Simplified View of Memory

A Quick Look

The Instruction Example

```
a = b + c;
```

- What are a, b, and c?

The Instruction Example

```
a = b + c;
```

- What are a, b, and c?
- They identify unique locations in Memory
  - Addresses

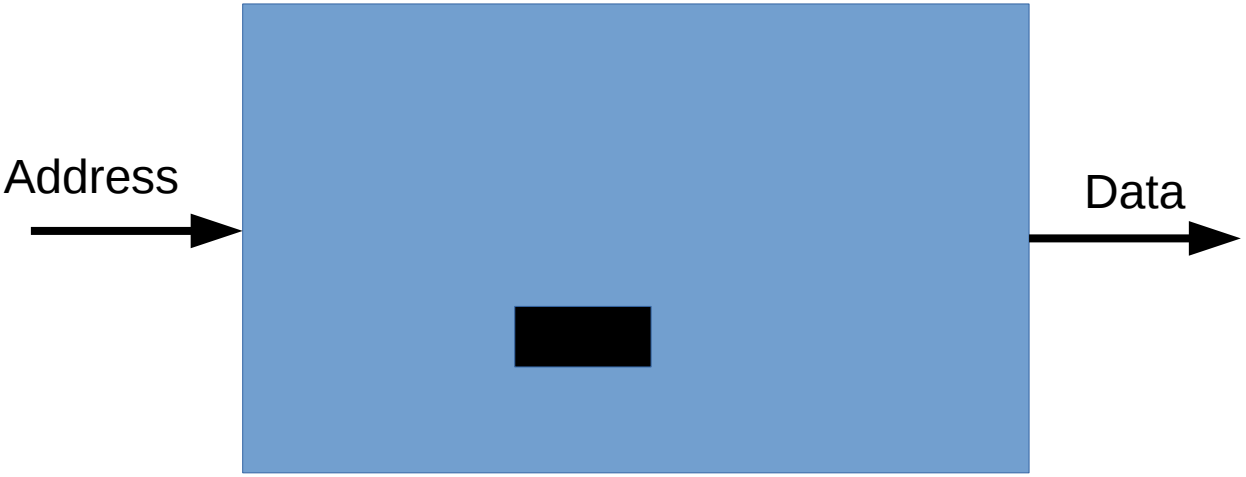
Simplified View of Memory

- Memory stores bits

Simplified View of Memory

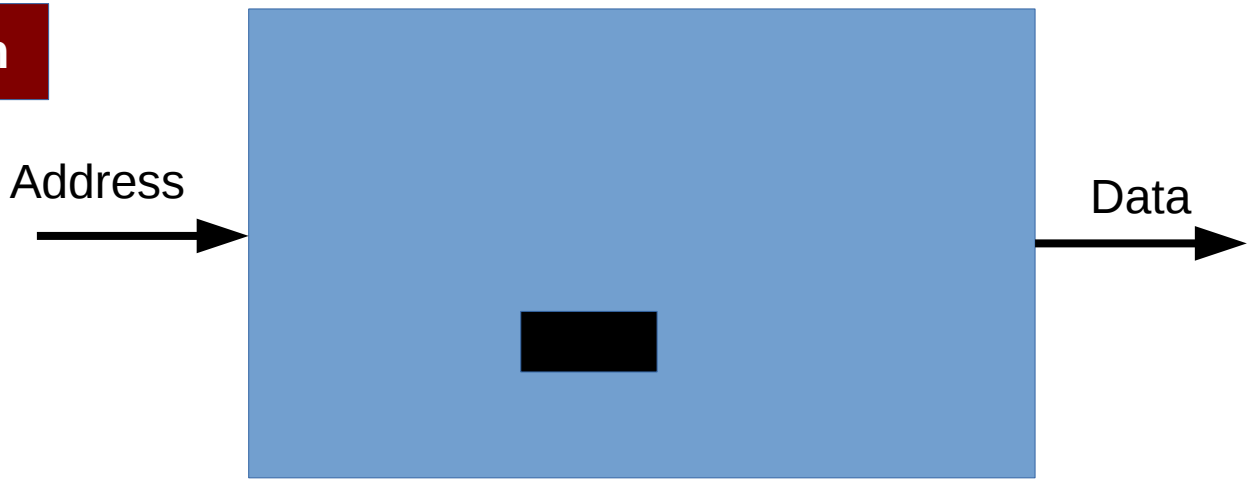
- Memory stores bits
- What operations can a memory perform?
  - Read and Write

