# Introduction to Programming

Week – *0*, Lecture – *2*

## Operating System Basics

SAURABH SRIVASTAVA

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

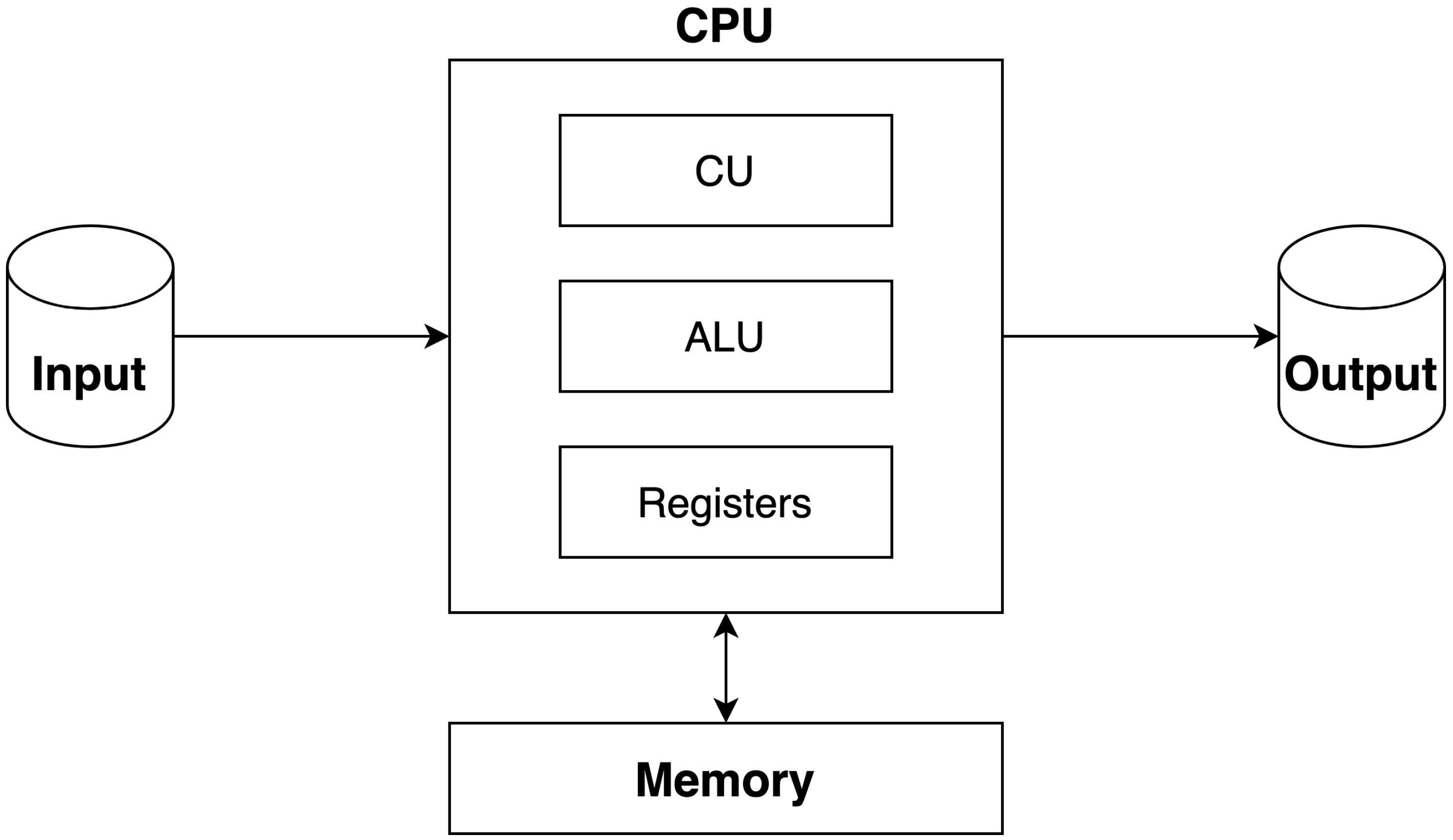
IIT KANPUR

# Revision

So you now know the basic elements of a Computer

- CPU, Input Devices, Output Devices and a Memory Element

# von Neumann Architecture



# Revision

So you now know the basic elements of a Computer

- CPU, Input Devices, Output Devices and a Memory Element

But think about how tedious it will be to use these electronic components

- We are talking about 0s and 1s – that too, lots of them !!

# Revision

So you now know the basic elements of a Computer

- CPU, Input Devices, Output Devices and a Memory Element

But think about how tedious it will be to use these electronic components

- We are talking about 0s and 1s – that too, lots of them !!

Modern Computers allow you to put an assistant for yourself which does this hard work

- We call this assistant the **Operating System** or **OS** – because it operates the “hardware” on our behalf
- Hardware is just a glorified term for all the underlying electronic components

# Revision

So you now know the basic elements of a Computer

- CPU, Input Devices, Output Devices and a Memory Element

But think about how tedious it will be to use these electronic components

- We are talking about 0s and 1s – that too, lots of them !!

Modern Computers allow you to put an assistant for yourself which does this hard work

- We call this assistant the **Operating System** or **OS** – because it operates the “hardware” on our behalf
- Hardware is just a glorified term for all the underlying electronic components

The Operating System adds a layer over the hardware

- You can talk to the Operating System, asking it to get the computations done from the CPU
- It also manages your Memory element – we usually call it the **Main Memory**

# Main Memory

Although not technically accurate, the colloquial term for Main Memory is “RAM”

# Main Memory

Although not technically accurate, the colloquial term for Main Memory is “RAM”

The Main Memory is where the Operating System puts instructions for the CPU to execute

# Main Memory

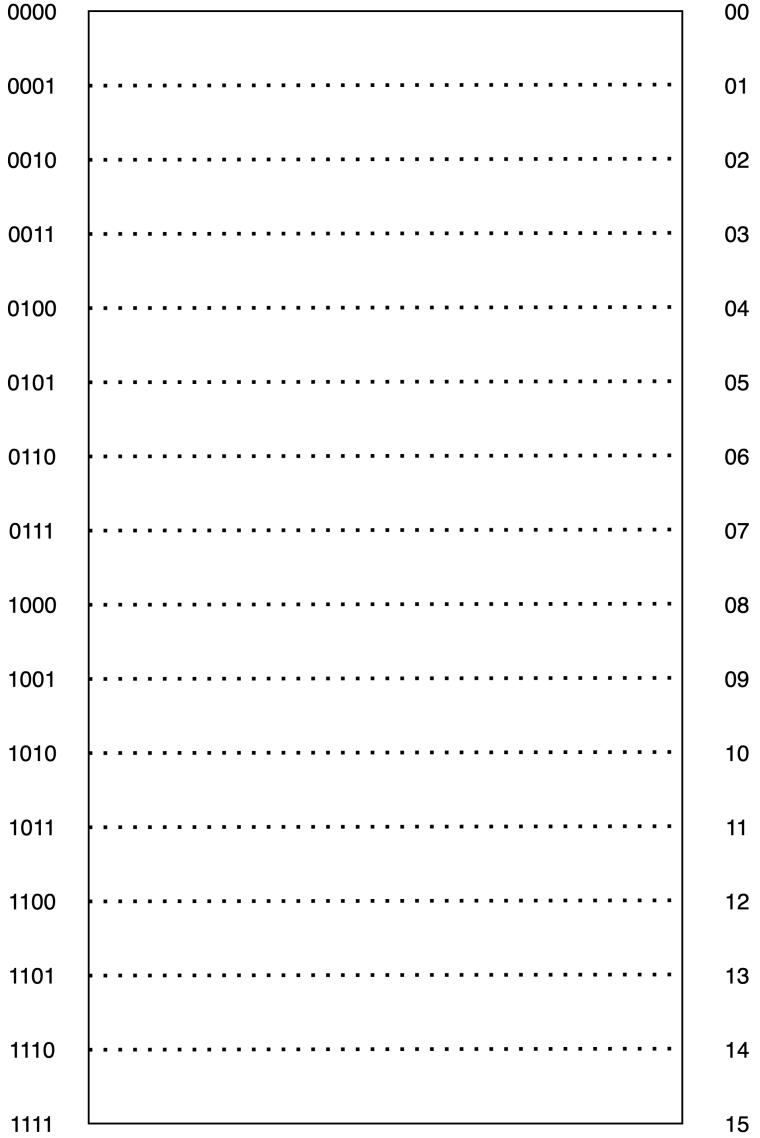
Although not technically accurate, the colloquial term for Main Memory is “RAM”

The Main Memory is where the Operating System puts instructions for the CPU to execute

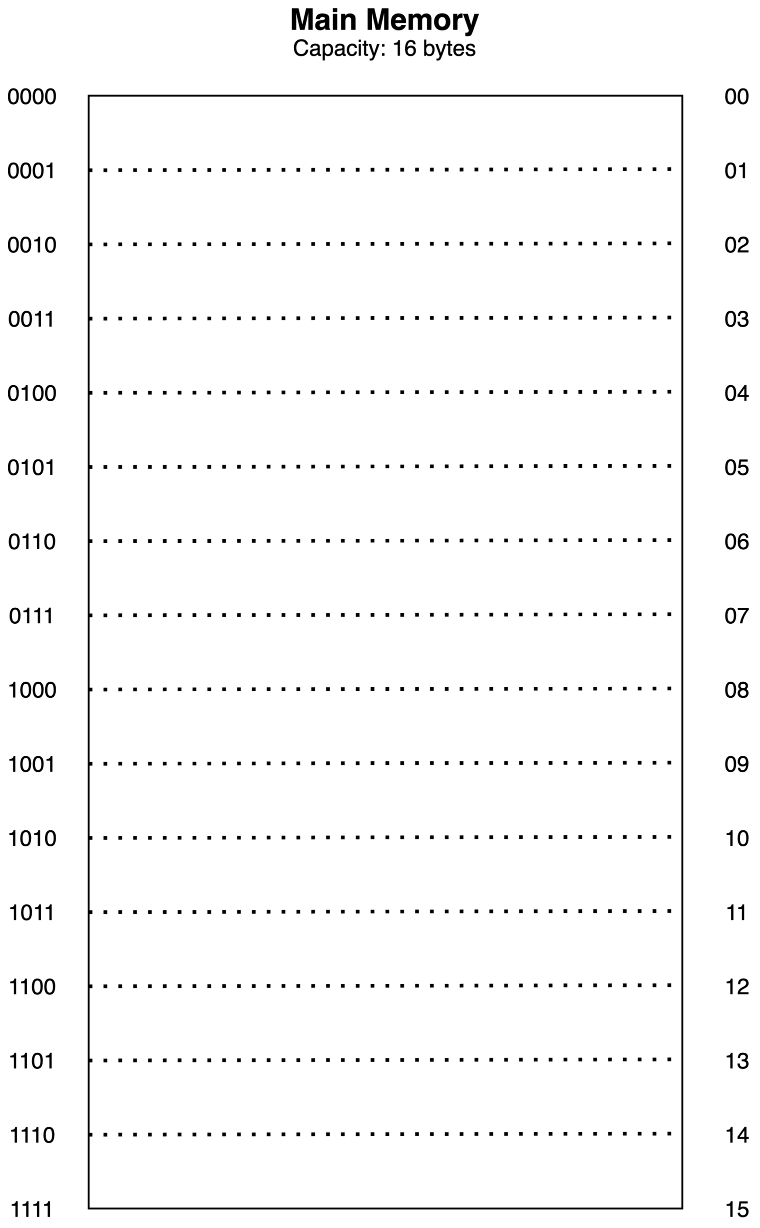
Any data, that is required to execute the instructions are also kept in the Main Memory

## Main Memory

Capacity: 16 bytes



| | |
|---|---|
| 0000 | 00 |
| 0001 | 01 |
| 0010 | 02 |
| 0011 | 03 |
| 0100 | 04 |
| 0101 | 05 |
| 0110 | 06 |
| 0111 | 07 |
| 1000 | 08 |
| 1001 | 09 |
| 1010 | 10 |
| 1011 | 11 |
| 1100 | 12 |
| 1101 | 13 |
| 1110 | 14 |
| 1111 | 15 |

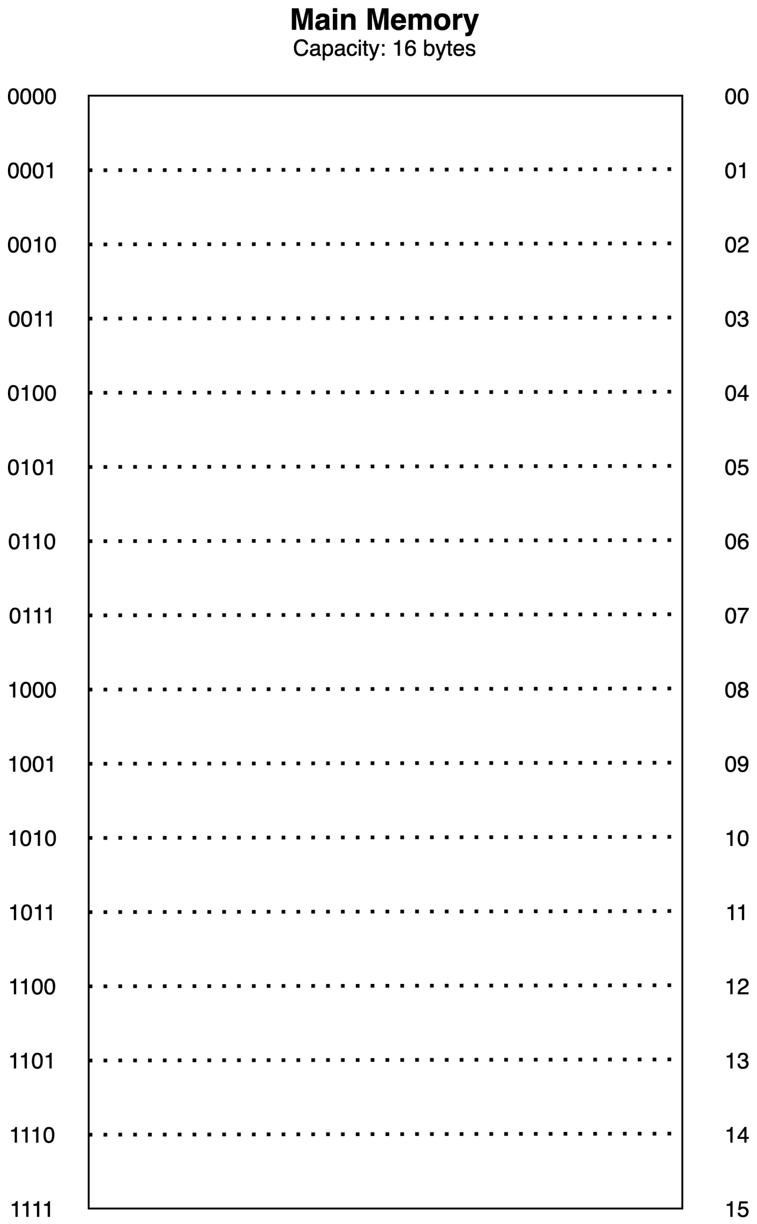


**Main Memory**

Capacity: 16 bytes

| | |
|---|---|
| 0000 | 00 |
| 0001 | 01 |
| 0010 | 02 |
| 0011 | 03 |
| 0100 | 04 |
| 0101 | 05 |
| 0110 | 06 |
| 0111 | 07 |
| 1000 | 08 |
| 1001 | 09 |
| 1010 | 10 |
| 1011 | 11 |
| 1100 | 12 |
| 1101 | 13 |
| 1110 | 14 |
| 1111 | 15 |

