MF20006: Introduction to Computer Science

# Lecture 2: Computer Architecture

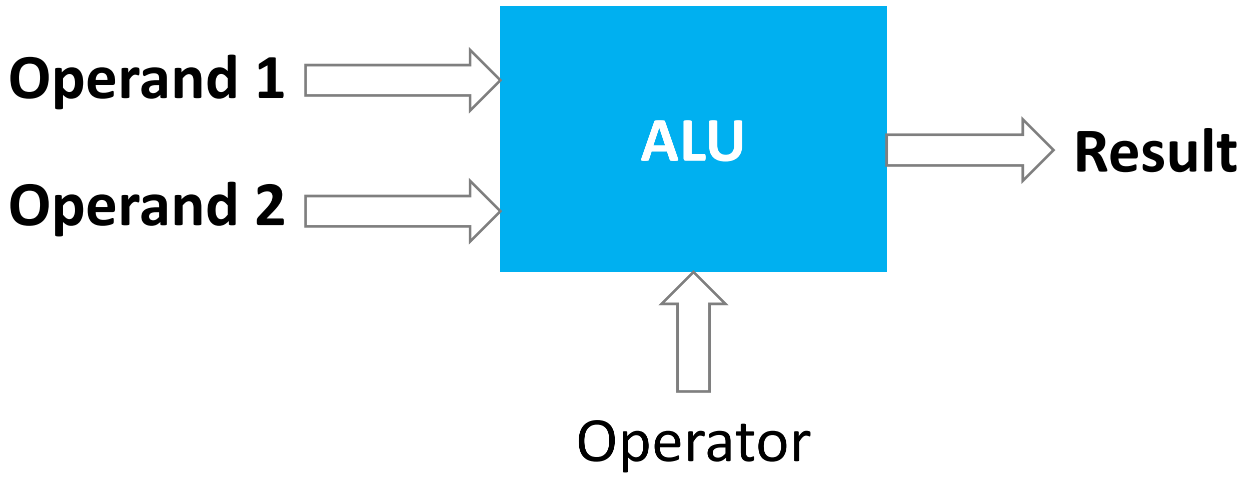
Hui Xu

xuh@fudan.edu.cn

## Outline

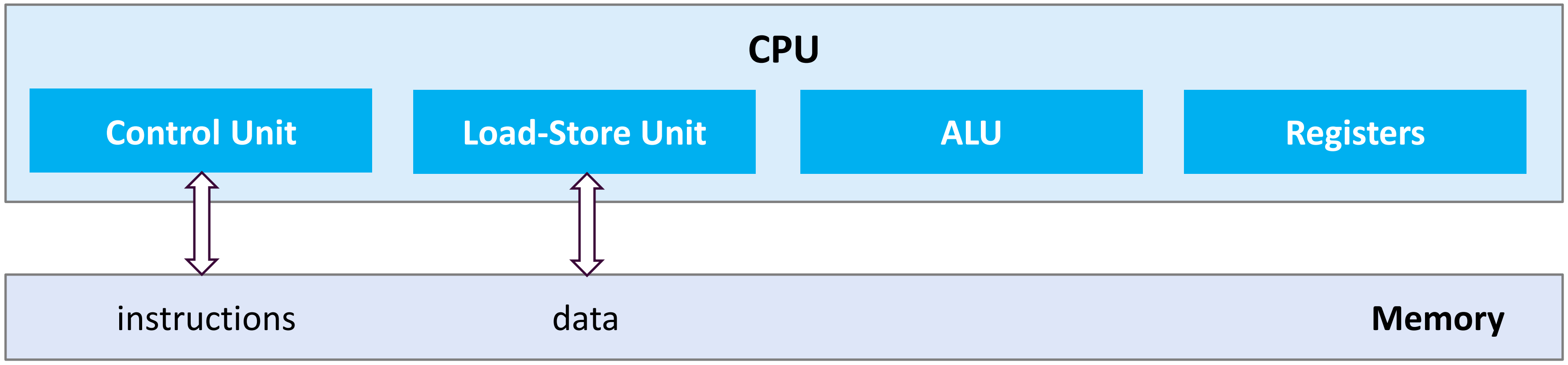
## 1. CPU and GPU

## Recall: Transistor => Logic Gate => Arithmetic Logic Unit

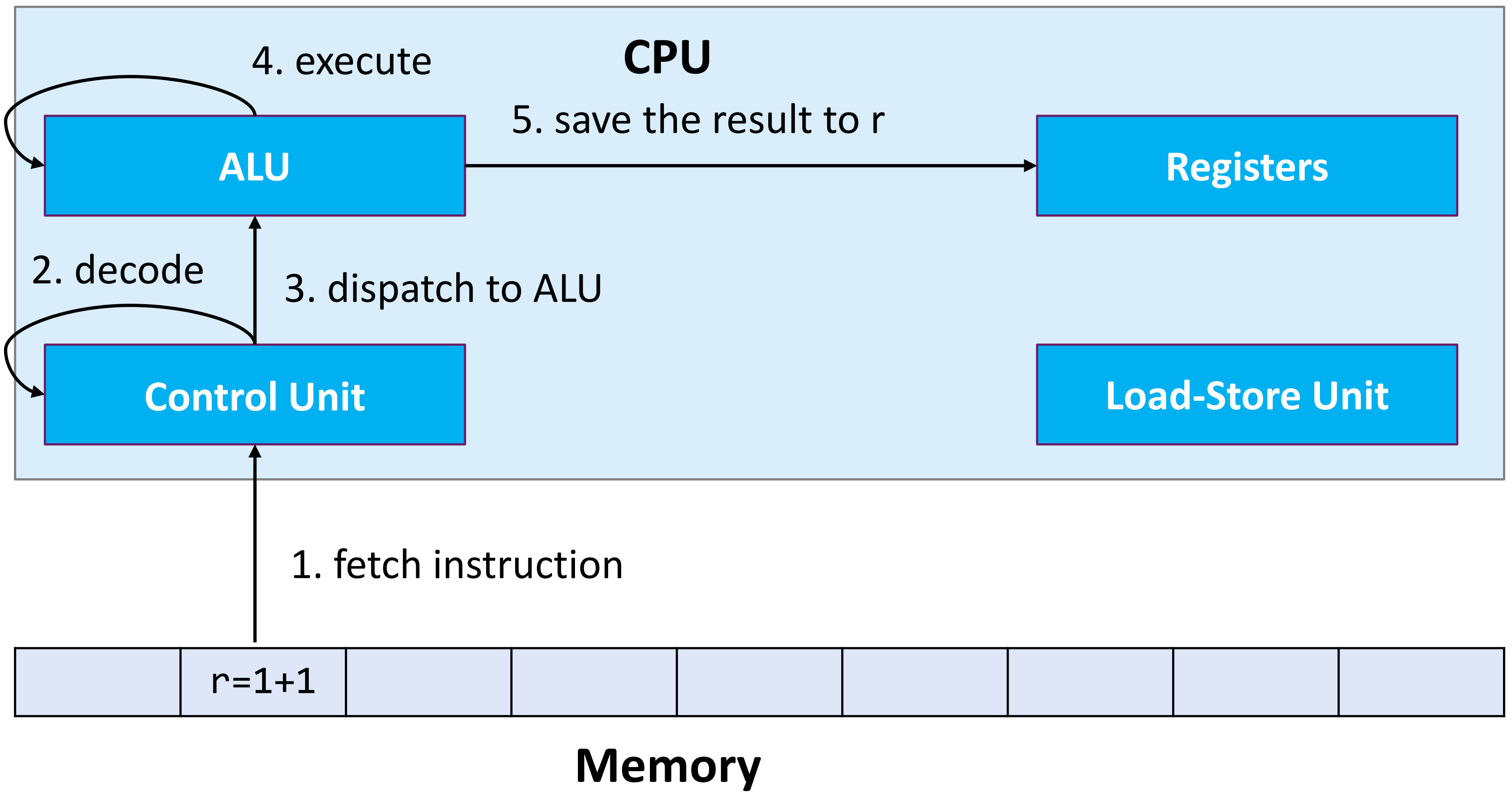


## CPU: Central Processing Unit

- ❑Arithmetic logic unit: Perform arithmetic/logic operations.
- ❑Control unit: Fetch instructions from memory; decode and dispatch.
- ❑Load-store unit: Load/store data from/to memory.
- ❑Registers: Supply operands to the ALU and save the execution result.

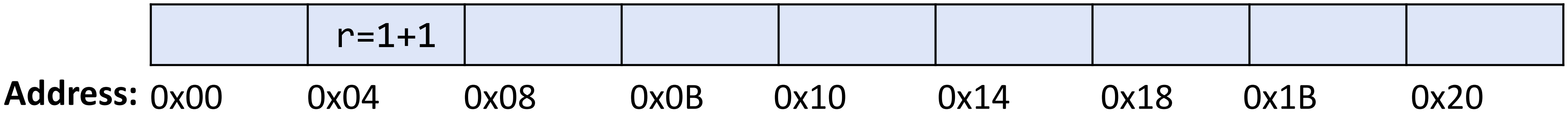


## Example: r = 1 + 1


CPU
4. execute
5. save the result to r
ALU
Registers
2. decode
3. dispatch to ALU
Control Unit
Load-Store Unit
1. fetch instruction
r=1+1
Memory

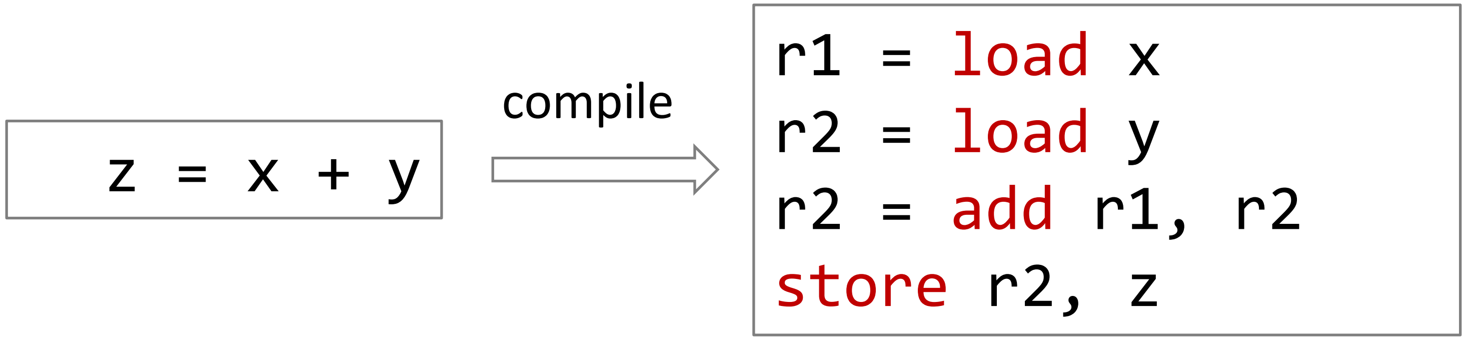
## Memory (Random-Access Memory or RAM)

- ❑Read or write a data item based on its address in memory.
- ❑Each data unit generally takes 1 byte or 8 bits.
- ❑Access any data with the same latency irrespective of their locations.



