# UoL logoUniversity of London

# Assessment Coversheet

Complete this coversheet and read the instructions below carefully.

**Candidate Number**:

WP1282

**Degree Title**:

BSc Computer Science

**Course/Module Title**:

How computers work

**Course/Module Code:**

CM1030

**Enter the numbers, and sub-sections, of the questions in the order in which you have attempted them:**

**Question 2**: a, b, c

**Question 3:** a, b, c

**Date**: 16.03.2021

**Instructions to Candidates**

1. Complete this coversheet and begin typing your answers on the page below, or, submit the coversheet with your handwritten answers (where handwritten answers are permitted or required as part of your online timed assessment).
2. Clearly state the question number, and any sub-sections, at the beginning of each answer and also note them in the space provided above.
3. For typed answers, use a plain font such as Arial or Calibri and font size 11 or larger.
4. Where permission has been given in advance, handwritten answers (including diagrams or mathematical formulae) must be done on light coloured paper using blue or black ink.
5. Reference your diagrams in your typed answers. Label diagrams clearly.

**The Examiners will attach great importance to legibility, accuracy and clarity of expression.**

**QUESTION 2**

1. A CPU is the main part of any computer and can be labelled the “brain” of the computer. But even the brain can’t do all work alone. A computer bus serves this role, connecting different parts of the computer so they can all work together (this is the logical level of abstraction). From a hardware perspective (physical level of abstraction) a computer bus nothing short of a bunch of wires that connect different pieces of hardware. There are different kinds of busses, but one of the most prominent ones is the system bus. A system bus is a single computer bus that connects 3 main parts of the computer: a CPU, a memory, and peripheral devices (using controllers). A system bus also combines 3 major buses: an address bus, a control bus, and a data bus. Figure 1 illustrates the basic idea.

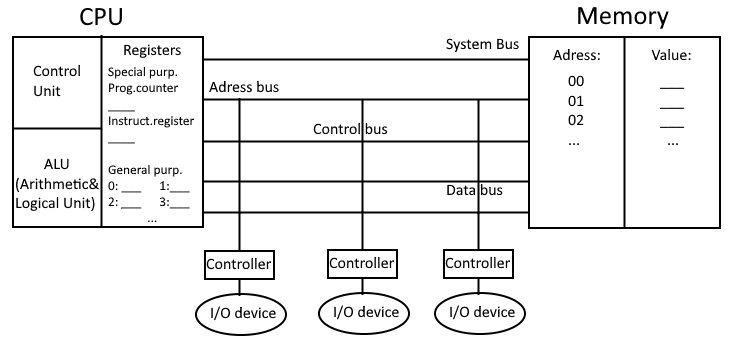


Figure 1 – System bus diagram

Through the address, bus CPU tells where to look for the data or program instruction in memory (In the CPU this information is stored in the special purpose register – program counter, with the use of DMA I/O devices, now can also use this bus to access memory w/o the CPU). The address bus is unidirectional – information flows from the CPU to the device. Using the control bus CPU can transmit commands to memory or another device and receive status information back from them. This bus is also used to synchronize operations with the CPU clock. And data bus allows the transfer of information between CPU and other devices, stored as values in memory cells. Address and data busses are bidirectional. Let us now present few examples of the use of the system bus:

The core of the computer's inner workings is the fetch-decode-execute cycle. Each cycle machine has some address stored in the program counter (PC). For example, let this be address “01”. The CPU then sends a fetch signal (control bus) to the memory using this address to retrieve the value residing in the memory cell “01”. The Memory sends the value (for example “F1”) of the desired address back to the CPU using the data bus. After that CPU increments the program counter (“01”->”02”) and starts decoding the instruction “F1”. In our example “F1” means “LOAD the register (general purpose) 1 with the bit pattern “01”. After decoding, this new instruction is loaded into the instruction register and executed, resulting in the value “01” in register “1”. After one machine cycle is complete, the next one begins, and cycles go on. This is but a simple example, that only involves CPU and memory. But, if in our example CPU utilized cache memory (fast copy of the small portion of the main memory within the CPU itself), then the whole process could be done only by CPU (and much faster) and its internal resources, without the need to interchange via the system bus.

Let us look at a more sophisticated example using all components. The CPU starts another cycle and fetches and decodes new instruction (look at prior example). This time this instruction is a request to save some data from the given hard drive cell (“FF”) to some cell (“11”) in memory. The CPU then executes this instruction in two steps: first step – LOAD some register (“3”) with the value from the Hard drive cell “FF” and then – STORE the value of the register “3” to the memory cell “11”. In both parts, instructions are sent via control bus, addresses for these instructions - via an address bus, and data values – by the data bus. This operation involves the CPU, the main memory, and the mass storage device (hard drive). The three-way operations involving several devices are relatively slow, and if in our example hard drive used DMA, it would save time and free system bus, because hard drive and memory could interchange data without the use of CPU. This last example also provides good insight into why this architecture becomes obsolete - the system bus becomes overcrowded (this is known as Von Neumann bottleneck).