This is how the Main Memory looks like



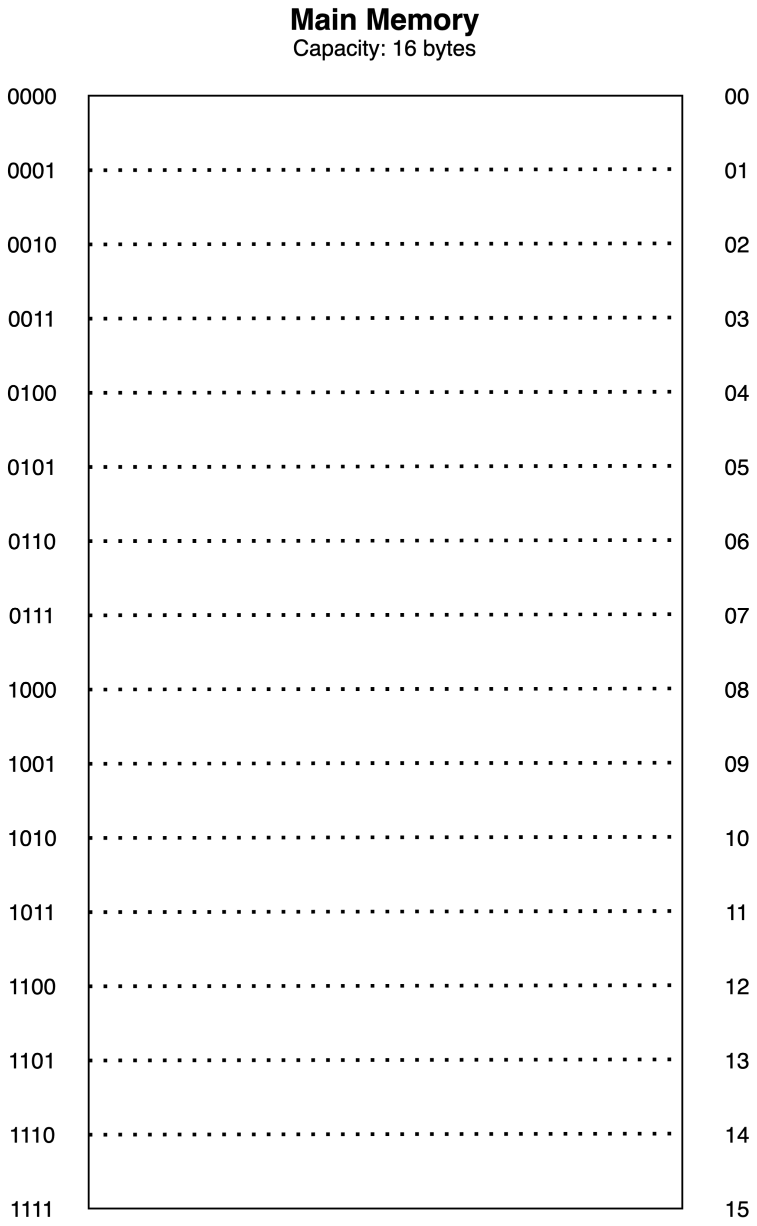
**Main Memory**

Capacity: 16 bytes

| | |
|---|---|
| 0000 | 00 |
| 0001 | 01 |
| 0010 | 02 |
| 0011 | 03 |
| 0100 | 04 |
| 0101 | 05 |
| 0110 | 06 |
| 0111 | 07 |
| 1000 | 08 |
| 1001 | 09 |
| 1010 | 10 |
| 1011 | 11 |
| 1100 | 12 |
| 1101 | 13 |
| 1110 | 14 |
| 1111 | 15 |

This is how the Main Memory looks like

Each Memory Location has an *address* (You can see the addresses here in both binary and decimal)



**Main Memory**

Capacity: 16 bytes

| | |
|---|---|
| 0000 | 00 |
| 0001 | 01 |
| 0010 | 02 |
| 0011 | 03 |
| 0100 | 04 |
| 0101 | 05 |
| 0110 | 06 |
| 0111 | 07 |
| 1000 | 08 |
| 1001 | 09 |
| 1010 | 10 |
| 1011 | 11 |
| 1100 | 12 |
| 1101 | 13 |
| 1110 | 14 |
| 1111 | 15 |

This is how the Main Memory looks like

Each memory location can store 1 or more bytes – we call this the *word length* (as of now, assume the length to be 1 byte)

Each Memory Location has an *address* (You can see the addresses here in both binary and decimal)

# Main Memory

Although not technically accurate, the colloquial term for Main Memory is “RAM”

The Main Memory is where the Operating System puts instructions for the CPU to execute

Any data, that is required to execute the instructions are also kept in the Main Memory

The word length is dependent on the processor, and the number of wires in the *buses*

- Don’t worry much about this as of now, you will study about Word lengths in Computer Organisation
- For now, just assume that every address in the memory can store 1 byte !!

# Main Memory

Although not technically accurate, the colloquial term for Main Memory is “RAM”

The Main Memory is where the Operating System puts instructions for the CPU to execute

Any data, that is required to execute the instructions are also kept in the Main Memory

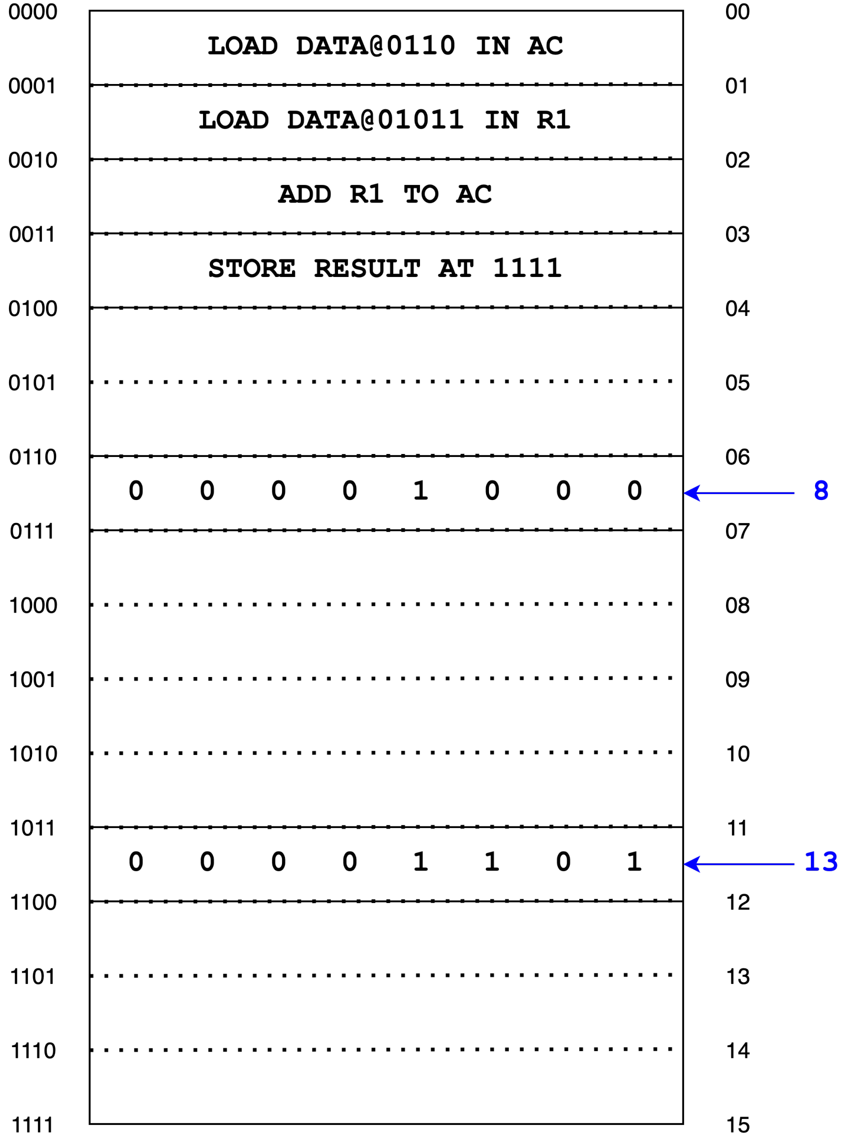
The word length is dependent on the processor, and the number of wires in the *buses*

- Don’t worry much about this as of now, you will study about Word lengths in Computer Organisation
- For now, just assume that every address in the memory can store 1 byte !!

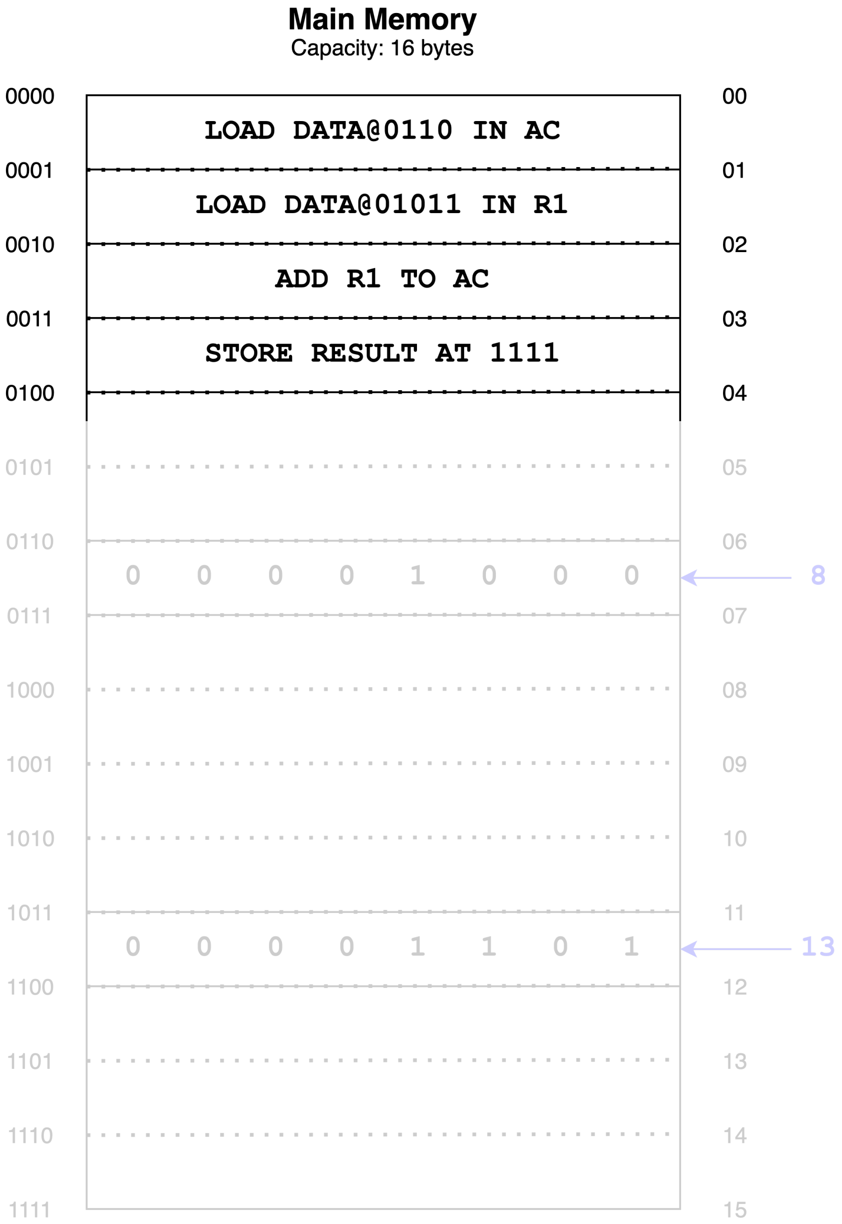
We can store both instructions, as well as data in the Main Memory

## Main Memory

Capacity: 16 bytes



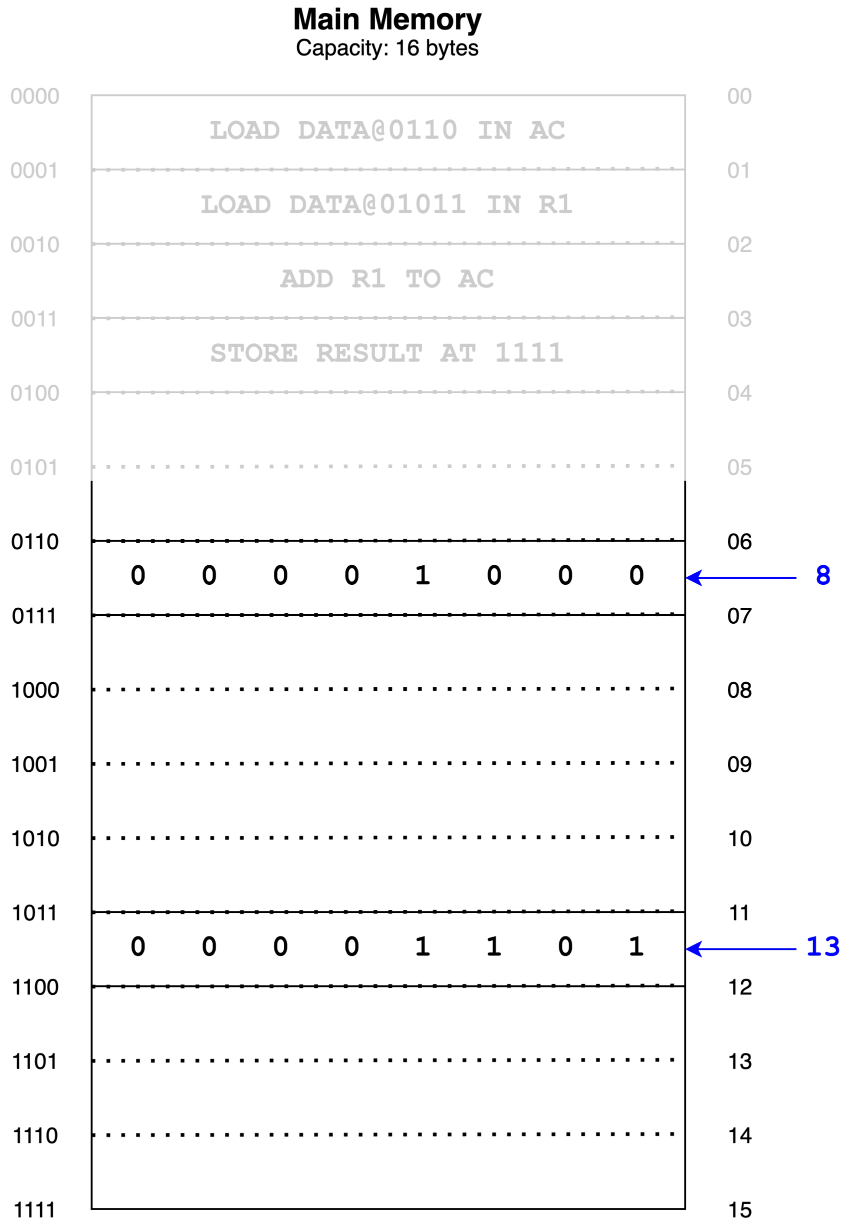
| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |



**Main Memory**
Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |

Some part of the memory has instructions



**Main Memory**
Capacity: 16 bytes

| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 | 06 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 | 11 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | | 14 |
| 1111 | | 15 |

Some part of the memory has instructions

While other parts carry data

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

Remember that we were going to instruct the Control Unit to perform computations?

