

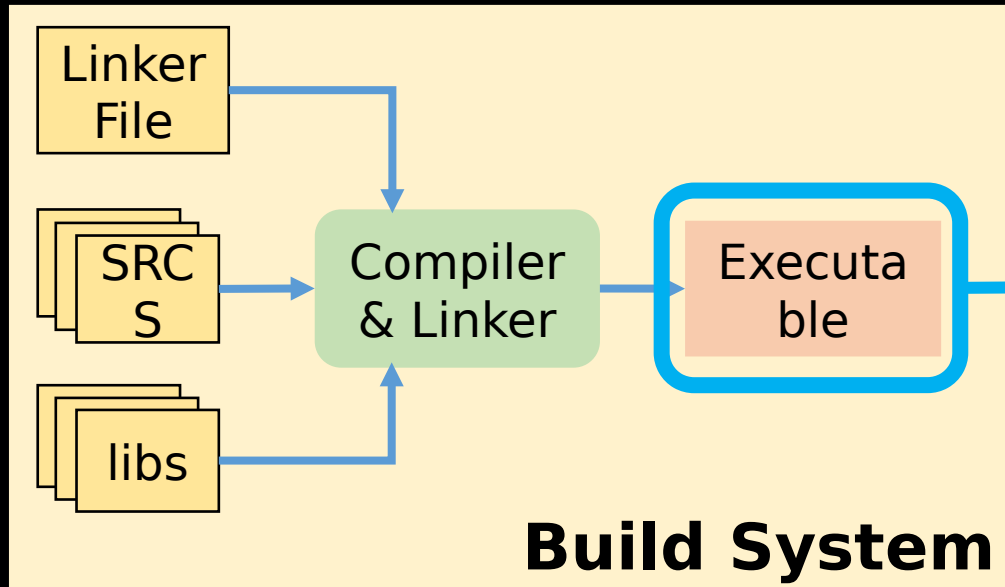
Attributes and Pragmas

Embedded Software Essentials

C2 M1 V7

Optimizations [S1]

- Optimizations will alter the implementation of code



```
b580      push    {r7, lr}
af00      add     r7, sp, #0
4b0a      ldr     r3, [pc, #40]
210a      movs   r1, #10
0018      movs   r0, r3
f000 f85f  bl     81e8 <clear_all>
4b08      ldr     r3, [pc, #32]
2200      movs   r2, #0
21aa      movs   r1, #170
0018      movs   r0, r3
```

Machine
Code

Assembly Code

Compile and Link with Optimizations

```
$ gcc main.c -o main.o -O2
```

Compiler Attributes [S2]

- **Attributes** can give specific details on how to compile code for
 - Variables
 - Structures & Structure Members
 - Functions

