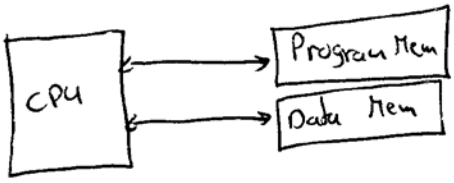
# PIC24F Memory Organization




Note: Harvard Architecture.

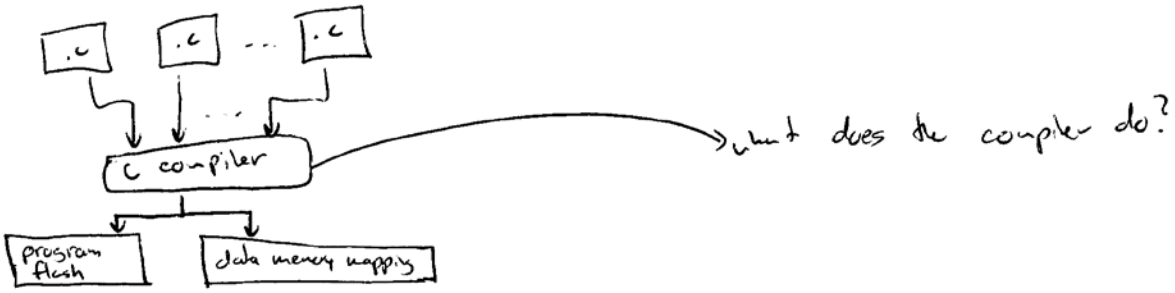
## Program Memory:

- Reset Address
- Device Configuration
- Interrupt Vector Tables
- User Flash Program Memory (FLASH) → non-volatile
  - ↳ generated by compiler

## Data Memory:

- Special Function Registers
- Data memory (RAM) → volatile
  - ↳ initialized by program
  - ↳ mapping of memory locations to variables, stack, heap done by compiler

## C Compiler:



Program Memory Sections:

- .text → software routines (defined by user or library code)
- .dinit → code to initialize data
- .isr → interrupt service routines (default routines)
- .const → constant values defined in C programs (and lib code) that are stored in program memory (for non-volatile)
- .prog → user defined regions of program memory.

**PIC C compiler:**

- uses .prog for user defined reserved regions of memory
- uses .text for user defined C code (including interrupts)
- uses .text for lib code
- uses .isr for default interrupt (i.e. programed into ivt for all interrupts not defined by user)

Data Memory Sections:

- .ndata → like data but near
- .bss → statically allocated variables that are not initialized (e.g. int count;)
- .data → statically allocated variables that are initialized (e.g. char a = 'h';)
- .dconst → constant values defined in C program (eg. text strings, const keyword)
- .heap → dynamically allocated data

What is **near** data?

- data located between 0x0000 and 0x1FFF.
- can be directly addressed using address field (13-bits) in instructions
  - eg. first 2kB of near memory are the SFR

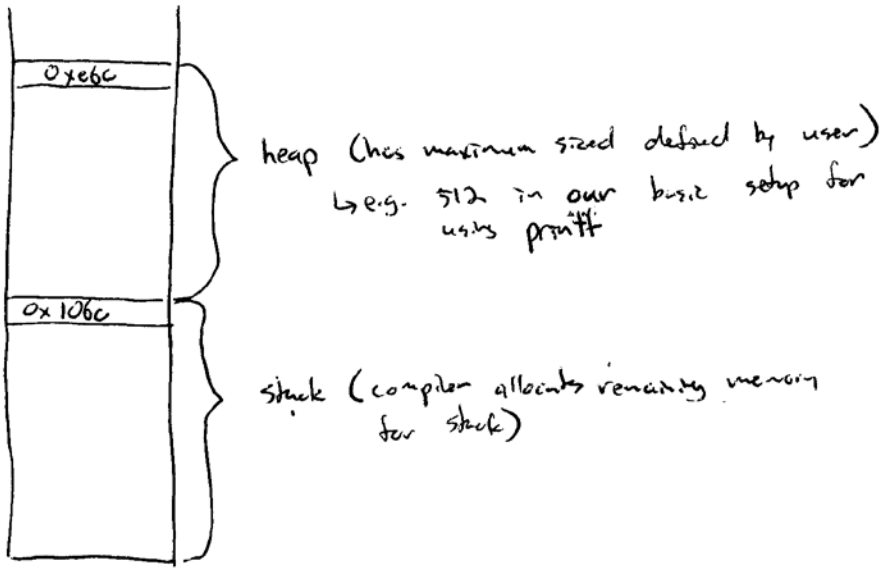
Heap: Region of memory used for dynamically allocated data

↳ e.g. malloc, calloc, free, realloc

Stack: Used for function calls and variables declared within function calls.

↳ on PIC, stack grows upward (increasing addresses)

Data Memory:




what happens on stack overflow?

↳ potential security problems (or at least corrupt data values)

SPLIM - Stack pointer limit value register

↳ set maximum value for stack

↳ calls stack error trap if push address exceeds SPLIM

↳ compiler can provide stack error trap

↳ but you can create one as well _stackError