# Forenklinger ved CPU Simuleringen

Iforhold til vår forenklede simulering er de fleste CPUer som Intel og AMD mer komplekse:

- Instruksjoner bruker mer tid enn en CPU-sykel på å utføres
- Hver instruksjon hentes inn fra RAM før den utføres (ikke ROM)
- En x86-instruksjon deles inn i flere små deler (mikro-operasjoner), uops

# CPU-løkke (hardware-nivå)

Pseudo-kode for den evige hardware-løkken som CPU'en kjører:

```
while(not HALT)
{
   IR = mem[PC];    # IR = Instruction Register
   PC++;            # PC = Program counter
   execute(IR);
   if(InterruptRequest)
   {
      savePC();
      loadPC(IRQ); # IRQ = Interrupt Request
                   # Hopper til Interrupt-rutine
   }
}
```

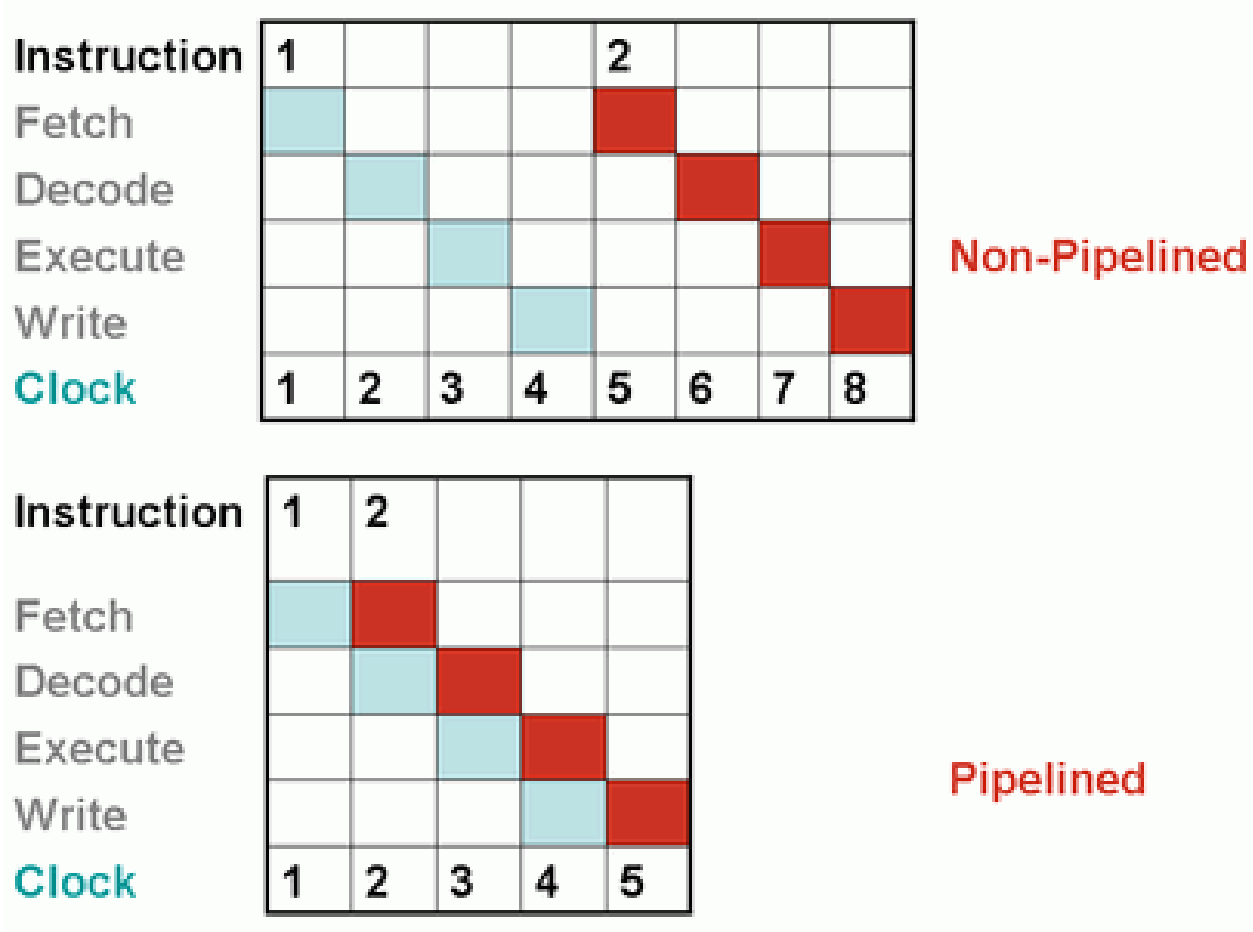
# Pipelining

En instruksjon kan deles inn i flere deler, stages, 14 er vanlig i Intel-CPUer.
Eksempel med 4 stages:

- Fetch (hent instruksjonen fra RAM)
- Decode (hvilke knapper skal trykkes på i ALU og Datapath)
- Execute (utfør instruksjonen)
- Write (skriv resultater til RAM)

Tid spares ved at neste instruksjon starter før den første er ferdig.

