EL-GY 6483 Real Time Embedded Systems

DEFINITION: EMBEDDED SYSTEM

- "Any device that includes a programmable computer but is not itself intended to be a general purpose computer."[1]
- "Information processing systems embedded into enclosing products." [2]
- Embedded software is software integrated with physical processes. The technical problem is managing time and concurrency in computational systems." [3]

1Wayne Wolf
2Peter Marwedel, TU Dortmund
3Edward Lee, Berkeley

EXAMPLES

Embedded systems:

- Mars rover
- Cardiac pacemaker
- Google glass

Not embedded systems:

- Laptop computer
- Desktop computer
- Cloud server

What about smartphones?

CHARACTERISTICS:

Real time: responds to input within a predictable, finite period (deadline). The correctness depends not only on the logical result but also the time it was delivered.

SYSTEM	TYPE	COMMENT
Missile Launcher	Hard	Missing a deadline is a total system failure
Weather Predictions	Firm	Result is not useful after its deadline. Occasional misses degrade quality of service, but are not catastrophic.
Live Video	Soft	Result is less useful after its deadline. Misses degrade quality of service

CHARACTERISTICS

Reactive: responds to input from physical environment.

This input is not sequential, often “messy”:

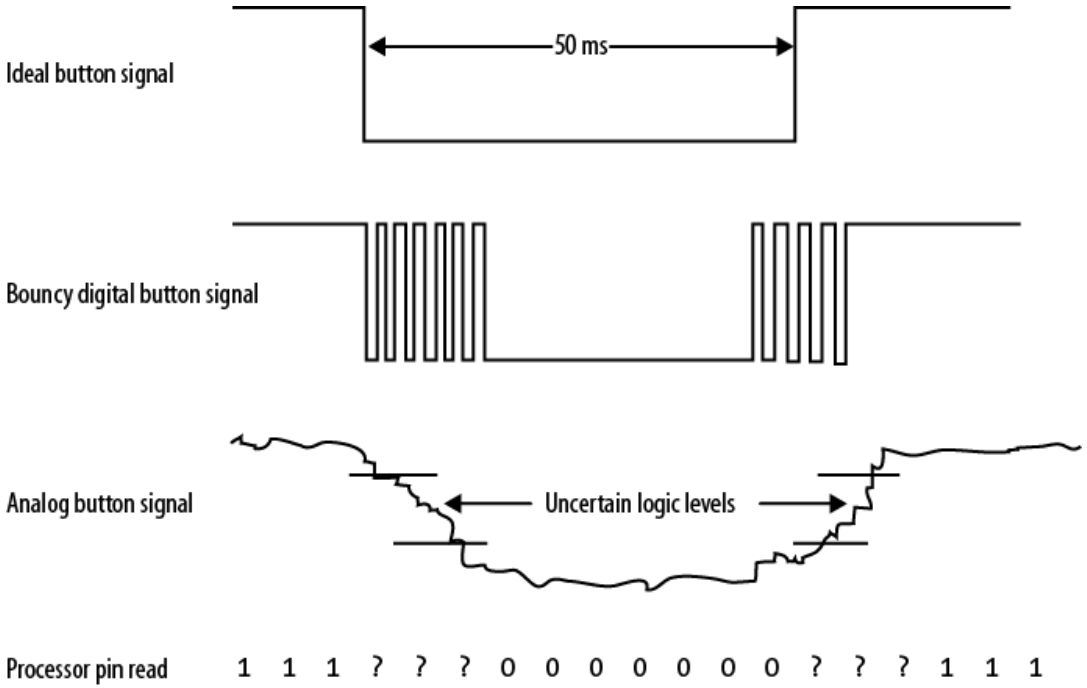


Image source: ”Making embedded systems”, Elicia White, O’Reilly

CHARACTERISTICS

Dependability: impact on physical world, may have safety implications.

Example: Therac-25

Therac-25 was a radiation therapy machine used in the 1980s. Because of programming errors, several people suffered massive radiation overdoses and some died.

