4CS015 – FUNDAMENTALS OF COMPUTING:

CPU ARCHITECTURE

Objectives

- By the end of this session you will be able to:
  - Describe the history of development of the CPU.
  - Explain how cache, pipelines and superscalar operation can improve performance.

History of Computer

Charles Babbage is considered to the father of computer.

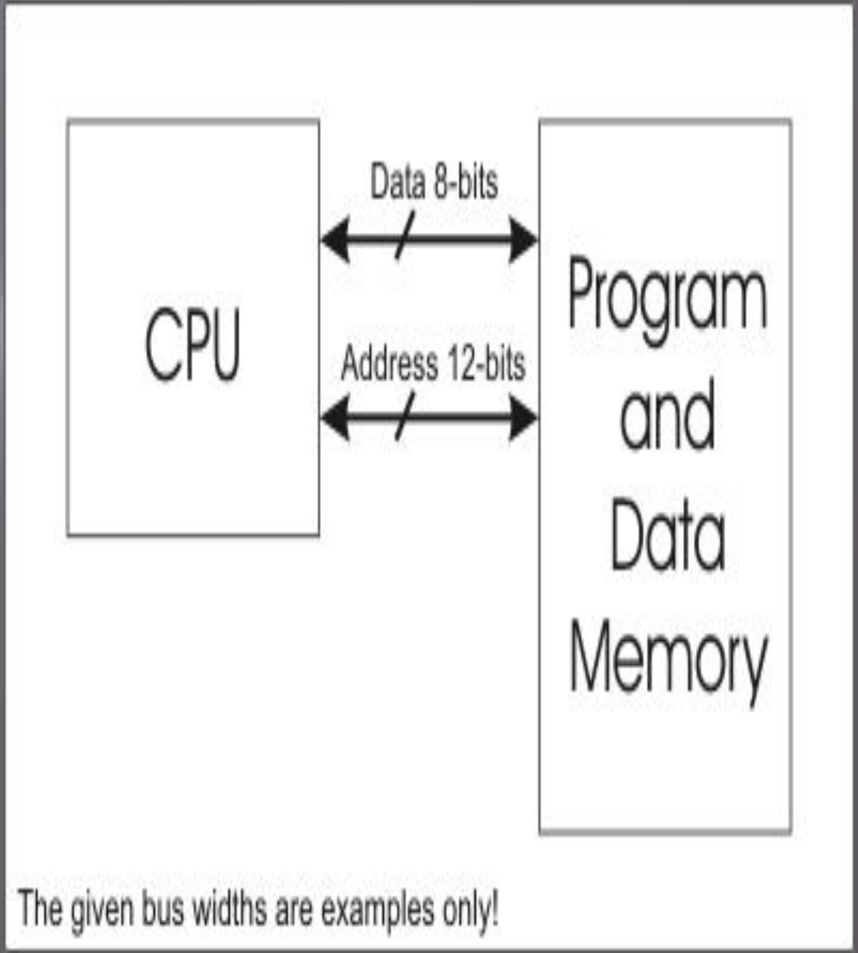
- He designed "**Difference Engine**" in 1822
- He designed a ***fully automatic analytical engine*** in 1842 for performing basic arithmetic functions
- His efforts established a number of principles that are fundamental to the design of any digital computer

History

- First computers programmed by wiring.
- **1944:**
  - EDVAC (**E**lectronic **D**iscrete **V**ariable **A**utomatic **C**omputer) – first conceptual stored program computer.
  - Von Neumann, Eckert, & Mauchly.
- **1948:**
  - First (prototype) stored program computer – Mark I at Manchester University.
  - Harvard Mark III & IV. Howard Aiken.
    - Harvard Architecture / fully automatic / controlled by paper tape / reliable / beginning modern computer era.

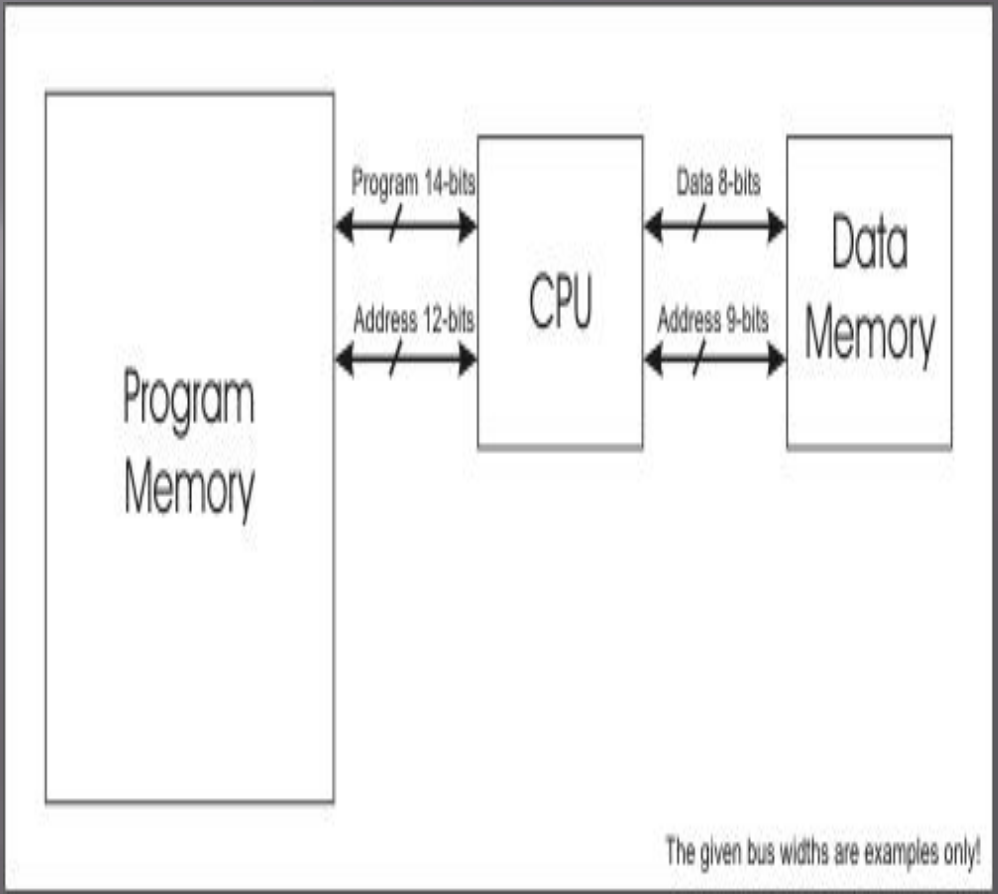
Von Neumann

- Classic computer architecture.
- Data and program instructions stored in same memory.
- Fetch-execute cycle.
- Suffers from Von Neumann ‘bottleneck’.



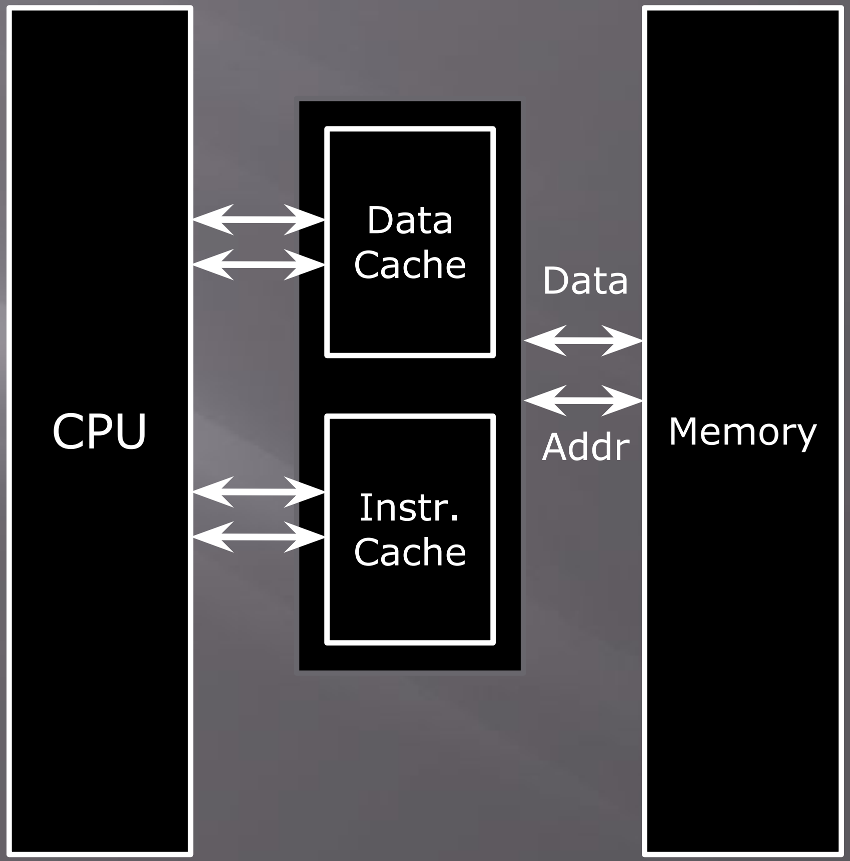
Harvard

- Data and programs stored in separate memory areas.
- Allows for faster operation.
- Simultaneous access of both data and programs
- Allows data and instruction bus to be diff sizes.