Operations on Memory



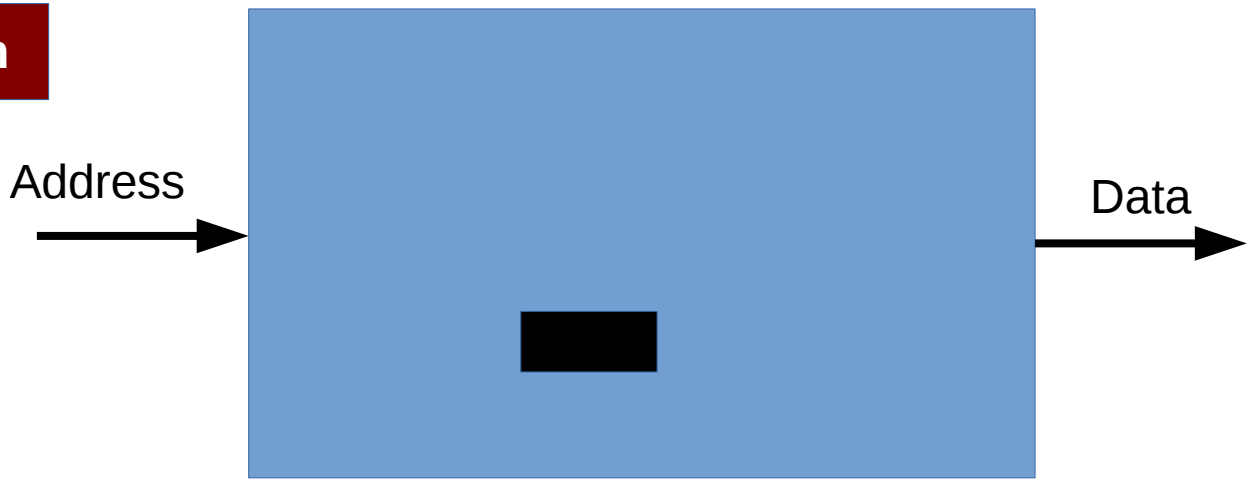
Operations on Memory

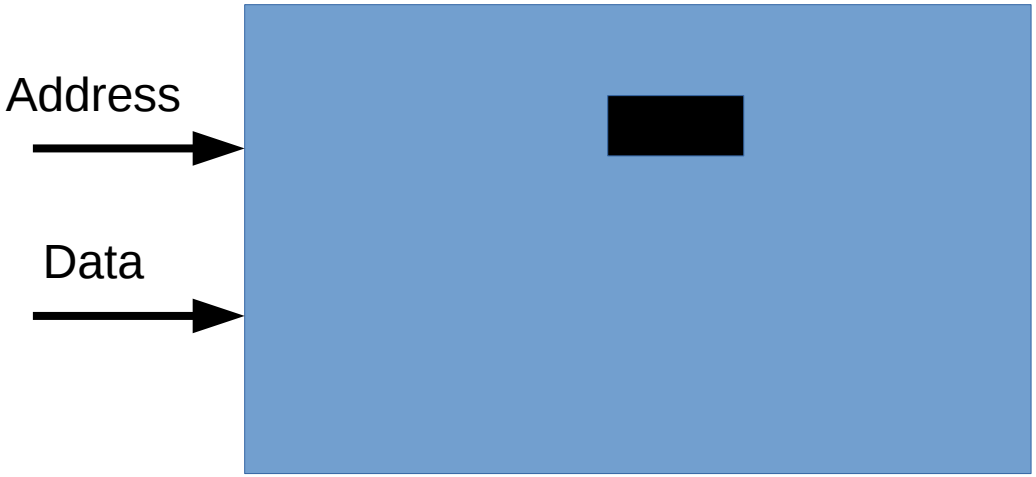
Read Operation



Operations on Memory

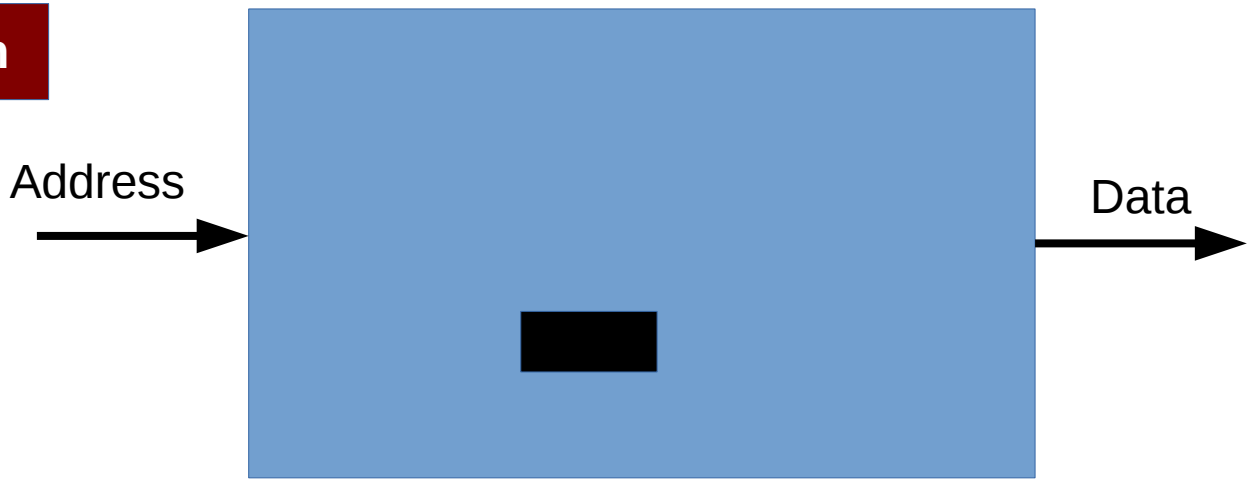
Read Operation

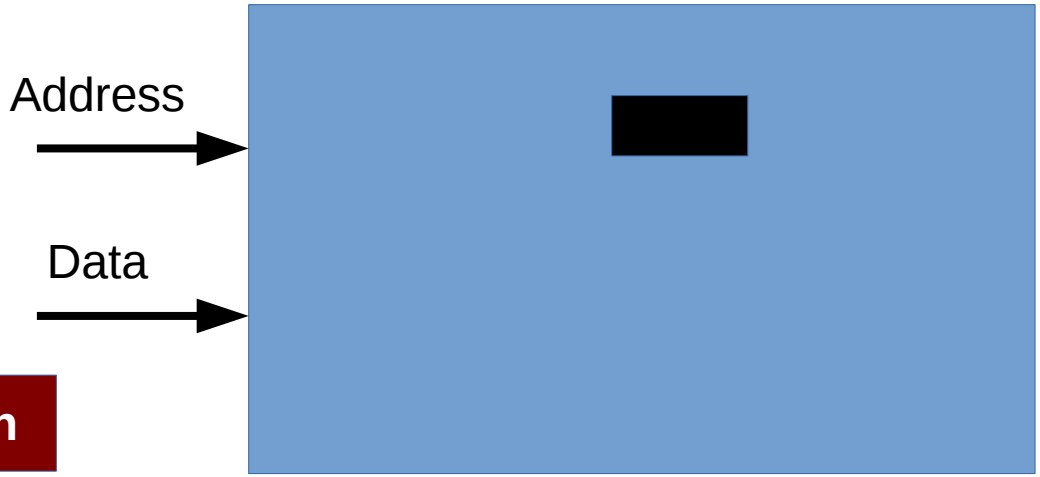




Operations on Memory

Read Operation





Write Operation

Memory Capacities

- 1 KB of memory
- How many bytes in
- KB =
- MB =
- GB =
- PB =
- EB =

Memory Capacities

- 1 KB of memory
- How many bytes in
- KB = 1024 = 2^{10} Bytes
- MB = 1024 KB= 2^{20} Bytes
- GB = 1024 MB = 2^{30} Bytes
- TerraByte = 1024 GB = 2^{40} Bytes
- PetaByte = 1024 TB = 2^{50} Bytes
- ExaByte = 1024 PB = 2^{60} Bytes
- ZetaByte = 1024 EB = 2^{70} Bytes
- YotaByte = 1024 ZB = 2^{80} Bytes

Memory Address Size

- 1 KB = 1024 Bytes = 2^{10} Bytes

Memory Address Size

- 1 KB = 1024 Bytes = 2^{10} Bytes
- Each byte has an address

Memory Address Size

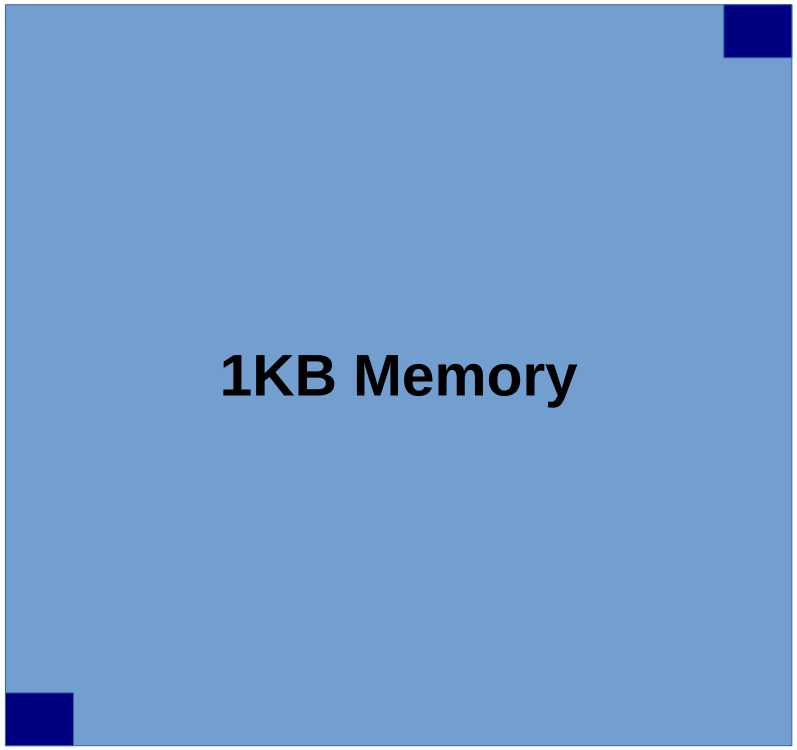
- 1 KB = 1024 Bytes = 2^{10} Bytes
- Each byte has an address
- First Address = 0; Last address = 1023

Memory Address Size

- 1 KB = 1024 Bytes = 2^{10} Bytes
- Each byte has an address
- First Address = 0; Last address = 1023
- No. of bits in the address
  - $\log_2$ Size
  - $\log_2 1024 = 10$ bits

