C PROGRAMMING FOR EMBEDDED SYSTEMS

EL-GY 6483 REAL TIME EMBEDDED SYSTEMS

C FOR EMBEDDED

Language	Programmers
C	60%
C++	21%
Assembly	5%
Java	3%
C#	2%
MATLAB/Labview	4%
Python	1%
.NET	1%
Other	4%

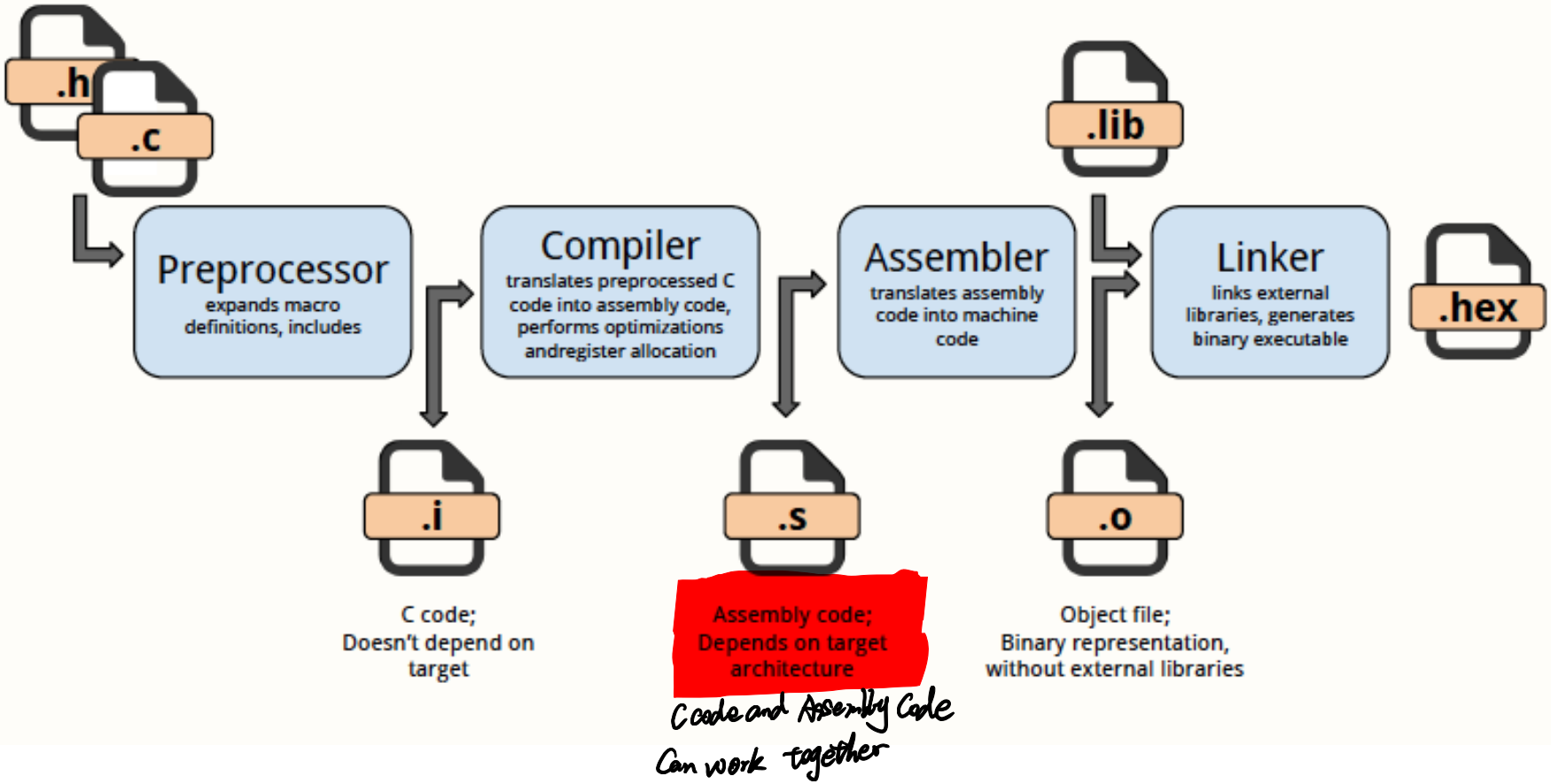
C FOR EMBEDDED

Some differences in programming for embedded systems:

- Compiling for a different target architecture
- Limited memory, processing power on target
- Can have input from external peripherals
- Reliability constraints