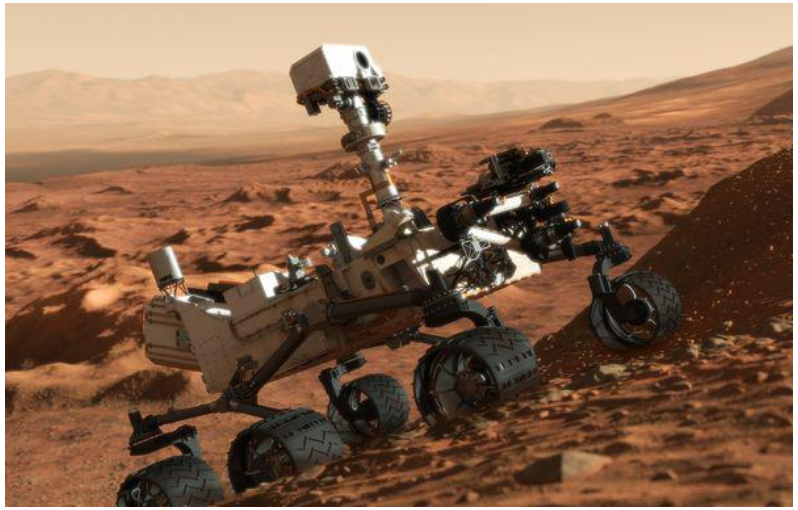
CHARACTERISTICS

Reliability: survival probability.

It is often impossible to fix bugs “in the field” and sometimes it’s impossible to replace the system.

Example: Mars Curiosity Rover

Chipset used is allowed to have one and only one “bluescreen” every 15 years (even under heavy radiation).



CHARACTERISTICS

Efficiency: use limited resources effectively.

Embedded system hardware is typically subject to strict size, weight, and manufacturing cost restraints.

MULTIDISCIPLINARY DESIGN

- Algorithms
- Operating systems
- Computer architecture
- Control theory
- Operations research
- Queuing theory
- Computer architecture
- Digital signal processing
- Electronics
- Human computer interaction

… to name a few

REVIEW: COMPUTER ORGANIZATION

UNITS OF DATA

- Bit: A single binary digit, that can have either value 0 or 1.
- Byte: 8 bits.
- Nibble: 4 bits.
- Word: varies by CPU, typically equals size of register. (For this class: 32 bits)

REPRESENTING DATA

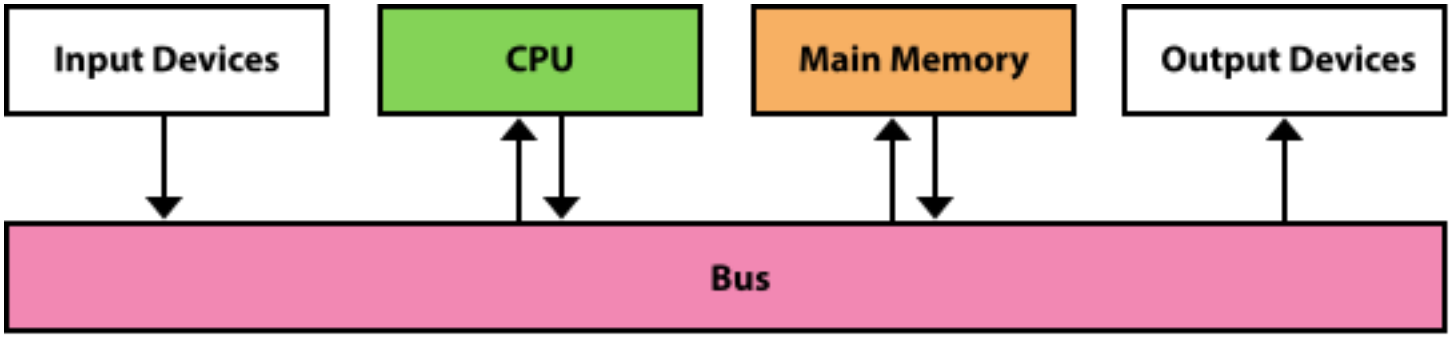
- Binary (base 2): 0's and 1's. One digit is a bit.
- Hexadecimal (base 16): 0,…9, A,B,C,D,E,F. One digit is a nibble.
- Octal (base 8): 0,…7. One digit is a byte.

