Operating Systems

Karim Lounis

Queen’s University, Canada

Winter’19, Jan-Apr 2019
Types of Operating systems (Terminology):

**Multiprogramming systems:** are systems that have multiple programs loaded into the memory at a time.

**Single-tasking systems:** are systems that have only one program loaded into the memory at a time e.g., MS-DOS.

**Time-sharing systems:** A.k.a., multitasking systems, r multiprogramming systems that switch between processes when their time slice is up e.g., modern OSs; Linux, Mac OS X, and Windows.

**Embedded OS:** designed for embedded devices (e.g., TinyOS, LiteOS).

**Real time OS:** designed for critical industrial facilities (e.g., IBM RTOS/360, PikeOS).

**Distributed OS:** designed for distributed infrastructures (e.g., Inferno, Plan 9 by Bell Labs).

**Network OS:** designed for networks administration and management (e.g., Windows 2018 server).
Overview:

Hardware and software in computer systems.
Memory unit.
CPU unit.
Single and multiprocessors systems.
Instruction execution cycle.
Computer Systems (Superficial view)

A computer system is mainly composed of four components:
Hardware Components
Hardware Components

The main hardware components in this course:
Hardware Components
Software Components
Software Components

The main software components in this course:
Von Neumann Architectural Model

Computer’s hardware can be represented by the Von Neumann architectural model:

![Diagram of the Von Neumann architectural model](image)
Hardware Components

Memory Unit
Memories

Memories are hardware components used to store data.

They can be volatile or non-volatile.
Memory Classification

Memories can be classified following their storage capacity, speed (access time), and cost.
Memory Classification (central memory)

RAM (Random Access Memory): Needs power to keep its content.
- **Static RAM**: or SRAM, uses latches components (flip-flops) to store data. Has smaller storage capacity but runs faster.
  - + used in internal CPU memory +
- **Dynamic RAM**: or DRAM, uses capacitors to store data. Needs refreshment. Has larger storage-capacity, but runs slower.
  - + used in central memory +

ROM (Read Only Memory): Does not need power to keep its content.
- **Programmable**: Can be erased and reprogrammed multiple times e.g., EPROM and EEPROM.
  - + used in memory-sticks and pendrives +
- **Non-Programmable**: Cannot be programmed twice e.g., ROM and PROM (only once, after manufacturing).
  - + used to store firmware +
Old-fasion central memory management

Each program is allocated a contiguous partition in the main memory:

```c
int a, b, c;
void main()
{
    int x, y, z;
    int *p;
    p = (int*) malloc(sizeof(int));
    *p = 10;
}
```
Hardware Components

Central Processing Unit
Processors

Processor or Microprocessor, is complex electronic circuit designed to execute machine instructions at a very high speed.

Processors are featured by their: frequency (in MHz or GHz), internal memory (cache & registers), word size (8-64 bit), and number of cores.
Processors

In Linux you can see the computer’s CPU features by executing the command `lscpu`:

```
root@Human-Device:~# lscpu
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 4
On-line CPU(s) list: 0-3
Thread(s) per core: 2
Core(s) per socket: 2
Socket(s): 1
NUMA node(s): 1
Vendor ID: GenuineIntel
CPU family: 6
Model: 42
Model name: Intel(R) Core(TM) i5-2410M CPU @ 2.30GHz
Stepping: 7
CPU MHz: 798.231
CPU max MHz: 2900.0000
CPU min MHz: 800.0000
BogoMIPS: 4589.81
Virtualization: VT-x
L1d cache: 32K
L1i cache: 32K
L2 cache: 256K
L3 cache: 3072K
NUMA node0 CPU(s): 0-3
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx rdtscp lm constant_tsc arch_perfmon pebs bts rep_good nopl xtopology nonstop_tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic popcnt tsc_deadline_timer aes xsave avx lahf_lm epb pti ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid xsaveopt dtherm ida arat pln pts flush_l1d
```
In Linux you can see the computer’s CPU features by executing the command `lscpu:`

```
Architecture: x86_64
CPU op-mode(s): 32-bit, 64-bit
Byte Order: Little Endian
CPU(s): 8
On-line CPU(s) list: 0-7
Thread(s) per core: 2
Core(s) per socket: 4
Socket(s): 1
NUMA node(s): 1
Vendor ID: AuthenticAMD
CPU family: 21
Model: 2
Model name: AMD FX(tm)-8350 Eight-Core Processor
Stepping: 0
CPU MHz: 1400.000
CPU max MHz: 4000.0000
CPU min MHz: 1400.0000
BogoMIPS: 8000.05
Virtualization: AMD-V
L1d cache: 16K
L1i cache: 64K
L2 cache: 2048K
L3 cache: 8192K
NUMA node0 CPU(s): 0-7
Flags: fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca cmov pat pse36 clflush mmx fxsr sse sse2 ht syscall nx mmxext fxsr_opt pdpe1gb rdtscp lm constant_tsc rep_good nopl nonstop_tsc extd_apicid aperfmperf pni pclmulqdq monitor ssse3 fma cx16 sse4_1 sse4_2 popcnt aes xsave avx f16c lahf_lm cmp_legacy svm extapic cr8_legacy abm sse4a misalignsse 3dnowprefetch osvw lfs vmptrld lwp fma4 tsc涉及 intel core perfctr_nb cpb hw_pst ate sbsd vmmcall bmi1 arat npt lbv svm_lock nrip_save tsc_scale vmcb_clean flus.hbyastd decodeassist pausefilter pfthreshold
```
Processors Registers

