**Logo

Description automatically generated**

**San Francisco Bay University**

**CE450 Fundamentals of Embedded Engineering**

**2022 Fall Midterm Exam**

**Student Name: Mahmud H. Omer ID: 19660**

1. What are embedded system basic requirements? What is ARM processor architecture?

ANS:

1. Embedded systems basically require hardware such as **power supply, processor, memory, timers, serial communication ports and system application specific circuits** and software that work together to perform specific tasks.
2. The ARM microcontroller stands for Advance RISC Machine, and it’s a processor which is mainly used in portable devices such as digital cameras, mobile phones, home networking modules and [wireless communication technologies](https://www.watelectronics.com/different-types-of-wireless-communication-technologies/) and other [embedded systems](https://www.watelectronics.com/classification-of-embedded-systems/) due to the benefits, such as low power consumption, reasonable performance. ARM architecture The architecture processor is an advanced reduced instruction set computing [RISC] machine and it’s a 32bit reduced instruction set computer (RISC) microcontroller. As shown in the figure below, the ARM architecture comprises **Arithmetic Logic Unit, Booth multiplier Barrel shifter, Control unit and Registers.**

**Diagram, schematic

Description automatically generated**

1. Analyze the following code step by step, like running it in debugging mode in language compiler. And then give the result through line-by-line explanation

*def Ton(now):*

*then = 42*

*def no(know):*

*no = then*

*return know \* now(know)*

*return no*

*>>> then, no = 7, 4*

*>>> now = lambda oh: oh \* no*

*>>> ok = Ton(now)(no)*

*>>> ok*

*?*

***ANS:***

The program is execution is defined step by step as follows

Steps 1: define function Ton

Step 2. assign then to 7 and no to 4

Step 3: assign now to lambda function that returns oh\*no

Step 4: assign ok to the return value of Ton(now)(no)

Step 5. call function Ton(now) -> calling lambda function to obtain now

Step 6. Call function no(know) that returns know\*now(know)

Step 7. Function Ton(now) returns no which is

Below is the visualization of the above reduced execution steps description

Diagram

Description automatically generated

def  Ton(now):

  then = 42

  def no(know):

    no = then

    return  know \* now(know)

  return no

then, no = 7, 4

now = lambda oh: oh \* no

ok = Ton(now)(no)

ok

ok = Ton(now)(no) , now = lambda oh: oh \* no

then = 42

def no(know): know = no = 4

return know\*now(4)

4\*(4\*no)

4\*4\*4 =64

Logical explanation:

1. As above required, analyze the following code.

*woo = 6*

*def much(woo):*

*if much == woo:*

*such = lambda woo: 5*

*def woo():*

*return such*

*return woo*

*such = lambda woo: 4*

*return woo()*

*>>> woo = much(much(much))(woo)*

*>>> woo*

*?*

ANS:

woo = 6

def  much(woo):

  if much == woo:

    such = lambda woo: 5

    def woo():

      return such

    return  woo

  such = lambda woo: 4

  return  woo()

woo = much(much(much))(woo)

woo

woo = much(much(much)) (woo)

* woo = woo(woo(woo)) (6)

return 5

return 5

return 5

woo = 5

A picture containing diagram

Description automatically generated

1. What is the running result after analyzing step by step as required in question*#1*?

*def horn(hood):*

*horn = hood*

*def hood (horn):*

*return horn*

*return horn(hood)*

*hood = lambda horn: horn(2)*

*>>> horn (hood)*

*?*

ANS:

def   horn(hood):

  horn = hood

  def  hood (horn):

    return horn

  return horn(hood)

hood = lambda horn: horn(2)

horn (hood)

hood = lambda horn: horn(2)

* horn (hood)

-> horn(horn(2))

-> hood(2)

horn = hood = 2

returns horn =2

Diagram

Description automatically generated

1. What is the running result? And explain why as above

*pear = "ni"*

*def apple(banana):*

*def plum(peach):*

*pear = lambda pear: peach(pear)*

*return pear*

*return plum(banana)("ck")*

*>>>apple(lambda peach: pear + peach)*

*?*

ANS:

pear = "ni"

def   apple(banana):

  def  plum(peach):

    pear = lambda pear: peach(pear)

    return pear

  return plum(banana)("ck")

apple(lambda peach: pear + peach)

apple(lambda peach: pear + peach)

* Banana = lambda peach: pear + peach

-> plum(banana)("ck")