Examples:

0xE in hex = 14 in decimal = 1110 in binary

0x64 in hex = 100 decimal = 1100100 in binary

SIMPLE MODEL

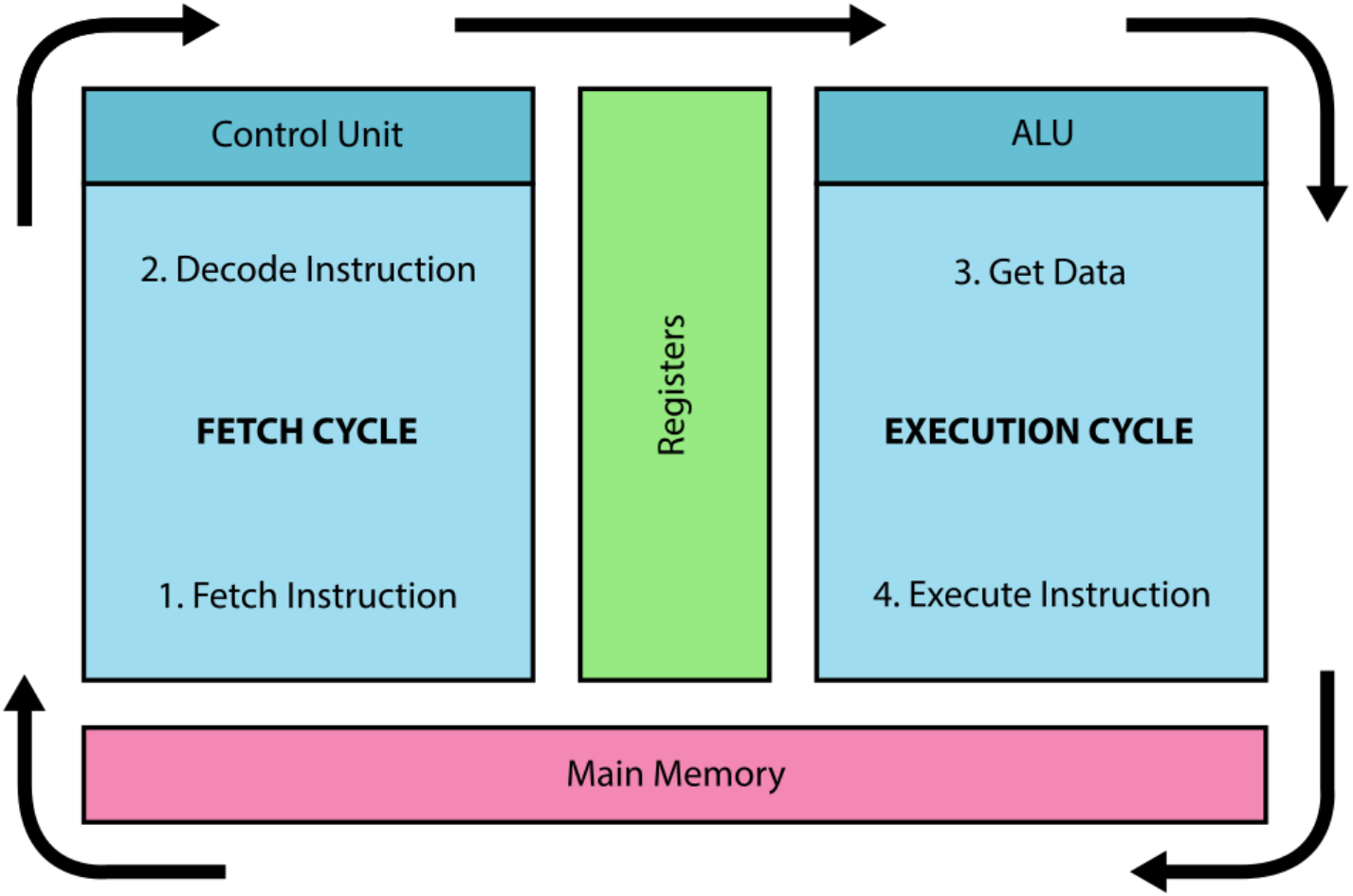


CPU

The role of the CPU is to execute a list of instructions (program), usually to manipulate data.

