ECE3073 Computer Systems

Practice Questions

Program Design and Analysis: Validation and Testing

i) Explain the meaning of the following terms:

a) Black box testing

Tests are generated without knowledge of the internal structure of the program.

b) White box (clear box) testing

Tests are generated using a knowledge of the program structure.

ii) The following C code has been compiled into NIOS-II assembly code as shown below:

```c
int i, an_array[4];
for (i=0; i<4; i++){
        an_array[i] += i;
}
```

```
0x00040764 <main+28>:  stw    zero,-20(fp)       // init i
0x00040768 <main+32>:  ldw    r2,-20(fp)         // get i
0x0004076c <main+36>:  cmpgei r2,r2,4            // check for end
0x00040770 <main+40>:  bne    r2,zero,0x407bc <main+116>
0x00040774 <main+44>:  ldw    r2,-20(fp)         // get i
0x00040778 <main+48>:  muli   r2,r2,4            // i*4
0x0004077c <main+52>:  addi   r3,fp,-20          // address of i
0x00040780 <main+56>:  add    r2,r2,r3           // get array base -4
0x00040784 <main+60>:  addi   r4,r2,4            // calc actual base
0x00040788 <main+64>:  ldw    r2,-20(fp)         // get i again
0x0004078c <main+68>:  muli   r2,r2,4
0x00040790 <main+72>:  addi   r3,fp,-20
0x00040794 <main+76>:  add    r2,r2,r3
0x00040798 <main+80>:  addi   r2,r2,4
0x0004079c <main+84>:  ldw    r3,0(r2)
0x000407a0 <main+88>:  ldw    r2,-20(fp)
0x000407a4 <main+92>:  add    r2,r3,r2
0x000407a8 <main+96>:  stw    r2,0(r4)
0x000407ac <main+100>: ldw    r2,-20(fp)
0x000407b0 <main+104>: addi   r2,r2,1
0x000407b4 <main+108>: stw    r2,-20(fp)
0x000407b8 <main+112>: br     0x40768 <main+32>
```

1) What is the memory address of *i* ?

Answer: fp - 20

2) What is the memory address of *an_array[0]* ?

Answer: fp -16

3) What address range is the code for *i=0* ?

Answer: 0x00040764 = Main+28

4) What address range is the code for that implements the loop termination?

Answer: Main+32 to main+40 inclusive

5) What is register is used for the address of *an_array[i]*?

R4 at main+60
R2 at main+80

6) What simple optimisations can be applied to the assembly code?

Answer:

Remove code at main+64 to main+80 inclusive and
Replace code at main+84 with ldw r3, 0(r4)
Also can:
Replace code at main+92 with add ***r3***, r2, r3
Replace code at main+96 with stw ***r3***, 0(r4)
which eliminates the need for code at main+100
(since r2=i is preserved from main+88)
The branch at main+112 can branch to main+36

```
0x00040764 <main+28>:  stw    zero,-20(fp)      i=0
0x00040768 <main+32>:  ldw    r2,-20(fp)        r2 = i
0x0004076c <main+36>:  cmpgei r2,r2,4           test i>= 4
0x00040770 <main+40>:  bne    r2,zero,0x407bc <main+116>  exit if
i>=4
0x00040774 <main+44>:  ldw    r2,-20(fp)        r2 = i
0x00040778 <main+48>:  muli   r2,r2,4           r2 = i*4
0x0004077c <main+52>:  addi   r3,fp,-20         r3 = &an_array[0]-4
0x00040780 <main+56>:  add    r2,r2,r3          r2 = &an_array[i]-4
0x00040784 <main+60>:  addi   r4,r2,4           r4 = &an_array[i]
0x00040788 <main+64>:  ldw    r2,-20(fp)        r2 = i
0x0004078c <main+68>:  muli   r2,r2,4           r2 = i*4
0x00040790 <main+72>:  addi   r3,fp,-20         r3 = &an_array[0]-4
0x00040794 <main+76>:  add    r2,r2,r3          r2 = &an_array[i]-4
0x00040798 <main+80>:  addi   r2,r2,4           r2 = &an_array[i]
0x0004079c <main+84>:  ldw    r3,0(r2)          r3 = an_array[i]
0x000407a0 <main+88>:  ldw    r2,-20(fp)        r2 = i
0x000407a4 <main+92>:  add    r2,r3,r2          r2 = an_array[i]+i
0x000407a8 <main+96>:  stw    r2,0(r4)          an_array[i] = r2
0x000407ac <main+100>: ldw    r2,-20(fp)        r2 = i
0x000407b0 <main+104>: addi   r2,r2,1           r2 = i+1
0x000407b4 <main+108>: stw    r2,-20(fp)        i = i+1
0x000407b8 <main+112>: br     0x40768 <main+32> goto main+32
```

iii)

