

COMPUTER STRUCTURE (I)

Computer Architecture and Operating Systems

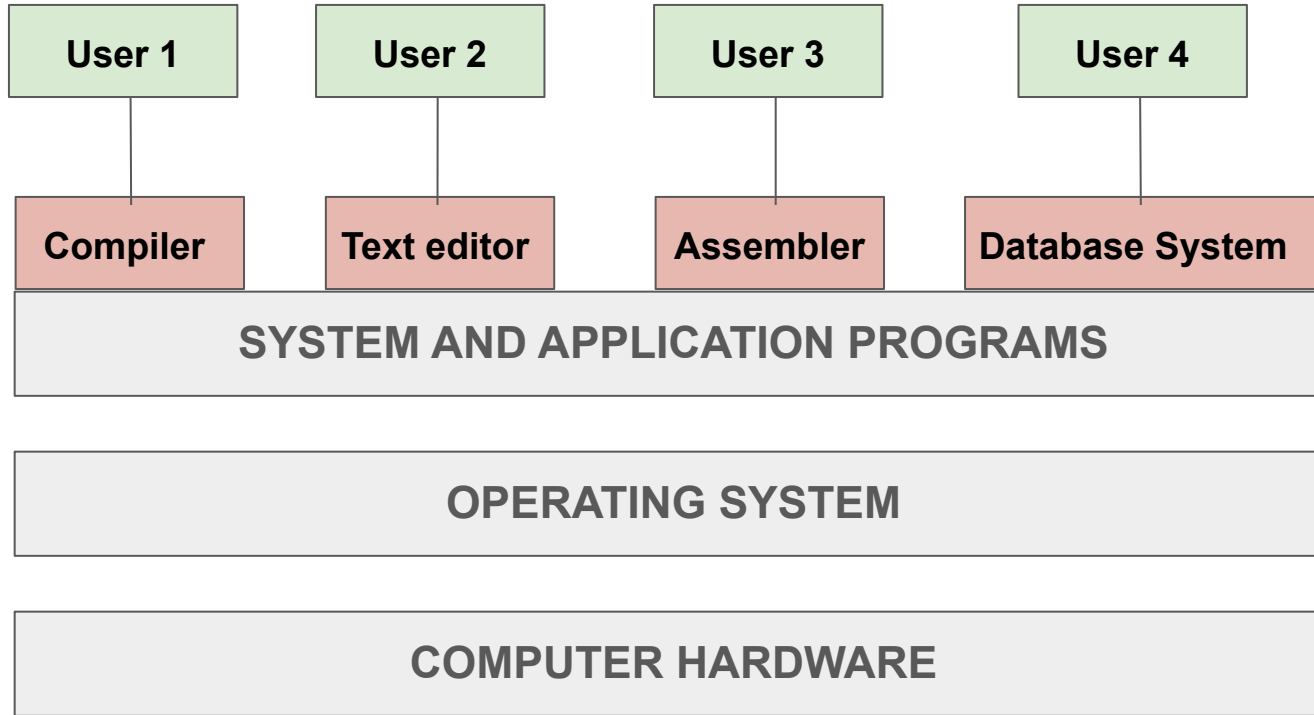
Content

- Computer
 - Conceptual view of a computer
 - Computer structure
- Instructions

What we are going to see is...

- What is a computer
- How a computer can be used and tools that can be useful.
- Know how a computer works and levels that are available on which we can work.
- Understand connection and dependence between levels

