 0x103           # [
    li $t142, 0x104           # ]
    li $t143, 0x105           # {
    li $t144, 0x106           # }
    li $t145, 0x107           # ;
    li $t146, 0x108           # :
    li $t147, 0x109           # .
    li $t148, 0x10a           # ?
    li $t149, 0x10b           # =
    li $t150, 0x10c           # +
    li $t151, 0x10d           # -
    li $t152, 0x10e           # *
    li $t153, 0x10f           # /
    li $t154, 0x100           # %
    li $t155, 0x101           # ^
    li $t156, 0x102           # |
    li $t157, 0x103           # ~
    li $t158, 0x104           # [
    li $t159, 0x105           # ]
    li $t160, 0x106           # {
    li $t161, 0x107           # }
    li $t162, 0x108           # ;
    li $t163, 0x109           # :
    li $t164, 0x10a           # .
    li $t165, 0x10b           # ?
    li $t166, 0x10c           # =
    li $t167, 0x10d           # +
    li $t168, 0x10e           # -
    li $t169, 0x10f           # *
    li $t170, 0x100           # /
    li $t171, 0x101           # %
    li $t172, 0x102           # ^
    li $t173, 0x103           # |
    li $t174, 0x104           # ~
    li $t175, 0x105           # [
    li $t176, 0x106           # ]
    li $t177, 0x107           # {
    li $t178, 0x108           # }
    li $t179, 0x109           # ;
    li $t180, 0x10a           # :
    li $t181, 0x10b           # .
    li $t182, 0x10c           # ?
    li $t183, 0x10d           # =
    li $t184, 0x10e           # +
    li $t185, 0x10f           # -
    li $t186, 0x100           # *
    li $t187, 0x101           # /
    li $t188, 0x102           # %
    li $t189, 0x103           # ^
    li $t190, 0x104           # |
    li $t191, 0x105           # ~
    li $t192, 0x106           # [
    li $t193, 0x107           # ]
    li $t194, 0x108           # {
    li $t195, 0x109           # }
    li $t196, 0x10a           # ;
    li $t197, 0x10b           # :
    li $t198, 0x10c           # .
    li $t199, 0x10d           # ?
    li $t200, 0x10e           # =
    li $t201, 0x10f           # +
    li $t202, 0x100           # -
    li $t203, 0x101           # *
    li $t204, 0x102           # /
    li $t205, 0x103           # %
    li $t206, 0x104           # ^
    li $t207, 0x105           # |
    li $t208, 0x106           # ~
    li $t209, 0x107           # [
    li $t210, 0x108           # ]
    li $t211, 0x109           # {
    li $t212, 0x10a           # }
    li $t213, 0x10b           # ;
    li $t214, 0x10c           # :
    li $t215, 0x10d           # .
    li $t216, 0x10e           # ?
    li $t217, 0x10f           # =
    li $t218, 0x100           # +
    li $t219, 0x101           # -
    li $t220, 0x102           # *
    li $t221, 0x103           # /
    li $t222, 0x104           # %
    li $t223, 0x105           # ^
    li $t224, 0x106           # |
    li $t225, 0x107           # ~
    li $t226, 0x108           # [
    li $t227, 0x109           # ]
    li $t228, 0x10a           # {
    li $t229, 0x10b           # }
    li $t230, 0x10c           # ;
    li $t231, 0x10d           # :
    li $t232, 0x10e           # .
    li $t233, 0x10f           # ?
    li $t234, 0x100           # =
    li $t235, 0x101           # +
    li $t236, 0x102           # -
    li $t237, 0x103           # *
    li $t238, 0x104           # /
    li $t239, 0x105           # %
    li $t240, 0x106           # ^
    li $t241, 0x107           # |
    li $t242, 0x108           # ~
    li $t243, 0x109           # [
    li $t244, 0x10a           # ]
    li $t245, 0x10b           # {
    li $t246, 0x10c           # }
    li $t247, 0x10d           # ;
    li $t248, 0x10e           # :
    li $t249, 0x10f           # .
    li $t250, 0x100           # ?
    li $t251, 0x101           # =
    li $t252, 0x102           # +
    li $t253, 0x103           # -
    li $t254, 0x104           # *
    li $t255, 0x105           # /
    li $t256, 0x106           # %
    li $t257, 0x107           # ^
    li $t258, 0x108           # |
    li $t259, 0x109           # ~
    li $t260, 0x10a           # [
    li $t261, 0x10b           # ]
    li $t262, 0x10c           # {
    li $t263, 0x10d           # }
    li $t264, 0x10e           # ;
    li $t265, 0x10f           # :
    li $t266, 0x100           # .
    li $t267, 0x101           # ?
    li $t268, 0x102           # =
    li $t269, 0x103           # +
    li $t270, 0x104           # -
    li $t271, 0x105           # *
    li $t272, 0x106           # /
    li $t273, 0x107           # %
    li $t274, 0x108           # ^
    li $t275, 0x109           # |
    li $t276, 0x10a           # ~
    li $t277, 0x10b           # [
    li $t278, 0x10c           # ]
    li $t279, 0x10d           # {
    li $t280, 0x10e           # }
    li $t281, 0x10f           # ;
    li $t282, 0x100           # :
    li $t283, 0x101           # .
    li $t284, 0x102           # ?
    li $t285, 0x103           # =
    li $t286, 0x104           # +
    li $t287, 0x105           # -
    li $t288, 0x106           # *
    li $t289, 0x107           # /
    li $t290, 0x108           # %
    li $t291, 0x109           # ^
    li $t292, 0x10a           # |
    li $t293, 0x10b           # ~
    li $t294, 0x10c           # [
    li $t295, 0x10d           # ]
    li $t296, 0x10e           # {
    li $t297, 0x10f           # }
    li $t298, 0x100           # ;
    li $t299, 0x101           # :
    li $t300, 0x102           # .
    li $t301, 0x103           # ?
    li $t302, 0x104           # =
    li $t303, 0x105           # +
    li $t304, 0x106           # -
    li $t305, 0x107           # *
    li $t306, 0x108           # /
    li $t307, 0x109           # %
    li $t308, 0x10a           # ^
    li $t309, 0x10b           # |
    li $t310, 0x10c           # ~
    li $t311, 0x10d           # [
    li $t312, 0x10e           # ]
    li $t313, 0x10f           # {
    li $t314, 0x100           # }
    li $t315, 0x101           # ;
    li $t316, 0x102           # :
    li $t317, 0x103           # .
    li $t318, 0x104           # ?
    li $t319, 0x105           # =
    li $t320, 0x106           # +
    li $t321, 0x107           # -
    li $t322, 0x108           # *
    li $t323, 0x109           # /
    li $t324, 0x10a           # %
    li $t325, 0x10b           # ^
    li $t326, 0x10c           # |
    li $t327, 0x10d           # ~
    li $t328, 0x10e           # [
    li $t329, 0x10f           # ]
    li $t330, 0x100           # {
    li $t331, 0x101           # }
    li $t332, 0x102           # ;
    li $t333, 0x103           # :
    li $t334, 0x104           # .
    li $t335, 0x105           # ?
    li $t336, 0x106           # =
    li $t337, 0x107           # +
    li $t338, 0x108           # -
    li $t339, 0x109           # *
    li $t340, 0x10a           # /
    li $t341, 0x10b           # %
    li $t342, 0x10c           # ^
    li $t343, 0x10d           # |
    li $t344, 0x10e           # ~
    li $t345, 0x10f           # [
    li $t346, 0x100           # ]
    li $t347, 0x101           # {
    li $t348, 0x102           # }
    li $t349, 0x103           # ;
    li $t350, 0x104           # :
    li $t351, 0x105           # .
    li $t352, 0x106           # ?
    li $t353, 0x107           # =
    li $t354, 0x108           # +
    li $t355, 0x109           # -
    li $t356, 0x10a           # *
    li $t357, 0x10b           # /
    li $t358, 0x10c           # %
    li $t359, 0x10d           # ^
    li $t360, 0x10e           # |
    li $t361, 0x10f           # ~
    li $t362, 0x100           # [
    li $t363, 0x101           # ]
    li $t364, 0x102           # {
    li $t365, 0x103           # }
    li $t366, 0x104           # ;
    li $t367, 0x105           # :
    li $t368, 0x106           # .
    li $t369, 0x107           # ?
    li $t370, 0x108           # =
    li $t371, 0x109           # +
    li $t372, 0x10a           # -
    li $t373, 0x10b           # *
    li $t374, 0x10c           # /
    li $t375, 0x10d           # %
    li $t376, 0x10e           # ^
    li $t377, 0x10f           # |
    li $t378, 0x100           # ~
    li $t379, 0x101           # [
    li $t380, 0x102           # ]
    li $t381, 0x103           # {
    li $t382, 0x104           # }
    li $t383, 0x105           # ;
    li $t384, 0x106           # :
    li $t385, 0x107           # .
    li $t386, 0x108           # ?
    li $t387, 0x109           # =
    li $t388, 0x10a           # +
    li $t389, 0x10b           # -
    li $t390, 0x10c           # *
    li $t391, 0x10d           # /
    li $t392, 0x10e           # %
    li $t393, 0x10f           # ^
    li $t394, 0x100           # |
    li $t395, 0x101           # ~
    li $t396, 0x102           # [
    li $t397, 0x103           # ]
    li $t398, 0x104           # {
    li $t399, 0x105           # }
    li $t400, 0x106           # ;
    li $t401, 0x107           # :
    li $t402, 0x108           # .
    li $t403, 0x109           # ?
    li $t404, 0x10a           # =
    li $t405, 0x10b           # +
    li $t406, 0x10c           # -
    li $t407, 0x10d           # *
    li $t408, 0x10e           # /
    li $t409, 0x10f           # %
    li $t410, 0x100           # ^
    li $t411, 0x101           # |
    li $t412, 0x102           # ~
    li $t413, 0x103           # [
    li $t414, 0x104           # ]
    li $t415, 0x105           # {
    li $t416, 0x106           # }
    li $t417, 0x107           # ;
    li $t418, 0x108           # :
    li $t419, 0x109           # .
    li $t420, 0x10a           # ?
    li $t421, 0x10b           # =
    li $t422, 0x10c           # +
    li $t423, 0x10d           # -
    li $t424, 0x10e           # *
    li $t425, 0x10f           # /
    li $t426, 0x100           # %
    li $t427, 0x101           # ^
    li $t428, 0x102           # |
    li $t429, 0x103           # ~
    li $t430, 0x104           # [
    li $t431, 0x105           # ]
    li $t432, 0x106           # {
    li $t433, 0x107           # }
    li $t434, 0x108           # ;
    li $t435, 0x109           # :
    li $t436, 0x10a           # .
    li $t437, 0x10b           # ?
    li $t438, 0x10c           # =
    li $t439, 0x10d           # +
    li $t440, 0x10e           # -
    li $t441, 0x10f           # *
    li $t442, 0x100           # /
    li $t443, 0x101           # %
    li $t444, 0x102           # ^
    li $t445, 0x103           # |
    li $t446, 0x104           # ~
    li $t447, 0x105           # [
    li $t448, 0x106           # ]
    li $t449, 0x107           # {
    li $t450, 0x108           # }
    li $t451, 0x109           # ;
    li $t452, 0x10a           # :
    li $t453, 0x10b           # .
    li $t454, 0x10c           # ?
    li $t455, 0x10d           # =
    li $t456, 0x10e           # +
    li $t457, 0x10f           # -
    li $t458, 0x100           # *
    li $t459, 0x101           # /
    li $t460, 0x102           # %
    li $t461, 0x103           # ^
    li $t462, 0x104           # |
    li $t463, 0x105           # ~
    li $t464, 0x106           # [
    li $t465, 0x107           # ]
    li $t466, 0x108           # {
    li $t467, 0x109           # }
    li $t468, 0x10a           # ;
    li $t469, 0x10b           # :
    li $t470, 0x10c           # .
    li $t471, 0x10d           # ?
    li $t472, 0x10e           # =
    li $t473, 0x10f           # +
    li $t474, 0x100           # -
    li $t475, 0x101           # *
    li $t476, 0x102           # /
    li $t477, 0x103           # %
    li $t478, 0x104           # ^
    li $t479, 0x105           # |
    li $t480, 0x106           # ~
    li $t481, 0x107           # [
    li $t482, 0x108           # ]
    li $t483, 0x109           # {
    li $t484, 0x10a           # }
    li $t485, 0x10b           # ;
    li $t486, 0x10c           # :
    li $t487, 0x10d           # .
    li $t488, 0x10e           # ?
    li $t489, 0x10f           # =
    li $t490, 0x100           # +
    li $t491, 0x101           # -
    li $t492, 0x102           # *
    li $t493, 0x103           # /
    li $t494, 0x104           # %
    li $t495, 0x105           # ^
    li $t496, 0x106           # |
    li $t497, 0x107           # ~
    li $t498, 0x108           # [
    li $t499, 0x109           # ]
    li $t500, 0x100           # {
    li $t501, 0x101           # }
    li $t502, 0x102           # ;
    li $t503, 0x103           # :
    li $t504, 0x104           # .
    li $t505, 0x105           # ?
    li $t506, 0x106           # =
    li $t507, 0x107           # +
    li $t508, 0x108           # -
    li $t509, 0x109           # *
    li $t510, 0x10a           # /
    li $t511, 0x10b           # %
    li $t512, 0x10c           # ^
    li $t513, 0x10d           # |
    li $t514, 0x10e           # ~
    li $t515, 0x10f           # [
    li $t516, 0x100           # ]
    li $t517, 0x101           # {
    li $t518, 0x102           # }
    li $t519, 0x103           # ;
    li $t520, 0x104           # :
    li $t521, 0x105           # .
    li $t522, 0x106           # ?
    li $t523, 0x107           # =
    li $t524, 0x108           # +
    li $t525, 0x109           # -
    li $t526, 0x10a           # *
    li $t527, 0x10b           # /
    li $t528, 0x10c           # %
    li $t529, 0x10d           # ^
    li $t530, 0x10e           # |
    li $t531, 0x10f           # ~
    li $t532, 0x100           # [
    li $t533, 0x101           # ]
    li $t534, 0x102           # {
    li $t535, 0x103           # }
    li $t536, 0x104           # ;
    li $t537, 0x105           # :
    li $t538, 0x106           # .
    li $t539, 0x107           # ?
    li $t540, 0x108           # =
    li $t541, 0x109           # +
    li $t542, 0x10a           # -
    li $t543, 0x10b           # *
    li $t544, 0x10c           # /
    li $t545, 0x10d           # %
    li $t546, 0x10e           # ^
    li $t547, 0x10f           # |
    li $t548, 0x100           # ~
    li $t549, 0x101           # [
    li $t550, 0x102           # ]
    li $t551, 0x103           # {
    li $t552, 0x104           # }
    li $t553, 0x105           # ;
    li $t554, 0x106           # :
    li $t555, 0x107           # .
    li $t556, 0x108           # ?
    li $t557, 0x109           # =
    li $t558, 0x10a           # +
    li $t559, 0x10b           # -
    li $t560, 0x10c           # *
    li $t561, 0x10d           # /
    li $t562, 0x10e           # %
    li $t563, 0x10f           # ^
    li $t564, 0x100           # |
    li $t565, 0x101           # ~
    li $t566, 0x102           # [
    li $t567, 0x103           # ]
    li $t568, 0x104           # {
    li $t569, 0x105           # }
    li $t570, 0x106           # ;
    li $t571, 0x107           # :
    li $t572, 0x108           # .
    li $t573, 0x109           # ?
    li $t574, 0x10a           # =
    li $t575, 0x10b           # +
    li $t576, 0x10c           # -
    li $t577, 0x10d           # *
    li $t578, 0x10e           # /
    li $t579, 0x10f           # %
    li $t580, 0x100           # ^
    li $t581, 0x101           # |
    li $t582, 0x102           # ~
    li $t583, 0x103           # [
    li $t584, 0x104           # ]
    li $t585, 0x105           # {
    li $t586, 0x106           # }
    li $t587, 0x107           # ;
    li $t588, 0x108           # :
    li $t589, 0x109           # .
    li $t590, 0x10a           # ?
    li $t591, 0x10b           # =
    li $t592, 0x10c           # +
    li $t593, 0x10d           # -
    li $t594, 0x10e           # *
    li $t595, 0x10f           # /
    li $t596, 0x100           # %
    li $t597, 0x101           # ^
    li $t598, 0x102           # |
    li $t599, 0x103           # ~
    li $t600, 0x104           # [
    li $t601, 0x105           # ]
    li $t602, 0x106           # {
    li $t603, 0x107           # }
    li $t604, 0x108           # ;
    li $t605, 0x109           # :
    li $t606, 0x10a           # .
    li $t607, 0x10b           # ?
    li $t608, 0x10c           # =
    li $t609, 0x10d           # +
    li $t610, 0x10e           # -
    li $t611, 0x10f           # *
    li $t612, 0x100           # /
    li $t613, 0x101           # %
    li $t614, 0x102           # ^
    li $t615, 0x103           # |
    li $t616, 0x104           # ~
    li $t617, 0x105           # [
    li $t618, 0x106           # ]
    li $t619, 0x107           # {
    li $t620, 0x108           # }
    li $t621, 0x109           # ;
    li $t622, 0x10a           # :
    li $t623, 0x10b           # .
    li $t624, 0x10c           # ?
    li $t625, 0x10d           # =
    li $t626, 0x10e           # +
    li $t627, 0x10f           # -
    li $t628, 0x100           # *
    li $t629, 0x101           # /
    li $t630, 0x102           # %
    li $t631, 0x103           # ^
    li $t632, 0x104           # |
    li $t633, 0x105           # ~
    li $t634, 0x106           # [
    li $t635, 0x107           # ]
    li $t636, 0x108           # {
    li $t637, 0x109           # }
    li $t638, 0x10a           # ;
    li $t639, 0x10b           # :
    li $t640, 0x10c           # .
