

Goals for This Lecture:

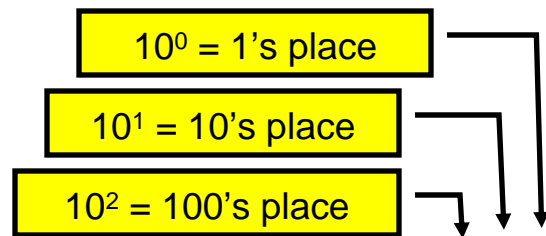
- Understand how integers are represented in binary form
- Understand what a bit, byte, & word are
- Understand the types of data stored in memory and/or disk: integer, real, complex, logical, and character
- Understand the basic form of FORTRAN statements
- Understand the basic form of a FORTRAN program

Representing data digitally

- Memory is composed of millions/billions of individual switches which are all in a state of being either ON or OFF
 - A switch can represent only the numbers 0 or 1
 - Each switch is called a *bit*
- To store larger numbers we group switches together and store numbers in binary form where each digit is either 1 or 0
- Usually the smallest grouping of numbers is 8 bits which is known as a *byte*
- Usually bytes are grouped together into 16 bit (2 bytes), 32 bit (4 bytes), 64 bit (8 bytes), or 128 bit (16 byte) *words*
 - Some computers don't use byte groupings and directly form groups 64 bits together into a word
 - Occasionally one finds odd sorts of machines that use groupings like 36 bit words

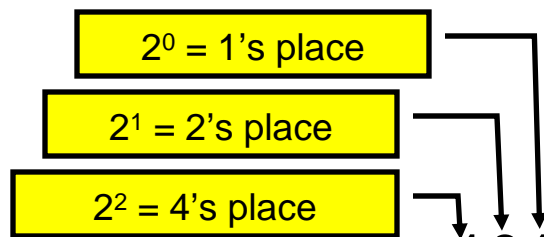
Binary Data Representation

- Base ten (decimal) representation: each digit multiplies a power of 10:



$$137_{10} = 1 \times 10^2 + 3 \times 10^1 + 7 \times 10^0$$

Base two (binary) representation: each digit multiplies a power of 2:



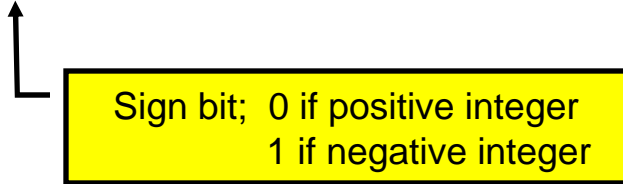
$$\begin{aligned} 101_2 &= 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \\ &= 5_{10} \text{ (in base 10 representation)} \end{aligned}$$

Binary Integers

- One byte of data can be used to positive store integers up to:

0	1	1	1	1	1	1	1
---	---	---	---	---	---	---	---

 $= 1 \times 2^6 + 1 \times 2^5 + 1 \times 2^4 + 1 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 = 127_{10}$



- Negative integers are stored via two's complement representation
- Complement the absolute value of the number (change zero's to one's and one's to zero)
- Add one to the complemented number to find the two's complement
- Therefore the value of -127_{10} is stored in two's complement form as:

1	0	0	0	0	0	0	1
---	---	---	---	---	---	---	---

- While the two's complement form may seem a bit weird it offers a big advantage in the design of ALU circuits:
 - To subtract one number from another one can form the two's complement of the first number and add it to the second number.
- In general a N-bit integer can store numbers in the range of $-2^{N-1} - 1$ to 2^{N-1}
- In the case of a two byte integer this range is -32768_{10} to 32767_{10}
- In most cases, FORTRAN integers are 4-bytes (unless otherwise specified) which gives a integer range of $-2^{31} - 1$ to 2^{31}

Other Representations

- There are a few other representations that are convenient on occasion:
 - Octal (Base 8) with possible values for each digit of:
 - 0,1,2,3,4,5,6,7
 - Used on computers that work in words
 - Hexadecimal (base 16) with possible values for each digit of:
 - 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
 - Used on computers that work in bytes
 - Each byte can be represented by two hexadecimal digits
 - Often used in setting RGB color combinations for web pages.
 - $127_{10} = 01111111_2 = EF_{16} = 241_7$
- Note: From this point forward omitted subscripts will indicate decimal (base 10) numbers

Types of data stored in memory and/or disk

- Integer data
 - Numbers such as 0, -343, 7714523
- Real data
 - Numbers such as 0.0, 3.14, 5.236×10^{23}
- Complex data
 - Numbers such as $2.2+3.4i$
- Logical data
 - Values of “true” & “false”
- Character data
 - Common characters found on the keyboard

Integer data

- Integers such as 48, -343, 7721559, etc.
- Positive & negative integers
- Stored in 1, 2, 4, or 8 byte form
- Smallest numbers that can be stored by n-bit integer is:
 -2^{n-1}
- Largest number that can be stored by n-bit integer is:
 $2^{n-1} - 1$

Complex data

- Complex numbers such as:
 $2.42+5.5i$, $3.4 \times 10^{-5} - 0.004i$
- Usually stored in 8, or 16 byte form as pairs of “real” or “floating-point” data types

Logical data

- Logical values of “true” & “false”
- Stored as 1, 2, or 4-byte variables
- Usually stored as integer values of “1” or “0”

