COMPUTER STRUCTURE (I)

Computer Architecture and Operating Systems

Content

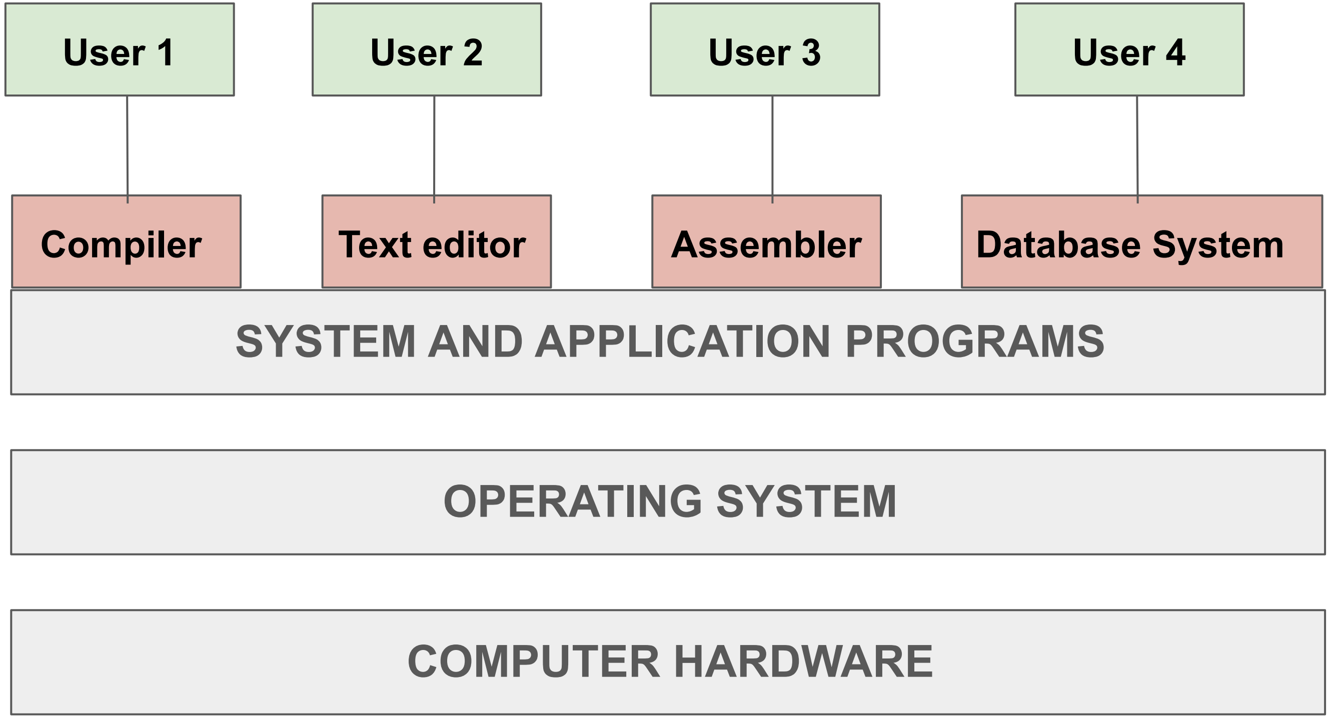
- Computer
  - Conceptual view of a computer
  - Computer structure