    li $t641, 0x10d           # ?
    li $t642, 0x10e           # =
    li $t643, 0x10f           # +
    li $t644, 0x100           # -
    li $t645, 0x101           # *
    li $t646, 0x102           # /
    li $t647, 0x103           # %
    li $t648, 0x104           # ^
    li $t649, 0x105           # |
    li $t650, 0x106           # ~
    li $t651, 0x107           # [
    li $t652, 0x108           # ]
    li $t653, 0x109           # {
    li $t654, 0x10a           # }
    li $t655, 0x10b           # ;
    li $t656, 0x10c           # :
    li $t657, 0x10d           # .
    li $t658, 0x10e           # ?
    li $t659, 0x10f           # =
    li $t660, 0x100           # +
    li $t661, 0x101           # -
    li $t662, 0x102           # *
    li $t663, 0x103           # /
    li $t664, 0x104           # %
    li $t665, 0x105           # ^
    li $t666, 0x106           # |
    li $t667, 0x107           # ~
    li $t668, 0x108           # [
    li $t669, 0x109           # ]
    li $t670, 0x100           # {
    li $t671, 0x101           # }
    li $t672, 0x102           # ;
    li $t673, 0x103           # :
    li $t674, 0x104           # .
    li $t675, 0x105           # ?
    li $t676, 0x106           # =
    li $t677, 0x107           # +
    li $t678, 0x108           # -
    li $t679, 0x109           # *
    li $t680, 0x10a           # /
    li $t681, 0x10b           # %
    li $t682, 0x10c           # ^
    li $t683, 0x10d           # |
    li $t684, 0x10e           # ~
    li $t685, 0x10f           # [
    li $t686, 0x100           # ]
    li $t687, 0x101           # {
    li $t688, 0x102           # }
    li $t689, 0x103           # ;
    li $t690, 0x104           # :
    li $t691, 0x105           # .
    li $t692, 0x106           # ?
    li $t693, 0x107           # =
    li $t694, 0x108           # +
    li $t695, 0x109           # -
    li $t696, 0x10a           # *
    li $t697, 0x10b           # /
    li $t698, 0x10c           # %
    li $t699, 0x10d           # ^
    li $t700, 0x10e           # |
    li $t701, 0x10f           # ~
    li $t702, 0x100           # [
    li $t703, 0x101           # ]
    li $t704, 0x102           # {
    li $t705, 0x103           # }
    li $t706, 0x104           # ;
    li $t707, 0x105           # :
    li $t708, 0x106           # .
    li $t709, 0x107           # ?
    li $t710, 0x108           # =
    li $t711, 0x109           # +
    li $t712, 0x10a           # -
    li $t713, 0x10b           # *
    li $t714, 0x10c           # /
    li $t715, 0x10d           # %
    li $t716, 0x10e           # ^
    li $t717, 0x10f           # |
    li
```

# MIPS Commands

- **Registers:** locations for storing information that can be quickly accessed.

# MIPS Commands

The screenshot shows a MIPS assembly debugger interface. At the top, there are tabs for "Show/Hide Demo", "User Guide", "Unit Tests", and "Docs". Below the tabs are buttons for "Addition", "Subtraction", "Max", "Looper", "Stack Test", and "Hello World". There are also buttons for "Code Gen", "Save String", "Interactive", "Decimal Decimal", and "Decimal Binary". A "Debug" button is also present.

The main area displays assembly code:

```
1 # Change '$HelloWorld' at the top of the stack
2
3 .data
4 $HelloWorld: .asciiz "$HelloWorld"
5
6 .text
7 la $t0, $HelloWorld
8 li $t1, 115901
9 sb $t1, ($t0)
10 li $t1, 115902
11 sb $t1, ($t0 + 1)
12 li $t1, 115903
13 sb $t1, ($t0 + 2)
14 li $t1, 115904
15 sb $t1, ($t0 + 3)
16 li $t1, 115905
17 sb $t1, ($t0 + 4)
18 li $t1, 115906
19 sb $t1, ($t0 + 5)
20 li $t1, 115907
21 sb $t1, ($t0 + 6)
22 li $t1, 115908
23 sb $t1, ($t0 + 7)
24 li $t1, 115909
25 sb $t1, ($t0 + 8)
26 li $t1, 115910
27 sb $t1, ($t0 + 9)
28 li $t1, 115911
29 sb $t1, ($t0 + 10)
30 li $t1, 115912
31 sb $t1, ($t0 + 11)
32 li $t1, 115913
33 sb $t1, ($t0 + 12)
34 li $t1, 115914
35 sb $t1, ($t0 + 13)
36 li $t1, 115915
37 sb $t1, ($t0 + 14)
38 li $t1, 115916
39 sb $t1, ($t0 + 15)
40 li $t1, 115917
41 sb $t1, ($t0 + 16)
42 li $t1, 115918
43 sb $t1, ($t0 + 17)
44 li $t1, 115919
45 sb $t1, ($t0 + 18)
46 li $t1, 115920
47 sb $t1, ($t0 + 19)
48 li $t1, 115921
49 sb $t1, ($t0 + 20)
50 li $t1, 115922
51 sb $t1, ($t0 + 21)
52 li $t1, 115923
53 sb $t1, ($t0 + 22)
54 li $t1, 115924
55 sb $t1, ($t0 + 23)
56 li $t1, 115925
57 sb $t1, ($t0 + 24)
58 li $t1, 115926
59 sb $t1, ($t0 + 25)
60 li $t1, 115927
61 sb $t1, ($t0 + 26)
62 li $t1, 115928
63 sb $t1, ($t0 + 27)
64 li $t1, 115929
65 sb $t1, ($t0 + 28)
66 li $t1, 115930
67 sb $t1, ($t0 + 29)
68 li $t1, 115931
69 sb $t1, ($t0 + 30)
70 li $t1, 115932
71 sb $t1, ($t0 + 31)
72 li $t1, 115933
73 sb $t1, ($t0 + 32) # (space)
74 li $t1, 115934
75 sb $t1, ($t0 + 33) # (space)
76 li $t1, 115935
77 sb $t1, ($t0 + 34) # (space)
78 li $t1, 115936
79 sb $t1, ($t0 + 35) # (space)
80 li $t1, 115937
81 sb $t1, ($t0 + 36) # (space)
82 li $t1, 115938
83 sb $t1, ($t0 + 37) # (space)
84 li $t1, 115939
85 sb $t1, ($t0 + 38) # (space)
86 li $t1, 115940
87 sb $t1, ($t0 + 39) # (space)
88 li $t1, 115941
89 sb $t1, ($t0 + 40) # (space)
90 li $t1, 115942
91 sb $t1, ($t0 + 41) # (space)
92 li $t1, 115943
93 sb $t1, ($t0 + 42) # (space)
94 li $t1, 115944
95 sb $t1, ($t0 + 43) # (space)
96 li $t1, 115945
97 sb $t1, ($t0 + 44) # (space)
98 li $t1, 115946
99 sb $t1, ($t0 + 45) # (space)
100 li $t1, 115947
101 sb $t1, ($t0 + 46) # (space)
102 li $t1, 115948
103 sb $t1, ($t0 + 47) # (space)
104 li $t1, 115949
105 sb $t1, ($t0 + 48) # (space)
106 li $t1, 115950
107 sb $t1, ($t0 + 49) # (space)
108 li $t1, 115951
109 sb $t1, ($t0 + 50) # (space)
110 li $t1, 115952
111 sb $t1, ($t0 + 51) # (space)
112 li $t1, 115953
113 sb $t1, ($t0 + 52) # (space)
114 li $t1, 115954
115 sb $t1, ($t0 + 53) # (space)
116 li $t1, 115955
117 sb $t1, ($t0 + 54) # (space)
118 li $t1, 115956
119 sb $t1, ($t0 + 55) # (space)
120 li $t1, 115957
121 sb $t1, ($t0 + 56) # (space)
122 li $t1, 115958
123 sb $t1, ($t0 + 57) # (space)
124 li $t1, 115959
125 sb $t1, ($t0 + 58) # (space)
126 li $t1, 115960
127 sb $t1, ($t0 + 59) # (space)
128 li $t1, 115961
129 sb $t1, ($t0 + 60) # (space)
130 li $t1, 115962
131 sb $t1, ($t0 + 61) # (space)
132 li $t1, 115963
133 sb $t1, ($t0 + 62) # (space)
134 li $t1, 115964
135 sb $t1, ($t0 + 63) # (space)
136 li $t1, 115965
137 sb $t1, ($t0 + 64) # (space)
138 li $t1, 115966
139 sb $t1, ($t0 + 65) # (space)
140 li $t1, 115967
141 sb $t1, ($t0 + 66) # (space)
142 li $t1, 115968
143 sb $t1, ($t0 + 67) # (space)
144 li $t1, 115969
145 sb $t1, ($t0 + 68) # (space)
146 li $t1, 115970
147 sb $t1, ($t0 + 69) # (space)
148 li $t1, 115971
149 sb $t1, ($t0 + 70) # (space)
150 li $t1, 115972
151 sb $t1, ($t0 + 71) # (space)
152 li $t1, 115973
153 sb $t1, ($t0 + 72) # (space)
154 li $t1, 115974
155 sb $t1, ($t0 + 73) # (space)
156 li $t1, 115975
157 sb $t1, ($t0 + 74) # (space)
158 li $t1, 115976
159 sb $t1, ($t0 + 75) # (space)
160 li $t1, 115977
161 sb $t1, ($t0 + 76) # (space)
162 li $t1, 115978
163 sb $t1, ($t0 + 77) # (space)
164 li $t1, 115979
165 sb $t1, ($t0 + 78) # (space)
166 li $t1, 115980
167 sb $t1, ($t0 + 79) # (space)
168 li $t1, 115981
169 sb $t1, ($t0 + 80) # (space)
170 li $t1, 115982
171 sb $t1, ($t0 + 81) # (space)
172 li $t1, 115983
173 sb $t1, ($t0 + 82) # (space)
174 li $t1, 115984
175 sb $t1, ($t0 + 83) # (space)
176 li $t1, 115985
177 sb $t1, ($t0 + 84) # (space)
178 li $t1, 115986
179 sb $t1, ($t0 + 85) # (space)
180 li $t1, 115987
181 sb $t1, ($t0 + 86) # (space)
182 li $t1, 115988
183 sb $t1, ($t0 + 87) # (space)
184 li $t1, 115989
185 sb $t1, ($t0 + 88) # (space)
186 li $t1, 115990
187 sb $t1, ($t0 + 89) # (space)
188 li $t1, 115991
189 sb $t1, ($t0 + 90) # (space)
190 li $t1, 115992
191 sb $t1, ($t0 + 91) # (space)
192 li $t1, 115993
193 sb $t1, ($t0 + 92) # (space)
194 li $t1, 115994
195 sb $t1, ($t0 + 93) # (space)
196 li $t1, 115995
197 sb $t1, ($t0 + 94) # (space)
198 li $t1, 115996
199 sb $t1, ($t0 + 95) # (space)
200 li $t1, 115997
201 sb $t1, ($t0 + 96) # (space)
202 li $t1, 115998
203 sb $t1, ($t0 + 97) # (space)
204 li $t1, 115999
205 sb $t1, ($t0 + 98) # (space)
206 li $t1, 115900
207 sb $t1, ($t0 + 99) # (space)
208 li $t1, 115901
209 sb $t1, ($t0 + 100) # (space)
210 li $t1, 115902
211 sb $t1, ($t0 + 101) # (space)
212 li $t1, 115903
213 sb $t1, ($t0 + 102) # (space)
214 li $t1, 115904
215 sb $t1, ($t0 + 103) # (space)
216 li $t1, 115905
217 sb $t1, ($t0 + 104) # (space)
218 li $t1, 115906
219 sb $t1, ($t0 + 105) # (space)
220 li $t1, 115907
221 sb $t1, ($t0 + 106) # (space)
222 li $t1, 115908
223 sb $t1, ($t0 + 107) # (space)
224 li $t1, 115909
225 sb $t1, ($t0 + 108) # (space)
226 li $t1, 115910
227 sb $t1, ($t0 + 109) # (space)
228 li $t1, 115911
229 sb $t1, ($t0 + 110) # (space)
230 li $t1, 115912
231 sb $t1, ($t0 + 111) # (space)
232 li $t1, 115913
233 sb $t1, ($t0 + 112) # (space)
234 li $t1, 115914
235 sb $t1, ($t0 + 113) # (space)
236 li $t1, 115915
237 sb $t1, ($t0 + 114) # (space)
238 li $t1, 115916
239 sb $t1, ($t0 + 115) # (space)
240 li $t1, 115917
241 sb $t1, ($t0 + 116) # (space)
242 li $t1, 115918
243 sb $t1, ($t0 + 117) # (space)
244 li $t1, 115919
245 sb $t1, ($t0 + 118) # (space)
246 li $t1, 115920
247 sb $t1, ($t0 + 119) # (space)
248 li $t1, 115921
249 sb $t1, ($t0 + 120) # (space)
250 li $t1, 115922
251 sb $t1, ($t0 + 121) # (space)
252 li $t1, 115923
253 sb $t1, ($t0 + 122) # (space)
254 li $t1, 115924
255 sb $t1, ($t0 + 123) # (space)
256 li $t1, 115925
257 sb $t1, ($t0 + 124) # (space)
258 li $t1, 115926
259 sb $t1, ($t0 + 125) # (space)
260 li $t1, 115927
261 sb $t1, ($t0 + 126) # (space)
262 li $t1, 115928
263 sb $t1, ($t0 + 127) # (space)
264 li $t1, 115929
265 sb $t1, ($t0 + 128) # (space)
266 li $t1, 115930
267 sb $t1, ($t0 + 129) # (space)
268 li $t1, 115931
269 sb $t1, ($t0 + 130) # (space)
270 li $t1, 115932
271 sb $t1, ($t0 + 131) # (space)
272 li $t1, 115933
273 sb $t1, ($t0 + 132) # (space)
274 li $t1, 115934
275 sb $t1, ($t0 + 133) # (space)
276 li $t1, 115935
277 sb $t1, ($t0 + 134) # (space)
278 li $t1, 115936
279 sb $t1, ($t0 + 135) # (space)
280 li $t1, 115937
281 sb $t1, ($t0 + 136) # (space)
282 li $t1, 115938
283 sb $t1, ($t0 + 137) # (space)
284 li $t1, 115939
285 sb $t1, ($t0 + 138) # (space)
286 li $t1, 115940
287 sb $t1, ($t0 + 139) # (space)
288 li $t1, 115941
289 sb $t1, ($t0 + 140) # (space)
290 li $t1, 115942
291 sb $t1, ($t0 + 141) # (space)
292 li $t1, 115943
293 sb $t1, ($t0 + 142) # (space)
294 li $t1, 115944
295 sb $t1, ($t0 + 143) # (space)
296 li $t1, 115945
297 sb $t1, ($t0 + 144) # (space)
298 li $t1, 115946
299 sb $t1, ($t0 + 145) # (space)
300 li $t1, 115947
301 sb $t1, ($t0 + 146) # (space)
302 li $t1, 115948
303 sb $t1, ($t0 + 147) # (space)
304 li $t1, 115949
305 sb $t1, ($t0 + 148) # (space)
306 li $t1, 115950
307 sb $t1, ($t0 + 149) # (space)
308 li $t1, 115951
309 sb $t1, ($t0 + 150) # (space)
310 li $t1, 115952
311 sb $t1, ($t0 + 151) # (space)
312 li $t1, 115953
313 sb $t1, ($t0 + 152) # (space)
314 li $t1, 115954
315 sb $t1, ($t0 + 153) # (space)
316 li $t1, 115955
317 sb $t1, ($t0 + 154) # (space)
318 li $t1, 115956
319 sb $t1, ($t0 + 155) # (space)
320 li $t1, 115957
321 sb $t1, ($t0 + 156) # (space)
322 li $t1, 115958
323 sb $t1, ($t0 + 157) # (space)
324 li $t1, 115959
325 sb $t1, ($t0 + 158) # (space)
326 li $t1, 115960
327 sb $t1, ($t0 + 159) # (space)
328 li $t1, 115961
329 sb $t1, ($t0 + 160) # (space)
330 li $t1, 115962
331 sb $t1, ($t0 + 161) # (space)
332 li $t1, 115963
333 sb $t1, ($t0 + 162) # (space)
334 li $t1, 115964
335 sb $t1, ($t0 + 163) # (space)
336 li $t1, 115965
337 sb $t1, ($t0 + 164) # (space)
338 li $t1, 115966
339 sb $t1, ($t0 + 165) # (space)
340 li $t1, 115967
341 sb $t1, ($t0 + 166) # (space)
342 li $t1, 115968
343 sb $t1, ($t0 + 167) # (space)
344 li $t1, 115969
345 sb $t1, ($t0 + 168) # (space)
346 li $t1, 115970
347 sb $t1, ($t0 + 169) # (space)
348 li $t1, 115971
349 sb $t1, ($t0 + 170) # (space)
350 li $t1, 115972
351 sb $t1, ($t0 + 171) # (space)
352 li $t1, 115973
353 sb $t1, ($t0 + 172) # (space)
354 li $t1, 115974
355 sb $t1, ($t0 + 173) # (space)
356 li $t1, 115975
357 sb $t1, ($t0 + 174) # (space)
358 li $t1, 115976
359 sb $t1, ($t0 + 175) # (space)
360 li $t1, 115977
361 sb $t1, ($t0 + 176) # (space)
362 li $t1, 115978
363 sb $t1, ($t0 + 177) # (space)
364 li $t1, 115979
365 sb $t1, ($t0 + 178) # (space)
366 li $t1, 115980
367 sb $t1, ($t0 + 179) # (space)
368 li $t1, 115981
369 sb $t1, ($t0 + 180) # (space)
370 li $t1, 115982
371 sb $t1, ($t0 + 181) # (space)
372 li $t1, 115983
373 sb $t1, ($t0 + 182) # (space)
374 li $t1, 115984
375 sb $t1, ($t0 + 183) # (space)
376 li $t1, 115985
377 sb $t1, ($t0 + 184) # (space)
378 li $t1, 115986
379 sb $t1, ($t0 + 185) # (space)
380 li $t1, 115987
381 sb $t1, ($t0 + 186) # (space)
382 li $t1, 115988
383 sb $t1, ($t0 + 187) # (space)
384 li $t1, 115989
385 sb $t1, ($t0 + 188) # (space)
386 li $t1, 115990
387 sb $t1, ($t0 + 189) # (space)
388 li $t1, 115991
389 sb $t1, ($t0 + 190) # (space)
390 li $t1, 115992
391 sb $t1, ($t0 + 191) # (space)
392 li $t1, 115993
393 sb $t1, ($t0 + 192) # (space)
394 li $t1, 115994
395 sb $t1, ($t0 + 193) # (space)
396 li $t1, 115995
397 sb $t1, ($t0 + 194) # (space)
398 li $t1, 115996
399 sb $t1, ($t0 + 195) # (space)
400 li $t1, 115997
401 sb $t1, ($t0 + 196) # (space)
402 li $t1, 115998
403 sb $t1, ($t0 + 197) # (space)
404 li $t1, 115999
405 sb $t1, ($t0 + 198) # (space)
