# Lec 21/01/25

SIMD single instr multiple data

SSE

AVX adv vector extension

same instr on multiple data points together. ⇒ hand coded library implementation to make use of direct instrs: specialised for FFT, convl etc!

→ **Stack Archi**: only few of the stack operands (top ones)
↳ Bring operands to stack first. Top 2 are operands and ans written on ToS (overwrite B)

→ **Accum. Archi**: Accum. registers
Load A → accum. reg.
Add B → B in memory (can use)
Store C → write result on accum. reg of A

→ **Reg-Mem Arc**: uses both reg. and mem in ALU. (intel x86)
saves no. of instr wrt ISA.

→ **Reg-Reg Arc**: all ALU ops only in reg.
"Load-Store reg archi". (ARM, MIPS, RISC-5)
↳ more no. of instr typically. coz we have to fetch into reg. Uniform latency of ALU ops.

→ No. of registers

## No. of registers

more registers → better performance.

but more reg ⇒ bitlines length / incr
and hence delay (RC) incr

(16) regs. found optimal < 32 bit / 64 bit today

→

**Instr Size:** reg-reg : 5(3) + 6 (opcode) = 21
(32 reg, 3 regs)

mem-mem > 32(3) = 96

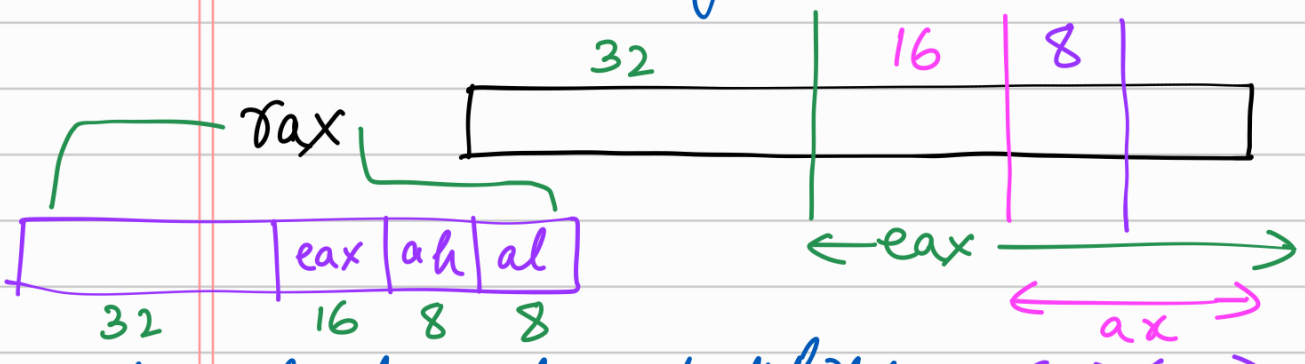
mem-m > r-r

**Code Density:** $\frac{\#\text{ of instr}}{\text{total size of code in B}}$

**Execution Time:** mem-mem ⇒ ALU non-determ ⇒ implementⁿ diff ⇒ reg-reg < time than mem-mem

**Pipelining:** reg-reg better, since uniform latency of ALU

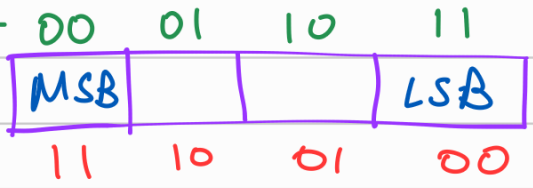
No. of bits for reg encoding : reg-reg needs more reg than mem-mem.




for backward compatibility and changing parts of a reg.

takes same amt of time ... of subpart

BE





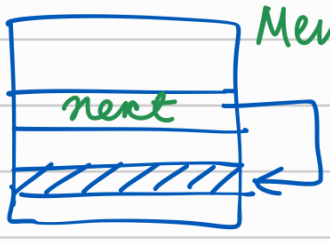
LE ⇒ LSB ⇒ 000
BE ⇒ MSB ⇒ 000
} same in both

→ **Addressing modes** (VAX left = destn)

N reg ⇒ each log N bits.

**Indexed** — one is like base and other like displacement eg. array start in R1 & offset in R2, R2 += 4 each loop.

**Mem Indirect** R3 →




Linked list better

Take R3 as an addr and go there. then treat it as a mem and go there

**Auto inc / dec** for loops.

when R3=n then reset to 0.

amt of inc/dec can be prog

(R3)+ ⇒ go then R3++

−(R3) ⇒ R3-- then go (last is (n-1) so R3 loaded with n)

**Scaled** : 100 + R2 + d × R3

⇓

scaling const, hidden.
can be programmed using insts.