Modified Harvard

- Hybrid approach – combination of von Neumann and Harvard
- Single Main Memory for both Data and Program
- Separate High-speed Memory Caches for Data and Instructions
- Simultaneous access of both data and programs from caches



CPU

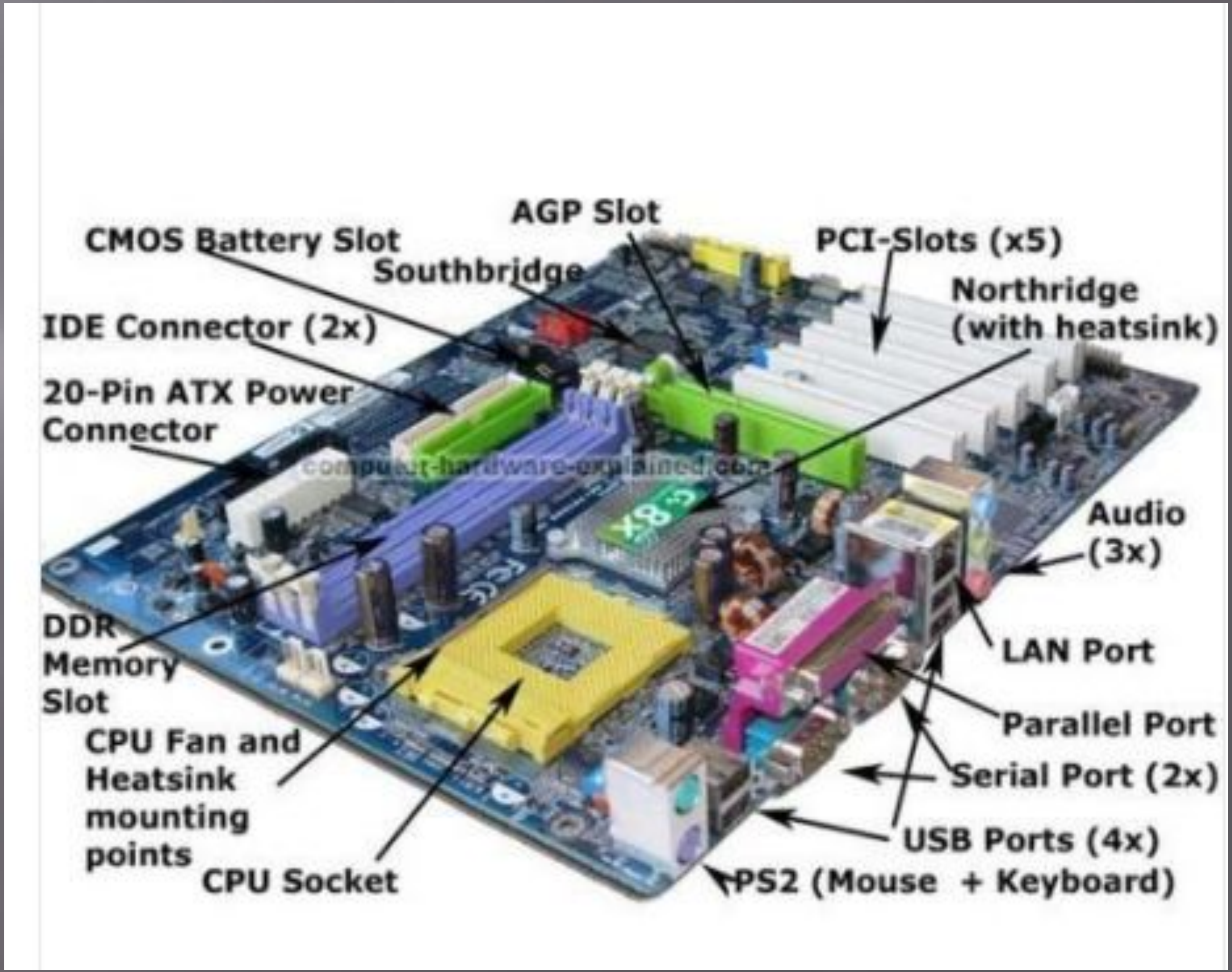
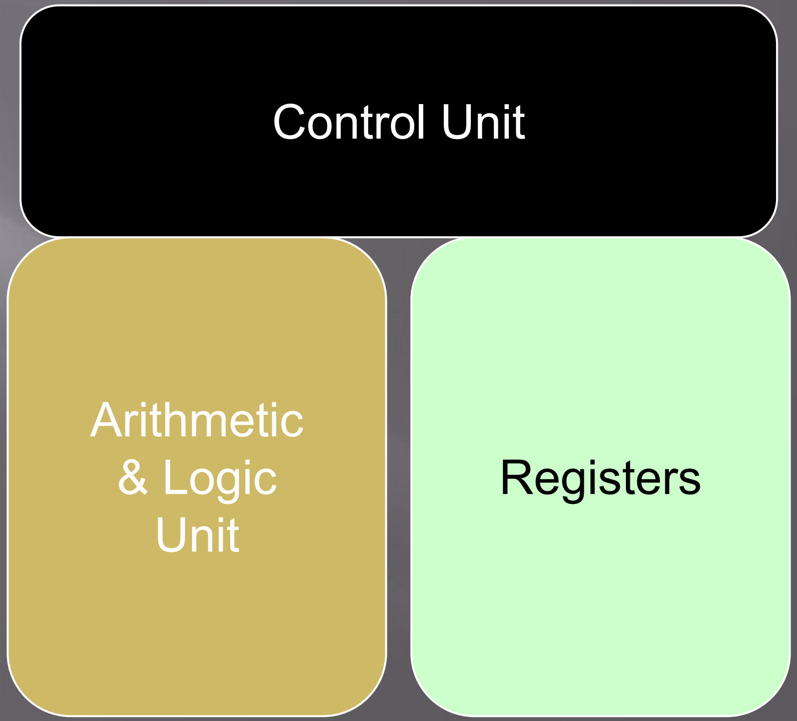


