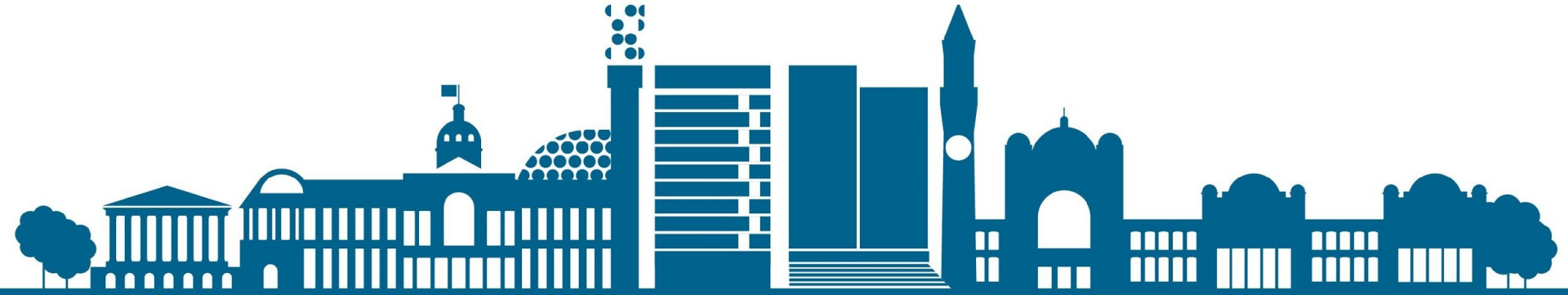
UNIVERSITY OF BIRMINGHAM

Computer Systems (2023/24)
Revision Lecture – Consolidation Week

Dr. Mian M. Hamayun
m.m.hamayun@bham.ac.uk



Module Mid Term MEQs [Dubai]

https://canvas.bham.ac.uk/courses/70900/quizzes/191241

Quiz 1 - Q5

Q5.1	Which numbers could the byte $(10001010)_2$ represent on an eight-bit computer, using unsigned, sign-and-magnitude and two's complement representations?
(a) ✓	**138**
(b) ✓	**-10**
(c) ✓	**-118**
(d)	-138
(e)	10
(f)	118
(g)	142
(h)	-142

Feedback:

When considered as an unsigned binary number, 10001010 represents the decimal number 2^7 + 2^3 + 2^1 = 138.

When considered as a signed binary number, 10001010 represents the decimal number -(2^3 + 2^1) = -10.

When considered as a two's complement binary number, 10001010 represents the decimal number -2^7 + 2^3 + 2^1 = -118.

Quiz 1 - Q6

Q6.1	Calculate the subtraction of the following two's complement fixed-point binary numbers: $(11001.001)_2 - (00001.100)_2$
(a)	**10111.101**
(b)	11010.101
(c)	10111.100
(d)	01000.011
(e)	01000.010

Feedback:

First we flip every bit and add 1 to the LSB of the second argument of the subtraction.

Second argument:	00001.100
Flip:	11110.011
Add 1 to LSB:	11110.100

Then we simply add this and the first argument:

```
   11001.001
 + 11110.100
-------------
(1)10111.101
```

Quiz 1 - Q7

Q7.1

Consider the following MIPS assembly code:

```
    lw $1, &a
    lw $2, &b
L1:
    bne $1, $2, L2
    sw $1, &a
    j exit
L2:
    bgt $1, $2, L3
    sub $2, $2, $1
    j L1
L3:
    sub $1, $1, $2
    j L1
exit:
```

Assuming initial values of a = 24 and b = 18, what will be the final values of a & b when the code reaches exit?

Quiz 1 – Q7 (2)

Feedback:

The above code implements the GCD algorithms, as given below:

```
int gcd(int a, int b)
{
      while (a != b)
            if (a > b)
                  a = a - b;
            else
                  b = b - a;
      return a;
}
```

When a=24 and b = 18, the code gives the final values of a = 6 and b = 18, where a is the GCD of the two input values. The table below shows how the values will be updated:

$1	24	6	6	**6**
$2	18	18	12	6

We only write back the value for $1 to **a**, therefore **b** still has the original value of 18.