406 li $t1, 115900
407 sb $t1, ($t0 + 199) # (space)
408 li $t1, 115901
409 sb $t1, ($t0 + 200) # (space)
410 li $t1, 115902
411 sb $t1, ($t0 + 201) # (space)
412 li $t1, 115903
413 sb $t1, ($t0 + 202) # (space)
414 li $t1, 115904
415 sb $t1, ($t0 + 203) # (space)
416 li $t1, 115905
417 sb $t1, ($t0 + 204) # (space)
418 li $t1, 115906
419 sb $t1, ($t0 + 205) # (space)
420 li $t1, 115907
421 sb $t1, ($t0 + 206) # (space)
422 li $t1, 115908
423 sb $t1, ($t0 + 207) # (space)
424 li $t1, 115909
425 sb $t1, ($t0 + 208) # (space)
426 li $t1, 115910
427 sb $t1, ($t0 + 209) # (space)
428 li $t1, 115911
429 sb $t1, ($t0 + 210) # (space)
430 li $t1, 115912
431 sb $t1, ($t0 + 211) # (space)
432 li $t1, 115913
433 sb $t1, ($t0 + 212) # (space)
434 li $t1, 115914
435 sb $t1, ($t0 + 213) # (space)
436 li $t1, 115915
437 sb $t1, ($t0 + 214) # (space)
438 li $t1, 115916
439 sb $t1, ($t0 + 215) # (space)
440 li $t1, 115917
441 sb $t1, ($t0 + 216) # (space)
442 li $t1, 115918
443 sb $t1, ($t0 + 217) # (space)
444 li $t1, 115919
445 sb $t1, ($t0 + 218) # (space)
446 li $t1, 115920
447 sb $t1, ($t0 + 219) # (space)
448 li $t1, 115921
449 sb $t1, ($t0 + 220) # (space)
450 li $t1, 115922
451 sb $t1, ($t0 + 221) # (space)
452 li $t1, 115923
453 sb $t1, ($t0 + 222) # (space)
454 li $t1, 115924
455 sb $t1, ($t0 + 223) # (space)
456 li $t1, 115925
457 sb $t1, ($t0 + 224) # (space)
458 li $t1, 115926
459 sb $t1, ($t0 + 225) # (space)
460 li $t1, 115927
461 sb $t1, ($t0 + 226) # (space)
462 li $t1, 115928
463 sb $t1, ($t0 + 227) # (space)
464 li $t1, 115929
465 sb $t1, ($t0 + 228) # (space)
466 li $t1, 115930
467 sb $t1, ($t0 + 229) # (space)
468 li $t1, 115931
469 sb $t1, ($t0 + 230) # (space)
470 li $t1, 115932
471 sb $t1, ($t0 + 231) # (space)
472 li $t1, 115933
473 sb $t1, ($t0 + 232) # (space)
474 li $t1, 115934
475 sb $t1, ($t0 + 233) # (space)
476 li $t1, 115935
477 sb $t1, ($t0 + 234) # (space)
478 li $t1, 115936
479 sb $t1, ($t0 + 235) # (space)
480 li $t1, 115937
481 sb $t1, ($t0 + 236) # (space)
482 li $t1, 115938
483 sb $t1, ($t0 + 237) # (space)
484 li $t1, 115939
485 sb $t1, ($t0 + 238) # (space)
486 li $t1, 115940
487 sb $t1, ($t0 + 239) # (space)
488 li $t1, 115941
489 sb $t1, ($t0 + 240) # (space)
490 li $t1, 115942
491 sb $t1, ($t0 + 241) # (space)
492 li $t1, 115943
493 sb $t1, ($t0 + 242) # (space)
494 li $t1, 115944
495 sb $t1, ($t0 + 243) # (space)
496 li $t1, 115945
497 sb $t1, ($t0 + 244) # (space)
498 li $t1, 115946
499 sb $t1, ($t0 + 245) # (space)
500 li $t1, 115947
501 sb $t1, ($t0 + 246) # (space)
502 li $t1, 115948
503 sb $t1, ($t0 + 247) # (space)
504 li $t1, 115949
505 sb $t1, ($t0 + 248) # (space)
506 li $t1, 115950
507 sb $t1, ($t0 + 249) # (space)
508 li $t1, 115951
509 sb $t1, ($t0 + 250) # (space)
510 li $t1, 115952
511 sb $t1, ($t0 + 251) # (space)
512 li $t1, 115953
513 sb $t1, ($t0 + 252) # (space)
514 li $t1, 115954
515 sb $t1, ($t0 + 253) # (space)
516 li $t1, 115955
517 sb $t1, ($t0 + 254) # (space)
518 li $t1, 115956
519 sb $t1, ($t0 + 255) # (space)
520 li $t1, 115957
521 sb $t1, ($t0 + 256) # (space)
522 li $t1, 115958
523 sb $t1, ($t0 + 257) # (space)
524 li $t1, 115959
525 sb $t1, ($t0 + 258) # (space)
526 li $t1, 115960
527 sb $t1, ($t0 + 259) # (space)
528 li $t1, 115961
529 sb $t1, ($t0 + 260) # (space)
530 li $t1, 115962
531 sb $t1, ($t0 + 261) # (space)
532 li $t1, 115963
533 sb $t1, ($t0 + 262) # (space)
534 li $t1, 115964
535 sb $t1, ($t0 + 263) # (space)
536 li $t1, 115965
537 sb $t1, ($t0 + 264) # (space)
538 li $t1, 115966
539 sb $t1, ($t0 + 265) # (space)
540 li $t1, 115967
541 sb $t1, ($t0 + 266) # (space)
542 li $t1, 115968
543 sb $t1, ($t0 + 267) # (space)
544 li $t1, 115969
545 sb $t1, ($t0 + 268) # (space)
546 li $t1, 115970
547 sb $t1, ($t0 + 269) # (space)
548 li $t1, 115971
549 sb $t1, ($t0 + 270) # (space)
550 li $t1, 115972
551 sb $t1, ($t0 + 271) # (space)
552 li $t1, 115973
553 sb $t1, ($t0 + 272) # (space)
554 li $t1, 115974
555 sb $t1, ($t0 + 273) # (space)
556 li $t1, 115975
557 sb $t1, ($t0 + 274) # (space)
558 li $t1, 115976
559 sb $t1, ($t0 + 275) # (space)
560 li $t1, 115977
561 sb $t1, ($t0 + 276) # (space)
562 li $t1, 115978
563 sb $t1, ($t0 + 277) # (space)
564 li $t1, 115979
565 sb $t1, ($t0 + 278) # (space)
566 li $t1, 115980
567 sb $t1, ($t0 + 279) # (space)
568 li $t1, 115981
569 sb $t1, ($t0 + 280) # (space)
570 li $t1, 115982
571 sb $t1, ($t0 + 281) # (space)
572 li $t1, 115983
573 sb $t1, ($t0 + 282) # (space)
574 li $t1, 115984
575 sb $t1, ($t0 + 283) # (space)
576 li $t1, 115985
577 sb $t1, ($t0 + 284) # (space)
578 li $t1, 115986
579 sb $t1, ($t0 + 285) # (space)
580 li $t1, 115987
581 sb $t1, ($t0 + 286) # (space)
582 li $t1, 115988
583 sb $t1, ($t0 + 287) # (space)
584 li $t1, 115989
585 sb $t1, ($t0 + 288) # (space)
586 li $t1, 115990
587 sb $t1, ($t0 + 289) # (space)
588 li $t1, 115991
589 sb $t1, ($t0 + 290) # (space)
590 li $t1, 115992
591 sb $t1, ($t0 + 291) # (space)
592 li $t1, 115993
593 sb $t1, ($t0 + 292) # (space)
594 li $t1, 115994
595 sb $t1, ($t0 + 293) # (space)
596 li $t1, 115995
597 sb $t1, ($t0 + 294) # (space)
598 li $t1, 115996
599 sb $t1, ($t0 + 295) # (space)
600 li $t1, 115997
601 sb $t1, ($t0 + 296) # (space)
602 li $t1, 115998
603 sb $t1, ($t0 + 297) # (space)
604 li $t1, 115999
605 sb $t1, ($t0 + 298) # (space)
606 li $t1, 115900
607 sb $t1, ($t0 + 299) # (space)
608 li $t1, 115901
609 sb $t1, ($t0 + 300) # (space)
```

Registers shown in the debugger:

S	T	V	Stack	Log
x0	10			
x1	9			
x2	8			
x3	7			
x4	6			
x5	5			
x6	4			
x7	3			
x8	2			
x9	1			
x10	0			

## Memory

Memory is organized as a sequence of bytes indexed by address.

Addressing is done via registers, memory locations are 32-bit words.

Register \$t0 contains the address of the string, \$t1 contains the value of the first byte.

After each instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

After the final instruction, the value of \$t1 is displayed in the log.

## MIPS Commands

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
  - **R Instructions:** Commands that use data in the registers:

# MIPS Commands

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
  - **R Instructions:** Commands that use data in the registers:  
add \$s1, \$s2, \$s3

## MIPS Commands

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
  - **R Instructions:** Commands that use data in the registers:  
add \$s1, \$s2, \$s3      (Basic form: OP rd, rs, rt)
  - **I Instructions:** instructions that also use intermediate values.

## MIPS Commands

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
  - **R Instructions:** Commands that use data in the registers:  
add \$s1, \$s2, \$s3      (Basic form: OP rd, rs, rt)
  - **I Instructions:** instructions that also use intermediate values.  
addi \$s1, \$s2, 100

## MIPS Commands

- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
  - **R Instructions:** Commands that use data in the registers:  
add \$s1, \$s2, \$s3      (Basic form: OP rd, rs, rt)
  - **I Instructions:** instructions that also use intermediate values.  
addi \$s1, \$s2, 100      (Basic form: OP rd, rs, imm)
  - **J Instructions:** instructions that jump to another memory location.

# MIPS Commands

The screenshot shows a MIPS assembly debugger interface. The assembly code window contains the following instructions:

```
1 # Shows 'Hello, world!' at the top of the stack
2 .text
3 .globl _start
4 .type _start, @function
5 _start:
6    li $t0, 115902
7    li $t1, 115901
8    li $t2, 115900
9    li $t3, 115901
10   li $t4, 115900
11   li $t5, 115901
12   li $t6, 115900
13   li $t7, 115901
14   li $t8, 115900
15   li $t9, 115901
16   li $t10, 115900
17   li $t11, 115901
18   li $t12, 115900
19   li $t13, 115901
20   li $t14, 115900
21   li $t15, 115901
22   li $t16, 115900
23   li $t17, 115901
24   li $t18, 115900
25   li $t19, 115901
26   li $t20, 115900
27   li $t21, 115901
28   li $t22, 115900
29   li $t23, 115901
30   li $t24, 115900
31   li $t25, 115901
32   li $t26, 115900
33   li $t27, 115901
34   li $t28, 115900
35   li $t29, 115901
36   li $t30, 115900
37   li $t31, 115901
38   li $t32, 115900
39   li $t33, 115901
40   li $t34, 115900
41   li $t35, 115901
42   li $t36, 115900
43   li $t37, 115901
44   li $t38, 115900
45   li $t39, 115901
46   li $t40, 115900
47   li $t41, 115901
48   li $t42, 115900
49   li $t43, 115901
50   li $t44, 115900
51   li $t45, 115901
52   li $t46, 115900
53   li $t47, 115901
54   li $t48, 115900
55   li $t49, 115901
56   li $t50, 115900
57   li $t51, 115901
58   li $t52, 115900
59   li $t53, 115901
60   li $t54, 115900
61   li $t55, 115901
62   li $t56, 115900
63   li $t57, 115901
64   li $t58, 115900
65   li $t59, 115901
66   li $t60, 115900
67   li $t61, 115901
68   li $t62, 115900
69   li $t63, 115901
70   li $t64, 115900
71   li $t65, 115901
72   li $t66, 115900
73   li $t67, 115901
74   li $t68, 115900
75   li $t69, 115901
76   li $t70, 115900
77   li $t71, 115901
78   li $t72, 115900
79   li $t73, 115901
80   li $t74, 115900
81   li $t75, 115901
82   li $t76, 115900
83   li $t77, 115901
84   li $t78, 115900
85   li $t79, 115901
86   li $t80, 115900
87   li $t81, 115901
88   li $t82, 115900
89   li $t83, 115901
90   li $t84, 115900
91   li $t85, 115901
92   li $t86, 115900
93   li $t87, 115901
94   li $t88, 115900
95   li $t89, 115901
96   li $t90, 115900
97   li $t91, 115901
98   li $t92, 115900
99   li $t93, 115901
100  li $t94, 115900
101  li $t95, 115901
102  li $t96, 115900
103  li $t97, 115901
104  li $t98, 115900
105  li $t99, 115901
106  li $t100, 115900
107  li $t101, 115901
108  li $t102, 115900
109  li $t103, 115901
110  li $t104, 115900
111  li $t105, 115901
112  li $t106, 115900
113  li $t107, 115901
114  li $t108, 115900
115  li $t109, 115901
116  li $t110, 115900
117  li $t111, 115901
118  li $t112, 115900
119  li $t113, 115901
120  li $t114, 115900
121  li $t115, 115901
122  li $t116, 115900
123  li $t117, 115901
124  li $t118, 115900
125  li $t119, 115901
126  li $t120, 115900
127  li $t121, 115901
128  li $t122, 115900
129  li $t123, 115901
130  li $t124, 115900
131  li $t125, 115901
132  li $t126, 115900
133  li $t127, 115901
134  li $t128, 115900
135  li $t129, 115901
136  li $t130, 115900
137  li $t131, 115901
138  li $t132, 115900
139  li $t133, 115901
140  li $t134, 115900
141  li $t135, 115901
142  li $t136, 115900
143  li $t137, 115901
144  li $t138, 115900
145  li $t139, 115901
146  li $t140, 115900
147  li $t141, 115901
148  li $t142, 115900
149  li $t143, 115901
150  li $t144, 115900
151  li $t145, 115901
152  li $t146, 115900
153  li $t147, 115901
154  li $t148, 115900
155  li $t149, 115901
156  li $t150, 115900
157  li $t151, 115901
158  li $t152, 115900
159  li $t153, 115901
160  li $t154, 115900
161  li $t155, 115901
162  li $t156, 115900
163  li $t157, 115901
164  li $t158, 115900
165  li $t159, 115901
166  li $t160, 115900
167  li $t161, 115901
168  li $t162, 115900
169  li $t163, 115901
170  li $t164, 115900
171  li $t165, 115901
172  li $t166, 115900
173  li $t167, 115901
174  li $t168, 115900
175  li $t169, 115901
176  li $t170, 115900
177  li $t171, 115901
178  li $t172, 115900
179  li $t173, 115901
180  li $t174, 115900
181  li $t175, 115901
182  li $t176, 115900
183  li $t177, 115901
184  li $t178, 115900
185  li $t179, 115901
186  li $t180, 115900
187  li $t181, 115901
188  li $t182, 115900
189  li $t183, 115901
190  li $t184, 115900
191  li $t185, 115901
192  li $t186, 115900
193  li $t187, 115901
194  li $t188, 115900
195  li $t189, 115901
196  li $t190, 115900
197  li $t191, 115901
198  li $t192, 115900
199  li $t193, 115901
200  li $t194, 115900
201  li $t195, 115901
202  li $t196, 115900
203  li $t197, 115901
204  li $t198, 115900
205  li $t199, 115901
206  li $t200, 115900
207  li $t201, 115901
208  li $t202, 115900
209  li $t203, 115901
210  li $t204, 115900
211  li $t205, 115901
212  li $t206, 115900
213  li $t207, 115901
214  li $t208, 115900
215  li $t209, 115901
216  li $t210, 115900
217  li $t211, 115901
218  li $t212, 115900
219  li $t213, 115901
220  li $t214, 115900
221  li $t215, 115901
222  li $t216, 115900
223  li $t217, 115901
224  li $t218, 115900
225  li $t219, 115901
226  li $t220, 115900
227  li $t221, 115901
228  li $t222, 115900
229  li $t223, 115901
230  li $t224, 115900
231  li $t225, 115901
232  li $t226, 115900
233  li $t227, 115901
234  li $t228, 115900
235  li $t229, 115901
236  li $t230, 115900
237  li $t231, 115901
238  li $t232, 115900
239  li $t233, 115901
240  li $t234, 115900
241  li $t235, 115901
242  li $t236, 115900
243  li $t237, 115901
244  li $t238, 115900
245  li $t239, 115901
246  li $t240, 115900
247  li $t241, 115901
248  li $t242, 115900
249  li $t243, 115901
250  li $t244, 115900
251  li $t245, 115901
252  li $t246, 115900
253  li $t247, 115901
254  li $t248, 115900
255  li $t249, 115901
256  li $t250, 115900
257  li $t251, 115901
258  li $t252, 115900
259  li $t253, 115901
260  li $t254, 115900
261  li $t255, 115901
262  li $t256, 115900
263  li $t257, 115901
264  li $t258, 115900
265  li $t259, 115901
266  li $t260, 115900
267  li $t261, 115901
268  li $t262, 115900
269  li $t263, 115901
270  li $t264, 115900
271  li $t265, 115901
272  li $t266, 115900
273  li $t267, 115901
274  li $t268, 115900
275  li $t269, 115901
276  li $t270, 115900
277  li $t271, 115901
278  li $t272, 115900
279  li $t273, 115901