1. Virtual memory is a technique that an operating system (OS) practices allowing the user to work exceeding his real random-access memory (RAM) limit. If applications that the user runs use up more memory addresses, than there are in his RAM and he doesn’t have/use virtual memory (VM) his computer is prone to crashing, because programs will try to access addresses that are not even there. To alleviate this problem, the operating system can use different techniques that are collectively known as “Virtual memory”. The first one - “Paged memory” (that’s why VM is often called paging) uses chunks of the same size called “page files”, while the second – “segmentation” uses chunks of the different size called “segments”. Both this method is often used in combination. The OS creates a mapping between addresses required and addresses available. When the required address is out of range of RAM, then the oldest “page” or “segment” is transferred to the hard drive, the mapping table is updated, and RAM space becomes free for further use. Virtual memory is an essential technique that has several pros and cons. The main advantage of the VM is that it bypasses the limit of the available RAM, allowing the user to continue work even if his computer doesn’t have enough memory to run big and/or multiple applications. A mass storage space is also much cheaper than RAM. Also, VM allows different applications to map their addresses to the same address in RAM, providing shareable resources and an efficient way to use the memory. But there are still some drawbacks, mainly – “thrashing”. When there is the active use of the memory pages stored onto the hard drive then the system is overloaded with multiple requests to move data back and forth between hard drive and memory, which causes massive loss of productivity (reasons in Q2-(a)). Virtual memory can be very helpful when I have to use an outdated PC to work with big data files, that are mostly read-only. In this case, a small RAM will be backed up by a cheap magnetic HDD to provide the means to work without visible loss of efficiency. But on another hand, if my OS will use my new SSD to page 100 of active "Chrome" tabs that I often switch between, I will soon find out that It was much cheaper just to buy another 32Gb of RAM to avoid slowdowns than to replace the expensive flash drive.
2. There are a lot of different devices out there, that use different techniques to carry the same task. Because we are not given specific examples, we will try to generalize the principle, using the concept of abstraction, adding small pieces of information that present different ways to implement said context.

Let us break this process into three main parts that are present in different devices: getting input data, processing it, and creating output for the user. There are different ways to calculate calorie burn, but they all use the same principle. They combine some general data and user-specific parameters to calculate the output. More sophisticated and expensive devices use more different user-specific data and more advanced methods (like regressions using multiple variables) to make more accurate calculations, while the cheap ones can use simple linear approximation with one variable. The input is also not the same. Some data is constant (gender, blood type, etc.), some is semi-constant (height, body type, etc.), some vary from day to day (weight, etc.), while other data can vary almost every minute (heart rate, body temperature, walking distance, etc.). In this example, we will assume that the aforementioned smartwatch uses both types of data. To collect the dynamic data, it will most certainly use some type (or multiple) sensors. Sensors are pieces of hardware (sometimes software) that collect physical data, store it and send it to some form of the processing unit. A good smartwatch almost certainly will collect body data, like heart rate and body temperature (sensors on the wristband) and activity data, like velocity and movement distance (using a built-in pedometer and accelerometer). This data is collected, updated, and sent to the watch's CPU in real-time (or near-realtime). As for the non-dynamic data, almost all watches come with user profiles. So user manually enters information about his gender, height, weight, and so on, when the watch is being set up.

The modern sports watch is all but a miniature personal computer, so it has all basic components - a CPU, volatile memory, mass storage device, and operating system. Non-dynamic and "old" dynamic data are most likely to be stored in the long-term memory because it is rarely updated and retrieved only when some advanced calculation is being cared out, or when the user is checking the history. "Up-to-date" dynamic data almost certainly passed from the memory to the CPU to create some "approximation" calculation on the fly and show the user average calorie burn. Also, we should mention exercise data that sports watches include. This watch must include some form of database that contains the list of exercises with "equations" that include some general statistical data in the form of parameters and input data as variables to estimate the calorie burn. For example, if I chose "squats" the watch's CPU will select the right equation from the database B = C + V1 \* A1 + V2 \* A2 +... + e. Where C is some constant, from the general data, A1, A2, ... - are parameters (example: 1.035, -0.389, ...) associated with variables V1, V2, ..., (example: avg. heart rate, body temperature, ...) and e is statistical value for the error. Parameters and statistical values are built into the exercises database, variables values are collected from the sensors and constant is calculated based on the user gender, height, etc. All these calculations are performed by the watch's CPU, using binary operations on the data fetched both from the RAM and long storage device.

