Data Memory

Embedded Software Essentials

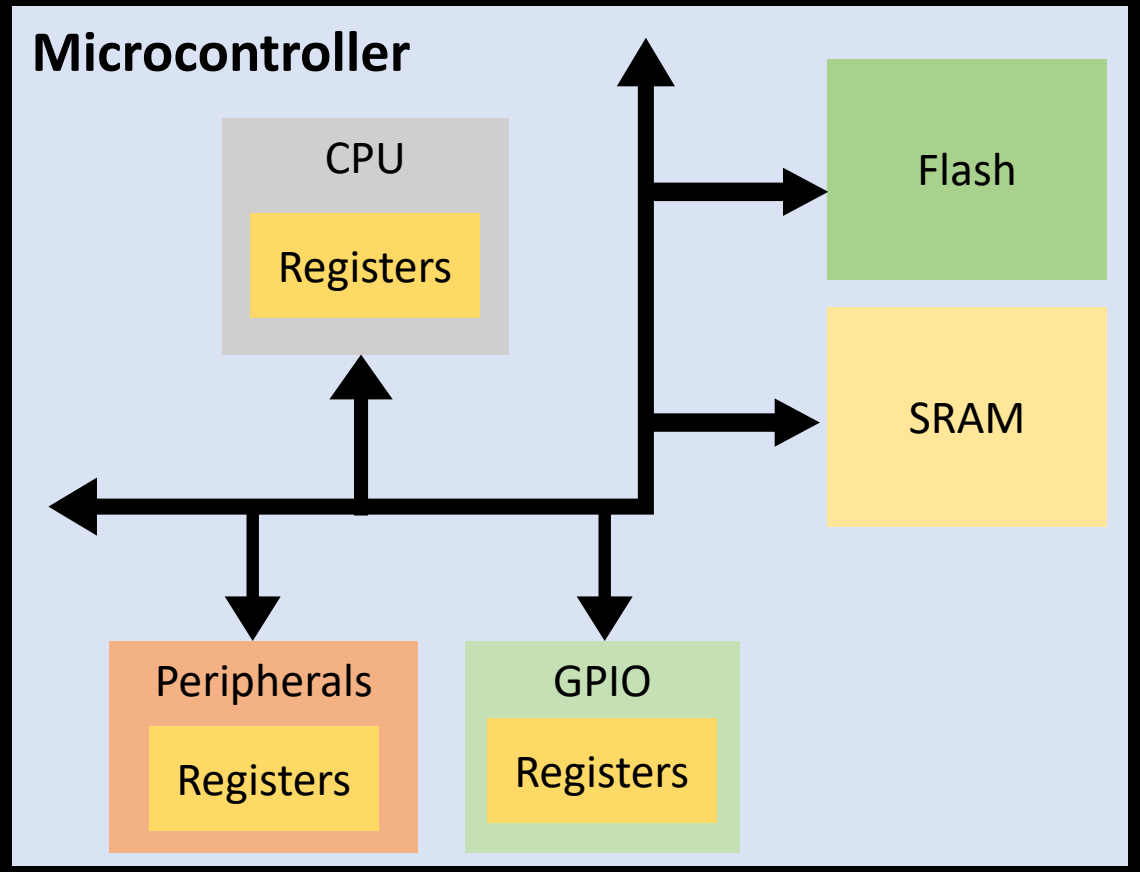
C1M3V4

Memory

- Three Main Types of Memory
  - Flash (non-volatile)
  - RAM (volatile)
  - Registers (volatile)

- Compilation tracks and maps memory needs program code and data into segments
  - Code Segment
  - Data Segment

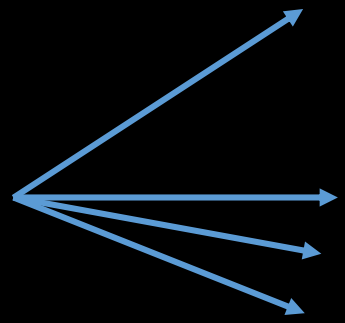
→ **Specified in the Linker File**



What is Data Memory?