LIFECYCLE OF A C PROGRAM FOR EMBEDDED

HOW C CODE BECOMES AN EXECUTABLE



SPECIFICS

DATA TYPES

No need to include it

C type	stdint.h type	Bits	Sign	Range
char	uint8_t	8	Unsigned	0 .. 255
signed char	int8_t	8	Signed	-128 .. 127
unsigned short	uint16_t	16	Unsigned	0 .. 65,535
short	int16_t	16	Signed	-32,768 .. 32,767
unsigned int	uint32_t	32	Unsigned	0 .. 4,294,967,295
int	int32_t	32	Signed	-2,147,483,648 .. 2,147,483,647
unsigned long long	uint64_t	64	Unsigned	0 .. 18,446,744,073,709,551,615
long long	int64_t	64	Signed	-9,223,372,036,854,775,808 .. 9,223,372,036,854,775,807

BITWISE OPERATIONS

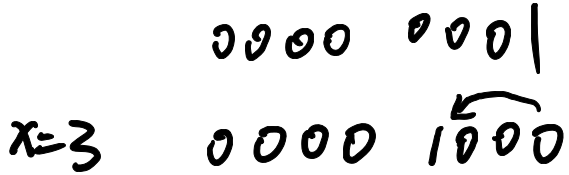
Bitwise operation	Symbol (in C)
AND	&
OR	|
XOR	^
NOT	~ !
Left Shift	<<
Right Shift	>>

EXAMPLE: CHECK A BIT

To check a bit, AND it with the bit you want to check:

```
bit = number & (1 << x);
```

That will put the value of bit x into the variable bit.



BIT FIELDS

To create a mask of certain bits.

```
#define BIT_MUTE_AUDIO   0x01
#define BIT_BACKLIGHT    0x02

unsigned int flags;

flags = (BIT_MUTE_AUDIO | BIT_BACKLIGHT);
```

=0x03

BIT OPERATIONS: EXERCISE

Answer the following question for C, using bitwise operators:

byte |= (1<<x)

xxxx xxxx
0000 1000

How do you set, clear and toggle a single bit in C/C++?

How to set, clear and toggle a bit in C/C++?

c++ | c | bit-manipulation | embedded

edited Nov 4 '13 at 18:58
Luchian Grigore
142k 22 217 370

asked Sep 7 '08 at 0:42
JeffV
9,414 15 64 99

IMPLIED DECLARATION

- int n = 0x7F2;
- int n = 0b1010;
- int n = (1<<3);

EXAMPLES

0x05 & 0x01 = ?

0x05 | 0x02 = ?

0x05 ^ 0x01 = ?

0x05 << 2 = ?

0x05 >> 1 = ?

0xF4 & 0x3A = ?

(1<<19) | (1<<12) = ?

ASSIGNMENT OPERATIONS

Table 2.10: Assignment Operators

Operator	Syntax	Equivalent Operation
+=	`i += j;`	`i = (i + j);`
-=	`i -= j;`	`i = (i - j);`
*=	`i *= j;`	`i = (i * j);`
/=	`i /= j;`	`i = (i / j);`
%=	`i %= j;`	`i = (i % j);`
&=	`i &= j;`	`i = (i & j);`
|=	`i |= j;`	`i = (i | j);`
^=	`i ^= j;`	`i = (i ^ j);`
<<=	`i <<= j;`	`i = (i << j);`
>>=	`i >>= j;`	`i = (i >> j);`

