Function Pointers

Embedded Software Essentials

C2 M2 V6

Function Pointers [S1a]

- Pointer that points to functions

- Defined just like a function
  - Return type
  - Function parameters
  - Pointer name

Example Declarations

```
void (* foo)();
int8_t void (* bar)( int8_t a, int8_t * b );
uint32_t (* func)( uint8_t param );
```

```
sizeof( ( void (*)) ) = sizeof( void* )
                      = sizeof( uint32_t* )
                      = 32-Bits![1]
```

[1]On our 32-bit ARM Architecture

Function Pointers [S1a]

- Pointer that points to functions
- Defined just like a function
  - Return type
  - Function parameters
  - Pointer name
- Dereferencing a function pointer will call a function

Example Declarations

```
void (* foo)();
int8_t void (* bar)( int8_t a, int8_t * b );
uint32_t (* func)( uint8_t param );
```

```
sizeof( void (*foo) ) = sizeof( void* )
                      = sizeof( uint32_t* )
                      = 32-Bits!¹
```

```
(* foo)();   or  foo();
```

[1]On our 32-bit ARM Architecture

Function Pointer Syntax [S2a]

- Declaration requires parentheses and a pointer *

$$\text{<type>}\ \underbrace{\text{(* <function\_pointer\_name>)}}_{\text{Must be inside parentheses}}\text{(<parameter list>) = <function-address>;}$$

Function Pointer Syntax [S2b]

- Declaration requires parentheses and a pointer *

$$\texttt{<type> }\underbrace{\texttt{(* <function\_pointer\_name>)}}_{\text{Must be inside parentheses}}\texttt{(<parameter list>) = <function-address>;}$$

```
int8_t (* foo)();
```

A function pointer variable declaration that returns a int8_t type

```
int8_t * foo();
```

A function declaration that returns a int8_t pointer type

```
/* Function Bar Prototype */
int8_t bar();

/* Function Pointer */
int8_t (* foo)() = &bar;
```

Function Pointer Syntax [S3a]

- Initialization and assignment to a function pointer should have matching return types and parameter list

```
<type> (* <function_pointer_name>)(<parameter list>) = <function-address>;
```

Should be consistent with function being assigned

Declarations:

```
int foo( int a, int b );        int (* fptr)( int c, int d );
```

Calling the functions:

```
ret = foo(1, 3)                 fptr = &foo;
                                ret = fptr(1, 3);
```

Function Pointer Syntax [S3b]

- Initialization and assignment to a function pointer should have matching return types and parameter list

```
<type> (* <function_pointer_name>)(<parameter list>) = <function-address>;
```

```
typedef int (* fptr_TYPE)( int c, int d );
```

Two function pointer declarations:

```
fptr_TYPE fptr1 = &foo;
fptr_TYPE fptr2 = &bar;
```

Defined functions:

```
int foo( int a, int b );
int bar( int c, int d );
```

Calling the functions:

```
ret = fptr1(1, 3);
ret = (*fptr2)(4, 5);
```

Function Pointer Array [S4a]

- Function pointers can be declared with an array

```
typedef void (* FuncPtr_t[2])();

FuncPtr_t example =
{
    foo,
    bar
};
```

Alternatively…

```
typedef void (* FuncPtr_t());

FuncPtr_t example[2] =
{
    foo,
    bar
};
```

Function Pointer Array [S4b]

- Function pointers can be declared with an array

```
typedef void (* FuncPtr_t[2])();

FuncPtr_t example =
{
    foo,
    bar
};
```