## Example: z = x + y

- ❑x and y are two integers stored in memory.
- ❑Compiling the code generates four instructions.



```
z = x + y
```

```
r1 = load x
r2 = load y
r2 = add r1, r2
store r2, z
```

High-level Code

Low-level Instructions

## Memory View

- ❑We may assume each instruction is encoded as 32 bits (4 bytes).
- ❑The four instructions are stored consecutively in memory with 16 bytes.
- ❑The variables x, y, and z are symbolic names that will be replaced with their corresponding memory addresses.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | `r1=load 0x18` | `r2=load 0x1B` | `r2=add r1,r2` | `store r2,0x20` | | x | y | z |
| **Address:** | 0x00 | 0x04 | 0x08 | 0x0B | 0x10 | 0x14 | 0x18 | 0x1B | 0x20 |

**Memory**

## Demonstration with Linux

Write a simple C program (you may generate via AI)

```
#: gcc target.c -o target
```

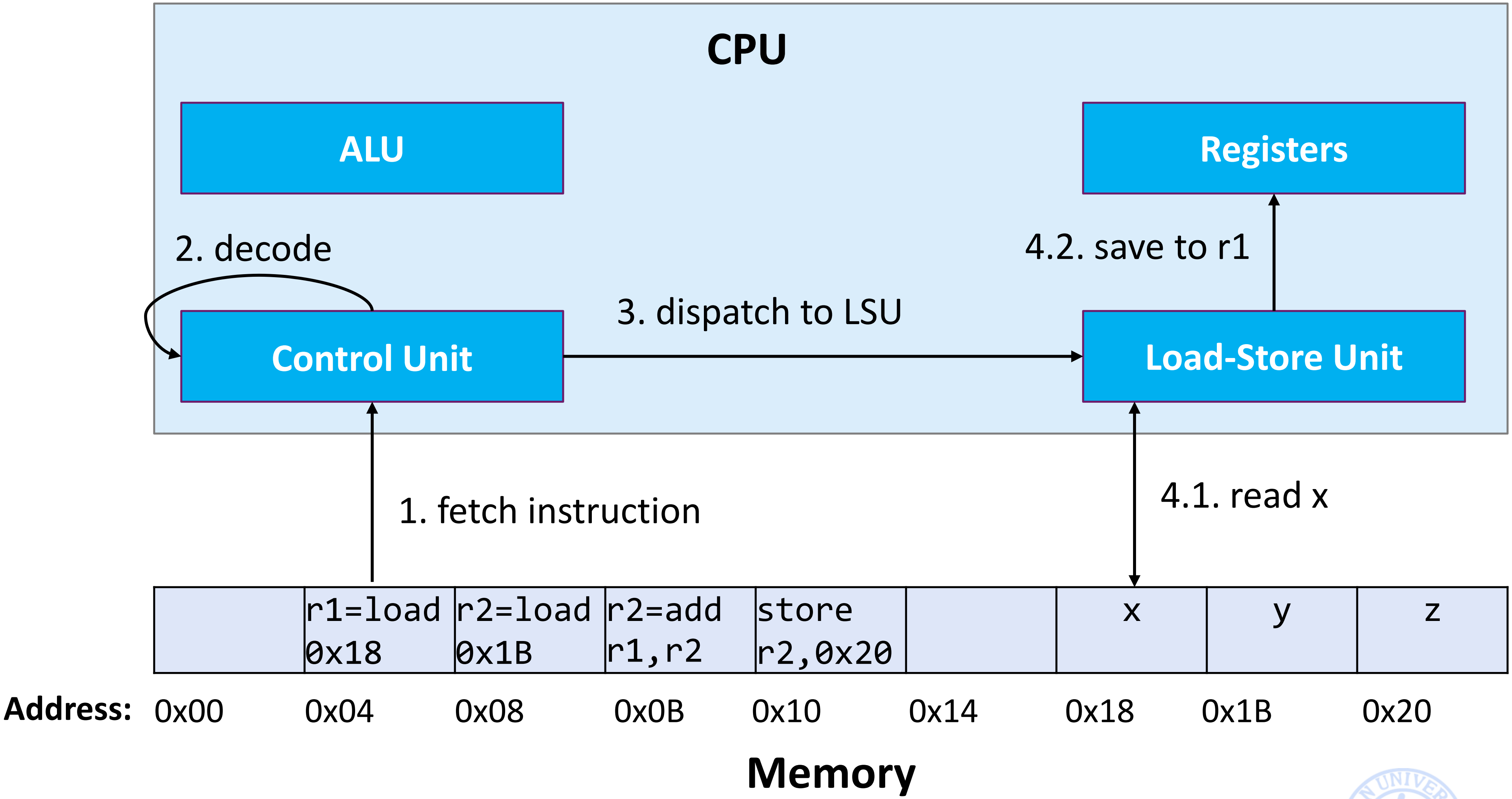
Single step execution via GDB

```
#: gdb target
(gdb) disass main
(gdb) break main
(gdb) run
(gdb) si
```

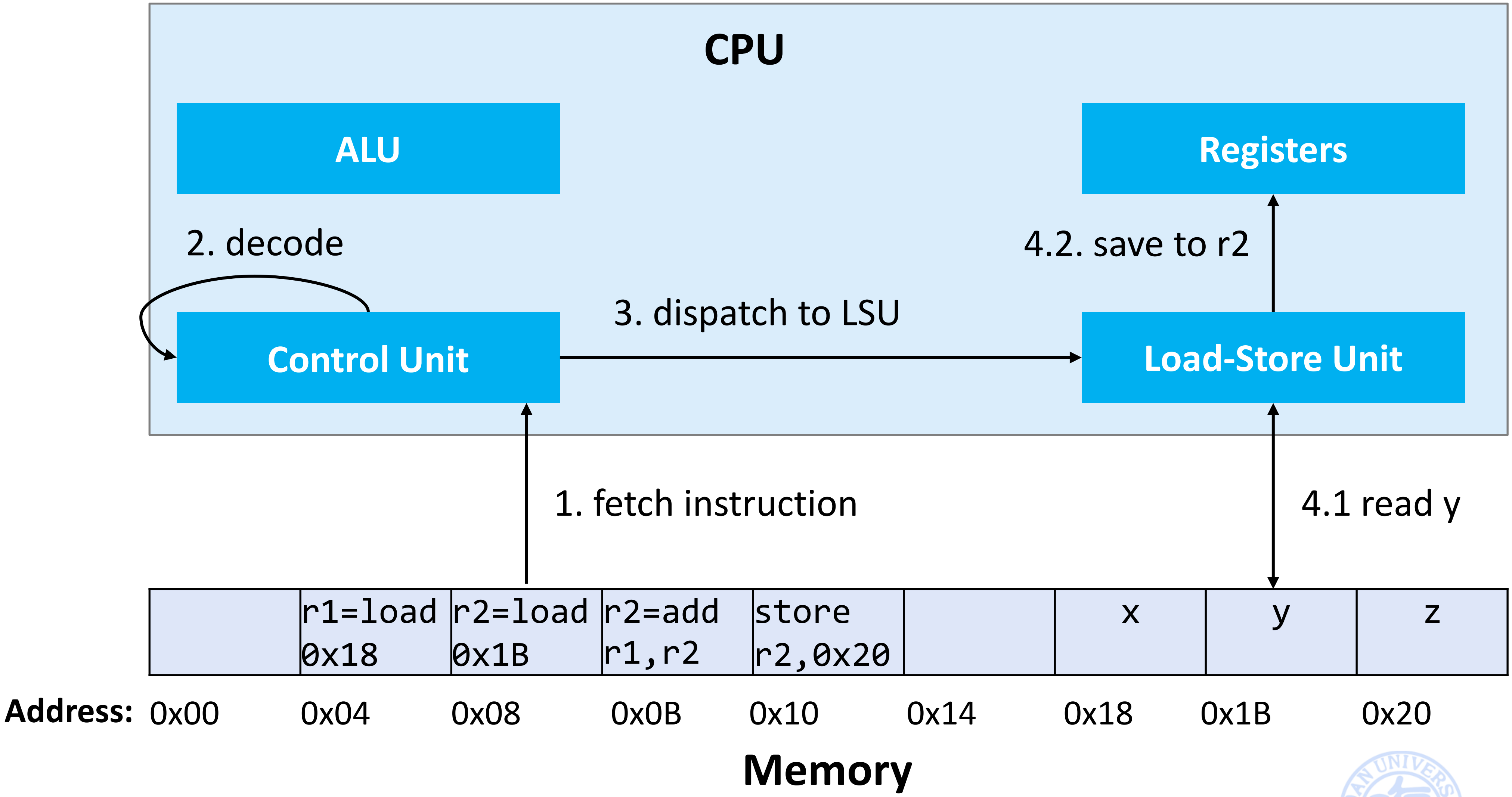
View the whole memory content of a process:

```
#: cat /proc/$pid/maps
```