- Control Unit: directs the operation of the units with timing and control signals. Includes two special registers:
  - Instruction Register (IR) contains the instruction that is being executed
  - Program Counter (PC) contains the address of the next instruction to be executed
- Arithmetic/Logic Unit: performs basic arithmetic operations (add, subtract, multiply) and logical operations (and, or not). Uses registers for fast access to data operands.

FETCH – DECODE – EXECUTE - STORE



MEMORY

Main/Primary memory: Volatile (RAM), keeps programs when they are running and immediate data

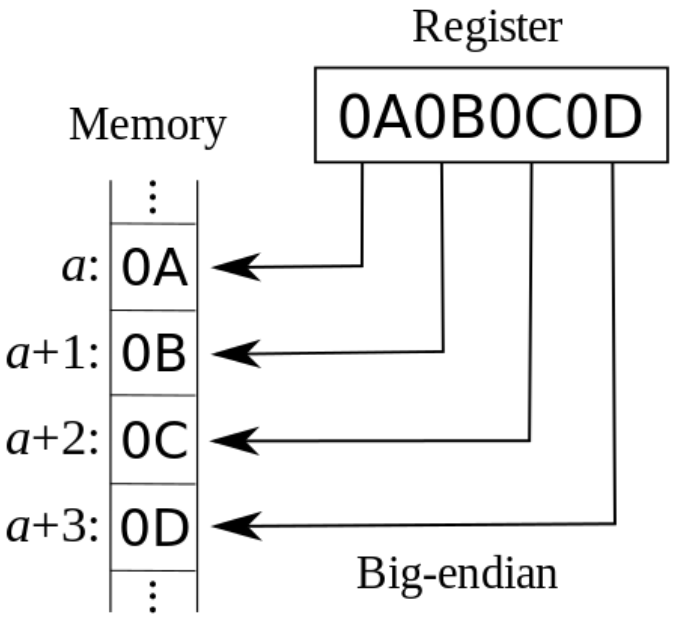
Secondary Memory: Non-volatile (ROM), long-term storage for programs and data

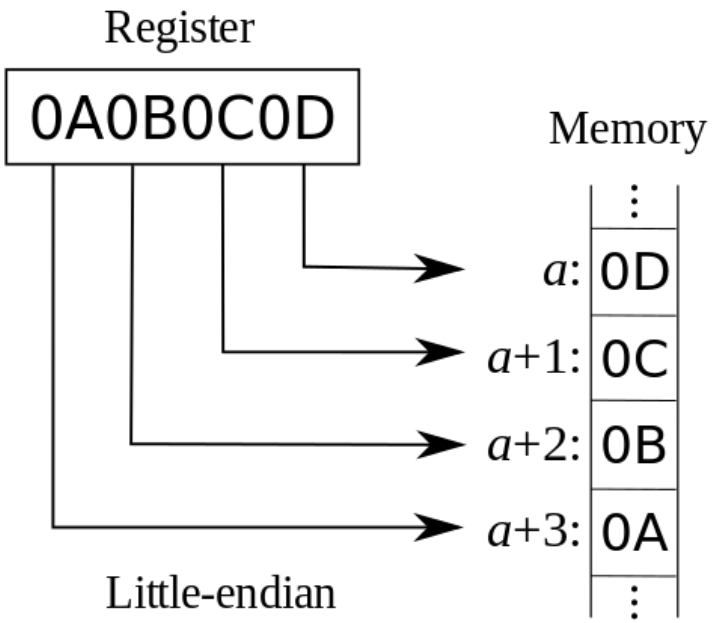
MEMORY ADDRESSING

A unique ID (address) is assigned to each byte of memory (byte-addressable) or each word of memory (word-addressable)

