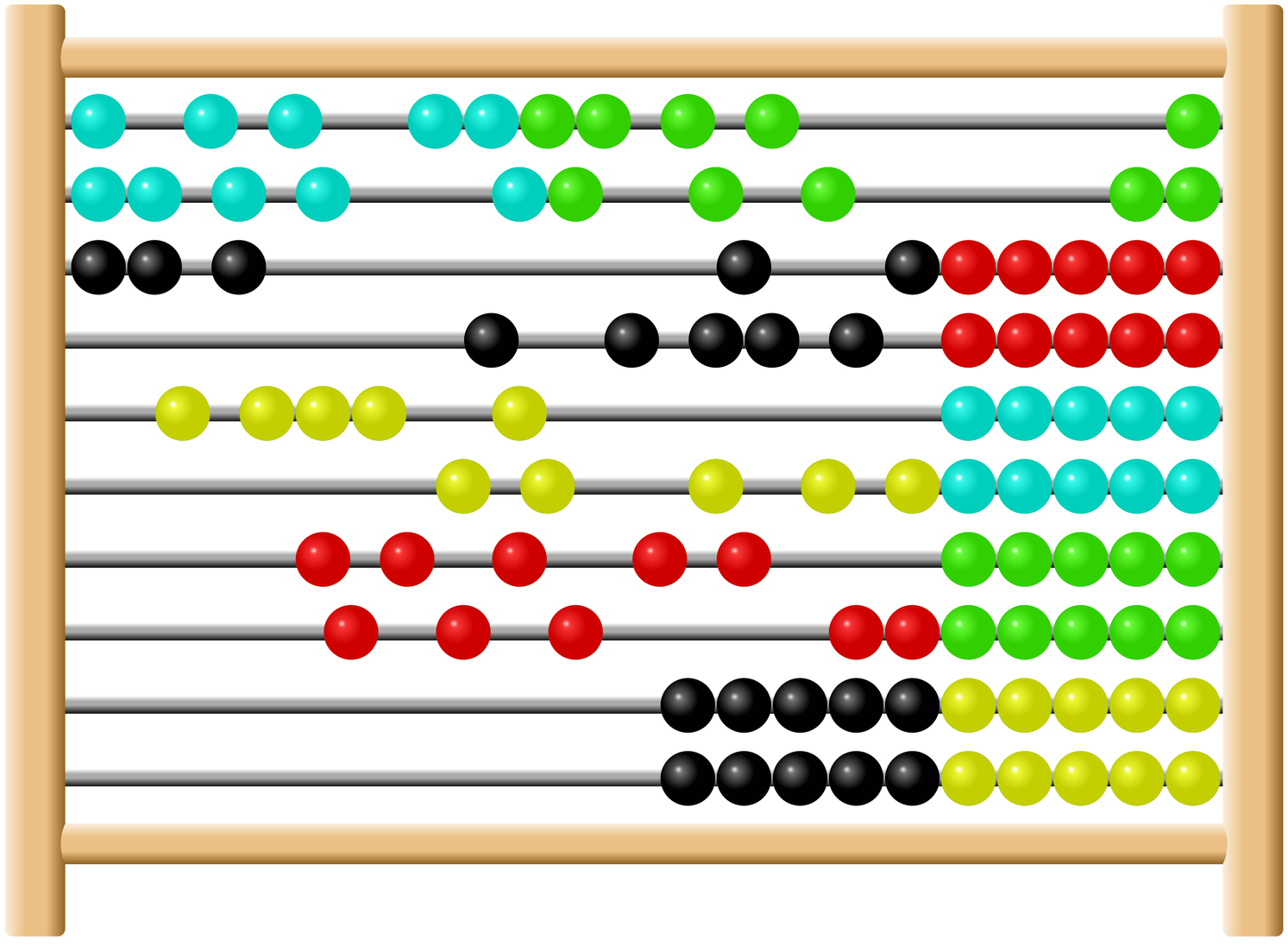
Overview

Language Fundamentals

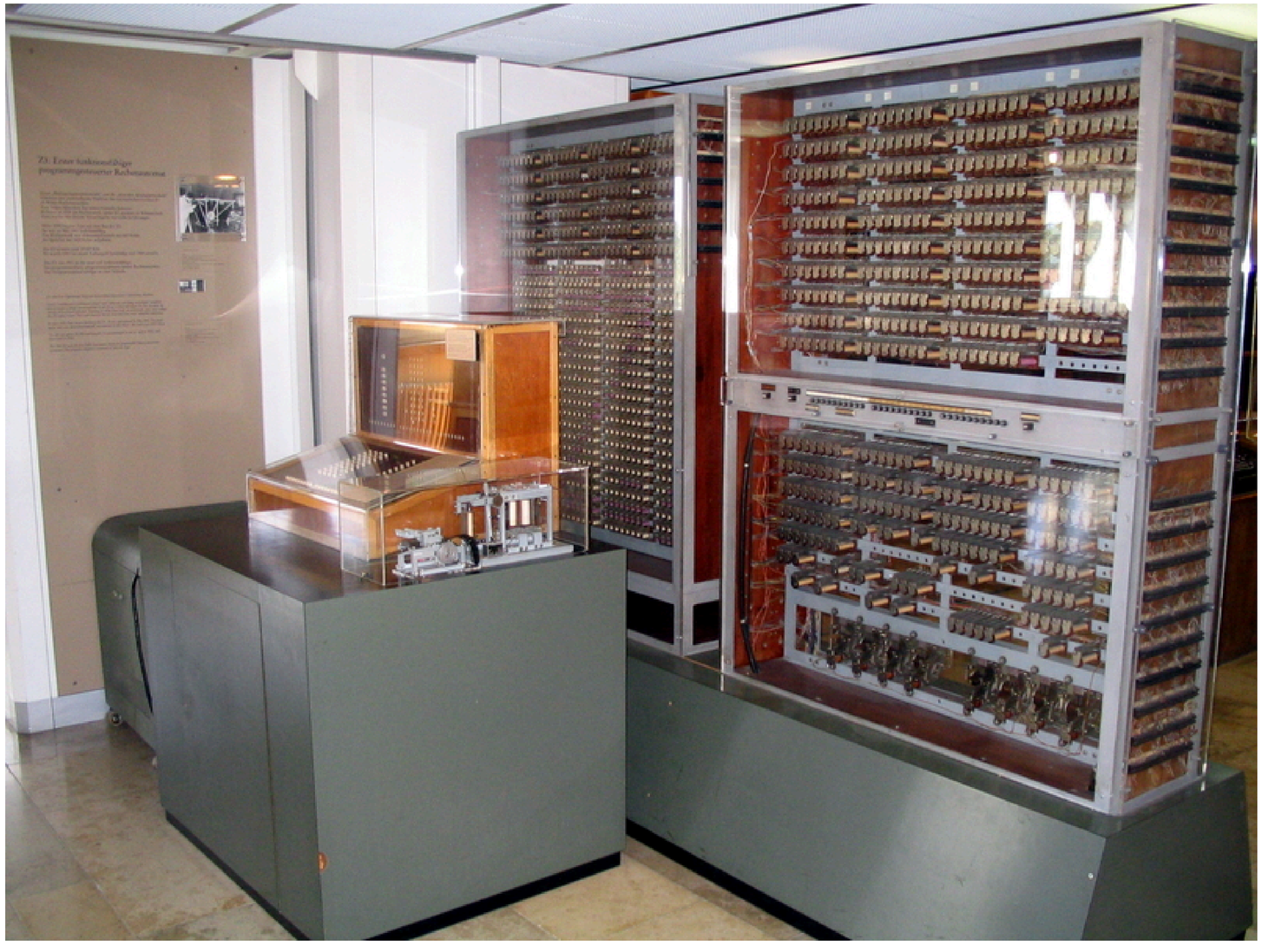
- Integer, ASCII and Unicode

Manual calculation: Abacus

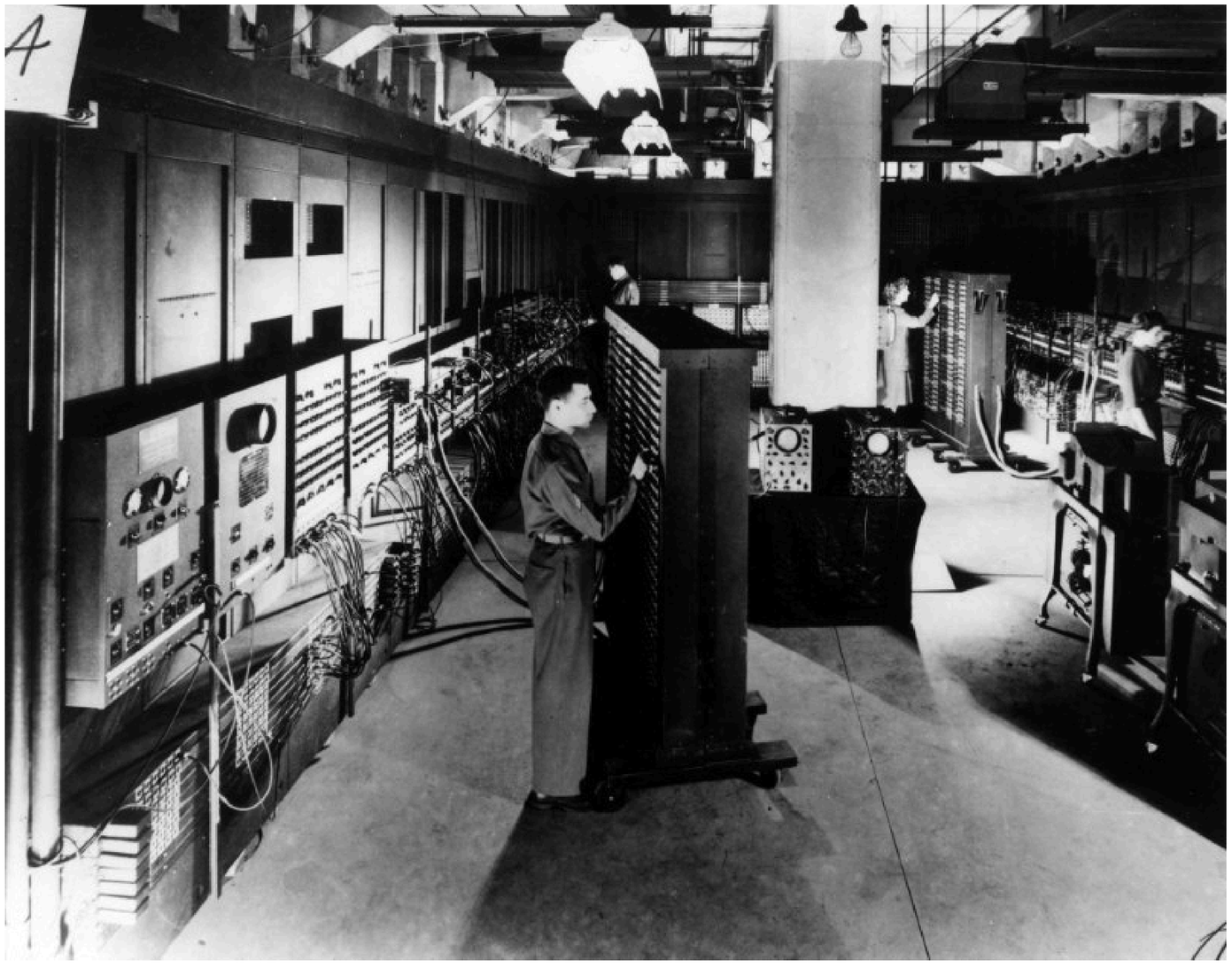


Mechanical calculation: Cash register

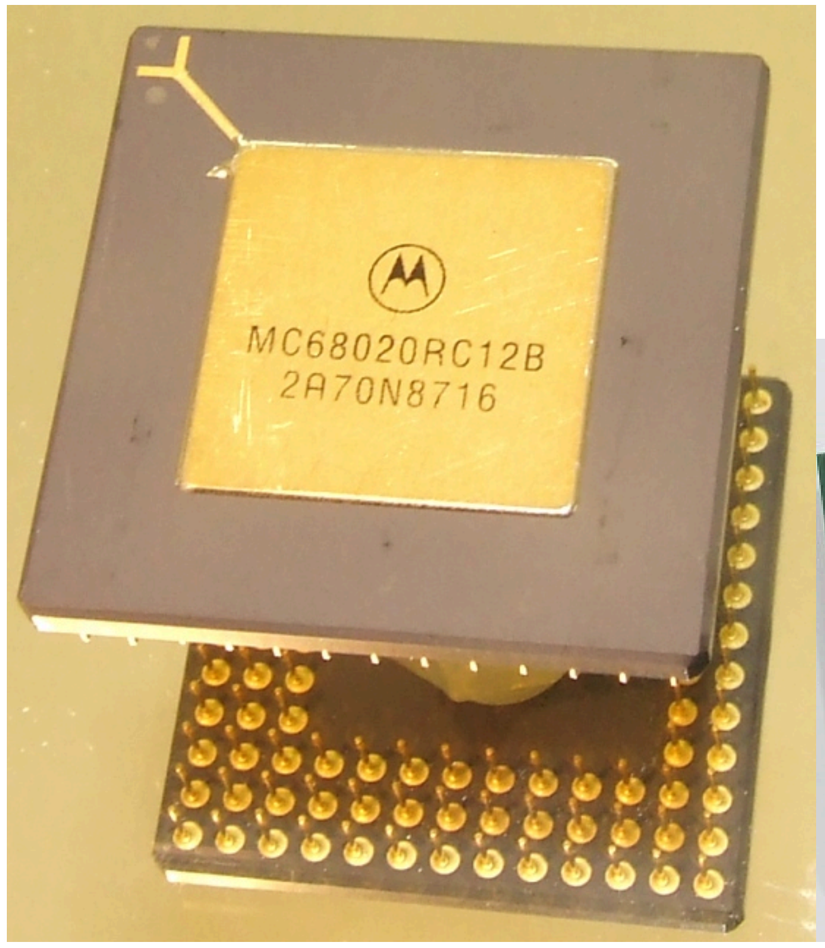
Electromechanical calculation: Zuse Z3

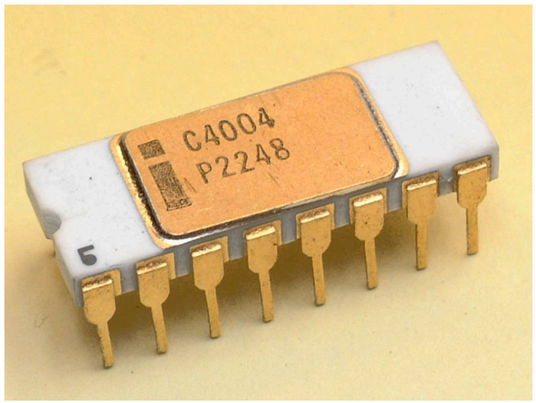


Vacuum Tube: Eniac

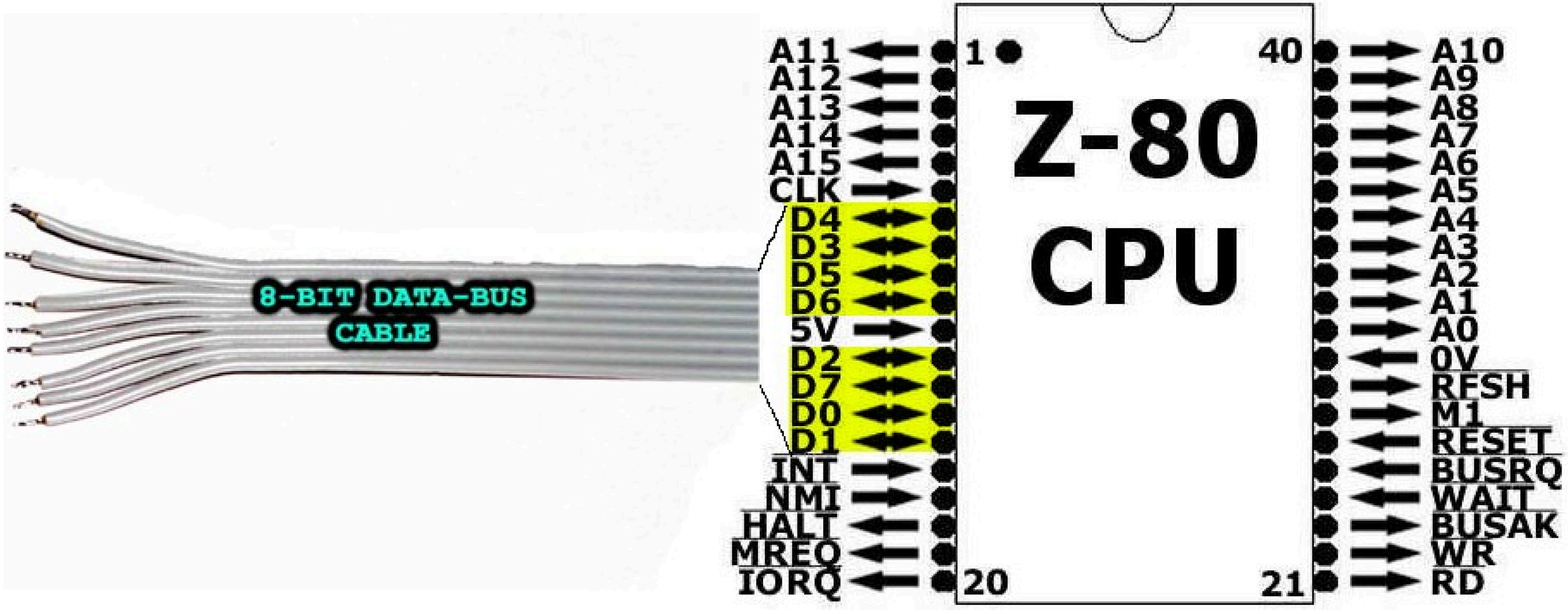


Transistor: Microprocessor ICs





Z80 8-bit data bus



Progress in hardware 1

Processor	Year	Address/ data bus	Transistors	Clock rate
Intel 4004	1971	12 / 4	2,300	740 kHz
Zilog Z80	1976	16 / 8	8,500	2.5 MHz
Motorola 68020	1984	32 / 32	190,000	12.5 MHz

Progress in hardware 2

Processor	Year	Address/ data bus	Transistors	Clock rate
Six-core Opteron	2009	64 / 64	904,000,000	1.8 GHz
Core i7 Broadwell	2016	64 / 64	3,200,000,000	3.6 GHz
Apple's ARM M1 Ultra	2022	64 / 64	114,000,000,000	3.2 GHz