Fig: Mother Board

CPU

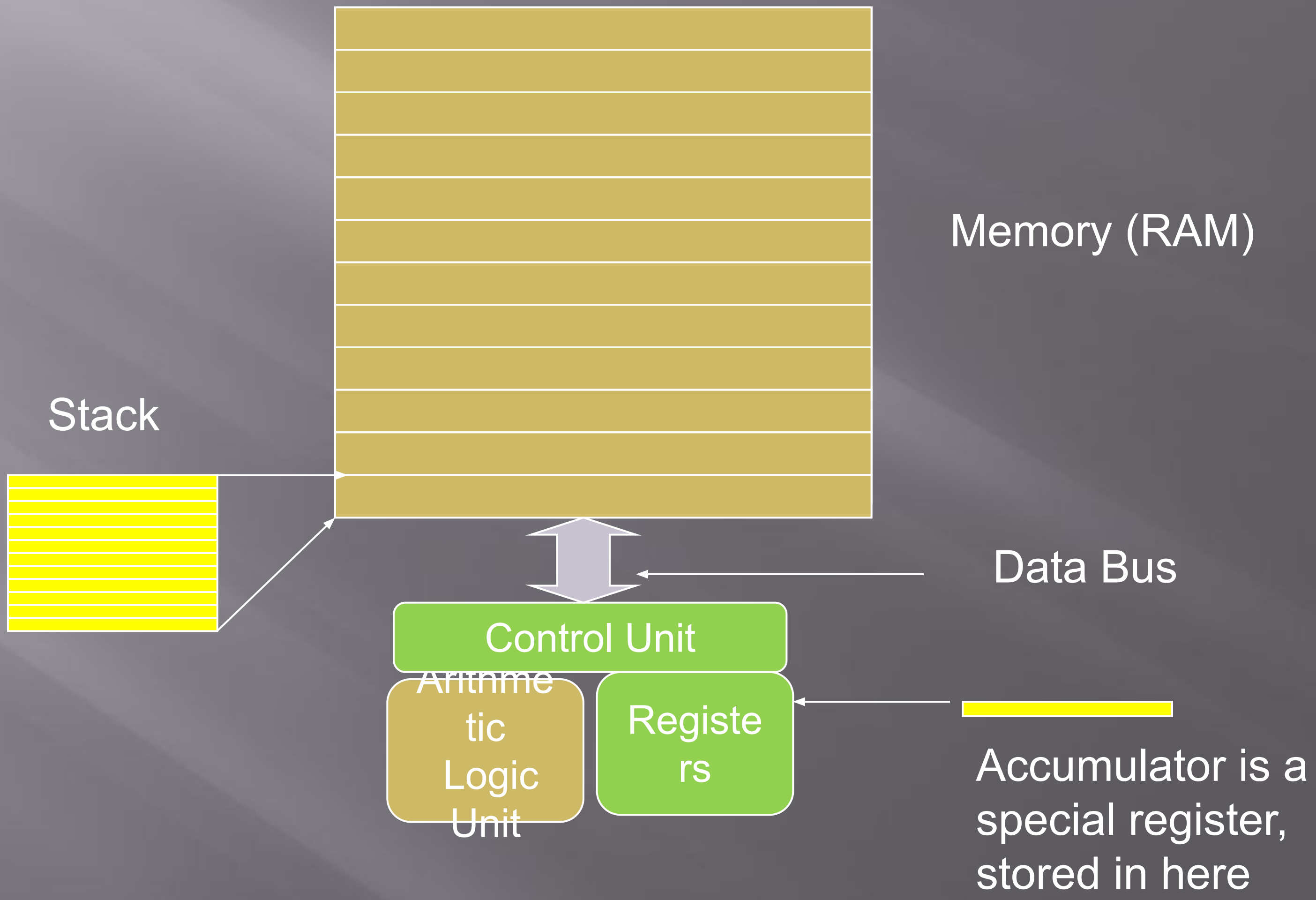
- CPU consists of transistors, combined together as gates.
- CPU works in cycles – fetch, decode, execute.
- Usually each step handled by different part of CPU.
- Where would adders be found?



Types of CPU design

- **Accumulator:**
  - All ALU operations work on data in accumulator (special register).
- **Stack:**
  - All ALU operations work on data stored on the stack.
- **Register-register:**
  - All ALU operations work on data stored in registers.

Example Architecture



Architecture Bit Sizing

- Most CPU's are rated by the number of bits they have:
  - 16 bit (8086).
  - 32 bit (80386).
  - 64 bit (Core 2)

Bus

- A bus is a high-speed internal connection. Buses are used to send control signals and data between the processor and other components.

Three types of bus are used.

- **Address bus** - carries memory addresses from the processor to other components such as primary storage and input/output devices. The address bus is ***unidirectional.***

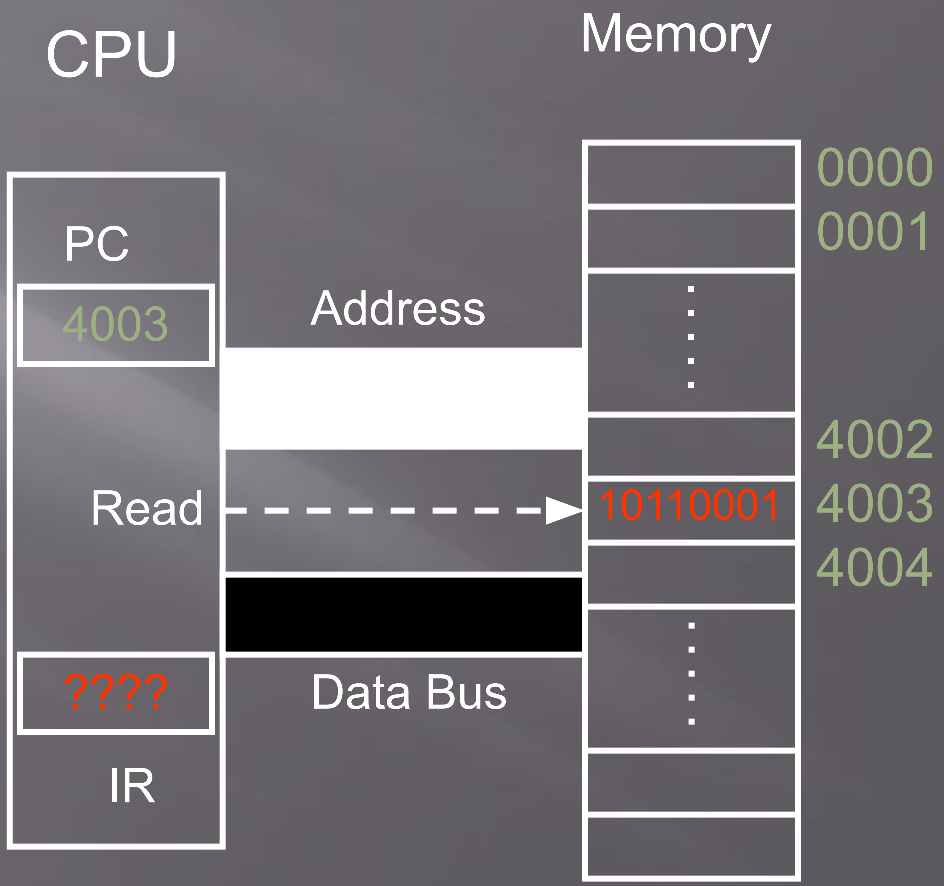
- **Data bus** - carries the data between the processor and other components. The data bus is ***bidirectional***.

- **Control bus** - carries control signals from the processor to other components. The control bus also carries the clock's pulses. The control bus is ***unidirectional.***

Data Bus

PC = Program counter. Register / holds the address of the current instruction.

- Determines how much data can be copied from / to memory at a time (cycle?).
- Advantages of Harvard vs. Von Neumann?

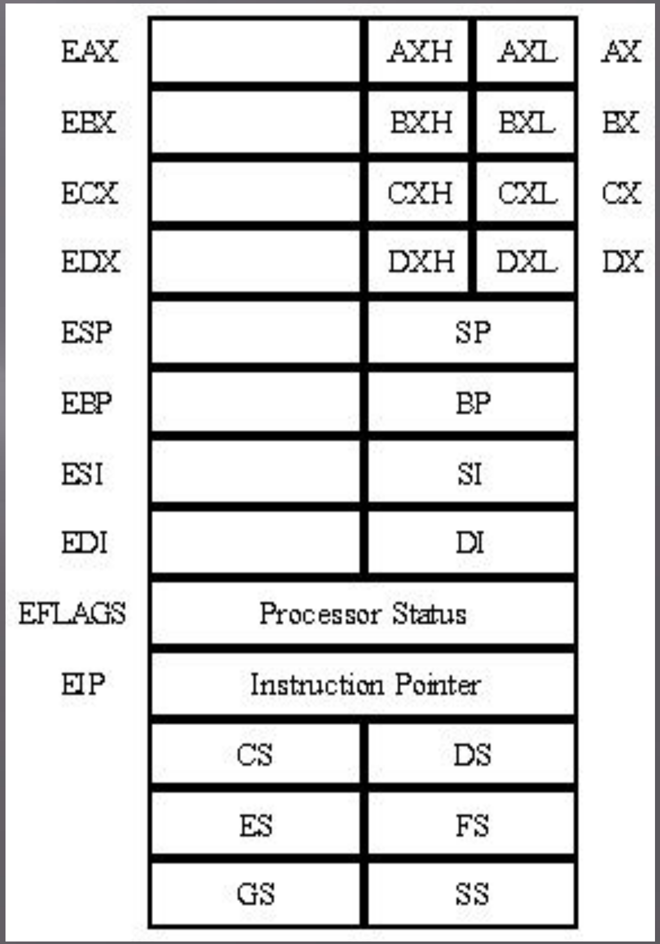


IR = Instruction register. Holds the current instruction while it is executed.

EAX,EBX,ECX,EDX: general purpose registers.
ESP: stack pointer (top).
EBP: Stack base Pointer.
EFLAGS: condition codes.
EIP: program counter (instruction pointer).

