Modifying Variables

```
IDL> x=3
IDL> y=x*2
IDL> x=4
```

What is `y` now?

A) 3

B) 4

C) 6

D) 8

E) None of the above / I don't know

Indexing

```
IDL> y = [1, !pi, 6, 2, 12]
```

Evaluate: `print,y[2]*y[3]`

A) 12.0000

B) 6 pi = 18.8496

C) pi = 3.14159

D) 24.0000

E) I don’t know

Plotting

- Reminder: `indgen` and `findgen` make arrays from `0` to `n-1`
- Plotting commands look like:

```
IDL> x=indgen(10)
IDL> y=2*x
IDL> plot,x,y
```

Do you have the textbook?

A) Yes

B) No, haven’t tried to get it

C) No, bookstore didn’t have it

D) No, other

Reading

How is the reading going?

A) I haven’t started yet

B) I did it all

C) I started, but it was too much

D) I started, and it seems reasonable

E) [anything else]

Exercise

How is the first exercise going?

A) I haven't started

B) I've started

C) I'm done, it was easy

D) I'm done, but it took forever

E) [anything else]

Remote Access

- There are instructions on the web site for getting remote access via mac/unix and windows
- Also “getting IDL” instructions

ASTR2600_class_materials / **handouts** /

- ClassSurvey.pdf
- Syllabus2013s.pdf
- XmingPuTTY.pdf
- idllicense.pdf
- sshaccess.pdf

Chapter 2: Data Types

Chapter 0: Binary

- We’re skipping Chapter 0, but need some of it now
- All numbers, letters, etc. are stored in binary on computers: they are sets of bits, which are 1’s and 0’s
- A “byte” is 8 bits: e.g., `0000 0000`

Binary representation

* `0000 0000 = 0`
* `0000 0001 = 1`
* `0000 0010 = 2`
* `0000 0011 = 3`
* `0000 0100 = 4`
* etc.

Binary Arithmetic

- 0+0=0
- 0+1=1
- 1+0=1
- 1+1=0, carry the 1

Binary representation

* `0000 0000 = 0`
* `0000 0001 = 1`
* `0000 0010 = 2 = 1+1`
* `0000 0011 = 3 = 1+2`
* `0000 0100 = 4 = 2+2`
* etc.

Binary Arithmetic

Evaluate:

```
 0000 0011
+0000 0101
```

A) 0000 0111

B) 0000 0110

C) 0000 1111

D) 0000 1000

E) None of the above / I don’t know

Binary Arithmetic

Evaluate:

```
 0000 0011 = 3
+0000 0101 = 5
```

A) 0000 0111 = 7

B) 0000 0110 = 6

C) 0000 1111 = 15

D) 0000 1000 = 8

E) None of the above / I don't know

Binary Overflow

- What about 252 + 6:

```
 1111 1100
+0000 0110

1 0000 0010 ??
```

- But there are only 8 bits, you can't have a 9-bit number.
- `0000 0010 = 2`, so 252+6 = 2!

Binary Overflow

- Similar to clocks: time can only be represented in HH:MM:SS
  - 02:59 + 5 minutes = 3:04
  - 12:55 + 10 minutes = 1:05

Two's Complement

- Revisit our overflow example:
- `1111 1100 = x`

  `+0000 0110 = 6`

  `0000 0010 = 2`

  `x + 6 = 2,` what is `x`?