280  li $t274, 115900
281  li $t275, 115901
282  li $t276, 115900
283  li $t277, 115901
284  li $t278, 115900
285  li $t279, 115901
286  li $t280, 115900
287  li $t281, 115901
288  li $t282, 115900
289  li $t283, 115901
290  li $t284, 115900
291  li $t285, 115901
292  li $t286, 115900
293  li $t287, 115901
294  li $t288, 115900
295  li $t289, 115901
296  li $t290, 115900
297  li $t291, 115901
298  li $t292, 115900
299  li $t293, 115901
300  li $t294, 115900
301  li $t295, 115901
302  li $t296, 115900
303  li $t297, 115901
304  li $t298, 115900
305  li $t299, 115901
306  li $t300, 115900
307  li $t301, 115901
308  li $t302, 115900
309  li $t303, 115901
310  li $t304, 115900
311  li $t305, 115901
312  li $t306, 115900
313  li $t307, 115901
314  li $t308, 115900
315  li $t309, 115901
316  li $t310, 115900
317  li $t311, 115901
318  li $t312, 115900
319  li $t313, 115901
320  li $t314, 115900
321  li $t315, 115901
322  li $t316, 115900
323  li $t317, 115901
324  li $t318, 115900
325  li $t319, 115901
326  li $t320, 115900
327  li $t321, 115901
328  li $t322, 115900
329  li $t323, 115901
330  li $t324, 115900
331  li $t325, 115901
332  li $t326, 115900
333  li $t327, 115901
334  li $t328, 115900
335  li $t329, 115901
336  li $t330, 115900
337  li $t331, 115901
338  li $t332, 115900
339  li $t333, 115901
340  li $t334, 115900
341  li $t335, 115901
342  li $t336, 115900
343  li $t337, 115901
344  li $t338, 115900
345  li $t339, 115901
346  li $t340, 115900
347  li $t341, 115901
348  li $t342, 115900
349  li $t343, 115901
350  li $t344, 115900
351  li $t345, 115901
352  li $t346, 115900
353  li $t347, 115901
354  li $t348, 115900
355  li $t349, 115901
356  li $t350, 115900
357  li $t351, 115901
358  li $t352, 115900
359  li $t353, 115901
360  li $t354, 115900
361  li $t355, 115901
362  li $t356, 115900
363  li $t357, 115901
364  li $t358, 115900
365  li $t359, 115901
366  li $t360, 115900
367  li $t361, 115901
368  li $t362, 115900
369  li $t363, 115901
370  li $t364, 115900
371  li $t365, 115901
372  li $t366, 115900
373  li $t367, 115901
374  li $t368, 115900
375  li $t369, 115901
376  li $t370, 115900
377  li $t371, 115901
378  li $t372, 115900
379  li $t373, 115901
380  li $t374, 115900
381  li $t375, 115901
382  li $t376, 115900
383  li $t377, 115901
384  li $t378, 115900
385  li $t379, 115901
386  li $t380, 115900
387  li $t381, 115901
388  li $t382, 115900
389  li $t383, 115901
390  li $t384, 115900
391  li $t385, 115901
392  li $t386, 115900
393  li $t387, 115901
394  li $t388, 115900
395  li $t389, 115901
396  li $t390, 115900
397  li $t391, 115901
398  li $t392, 115900
399  li $t393, 115901
400  li $t394, 115900
401  li $t395, 115901
402  li $t396, 115900
403  li $t397, 115901
404  li $t398, 115900
405  li $t399, 115901
406  li $t400, 115900
407  li $t401, 115901
408  li $t402, 115900
409  li $t403, 115901
410  li $t404, 115900
411  li $t405, 115901
412  li $t406, 115900
413  li $t407, 115901
414  li $t408, 115900
415  li $t409, 115901
416  li $t410, 115900
417  li $t411, 115901
418  li $t412, 115900
419  li $t413, 115901
420  li $t414, 115900
421  li $t415, 115901
422  li $t416, 115900
423  li $t417, 115901
424  li $t418, 115900
425  li $t419, 115901
426  li $t420, 115900
427  li $t421, 115901
428  li $t422, 115900
429  li $t423, 115901
430  li $t424, 115900
431  li $t425, 115901
432  li $t426, 115900
433  li $t427, 115901
434  li $t428, 115900
435  li $t429, 115901
436  li $t430, 115900
437  li $t431, 115901
438  li $t432, 115900
439  li $t433, 115901
440  li $t434, 115900
441  li $t435, 115901
442  li $t436, 115900
443  li $t437, 115901
444  li $t438, 115900
445  li $t439, 115901
446  li $t440, 115900
447  li $t441, 115901
448  li $t442, 115900
449  li $t443, 115901
450  li $t444, 115900
451  li $t445, 115901
452  li $t446, 115900
453  li $t447, 115901
454  li $t448, 115900
455  li $t449, 115901
456  li $t450, 115900
457  li $t451, 115901
458  li $t452, 115900
459  li $t453, 115901
460  li $t454, 115900
461  li $t455, 115901
462  li $t456, 115900
463  li $t457, 115901
464  li $t458, 115900
465  li $t459, 115901
466  li $t460, 115900
467  li $t461, 115901
468  li $t462, 115900
469  li $t463, 115901
470  li $t464, 115900
471  li $t465, 115901
472  li $t466, 115900
473  li $t467, 115901
474  li $t468, 115900
475  li $t469, 115901
476  li $t470, 115900
477  li $t471, 115901
478  li $t472, 115900
479  li $t473, 115901
480  li $t474, 115900
481  li $t475, 115901
482  li $t476, 115900
483  li $t477, 115901
484  li $t478, 115900
485  li $t479, 115901
486  li $t480, 115900
487  li $t481, 115901
488  li $t482, 115900
489  li $t483, 115901
490  li $t484, 115900
491  li $t485, 115901
492  li $t486, 115900
493  li $t487, 115901
494  li $t488, 115900
495  li $t489, 115901
496  li $t490, 115900
497  li $t491, 115901
498  li $t492, 115900
499  li $t493, 115901
500  li $t494, 115900
501  li $t495, 115901
502  li $t496, 115900
503  li $t497, 115901
504  li $t498, 115900
505  li $t499, 115901
506  li $t500, 115900
507  li $t501, 115901
508  li $t502, 115900
509  li $t503, 115901
510  li $t504, 115900
511  li $t505, 115901
512  li $t506, 115900
513  li $t507, 115901
514  li $t508, 115900
515  li $t509, 115901
516  li $t510, 115900
517  li $t511, 115901
518  li $t512, 115900
519  li $t513, 115901
520  li $t514, 115900
521  li $t515, 115901
522  li $t516, 115900
523  li $t517, 115901
524  li $t518, 115900
525  li $t519, 115901
526  li $t520, 115900
527  li $t521, 115901
528  li $t522, 115900
529  li $t523, 115901
530  li $t524, 115900
531  li $t525, 115901
532  li $t526, 115900
533  li $t527, 115901
534  li $t528, 115900
535  li $t529, 115901
536  li $t530, 115900
537  li $t531, 115901
538  li $t532, 115900
539  li $t533, 115901
540  li $t534, 115900
541  li $t535, 115901
542  li $t536, 115900
543  li $t537, 115901
544  li $t538, 115900
545  li $t539, 115901
546  li $t540, 115900
547  li $t541, 115901
548  li $t542, 115900
549  li $t543, 115901
550  li $t544, 115900
551  li $t545, 115901
552  li $t546, 115900
553  li $t547, 115901
554  li $t548, 115900
555  li $t549, 115901
556  li $t550, 115900
557  li $t551, 115901
558  li $t552, 115900
559  li $t553, 115901
560  li $t554, 115900
561  li $t555, 115901
562  li $t556, 115900
563  li $t557, 115901
564  li $t558, 115900
565  li $t559, 115901
566  li $t560, 115900
567  li $t561, 115901
568  li $t562, 115900
569  li $t563, 115901
570  li $t564, 115900
571  li $t565, 115901
572  li $t566, 115900
573  li $t567, 115901
574  li $t568, 115900
575  li $t569, 115901
576  li $t570, 115900
577  li $t571, 115901
578  li $t572, 115900
579  li $t573, 115901
580  li $t574, 115900
581  li $t575, 115901
582  li $t576, 115900
583  li $t577, 115901
584  li $t578, 115900
585  li $t579, 115901
586  li $t580, 115900
587  li $t581, 115901
588  li $t582, 115900
589  li $t583, 115901
590  li $t584, 115900
591  li $t585, 115901
592  li $t586, 115900
593  li $t587, 115901
594  li $t588, 115900
595  li $t589, 115901
596  li $t590, 115900
597  li $t591, 115901
598  li $t592, 115900
599  li $t593, 115901
600  li $t594, 115900
601  li $t595, 115901
602  li $t596, 115900
603  li $t597, 115901
604  li $t598, 115900
605  li $t599, 115901
606  li $t600, 115900
607  li $t601, 115901
608  li $t602, 115900
609  li $t603, 115901
610  li $t604, 115900
611  li $t605, 115901
612  li $t606, 115900
613  li $t607, 115901
614  li $t608, 115900
615  li $t609, 115901
616  li $t610, 115900
617  li $t611, 115901
618  li $t612, 115900
619  li $t613, 115901
620  li $t614, 115900
621  li $t615, 115901
622  li $t616, 115900
623  li $t617, 115901
624  li $t618, 115900
625  li $t619, 115901
626  li $t620, 115900
627  li $t621, 115901
628  li $t622, 115900
629  li $t623, 115901
630  li $t624, 115900
631  li $t625, 115901
632  li $t626, 115900
633  li $t627, 115901
634  li $t628, 115900
635  li $t629, 115901
636  li $t630, 115900
637  li $t631, 115901
638  li $t632, 115900
639  li $t633, 115901
640  li $t634, 115900
641  li $t635, 115901
642  li $t636, 115900
643  li $t637, 115901
644  li $t638, 115900
645  li $t639, 115901
646  li $t640, 115900
647  li $t641, 115901
648  li $t642, 115900
649  li $t643, 115901
650  li $t644, 115900
651  li $t645, 115901
652  li $t646, 115900
653  li $t647, 115901
654  li $t648, 115900
655  li $t649, 115901
656  li $t650, 115900
657  li $t651, 115901
658  li $t652, 115900
659  li $t653, 115901
660  li $t654, 115900
661  li $
```

# MIPS Commands

The screenshot shows a MIPS assembly editor interface. At the top, there's a menu bar with 'File', 'Edit', 'Get', 'Show/Hide Demo', 'Addition', 'Subtraction', 'Multiplication', 'Division', 'Hello World', 'Code Gen', 'Save String', 'Interactive', 'Decimal Decimal', 'Decimal Binary', and 'Debug'. Below the menu is a toolbar with 'Step', 'Run', 'Break', 'Create auto stepping', and 'Stop' buttons. A status bar at the bottom right shows 'User Guide | Unit Tests | Docs'. The main area has tabs for 'S' (selected), 'T', 'A', 'V', 'Stack', and 'Log'. The assembly code in the 'S' tab is:

```
1 # Shows 'Hello world' at the top of the stack
2 .text
3 .globl _start
4 .data
5 _start: .asciz "Hello world\n"
6 .text
7 _start: li $t0, 111901
8 sb $t0, 111901
9 li $t1, 111902
10 sb $t1, 111902
11 li $t2, 111903
12 sb $t2, 111903
13 li $t3, 111904
14 sb $t3, 111904
15 li $t4, 111905
16 sb $t4, 111905
17 li $t5, 111906
18 sb $t5, 111906
19 li $t6, 111907
20 sb $t6, 111907
21 li $t7, 111908
22 sb $t7, 111908
23 li $t8, 111909
24 sb $t8, 111909
25 li $t9, 111910
26 sb $t9, 111910
27 li $t10, 111911
28 sb $t10, 111911
29 li $t11, 111912
30 sb $t11, 111912
31 li $t12, 111913
32 sb $t12, 111913
33 li $t13, 111914
34 sb $t13, 111914
35 li $t14, 111915
36 sb $t14, 111915
37 li $t15, 111916
38 sb $t15, 111916
39 li $t16, 111917
40 sb $t16, 111917
41 li $t17, 111918
42 sb $t17, 111918
43 li $t18, 111919
44 sb $t18, 111919
45 li $t19, 111920
46 sb $t19, 111920
47 li $t20, 111921
48 sb $t20, 111921
49 li $t21, 111922
50 sb $t21, 111922
51 li $t22, 111923
52 sb $t22, 111923
53 li $t23, 111924
54 sb $t23, 111924
55 li $t24, 111925
56 sb $t24, 111925
57 li $t25, 111926
58 sb $t25, 111926
59 li $t26, 111927
60 sb $t26, 111927
61 li $t27, 111928
62 sb $t27, 111928
63 li $t28, 111929
64 sb $t28, 111929
65 li $t29, 111930
66 sb $t29, 111930
67 li $t30, 111931
68 sb $t30, 111931
69 li $t31, 111932
70 sb $t31, 111932
71 li $t32, 111933
72 sb $t32, 111933
73 li $t33, 111934
74 sb $t33, 111934
75 li $t34, 111935
76 sb $t34, 111935
77 li $t35, 111936
78 sb $t35, 111936
79 li $t36, 111937
80 sb $t36, 111937
81 li $t37, 111938
82 sb $t37, 111938
83 li $t38, 111939
84 sb $t38, 111939
85 li $t39, 111940
86 sb $t39, 111940
87 li $t40, 111941
88 sb $t40, 111941
89 li $t41, 111942
90 sb $t41, 111942
91 li $t42, 111943
92 sb $t42, 111943
93 li $t43, 111944
94 sb $t43, 111944
95 li $t44, 111945
96 sb $t44, 111945
97 li $t45, 111946
98 sb $t45, 111946
99 li $t46, 111947
100 sb $t46, 111947
101 li $t47, 111948
102 sb $t47, 111948
103 li $t48, 111949
104 sb $t48, 111949
105 li $t49, 111950
106 sb $t49, 111950
107 li $t50, 111951
108 sb $t50, 111951
109 li $t51, 111952
110 sb $t51, 111952
111 li $t52, 111953
112 sb $t52, 111953
113 li $t53, 111954
114 sb $t53, 111954
115 li $t54, 111955
116 sb $t54, 111955
117 li $t55, 111956
118 sb $t55, 111956
119 li $t56, 111957
120 sb $t56, 111957
121 li $t57, 111958
122 sb $t57, 111958
123 li $t58, 111959
124 sb $t58, 111959
125 li $t59, 111960
126 sb $t59, 111960
127 li $t60, 111961
128 sb $t60, 111961
129 li $t61, 111962
130 sb $t61, 111962
131 li $t62, 111963
132 sb $t62, 111963
133 li $t63, 111964
134 sb $t63, 111964
135 li $t64, 111965
136 sb $t64, 111965
137 li $t65, 111966
138 sb $t65, 111966
139 li $t66, 111967
140 sb $t66, 111967
141 li $t67, 111968
142 sb $t67, 111968
143 li $t68, 111969
144 sb $t68, 111969
145 li $t69, 111970
146 sb $t69, 111970
147 li $t70, 111971
148 sb $t70, 111971
149 li $t71, 111972
150 sb $t71, 111972
151 li $t72, 111973
152 sb $t72, 111973
153 li $t73, 111974
154 sb $t73, 111974
155 li $t74, 111975
156 sb $t74, 111975
157 li $t75, 111976
158 sb $t75, 111976
159 li $t76, 111977
160 sb $t76, 111977
161 li $t77, 111978
162 sb $t77, 111978
163 li $t78, 111979
164 sb $t78, 111979
165 li $t79, 111980
166 sb $t79, 111980
167 li $t80, 111981
168 sb $t80, 111981
169 li $t81, 111982
170 sb $t81, 111982
171 li $t82, 111983
172 sb $t82, 111983
173 li $t83, 111984
174 sb $t83, 111984
175 li $t84, 111985
176 sb $t84, 111985
177 li $t85, 111986
178 sb $t85, 111986
179 li $t86, 111987
180 sb $t86, 111987
181 li $t87, 111988
182 sb $t87, 111988
183 li $t88, 111989
184 sb $t88, 111989
185 li $t89, 111990
186 sb $t89, 111990
187 li $t90, 111991
188 sb $t90, 111991
189 li $t91, 111992
190 sb $t91, 111992
191 li $t92, 111993
192 sb $t92, 111993
193 li $t93, 111994
194 sb $t93, 111994
195 li $t94, 111995
196 sb $t94, 111995
197 li $t95, 111996
198 sb $t95, 111996
199 li $t96, 111997
200 sb $t96, 111997
201 li $t97, 111998
202 sb $t97, 111998
203 li $t98, 111999
204 sb $t98, 111999
205 li $t99, 111900
206 sb $t99, 111900
207 li $t100, 111901
208 sb $t100, 111901
209 li $t101, 111902
210 sb $t101, 111902
211 li $t102, 111903
212 sb $t102, 111903
213 li $t103, 111904
214 sb $t103, 111904
215 li $t104, 111905
216 sb $t104, 111905
217 li $t105, 111906
218 sb $t105, 111906
219 li $t106, 111907
220 sb $t106, 111907
221 li $t107, 111908