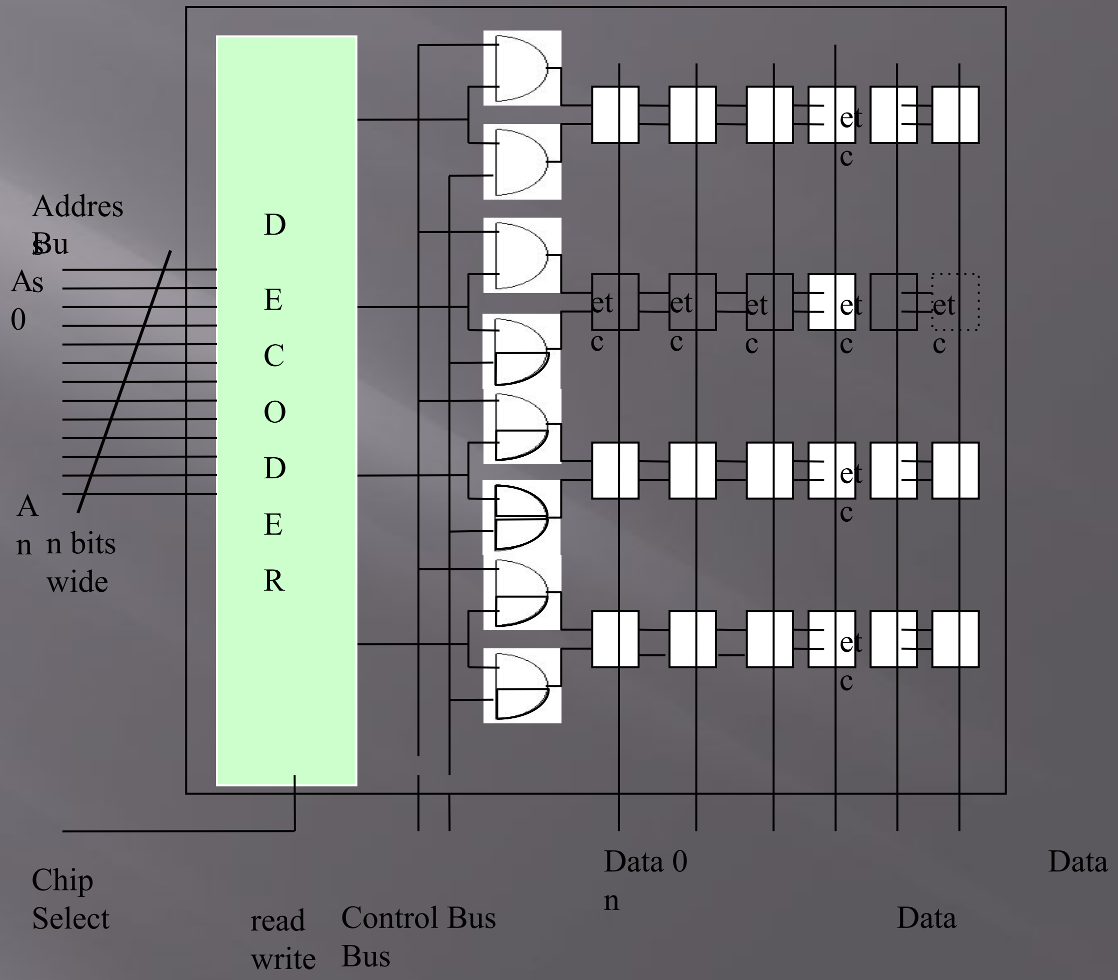
Register Sizing

- Registers are the internal ‘memory’ of a CPU.
- They determine the maximum memory that can be addressed (PC).
- They determine the size of ALU operations.
- Can be 8-bit, 16-bit, 32-bit, 64-bt (or more!)



Address Bus

- Determines how much memory can be accessed.
- Matched with program counter / pointer



Address Bus

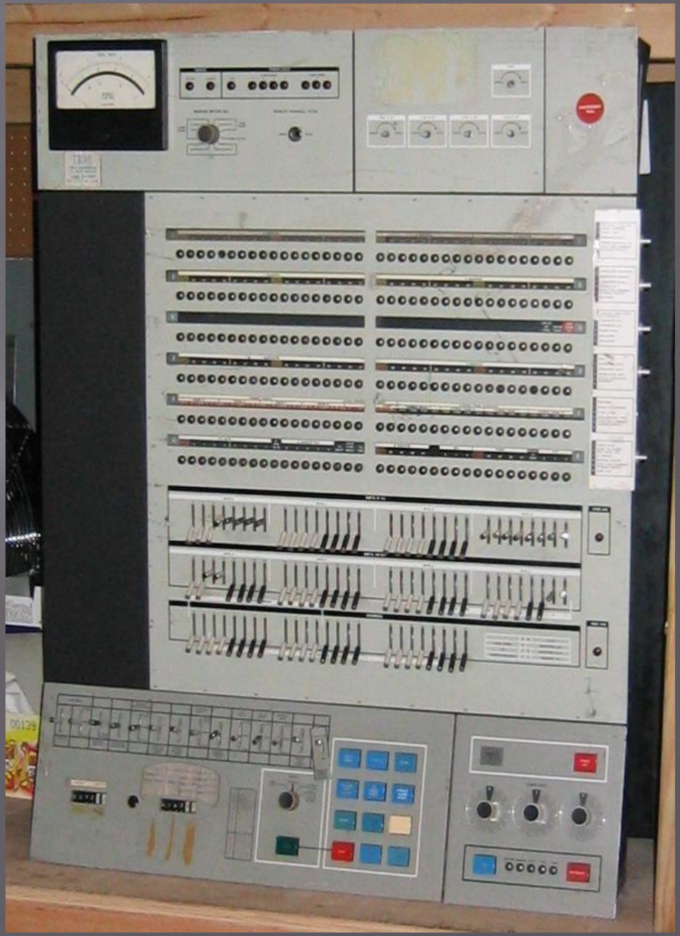
- Each ‘line’ in memory corresponds to a bit in a register (the MAR – memory address register).
- So 8 lines = 8 bits = 2^{8} = ??
- 32 lines = 32 bits = 2^{32} = 4GB.
- 64 lines = 64 bits = 2^{64} = 18.5 EB (exabytes), $1.85\text{x}10^{19}$bytes

Next Steps…

- Late fifties / early sixties number computers available increased rapidly.
- Generally each generation had a different architecture than previous one (enhancements / improvements).
  - Different instruction sets.
  - Machines customised for customers.
- Was this a problem?
- IBM solution: system 360.

System 360

- System 360 popularised use of microcode across the ‘range’.
  - **i.e.** same instruction set.
  - **Example:** Multiplication.
- Any program written for one model would work on all.
- Binary compatibility.



CISC - Complex Instruction Set Computer

- Originally, CPU speed and Memory speed the same.
- Instructions varied in size, 1 – 5 words (16 bit).
- Depending on data bus size, could take up to 5 cycles / instruction.
- Decoding may include microcode.
- **CISC** – designed to make programming easier – either for assembly programmer or compiler programmer.

Evolution

- Mid seventies – John Cocke @ IBM initiated research on performance of microcoded CPU’s.
- 1980 – Patterson built on this work, coined term RISC.
- **Conclusions:**
  - Most programs use only small % of available instructions.
  - Microcoded instructions not always best / most efficient way of performing task.

Instruction Usage

Instruction type	Usage X86	Usage (ARM)
Data movement	38%	43%
Control flow	22%	23%
Arithmetic operations	14%	15%
Comparisons	16%	13%
Logical operations	6%	5%
Other	4%	1%

- Which is the most important type of instruction to optimise?

RISC - Reduced instruction set computer

- RISC should really be Reduced Complexity Instructions (RCISC)
- Ideal specification:
  - Each instruction executes in one cycle – increases processing efficiency.
  - No microcode.
  - All instructions work on registers, except load / store.
  - May need more store as have more instructions.

RISC v. CISC

- **Performance:**
- RISC optimises cycles / instruction.
- CISC optimises instructions per program.
- Trade – off.
- How do we optimise cycle time?

Example: VAX Vs. MIPS

Description	Value (mean)
VAX CPI	9.9
MIPS CPI	1.7
CPI Ratio (VAX/MIPS)	5.8
Instruction Ratio (MIPS/VAX)	2.2
RISC factor	2.7

- **Cycles Per Instruction (CPI):** instructions may take one or more cycles to complete.
- **VAX** – CISC architecture. **MIPS** – RISC architecture.
- RISC factor = CPI ratio / Instruction ratio.

Modern Enhancements

- How do we improve cycle times?
- Cycle time = **I-time** + **E-time** where
  - **I-time** = time taken to fetch instruction from memory (or cache).
  - **E-time** – time taken by ALU to execute instruction.

Clock Speed

- System clock like ‘heartbeat’ for system.
- All system synchronised to clock.
- Ideal condition is CPU performs one instruction (or more!) per clock cycle.

Clock Cycles

- Cycles measured in Hertz.
- 1 MHz = 1 million cycles/sec.
- 1 GHz = 1,000 million cycles/sec.