Address	Content	Name	Type	Value
0x90000000	0x00			
0x90000001	0x00			
0x90000002	0x00	anInt	int	0x000000FF(255)
0x90000003	0xFF			
0x90000004	0xFF	aShort	short	0xFFFF(-1)
0x90000005	0xFF			
0x90000006	0x1F			
0x90000007	0xFF			
0x90000008	0xFF			
0x90000009	0xFF			
0x9000000A	0xFF	aDouble	double	0x1FFFFFFFFFFFFFF
0x9000000B	0xFF			(4.4501477170144023E-308)
0x9000000C	0xFF			
0x9000000D	0xFF			
0x9000000E	0x90			
0x9000000F	0x00			
0x90000010	0x00	ptrAnInt	int*	0x90000000
0x90000011	0x00			

ENDIANNESS


Register
0A0B0C0D
Memory
a: 0A
a+1: 0B
a+2: 0C
a+3: 0D
Big-endian


Register
0A0B0C0D
Memory
a: 0D
a+1: 0C
a+2: 0B
a+3: 0A
Little-endian

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%

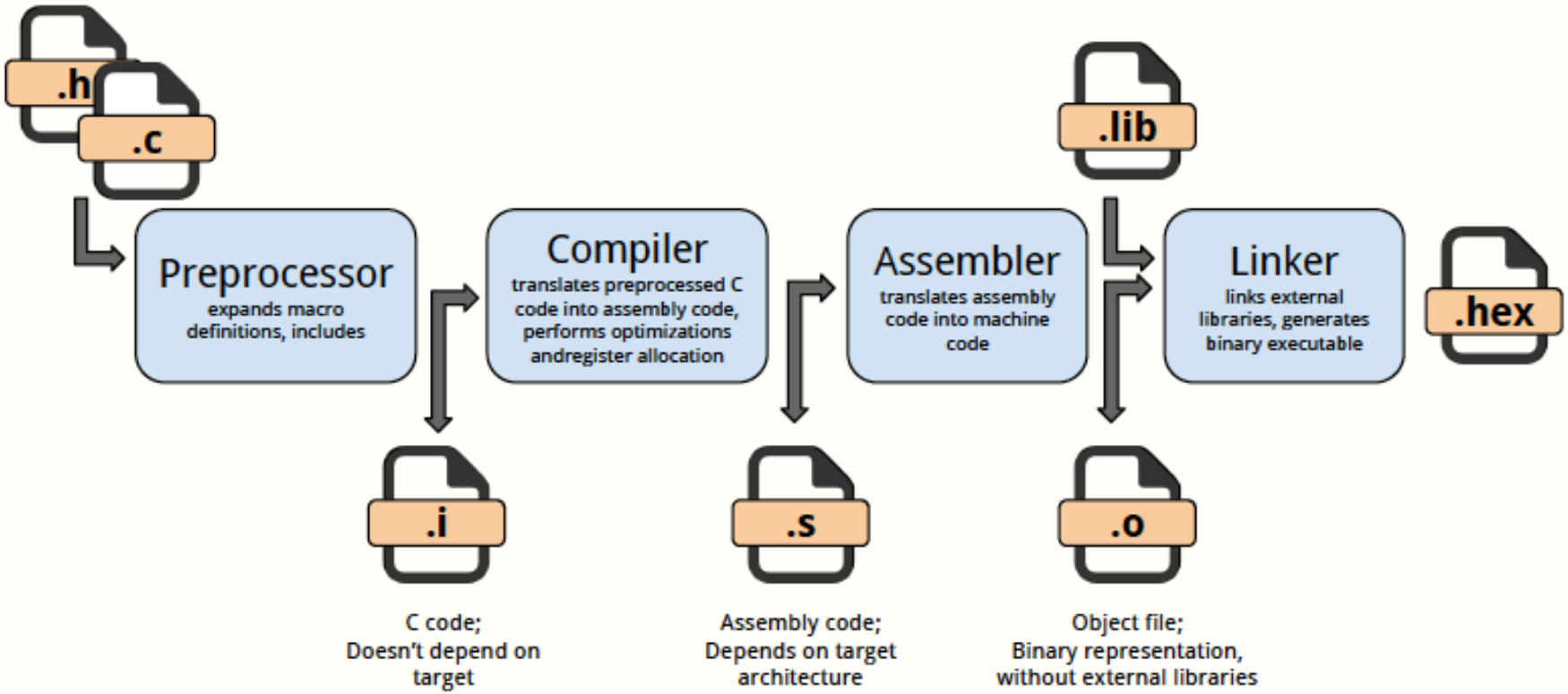
C FOR EMBEDDED

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

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);
```

BIT OPERATIONS: EXERCISE

Answer the following question for C, using bitwise operators:

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);`

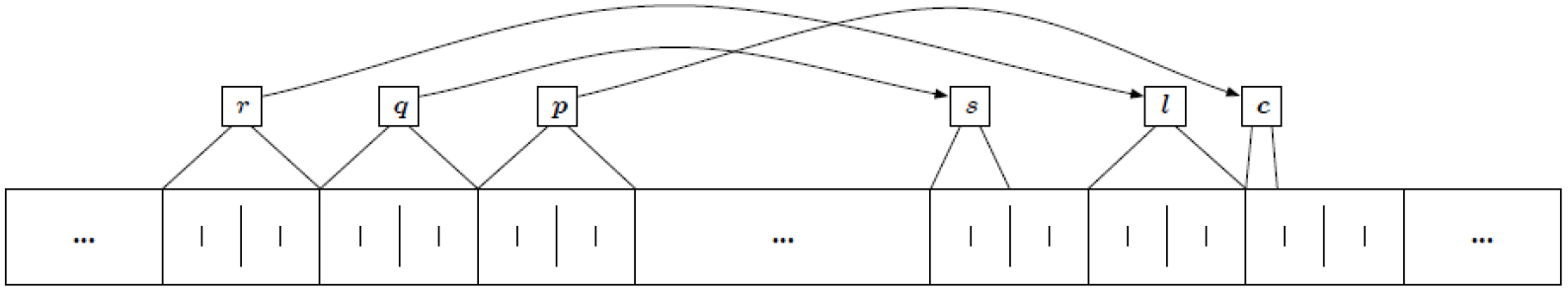
INCREMENT/DECREMENT OPERATIONS

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) */
```