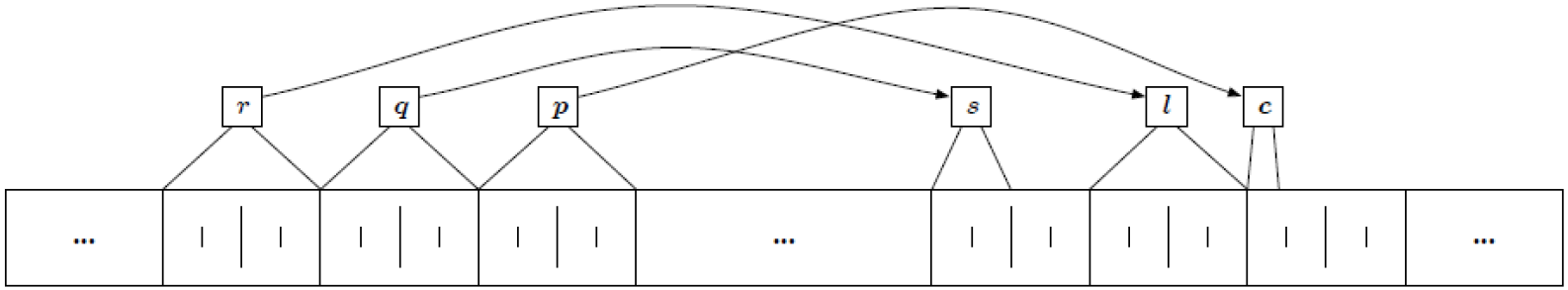
INCREMENT/DECREMENT OPERATIONS

Table 2.8: Increment and Decrement Operators

Operator	Operation
++	increment value by 1; either before or after the variable is used
--	decrement value by 1; either before or after the variable is used

Statement	x Before	n After	x After
`n = x++;`	10	10	11
`n = ++x;`	10	11	11
`n = x--;`	10	10	9
`n = --x;`	10	9	9

POINTERS




```
char *p;
short *q;
long *r;
```

```
p = &c;
q = &s;
r = &l;
```

```
p = &c;
c = 0;
*p = 10;

/* now it is true that (c == 10) */
```

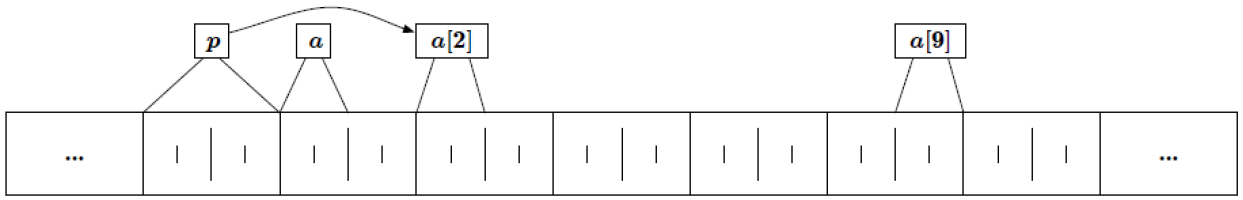
POINTERS EXAMPLE

Table 2.13: Pointer Indexing Operations

	Before			After			
Instruction	&c = 100	101	p	&c = 100	101	p	*p
`c = *p + 1;`	5	0	100	6	0	100	6
`*p += 1;`	5	0	100	6	0	100	6
`++*p;`	5	0	100	6	0	100	6
`(*p)++;`	5	0	100	6	0	100	6
`*p++;`	5	0	100	5	0	101	0

ANOTHER POINTER EXAMPLE

```
short *p;
short a[10];

p = &(a[2]);

/* The following expressions are true. */
*(p) == a[2];
*(p+1) == a[3];
```



POSSIBLE??

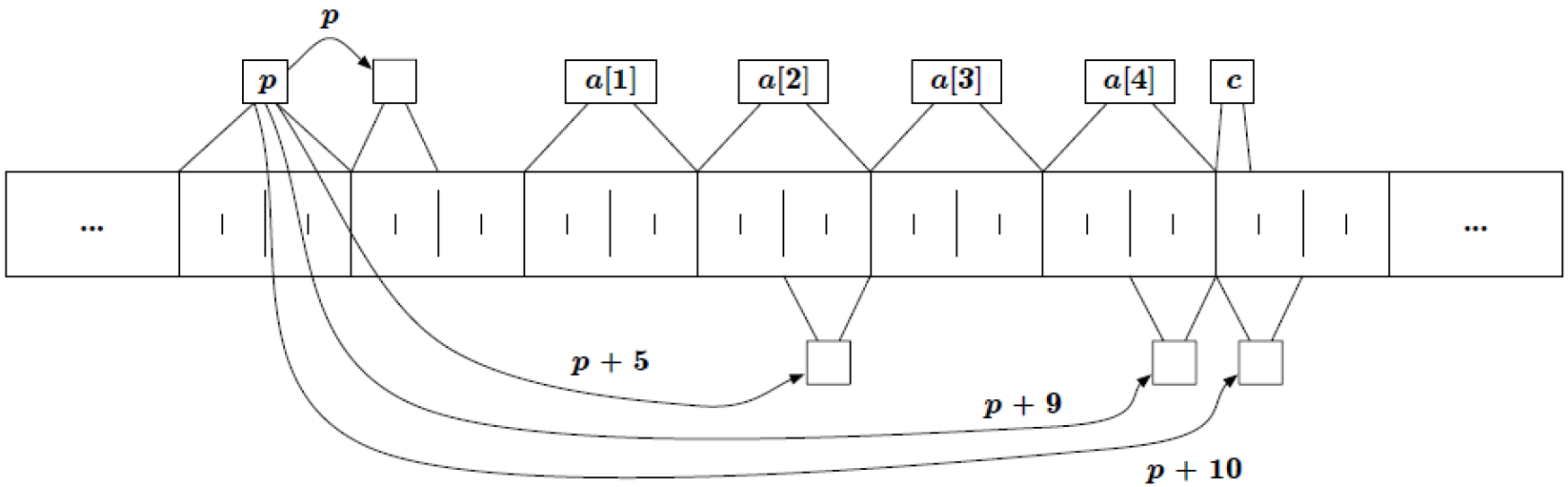
```
short *p;
long a[5];
char c;

p = (short *) (&(a[0]));
```