- Data memory stores our program's operands

C-Program variables get stored in Data Memory



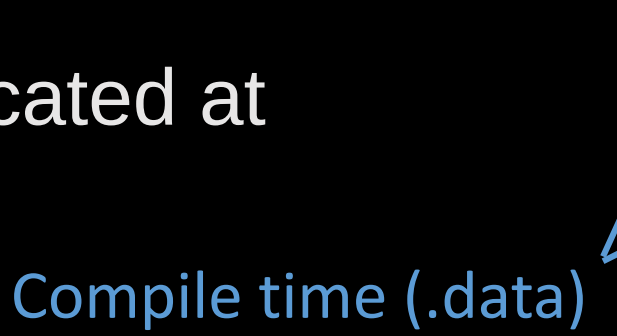
```c
int varA;

int main(){
  int varB = 6;
  int varC = 12;
  int * varB_p = &varB;

  varA = varC + *varB_p;
  return 0;
}
```

- Data gets **loaded** into registers, and **stored** back into memory

Data Allocation

- Data represents more than just variables declared
  - Can be allocated with linker or dynamically at runtime

- Data memory can be allocated at
  - Compile Time
  - Runtime




```
int varA = 2;

int main(){
  static int varB = 6;
  int varC = 12;

  /* More Code Here */
  return 0;
}
```

Many characteristics of data determine how to be mapped into the Data Segment

Data Segment

- The Data Segment is a container for the various types of allocated data that get mapped into physical memory

- Four main Sub-Segments
  - Stack
  - Heap
  - Data
  - BSS (Block Started by Symbol)

Each varies with size depending on the program

Data Segment

- Stack: Temporary Data Storage like local variables
- Heap: Dynamic data storage
- Data: Non-Zero Initialized global and static data
- BSS: Zero initialized and Uninitialized global and static data

```
int A_BSS;
int B_BSS = 0;
int C_DATA = 1;

void foo(int D_STACK_REG){
  int E_STACK_REG;
  int F_STACK_REG = 1;
  static int G_BSS;
  static int H_BSS = 0;
  static int I_DATA = 1;

  /* More Code Here */

  return;
}
```

Example Linker Script Contents

```
MEMORY
{
    MAIN  (RX) : origin = 0x00000000, length = 0x00040000
    DATA  (RW) : origin = 0x20000000, length = 0x00010000
}
```

Physical Memory Regions

```
SECTIONS
{
    .intvecs :   > 0x00000000
    .text :   > MAIN
    .const :   > MAIN
    .cinit :   > MAIN
    .pinit :   > MAIN
    .data :   > DATA
    .bss :   > DATA
    .heap :   > DATA
    .stack :   > DATA (HIGH)
}
```

Compiled Memory Sections

Example Linker Script Contents

```
MEMORY
{
    MAIN  (RX) : origin = 0x00000000, length = 0x00040000
    DATA  (RW) : origin = 0x20000000, length = 0x00010000
}
```