We talked about hardware, now let us mention software. Watches software is similar to that of the PC, so it also has an operating system. This OS also has 2 layers - kernel, that control inner workings (like file manager, memory manager, drivers of the different sensors, etc.) and user interface, most likely in the form of the graphical user interface (GUI).

This is an example with a smartwatch, but we can see, that this idea will advance further into a "system" that consists of several devices, similar to this watch. These devices can combine functions of different accessories (like a walkman, a headband, etc.) and sensors. They will connect (using wireless technology like Bluetooth or WiFi) to the smartphone as to the central unit, utilizing more powerful hardware, allowing more flexibility and precision, and further developing the user's body area network (BAN).

**QUESTION 3**

1. First of all, let us look at the photograph from the perspective of logical abstraction. Photography is graphical data, and as any other data, is represented in the computer as a collection of primitive components. For the images, a computer uses colored dots. We should note that there could be only two colors (black and white photo or monochrome image) or millions of colors (color photos and images use many tones, as different combinations of the main 3 colors: red, green, and blue). These simple colored dots are known as pixels (short from "picture elements") and the image is represented as a collection of pixels if called "pix-map" or "bitmap" (we will explain this a little bit later).

Besides the collection of pixels themselves, pixmap also contains image information. Size of the picture/photo, like width and height in pixels. For example, size: 4000\*3000 means width of 4000 pixels and height of 3000 pixels. Another important piece of information is the color pallet used. As we mentioned before, each color can be coded as a combination of 3 main colors (for example, color red coded as a combination of Red:255, Green:0, and Blue:0 or [255,0,0]), or as a set of brightness (called "luminescence" - Y') and two "chrominance" components - U and V(for example, the color green is coded like Y':100, U:100, V:100). Last, but not least - the color depth, as the number of bits per pixel. And now, we delve a little deeper, to the physical level of abstraction. As for all data, a computer uses binary to interpret these colored dots.

Bitmap essentially just a long bit stream, that encodes images in the table fashion, where the location of every pixel in the image is set as row and column position. So, with this in mind, the color depth is just a number saying how much information an image/photo uses to represent a single color. For example, depth 8 bits/pixel means that the computer will use 2^8 = 256 tones to represent a color. If we use the RGB model, this means that the color of each pixel is a combination of 256 tones of red, 256 tones of green, and 256 tones of blue. Higher dimensions mean more details and more quality, same for the color depth, but the size of the photography will also become bigger.

So, to tie it all together: on the physical level photography is just a long string of binary numbers. On the logical level, this binary is translated by the computer into information about the dimensions of the photo, the color palette used, and the table of pixel values. The intersection of the row and column is the position of each pixel, and the value in the table is the color. And on the next level, the software will use this data to translate this information into photography on the screen.

1. The Internet is a network of networks and all smaller networks can be very different. Computers use different sets of universal rules, called protocols, to communicate between different networks. There are different "layers" in Internet communication, the formal OSI model uses 7 layers, but in this example, we will use a simple 4 layers model, that uses all essential levels: application layer, transport layer, network layer, and link (or data link) layer. On each level, several protocols regulate efficient transmission. Let us explain a little bit more about each of them:

From the name, we can understand that application layer protocols mostly regulate communications in the context of the task, specific for different applications. Because modern PC-s are built around interactions with the user, most Internet transmissions are also starting from this layer. Some protocols define major internet activities, like IMAP/SMTP/POP3 for the message exchange, FTP for the files, HTTP/HTTPS for the web browsing (exchange of the hypertext documents), DNS for the routing, Telnet for the VoIP, SSH for remote control and many others. HTTP (short for HyperText Transfer Protocol) is used by special applications (browsers) to exchange requests between the user computer (called a client) and another machine that hosts desired information (called a server). The browser on the client's computer sends the request to the server to get some web document (encoded using HTML). HTML is the text format, that uses special codewords called tags, and combines this text with hyperlinks - connectors to other web documents or files. If this request is acknowledged and approved, the server will respond, sending a web document to the client computer, that will be visualized in the browser (using tags). HTTPS is just an extension of the HTTP protocol that uses encryption to secure the connection between a client and a server.

To carry application requests over the Internet, a computer uses Transport level protocols. They break the initial request into smaller chunks called packets, suitable for Internet transportation, mark them, so the server will be able to reverse the process. In some sense, protocols of this level (TCP and UDP) work as an envelope. Both TCP and UDP are similar in what they do, but with a small difference. Before sending a message further, TCP sends its request, to make sure, that a connection to the server can be established. When the client computer receives the acknowledgment, it sends packets further, while UDP sends them straight without response. TCP is "slower", but more reliable, but both protocols find their uses (for example "on-line" services like VoIP and streaming rely on the UDP more).