->plum(lambda peach: pear + peach) ("ck")

->pear = lambda pear: peach(pear)

pear = lambda pear: lambda peach: pear + peach

where pear = "ni" and peach = "ck"

returns "nick"

Diagram

Description automatically generated with medium confidence

1. What would Python print? And explain why as above

*x = "x"*

*g = x*

*def x(x):*

*g = "h"*

*if x == g:*

*return x + "i"*

*x = lambda x: x(g)*

*return lambda g: x(g)*

*>>> x = x(x)(x)*

*>>> x*

*?*

ANS:

x = "x"

g = x

def x(x):

  g = "h"

  if x == g:

    return x + "i"

  x = lambda x: x(g)

  return lambda g: x(g)

x = x(x)(x)

x

x = x(x)(x)

* x(x)("x") =x("x”) ("x")

returns x(“h”)

-> x == g :True as x = g = ‘h’

-> returns ‘h’ +’i’ = ‘hi’

Return value : ‘hi’

Diagram

Description automatically generated with medium confidence

1. Define a function with one argument like a positive number *x* and returns the powers of two, which is nearest to *x*. If *x* is exactly between two powers of two, return the larger.

*def nrst\_two(x):*

*"""*

*>>> nrst\_two(8) # 2^3 is 8*

*8.0*

*>>> nrst\_two (11.5) # 11.5 is closer to 8(=2^3) than 16(=2^4)*

*8.0*

*>>> nrst\_two (14) # 14 is closer to 16 than 8*

*16.0*

*>>> nrst\_two (2019) # 2^10 = 1024; 2^11=2048.0*

*2048.0*

*>>> nrst\_two (0.1)*

*0.125*

*>>> nrst\_two (0.75) # Tie between ½(=) and 1(=)*

*1.0*

*>>> nrst\_two (1.5) # Tie between 1 and 2(=)*

*2.0*

*"""*

*ANS:*

|  |
| --- |
| #ANS  def nrst\_two(X):     x = round(X)     for i in range(x):         if 2\*\*i >=X:           if (X -2\*\*(i-1))> (2\*\*(i) -X):              return 2\*\*i           elif (X -2\*\*(i-1))< (2\*\*(i) -X):              return 2\*\*(i-1)           else:              return 2\*\*i  nrst\_two(8)  nrst\_two (11.5)  nrst\_two (0.75)  nrst\_two (1.5) |

1. Create a function in recursion to check whether input argument is palindrome if it reads the same forwards and backwards.

*def is\_plndrm(n):*

*"""*

*>>> is\_plndrm (45654)*

*True*

*>>> is\_plndrm (42)*

*False*

*>>> is\_plndrm (2019)*

*False*

*>>> is\_plndrm (10101)*

*True*

*"""*

*ANS:*

|  |
| --- |
| def is\_plndrm(n):      word = str(n)      if len(word) < 2:          return True      if word[0] != word[-1]:          return False      return is\_plndrm(word[1:-1])  print(is\_plndrm(45654))  print(is\_plndrm(42))  print(is\_plndrm(2019))  print(is\_plndrm(10101)) |

*output:*

|  |
| --- |
| True  False  False  True |

1. Define a function "*has\_subls*" in recursive call with two lists as arguments, *ls* and s*ubls*, and returns if the elements of *subls* appear in order anywhere within *ls*.

*def has\_subls (ls, subls):*

*"""Returns if the elements of subls appear in order anywhere within list ls.*

*>>> has\_subls ([], [])*

*True*

*>>> has\_subls ([3, 3, 2, 1], [])*

*True*

*>>> has\_subls ([], [3, 3, 2, 1])*

*False*

*>>> has\_subls ([3, 3, 2, 1], [3, 2, 1])*

*True*

*>>> has\_subls ([3, 2, 1], [3, 2, 1])*

*True*

*"""*

*ANS:*

|  |
| --- |
| def has\_subls(ls, subls):      if (set(subls) & set(ls)) == set(subls):          return True      elif len(ls) != len(subls):          return False      elif len(ls) == 0:          return True      elif ls[0] != subls[0]:          return False      else:          return has\_subls(ls[1:], subls[1:])  print(has\_subls([], []))  print(has\_subls([3, 3, 2, 1], []))  print(has\_subls([], [3, 3, 2, 1]))  print(has\_subls([3, 3, 2, 1], [3, 2, 1]))  print(has\_subls([3, 2, 1], [3, 2, 1])) |