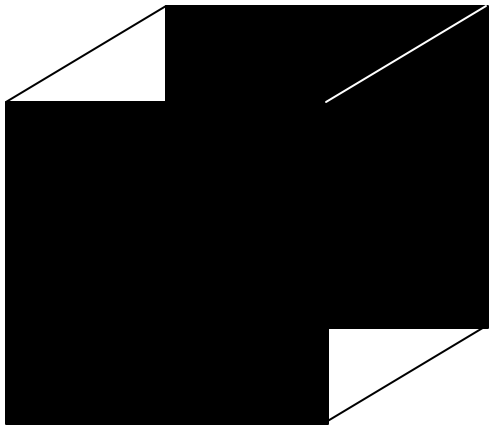
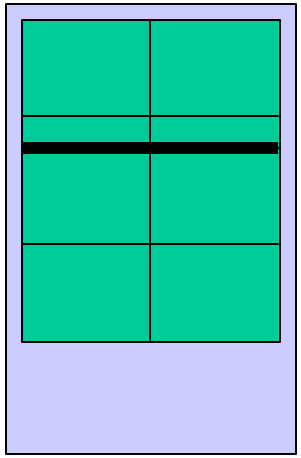
# Simple CPU

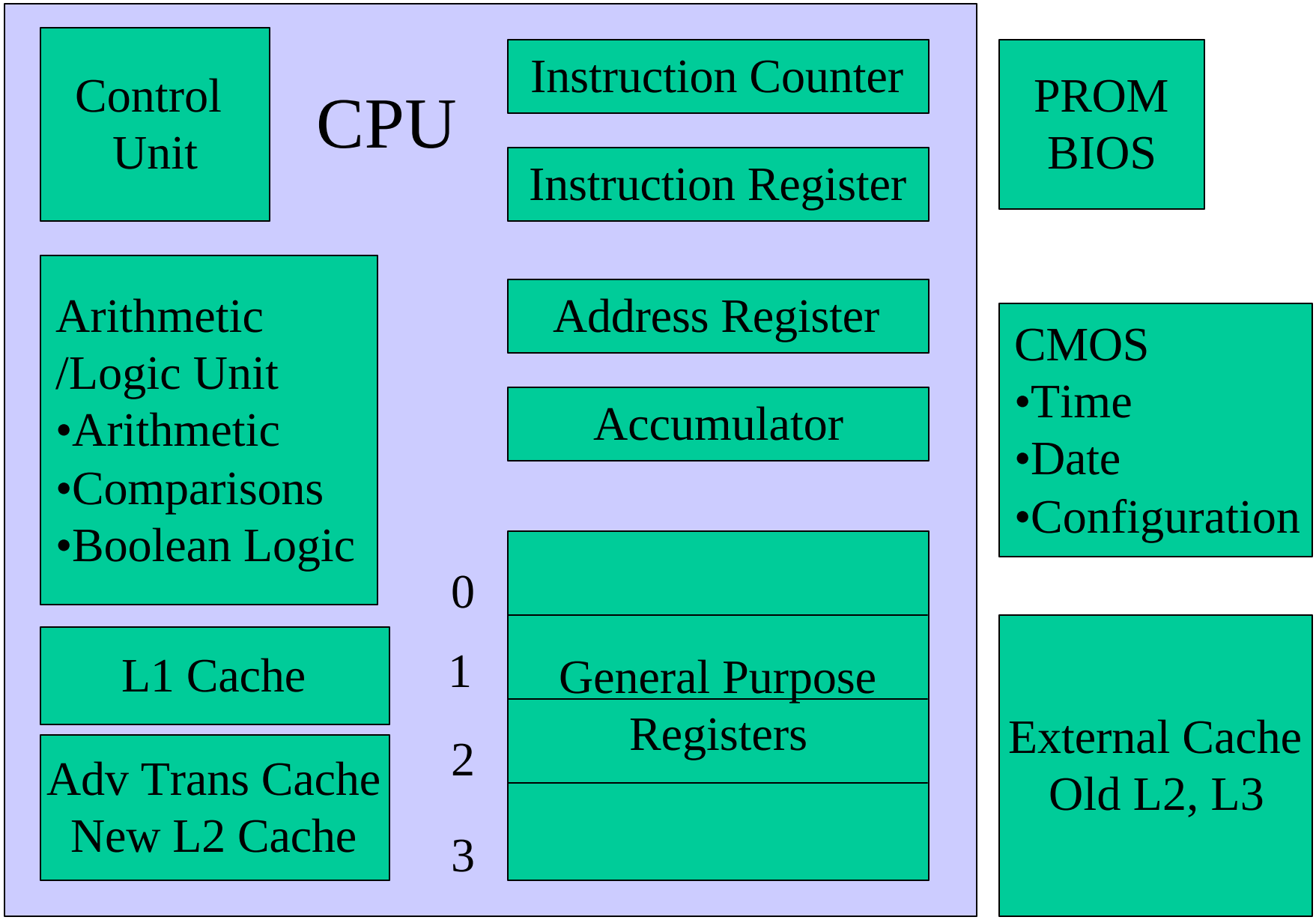


Black Box

or



Open Window


Control Unit
CPU
Instruction Counter
Instruction Register
PROM BIOS
Arithmetic /Logic Unit
•Arithmetic
•Comparisons
•Boolean Logic
Address Register
Accumulator
CMOS
•Time
•Date
•Configuration
0
1
2
3
General Purpose Registers
L1 Cache
Adv Trans Cache New L2 Cache
External Cache Old L2, L3

# Instruction Counter (IC)

- This special register holds the address of the next instruction to fetch.
- This automatically increments after fetching an instruction.
- The results of comparison instruction execution are stored in the Instruction Counter.

# Instruction Register (IR)

- This register holds the bit pattern retrieved from memory during the instruction FETCH cycle.
- The CPU attempts to interpret contents of the instruction register as a machine language instruction.

# Bit Patterns in Memory

- Until a bit pattern is loaded into the instruction, the computer neither knows, nor cares, whether that bit pattern represents and instruction or data.
- Sometimes, a bit pattern is used both as an instruction and as data.
  - Dynamic Programming routinely does this.

# Address Register (ADDR)

- This register holds the address of the data to be fetched from memory.
- The value loaded into the address register is taken from the address field of an instruction.

# Accumulator (Acc)

- The accumulator holds data that the Arithmetic / Logic Unit operates on.
- The results of arithmetic and Boolean operations are stored in the Accumulator.

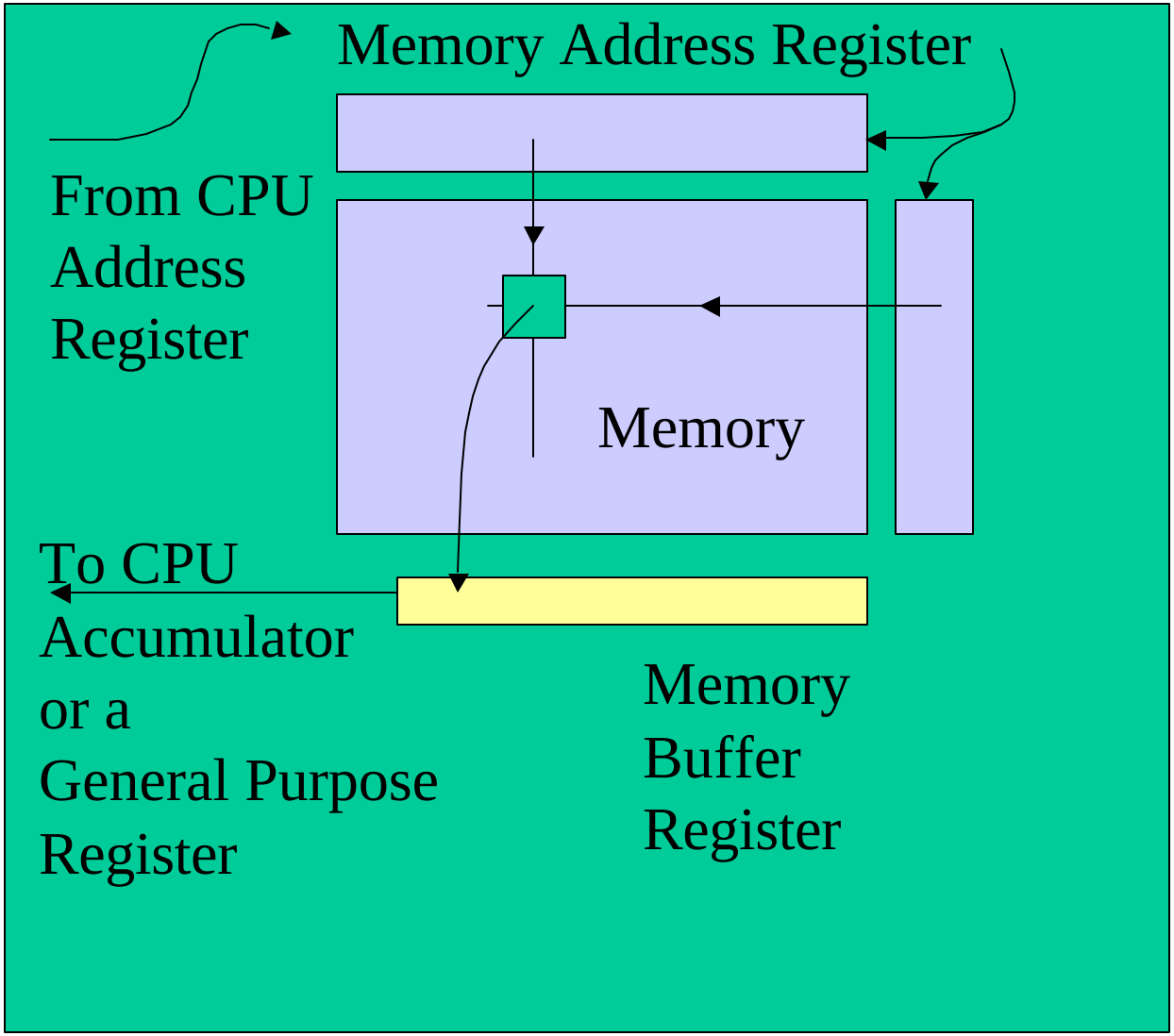
# Accumulator Features

- The Accumulator generally has more bits than normal memory locations to hold extra precision results from multiplication and division operations.
- Special hardware logic associated with the Accumulator is used to do special operations on the data in the Accumulator.
- Faster than main memory.