- 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

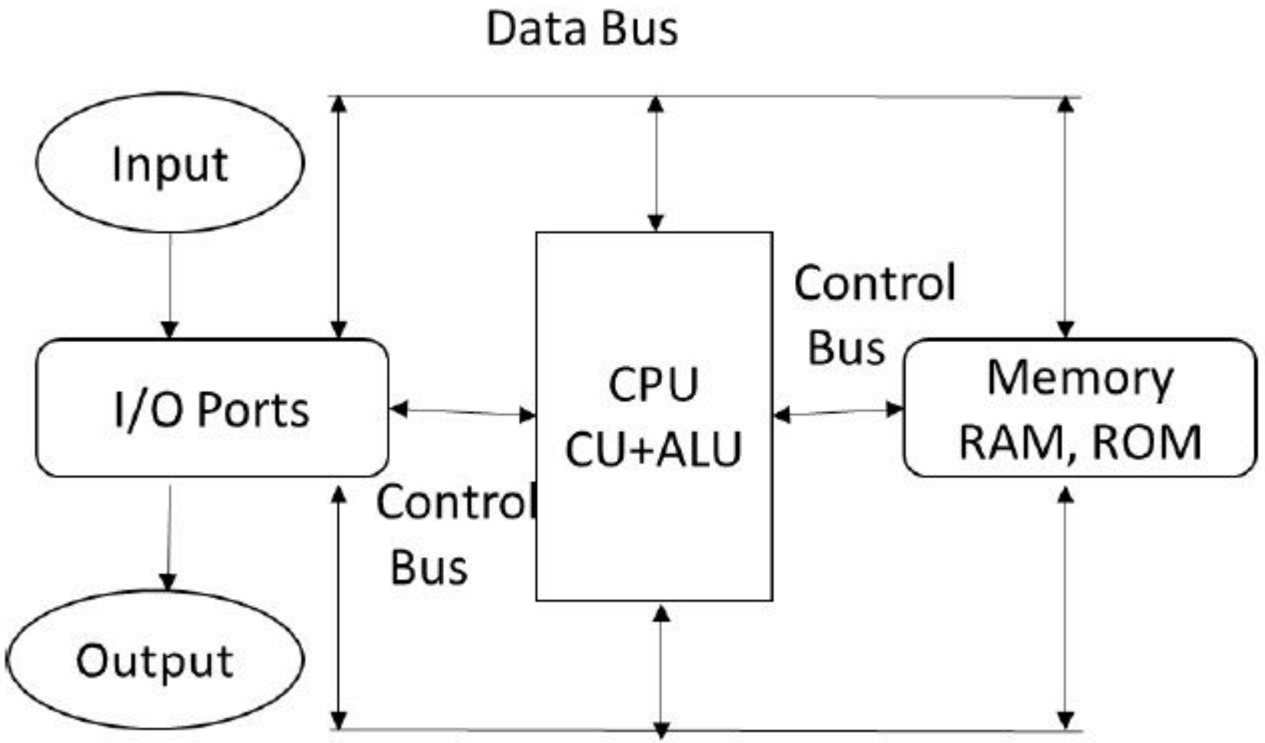
Conceptual view of a computer system



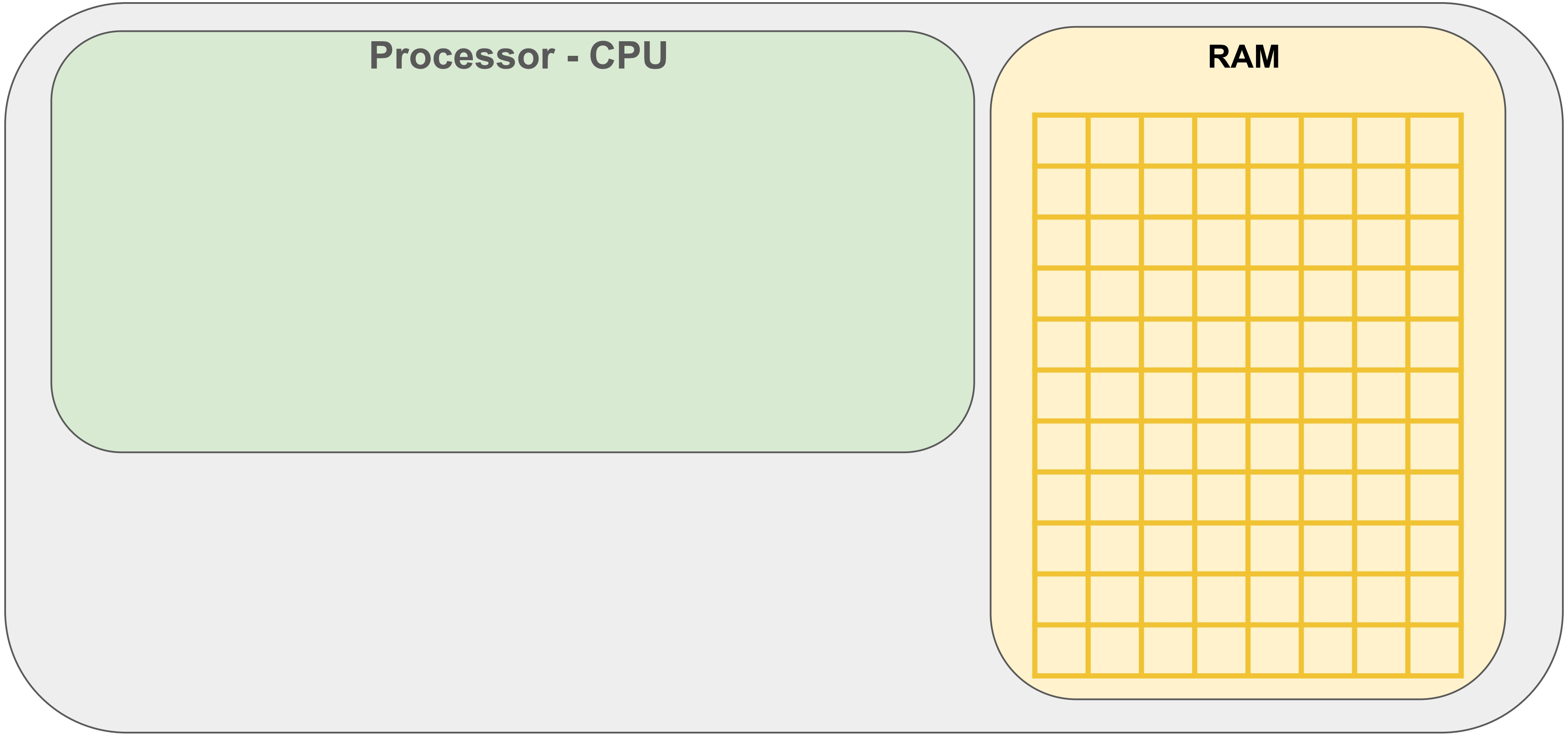
System examples

FRONTIER
OAK RIDGE National Laboratory
U.S. DEPARTMENT OF ENERGY
Hewlett Packard Enterprise
AMD
FRONTIER