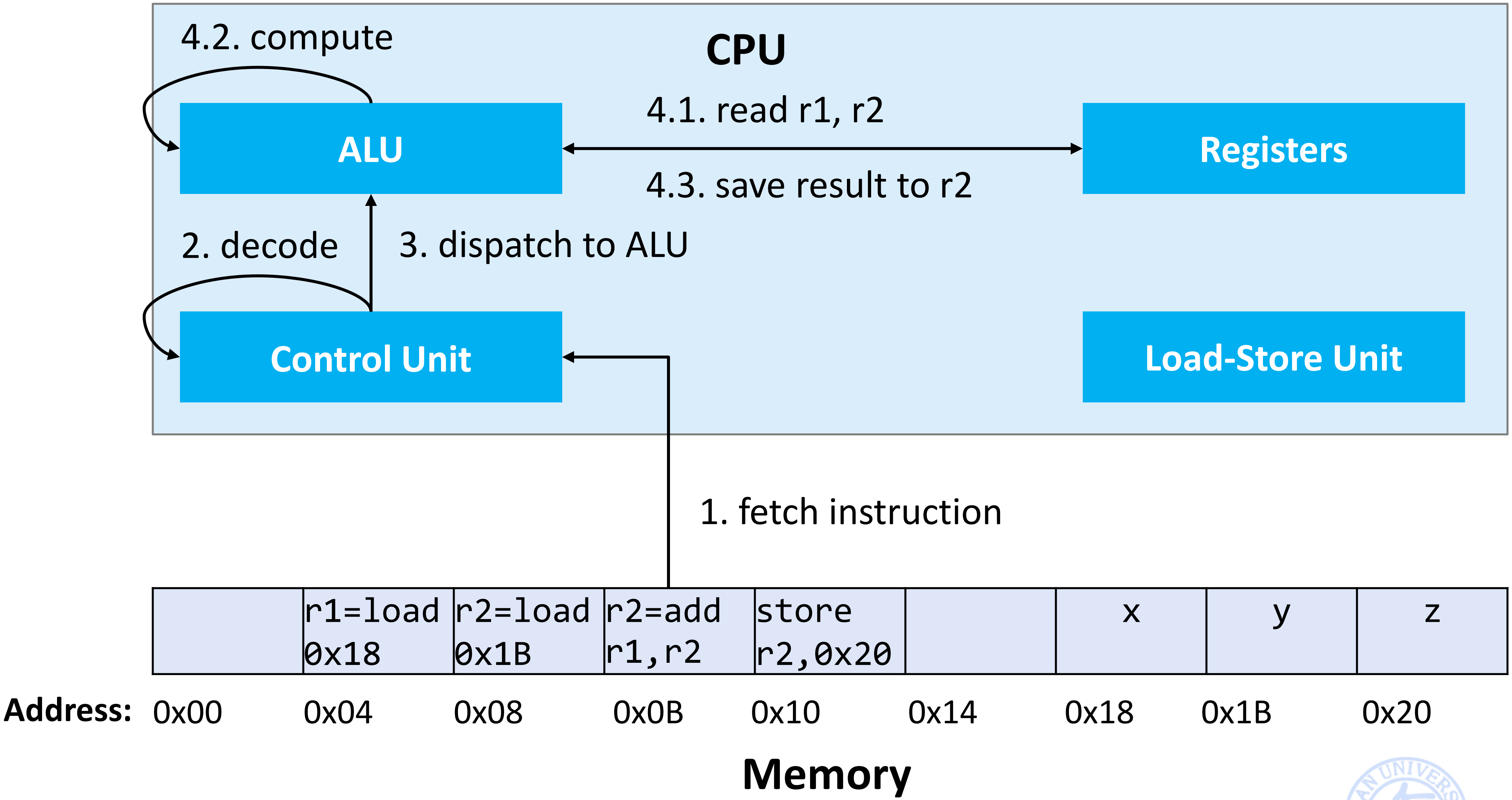
## Step 1: r1 = load x



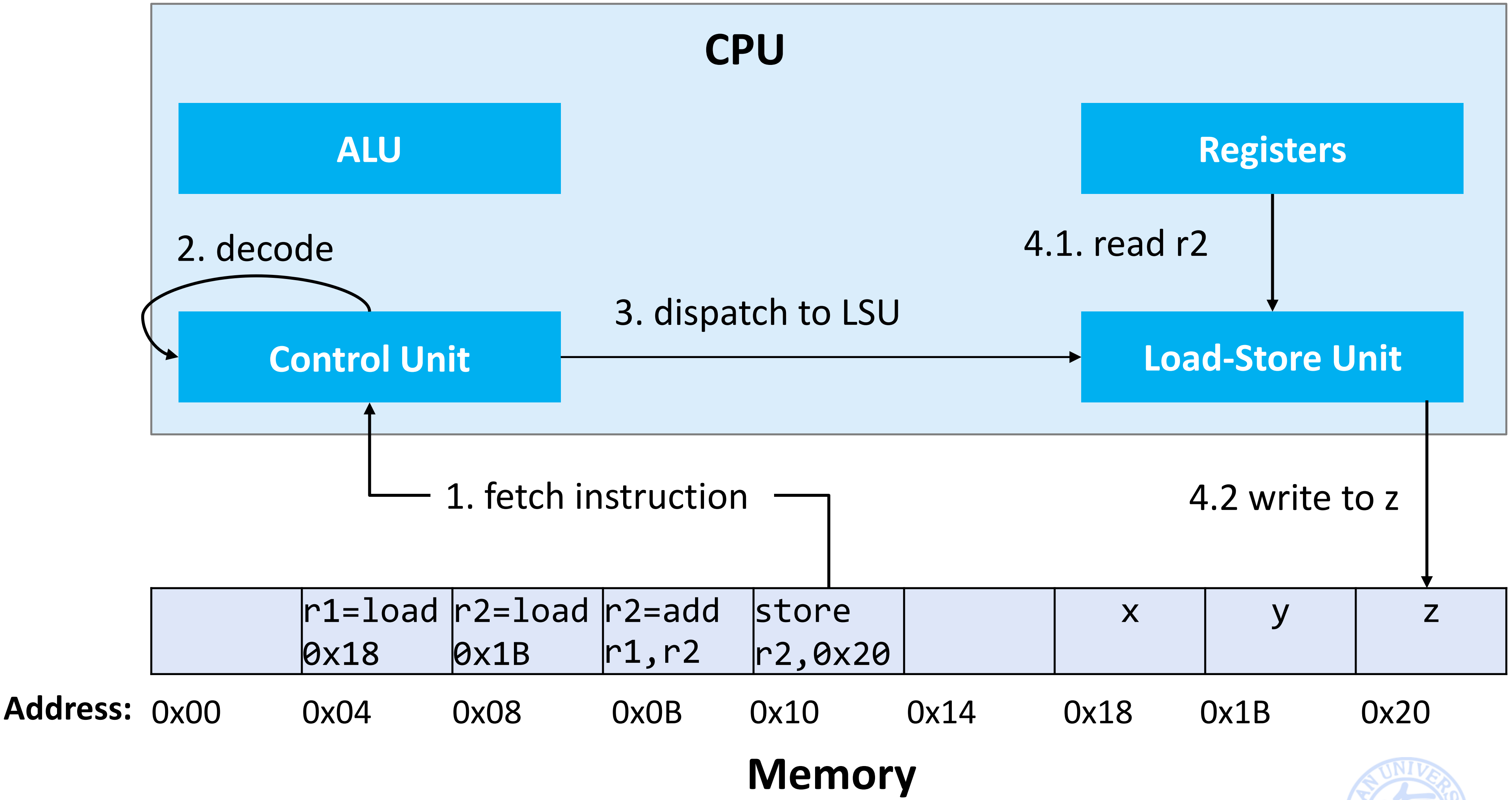
## Step 2: r1 = load y



## Step 3: r2=add r1, r2

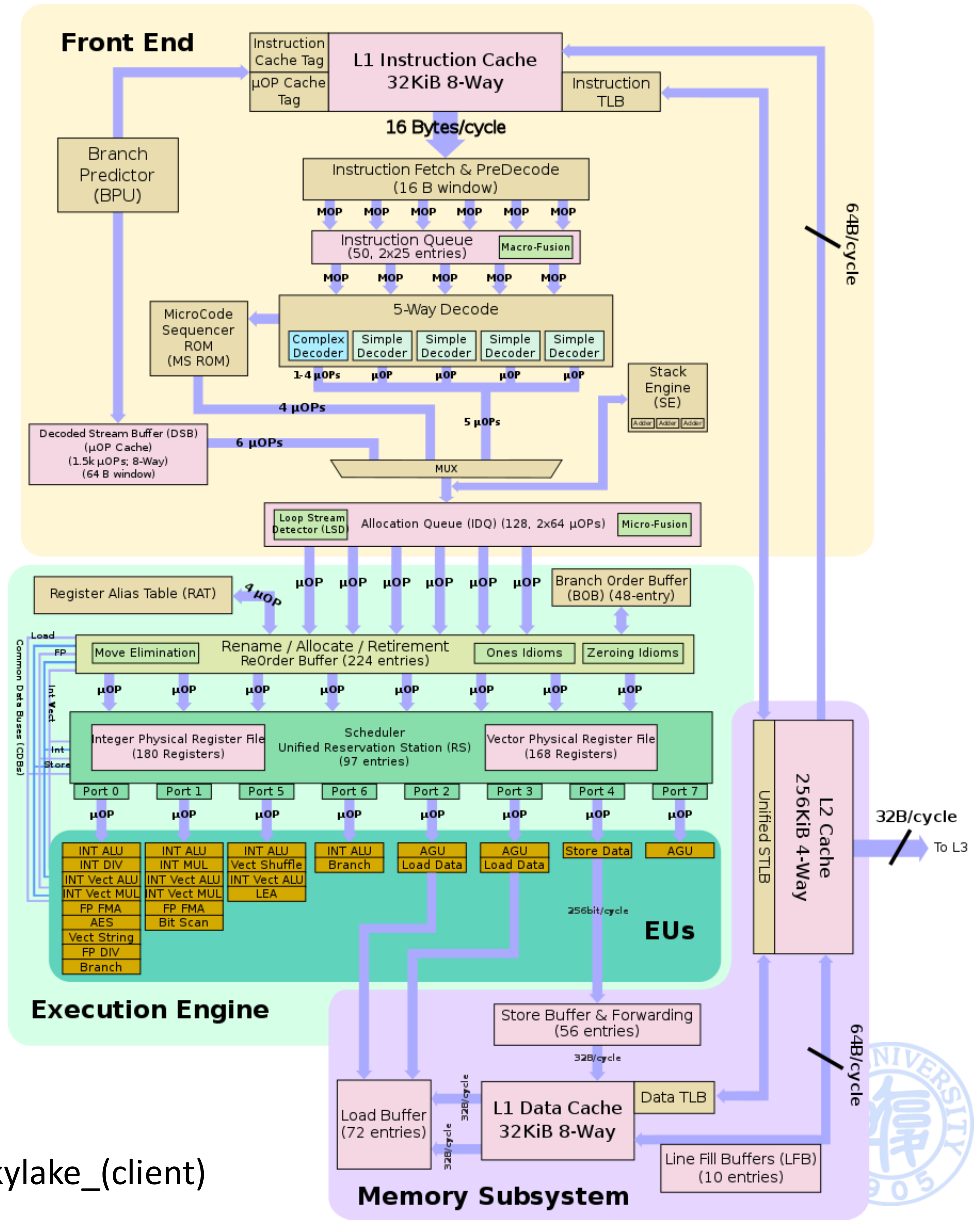


## Step 4: store r2, z



# Sample CPU Core: Skylake

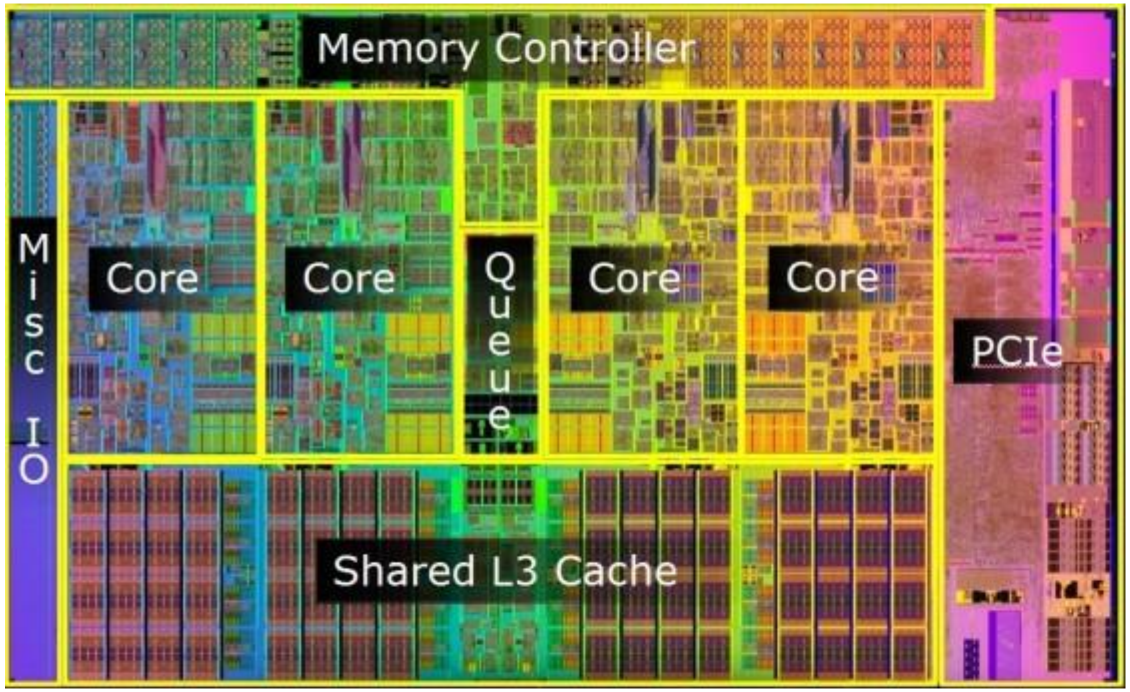
- ❑**Frontend:**
  - ➢Fetch, decode, …
- ❑**Execution engine (backend):**
  - ➢ALU, load-store unit, …
- ❑**Memory subsystem**
  - ➢Cache



Front End
Instruction Cache Tag
µOP Cache Tag
L1 Instruction Cache 32KiB 8-Way
Instruction TLB
16 Bytes/cycle
Branch Predictor (BPU)
Instruction Fetch & PreDecode (16 B window)
MOP MOP MOP MOP MOP MOP
Instruction Queue (50, 2x25 entries)
Macro-Fusion
MOP MOP MOP MOP MOP
MicroCode Sequencer ROM (MS ROM)
5-Way Decode
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
Simple Decoder
1-4 µOPs
µOP µOP µOP µOP
Stack Engine (SE)
Adder Adder Adder
4 µOPs
5 µOPs
6 µOPs
Decoded Stream Buffer (DSB) (µOP Cache) (1.5k µOPs; 8-Way) (64 B window)
MUX
Loop Stream Detector (LSD)
Allocation Queue (IDQ) (128, 2x64 µOPs)
Micro-Fusion
64B/cycle
Register Alias Table (RAT)
4 µOP
µOP µOP µOP µOP µOP µOP
Branch Order Buffer (BOB) (48-entry)
Load
FP
Int Vect
Int
Store
Common Data Buses (CDBs)
Move Elimination
Rename / Allocate / Retirement ReOrder Buffer (224 entries)
Ones Idioms
Zeroing Idioms
µOP µOP µOP µOP µOP µOP µOP µOP
Integer Physical Register File (180 Registers)
Scheduler Unified Reservation Station (RS) (97 entries)
Vector Physical Register File (168 Registers)
Port 0
Port 1
Port 5
Port 6
Port 2
Port 3
Port 4
Port 7
µOP µOP µOP µOP µOP µOP µOP µOP
INT ALU, INT DIV, INT Vect ALU, INT Vect MUL, FP FMA, AES, Vect String, FP DIV, Branch
INT ALU, INT MUL, INT Vect ALU, INT Vect MUL, FP FMA, Bit Scan
INT ALU, Vect Shuffle, INT Vect ALU, LEA
INT ALU, Branch
AGU, Load Data
AGU, Load Data
Store Data
AGU
256bit/cycle
EUs
Execution Engine
Unified STLB
L2 Cache 256KiB 4-Way
32B/cycle
To L3
Store Buffer & Forwarding (56 entries)
32B/cycle
Load Buffer (72 entries)
32B/cycle
32B/cycle
L1 Data Cache 32KiB 8-Way
Data TLB
Line Fill Buffers (LFB) (10 entries)
64B/cycle
Memory Subsystem

https://en.wikichip.org/wiki/intel/microarchitectures/skylake_(client)

## Intel i7 with Four Cores



## Demonstration (PC)

```
#: lscpu
Architecture:                x86_64
  CPU op-mode(s):            32-bit, 64-bit