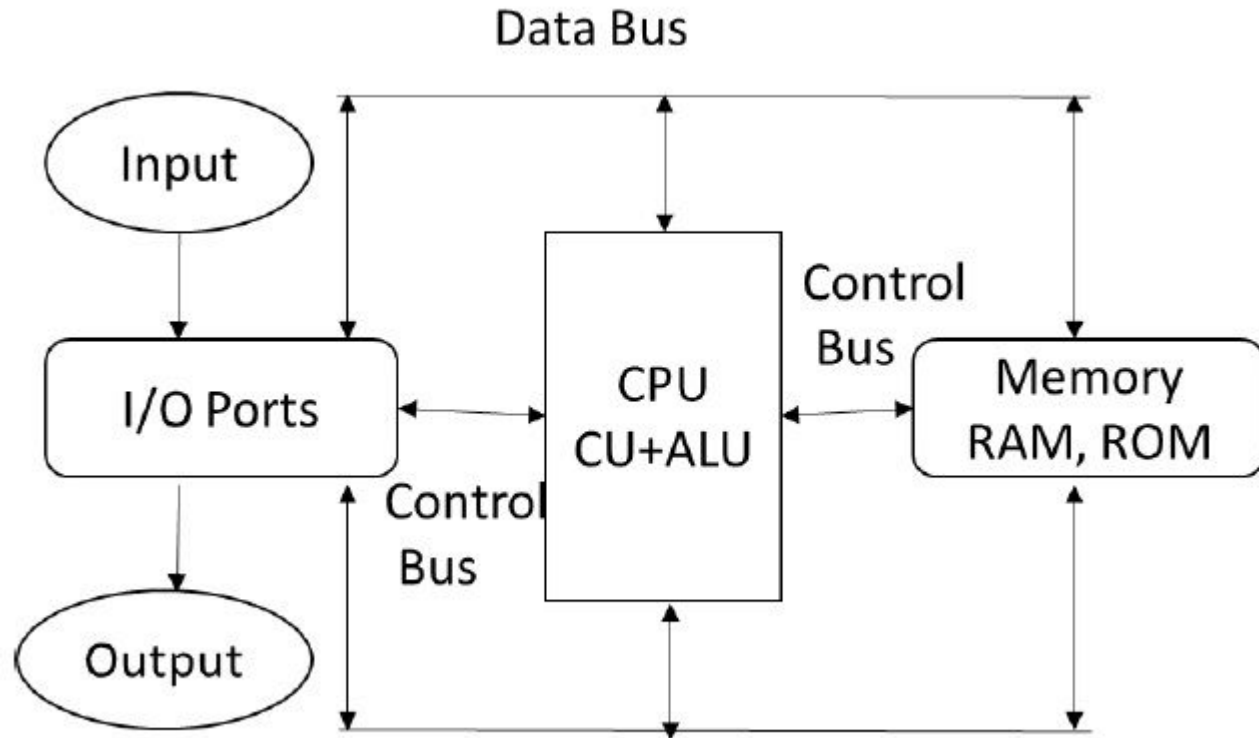
Conceptual view of a computer system



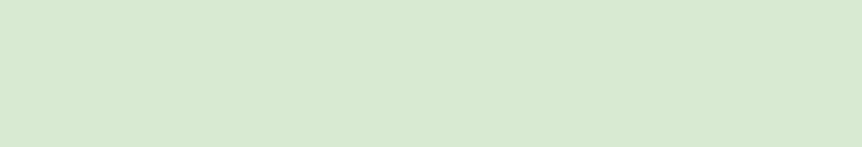
System examples



Computer structure



The diagram illustrates a computer system with two main components: a Processor - CPU and RAM. The Processor - CPU is represented by a light green rounded rectangle on the left. The RAM is represented by a light yellow rounded rectangle on the right, which contains a grid of 10 rows and 8 columns, totaling 80 small squares, symbolizing memory storage.



Processor - CPU

[illegible][illegible]

Memory

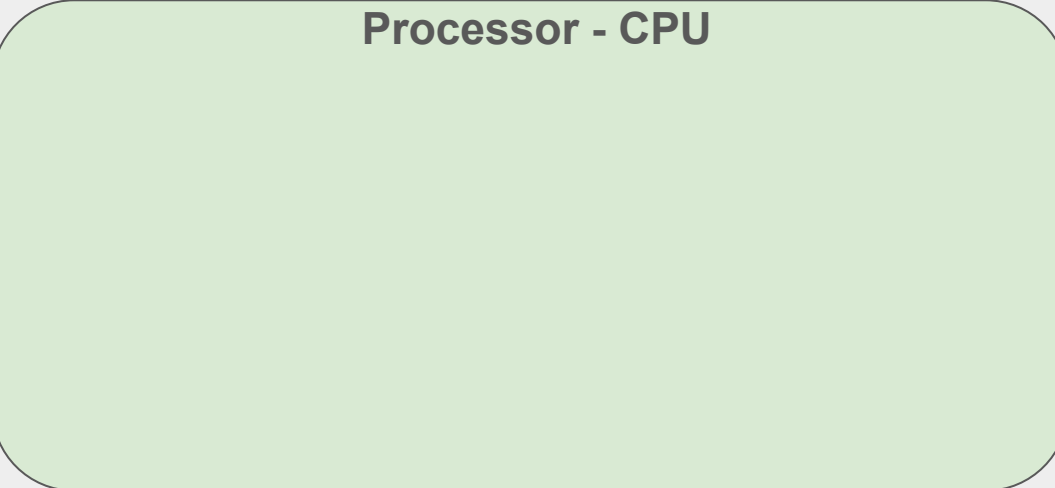
- Composed by cells that can store 1 BIT
- Positional organized, usually in 8 BITS (1 BYTE)
- Each position is identified by one address
- Addresses are composed by groups of 0s and 1s
- Memory position content is a group of 0s and 1s
- Content of memory can be data or instructions
- Memory positions can be read (retrieving information) or write (storing information)

The diagram illustrates the components of a computer system, divided into two main sections: the Processor - CPU and RAM.

Processor - CPU: This section is represented by a large green rounded rectangle. Below it are three smaller green rounded rectangles, each representing a different type of bus or interface:

- Address:** The top bus, labeled "Address".
- Data:** The middle bus, labeled "Data".
- Control:** The bottom bus, labeled "Control".

RAM: This section is represented by a yellow rounded rectangle. It contains a grid of 10 rows and 8 columns, representing memory cells. The grid is labeled "RAM Content" at the top and "Address" on the left side, indicating the memory address for each row.