- Remember ideal: > 1 instruction/cycle!

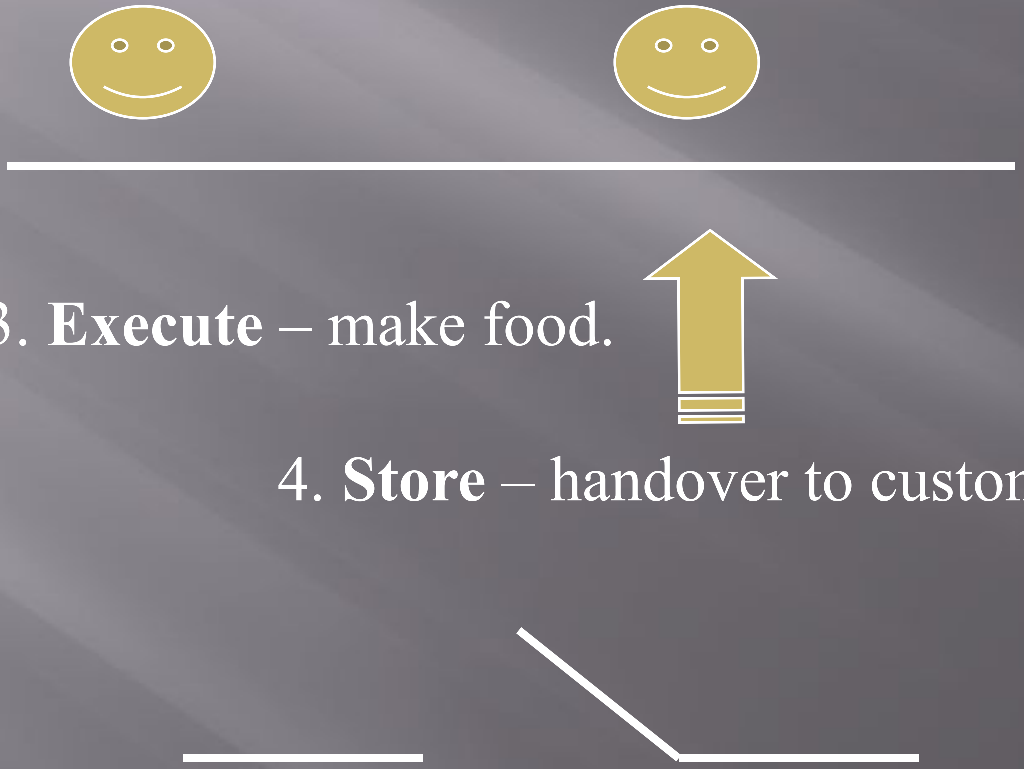
Operations (Original CISC)

- **Analogy:** The fast food outlet…

1. **Fetch** – press button, open door.

2. **Decode** – take order.

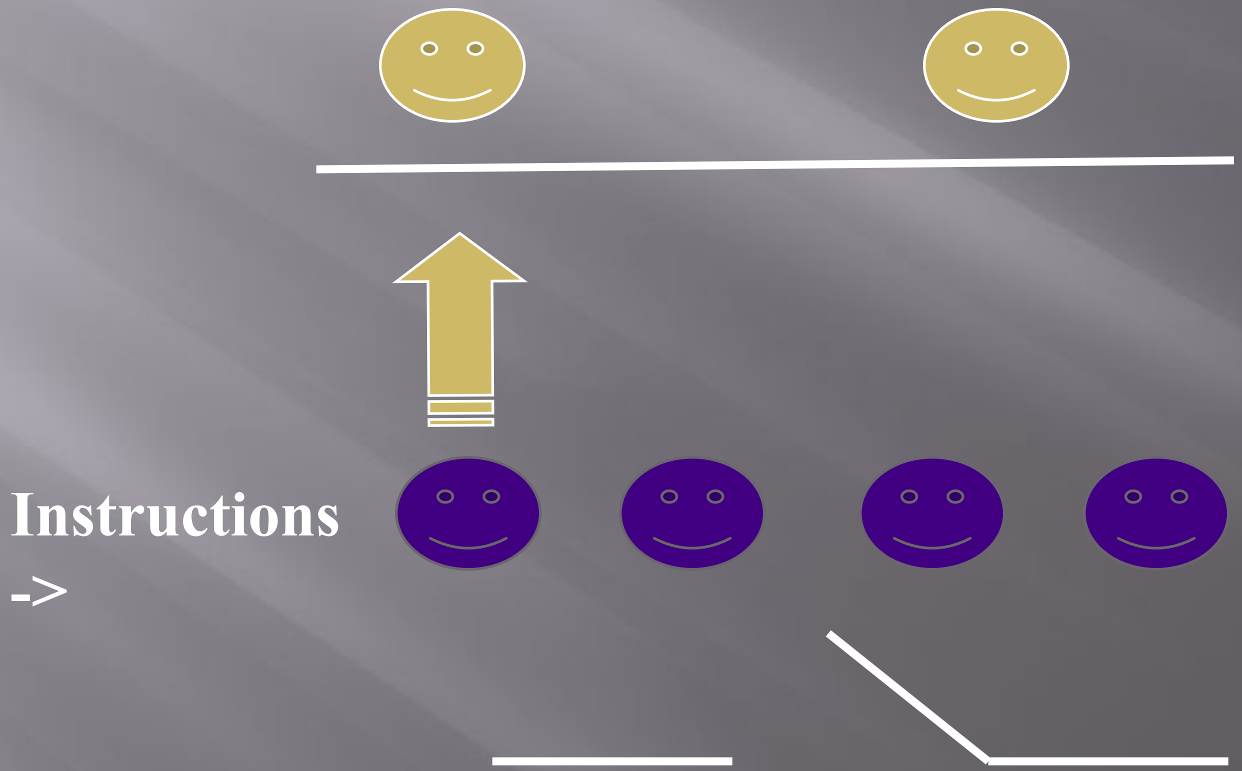
Operations



3. **Execute** – make food.

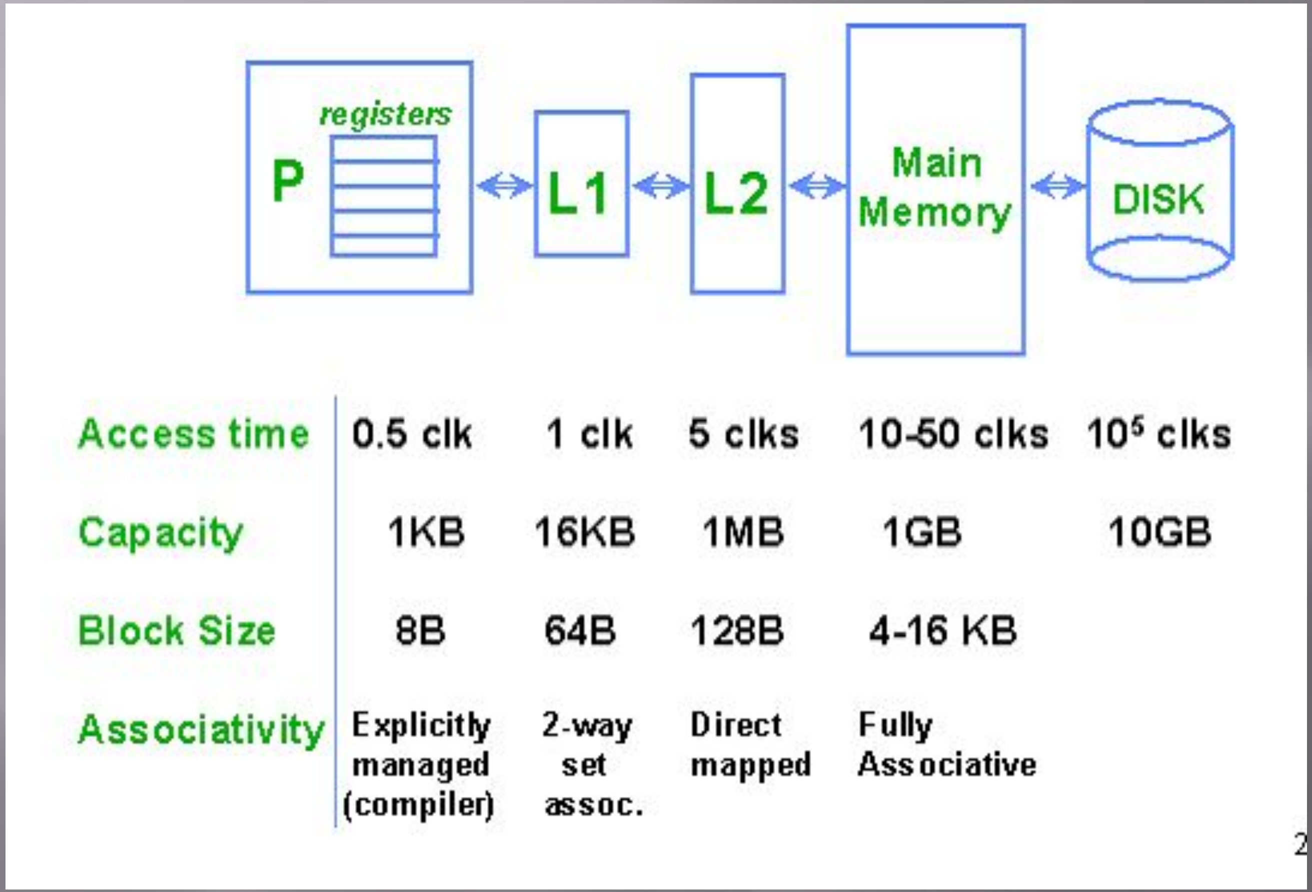
4. **Store** – handover to customer.