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

Remember that we were going to instruct the Control Unit to perform computations?

- Let us take the example of Addition

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

Remember that we were going to instruct the Control Unit to perform computations?

- Let us take the example of Addition
- We want the Control Unit to add two numbers – `8` and `13`

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

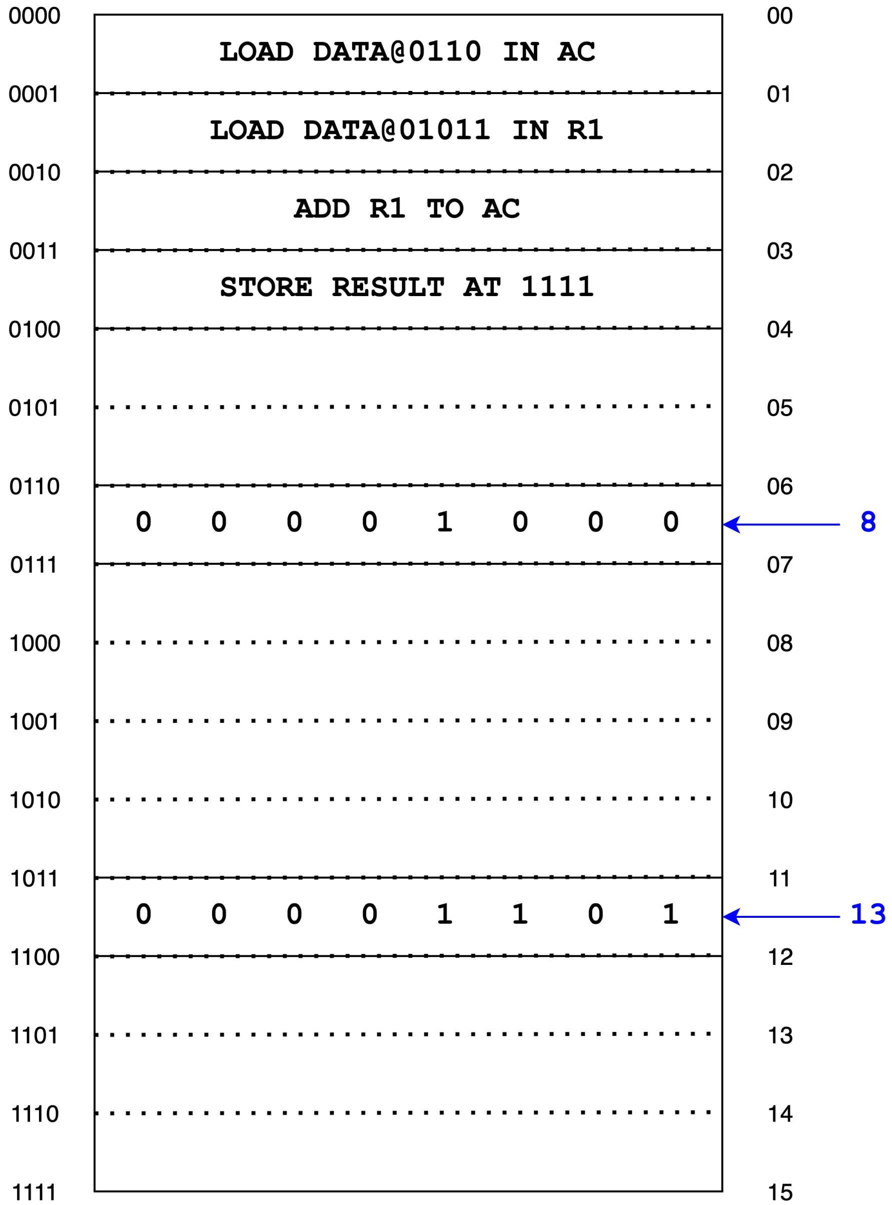
Remember that we were going to instruct the Control Unit to perform computations?

- Let us take the example of Addition
- We want the Control Unit to add two numbers – `8` and `13`

How can we arrange the relevant instructions and data in the Memory?

## Main Memory

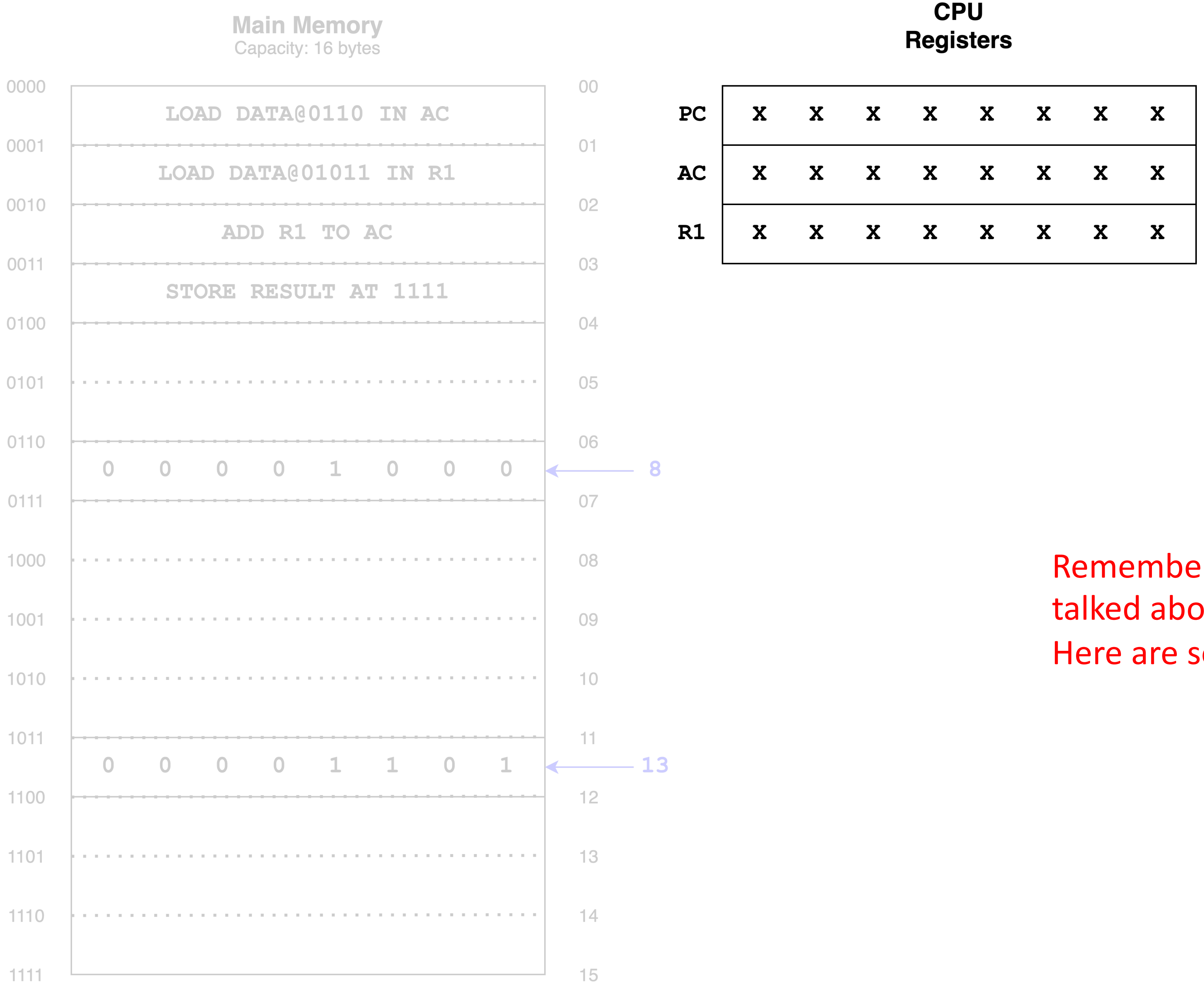
Capacity: 16 bytes



| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |

## CPU Registers

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | X | X | X | X | X | X | X | X |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |



## Main Memory

Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | ← 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | ← 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |

## CPU Registers

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | X | X | X | X | X | X | X | X |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |

Remember the *Registers* we talked about?

Here are some examples !!

## Main Memory

Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | ← 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | ← 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |

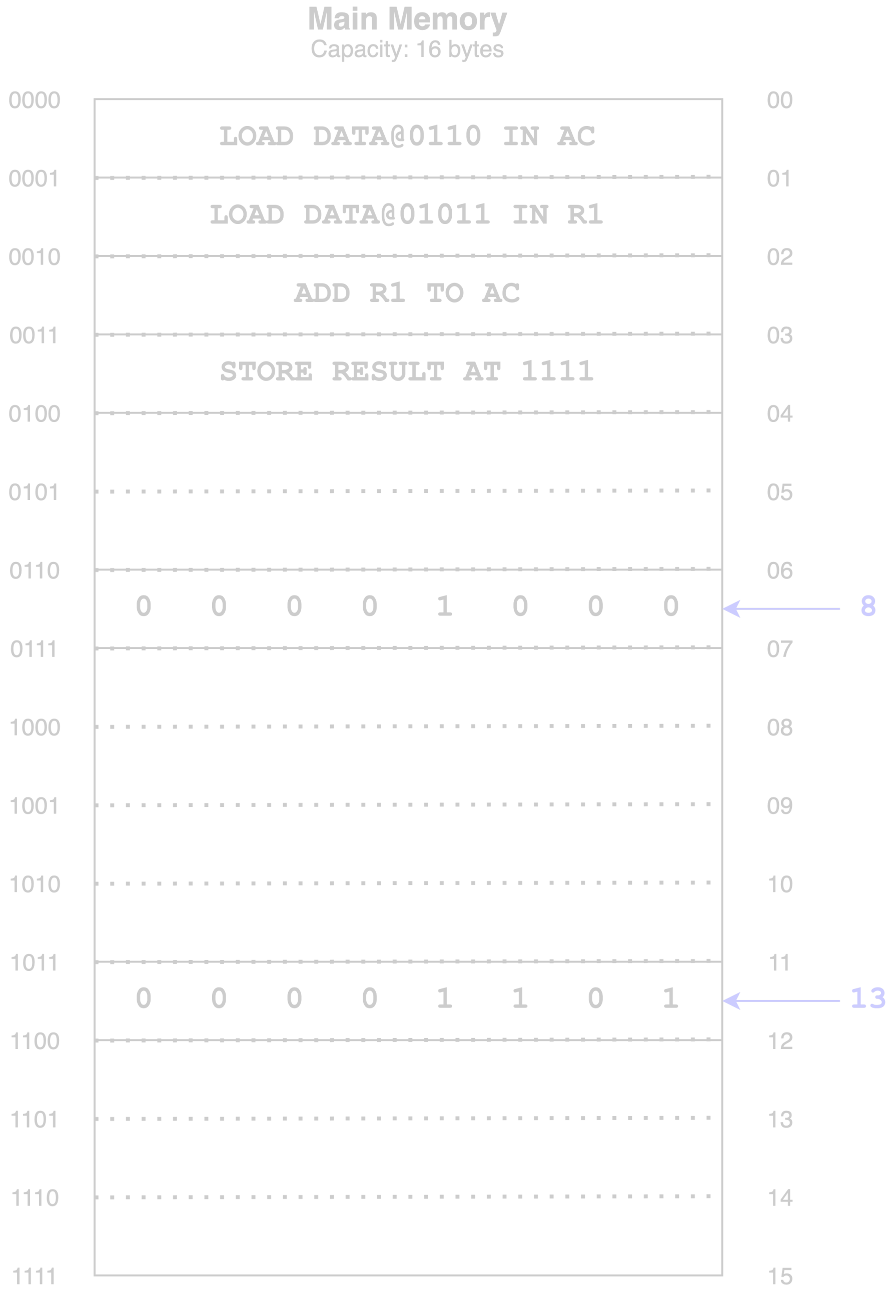
## CPU Registers

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | X | X | X | X | X | X | X | X |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |

Remember the *Registers* we talked about?

Here are some examples !!

There are two special Registers named PC and AC here



Main Memory
Capacity: 16 bytes

| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 | 06 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 | 11 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | | 14 |
| 1111 | | 15 |

**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | X | X | X | X | X | X | X | X |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |

Remember the *Registers* we talked about?
Here are some examples !!

There are two special Registers named PC and AC here

… and there can be others, called R1, R2, R3 and so on

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

Remember that we were going to instruct the Control Unit to perform computations?

- Let us take the example of Addition
- We want the Control Unit to add two numbers – `8` and `13`

AC stands for Accumulator Register

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

Remember that we were going to instruct the Control Unit to perform computations?

- Let us take the example of Addition
- We want the Control Unit to add two numbers – `8` and `13`

AC stands for Accumulator Register

- It is the primary register that ALU uses for its operations

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

Remember that we were going to instruct the Control Unit to perform computations?

- Let us take the example of Addition
- We want the Control Unit to add two numbers – `8` and `13`

AC stands for Accumulator Register

- It is the primary register that ALU uses for its operations
- For any single operand operations, the Control Unit simply stores the operand in AC

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

Remember that we were going to instruct the Control Unit to perform computations?

- Let us take the example of Addition
- We want the Control Unit to add two numbers – `8` and `13`

AC stands for Accumulator Register

- It is the primary register that ALU uses for its operations
- For any single operand operations, the Control Unit simply stores the operand in AC
- … and instructs the ALU to perform the operation

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

Remember that we were going to instruct the Control Unit to perform computations?

- Let us take the example of Addition
- We want the Control Unit to add two numbers – `8` and `13`

AC stands for Accumulator Register

- It is the primary register that ALU uses for its operations
- For any single operand operations, the Control Unit simply stores the operand in AC
- … and instructs the ALU to perform the operation
- For two operand operations, ALU can use one of the registers R1, R2 etc.

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

Remember that we were going to instruct the Control Unit to perform computations?

- Let us take the example of Addition
- We want the Control Unit to add two numbers – `8` and `13`

AC stands for Accumulator Register

- It is the primary register that ALU uses for its operations
- For any single operand operations, the Control Unit simply stores the operand in AC
- … and instructs the ALU to perform the operation
- For two operand operations, ALU can use one of the registers R1, R2 etc.

PC stands for Program Counter

# Revisiting CPU – A sample program

Let us revisit our CPU again as well

Remember that we were going to instruct the Control Unit to perform computations?