Quiz 1 – Q9

Q9.1	In a hypothetical computer, real numbers are represented as 11-bit floating point binary numbers in the following way: • The **first** bit is the sign bit of the number (i.e. 1 is negative) • The next **three** bits are the exponent, which is represented in **two's complement** and **not as a biased/offset number** • The final **seven** bits are the magnitude of the number On such a computer, how would the number **-3.328125** be represented?
(a) ✓	**10011010101**
(b)	10011110101 *(junk)*
(c)	11101010100 *(assumed it was a fixed point number)*
(d)	10101101010 *(forgot about the hidden 1.)*

Feedback:

The decimal number -3.328125 would be represented in fixed-point binary as 11.010101.

We then normalise this number (meaning that it starts with "1."), which gives us 1.1010101 x 2^1.

This means the exponent is 1, which is represented in 3-bit two's complement as 001, and the magnitude is 11010101.

Finally, the sign bit is clearly 1 as the number is negative.

Therefore, the number is represented on the hypothetical computer as 1 001 1010101.

Quiz 1 – Q10

Q10.1	A processor is operating at 50MHz. Each instruction takes a minimum of 8 cycles to execute. The processor has an 8 stage pipeline. If a program starts execution at time 0, what is the theoretical maximum number of instructions that will have completed their execution at the end of 10 milliseconds?
(a)	62,500 instructions
(b)	49,993 instructions
(c)	499,930 instructions
(d)	499,992 instructions
(e)	**499,993 instructions**

Feedback:

Speed = 50MHz = 50,000,000 cycles/second = 50,000 cycles/ms

The processor will execute 500,000 cycles in 10 milliseconds

With pipelining:

Number of clock cycles taken by the first instruction = 8 cycles.

After the first instruction has completely executed, one instruction comes out per cycle.

So, the number of cycles taken by each remaining instruction = 1 cycle.

Number of executed instructions = 1 + (500,000 cycles – 8 cycles)

= 1 + 499,992 = 499,993 instructions

In practice, there are factors that mean that this theoretical maximum is never reached.

2019 – Q1(b)

$(-29.5)_{10}$ ✓

$(+13)_{10}$ $(26)_{10}$

(b) The table below contains a number represented in floating point notation using 16 bits. What decimal fraction does it represent? **[6 marks]**

Sign	Exponent	Mantissa
1	10011	11 0110 0000

offset $= 2^{n-1} - 1$

$\Rightarrow 2^{4} - 1 \Rightarrow 15$

$1.11\,0110\,0000 \times 2^{4}$

2019 – Q1(b) [Answer]

Sign = –ve
Offset = $2^{n-1} - 1 = 2^{5-1} - 1 = 15$
Exponent (with offset) = 10011 = 19
Exponent = 19 – 15 = 4
Fraction = 1.1101100000

So the binary fraction (in standard notation) is:
$-1.11011 \times 2^4 \Rightarrow -11101.1 \Rightarrow$ **-29.5**

11101.1

↓

−29.5

2019 – Q2(b)

(b) Consider the following IJVM instructions:

Hex	Mnemonic	Meaning
0x10	BIPUSH *byte*	Push byte onto stack
0x59	DUP	Copy top word on stack and push onto stack
0xA7	GOTO *offset*	Unconditional branch
0x60	IADD	Pop two words from stack; push their sum
0x7E	IAND	Pop two words from stack; push Boolean AND
0x99	IFEQ *offset*	Pop word from stack and branch if it is zero
0x9B	IFLT *offset*	Pop word from stack and branch if it is less than zero
0x9F	IF_ICMPEQ *offset*	Pop two words from stack; branch if equal
0x84	IINC *varnum const*	Add a constant to a local variable
0x15	ILOAD *varnum*	Push local variable onto stack
0xB6	INVOKEVIRTUAL *disp*	Invoke a method
0x80	IOR	Pop two words from stack; push Boolean OR
0xAC	IRETURN	Return from method with integer value
0x36	ISTORE *varnum*	Pop word from stack and store in local variable
0x64	ISUB	Pop two words from stack; push their difference
0x13	LDC_W *index*	Push constant from constant pool onto stack
0x00	NOP	Do nothing
0x57	POP	Delete word on top of stack
0x5F	SWAP	Swap the two top words on the stack
0xC4	WIDE	Prefix instruction; next instruction has a 16-bit index

Convert the following assembly code into Machine Code using the above instructions table. For variables a, b and r you can use 0x01, 0x02 and 0x03 as variable numbers. The address of first instruction is 0x00.

