Høyskolen Kristiania

Digital technology

TK1104-1 22H

Lecturer: Toktam Ramezanifarkhani

Toktam.Ramezanifarkhani@kristiania.no

Toktamr@ifi.uio.no

Elsa Lossius

ello008@student.kristiania.no

Høyskolen Kristiania

Instruction programming Computer Organization

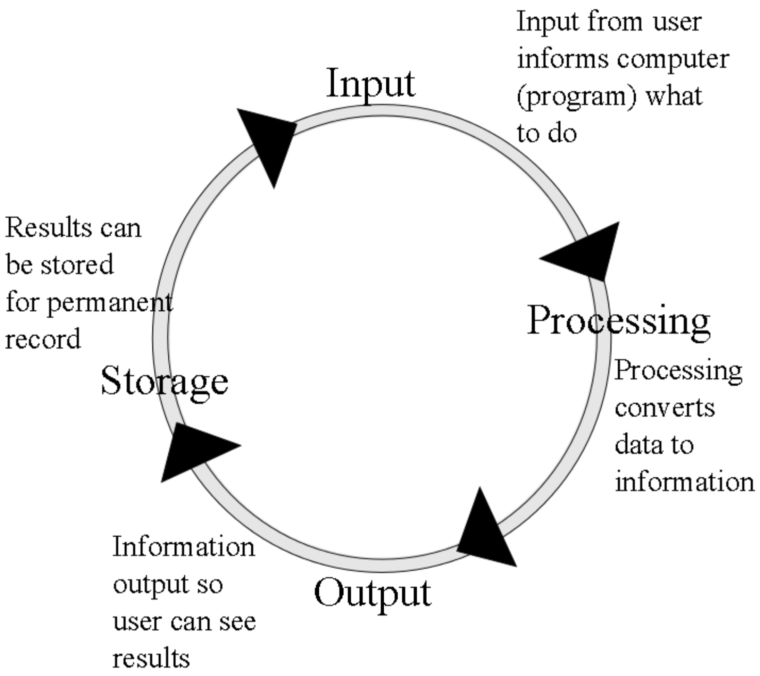
CPU
Programming Computer Organization

Høyskolen Kristiania

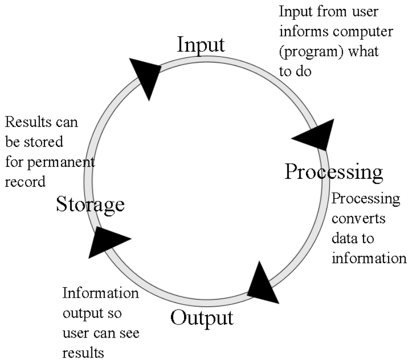
Computer Organization

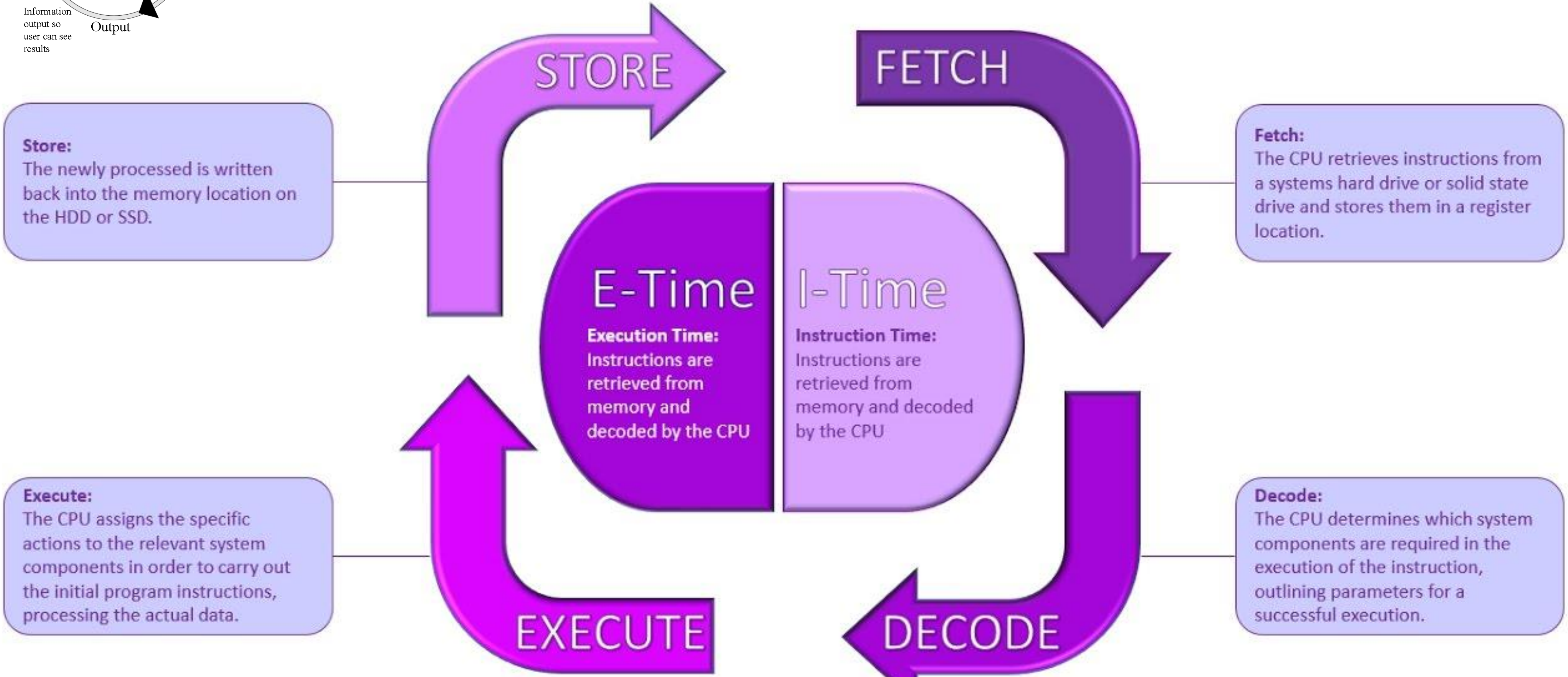
- All computers perform IPOS

  Input, Processing, Output, Storage
- Here, we concentrate on how IPOS is carried out through the fetch-execute cycle
- This requires that we study
  - the structure of the components in the computer
  - the function of those components
    - how the CPU works
    - the role of memory
    - the role of I/O devices



