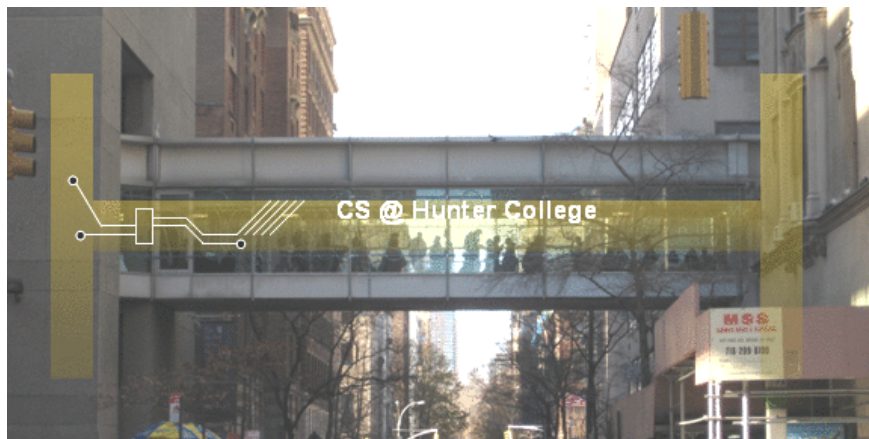


CSci 127: Introduction to Computer Science



hunter.cuny.edu/csci

Frequently Asked Questions

From email and tutoring.

- **When is the final? Is there a review sheet?**

Frequently Asked Questions

From email and tutoring.

- **When is the final? Is there a review sheet?**

The official final is the last day of class 10-12pm.

Frequently Asked Questions

From email and tutoring.

- **When is the final? Is there a review sheet?**

The official final is the last day of class 10-12pm.

Instead of a review sheet, we have:

Frequently Asked Questions

From email and tutoring.

- **When is the final? Is there a review sheet?**

The official final is the last day of class 10-12pm.

Instead of a review sheet, we have:

- ▶ *All previous final exams (and answer keys) on the website.*

Frequently Asked Questions

From email and tutoring.

- **When is the final? Is there a review sheet?**

The official final is the last day of class 10-12pm.

Instead of a review sheet, we have:

- ▶ *All previous final exams (and answer keys) on the website.*
- ▶ *Our TA Lola is happy to review concepts and old exam questions.*

Frequently Asked Questions

From email and tutoring.

- **When is the final? Is there a review sheet?**

The official final is the last day of class 10-12pm.

Instead of a review sheet, we have:

- ▶ *All previous final exams (and answer keys) on the website.*
- ▶ *Our TA Lola is happy to review concepts and old exam questions.*
- ▶ *There will be opportunity for some practice and an ungraded mock exam available on Gradescope.*

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

- Example of **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.')
```

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: 9 Classes in 10 Minutes



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

Class 1: print(), loops, comments, & turtles

Class 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*

Class 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)

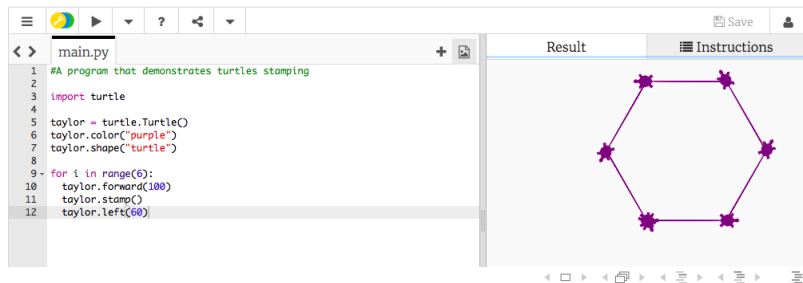
```
#This program prints: Hello, World!
```

← (this one also)

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

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

- As well as definite loops & the turtle package:



Class 1: variables, data types, more on loops & range()

Class 1: variables, data types, more on loops & range()

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

Class 1: 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

Class 1: 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

Class 1: 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

Class 1: 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

Class 1: 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 1: 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.

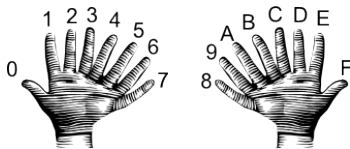
Class 1: variables, data types, more on loops & range()

- 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)
```

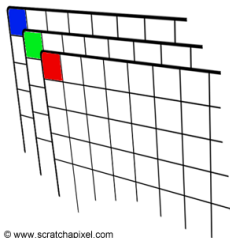
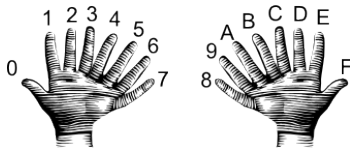
Class 2: colors, hex, slices, numpy & images

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








Class 2: colors, hex, slices, numpy & images

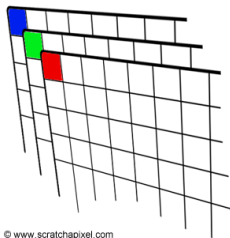
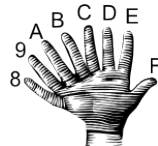
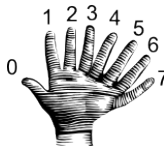
Color Name	HEX	Color
Black	#000000	
Navy	#000080	
DarkBlue	#00008B	
MediumBlue	#0000CD	
Blue	#0000FF	



© www.scratchapixel.com

Class 2: 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

Class 3: design problem (cropping images) & decisions



Class 3: design problem (cropping images) & decisions



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

Class 3: 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.

Class 3: 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.

Class 3: 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')
```

Class 4: 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.")

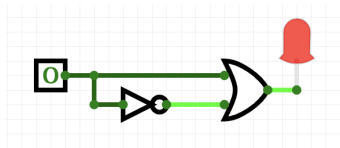
visibility = 0.2
winds = 40
conditions = "blowing snow"
if (winds > 35) and (visibility < 0.25) and \
    (conditions == "blowing snow" or conditions == "heavy snow"):
    print("Blizzard!")
```

Class 4: 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		in2	returns:
False	and	False	False
False	and	True	False
True	and	False	False
True	and	True	True



Class 5: structured data, pandas, & more design

```
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,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,,727,7681
1771,21883,3623,,,2847,28423
1790,33131,45049,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,40203,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419801,45648,37393,33829,1470103
1880,1164673,599495,56559,51980,38991,1911690
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,24372702
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018296,469042,732016,116511,2620048
1930,1867312,2560461,1079129,1565258,159346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738275,1500849,1452177,191555,78921957
1960,1698281,2627319,1809578,1424815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1801325,1168872,352121,7071639
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8006278
2010,1548473,2504790,2230722,1385108,448730,8175123
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

Class 5: 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,,,,,
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
1698,4937,2017,,,727,7681
1771,21883,3623,,,2847,28423
1790,33131,45049,6159,1781,3827,49447
1800,60515,5740,6642,1755,4563,79215
1810,96373,40203,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419801,45468,37393,33829,1470183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1166582,152899,200507,67021,24372702
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018256,469042,732018,116511,5620848
1930,1867312,2580461,1079129,1565258,158346,6930446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738275,1500849,1452177,191555,78931957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7094862
1980,1428285,2230936,1801325,1168872,352121,7071639
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8006278
2010,1494873,2504790,2230722,1385108,448730,81751523
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

Class 5: 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,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,45049,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,40203,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419921,45468,37393,33829,1470193
1880,1164673,599495,56559,51980,38991,1911690
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,2437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018256,469042,732016,116511,5620048
1930,1867312,2580461,1079129,1265258,159346,4590446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738075,1500849,1451277,191555,78991957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1801325,1168872,352121,7071639
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1494873,2504790,2230722,1385108,448735,81751123
2015,1644518,2636735,2339150,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

Class 5: 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,Manhattan,Brooklyn,Queens,Bronx,Staten Island>Total
1698,4937,2017,,,727,7681
1771,21863,3623,,,2847,28423
1790,33131,45049,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,40203,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419801,45468,37393,33829,1470183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,2437202
1910,2331542,1634351,284041,430980,85969,4766883
1920,2284103,2018256,469042,732016,116511,4620048
1930,1867312,2580461,1079129,1265258,158346,4590446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738275,1500849,1452277,291555,78991957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7894862
1980,1428285,2230936,1801325,1168872,352121,7071639
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1332650,443728,8008278
2010,1484873,2504700,2230722,1385108,448730,81751523
2015,1644518,2636735,2339150,1455444,476558,8550405
```

```
pop.plot(x="Year")
plt.show()
```

nycHistPop.csv

In Lab 6

Class 5: structured data, pandas, & more design

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

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

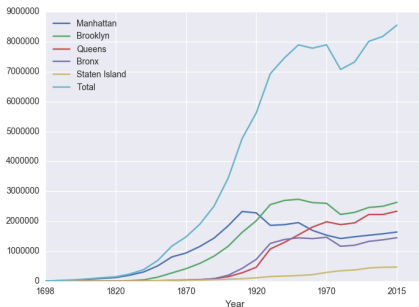
Source: https://en.wikipedia.org/wiki/Demographics_of_New_York_City,
All population figures are consistent with present-day boundaries.
First census after the consolidation of the five boroughs.

```
Year,Manhattan,Brooklyn,Queens,Bronx,Staten Island,Total
1698,4937,2017,,727,7681
1771,21863,3623,,2847,28423
1790,33131,4548,6159,1781,3827,49447
1800,40515,5740,6642,1755,4563,79215
1810,96373,8003,7444,2267,5347,119734
1820,123706,11187,8246,2782,6135,152056
1830,202589,20535,9049,3023,7082,242278
1840,312710,47613,14480,3344,10965,391114
1850,515547,138882,18593,8032,15061,696115
1860,813649,279122,32963,23593,25492,1174779
1870,942292,419921,45468,37393,33829,1470183
1880,1164673,599495,56559,51980,38991,1911698
1890,1441216,838547,87050,88908,51692,2507414
1900,1650093,1146582,152899,200507,67021,3437202
1910,2331542,1634351,284041,430989,85969,4766883
1920,2284103,2018256,469042,732016,116531,5620048
1930,1867312,2560451,1079129,1265598,159346,6906446
1940,1889924,2698285,1297634,1394711,174441,7454995
1950,1940101,2738275,1505049,1452177,291559,7892957
1960,1698281,2627319,1809578,1624815,221993,7781984
1970,1539233,2602012,1986473,1471701,295443,7094862
1980,1428285,2210936,1801325,1168872,352121,7071439
1990,1487536,2300644,1951598,1203789,378977,7322564
2000,1537195,2465326,2229379,1326550,443728,8006278
2010,1484873,2504700,2230722,1385108,468730,8175133
2015,1644518,2636735,2339155,1455444,476558,8550405
```

nycHistPop.csv

In Lab 6

```
pop.plot(x="Year")
plt.show()
```



Class 6: 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()
```

Class 6: 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.

Class 6: 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()`

Class 6: 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:

Class 6: 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")`

Class 6: 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,

Class 6: 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.

Class 7: 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)
```

Class 7: function parameters, github

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

```
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)
```

Class 7: function parameters, github

- 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**.

```
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)
```

Class 7: 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**

Class 7: 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.

Class 7: function parameters, github

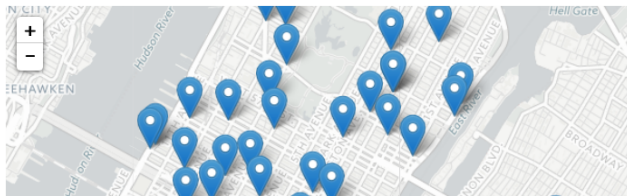
```
def totalWithTax(food,tip):  
    total = 0  
    tax = 0.0875  
    total = food + food * tax  
    total = total + tip  
    return(total)  
  
lunch = float(input('Enter lunch total: '))  
lTip = float(input('Enter lunch tip: '))  
lTotal = totalWithTax(lunch, lTip)  
print('Lunch total is', lTotal)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

Formal Parameters

Actual Parameters

- 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.

Class 8: 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)
```