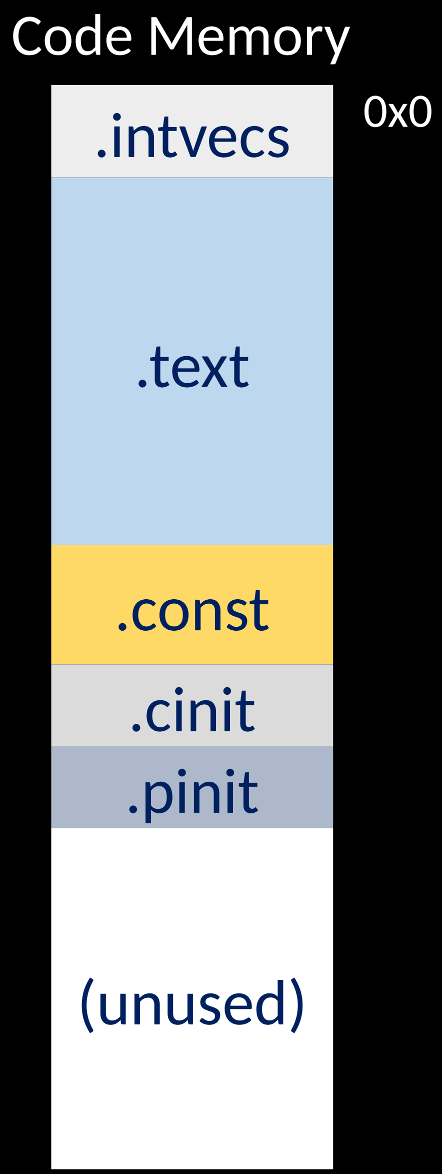
```
typedef enum
{
  FP_FOO = 0,
  FP_BAR = 1,
} FP_e;
```

Example Calls:

```
example[FP_FOO]();
example[FP_BAR]();
```

Interrupt Vector Table [S5a]

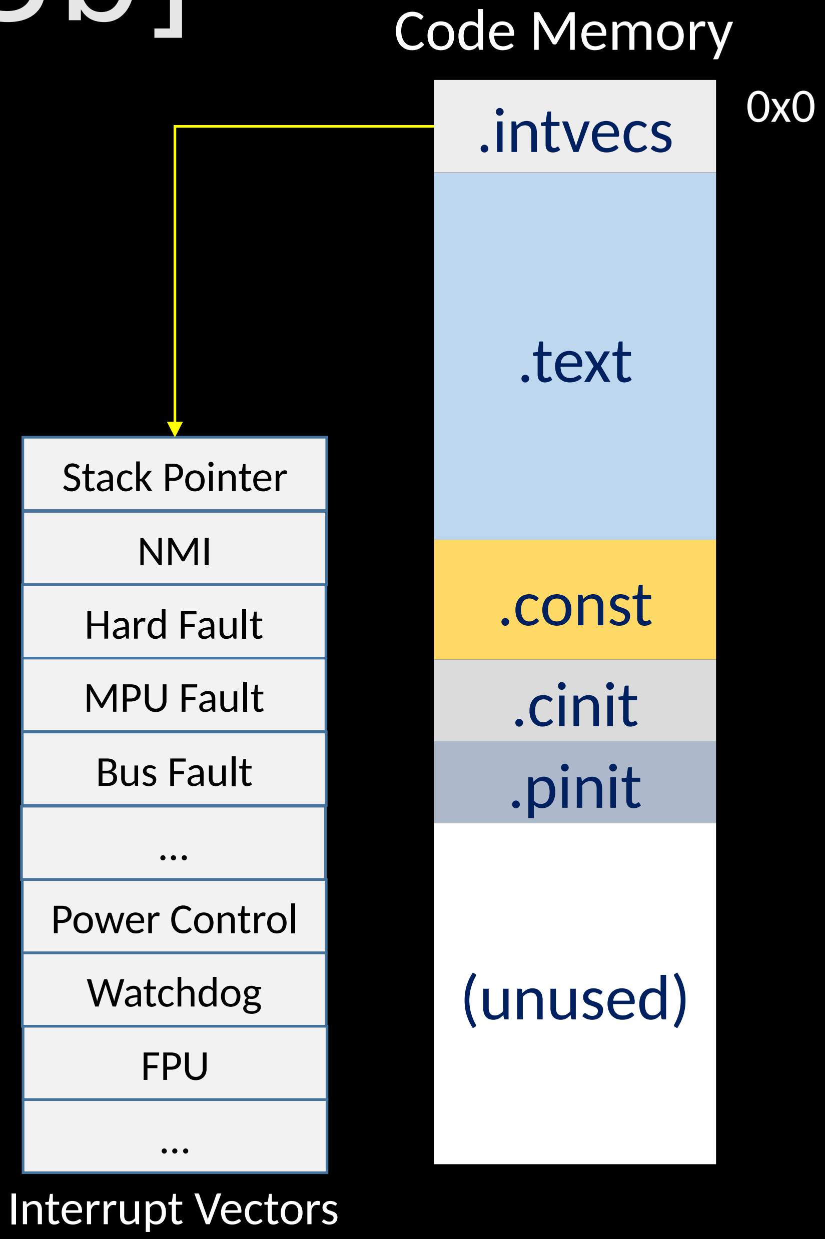
- Interrupts are special events that request the CPU to perform a specific operation