The Fetch-Execute Cycle



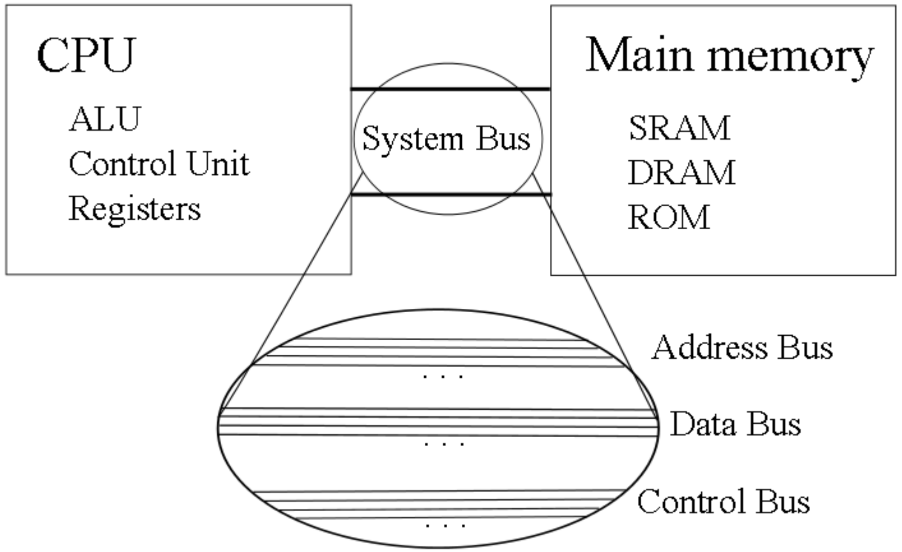


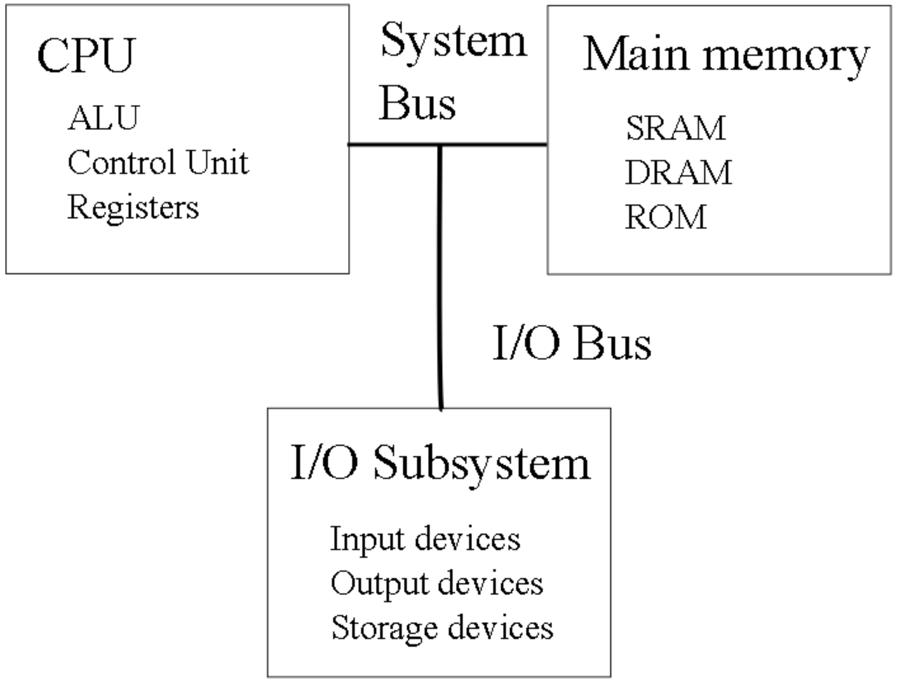
The Components of the CPU

- Arithmetic-Logic Unit
  - Contains circuits to perform arithmetic and logic operations
    - adder/subtracter
    - negater (convert + to -, - to +)
    - multiplier
    - divider
    - shifter
    - rotate
    - comparator
  - Sets status flags

- Control Unit
  - Operates the fetch-execute cycle
  - Decodes instructions
  - Sends out control signals
- Registers
  - Data register(s)
  - Control unit registers

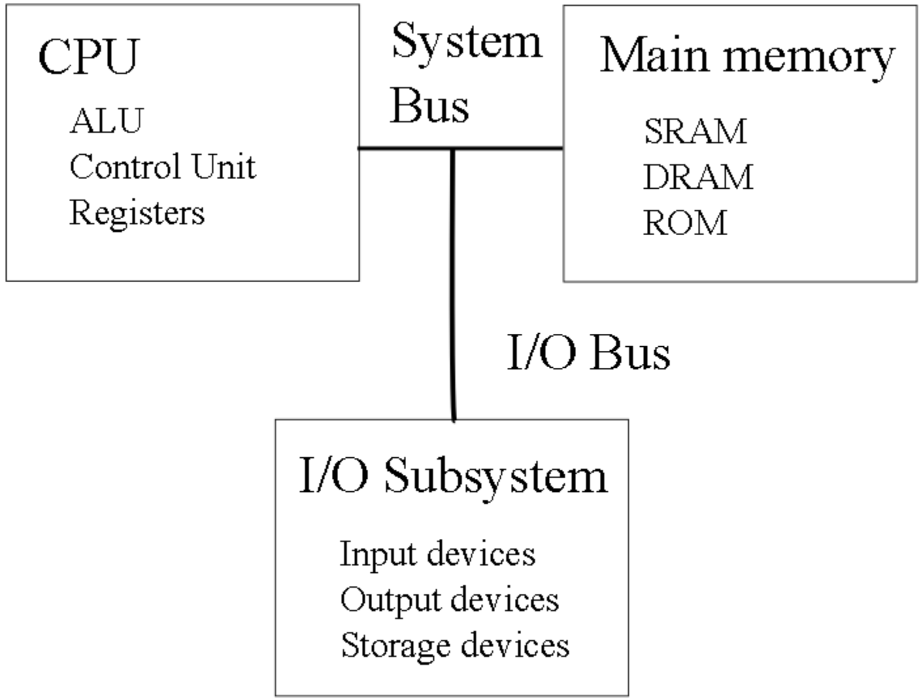
The Structure of the Computer





The Bus

- Address bus:
  - CPU sends address to memory or I/O subsystem
  - Address is the location of the item being moved
- Control bus:
  - CPU sends out commands to other devices
    - read, write for memory
    - input, output, are you available for I/O devices
  - Devices send signals back to the CPU such as interrupt
- Data bus:
  - Used to send data and program instructions
    - from memory to the CPU
    - from the CPU to memory
    - between I/O device and the CPU
    - between I/O device and memory
  - Size of data bus is often size of computer's word



Registers

Høyskolen Kristiania

- Temporary storage in the CPU
  - Store values used during the fetch execute cycle
  - PC – program counter
    - Memory location of next instruction, used during instruction fetch
  - Data registers
    - To store temporary results during execution
    - Some computers have one, the accumulator (AC), others have several, maybe dozens (eax, ebx, ecx, edx or R0, R1, R2, …, R31)
  - IR – instruction register
  - Current instruction, used during decoding
- Status flags
  - To store information about the result of the previous ALU operation
    - positive, negative, zero, even or odd parity, carry, overflow, interrupt

EBX is a "base" register

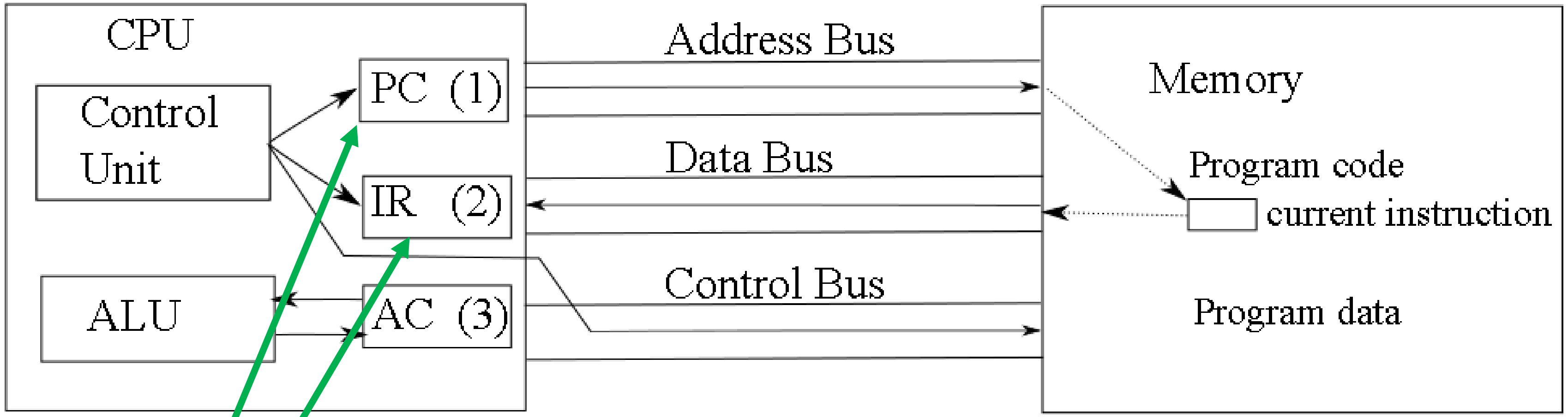
EAX is an "accumulator"