Class 9: 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)
```

Class 9: 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

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

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

Class 9: 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.

Class 9: 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.`

Class 9: 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: 9 Classes in 10 Minutes



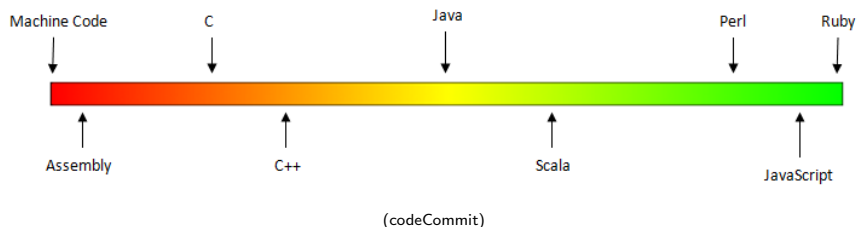
- Input/Output (I/O): `input()` and `print()`;
pandas for CSV files
- Types:
 - ▶ Primitive: `int`, `float`, `bool`, `string`;
 - ▶ Container: lists (but not dictionaries/hashtables or tuples)
- Objects: turtles (used but did not design our own)
- Loops: definite & indefinite
- Conditionals: `if-elif-else`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

Today's Topics



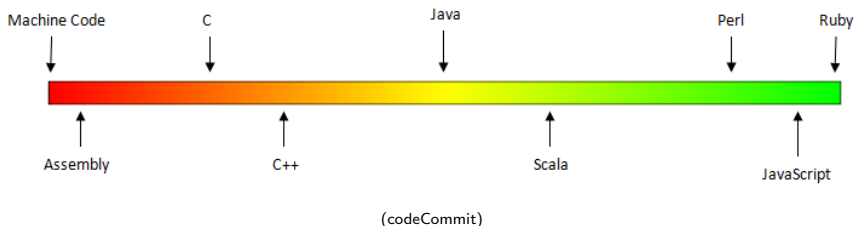
- Design Patterns: Searching
- Python Recap
- **Machine Language**
- Machine Language: Jumps & Loops
- Binary & Hex Arithmetic
- Final Exam: Format

Low-Level vs. High-Level Languages



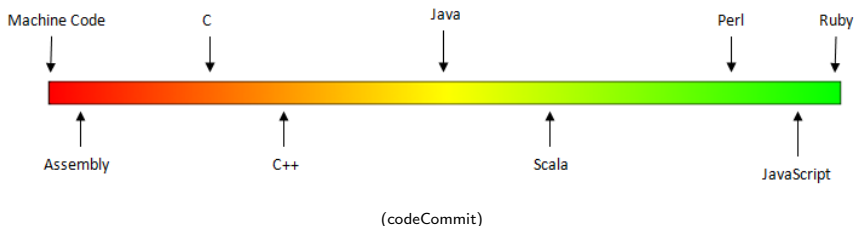
- Can view programming languages on a continuum.

Low-Level vs. High-Level Languages



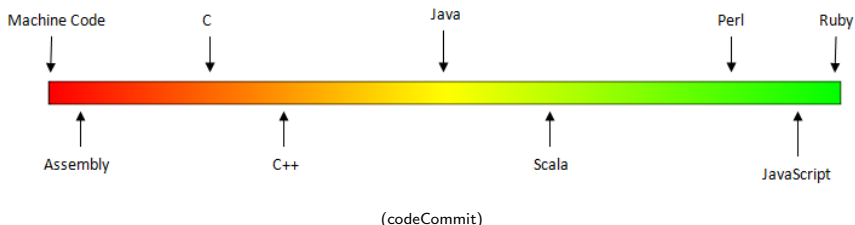
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages**

Low-Level vs. High-Level Languages



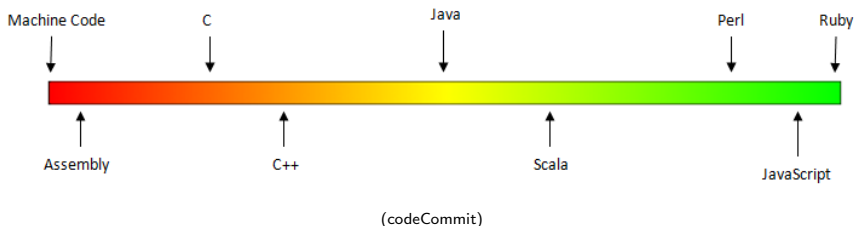
- Can view programming languages on a continuum.
- Those that directly access machine instructions & memory and have little abstraction are **low-level languages** (e.g. machine language, assembly language).

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.

Processing



Dies ist ein Blindtext. An ihm lässt sich vieles über die Schrift ablesen, in der er gesetzt ist. Auf den ersten Blick wird der Grauwert der Schriftfläche sichtbar. Dann kann man prüfen, wie gut die Schrift zu lesen ist und wie sie auf den Leser wirkt. Dies ist ein Blindtext. An ihm lässt sich

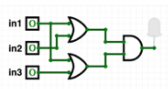


**Data
&
Instructions**

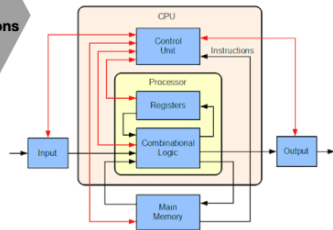
```
def totalWithTax(food,tip):
    total = 0
    tax = 0.0875
    total = food + food * tax
    total = total + tip
    return(total)
```



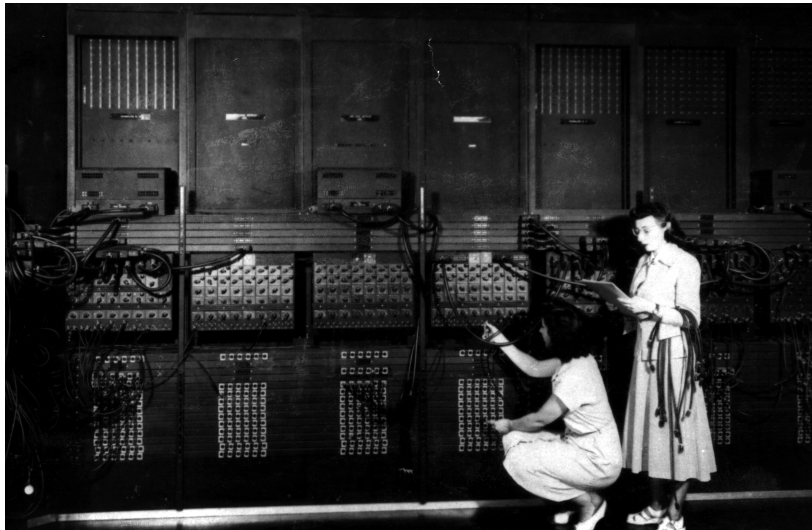
**Data
&
Instructions**



Circuits (switches)
On/Off 1/0 Logic
Billions of switches/bits



Machine Language



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

Machine Language

```
1 FOX 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 NUmxDI2C .A .X .Y SP DP DB
; 00 E012 00110000 0000 0000 0002 CFFF 0000 00
g 2000

