To develop an embedded product for an autonomous car that detects objects and takes corrective actions while driving, the requirements must cover the hardware, software, and environmental constraints. Below is a comprehensive list of requirements to guide the selection of a microcontroller for this project.

1. Functional Requirements

1. **Object Detection**
   - Support for interfacing sensors such as LIDAR, ultrasonic, RADAR, and cameras.
   - Real-time image processing capability for object recognition and classification.
2. **Corrective Action**
   - Ability to control actuators for steering, braking, and acceleration.
   - High-speed decision-making to avoid collisions or maintain safe distances.
3. **Communication**
   - Support for CAN, LIN, and Ethernet for communication with other car systems.
   - Ability to interface with GPS and IMU sensors for positioning and orientation.
4. **Fail-Safe Mechanisms**
   - Redundant systems to ensure reliability in case of hardware or software failure.
   - Automatic transition to manual driving in case of system failure.

2. Performance Requirements

1. **Processing Power**
   - High-performance ARM Cortex-M or Cortex-A cores capable of handling complex AI/ML algorithms.
   - Minimum clock speed: 200 MHz.
   - Support for hardware accelerators (e.g., DSP or AI inference engines).
2. **Memory**
   - Flash memory: ≥ 2 MB for program storage.
   - RAM: ≥ 512 KB for real-time processing.
3. **Real-Time Operation**
   - Must support real-time operating systems (RTOS) for deterministic behavior.
   - Low latency for sensor input to action output (≤ 50 ms).
4. **Power Consumption**

- Optimized power consumption for automotive environments, with sleep modes and low-power states.

3. Hardware Requirements

1. **Interfaces**
    - Multiple UART, SPI, I2C, and GPIOs for connecting peripherals.
    - Support for high-speed data interfaces like USB or PCIe.
2. **Robustness**
    - Temperature range: -40°C to 125°C.
    - Vibration and shock resistance per automotive standards.
3. **Safety Standards Compliance**
    - Compliance with ISO 26262 for functional safety (ASIL-B or higher recommended).
4. **Analog and Digital Input/Output**
    - Support for ADCs and DACs for sensor inputs and control outputs.

4. Software Requirements

1. **Development Tools**
    - Support for standard toolchains like GCC, Keil, IAR, or vendor-specific IDEs.
    - Debugging capabilities with JTAG or SWD.
2. **Connectivity**
    - Support for wireless communication standards like Wi-Fi, Bluetooth, or 5G for over-the-air updates.
3. **AI and Machine Learning**
    - Compatibility with AI frameworks like TensorFlow Lite or ONNX for embedded systems.
    - Hardware or software-based ML acceleration.
4. **Firmware Update**
    - Secure bootloader for over-the-air firmware updates (OTA).
    - Encryption and authentication for updates.

5. Environmental Constraints

1. **Automotive Standards**

- Must comply with AEC-Q100 for automotive-grade microcontrollers.
- Electromagnetic compatibility (EMC) and susceptibility (EMS) compliance.

2. **Power Supply**
    - Operate within 12V DC automotive systems with tolerance for voltage spikes.

6. Cost and Scalability

1. **Cost Constraints**
    - Affordable while meeting all performance and safety requirements.
2. **Scalability**
    - Easily scalable for integration into different vehicle models.

The **TI TDA4VM** is the best choice because it is specifically designed for automotive applications, particularly for ADAS and autonomous driving systems. It provides advanced **AI/ML acceleration** with dedicated hardware, including **vision processing units (VPUs)** and AI engines, which are essential for real-time object detection and decision-making. This processor meets **ISO 26262** standards up to **ASIL-D**, ensuring high levels of functional safety for critical automotive systems. The TDA4VM also supports a wide range of connectivity options, including **Wi-Fi**, **Bluetooth**, and **5G**, enabling seamless over-the-air updates and communication with other vehicle systems. With its support for **secure bootloaders**, **OTA firmware updates**, and robust environmental capabilities (temperature range, vibration, and shock resistance), it is built to handle the harsh conditions of the automotive environment. Furthermore, its scalable architecture allows easy integration into various vehicle models, making it a versatile and future-proof choice for automotive applications.