To execute a program

Høyskolen Kristiania

- CPU performs the fetch-execute cycle
  - Fetch next instruction from memory
  - Decode the instruction
  - Execute the instruction
  - Store the result

Fetch-Execute Cycle



CPU needs to know What to do.
CPU needs:
Where is the instruction
What is the instruction

1. Control unit moves PC to Address Bus and signals Memory "read" command over Control Bus, Memory returns instruction over Data Bus to be stored in IR
2. Control Unit decodes instruction in IR
3. Execute instruction in the ALU using datum in AC, putting result back in the AC

Processor operation with a program

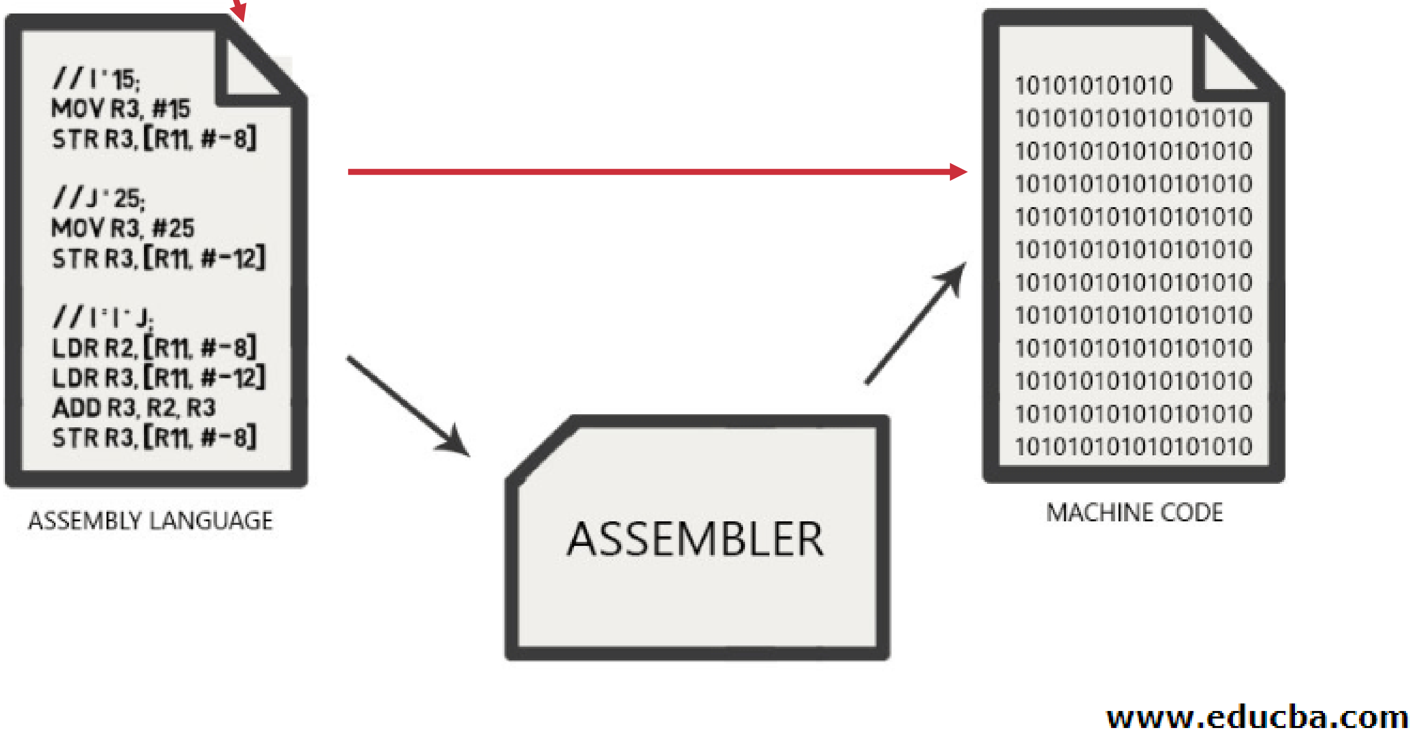
```
void main( )                                          // start of the program
{               int a, b, c;                // use 3 integer variables
                scanf("%d", &a);            // input a
                scanf("%d", &b);            // input b
                if(a < b)                   // compare a to b, if a is less then b
                     c=a + b;               // then set c to be their sum
                else c=a - b;               // otherwise set c to be their difference
                printf("%d", c);                      // output the result, c }
```

High level language
- C #, JAVA (and many others)
- Interpreted in an environment
(eg a virtual machine)

Low level language
- C, C ++ (and more)
- Compiled to Assembly and then
machine instructions

Machine language (IA32, IA64, etc.)
- Assembler language
(each command corresponds to a
machine instruction: mov eax, [00AF3B13])

Instructions for the processor = programs
- Retrieved to the processor from memory (RAM)



Example: A Program

- We use the following C program to better understand the fetch-execute cycle

```
#include <stdio.h>                    // input/output library

void main( )                          // start of the program
{
        int a, b, c;                  // use 3 integer variables
        scanf("%d", &a);              // input a
        scanf("%d", &b);              // input b
        if(a < b)                     // compare a to b, if a is less then b
            c=a + b;                  // then set c to be their sum
        else c=a - b;                 // otherwise set c to be their difference
        printf("%d", c);              // output the result, c
}
```

Program in Assembly Language

```
void main( )

{         int a, b, c;

          scanf("%d", &a);

          scanf("%d", &b);

          if(a < b)
               c=a + b;

          else c=a - b;
          printf("%d", c);

}
```

```
          Input 33        // assume 33 is the keyboard, input a value from keyboard
          Store a         // and store the value in the variable a
          Input 33        // repeat for b
          Store b
          Load a          // move a from memory to CPU, a location called the accumulator
          Subt  b         // subtract b from the accumulator (accumulator = a – b)
          Jge    else     // if the result is greater than or equal to 0, go to location "else"
          Load a          // otherwise, here we do the then clause, load a into accumulator
          Add   b         // add b (accumulator is now a + b)
          Store c         // store the result (a + b) in c
          Jump next       // go to the location called next
else:     Load a          // here is the else clause, load a into the accumulator
          Subt  b         // subtract b (accumulator is now a – b)
          Store c         // store the result (a – b) into c
next:     Load   c        // load c into the accumulator
          Output 2049     // send the accumulator value to the output device 2049, assume
                          //      this is the monitor
          Halt            // end the program
```

Program in Assembly Language

```
          Input 33        // assume 33 is the keyboard, input a value from keyboard
          Store a         // and store the value in the variable a
          Input 33        // repeat the input for b
          Store b
          Load a          // move a from memory to CPU, a location called the accumulator
          Subt  b         // subtract b from the accumulator (accumulator = a – b)
          Jge    else     // if the result is greater than or equal to 0, go to location “else”
          Load a          // otherwise, here we do the then clause, load a into accumulator
          Add   b         // add b (accumulator is now a + b)
          Store c         // store the result (a + b) in c
          Jump next       // go to the location called next
else:     Load a          // here is the else clause, load a into the accumulator
          Subt  b         // subtract b (accumulator is now a – b)
          Store c         // store the result (a – b) into c
next:     Load   c        // load c into the accumulator
          Output 2049     // send the accumulator value to the output device 2049, assume
                          //      this is the monitor
          Halt            // end the program
```