Simple facts:

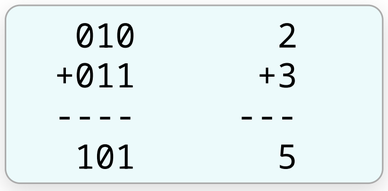
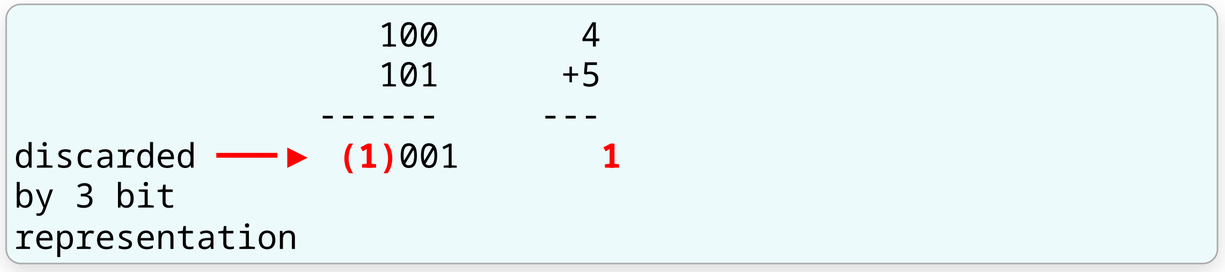
*There are only **10** types of people in the world:*

Those who understand binary and those who don't.

Unsigned 3 bit integer representation

0	0	0	0	$0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$
1	0	0	1	$0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
2	0	1	0	$0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$
3	0	1	1	$0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$
4	1	0	0	$1 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0$
5	1	0	1	$1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$
6	1	1	0	$1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0$
7	1	1	1	$1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

Binary system addition

Within limits: o.K.		**Caution: Overflow!**
		

3 bit two-complement representation

-4	1	0	0	$-2^{(3-1)}$