Processor - CPU

Address

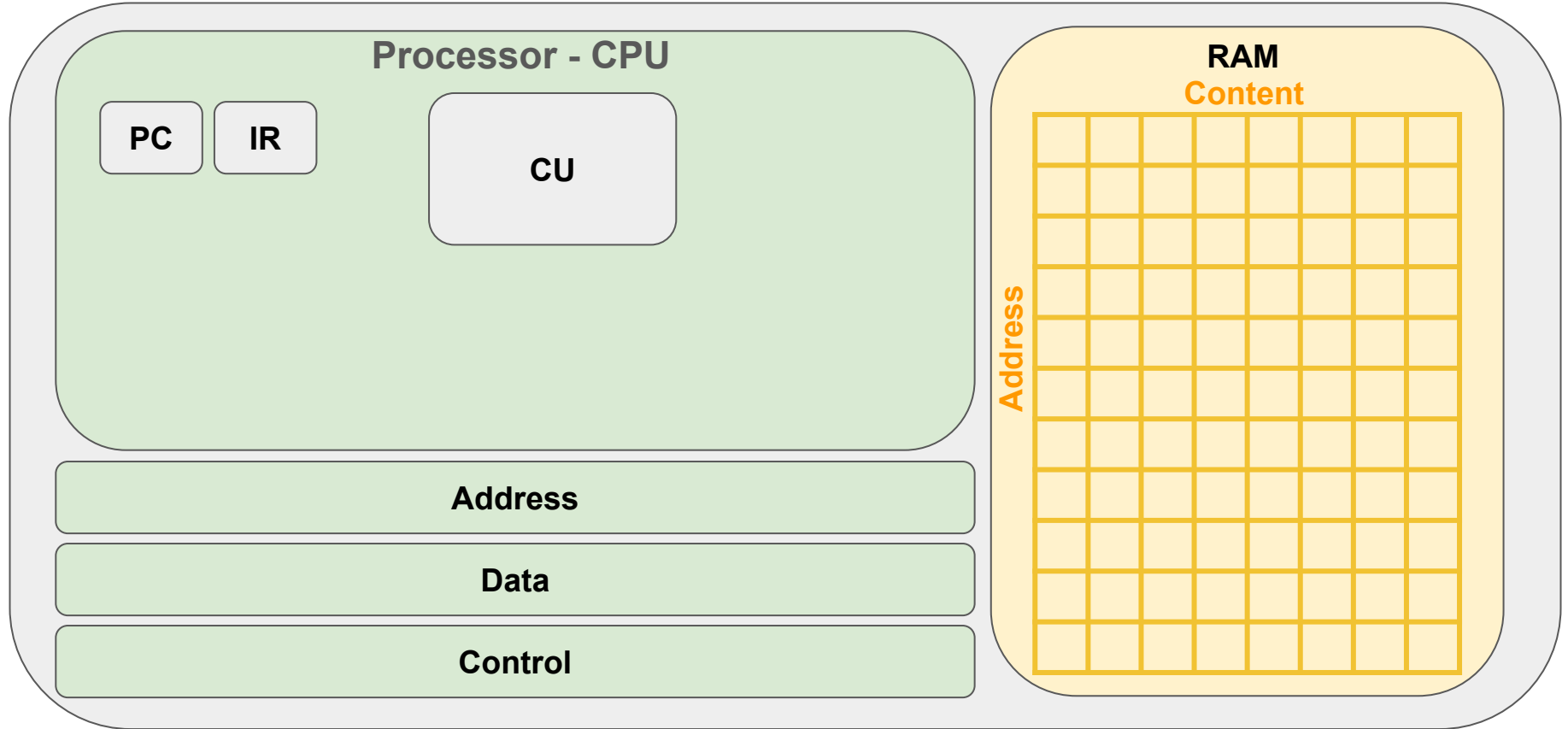
Data

Control

The diagram illustrates the structure of Random Access Memory (RAM). It features a grid with 12 rows and 8 columns. The vertical axis is labeled 'Address' and the horizontal axis is labeled 'Content'. The grid is divided into 96 individual cells, representing the memory locations.

Address

The computer



The processor

- **Program counter (PC):** It holds the address where next instruction which is to be executed, is stored.
- **Instruction Register (IR):** It holds the instruction which is executing.
- **Control Unit (CU):** It directs the operation of the processor. A CU typically uses a binary decoder to convert coded instructions into timing and control signals that direct the operation of the other units (memory, arithmetic logic unit and input and output devices, etc.).

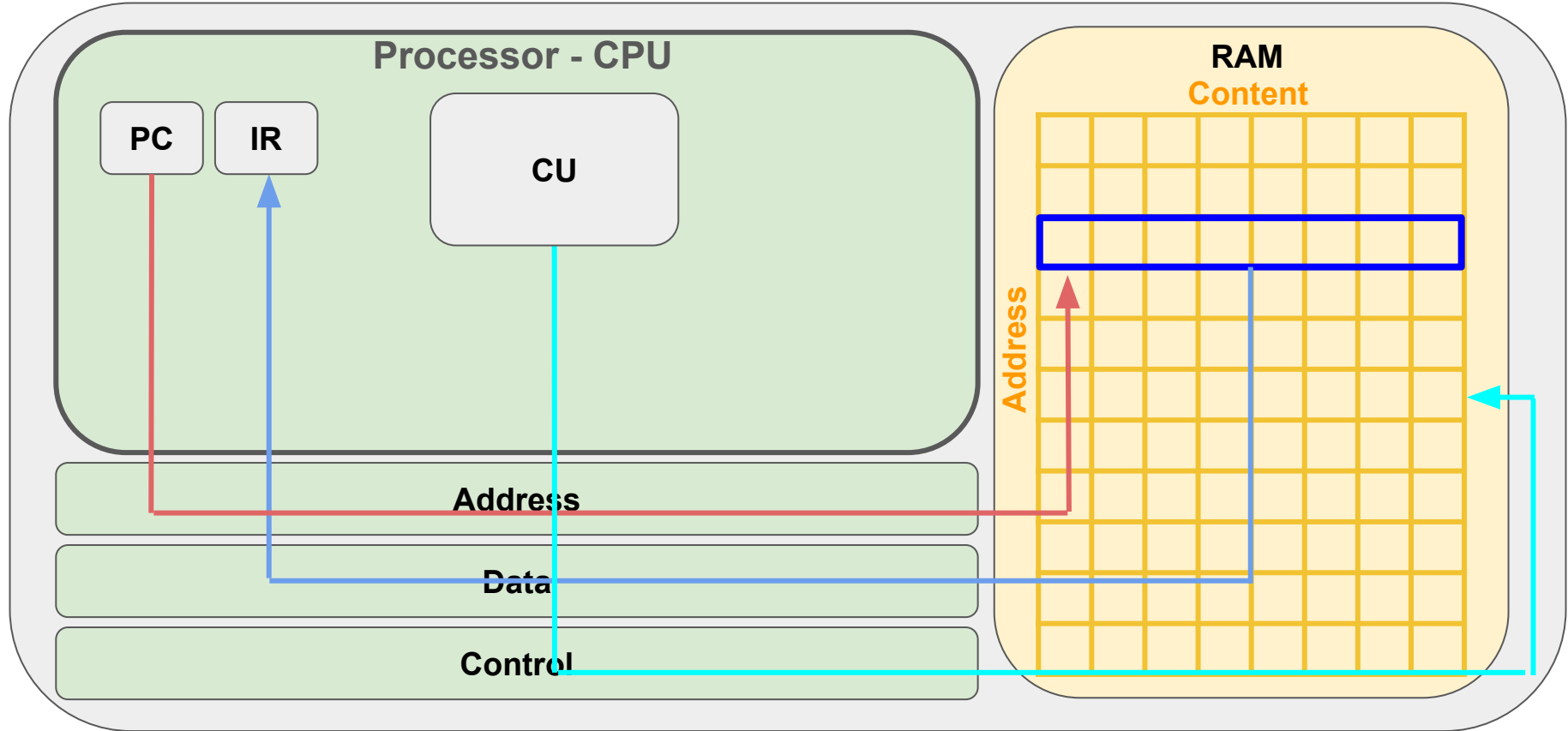
Machine cycle

- A machine cycle consists of the steps that a computer's processor executes whenever it receives a machine language instruction.
- It is the most basic CPU operation, and modern CPUs are able to perform millions of machine cycles per second