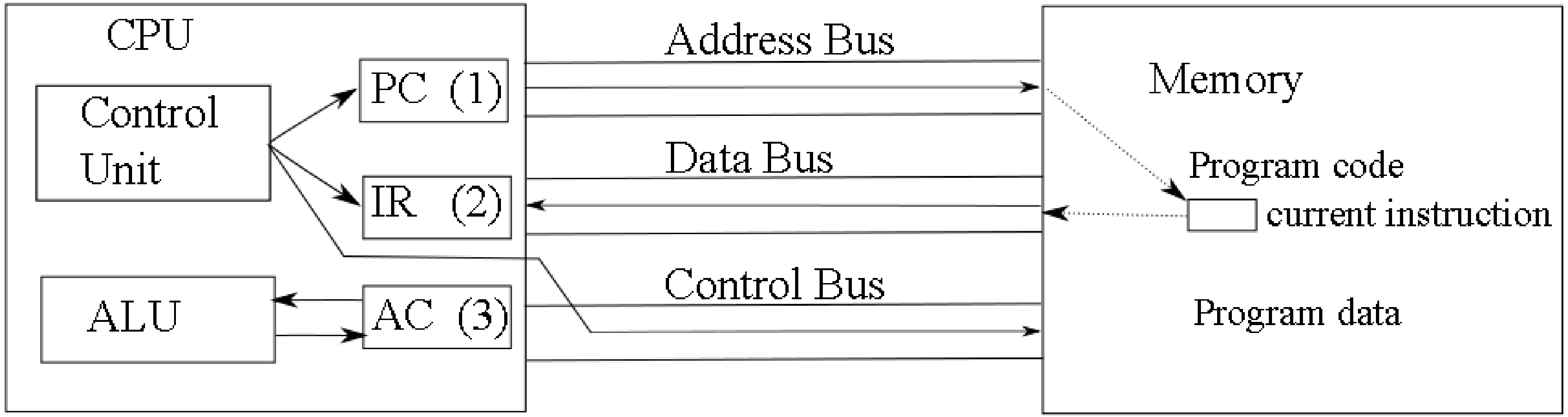
Program in Machine Language

- Assembly code version of our C program is stored in the computer in machine language
  - The first four instructions might look like this:

1000100 0000000000000000000100001 – input (from keyboard)
1000111 0010011000100101101010001 – store the datum in a
1000100 0000000000000000000100001 – input (from keyboard)
1000111 0010011000100101101010010 – store the datum in b

op code operand (datum)

Fetch-Execute Cycle – (reminder)



CPU needs to know What to do.
CPU needs:
Where is the instruction
What is the instruction

1. Control unit moves PC to Address Bus and signals Memory "read" command over Control Bus, Memory returns instruction over Data Bus to be stored in IR
2. Control Unit decodes instruction in IR
3. Execute instruction in the ALU using datum in AC, putting result back in the AC

When memory sends an instruction to CPU, it uses data bus
When CPU sends the instruction to the memory, control bus is used

Fetch-Execute Cycle: Details

```
        Input 33
        Store a
        Input 33
        Store b
        Load a
        Subt  b
        Jge     else
        Load a
        Add   b
        Store c
        Jump next
else:   Load a
        Subt  b
        Store c
next:   Load   c
        Output 2049
        Halt
```

- Fetch:
  - PC stores address of next instruction
  - Fetch instruction at PC location
  - Increment PC
  - Instruction sent over data bus
  - Store instruction in IR
- Decode:
  - Decode opcode portion in IR
  - Determine operand(s) from instruction in IR
- Execute:
  - Issue command(s) to proper circuits
  - Use data register(s)
- Store result
  - In AC (or data register), or memory

Fetch-Execute Cycle: Example

- Assume our program starts at location 5,000,000
  - PC: 5,000,000
  - IR: -------
- Fetch instruction
  - PC: 5,000,000
  - IR: 1000100 0000000000000000000100001
  - Increment PC to 5,000,001
- Decode instruction
  - Input operation (obtain input from keyboard)
- Execute:
  - Take input value from keyboard
  - Move to AC

The address of the starting block of the program in the memory is 5,000,000

	Input 33	Store a	

Continued

- Fetch instruction
  - PC: 5,000,001
  - IR: 1000111 0010011000100101101010001
  - Increment PC to 5,000,002
- Decode instruction
  - Store datum to memory location 0010011000100101101010001 (memory location storing variable a)
- Execute:
  - Move datum from AC over data bus to memory location a
    - NOTE: the next two instructions are almost identical except that the second input's datum (from the third instruction) is sent to memory location b instead of a)

Continued

```
         Input 33
         Store a
         Input 33
         Store b
         Load a
         Subt  b
         Jge     else
         Load a
         Add   b
         Store c
         Jump next
else:    Load a
         Subt  b
         Store c
next:    Load   c
         Output 2049
         Halt
```

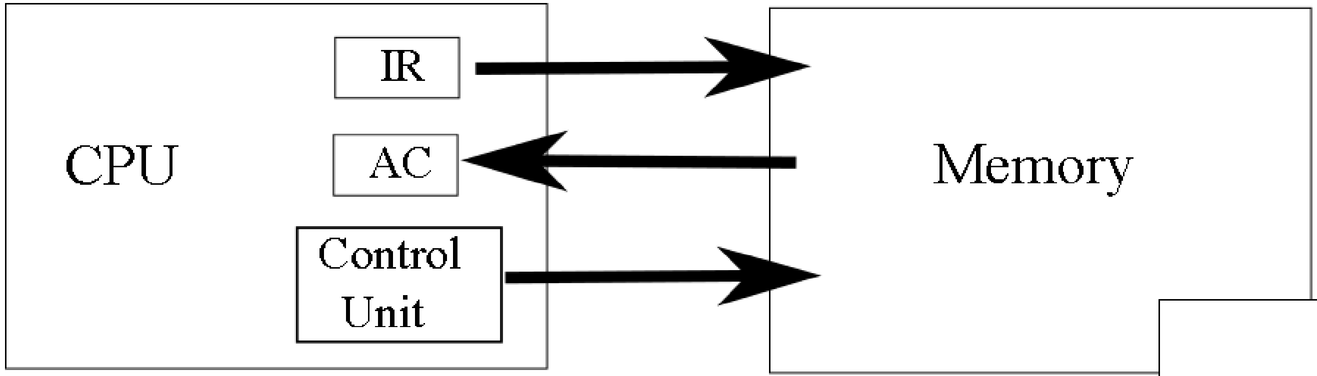
- Load a
  - Fetch instruction at 5,000,004
  - Increment PC to 5,000,005
  - Decode – load instruction, operand is a from memory
  - Execute – loads datum at location a into AC
- Subt b
  - Fetch instruction at 5,000,005
  - Increment PC to 5,000,006
  - Decode – subtract instruction, operands are AC register and b from memory
  - Execute – fetch b from memory, send AC and b to subtracter circuit
  - Store result in AC
  - Set status flags as appropriate (negative, positive, zero, carry, overflow)

Continued

```
         Input 33
         Store a
         Input 33
         Store b
         Load a
         Subt  b
         Jge    else
         Load a
         Add   b
         Store c
         Jump next
else:    Load a
         Subt  b
         Store c
next:    Load   c
         Output 2049
         Halt
```