Attributes are **NOT**
part of the C-
standard

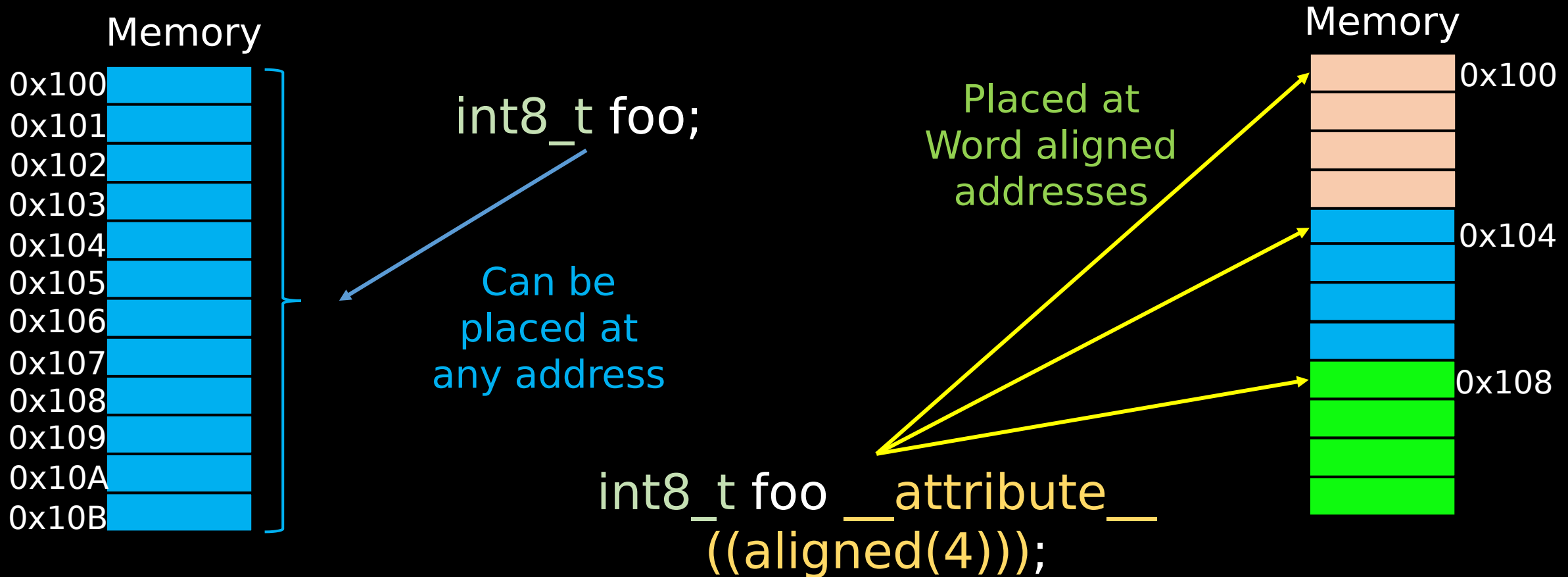


Not Portable
Across
Compilers

```
struct struct_name {  
    int8_t  var1;  
    int32_t var2;  
    int8_t  var3;  
} __attribute__((packed));
```

Alignment Attributes [S3]

- Alignment attributes specify memory alignment for data
 - Power of two: 2, 4, 8, 16



Alignment on a Structure [S4]

- Structures and structure members can be aligned

At a minimum,
structure requires 6
Bytes



```
typedef struct {  
    int8_t  var1;  
    int32_t var2;  
    int8_t  var3;  
} str1;
```

sizeof(str2) = 12 Bytes

```
typedef struct {  
    int8_t  var1 __attribute__((aligned(4)));  
    int32_t var2 __attribute__((aligned(4)));  
    int8_t  var3 __attribute__((aligned(4)));  
}
```

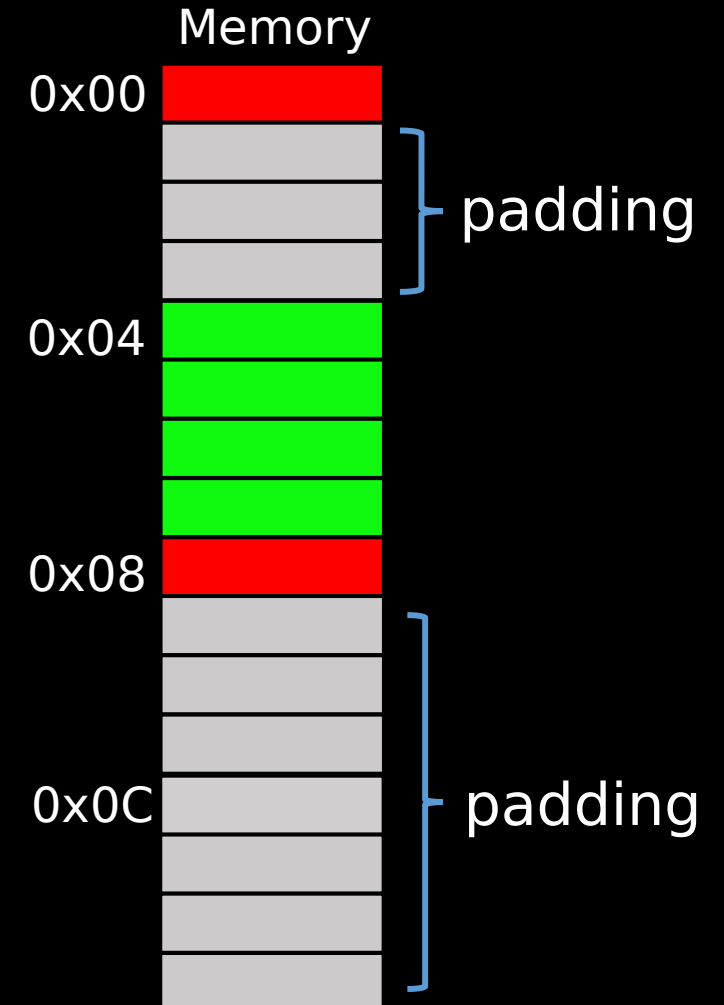
Alignment on a Structure [S5]