Definition

**Register:** It is a very high speed one-word memory inside the CPU, generally made of electronic latches.

The CPU holds multiple registers for different purposes:

- **General purpose registers:** Used for unary and binary operations e.g., the Accumulator AX, Base register BX, Counting register CX, and Data register DX, or sometimes R0,…,R15.

- **Index registers:** Used for addressing e.g., MAR (Memory Address Register), BP (Base Pointer), SP (Stack Pointer), IP (Instruction Pointer) a.k.a., PC (Program Counter), CIR (Current Instruction Register) or IR, and ISR SP (Interrupt handler Stack pointer).
Processors Registers

- **Segment registers**: Used for determining the start and the end of a segment e.g., CS (Code Segment), DS (Data Segment), SS (Stack Segment), and ES (Extended Segment).

- **State register**: Used for storing the state of the current instruction e.g., Carry flag, parity flag, sign flag, ..., overflow, interruption flag.

- **Other register**: Such as the MDR (Memory Data Register) or MBR (Memory Buffer Register) used to transfer data between the CPU and the memory.
Processor Instruction Set

**Processor instruction set**: is the set of machine instructions that a given processor is able to decode and execute (supported instructions).

There exist many: x86, ARM, VAX, MIPS, ..., Alpha. In this course, most of the examples given in assembly language use the x86 instructions set:

- `Mov Ax, 0x1B25` //Moves the value 0x1b25 to the register AX.
- `Mov Al, 0x25` //Moves the value 0x25 to lower part of AX.
- `Inc Ax` //Increments the value in the register Ax by 1.
- `Dec Ax` //Decrements the value in the register Ax by 1.
- `Push Ax` //pushes to content of Ax into the stack (SP-2).
- `Pop Ax` //pops the content pointed at [SP] in the stack into Ax register.
Processor Instruction Set

Jmp T  //Jumps to the instruction labeled with T.

Cmp Ax, 1  //Compares the value stored in the Ax register with 1.

Je T  //Jumps to T if the previously compared values were equal.

Jle T  //Jumps if the 1st operand is ≤ to the 2nd.

Loop T  //Decrements Cx and jumps to T if Cx≠ 0.

Xor Ax, Bx  //Xor of Ax and Bx and stores result in Ax.

Halt  //Terminates program.
Processor Instruction Set

Definition

**Machine code:** is a sequence of bytes 01010101 01010111...01010100 that can be interpreted and executed by a dedicated CPU.

Each instruction in the assembly language is transformed into a machine code by the assembler program (e.g., Netwide Assembler a.k.a., nasm).

Each instruction in machine code is generally composed of an **Operation code** (opcode) and **operand**.