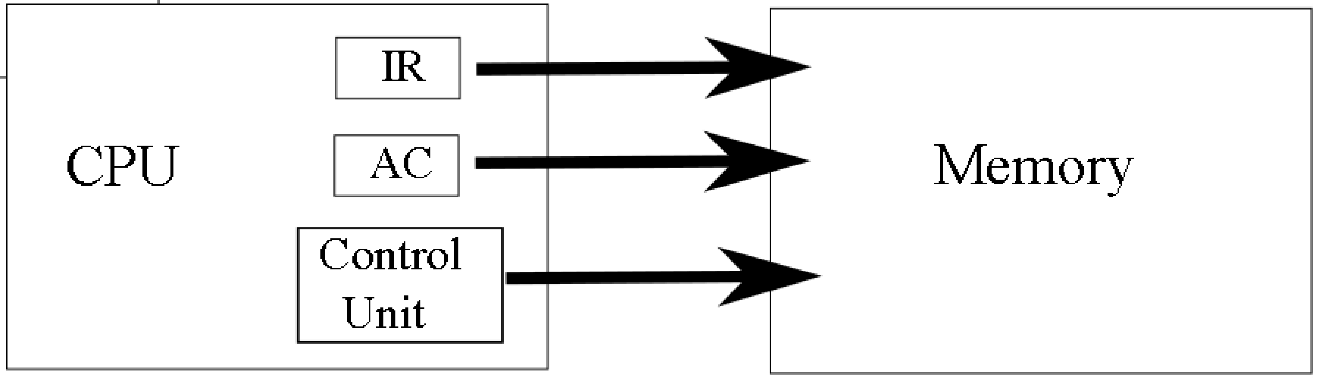
- Jge else –a *branch* instruction
  - Fetch instruction at 5,000,006
  - Increment PC to 5,000,007
  - Decode instruction
  - Execute instruction – if positive or zero flag are set (1) reset PC to 5,000,011 (branches to “else”) otherwise, end instruction
- Next instruction fetch is 5,000,007 or 5,000,011
- Next few instructions executed depend on the previous conditional branch
  - Either “Load a”, “Add b”, “Store c”, “Jump next”
  - Or “Load a”, “Subt b”, “Store c”
- Jump next – PC altered to 5,000,014 (location of “next”)
- Output 2049 – outputs value in AC to device 2049 (the monitor)
- Halt – ends the program

Execute Memory Read instruction after Decode:



1. Address from IR to address bus
   Control unit signals memory read over control bus
2. Memory accesses address
   Returns datum over data bus
3. Datum stored in AC

Execute Memory Write instruction after Decode:



1. Address from IR to address bus
   Datum from AC over data bus
   Control unit signals memory write over control bus
2. Memory accesses address
   Stores datum from data bus to memory location

Tutorials

- Demonstration of the fetch execute cycle
- Memory read operation
- Memory write operation

https://www.youtube.com/watch?v=xs5oq-i_rTc&t=65s
Fetch Decode Execute Cycle in more detail
https://youtu.be/jFDMZpkUWCw

CPU Performance

Microcode and System Clock

- Each clock cycle, the control unit issues instructions to the devices in the computer (1 part of the fetch-execute cycle, not a full instruction)
- These instructions are in the form of microcode
  - 1 bit per line on the control bus
- Example: instruction fetch
  - Move PC to address bus, Signal memory read, might look like
    - 10000000000000100000000000000000000000000000000000
  - 1 clock cycle = 1 microinstruction executed
- Fetch-execute cycle might have between 5 and 30 steps depending on architecture
- Clock speeds are given in GHz
  - 1 GHz = 1 billion clock cycles per second, or 1 clock cycle executes in 1 billionth of a second (1 nanosecond)

Measuring CPU Performance

- A faster clock does not necessarily mean a faster CPU

  CPU1 2.5 GHz, 12 stage fetch-execute cycle requiring 20 cycles to complete 1 instruction
  CPU2 1 GHz, 5 stage fetch-execute cycle requiring 8 cycles to complete 1 instruction
  CPU1 = 20 / 2.5 GHz = 8 nanoseconds / instruction
  CPU2 = 8 / 1 GHz = 8 nanoseconds / instruction

- Other impacts of CPU performance include
  - Word size – size of datum being moved/processed
  - Cache performance (amount and usage of cache)
  - The program being run (some programs are slower than others)
  - How is the OS load
  - Virtual memory performance
  - Any parallel processing hardware?
- Best measure of CPU performance is to examine benchmark results

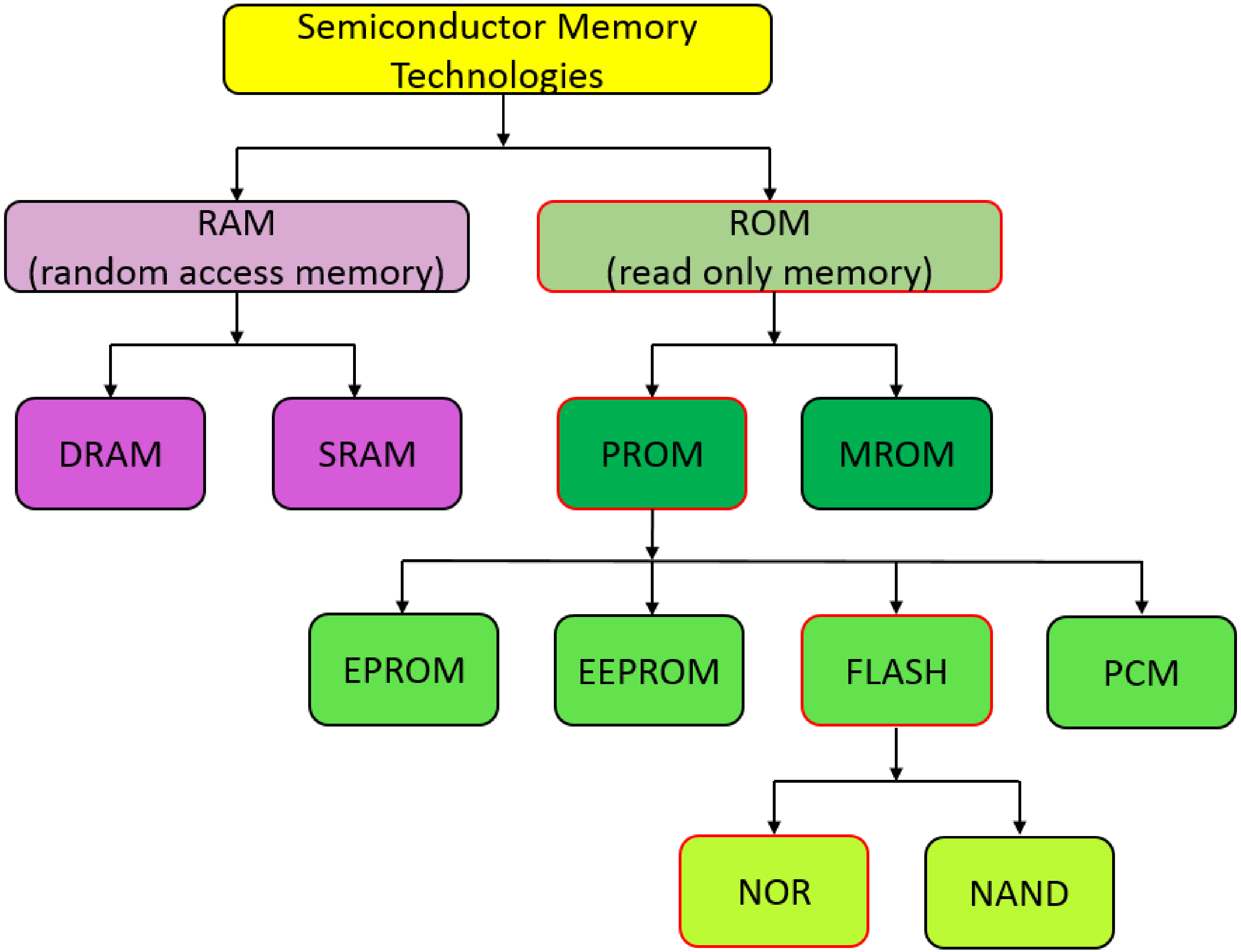
Memory

Role of Memory

- Memory is referenced at every instruction
  - 1 instruction fetch
  - Possibly 1 or more data accesses
    - data read as in Subt b
    - data write as in Store a
  - In Intel x86 architecture, Add x, 5 involves 3 memory references
    - instruction fetch
    - load x for addition
    - store result in x