-3	1	0	1	
-2	1	1	0	
-1	1	1	1	
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	$2^{(3-1)}-1$

3 bit two complement rationale: “Usual” addition

Within limits: o.K.		**Caution: Overflow!**
`` 101 -3 +010 +2 ---- --- 111 -1```		``` 100 -4 101 -3 ---- --- 1001 1```

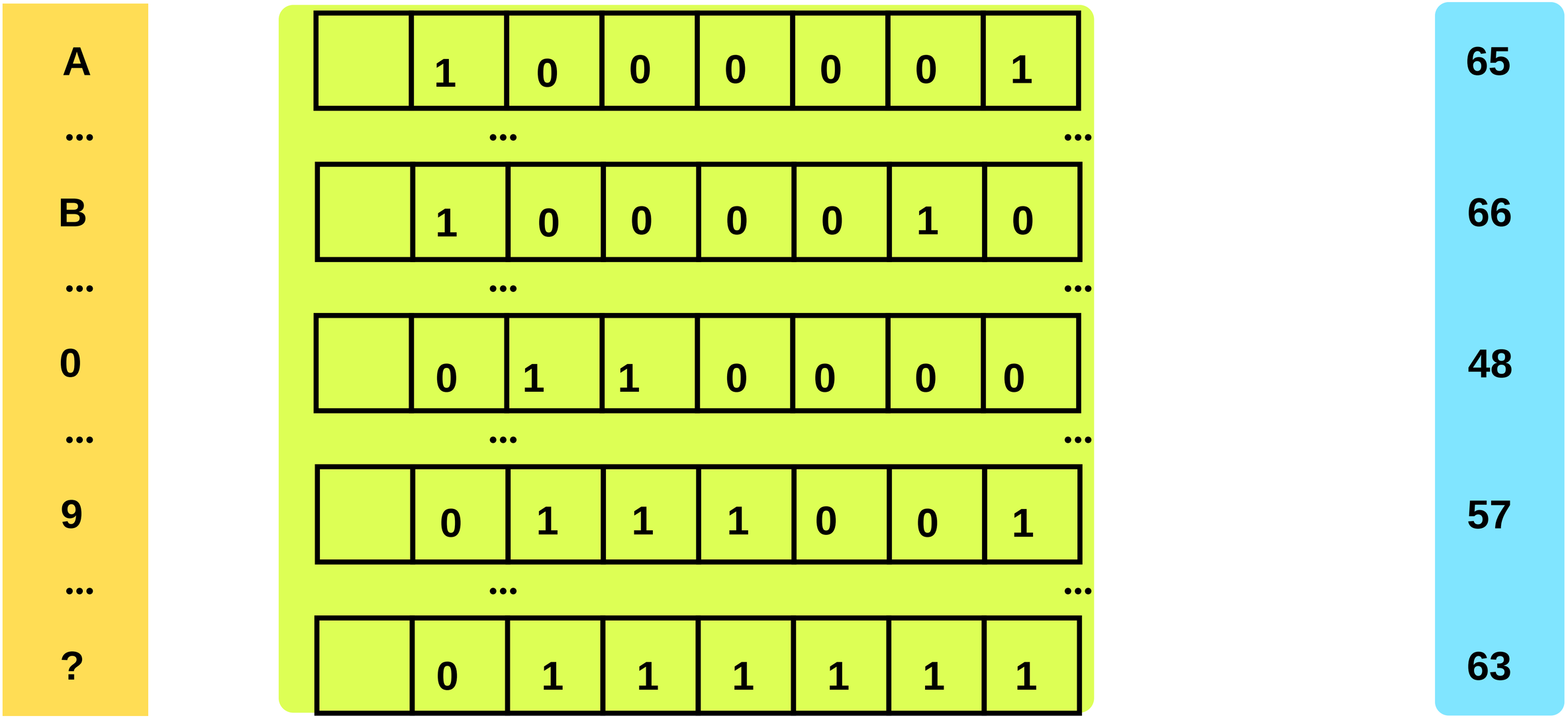
Signed 8 bit integer binary representation

-128	1 0 0 0 0 0 0 0	$-2^{(8-1)}$
-127	1 0 0 0 0 0 0 1	
-1	1 1 1 1 1 1 1 1	
...	...	
0	0 0 0 0 0 0 0 0	
1	0 0 0 0 0 0 0 1	
...	...	
126	0 1 1 1 1 1 1 0	
127	0 1 1 1 1 1 1 1	$2^{(8-1)} - 1$

Followup exercise

5. Hotel key cards

7-bit ASCII



7-bit ASCII with even parity bit

A	0	1	0	0	0	0	0	1	65
B	0	1	0	0	0	0	1	0	66
C	1	1	0	0	0	0	1	1	67
D	0	1	0	0	0	1	0	0	68

Western European characters: ISO Latin 1 encoding

A	0	1	0	0	0	0	0	1	65
µ	1	0	1	1	0	1	0	1	181
¾	1	0	1	1	1	1	1	0	190
Å	1	1	0	0	0	1	0	1	197
Ç	1	1	0	0	0	1	1	1	199
Ä	1	1	0	0	0	1	0	0	196
ß	1	1	0	1	1	1	1	1	223