  Address sizes:             39 bits physical, 48 bits virtual
  Byte Order:                Little Endian
CPU(s):                      8
  On-line CPU(s) list:       0-7
Vendor ID:                   Genuine Intel
  Model name:                Intel(R) Core(TM) i7-9700T CPU @ 2.00GHz
    CPU family:              6
    Model:                   158
    Thread(s) per core:      1
    Core(s) per socket:      8
    Socket(s):               1
    Stepping:                13
    CPU max MHz:             4300.0000
    CPU min MHz:             800.0000
...
```

## Demonstration (Server)

```
#: lscpu
Architecture:            x86_64
  CPU op-mode(s):        32-bit, 64-bit
  Address sizes:         46 bits physical, 48 bits virtual
  Byte Order:            Little Endian
CPU(s):                  80
  On-line CPU(s) list:   0-79
Vendor ID:               GenuineIntel
  Model name:            Intel(R) Xeon(R) Gold 6242R CPU @ 3.10GHz
    CPU family:          6
    Model:               85
    Thread(s) per core:  2
    Core(s) per socket:  20
    Socket(s):           2
    Stepping:            7
    CPU max MHz:         4100.0000
    CPU min MHz:         1200.0000
...
```

## CPU Frequency

- **How many clock cycles the CPU completes per second.**
  - Shorter clock period => Higher CPU frequency
- **Clock ticks synchronize all operations inside the CPU**
  - Clock cycle is determined by the critical path of the CPU’s circuits.
  - Signals travel from one register → combinational logic → next register in one cycle.
  - Smaller transistor size => Faster signal propagation => Shorter critical path.
  - Simple instruction take 1 cycle (*e.g.,* adding two registers).
  - Complex instruction may take several cycles (*e.g.,* memory access, division).

## Question

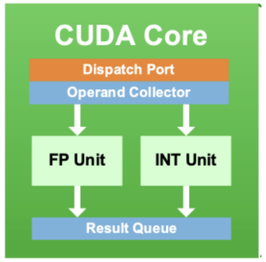
- If a CPU employs 32 bits to represent a memory address, what is the maximum memory space the CPU can support?

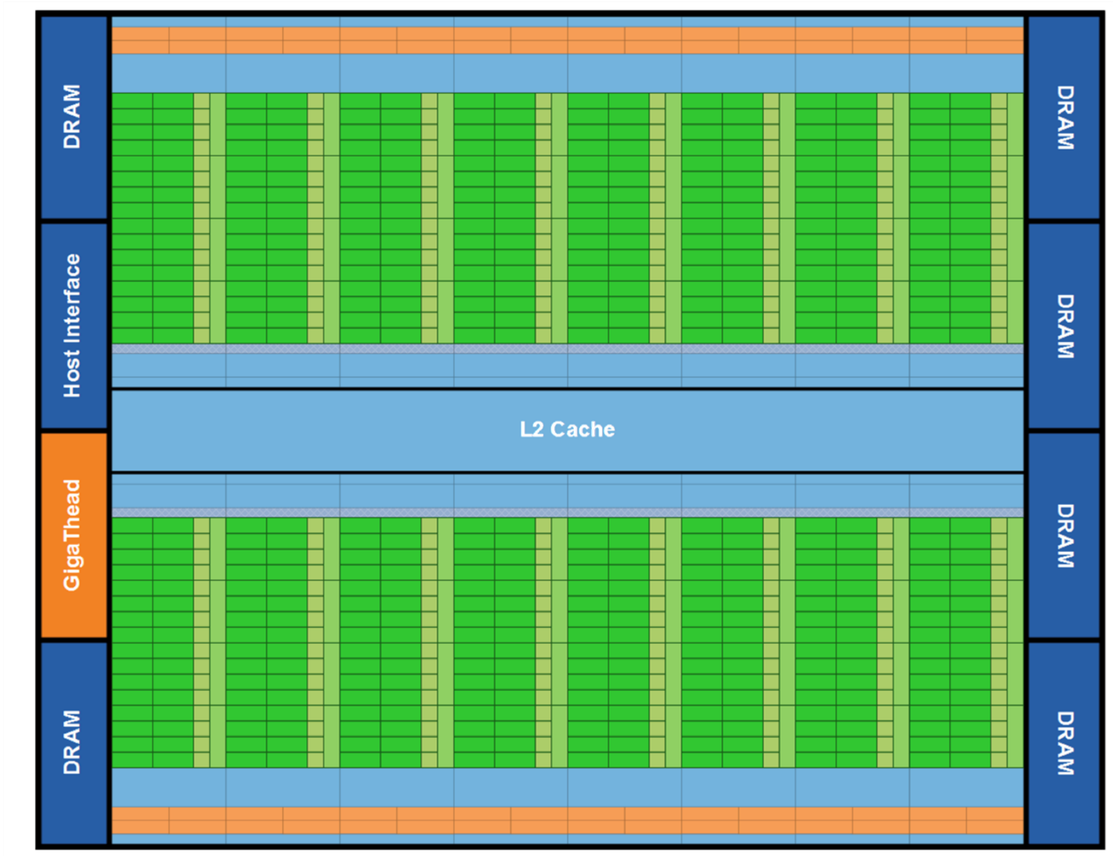
## General-Purpose Computing with Graphics Processing Units

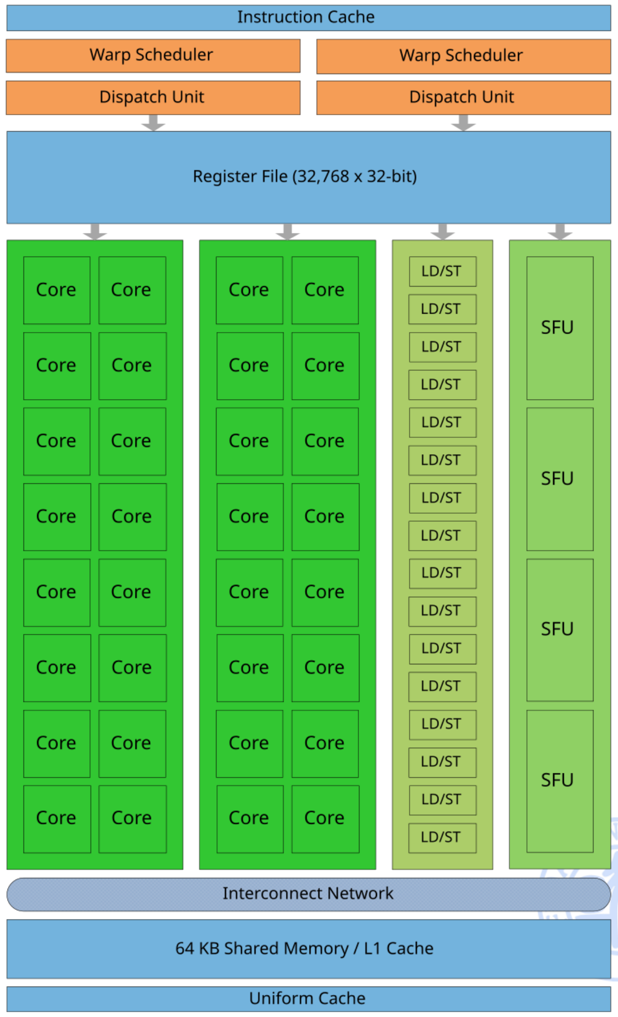
- ❑GPU has thousands of cores and a highly parallel architecture.
- ❑It was originally designed for graphics.
- ❑GPGPU uses GPU to perform computation typically handled by CPU

# Example GPU: Nvidia Fermi

- ❑A GPU has multiple streaming multiprocessors (SMs).
- ❑Each SM consists of 32 cores.