Computer structure



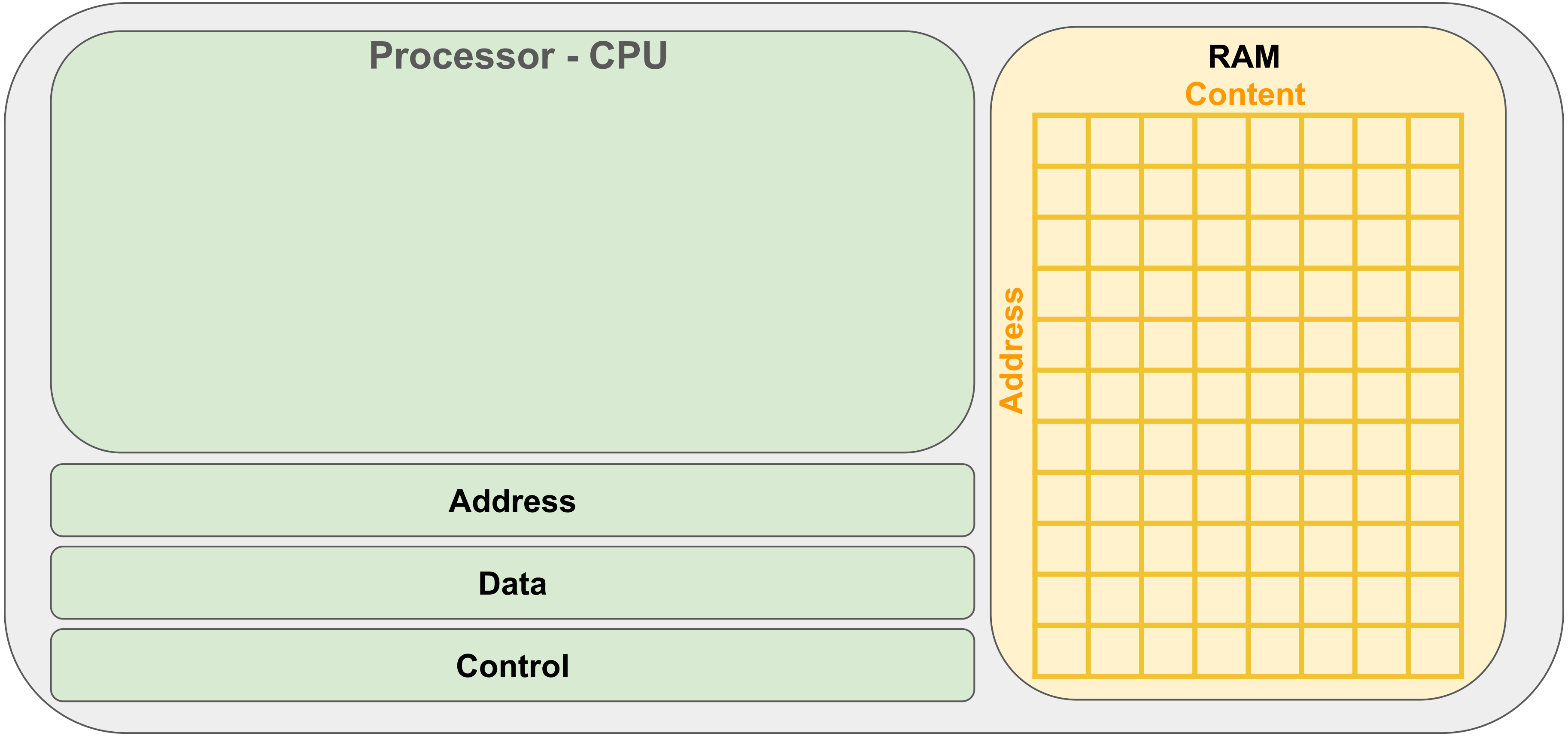
The computer



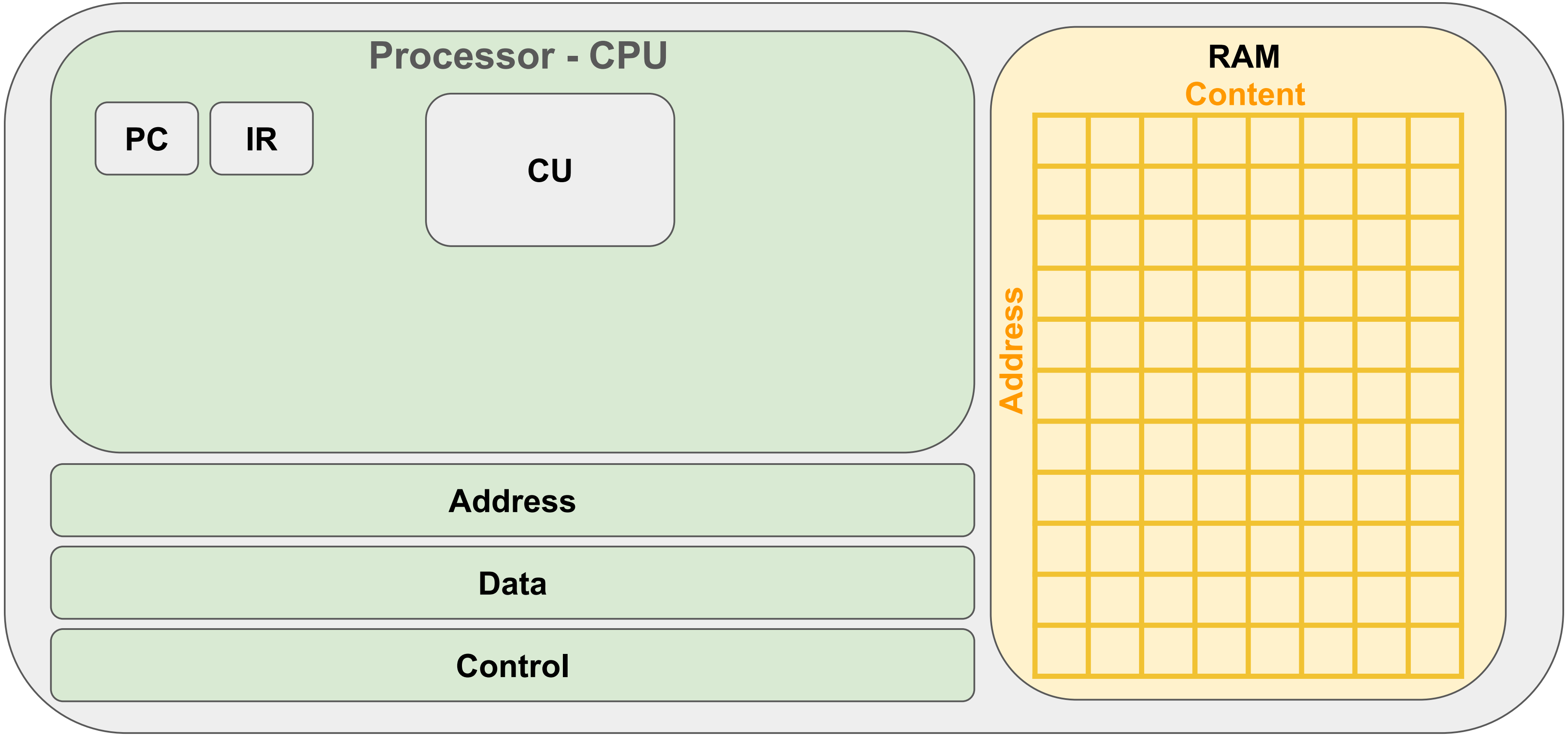
Memory

- 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 computer