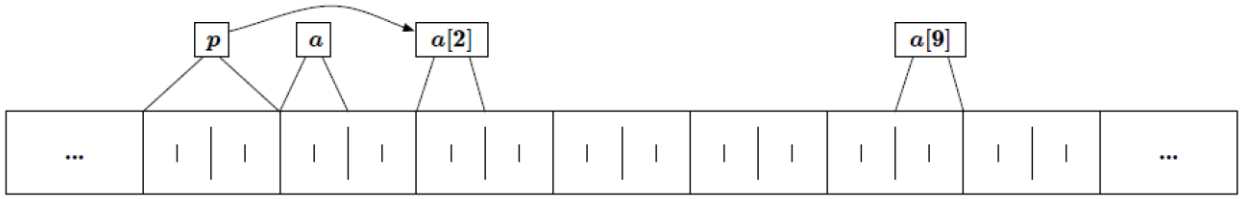
POINTERS EXAMPLE

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

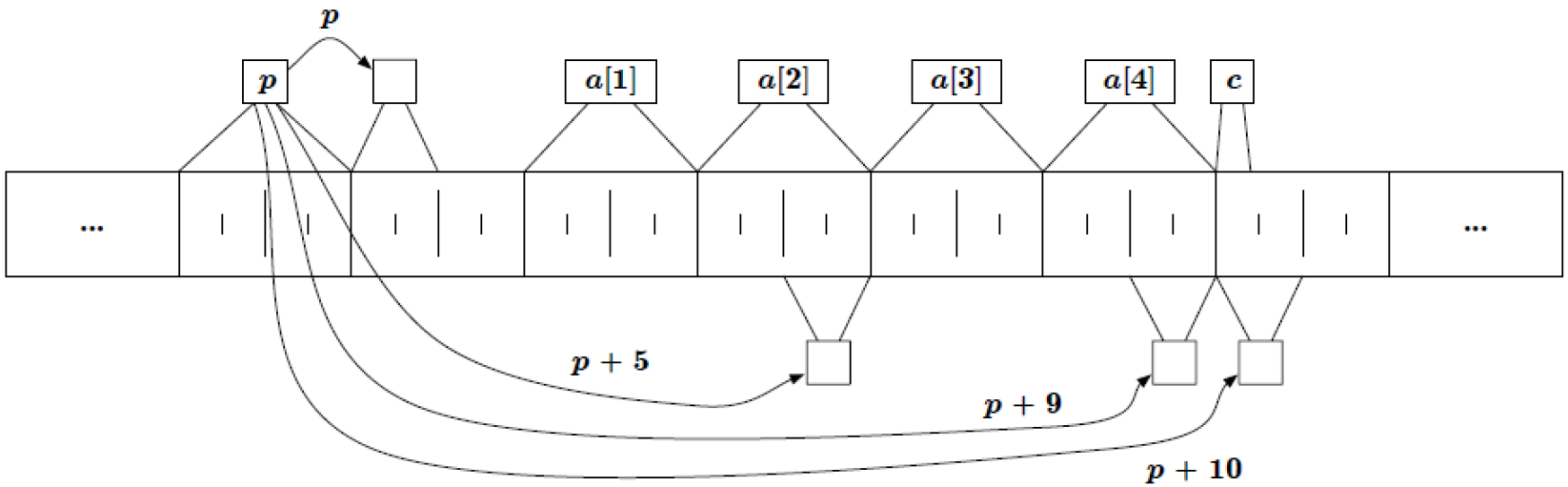
```
short *p;
short a[10];

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]));
```

POSSIBLE?? -YES



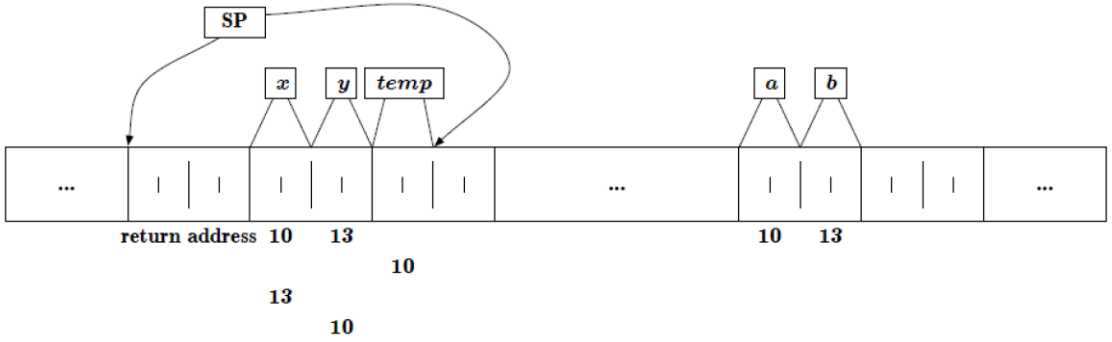
FUNCTION POINTERS, POSSIBLE??

Yes, but can be tricky.