BREAK

PB PC NUmxDI2C .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 00:UU.....
█
```

(wiki)

Machine Language

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



The screenshot shows a MIPS assembly program in a debugger. The program consists of several instructions: `li $t0, 0x1234`, `li $t1, 0x5678`, `add $t2, $t0, $t1`, `sw $t2, 0($t0)`, `lw $t3, 0($t0)`, `add $t4, $t3, $t2`, `sw $t4, 0($t1)`, `lw $t5, 0($t1)`, `add $t6, $t5, $t4`, `sw $t6, 0($t2)`, `lw $t7, 0($t2)`, `add $t8, $t7, $t6`, `sw $t8, 0($t3)`, `lw $t9, 0($t3)`, `add $t10, $t9, $t8`, `sw $t10, 0($t4)`, `lw $t11, 0($t4)`, `add $t12, $t11, $t10`, `sw $t12, 0($t5)`, `lw $t13, 0($t5)`, `add $t14, $t13, $t12`, `sw $t14, 0($t6)`, `lw $t15, 0($t6)`, `add $t16, $t15, $t14`, `sw $t16, 0($t7)`, `lw $t17, 0($t7)`, `add $t18, $t17, $t16`, `sw $t18, 0($t8)`, `lw $t19, 0($t8)`, `add $t20, $t19, $t18`, `sw $t20, 0($t9)`, `lw $t21, 0($t9)`, `add $t22, $t21, $t20`, `sw $t22, 0($t10)`, `lw $t23, 0($t10)`, `add $t24, $t23, $t22`, `sw $t24, 0($t11)`, `lw $t25, 0($t11)`, `add $t26, $t25, $t24`, `sw $t26, 0($t12)`, `lw $t27, 0($t12)`, `add $t28, $t27, $t26`, `sw $t28, 0($t13)`, `lw $t29, 0($t13)`, `add $t30, $t29, $t28`, `sw $t30, 0($t14)`, `lw $t31, 0($t14)`, `add $t32, $t31, $t30`, `sw $t32, 0($t15)`, `lw $t33, 0($t15)`, `add $t34, $t33, $t32`, `sw $t34, 0($t16)`, `lw $t35, 0($t16)`, `add $t36, $t35, $t34`, `sw $t36, 0($t17)`, `lw $t37, 0($t17)`, `add $t38, $t37, $t36`, `sw $t38, 0($t18)`, `lw $t39, 0($t18)`, `add $t40, $t39, $t38`, `sw $t40, 0($t19)`, `lw $t41, 0($t19)`, `add $t42, $t41, $t40`, `sw $t42, 0($t20)`, `lw $t43, 0($t20)`, `add $t44, $t43, $t42`, `sw $t44, 0($t21)`, `lw $t45, 0($t21)`, `add $t46, $t45, $t44`, `sw $t46, 0($t22)`, `lw $t47, 0($t22)`, `add $t48, $t47, $t46`, `sw $t48, 0($t23)`, `lw $t49, 0($t23)`, `add $t50, $t49, $t48`, `sw $t50, 0($t24)`, `lw $t51, 0($t24)`, `add $t52, $t51, $t50`, `sw $t52, 0($t25)`, `lw $t53, 0($t25)`, `add $t54, $t53, $t52`, `sw $t54, 0($t26)`, `lw $t55, 0($t26)`, `add $t56, $t55, $t54`, `sw $t56, 0($t27)`, `lw $t57, 0($t27)`, `add $t58, $t57, $t56`, `sw $t58, 0($t28)`, `lw $t59, 0($t28)`, `add $t60, $t59, $t58`, `sw $t60, 0($t29)`, `lw $t61, 0($t29)`, `add $t62, $t61, $t60`, `sw $t62, 0($t30)`, `lw $t63, 0($t30)`, `add $t64, $t63, $t62`, `sw $t64, 0($t31)`, `lw $t65, 0($t31)`, `add $t66, $t65, $t64`, `sw $t66, 0($t32)`, `lw $t67, 0($t32)`, `add $t68, $t67, $t66`, `sw $t68, 0($t33)`, `lw $t69, 0($t33)`, `add $t70, $t69, $t68`, `sw $t70, 0($t34)`, `lw $t71, 0($t34)`, `add $t72, $t71, $t70`, `sw $t72, 0($t35)`, `lw $t73, 0($t35)`, `add $t74, $t73, $t72`, `sw $t74, 0($t36)`, `lw $t75, 0($t36)`, `add $t76, $t75, $t74`, `sw $t76, 0($t37)`, `lw $t77, 0($t37)`, `add $t78, $t77, $t76`, `sw $t78, 0($t38)`, `lw $t79, 0($t38)`, `add $t80, $t79, $t78`, `sw $t80, 0($t39)`, `lw $t81, 0($t39)`, `add $t82, $t81, $t80`, `sw $t82, 0($t40)`, `lw $t83, 0($t40)`, `add $t84, $t83, $t82`, `sw $t84, 0($t41)`, `lw $t85, 0($t41)`, `add $t86, $t85, $t84`, `sw $t86, 0($t42)`, `lw $t87, 0($t42)`, `add $t88, $t87, $t86`, `sw $t88, 0($t43)`, `lw $t89, 0($t43)`, `add $t90, $t89, $t88`, `sw $t90, 0($t44)`, `lw $t91, 0($t44)`, `add $t92, $t91, $t90`, `sw $t92, 0($t45)`, `lw $t93, 0($t45)`, `add $t94, $t93, $t92`, `sw $t94, 0($t46)`, `lw $t95, 0($t46)`, `add $t96, $t95, $t94`, `sw $t96, 0($t47)`, `lw $t97, 0($t47)`, `add $t98, $t97, $t96`, `sw $t98, 0($t48)`, `lw $t99, 0($t48)`, `add $t100, $t99, $t98`, `sw $t100, 0($t49)`, `lw $t101, 0($t49)`, `add $t102, $t101, $t100`, `sw $t102, 0($t50)`, `lw $t103, 0($t50)`, `add $t104, $t103, $t102`, `sw $t104, 0($t51)`, `lw $t105, 0($t51)`, `add $t106, $t105, $t104`, `sw $t106, 0($t52)`, `lw $t107, 0($t52)`, `add $t108, $t107, $t106`, `sw $t108, 0($t53)`, `lw $t109, 0($t53)`, `add $t110, $t109, $t108`, `sw $t110, 0($t54)`, `lw $t111, 0($t54)`, `add $t112, $t111, $t110`, `sw $t112, 0($t55)`, `lw $t113, 0($t55)`, `add $t114, $t113, $t112`, `sw $t114, 0($t56)`, `lw $t115, 0($t56)`, `add $t116, $t115, $t114`, `sw $t116, 0($t57)`, `lw $t117, 0($t57)`, `add $t118, $t117, $t116`, `sw $t118, 0($t58)`, `lw $t119, 0($t58)`, `add $t120, $t119, $t118`, `sw $t120, 0($t59)`, `lw $t121, 0($t59)`, `add $t122, $t121, $t120`, `sw $t122, 0($t60)`, `lw $t123, 0($t60)`, `add $t124, $t123, $t122`, `sw $t124, 0($t61)`, `lw $t125, 0($t61)`, `add $t126, $t125, $t124`, `sw $t126, 0($t62)`, `lw $t127, 0($t62)`, `add $t128, $t127, $t126`, `sw $t128, 0($t63)`, `lw $t129, 0($t63)`, `add $t130, $t129, $t128`, `sw $t130, 0($t64)`, `lw $t131, 0($t64)`, `add $t132, $t131, $t130`, `sw $t132, 0($t65)`, `lw $t133, 0($t65)`, `add $t134, $t133, $t132`, `sw $t134, 0($t66)`, `lw $t135, 0($t66)`, `add $t136, $t135, $t134`, `sw $t136, 0($t67)`, `lw $t137, 0($t67)`, `add $t138, $t137, $t136`, `sw $t138, 0($t68)`, `lw $t139, 0($t68)`, `add $t140, $t139, $t138`, `sw $t140, 0($t69)`, `lw $t141, 0($t69)`, `add $t142, $t141, $t140`, `sw $t142, 0($t70)`, `lw $t143, 0($t70)`, `add $t144, $t143, $t142`, `sw $t144, 0($t71)`, `lw $t145, 0($t71)`, `add $t146, $t145, $t144`, `sw $t146, 0($t72)`, `lw $t147, 0($t72)`, `add $t148, $t147, $t146`, `sw $t148, 0($t73)`, `lw $t149, 0($t73)`, `add $t150, $t149, $t148`, `sw $t150, 0($t74)`, `lw $t151, 0($t74)`, `add $t152, $t151, $t150`, `sw $t152, 0($t75)`, `lw $t153, 0($t75)`, `add $t154, $t153, $t152`, `sw $t154, 0($t76)`, `lw $t155, 0($t76)`, `add $t156, $t155, $t154`, `sw $t156, 0($t77)`, `lw $t157, 0($t77)`, `add $t158, $t157, $t156`, `sw $t158, 0($t78)`, `lw $t159, 0($t78)`, `add $t160, $t159, $t158`, `sw $t160, 0($t79)`, `lw $t161, 0($t79)`, `add $t162, $t161, $t160`, `sw $t162, 0($t80)`, `lw $t163, 0($t80)`, `add $t164, $t163, $t162`, `sw $t164, 0($t81)`, `lw $t165, 0($t81)`, `add $t166, $t165, $t164`, `sw $t166, 0($t82)`, `lw $t167, 0($t82)`, `add $t168, $t167, $t166`, `sw $t168, 0($t83)`, `lw $t169, 0($t83)`, `add $t170, $t169, $t168`, `sw $t170, 0($t84)`, `lw $t171, 0($t84)`, `add $t172, $t171, $t170`, `sw $t172, 0($t85)`, `lw $t173, 0($t85)`, `add $t174, $t173, $t172`, `sw $t174, 0($t86)`, `lw $t175, 0($t86)`, `add $t176, $t175, $t174`, `sw $t176, 0($t87)`, `lw $t177, 0($t87)`, `add $t178, $t177, $t176`, `sw $t178, 0($t88)`, `lw $t179, 0($t88)`, `add $t180, $t179, $t178`, `sw $t180, 0($t89)`, `lw $t181, 0($t89)`, `add $t182, $t181, $t180`, `sw $t182, 0($t90)`, `lw $t183, 0($t90)`, `add $t184, $t183, $t182`, `sw $t184, 0($t91)`, `lw $t185, 0($t91)`, `add $t186, $t185, $t184`, `sw $t186, 0($t92)`, `lw $t187, 0($t92)`, `add $t188, $t187, $t186`, `sw $t188, 0($t93)`, `lw $t189, 0($t93)`, `add $t190, $t189, $t188`, `sw $t190, 0($t94)`, `lw $t191, 0($t94)`, `add $t192, $t191, $t190`, `sw $t192, 0($t95)`, `lw $t193, 0($t95)`, `add $t194, $t193, $t192`, `sw $t194, 0($t96)`, `lw $t195, 0($t96)`, `add $t196, $t195, $t194`, `sw $t196, 0($t97)`, `lw $t197, 0($t97)`, `add $t198, $t197, $t196`, `sw $t198, 0($t98)`, `lw $t199, 0($t98)`, `add $t200, $t199, $t198`, `sw $t200, 0($t99)`, `lw $t201, 0($t99)`, `add $t202, $t201, $t200`, `sw $t202, 0($t100)`, `lw $t203, 0($t100)`, `add $t204, $t203, $t202`, `sw $t204, 0($t101)`, `lw $t205, 0($t101)`, `add $t206, $t205, $t204`, `sw $t206, 0($t102)`, `lw $t207, 0($t102)`, `add $t208, $t207, $t206`, `sw $t208, 0($t103)`, `lw $t209, 0($t103)`, `add $t210, $t209, $t208`, `sw $t210, 0($t104)`, `lw $t211, 0($t104)`, `add $t212, $t211, $t210`, `sw $t212, 0($t105)`, `lw $t213, 0($t105)`, `add $t214, $t213, $t212`, `sw $t214, 0($t106)`, `lw $t215, 0($t106)`, `add $t216, $t215, $t214`, `sw $t216, 0($t107)`, `lw $t217, 0($t107)`, `add $t218, $t217, $t216`, `sw $t218, 0($t108)`, `lw $t219, 0($t108)`, `add $t220, $t219, $t218`, `sw $t220, 0($t109)`, `lw $t221, 0($t109)`, `add $t222, $t221, $t220`, `sw $t222, 0($t110)`, `lw $t223, 0($t110)`, `add $t224, $t223, $t222`, `sw $t224, 0($t111)`, `lw $t225, 0($t111)`, `add $t226, $t225, $t224`, `sw $t226, 0($t112)`, `lw $t227, 0($t112)`, `add $t228, $t227, $t226`, `sw $t228, 0($t113)`, `lw $t229, 0($t113)`, `add $t230, $t229, $t228`, `sw $t230, 0($t114)`, `lw $t231, 0($t114)`, `add $t232, $t231, $t230`, `sw $t232, 0($t115)`, `lw $t233, 0($t115)`, `add $t234, $t233, $t232`, `sw $t234, 0($t116)`, `lw $t235, 0($t116)`, `add $t236, $t235, $t234`, `sw $t236, 0($t117)`, `lw $t237, 0($t117)`, `add $t238, $t237, $t236`, `sw $t238, 0($t118)`, `lw $t239, 0($t118)`, `add $t240, $t239, $t238`, `sw $t240, 0($t119)`, `lw $t241, 0($t119)`, `add $t242, $t241, $t240`, `sw $t242, 0($t120)`, `lw $t243, 0($t120)`, `add $t244, $t243, $t242`, `sw $t244, 0($t121)`, `lw $t245, 0($t121)`, `add $t246, $t245, $t244`, `sw $t246, 0($t122)`, `lw $t247, 0($t122)`, `add $t248, $t247, $t246`, `sw $t248, 0($t123)`, `lw $t249, 0($t123)`, `add $t250, $t249, $t248`, `sw $t250, 0($t124)`, `lw $t251, 0($t124)`, `add $t252, $t251, $t250`, `sw $t252, 0($t125)`, `lw $t253, 0($t125)`, `add $t254, $t253, $t252`, `sw $t254, 0($t126)`, `lw $t255, 0($t126)`, `add $t256, $t255, $t254`, `sw $t256, 0($t127)`, `lw $t257, 0($t127)`, `add $t258, $t257, $t256`, `sw $t258, 0($t128)`, `lw $t259, 0($t128)`, `add $t260, $t259, $t258`, `sw $t260, 0($t129)`, `lw $t261, 0($t129)`, `add $t262, $t261, $t260`, `sw $t262, 0($t130)`, `lw $t263, 0($t130)`, `add $t264, $t263, $t262`, `sw $t264, 0($t131)`, `lw $t265, 0($t131)`, `add $t266, $t265, $t264`, `sw $t266, 0($t132)`, `lw $t267, 0($t132)`, `add $t268, $t267, $t266`, `sw $t268, 0($t133)`, `lw $t269, 0($t133)`, `add $t270, $t269, $t268`, `sw $t270, 0($t134)`, `lw $t271, 0($t134)`, `add $t272, $t271, $t270`, `sw $t272, 0($t135)`, `lw $t273, 0($t135)`, `add $t274, $t273, $t272`, `sw $t274, 0($t136)`, `lw $t275, 0($t136)`, `add $t276, $t275, $t274`, `sw $t276, 0($t137)`, `lw $t277, 0($t137)`, `add $t278, $t277, $t276`, `sw $t278, 0($t138)`, `lw $t279, 0($t138)`, `add $t280, $t279, $t278`, `sw $t280, 0($t139)`, `lw $t281, 0($t139)`, `add $t282, $t281, $t280`, `sw $t282, 0($t140)`, `lw $t283, 0($t140)`, `add $t284, $t283, $t282`, `sw $t284, 0($t141)`, `lw $t285, 0($t141)`, `add $t286, $t285, $t284`, `sw $t286, 0($t142)`, `lw $t287, 0($t142)`, `add $t288, $t287, $t286`, `sw $t288, 0($t143)`, `lw $t289, 0($t143)`, `add $t290, $t289, $t288`, `sw $t290, 0($t144)`, `lw $t291, 0($t144)`, `add $t292, $t291, $t290`, `sw $t292, 0($t145)`, `lw $t293, 0($t145)`, `add $t294, $t293, $t292`, `sw $t294, 0($t146)`, `lw $t295, 0($t146)`, `add $t296, $t295, $t294`, `sw $t296, 0($t147)`, `lw $t297, 0($t147)`, `add $t298, $t297, $t296`, `sw $t298, 0($t148)`, `lw $t299, 0($t148)`, `add $t300, $t299, $t298`, `sw $t300, 0($t149)`, `lw $t301, 0($t149)`, `add $t302, $t301, $t300`, `sw $t302, 0($t150)`, `lw $t303, 0($t150)`, `add $t304, $t303, $t302`, `sw $t304, 0($t151)`, `lw $t305, 0($t151)`, `add $t306, $t305, $t304`, `sw $t306, 0($t152)`, `lw $t307, 0($t152)`, `add $t308, $t307, $t306`, `sw $t308, 0($t153)`, `lw $t309, 0($t153)`, `add $t310, $t309, $t308`, `sw $t310, 0($t154)`, `lw $t311, 0($t154)`, `add $t312, $t311, $t310`, `sw $t312, 0($t155)`, `lw $t313, 0($t155)`, `add $t314, $t313, $t312`, `sw $t314, 0($t156)`, `lw $t315, 0($t156)`, `add $t316, $t315, $t314`, `sw $t316, 0($t157)`, `lw $t317, 0($t157)`, `add $t318, $t317, $t316`, `sw $t318, 0($t158)`, `lw $t319, 0($t158)`, `add $t320, $t319, $t318`, `sw $t320, 0($t159)`, `lw $t321, 0($t159)`, `add $t322, $t321, $t320`, `sw $t322, 0($t160)`, `lw $t323, 0($t160)`, `add $t324, $t323, $t322`, `sw $t324, 0($t161)`, `lw $t325, 0($t161)`, `add $t326, $t325, $t324`, `sw $t326, 0($t162)`, `lw $t327, 0($t162)`, `add $t328, $t327, $t326`, `sw $t328, 0($t163)`, `lw $t329, 0($t163)`, `add $t330, $t329, $t328`, `sw $t330, 0($t164)`, `lw $t331, 0($t164)`, `add $t332, $t331, $t330`, `sw $t332, 0($t165)`, `lw $t333, 0($t165)`, `add $t334, $t333, $t332`, `sw $t334, 0($t166)`, `lw $t335, 0($t166)`, `add $t336, $t335, $t334`, `sw $t336, 0($t167)`, `lw $t337, 0($t167)`, `add $t338, $t337, $t336`, `sw $t338, 0($t168)`, `lw $t339, 0($t168)`, `add $t340, $t339, $t338`, `sw $t340, 0($t169)`, `lw $t341, 0($t169)`, `add $t342, $t341, $t340`, `sw $t342, 0($t170)`, `lw $t343, 0($t170)`, `add $t344, $t343, $t342`, `sw $t344, 0($t171)`, `lw $t345, 0($t171)`, `add $t346, $t345, $t344`, `sw $t346, 0($t172)`, `lw $t347, 0($t172)`, `add $t348, $t347, $t346`, `sw $t348, 0($t173)`, `lw $t349, 0($t173)`, `add $t350, $t349, $t348`, `sw $t350, 0($t174)`, `lw $t351, 0($t174)`, `add $t352, $t351, $t350`, `sw $t352, 0($t175)`, `lw $t353, 0($t175)`, `add $t354, $t353, $t352`, `sw $t354, 0($t176)`, `lw $t355, 0($t176)`, `add $t356, $t355, $t354`, `sw $t356, 0($t177)`, `lw $t357, 0($t177)`, `add $t358, $t357, $t356`, `sw $t358, 0($t178)`, `lw $t359, 0($t178)`, `add $t360, $t359, $t358`, `sw $t360, 0($t179)`, `lw $t361, 0($t179)`, `add $t362, $t361, $t360`, `sw $t362, 0($t180)`, `lw $t363, 0($t180)`, `add $t364, $t363, $t362`, `sw $t364, 0($t181)`, `lw $t365, 0($t181)`, `add $t366, $t365, $t364`, `sw $t366, 0($t182)`, `lw $t367, 0($t182)`, `add $t368, $t367, $t366`, `sw $t368, 0($t183)`, `lw $t369, 0($t183)`, `add $t370, $t369, $t368`, `sw $t370, 0($t184)`, `lw $t371, 0($t184)`, `add $t372, $t371, $t370`, `sw $t372, 0($t185)`, `lw $t373, 0($t185)`, `add $t374, $t373, $t372`, `sw $t374, 0($t186)`, `lw $t375, 0($t186)`, `add $t376, $t375, $t374`, `sw $t376, 0($t187)`, `lw $t377, 0($t187)`, `add $t378, $t377, $t376`, `sw $t378, 0($t188)`, `lw $t379, 0($t188)`, `add $t380, $t379, $t378`, `sw $t380, 0($t189)`, `lw $t381, 0($t189)`, `add $t382, $t381, $t380`, `sw $t382, 0($t190)`, `lw $t383, 0($t190)`, `add $t384, $t383, $t382`, `sw $t384, 0($t191)`, `lw $t385, 0($t191)`, `add $t386, $t385, $t384`, `sw $t386, 0($t192)`, `lw $t387, 0($t192)`, `add $t388, $t387, $t386`, `sw $t388, 0($t193)`, `lw $t389, 0($t193)`, `add $t390, $t389, $t388`, `sw $t390, 0($t194)`, `lw $t391, 0($t194)`, `add $t392, $t391, $t390`, `sw $t392, 0($t195)`, `lw $t393, 0($t195)`, `add $t394, $t393, $t392`, `sw $t394, 0($t196)`, `lw $t395, 0($t196)`, `add $t396, $t395, $t394`, `sw $t396, 0($t197)`, `lw $t397, 0($t197`

Machine Language

```
002000 c2 30      REP #K30
002002 10        CLC
002003 F0        SED
002004 40 34 12    LSH #01234
002007 60 21 43    RSC #04321
00200A 0F 03 7F 01 STN #017F03
00200C 00        CLJ
00200F E2 30      SEP #K30
002011 00        BRX
002012

P  PC  Mem32C  A  X  Y  SP  BP  B0
: 00 E012 00110000 0000 0000 0002 C7FF 0000 00
$ 2000

BREAK

P  PC  Mem32C  A  X  Y  SP  BP  B0
: 00 2013 00110000 5555 0000 0002 C7FF 0000 00
n 1103 7403
007F03 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.