222 sb $t107, 111908
223 li $t108, 111909
224 sb $t108, 111909
225 li $t109, 111910
226 sb $t109, 111910
227 li $t110, 111911
228 sb $t110, 111911
229 li $t111, 111912
230 sb $t111, 111912
231 li $t112, 111913
232 sb $t112, 111913
233 li $t113, 111914
234 sb $t113, 111914
235 li $t114, 111915
236 sb $t114, 111915
237 li $t115, 111916
238 sb $t115, 111916
239 li $t116, 111917
240 sb $t116, 111917
241 li $t117, 111918
242 sb $t117, 111918
243 li $t118, 111919
244 sb $t118, 111919
245 li $t119, 111920
246 sb $t119, 111920
247 li $t120, 111921
248 sb $t120, 111921
249 li $t121, 111922
250 sb $t121, 111922
251 li $t122, 111923
252 sb $t122, 111923
253 li $t123, 111924
254 sb $t123, 111924
255 li $t124, 111925
256 sb $t124, 111925
257 li $t125, 111926
258 sb $t125, 111926
259 li $t126, 111927
260 sb $t126, 111927
261 li $t127, 111928
262 sb $t127, 111928
263 li $t128, 111929
264 sb $t128, 111929
265 li $t129, 111930
266 sb $t129, 111930
267 li $t130, 111931
268 sb $t130, 111931
269 li $t131, 111932
270 sb $t131, 111932
271 li $t132, 111933
272 sb $t132, 111933
273 li $t133, 111934
274 sb $t133, 111934
275 li $t134, 111935
276 sb $t134, 111935
277 li $t135, 111936
278 sb $t135, 111936
279 li $t136, 111937
280 sb $t136, 111937
281 li $t137, 111938
282 sb $t137, 111938
283 li $t138, 111939
284 sb $t138, 111939
285 li $t139, 111940
286 sb $t139, 111940
287 li $t140, 111941
288 sb $t140, 111941
289 li $t141, 111942
290 sb $t141, 111942
291 li $t142, 111943
292 sb $t142, 111943
293 li $t143, 111944
294 sb $t143, 111944
295 li $t144, 111945
296 sb $t144, 111945
297 li $t145, 111946
298 sb $t145, 111946
299 li $t146, 111947
300 sb $t146, 111947
301 li $t147, 111948
302 sb $t147, 111948
303 li $t148, 111949
304 sb $t148, 111949
305 li $t149, 111950
306 sb $t149, 111950
307 li $t150, 111951
308 sb $t150, 111951
309 li $t151, 111952
310 sb $t151, 111952
311 li $t152, 111953
312 sb $t152, 111953
313 li $t153, 111954
314 sb $t153, 111954
315 li $t154, 111955
316 sb $t154, 111955
317 li $t155, 111956
318 sb $t155, 111956
319 li $t156, 111957
320 sb $t156, 111957
321 li $t157, 111958
322 sb $t157, 111958
323 li $t158, 111959
324 sb $t158, 111959
325 li $t159, 111960
326 sb $t159, 111960
327 li $t160, 111961
328 sb $t160, 111961
329 li $t161, 111962
330 sb $t161, 111962
331 li $t162, 111963
332 sb $t162, 111963
333 li $t163, 111964
334 sb $t163, 111964
335 li $t164, 111965
336 sb $t164, 111965
337 li $t165, 111966
338 sb $t165, 111966
339 li $t166, 111967
340 sb $t166, 111967
341 li $t167, 111968
342 sb $t167, 111968
343 li $t168, 111969
344 sb $t168, 111969
345 li $t169, 111970
346 sb $t169, 111970
347 li $t170, 111971
348 sb $t170, 111971
349 li $t171, 111972
350 sb $t171, 111972
351 li $t172, 111973
352 sb $t172, 111973
353 li $t173, 111974
354 sb $t173, 111974
355 li $t174, 111975
356 sb $t174, 111975
357 li $t175, 111976
358 sb $t175, 111976
359 li $t176, 111977
360 sb $t176, 111977
361 li $t177, 111978
362 sb $t177, 111978
363 li $t178, 111979
364 sb $t178, 111979
365 li $t179, 111980
366 sb $t179, 111980
367 li $t180, 111981
368 sb $t180, 111981
369 li $t181, 111982
370 sb $t181, 111982
371 li $t182, 111983
372 sb $t182, 111983
373 li $t183, 111984
374 sb $t183, 111984
375 li $t184, 111985
376 sb $t184, 111985
377 li $t185, 111986
378 sb $t185, 111986
379 li $t186, 111987
380 sb $t186, 111987
381 li $t187, 111988
382 sb $t187, 111988
383 li $t188, 111989
384 sb $t188, 111989
385 li $t189, 111990
386 sb $t189, 111990
387 li $t190, 111991
388 sb $t190, 111991
389 li $t191, 111992
390 sb $t191, 111992
391 li $t192, 111993
392 sb $t192, 111993
393 li $t193, 111994
394 sb $t193, 111994
395 li $t194, 111995
396 sb $t194, 111995
397 li $t195, 111996
398 sb $t195, 111996
399 li $t196, 111997
400 sb $t196, 111997
401 li $t197, 111998
402 sb $t197, 111998
403 li $t198, 111999
404 sb $t198, 111999
405 li $t199, 111900
406 sb $t199, 111900
407 li $t200, 111901
408 sb $t200, 111901
409 li $t201, 111902
410 sb $t201, 111902
411 li $t202, 111903
412 sb $t202, 111903
413 li $t203, 111904
414 sb $t203, 111904
415 li $t204, 111905
416 sb $t204, 111905
417 li $t205, 111906
418 sb $t205, 111906
419 li $t206, 111907
420 sb $t206, 111907
421 li $t207, 111908
422 sb $t207, 111908
423 li $t208, 111909
424 sb $t208, 111909
425 li $t209, 111910
426 sb $t209, 111910
427 li $t210, 111911
428 sb $t210, 111911
429 li $t211, 111912
430 sb $t211, 111912
431 li $t212, 111913
432 sb $t212, 111913
433 li $t213, 111914
434 sb $t213, 111914
435 li $t214, 111915
436 sb $t214, 111915
437 li $t215, 111916
438 sb $t215, 111916
439 li $t216, 111917
440 sb $t216, 111917
441 li $t217, 111918
442 sb $t217, 111918
443 li $t218, 111919
444 sb $t218, 111919
445 li $t219, 111920
446 sb $t219, 111920
447 li $t220, 111921
448 sb $t220, 111921
449 li $t221, 111922
450 sb $t221, 111922
451 li $t222, 111923
452 sb $t222, 111923
453 li $t223, 111924
454 sb $t223, 111924
455 li $t224, 111925
456 sb $t224, 111925
457 li $t225, 111926
458 sb $t225, 111926
459 li $t226, 111927
460 sb $t226, 111927
461 li $t227, 111928
462 sb $t227, 111928
463 li $t228, 111929
464 sb $t228, 111929
465 li $t229, 111930
466 sb $t229, 111930
467 li $t230, 111931
468 sb $t230, 111931
469 li $t231, 111932
470 sb $t231, 111932
471 li $t232, 111933
472 sb $t232, 111933
473 li $t233, 111934
474 sb $t233, 111934
475 li $t234, 111935
476 sb $t234, 111935
477 li $t235, 111936
478 sb $t235, 111936
479 li $t236, 111937
480 sb $t236, 111937
481 li $t237, 111938
482 sb $t237, 111938
483 li $t238, 111939
484 sb $t238, 111939
485 li $t239, 111940
486 sb $t239, 111940
487 li $t240, 111941
488 sb $t240, 111941
489 li $t241, 111942
490 sb $t241, 111942
491 li $t242, 111943
492 sb $t242, 111943
493 li $t243, 111944
494 sb $t243, 111944
495 li $t244, 111945
496 sb $t244, 111945
497 li $t245, 111946
498 sb $t245, 111946
499 li $t246, 111947
500 sb $t246, 111947
501 li $t247, 111948
502 sb $t247, 111948
503 li $t248, 111949
504 sb $t248, 111949
505 li $t249, 111950
506 sb $t249, 111950
507 li $t250, 111951
508 sb $t250, 111951
509 li $t251, 111952
510 sb $t251, 111952
511 li $t252, 111953
512 sb $t252, 111953
513 li $t253, 111954
514 sb $t253, 111954
515 li $t254, 111955
516 sb $t254, 111955
517 li $t255, 111956
518 sb $t255, 111956
519 li $t256, 111957
520 sb $t256, 111957
521 li $t257, 111958
522 sb $t257, 111958
523 li $t258, 111959
524 sb $t258, 111959
525 li $t259, 111960
526 sb $t259, 111960
527 li $t260, 111961
528 sb $t260, 111961
529 li $t261, 111962
530 sb $t261, 111962
531 li $t262, 111963
532 sb $t262, 111963
533 li $t263, 111964
534 sb $t263, 111964
535 li $t264, 111965
536 sb $t264, 111965
537 li $t265, 111966
538 sb $t265, 111966
539 li $t266, 111967
540 sb $t266, 111967
541 li $t267, 111968
542 sb $t267, 111968
543 li $t268, 111969
544 sb $t268, 111969
545 li $t269, 111970
546 sb $t269, 111970
547 li $t270, 111971
548 sb $t270, 111971
549 li $t271, 111972
550 sb $t271, 111972
551 li $t272, 111973
552 sb $t272, 111973
553 li $t273, 111974
554 sb $t273, 111974
555 li $t274, 111975
556 sb $t274, 111975
557 li $t275, 111976
558 sb $t275, 111976
559 li $t276, 111977
560 sb $t276, 111977
561 li $t277, 111978
562 sb $t277, 111978
563 li $t278, 111979
564 sb $t278, 111979
565 li $t279, 111980
566 sb $t279, 111980
567 li $t280, 111981
568 sb $t280, 111981
569 li $t281, 111982
570 sb $t281, 111982
571 li $t282, 111983
572 sb $t282, 111983
573 li $t283, 111984
574 sb $t283, 111984
575 li $t284, 111985
576 sb $t284, 111985
577 li $t285, 111986
578 sb $t285, 111986
579 li $t286, 111987
580 sb $t286, 111987
581 li $t287, 111988
582 sb $t287, 111988
583 li $t288, 111989
584 sb $t288, 111989
585 li $t289, 111990
586 sb $t289, 111990
587 li $t290, 111991
588 sb $t290, 111991
589 li $t291, 111992
590 sb $t291, 111992
591 li $t292, 111993
592 sb $t292, 111993
593 li $t293, 111994
594 sb $t293, 111994
595 li $t294, 111995
596 sb $t294, 111995
597 li $t295, 111996
598 sb $t295, 111996
599 li $t296, 111997
600 sb $t296, 111997
601 li $t297, 111998
602 sb $t297, 111998
603 li $t298, 111999
604 sb $t298, 111999
605 li $t299, 111900
606 sb $t299, 111900
607 li $t300, 111901
608 sb $t300, 111901
609 li $t301, 111902
610 sb $t301, 111902
611 li $t302, 111903
612 sb $t302, 111903
613 li $t303, 111904
614 sb $t303, 111904
615 li $t304, 111905
616 sb $t304, 111905
617 li $t305, 111906
618 sb $t305, 111906
619 li $t306, 111907
620 sb $t306, 111907
621 li $t307, 111908
622 sb $t307, 111908
623 li $t308, 111909
624 sb $t308, 111909
625 li $t309, 111910
626 sb $t309, 111910
627 li $t310, 111911
628 sb $t310, 111911
629 li $t311, 111912
630 sb $t311, 111912
631 li $t312, 111913
632 sb $t312, 111913
633 li $t313, 111914
634 sb $t313, 111914
635 li $t314, 111915
636 sb $t314, 111915
637 li $t315, 111916
638 sb $t315, 111916
639 li $t316, 111917
640 sb $t316, 111917
641 li $t317, 111918
642 sb $t317, 111918
643 li $t318, 111919
644 sb $t318, 111919
645 li $t319, 111920
646 sb $t319, 111920
647 li $t320, 111921
648 sb $t320, 111921
649 li $t321, 111922
650 sb $t321, 111922
651 li $t322, 111923
652 sb $t322, 111923
653 li $t323, 111924
654 sb $t323, 111924
655 li $t324, 111925
656 sb $t324, 111925
657 li $t325, 111926
658 sb $t325, 111926
659 li $t326, 111927
660 sb $t326, 111927
661 li $t327, 111928
662 sb $t327, 111928
663 li $t328, 111929
664 sb $t328, 111929
665 li $t329, 111930
666 sb $t329, 111930
667 li $t330, 111931
668 sb $t330, 111931
669 li $t331, 111932
670 sb $t331, 111932
671 li $t332, 111933
672 sb $t332, 111933
673 li $t333, 111934
674 sb $t333, 111934
675 li $t334, 111935
676 sb $t334, 111935
677 li $t335, 111936
678 sb $t335, 111936
679 li $t336, 111937
680 sb $t336, 111937
681 li $t337, 111938
682 sb $t337, 111938
683 li $t338, 111939
684 sb $t338, 111939
685 li $t339, 111940
686 sb $t339, 111940
687 li $t340, 111941
688 sb $t340, 111941
689 li $t341, 111942
690 sb $t341, 111942
691 li $t342, 111943
692 sb $t342, 111943
693 li $t343, 111944
694 sb $t343, 111944
695 li $t344, 111945
696 sb $t3
```

# In Pairs or Triples:

Line: 3 Go! Show/Hide Demos User Guide | Unit Tests | Docs

Addition Doubler Stav Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

```
1 # Store 'Hello world!' at the top of the stack
2 ADDI $sp, $sp, -13
3 ADDI $t0, $zero, 72 # H
4 SB $t0, 0($sp)
5 ADDI $t0, $zero, 101 # e
6 SB $t0, 1($sp)
7 ADDI $t0, $zero, 108 # l
8 SB $t0, 2($sp)
9 ADDI $t0, $zero, 108 # l
10 SB $t0, 3($sp)
11 ADDI $t0, $zero, 111 # o
12 SB $t0, 4($sp)
13 ADDI $t0, $zero, 32 # (space)
14 SB $t0, 5($sp)
15 ADDI $t0, $zero, 119 # w
16 SB $t0, 6($sp)
17 ADDI $t0, $zero, 111 # o
18 SB $t0, 7($sp)
19 ADDI $t0, $zero, 114 # r
20 SB $t0, 8($sp)
21 ADDI $t0, $zero, 108 # l
22 SB $t0, 9($sp)
23 ADDI $t0, $zero, 100 # d
24 SB $t0, 10($sp)
25 ADDI $t0, $zero, 33 # !
26 SB $t0, 11($sp)
27 ADDI $t0, $zero, 0 # (null)
28 SB $t0, 12($sp)
29
30 ADDI $v0, $zero, 4 # 4 is for print string
31 ADDI $a0, $sp, 0      # print to the log
32 syscall
```

Step Run  Enable auto switching

S	T	A	V	Stack	Log
s0:				10	
s1:				9	
s2:				9	
s3:				22	
s4:				696	
s5:				976	
s6:				927	
s7:				418	

Write a program that prints out the alphabet: a b c d ... x y z

# WeMIPS

The screenshot shows the WeMIPS debugger interface. At the top, there are tabs for 'Live', '3', 'Data', and 'ShowWhile Device'. Below these are navigation links: 'Addition Doubler', 'Stax', 'Looper', 'Stack Test', and 'Hello World'. Underneath are buttons for 'Code Gen Save String', 'Interactive', 'Binary2 Decimal', and 'Decimal2 Binary'. A 'Debug' button is also present.

The main area contains assembly code:

```
# Store 'Hello world!' at the top of the stack
1    ADDI $t0,$zero,72 # N
2    ADDI $t1,$zero,101 # e
3    ADDI $t2,$zero,101 # m
4    ADDI $t3,$zero,101 # l
5    ADDI $t4,$zero,101 # o
6    ADDI $t5,$zero,101 # r
7    ADDI $t6,$zero,101 # i
8    ADDI $t7,$zero,101 # n
9    ADDI $t8,$zero,101 # d
10   ADDI $t9,$zero,33 # !
11   ADDI $t10,$zero,0 # null
12   ADDI $t11,$zero,4 # 4 is for print string
13   ADDI $t12,$zero,0 # point to the log
14
15   # syscall
```

To the right, there is a table titled 'Registers' with columns for Step, T, A, V, Stack, and Log. The data is as follows:

Step	T	A	V	Stack	Log
s0	10				
s1	9				
s2	8				
s3	22				
s4	695				
s5	976				
s6	977				
s7	419				

(Demo with WeMIPS)

# Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- **Machine Language: Jumps & Loops**
- Binary & Hex Arithmetic
- Final Exam: Format

# Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.



# Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.



# Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.



# Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:



# Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
  - ▶ **Unconditional:** `j Done` will jump to the address with label `Done`.



# Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
  - Unconditional:** `j Done` will jump to the address with label `Done`.
  - Branch if Equal:** `beq $s0 $s1 DoAgain` will jump to the address with label `DoAgain` if the registers `$s0` and `$s1` contain the same value.



# Loops & Jumps in Machine Language

- Instead of built-in looping structures like `for` and `while`, you create your own loops by “jumping” to the location in the program.
- Can indicate locations by writing **labels** at the beginning of a line.
- Then give a command to jump to that location.