Cache

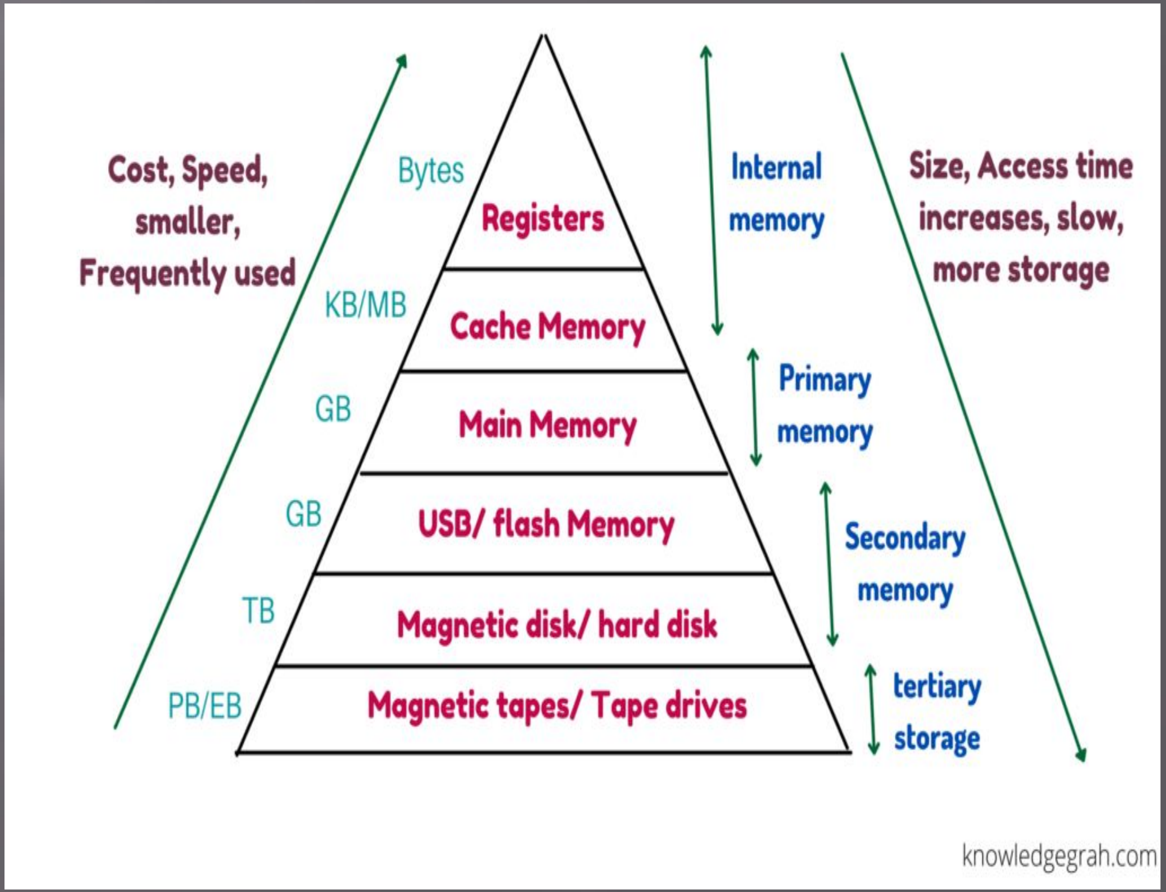


- Expensive and high speed memory.
- Speed up memory retrieval.
- Relatively small amount.

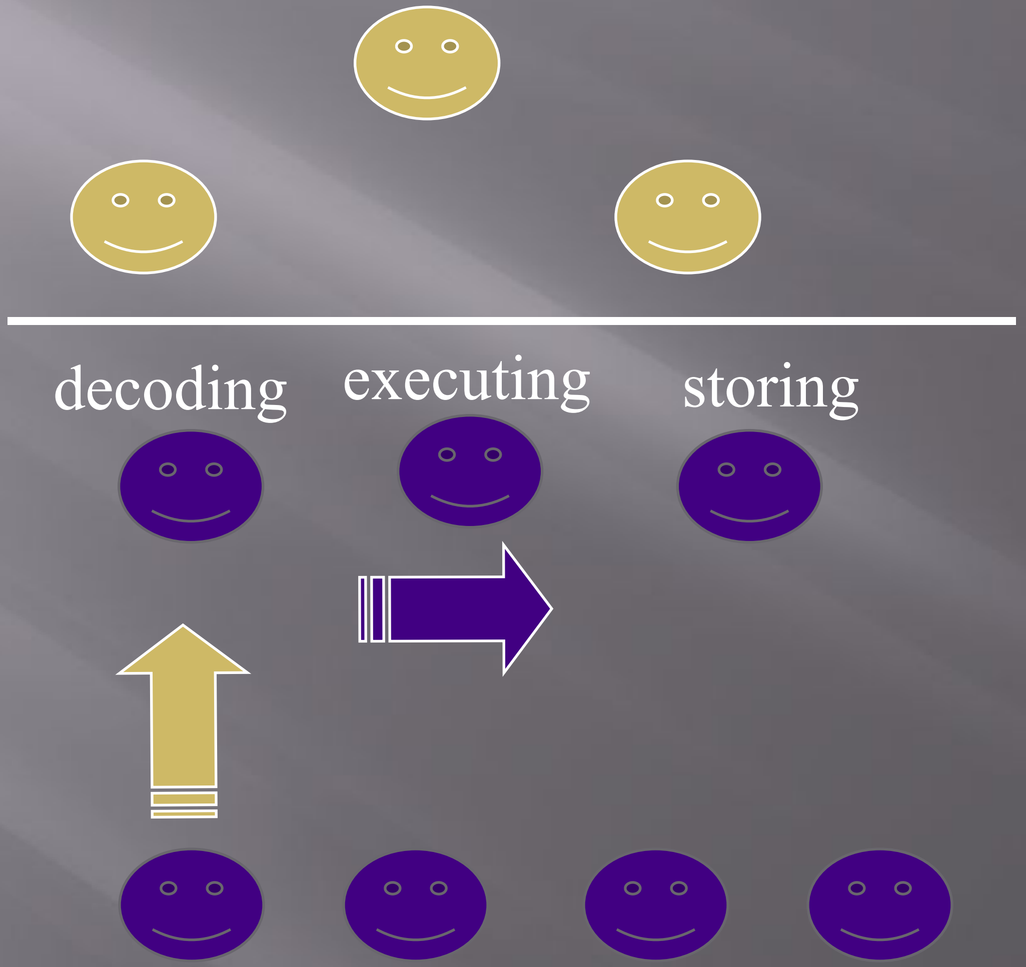
Cache



Access time	0.5 clk	1 clk	5 clks	10-50 clks	10^5 clks
Capacity	1KB	16KB	1MB	1GB	10GB
Block Size	8B	64B	128B	4-16 KB	
Associativity	Explicitly managed (compiler)	2-way set assoc.	Direct mapped	Fully Associative	