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

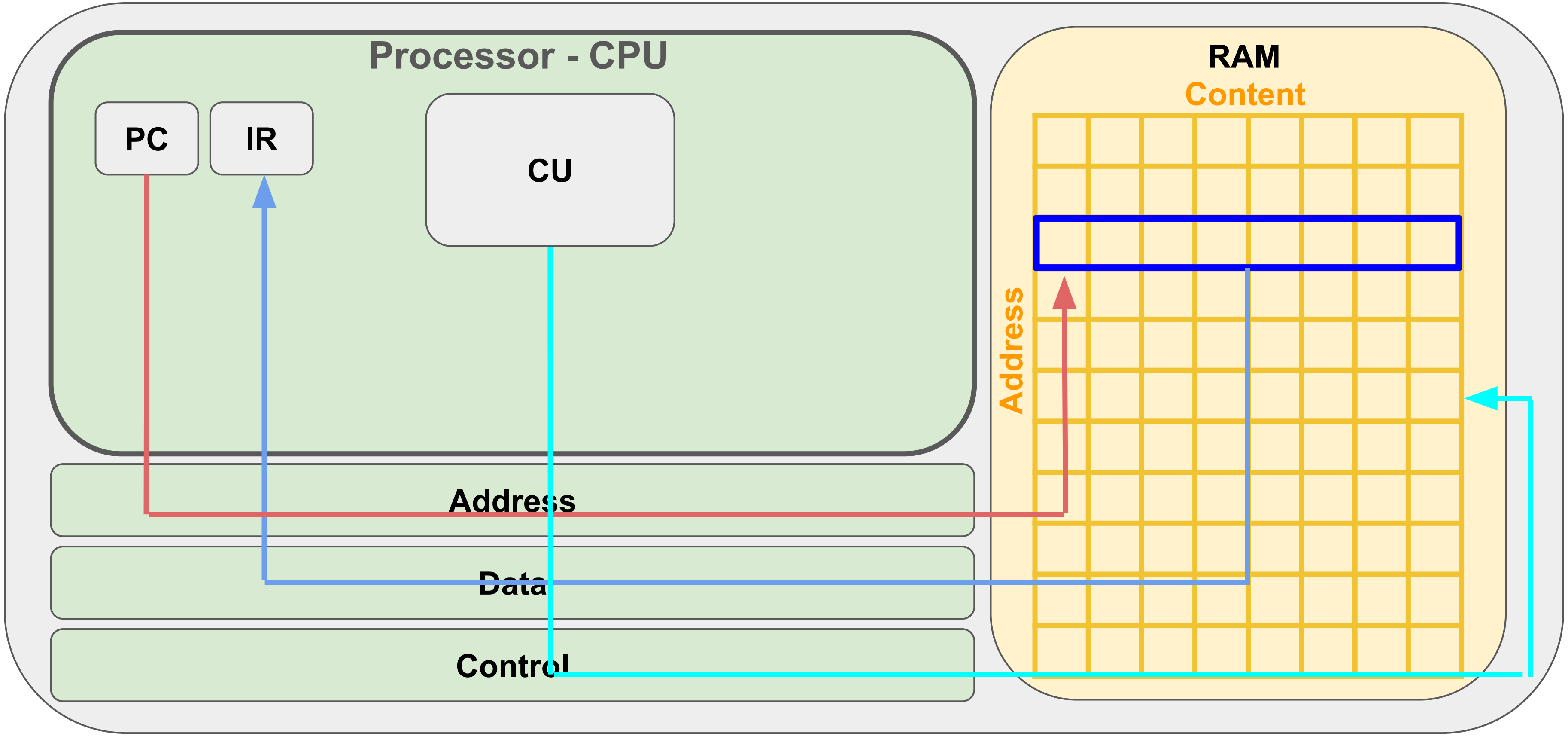
- A machine cycle consists of the steps that a computer’s processor executes whenever it receives a machine language instruction.