Character data

- Character data
- For english:
 - 26 upper-case letters (A-Z)
 - 26 lower-case letters (a-z)
 - 10 digits (0-9)
 - common symbols such as “:;()[]{}!~@#\$\$%^&*.
- Stored as 1-byte w/ numerical value determined by convention
 - ASCII (American Standard Code for Information Interchange)
 - Found in appendix A of Chapman
 - Other schemes:
 - EBCDIC (Extended Binary Coded Decimal Interchange Code)
 - Previously Used by IBM on mainframe systems
 - Unicode
 - 2-byte system
 - International (any language can be used)

The FORTRAN Character set

- The following 84 characters are valid for use in a FORTRAN program:
 - Upper-case letters: A-Z
 - lower-case letters: a-z
 - 10 digits: 0-9
 - Underscore character: _
 - Arithmetic symbols: + - * /
 - Sixteen miscellaneous symbols: (). = , ' \$: ! " % & ; < > ?
 - Blank space (a.k.a. white space)
- Chapman claims 86 characters but arithmetic is incorrect (and “**” is not it’s own symbol just “*” repeated twice)
- FORTRAN is **not case sensitive**
 - upper or lower case characters in statements are treated the same

FORTRAN Statements

- Free form
 - Statements can be entered anywhere on a line with each line up to 132 characters long
 - Statements that take more than one line can be continued to the next line by placing a continuation character “&” at the end of each incomplete line.
 - A statement can be labeled (rare in modern FORTRAN) by a number at the beginning of the statement with a value of 1-99999
 - Any characters following a ! (unless it is inside a pair of quotes) are considered *comments* and are ignored by the compiler
 - Files usually named in the form file.f90 or file.f95
- Fixed form (historical carryover from punch-card days):
 - Statements must begin after column 6 and end by column 72
 - A character in column 6 indicates a continuation of the previous statement
 - Columns 1-5 can be used for numerical statement labels in the range 1-99999
 - A “c” in column 1 indicates a comment statement (ignored by compiler)

Structure of a FORTRAN program

- Every program must have the following three sections (in this top-down order):
 - Declaration section
 - Consists of a group of non-executable statements at the beginning, i.e. top, of the program that define the name of the program and any variables the program may use
 - Execution section
 - Consists of statements describing the actions to be taken by the program
 - Termination section
 - Consists of statements terminating execution of the program and telling the compiler that the program is complete

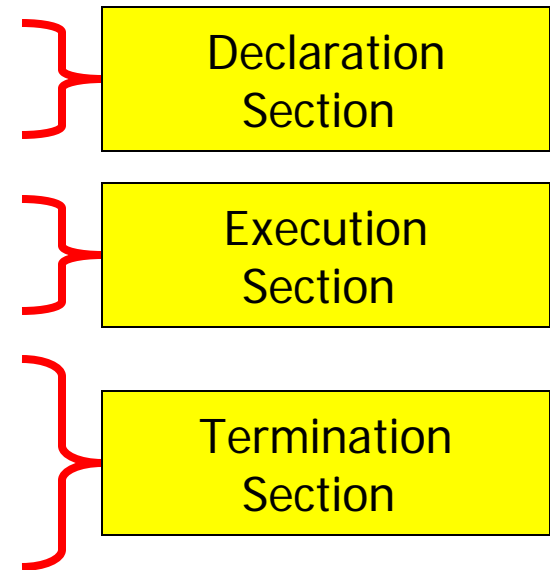
The “Hello World” FORTRAN program

```
program hello_world
```

```
write(*,*) “Hello World!”
```

```
stop
```

```
end program hello_world
```



A More Complex Example

```
! Purpose: multiply some numbers together
! Author:  F. Douglas Swesty
! Date:    9/7/2005
program my_first_program

integer :: i,j,k ! Declare three integer variables

i = 5           ! Assign a value of 5 to i
j = 3           ! Assign a value of 3 to j

k = i*j        ! Multiply I times j and assign value to k

write(*,*) " k = ",k

stop           ! Stop execution of the program

end program my_first_program
```

Assignments

- Read Sections 2.4-2.6, 2.10 of Chapman (FORTRAN 90/95)

Constants & Variables

- FORTRAN Constant: A data object that is defined before the program is executed and whose value never changes
- FORTRAN Variable: A data object that can change it's value during the execution of the program
- Five intrinsic, i.e. built-in, types of variables & constants in FORTRAN:
 - integer
 - real
 - complex
 - logical
 - character

Declaration of Variables

- Variables should always be explicitly declared in a declaration statement

integer :: var1, var2, var3, ...

real :: var1, var2, var3,...

complex :: var1, var2, var3,...

logical :: var1, var2, var3, ...

character :: var1, var2, var3, ...

- Variables that are not declared are implicitly typed:
 - Undeclared variables whose names begin with i-n are considered to be integers, all others are considered to be reals
 - NEVER, EVER make use of this feature!
 - Always turn this feature off with an **implicit none** statement

Variable Naming

- Valid variable names:
 - Can contain up to 31 characters
 - Valid characters are the characters a-z, A-Z, 0-9, and the underscore “_”
 - Names must begin with an alphabetic character
- Good programming practice:
 - Pick informative names for your variables
 - `temperature`, `density`, `velocity` are better names than `x1`, `x2`, `x3`