# General Purpose Registers

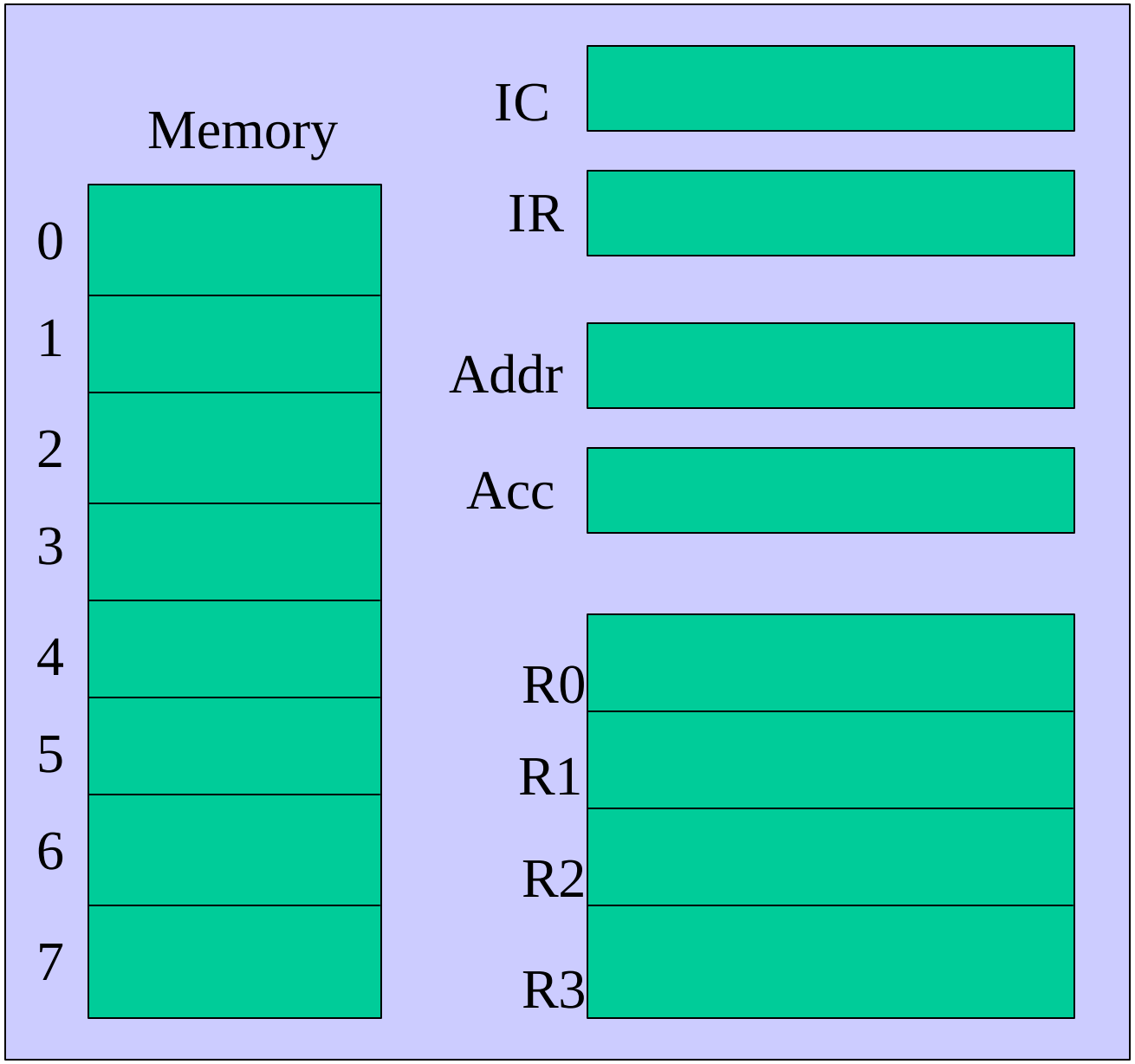
- Used to temporarily hold counters, addresses, and results of computation.
- Word size is generally larger than main memory locations.
- Often, special logic supports special operations between registers.
- Faster than main memory.
- Small microprocessors usually have at least 8 general purpose registers.

# Simple Memory Module

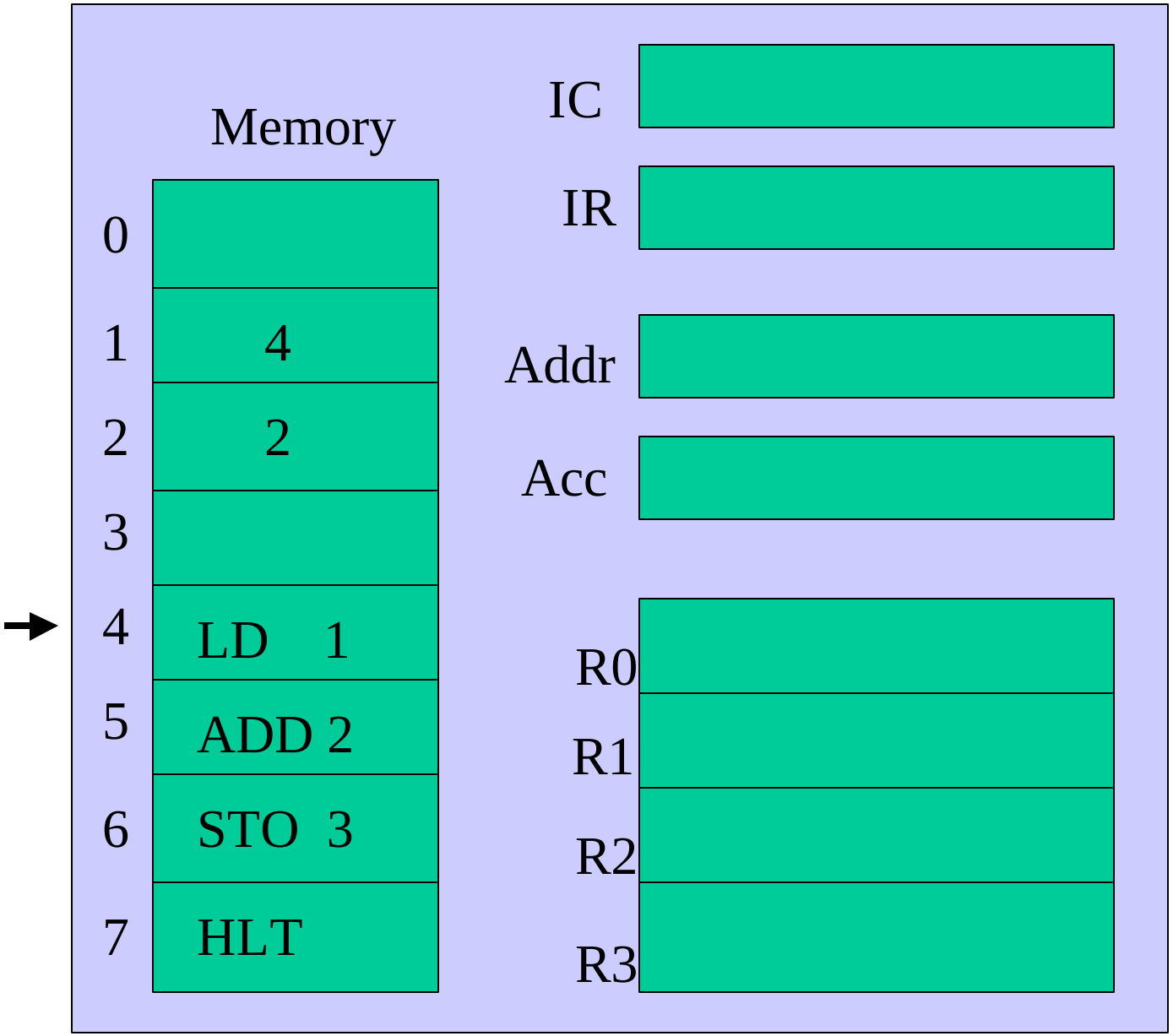


# Instruction Anatomy

- Our simple instructions have two parts in the format of OPR X : Operation Code|Argument Field
- Test and Jump instructions modify the Instruction Counter.
- Immediate instructions load the argument into the Accumulator.
- Other instructions load the argument into Address Register.
- Real machine instructions are often more complicated.