- Random Access Memory: Dynamic and Static
  - There are several types of RAM
  - DRAM – “main memory”
    - made of capacitors
    - requires timely refreshing
    - slow but very cheap
  - SRAM – cache and registers
    - much faster-made of flip fops
    - but more expensive
  - ROM – read only memory
    - used to store boot program, BIOS
    - information permanent (cannot be altered)
    - even more expensive

Memory



RAM (random access memory)
- cannot hold the data without the power, whereas

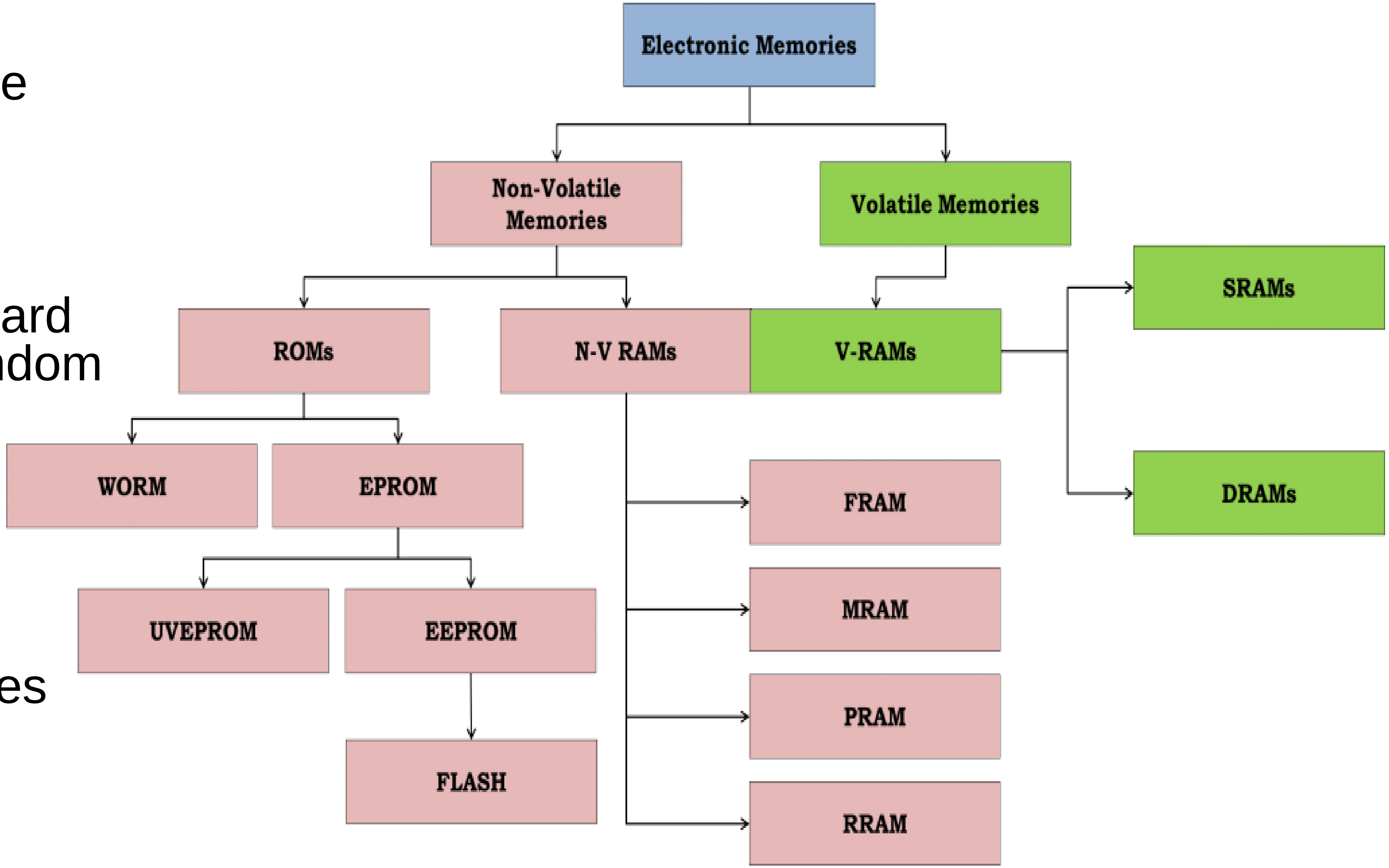
ROM (read-only memory)
- can hold the data even without the power
- With RAM, writing data is a much faster and lightening process, whereas ROM, writing data speed is much slower as compared to RAM.

RAM and ROM: both are random access and not sequential

- https://www.mphysicstutorial.com/2020/12/semiconductor-memory-types-ram-rom-dram.html
- https://m.facebook.com/Tipsoncomputermaintenanceandknowledge11/posts/ram-random-access-memory-and-rom-read-only-memory-are-the-two-important-memory-t/391442481220798/