Physical Memory Regions

Data Segment maps to data memory

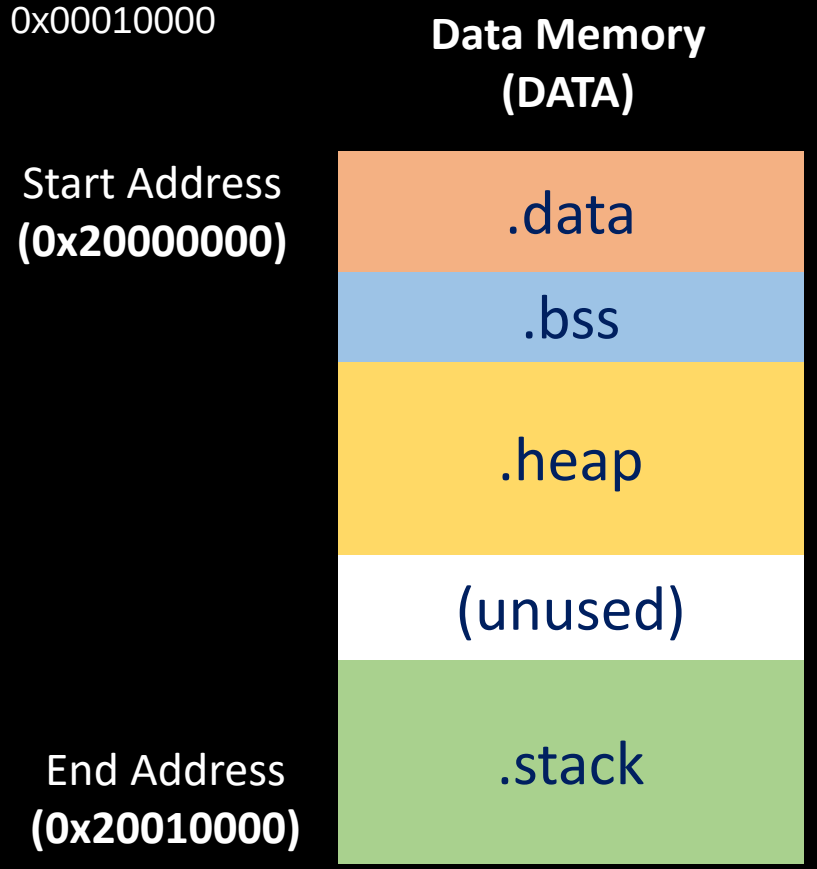
- **Contains multiple sub-segments**

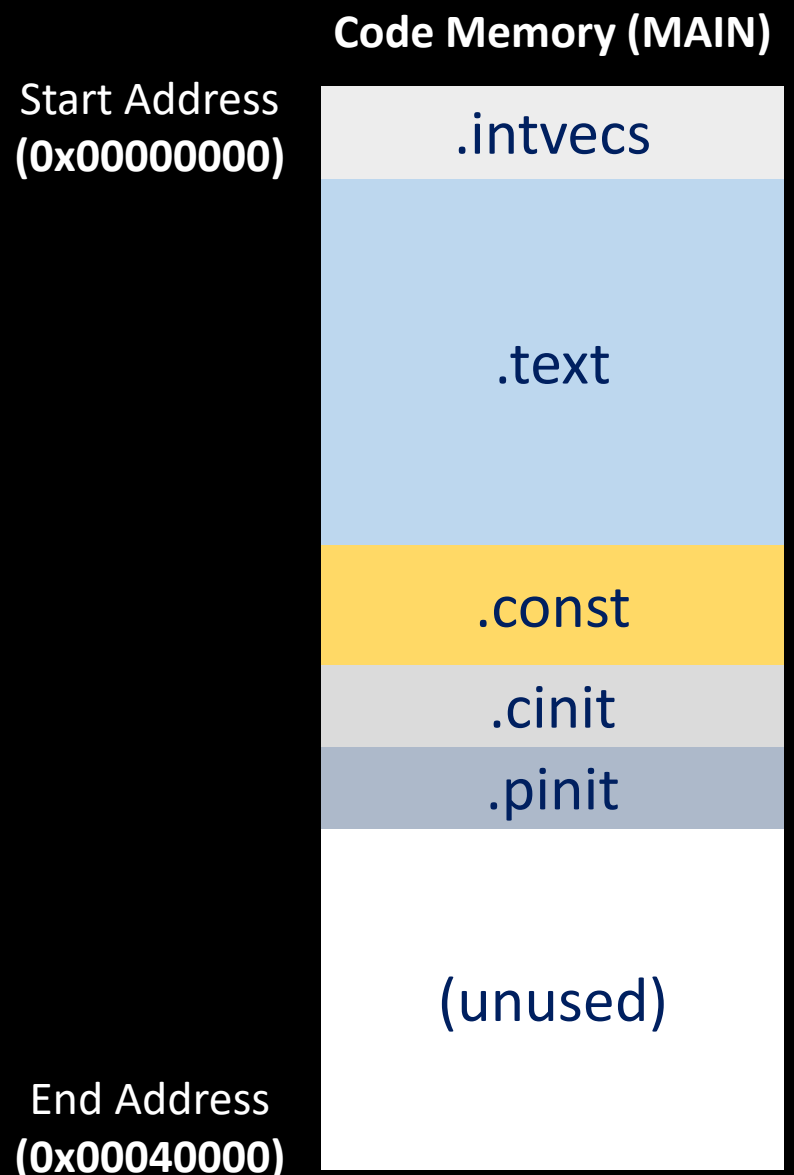
```
SECTIONS
{
    .intvecs :   > 0x00000000
    .text :   > MAIN
    .const :   > MAIN
    .cinit :   > MAIN
    .pinit :   > MAIN
    .data :   > DATA
    .bss :   > DATA
    .heap :   > DATA
    .stack :   > DATA (HIGH)
}
```