Unicode UTF-8 samples

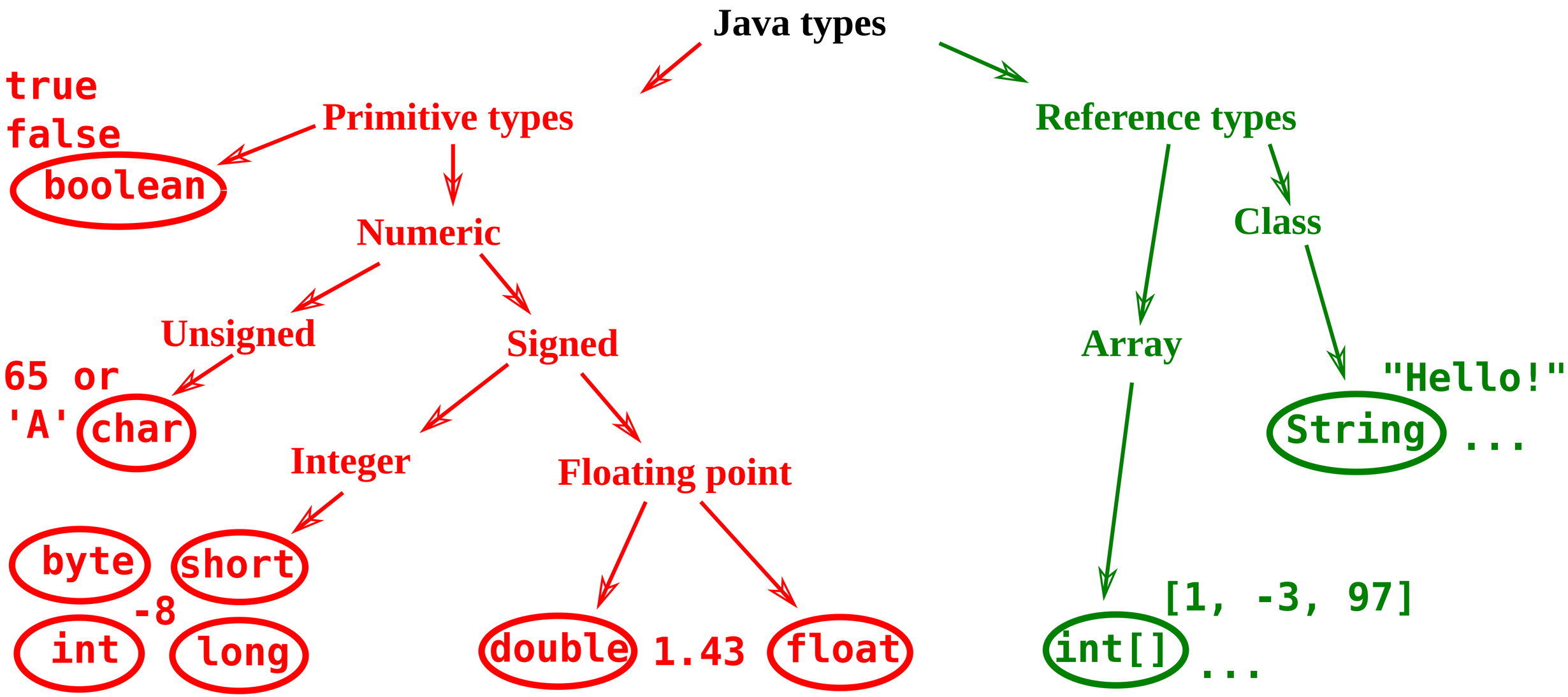
A	0 1 0 0 0 0 0 1	65
ښ	0 0 0 0 0 1 1 0 1 0 0 1 1 1 0 0	1692
	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 0 1 1 0 0 0 0 0 1 1 1 0	128526

Overview

Language Fundamentals

- Primitive types

Java types


Java types
Primitive types
Reference types
true
false
boolean
Numeric
Class
Unsigned
Signed
Array
65 or
'A'
char
Integer
Floating point
"Hello!"
String
...
byte
short
-8
int
long
double
1.43
float
[1, -3, 97]
int[]
...

Java signed integer primitive types

Name	Bytes	Type	Range	Literal samples
`byte`	1 / **8 bit**	Signed integer	$\left[-2^{7}, 2^{7}-1\right]$	-
`short`	2 / **16 bit**	Signed integer	$\left[-2^{15}, 2^{15}-1\right]$	-
`int`	4 / **32 bit**	Signed integer	$\left[-2^{31}, 2^{31}-1\right]$	0, 1, +1, -42, 0b1101, 017, 0xC
`long`	8 / **64 bit**	Signed integer	$\left[-2^{63}, 2^{63}-1\right]$	0L, 1L, +1L, -42L, 0b1101L, 017L, 0xCL

`int` literals explained

Literal	Discriminator	Type	Value
29	base **10**	Decimal	$2 \times 10^1 + 9 \times 10^0$
0b11101	**0b**, base **2**	Binary	$1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2$ $+0 \times 2^1 + 1 \times 2^0$
0x1D	**0x**, base **16**	Hexadecimal	$1 \times 16^1 + 13 \times 16^0$
035	**0**, base **8**	Octal	$3 \times 8^1 + 5 \times 8^0$

Java unsigned integer, floating point and boolean primitive types

Name	Bytes	Type	Range	Literal samples
`char`	2 / **16 bit**	Unsigned integer	$\left[0, 2^{16} - 1\right]$	'A', '*', '-', 'Ç', '⇶' ...
`float`	4	Floating point	$\pm 1.18 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$	3.14f, 0.022f, -3.4E-24f
`double`	8	Floating point	$\pm 2.23 \times 10^{-308}$ to $\pm 1.8 \times 10^{308}$	3.14, 0.022, -3.4E-24, 3.14d,...
`boolean`	?	Logical value	not applicable	`true`, `false`

Followup exercises

6. Literal samples
7. Literals of type int
8. Integer overflow
9. Strange sum result
10. Correcting the error

Overview

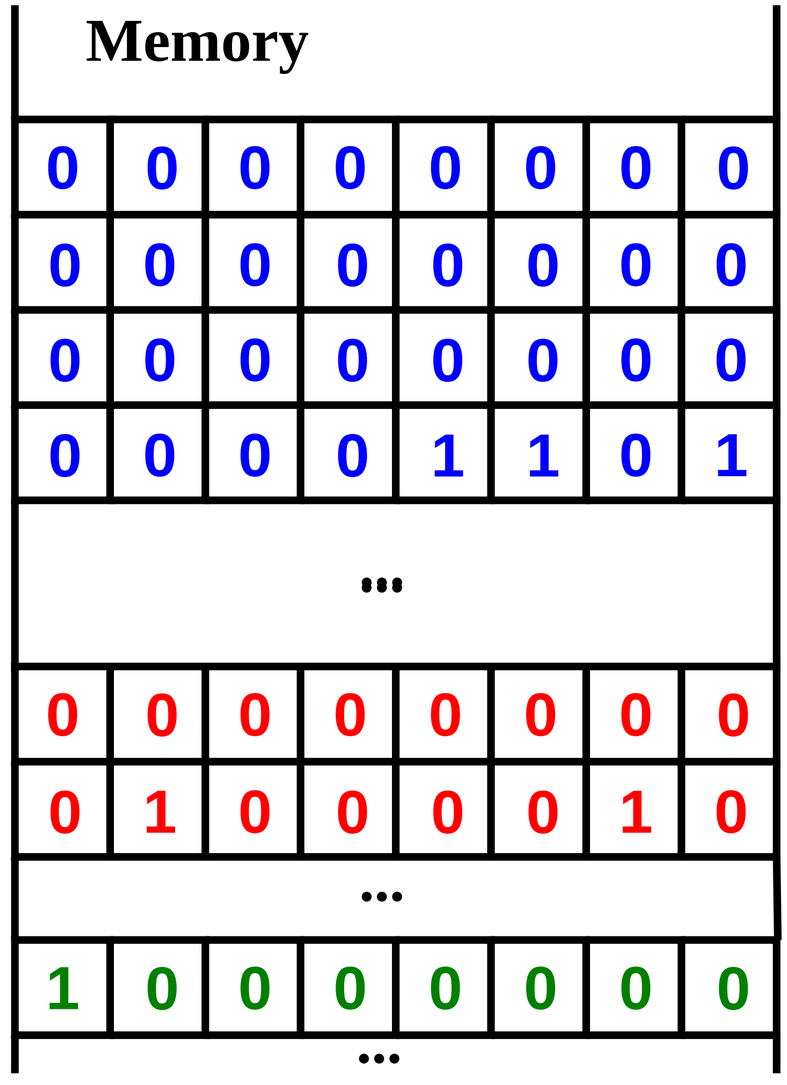
Language Fundamentals

- Variables

Variables: Handles to memory

Code:

```
int a = 13;

char c = 'B';

byte b = -128;
```



Local variable declaration

```
// Variable declaration:
//     Variable's type is double
//     Variable's name is «pi» (identifier)

double pi;
```

```
{type name} {variable name} ;
```