## Parameters of Different NVidia GPUs

| Arch | Product | #. of Cores | | TFLOPS (FP32) | Memory | Bandwidth GB/s | Price |
|---|---|---|---|---|---|---|---|
| | | CUDA | Tensor | | | | |
| Pascal | GTX 1080 | 2560 | - | 8.9 | 8GB GDDR5X | 352 | $599 |
| Pascal | P100 | 3584 | - | 9.3 | 16GB HBM2 | 732 | $5,000+ |
| Turing | RTX 2080 | 2944 | 46 | 10.1 | 8GB GDDR6 | 448 | $699 |
| Ampere | RTX 3080 | 8704 | 328 | 29.8 | 10GB GDDR6X | 912 | $699 |
| Ampere | A100 | 6912 | 432 | 19.5 | 40/80GB HBM2e | 1555 | $11,000 |
| Hopper (PCI-e/SXM) | H100 | 16896 | 528 | 51 | 80GB HBM3 | 2039 | $30,000 |
| Ada Lovelace | RTX 4090 | 16384 | 512 | 82.6 | 24GB GDDR6X | 1008 | $1,599 |
| Blackwell | RTX 5090 | 21760 | 680 | 104.8 | 32 | 1792 | $1,999 |

https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units

## Demonstration

```
#: lspci | grep -i nvidia
01:00.0 VGA compatible controller: NVIDIA Corporation TU102 [GeForce RTX 2080 Ti Rev. A] (rev a1)
01:00.1 Audio device: NVIDIA Corporation TU102 High Definition Audio Controller (rev a1)
01:00.2 USB controller: NVIDIA Corporation TU102 USB 3.1 Host Controller (rev a1)
01:00.3 Serial bus controller [0c80]: NVIDIA Corporation TU102 USB Type-C UCSI Controller (rev a1)
02:00.0 VGA compatible controller: NVIDIA Corporation TU102 [GeForce RTX 2080 Ti Rev. A] (rev a1)
02:00.1 Audio device: NVIDIA Corporation TU102 High Definition Audio Controller (rev a1)
02:00.2 USB controller: NVIDIA Corporation TU102 USB 3.1 Host Controller (rev a1)
02:00.3 Serial bus controller [0c80]: NVIDIA Corporation TU102 USB Type-C UCSI Controller (rev a1)

#: nvidia-smi
Mon Sep 15 11:16:32 2025
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 525.89.02    Driver Version: 525.89.02    CUDA Version: 12.0     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  NVIDIA GeForce ...  Off  | 00000000:01:00.0 Off |                  N/A |
| 27%   44C    P0    58W / 250W |      0MiB / 11264MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
|   1  NVIDIA GeForce ...  Off  | 00000000:02:00.0 Off |                  N/A |
| 32%   45C    P0    31W / 250W |      0MiB / 11264MiB |      0%      Default |
|                               |                      |                  N/A |
+-------------------------------+----------------------+----------------------+
```

## 2. ISA and Assembly Code

## ISA: Instruction Set Architecture

- **ISA is the interface that defines what a processor can do and how software communicates with the hardware.**
- **RISC: Reduced Instruction Set Computer**
  - Separate data load/store and computation into different instructions
  - *e.g.*, AArch/ARM、RISC-V
- **CISC: Complex Instruction Set Computer**
  - An instruction can do both data load/store and computation
  - *e.g.*, X86, X86-64

## ARMv8-A Instructions

- ❑The operands of instructions can be registers, instant numbers, *etc.*
- ❑In the following example:
  - ➢x0..x5 are registers
  - ➢#5, #10 are instant numbers

| Instruction | Semantic |
|---|---|
| `mov x0, #5` | `x0 = 5` |
| `mov x1, #10` | `x1 = 10` |
| `add x2, x0, x1` | `x2 = x0 + x1` |
| `sub  x3, x0, x1` | `x3 = x0 - x1` |
| `mul  x4, x0, x1` | `x4 = x0 * x1` |
| `sdiv x5, x0, x1` | `x5 = x0 / x1` |

## ARMv8-A Instructions: Data Load/Store

- Memory address as an operand:
  - [x] indicates the memory address pointed by x

| Instruction | Semantic |
|---|---|
| `ldr x1, [x0]` | `x1 = [x0]` |
| `ldr x2, [x0, #12]` | `x2 = [x0 + 12]` |
| `add x3, x1, x2` | `x3 = x1 + x2,` |
| `str x3, [x0, #8]` | `[x0 + 8] = x3` |

| | |
|---|---|
| `add x2, x1, [x0]` | Not allowed (CISC operation) |

## ARMv8-A Instructions: Control

- ❑ **Control instructions change the normal flow of execution.**
  - ➢ *e.g.,* jumping to another part of code.

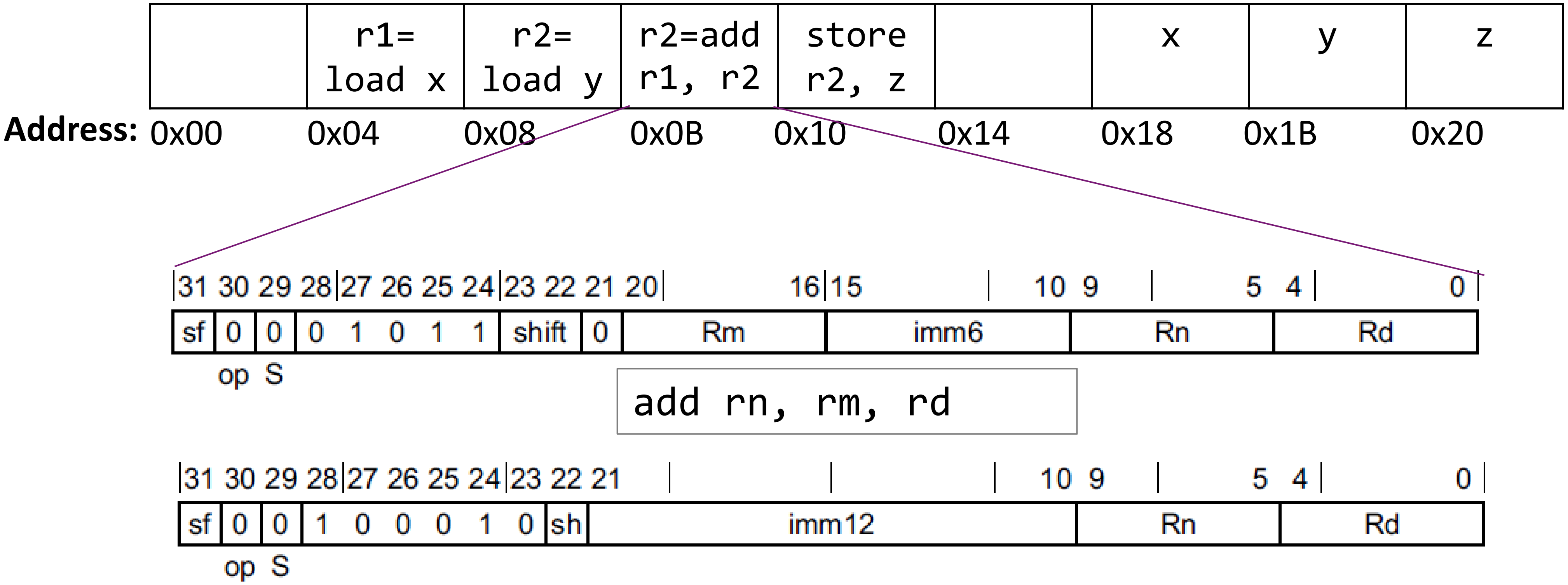
| address | Instructions |
|---|---|
| 0x20 | `cmp      x0, x1` |
| 0x24 | `ble     .addr1` |
| ... | ... |
| 0x104 | `.addr1:` |
| ... | ... |

```
[flag] = compare(x0, x1)
if(flag:<=) PC = 0x104
...
```

PC is a register that always holds the memory address of the next instruction.

## Instructions in Memory

- 32bit fixed-length for ARMv8-A.
- Limitation: Cannot employ some large numbers as operands.



```
add rn, rm, rd
```

```
add rd, rn, imm12
add rd, rn, #4095
add rd, rn, #4097
```

Invalid, because 4097 needs 13 bits.