Two's Complement

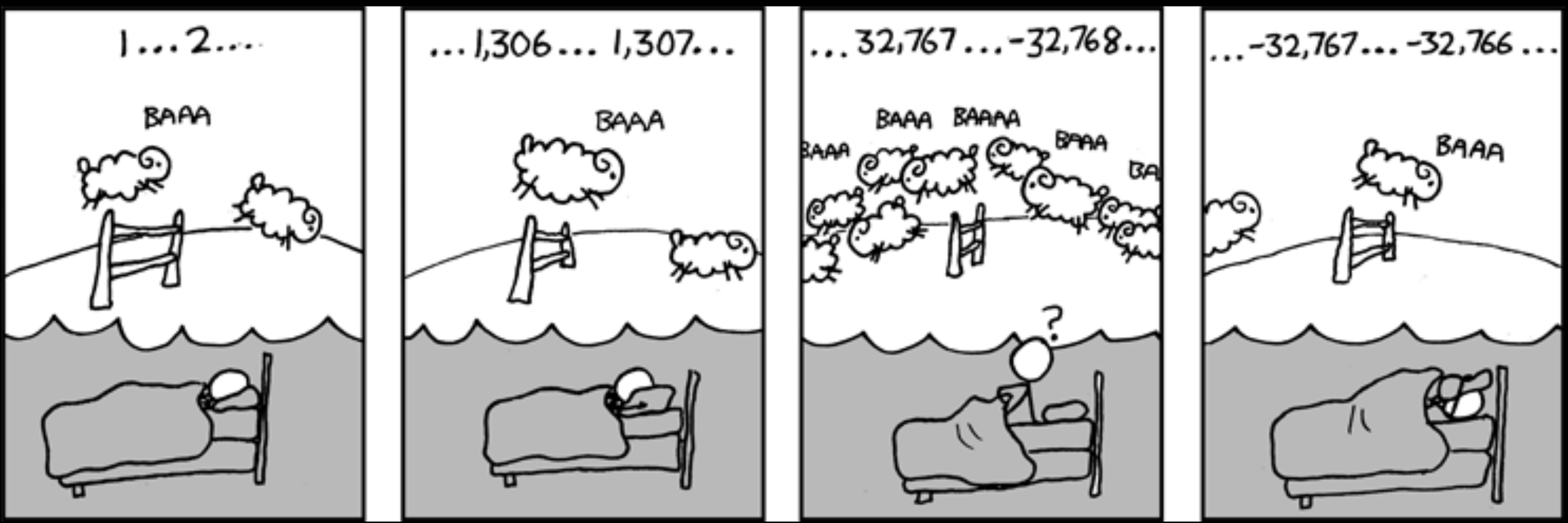
- `x + 6 = 2`, therefore `x=-4`
- `1111 1100 = -4`

  `+0000 0110 = 6`

  `0000 0010 = 2`

"two's complement" is an alternative interpretation of the same number

Two’s Complement



Two’s Complement

- 1### #### are negative numbers
- 0### #### are positive numbers

Two's Complement

Two's Complement numbers range from `1000 0000` to `0111 1111`. What is the corresponding range of `integers?`

A) `1,111,111 : 10,000,000`

B) `0 : 255`

C) `1 : 256`

D) `-128 : 127`

E) None of the above / I don't know

Why Two's Complement?

- Need some way to represent negative numbers
- But, not all data types have them

Integer Data Types

- byte : 0–255
- short : -32678-32677
- long : -2147483648-2147483647

```
IDL> print,2b^8b,2b^8b-1b
   0 255
```

```
IDL> print,2s^15,2s^15-1
  -32768   32767
```

```
IDL> print,2L^31L,2L^31L-1L
 -2147483648  2147483647
```

Integer Data Types

- Even though our example was -128 : 127, there is no “signed byte” type
- There ARE *un*signed long/short:

```
IDL> print,2US^15US,2L^15US-1US,2US^16US-1,2US^16US
   32768       32767   65535       0
IDL> print,2UL^31UL,2L^31UL-1UL,2UL^32UL-1,2UL^32UL
  2147483648  2147483647  4294967295          0
```

Types and Issues

Evaluate: `print,16b*16b*2b`

A) 0 type Byte

B) 256 type Byte

C) 512 type Byte

D) 512 type Integer

E) I don’t know / None of the above

Types and Issues

Evaluate: `print,16b*16b*2b`

A) 0 type Byte Correct: `16b*16b = 0b, 0b*2b=0b`

B) 256 type Byte Not possible

C) 512 type Byte Not possible

D) 512 type Integer What it probably “should” be

E) I don’t know / None of the above

Promotion

- We just saw that `16b*16b*2b = 0`
- What is `16b*16s*2b`?