Mov Ax, [0xF565] | JMP Label

Some instructions just consists of an **operation code**:

Nop | Inc Ax | Add Ax, Bx | Mov Ax, Bx

Instructions may have difference size e.g., in Intel, 1-Byte to 14-Bytes.
Consider the following x86-assembly program:

\[
\begin{align*}
&\text{Mov Ax, 0X0001} \\
&\text{Mov Cx, 0X0001} \\
&\text{T Mul Ax, Cx} \\
&\text{Inc Cx} \\
&\text{Cmp Cx, 0X0005} \\
&\text{Jle T} \\
&\text{Halt}
\end{align*}
\]

At the end of this program the register Ax contains the value 0x0078.
Single-Processor Systems

In a single-processor system, there is only one CPU (Central Process Unit) capable of executing a general-purpose instruction set.

Such systems do have other special-purpose processors such as device-specific processor (or controllers), or a general-purpose processor on mainframes.

A system with $n$ special-purpose processors and one general-purpose processor is a single-processor system.
Multiprocessor Systems

In a multiprocessor system, there are two or more CPUs capable of executing a **general-purpose** instruction set, and sharing the computer bus, [clock], memory, and peripheral devices.

![Motherboard of a multiprocessor computer](image)

**Figure 2:** Motherboard of a multiprocessor computer

- Appeared in servers then migrated to desktops, laptops, and recently smartphones and tablets.
- Cheaper than multiple single-core systems and provide better throughput, response time, turn around time, and higher reliability.
Multiprocessor Systems

There exist two types of multiprocessor systems:

1. **AMP (Asymmetric MultiProcessing):** In which processors are not treated in the same way by the OS e.g., one CPU executes OS code only, or is attached to certain devices only.

2. **SMP (Symmetric MultiProcessing):** Hardware architecture where two or more identical processors are connected to a single, shared main memory, have full access to all input and output devices, and are controlled by a single operating system instance that treats all processors equally, reserving none for special purposes.

Nowadays, operating systems such as Windows, Mac OS X, and Linux provides support for SMP architectures.

Modern CPU design: Include multiple computing cores on a single chip, they are called multicore e.g., Intel Core i3, i5, and i9.
Multiprocessor Systems

Multiprocessor systems can also be classified into:

1. **Tightly coupled multiprocessor:** The set of CPUs share one main memory (synchronization is provided) e.g., Intel Core i3, i5, and i9.
   - Use synchronization mechanism such as semaphores and monitors.

2. **Loosely coupled multiprocessor:** Each CPU has its own memory. Generally, CPUs are connected via a Network e.g., Clusters and Distributed systems.
   - Use message passing mechanism.
Multiprocessor Systems

**Clustered Systems:** Are composed of several computer systems (a.k.a., hosts) connected via a local area network. Each of the computer systems can be a single-processor system or a multicore system.

- Clusters emerged: low-cost microprocessors, high-speed networks, and software for high-performance distributed computing.
- Clusters are used to share storage and provide high availability.
- Can provide HPC (High Performance Computing) environments.
Instructions Execution

Instructions Execution
Some Initial Assumptions

Let us consider the following simplifying assumptions:

1. We work on a uniprocessor computer.
2. No interaction between processes i.e., if \( n \) processes are executing, no process should affect the execution (behavior) of the other ones.
3. The OS adopts the old-fashioned memory management partitioning:
Instruction Execution Cycle

The processing of each program instruction goes through three main phases: **FETCH, DECODE, & EXECUTE**

1. **FETCH:** Getting program’s next instruction from memory.
   - 1.1. Get instruction address from IP, place it in the MAR and send a read-request to RAM to retrieve the content of [MAR].
   - 1.2. After access time, the content is placed on the MBR.
   - 1.3. Store the instruction code in CIR.

2. **DECODE:** Instruction decoding and operands fetching.
   - 2.1. The CU decodes and transforms the instruction into a sequence of elementary operations.
   - 2.2. If the instruction requires operands from memory, the CPU gets them in the MBR after issuing a fetch operand operation.
   - 2.3. The operand is stored in one of the general purpose register and the IP is updated.
**Instruction Execution Cycle**

**EXECUTE:** Instruction execution

3.1. The ALU executes the instruction.

3.2. The state register is updated.

Instructions could be:

- **Data transfer:** from and to memory or between registers.
  
  e.g., `Mov Ax, [0x52F3]` or `Mov Ax, Bx`

- **Arithmetic operation:** Addition, subtraction, division, and multiplication.
  
  e.g., `Div Ax, Cx` or `Add Ax, Bx`

- **Logical operation:** AND, OR, NOT, and comparison.
  
  e.g., `Cmp Ax, 0x0001` or `XOR Ax, Ax`

- **Sequence control:** Branch and tests.
  
  e.g., `Je X` or `jmp X`
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)
...
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax
...