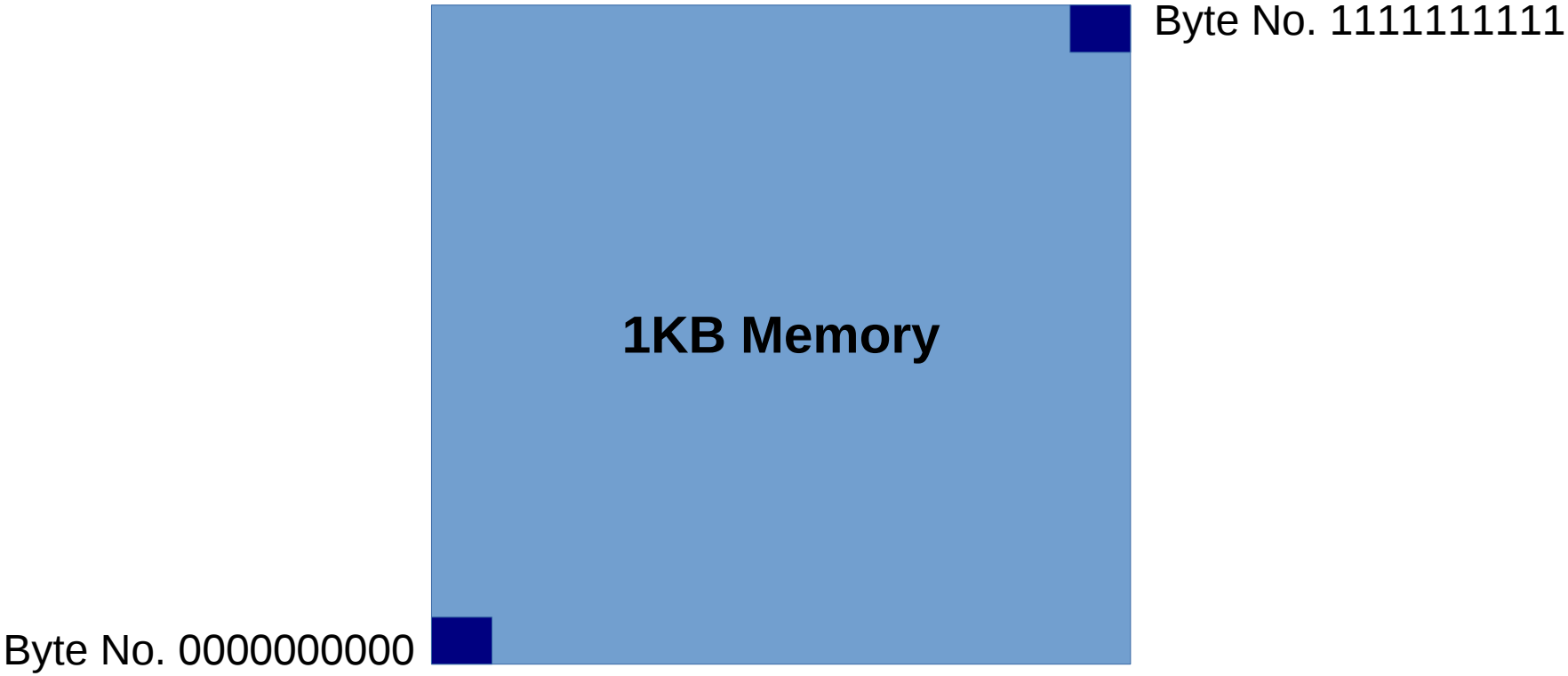
Simplified 1KB Memory

- 1 KB = 1024 Bytes = 2^{10} Bytes
- 10 bit address



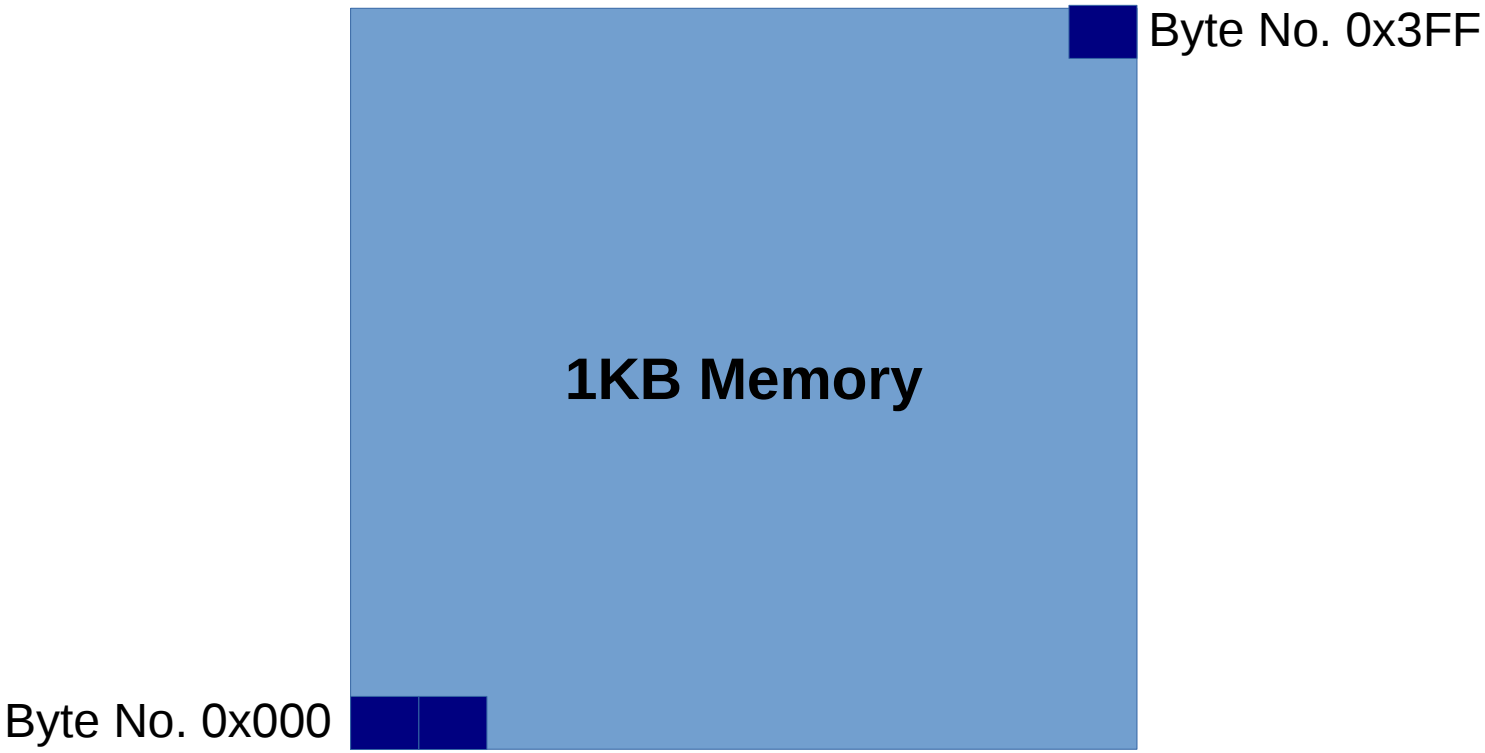
Simplified 1KB Memory

- 1 KB = 1024 Bytes = 2^{10} Bytes
- 10 bit address



Simplified 1KB Memory

- 1 KB = 1024 Bytes = 2^{10} Bytes
- 10 bit address



Simplified Memory

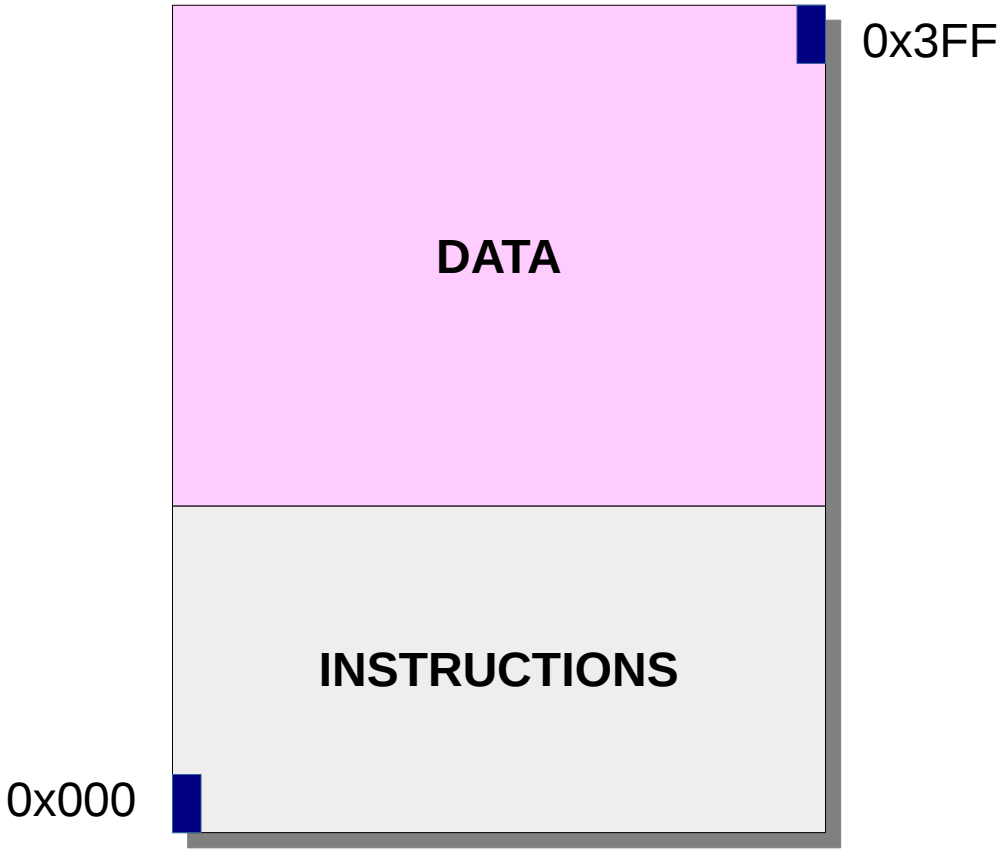
- Recall: Instructions and Data reside in the memory