- Different kinds of jumps:
  - ▶ **Unconditional:** `j Done` will jump to the address with label `Done`.
  - ▶ **Branch if Equal:** `beq $s0 $s1 DoAgain` will jump to the address with label `DoAgain` if the registers `$s0` and `$s1` contain the same value.
  - ▶ See reading for more variations.



## Jump Demo

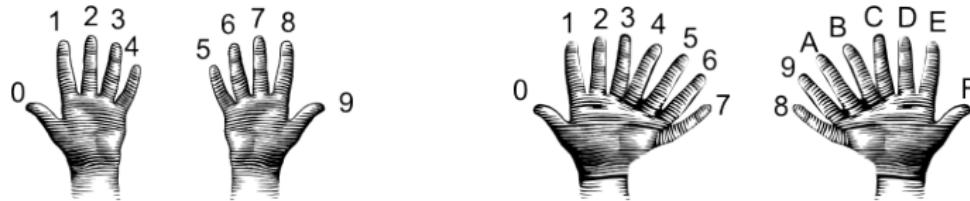
## (Demo with WeMIPS)

# Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- **Binary & Hex Arithmetic**
- Final Exam: Format

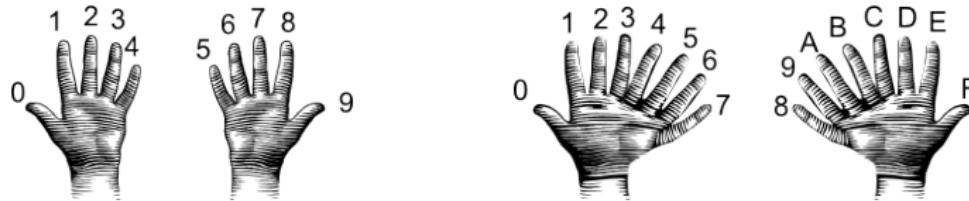
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.

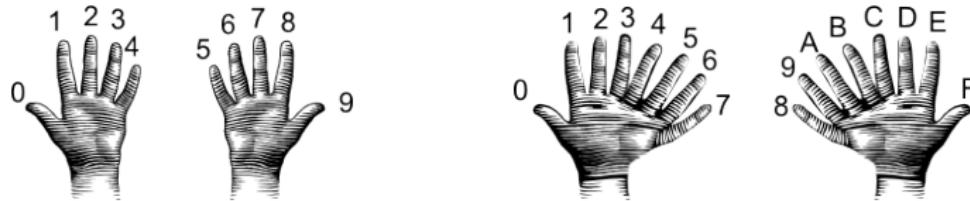
# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:
  - Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.

# Hexadecimal to Decimal: Converting Between Bases

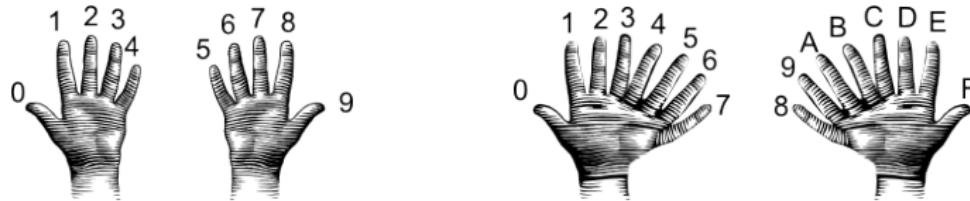


(from i-programmer.info)

- From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
- Convert second digit to decimal and add to total.
- Example: what is 2A as a decimal number?

# Hexadecimal to Decimal: Converting Between Bases

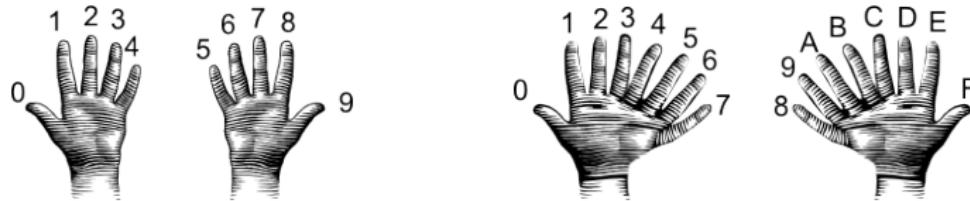


(from i-programmer.info)

- From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
- Convert second digit to decimal and add to total.
- Example: what is 2A as a decimal number?  
2 in decimal is 2.

# Hexadecimal to Decimal: Converting Between Bases



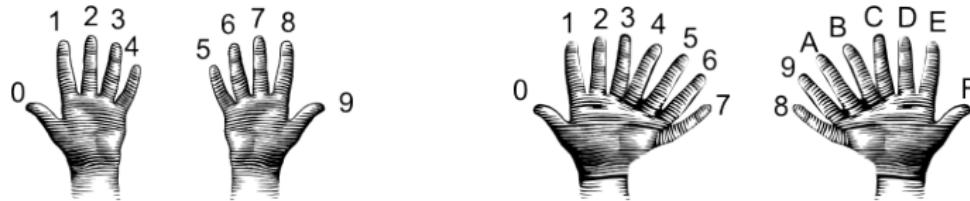
(from i-programmer.info)

- From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
- Convert second digit to decimal and add to total.
- Example: what is 2A as a decimal number?

2 in decimal is 2.  $2 \times 16$  is 32.

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

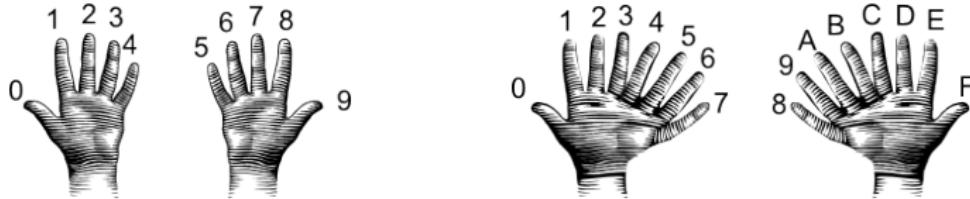
- From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
- Convert second digit to decimal and add to total.
- Example: what is 2A as a decimal number?

2 in decimal is 2.  $2 \times 16$  is 32.

A in decimal digits is 10.

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:

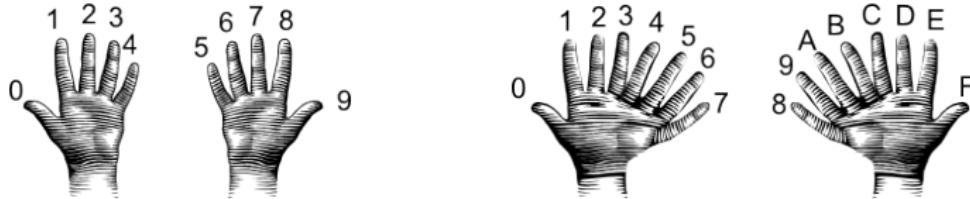
- Convert first digit to decimal and multiple by 16.
- Convert second digit to decimal and add to total.
- Example: what is 2A as a decimal number?

2 in decimal is 2.  $2 \times 16$  is 32.

A in decimal digits is 10.

$32 + 10$  is 42.

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?

2 in decimal is 2.  $2 \times 16$  is 32.

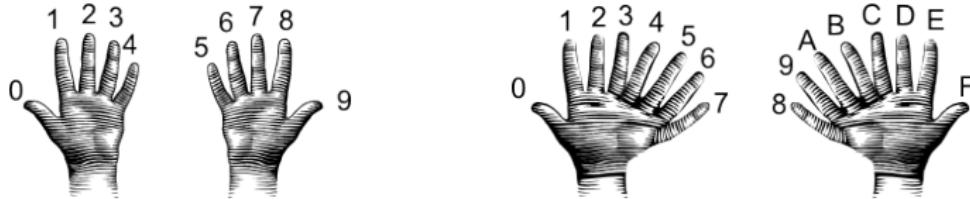
A in decimal digits is 10.

$32 + 10$  is 42.

Answer is 42.

- Example: what is 99 as a decimal number?

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?

2 in decimal is 2.  $2 \times 16$  is 32.

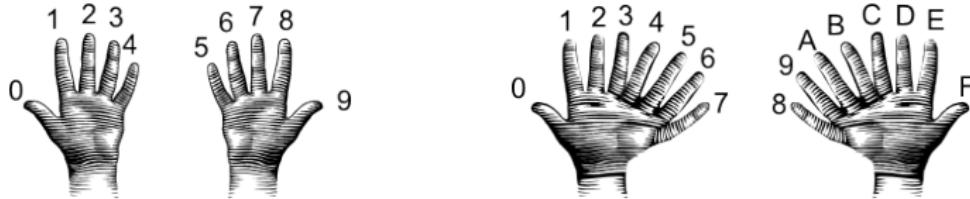
A in decimal digits is 10.

$32 + 10$  is 42.

Answer is 42.

- Example: what is 99 as a decimal number?
  - 9 in decimal is 9.

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?

2 in decimal is 2.  $2 \times 16$  is 32.

A in decimal digits is 10.

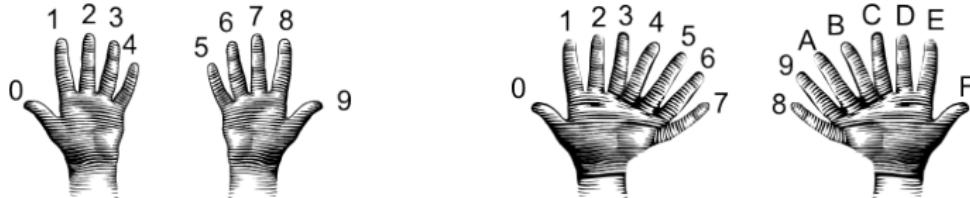
$32 + 10$  is 42.

Answer is 42.

- Example: what is 99 as a decimal number?

9 in decimal is 9.  $9 \times 16$  is 144.

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?

2 in decimal is 2.  $2 \times 16$  is 32.

A in decimal digits is 10.

$32 + 10$  is 42.

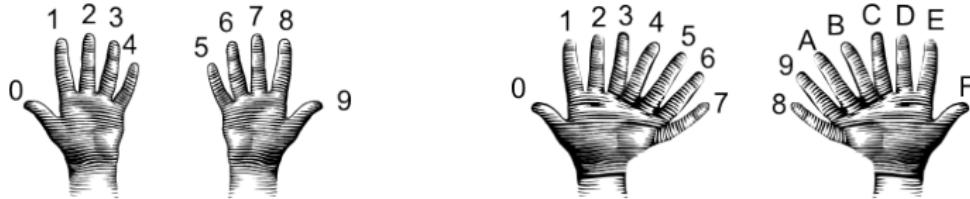
Answer is 42.

- Example: what is 99 as a decimal number?

9 in decimal is 9.  $9 \times 16$  is 144.

9 in decimal digits is 9

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?

2 in decimal is 2.  $2 \times 16$  is 32.

A in decimal digits is 10.

$32 + 10$  is 42.

Answer is 42.

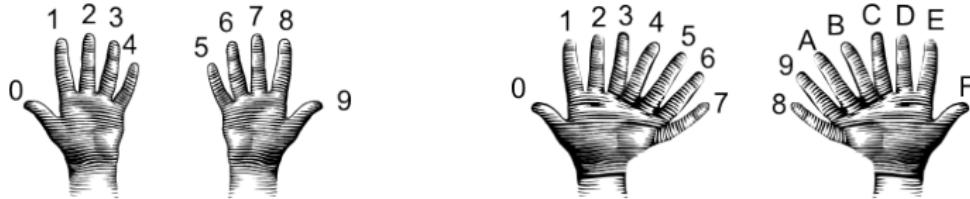
- Example: what is 99 as a decimal number?

9 in decimal is 9.  $9 \times 16$  is 144.

9 in decimal digits is 9

$144 + 9$  is 153.

# Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal:

- Convert first digit to decimal and multiple by 16.
  - Convert second digit to decimal and add to total.
  - Example: what is 2A as a decimal number?

2 in decimal is 2.  $2 \times 16$  is 32.

A in decimal digits is 10.

$32 + 10$  is 42.

Answer is 42.

- Example: what is 99 as a decimal number?

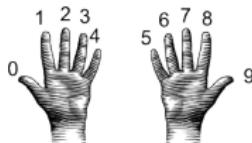
9 in decimal is 9.  $9 \times 16$  is 144.

9 in decimal digits is 9

$144 + 9$  is 153.

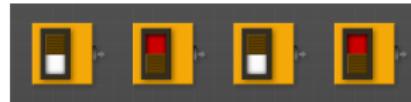
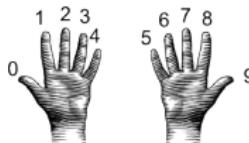
Answer is 153.

# Decimal to Binary: Converting Between Bases



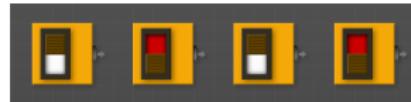
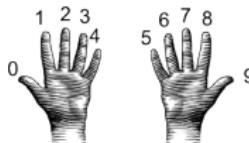
- From decimal to binary:
  - Divide by  $128 (= 2^7)$ . Quotient is the first digit.

# Decimal to Binary: Converting Between Bases



- From decimal to binary:
  - Divide by  $128 (= 2^7)$ . Quotient is the first digit.
  - Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.

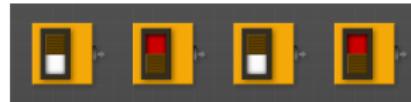
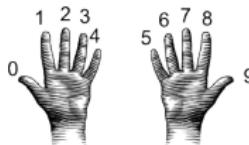
# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.

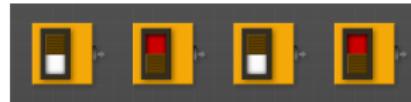
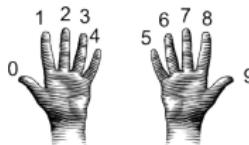
# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.

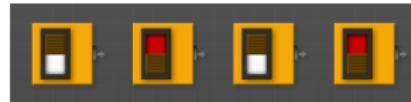
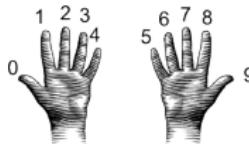
# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.

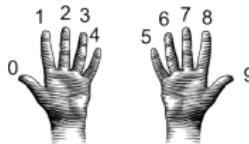
# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.

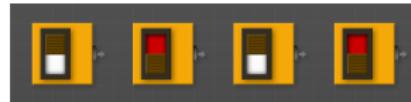
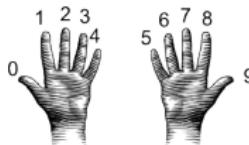
# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.

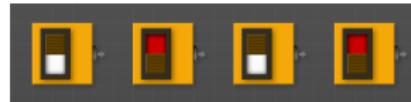
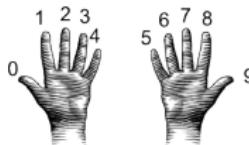
# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.

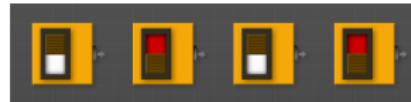
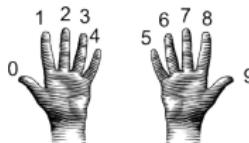
# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

# Decimal to Binary: Converting Between Bases

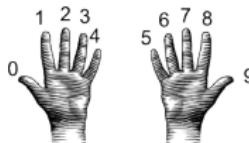


- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2.

# Decimal to Binary: Converting Between Bases

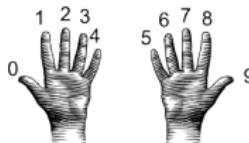


- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1:

# Decimal to Binary: Converting Between Bases



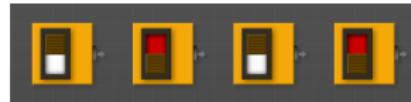
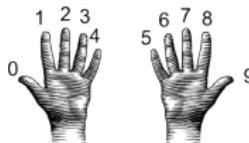
- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2.

# Decimal to Binary: Converting Between Bases



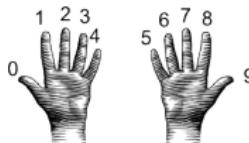
- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0:

# Decimal to Binary: Converting Between Bases



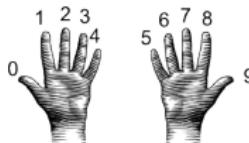
- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

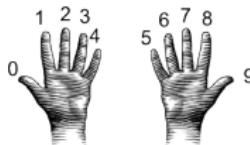
- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2.

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

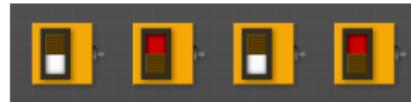
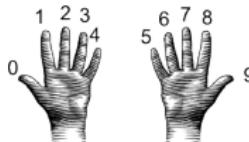
- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0:

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

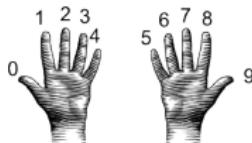
- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

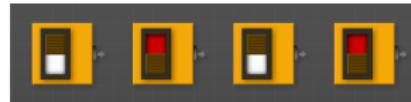
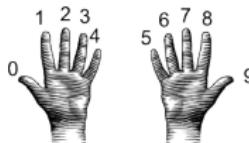
130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2.