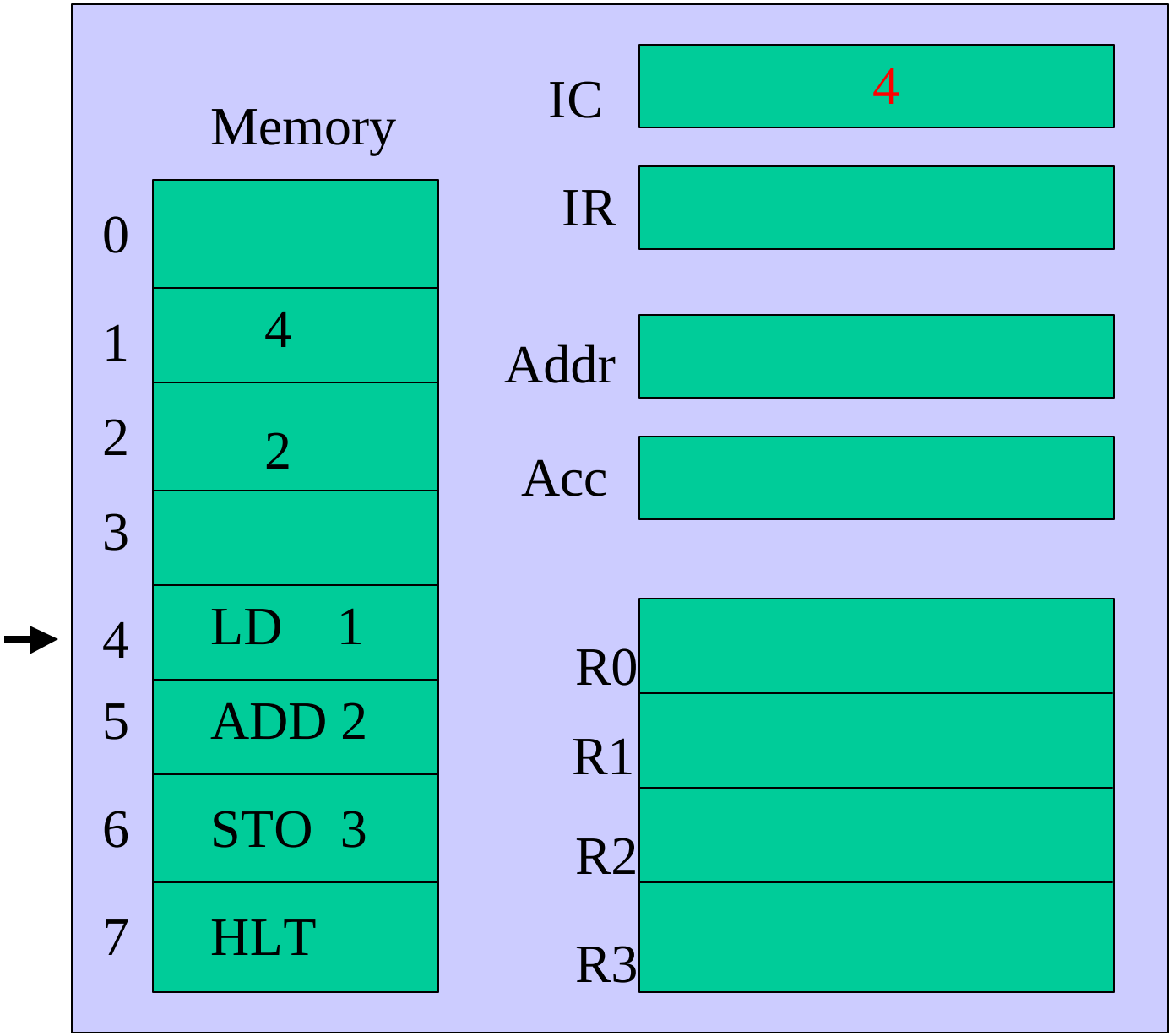



```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```



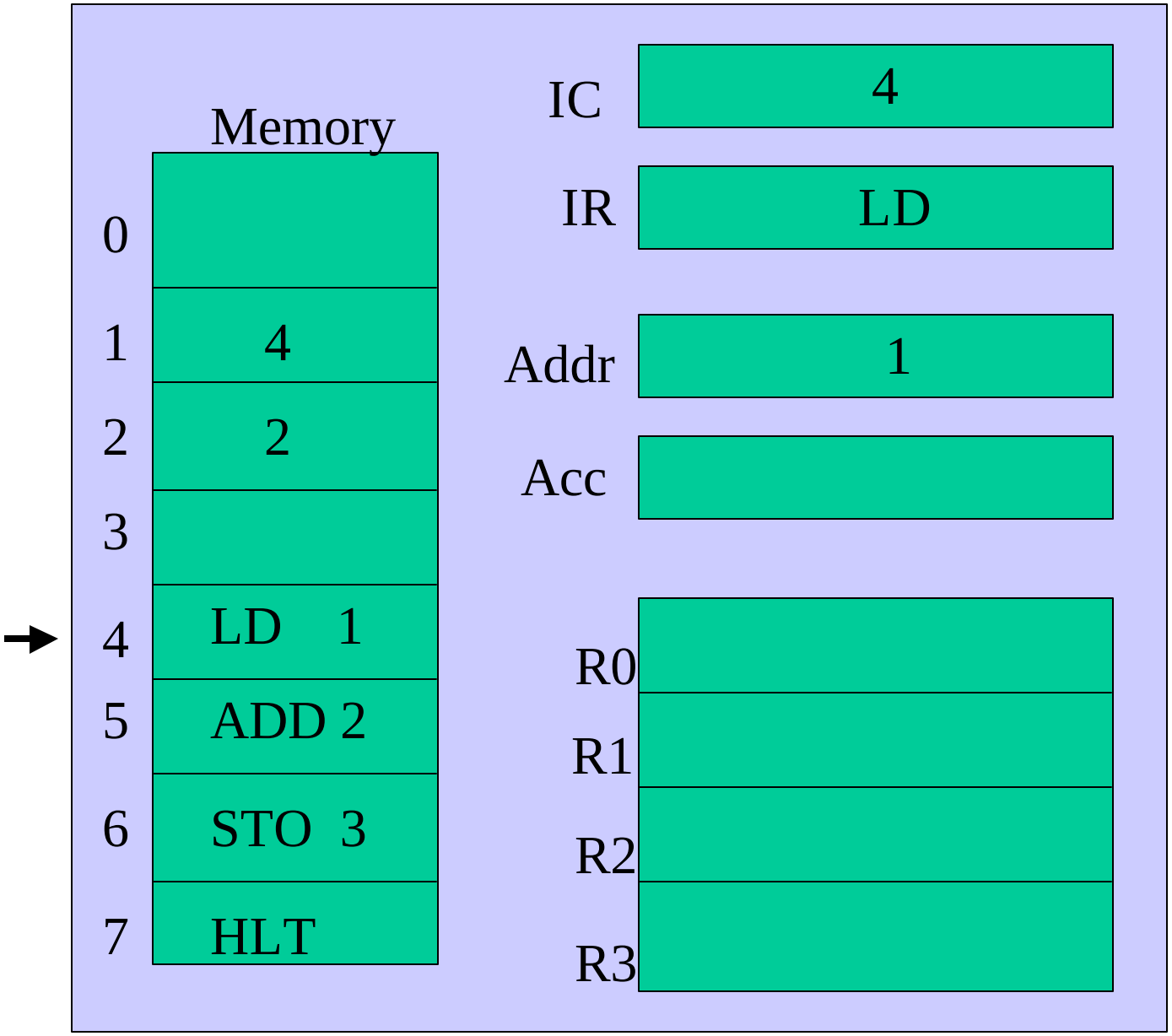

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Load program into memory, starting at ORiGin location 1.



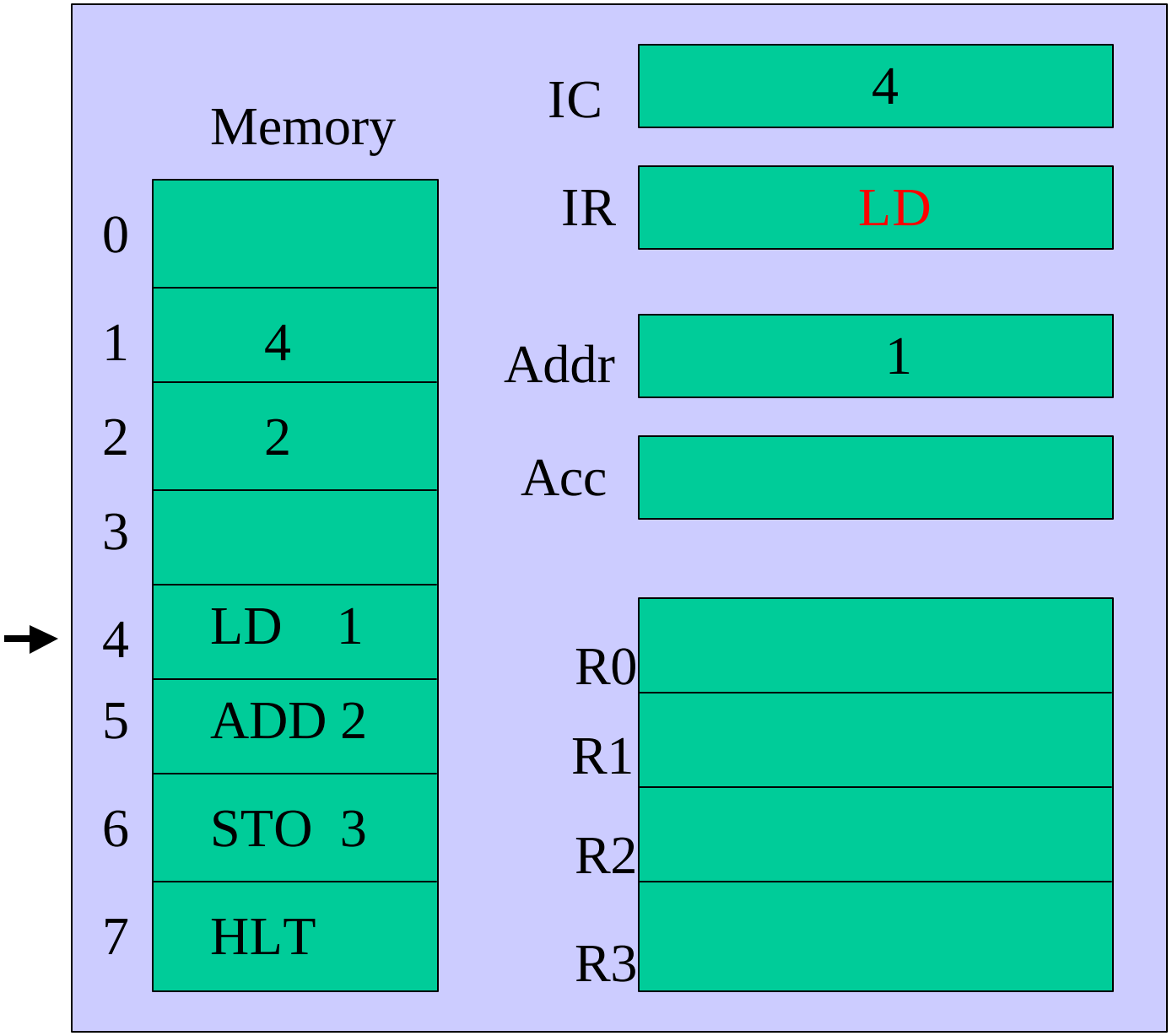