Compiled Memory Sections

Memory Segments

```
MEMORY
{
    MAIN  (RX) : origin = 0x00000000, length = 0x00040000
    DATA  (RW) : origin = 0x20000000, length = 0x00010000
}

SECTIONS
{
    .intvecs :   > 0x00000000
    .text :   > MAIN
    .const :   > MAIN
    .cinit :   > MAIN
    .pinit :   > MAIN
    .data :   > DATA
    .bss :   > DATA
    .heap :   > DATA
    .stack :   > DATA (HIGH)
}
```

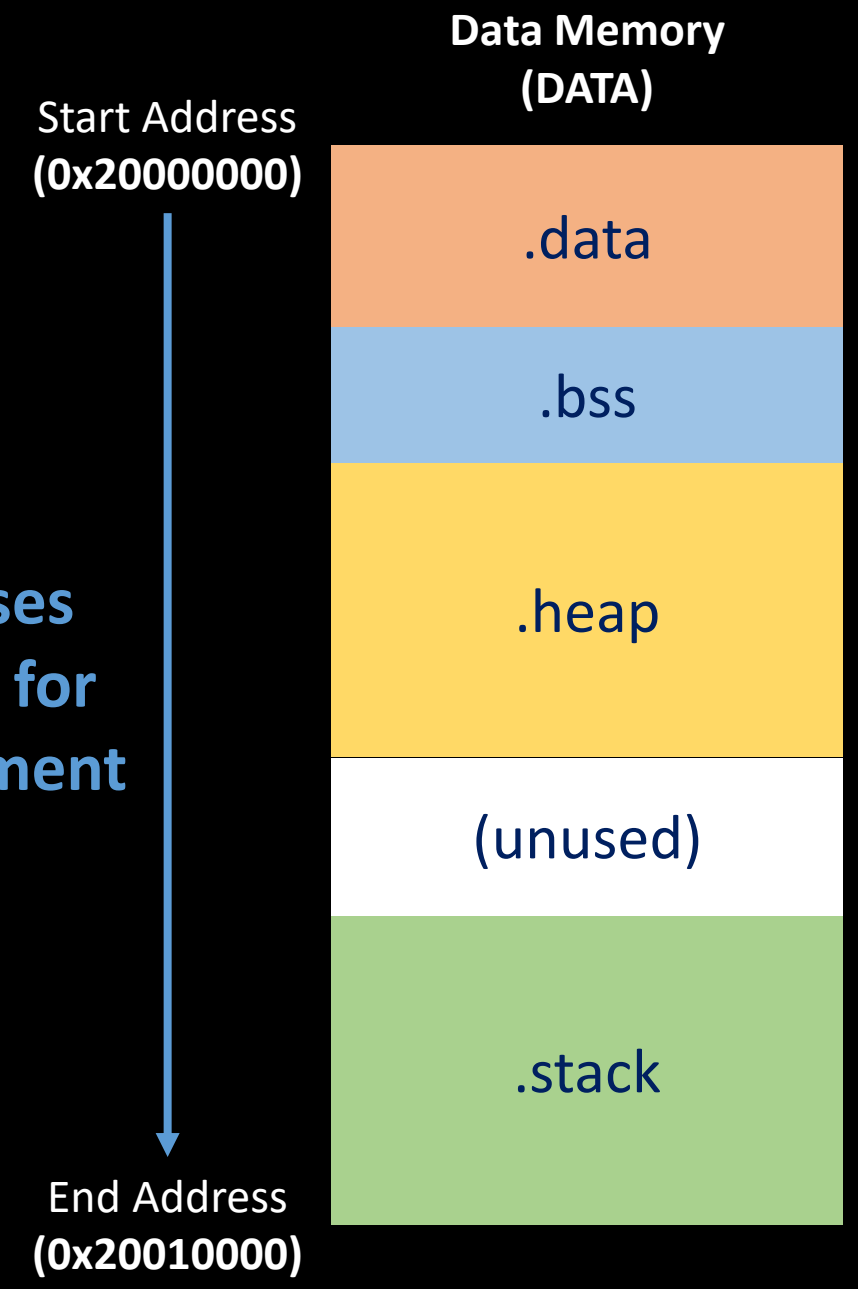




Data Segments

- Data sub-segments are allocated contiguously except for the stack
  - This Stack descends from high to low addresses

```
SECTIONS
{
   ….
   .data :   > DATA
   .bss :   > DATA
   .heap :   > DATA
   .stack :   > DATA (HIGH)
}
```



Allocated Data Characteristics

- Allocated Data can have varying
  - Size
  - Access
  - Scope
  - Location
  - Creation time
  - Lifetime

Specified by utilizing

- **Variable types**
- **Type Qualifiers**
- **Type Modifiers**
- **Storage Classes**
- **Compiler Attributes**
- **Specialized Functions**

- Data allocation is not limited to static allocation at compile time, but also dynamic allocation at runtime.

Variable Lifetime and Scope

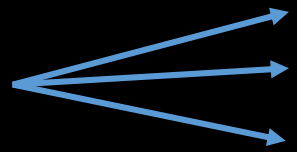
- Data memory can exist for different lengths of time
  - Lifetime of function or block
  - Lifetime of program
  - Longer than a function, less than a program

```
int varA;