Simplified Memory

- Recall: Instructions and Data reside in the memory

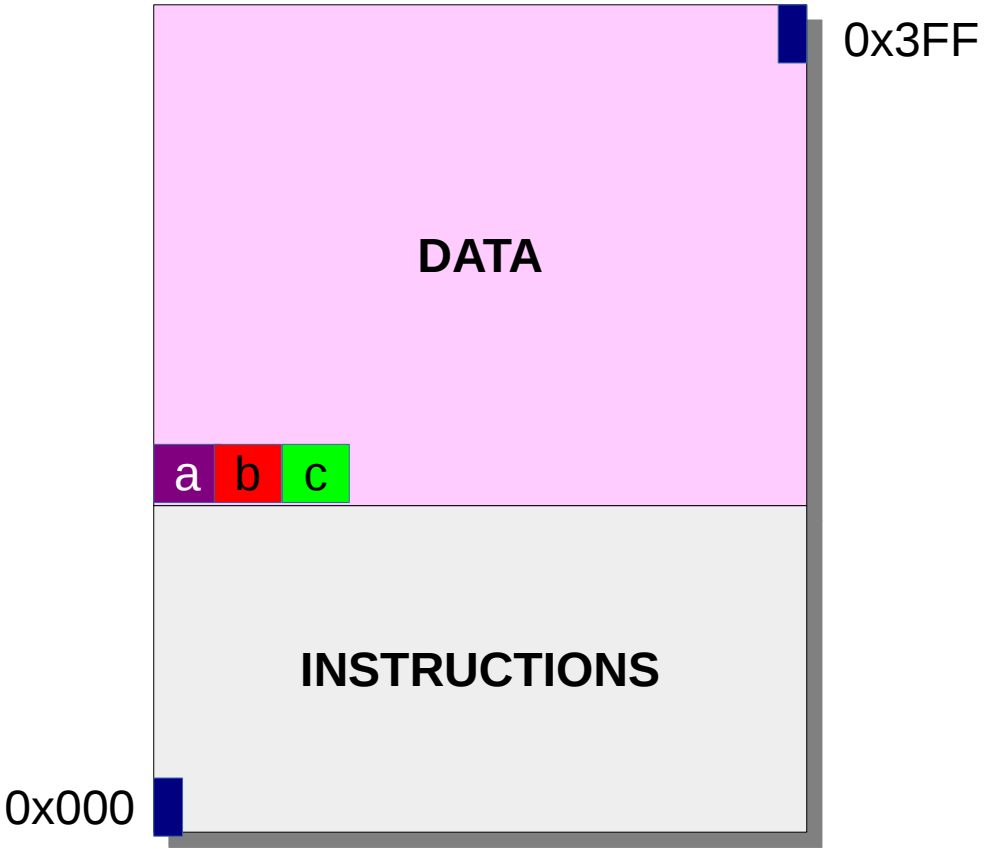
Simplified Memory

- Recall: Instructions and Data reside in the memory



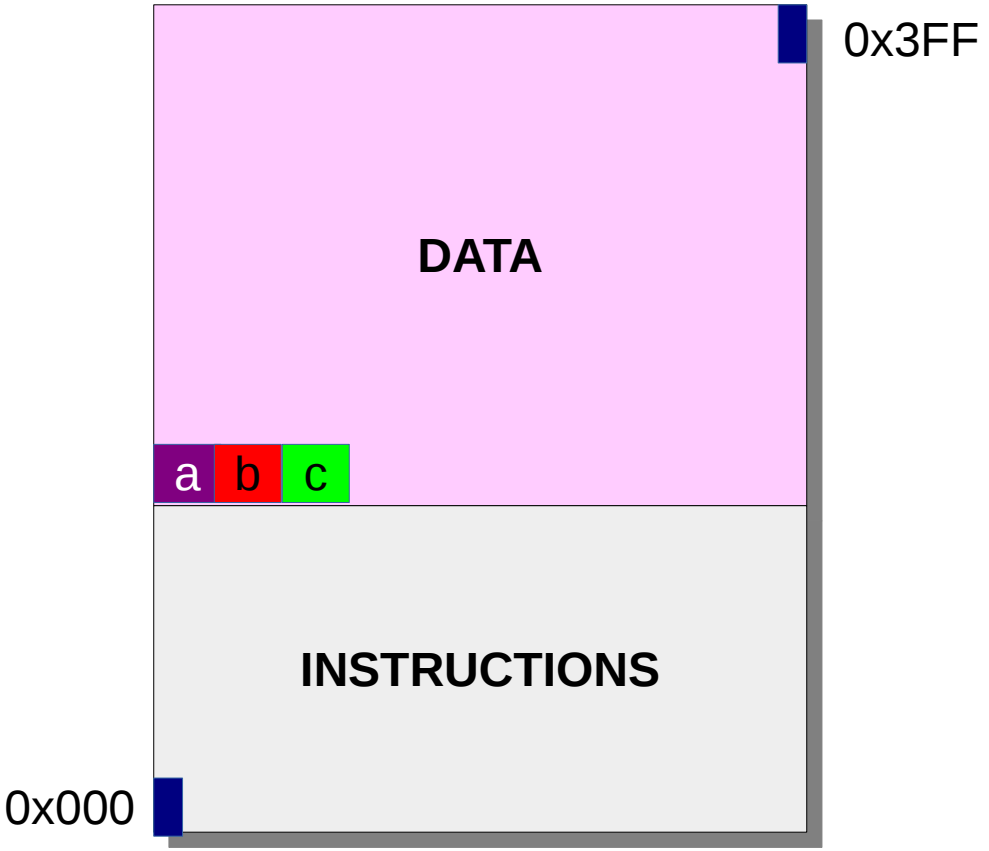
Simplified Program Memory

- Instructions and Data reside in the memory
- Assume each of a, b, c are 4 Bytes



Simplified Program Memory

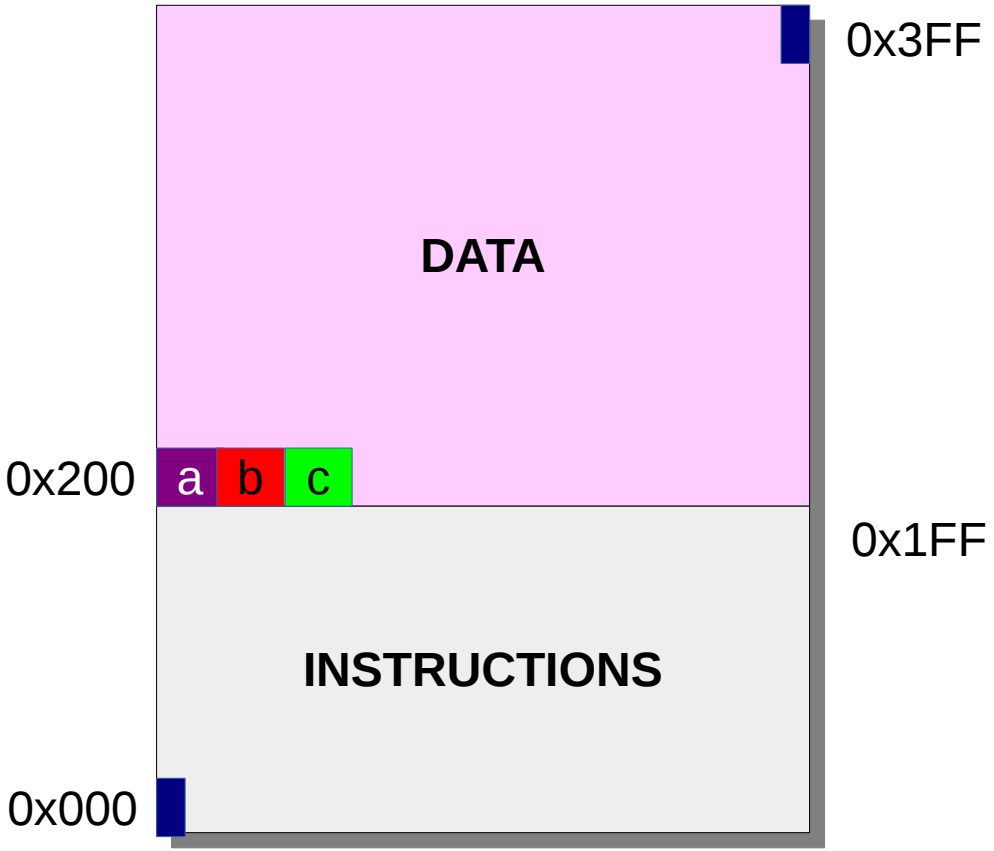
- Instructions and Data reside in the memory
- Assume each of a, b, c are 4 Bytes



If Instructions and data occupy equal bytes in the memory, What are the addresses of a, b, and c?