Steps of a machine cycle

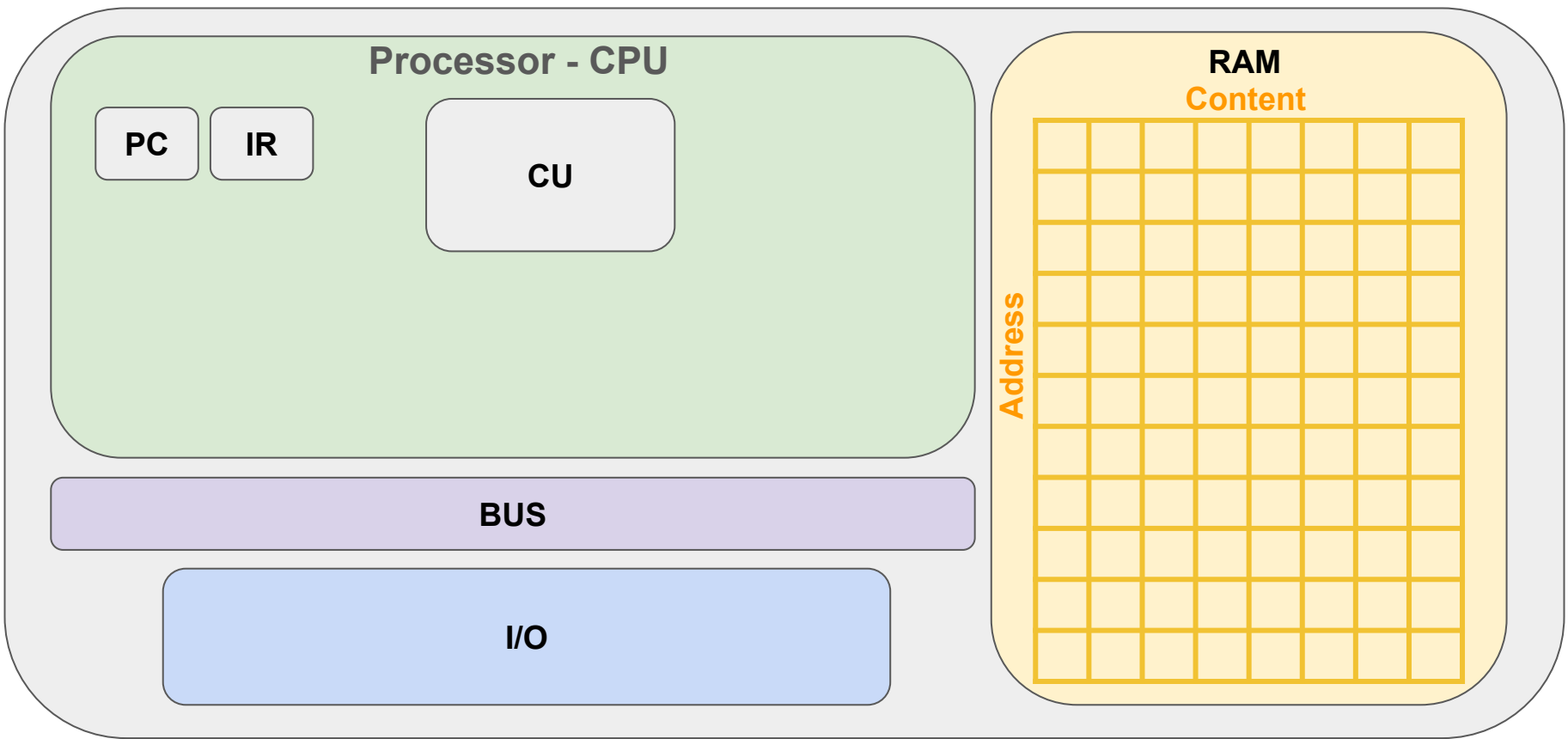
- **Fetch:** The control unit requests instructions from the main memory that is stored at a memory's location as indicated by the program counter (instruction counter)
- **Decode:** Received instructions are decoded in the instruction register. This involves breaking the operand field into its components based on the instruction's operation code (opcode)
- **Execute:** This involves the instruction's opcode as it specifies the CPU operation required. The program counter indicates the instruction sequence for computer. These instructions are arranged into the instructions register and as each are executed, it increments the program counter so that the next instruction is stored in memory. Appropriate circuitry is then activated to perform the requested task. As soon as instructions have been executed, it restarts the machine cycle that begins the fetch step.

The computer



The diagram illustrates the architecture of a computer system, divided into four main components:

- Processor - CPU:** A green rounded rectangle containing three sub-components:
 - PC (Program Counter):** A small white rounded rectangle.
 - IR (Instruction Register):** A small white rounded rectangle.
 - CU (Control Unit):** A larger white rounded rectangle.
- BUS:** A horizontal purple bar connecting the Processor - CPU to the I/O component.
- I/O (Input/Output):** A horizontal blue rounded rectangle at the bottom of the system.
- RAM (Random Access Memory):** A yellow rounded rectangle on the right side, containing a grid of 10 rows and 10 columns. The grid is labeled **Address** on the left and **Content** at the top.