Machine Language



The screenshot shows a MIPS assembly program being executed in a debugger. The assembly code is as follows:

```
002000 c2 30      REP #30
002002 10          CLC
002003 f0          SED
002004 40 34 12    L32 $t0, 001224
002007 60 21 43    RLC $t0, 004321
00200a 0f 03 7f 01  ST4 $t0, 0017f03
00200c 00          CLJ
00200f e2 30      SEP #30
002011 00          BRK
002012
```

Below the assembly code, the debugger shows the current state of the processor:

```
PC: 002012 00110000 0000 0000 0002 c7ff 0000 00
$2000
BREAK
PC: 002013 00110000 5555 0000 0002 c7ff 0000 00
$2003 7f03
0017f03 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

[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
32 syscall # print to the log
```

Step	Run	<input checked="" type="checkbox"/> 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

Line: 3 dis

Show/Hide Demos

User Guide | Unit Tests | Docs

Addition Doubler Stop Looper Stack Test Hello World

Code Gen Save String Interactive Binary2 Decimal Decimal2 Binary

Debug

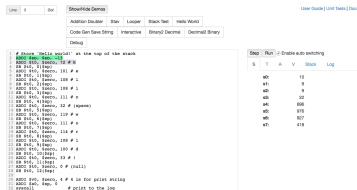
```
1 # Store "hello world!" at the top of the stack
2 ADDUI $a0, $zero, 32 # 8
3 ROR $t0, $t0, 1($a0)
4 ADDUI $t0, $zero, 101 # e
5 DD $t0, 1($a0)
6 ADDUI $t0, $zero, 108 # l
7 DD $t0, 1($a0)
8 ADDUI $t0, $zero, 108 # l
9 DD $t0, 1($a0)
10 ADDUI $t0, $zero, 111 # o
11 DD $t0, 1($a0)
12 ADDUI $t0, $zero, 32 # (space)
13 DD $t0, 1($a0)
14 ADDUI $t0, $zero, 119 # w
15 DD $t0, 1($a0)
16 ADDUI $t0, $zero, 114 # u
17 DD $t0, 1($a0)
18 ADDUI $t0, $zero, 108 # d
19 DD $t0, 1($a0)
20 ADDUI $t0, $zero, 108 # d
21 DD $t0, 1($a0)
22 ADDUI $t0, $zero, 103 # d
23 DD $t0, 1($a0)
24 ADDUI $t0, $zero, 33 # !
25 DD $t0, 1($a0)
26 ADDUI $t0, $zero, 0 # (null)
27 DD $t0, 1($a0)
28 ADDUI $t0, $zero, 6 # 4 in for print string
29 ADDUI $a0, $a0, 0
30 syscall # print to the log
```

Step Run v Enable auto-switching

S	T	A	V	Stack	Log
a0:				10	
t0:				9	
a0:				9	
a0:				22	
a0:				695	
a0:				970	
a0:				927	
a0:				418	

(Demo with WeMIPS)

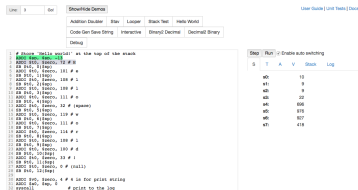
MIPS Commands



```
1 # Please "hello world" as the sign of the success
2 ADDI $0, $zero, 12 # 12
3 SW $0, 0($0)
4 ADDI $0, $zero, 191 # 191
5 SW $0, 100($0)
6 ADDI $0, $zero, 100 # 100
7 SW $0, 200($0)
8 ADDI $0, $zero, 100 # 100
9 SW $0, 300($0)
10 ADDI $0, $zero, 111 # 111
11 SW $0, 400($0)
12 ADDI $0, $zero, 12 # 12 (again)
13 SW $0, 500($0)
14 ADDI $0, $zero, 119 # 119
15 SW $0, 600($0)
16 ADDI $0, $zero, 111 # 111
17 SW $0, 700($0)
18 ADDI $0, $zero, 114 # 114
19 SW $0, 800($0)
20 ADDI $0, $zero, 100 # 100
21 SW $0, 900($0)
22 ADDI $0, $zero, 100 # 100
23 SW $0, 1000($0)
24 ADDI $0, $zero, 10 # 10
25 SW $0, 1100($0)
26 ADDI $0, $zero, 0 # (null)
27 SW $0, 1200($0)
28 ADDI $0, $zero, 4 # 4 is for print string
29 ADDI $0, $zero, 4
30 syscall
31 # print to the log
```

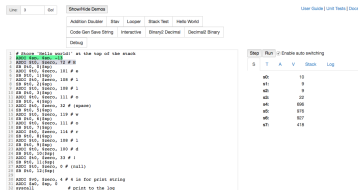
- **Registers:** locations for storing information that can be quickly accessed.