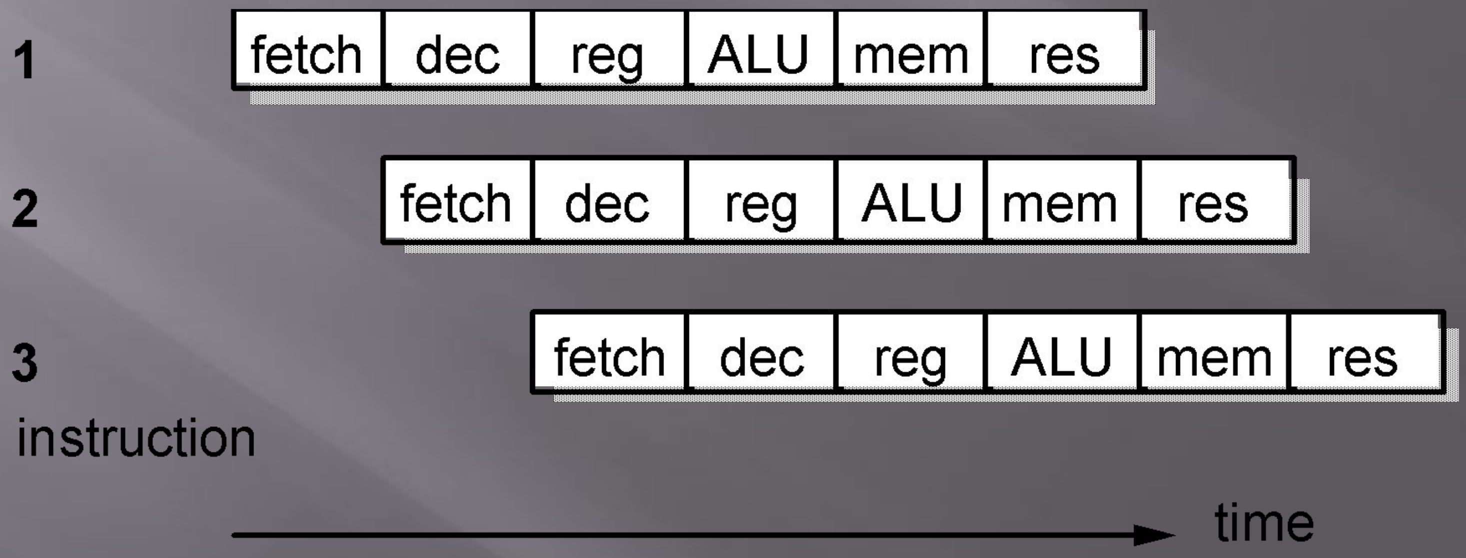
Pipeline



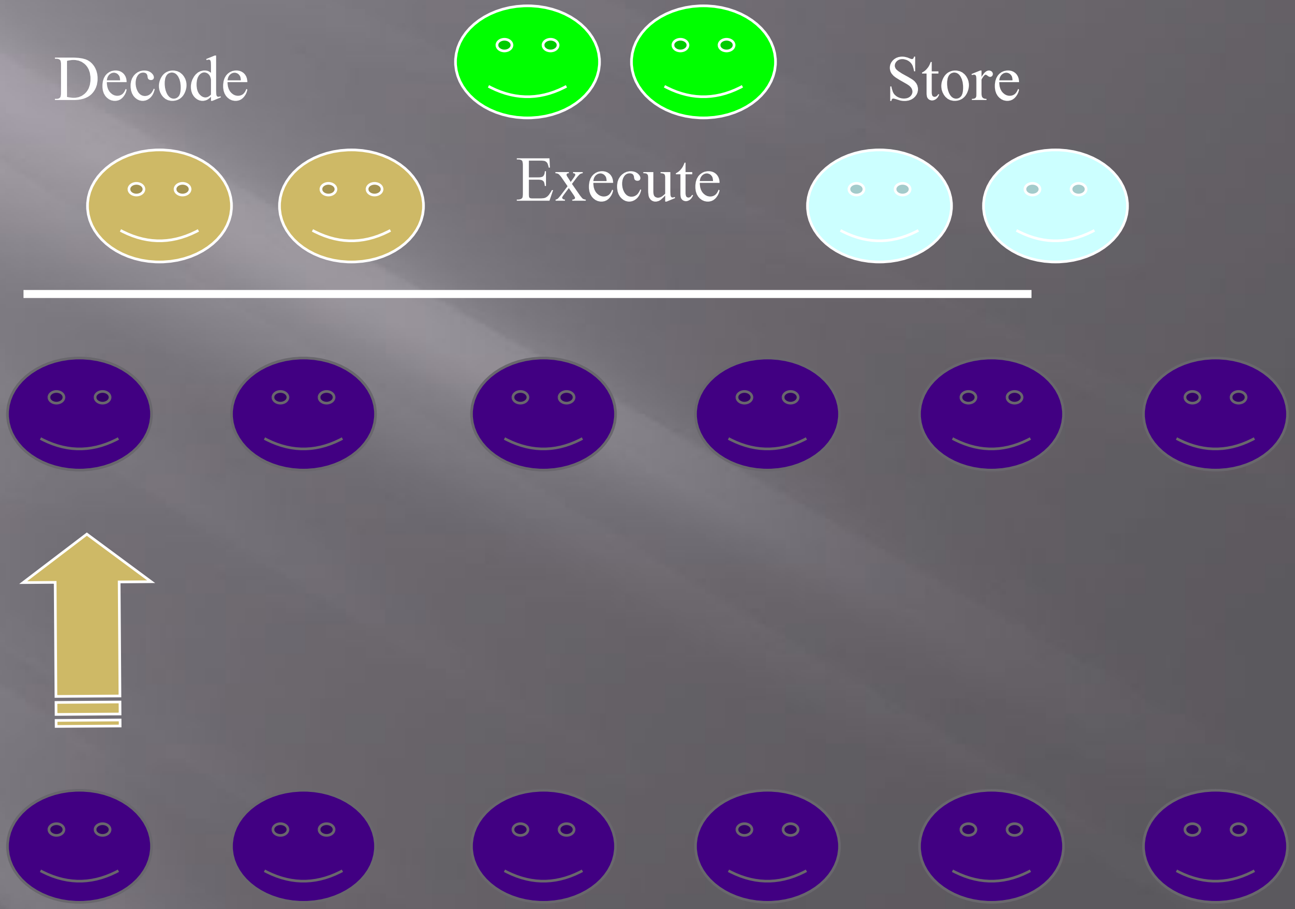
simultaneously

Pipelines

- Concept of using each functional unit simultaneously.
- Theoretically:
  - N stage pipeline = n* increase in speed.
  - Problems in practice? **Pipeline Hazards.**
  - **Instruction 2** operand is the output from **Instruction 1**.
    - Stall / read-after-write hazard.
  - Branching.



Superscalar



Multiple,
independent,
functional units.

Instruction-level parallelism.

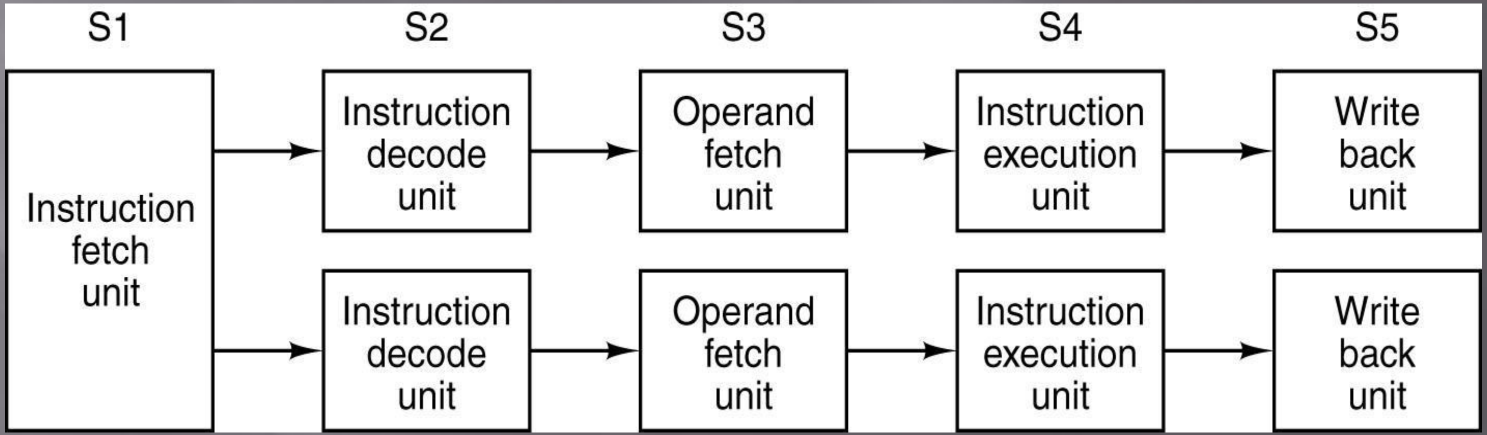
Superscalar

Which instructions can be perform simultaneously?