HOW CPU WORKS

How CPU works

CPU stands for Central Processing Unit. It is the brain of the computer. there is a wire lots of different wires Carrying information around CPU. in every CPU there is a particular wire that turns ON and OFF at a steady ready rate to keep everything in sync, it is called clock. Modern CPU's are measured in Gigahertz. GHz The CPU fits into mother board. the mother board holds the Components of Computer to connect to each other. Rig Right of mother is the place for RAM [Random-Access Memory] & it contains all the data that is processed by the CPU. RAM Consists of a list of addresses, and at each is a piece of data. CPU normally requests and process each piece of data in order one after the another. however it can randomly access too when needed. when Computer first starts running a program it sends the address to RAM to begin referring that program.

The RAM address Consist of a Series of 1's and 0's representing on and off wire RAM doesn't do anything until CPU turns ON SET or Enable wire. When Enable is turned ON RAM automatically Sends the data at that address to CPU. Once CPU finishes processing that data, it then Sends another address to RAM. If CPU wants to save data, ~~it turns~~ it Sends data to that address & turns ON SET then RAM overwrites data at that address. What's the data in RAM? It Can be instructions, Number, letters, addresses.

Instruction Set of CPU

each CPU has it's own set of instruction

* LOAD → ~~as~~ load a number from RAM to CPU

* ADD two numbers together

* STORE a number from CPU back to RAM

* COMPARE One number with another

* Jump if Condition to another address in RAM

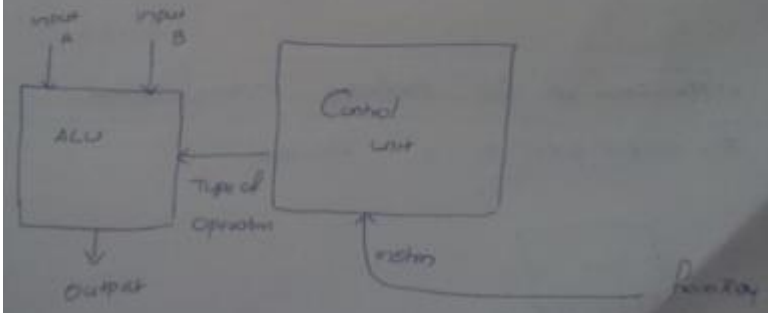
* Jump to another address in RAM

* Output to a device as monitor

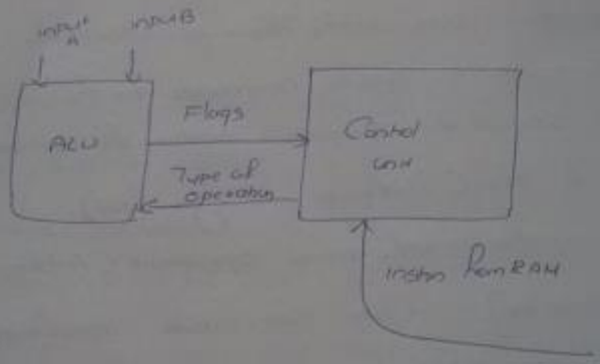
* Input from a device Such as Keyboard

the first Component of CPU is the Control Unit, It receives order from RAM in the form of instructions and breaks them down to specific commands for the other Components One of the Component under the command of Control unit is ALU (Arithmetic and Logic Unit)

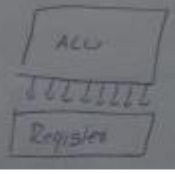
It performs all the mathematical operations (Addition, Subtraction, Comparison etc], it has two inputs Input A and Input B, if we are adding two no.s



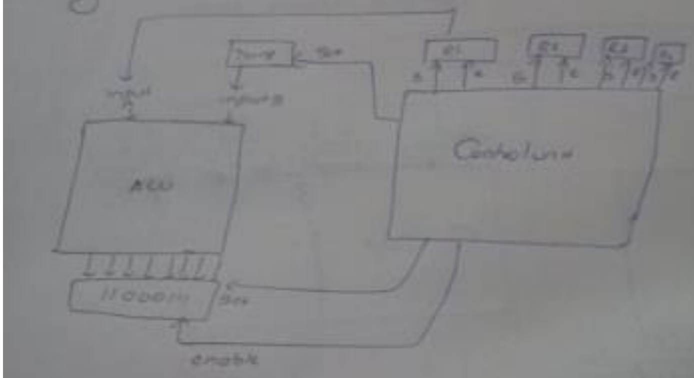
Sometimes, depending upon type of instrn, the output from ALU can be ignored. for instance, if it's a Compare instrn, ALU doesn't need to output an answer, instead tell the Control unit how the two numbers compared. for this ALU uses flags eg help control unit decide what to do when it receive next instrn like jumpzo.