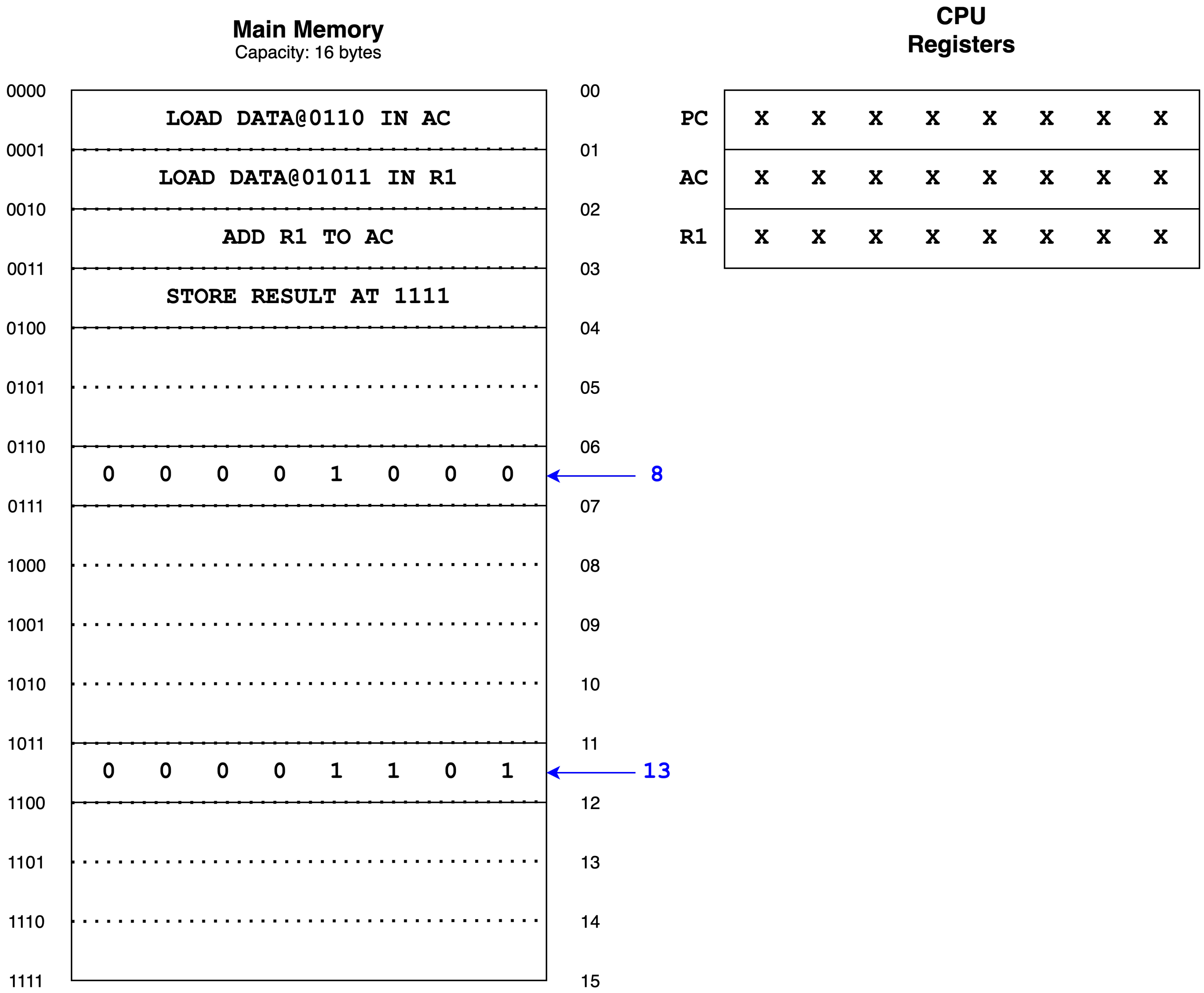
- Let us take the example of Addition
- We want the Control Unit to add two numbers – `8` and `13`

AC stands for Accumulator Register

- It is the primary register that ALU uses for its operations
- For any single operand operations, the Control Unit simply stores the operand in AC
- … and instructs the ALU to perform the operation
- For two operand operations, ALU can use one of the registers R1, R2 etc.

PC stands for Program Counter

- It stores the address of *the next instruction to execute* – let us walk through an example to understand

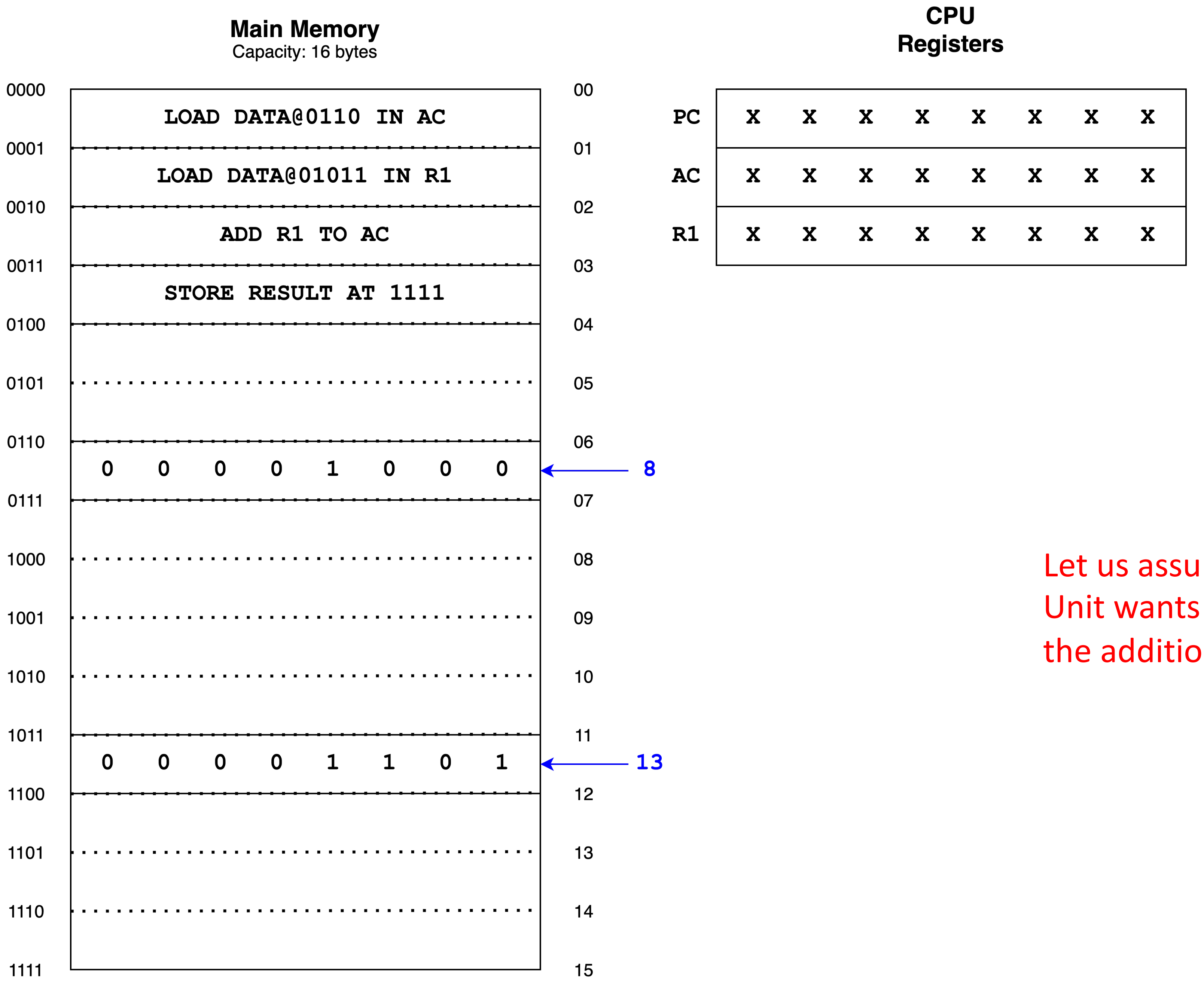


**Main Memory**
Capacity: 16 bytes

| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 | 06 ← 8 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 | 11 ← 13 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | | 14 |
| 1111 | | 15 |

**CPU**
**Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | X | X | X | X | X | X | X | X |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |



**Main Memory**

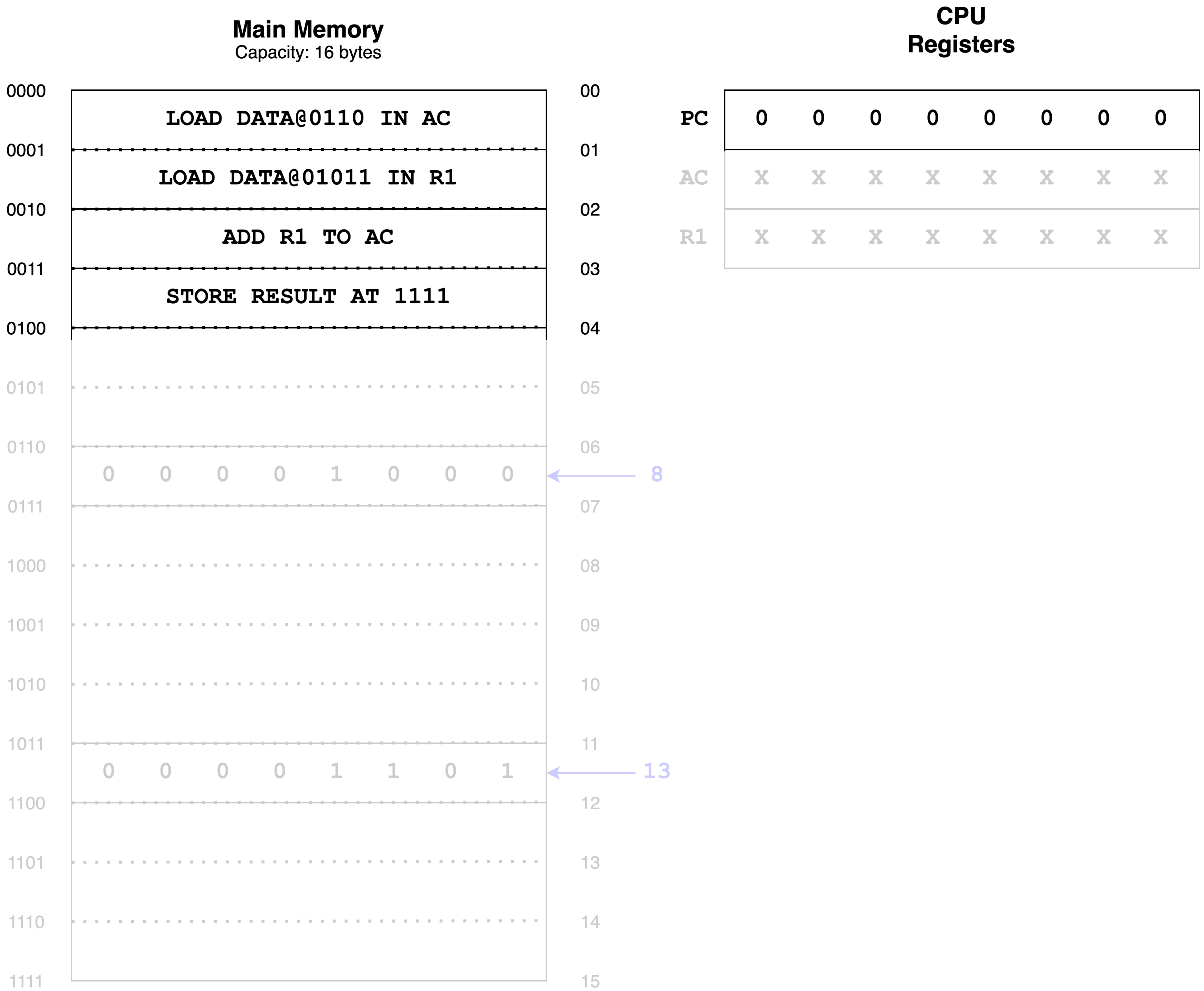
Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |

**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | X | X | X | X | X | X | X | X |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |

Let us assume that the Control Unit wants to use the ALU for the addition operation



**Main Memory**
Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |

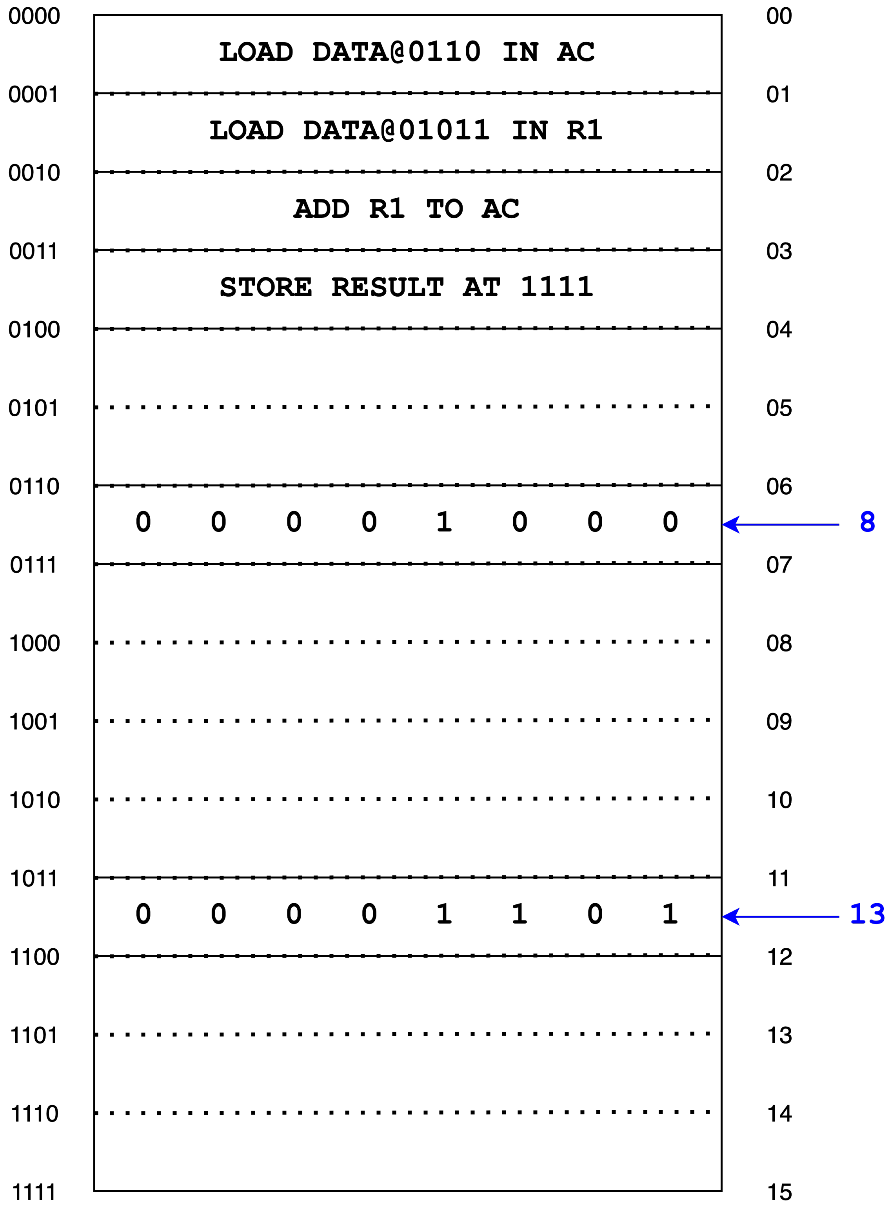
**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |

The first step is to set the value of PC to the beginning of the "addition program"

**Main Memory**

Capacity: 16 bytes



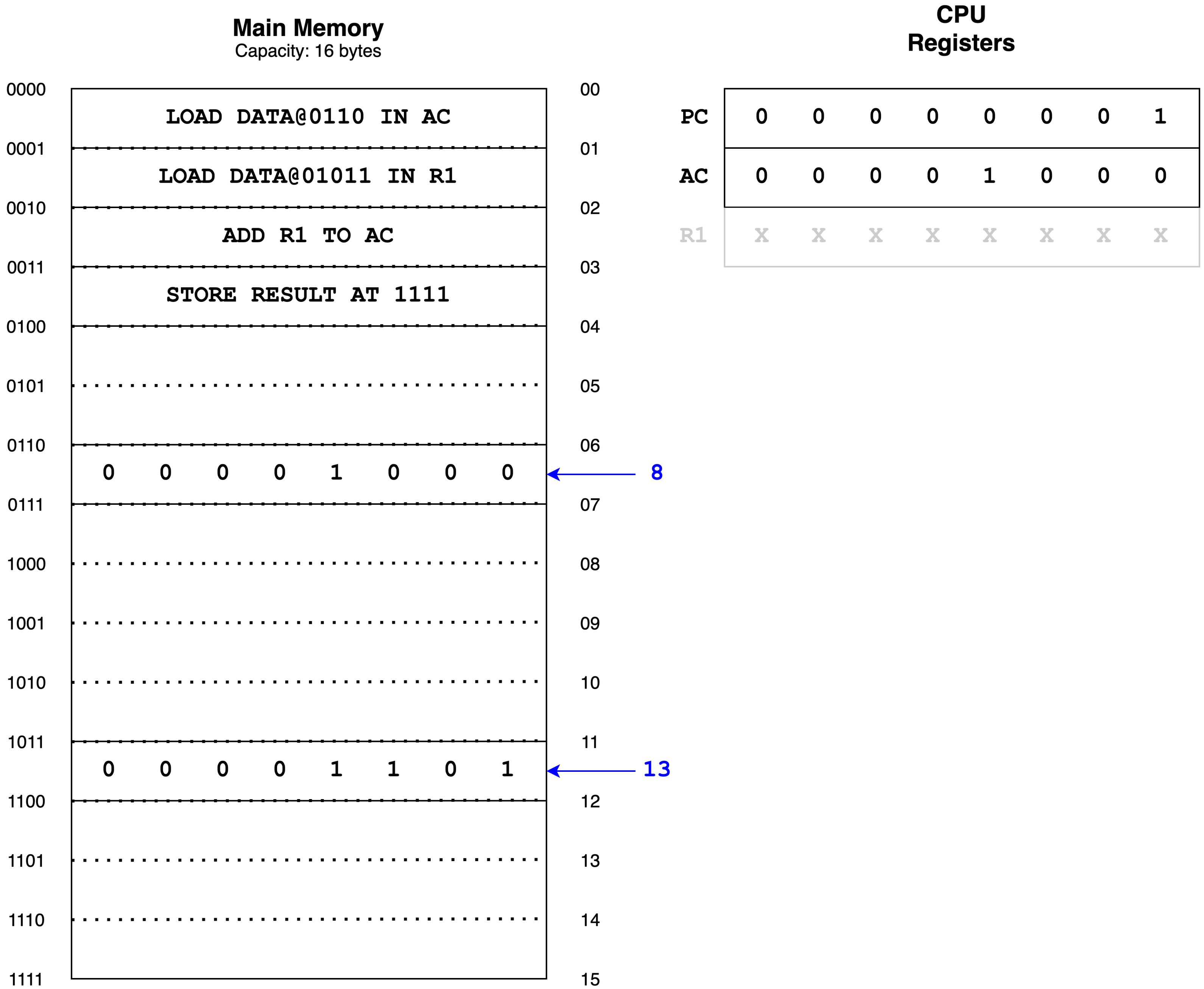
| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 | 06 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 | 11 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | | 14 |
| 1111 | | 15 |

**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |

The CU reads the instruction at this location in the Memory

The instruction says –
”load the data stored at address `0110` into the AC register”



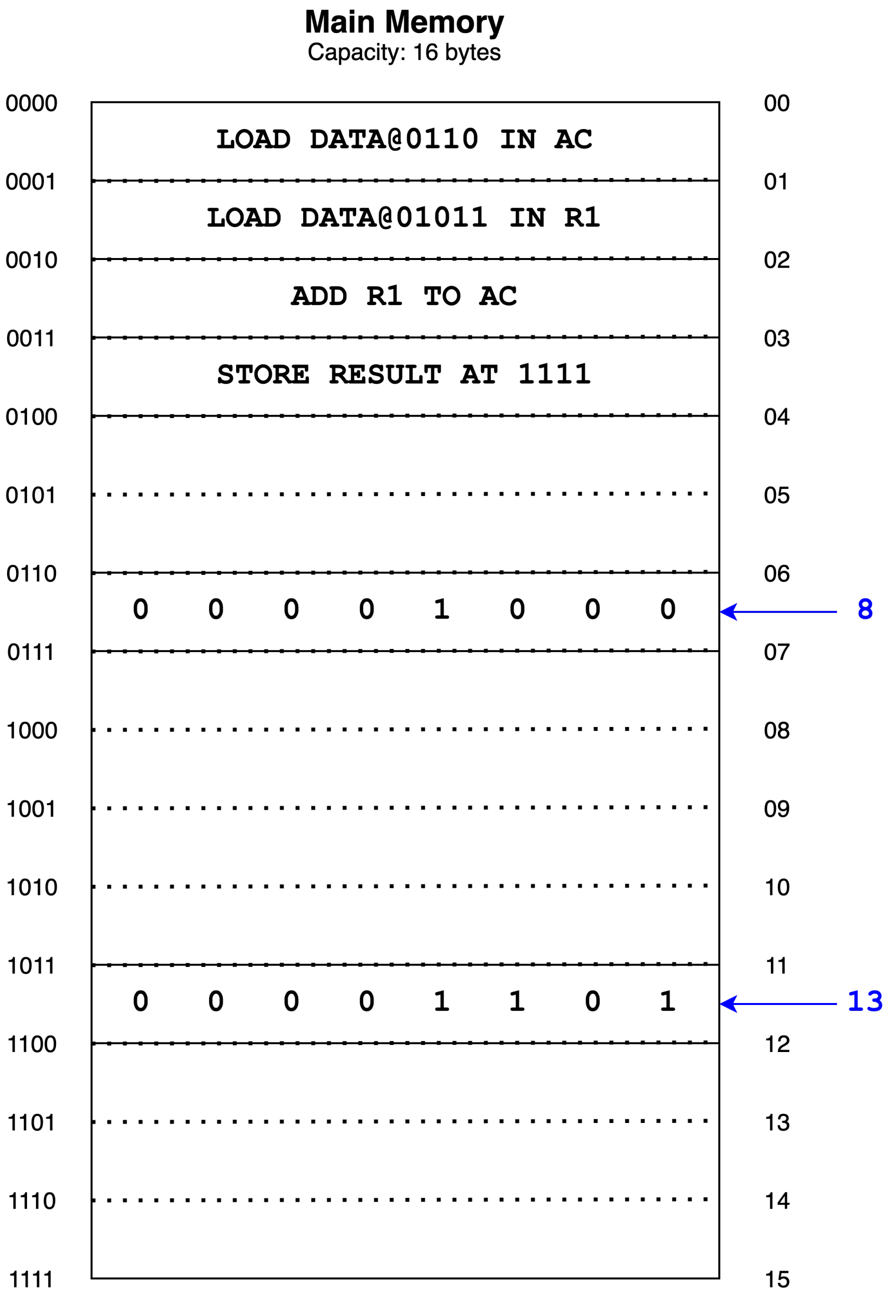
**Main Memory**
Capacity: 16 bytes

| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 ← 8 | 06 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 ← 13 | 11 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | | 14 |
| 1111 | | 15 |

**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| AC | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| R1 | X | X | X | X | X | X | X | X |

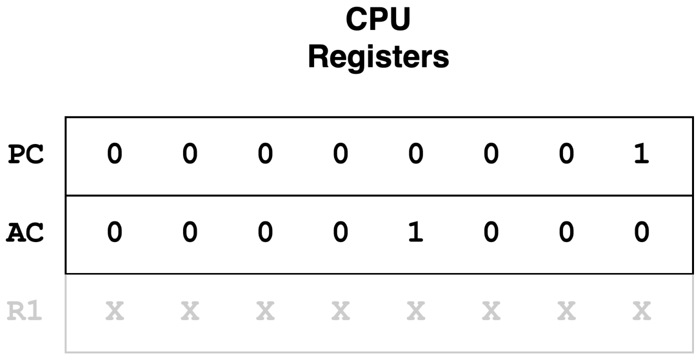
After this instruction gets executed, AC contains the data from the `0110` location



**Main Memory**

Capacity: 16 bytes

| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 | 06 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 | 11 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | | 14 |
| 1111 | | 15 |

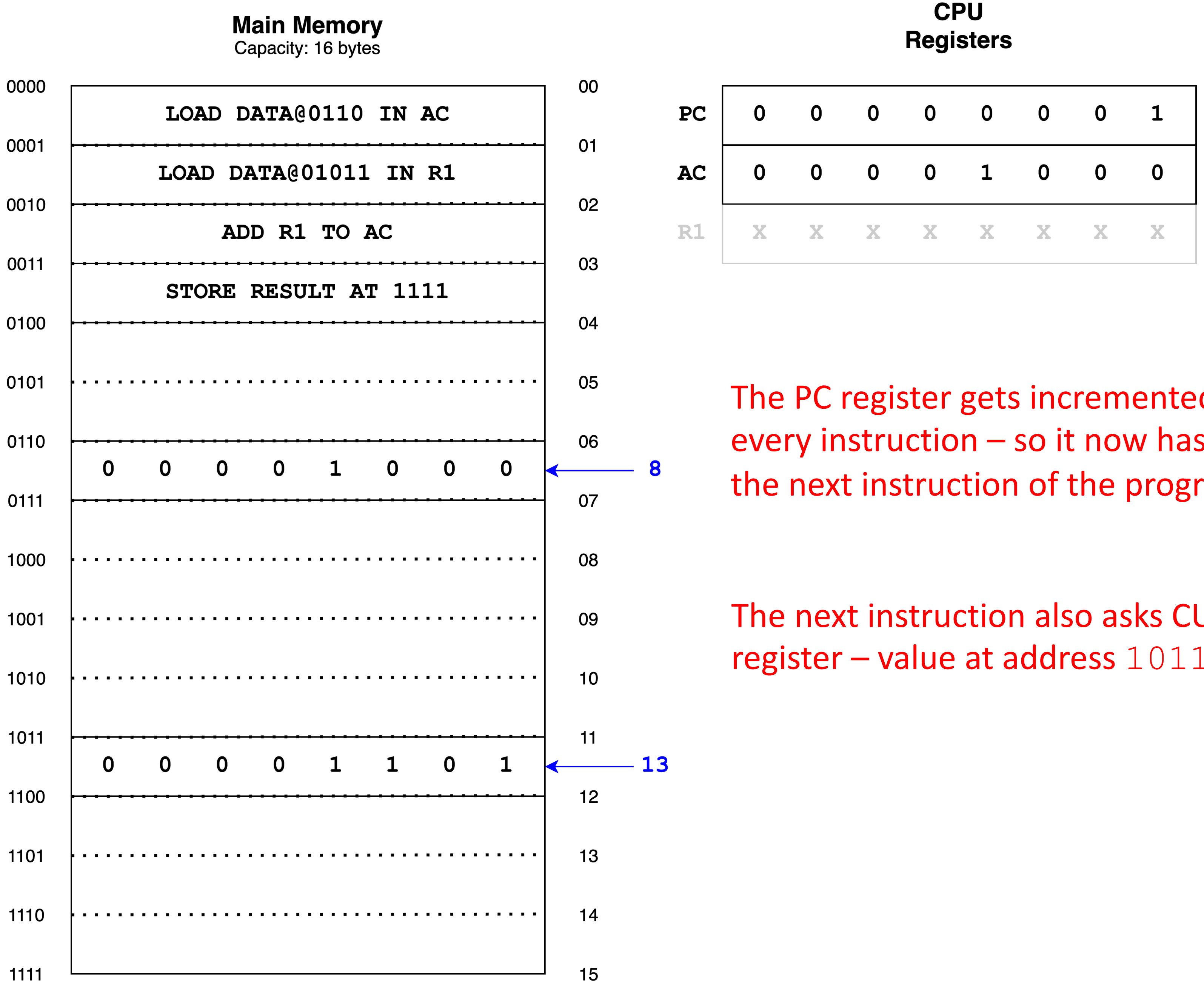


**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| AC | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| R1 | X | X | X | X | X | X | X | X |

After this instruction gets executed, AC contains the data from the 0110 location

The PC register gets incremented by 1 after execution of every instruction – so it now has a value which points to the next instruction of the program



**Main Memory**
Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |

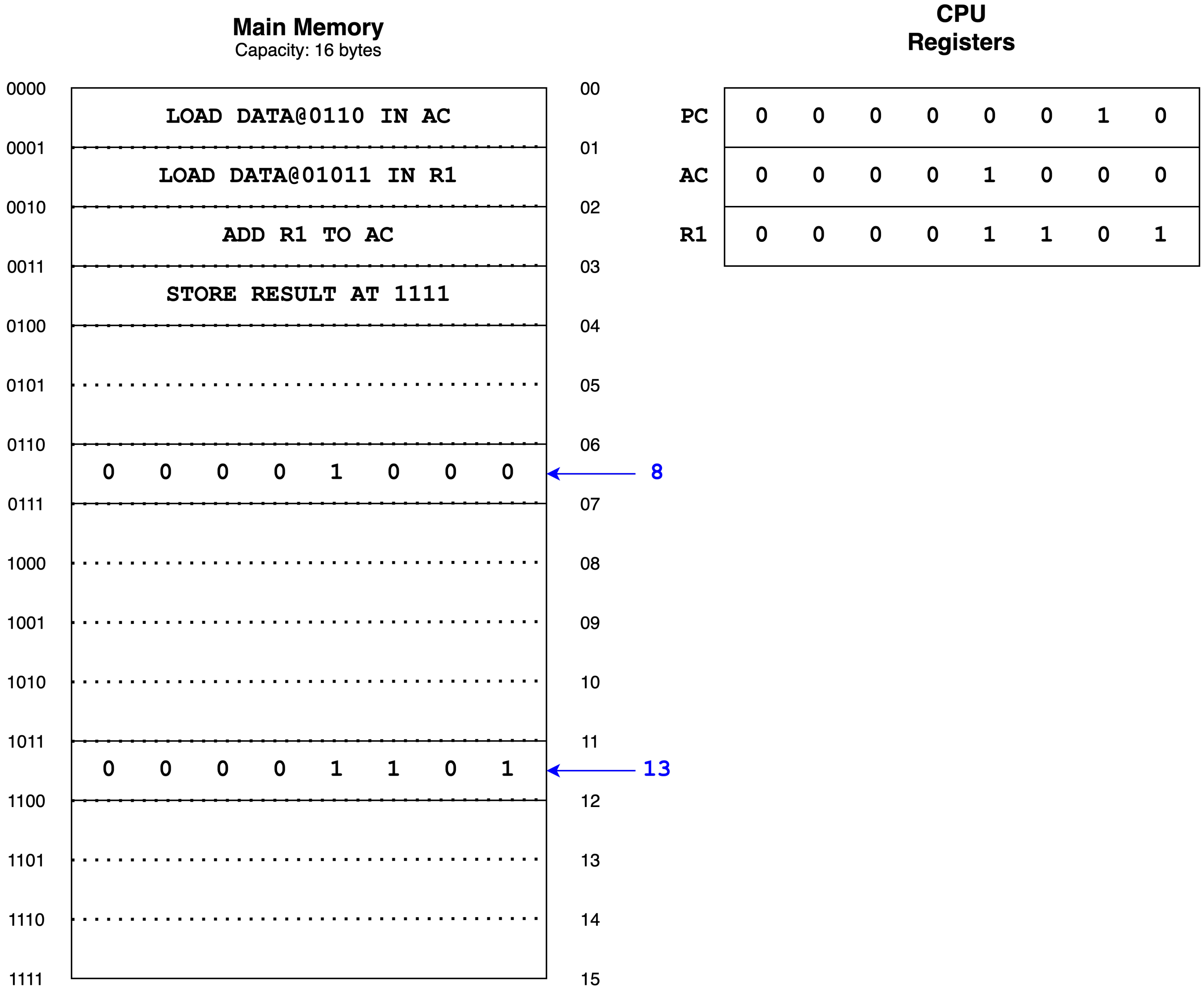
**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| AC | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| R1 | X | X | X | X | X | X | X | X |

