Q: (1) Which method is used to access the I / O devices by repeatedly checking the status flags ?

A: The method used to access I/O devices by repeatedly checking the status flags is **polling**.

Q: (1) What is said to be the Individual control words of the micro routine ?

A: The individual control words of a microroutine are typically called **micro-instructions** or **control words**. There isn't a single, universally accepted term beyond that. The specific name might vary depending on the architecture or textbook being used.

Q: (1) Why a Cache memory is considered as the onboard storage ?

A: Cache memory isn't considered onboard storage in the same way that RAM or ROM is. The term "onboard" usually refers to memory that's physically integrated onto the main circuit board (motherboard) of a computer. While cache is *physically* located on the CPU or very close to it (e.g., on a separate chip but still within the same package), it's not generally thought of as "onboard storage" in the broader context.

The distinction is more about its *function* than its physical location. Onboard storage typically refers to the primary memory directly accessible to the CPU (RAM) and permanent storage built into the system (ROM/BIOS). Cache is a *high-speed buffer* that acts as an intermediary between the CPU and main memory (RAM). It's not directly addressed by software in the same way RAM is. It's managed automatically by the hardware to improve performance.

Therefore, while it might be *physically close* to the CPU, cache's primary role as a performance-enhancing buffer, and its different access mechanisms, differentiate it from what is typically referred to as "onboard storage".

Q: (IV) When the architecture supports one instruction per cycle , is called as

A: When the architecture supports one instruction per cycle, it is called **single-cycle architecture** or **single-cycle implementation**.

Q: (V) Write the applications of Octal System .

A: The octal number system (base-8) has limited direct applications compared to decimal (base-10) or binary (base-2). Its main use stems from its convenient relationship with the binary system. Here are some key applications:

* **Early Computing:** Before widespread use of hexadecimal, octal was sometimes used as a shorthand representation for binary data. Three binary digits (bits) can be represented by one octal digit, making it easier for programmers to read and write binary code. This was particularly relevant in systems with word sizes that were multiples of three bits.

* **File Permissions in Unix-like Systems:** Unix-like operating systems (Linux, macOS, etc.) often use octal numbers to represent file permissions. Each octal digit represents read, write, and execute permissions for the owner, group, and others. This concise representation allows for easy manipulation of file permissions. (e.g., `755` represents read, write, and execute for the owner, read and execute for the group and others).

* **Digital Display Systems:** While less common now, octal was occasionally used in early digital display systems because of its compact representation of binary data.

* **Representation of Binary Data:** In some specialized contexts, octal can still be

used as a more human-readable representation of binary data, though hexadecimal (base-16) is generally preferred for this purpose due to its even more compact representation.

* **Education:** It serves as a valuable tool in computer science education to help students understand different number systems and the conversion processes between them.

It's important to note that the prominence of octal has significantly diminished with the rise of hexadecimal. Hexadecimal (base-16) provides an even more compact representation of binary data (4 bits per digit) and is now the more commonly used system for this purpose.

Q: (VI) How many different combinational inputs can generated by a 3 - bit Full ADDRER ?
A: A 3-bit full adder has three inputs (A, B, and Cin) for each bit position. Since each input can be either 0 or 1, there are 2 possibilities for each input. With three inputs, the total number of different combinational inputs is 2 * 2 * 2 = 8.

Q: [VII] What is an Interrupt ?
A: An interrupt is a signal that temporarily suspends the normal execution of a program to handle a more urgent event. Think of it as a sudden interruption that demands immediate attention. The processor stops what it's doing, saves its current state, and jumps to a special routine (an *interrupt handler* or *interrupt service routine* – ISR) to deal with the event. Once the event is handled, the processor restores its previous state and resumes the interrupted program from where it left off.

Here's a breakdown:

* **Signal:** An interrupt is triggered by a hardware or software event.

* **Hardware Interrupts:** These are triggered by external devices like the keyboard, mouse, hard drive, or network card. For example, pressing a key on the keyboard generates a hardware interrupt, signaling the CPU to read the keystroke.

* **Software Interrupts:** These are triggered by software instructions, often used for system calls or exceptions (errors). For instance, a program might generate a software interrupt to request access to a file.

* **Interrupt Handler (ISR):** A dedicated piece of code that processes the interrupt. It handles the specific event that triggered the interrupt. For example, a keyboard interrupt handler would read the key pressed and add it to the keyboard buffer.

* **Context Switching:** The CPU saves the current program's state (registers, program counter, etc.) before executing the ISR. After the ISR completes, it restores this state to allow the original program to continue seamlessly.

Why are Interrupts Important?

Interrupts are crucial for several reasons:

* **Responsiveness:** They allow the system to respond quickly to external events without constantly polling (checking) devices. This is much more efficient.

* **Efficiency:** They enable multitasking and concurrent processing, allowing the CPU to handle multiple tasks seemingly at the same time.

* **Real-time capabilities:** They are essential for systems requiring immediate responses, such as those used in industrial control or embedded systems.

Example:

Imagine you're reading a book (your program). Suddenly, the phone rings (an interrupt). You stop reading, answer the phone, handle the call, and then return to your book where you left off. The interrupt handling is analogous to answering the phone, and restoring your state is like picking up your book where you left off.

Q: (VII) Which device helps to read the control words sequentially ?
A: A **control unit** helps to read control words sequentially. The control unit fetches instructions from memory and decodes them, determining the sequence of operations to be performed. These instructions often involve control words specifying the actions to be taken.

Q: (IX) Evaluate the Storage Capacity of the memory where m = number of memory locations and n - word size .
A: The storage capacity of a memory is determined by the number of memory locations (m) and the word size (n). The total storage capacity is calculated as:

Capacity = m * n

Where:

* **m** is the number of memory locations (often expressed in bytes, kilobytes, megabytes, etc.)
* **n** is the word size (usually in bits, e.g., 8 bits, 16 bits, 32 bits, 64 bits)

Therefore, to find the storage capacity, you simply multiply the number of memory locations by the word size. The resulting value will represent the total amount of data the memory can store, often expressed in bits or bytes. You may need to convert to larger units (KB, MB, GB, etc.) depending on the magnitude of the result.

Q: (Let the number of instructions be n = 64 , find out the speedup value for a 4 segment linear pipeline .
A: The speedup of a pipeline is determined by comparing the execution time of a single instruction in a non-pipelined system versus the execution time in a pipelined system.

Non-pipelined execution:

* Each instruction takes a time of 1 time unit to complete.
* Total execution time for n instructions is n time units.
* For n = 64, the total time is 64 time units.

Pipelined execution:

* With a 4-segment pipeline, the pipeline takes 4 time units to fill completely. After that, one instruction completes every time unit.
* Total execution time = (pipeline fill time) + ((number of instructions - 1) * time per

instruction)

* In a 4-segment pipeline, the pipeline fill time is 4 time units.

* The time per instruction after the pipeline is full is 1 time unit.

* So, the total time = 4 + (64 - 1) * 1 = 67 time units.

Speedup Calculation:

Speedup = Execution time (non-pipelined) / Execution time (pipelined)

Speedup = 64 / 67 ≈ 0.955

Therefore, the speedup for a 4-segment linear pipeline with 64 instructions is approximately **0.955**. Note that this is *less* than 1, which seems counterintuitive. This is because we've analyzed a more realistic scenario including the pipeline fill time. A naive calculation, ignoring the initial fill time, would suggest a speedup of 4, but this isn't the actual performance. The speedup approaches 4 as the number of instructions (n) increases significantly.