## Sample Program

- The following code computes the largest Fibonacci number less than 100 and saves the result to the register x0 (aarch64) or eax (x86).
  - 1+1=2, 1+2=3, 2+3=5, 3+5=8, 5+8=13, 8+13=21, 13+21=34, 21+34=55, 34+55=89

```
_main:
    mov x0, #1
    mov x2, #1

_loop:
    mov x1, x0
    mov x0, x2
    add x2, x2, x1
    cmp x2, #100
    ble _loop
    ret
```

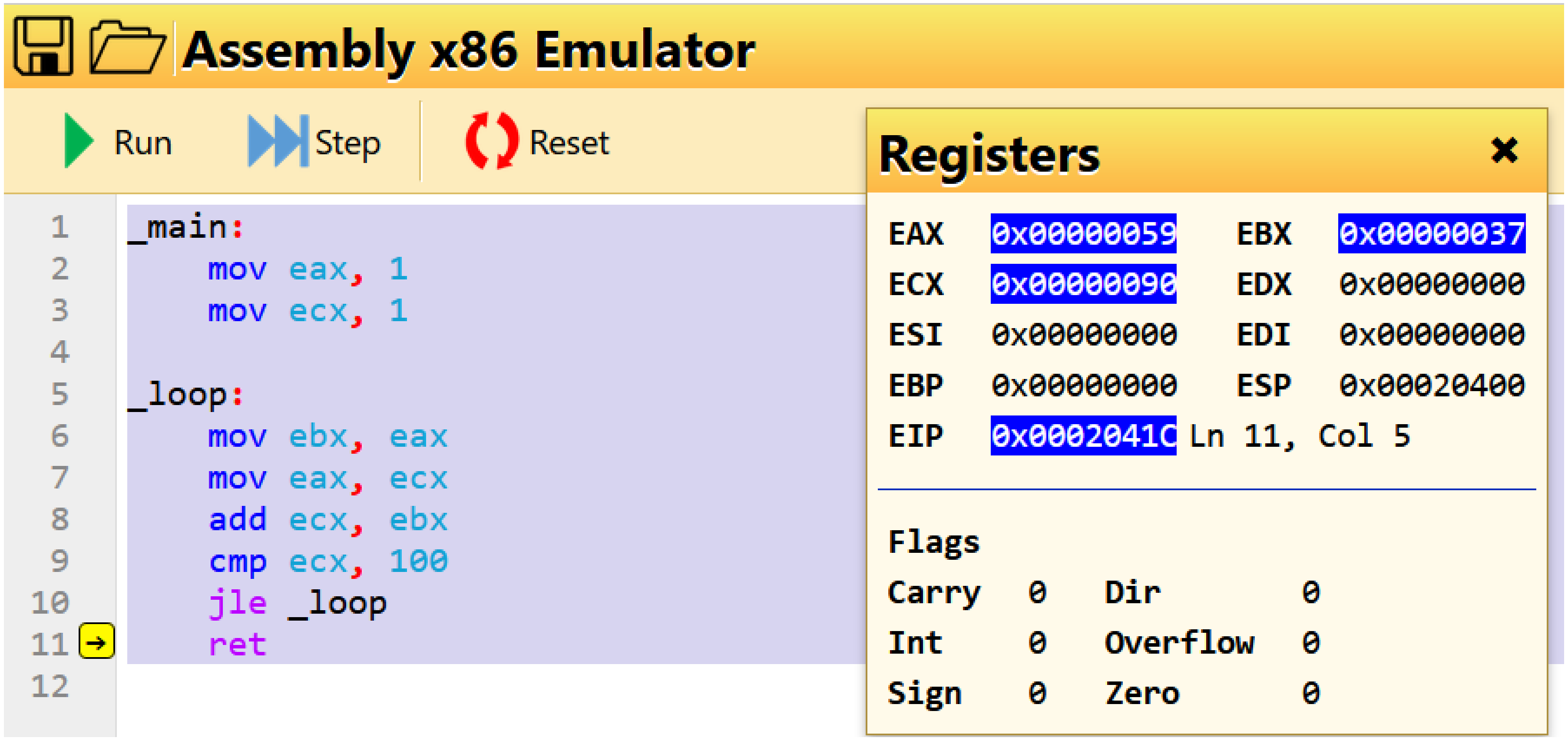
**AArch64 Version**

```
_main:
    mov eax, 1
    mov ecx, 1

_loop:
    mov ebx, eax
    mov eax, ecx
    add ecx, ebx
    cmp ecx, 100
    jle _loop
    ret
```

ecx = ecx + ebx

**X86 Version**

## Examine The Code via an Emulator



```
_main:
    mov eax, 1
    mov ecx, 1

_loop:
    mov ebx, eax
    mov eax, ecx
    add ecx, ebx
    cmp ecx, 100
    jle _loop
    ret
```

```
EAX  0x00000059   EBX  0x00000037
ECX  0x00000090   EDX  0x00000000
ESI  0x00000000   EDI  0x00000000
EBP  0x00000000   ESP  0x00020400
EIP  0x0002041C Ln 11, Col 5

Flags
Carry  0   Dir       0
Int    0   Overflow  0
Sign   0   Zero      0
```

https://carlosrafaelgn.com.br/Asm86/

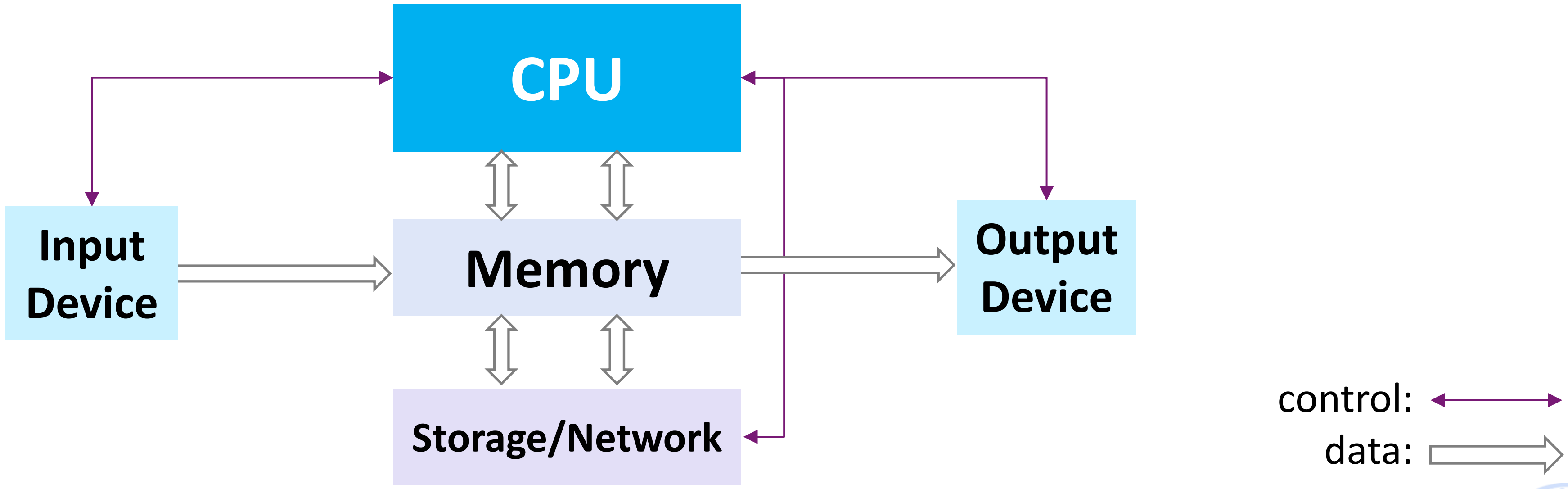
## Exercise

- **Modify the previous code to compute the largest accumulated value less than 100, starting from 1.**
  - result = Max(1+2+3+4+5+…), s.t. result < 100

## 3. I/O and More

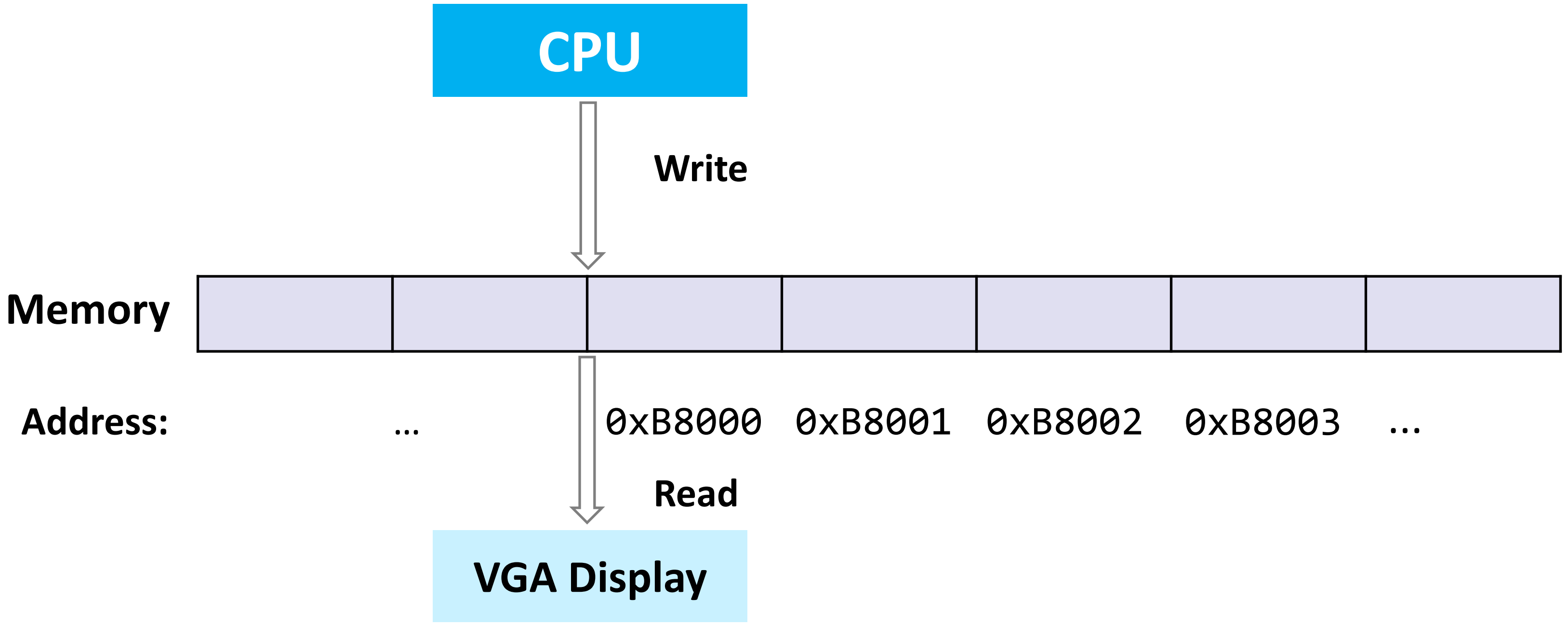
## Computer Architecture

- **Besides CPU and memory, there are peripheral devices**
  - Input device: keyboard, mouse, microphone, *etc.*
  - Output device: display, printer, *etc.*
  - Storage/Network: hard disk, USB flash drive, *etc*.
- **How does the CPU read/write peripheral devices?**



## Memory-Mapped I/O

- ❑Read/write data to I/O devices via memory access.
- ❑Addressing achieved via PCI (peripheral component interconnect).