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

To start program, load “4” into the Instruction Counter (IC)



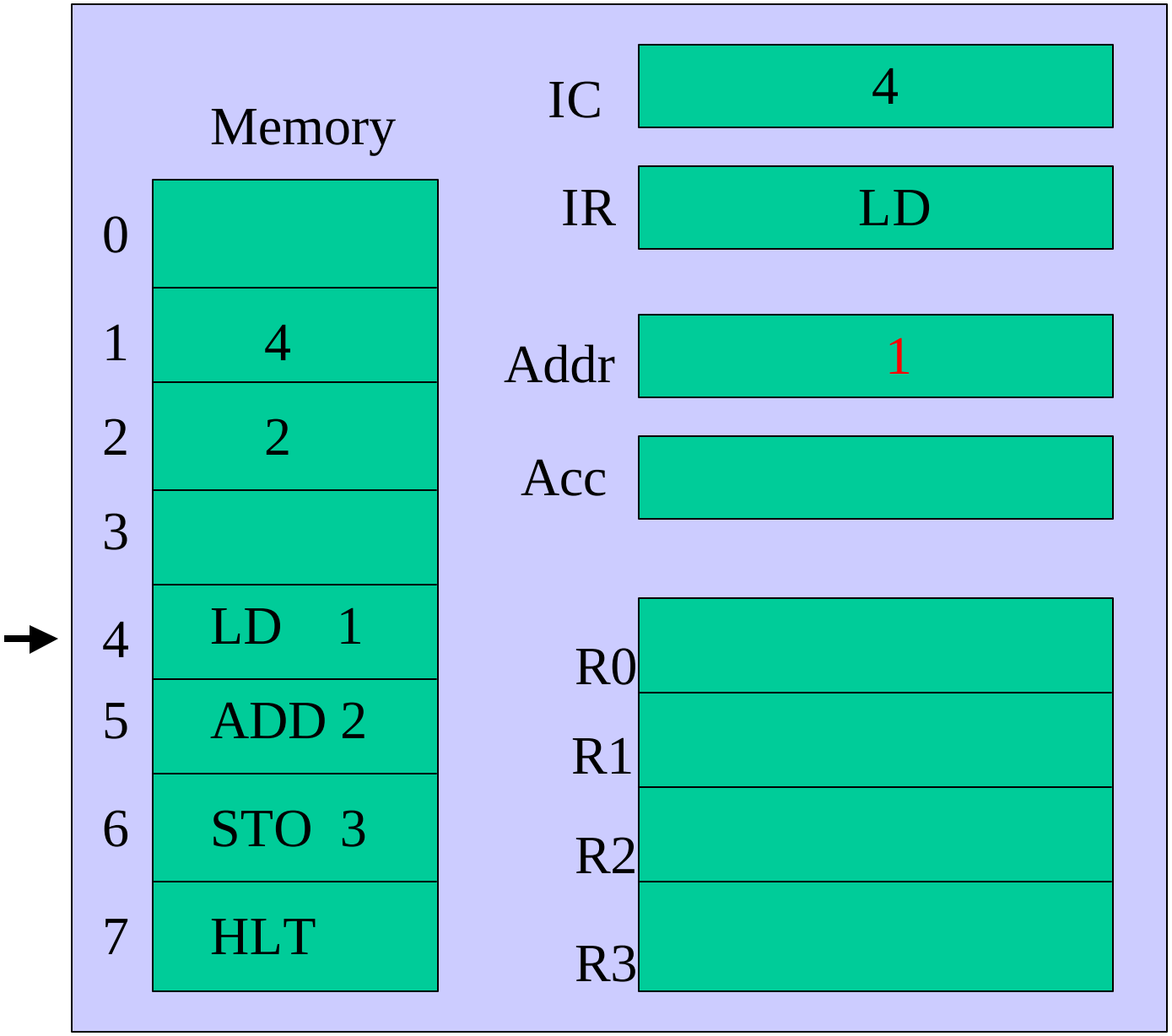

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Fetch the contents of memory location 4.



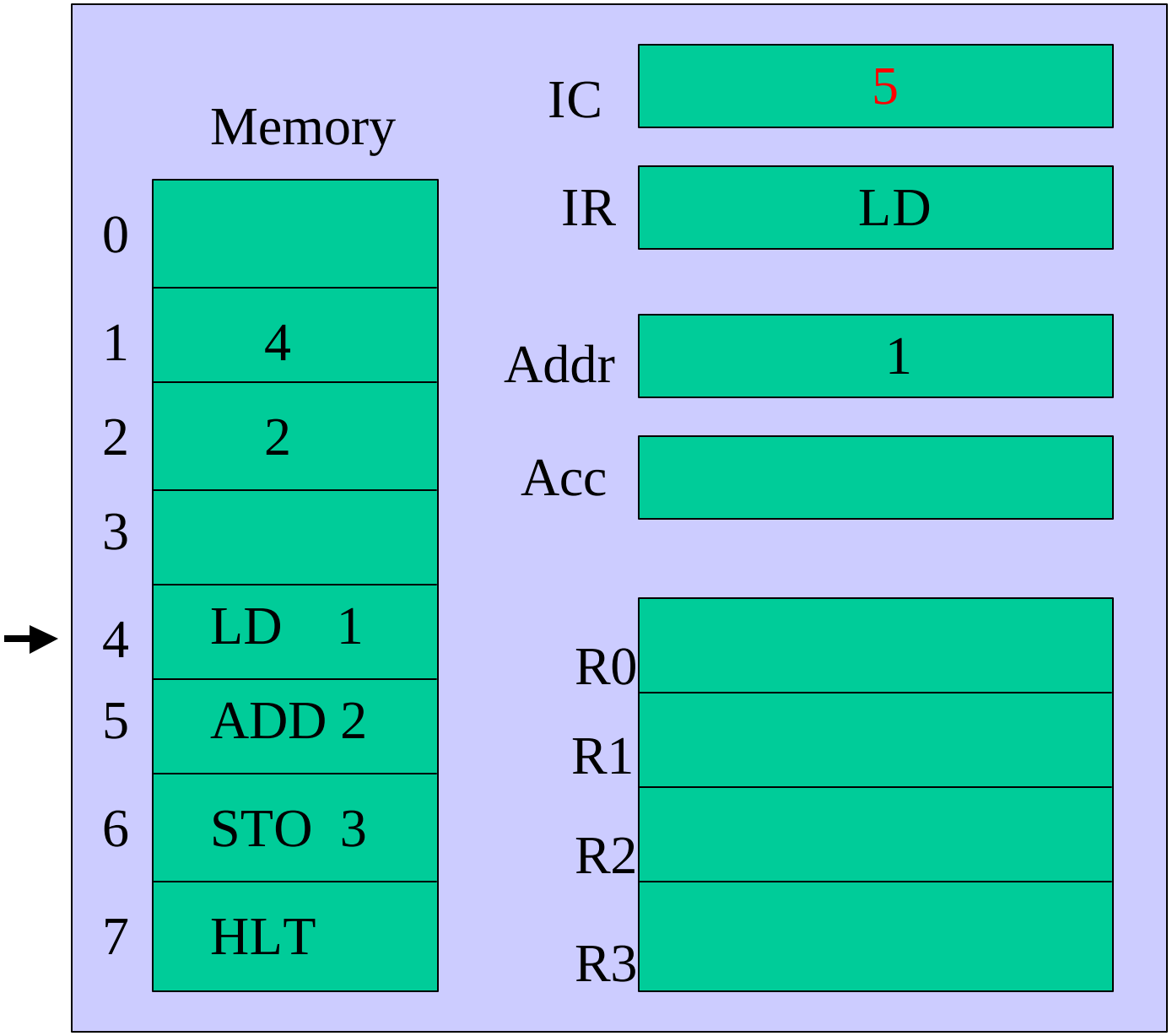

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Place