Memory

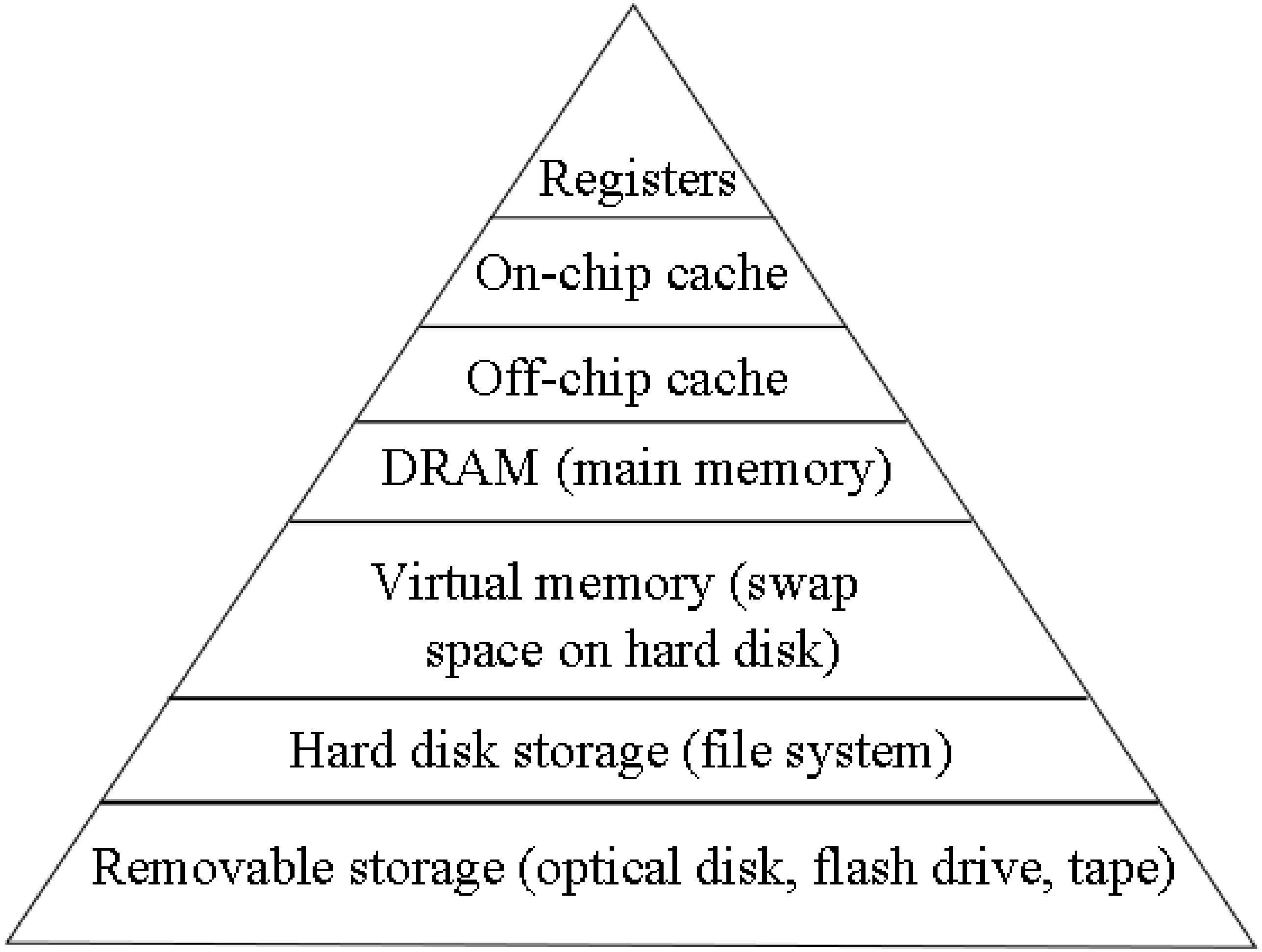
- HDD are non volatile
- The new version of ROM are not just readable anymore.
- Modern computer hard disks are using Random Access



- Tape Storage devices (used for offline backups) use Sequential Access.

https://sites.unica.it/dealab/organic-memories/

Memory Hierarchy



Self-Study

Using The Memory Hierarchy

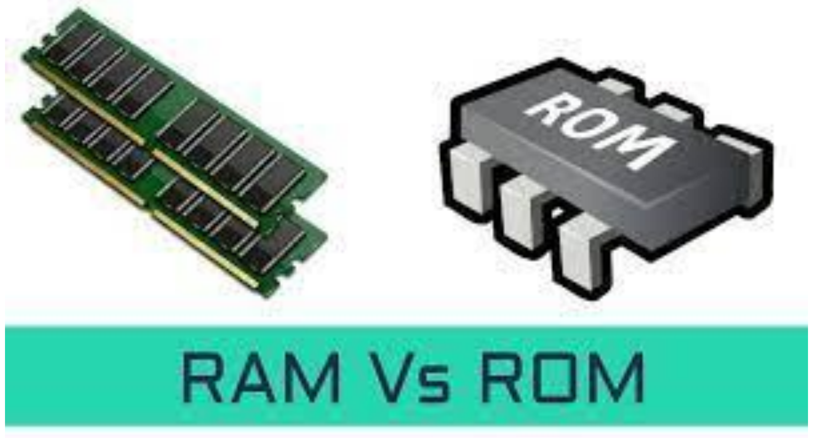
Høyskolen Kristiania

- The goal is to access only the highest levels of the hierarchy
  - On a miss, move down to the next level
    - cache hit rates are as high as 98-99%
    - misses happen in 1 to 2 of every 100 accesses
  - Bring item up to higher level
  - Bring its neighbors up as well in hopes of using them
- Lower levels act as “backstops”
  - On-chip caches are 32KB to 64KB today
  - Off-chip cache might be as large as 8MB
  - Main memory (DRAM) up to 8GB
  - Use hard disk space for memory’s backstop, known as swap space or virtual memory
- When something is not in memory
  - Need to swap it from disk
  - May require discarding something from memory

Memory

- Storage of data and programs
  - Ferrite cores (https://www.youtube.com/watch?v=HrlkEQvvxUA)
  - RAM (Random Access Memory)
  - ROM (Read Only Memory), PROM, EPROM, Flash-RAM

- «Virtual memory»
  - Fast cache
  - Level 1 - internal (on-die, on-chip); ~ KB
- Level 2 - internal / external; ~ MB
  - Level 3 - externally ~ 10 MB



The Role of I/O

- I/O – input and output
  - All I/O takes place in the I/O subsystem
  - Devices connect to computer by expansion cards and ports
- Earliest form of I/O was punch cards (input) and printer (output) with magnetic tape used as intermediate storage
  - No direct interaction with computer
- Today, we expect to interact directly with the computer
  - Pointing devices, keyboard, microphone, monitor, speakers
- To improve interactivity, human computer interaction (HCI) combines
  - Computer science
  - Psychology
  - Design
  - Health
- And ergonomics
  - Reduce stress on the body (repetitive stress injuries very prevalent, particularly Carpal Tunnel Syndrome)
  - Improve accessibility for people with handicaps through larger monitors, speech recognition, Braille output devices
  - See for instance Rehabilitation Act section 508

The Portable Computer

Høyskolen Kristiania

- We no longer view the computer as a stationary device
  - Laptops, notebooks
  - Handheld devices
- Recent and near-future I/O devices
  - Wearables
  - Touch screens
  - Virtual reality interfaces
  - Sensor networks
  - Plug and play
- With wireless access we have
  - Interaction anywhere

Tutorials

- Personal Computer Architecture

https://www.youtube.com/watch?v=_I8CLQazom0&list=PLTd6ceoshprfg23JMtwGysCm4tlc0I1ou&index=2