- Structure is aligned, all members aligned

```
typedef struct {  
    int8_t  var1;  
    int32_t var2;  
    int8_t  var3;  
} str3 __attribute__((aligned));
```

`sizeof(str3)` □ **16 Bytes**

Aligned structure members size
would require 12 bytes, Not a
power of 2!

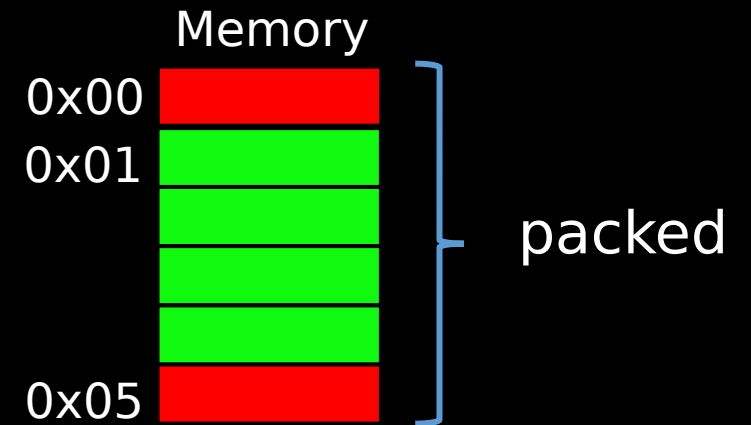


Alignment on a Structure [S6]

- Structure is packed, alignment ignored

```
typedef struct {  
    int8_t  var1;  
    int32_t var2;  
    int8_t  var3;  
} str4 __attribute__((packed));
```

`sizeof(str4)` \square 6 Bytes



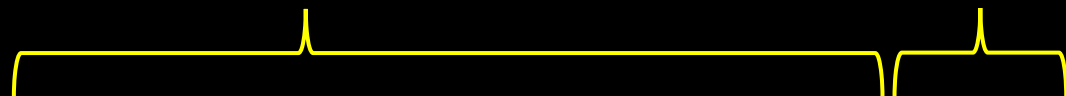
When structure is packed,
members will be
unaligned!!!

Function Attributes [S7]

- Compiler Attributes can apply to Functions
 - Inline – Skips calling convention, copies function body into calling code

Compiler will
NOT ignore this

Compiler might
ignore this



```
__attribute__((always_inline)) inline int32_t add( int32_t  
x, int32_t y ) {  
    return (x + y);  
}
```

inline keyword is a c99 Feature ☐ Not supported in c89
always_inline is a GCC attribute ☐ Not supported on other
compilers

Function Pragma [S8]

- Pragma provide special instructions to the compiler
 - Push/Pop – Add extra option for compilation
 - Optimize – Specify a certain level of optimization block of code

Only this code will
have Zero
Optimizations applied

```
#pragma GCC push
#pragma GCC optimize ("O0")
int32_t add( int32_t x, int32_t
y )
{
    return (x + y);
}
#pragma GCC pop
```