MIPS Commands



```
1 # Store "hello world!" in the top of the stack
2 ADDI $s0, $zero, 12 # 12
3 SW $s0, 0($zero)
4 ADDI $s1, $zero, 191 # 191
5 SW $s1, 16($zero)
6 ADDI $t0, $zero, 100 # 100
7 SW $t0, 32($zero)
8 ADDI $t1, $zero, 111 # 111
9 SW $t1, 48($zero)
10 ADDI $s0, $zero, 12 # (address)
11 SW $s0, 64($zero)
12 ADDI $t0, $zero, 111 # 111
13 SW $t0, 80($zero)
14 ADDI $t1, $zero, 114 # 114
15 SW $t1, 96($zero)
16 ADDI $s0, $zero, 100 # 100
17 SW $s0, 112($zero)
18 ADDI $t0, $zero, 100 # 100
19 SW $t0, 128($zero)
20 ADDI $t1, $zero, 111 # 111
21 SW $t1, 144($zero)
22 ADDI $s0, $zero, 114 # 114
23 SW $s0, 160($zero)
24 ADDI $t0, $zero, 111 # 111
25 SW $t0, 176($zero)
26 ADDI $t1, $zero, 114 # 114
27 SW $t1, 192($zero)
28 ADDI $s0, $zero, 111 # 111
29 SW $s0, 208($zero)
30 ADDI $t0, $zero, 114 # 114
31 SW $t0, 224($zero)
32 ADDI $t1, $zero, 111 # 111
33 SW $t1, 240($zero)
34 ADDI $s0, $zero, 114 # 114
35 SW $s0, 256($zero)
36 ADDI $t0, $zero, 111 # 111
37 SW $t0, 272($zero)
38 ADDI $t1, $zero, 114 # 114
39 SW $t1, 288($zero)
40 ADDI $s0, $zero, 111 # 111
41 SW $s0, 304($zero)
42 ADDI $t0, $zero, 114 # 114
43 SW $t0, 320($zero)
44 ADDI $t1, $zero, 111 # 111
45 SW $t1, 336($zero)
46 ADDI $s0, $zero, 114 # 114
47 SW $s0, 352($zero)
48 ADDI $t0, $zero, 111 # 111
49 SW $t0, 368($zero)
50 ADDI $t1, $zero, 114 # 114
51 SW $t1, 384($zero)
52 ADDI $s0, $zero, 111 # 111
53 SW $s0, 400($zero)
54 ADDI $t0, $zero, 114 # 114
55 SW $t0, 416($zero)
56 ADDI $t1, $zero, 111 # 111
57 SW $t1, 432($zero)
58 ADDI $s0, $zero, 114 # 114
59 SW $s0, 448($zero)
60 ADDI $t0, $zero, 111 # 111
61 SW $t0, 464($zero)
62 ADDI $t1, $zero, 114 # 114
63 SW $t1, 480($zero)
64 ADDI $s0, $zero, 111 # 111
65 SW $s0, 496($zero)
66 ADDI $t0, $zero, 114 # 114
67 SW $t0, 512($zero)
68 ADDI $t1, $zero, 111 # 111
69 SW $t1, 528($zero)
70 ADDI $s0, $zero, 114 # 114
71 SW $s0, 544($zero)
72 ADDI $t0, $zero, 111 # 111
73 SW $t0, 560($zero)
74 ADDI $t1, $zero, 114 # 114
75 SW $t1, 576($zero)
76 ADDI $s0, $zero, 111 # 111
77 SW $s0, 592($zero)
78 ADDI $t0, $zero, 114 # 114
79 SW $t0, 608($zero)
80 ADDI $t1, $zero, 111 # 111
81 SW $t1, 624($zero)
82 ADDI $s0, $zero, 114 # 114
83 SW $s0, 640($zero)
84 ADDI $t0, $zero, 111 # 111
85 SW $t0, 656($zero)
86 ADDI $t1, $zero, 114 # 114
87 SW $t1, 672($zero)
88 ADDI $s0, $zero, 111 # 111
89 SW $s0, 688($zero)
90 ADDI $t0, $zero, 114 # 114
91 SW $t0, 704($zero)
92 ADDI $t1, $zero, 111 # 111
93 SW $t1, 720($zero)
94 ADDI $s0, $zero, 114 # 114
95 SW $s0, 736($zero)
96 ADDI $t0, $zero, 111 # 111
97 SW $t0, 752($zero)
98 ADDI $t1, $zero, 114 # 114
99 SW $t1, 768($zero)
100 ADDI $s0, $zero, 111 # 111
101 SW $s0, 784($zero)
102 ADDI $t0, $zero, 114 # 114
103 SW $t0, 800($zero)
104 ADDI $t1, $zero, 111 # 111
105 SW $t1, 816($zero)
106 ADDI $s0, $zero, 114 # 114
107 SW $s0, 832($zero)
108 ADDI $t0, $zero, 111 # 111
109 SW $t0, 848($zero)
110 ADDI $t1, $zero, 114 # 114
111 SW $t1, 864($zero)
112 ADDI $s0, $zero, 111 # 111
113 SW $s0, 880($zero)
114 ADDI $t0, $zero, 114 # 114
115 SW $t0, 896($zero)
116 ADDI $t1, $zero, 111 # 111
117 SW $t1, 912($zero)
118 ADDI $s0, $zero, 114 # 114
119 SW $s0, 928($zero)
120 ADDI $t0, $zero, 111 # 111
121 SW $t0, 944($zero)
122 ADDI $t1, $zero, 114 # 114
123 SW $t1, 960($zero)
124 ADDI $s0, $zero, 111 # 111
125 SW $s0, 976($zero)
126 ADDI $t0, $zero, 114 # 114
127 SW $t0, 992($zero)
128 ADDI $t1, $zero, 111 # 111
129 SW $t1, 1008($zero)
130 ADDI $s0, $zero, 114 # 114
131 SW $s0, 1024($zero)
132 ADDI $t0, $zero, 111 # 111
133 SW $t0, 1040($zero)
134 ADDI $t1, $zero, 114 # 114
135 SW $t1, 1056($zero)
136 ADDI $s0, $zero, 111 # 111
137 SW $s0, 1072($zero)
138 ADDI $t0, $zero, 114 # 114
139 SW $t0, 1088($zero)
140 ADDI $t1, $zero, 111 # 111
141 SW $t1, 1104($zero)
142 ADDI $s0, $zero, 114 # 114
143 SW $s0, 1120($zero)
144 ADDI $t0, $zero, 111 # 111
145 SW $t0, 1136($zero)
146 ADDI $t1, $zero, 114 # 114
147 SW $t1, 1152($zero)
148 ADDI $s0, $zero, 111 # 111
149 SW $s0, 1168($zero)
150 ADDI $t0, $zero, 114 # 114
151 SW $t0, 1184($zero)
152 ADDI $t1, $zero, 111 # 111
153 SW $t1, 1200($zero)
154 ADDI $s0, $zero, 114 # 114
155 SW $s0, 1216($zero)
156 ADDI $t0, $zero, 111 # 111
157 SW $t0, 1232($zero)
158 ADDI $t1, $zero, 114 # 114
159 SW $t1, 1248($zero)
160 ADDI $s0, $zero, 111 # 111
161 SW $s0, 1264($zero)
162 ADDI $t0, $zero, 114 # 114
163 SW $t0, 1280($zero)
164 ADDI $t1, $zero, 111 # 111
165 SW $t1, 1296($zero)
166 ADDI $s0, $zero, 114 # 114
167 SW $s0, 1312($zero)
168 ADDI $t0, $zero, 111 # 111
169 SW $t0, 1328($zero)
170 ADDI $t1, $zero, 114 # 114
171 SW $t1, 1344($zero)
172 ADDI $s0, $zero, 111 # 111
173 SW $s0, 1360($zero)
174 ADDI $t0, $zero, 114 # 114
175 SW $t0, 1376($zero)
176 ADDI $t1, $zero, 111 # 111
177 SW $t1, 1392($zero)
178 ADDI $s0, $zero, 114 # 114
179 SW $s0, 1408($zero)
180 ADDI $t0, $zero, 111 # 111
181 SW $t0, 1424($zero)
182 ADDI $t1, $zero, 114 # 114
183 SW $t1, 1440($zero)
184 ADDI $s0, $zero, 111 # 111
185 SW $s0, 1456($zero)
186 ADDI $t0, $zero, 114 # 114
187 SW $t0, 1472($zero)
188 ADDI $t1, $zero, 111 # 111
189 SW $t1, 1488($zero)
190 ADDI $s0, $zero, 114 # 114
191 SW $s0, 1504($zero)
192 ADDI $t0, $zero, 111 # 111
193 SW $t0, 1520($zero)
194 ADDI $t1, $zero, 114 # 114
195 SW $t1, 1536($zero)
196 ADDI $s0, $zero, 111 # 111
197 SW $s0, 1552($zero)
198 ADDI $t0, $zero, 114 # 114
199 SW $t0, 1568($zero)
200 ADDI $t1, $zero, 111 # 111
201 SW $t1, 1584($zero)
202 ADDI $s0, $zero, 114 # 114
203 SW $s0, 1600($zero)
204 ADDI $t0, $zero, 111 # 111
205 SW $t0, 1616($zero)
206 ADDI $t1, $zero, 114 # 114
207 SW $t1, 1632($zero)
208 ADDI $s0, $zero, 111 # 111
209 SW $s0, 1648($zero)
210 ADDI $t0, $zero, 114 # 114
211 SW $t0, 1664($zero)
212 ADDI $t1, $zero, 111 # 111
213 SW $t1, 1680($zero)
214 ADDI $s0, $zero, 114 # 114
215 SW $s0, 1696($zero)
216 ADDI $t0, $zero, 111 # 111
217 SW $t0, 1712($zero)
218 ADDI $t1, $zero, 114 # 114
219 SW $t1, 1728($zero)
220 ADDI $s0, $zero, 111 # 111
221 SW $s0, 1744($zero)
222 ADDI $t0, $zero, 114 # 114
223 SW $t0, 1760($zero)
224 ADDI $t1, $zero, 111 # 111
225 SW $t1, 1776($zero)
226 ADDI $s0, $zero, 114 # 114
227 SW $s0, 1792($zero)
228 ADDI $t0, $zero, 111 # 111
229 SW $t0, 1808($zero)
230 ADDI $t1, $zero, 114 # 114
231 SW $t1, 1824($zero)
232 ADDI $s0, $zero, 111 # 111
233 SW $s0, 1840($zero)
234 ADDI $t0, $zero, 114 # 114
235 SW $t0, 1856($zero)
236 ADDI $t1, $zero, 111 # 111
237 SW $t1, 1872($zero)
238 ADDI $s0, $zero, 114 # 114
239 SW $s0, 1888($zero)
240 ADDI $t0, $zero, 111 # 111
241 SW $t0, 1904($zero)
242 ADDI $t1, $zero, 114 # 114
243 SW $t1, 1920($zero)
244 ADDI $s0, $zero, 111 # 111
245 SW $s0, 1936($zero)
246 ADDI $t0, $zero, 114 # 114
247 SW $t0, 1952($zero)
248 ADDI $t1, $zero, 111 # 111
249 SW $t1, 1968($zero)
250 ADDI $s0, $zero, 114 # 114
251 SW $s0, 1984($zero)
252 ADDI $t0, $zero, 111 # 111
253 SW $t0, 2000($zero)
254 ADDI $t1, $zero, 114 # 114
255 SW $t1, 2016($zero)
256 ADDI $s0, $zero, 111 # 111
257 SW $s0, 2032($zero)
258 ADDI $t0, $zero, 114 # 114
259 SW $t0, 2048($zero)
260 ADDI $t1, $zero, 111 # 111
261 SW $t1, 2064($zero)
262 ADDI $s0, $zero, 114 # 114
263 SW $s0, 2080($zero)
264 ADDI $t0, $zero, 111 # 111
265 SW $t0, 2096($zero)
266 ADDI $t1, $zero, 114 # 114
267 SW $t1, 2112($zero)
268 ADDI $s0, $zero, 111 # 111
269 SW $s0, 2128($zero)
270 ADDI $t0, $zero, 114 # 114
271 SW $t0, 2144($zero)
272 ADDI $t1, $zero, 111 # 111
273 SW $t1, 2160($zero)
274 ADDI $s0, $zero, 114 # 114
275 SW $s0, 2176($zero)
276 ADDI $t0, $zero, 111 # 111
277 SW $t0, 2192($zero)
278 ADDI $t1, $zero, 114 # 114
279 SW $t1, 2208($zero)
280 ADDI $s0, $zero, 111 # 111
281 SW $s0, 2224($zero)
282 ADDI $t0, $zero, 114 # 114
283 SW $t0, 2240($zero)
284 ADDI $t1, $zero, 111 # 111
285 SW $t1, 2256($zero)
286 ADDI $s0, $zero, 114 # 114
287 SW $s0, 2272($zero)
288 ADDI $t0, $zero, 111 # 111
289 SW $t0, 2288($zero)
290 ADDI $t1, $zero, 114 # 114
291 SW $t1, 2304($zero)
292 ADDI $s0, $zero, 111 # 111
293 SW $s0, 2320($zero)
294 ADDI $t0, $zero, 114 # 114
295 SW $t0, 2336($zero)
296 ADDI $t1, $zero, 111 # 111
297 SW $t1, 2352($zero)
298 ADDI $s0, $zero, 114 # 114
299 SW $s0, 2368($zero)
300 ADDI $t0, $zero, 111 # 111
301 SW $t0, 2384($zero)
302 ADDI $t1, $zero, 114 # 114
303 SW $t1, 2400($zero)
304 ADDI $s0, $zero, 111 # 111
305 SW $s0, 2416($zero)
306 ADDI $t0, $zero, 114 # 114
307 SW $t0, 2432($zero)
308 ADDI $t1, $zero, 111 # 111
309 SW $t1, 2448($zero)
310 ADDI $s0, $zero, 114 # 114
311 SW $s0, 2464($zero)
312 ADDI $t0, $zero, 111 # 111
313 SW $t0, 2480($zero)
314 ADDI $t1, $zero, 114 # 114
315 SW $t1, 2496($zero)
316 ADDI $s0, $zero, 111 # 111
317 SW $s0, 2512($zero)
318 ADDI $t0, $zero, 114 # 114
319 SW $t0, 2528($zero)
320 ADDI $t1, $zero, 111 # 111
321 SW $t1, 2544($zero)
322 ADDI $s0, $zero, 114 # 114
323 SW $s0, 2560($zero)
324 ADDI $t0, $zero, 111 # 111
325 SW $t0, 2576($zero)
326 ADDI $t1, $zero, 114 # 114
327 SW $t1, 2592($zero)
328 ADDI $s0, $zero, 111 # 111
329 SW $s0, 2608($zero)
330 ADDI $t0, $zero, 114 # 114
331 SW $t0, 2624($zero)
332 ADDI $t1, $zero, 111 # 111
333 SW $t1, 2640($zero)
334 ADDI $s0, $zero, 114 # 114
335 SW $s0, 2656($zero)
336 ADDI $t0, $zero, 111 # 111
337 SW $t0, 2672($zero)
338 ADDI $t1, $zero, 114 # 114
339 SW $t1, 2688($zero)
340 ADDI $s0, $zero, 111 # 111
341 SW $s0, 2704($zero)
342 ADDI $t0, $zero, 114 # 114
343 SW $t0, 2720($zero)
344 ADDI $t1, $zero, 111 # 111
345 SW $t1, 2736($zero)
346 ADDI $s0, $zero, 114 # 114
347 SW $s0, 2752($zero)
348 ADDI $t0, $zero, 111 # 111
349 SW $t0, 2768($zero)
350 ADDI $t1, $zero, 114 # 114
351 SW $t1, 2784($zero)
352 ADDI $s0, $zero, 111 # 111
353 SW $s0, 2800($zero)
354 ADDI $t0, $zero, 114 # 114
355 SW $t0, 2816($zero)
356 ADDI $t1, $zero, 111 # 111
357 SW $t1, 2832($zero)
358 ADDI $s0, $zero, 114 # 114
359 SW $s0, 2848($zero)
360 ADDI $t0, $zero, 111 # 111
361 SW $t0, 2864($zero)
362 ADDI $t1, $zero, 114 # 114
363 SW $t1, 2880($zero)
364 ADDI $s0, $zero, 111 # 111
365 SW $s0, 2896($zero)
366 ADDI $t0, $zero, 114 # 114
367 SW $t0, 2912($zero)
368 ADDI $t1, $zero, 111 # 111
369 SW $t1, 2928($zero)
370 ADDI $s0, $zero, 114 # 114
371 SW $s0, 2944($zero)
372 ADDI $t0, $zero, 111 # 111
373 SW $t0, 2960($zero)
374 ADDI $t1, $zero, 114 # 114
375 SW $t1, 2976($zero)
376 ADDI $s0, $zero, 111 # 111
377 SW $s0, 2992($zero)
378 ADDI $t0, $zero, 114 # 114
379 SW $t0, 3008($zero)
380 ADDI $t1, $zero, 111 # 111
381 SW $t1, 3024($zero)
382 ADDI $s0, $zero, 114 # 114
383 SW $s0, 3040($zero)
384 ADDI $t0, $zero, 111 # 111
385 SW $t0, 3056($zero)
386 ADDI $t1, $zero, 114 # 114
387 SW $t1, 3072($zero)
388 ADDI $s0, $zero, 111 # 111
389 SW $s0, 3088($zero)
390 ADDI $t0, $zero, 114 # 114
391 SW $t0, 3104($zero)
392 ADDI $t1, $zero, 111 # 111
393 SW $t1, 3120($zero)
394 ADDI $s0, $zero, 114 # 114
395 SW $s0, 3136($zero)
396 ADDI $t0, $zero, 111 # 111
397 SW $t0, 3152($zero)
398 ADDI $t1, $zero, 114 # 114
399 SW $t1, 3168($zero)
400 ADDI $s0, $zero, 111 # 111
401 SW $s0, 3184($zero)
402 ADDI $t0, $zero, 114 # 114
403 SW $t0, 3200($zero)
404 ADDI $t1, $zero, 111 # 111
405 SW $t1, 3216($zero)
406 ADDI $s0, $zero, 114 # 114
407 SW $s0, 3232($zero)
408 ADDI $t0, $zero, 111 # 111
409 SW $t0, 3248($zero)
410 ADDI $t1, $zero, 114 # 114
411 SW $t1, 3264($zero)
412 ADDI $s0, $zero, 111 # 111
413 SW $s0, 3280($zero)
414 ADDI $t0, $zero, 114 # 114
415 SW $t0, 3296($zero)
416 ADDI $t1, $zero, 111 # 111
417 SW $t1, 3312($zero)
418 ADDI $s0, $zero, 114 # 114
419 SW $s0, 3328($zero)
420 ADDI $t0, $zero, 111 # 111
421 SW $t0, 3344($zero)
422 ADDI $t1, $zero, 114 # 114
423 SW $t1, 3360($zero)
424 ADDI $s0, $zero, 111 # 111
425 SW $s0, 3376($zero)
426 ADDI $t0, $zero, 114 # 114
427 SW $t0, 3392($zero)
428 ADDI $t1, $zero, 111 # 111
429 SW $t1, 3408($zero)
430 ADDI $s0, $zero, 114 # 114
431 SW $s0, 3424($zero)
432 ADDI $t0, $zero, 111 # 111
433 SW $t0, 3440($zero)
434 ADDI $t1, $zero, 114 # 114
435 SW $t1, 3456($zero)
436 ADDI $s0, $zero, 111 # 111
437 SW $s0, 3472($zero)
438 ADDI $t0, $zero, 114 # 114
439 SW $t0, 3488($zero)
440 ADDI $t1, $zero, 111 # 111
441 SW $t1, 3504($zero)
442 ADDI $s0, $zero, 114 # 114
443 SW $s0, 3520($zero)
444 ADDI $t0, $zero, 111 # 111
445 SW $t0, 3536($zero)
446 ADDI $t1, $zero, 114 # 114
447 SW $t1, 3552($zero)
448 ADDI $s0, $zero, 114 # 114
449 SW $s0, 3568($zero)
450 ADDI $t0, $zero, 111 # 111
451 SW $t0, 3584($zero)
452 ADDI $t1, $zero, 114 # 114
453 SW $t1, 3600($zero)
454 ADDI $s0, $zero, 114 # 114
455 SW $s0, 3616($zero)
456 ADDI $t0, $zero, 111 # 111
457 SW $t0, 3632($zero)
458 ADDI $t1, $zero, 114 # 114
459 SW $t1, 3648($zero)
460 ADDI $s0, $zero, 114 # 114
461 SW $s0, 3664($zero)
462 ADDI $t0, $zero, 111 # 111
463 SW $t0, 3680($zero)
464 ADDI $t1, $zero, 114 # 114
465 SW $t1, 3696($zero)
466 ADDI $s0, $zero, 114 # 114
467 SW $s0, 3712($zero)
468 ADDI $t0, $zero, 111 # 111
469 SW $t0, 3728($zero)
470 ADDI $t1, $zero, 114 # 114
471 SW $t1, 3744($zero)
472 ADDI $s0, $zero, 114 # 114
473 SW $s0, 3760($zero)
474 ADDI $t0, $zero, 111 # 111
475 SW $t0, 3776($zero)
476 ADDI $t1, $zero, 114 # 114
477 SW $t1, 3792($zero)
478 ADDI $s0, $zero, 114 # 114
479 SW $s0, 3808($zero)
480 ADDI $t0, $zero, 111 # 111
481 SW $t0, 3824($zero)
482 ADDI $t1, $zero, 114 # 114
483 SW $t1, 3840($zero)
484 ADDI $s0, $zero, 114 # 114
485 SW $s0, 3856($zero)
486 ADDI $t0, $zero, 111 # 111
487 SW $t0, 3872($zero)
488 ADDI $t1, $zero, 114 # 114
489 SW $t1, 3888($zero)
490 ADDI $s0, $zero, 114 # 114
491 SW $s0, 3904($zero)
492 ADDI $t0, $zero, 111 # 111
493 SW $t0, 3920($zero)
494 ADDI $t1, $zero, 114 # 114
495 SW $t1, 3936($zero)
496 ADDI $s0, $zero, 114 # 114
497 SW $s0, 3952($zero)
498 ADDI $t0, $zero, 111 # 111
499 SW $t0, 3968($zero)
500 ADDI $t1, $zero, 114 # 114
501 SW $t1, 3984($zero)
502 ADDI $s0, $zero, 114 # 114
503 SW $s0, 4000($zero)
504 ADDI $t0, $zero, 111 # 111
505 SW $t0, 4016($zero)
506 ADDI $t1, $zero, 114 # 114
507 SW $t1, 4032($zero)
508 ADDI $s0, $zero, 114 # 114
509 SW $s0, 4048($zero)
510 ADDI $t0, $zero, 111 # 111
511 SW $t0, 4064($zero)
512 ADDI $t1, $zero, 114 # 114
513 SW $t1, 4080($zero)
514 ADDI $s0, $zero, 114 # 114
515 SW $s0, 4096($zero)
516 ADDI $t0, $zero, 111 # 111
517 SW $t0, 4112($zero)
518 ADDI $t1, $zero, 114 # 114
519 SW $t1, 4128($zero)
520 ADDI $s0, $zero, 114 # 114
521 SW $s0, 4144($zero)
522 ADDI $t0, $zero, 111 # 111
523 SW $t0, 4160($zero)
524 ADDI $t1, $zero, 114 # 114
525 SW $t1, 4176($zero)
526 ADDI $s0, $zero, 114 # 114
527 SW $s0, 4192($zero)
528 ADDI $t0, $zero, 111 # 111
529 SW $t0, 4208($zero)
530 ADDI $t1, $zero, 114 # 114
531 SW $t1, 4224($zero)
532 ADDI $s0, $zero, 114 # 114
533 SW $s0, 4240($zero)
534 ADDI $t0, $zero, 111 # 111
535 SW $t0, 4256($zero)
536 ADDI $t1, $zero, 114 # 114
537 SW $t1, 4272($zero)
538 ADDI $s0, $zero, 114 # 114
539 SW $s0, 4288($zero)
540 ADDI $t0, $zero, 111 # 111
541 SW $t0, 4304($zero)
542 ADDI $t1, $zero, 114 # 114
543 SW $t1, 4320($zero)
544 ADDI $s0, $zero, 114 # 114
545 SW $s0, 4336($zero)
546 ADDI $t0, $zero, 111 # 111
547 SW $t0, 4352($zero)
548 ADDI $t1, $zero, 114 # 114
549 SW $t1, 4368($zero)
550 ADDI $s0, $zero, 114 # 114
551 SW $s0, 4384($zero)
552 ADDI $t0, $zero, 111 # 111
553 SW $t0, 4400($zero)
554 ADDI $t1, $zero, 114 # 114
555 SW $t1, 4416($zero)
556 ADDI $s0, $zero, 114 # 114
557 SW $s0, 4432($zero)
558 ADDI $t0, $zero, 111 # 111
559 SW $t0, 4448($zero)
560 ADDI $t1, $zero, 114 # 114
561 SW $t1, 4464($zero)
562 ADDI $s0, $zero, 114 # 114
563 SW $s0, 4480($zero)
564 ADDI $t0, $zero, 111 # 111
565 SW $t0, 4496($zero)
566 ADDI $t1, $zero, 114 # 114
567 SW $t1, 4512($zero)
568 ADDI $s0, $zero, 114 # 114
569 SW $s0, 4528($zero)
570 ADDI $t0, $zero, 111 # 111
571 SW $t0, 4544($zero)
572 ADDI $t1, $zero, 114 # 114
573 SW $t1, 4560($zero)
574 ADDI $s0, $zero, 114 # 114
575 SW $s0, 4576($zero)
576 ADDI $t0, $zero, 111 # 111
577 SW $t0, 4592($zero)
578 ADDI $t1, $zero, 114 # 114
579 SW $t1, 4608($zero)
580 ADDI $s0, $zero, 114 # 114
581 SW $s0, 4624($zero)
582 ADDI $t0, $zero, 111 # 111
583 SW $t0, 4640($zero)
584 ADDI $t1, $zero, 114 # 114
585 SW $t1, 4656($zero)
586 ADDI $s0, $zero, 114 # 114
587 SW $s0, 4672($zero)
588 ADDI $t0, $zero, 111 # 111
589 SW $t0, 4688($zero)
590 ADDI $t1, $zero, 114 # 114
591 SW $t1, 4704($zero)
592 ADDI $s0, $zero, 114 # 114
593 SW $s0, 4720($zero)
594 ADDI $t0, $zero, 111 # 111
595 SW $t0, 4736($zero)
596 ADDI $t1, $zero, 114 # 114
597 SW $t1, 4752($zero)
598 ADDI $s0, $zero, 114 # 114
599 SW $s0, 4768($zero)
600 ADDI $t0, $zero, 111 # 111
601 SW $t0, 4784($zero)
602 ADDI $t1, $zero, 114 # 114
603 SW $t1, 4800($zero)
604 ADDI $s0, $zero, 114 # 114
605 SW $s0, 4816($zero)
606 ADDI $t0, $zero, 111 # 111
607 SW $t0, 4832($zero)
608 ADDI $t1, $zero, 114 # 114
609 SW $t1, 4848($zero)
610 ADDI $s0, $zero, 114 # 114
611 SW $s0, 4864($zero)
612 ADDI $t0, $zero, 111 # 111
613 SW $t0, 4880($zero)
614 ADDI $t1, $zero, 114 # 114
615 SW $t1, 4896($zero)
616 ADDI $s0, $zero, 114 # 114
617 SW $s0, 4912($zero)
618 ADDI $t0, $zero, 111 # 111
619 SW $t0, 4928($zero)
620 ADDI $t1, $zero, 114 # 114
621 SW $t1, 4944($zero)
622 ADDI $s0, $zero, 114 # 114
623 SW $s0, 4960($zero)
624 ADDI $t0, $zero, 111 # 111
625 SW $t0, 4976($zero)
626 ADDI $t1, $zero, 114 # 114
627 SW $t1, 4992($zero)
628 ADDI $s0, $zero, 114 # 114
629 SW $s0, 5008($zero)
630 ADDI $t0, $zero, 111 # 111
631 SW $t0, 5024($zero)
632 ADDI $t1, $zero, 114 # 114
633 SW $t1, 5040($zero)
634 ADDI $s0, $zero, 114 # 114
635 SW $s0, 5056($zero)
636 ADDI $t0, $zero, 111 # 111
637 SW $t0, 5072($
```