$$\left\{\begin{array}{l} E = a + b \\ F = c + d \\ G = e * f \end{array}\right.$$

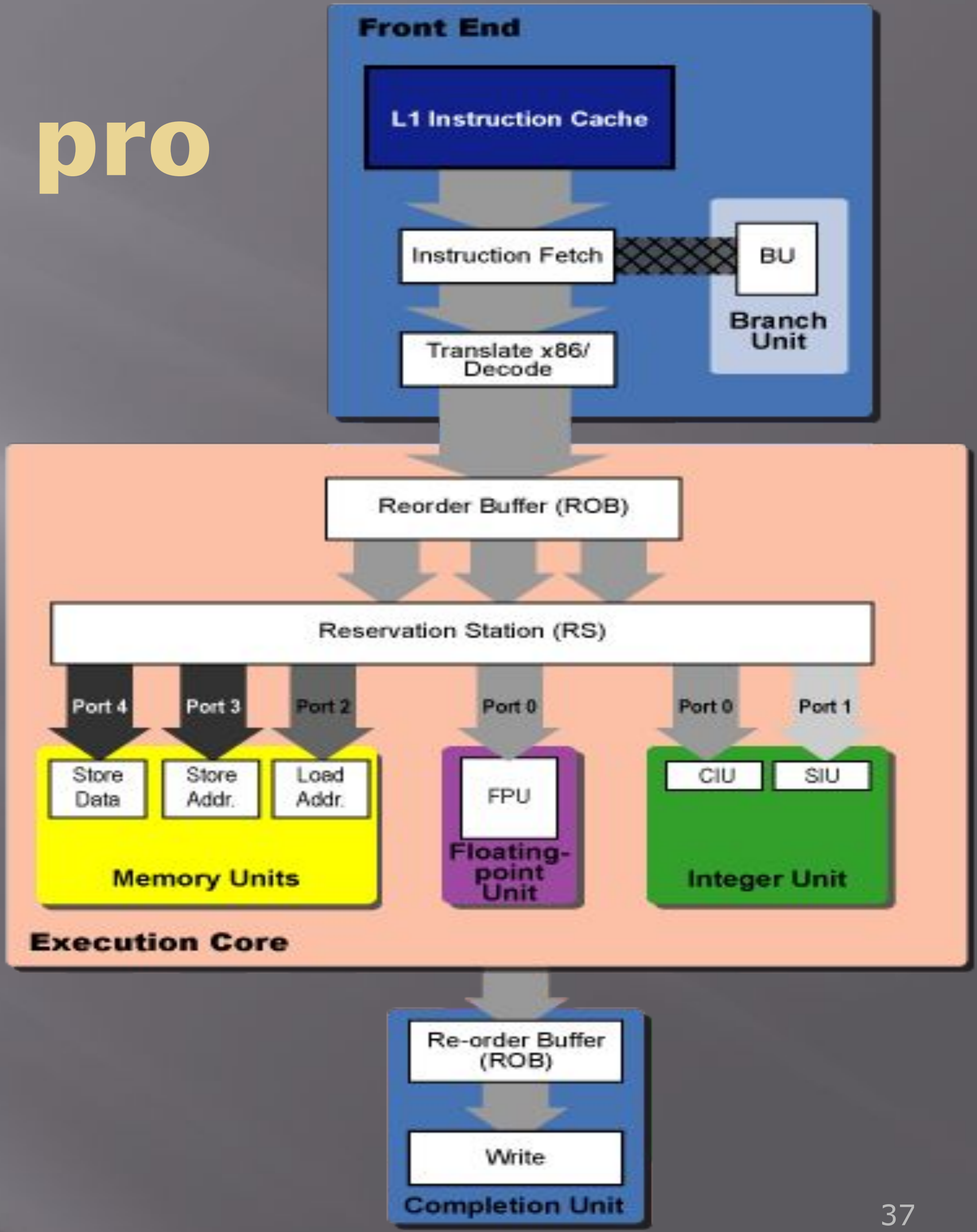
- Concept of adding more functional units.
- Theoretically:
  - N units = N instructions processing in parallel.
  - Problems in practice? **Order of instruction?**

Common instruction fetch unit.



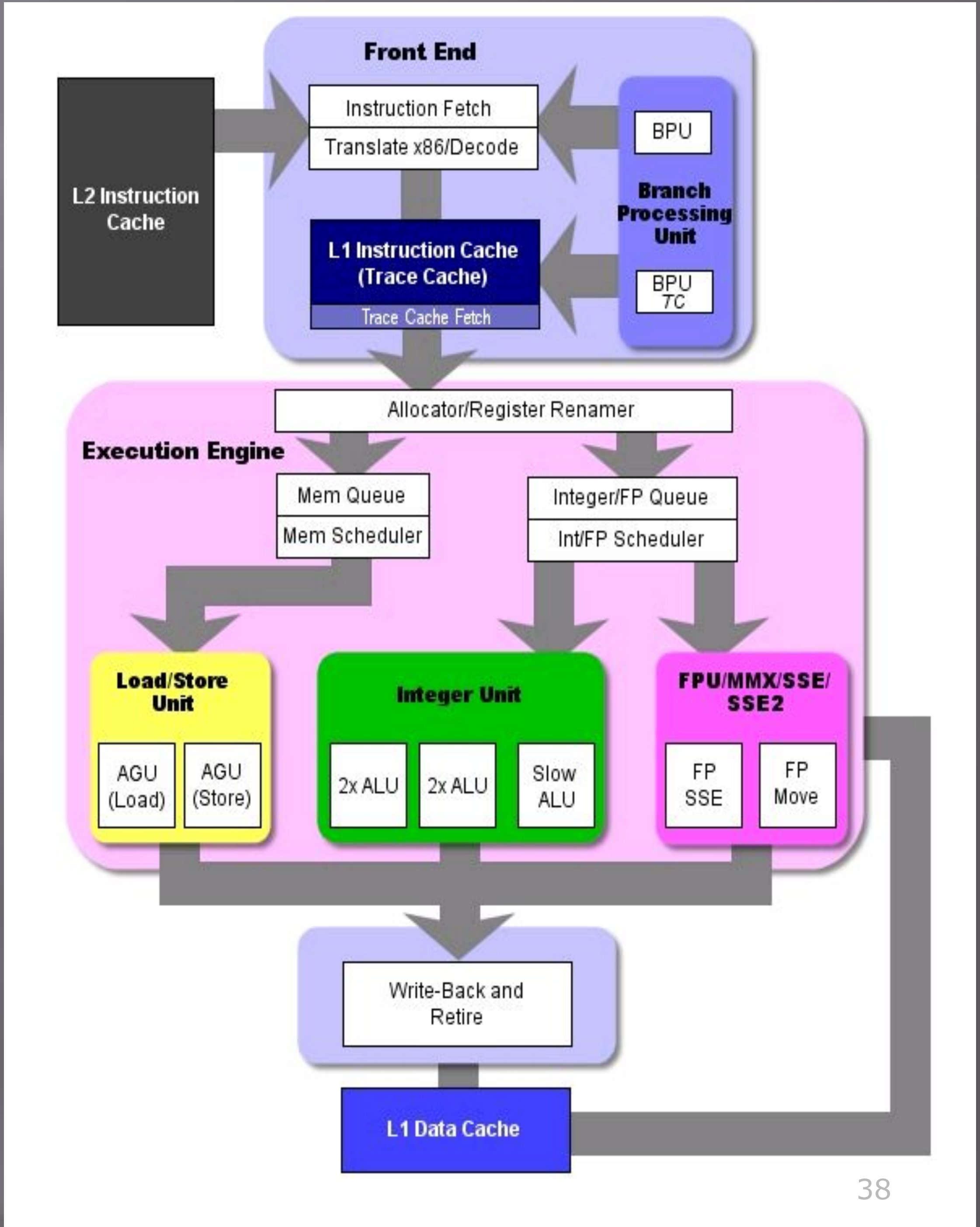
Pentium pro

- Division between front end and back end.
- Front end – fetch & decodes, convert to micro-ops.
- Reorder buffer (ROB).
- Reservation station.



Pentium 4

- 16k trace cache (12,000 micro-ops) (L1).
- 8 general purpose registers.
- 128 rename registers.
- 2 simple ALU (double speed).
- 1 complex ALU.
- FPU (dual pipeline, stack based).
- SSE(Streaming SIMD Extension).



Summary

- Storage options (stack, accumulator or register).
- CISC vs. RISC.
- Modern enhancements (cache, pipelines, superscalar).
- Modern processor examples.