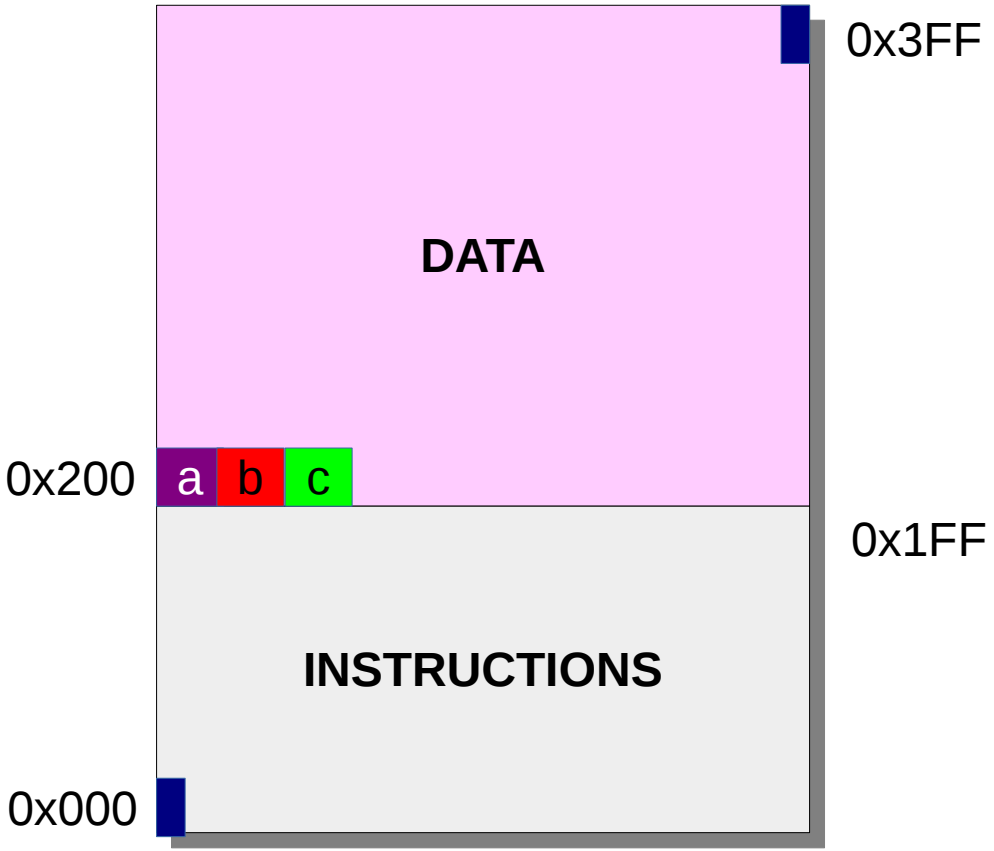
Data Memory

- Address range = 0 to 1023 Bytes = 1024 Bytes



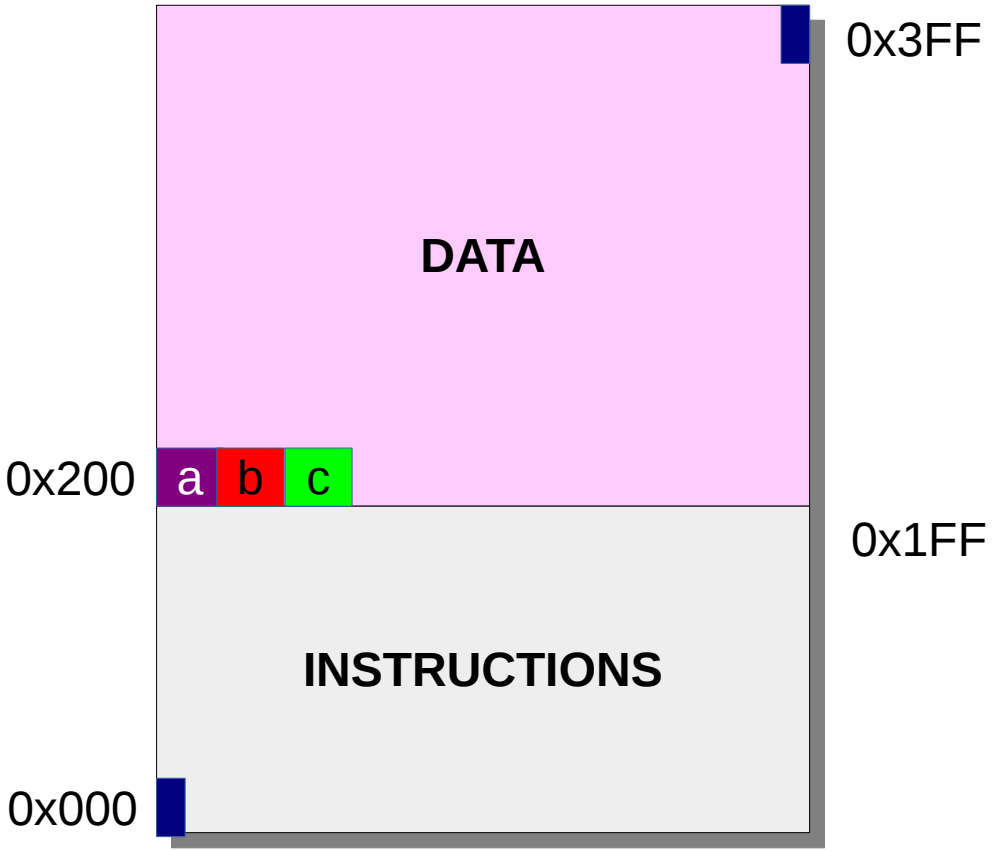
Data Memory

- Address range = 0 to 1023 Bytes = 1024 Bytes
- Instructions are in 512B; Data fills the rest of the 512B



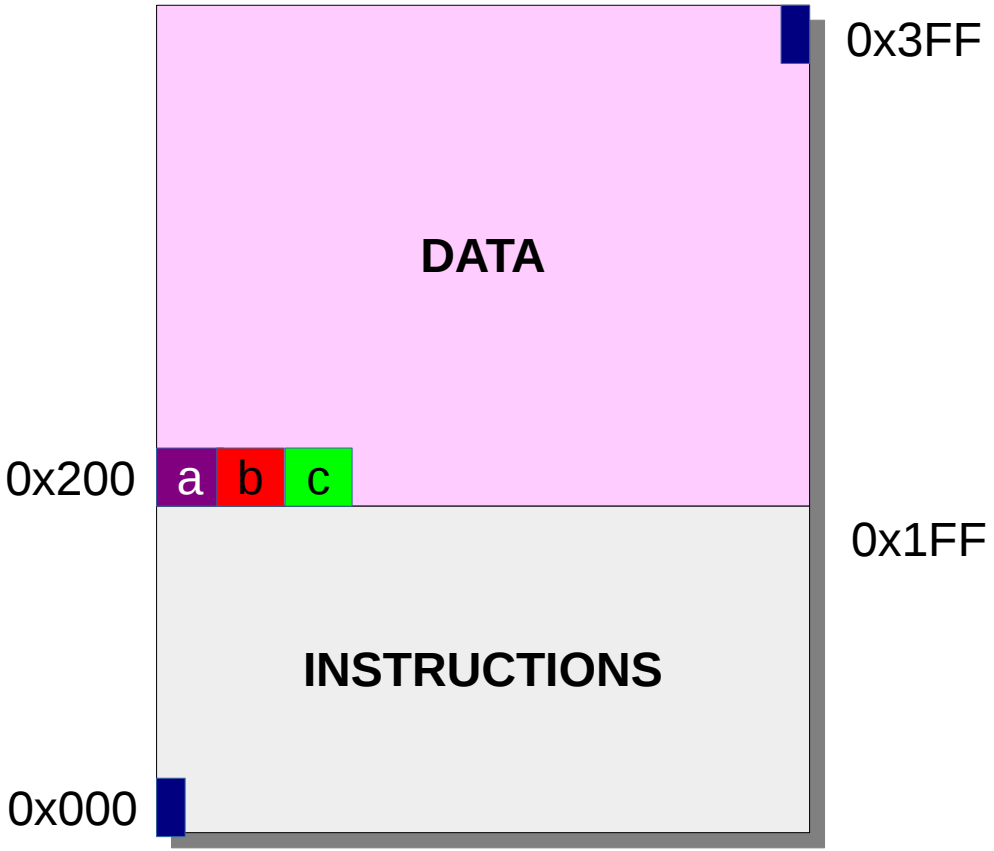
Data Memory

- Address range = 0 to 1023 Bytes = 1024 Bytes
- Instructions are in 512B; Data fills the rest of the 512B
- Instructions Address Range = 0 to 511 = 0 to 111111111 = 0x000 to 0x1FF



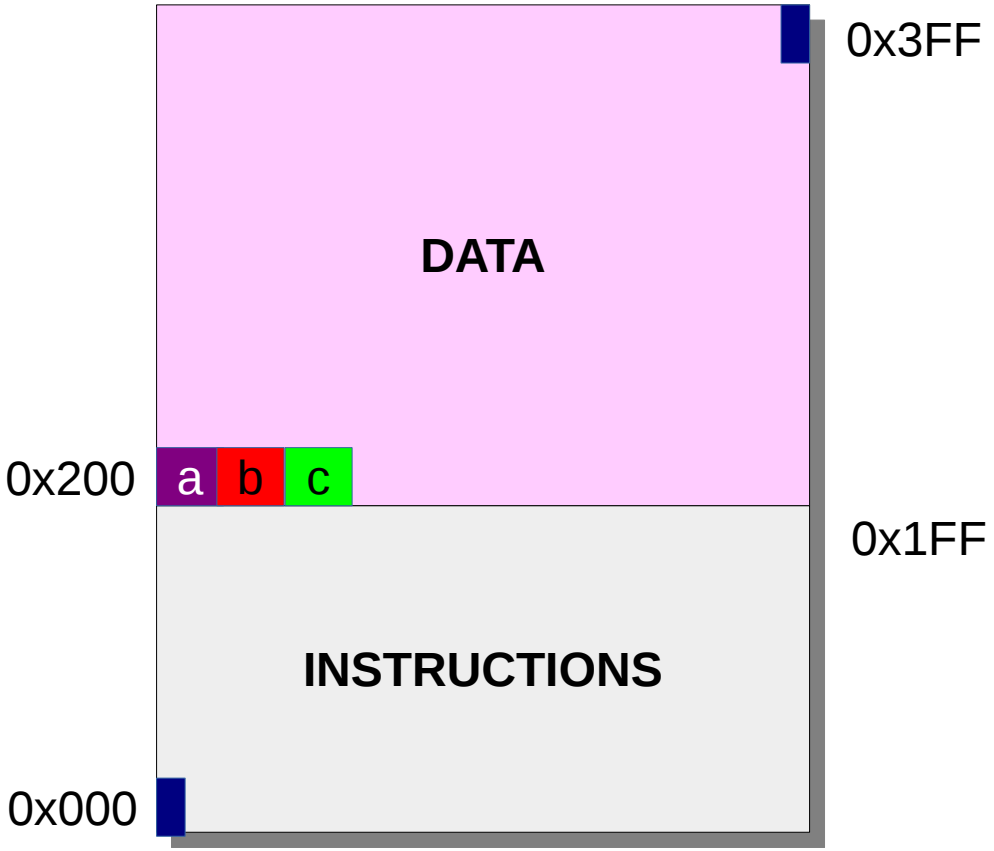
Data Memory

- Address range = 0 to 1023 Bytes = 1024 Bytes
- Instructions are in 512B; Data fills the rest of the 512B
- Instructions Address Range = 0 to 511 = 0 to 111111111 = 0x000 to 0x1FF
- Data Address range = 0x200 to 0x3FF