Interrupt Vector Table [S5b]

- Interrupts are special events that request the CPU to perform a specific operation
  - E.g. Timers, GPIO, CPU Exception

- Interrupt Service Routine (ISR): Function to be called in response to an interrupt request

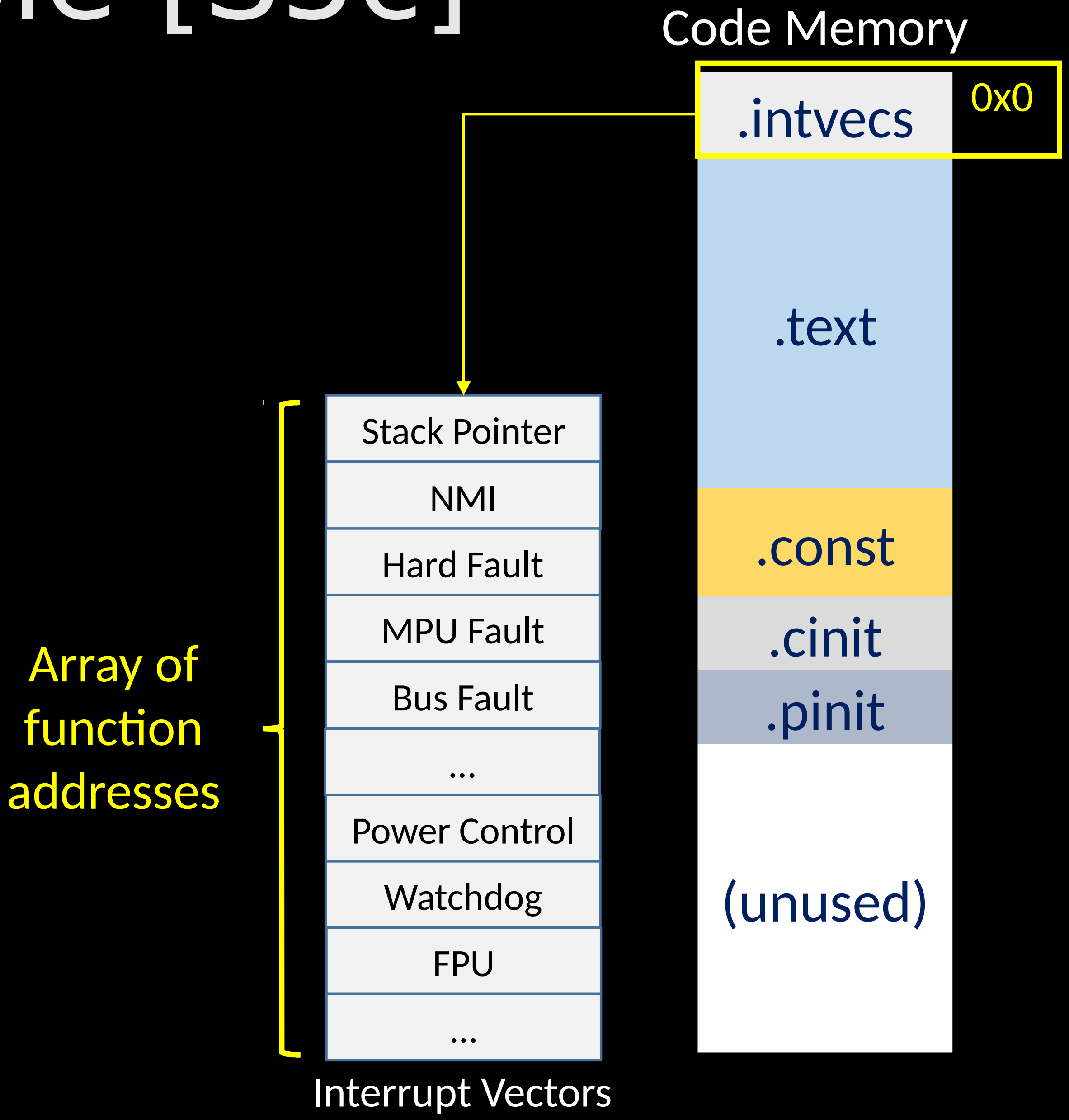


Interrupt Vector Table [S5c]

