An embedded system is a computer system embedded in another device.

Indicate which are embedded systems.

1)

Desktop PC

True

False

**Correct**

Not embedded in another device.

2)

Electronic drums (musical instrument)

True

False

3)

Elevator (lift)

True

False

**Correct**

Computers not only open/close doors and move the elevator, but they do so increasingly smoothly and quickly today.

4)

Cloud server

True

False

**Correct**

A server is a computer that provides data or computation via the Internet. They are big. Companies like Google and Amazon maintain thousands of servers in football-field sized rooms. [Wikipedia: Server farms.](http://en.wikipedia.org/wiki/Server_farm)

5)

Cardiac pacemaker

True

False

**Correct**

Not only does a pacemaker, which delivers electrical pulses to the heart, have an embedded computer, but the pacemaker itself is embedded in a human. Other embedded systems on or in humans include brain pulse delivery system for fighting migraines or seizures, automatic medicine delivery like insulin pumps, artificial limbs, etc.

Integrated circuits (a.k.a. ICs or chips), on which microprocessors are implemented, have been doubling in transistor capacity roughly every 18 months, a trend known as Moore's Law ([Wikipedia: Moore's Law](http://en.wikipedia.org/wiki/Moore%27s_law)).

1)

Moore's Law suggests IC capacity doubles about every \_\_\_ months.



**Correct**

18

Moore's prediction back in the 1960s was actually a doubling every 2 years.

2)

Consider a chip in a smartphone today. If Moore's Law holds, that chip in 6 years would hold how many times more transistors?



**Correct**

16

Capacity would double 4 times in 6 years, so 2\*2\*2\*2 yields 16. So in just 6 years, a smartphone could become 16 times more capable.

3)

Consider a chip in a smartphone today. If Moore's Law holds, the transistors within that chip in 12 years would occupy what fraction of the chip's current size? Write answer as a fraction in the form: 1 / 4



**Correct**

1 / 256

So a smartphone's chip would become almost too small to see. Some new products will become possible because of that shrinking.

A useful abstraction is to consider only two voltage ranges, a "low" range (such as 0 Volts to 0.3 Volts) that is abstracted to 0, and a "high range" (such as 0.7 Volts to 1.2 Volts) that is abstracted to 1. A bit (short for "binary digit") is one digit of such a two-valued item. A bit's changing value over time is called a digital signal.

A switch is an electromechanical component with a pair of electrical contacts. The contacts are in one of two mechanically controlled states: closed or open. When closed, the contacts are electrically connected. When open, the contacts are electrically disconnected.

A light emitting diode or LED is a semiconductor with a pair of contacts. When a small electrical current is applied to the LED contacts, the LED illuminates.

A microcontroller is a programmable component that reads digital inputs and writes digital outputs according to some internally-stored program that computes.

The example statement B0 = A2 && A1 && A0 sets the microcontroller output B0 to 1 if inputs A2, A1, and A0 are all 1. The while (1) { <statements> } loop is a common feature of a C program for embedded systems and is called an infinite loop, causing the contained statements to repeat continually. (LOOK AT CODE EXAMPLE 1 IN FOLDER)

The switch and buttons are examples of sensors, which convert physical phenomena into digital inputs to the embedded system. The LED is an example of an actuator, which converts digital outputs from the embedded system into physical phenomena.

1)

A button outputs 1 Volt when pressed, 0 Volts otherwise.

True

False

**Correct**

The actual voltage varies on the electrical setup, but a common abstraction is to consider a press as a 1 (not 1 Volts, just 1).

2)

An LED's electronics are designed to turn on when the input voltage exceeds 1 Volt.

True

False

**Correct**

The actual voltage varies depending on the LED. But an abstraction is to say that a 1 turns on an LED (not 1 Volts, just 1).

3)

A microcontroller performs computations that convert input values to output values.

True

False

**Correct**

A microcontroller carries out the computing part of most embedded systems.

4)

A microcontroller cannot be programmed.

True

False

**Correct**

In contrast, a key feature of a microcontroller is that a programmer can program a microcontroller to perform specific tasks, typically using a language like C or C++.

Set B0 to 1 when A0 is 1 and A1 is 1, or when A0 is 1 and A2 is 1. Ex: B0 = (A0 && A1) || (A0 && A2);

Modify the program to set B0 to 1 when the number of 1s on A2, A1, and A0 is two or more (i.e., when A2A1A0 are 011, 110, 101, or 111). Ex: B0 = (A0 && A1) || (A0 && A2) || (A1 && A2) || (A0 && A1 && A2);

1)

Pressing "Compile" translates the program code to executable machine code.

True

False

**Correct**

Compilation messages appear in the textarea at the bottom of RIMS.

2)

When a program is executing (by pressing Run), the values of inputs A0, A1, A2, ..., cannot be changed.

True

False

**Correct**

In fact, during execution is when the user will want to change the input values, to see how the program responds.

3)

#include "RIMS.h" is required atop all C files for RIMS.

True

False

**Correct**

RIMS.h defines many RIM-specific properties, such as definitions of A and B.

4)

The "Slowest" simulation speed causes a highlighting of the next C statement to execute.

True

False

**Correct**

Using the "Slowest" simulation speed may be useful to observe the program's execution.