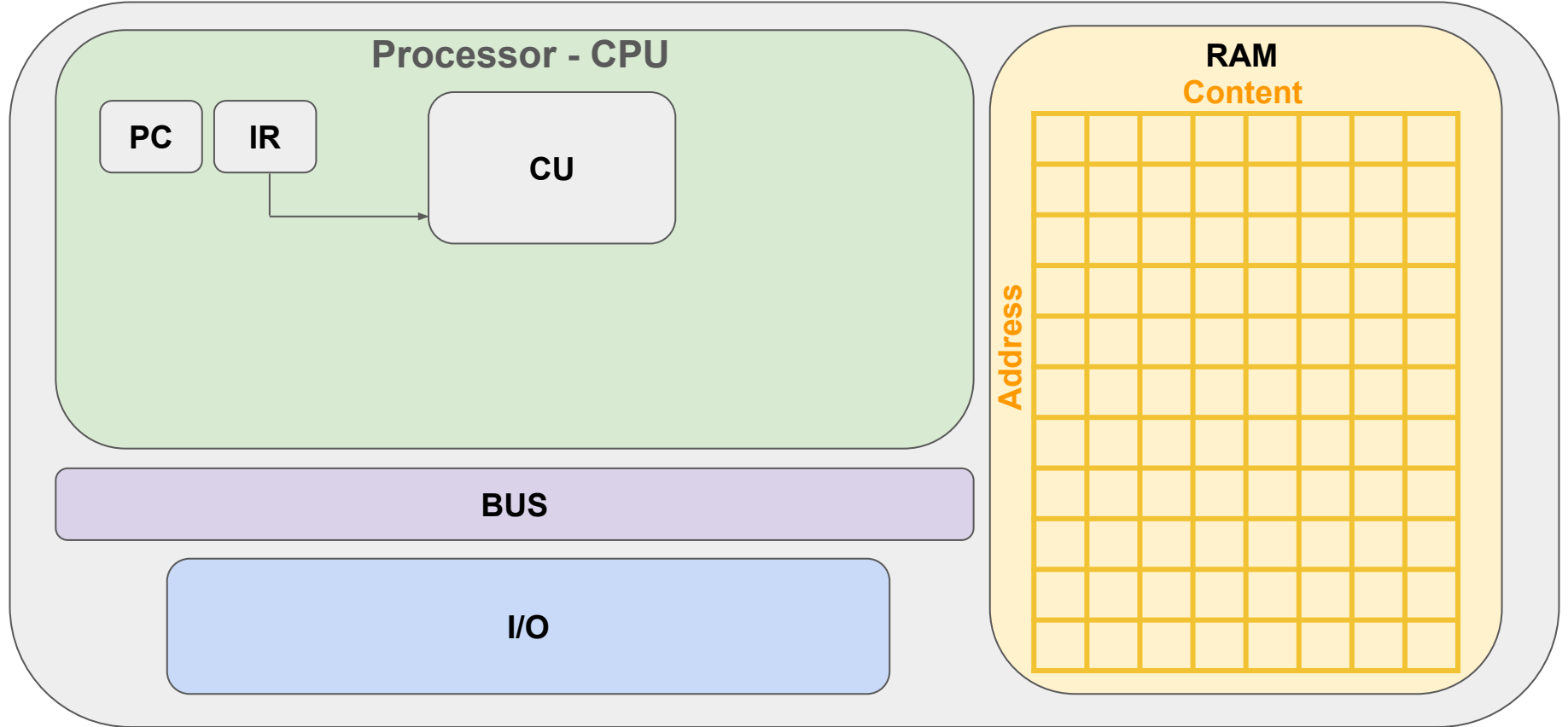


Computer

Cycle:

- Instruction fetch:
 - Send the address stored in PC
 - Memory content read
 - Getting instruction and store it in IR
 - Update - increase the value of PC by 1
- Instruction decode

The computer



Instructions

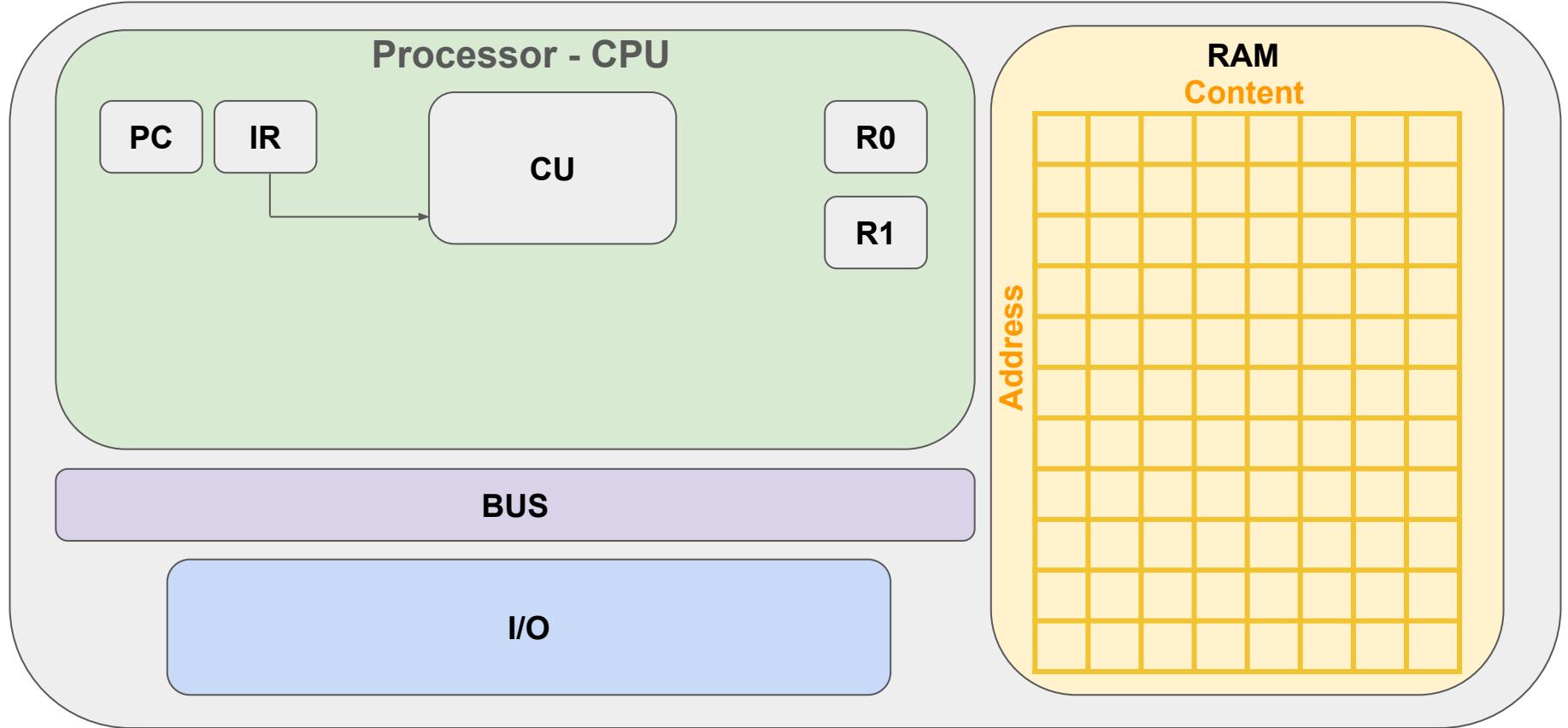


There are instruction sets with 1, 2 or 3 operands

Operands can be:

- Immediate
- Register
- Memory address

The computer



Computer

Cycle:

- Instruction fetch:
 - Send the address stored in PC
 - Memory content read
 - Getting instruction and store it in IR
 - Update - increase the value of PC by 1
- Instruction decode
- Instruction execution

C program

```
#include <stdio.h>
```

```
int A, B, C;      →      Variable declaration
```

```
main() {
```

```
    A = 5;      |      →      Value assignment
```

```
    B = 7;
```

```
    C = A + B;      →      Operation between variables
```

```
    printf("A + B = %ld", C);
```

```
}
```

C program

```
#include <stdio.h>
```

int A, B, C; \longrightarrow Variable declaration

```
main() {
```

A = 5;

B = 7;

$$C = A + B;$$

```
printf("A + B = %ld", C);
```

}

[illegible]

Variable types

int A;

double B;

char C;

A

A+1

A+2

A+3

B

B+1

B+2

B+3

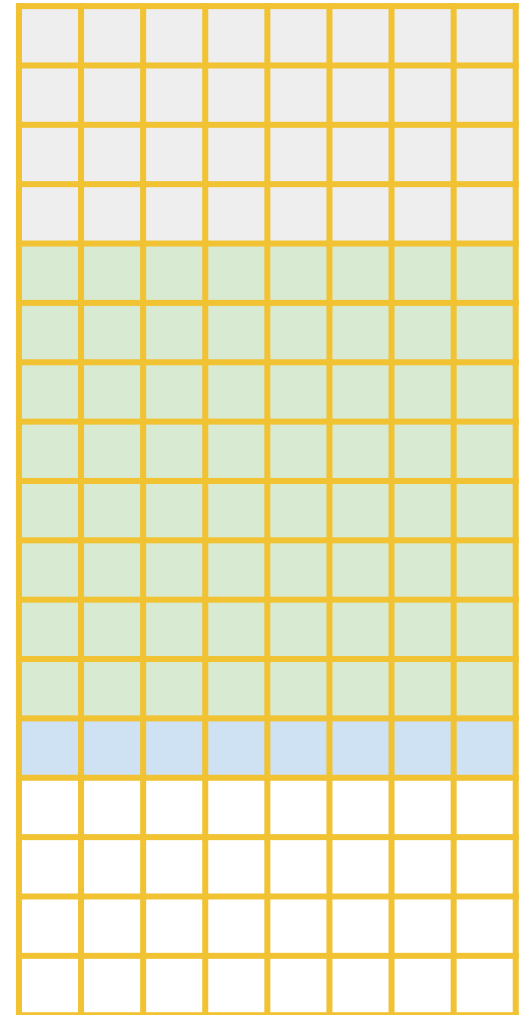
B+4

B+5

B+6

B+7

C



Variable types

```
int Vector[10];
```

Vector[0]

Vector

Vector+1

Vector+2

Vector+3

Vector[1]

Vector+4

Vector+5

Vector+6

Vector+7

Vector[2]

Vector+8

Vector+9

Vector+10

Vector+11

Vector[3]

Vector+12

Vector+13

Vector+14

Vector+15

...

Instruction set

- Instruction types
- Addressing modes

Instruction types

- Data movement instructions
- Arithmetic and logical instructions
- Control flow instruction

Data movement instructions

MOS 6502

LDA SOURCE

Imm
Mem

STA DESTINATION

Mem

INTEL x86

MOV DESTINATION, SOURCE

Reg, Imm
Reg, Reg
Reg, Mem
Mem, Imm
Mem, Reg

~~Mem, Mem~~

RISC-V

LD DESTINATION, SOURCE

rd, imm(rs1)
(lb, lw)

SD SOURCE, DESTINATION

rs2, imm(rs1)
(sb, sw)

Arithmetic and logical instructions

Arithmetic operations:

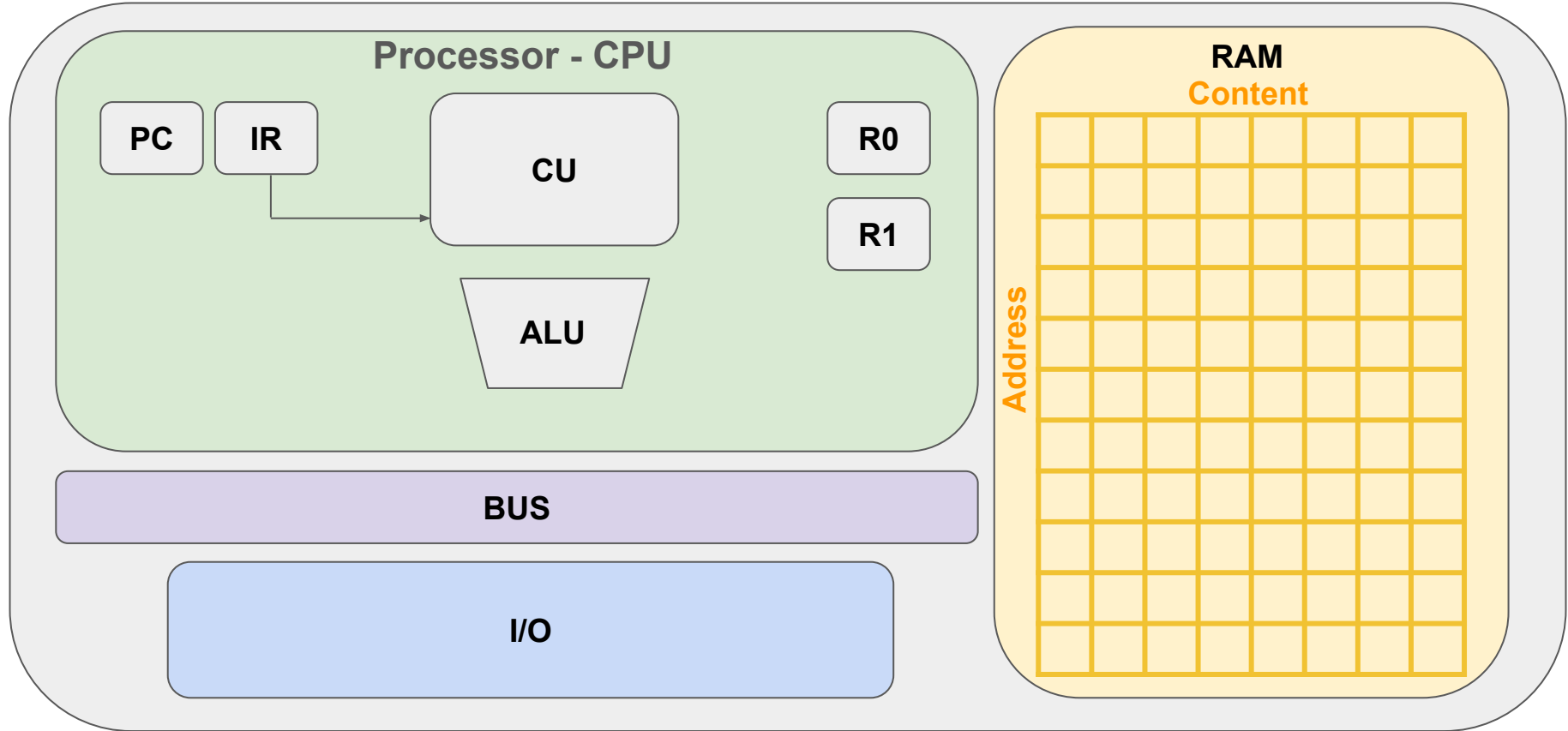
Result <- Operand_1 OP Operand_2

A = B + C;

Result <- OP Operand_1

A++;

The computer



Arithmetic and logical instructions

MOS 6502

ADD SOURCE

Imm

Mem

$Acc \leftarrow (Acc) + (SOURCE)$

SUB, ADC, SBB, ...

CMP SOURCE

INTEL X86

ADD DESTINATION, SOURCE
 $DESTINATION \leftarrow (DESTINATION) + (SOURCE)$

SUB, ADC, SBB, ...
 $CMP Op1, Op2 (Op1 - Op2)$

RISC-V

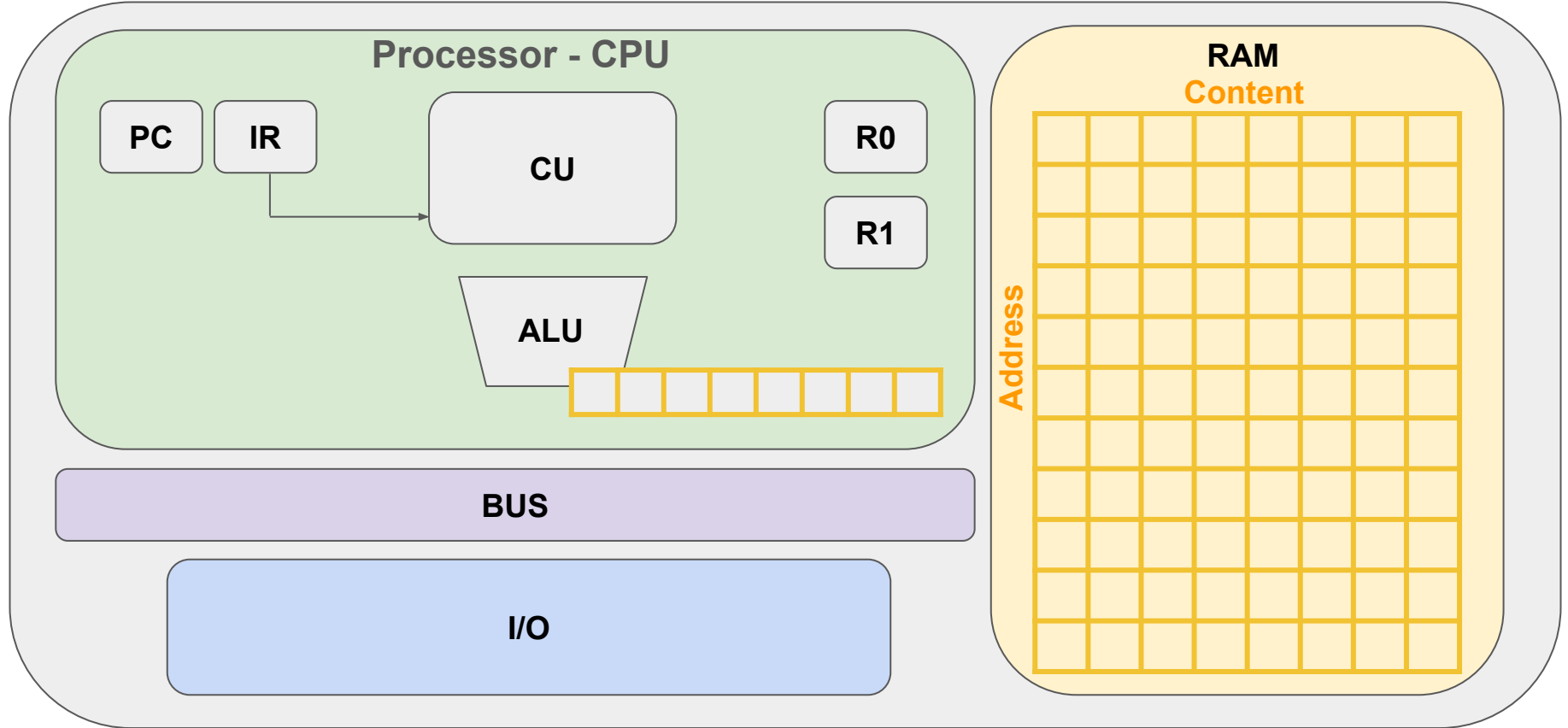
OP rd, rs1, rs2
 $rd \leftarrow (rs1) OP (rs2)$

ADD rd, rs1, rs2
SUB rd, rs1, rs2

Opi rd, rs1, imm
 $rd \leftarrow (rs1) OP imm$

ADDI rd, rs1, imm
SUBI rd, rs1, imm

The computer



Arithmetic and logical instructions

MOS 6502

ADD SOURCE

Imm

Mem

Acc \leftarrow (Acc) + (SOURCE)

SUB, ADC, SBB, ...

CMP SOURCE

~~MUL, DIV~~

INTEL X86

ADD DESTINATION, SOURCE
DESTINATION \leftarrow
(DESTINATION) + (SOURCE)

SUB, ADC, SBB, ...