Use the given format on the next page to provide your answer. **[6 marks]**

2019 – Q2(b)

```
        iload a
        iload b
        isub

        iflt else

        iload b
        istore r
        goto end_if

else:
        iload a
        istore r

end_if:
        iload r
        ireturn
```

Answer Format:

IJVM Instruction	Instruction Address	Machine Code

2019 – Q2(b) [Answer]

JVM Instruction	Instruction Address	Machine Code
iload a	0x00	0x15 0x01
iload b	0x02	0x15 0x02
isub	0x04	0x64
iflt else	0x05	0x9B 0x08
iload b	0x07	0x15 0x02
istore r	0x09	0x36 0x03
goto end_if	0x0B	0xA7 0x06
iload a	0x0D	0x15 0x01
istore r	0x0F	0x36 0x03
iload r	0x11	0x15 0x03
ireturn	0x13	0xAC

0x0D-0x05
=
0x08

0x11-0x0B
=
16^1+16^0 - 11
=
17-11
=
0x06

2019 – Q2(c)

x y − z ^ w

(c) Apply the Dijkstra's Shunting-Yard Algorithm to convert the following expression into Reverse Polish Notation: **[6 marks]**

⇒ (x – y) ^ z + w

Use the following format to show your working:

Output	Stack (Top on Right)	Input
	(	(

x

x

x y

x y −

((− (∅

→ x → − → y →) → ^ → z

2019 – Q2(c) [Answer]

Output	Stack (Top on Right)	Input
Empty	Empty	(x – y) ^ z + w
Empty	(	x – y) ^ z + w
x	(	– y) ^ z + w
x	(–	y) ^ z + w
x y	(–	)
x y –	Empty	^ z + w
x y –	^	z + w
x y – z	^	+ w
x y – z ^	+	w
x y – z ^ w	+	Empty
x y – z ^ w +	Empty	Empty

2019 – Q3(b)(i)

(b) Indicate the time complexity (with respect to N) of the following functions from the given choices? Justify your answer.

(i) Function #1 **[3 marks]**

```
procedure D(S : string)
    int N :int := length(S)
    int k := 0
    for i := 1..N
        k += 1
    end for
    for i := 1.. N
        for j := 1..N
            k += 1
        end for
    end for
    for i := 1..N
        for j := 1..N
            k += 1
        end for
    end for
end D
```

$O(N^2)$	$O(N)$
$O(\log N)$	$O(N \log N^2)$
$O(N \log N)$	$O(1)$
infinite	$O(2^N)$

None of the above

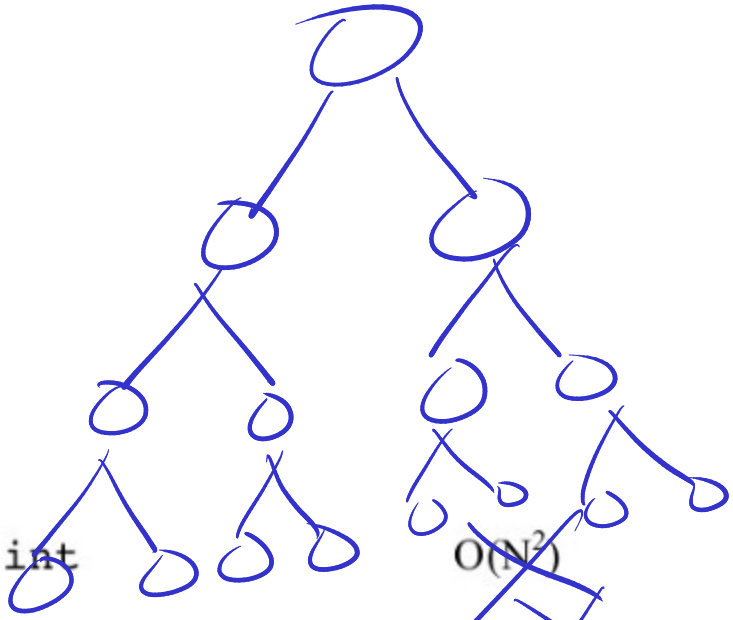
2019 – Q3(b)(i) [Answer]

Complexity for Function # 1: $O(N^2)$
Justification: The function contains 5 for loops, however only 2 of these are nested. Therefore, the time-complexity is $O(N^2)$ as this indicates the maximum required time for this function.