1)

Pressing "Break" crashes the program.

True

False

**Correct**

Pressing "Break" temporarily stops program execution and highlights the next statement to execute. Pressing Continue resumes normal program execution.

2)

Pressing "Step" executes one C statement.

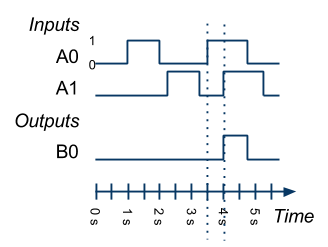
True

False

**Correct**

Specifically, pressing "Step" executes the currently highlighted C statement, then highlights the next C statement.

A timing diagram shows time proceeding to the right, and plots the value of bit signals as either 1 (high) or 0 (low). The figure below shows sample input values for the above example1.c program that continually computes B0 = A1 && A0. A0 is 0 from time 0 ms to 1 ms, when its value changes to 1. A0 stays 1 until 2 ms, when its value changes to 0. And so on. The 1 and 0 values are labeled for signal A0, but are usually implicit as for A1.



The timing diagram shows that B0 is 1 during the time interval when both A0 and A1 are 1, namely between 4 ms and 5 ms.

A change on a signal is called an event. Events typically refer to a one-bit signal changing from 0 to 1 or from 1 to 0, but can refer to changes on an integer (multi-bit) or other signal.

 If a signal changes from 0 to 1, the event is called rising. 1 to 0 is called falling. A pulse is a signal portion started by a rising event and ended by a falling event (looks like a camel hump).

1)

A0 is 1 at time 1.5 s.

True

False

**Correct**

A0 changed to 0 at 1 s, then back to 0 at 2 s.

2)

A0 and A1 are both 1 for just over 0.5 s.

True

False

**Correct**

A0 and A1 are both 1 starting at time 4 s, A0 falling back to 0 just before 5 s.

3)

The vertical dotted line indicates that B0 changing to 1 causes A1 to change to 1.

True

False

**Correct**

B0 is an output so can't cause an input to change. A1 changing to 1 might have caused B0 to change (to make that clear, an arrow could be drawn from the rising A1 signal at 4 s to the rising B0 signal).

4)

B0 exhibits three events: 0, 1, and 0.

True

False

**Correct**

An event is a change. B0 only changes twice: from 0 to 1 at 4 s, and from 1 to 0 at just before 5 s.

5)

A0 exhibits five pulses: 0, then 1, 0, 1, and finally 0 again.

True

False

**Correct**

A pulse is 1 preceded and followed by 0s. A0 only has two pulses.

Timing diagrams may include an arrow to indicate that a particular input event triggered or caused some output event, as shown below where each rise of A1 triggered a change on output B.



1)

The timing diagram indicates that shortly after time 2 s a rising event on A1 caused B to change from 19 to 23.

True

False

**Correct**

The arrow indicates that the A1 event triggered the output change.

2)

The timing diagram indicates that at around time 4 s a change on B from 23 to 0 triggered a rising event on A1.

True

False

**Correct**

An output event cannot cause an input event. The causality is the other direction, which is why the arrow points the other direction too.

3)

The timing diagram indicates that just after time 5 s the falling event on A1 caused B to change from 0 to 1.

True

False

**Correct**

Although the change occurs at that time, the timing diagram has no indication of whether A1's fall triggered the change. The change could have been coincidental.

4)

For the time during which B is 0 or 1, the timing diagram could have shown the signal as high or low, similar to how A1 is shown as high or low.

True

False

**Correct**

B represents an integer, not a bit or Boolean. Thus, drawing the signal as high or low just because the values happen to be 0 or 1 would not be appropriate, as an integer 0 or 1 has a different meaning than a Boolean 0 or 1.

Because testing usually can't cover all input combinations, testing should cover border cases, which are fringe or extreme cases such as all inputs being 0s and all inputs being 1s, and then various normal cases.

If code has branches, then good testing also ensures that every statement in the code is executed at least once, known as 100% code coverage.

To manually trace a program means to mentally execute the program, perhaps with the aid of paper and pencil.

A trace statement is a statement that prints information about a program's execution, such as what region of code is currently executing, or such as the value of a variable at some point in the program.

The basic method for printing a trace statement is to use printf("Trace text here.\r\n");. printf() supports arguments for printing the values of different data types, like char, int, etc.

Input value combinations, known as test vectors, can be described in RIMS rather than each input value combination being generated by clicking on switches.

The check output event is also known as an assertion. An assertion compares the B7-B0 outputs to the given expected value, and prints a warning if those values do not match. Assertion statements provide a mechanism for detecting when a program is not behaving as intended.

1)

A minimum testing requirement is to test all possible input combinations of a system.

True

False

**Correct**

Usually, too many possible combinations exist for such complete testing to be possible.

2)

A test vector is a particular combination of input values.

True

False

**Correct**

A programmer strives to create a good set of test vectors to test a program.

3)

A border case is a typical input value to a program.

True

False

**Correct**

A border case is an extreme or unusual value that could appear, such as all 0s, all 1s, or very large or small numbers. A programmer should think carefully about various extreme cases that might occur.

4)

A trace statement prints information about an executing program.

True

False

**Correct**

Adding temporary print statements is likely the simplest and most widely-used technique for debugging a program.