FUNCTION POINTERS, POSSIBLE??

Yes, but can be tricky.

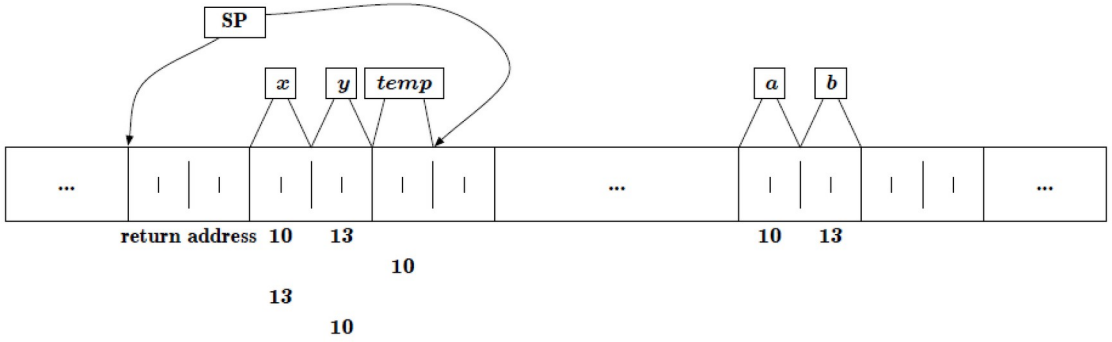
- callbacks into RTOSes;
- ISR handling;
- I/O port interfacing to higher level;

POSSIBLE?? - YES



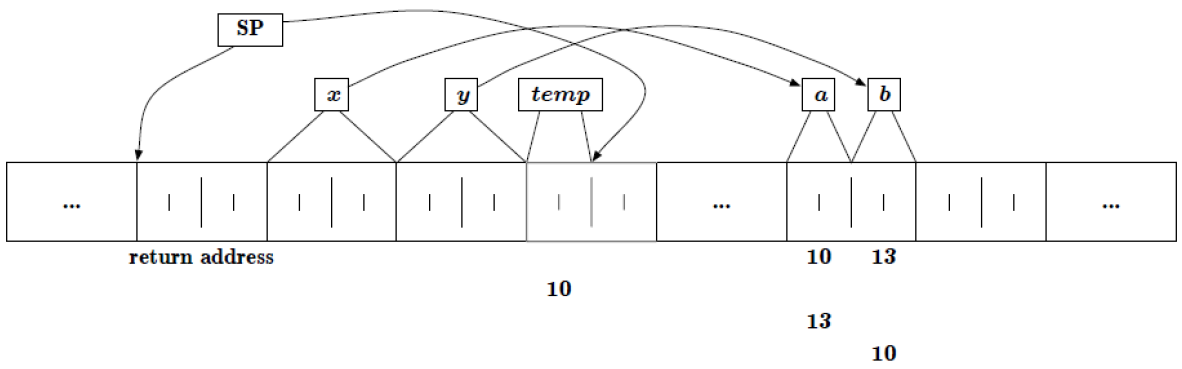
PASSING BY VALUE VS. REFERENCE

```
void main (void)
{
   short a = 10;
   short b = 13;

   swap (a,b);

   /* a == ?, b == ? */
}

void swap (short x, short y)
{
   short temp;

   temp = x;
   x = y;
   y = temp;
}
```