From the application level, each message carries the address of the destination. Packets carry this information to network-layer protocols. The most popular protocols of this layer are IP (IPv4 and IPv6). These protocols use the web address (URL) and forwarding tables (the data table that matches the URL to the IP address in the form of the bit stream) to create a route for the packet to reach its destination. When the route is established, the packet is forwarded next, to the last layer. This protocol is used by routers, to ensure that each message will find its destination.

Datalink layer protocols a related to the transition between different network types. The two main protocols of this layer are Ethernet and Wi-Fi. They related to the different types of physical connections (wires vs radio waves) and network architecture (bus vs star). Ethernet protocol defines how the message finds its final destination in hub-type networks, connected through bridges, repeaters, and switches using wires.

When forwarded, message hops from intermediate points in designed routes, using different protocols for different inner networks, using the network and link protocols to continue its journey between routers on the way to its goal. After that, the message will go back to the transport level, where all packets are composed back to the initial message, sent back to the application layer.

1. In the first step, we need to connect to the web store page. We can access the website directly (if we already know the website) by typing the address directly into the address bar of the browser or start by using a search engine, to find different websites where we can buy cloth. In case we use direct access, we almost certainly will enter a URL address (for example https://www.amazon.com/) of the shop, and not the IP address. Using a search engine is similar, we just add one intermediate step, requesting a web store address from the search engine's server.

It is easy for the human to remember the address in the text form, then in the form of a number stream. When we click to follow this address, the browser will create an HTTPS request (in the case of web commerce security is one of the top priorities). This request will go through all Internet layers and use several different protocols to create a request, to establish a connection, to retrieve an IP address from the URL and create the route to the server, and how to handle different networks(Q3(b)). This request reaches the amazon server and asks for the store webpage. Here, this request will be handled based on the setup that the server uses. Popular setups, called stacks, include different pieces of hardware and software and can vary in how they handle the request, but the basic principle is the same. All web commerce sites have a database, server software that handles incoming requests, and a mechanism in which they respond to these requests. There are different types of databases, like relational or non-relational databases, but they all store information about products. Information like name, dimensions, description, links to the assets files (like images or maybe even video or audio samples), and others. All web commerce sites store information about their clients, also using databases. Server software gets HTTPS request from the client, decodes it, and starts preparing the answer. All modern web stores use page templates, that filled with data from the database, depending on the contents of the request. Some stacks generate the requested page, on the server machine, and send it back in the answer to the HTTPS request (LAMP one of the most popular server-side stacks). Others send back the page template and data from the database, together with the set of instructions to the client. Using these instructions template will compile the page, that was requested on the client's machine (servers using MEAN stack follow a similar algorithm). Both stacks find their uses, but in modern days, client-side interaction becomes more and more dominant. There are several reasons for that. First of all, a big online store like amazon handles millions of requests in a minute, and to create all responses on the server-side is very cumbersome. Using client-side server architecture, can alleviate the load and free some resources. And the second reason is that often we don't need to send the whole new page to the user, just some new data from the database, so the user will also not experience any discomfort. In our example, Amazon uses client-side interactions, so its stack is akin to the MEAN.

This is a typical example of how a client-server model handles a single request. But buying a product requires multiple requests, which result in different pages. But the whole procedure of buying can be broken down into several similar states. Searching product state: we make requests to the database, using some form of filters to narrow down our preferences. The server will respond with pages that look like lists of items that can interest the customer, based on his request. When we narrow down our search to a single item, we load the product page. This is the next stage of the website. If we don't like the item, we return and repeat steps 1 and 2 until we decide to buy something. When we add an item to the cart, we can continue steps 1-3, or go to checkout. After we ready to buy, we need to login. This state requires creating a secure connection because user data contains a lot of private sensitive information. If the server confirms that our login information valid (it checks input against information in the customer's database) we continue to the "logged in" state. If there is no information about our delivery address or billing details in the account, or we want to change them, we will be asked to fill in this information. After everything is ready we send the final request, confirming the transaction.

Now we are waiting for the next state. The server then connects to the bank and charges us for the product. If everything is okay, then the server will respond with a new page, saying that we bought the product. The last state is again handled by the server. Now, behind the scenes, the server connects with the delivery company and requests to schedule the delivery. In the same way, it transfers our delivery data to this company, and when everything is done, it will contact us back, sending some sort of confirmation message with the tracking number so we can control the delivery process.