MIPS Commands



The screenshot shows a MIPS simulator interface. On the left, there's a text area with assembly code. On the right, there's a register window showing the values of registers \$0 through \$31.

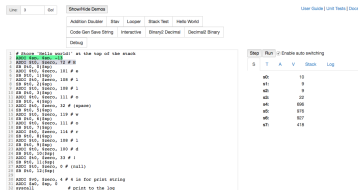
```
1 # Store "hello world" in the top of the stack
2 ADDI $0, $zero, 128
3 SW $0, 128($0)
4 ADDI $0, $zero, 191 # n
5 SW $0, 128($0)
6 ADDI $0, $zero, 128 # 1
7 SW $0, 128($0)
8 ADDI $0, $zero, 131 # n
9 SW $0, 128($0)
10 ADDI $0, $zero, 12 # (space)
11 SW $0, 128($0)
12 ADDI $0, $zero, 131 # n
13 SW $0, 128($0)
14 ADDI $0, $zero, 114 # n
15 SW $0, 128($0)
16 ADDI $0, $zero, 128 # 1
17 SW $0, 128($0)
18 ADDI $0, $zero, 130 # n
19 SW $0, 128($0)
20 ADDI $0, $zero, 33 # 1
21 SW $0, 128($0)
22 ADDI $0, $zero, 0 # (null)
23 SW $0, 128($0)
24 ADDI $0, $zero, 4 # 4 is for print ending
25 ADDI $0, $zero, 0 # print to the log
26 syscall
```

The register window on the right shows the following values:

Register	Value
\$0	0
\$1	0
\$2	0
\$3	0
\$4	0
\$5	0
\$6	0
\$7	0
\$8	0
\$9	0
\$10	0
\$11	0
\$12	0
\$13	0
\$14	0
\$15	0
\$16	0
\$17	0
\$18	0
\$19	0
\$20	0
\$21	0
\$22	0
\$23	0
\$24	0
\$25	0
\$26	0
\$27	0
\$28	0
\$29	0
\$30	0
\$31	0

- **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



```
# Choose "hello world" as the top of the stack
1 addi $v0, $zero, 4
2 ori $a0, $zero, 131 # a
3 sb $t0, 0($zero)
4 addi $v0, $zero, 100 # 1
5 sb $t0, 4($zero)
6 addi $v0, $zero, 111 # s
7 sb $t0, 8($zero)
8 addi $v0, $zero, 12 # \n
9 sb $t0, 12($zero)
10 addi $v0, $zero, 111 # s
11 sb $t0, 16($zero)
12 addi $v0, $zero, 10 # \n
13 sb $t0, 20($zero)
14 addi $v0, $zero, 114 # .
15 sb $t0, 24($zero)
16 addi $v0, $zero, 100 # 1
17 sb $t0, 28($zero)
18 addi $v0, $zero, 100 # a
19 sb $t0, 32($zero)
20 addi $v0, $zero, 33 # !
21 sb $t0, 36($zero)
22 addi $v0, $zero, 0 # (null)
23 sb $t0, 40($zero)
24 addi $v0, $zero, 4 # a is for print ending
25 addi $v0, $zero, 4 # print to the log
26 syscall
```

- **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

```
1 # Please "hello world" as the sign of the success
2 addi $v0, $zero, 12 # 12
3 li $t0, 123456
4 addi $v0, $zero, 181 # 181
5 li $t1, 1234567
6 addi $v0, $zero, 100 # 100
7 li $t2, 12345678
8 addi $v0, $zero, 100 # 100
9 li $t3, 123456789
10 addi $v0, $zero, 111 # 111
11 li $t4, 1234567890
12 addi $v0, $zero, 12 # (space)
13 li $t5, 12345678901
14 addi $v0, $zero, 111 # 111
15 li $t6, 123456789012
16 addi $v0, $zero, 114 # 114
17 li $t7, 1234567890123
18 addi $v0, $zero, 100 # 100
19 li $t8, 12345678901234
20 addi $v0, $zero, 100 # 100
21 li $t9, 123456789012345
22 addi $v0, $zero, 10 # (newline)
23 li $t10, 1234567890123456
24 addi $v0, $zero, 0 # (null)
25 li $t11, 12345678901234567
26 addi $v0, $zero, 4 # 4 is for print ending
27 addi $v0, $zero, 4
28 syscall
29 # print to the log
```

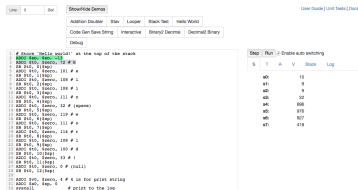
- **Registers:** locations for storing information that can be quickly accessed. Names start with '\$': \$s0, \$s1, \$t0, \$t1,...
- **R Instructions:** Commands that use data in the registers:
add \$s1, \$s2, \$s3 (Basic form: OP rd, rs, rt)
- **I Instructions:** instructions that also use intermediate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

MIPS Commands

[illegible]

- **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



```
1 # Print "Hello world!" as the top of the stack
2 ADDI $t0, $zero, 12 # 12
3 SW $t0, 0($zero)
4 ADDI $t0, $zero, 181 # 181
5 SW $t0, 100($zero)
6 ADDI $t0, $zero, 100 # 100
7 SW $t0, 200($zero)
8 ADDI $t0, $zero, 100 # 100
9 SW $t0, 300($zero)
10 ADDI $t0, $zero, 111 # 111
11 SW $t0, 400($zero)
12 ADDI $t0, $zero, 12 # (again)
13 SW $t0, 500($zero)
14 ADDI $t0, $zero, 111 # 111
15 SW $t0, 600($zero)
16 ADDI $t0, $zero, 114 # 114
17 SW $t0, 700($zero)
18 ADDI $t0, $zero, 100 # 100
19 SW $t0, 800($zero)
20 ADDI $t0, $zero, 100 # 100
21 SW $t0, 900($zero)
22 ADDI $t0, $zero, 10 # 10
23 SW $t0, 1000($zero)
24 ADDI $t0, $zero, 0 # (null)
25 SW $t0, 1100($zero)
26 ADDI $t0, $zero, 4 # 4 is for print string
27 ADDI $t0, $zero, 0
28 syscall # print to the log
```

Register	Value
\$t0	10
\$t1	9
\$t2	8
\$t3	7
\$t4	6
\$t5	5
\$t6	4
\$t7	3

- **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

[illegible]

- **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.
j done

MIPS Commands

[illegible]

- **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.
j done (Basic form: OP label)

Challenge:

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 # i
```

```
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
```

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

Line 3

Out

Show/Hide Details

Addition Doubler

Stop

Looper

Stack Test

Hello World

Code Gen Save Setting

Interactive

Binary2 Decimal

Decimal2 Binary

Debug

```

1 # Store 'Hello world' at the top of the stack
2 ADDI $a0, $a0, 0
3 ADDI $k0, $a0, 0, 12 # $k0
4 CD $k0, 0($a0)
5 ADDI $t0, $a0, 101 # $t0
6 CD $t0, 1($a0)
7 ADDI $t0, $a0, 108 # $t0
8 CD $t0, 2($a0)
9 ADDI $t0, $a0, 109 # $t0
10 CD $t0, 3($a0)
11 ADDI $t0, $a0, 11 # $t0
12 CD $t0, 4($a0)
13 ADDI $t0, $a0, 32 # ($a0)
14 CD $t0, 5($a0)
15 ADDI $t0, $a0, 119 # $t0
16 CD $t0, 6($a0)
17 ADDI $t0, $a0, 11 # $t0
18 CD $t0, 7($a0)
19 ADDI $t0, $a0, 114 # $t0
20 CD $t0, 8($a0)
21 ADDI $t0, $a0, 108 # $t0
22 CD $t0, 9($a0)
23 ADDI $t0, $a0, 109 # $t0
24 CD $t0, 10($a0)
25 ADDI $t0, $a0, 33 # $t0
26 ADDI $t0, $a0, 0 # (null)
27 CD $t0, 11($a0)
28 ADDI $v0, $a0, 4 # 4 in for print string
29 ADDI $a0, $a0, 0
30 syscall
31 # print to the log

```

Step

Run

Enable auto switching

S	T	A	V	Stack	Log
				\$t0	10
				\$t0	9
				\$t0	9
				\$t0	22
				\$t0	908
				\$t0	976
				\$t0	907
				\$t0	418

(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.



The screenshot shows a debugger window with two panes. The left pane displays assembly code with instructions like `movl $0, %eax`, `movl $1, %ecx`, and `jmp $0x00000000`. The right pane shows the state of registers, including `%eax`, `%ecx`, and `%edx`.

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.



The screenshot shows a code editor with assembly code. On the left, a list of instructions is visible, with some lines highlighted in green. On the right, a larger window shows a snippet of assembly code. The code includes labels like `loop_start` and `loop_end`, and instructions like `jmp` (jump) and `inc` (increment). The code is written in a low-level, assembly-like syntax.

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:



The screenshot shows a debugger window with two panes. The left pane displays assembly code with several labels (e.g., `start`, `loop`) and instructions like `jmp` (jump). The right pane shows a hex dump of the memory. The assembly code includes a loop structure where a label is followed by instructions that eventually lead to a jump back to the label.

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

Line: 18 Go!

Show/Hide Demos

[User Guide](#) | [Unit Tests](#) | [Docs](#)