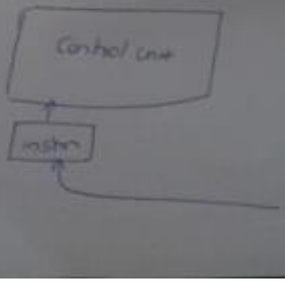
now in the case of o/p producing instrn, ~~where~~ ~~does~~ the output goes to a Register.

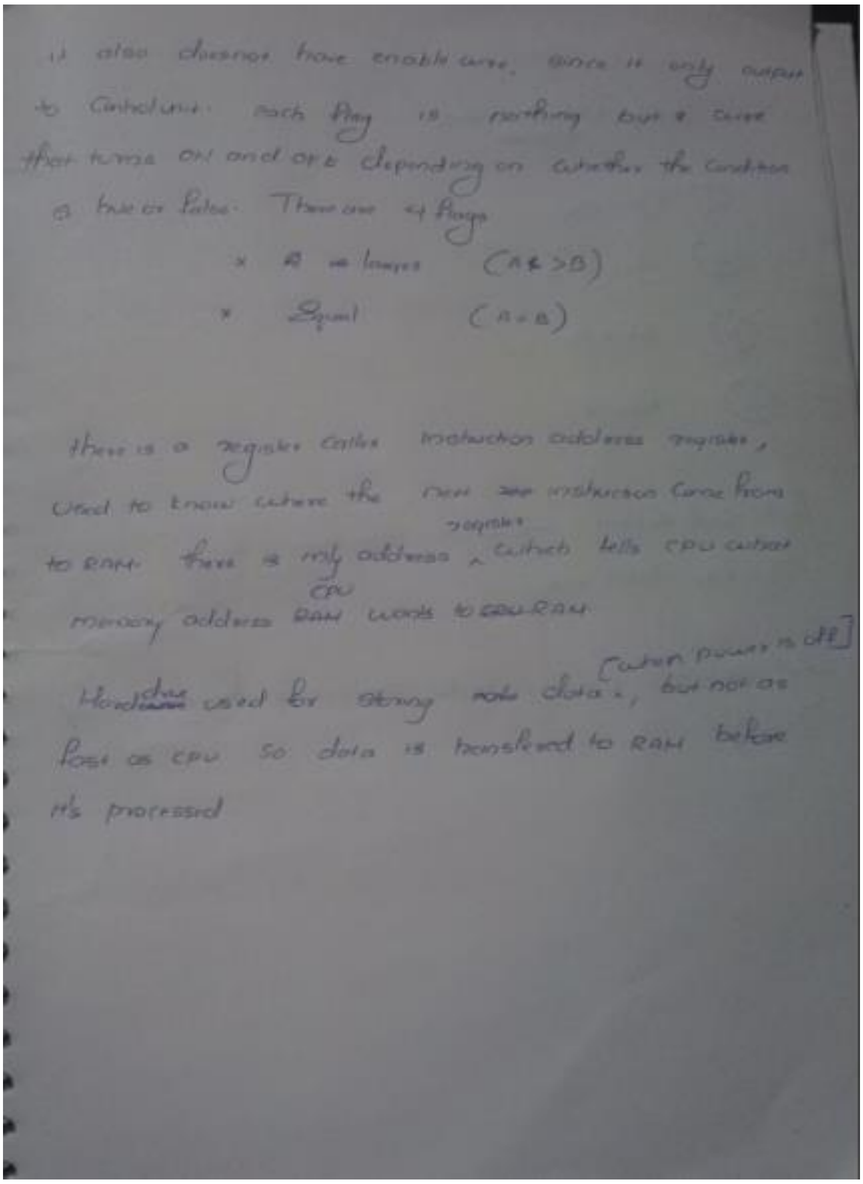


the four registers are used for Storing numbers Between operations the disadvantage of bus is that we can only have one number on it at a time. because of this ALU uses Temporary Register for inputB, Temporary register only have set wire, as since it only outputs to ALU.



the instruction itself is Stored in an instruction register





it also doesnot have enable wire, since it only output to Controlunit. each flag is nothing but a wire that turns ON and OFF depending on whether the condition is true or false. There are 4 flags

- ~~A~~ ~~=~~ larger (A>B)
- Equal (A=B)

there is a register called instruction address register, used to know where the next ~~one~~ instruction come from to RAM. there is mly address register which tells CPU what memory address CPU ~~RAM~~ wants to ~~CPU~~ RAM

Harddisc used for storing ~~more~~ data [when power is off], but not as fast as CPU so data is transfered to RAM before it's processed