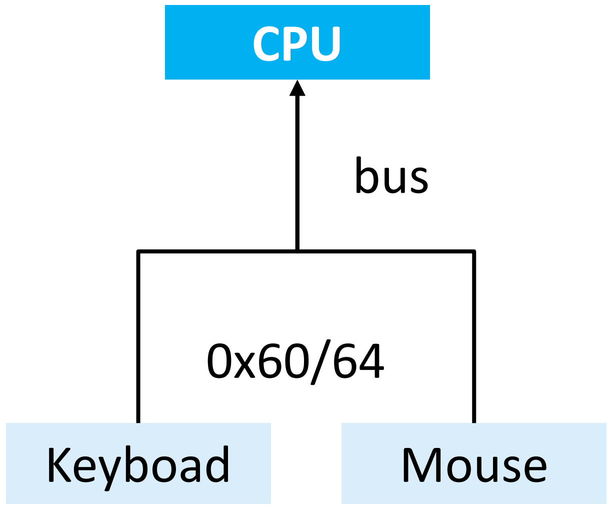
## Output: Video as an Example

- VGA (video graphics array): video display controller 15-pin.
- Mapped to the PC's memory: 0xA0000 - 0xBFFFF.
- 0xB8000 for color text mode (32 KB).

```
mov ax, 0xB8000    ; 0xB8000 is the VGA address
mov es, ax         ; move the content of ax to es
mov di, 0          ; move the value 0 to di register
mov al, 'A'        ; move the value 'A' to al register
mov ah, 0x0F       ; move the value 0x0F (color) to ah
mov [es:di], al    ; move the content of al to memory address:
                     es+di
mov [es:di+1], ah ; move the content of ah to memory address:
                     es+di+1
```

## Alternative Approach: Port-based I/O

- **Read/write data to I/O devices via port.**
- **For example, the PS/2 interface employs two ports:**
  - 0x60 for command
  - 0x64 for data



## Handling Input is More Complicated

- When/how many data arrives is not controlled by the CPU.
  - Polling: let the CPU check particular signals periodically? How frequently?
  - By interrupt (trigger-based) via particular CPU pins

**Reading the keyboard via polling**

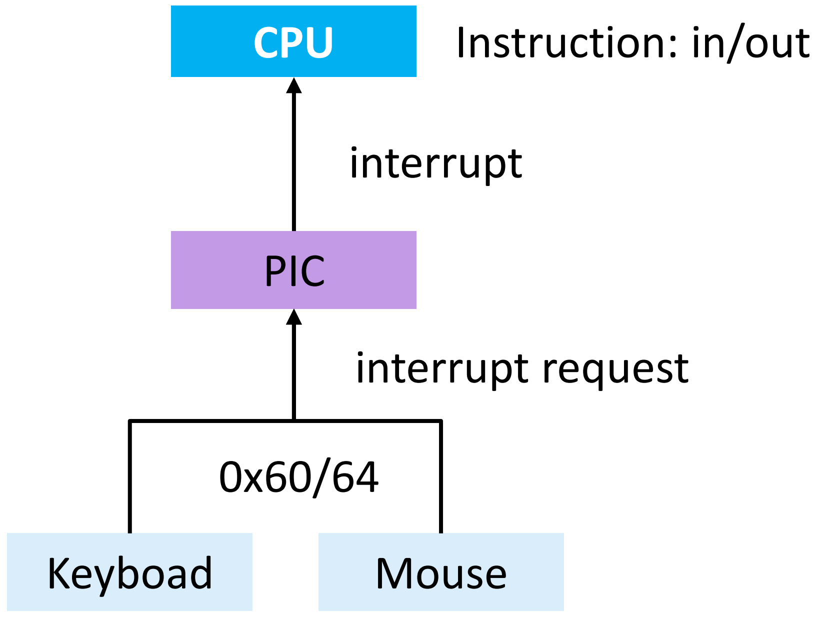
```
wait_for_input:
    in al, 0x64   ; Read the signal from port 0x64 into AL
    test al, 0x01 ; Test whether bit 0 of AL is set (if data is ready)
    jz wait_for_input ; If zero, keep polling until input is available

read_input:
    in al, 0x60   ; Read the character from port 0x60
```

**Reading the keyboard via interrupts**

```
in al, 0x60 ; ; Read the byte from port 0x60
...
out 0x20, 0x20  ; Send End-Of-Interrupt command to port 0x20
```

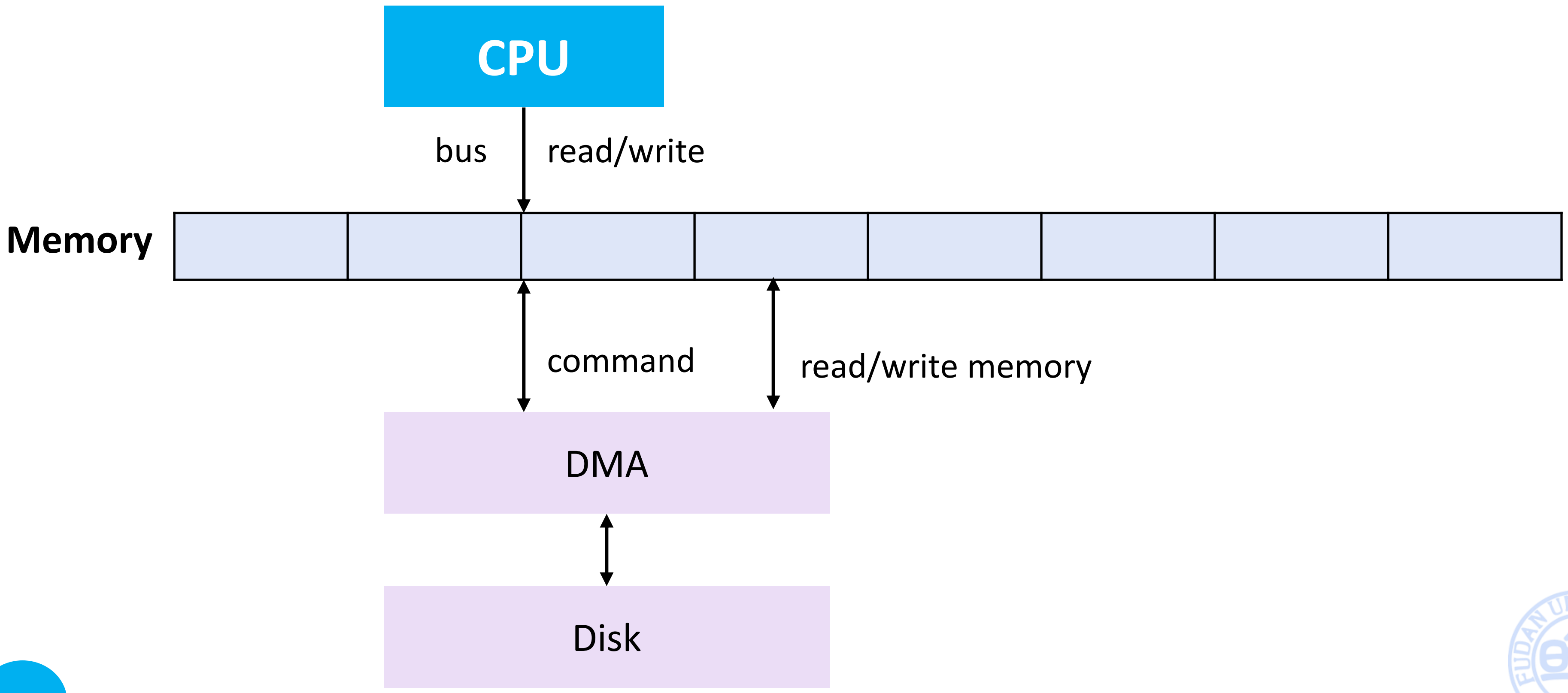
## Input is More Complicated

- **What if data arrives from multiple input devices simultaneously?**
  - First come first serve? May lead to denial-of-service
  - Priority/queuing via PIC (programmable interrupt controller)



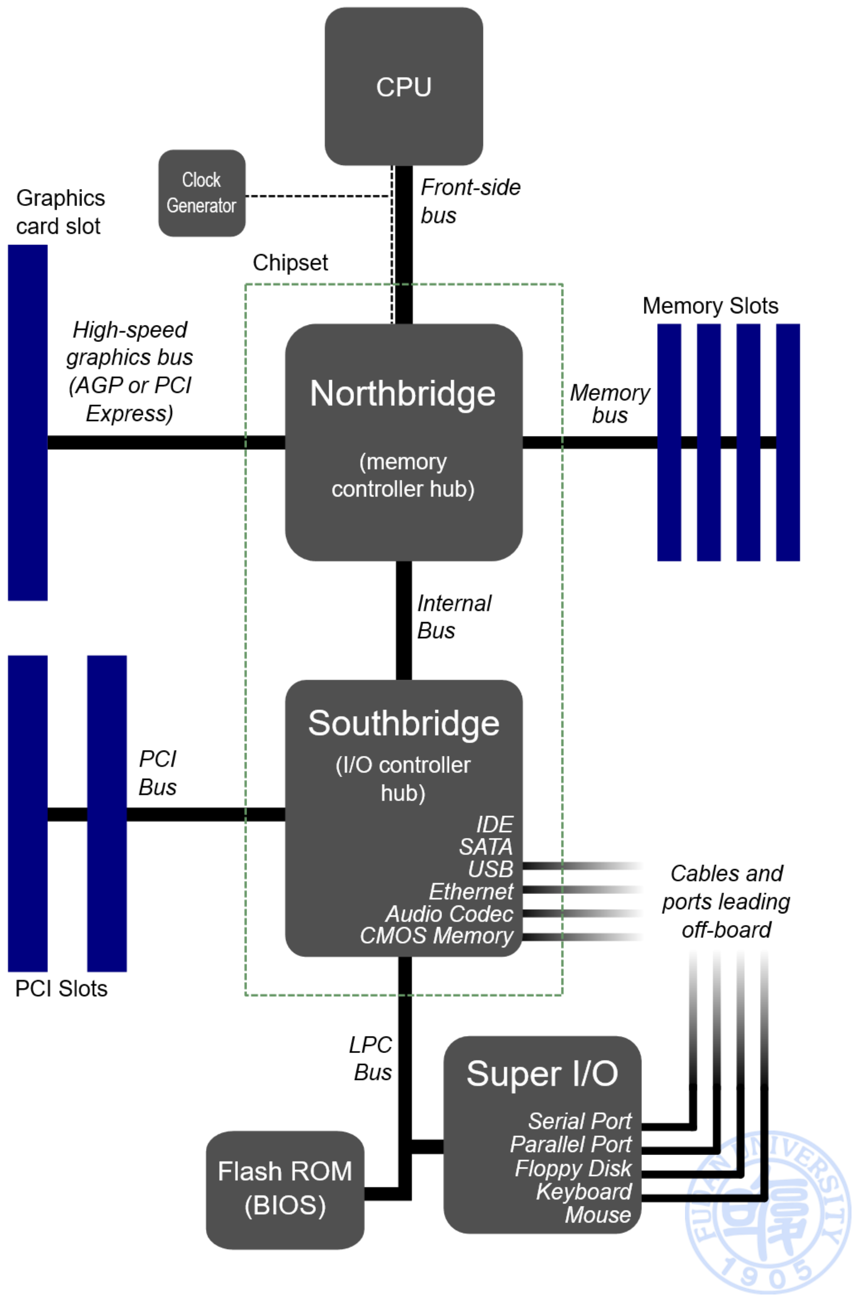
## Disk I/O

- Disk read/write operations are much slower than memory.
- Offload the workload to disk controller via direct memory access.