# Pipelining

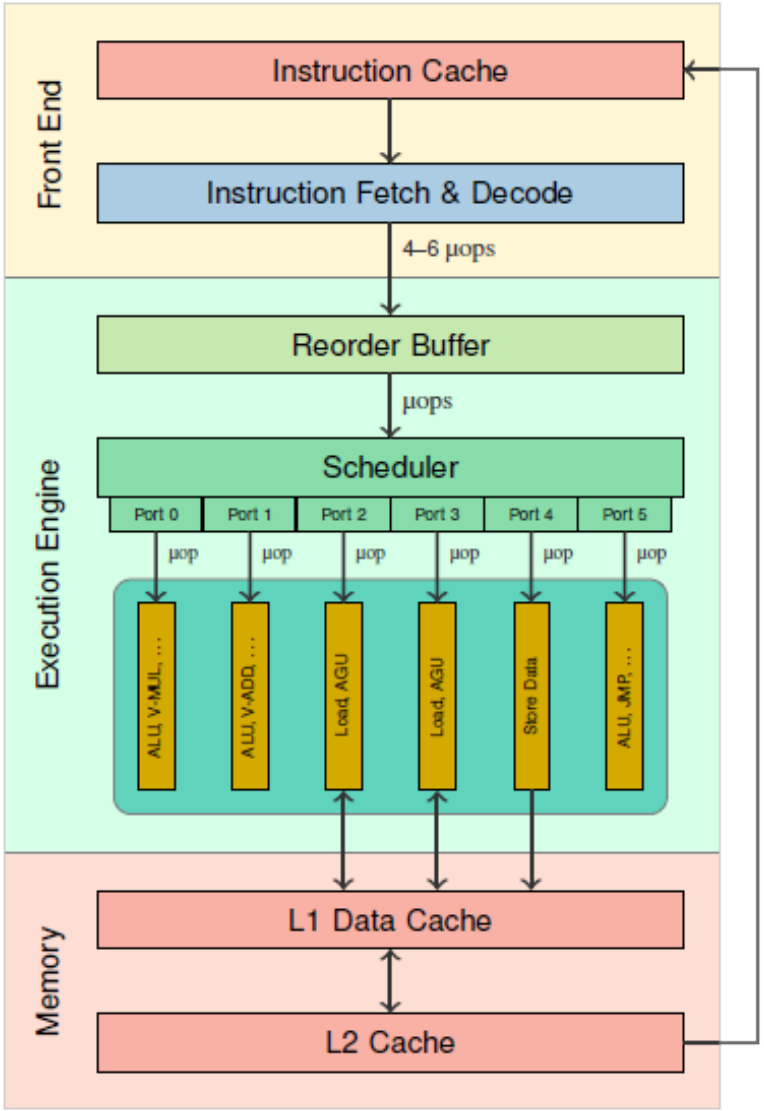


# Intel mikroarkitekturer

En mikroarkitektur er hvordan et instruksjonsett er implementert i en CPU.

| År | arkitektur (CPU) | pipeline stages | Max MHz | nm |
|---|---|---|---|---|
| 1978 | 8086 | 2 | 5 | 3000 |
| 1985 | 486 | 5 | 33 | 1000 |
| 1995 | P6 (Pentium Pro) | 14 | 450 | 250 |
| 2000 | NetBurst (Pentium 4) | 20 | 2000 | 180 |
| 2004 | NetBurst (Pentium 4) | 31 | 3800 | 90 |
| 2011 | Sandy Bridge (core i7) | 14 | 4000 | 32 |
| 2015 | Skylake (core i7) | 14 | 4200 | 14 |
| 2019 | Cascade Lake (core i9) | 14 | 4400 | 14 |