int main(){
  int varB = 6;
  int varC = 12;
  int * varB_p = &varB;

  varA = varC + *varB_p;
  return 0;
}
```

Variable Lifetime and Scope

- Data memory can exist for different lengths of time
  - Lifetime of function or block
  - Lifetime of program
  - Longer than a function, less than a program

Local variables will exist
for Length of this function
or block of code

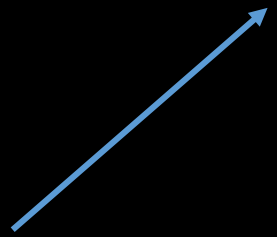


```
int varA;

int main(){
   int varB = 6;
   int varC = 12;
   int * varB_p = &varB;

   varA = varC + *varB_p;
   return 0;
}
```

Variable Lifetime and Scope

- Data memory can exist for different lengths of time
  - Lifetime of function or block
  - Lifetime of program
  - Longer than a function, less than a program

Global variables will exist for Length of the program and allocated at compile time



```
int varA;

int main(){
  int varB = 6;
  int varC = 12;
  int * varB_p = &varB;

  varA = varC + *varB_p;
  return 0;
}
```

Variable Lifetime and Scope

- Data memory can exist for different lengths of time
  - Lifetime of function or block
  - Lifetime of program
  - Longer than a function, less than a program

Variables can have different scope

- Global
- Local
  - Function
  - Block

```
int varA;

int main(){
  int varB = 6;
  int varC = 12;
  int * varB_p = &varB;

  varA = varC + *varB_p;
  return 0;
}
```