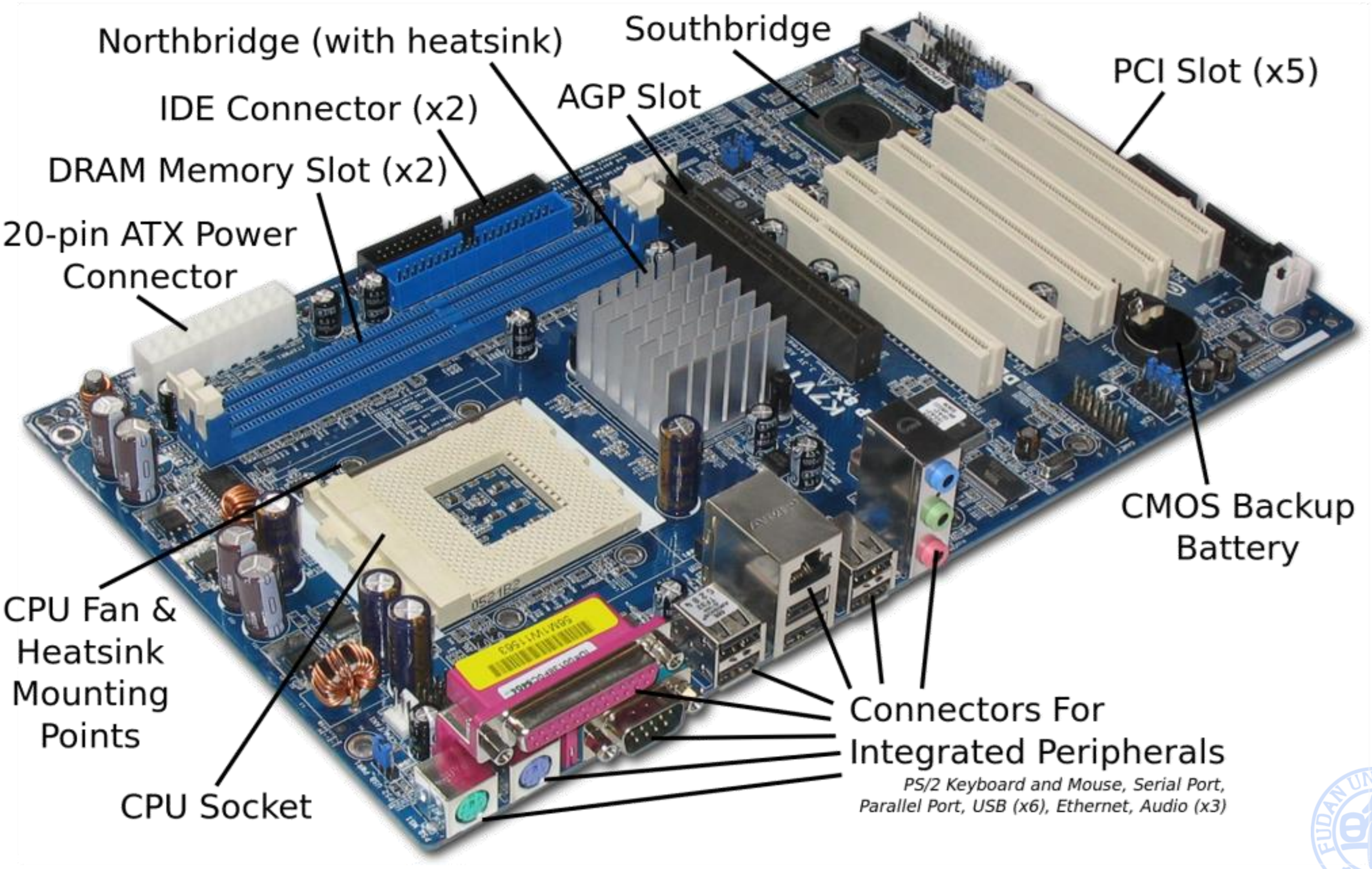


## Motherboard

- ❑Northbridge: handle high-performance tasks, connected directly to a CPU.
- ❑Southbridge: handle low speed tasks, connected via the northbridge



## Motherboard


Northbridge (with heatsink)
Southbridge
PCI Slot (x5)
AGP Slot
IDE Connector (x2)
DRAM Memory Slot (x2)
20-pin ATX Power Connector
CMOS Backup Battery
CPU Fan & Heatsink Mounting Points
CPU Socket
Connectors For Integrated Peripherals
PS/2 Keyboard and Mouse, Serial Port, Parallel Port, USB (x6), Ethernet, Audio (x3)

## Demonstration

```
#: lspci
00:00.0 Host bridge: Intel Corporation 8th/9th Gen Core 8-core Desktop Processor Host Bridge/DRAM
Registers [Coffee Lake S] (rev 0d)
00:02.0 VGA compatible controller: Intel Corporation CoffeeLake-S GT2 [UHD Graphics 630] (rev 02)
00:08.0 System peripheral: Intel Corporation Xeon E3-1200 v5/v6 / E3-1500 v5 / 6th/7th/8th Gen Core
Processor Gaussian Mixture Model
00:12.0 Signal processing controller: Intel Corporation Cannon Lake PCH Thermal Controller (rev 10)
00:14.0 USB controller: Intel Corporation Cannon Lake PCH USB 3.1 xHCI Host Controller (rev 10)
00:14.2 RAM memory: Intel Corporation Cannon Lake PCH Shared SRAM (rev 10)
00:14.3 Network controller: Intel Corporation Cannon Lake PCH CNVi WiFi (rev 10)
00:15.0 Serial bus controller: Intel Corporation Cannon Lake PCH Serial IO I2C Controller #0 (rev 10)
00:16.0 Communication controller: Intel Corporation Cannon Lake PCH HECI Controller (rev 10)
00:17.0 SATA controller: Intel Corporation Cannon Lake PCH SATA AHCI Controller (rev 10)
00:1b.0 PCI bridge: Intel Corporation Cannon Lake PCH PCI Express Root Port #17 (rev f0)
00:1f.0 ISA bridge: Intel Corporation Q370 Chipset LPC/eSPI Controller (rev 10)
00:1f.3 Audio device: Intel Corporation Cannon Lake PCH cAVS (rev 10)
00:1f.4 SMBus: Intel Corporation Cannon Lake PCH SMBus Controller (rev 10)
00:1f.5 Serial bus controller: Intel Corporation Cannon Lake PCH SPI Controller (rev 10)
00:1f.6 Ethernet controller: Intel Corporation Ethernet Connection (7) I219-LM (rev 10)
01:00.0 Non-Volatile memory controller: Phison Electronics Corporation PS5013 E13 NVMe Controller (rev 01)
```

## Demonstration

```
#: sudo lspci -vv
00:00.0 Host bridge: Intel Corporation 8th/9th Gen Core 8-core Desktop Processor Host Bridge/DRAM Registers
[Coffee Lake S] (rev 0d)
        DeviceName: Onboard - Other
        Subsystem: Dell 8th/9th Gen Core 8-core Desktop Processor Host Bridge/DRAM Registers [Coffee Lake S]
        Control: I/O- Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR+ FastB2B- DisINTx-
        Status: Cap+ 66MHz- UDF- FastB2B+ ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort+ >SERR- <PERR- INTx-
        Latency: 0
        Capabilities: [e0] Vendor Specific Information: Len=10 <?>
        Kernel driver in use: skl_uncore
        Kernel modules: ie31200_edac

00:02.0 VGA compatible controller: Intel Corporation CoffeeLake-S GT2 [UHD Graphics 630] (rev 02) (prog-if 00 [VGA
controller])
        DeviceName: Onboard - Video
        Subsystem: Dell CoffeeLake-S GT2 [UHD Graphics 630]
        Control: I/O+ Mem+ BusMaster+ SpecCycle- MemWINV- VGASnoop- ParErr- Stepping- SERR- FastB2B- DisINTx+
        Status: Cap+ 66MHz- UDF- FastB2B- ParErr- DEVSEL=fast >TAbort- <TAbort- <MAbort- >SERR- <PERR- INTx-
        Latency: 0, Cache Line Size: 64 bytes
        Interrupt: pin A routed to IRQ 142
        Region 0: Memory at 90000000 (64-bit, non-prefetchable) [size=16M]
        Region 2: Memory at 80000000 (64-bit, prefetchable) [size=256M]
        Region 4: I/O ports at 3000 [size=64]
        Expansion ROM at 000c0000 [virtual] [disabled] [size=128K]
        Capabilities: [40] Vendor Specific Information: Len=0c <?>
...
```

## Several Concepts

- **Hardware: Physical components of a computer system.**
- **Firmware: Specialized software embedded in hardware devices to control or manage the hardware directly.**
  - *e.g.,* BIOS (basic input/output system) for interrupt handling.
- **Software: Programs that run on the hardware.**
  - *e.g.,* operating systems and applications.

## Role of BIOS / UEFI

- ❑BIOS/UEFI runs before the OS, during system boot.
- ❑Assigns and initializes port addresses for some devices.
- ❑Scans devices (PCI, ACPI, etc.) and assigns MMIO ranges and writes assigned addresses into device registers.

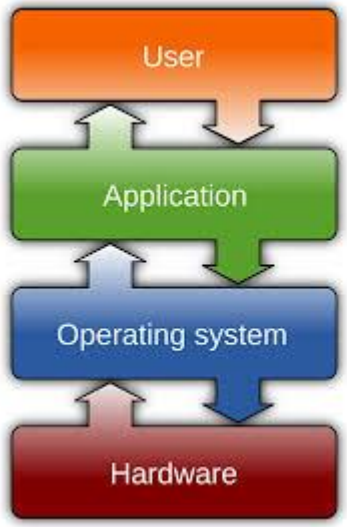
## Role of the Operating System

❑**After BIOS/UEFI initializes hardware, the OS takes over.**

➢Talking to the hardware directly is difficult, error-prone, and hardware-specific.

❑**OS manages hardware resources and provides services for apps.**

➢Applications cannot talk to the hardware directly, ensuring safe device operations.



## Summary

- A CPU consists of a control unit, ALU, LSU, and registers.
- Each CPU implements an ISA, programmed using assembly instructions.
- CPU communicates with peripheral devices via I/O ports, memory-mapped I/O, and interrupts.
- OS manages hardware and simplifies the usage of computer.