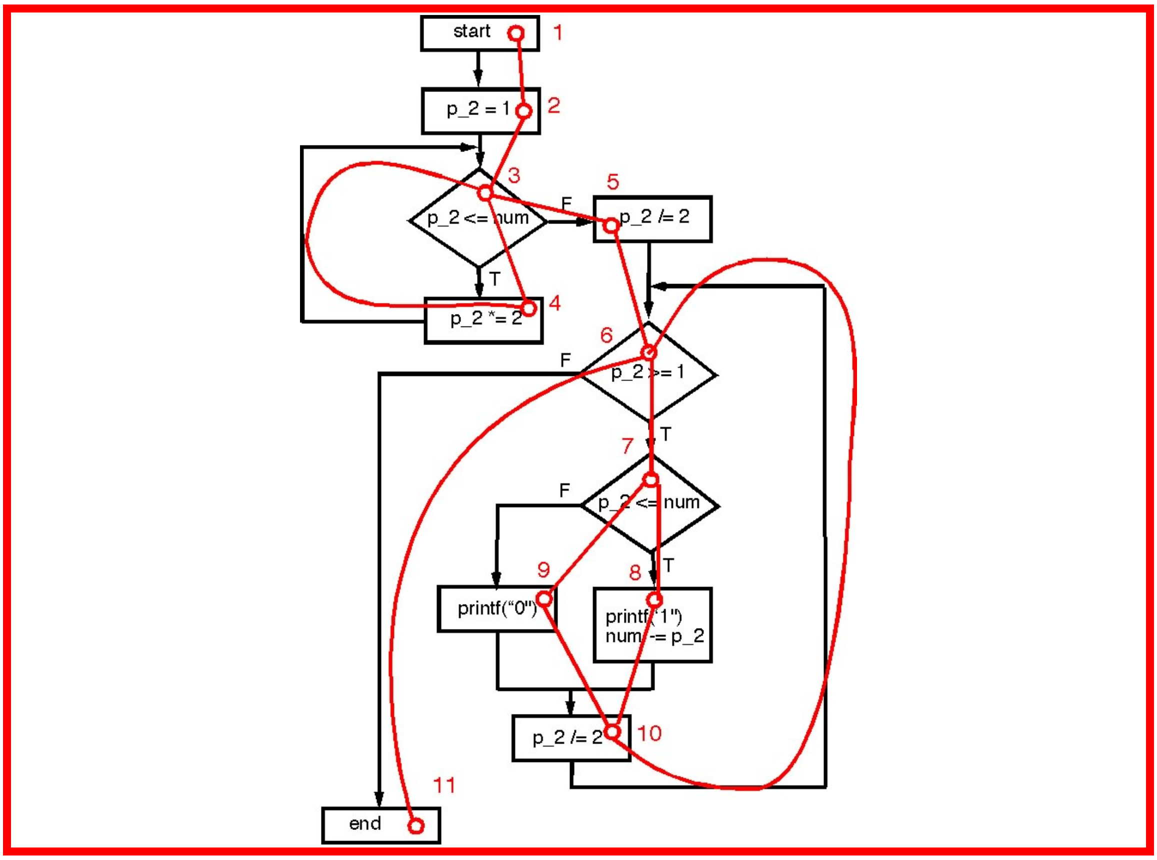
This question refers to the following C code. The questions follow the code.

```
//*****************************************************************************
// Function Binary – prints to cout a number in binary using a minimum number of bits.
// Input parameter(s): num – a non negative integer (ie >=0)
// Ouput parameter(s): none
// Returns: void
//*****************************************************************************
void Binary(int num)                              // 1
{
 int power_2=1;                                   // 2

 while (power_2  <=  num)                         // 3
   power_2  *= 2;                                 // 4

 power_2  /= 2;                                   // 5
 // power_2 is the largest power of 2 <= num
 while (power_2 >= 1)                             // 6
 {
    if (power_2  <=  num)                         // 7
    {
      printf("1");                                // 8
      num  -=  power_2;
    } else
    {
      printf("0");                                // 9
    } // if else
    power_2   /=  2;                              // 10
 } // while
} // Binary                                       // 11
```

1) Draw a CDFG flow graph next to the code above for the function *Binary* where each node in the graph is numbered and refers to sections of code that ***you label on the code above***. Any decision node must label the output paths as *true* or *false*.



2) Find the cyclomatic complexity of the function *Binary* shown above.

p = 1 (one connected part)
n = 11 (nodes)
e = 13 (edges)

M = e – n + 2p

M = 13 – 11 + 2 * 1 = 4 (upper bound on paths)

3) List a set of independent paths through your graph from Question 4.1 and for each path define the value of *num* that gives rise to that path.

Path 1: 1, 2, 3, 5, 6, 11 (num = 0)
Path 2: 1, 2, 3, 4, 3, 5, 6, 7, 8, 10, 6, 11 (num = 1)
Path 3: 1, 2, 3. 4. 3, 4, 3, 5, 6, 7, 8, 10. 6, 7, 9, 10, 6, 11 (num = 2)

4) Is the number of independent paths equal to the cyclomatic complexity in this case? If not explain why this is not the case.

No it is less, but cyclomatic complexity gives an upper bound

5) By examining the *printf* output corresponding to each path in your answer above, identify an error in the code for the function *Binary*. Note that all comments can be

assumed to be free of errors. Correct the error using one *if* statement with one predicate and an existing statement. That is change one line from

line_of_code;

to

if (*some_predicate) line_of_code;*

5: if(num!=0) p_2 /= 2