GNUC Support [S9]

- Embedded teams can support multiple chipset platforms and multiple architectures
 - Different architecture *may* require different compiler

__attribute__(x) is
only a GCC compiler
keyword,
Throws errors for
other compilers

```
#ifndef (__GNUC__)  
#define __attribute_  
(x)  
#endif
```

Define as nothing
for Non-GNU C
compilers

Ignore all slides after this

Unused slide material

Introduction [S1.3.6.a]

Attributes and pragmas are compiler directives (i.e. not part of C)
Can use them on functions and variables to convey special information to compiler.

```
/* Use #pragma to specify compiler directives */
#pragma Otime /* Optimize for execution time */
*/

/* Using attributes on functions */
void Mandelbrot16(uint16_t n, uint16_t c); __attribute__((noreturn));
#pragma Ospace __attribute__((always_inline)); /* Optimize for code space */
struct __attribute__((packed)) PackedStruct { /* sizeof(PackedStruct) = 5 bytes */
    uint8_t varx __attribute__((mode(__pointer__)));
    uint32_t vary __attribute__((aligned(16))); /* allocate 'vary' on 16-bit boundary */
};
```

Compiler Specific [S1.3.6.b]

Function attributes and pragmas are compiler-dependent, though some common ones may be shared between them.

/* Only valid for MIPS */

```
void __attribute__ ((interrupt, use_shadow_register_set)) v1  
();
```

/* Gives error unless using GCC Solaris compiler */

```
#pragma fini (fnc1, fnc2, fnc3, fnc4);
```

/* Works for GCC ARM compiler */

```
#pragma thumb
```

```
void __attribute__ ((interrupt, use_shadow_register_set)) v1  
();
```

```
static int max(int x, int y) __attribute__((always_inline));
```

Attributes at Compile Time [S1.3.6.c]

Attributes can be turned on/off using compile time switches

```
/* If compiler is not GNU C, then omit '__attribute__' */  
#ifndef __GNUC_  
#define __attribute__(x) /* Nothing */  
#endif
```

```
/* Can also use pragmas to enable/disable optimization at  
certain parts */  
#pragma GCC push_options  
#pragma GCC optimize ("O0")  
    // code section here  
#pragma GCC pop_options
```

Aligned [S1.3.6.d]

By default, strongly declared symbols have definitions.

Symbols declared *weak* don't need definitions – i.e. can have multiple definitions.

/

- * **Forces compiler to ensure 'S' or 'some_int_var'**
- * **will be allocated and aligned *at least* on a 8-bit boundary.**

*****/

```
struct S {short f[3]; } __attribute__((aligned (8)));  
typedef int some_int_var __attribute__((aligned (8)));
```

Packed [S1.3.6.e]

Using the *packed* attribute on a **struct** or **union** makes each its members also *packed*.

/ Members of packed_struct are packed, but internal layout of ustruct is not packed. The unpacked_struct must be packed separately. */*

```
struct unpacked_struct{  
    uint8_t c1;  
    uint32_t c2;  
};
```

```
struct __attribute__((__packed__)) packed_struct {  
    uint8_t d1;  
    uint32_t d2;  
    struct unpacked_struct ustruct;  
};
```


Target [S1.3.6.f]

Target attribute allows user to specify target-specific compilation options.

/* Equivalent to compiling somefunc with '-march=core2' and '-sse4a' target options. */

uint32_t somefunc (void) __attribute__ ((__target__ ("march=core2", "sse4a")));

Pragma Optimizations [S1.3.6.g]

Use pragma to specify optimization levels and types

#pragma Otime void function1(){ ... } execution time */	/* Optimize function1 for
#pragma push	/* Save current pragma state */
#pragma O2 uint32_t function2(){ ... }	/* Optimize at level 3 */ /* Optimize function2 at O3*/
#pragma Ospace uint8_t function3(){ ... } size */	/* Optimize function3 for code
#pragma pop pragma state */	/* Restores previously saved