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

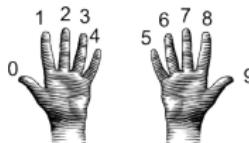
130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0:

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

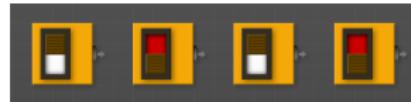
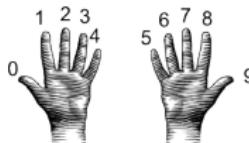
130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

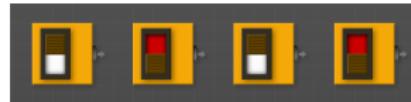
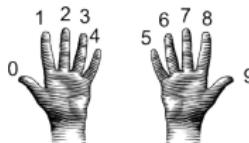
2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

2/8 is 0 rem 2.

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

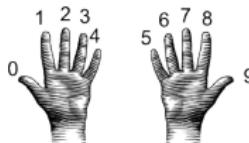
2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

2/8 is 0 rem 2. Next digit is 0:

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

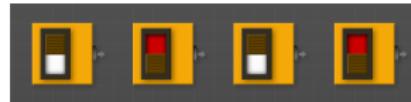
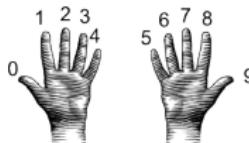
2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

2/8 is 0 rem 2. Next digit is 0: 10000...

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

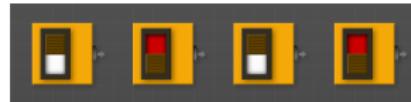
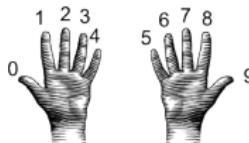
2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2.

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

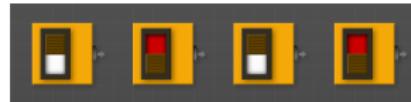
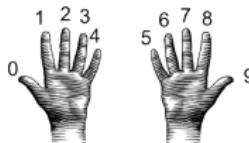
2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0:

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

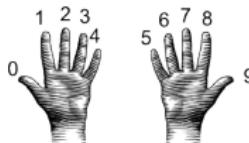
2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0: 100000...

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- ▶ Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- ▶ Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- ▶ Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- ▶ Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- ▶ Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- ▶ Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- ▶ Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- ▶ The last remainder is the last digit.
- ▶ Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

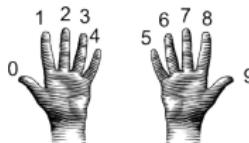
2/16 is 0 rem 2. Next digit is 0: 1000...

2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0: 100000...

2/2 is 1 rem 0.

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

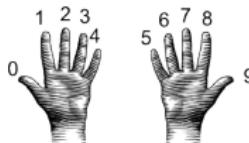
2/16 is 0 rem 2. Next digit is 0: 1000...

2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0: 100000...

2/2 is 1 rem 0. Next digit is 1:

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

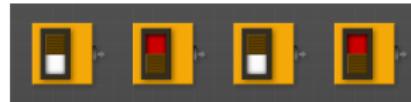
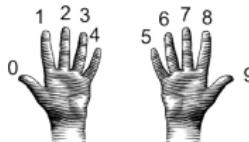
2/16 is 0 rem 2. Next digit is 0: 1000...

2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0: 100000...

2/2 is 1 rem 0. Next digit is 1: 1000001...

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

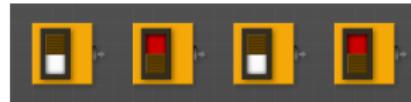
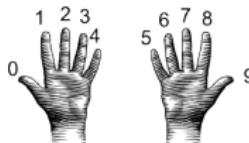
2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0: 100000...

2/2 is 1 rem 0. Next digit is 1: 1000001...

Adding the last remainder: 10000010

# Decimal to Binary: Converting Between Bases



- From decimal to binary:

- Divide by  $128 (= 2^7)$ . Quotient is the first digit.
- Divide remainder by  $64 (= 2^6)$ . Quotient is the next digit.
- Divide remainder by  $32 (= 2^5)$ . Quotient is the next digit.
- Divide remainder by  $16 (= 2^4)$ . Quotient is the next digit.
- Divide remainder by  $8 (= 2^3)$ . Quotient is the next digit.
- Divide remainder by  $4 (= 2^2)$ . Quotient is the next digit.
- Divide remainder by  $2 (= 2^1)$ . Quotient is the next digit.
- The last remainder is the last digit.
- Example: what is 130 in binary notation?

130/128 is 1 rem 2. First digit is 1: 1...

2/64 is 0 rem 2. Next digit is 0: 10...

2/32 is 0 rem 2. Next digit is 0: 100...

2/16 is 0 rem 2. Next digit is 0: 1000...

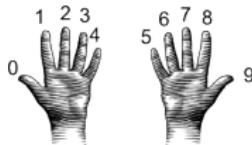
2/8 is 0 rem 2. Next digit is 0: 10000...

2/4 is 0 remainder 2. Next digit is 0: 100000...

2/2 is 1 rem 0. Next digit is 1: 1000001...

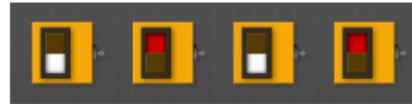
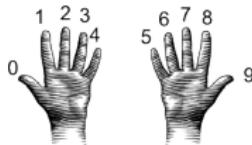
Adding the last remainder: 10000010

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

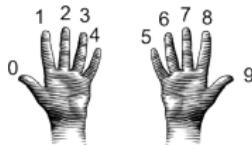
# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

$99/128$  is 0 rem 99.

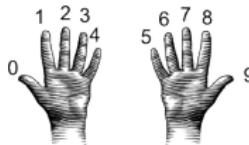
# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0:

# Decimal to Binary: Converting Between Bases

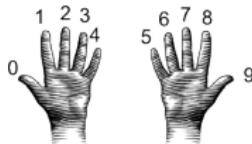


- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35.

# Decimal to Binary: Converting Between Bases

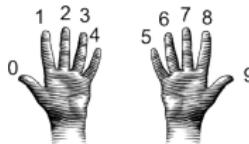


- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1:

# Decimal to Binary: Converting Between Bases

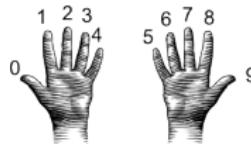


- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

# Decimal to Binary: Converting Between Bases



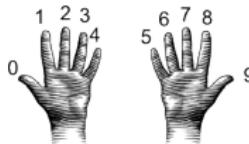
- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3.

# Decimal to Binary: Converting Between Bases



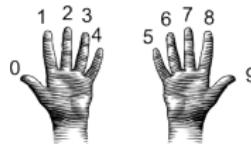
- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1:

# Decimal to Binary: Converting Between Bases



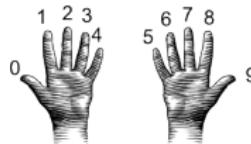
- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

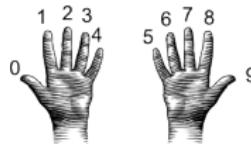
99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3.

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

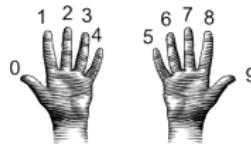
99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0:

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

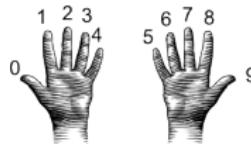
99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

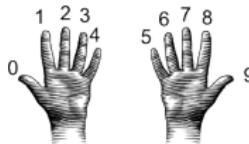
99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3.

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

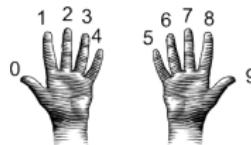
99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0:

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

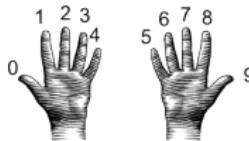
99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

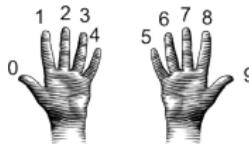
35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3.

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

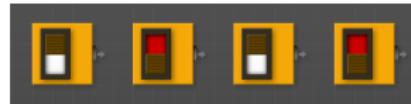
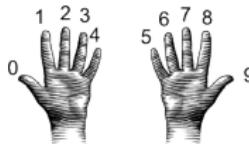
35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0:

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

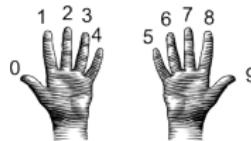
35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

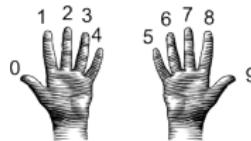
3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1.

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

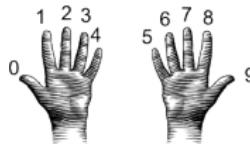
3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1. Next digit is 1:

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

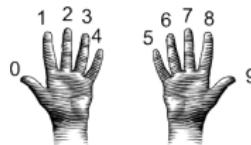
3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1. Next digit is 1: 0110001...

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

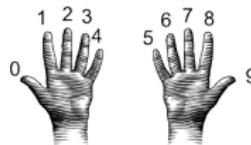
3/8 is 0 rem 3. Next digit is 0: 01100...

3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1. Next digit is 1: 0110001...

Adding the last remainder: 01100011

# Decimal to Binary: Converting Between Bases



- Example: what is 99 in binary notation?

99/128 is 0 rem 99. First digit is 0: 0...

99/64 is 1 rem 35. Next digit is 1: 01...

35/32 is 1 rem 3. Next digit is 1: 011...

3/16 is 0 rem 3. Next digit is 0: 0110...

3/8 is 0 rem 3. Next digit is 0: 01100...

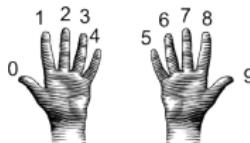
3/4 is 0 remainder 3. Next digit is 0: 011000...

3/2 is 1 rem 1. Next digit is 1: 0110001...

Adding the last remainder: 01100011

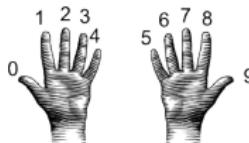
Answer is 1100011.

# Binary to Decimal: Converting Between Bases



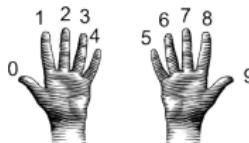
- From binary to decimal:
  - Set sum = last digit.

# Binary to Decimal: Converting Between Bases



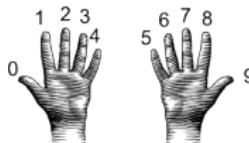
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by  $2^1$ . Add to sum.

# Binary to Decimal: Converting Between Bases



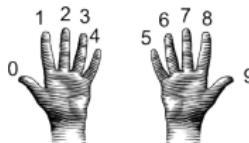
- From binary to decimal:
  - Set sum = last digit.
  - Multiply next digit by  $2 = 2^1$ . Add to sum.
  - Multiply next digit by  $4 = 2^2$ . Add to sum.

# Binary to Decimal: Converting Between Bases



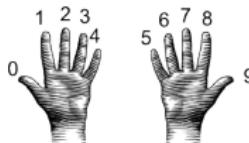
- From binary to decimal:
  - ▶ Set sum = last digit.
  - ▶ Multiply next digit by  $2^1$ . Add to sum.
  - ▶ Multiply next digit by  $2^2$ . Add to sum.
  - ▶ Multiply next digit by  $2^3$ . Add to sum.

# Binary to Decimal: Converting Between Bases



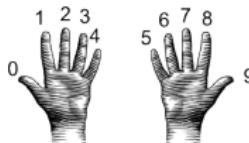
- From binary to decimal:
  - ▶ Set sum = last digit.
  - ▶ Multiply next digit by  $2 = 2^1$ . Add to sum.
  - ▶ Multiply next digit by  $4 = 2^2$ . Add to sum.
  - ▶ Multiply next digit by  $8 = 2^3$ . Add to sum.
  - ▶ Multiply next digit by  $16 = 2^4$ . Add to sum.

# Binary to Decimal: Converting Between Bases



- From binary to decimal:
  - ▶ Set sum = last digit.
  - ▶ Multiply next digit by  $2^1$ . Add to sum.
  - ▶ Multiply next digit by  $2^2$ . Add to sum.
  - ▶ Multiply next digit by  $2^3$ . Add to sum.
  - ▶ Multiply next digit by  $2^4$ . Add to sum.
  - ▶ Multiply next digit by  $2^5$ . Add to sum.

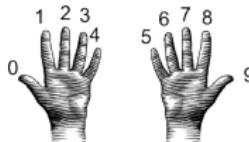
# Binary to Decimal: Converting Between Bases



- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.

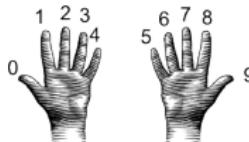
# Binary to Decimal: Converting Between Bases



- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.

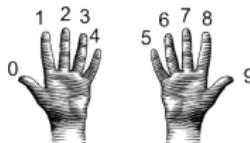
# Binary to Decimal: Converting Between Bases



- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.

# Binary to Decimal: Converting Between Bases

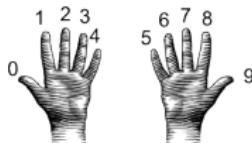


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with:

# Binary to Decimal: Converting Between Bases



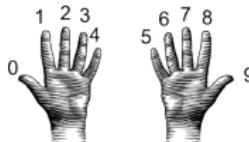
- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2 = 2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with: 1

$0 * 2 = 0$ . Add 0 to sum:

# Binary to Decimal: Converting Between Bases

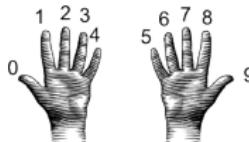


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with:                    1  
 $0 \times 2 = 0$ . Add 0 to sum:        1

# Binary to Decimal: Converting Between Bases



- From binary to decimal:

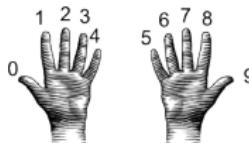
- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with: 1

$0 * 2 = 0$ . Add 0 to sum: 1

$1 * 4 = 4$ . Add 4 to sum:

# Binary to Decimal: Converting Between Bases



- From binary to decimal:

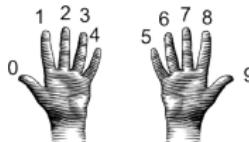
- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with: 1

$0 * 2 = 0$ . Add 0 to sum: 1

$1 * 4 = 4$ . Add 4 to sum: 5

# Binary to Decimal: Converting Between Bases



- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

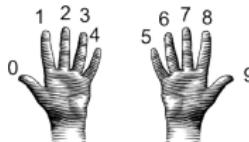
Sum starts with: 1

$0 * 2 = 0$ . Add 0 to sum: 1

$1 * 4 = 4$ . Add 4 to sum: 5

$1 * 8 = 8$ . Add 8 to sum:

# Binary to Decimal: Converting Between Bases

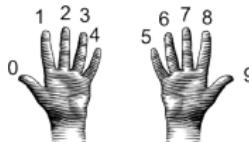


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with: 1  
 $0*2 = 0$ . Add 0 to sum: 1  
 $1*4 = 4$ . Add 4 to sum: 5  
 $1*8 = 8$ . Add 8 to sum: 13

# Binary to Decimal: Converting Between Bases



- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with: 1

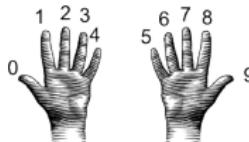
$0*2 = 0$ . Add 0 to sum: 1

$1*4 = 4$ . Add 4 to sum: 5

$1*8 = 8$ . Add 8 to sum: 13

$1*16 = 16$ . Add 16 to sum:

# Binary to Decimal: Converting Between Bases

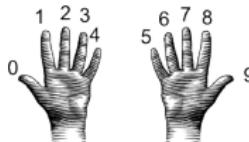


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with: 1  
 $0*2 = 0$ . Add 0 to sum: 1  
 $1*4 = 4$ . Add 4 to sum: 5  
 $1*8 = 8$ . Add 8 to sum: 13  
 $1*16 = 16$ . Add 16 to sum: 29

# Binary to Decimal: Converting Between Bases

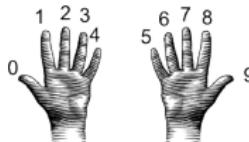


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with: 1  
 $0*2 = 0$ . Add 0 to sum: 1  
 $1*4 = 4$ . Add 4 to sum: 5  
 $1*8 = 8$ . Add 8 to sum: 13  
 $1*16 = 16$ . Add 16 to sum: 29  
 $1*32 = 32$ . Add 32 to sum:

# Binary to Decimal: Converting Between Bases

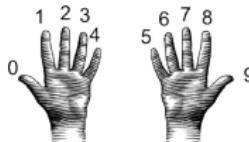


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with: 1  
 $0*2 = 0$ . Add 0 to sum: 1  
 $1*4 = 4$ . Add 4 to sum: 5  
 $1*8 = 8$ . Add 8 to sum: 13  
 $1*16 = 16$ . Add 16 to sum: 29  
 $1*32 = 32$ . Add 32 to sum: 61

# Binary to Decimal: Converting Between Bases

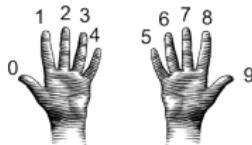


- From binary to decimal:

- Set sum = last digit.
- Multiply next digit by  $2 = 2^1$ . Add to sum.
- Multiply next digit by  $4 = 2^2$ . Add to sum.
- Multiply next digit by  $8 = 2^3$ . Add to sum.