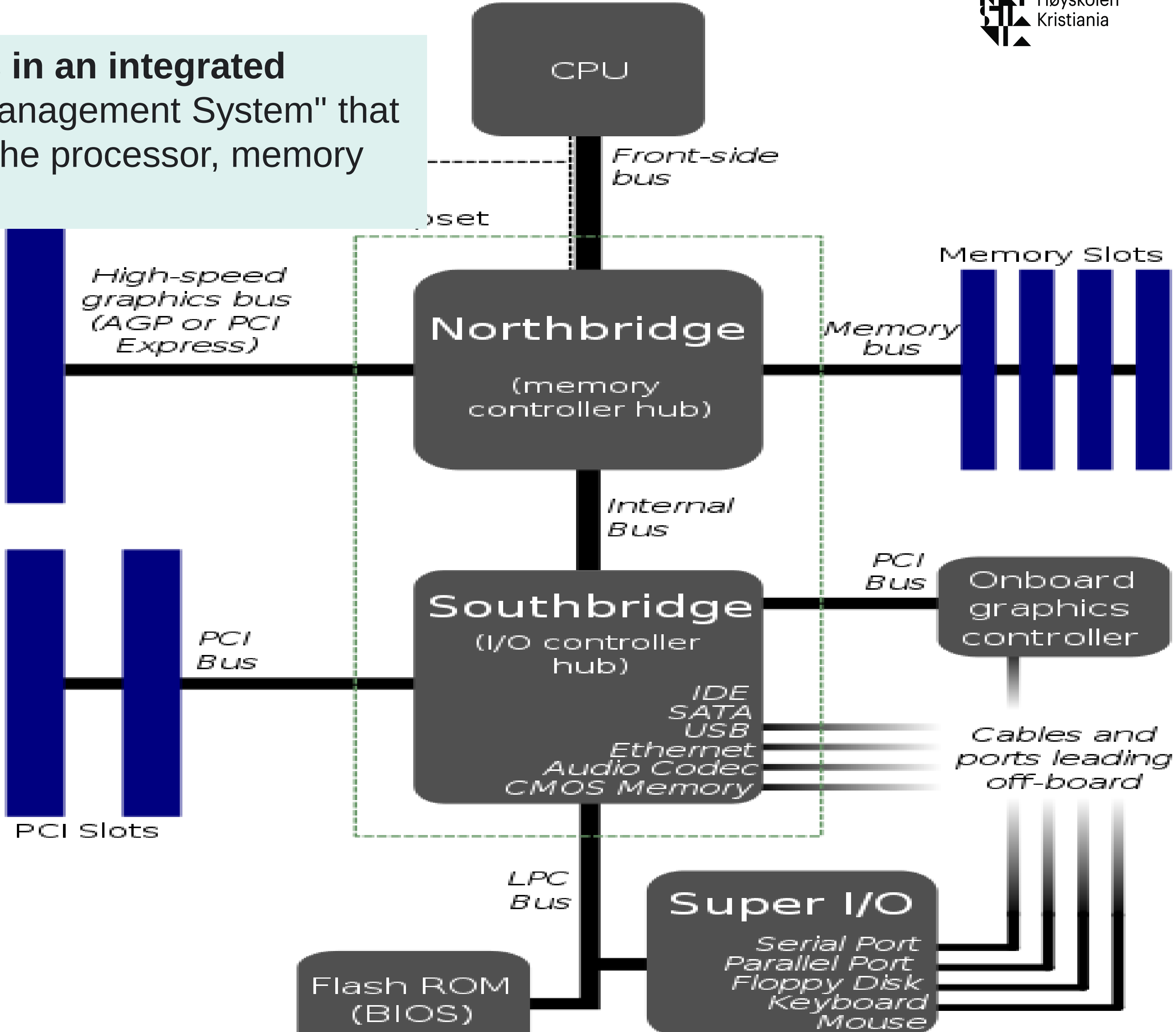
(https://www.youtube.com/watch?list=PLTd6ceoshprfg23JMtwGysCm4tlc0I1ou&v=_I8CLQazom0&feature=emb_logo)

CPU in Motherboard

Motherboard

a set of electronic components in an integrated circuit known as a "Data Flow Management System" that manages the data flow between the processor, memory and peripherals

- Conn
  - C
  - Chipset
  - Buses
  - Memory tracks and RAM
  - Expansion slots and extra cards
  - Gates and «plugs»



Modern Motherboard Ports & Chipsets

DIMM (dual in-line memory module) slots are the place on your **motherboard** where the **RAM** goes

1. CPU socket

2. Chipset

3. DIMM/RAM slots

X16 ideal for large and power hungry cards like a

To connect hard drives to motherboards

10. USB 3.1 Gen

The BIOS is the pro **stores the date, tim** computer. ... CMOS refer to the chip that

13. CPU power connector

This slides is related to the harware.
You can self-read on canvas.

16. Fan headers

17. Front panel header

18. VRM heatsink

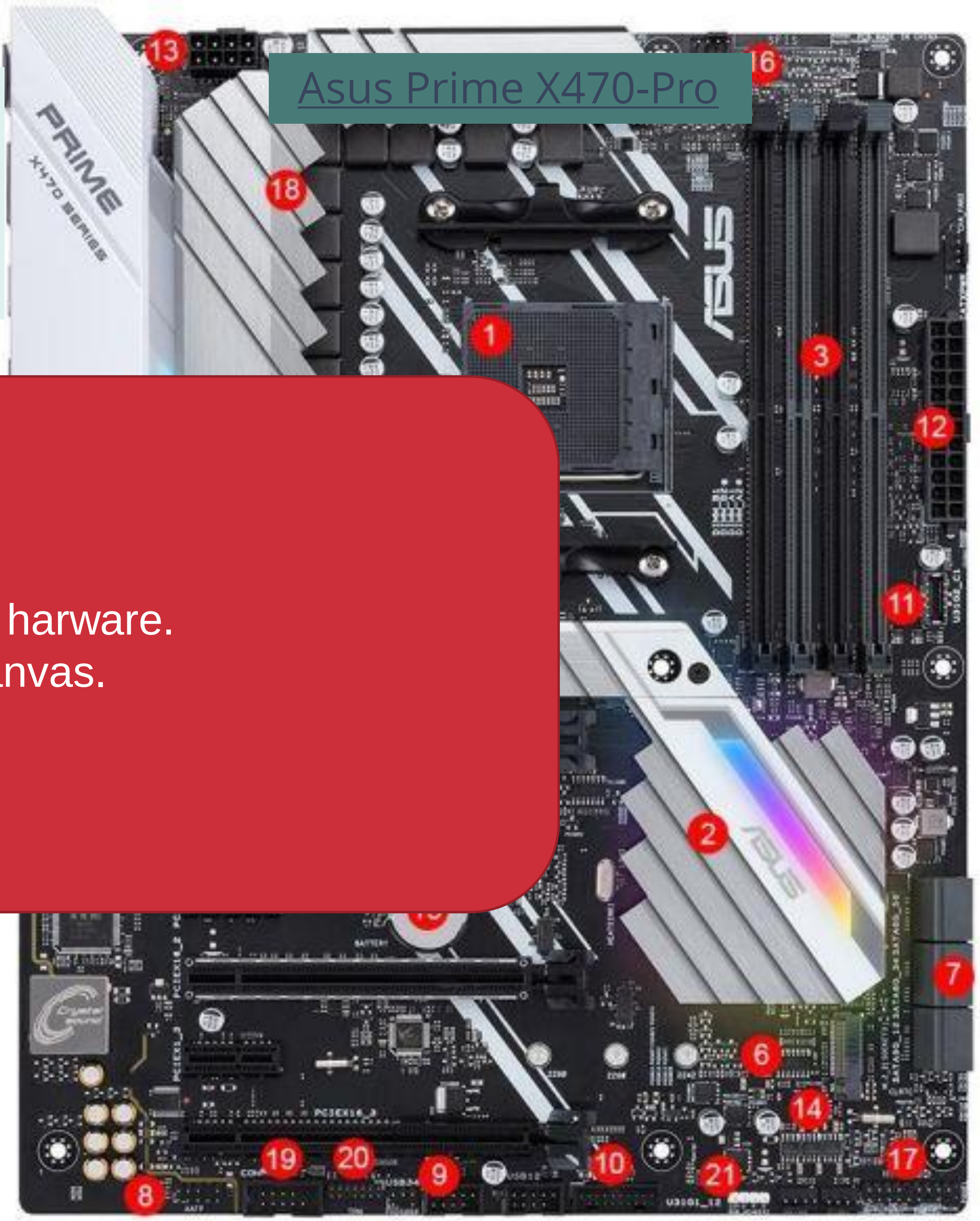
Voltage Regulator Thermal Module Pad

19. COM/Serial header

20. TPM header

21. RGB header

Asus Prime X470-Pro



End

Høyskolen Kristiania

Lab assignment - Today

- Review the lecture
  - Get into groups of two-three
  - Read and help each other to “Decode what is fetched” in the lecture time ☺
- Work on your assignment on Metaverse,
  - Find your group
  - Choose your topic and some references
  - Separate duties
- Outcomes
  - Practice the group work
  - Learn for your exam