- **CPU:** Is an integrated circuit chip that processes signals to control all devices within a machine. It is made up of three components:
    - **Control Unit:** send signals to other parts of computer system. Manages fetch, decode and execute program instructions. (read/write/clock/reset/interrupt).

    - **Arithmetic Logic Unit:** commonly called accumulator, does all operations for you: add subtract multiplication. Where data is actually processed in cpu

    - **Registers**: very fast small storage mechanism, (2 byte of data), sit inside cpu.

- The different types of **registers** include
    - **Program counter:** points to the memory location of next instruction.
    - **Memory address register:** holds the address of a location in the memory. Ie y= 3
    - **Memory data register:** load data by reading from memory address register calls y = 3
    - **Instruction register:** keep current instruction of what cpu is doing so it doesn't forget
    - **General program register:** use by programmers

- There are two broad types of registers these include:

    - **Register-Memory:**
        - Allows memory words to be fetched into registers
        - Allows register to be stored back into memory
    - **Register-Register:**
        - Puts results back into register
        - Performs some operations
        - Fetches 2 operands from register

- Any instruction needs to have resister. Operation need/can take 2 resister such as: 2 x register-register or 1 register + 1 memory. But we cant have two memory position for an operation for machine level program. Atleast one must be register-register. The reasons why is because we have 1 mdr and one mar

- Different types of instructions that may be performed include:
    - **Data manipulation**- add/subtract, increment/decrement, shift (empty space is 0)/rotate (two end ones go side by side), Immediate operand (a number)
    - **Data staging:** load (reading) /store (writing) data from memory.
    - **Controling:** Conditional/unconditional branching in program flow. Subroutine call and return

- A sequence of instructions (more than one instruction). (A computer program is made up of a set of instructions) will be written in a list using assembly language.
    - Exe file is a binary file which is coded in machine level program
    - ***Compiler: convert (3+3) to assembly code → assembler to machine code (binary)***
    - ***So machine code (binary) → assembly code (mov 3) → high level programming language (3 + 3)***

    - ***High level programming → low level programming***

    - ***Problem-orientated language, 3+3: (lvl 5) → Assembly code, mov 3: (lvl 2) [ISA] → Machine code, 0101: (lvl 1) [Digital logic level)***

- The sequence of instruction execution includes:
    - Step one: Program counter we fetch from memory to instruction register
    - Step two: Move program counter to point to next instruction
    - Step three: Determine type of instruction fetched (is it add/division operation etc)
    - Step four: If we need to fetch any word/variable in memory into cpu register. Link up all resources that we need
    - Step 5: execute instruction
    - Step 6: repeat for every instruction + store data into memory postion (if needed)

  Note: Maybe more complicated

- All computer modern should aim to/try to do (cpu designer should follow)
    - All instructions (ie: programming) are executable by hardware

    - Maximize rate at which instructions are issued

    - Instruction (complex for hardware) easy to decode. Do minimal operaration that interact with memory. If we do this, it will help with (NOTE: MAYBE MISREAD IT. The below might be how we make instruction easier to decode)
        - Regular instruction
        - Fixed length (no need for more control instruction to clarify)
        - Small number of fields
        - Small number of fields

- Provide plenty of register because: ram/memory access is slow, transferring between memory and register takes time and to do as much as possible with register
- **Parallelism**: were we use multiple processors and divide the load to perform instruction. Aims to increase instruction rate

We only have a few register is because:

We don't need that many as it wont be used.
We need to control so control unit more complex
One register break then cpu breaks

- Maximize instruction rate issued
  - By doing maximizing this we can have massive performance gains
  - ***1cpu can do multiple Processor at same time = not all instruction at same time but alternates instructions***
  - This is done through **parallel** instruction execution/ (multithreading: extension)

- There are four different types of parallelism: maximizes instruction rate issued. It uses multiple processors to get the task done.
- So instead of using one processor to do a task we use two processors and divide the task so we can do task faster. (we can't just increase the processor to make it faster as it will overheat)

  - **Instruction level parallelism/pipelining**: Multiple task operate simultaneously using different resources.
    While instruction 1 is dealing with 2. Instruction 2 can be dealt with in step 1. Ie 1 → 1 + 2 → 1 + 2 + 3. 9 cycles instead of 25.

  - **Superscalar architecture**: Dual five stage pipeline with common instruction unit. Instruction fetch units + 2: decoder + Operands unit + execute instruction unit + write back unit. Getting instruction at a single fetch rate

  - **Processor level parallelism:** Multiple processors. Independent cpus. Contain: CPU + Memory

  - **Multiprocessors/Processor level parallelism 2**: performing multiple task at the same time share memory

○ *START OF MICROARCITECTURE LEVEL*

- Microarchitecture level (hardware part of the computer tier list. Contains info for Instruction arch level. Not visibile):
  - Given an instruction/program we don't use all the hardware and every part only use portion and function of ALU.

    - Small programs sits in rom and cpu
    - Which route to use and when to use it.
    - Fetches instructions and all that is needed from memory
    - Executes the instruction as a series of digital logic operation
    - Perform a fetch-execute cycle if the ISA needs it
        - Locate operands in memory
        - Read them
        - Store results from operaration back to memory (read/write not done on same path. Reading register one patch reading register one path)

- If 9 register read/write and we want to do operation their need to be a path of 9 lines so data can be coming from each register.

So 2 + 2 = expression
+ is the operation
2 and 2 is operands

- **Operands**: are the data on which the operation is to be performed they are found in processor memory or register. They are needed to perform + operation = instructions

- **Adressing mode**: The different ways in which the location of an operand is specified (presented):

    - **Immediate addressing:**
        - Direct give value to resistor
        - No memory reference
        - Fast
        - Limited range
    - **Direct addressing:**
        - In Memory contains: Operand gives it's value to resistor
        - Limited address space (since operands contain values)
    - **Indirect addressing:**
        - So memory cell tells us address of operands. And then we take its value and put it in register

ADRESSING MODE IS DIFFERENT TECHNIQUE FOR INSTRUCTION EXECUTION. This is how we get values/operands to register [Step 4]

START OF INSTRUCTIONAL SET ARCHITECTURE LEVEL

- **Instructional Set Architecture:** acts as a bridge between compiler (level 5) and hardware (level 1) *remember high level programming to low.
    - Hardware execute java on hardware directly is too hard

    - Translates programs in high level languages to common intermediate form
    - Programmers can write in ISA level (assembly code)
    - Language both understood by compiler and hardware
    - **Implemented by microarchitecture in hardware**
    - Features needed by compiler are added by isa
    - Featured deemed to complex to implement are left to the compiler to implement. && ABOVE IS JUST CHARACTERISTICS

- Good properties of ISA include: Make a good set of instructions implemented effectively
    - Mindful of future technology so its not excluded in instruction sets.
    - Design poor = poor performance
    - Design poor = more difficult to implement and require more gates and therefore cost more money
    - A design that takes advantage of a current technology might not be best for future
- Provide a clean target for compiled code
    - Regulatory and completeness in range of options

- Properties that belong to ira include:
    - What memory model aviable
    - What registers are available
    - What data types & instruction available
    - Recognize ISA-code: is what a compiler outputs

- Properties that don't belong to IRA include:
    - Other issues that are not part of the ira as compiler doesn't need to understand them
    - Hardware parralism and superscalar design etc
    - Operation unit and alu
    - Optimization of cpu ie: convert int to float

Arithmetic Logic Unit: also called accumulator, has logic unit.

---

*ADD/DO TUTORIAL 3
* https://stackoverflow.com/questions/2684364/why-arent-programs-written-in-assembly-more-often