Program written in C programming language
...
C = A + B;
...

RAM
0xFFFFF
0xFFFFE
0xFFFFD
...
0xA524
0xA523
0x002C
...
0xA522
0xA521
0xA520
...
0x5103
0x5102
0x5101
...
0x5100
0x0002
0x0000

CPU

MBR

Ax
Cx
Bx
Dx
Sp
Bp

Arithmetic and Logic Unit

Control Unit

Decoder

Control Bus

Data Bus 16-bit

Address Bus 16-bit
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086):
...
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax
...

Program written in C programming language:
...
C = A + B;
...

Karim Lounis (Queen's University, Canada)
Example (Instructions Execution Cycle)

```
... Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax ...
```

Program written in C programming language
```
..., 
C = A + B;
..., 
```
Example (Instructions Execution Cycle)

**Assembly Language (Intel 8086)**

...  
Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...

**Program written in C programming language**

```c
...;  
C = A + B;  
...;
```

**CPU**

- AX
- BX
- SP
- CIR
- CX
- DX
- BP

**MBR**

`Mov AX, [0xA522]`

**Control Unit**

**Data Bus 16-bit**

**Address Bus 16-bit**

**RAM**

- 0xFFFF
- 0xFFFF
- ...
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

... Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax ...

Program written in C programming language

...;
C = A + B;
...

RAM

0x0000 0x0001 0x0002 0xEF
0xFE 0xFF

CPU

MBR Mov AX, [0xA522]
Mov AX, [0xA522]

Mov AX, [0xA522]

CIR

Decoder

Control Bus

Data Bus 16-bit

Address Bus 16-bit

IP 0x5100
MAR 0x5100

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)
...
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax
...

Program written in C programming language
....
C = A + B;
....

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

...  
Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...

Program written in C programming language

...;
  C = A + B;
  ...

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Program written in C programming language

As an example:

```
...;
C = A + B;
...;
```

Instruction cycles:
1. Fetch instruction
2. Decode instruction
3. Execute instruction
4. Write back result

Assembly Language (Intel 8086)

```
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax
...;
```

CPU
- AX
- BX
- SP
- CX
- DX
- IP
- MAR
- MBR
- Ctrl
- Decoder
- ALU

Data Bus 16-bit
Address Bus 16-bit
Control Bus

RAM

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)
...
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax...

Program written in C programming language
...;
C = A + B;
...;

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

...  
Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...

Program written in C programming language

...;  
C = A + B;  
...;

RAM

<table>
<thead>
<tr>
<th>Address</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0xFFFF</td>
<td>0xB88</td>
</tr>
<tr>
<td>0xFEBF</td>
<td>0xA524</td>
</tr>
<tr>
<td>0xB100</td>
<td>0xA523</td>
</tr>
<tr>
<td>0x0002</td>
<td>0xA522</td>
</tr>
<tr>
<td>0x5103</td>
<td>0xB88</td>
</tr>
<tr>
<td>0x5102</td>
<td>0xA523</td>
</tr>
<tr>
<td>0x5101</td>
<td>0xA522</td>
</tr>
<tr>
<td>0x5100</td>
<td>0xA522</td>
</tr>
<tr>
<td>0x0000</td>
<td>0xA522</td>
</tr>
</tbody>
</table>

CPU

- MBR: 0x002C
- AX: 0x002C
- BX: 0x5101
- SP: 0xA522
- Decoder
- Control Unit
- Control Bus
- Data Bus 16-bit
- Address Bus 16-bit

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

...  
Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...

Program written in C programming language

...;

C = A + B;

...

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Program written in C programming language

...;
C = A + B;
...

Assembly Language (Intel 8086)

... 
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax
...

RAM

0xFFFF
8xFFFF

CPU

0x002C
0x5101

MBR

Mov Bx, [0xA523]
Mov AX, [0xA522]

Control Bus

Decompressor

Address Bus 16-bit

Data Bus 16-bit

Decoder

0x5101

0x5101

0x5101
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)
...
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax
...