CMP Op1, Op2 (Op1 - Op2)

MUL Op1

MUL BX

DX:AX \leftarrow (AX) * (BX)

DIV Op1

DIV BX

DX:AX/BX

Q(AX) i R(DX)

RISC-V

OP rd, rs1, rs2
rd \leftarrow (rs1) OP (rs2)

ADD rd, rs1, rs2
SUB rd, rs1, rs2

Opi rd, rs1, imm
rd \leftarrow (rs1) OP imm

ADDI rd, rs1, imm
SUBI rd, rs1, imm

Operate instructions

MOS 6502

INC SOURCE
Mem
 $\text{Mem} \leftarrow (\text{Mem}) + 1$

DEC SOURCE
Mem
 $\text{Mem} \leftarrow (\text{Mem}) - 1$

INX, DEX

INTEL X86

INC SOURCE
Mem
Reg
 $\text{SOURCE} \leftarrow (\text{SOURCE}) + 1$

DEC SOURCE
Mem
Reg
 $\text{SOURCE} \leftarrow (\text{SOURCE}) - 1$

RISC-V

ADDI rd, rs1, 1

Define directive - IA

Directive	Purpose	Storage Space
DB	Define Byte	allocates 1 byte
DW	Define Word	allocates 2 bytes
DD	Define Doubleword	allocates 4 bytes
DQ	Define Quadword	allocates 8 bytes
DT	Define Ten Bytes	allocates 10 bytes

choice	DB	'y'
number	DW	12345
neg_number	DW	-12345
big_number	DQ	123456789
real_number1	DD	1.234
real_number2	DQ	123.456

Intel Registers

There are ten 32-bit and six 16-bit processor registers in IA-32 architecture. The registers are grouped into three categories –

- General registers
- Control registers
- Segment registers

The general registers are further divided into the following groups –

- Data registers
- Pointer registers
- Index registers

32-bit registers



31

16 15

8 7

0

EAX

EBX

ECX

EDX

		AH	AL
	BH	BL	
	CH	CL	
	DH	DL	

16-bit registers



AX Accumulator

BX Base

CX Counter

DX Data

Adapt to Intel x86

```
int A, B, C, D;
```

```
A = B + C * D;
```

Adapt to Intel x86

A dd

B dd

C dd

D dd

MOV R1, [C]

MUL R1, [D]

ADD R1, [B]

MOV [A], R1

int A, B, C, D

A = B + C * D

Arithmetic and logical instructions

Logical operations

Result <- Op1 OP Op2

AND EAX, EBX (Bitwise operation)

OR and XOR

Result <- OP Operand_1

NOT EAX (Bitwise operation)

SHL Op1, Imm => Multiply per 2^{Imm} (SAL)

SHR Op1, Imm => Divide by 2^{Imm} (SAR)

Control flow instructions

Unconditional jump:

`jmp address`

`PC <- address`

Conditional jump:

`jne offset (jge, jg, je, jle, jl, jc, ...)`

`PC <- (PC) + offset`

Usually, offset is a 8 bit value

Jumps with return

RISC-V

Conditional jump:

`beq rs1, rs2, imm`

`bne, blt, bge, ...`

Example

```
int A, B, C;  
  
if(A >= B)  
    C = C + 1;  
  
else  
    C = C - 1;
```

```
A dd  
B dd  
C dd  
  
    MOV R1, [A]  
    MOV R2, [B]  
    CMP R1, R2  
    JL elseif  
    INC [C]  
    JMP FI  
elseif: DEC [C]  
fi:
```


Example

```
int A, B, C, D;
```

```
WHILE (A < B) {  
    C = C + D;  
    A = A + 1;  
}
```

```
A dd  
B dd  
C dd  
D dd
```

```
while:    MOV R1, [A]  
          MOV R2, [B]  
          CMP R1, R2  
          JGE endwhile  
          MOV R3, [D]  
          ADD [C], R3  
          INC R1  
          JMP while  
endwhile: MOV [A], R1
```

Exercise

```
int base, exponent, iter, result;
```

```
result = 1;
```

```
iter = 1;
```

```
while(iter <= exponent) {  
    result = result * base;  
    iter++;  
}
```



Exercise

```
int base, exponent, iter, result;
```

```
result = 1;
```

```
iter = 1;
```

```
while(iter <= exponent) {  
    result = result * base;  
    iter++;  
}
```

```
base dd
```

```
exponent dd
```

```
iter dd 1
```

```
result dd 1
```

```
MOV eax, [result]
```

```
MOV esi, [iter]
```

```
MOV ebx, [exponent]
```

```
bucle: CMP esi, ebx
```

```
JG fin
```

```
MUL eax, [base]
```

```
INC esi
```

```
JMP bucle
```

```
fin: MOV [result], eax
```

Exercise

Write this code fragment to assembler:

```
While (A < B)
{
    C = C + C;
    A = A + 1;
}
```

Exercise

Option 1:

```
0.      MOV R1, [A]
1.      MOV R2, [B]
2. WHILE: CMP R1, R2
3.      JGE END-WHILE
4.      MOV R3, [C]
5.      MOV R4, R3
6.      ADD R3, R4
7.      MOV [C], R3
8.      INC R1
9.      MOV [A], R1
10.     JMP WHILE
```

Option 2:

```
0.      MOV R1, [A]
1.      MOV R2, [B]
2.      MOV R3, [C]
3. WHILE: CMP R1, R2
4.      JGE END – WHILE
5.      ADD R3, R3
6.      MOV [C], R3
7.      INC R1
8.      MOV [A], R1
9.      JMP WHILE
10. END – WHILE:
```

Exercise

Store 1 in R1 if content of R2 is greater or equal than R3 content and R4 content is less than R5 content. if the above is not fulfilled, we store 0 in R1.

Exercise

```
0. IF:          CMP R2, R3
1.              JL ELSE
2.              CMP R4, R5
3.              JGE ELSE
4.              MOV R1, 1
5.              JMP SEQUENT
6.              MOV R1, 0
7. ELSE:
8. SEQUENT:
```

COMPUTER STRUCTURE (I)

Computer Architecture and Operating Systems