- It is the most basic CPU operation, and modern CPUs are able to perform millions of machine cycles per second

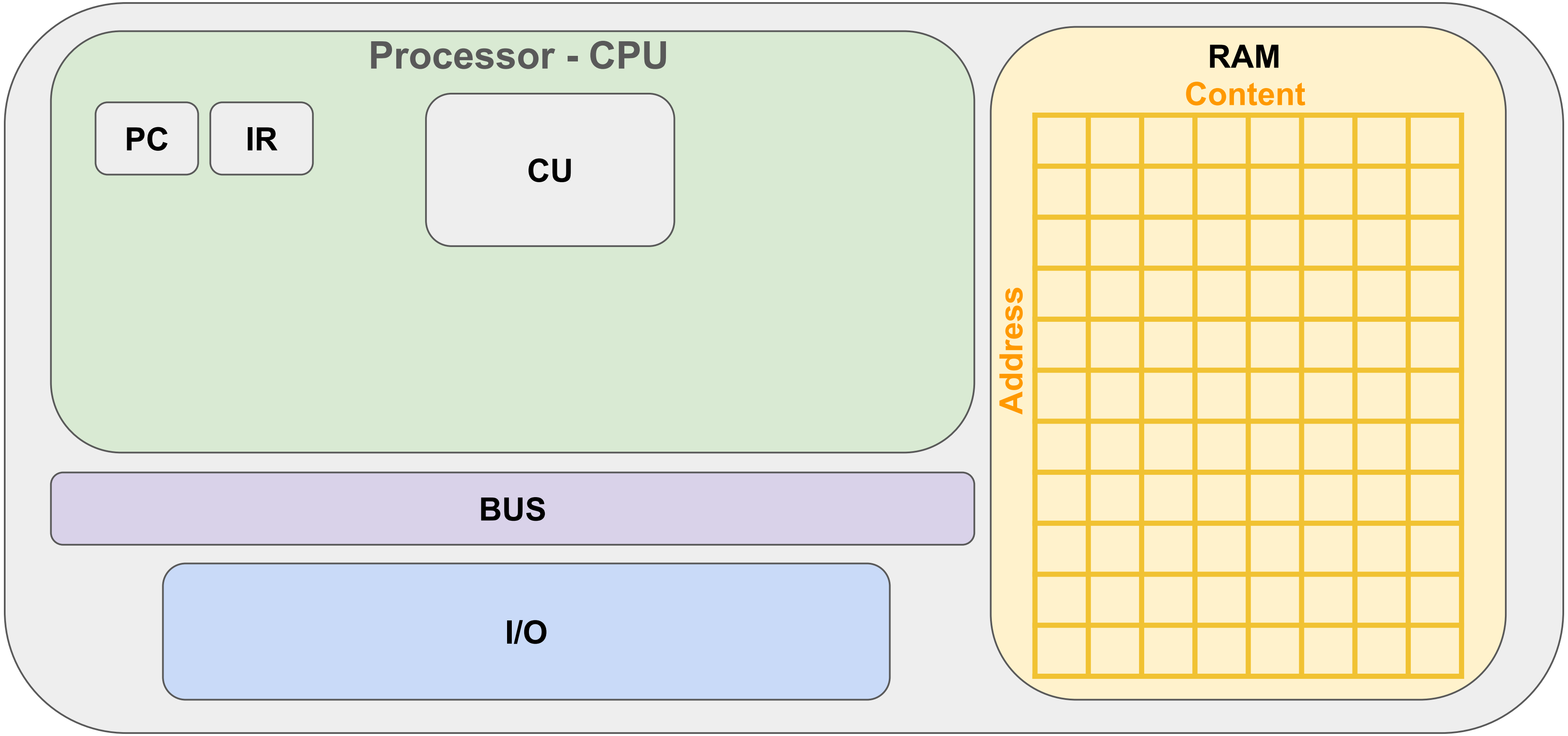
Steps of a machine cycle

- **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 computer

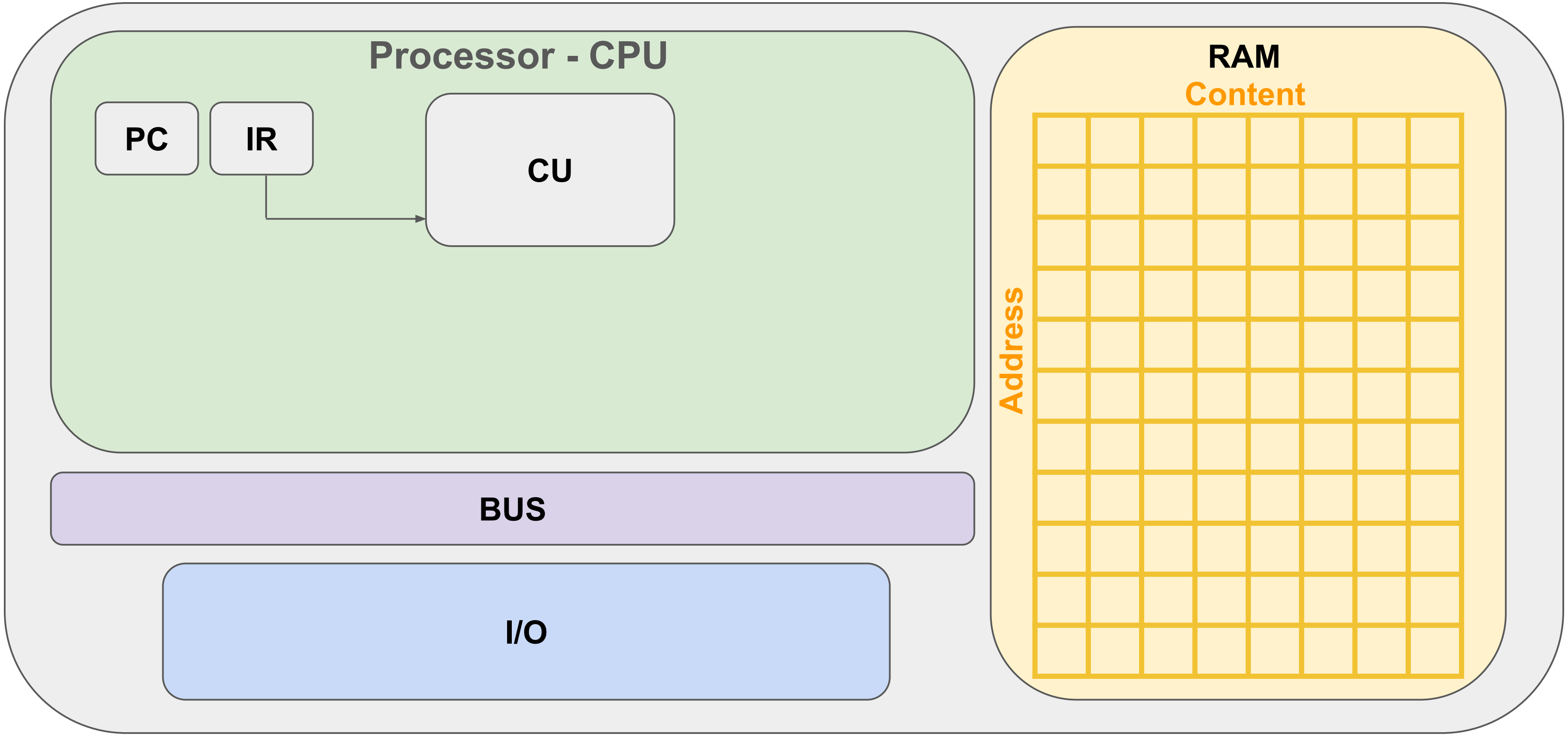


Computer

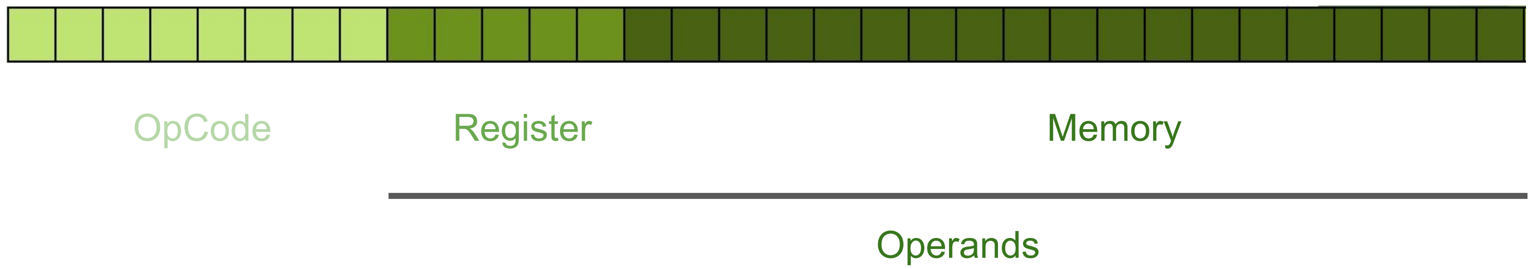
Cycle:

- 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

The computer



Instructions

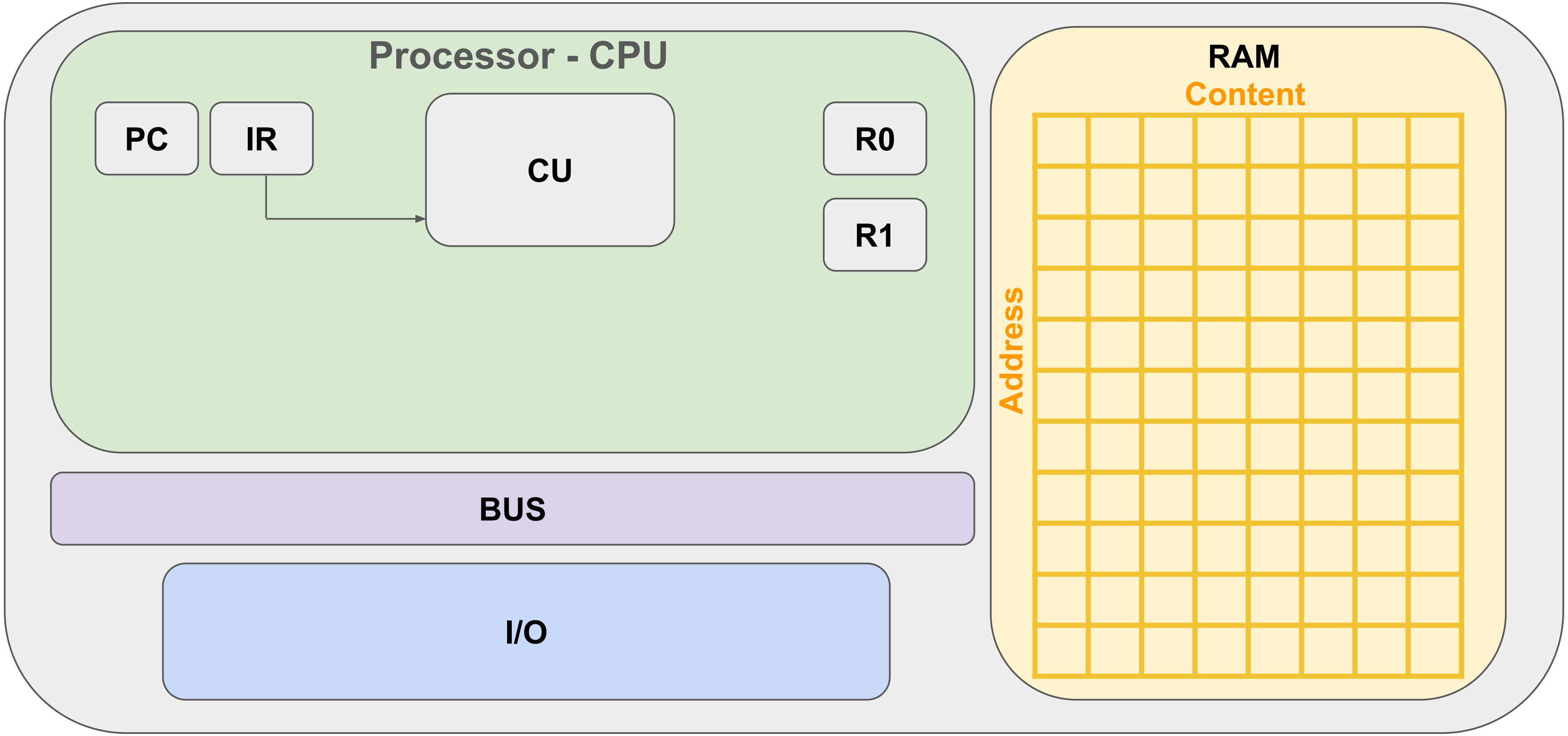


There are instruction sets with 1, 2 or 3 operands

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

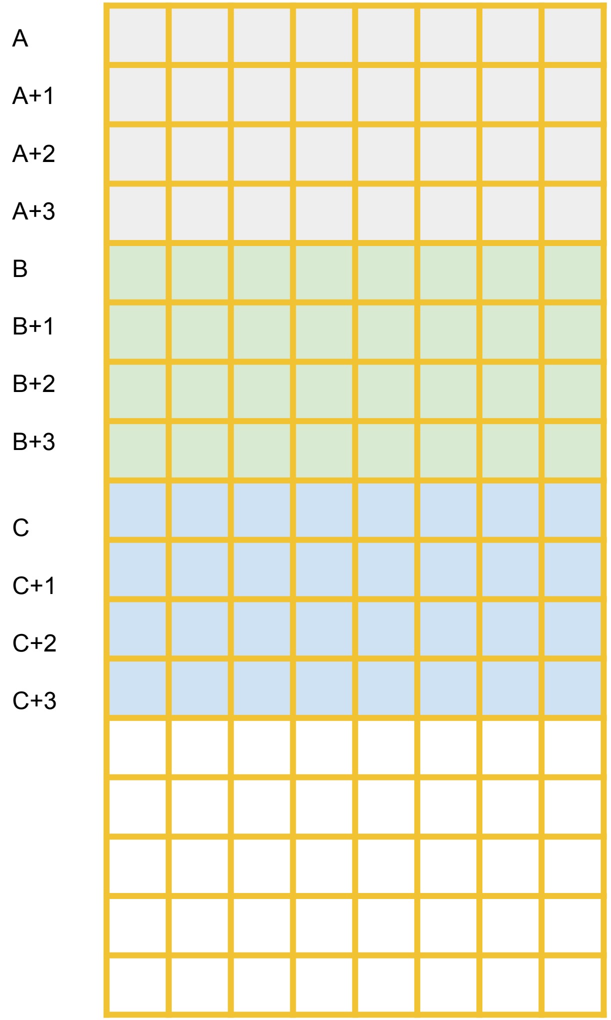
C program

```
#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

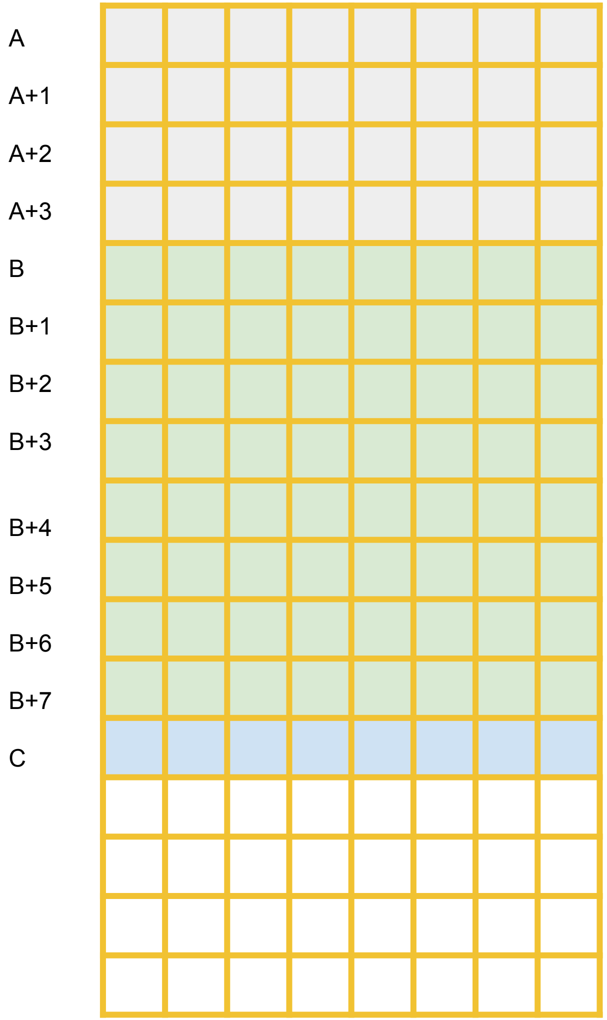
```
#include stdio.h
int A, B, C;
main() {
  A = 5;
  B = 7;
  C = A + B;
  printf(“A + B = %ld”; C);
}
```

$\longrightarrow$ Variable declaration



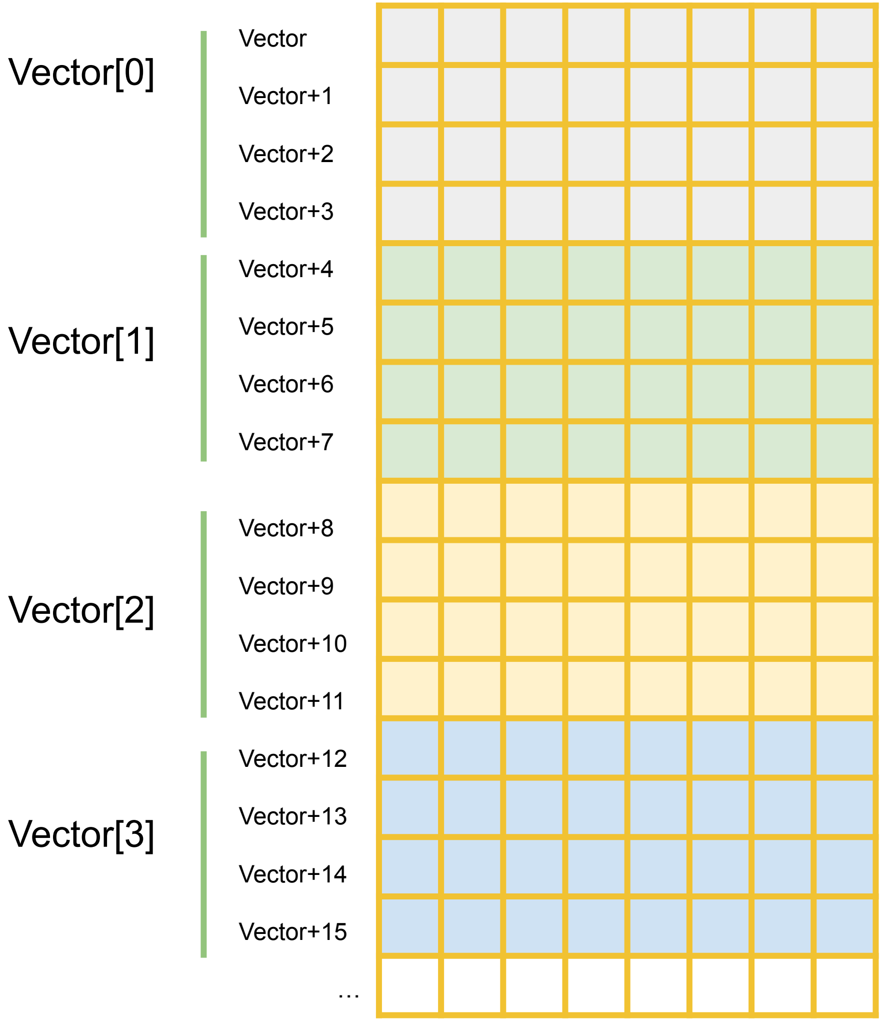
Variable types