Declare, assign and use

```
double pi;       // Variable declaration

pi = 3.1415926; // Assigning value to variable

// Print a circle's area of radius 2.0

System.out.println(pi * 2.0 * 2.0);
```

Combining declaration and initialization

Separate declaration and initialization

```
double pi;       // Declaration of variable p
pi = 3.1415926; // Value assignment
```

Combined declaration and initialization

```
double pi = 3.1415926;
```

Compound declarations

Separate declarations	Equivalent compound declaration
`int a;` `int b = 22;` `int c;`	`int a,` `    b = 22,` `    c;`

Identifier in Java™:

- Variable names.
- Class names
- Method names, e.g.:

```
public static void main(String [] args)
```

Identifier name examples:

Identifier rules	Legal	Illegal
• Start with a letter, “_” or “$” • **May** be followed by letters, digits, “_” or “$” • Must not match a reserved keyword or literal	• `$test` • `_$` • `count` • `blue`	• **`2sad`** • **`switch`** • **`true`**

Java™ keywords.

Reserved keywords

```
abstract  char      else     if          long       return       this      while
assert    class     enum     goto        native     short        throw     _ (underscore)
boolean   const     extends  implements  new        static       throws
break     continue  final    import      package    strictfp      transient
byte      default   finally  instanceof  private    super        try
case      do        float    int         protected  switch       void
catch     double    for      interface   public     synchronized volatile
```

Reserved boolean and null literal values

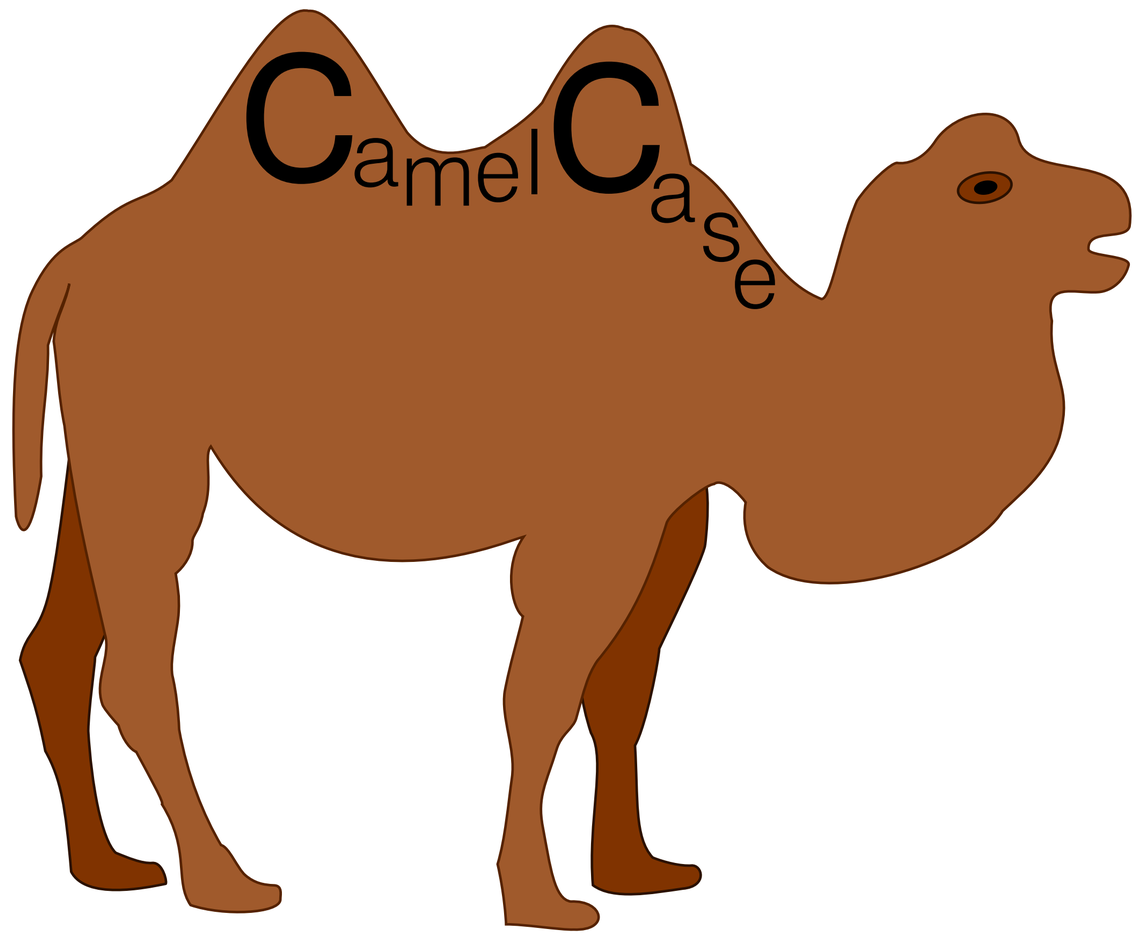
```
true   false   null
```

Contextual keywords

```
exports  non-sealed opens    provides  requires  to          uses  when  yield
module   open       permits  record    sealed    transitive  var   with
```

Variable naming conventions

- Start with a small letter like `africa` rather than `Africa`.
- Use “camel case” e.g. `myFirstCode`.



- Do not start with _ or $.

Constant variables

Modifier `final` prohibits changes:

```
final double PI = 3.1415926;
...
PI = 1.0; // Compile time error: Constant cannot be modified
```

Note

`final` variables by convention are being capitalized

Case sensitivity

Variable names are case sensitive:

```
int count = 32;
int Count = 44;
System.out.println(count + ":" + Count);
```

Resulting output:

```
32:44
```

Followup exercise

11. Legal variable names

Define before use

Correct:

```
double f;
f = -4.55;
```

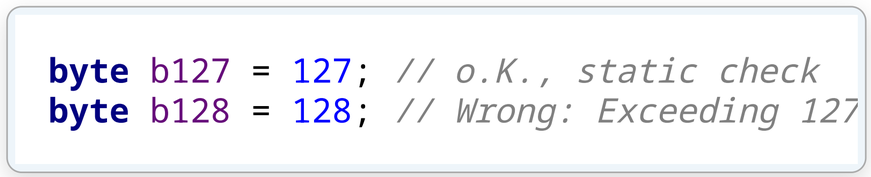
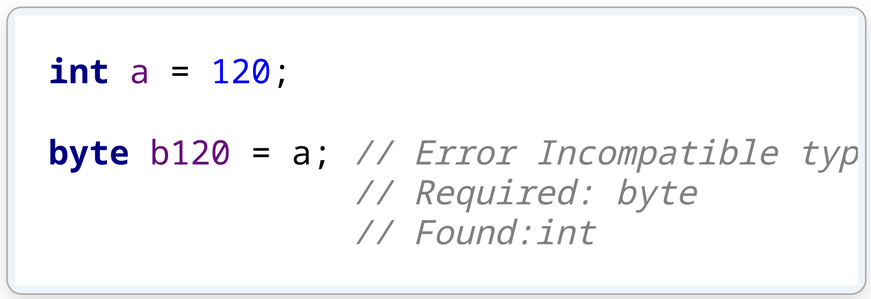
Wrong:

```
f = -4.55;
double f;
```

Type safety

```
int i = 2;
int j = i;   // o.K.: Assigning int to int
long l = i;  // o.K.: Widening conversion

i = l;       // Wrong: Narrowing

boolean b = true;
i = b;       // Error: int and boolean are incompatible types
i = 4.3345   // Error: Cannot assign double to int
i = "Hello"; // Even worse: Assigning a String to an int
```

Compile time analysis

	Performing static range check
	No static check on `int` variable's value

Followup exercise

12. Benefits of `final`

Type inference (JDK™ 10)

```
var count = 30; // o.K., type can be inferred from int literal

var index;      // Wrong: No way to infer type here.
index = 3;
```

Forcing conversions

```java
long l = 4345;

int i = (int) l; // Casting long to int
System.out.println("i carrying long: " + i);

double d = 44.2323;
i = (int) d; // Casting double to int
System.out.println("i carrying double: " + i);
```

```
i carrying long: 4345
i carrying double: 44
```

Watch out!

```java
long l = 3000000000000000L;
int i = (int) l;
System.out.println("i carrying long:" + i);

double d = 443000000000000.0;
i = (int) d;
System.out.println("i carrying double:" + i);
```

```
i carrying long:-296517632
i carrying double:2147483647
```

Casting long to int

```
long a = 3000000000000000L;
```

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1
0	1	1	1	0	1	0	0
1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	0
0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

Type cast `long` to `int`:

```
int b = (int) a;
```

Value: -296517632

1	1	1	1	1	0	1	1
1	1	0	1	1	1	1	0
0	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0

Don’t worry, be happy …

«C» programming language miracles:

```c
#include <stdio.h>

void main(void) {
  double measure = 65234.5435;
  short velocity;
  velocity = measure;
  printf("Velocity=%d\n", velocity);
}
```

Uups:

```
Velocity=-302
```

... and watch the outcome

Development costs:

~$7 billion.