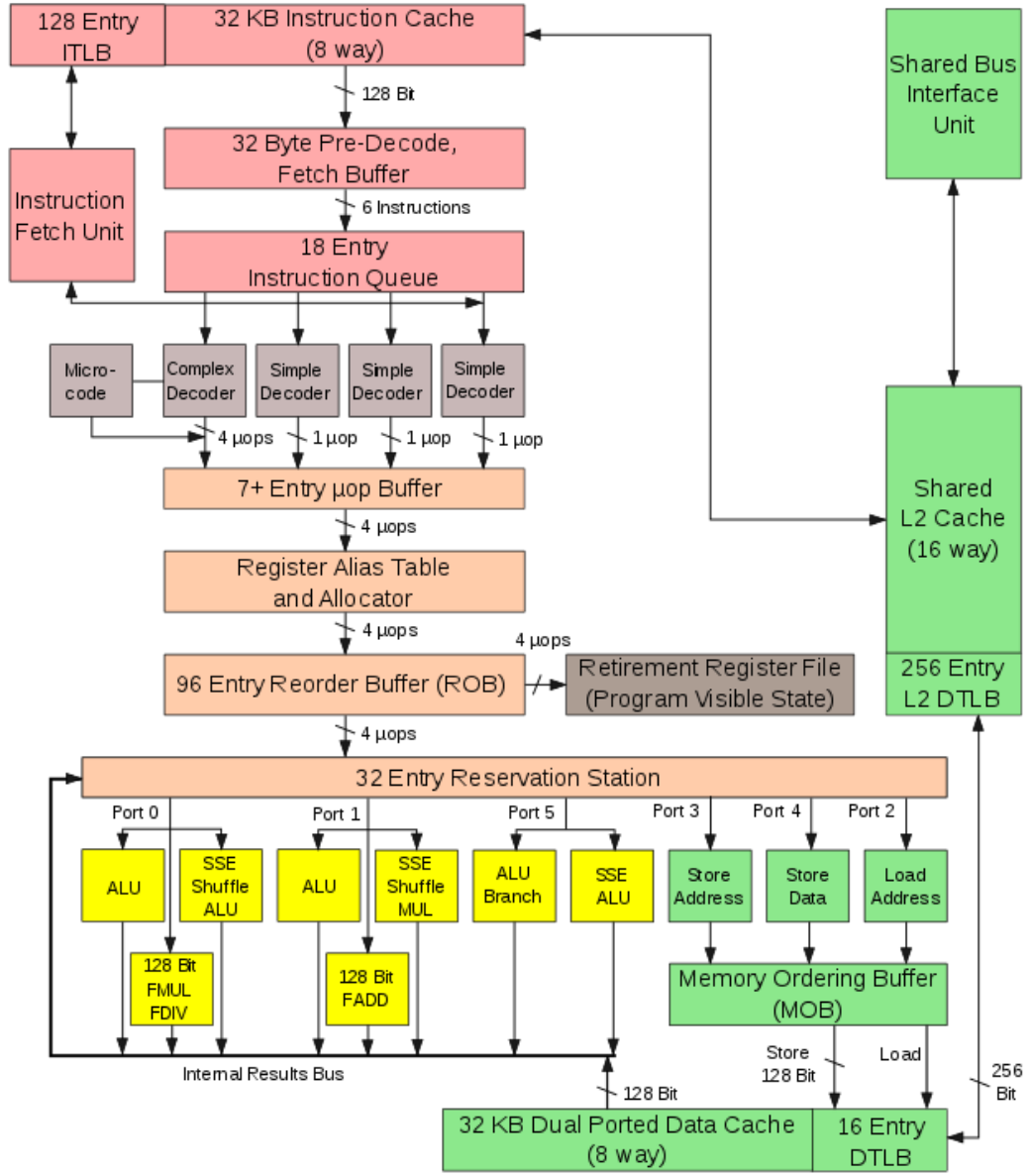
# Superscalar arkitektur



# Superscalar arkitektur

- En skalær prosessor utfører instruksjoner en for en (som simuleringen)
- En superskalær prosessor har flere parallelle enheter som utfører mikro-operasjoner
- For eksempel 2 ALUer, 1 FPU, load, store
- Utfører operasjoner samtidig
- Operasjoner kan utføres out-of-order (i en annen rekkefølge enn det sekvensielle programmet tilsier)

# Intel Core 2




Intel Core 2 Architecture

# branch prediction

- Ved en branch i programmet (if-test), vet man ikke hva neste instruksjon er
- Stort problem for pipelining, må vente på resultatet fra forrige instruksjon
- Gjetter, basert på erfaring, hvilken branch (gren) som følges i programmet og utfører den
- Speculative execution, må gjøres om hvis feil
- I et superskalær arkitektur kan begge grener delvis utføres på forhånd

# Meltdown

- Et hardware-sikkerhetshull funnet i 2018
- Rammet Intel, ARM og IBM-prosessorer
- Meltdown utnytter at både koden som sjekker om prosessen kan lese fra RAM og lesingen fra RAM delvis utføres
- Meltdown kan dermed lese data fra andre prosesser som er cache't men ennå ikke fjernet pga feil branch
- Spectre brukte lignende metoder til å lese passord og sensitive data
- Betegnet som sikkerhets-katastrofe
- Både CPU design og operativsystemer ble endret for å hindre Meltdown og Spectre i å virke

# Viktig å huske fra datamaskinarkitektur

- Alt CPU'en gjør er å slavisk utføre maskininstruksjoner en for en i en evigvarende løkke
- Ingen en til en forbindelse mellom instruksjoner i høynivkode og maskinkode
- En linje kode i høynivåspråk fører ofte til mange maskininstruksjoner i det kompilerte programmet

# CPU-løkke (hardware-nivå)

Pseudo-kode for den evige hardware-løkken som CPU'en kjører:

```
while(not HALT)
{
   IR = mem[PC];    # IR = Instruction Register
   PC++;            # PC = Program counter
   execute(IR);
   if(InterruptRequest)
   {
      savePC();
      loadPC(IRQ); # IRQ = Interrupt Request
                   # Hopper til Interrupt-rutine
   }
}
```