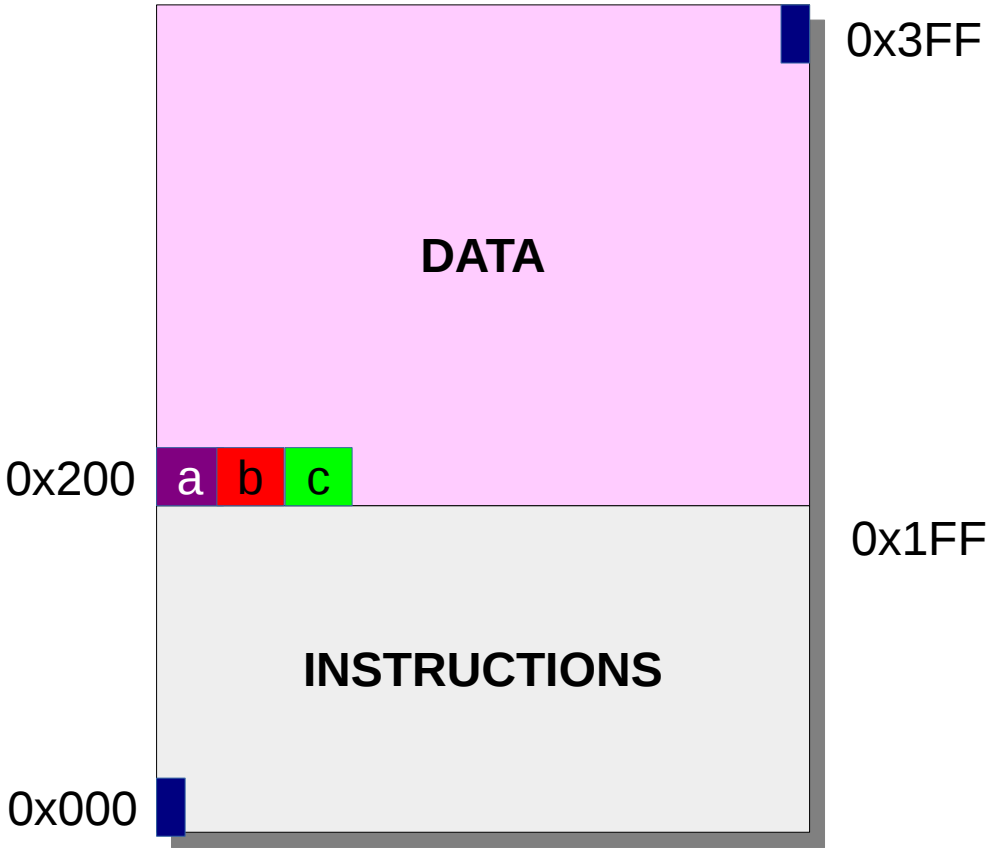
Data Memory

- Address of first variable = 512; second variable = 512+4; third variable = 512+8; and so on.



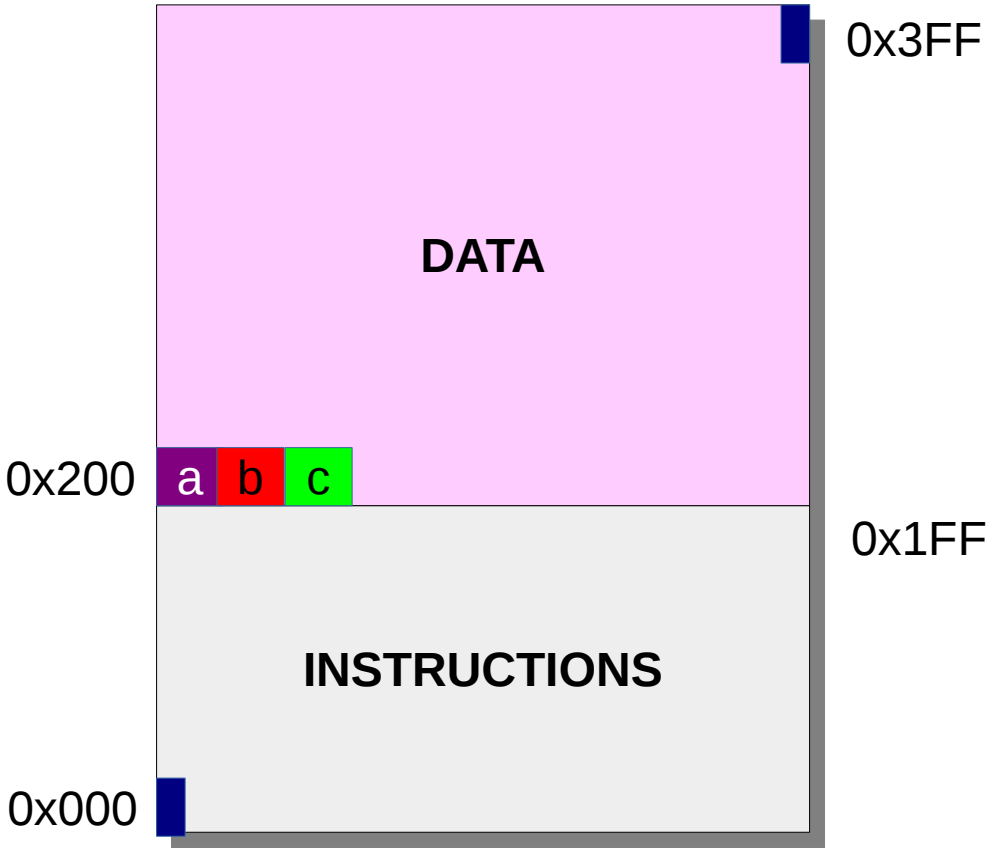
Data Memory

- Address of first variable = 512; second variable = 512+4; third variable = 512+8; and so on.
- Address of first variable = 0x200; second variable = 0x200+4; third variable = 0x200+8; and so on.
- What is the address of the 10th variable?



Data Memory

- Address of first variable = 512; second variable = 512+4; third variable = 512+8; and so on.
- Address of first variable = 0x200; second variable = 0x200+4; third variable = 0x200+8; and so on.



Data Addresses

Variable	Decimal Address	
1	512	0x200 + 0
2	512 + 4 = 516	0x200 + 1*4
3	512 + 8 = 520	0x200 + 2*4
i^{th}	512 + (i-1)*4	0x200 + (i-1)*4

Data Addresses

Variable	Decimal Address	
1	512	0x200 + 0
2	512 + 4 = 516	0x200 + 1*4
3	512 + 8 = 520	0x200 + 2*4
i^{th}	512 + (i-1)*4	0x200 + (i-1)*4

- The address 512 (0x200) is called the Base Address (B)

Data Addresses

Variable	Decimal Address	
1	512	0x200 + 0
2	512 + 4 = 516	0x200 + 1*4
3	512 + 8 = 520	0x200 + 2*4
i^{th}	512 + (i-1)*4	0x200 + (i-1)*4

- The address 512 (0x200) is called the Base Address (B)
- *i* is the index; *i* ranges from *(0, ..., n-1)*; Total elements = n.

Data Addresses

Variable	Decimal Address	
1	512	0x200 + 0
2	512 + 4 = 516	0x200 + 1*4
3	512 + 8 = 520	0x200 + 2*4
i^{th}	512 + (i-1)*4	0x200 + (i-1)*4

- The address 512 (0x200) is called the Base Address (B)
- *i* is the index; *i* ranges from *(0, ..., n-1)*; Total elements = n.
- The ith variable can be accessed at address

$$Address_i = BaseAddress + i * VariableSize$$

Data Addresses

Variable	Decimal Address	
1	512	0x200 + 0
2	512 + 4 = 516	0x200 + 1*4
3	512 + 8 = 520	0x200 + 2*4
i^{th}	512 + (i-1)*4	0x200 + (i-1)*4

- The address 512 (0x200) is called the Base Address (B)
- *i* is the index; *i* ranges from *(0, ..., n-1)*; Total elements = n.
- The ith variable can be accessed at address

$$Address_i = BaseAddress + i * VariableSize$$

- Where can such a scheme be useful?