Program written in C programming language
...;
C = A + B;
...;

CPU
- AX: 0x002C
- BX: 0x0000
- SP: 0x0000
- IP: 0x5101

MBR
- Mov Bx, [0xA523]

CIR
- Mov BX, [0xA523]

Decoder
- Mov AX, [0xA522]

Control Unit

Address Bus 16-bit
- Mov [0xA524], Ax
- Add Ax, Bx
- Mov Bx, [0xA523]
- Mov Ax, [0xA522]

Data Bus 16-bit

Address Bus 16-bit
- Mov [0xA524], Ax
- Add Ax, Bx
- Mov Bx, [0xA523]
- Mov Ax, [0xA522]

RAM
- 0xFFFFE
- 0xFFFFF
- 0xFB88
- 0x0100
- 0x002C
- 0xA523
- 0xA524
- 0x5103
- 0x5102
- 0x5101
- 0x5100
- 0x0002
- 0x0001
- 0x0000
Example (Instructions Execution Cycle)

```
...  
Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...
```

```
Program written in C programming language
...
C = A + B;
...;
```

Assembly Language (Intel 8086)

```
...  
Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...
```
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)
...
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax
...

Program written in C programming language
...
C = A + B;
...

RAM

CPU

MBR
Mov Bx, [0xA523]

Control Bus

CIR
Mov Bx, [0xA523]

Decoder

Data Bus 16-bit

Address Bus 16-bit

Arithmetic and Logic Unit

Control Unit

IP
0x5102

MAR
0x5101

AX
0x002C

CX

BX

SP

0xFFEE

0xFFFD

0x012C

0x0100

0x002C

0xA524

0xA523

0xA522

0x5103

0x5102

0x5101

0x5100

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

...  
Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...

Program written in C programming language

...  
C = A + B;  
...

CPU

MBR  
Mov Bx, [0xA523]  
Mov BX, [0xA523]

CIR  
Mov BX, [0xA523]

Decoder

Control Unit

Arithmetic and Logic Unit

IP  
0x5102

MAR  
0xA523

RAM

<table>
<thead>
<tr>
<th>0x0000</th>
<th>0x0001</th>
<th>0x0002</th>
<th>0x0003</th>
<th>0x0004</th>
<th>0x0005</th>
<th>0x0006</th>
<th>0x0007</th>
<th>0x0008</th>
<th>0x0009</th>
<th>0x000A</th>
<th>0x000B</th>
<th>0x000C</th>
<th>0x000D</th>
<th>0x000E</th>
<th>0x000F</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0000</td>
<td>0x0001</td>
<td>0x0002</td>
<td>0x0003</td>
<td>0x0004</td>
<td>0x0005</td>
<td>0x0006</td>
<td>0x0007</td>
<td>0x0008</td>
<td>0x0009</td>
<td>0x000A</td>
<td>0x000B</td>
<td>0x000C</td>
<td>0x000D</td>
<td>0x000E</td>
<td>0x000F</td>
</tr>
</tbody>
</table>

Data Bus 16-bit

Address Bus 16-bit

Control Bus

0xF888 0xA524
0x100 0xA523
0x002C 0xA522
**Example (Instructions Execution Cycle)**

Assembly Language (Intel 8086)

```
... Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...
```

Program written in C programming language

```
...;  
C = A + B;  
...;
```

**CPU**

- AX: 0x002C
- BX: 0x002C
- SP: 0x002C
- IP: 0x5102
- MAR: 0xA523

**RAM**

```
0xFFFF
0xEFFD
0x5110
0x510F
...
```

**Memory Bus (MBR)**: 0x0100

**Data Bus 16-bit**

- Mov BX, [0xA523]

**Control Bus**

- Mov BX, [0xA523]

**Address Bus 16-bit**

- Mov [0xA524], Ax
- Add Ax, Bx
- Mov Bx, [0xA523]
- Mov Ax, [0xA522]
- Mov Ax, [0xA523]

**Karim Lounis (Queen’s University, Canada)**
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

... Mov Ax, [0xA522] Mov Bx, [0xA523] Add Ax, Bx Mov [0xA524], Ax ...

Program written in C programming language

...;
C = A + B;
...;

CPU

MBR 0x0100

AX 0x002C
BX 0x0100
SP

Mov BX, [0xA523]

CIR

CX

DX

Control Unit

Decoder

Data Bus 16-bit

0x0100

0x002C

Mov [0xA524], Ax Add Ax, Bx Mov Bx, [0xA523] Mov Ax, [0xA522] Mov [0xA523], Ax

Address Bus 16-bit

RAM

0xFFFF 0xBFFF BFFFF 0xA524

0xFB88 0xA523

0x0100 0xA522

0xA5103 0xA5102 0xA5101 0xA5100

0x0002 0x0001 0x0000
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

...  
Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax
...