Variable Lifetime and Scope

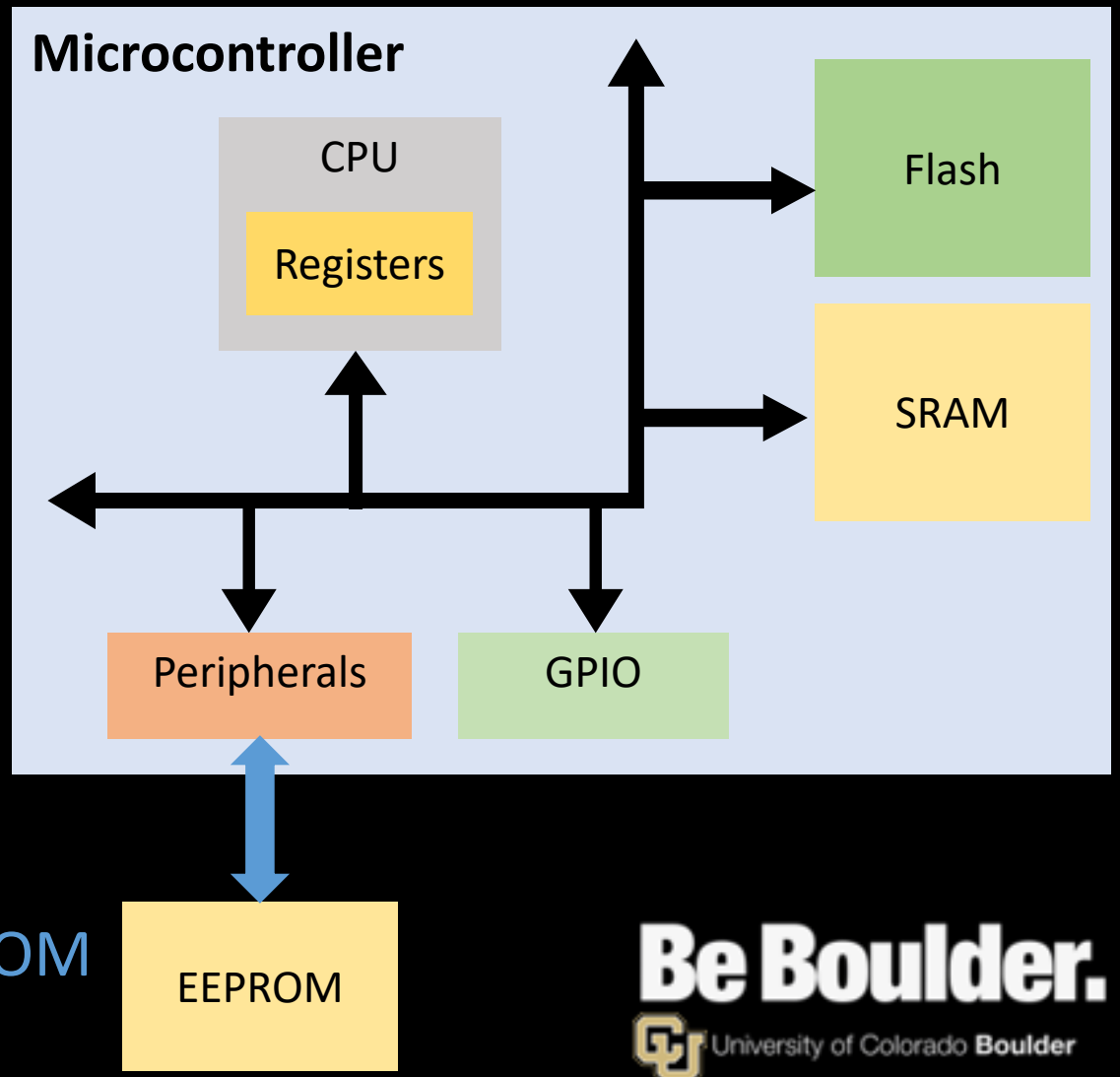
- Data memory can exist for different lengths of time
  - Lifetime of function or block
  - Lifetime of program
  - Longer than a function, less than a program

Dynamic Allocation is Data that is allocated at runtime but and managed directly by the a software programmer

```
void * malloc(size_t size);
void free(void * ptr);
```

Data Memory Locations

- Data memory in RAM is volatile
  - Data is lost when powered off
- Extra Non-Volatile memory may be included in your embedded system either on or off-chip

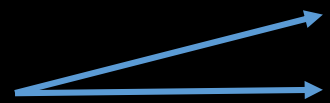


Off-Chip EEPROM example

Data Memory Locations

- Data memory in RAM is volatile
  - Data is lost when powered off

Initial values come from non-volatile memory



```
int main(){
  int varB = 6;
  int varC = 12;
  int * varB_p = &varB;

  varA = varC + *varB_p;
  return 0;
}
```

- Data must get initialized at the beginning of execution or function