Assembly Code revisited

```
Load R1, b
Load R2, c
Add R3, R1, R2
Store R3, a
```

Memory expects an address
a, b, and c have to be converted to their corresponding addresses!
Compiler populates and maintains a table with this info.

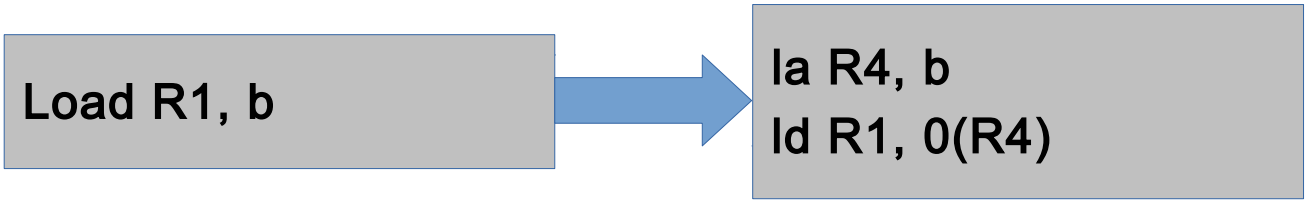
Assembly Code revisited

```
Load R1, b
Load R2, c
Add R3, R1, R2
Store R3, a
```

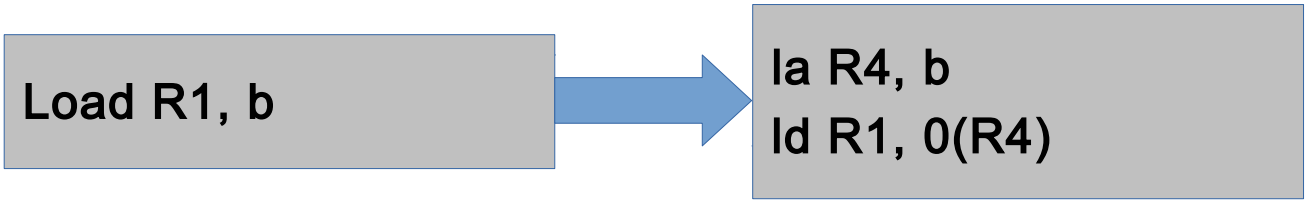
Memory expects an address
a, b, and c have to be converted
to their corresponding addresses!
Compiler populates and maintains
a table with this info.

- Address calculation and maintenance is one of the jobs of the Compiler

Correct Code

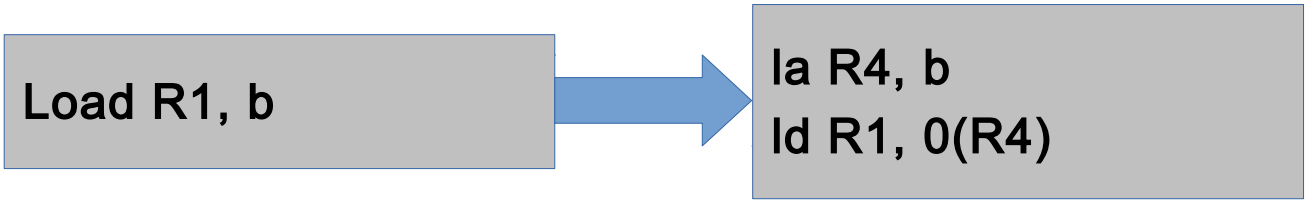


Correct Code



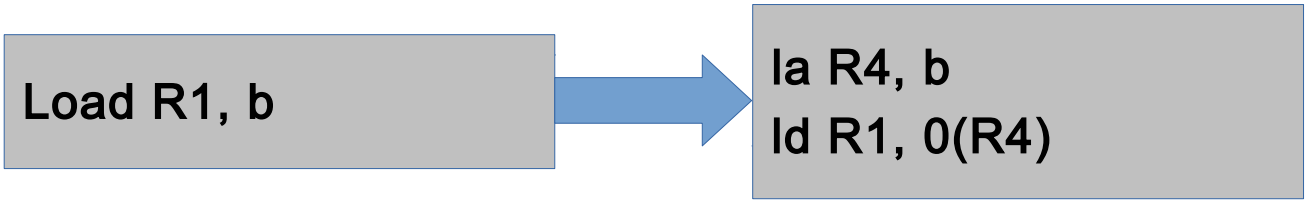
- R4 contains the Address of variable ‘b’

Correct Code



- R4 contains the Address of variable ‘b’
- 0(R4): Address = [R4 + 0]

Correct Code



- Would these instructions work?
  - ld R1, R4
  - ld R1, (R4)

MIPS ISA & Assembly Language

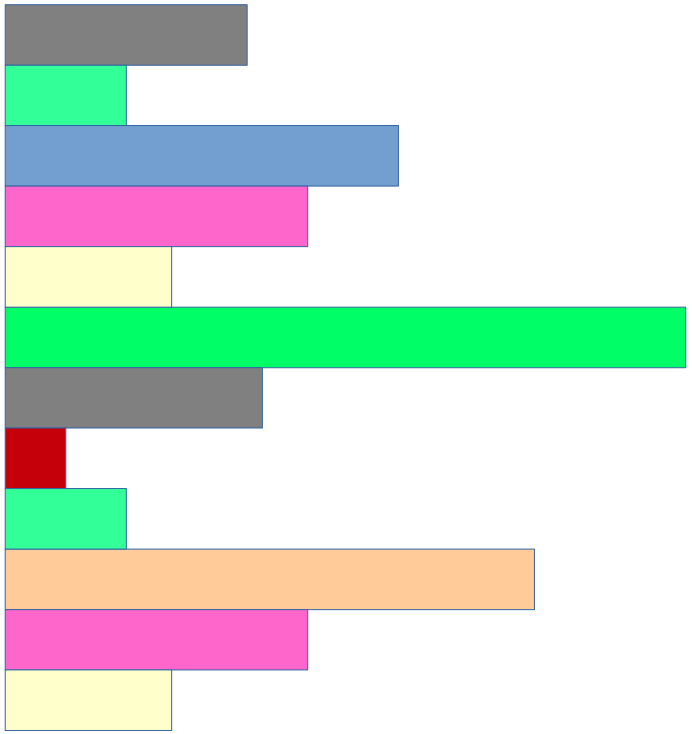
- Simple and elegant ISA

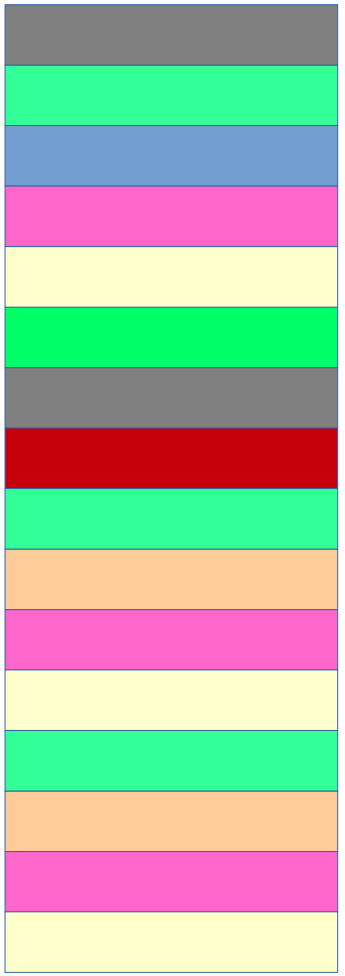
MIPS ISA & Assembly Language

- Simple and elegant ISA
- One of the first RISC ISAs

CISC vs. RISC

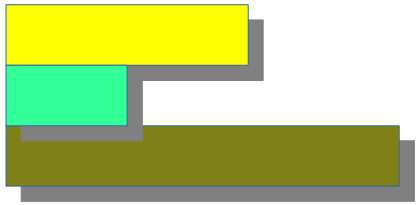
- Instruction Sizes





CISC vs. RISC

ADD R3, R1, (R2)



ADD R3, R1, R2

Simple Architecture (1980s)

- IBM 801, Berkely RISC Processor, and Stanford MIPS
- Simple load-store architecture
- Fixed-format 32-bit instructions
- Efficient pipelining
- **Reduced Instruction Set Computers (RISC)**

John Cocke (Turing award, 1987) is the father of IBM 801.