the operation code into the Instruction Register (IR).



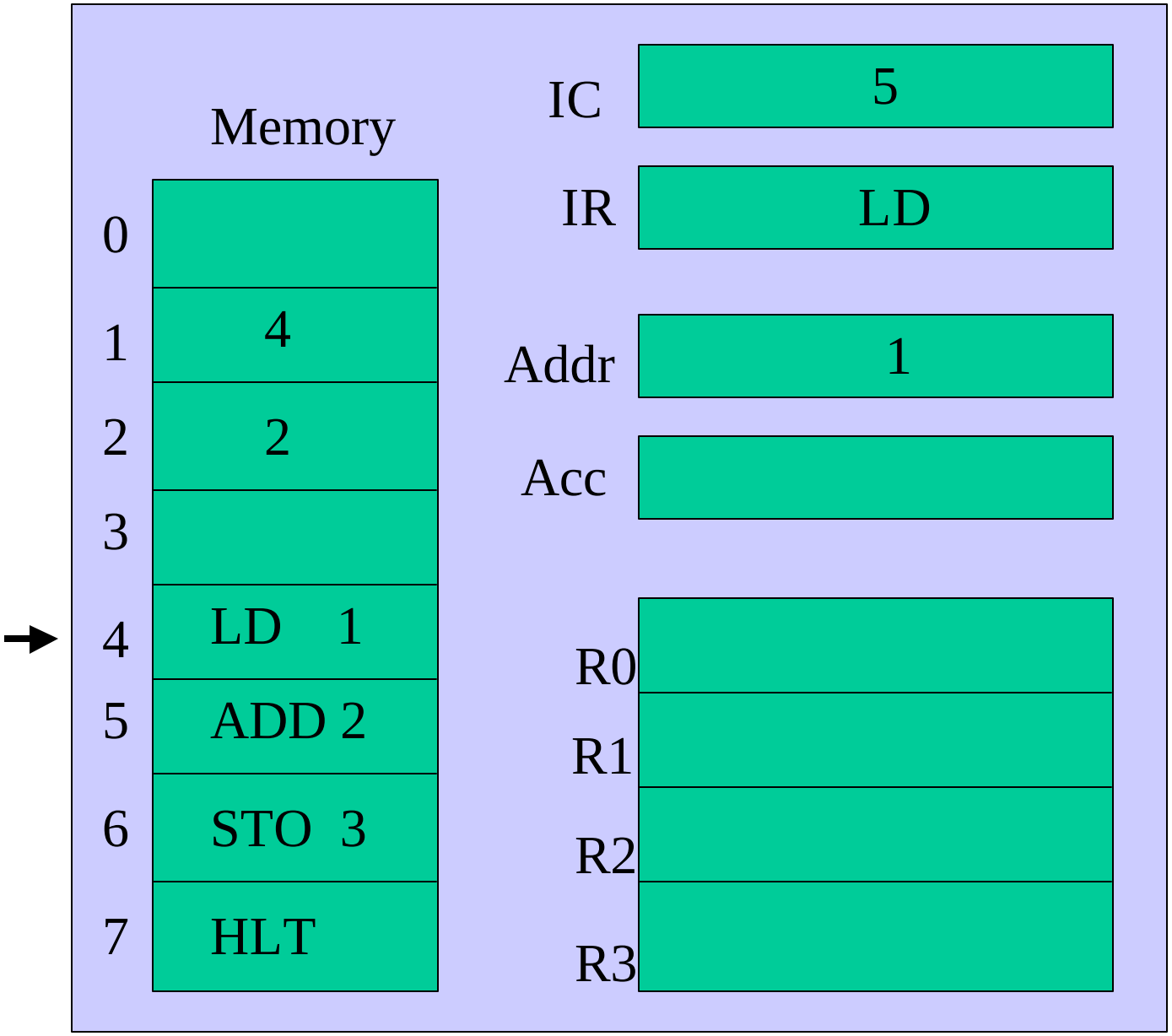

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Place the argument field into the Address Register (ADDR).



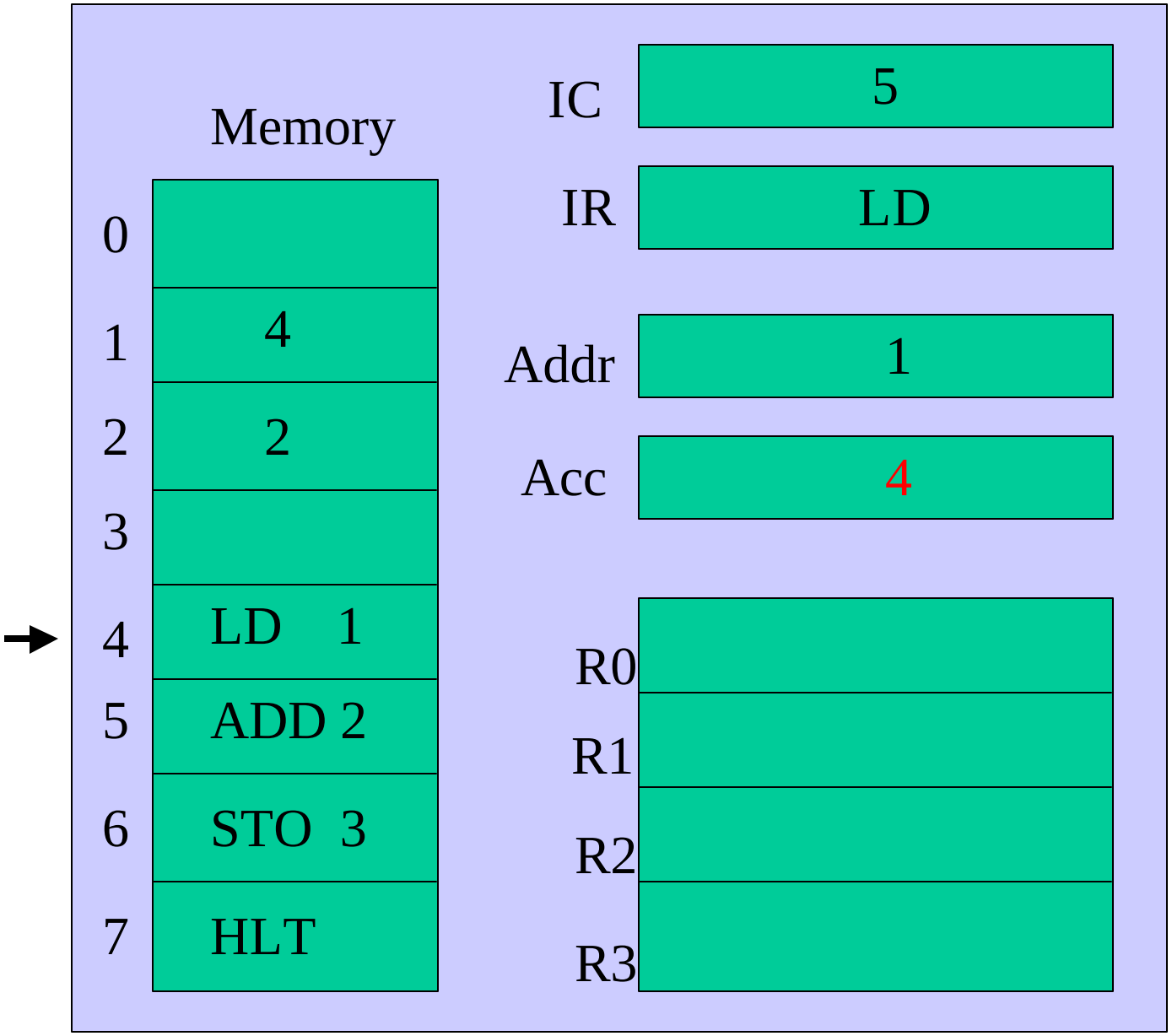

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Increment the Instruction Counter.