After this instruction gets executed, AC contains the data from the `0110` location

The PC register gets incremented by 1 after execution of every instruction – so it now has a value which points to the next instruction of the program

The next instruction also asks CU to load a value in a register – value at address `1011` into register R1



**Main Memory**

Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | ← 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | ← 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |

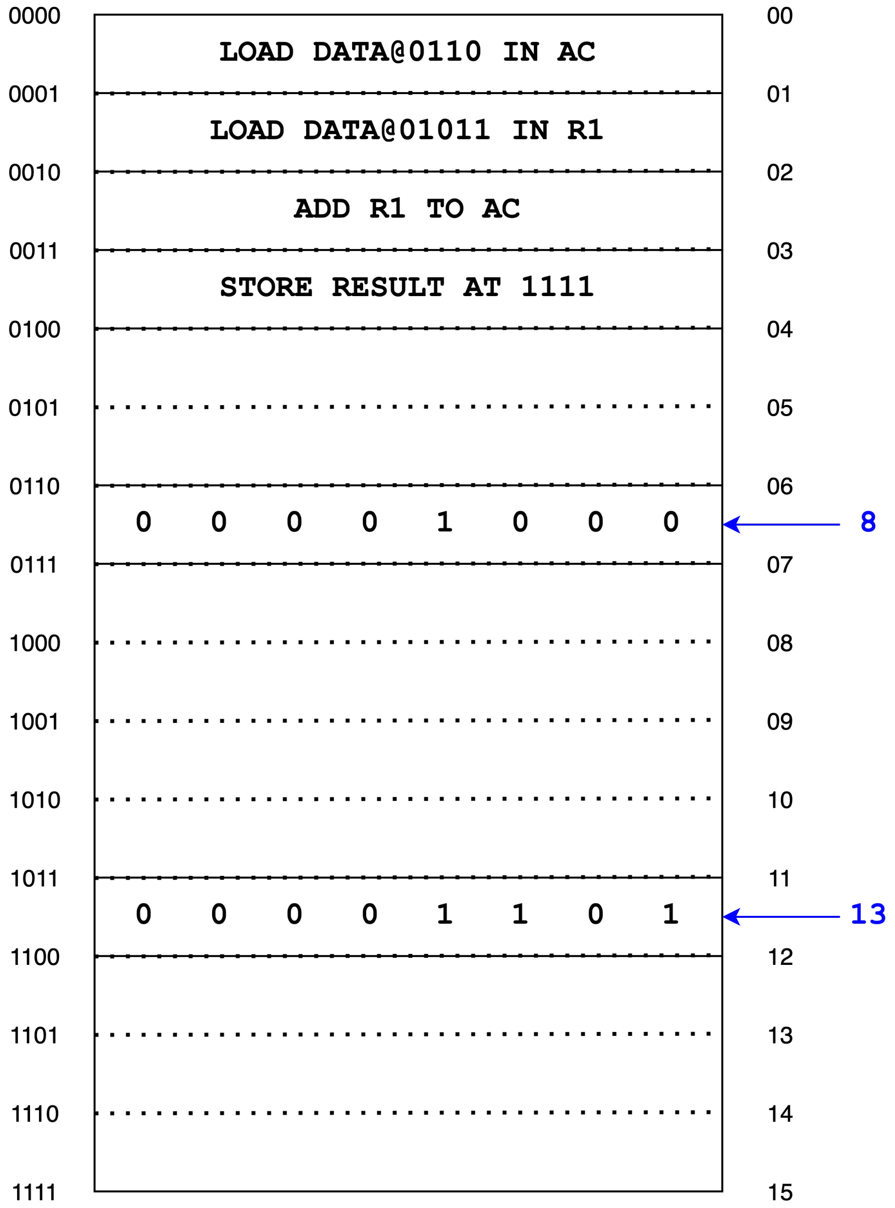
**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| AC | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| R1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

The other operand for addition is now in another register – R1

**Main Memory**

Capacity: 16 bytes



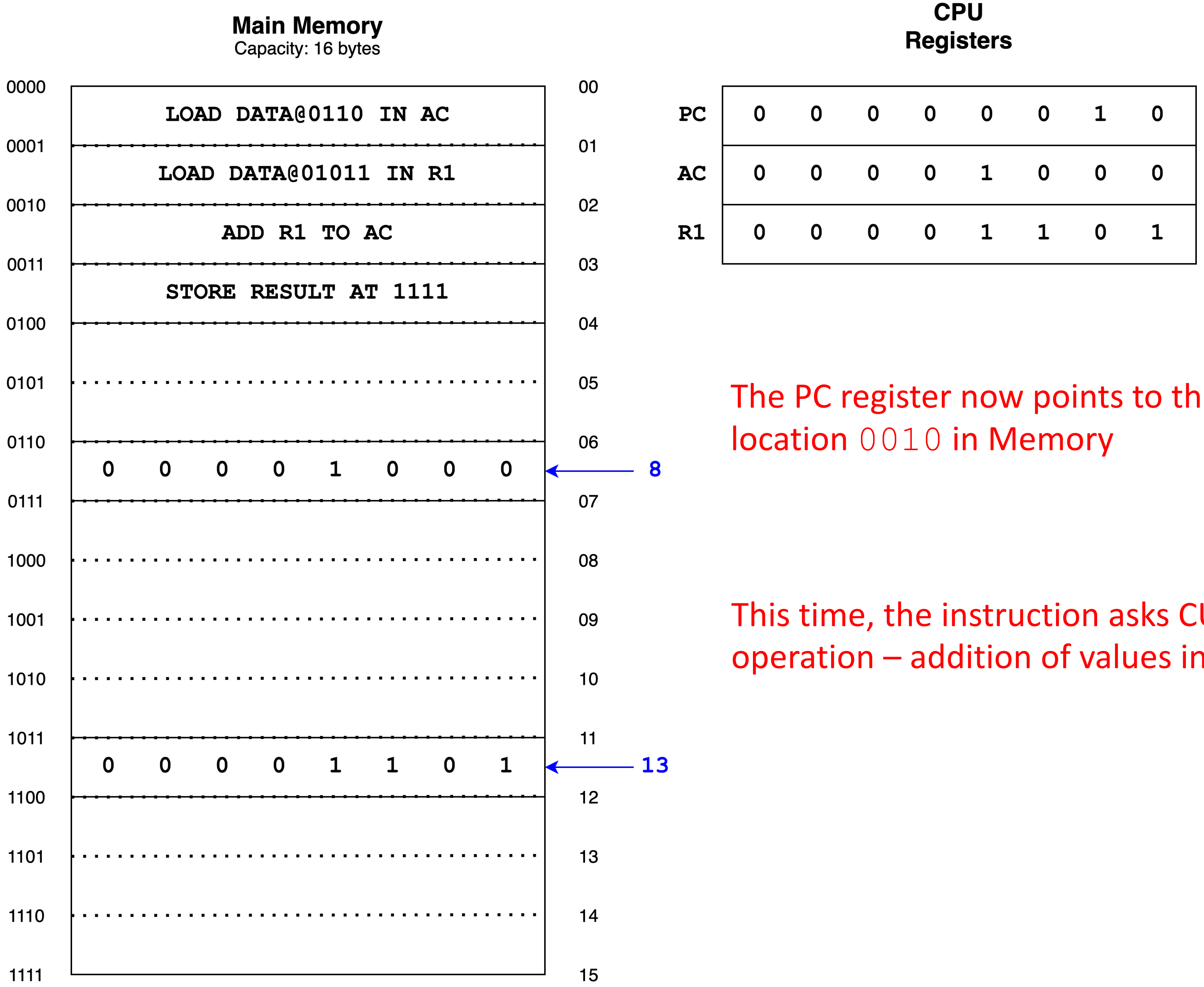
| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 ← 8 | 06 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 ← 13 | 11 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | | 14 |
| 1111 | | 15 |

**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| AC | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| R1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

The other operand for addition is now in another register – R1

The PC register now points to the next instruction – at location `0010` in Memory



**Main Memory**
Capacity: 16 bytes

| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 ← 8 | 06 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 ← 13 | 11 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | | 14 |
| 1111 | | 15 |

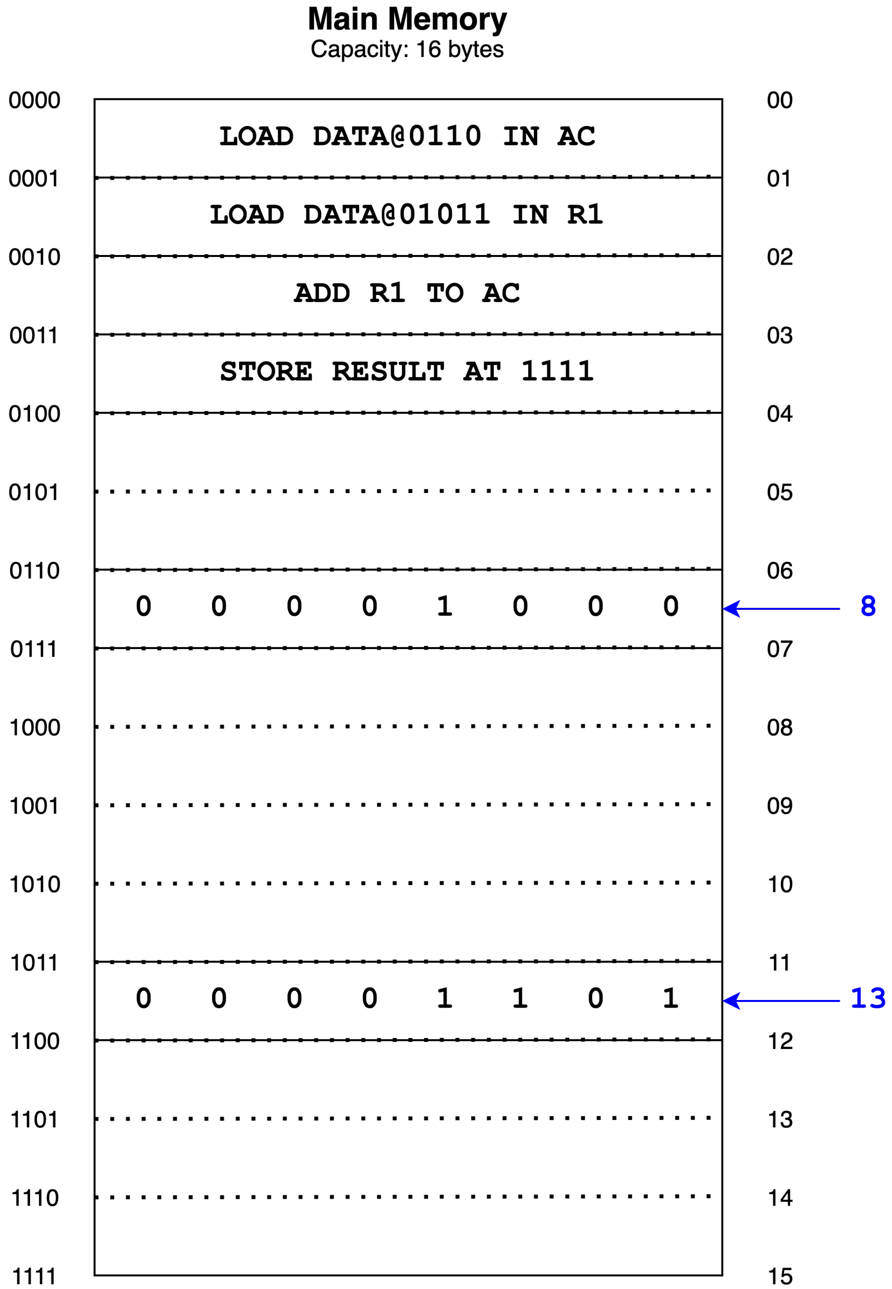
**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| AC | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| R1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

The other operand for addition is now in another register – R1

The PC register now points to the next instruction – at location `0010` in Memory

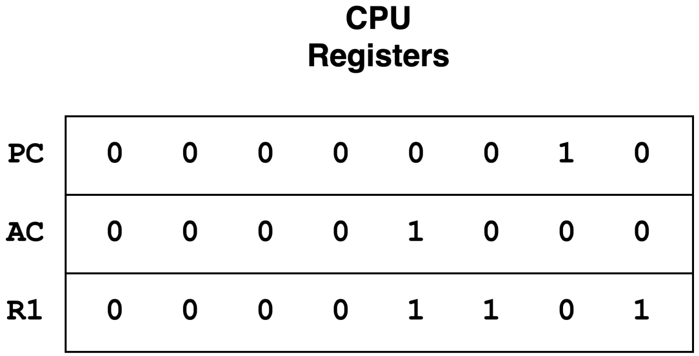
This time, the instruction asks CU to perform an arithmetic operation – addition of values in registers R1 and AC