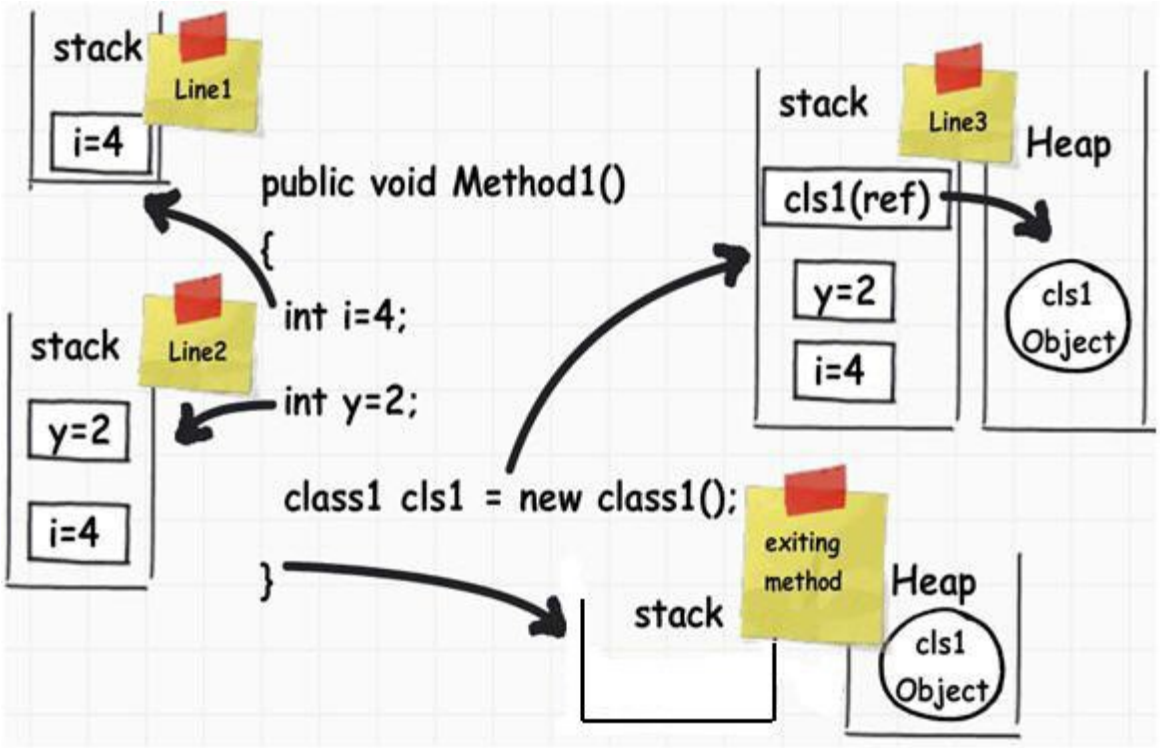


PASSING BY VALUE VS. REFERENCE

```
void main (void)
{
   short a = 10;
   short b = 13;

   /* Now we pass the address of the variables we want to change. */
   swap (&a,&b);

   /* a == ?, b == ? */
}

void swap (short *x, short *y)
{
   short temp;

   temp = *x;
   *x = *y;
   *y = temp;
}
```



MEMORY MANAGEMENT



Source: ”What and where are the stack and heap?” Answer by Snow Crash, http://stackoverflow.com/questions/79923/what-and-where-are-the-stack-and-heap

MEMORY MANAGEMENT

```
int foo() {
  char *pBuffer; //<--nothing allocated yet (excluding the pointer itself,
which is                                  //allocated here on the stack).
  bool b = true; // Allocated on the stack.
  if(b)
  {
     //Create 500 bytes on the stack
    char buffer[500];
    //Create 500 bytes on the heap
    pBuffer = new char[500]; }//<-- buffer is deallocated here, pBuffer
is not
  }//<--- oops there's a memory leak, I should have called delete[]
 pBuffer;
```

Source: "What and where are the stack and heap?" Answer by Snow Crash, http://stackoverflow.com/questions/79923/what-and-where-are-the-stack-and-heap

MEMORY MANAGEMENT

Some coding standards (e.g. for high-reliability embedded systems) forbid dynamic memory allocation. Why?

VOLATILE

- We specify volatile variables when using interrupts and I/O ports
- Tells compiler that variables can be changed outside of the code

VOLATILE

A programmer writes the following function to get the square of a volatile integer parameter pointed to by*p.

However, when he tests it, it returns ‘6’ – which is not a square of an integer value!

Why does this happen, and how can he modify his code so that it will always return a valid square?

```
int square(volatile int *p)
{
return *p * *p;
}
```

TYPE QUALIFIERS

When might we declare

const volatile int n;

?