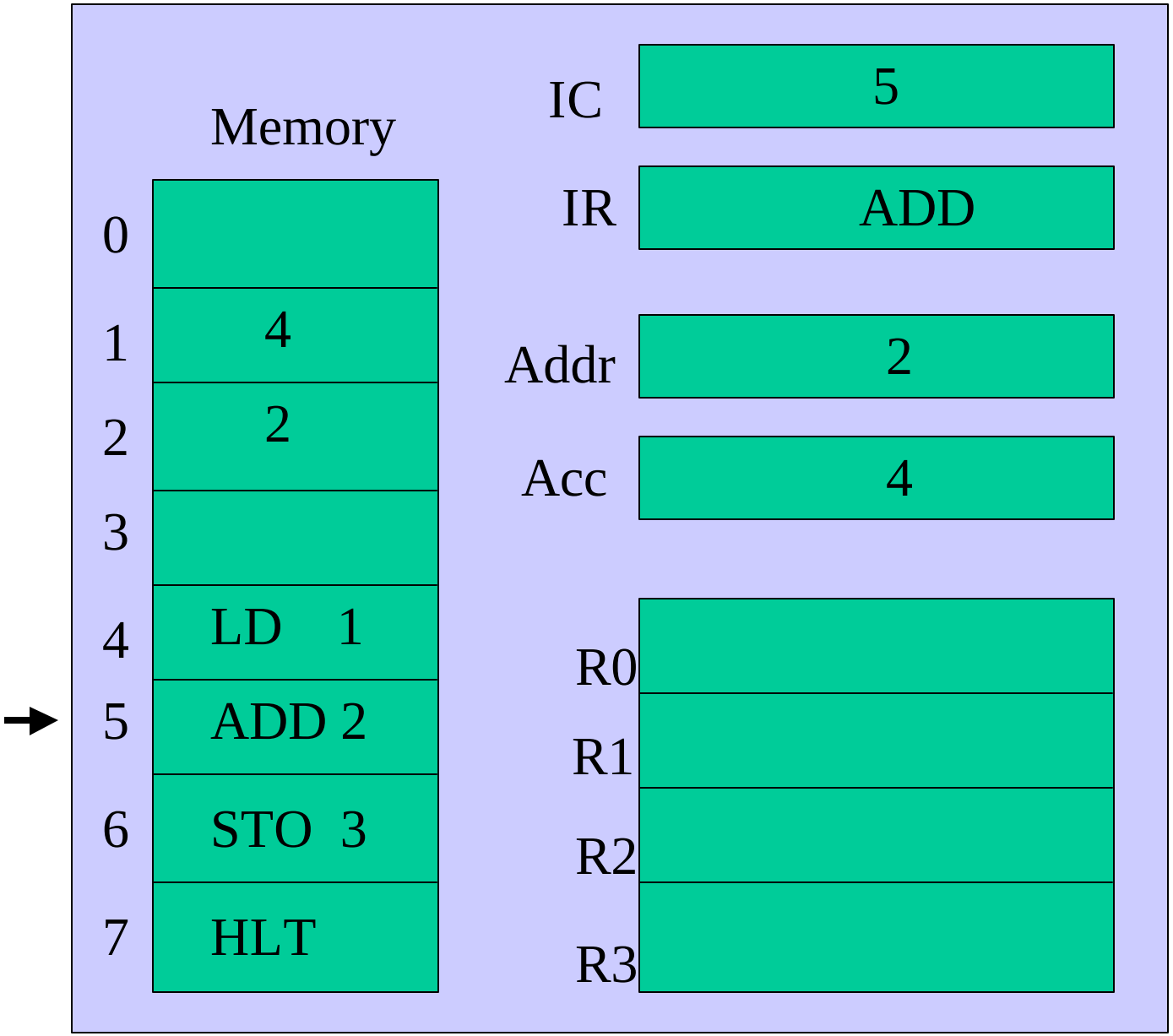

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Execute the Load instruction.



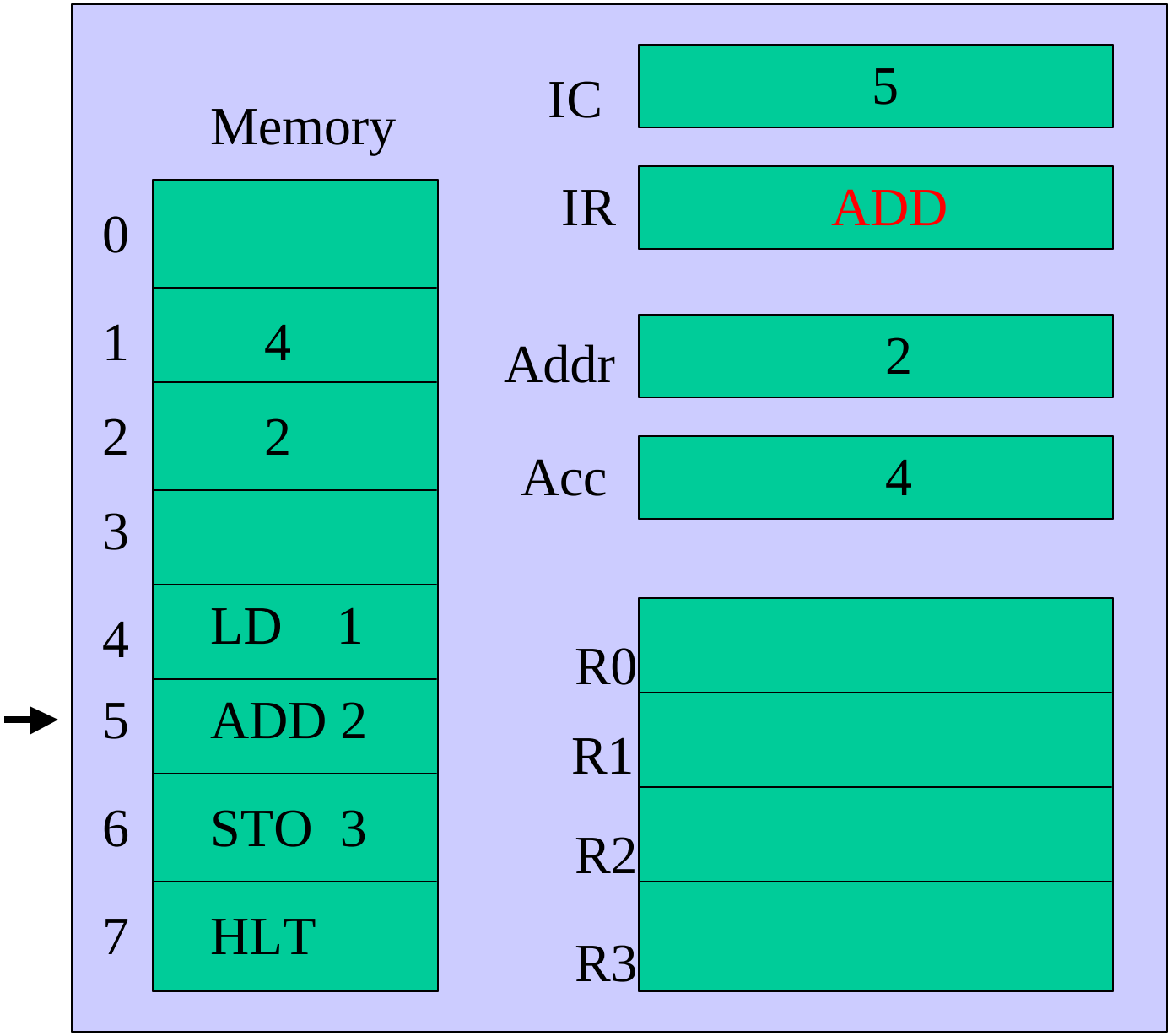

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Load contents of location 1 into the Accumulator.



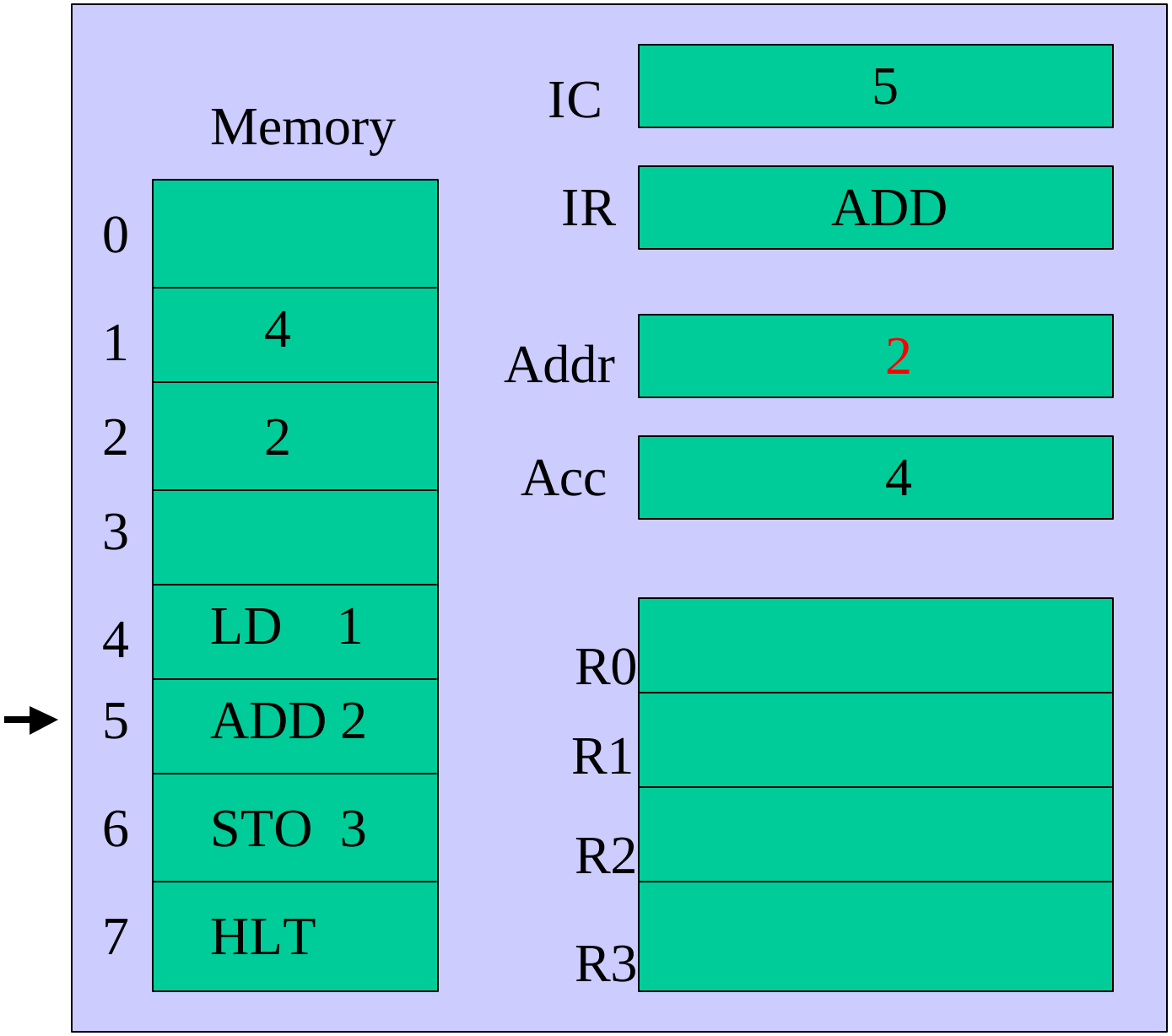