**Main Memory**

Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |



**CPU Registers**

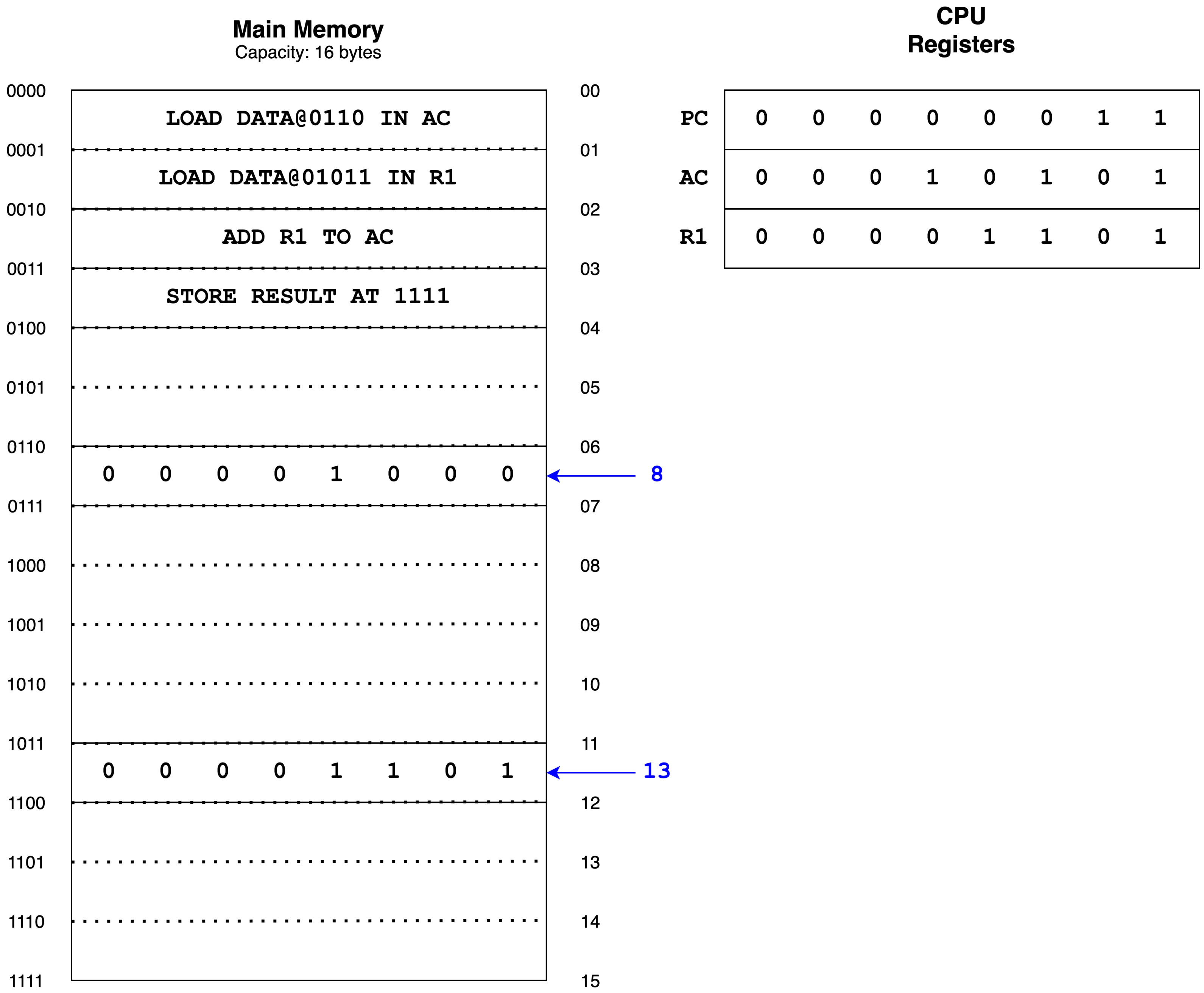
| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| AC | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| R1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

The other operand for addition is now in another register – R1

The PC register now points to the next instruction – at location `0010` in Memory

This time, the instruction asks CU to perform an arithmetic operation – addition of values in registers R1 and AC

Now, the CU activates ALU, which adds the contents of the two registers, and overwrites it back in the AC register



**Main Memory**

Capacity: 16 bytes

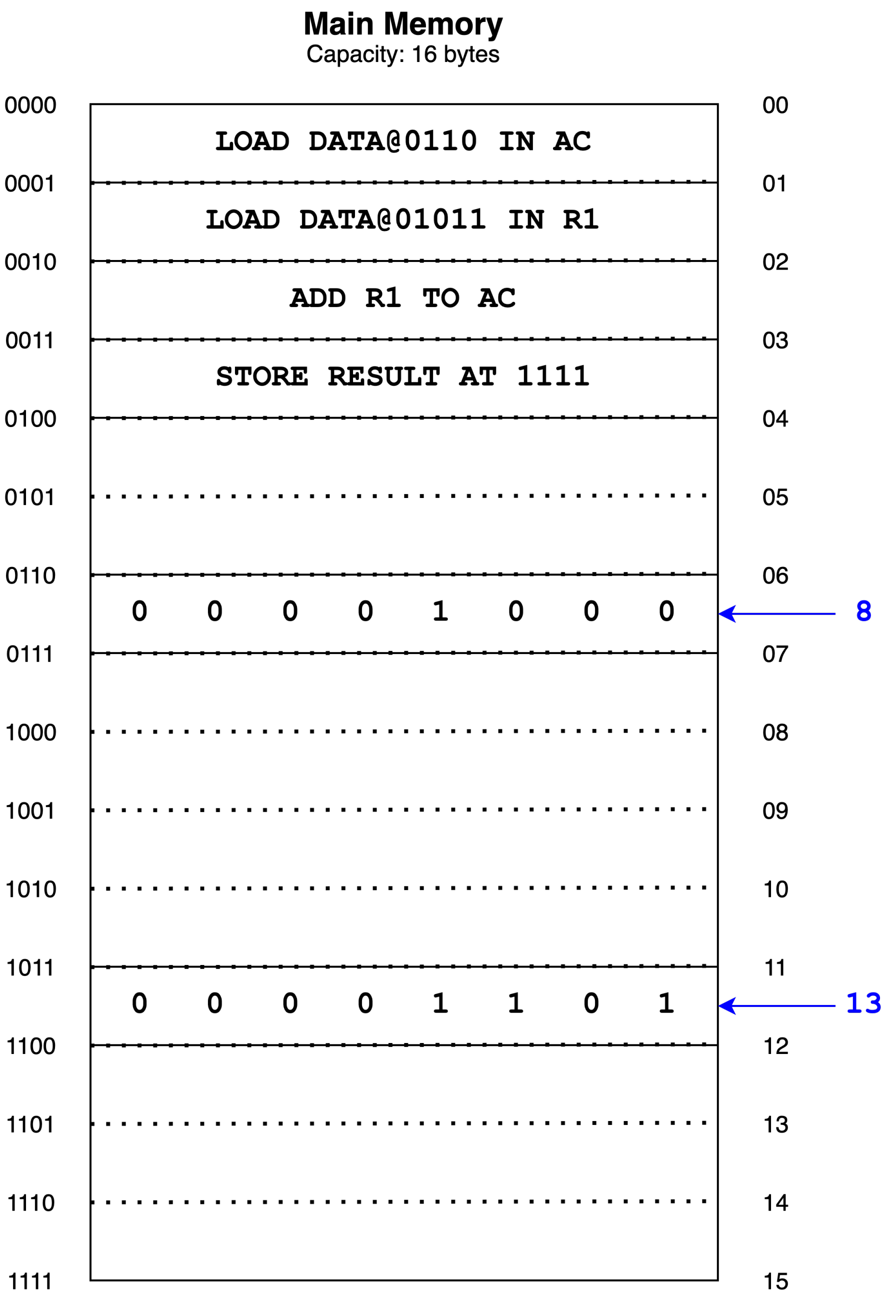
| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 | 06 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 | 11 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | | 14 |
| 1111 | | 15 |

**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| AC | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| R1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

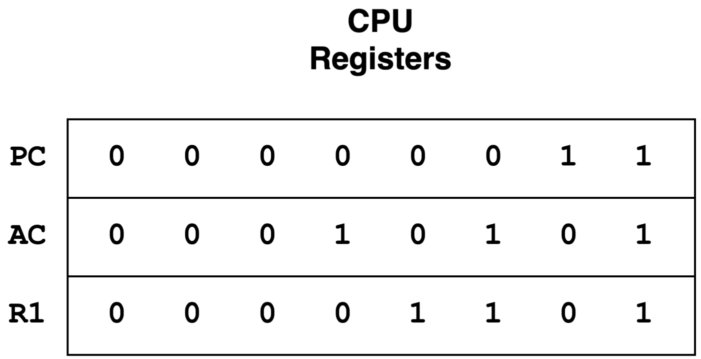
The AC register now has the sum of two numbers –

8 + 13 = 21



**Main Memory**
Capacity: 16 bytes

| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 ← 8 | 06 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 ← 13 | 11 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | | 14 |
| 1111 | | 15 |

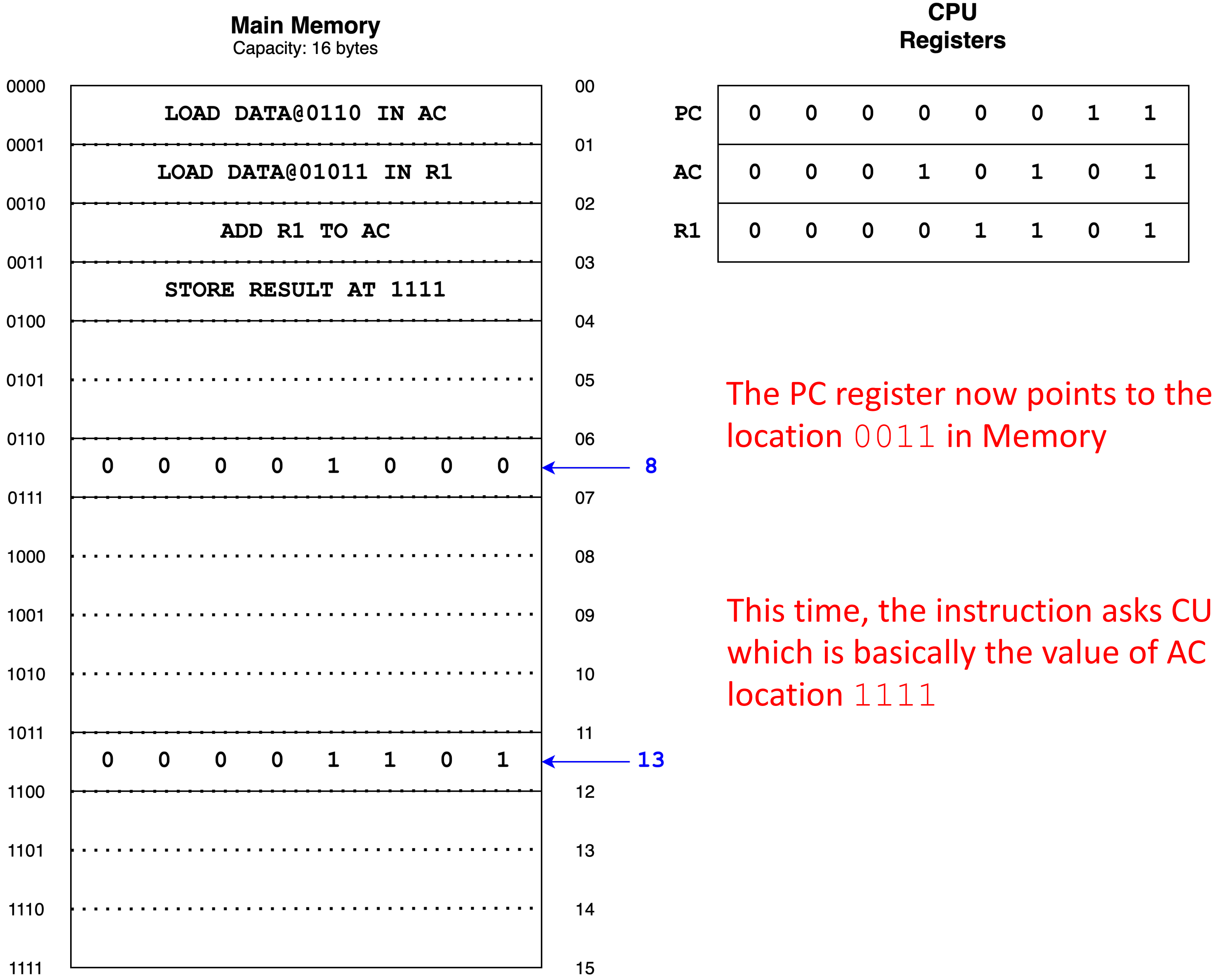


**CPU Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| AC | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| R1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

The AC register now has the sum of two numbers – `8 + 13 = 21`

The PC register now points to the next instruction – at location `0011` in Memory



**Main Memory**
Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | ← 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | ← 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | | 15 | |

**CPU**
**Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| AC | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| R1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |

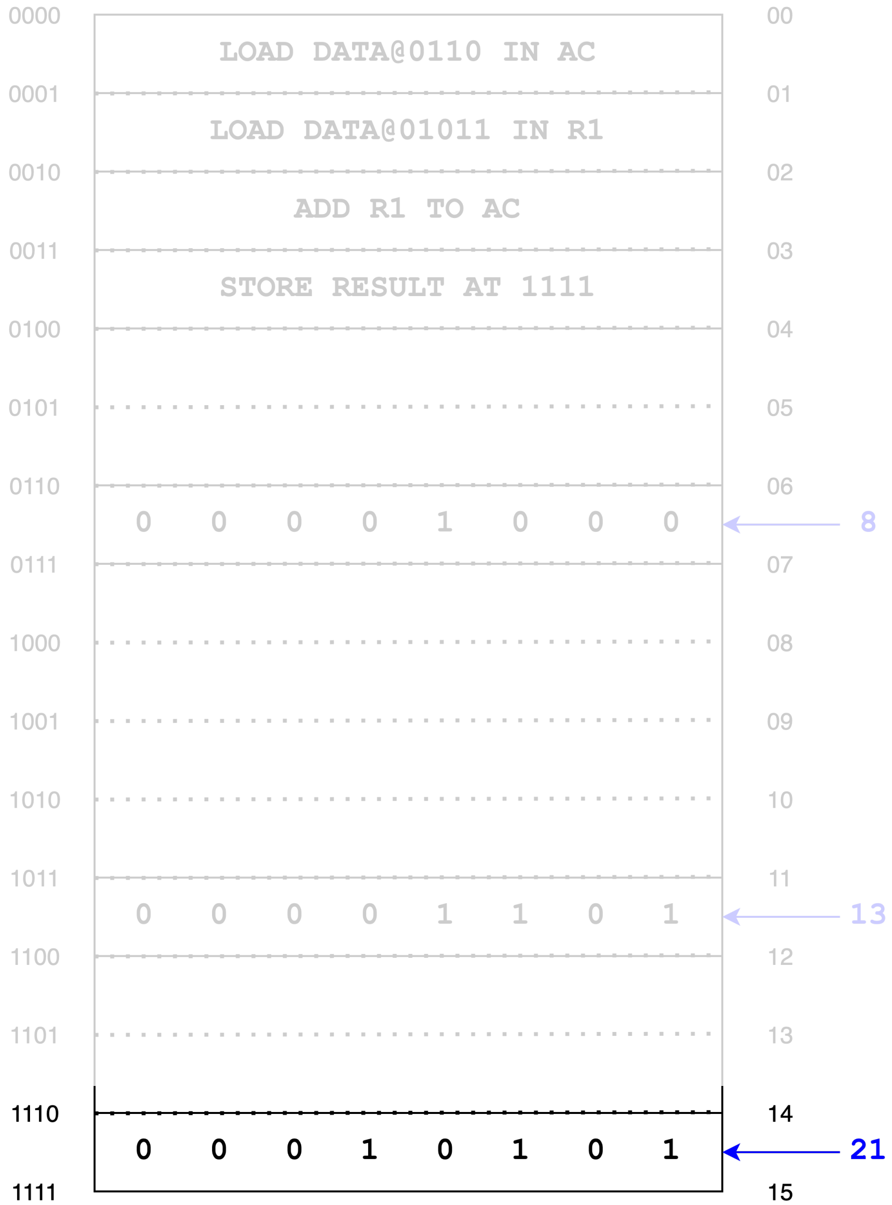
The AC register now has the sum of two numbers – `8 + 13 = 21`