Rocket and cargo

~$500 million.

related video and explanations

From the report

The cause of the failure was a software error in the inertial reference system.

Specifically, a **64 bit floating point number** relating to the horizontal velocity of the rocket with respect to the platform was converted to a **16 bit signed integer**.

The number was larger than 32,767, the largest integer possible in a 16 bit signed integer, **and thus the conversion failed**.

Related video explanation

Followup exercises

13. «C» vs. Java.
14. Assignment and type safety
15. Inventing `tinyint`.
16. An `int`'s minimum and maximum value

Dynamic typing in PERL

```perl
$test = 44;  # Assigning an int
print $test, "\n";

$test = "Jim"; # Assigning a string
print $test, "\n";

$cmp = 43.55; # A float

if ($test == $cmp) { # comparing string against float
   print "Equal\n";
} else {
   print "Different\n";
}
```

```
44
Jim
Different
```

Dynamic typing in PERL, part 2

```perl
$a = 2; # An integer
$b = 3; # Another integer

print '$a + $b = ', $a + $b, "\n";

$jim = "Jim";          # A string
print '$jim + $a = ' , $jim + $a, "\n";
```

```
$a + $b = 5
$jim + $a = 2
```

Using final

```
//Bad!
double pi = 3.141592653589793;
...
pi = -4; // Woops, accidential and erroneous redefinition
```

```
//Good
final double PI = 3.141592653589793;
...
PI = -4; // Compile time error:
         // Cannot assign a value to final variable 'pi'
```

Reference type examples

```
GpsPosition start = new GpsPosition(48.7758, 9.1829);
String name = "Simon";
LocalDate birthday = LocalDate.of(1990, Month.JULY, 5);
```

Overview

Language Fundamentals

➟ Literals

`float` and `double`

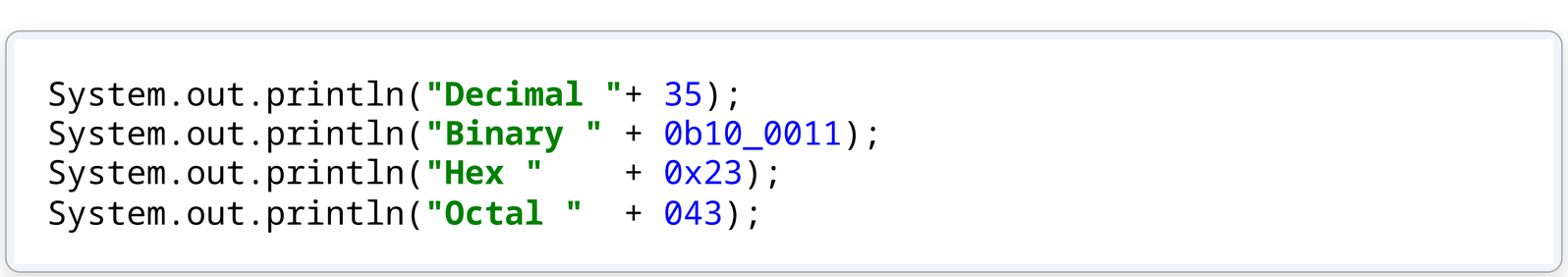
double d = 0.1;

0	0	1	1	1	1	1	1
1	0	1	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	0	1
1	0	0	1	1	0	1	0

float f = 0.1F

0	0	1	1	1	1	0	1
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	1

Four ways representing 35

Code	Result
 ``` System.out.println("Decimal "+ 35); System.out.println("Binary " + 0b10_0011); System.out.println("Hex " + 0x23); System.out.println("Octal " + 043); ```	 ``` Decimal 35 Binary 35 Hex 35 Octal 35 ```

Choose your output representation

```
System.out.println("35 as Binary (Base 2):       " + Integer.toString(35, 2));
System.out.println("35 as Ternary (Base 3):      " + Integer.toString(35, 3));
System.out.println("35 as Octal (Base 8):        " + Integer.toString(35, 8));
System.out.println("35 as Hexadecimal (Base 16): " + Integer.toString(35, 16));
```

results in:

```
35 as Binary (Base 2):       100011
35 as Ternary (Base 3):      1022
35 as Octal (Base 8):        43
35 as Hexadecimal (Base 16): 23
```

Followup exercises

17. Pretty may not be pretty
18. Strange output

Know your limits!

```
System.out.println(1000000000);    // o.K.
System.out.println(2147483647);    // o.K.: Largest int value 2^31 - 1

System.out.println(10000000000L);  // o.K.: Using type long
System.out.println(10000000000 );  // Compile time error: Integer number
                                   // larger than 2147483647 or
                                   // 2^31 - 1, Integer.System.out.println("i carrying double: " + i))
```

Literal examples

```
System.out.println("Hello");     // A String literal

System.out.println(33452);       // An int literal

System.out.println(34.0223);     // A double (floating point) literal

System.out.println(2147483648L); // A long literal
```

`int` literals

```
System.out.println("Value 1: " + 29);
System.out.println("Value 2: " + 0b11101);
System.out.println("Value 3: " + 0x1D);
System.out.println("Value 4: " + 035);
```

```
Value 1: 29
Value 2: 29
Value 3: 29
Value 4: 29
```

Followup exercises

19. Poor mans ASCII table
20. Integer value hexadecimal representation
21. Binary literals
22. Testing the limits (Difficult)
23. Why using braces in `System.out.println(...)`?
24. Composing strings of literals and variables
25. Escaping double quotes
26. Supplementary string exercises

Just kidding ...

```
int year = MMXIV; // Roman numerals representation

System.out.println("Olympic winter games: " + year);
```

Could this happen?

```
Olympic winter games: 2014
```

Overview

Language Fundamentals

➟ Arithmetic limitations

Strange things I

```
byte count = 91;    // o.K.

int i = 91;

byte count2 = i;    // Compile error: Incompatible types
                    // Required: byte Found: int

byte points = 130; // Compile error: Incompatible types
                    // Required: byte Found: int
```

Strange things II

```
final int i = 91;

byte count = i;    //  o.K.
```

Limited precision

```
float float2Power31 = Integer.MAX_VALUE + 1f; // 2^31

float floatDoubleMAX_VALUE = 2 * float2Power31 * float2Power31 - 1f; // 2^63 - 1

System.out.format( "   Float value: %f\n", floatDoubleMAX_VALUE);
System.out.println("Expected value: "   +  Long.MAX_VALUE);
```

Result:

```
   Float value:  9223372036854776000.000000
Expected value: -9223372036854775807
-----------------------------------------
    Difference:                   193
```

Nearest `float` to 2.1

```
//Print 15 fractional digits
DecimalFormat df = new DecimalFormat("#.###############");

System.out.println(df.format(Float.intBitsToFloat(
  0b0_10000000_00001100110011001100110
)));

// The smallest possible step above previous value
System.out.println(df.format(Float.intBitsToFloat(
  0b0_10000000_00001100110011001100111
)));
```