MIPS ISA & Assembly Language

- Simple and elegant ISA
- One of the first RISC ISAs
- Influenced a whole revolution in the processor industry

MIPS ISA & Assembly Language

- Simple and elegant ISA
- One of the first RISC ISAs
- Influenced a whole revolution in the processor industry
- Entire ISA at the back of the textbook

MIPS ISA & Assembly Language

- Simple and elegant ISA
- One of the first RISC ISAs
- Influenced a whole revolution in the processor industry
- Entire ISA at the back of the textbook
- Use the simulator SPIM (xspim, qtspim) for simulating MIPS assembly programs

Accurate Code

```
Load R1, b
Load R2, c
Add R3, R1, R2
Store R3, a
```

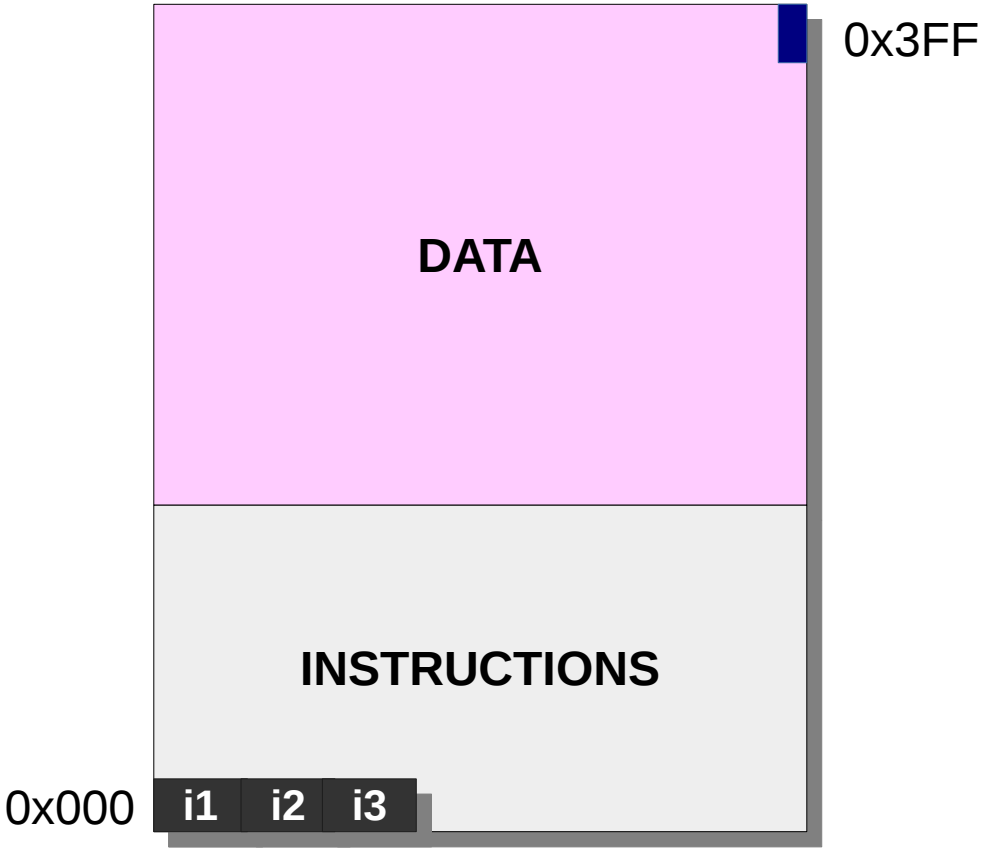
```
la R4, b
la R5, c
la R6, a
ld R1, 0(R4)
ld R2, 0(R5)
add R3, R1, R2
st R3, 0(R6)
```

Accurate MIPS Code

```
Load R1, b
Load R2, c
Add R3, R1, R2
Store R3, a
```

```
la $t0, b
la $t1, c
la $t2, a
ld $t3, 0($t0)
ld $t4, 0($t1)
add $t5, $t3, $t4
st $t5, 0($t2)
```

Instruction Memory



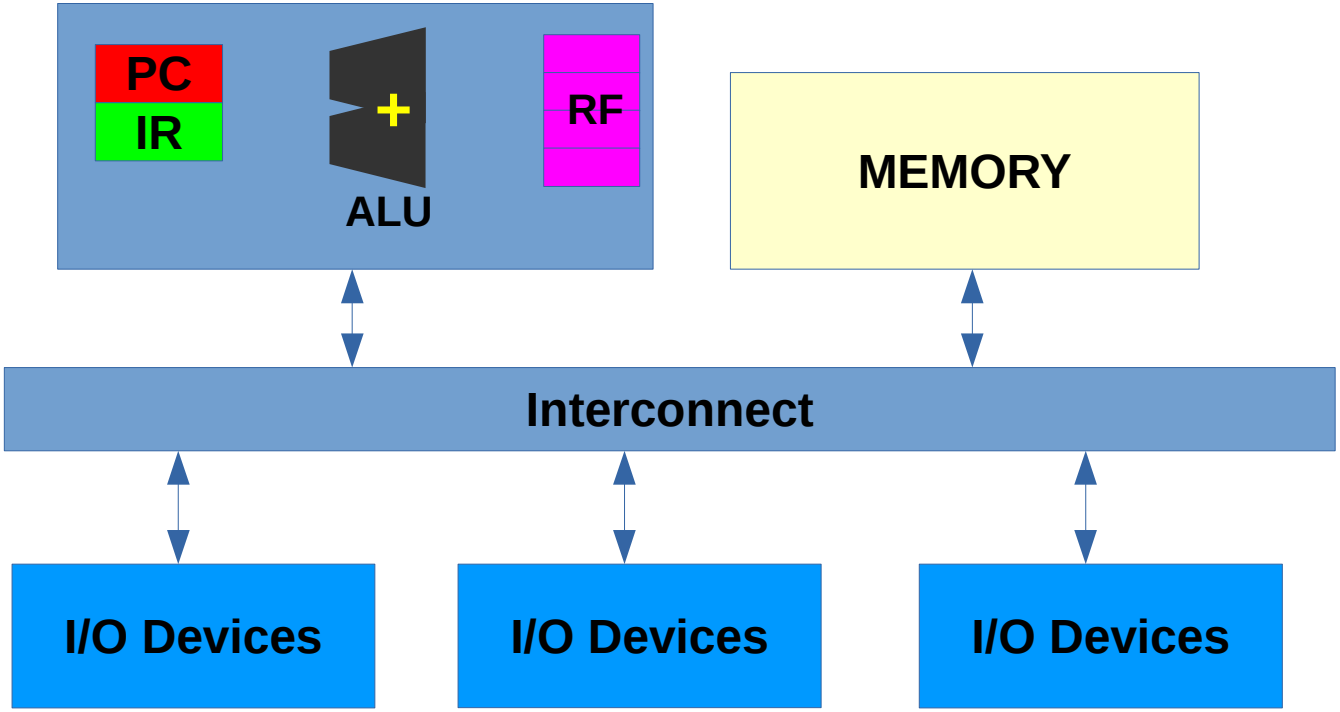
Special Registers for Instructions

- The instruction to be executed should be present inside the processor
  - Instruction Register

Special Registers for Instructions

- The instruction to be executed should be present inside the processor
  - Instruction Register
- Program Counter keeps track of the next instruction to be executed

The Computer System



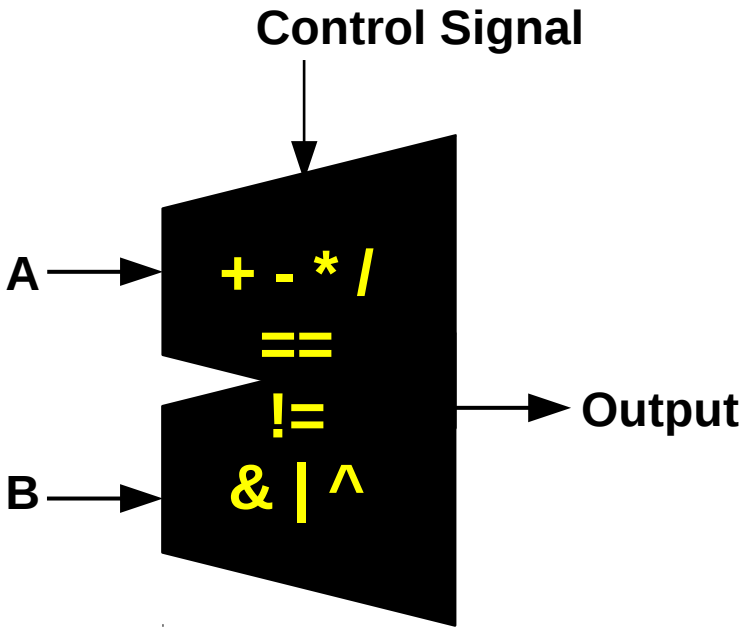
Datapath and Control Unit

- Datapath: Components in which instructions and data are handled in a processor
  - Register file, ALU, Special registers, ...

Datapath and Control Unit

- Datapath: Components in which instructions and data are handled in a processor
  - Register file, ALU, Special registers, ...
- Control Unit: Manages these components
  - Ensures correct execution of the instruction

Control Unit – Example Use



Arithmetic and Logic Unit

Computer System