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Fetch the contents of location 5.



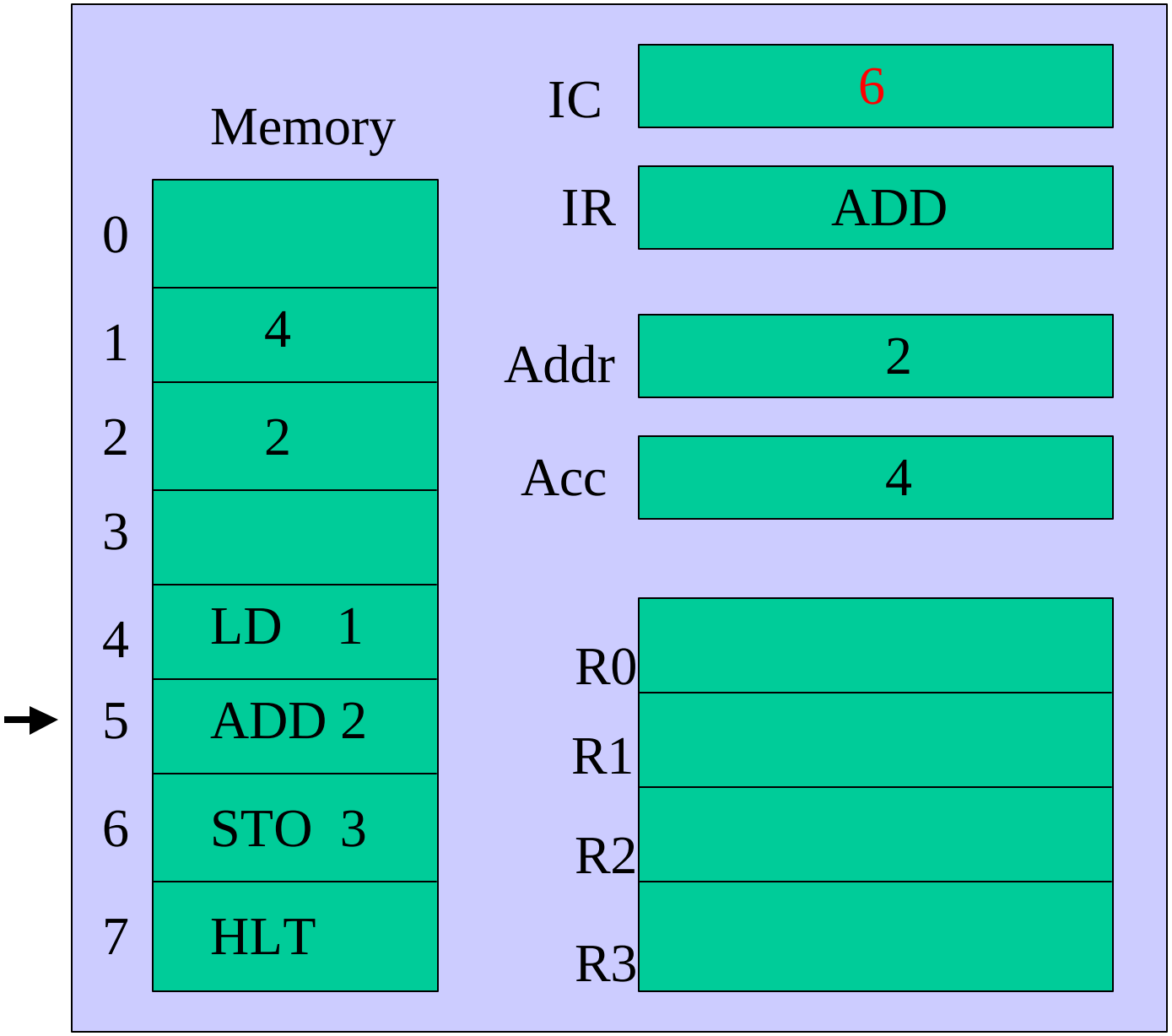

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Place the operation code into the Instruction Register.



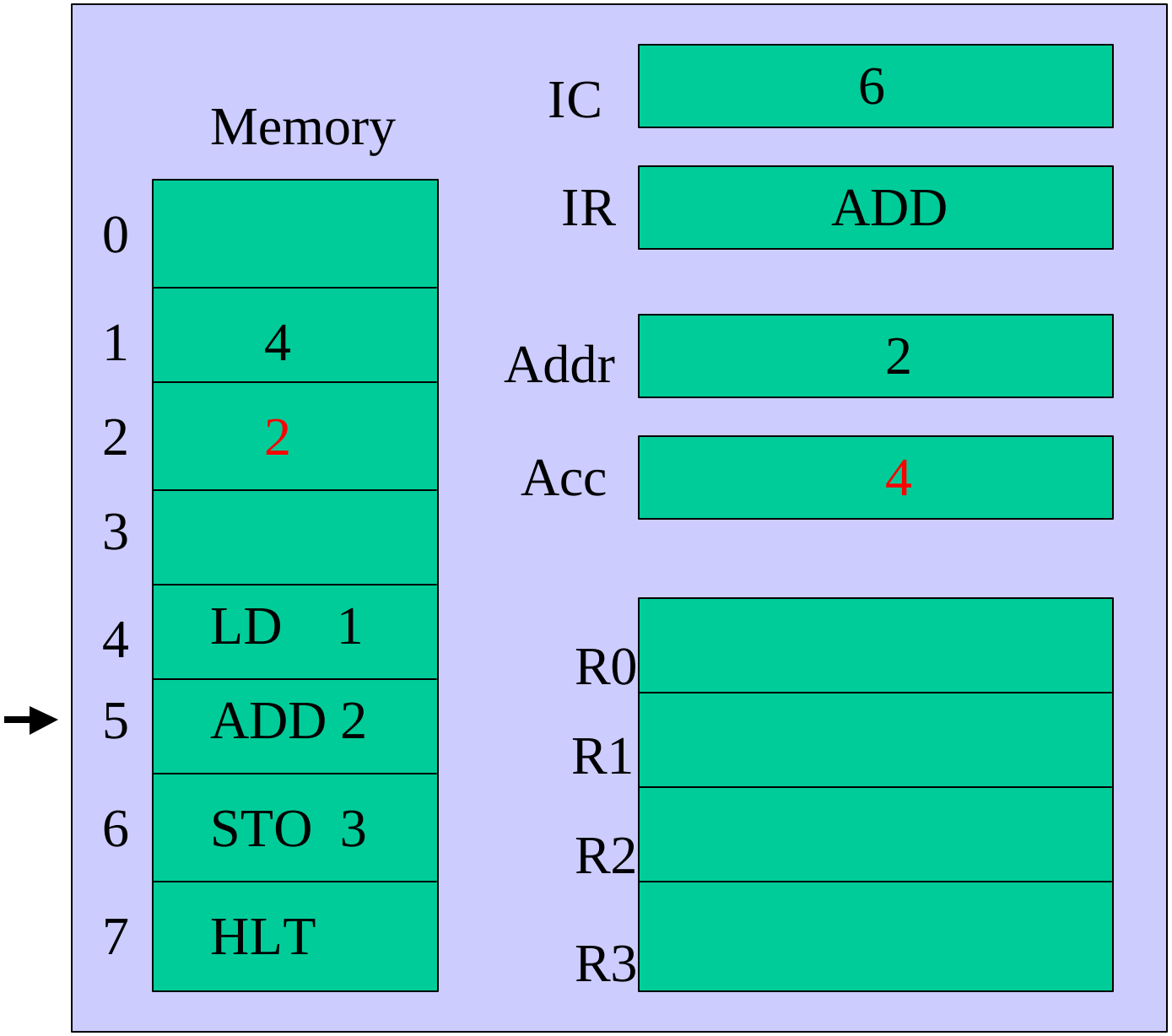