Program written in C programming language

...,  
C = A + B;  
...;

RAM

0xFFFF  
0xEEE5

0x00100

Data Bus 16-bit

Control Bus

Address Bus 16-bit

0xFFFF  
0xA524

0x0100  
0xA523

0x002C  
0xA522

0x5103  
0x5102

0x5101  
0x5100

0x0002  
0x0000
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)
...
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0x524], Ax
...

Program written in C programming language
...
C = A + B;
...

RAM
0xFFFFFFFF
0xFFFFFFFF
0x5100
0x5101
0x5102
0x5103
0xA522
0xA523
0xA524
0xFB88
...

CPU

ARITHMETIC AND LOGIC UNIT

IP 0x5102
MAR 0x5102

MBR
Add Ax, Bx

CIR
Mov BX, [0xA523]

DX

Decoer

BP

Control Unit

ADDRES S BUS 16-bit

Data Bus 16-bit

Control Bus

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

...  
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax
...

Program written in C programming language

...;
C = A + B;
...;

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Program written in C programming language:
```c
...;
C = A + B;
...;
```

Assembly Language (Intel 8086):
```assembly
...  
Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...  
```

### CPU
- **MBR**: Add Ax, Bx
- **ARITHMETIC AND LOGIC UNIT**
  - **AX**: 0x002C
  - **BX**: 0x0100
  - **SP**: 
  - **IP**: 0x5102
- **CIR**: Add Ax, Bx
- **Decoder**: Add Ax, Bx
- **Control Unit**: Add Ax, Bx
- **Address Bus 16-bit**: 0x5101
- **Data Bus 16-bit**: Add Ax, Bx

### RAM
- Memory addresses:
  - 0xFFFFE
  - 0xFFFFD
  - 0xA522
  - 0xA523
  - 0xA524
- Instructions:
  - Mov [0xA524], Ax
  - Add Ax, Bx
  - Mov Bx, [0xA523]
  - Mov Ax, [0xA522]
  - Mov Ax, [0xA524]
  - Mov Bx, [0xA523]
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

...  
Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...

Program written in C programming language

C = A + B;

...

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)
Example (Instructions Execution Cycle)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)
...
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax...

Program written in C programming language
...
C = A + B;
...

RAM
0xFFFF
0xEEEE

CPU

0x012C
0x0100
0x5103
0x5103

MBR
Add Ax, Bx
Add Ax, Bx

CIR

DX

Decoder

Control Bus

0xFB88
0x0100
0x002C
0xA524
0xA523
0xA522

0x5103
0x5102
0x5101
0x5100
0x0002
0x0001
0x0000

Data Bus 16-bit

Address Bus 16-bit

Arithmetic and Logic Unit

Control Unit

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

... Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...

Program written in C programming language

...:
C = A + B;
...:

Karim Lounis (Queen’s University, Canada)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

... Mov Ax, [0xA522]  
Mov Bx, [0xA523]  
Add Ax, Bx  
Mov [0xA524], Ax  
...

Program written in C programming language

... C = A + B;  
..;

CPU

<table>
<thead>
<tr>
<th>AX</th>
<th>0x012C</th>
<th>CX</th>
</tr>
</thead>
<tbody>
<tr>
<td>BX</td>
<td>0x0100</td>
<td>DX</td>
</tr>
<tr>
<td>SP</td>
<td></td>
<td>BP</td>
</tr>
</tbody>
</table>

