Assembler Utility program converts assembly → machine code

Linker Utility program combines individual files created by an Assembler into single executable programme
Linker links the libraries used inside combines with objcode

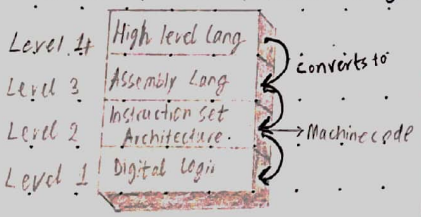
* Assembly Lang – portable = false;
each assembly lang designed for specific processor

Uses of Asm Lang:
1. Used in writing embedded programs of electronic devices
2. Computer games
3. Used in device drivers ~ files that Allow hardware devices to communicate to OS
4. Writing code for older processors
5. Cryptographic Algorithms
6. Making Computer Viruses
7. Used in making Operating Systems
8. used in reverse engineering of sourcecode. Used by hackers, crackers, competitors to predict source code of a company generated software

Os Loader. Loads programs, places programs then into memory, initilizes some regs, prog then executes

Debugger Utility prog, steps you thraugh prog while its running displays realtime reg. vals

Listing file contains copy of source code + numeric address of each instruction + symbol table



Instruction Execution cycle (instruction Pointer)
1. CPU fetches instructions from instruction queue then increments
2. CPU decodes instruction
3. If instruction has operands CPU fetches them from registers and memory
4. CPU then executes instruction simultaneously updating CPU flags
5. If output Operand present in ins. CPU stores result in operand

Execution unit 1st division of CPU, executes Instructions

Bus Interface unit 2nd division of CPU, transmits addresses, data bw Execution unit and memory / I/o devices

Instruction prefetch BI unit fetches 6 Bytes of next instruction onto instruction queue

Cache Computer memory creates speed bottleneck when accessing variables
To reduce amount of time spent in reading and writing most recently used instructions and data are stored in high speed memory. **Cache**
if there is a loop clock cycles would be saved

cache hit when processor finds data in cache mem.

cache miss when processor cant find data in cache mem

Level1 cache (primary cache) stored in CPU, vv fast but less space

Level2 cache (secondary cache) connected to CPU by buses thora sa slower but has Large mem

* Cache faster than RAM because · Cache = static. RAM (no need to refresh)
RAM = dynamic ram (needs to be refreshed)

Loading + Executing a Programme.
1. Programme loaded into memory by Loader
2. Os initially searches for programme in disk for execution
3. If not found, gen. Error else Os retrieves info about programme file (file size, physical loc)
4. Os then allocates memory block into next available mem Location
5. Programmes size and Location stored in descriptor table
6. Os begins executing first machine instruction, the program is now called process
7. Os assigns process an ID, while execution Os responds to all request of process
8. when process ends, it is removed from memory

Virtual Machine partially uses hard drive of system space, completely separated from main system
- **System VM** a complete system and platform for all processes which mimics the actual operating system
- **Process VM** a virtual environment of an OS is created while using that app, the environment will be destroyed as soon as we exit the app.

Mode of Operations x86 processors manage memory according to the basic modes of operation
- **Real Address Mode** 1 mb of memory can be addressed by a prog. processor can run only 1 prog at a time. programme can access any part of memory. MS DOS, Windows 98 runs in this Memory mode
- **Protected Mode** 4gb of memory can be addressed by a prog. Os assigns mem to each progs. Win 2000, XP, Linux runs in this mem mode
- **System management mode** Power saving and system error handling introduced
- **Virtual 8086 mode** Here Processor creates Virtual x8086 machine. 1MB address space for each running prog. XP makes multiple of these to Prevent Prog. crashes affect other prog

CPU performs arthimetic, logic, controlling, Input/output operations

Control unit Uses binary decoder to convert coded ins into Signals

clock makes simultaneous operations possible synchronises cpu and components, processor has a clock generator which generate pulses. its frequency = = clock cycle
Kitni taizi se processor Instruction executes
cpu frequency varies to save Power and temperature
a Instruction requires 1 clock cycle to execute

Memory storage unit (RAM) is where all instructions and data is held
This storage unit recieves request from cpu for copying data into CPU to be executed Line by Line then transferred back into ram.

Bus group of parallel wires that transfer data
- **data bus** transfers ins. bw CPU & RAM + ROM
- **I/o bus** tranfers data bw CPU & input/output devices
- **Control bus** uses (High 1) or (Low 0) to turn devices on or off or control thier actions (read or write...)
- **Address bus** holds address of currently executing instruction & address

32 bit Architecture means
cpu can transfer 32 bits of data per clock cycle

Segmentation: main memory of computer divided into segments
in real mode allows processes to easily share data, No internal fragmentation
allows 16 bit registers to address 1Mb
w/o segmentation it would require 20 bit regs

addressable memory ; The memory address which you can access

segment consecutive memory bytes, Max segment size is 16 bits
a memory loc is specified through an offset
.code .stack .data .extra (3 segments)
These are 6 segments accessible by a prog

★ code segment register points to .code segment
★ data segment register points .data segment
★ Stack Segment register points .stack segment
★ Extra segment register points to .extra segments

Segment address : A1F8 16 bits

Whats its Linear address when offset is 04C0

A1F80
04C0
A2440

Segments Variable sized memory blocks for code & data
In protected mode

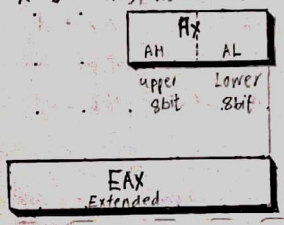
Segment descriptor contains segments base address, size, usage

Segment descriptor table table contains segment descriptors
Ds Ko segments track kine mein asani

Segment registers points to segment descriptor tables

registors high speed storage locations inside CPU

★ 8 × 32 bit general regs
★ 6 × 16 bit segment regs
★ 1 × processor status flags reg
★ 1 × Instruction Pointer



Segment registors
★ In real address mode 16 bit segment regs store segment addresses
★ In protected mode segment regs hold pointers to segment descriptor table

Instruction Pointer
EIP contains address of next instruction to be executed

(32bit) General registers
EAX favoured for arithmetic operations (Accumulator reg)
EBX holds pointer to data (Base reg)
ECX used in loops (counter reg)
EDX used in input/output operations (data reg)
ESP pointer to the top of Stack (stack pointer)
EBP pointer to the bottom of stack (stack base pointer)
ESI (source index)
EDI (destination index)

flags
Control flags
Direction flag (DF)
Interrupt flag (IF) determines whether interrupt can occur
Trap flag (TF)
Status flags
carry flag (CF) carry (1), no carry (0)
overflow flag (OF) when data too large to fit = (1)
sign flag (SF) when Arithmetic gives (-) result = (1)
Zero flag (ZF) 0 ans in Arithmetic = (1)
parity flag (PF) even parity = (1)

Instruction becomes executable when Assembled
Label is a identifier acts as Label for data
Data Label: Identifies location of variable, must be unique
Code Label: used in jumping and looping, must end with :

Mnemonics MOV, ADD, SUB, MUL, INC ...
operands eax, [Label], 96Ah, 0110b
Comments ; single line, @ code @ block

12345669h

	Big endian		Little endian
0000	12	0000	69
0001	34	0001	56
0002	56	0002	34
0003	69	0003	12

mov EAX, 26.E+5
(sign) (exponent) (constant)
(integer) (sign)

Directives commands that are part of assembler syntax, but not related to x86 instruction set
.data, .code, .stack
BYTE, SBYTE, WORD