The PC register now points to the next instruction – at location `0011` in Memory

This time, the instruction asks CU to store the "result" – which is basically the value of AC register, to the memory location `1111`

## Main Memory

Capacity: 16 bytes



| | | |
|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |
| 0001 | LOAD DATA@01011 IN R1 | 01 |
| 0010 | ADD R1 TO AC | 02 |
| 0011 | STORE RESULT AT 1111 | 03 |
| 0100 | | 04 |
| 0101 | | 05 |
| 0110 | 0 0 0 0 1 0 0 0 | 06 ← 8 |
| 0111 | | 07 |
| 1000 | | 08 |
| 1001 | | 09 |
| 1010 | | 10 |
| 1011 | 0 0 0 0 1 1 0 1 | 11 ← 13 |
| 1100 | | 12 |
| 1101 | | 13 |
| 1110 | 0 0 0 1 0 1 0 1 | 14 ← 21 |
| 1111 | | 15 |

## CPU Registers

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | X | X | X | X | X | X | X | X |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |

The CU now transfers the content of AC to location `1110` of the Memory

## Main Memory

Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | | 14 | |
| 1111 | 0 0 0 1 0 1 0 1 | 15 | 21 |

## CPU Registers

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | x | x | x | x | x | x | x | x |
| AC | x | x | x | x | x | x | x | x |
| R1 | x | x | x | x | x | x | x | x |

The content of the registers now *do not matter*, as the program has been executed successfully

## Main Memory

Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | ← 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | ← 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | 0 0 0 1 0 1 0 1 | 14 | ← 21 |
| 1111 | | 15 | |

## CPU Registers

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | X | X | X | X | X | X | X | X |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |

The content of the registers now *do not matter*, as the program has been executed successfully

Basically, at this point, the CU has essentially loaded the starting address of the next program to execute, and the cycle repeats

**Main Memory**

Capacity: 16 bytes

| | | | |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 | |
| 0001 | LOAD DATA@01011 IN R1 | 01 | |
| 0010 | ADD R1 TO AC | 02 | |
| 0011 | STORE RESULT AT 1111 | 03 | |
| 0100 | | 04 | |
| 0101 | | 05 | |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | ← 8 |
| 0111 | | 07 | |
| 1000 | | 08 | |
| 1001 | | 09 | |
| 1010 | | 10 | |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | ← 13 |
| 1100 | | 12 | |
| 1101 | | 13 | |
| 1110 | 0 0 0 1 0 1 0 1 | 14 | ← 21 |
| 1111 | | 15 | |

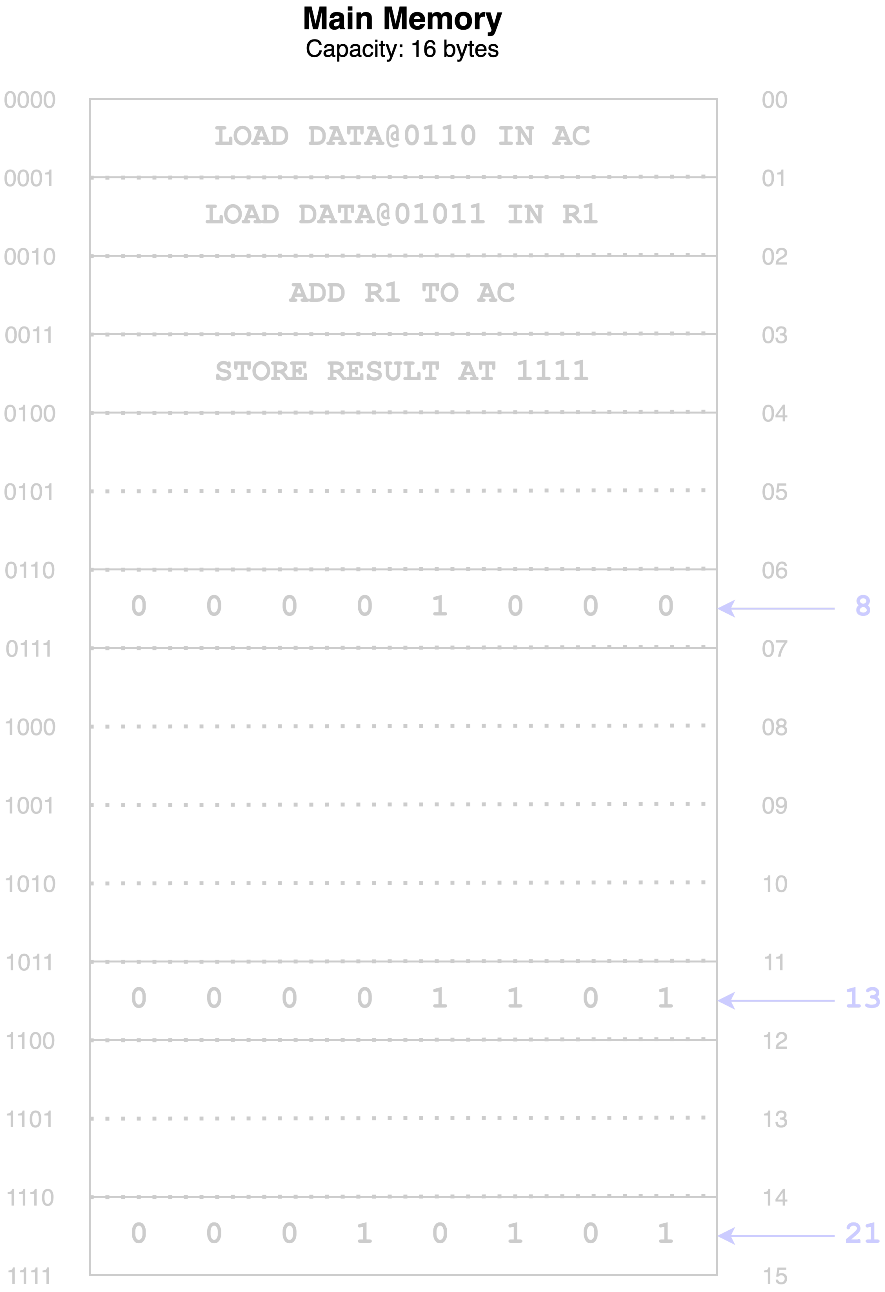
**CPU**
**Registers**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| PC | X | X | X | X | X | X | X | X |
| AC | X | X | X | X | X | X | X | X |
| R1 | X | X | X | X | X | X | X | X |

The content of the registers now *do not matter*, as the program has been executed successfully

Basically, at this point, the CU has essentially loaded the starting address of the next program to execute, and the cycle repeats

Congratulations !! You are now a programmer !!



**Main Memory**
Capacity: 16 bytes

|  |  |  |  |
|---|---|---|---|
| 0000 | LOAD DATA@0110 IN AC | 00 |  |
| 0001 | LOAD DATA@01011 IN R1 | 01 |  |
| 0010 | ADD R1 TO AC | 02 |  |
| 0011 | STORE RESULT AT 1111 | 03 |  |
| 0100 |  | 04 |  |
| 0101 |  | 05 |  |
| 0110 | 0 0 0 0 1 0 0 0 | 06 | ← 8 |
| 0111 |  | 07 |  |
| 1000 |  | 08 |  |
| 1001 |  | 09 |  |
| 1010 |  | 10 |  |
| 1011 | 0 0 0 0 1 1 0 1 | 11 | ← 13 |
| 1100 |  | 12 |  |
| 1101 |  | 13 |  |
| 1110 | 0 0 0 1 0 1 0 1 | 14 | ← 21 |
| 1111 |  | 15 |  |

**CPU**
**Registers**

|  |  |  |  |  |  |  |  |  |
|---|---|---|---|---|---|---|---|---|
| PC | x | x | x | x | x | x | x | x |
| AC | x | x | x | x | x | x | x | x |
| R1 | x | x | x | x | x | x | x | x |

The content of the registers now *do not matter*, as the program has been executed successfully

Basically, at this point, the CU has essentially loaded the starting address of the next program to execute, and the cycle repeats

Congratulations !! You are now a programmer !!

This "program" is a simplified version of an "assembly program" – done in an assembly language

# The *Instructions* in an Assembly Program

We saw the instructions written as `LOAD`, `ADD` or `STORE`

# The *Instructions* in an Assembly Program

We saw the instructions written as `LOAD`, `ADD` or `STORE`

However, in the memory, they too are just some “sequence of bits”

# The *Instructions* in an Assembly Program

We saw the instructions written as `LOAD`, `ADD` or `STORE`

However, in the memory, they too are just some “sequence of bits”

The instruction `LOAD DATA@0110 IN AC` could be stored as `00011011`

# The *Instructions* in an Assembly Program

We saw the instructions written as `LOAD`, `ADD` or `STORE`

However, in the memory, they too are just some “sequence of bits”

The instruction `LOAD DATA@0110 IN AC` could be stored as `00011011`

- Here, `00` is the code for the *load operation*
- `0110` is the *location of the operand*
- …and `11` represents the AC register

# The *Instructions* in an Assembly Program

We saw the instructions written as `LOAD`, `ADD` or `STORE`

However, in the memory, they too are just some “sequence of bits”

The instruction `LOAD DATA@0110 IN AC` could be stored as `00011011`

- Here, `00` is the code for the *load operation*
- `0110` is the *location of the operand*
- …and `11` represents the AC register

Other operations, like STORE and ADD may have codes like `01` and `11`

# The *Instructions* in an Assembly Program

We saw the instructions written as `LOAD`, `ADD` or `STORE`

However, in the memory, they too are just some “sequence of bits”

The instruction `LOAD DATA@0110 IN AC` could be stored as `00011011`

- Here, `00` is the code for the *load operation*
- `0110` is the *location of the operand*
- …and `11` represents the AC register

Other operations, like STORE and ADD may have codes like `01` and `11`

Other registers, like R1 and R2 may be represented by codes `01` and `10`

# The *Instructions* in an Assembly Program

We saw the instructions written as `LOAD`, `ADD` or `STORE`

However, in the memory, they too are just some “sequence of bits”

The instruction `LOAD DATA@0110 IN AC` could be stored as `00011011`

- Here, `00` is the code for the *load operation*
- `0110` is the *location of the operand*
- …and `11` represents the AC register

Other operations, like STORE and ADD may have codes like `01` and `11`

Other registers, like R1 and R2 may be represented by codes `01` and `10`

What differs, is how these bits are interpreted – they can be interpreted as instruction or data

# The Input Device

We understand now that if a program is stored in the memory, the CU can execute it

# The Input Device

We understand now that if a program is stored in the memory, the CU can execute it

But how do we store it there?

# The Input Device

We understand now that if a program is stored in the memory, the CU can execute it

But how do we store it there?

This is where *input devices* come into picture

# The Input Device

We understand now that if a program is stored in the memory, the CU can execute it

But how do we store it there?

This is where *input devices* come into picture

A typical input device is keyboard

# The Input Device

We understand now that if a program is stored in the memory, the CU can execute it

But how do we store it there?

This is where *input devices* come into picture

A typical input device is keyboard

The Operating System ”understands” how your “input device” works !!

# The Input Device

We understand now that if a program is stored in the memory, the CU can execute it

But how do we store it there?

This is where *input devices* come into picture

A typical input device is keyboard

The Operating System ”understands” how your “input device” works !!

The input devices are connected to the memory by buses

# The Input Device

We understand now that if a program is stored in the memory, the CU can execute it

But how do we store it there?

This is where *input devices* come into picture

A typical input device is keyboard

The Operating System ”understands” how your “input device” works !!

The input devices are connected to the memory by buses

The Operating System contains “assembly programs to transfer data” from the device to memory

- That is all you need to know for now !! “Somehow” the OS knows when and how to do this

# The Output Device

The sum that we did using the program needs to be “retrieved” too

# The Output Device

The sum that we did using the program needs to be “retrieved” too

An Output Device can fetch this data from the memory for us

# The Output Device

The sum that we did using the program needs to be “retrieved” too

An Output Device can fetch this data from the memory for us

Again, just like the Input Device, the OS knows “how to manage” the device

# The Output Device

The sum that we did using the program needs to be “retrieved” too

An Output Device can fetch this data from the memory for us

Again, just like the Input Device, the OS knows “how to manage” the device

A typical output device is your Laptop’s Screen or a Monitor

- Another popular output device is Printer

# The Output Device

The sum that we did using the program needs to be “retrieved” too

An Output Device can fetch this data from the memory for us

Again, just like the Input Device, the OS knows “how to manage” the device

A typical output device is your Laptop’s Screen or a Monitor

- Another popular output device is Printer

The Operating System has programs which can send data from the memory to the device

- Again… that is all you need to know at this stage !!

# To summarise…

The Operating System is a *collection of programs* which can

- Manage Input Devices

# To summarise…

The Operating System is a *collection of programs* which can

- Manage Input Devices
- Manage Output Devices

# To summarise…

The Operating System is a *collection of programs* which can

- Manage Input Devices
- Manage Output Devices
- Manage Main Memory

# To summarise…

The Operating System is a *collection of programs* which can

- Manage Input Devices
- Manage Output Devices
- Manage Main Memory
- Co-ordinate with the Control Unit to get some computation done

# To summarise…

The Operating System is a *collection of programs* which can

- Manage Input Devices
- Manage Output Devices
- Manage Main Memory
- Co-ordinate with the Control Unit to get some computation done

It provides an "easier" interface for us to get our jobs done

# To summarise…

The Operating System is a *collection of programs* which can

- Manage Input Devices
- Manage Output Devices
- Manage Main Memory
- Co-ordinate with the Control Unit to get some computation done

It provides an “easier” interface for us to get our jobs done

- Easier, because we can, in theory, directly manage the CPU as well !!

# To summarise…

The Operating System is a *collection of programs* which can

- Manage Input Devices
- Manage Output Devices
- Manage Main Memory
- Co-ordinate with the Control Unit to get some computation done

It provides an “easier” interface for us to get our jobs done

- Easier, because we can, in theory, directly manage the CPU as well !!
- But that will involve speaking in 0100100010101111O110… Sorry I got a little carried away :P

# To summarise…

The Operating System is a *collection of programs* which can

- Manage Input Devices
- Manage Output Devices
- Manage Main Memory
- Co-ordinate with the Control Unit to get some computation done

It provides an “easier” interface for us to get our jobs done

- Easier, because we can, in theory, directly manage the CPU as well !!
- But that will involve speaking in 0100100010101110110… Sorry I got a little carried away :P

You’ll have a whole subject dedicated to Operating Systems, so I leave it here for now

# Homework !!

Read more about the type of operations that a typical Assembly Language may have

- Just reading this tutorial maybe more than enough for now: https://www.tutorialspoint.com/assembly_programming/assembly_introduction.htm