2019 – Q3(b)(ii)

(ii) Function #2 **[3 marks]**

```
function Rec2(N : int) : int
    if N == 1 then
        return 1
    else
        Rec2(N-1)
        Rec2(N-1)
    end if
end Rec
```

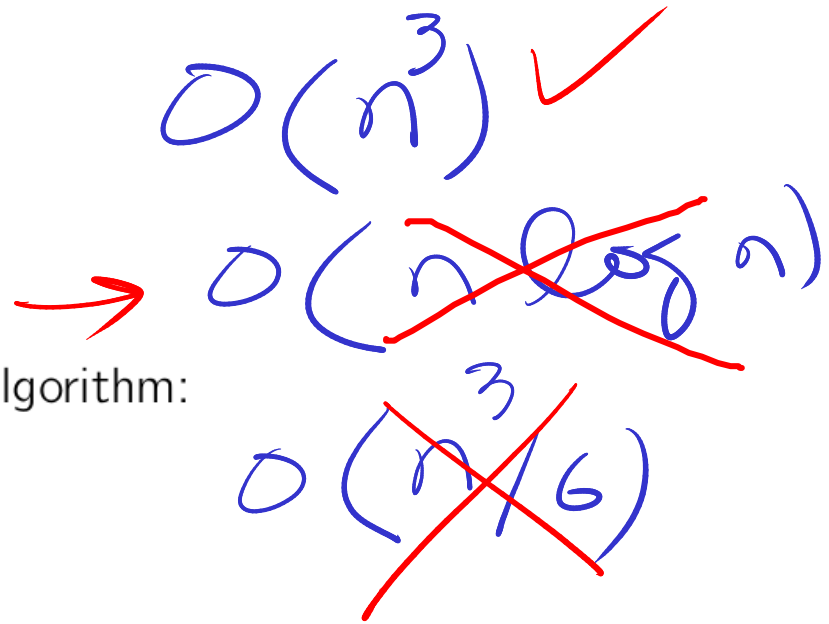


- $O(N^2)$
- $O(\log N)$
- $O(N \log N)$
- infinite
- $O(N)$
- $O(N \log N^2)$
- $O(1)$
- $O(2^N)$
- None of the above

2019 – Q3(b)(ii) [Answer]

Complexity for Function # 2: $O(2^N)$
Justification: The function contains 2 recursive calls for each of its calls. It creates a binary tree-like call structure i.e. 1 call leads to 2 calls, which in turn lead to 4 calls and so on. Therefore the time-complexity is exponential.

2020 – Q1(b)



(b) You have been given the following algorithm:

```
for i=1 to n
    for j=1 to n/2
        for k=1 to n/3
            a=a+i*j*k
```

(i) Derive and justify its time complexity using 'Big-O' notation

2020 – Q1(b)

$O(n^3)$ 250ms?

(ii) You have been given the following execution times for a program which implements this algorithm. The program has two parts. The first is constant and independent of n – a fixed overhead, the second implements the algorithm above.

N	250	500
Execution time	110 msec	180 msec

1000 ?

Estimate and justify how long it would take to execute for n=1000

T1 = n^3 = 110ms
T2 = (2n)^3 = 180ms
T2 - T1 = 7n^3 = 70ms
n^3 = 10ms
fixed time = T1 - n^3 = 110 - 10 = 100ms
T3 = (4n)^3 = 100 + 64*10 = 740ms

You can use stack or memory to store parameters,
but you cannot control Cache, it it controlled by hardware.

2020 - Q2(b)
(b)(iii)
Context-switch between processes:
1. I/O process
2. Run of out time
3.

Actions:

Advantages: switch more often, although we have more overhead, the response will be quicker.
Disadvantages: switch less often, we can save some time on overhead, but the response will be slower.

In computing and various technical contexts, "overhead" refers to the additional resources or work required beyond the essential or primary task.