- Interrupts are special events that request the CPU to perform a specific operation
  - E.g. Timers, GPIO, CPU Exception

- Interrupt Service Routine (ISR): Function to be called in response to an interrupt request

- Placed at address 0x0 in code



Vector Table [S6a]

- Definition requires both linker mapping and C/assembly code

Linker Script Excerpt

```
SECTIONS
{
    .intvecs : >
0x00000000
    .text :     > MAIN
    .const :    > MAIN
    …
```

MSP432 Startup File Excerpt

```
#pragma DATA_SECTION(interruptVectors, ".intvecs")
void (* const interruptVectors[])(void) =
{
    (void (*)(void))((uint32_t)&__STACK_END), /* Initial stack
pointer */
    reset_ISR,                                /* Reset handler
   */
    nmi_ISR,                                  /* NMI handler
   */
    fault_ISR,                                /* Hard fault handler
   */
```

Vector Table [S6b]

- Definition requires both linker mapping and C/assembly code

Linker Script Excerpt

```
SECTIONS
{
    .intvecs : >
0x00000000
    .text :     > MAIN
    .const :    > MAIN
    …
```

MSP432 Startup File Excerpt

```
#pragma DATA_SECTION(interruptVectors, ".intvecs")
void (* const interruptVectors[])(void) =
{
    (void (*)(void))((uint32_t)&__STACK_END), /* Initial stack
pointer */
    reset_ISR,                                /* Reset handler
   */
    nmi_ISR,                                  /* NMI handler
   */
    fault_ISR,                                /* Hard fault handler
   */
    mpu_ISR,                                  /* MPU fault handler
```

Vector Table [S7a]

- Vector table is an array of function addresses
  - Used to “jump” into a routine when interrupt occurs

MSP432 Startup File Excerpt

```
#pragma DATA_SECTION(interruptVectors,
".intvecs")
void (* const interruptVectors[])(void) =
{
  (void (*)(void))((uint32_t)&__STACK_END),
  reset_ISR,
  nmi_ISR,
  fault_ISR,
  mpu_ISR,
  busfault_ISR,
  …  /* More Interrupt handlers */
```

Vector Table [S7b]

- Vector table is an array of function addresses
  - Used to "jump" into a routine when interrupt occurs

MSP432 Startup File Excerpt

Function pointer declaration

All Interrupt Subroutines are type void functions

```
#pragma DATA_SECTION(interruptVectors,
".intvecs")
void (* const interruptVectors[])(void) =
{
  (void (*)(void))((uint32_t)&__STACK_END),
  reset_ISR,
  nmi_ISR,
  fault_ISR,
  mpu_ISR,
  busfault_ISR,
  …  /* More Interrupt handlers */
```

Vector Table [S7c]

- Vector table is an array of function addresses
  - Used to “jump” into a routine when interrupt occurs

Array should be read only functions set at compile

MSP432 Startup File Excerpt

```
#pragma DATA_SECTION(interruptVectors,
".intvecs")
void (* const interruptVectors[])(void) =
{
  (void (*)(void))((uint32_t)&__STACK_END),
  reset_ISR,
  nmi_ISR,
  fault_ISR,
  mpu_ISR,
  busfault_ISR,
  …  /* More Interrupt handlers */
```

Vector Table [S8a]

- Vector table is an array of function addresses
  - Used to “jump” into a routine when interrupt occurs

First element is the initial stack pointer to initialize the Core CPU Registers

High Priority ARM Core Exceptions

MSP432 Startup File Excerpt

```
#pragma DATA_SECTION(interruptVectors,
".intvecs")
void (* const interruptVectors[])(void) =
{
   (void (*)(void))((uint32_t)&__STACK_END),
   reset_ISR,
   nmi_ISR,
   fault_ISR,
   mpu_ISR,
   busfault_ISR,
   …  /* More Interrupt handlers */
```