Promotion

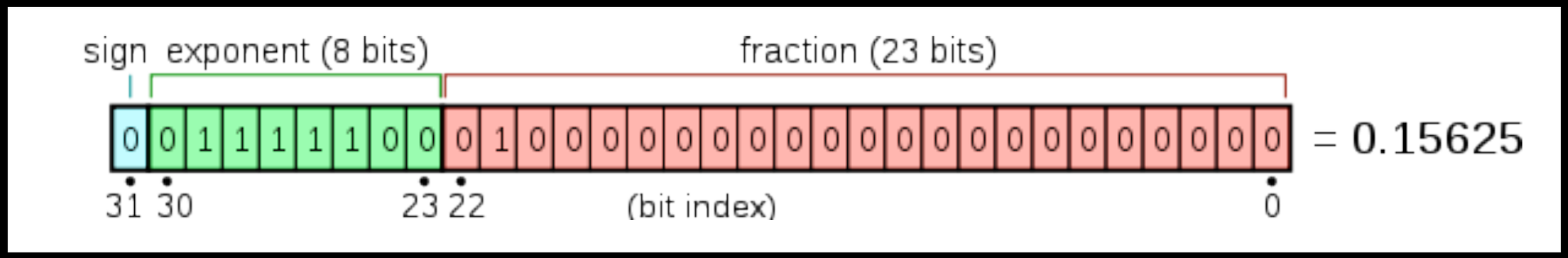
- We just saw that `16b*16b*2b = 0`
- What is `16b*16s*2b`?
  - `16b*16s*2b = 512s`
  - Numbers get “promoted” to the type highest number of bytes...
  - BUT, `16b*16b*2s = 0`! Evaluation is left-to-right

Order of Promotion

- byte (`B`) — 8 bits
- short int (`S`, `US`) — 16 bits
- long int (`L`, `UL`) — 32 bits
- `long64` — 64 bits
- float (`E`)
- double (`D`)

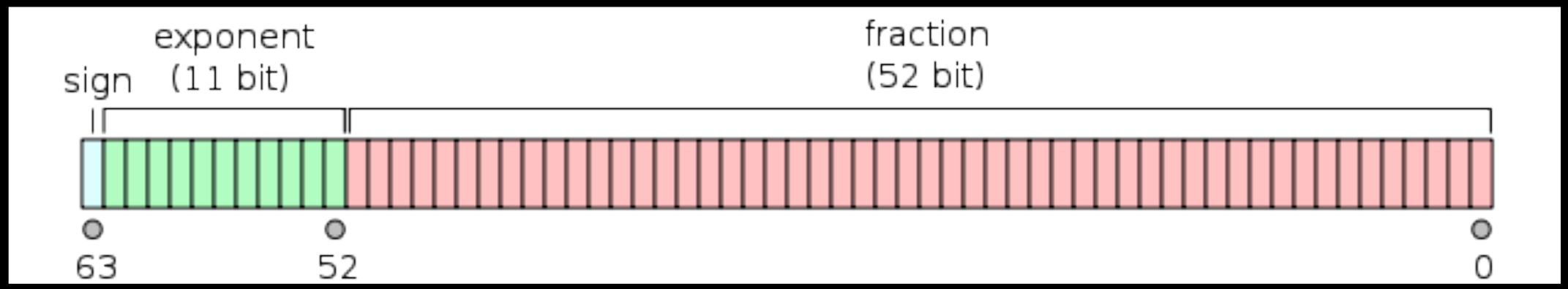
Floating Points

- Binary representation of decimals
- A “float” has 32 bits (4 bytes)
- Has about 7 digits of accuracy
- Can represent numbers from `1.2e-38` to `3.4e38` (positive or negative)



Double Precision

- 64 bits
- 15 digits of accuracy
- `1e-308` to `1e308` (positive or negative)