```
2.099999904632568
2.100000143051147
```

FloatConverter

IEEE 754 Converter (JavaScript), V0.21

	Sign	Exponent	Mantissa
Value:	+1	2^3	1.3875000476837158
Encoded as:	0	130	3250586
Binary:	☐	☑ ☐ ☐ ☐ ☐ ☐ ☐ ☐	☐ ☑ ☑ ☐ ☐ ☐ ☑ ☑ ☐ ☐ ☑ ☑ ☐ ☐ ☑ ☐ ☐ ☑ ☑ ☐ ☑ ☑ ☐ ☑ ☐

You entered: 11.1

Value actually stored in float: 11.100000381469726 5625

Error due to conversion: 3.814697265625E-7

Binary Representation: 01000001001100011001100110011010

Hexadecimal Representation: 0x4131999a

+1

-1

Overview

Language Fundamentals

➟ Conversions

Widening from `byte` literal to `short`

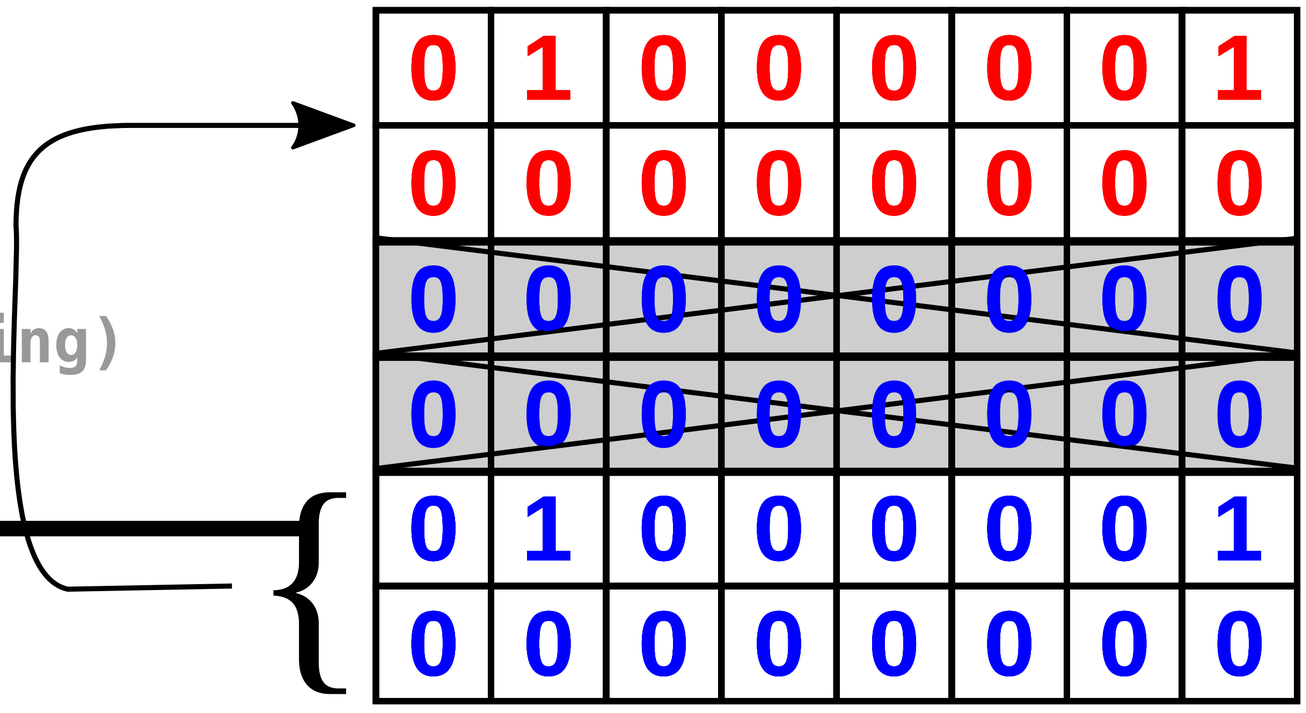
```
byte b = 42;

short s = b;

System.out.println(s);
```

0	0	0	0	0	0	0	0
0	0	1	0	1	0	1	0
0	0	1	0	1	0	1	0

```
Result: 42
```

Narrowing from `int` literal to `char` variable

```
System.out.println(65);

char c = 65; // Narrowing

System.out.println(c);


//Type cast char to int (widening)
System.out.println((int) c);
```



Result:

```
65
A
65
```

A widening “ladder”

```
byte   b = 42;  // Narrowing: constant int literal to byte
short  s = b;   // Widening
int    i = s;   // Widening
long   l = i;   // Widening
float  f = l;   // Widening
double d = f;   // Widening
```

A narrowing “ladder”

```
double d = 14.23;
float  f  = (float) d;  // Narrowing
long   l  = (long)  f;  // Narrowing
int    i  = (int)   l;  // Narrowing
short  s  = (short) i;  // Narrowing
byte   b  = (byte)  s;  // Narrowing
```

Followup exercises

27. `int` and `char`
28. `float` vs. `double`
29. `int` to `char` narrowing problems
30. Get a `byte` from 139
31. Ariane, I miss you!
32. Reducing `long` to `int` (difficult)

Overview

Language Fundamentals

- Operators and expressions
  - Arithmetic and logical operators

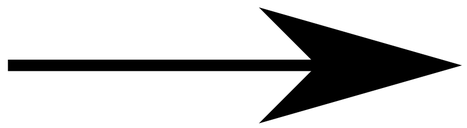
The binary plus operator

```
byte a = 4;

int result = a + 5;

System.out.println(result);
```

Operand
Type: `byte`

Operator

Operand
Type: `int`

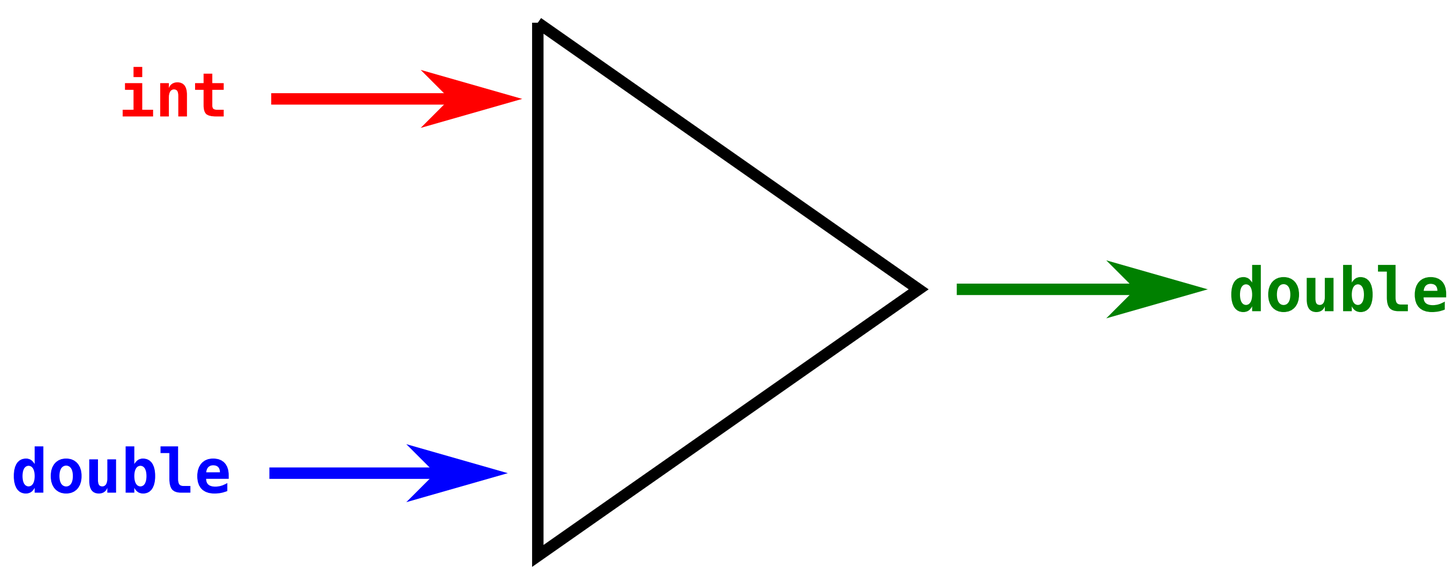


Output: `9`

Type: `int`

Binary operator output type

```
int a = 7;
double d = 2.3;
System.out.println(a + d);
```



Followup exercises

33. Calculating a circle's area
34. Dividing values
35. Strange things with operator ++
36. Adding values
37. Representational float and double miracles

Detecting arithmetic overflow (Java 8+)

```
try {
    int sum = Math.addExact(2147480000, 2147480000);
    System.out.println("sum = " + sum);
} catch (ArithmeticException ex) {
    System.err.println("Problem: " + ex.getMessage());
}
```