```
int A;

double B;

char C;
```



Variable types

int Vector[10];



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)

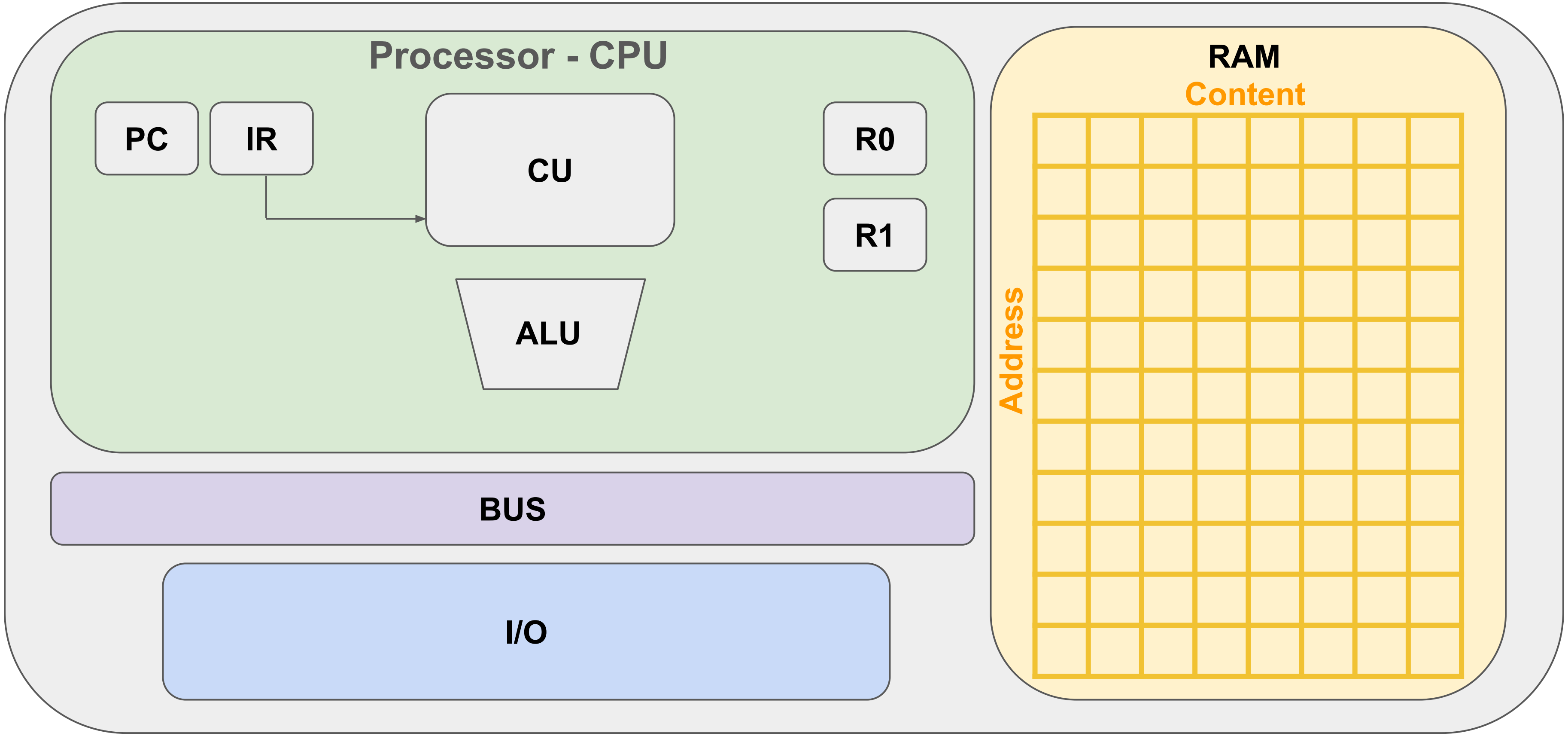
(sb, sw)

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 <- (Acc) + (SOURCE)

SUB, ADC, SBB, …
CMP SOURCE
```