- Multiply next digit by  $16 = 2^4$ . Add to sum.
- Multiply next digit by  $32 = 2^5$ . Add to sum.
- Multiply next digit by  $64 = 2^6$ . Add to sum.
- Multiply next digit by  $128 = 2^7$ . Add to sum.
- Sum is the decimal number.
- Example: What is 111101 in decimal?

Sum starts with:	1
$0*2 = 0$ . Add 0 to sum:	1
$1*4 = 4$ . Add 4 to sum:	5
$1*8 = 8$ . Add 8 to sum:	13
$1*16 = 16$ . Add 16 to sum:	29
$1*32 = 32$ . Add 32 to sum:	61

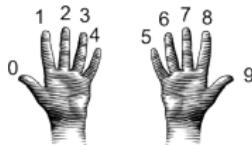
# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:

# Binary to Decimal: Converting Between Bases



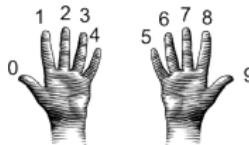
- Example: What is 10100100 in decimal?

Sum starts with:

0

$0 \times 2 = 0$ . Add 0 to sum:

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

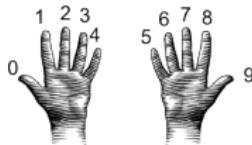
Sum starts with:

0

$0 \times 2 = 0$ . Add 0 to sum:

0

# Binary to Decimal: Converting Between Bases



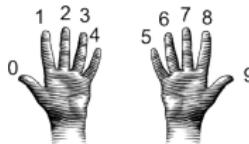
- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 * 2 = 0$ . Add 0 to sum: 0

$1 * 4 = 4$ . Add 4 to sum:

# Binary to Decimal: Converting Between Bases



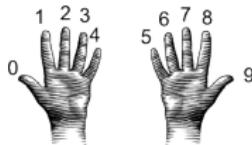
- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 * 2 = 0$ . Add 0 to sum: 0

$1 * 4 = 4$ . Add 4 to sum: 4

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

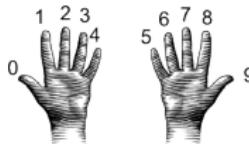
Sum starts with: 0

$0 \times 2 = 0$ . Add 0 to sum: 0

$1 \times 4 = 4$ . Add 4 to sum: 4

$0 \times 8 = 0$ . Add 0 to sum:

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

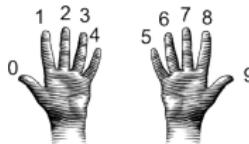
Sum starts with: 0

$0 \times 2 = 0$ . Add 0 to sum: 0

$1 \times 4 = 4$ . Add 4 to sum: 4

$0 \times 8 = 0$ . Add 0 to sum: 4

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with: 0

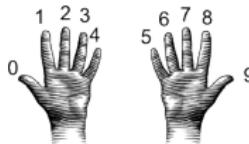
$0 * 2 = 0$ . Add 0 to sum: 0

$1 * 4 = 4$ . Add 4 to sum: 4

$0 * 8 = 0$ . Add 0 to sum: 4

$0 * 16 = 0$ . Add 0 to sum:

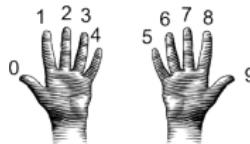
# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0.$ Add 0 to sum:	0
$1 \times 4 = 4.$ Add 4 to sum:	4
$0 \times 8 = 0.$ Add 0 to sum:	4
$0 \times 16 = 0.$ Add 0 to sum:	4

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$ . Add 0 to sum: 0

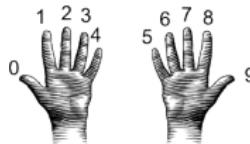
$1 \times 4 = 4$ . Add 4 to sum: 4

$0 \times 8 = 0$ . Add 0 to sum: 4

$0 \times 16 = 0$ . Add 0 to sum: 4

$1 \times 32 = 32$ . Add 32 to sum:

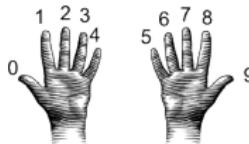
# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0.$ Add 0 to sum:	0
$1 \times 4 = 4.$ Add 4 to sum:	4
$0 \times 8 = 0.$ Add 0 to sum:	4
$0 \times 16 = 0.$ Add 0 to sum:	4
$1 \times 32 = 32.$ Add 32 to sum:	36

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$ . Add 0 to sum: 0

$1 \times 4 = 4$ . Add 4 to sum: 4

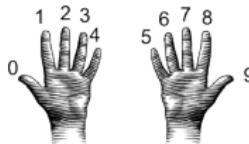
$0 \times 8 = 0$ . Add 0 to sum: 4

$0 \times 16 = 0$ . Add 0 to sum: 4

$1 \times 32 = 32$ . Add 32 to sum: 36

$0 \times 64 = 0$ . Add 0 to sum:

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$ . Add 0 to sum: 0

$1 \times 4 = 4$ . Add 4 to sum: 4

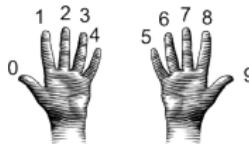
$0 \times 8 = 0$ . Add 0 to sum: 4

$0 \times 16 = 0$ . Add 0 to sum: 4

$1 \times 32 = 32$ . Add 32 to sum: 36

$0 \times 64 = 0$ . Add 0 to sum: 36

# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with: 0

$0 \times 2 = 0$ . Add 0 to sum: 0

$1 \times 4 = 4$ . Add 4 to sum: 4

$0 \times 8 = 0$ . Add 0 to sum: 4

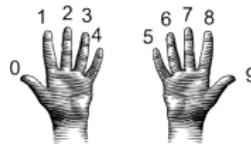
$0 \times 16 = 0$ . Add 0 to sum: 4

$1 \times 32 = 32$ . Add 32 to sum: 36

$0 \times 64 = 0$ . Add 0 to sum: 36

$1 \times 128 = 0$ . Add 128 to sum:

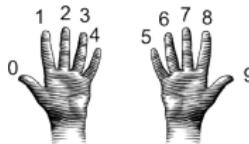
# Binary to Decimal: Converting Between Bases



- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0$ . Add 0 to sum:	0
$1 \times 4 = 4$ . Add 4 to sum:	4
$0 \times 8 = 0$ . Add 0 to sum:	4
$0 \times 16 = 0$ . Add 0 to sum:	4
$1 \times 32 = 32$ . Add 32 to sum:	36
$0 \times 64 = 0$ . Add 0 to sum:	36
$1 \times 128 = 128$ . Add 128 to sum:	164

# Binary to Decimal: Converting Between Bases

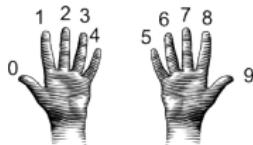


- Example: What is 10100100 in decimal?

Sum starts with:	0
$0 \times 2 = 0.$ Add 0 to sum:	0
$1 \times 4 = 4.$ Add 4 to sum:	4
$0 \times 8 = 0.$ Add 0 to sum:	4
$0 \times 16 = 0.$ Add 0 to sum:	4
$1 \times 32 = 32.$ Add 32 to sum:	36
$0 \times 64 = 0.$ Add 0 to sum:	36
$1 \times 128 = 128.$ Add 128 to sum:	164

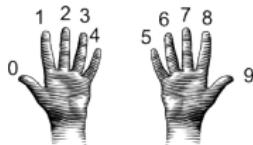
The answer is 164.

# Design Challenge: Incrementers



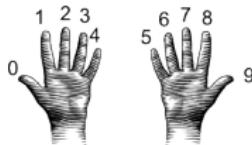
- Simplest arithmetic: add one ("increment") a variable.

# Design Challenge: Incrementers



- Simplest arithmetic: add one (“increment”) a variable.
- Example: Increment a decimal number:

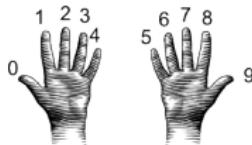
# Design Challenge: Incrementers



- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```

# Design Challenge: Incrementers

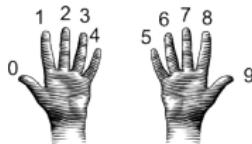


- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```

- Challenge: Write an algorithm for incrementing numbers expressed as words.

# Design Challenge: Incrementers

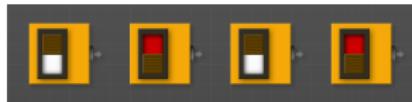
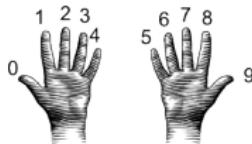


- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```

- Challenge: Write an algorithm for incrementing numbers expressed as words.  
Example: "forty one" → "forty two"

# Design Challenge: Incrementers



- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

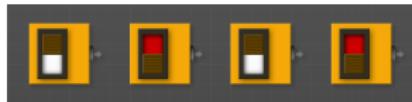
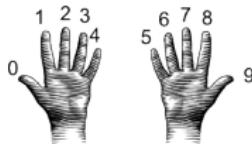
```
def addOne(n):  
    m = n+1  
    return(m)
```

- Challenge: Write an algorithm for incrementing numbers expressed as words.

Example: "forty one" → "forty two"

*Hint: Convert to numbers, increment, and convert back to strings.*

# Design Challenge: Incrementers

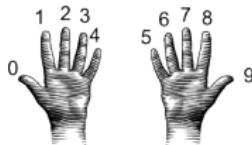


- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```

- Challenge: Write an algorithm for incrementing numbers expressed as words.  
Example: "forty one" → "forty two"  
*Hint: Convert to numbers, increment, and convert back to strings.*
- Challenge: Write an algorithm for incrementing binary numbers.

# Design Challenge: Incrementers



- Simplest arithmetic: add one ("increment") a variable.
- Example: Increment a decimal number:

```
def addOne(n):  
    m = n+1  
    return(m)
```

- Challenge: Write an algorithm for incrementing numbers expressed as words.  
Example: "forty one" → "forty two"  
*Hint: Convert to numbers, increment, and convert back to strings.*
- Challenge: Write an algorithm for incrementing binary numbers.  
Example: "1001" → "1010"

# Recap



- Searching through data is a common task— built-in functions and standard design patterns for this.

# Recap



- Searching through data is a common task— built-in functions and standard design patterns for this.
- Programming languages can be classified by the level of abstraction and direct access to data.

# Recap



- Searching through data is a common task— built-in functions and standard design patterns for this.
- Programming languages can be classified by the level of abstraction and direct access to data.

# Today's Topics



- Design Patterns: Searching
- Python Recap
- Machine Language
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- **Final Exam: Format**

## Final Overview: Administration

- The exam will be administered through Gradescope.

# Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **May 18, 9am-10:30am**

# Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **May 18, 9am-10:30am**
- There will be a different Gradescope Course called **CSci 127 Final Exam**

# Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **May 18, 9am-10:30am**
- There will be a different Gradescope Course called **CSci 127 Final Exam**
- Prior to the exam you will be added to the final exam course for your exam version.

# Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **May 18, 9am-10:30am**
- There will be a different Gradescope Course called **CSci 127 Final Exam**
- Prior to the exam you will be added to the final exam course for your exam version.
- The only assignment in that course will be your final exam.

# Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on **May 18, 9am-10:30am**
- There will be a different Gradescope Course called **CSci 127 Final Exam**
- Prior to the exam you will be added to the final exam course for your exam version.
- The only assignment in that course will be your final exam.
- The morning of the exam: log into Gradescope, find the **CSci 127 Final Exam** course and open the assignment.

# Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.

# Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
  - ▶ With notes, examples, programs: what will help you on the exam.

# Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ Best if you design/write yours since excellent way to study.

# Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:

# Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
  - ▶ 10 questions, each worth 10 points.

# Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
  - ▶ 10 questions, each worth 10 points.
  - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.

# Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
  - ▶ 10 questions, each worth 10 points.
  - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
  - ▶ Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.

# Final Overview: Format

- Although the exam is remote, we still suggest you prepare 1 piece of **8.5" x 11"** paper.
  - ▶ With notes, examples, programs: what will help you on the exam.
  - ▶ Best if you design/write yours since excellent way to study.
- The exam format:
  - ▶ 10 questions, each worth 10 points.
  - ▶ Questions correspond to the course topics, and are variations on the programming assignments, lab exercises, and lecture design challenges.
  - ▶ Style of questions: what does the code do? short answer, write functions, top down design, & write complete programs.
- Past exams available on webpage (includes answer keys).

# Exam Options

## Exam Times:

FINAL EXAM, VERSION 3  
CSci 127: Introduction to Computer Science  
Hunter College, City University of New York

19 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart watch, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, among others) as serious violations of the Honor Code. Such acts are considered a violation against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and pursues cases of academic dishonesty according to the Hunter College Academic Integrity Procedure.

<i>I acknowledge that all content of academic dishonesty will be reported to the Office of Student Life and will result in sanctions.</i>
Name: _____
Signature: _____
Email: _____
Signature: _____

# Exam Options

## Exam Times:

- Default: Regular Time: Monday, 18 May, 9-10:30am.
- Alternate Time: Reading Day, Friday, 15 May, 8:00am-9:30am.
- Accessibility Testing: For double time must contact Prof. Ligorio by 15 May.

FINAL EXAM, VERSION 3  
CSci 127: Introduction to Computer Science  
Hunter College, City University of New York

19 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart phone, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, among others) as serious violations of the College's Honor Code and as violations of the College's policies against the values of intellectual honesty. The College is committed to enforcing the CUNY Policy on Academic Integrity and pursues cases of academic dishonesty according to the Hunter College Academic Integrity Procedure.

Indicate that all content is available electronically and/or referred to the links of Hunter and all used in exam.
Name:
Signature:
Email:
Signature:

# Exam Options

## Exam Times:

- Default: Regular Time: Monday, 18 May, 9-10:30am.
- Alternate Time: Reading Day, Friday, 15 May, 8:00am-9:30am.
- Accessibility Testing: For double time must contact Prof. Ligorio by 15 May.

## Grading Options:

FINAL EXAM, VERSION 3  
CSci 127: Introduction to Computer Science  
Hunter College, City University of New York

19 December 2018

**Exam Rules**

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart phone, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, commercializing, and other forms of academic dishonesty) as serious violations of the College's mission against the values of individual integrity. The College is committed to enforcing the CUNY Policy on Academic Integrity and pursues cases of academic dishonesty according to the Hunter College Academic Integrity Procedure.

Indicates that all content is available electronically and is referred to the Honor of the Student and all credit is earned.
Name: _____
SugID# _____
Email: _____
Signature: _____

# Exam Options

## Exam Times:

- Default: Regular Time: Monday, 18 May, 9-10:30am.
- Alternate Time: Reading Day, Friday, 15 May, 8:00am-9:30am.
- Accessibility Testing: For double time must contact Prof. Ligorio by 15 May.

## Grading Options:

- Default: Letter Grade.
- Credit/NoCredit grade— availability depends on major and academic standing.

FINAL EXAM, VERSION 3  
CSci 127: Introduction to Computer Science  
Hunter College, City University of New York

19 December 2018

### Exam Rules

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have with you pens and pencils, and your note sheet.
- You may not use a computer, calculator, tablet, smart phone, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, fabrication, and aiding and abetting) as serious violations of the Honor Code and as violations against the values of individual integrity. The College is committed to enforcing the CUNY Policy on Academic Integrity and punishes acts of academic dishonesty according to the Hunter College Academic Integrity Procedure.

Indicate that all content is available electronically and/or referred to the links of Electronic and all content in questions.
Name:
SugnOff:
Email:
Signature:

# Exam Options

## Exam Times:

- Default: Regular Time: Monday, 18 May, 9-10:30am.
- Alternate Time: Reading Day, Friday, 15 May, 8:00am-9:30am.
- Accessibility Testing: For double time must contact Prof. Ligorio by 15 May.

## Grading Options:

- Default: Letter Grade.
- Credit/NoCredit grade— availability depends on major and academic standing.

Survey for your choices will be available next lecture.

# Exam Options

## Exam Times:

- Default: Regular Time: Monday, 18 May, 9-10:30am.
- Alternate Time: Reading Day, Friday, 15 May, 8:00am-9:30am.
- Accessibility Testing: For double time must contact Prof. Ligorio by 15 May.

FINAL EXAM, VERSION 3  
CSci 122: Introduction to Computer Science  
Hunter College, City University of New York

19 December 2018

**Exam Rules**

- Show all your work. Your grade will be based on the work shown.
- The exam is closed book and closed notes with the exception of an 8.5" x 11" piece of paper that you can write on.
- When taking the exam, you may have three 200 page and pencil, and your note sheet.
- You may not use a computer, calculator, tablet, smart phone, or other electronic device.
- Do not open this exam until instructed to do so.

Hunter College regards acts of academic dishonesty (e.g., plagiarism, cheating or communism, among others) as serious violations of the Honor Code. Such acts are considered a violation against the values of individual integrity. The College is committed to enforcing the CUNY Policy on Academic Integrity and punishes acts of academic dishonesty according to the Hunter College Academic Integrity Procedure.

Information that will be checked electronically will be reported to the Office of Student Life and will result in sanctions.
Name:
SugID#
Email:
Signature:

## Grading Options:

- Default: Letter Grade.
- Credit/NoCredit grade— availability depends on major and academic standing.

Survey for your choices will be available next lecture.

No survey answer implies you will take the exam on 18 May, regular time and letter grade.