```
1
2 ADDI $sp, $sp, -27      # Set up stack
3 ADDI $s3, $zero, 1     # Store 1 in a register
4 ADDI $t0, $zero, 97    # Set $t0 at 97 (a)
5 ADDI $s2, $zero, 26    # Use to test when you reach 26
6 SETUP: SB $t0, 0($sp)   # Next letter in $t0
7 ADDI $sp, $sp, 1       # Increment the stack
8 SUB $s2, $s2, $s3      # Decrease the counter by 1
9 ADDI $t0, $t0, 1       # Increment the letter
10 BEQ $s2, $zero, DONE   # Jump to done if $s2 == 0
11 J SETUP                # Else, jump back to SETUP
12 DONE: ADDI $t0, $zero, 0 # Null (0) to terminate string
13 SB $t0, 0($sp)         # Add null to stack
14 ADDI $sp, $sp, -26     # Set up stack to print
15 ADDI $v0, $zero, 4     # 4 is for print string
16 ADDI $a0, $sp, 0       # Set $a0 to stack pointer
17 syscall                # Print to the log
```

(Demo
with
WeMIPS)

Step **Run** ☒ Enable auto switching

S T A V Stack Log

Clear Log

Emulation complete, returning to line 1

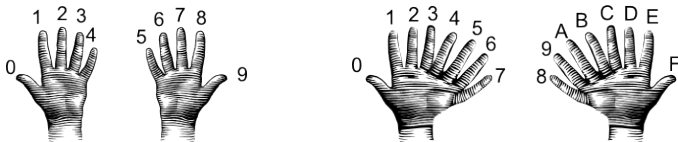
abcdefghijklmnopqrstuvwxyz

Today's Topics



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

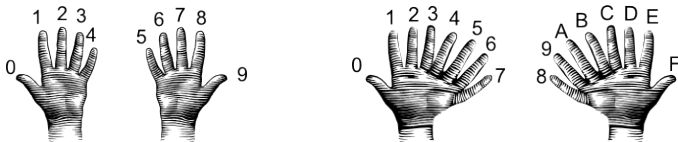
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.

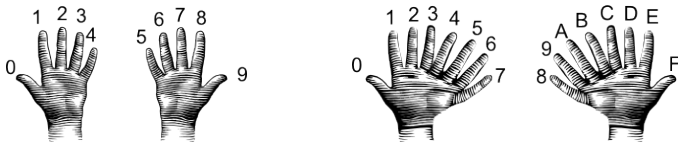
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ Convert first digit to decimal and multiple by 16.
 - ▶ Convert second digit to decimal and add to total.

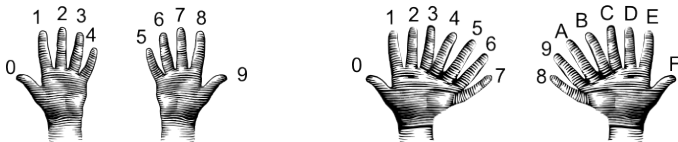
Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):
 - ▶ 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



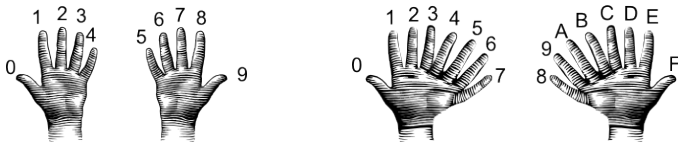
(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):

- ▶ 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



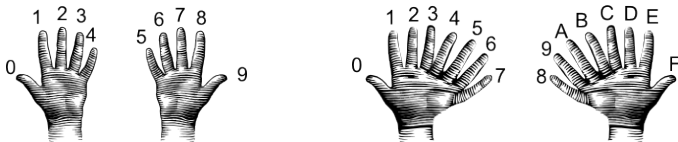
(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):

- ▶ 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. 2×16 is 32.

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

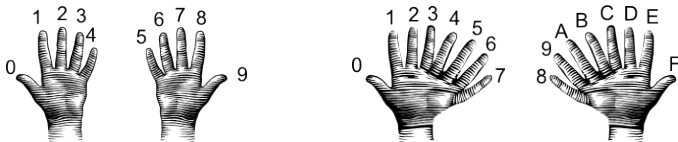
- From hexadecimal to decimal (assuming two-digit numbers):

- ▶ 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. 2×16 is 32.

A in decimal digits is 10.

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):

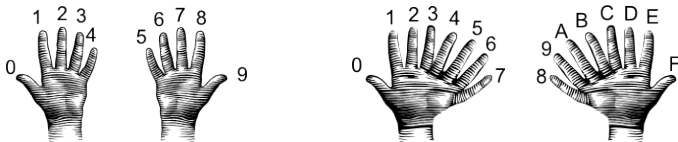
- ▶ Convert first digit to decimal and multiple by 16.
- ▶ Convert second digit to decimal and add to total.
- ▶ Example: what is 2A as a decimal number?

2 in decimal is 2. 2×16 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 (assuming two-digit numbers):

- ▶ 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. 2×16 is 32.

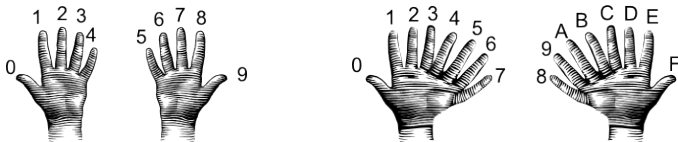
A in decimal digits is 10.

$32 + 10$ 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 (assuming two-digit numbers):

- ▶ 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. 2×16 is 32.

A in decimal digits is 10.

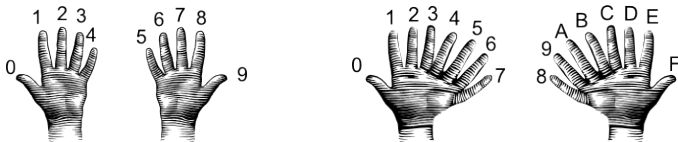
$32 + 10$ is 42.

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 (assuming two-digit numbers):

- ▶ 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. 2×16 is 32.

A in decimal digits is 10.

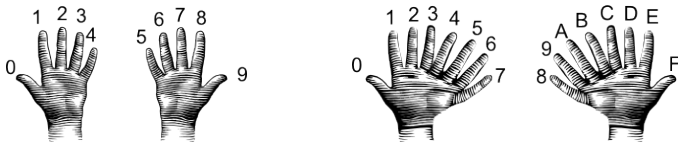
$32 + 10$ is 42.

Answer is 42.

- ▶ Example: what is 99 as a decimal number?

9 in decimal is 9. 9×16 is 144.

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):

- ▶ 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. 2×16 is 32.

A in decimal digits is 10.

$32 + 10$ is 42.

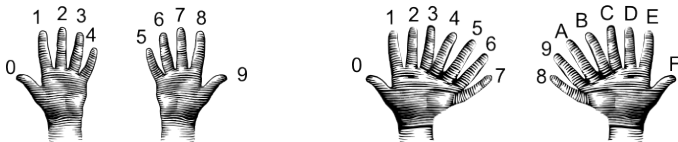
Answer is 42.

- ▶ Example: what is 99 as a decimal number?

9 in decimal is 9. 9×16 is 144.

9 in decimal digits is 9

Hexadecimal to Decimal: Converting Between Bases



(from i-programmer.info)

- From hexadecimal to decimal (assuming two-digit numbers):

- ▶ 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. 2×16 is 32.

A in decimal digits is 10.

$32 + 10$ is 42.

Answer is 42.

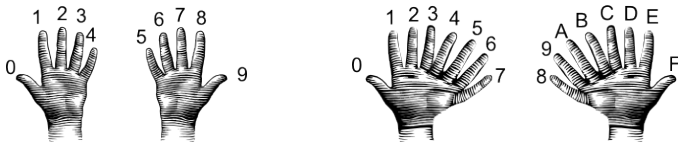
- ▶ Example: what is 99 as a decimal number?

9 in decimal is 9. 9×16 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 (assuming two-digit numbers):

- ▶ 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. 2×16 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. 9×16 is 144.

9 in decimal digits is 9

$144 + 9$ is 153.

Answer is 153.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From decimal to binary:

- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

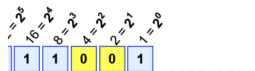


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

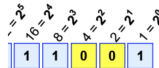


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

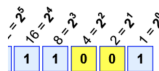


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

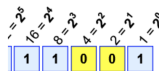


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

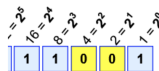


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

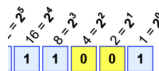


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

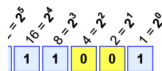


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



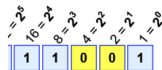
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

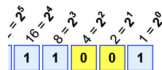


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

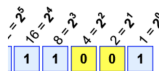
From decimal to binary:

- ▶ 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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

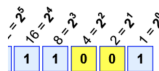
From decimal to binary:

- ▶ 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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

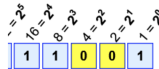
From decimal to binary:

- ▶ 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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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.

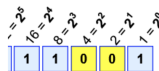
Example: what is 130 in binary notation?

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

From decimal to binary:

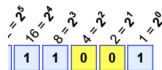
- ▶ Divide by 128 ($= 2^7$). Quotient is the first digit.
- ▶ Divide remainder by 64 ($= 2^6$). Quotient is the next digit.
- ▶ Divide remainder by 32 ($= 2^5$). Quotient is the next digit.
- ▶ Divide remainder by 16 ($= 2^4$). Quotient is the next digit.
- ▶ Divide remainder by 8 ($= 2^3$). Quotient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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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?

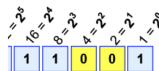
130/128 is 1 rem 2. First digit is 1: 1...

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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?

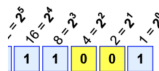
130/128 is 1 rem 2. First digit is 1: 1...

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



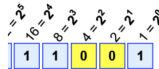
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



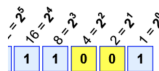
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



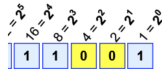
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



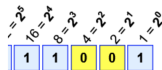
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



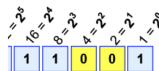
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



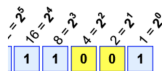
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



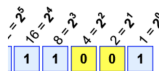
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



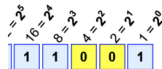
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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...

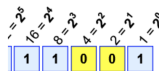
2/4 is 0 remainder 2. Next digit is 0: 100000...

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

Adding the last remainder:

10000010

Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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...

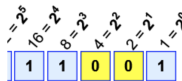
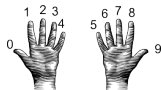
2/4 is 0 remainder 2. Next digit is 0: 100000...

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

Adding the last remainder: 10000010



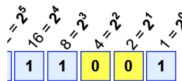
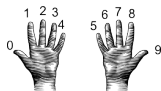
Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

Decimal to Binary: Converting Between Bases

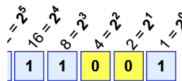
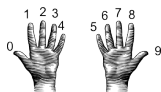


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

99/128 is 0 rem 99.

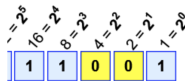
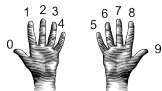
Decimal to Binary: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?
99/128 is 0 rem 99. First digit is 0:

Decimal to Binary: Converting Between Bases



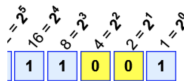
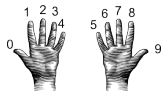
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



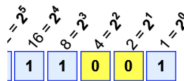
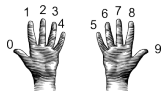
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



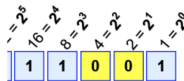
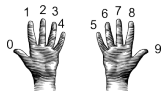
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

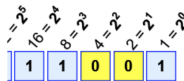
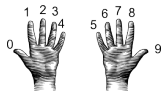
- Example: what is 99 in binary notation?

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

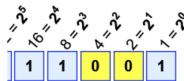
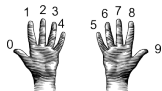
- Example: what is 99 in binary notation?

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

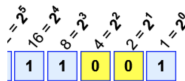
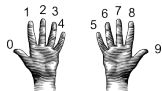
- Example: what is 99 in binary notation?

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

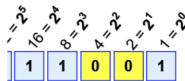
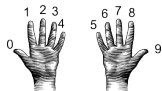
99/128 is 0 rem 99. First digit is 0: 0...

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

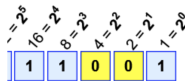
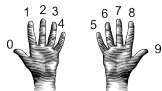
99/128 is 0 rem 99. First digit is 0: 0...

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

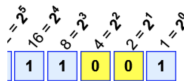
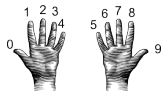
99/128 is 0 rem 99. First digit is 0: 0...

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

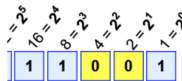
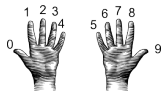
99/64 is 1 rem 35. Next digit is 1: 01...

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: what is 99 in binary notation?

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

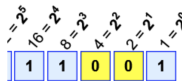
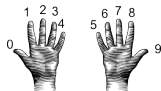
99/64 is 1 rem 35. Next digit is 1: 01...

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

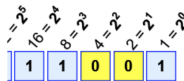
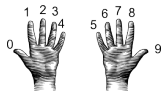


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

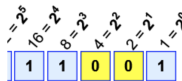
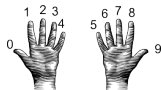


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

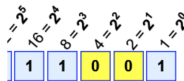
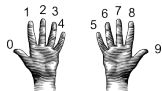


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

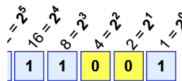
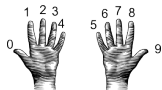
35/32 is 1 rem 3. Next digit is 1: 011...

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

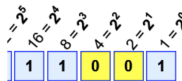
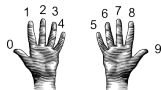
3/16 is 0 rem 3. Next digit is 0: 0110...

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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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...

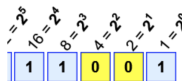
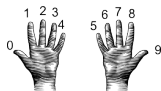
3/16 is 0 rem 3. Next digit is 0: 0110...

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

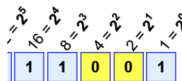
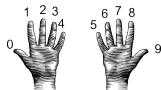


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

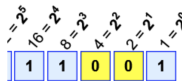
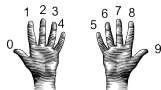


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



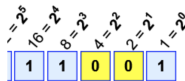
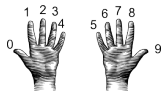
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

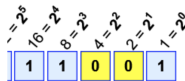
Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

Binary to Decimal: Converting Between Bases

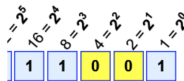
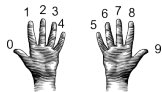


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- From binary to decimal:

- ▶ Set sum = last digit.
- ▶ Multiply next digit by $2 = 2^1$. Add to sum.

Binary to Decimal: Converting Between Bases

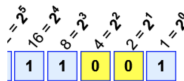
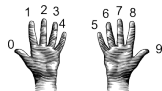


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

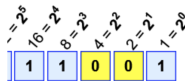
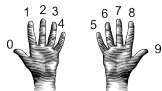


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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.

Binary to Decimal: Converting Between Bases

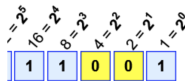
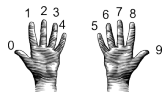


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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

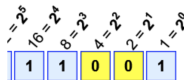
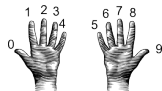


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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.

Binary to Decimal: Converting Between Bases

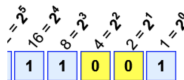


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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.

Binary to Decimal: Converting Between Bases

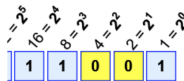


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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.

Binary to Decimal: Converting Between Bases

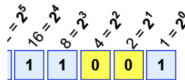
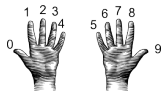


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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.

Binary to Decimal: Converting Between Bases



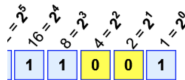
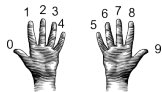
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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:

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

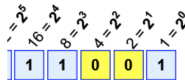
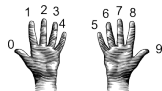
● From binary to decimal:

- ▶ 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

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

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

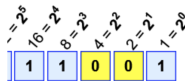
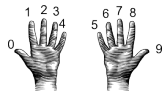
● From binary to decimal:

- ▶ 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

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

Binary to Decimal: Converting Between Bases



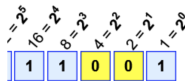
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum:

Binary to Decimal: Converting Between Bases



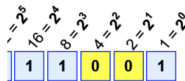
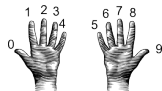
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5

Binary to Decimal: Converting Between Bases



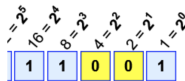
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5
 $1 \times 8 = 8$. Add 8 to sum:

Binary to Decimal: Converting Between Bases



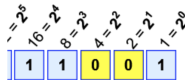
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13

Binary to Decimal: Converting Between Bases



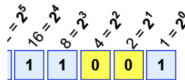
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5
 $1 \times 8 = 8$. Add 8 to sum: 13
 $1 \times 16 = 16$. Add 16 to sum:

Binary to Decimal: Converting Between Bases



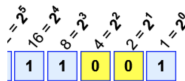
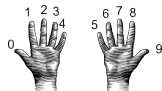
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13
$1 \times 16 = 16$. Add 16 to sum:	29

Binary to Decimal: Converting Between Bases



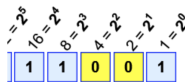
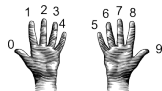
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5
 $1 \times 8 = 8$. Add 8 to sum: 13
 $1 \times 16 = 16$. Add 16 to sum: 29
 $1 \times 32 = 32$. Add 32 to sum:

Binary to Decimal: Converting Between Bases



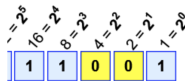
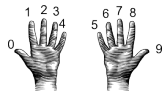
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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
 $0 \times 2 = 0$. Add 0 to sum: 1
 $1 \times 4 = 4$. Add 4 to sum: 5
 $1 \times 8 = 8$. Add 8 to sum: 13
 $1 \times 16 = 16$. Add 16 to sum: 29
 $1 \times 32 = 32$. Add 32 to sum: 61

Binary to Decimal: Converting Between Bases



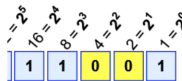
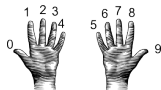
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

● 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
$0 \times 2 = 0$. Add 0 to sum:	1
$1 \times 4 = 4$. Add 4 to sum:	5
$1 \times 8 = 8$. Add 8 to sum:	13
$1 \times 16 = 16$. Add 16 to sum:	29
$1 \times 32 = 32$. Add 32 to sum:	61

Binary to Decimal: Converting Between Bases

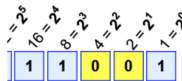
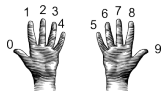


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with:

Binary to Decimal: Converting Between Bases



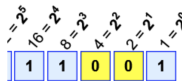
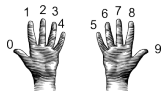
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

Binary to Decimal: Converting Between Bases



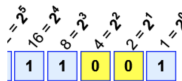
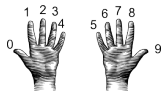
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

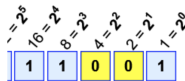
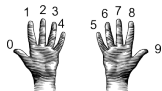
- Example: What is 10100100 in decimal?

Sum starts with: 0

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

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

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

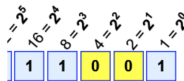
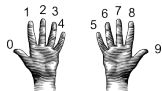
- Example: What is 10100100 in decimal?

Sum starts with: 0

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

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

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

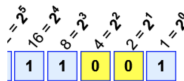
Sum starts with: 0

$0 \times 2 = 0$. 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

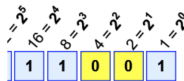
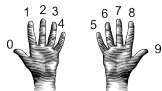
Sum starts with: 0

$0 \times 2 = 0$. 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

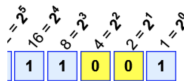
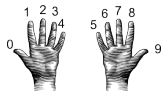
$0 \times 2 = 0$. Add 0 to sum: 0

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

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

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

Binary to Decimal: Converting Between Bases



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

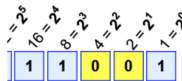
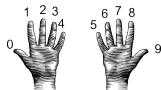
$0 \times 2 = 0$. Add 0 to sum: 0

$1 \times 4 = 4$. 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: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- Example: What is 10100100 in decimal?

Sum starts with: 0

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

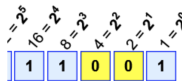
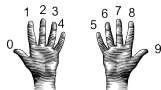
$1 \times 4 = 4$. Add 4 to sum: 4

$0 \times 8 = 0$. 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

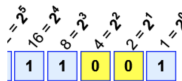
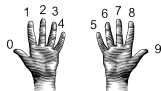


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

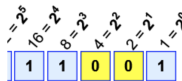
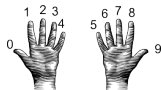


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

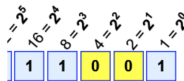
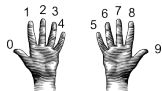


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

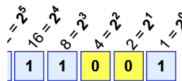
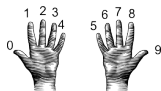


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

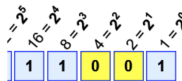
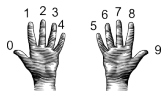


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



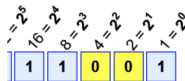
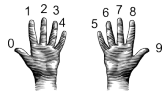
Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

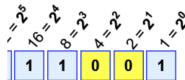
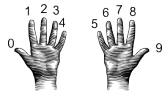
Design Challenge: Incrementers



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

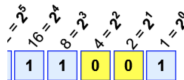
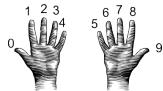
Design Challenge: Incrementers



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

Design Challenge: Incrementers

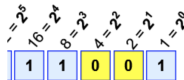
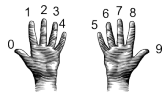


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

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

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

Design Challenge: Incrementers

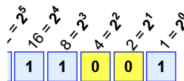


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

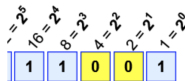
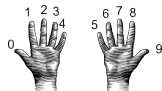


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

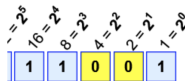
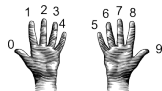


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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

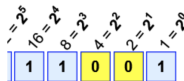
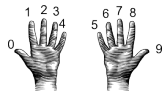


Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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



Example: $1 \times 16 + 1 \times 8 + 1 \times 1 = 16 + 8 + 1 = 25$

- 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.

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 during the time of the exam
- The exam format:

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- The exam format:
 - ▶ Like a long Lab Quiz, you scroll down to answer all questions.

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- The exam format:
 - ▶ Like a long Lab Quiz, you scroll down to answer all questions.
 - ▶ Questions roughly correspond to the 10 parts from old exams, but will appear as a larger number of questions on Gradescope

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- The exam format:
 - ▶ Like a long Lab Quiz, you scroll down to answer all questions.
 - ▶ Questions roughly correspond to the 10 parts from old exams, but will appear as a larger number of questions on Gradescope
 - ▶ Questions are variations on the programming assignments, lab exercises, and lecture design challenges.

Final Overview: Administration

- The exam will be administered through Gradescope.
- The exam will be available on Gradescope only on during the time of the exam
- The exam format:
 - ▶ Like a long Lab Quiz, you scroll down to answer all questions.
 - ▶ Questions roughly correspond to the 10 parts from old exams, but will appear as a larger number of questions on Gradescope
 - ▶ Questions 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.

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.

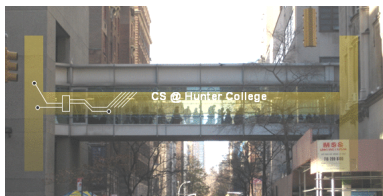
Final Overview: Format

- 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.
 - ▶ Avoid scrambling through web searches and waste time during 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.
 - ▶ Avoid scrambling through web searches and waste time during the exam.
- Past exams available on webpage (includes answer keys).

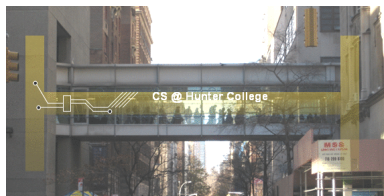
Class Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab

Class Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab
- Take the Lab Quiz on Gradescope by 9pm on Today

Class Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab
- Take the Lab Quiz on Gradescope by 9pm on Today
- Submit this class's programming assignments (programs 50-52)

Class Reminders!



Before next lecture, don't forget to:

- Review this class's Lecture and Lab
- Take the Lab Quiz on Gradescope by 9pm on Today
- Submit this class's programming assignments (programs 50-52)
- At any point, visit our TA for help!!!