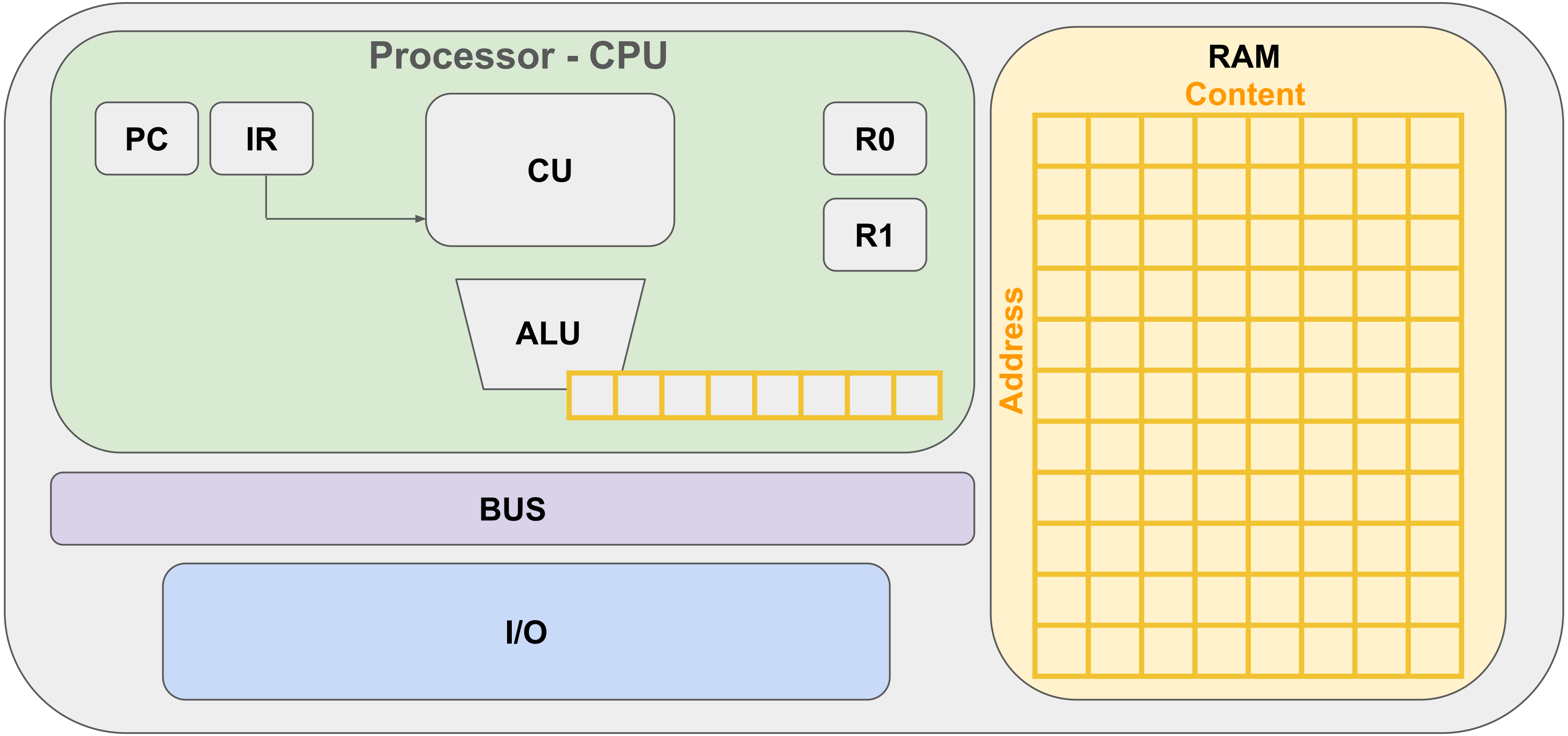
INTEL X86

```
ADD DESTINATION, SOURCE
DESTINATION <-
(DESTINATION)  + (SOURCE)

SUB, ADC, SBB, …
CMP Op1, Op2 (Op1 - Op2)
```

RISC-V

```
OP rd, rs1, rs2
   rd <- (rs1) OP (rs2)

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

Opi rd, rs1, imm
  rd <- (rs1) OP imm

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

The computer



Arithmetic and logical instructions

MOS 6502

ADD SOURCE
        Imm
        Mem
  Acc <- (Acc) + (SOURCE)

SUB, ADC, SBB, …
CMP SOURCE

~~MUL, DIV~~

INTEL X86

ADD DESTINATION, SOURCE
DESTINATION <-
(DESTINATION) + (SOURCE)

SUB, ADC, SBB, …
CMP Op1, Op2 (Op1 - Op2)

MUL Op1
  MUL BX
  DX:AX <- (AX) * (BX)

DIV Op1
  DIV BX
  DX:AX/BX
  Q(AX) i R(DX)

RISC-V

OP rd, rs1, rs2
   rd <- (rs1) OP (rs2)

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

Opi rd, rs1, imm
  rd <- (rs1) OP imm

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

Operate instructions

MOS 6502

INC SOURCE
- Mem
- Mem <- (Mem) + 1

DEC SOURCE
- Mem
- Mem <- (Mem) - 1

INX, DEX

INTEL X86

INC SOURCE
- Mem
- Reg

SOURCE <- (SOURCE) + 1

DEC SOURCE
- Mem
- Reg

SOURCE <- (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
```

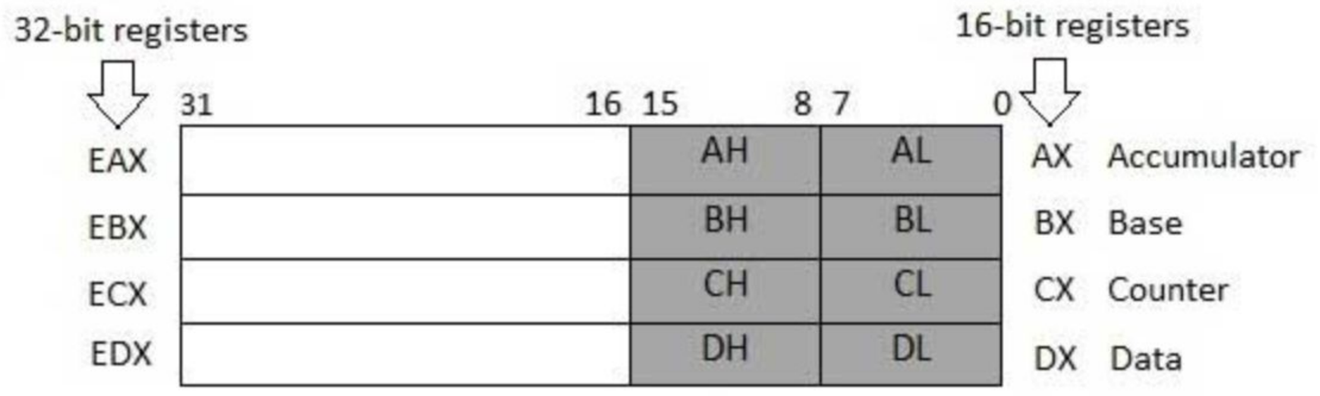
Intel Registers

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



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
```

int A, B, C, D

```
MOV R1, [C]
MUL R1, [D]
ADD R1, [B]
MOV [A], R1
```

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

           MOV R1, [A]
           MOV R2, [B]
while:     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

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

result = 1;
iter = 1;

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

```asm
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
```

Opction 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 SEGUENT
6.              MOV R1, 0
7. ELSE:
8. SEGUENT:
```

COMPUTER STRUCTURE (I)

Computer Architecture and Operating Systems