Arithmetic and Logic Unit

IP  | 0x5103 | MAR   | 0x5103 |

Control Unit

Data Bus 16-bit

Control Bus

Address Bus 16-bit

RAM

0x0000 0x0001 0x0002 0x0003 0x0004 0x0005 0x0006 0x0007 0x0008 0x0009 0x000A 0x000B 0x000C 0x000D 0x000E 0x000F 0x0010 0x0011 0x0012 0x0013 0x0014 0x0015 0x0016 0x0017 0x0018 0x0019 0x001A 0x001B 0x001C 0x001D 0x001E 0x001F 0x0020 0x0021 0x0022 0x0023 0x0024 0x0025 0x0026 0x0027 0x0028 0x0029 0x002A 0x002B 0x002C 0x002D 0x002E 0x002F 0x0030 0x0031 0x0032 0x0033 0x0034 0x0035 0x0036 0x0037 0x0038 0x0039 0x003A 0x003B 0x003C 0x003D 0x003E 0x003F 0x0040 0x0041 0x0042 0x0043 0x0044 0x0045 0x0046 0x0047 0x0048 0x0049 0x004A 0x004B 0x004C 0x004D 0x004E 0x004F 0x0050 0x0051 0x0052 0x0053 0x0054 0x0055 0x0056 0x0057 0x0058 0x0059 0x005A 0x005B 0x005C 0x005D 0x005E 0x005F 0x0060 0x0061 0x0062 0x0063 0x0064 0x0065 0x0066 0x0067 0x0068 0x0069 0x006A 0x006B 0x006C 0x006D 0x006E 0x006F 0x0070 0x0071 0x0072 0x0073 0x0074 0x0075 0x0076 0x0077 0x0078 0x0079 0x007A 0x007B 0x007C 0x007D 0x007E 0x007F 0x0080 0x0081 0x0082 0x0083 0x0084 0x0085 0x0086 0x0087 0x0088 0x0089 0x008A 0x008B 0x008C 0x008D 0x008E 0x008F 0x0090 0x0091 0x0092 0x0093 0x0094 0x0095 0x0096 0x0097 0x0098 0x0099 0x009A 0x009B 0x009C 0x009D 0x009E 0x009F 0x00A0 0x00A1 0x00A2 0x00A3 0x00A4 0x00A5 0x00A6 0x00A7 0x00A8 0x00A9 0x00AA 0x00AB 0x00AC 0x00AD 0x00AE 0x00AF 0x00B0 0x00B1 0x00B2 0x00B3 0x00B4 0x00B5 0x00B6 0x00B7 0x00B8 0x00B9 0x00BA 0x00BB 0x00BC 0x00BD 0x00BE 0x00BF 0x00C0 0x00C1 0x00C2 0x00C3 0x00C4 0x00C5 0x00C6 0x00C7 0x00C8 0x00C9 0x00CA 0x00CB 0x00CC 0x00CD 0x00CE 0x00CF 0x00D0 0x00D1 0x00D2 0x00D3 0x00D4 0x00D5 0x00D6 0x00D7 0x00D8 0x00D9 0x00DA 0x00DB 0x00DC 0x00DD 0x00DE 0x00DF 0x00E0 0x00E1 0x00E2 0x00E3 0x00E4 0x00E5 0x00E6 0x00E7 0x00E8 0x00E9 0x00EA 0x00EB 0x00EC 0x00ED 0x00EE 0x00EF 0x00F0 0x00F1 0x00F2 0x00F3 0x00F4 0x00F5 0x00F6 0x00F7 0x00F8 0x00F9 0x00FA 0x00FB 0x00FC 0x00FD 0x00FE 0x00FF
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)

...;
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax
...

Program written in C programming language

...;
C = A + B;
...

Karim Lounis (Queen's University, Canada)
Example (Instructions Execution Cycle)

Assembly Language (Intel 8086)
...
Mov Ax, [0xA522]
Mov Bx, [0xA523]
Add Ax, Bx
Mov [0xA524], Ax
...

Program written in C programming language
...
C = A + B;
...

Karim Lounis (Queen's University, Canada)
Example (Instructions Execution Cycle)
Isn’t the CPU a shared resource?

The operating system has to arrange that one CPU is used to execute: the **Operating system kernel code** & a set of **processes** (from OS & user).

$E_1$: What happens if the user executes one of the C-programs below?

```c
.. while (1) for (int i=0; i<1;) do
    X: goto Y;
    { 
    } { 
    } 
Y: goto X;
.. 
   } } 
   } 
   } } } while(!0);
```

$E_2$: What happens if the user types something into the keyboard?

$E_3$: What happens if network traffic arrives through the network card?

$E_4$: What happens if the user types something into the keyboard?

$E_5$: What happens if the user moves its mouse?

$E_6$: What happens if the user’s program creates an error (e.g., 0/0)?
End