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

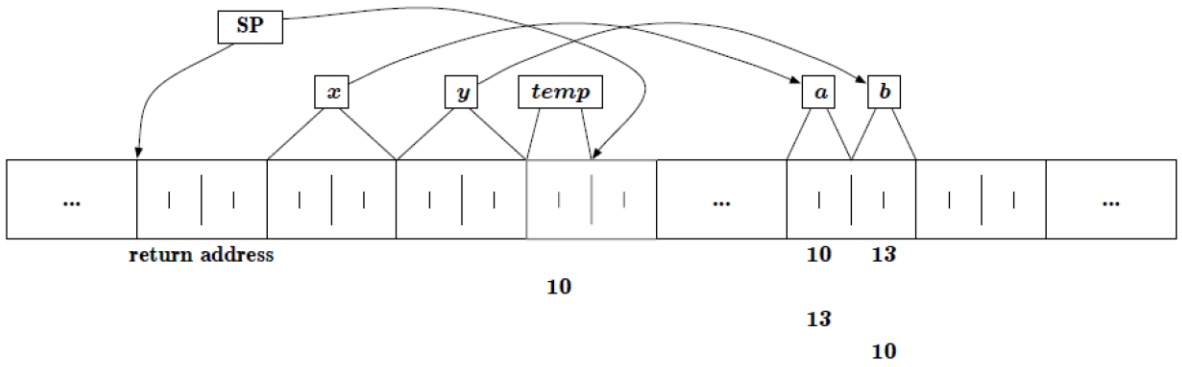
PASSING BY VALUE VS. REFERENCE

```
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;
}
```

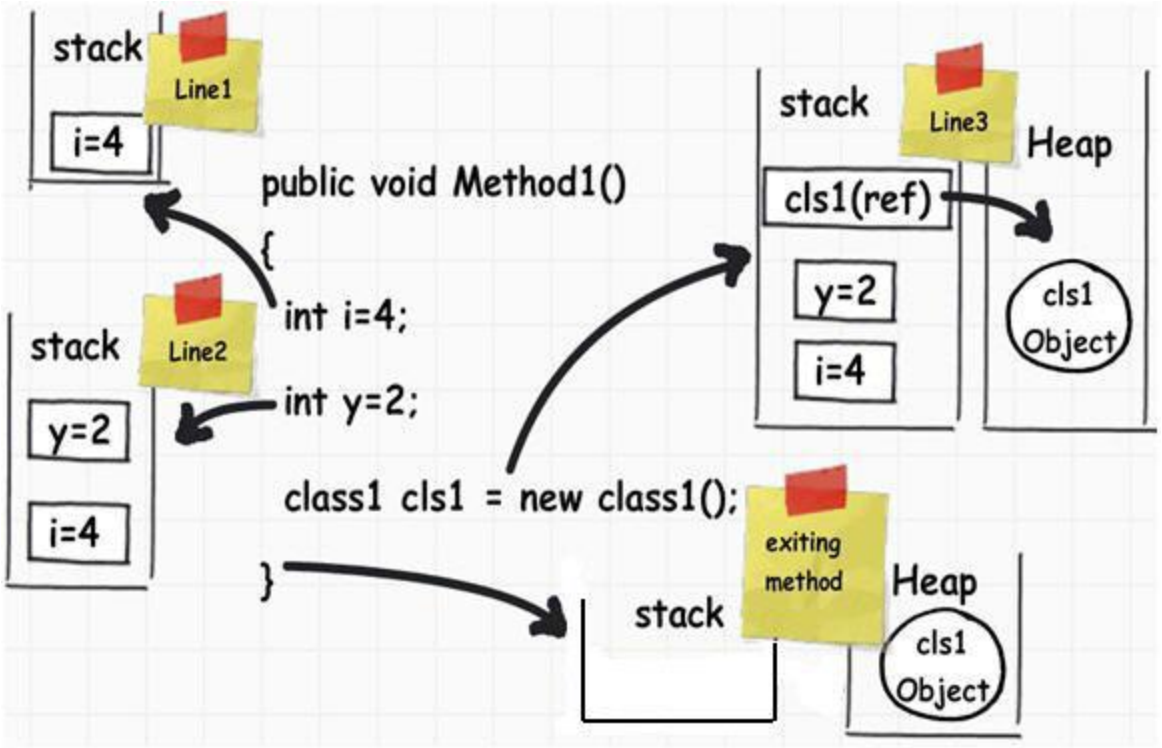


PASSING BY VALUE VS. REFERENCE

```
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;

?

Typical Embedded C

```
void main()
{
        dothis = 1;
        dothat = 2;
        while(1)
        {
                dothisoverandover = 1;
        }
}
```