Declaring Floats and Doubles

```
IDL> help,5.
<Expression>    FLOAT     =       5.00000
IDL> help,5e0
<Expression>    FLOAT     =       5.00000
IDL> help,0.5e1
<Expression>    FLOAT     =       5.00000
IDL> help,5.0
<Expression>    FLOAT     =       5.00000
IDL> help,.5e1
<Expression>    FLOAT     =       5.00000
IDL> help,5*1e0
<Expression>    FLOAT     =       5.00000
IDL> help,5e
<Expression>    FLOAT     =       5.00000
IDL> help,50e-1
<Expression>    FLOAT     =       5.00000
```

Floating Point Difficulties

- You can lose data when adding/ subtracting numbers

```
IDL> x=1e6
IDL> y=1e-6
IDL> z=x+y
IDL> print,x,y,z
   1.00000e+06  1.00000e-06  1.00000e+06
IDL> print,z-x
       0.00000
```

Floating Point Arithmetic

- There are some really scary features of floating point arithmetic
  - finite precision means there may be “rounding error” in any given number
  - Usually minor, but can cause domain errors

```
IDL> print,0.0314,0.314159,0.314159-0.314
     0.0314000      0.314159  0.000158995
IDL> print,314159L-314000L
           159
```

Promotion Again

Evaluate: `16D * 16B * 2B`

A) 0 type Byte

B) 0 type Double

C) 512 type Integer

D) 512 type Double

E) I don’t know / None of the above

Strings

- Characters and words
  - e.g. `“Hello”, “t”`
- Each letter is 1 byte
- You can add strings:

```
IDL> x="hello"
IDL> y=" "
IDL> z="there"
IDL> print,x+y+z
hello there
```

Strings

- You can’t add strings and numbers:

```
IDL> print,"five "+5
% Type conversion error: Unable to convert given STRING to Integer.
% Detected at: $MAIN$
      5
```

- But you can convert numbers to strings:

```
IDL> print,"five "+string(5)
five          5
```

String Manipulation

- Extract substrings

```
IDL> print,strsplit("Item1,Item2,Item3",",")
           0           6          12

IDL> print,strpos("Hello there","there")
           6
IDL> print,strlen("there")
           5
IDL> print,strmid("Hello there",6,5)
there
```

String Manipulation

- Extract substrings

```
                    0123456123456
IDL> print,strsplit("Item1,Item2,Item3",",")
           0           6           12

                    0123456
IDL> print,strpos("Hello there","there")
           6
                    12345
IDL> print,strlen("there")
           5
IDL> print,strmid("Hello there",6,5)
                    01234512345
there
```


WHENEVER I LEARN A NEW SKILL I CONCOCT ELABORATE FANTASY SCENARIOS WHERE IT LETS ME SAVE THE DAY.
OH NO! THE KILLER MUST HAVE FOLLOWED HER ON VACATION!
BUT TO FIND THEM WE'D HAVE TO SEARCH THROUGH 200 MB OF EMAILS LOOKING FOR SOMETHING FORMATTED LIKE AN ADDRESS!
IT'S HOPELESS!
EVERYBODY STAND BACK.
I KNOW REGULAR EXPRESSIONS.
PERL!
TAP TAP

Regular Expressions

- Advanced string manipulation
- A language in themselves
- Not a main topic in this course, but absurdly useful if you want to try yourself
  - e.g. Find the word between “quick” and “fox”:

```
IDL> s = "the quick brown fox jumps over the lazy dog"
IDL> r = stregex(s,"quick (.*) fox",/extract,/sub)
IDL> print,"Whole match: ",r[0]
Whole match: quick brown fox
IDL> print,"Between quick & fox: ",r[1]
Between quick & fox: brown
```

Variable Declaration

- “Declaration” means specifying the type of a variable before specifying its value
- IDL and python do not require declaration
- C++ and others DO
  - Not declaring variables is easier, but makes mistakes harder to catch

```
IDL> massSum = 5
IDL> print,massum
% PRINT: Variable is undefined: MASSUM.
% Execution halted at: $MAIN$
```