```
Problem: integer overflow
```

Dividing by zero

```java
double f = 34.3 / 0;
System.out.println("Value: " + f);
```

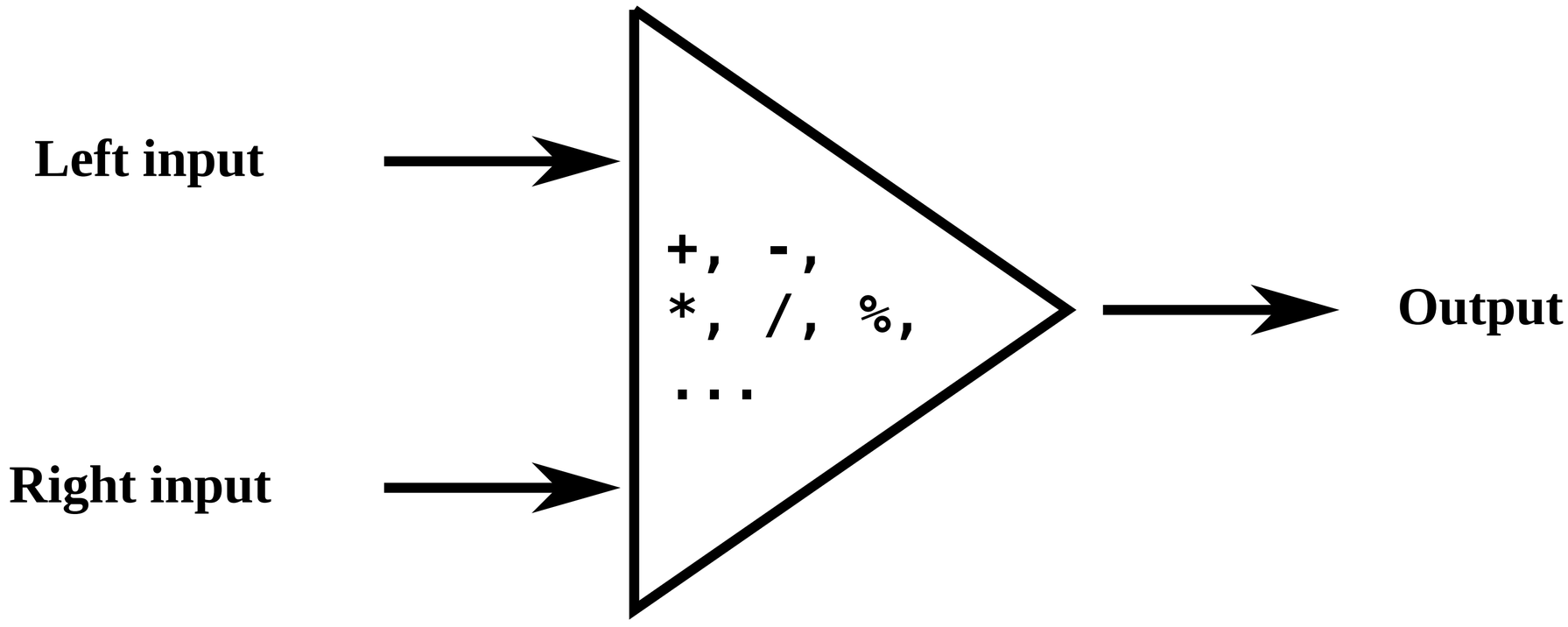
```
Value: Infinity
```

Watch out: “Silent” error!

Followup exercise

38. Expressions involving infinity

Generic binary operator



The modulus operator %

Get a division's remainder:

```
// Fair distribution of nuggets
// among gold diggers
int nuggets = 11,
    diggers = 3;

System.out.println("Nuggets per digger: " + nuggets / diggers);
System.out.println("Remaining nuggets: " + nuggets % diggers);
```

```
Nuggets per digger: 3
Remaining nuggets: 2
```

Binary operator type examples

Left	Right	Out	Examples
`boolean`	`boolean`	`boolean`	`|, &, &&, ||, ^`
`int`	`int`	`int`	+, -, *, /, % (read here!)
`int`	`long`	`long`	
`byte`	`byte`	`int`	
`double`	`float`	`double`	
`int`	`float`	`float`	
`char`	`byte`	`int`	

No binary + operator yielding `byte`

```
byte a = 1, b = 2;

byte sum = a + b; // Error: Incompatible types. (1)
                  // Required: byte
                  // Found: int
```

39. `int` to `short` assignment
40. `int` to `short` assignment using `final`
41. Calculating a circle's area avoiding accidental redefinition
42. Turning weeks into seconds
43. Turning seconds into weeks
44. Using predefined Java™ standard library constants
45. Converting temperature values
46. Time unit conversion
47. Interest calculation
48. Summing short and char

The logical “and” operator &

Boolean “and” of two operands:

```
boolean examSuccess = true,
        registered = false;

boolean examPassed = examSuccess & registered;

System.out.println("Exam passed:" + examPassed);
```

```
Exam passed:false
```

Followup exercise

49. Operator & vs. &&

Overview

Language Fundamentals

- Operators and expressions
  - Assignment operators

The += operator

Increment variable by right hand value:

```
int a = 4;

a = a + 2;

System.out.println("Value:" + a);
```

```
int a = 4;

a += 2;

System.out.println("Value:" + a);
```

```
Value:6
```

Followup exercise

50. Strange addition

The &= operator

Logical and operation:

```
boolean examSuccess = true,
        registered = false;

examSuccess = examSuccess & registered;

System.out.println(
  "Exam success:" + examSuccess);
```

```
boolean examSuccess = true,
        registered = false;

examSuccess &= registered;

System.out.println(
  "Exam success:" + examSuccess);
```

```
Exam success:false
```

Arithmetic assignment operators

		Example using %=
=	Assign right to left operand	
+=	Assign sum of operands to left operand	
-=	Assign difference of operands to left operand	
*=	Assign product of operands to left operand	
/=	Assign quotient of operands to left operand	
%=	Assign remainder of operands to left operand	

```
int value = 13;
value %= 5;
System.out.println("Result=" + value);
```

```
Result=3
```

Logical assignment operators

&=	Assign logical “and” of operands to left operand
| =	Assign logical “or” of operands to left operand

Followup exercise

51. Understanding +=

Overview

Language Fundamentals

- Operators and expressions
- Unary operators

Increment operator ++

Increment variable by 1:	Shorthand version:
`int a = 4;` `a = a + 1;` `System.out.println("Value:" + a);`	`int a = 4;` `a++;` `System.out.println("Value:" + a);`

```
Value:5
```

Different range behaviour!

Increment variable by 1:	Shorthand version:
`byte value = 127; // Max possible value` **`value = value + 1; // Error: Required type: byte`** **`// Provided:int`** `System.out.println(value);`	`byte value = 127; // Max possible value` **`value++; // o.K., will cycle through range`** `System.out.println(value);`
Does not compile	`Value:-128`

Cast required

Increment variable by 1:

```
byte value = 127; // Max possible value

value = (byte)(value + 1); // cast required,
                           // possible overflow

System.out.println(value);
```

Shorthand version:

```
byte value = 127; // Max possible value

value++; // cycle through range,
         // no cast required.

System.out.println(value);
```

```
Value:-128
```

Prefix and postfix notation

pre-increment		post-increment	
`int a = 4;` `int b = ++a;` `System.out.println(b);`		`int a = 4;` `int b = a++;` `System.out.println(b);`	
Output:	5	Output:	4

Followup exercise

Operator examples

Shorthand	Operation	Explicit
`//Integer operations` `i--;` `i += k;` `i %= 3;` `// boolean` `  - nothing appropriate -`	`Decrement by one` `Raise by k's value` `assign modulus of 3` `Switching true <--> false`	`//Integer operations` `i = i - 1;` `i = i + k;` `i = i % 3;` `// boolean` `b = !b;`

Followup exercises

Overview

Language Fundamentals

➟ Comments

Java™ comment flavors

Multi line comment:

```
int a;
/*  We define a variable. Then
    subsequently a value is being assigned */
a = 33;
```

End of line comment:

```
int a;  // We define a variable.
a = 33; // Then subsequently a value is being assig
```

Inline comments

```
int strength = a /* fixed value */ + b /* age */ + c /* courage */;
```

being run-time equivalent to:

```
int strength = a + b + c;
```

Javadoc™ comments

```
/**
 * Describing rectangles. (1)
 */
public class Rectangle {

  /**
   * Setting a rectangle's width.
   *
   * @param width The rectangle's new width. (2)
   *
   *
   */
  public void setWidth(double width) {
      // Implementation yet missing
  }
  ...
}
```