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Place the argument field into the Address Register.



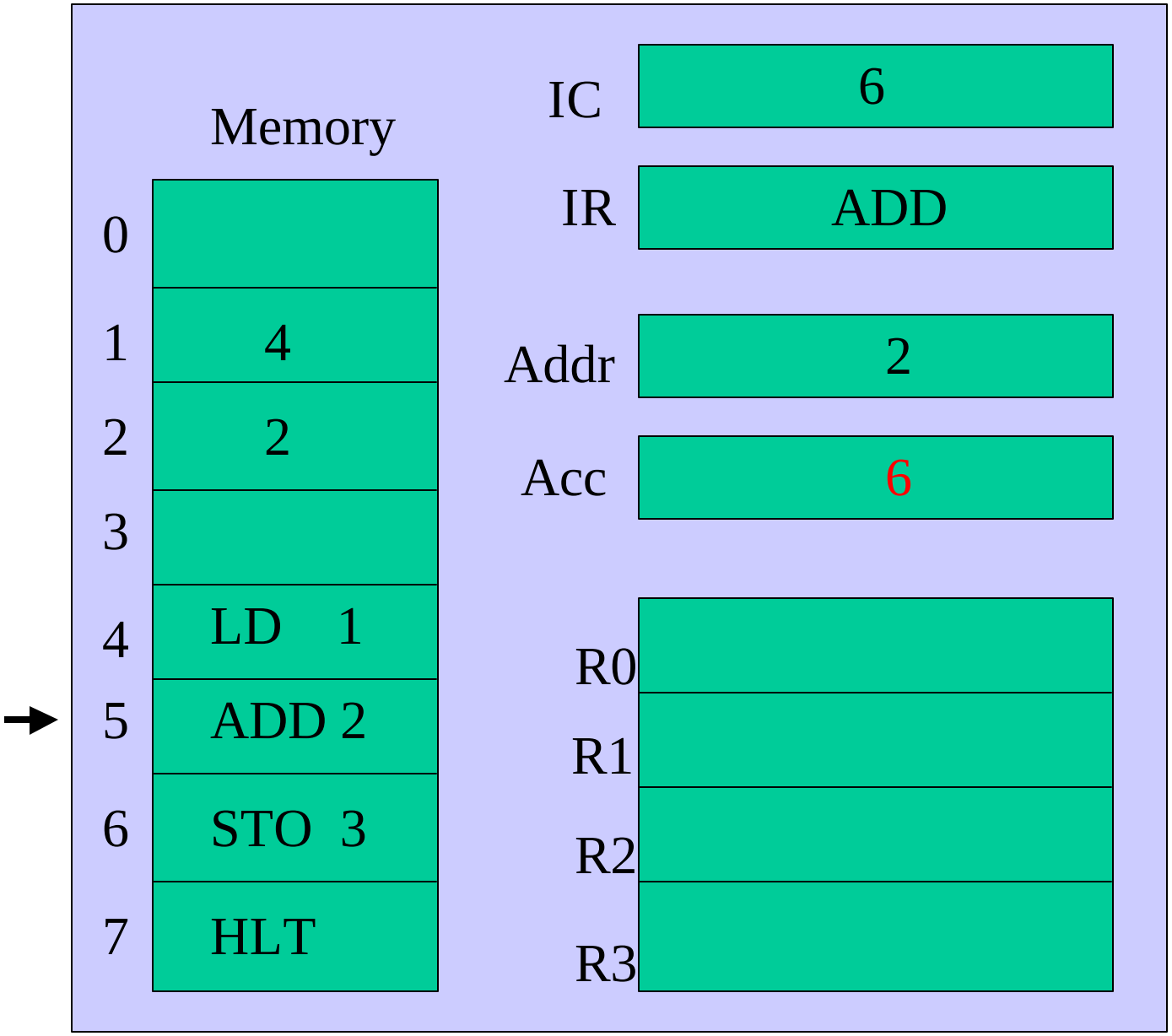

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Increment the Instruction Counter.



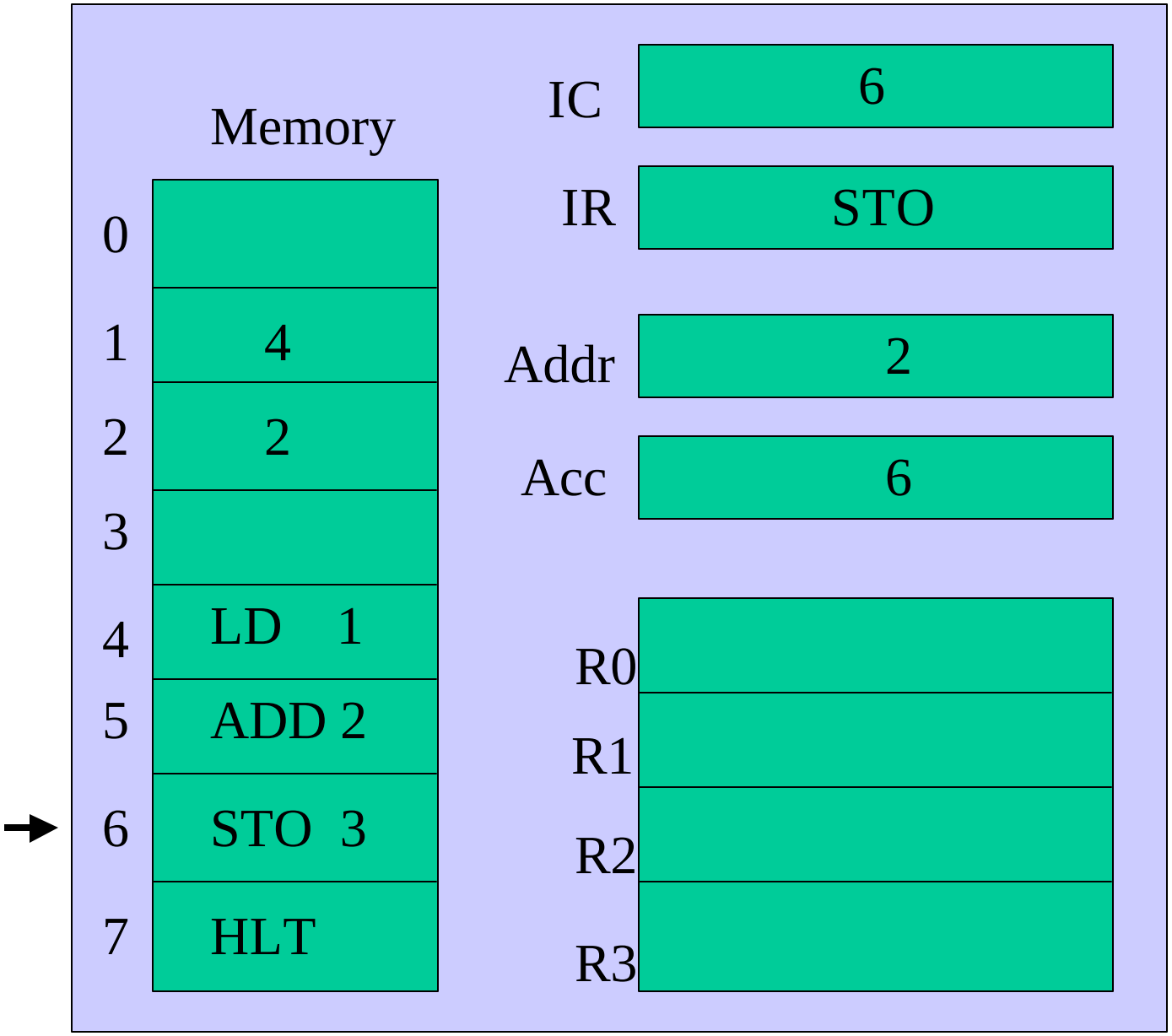

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Execute the ADD instruction.



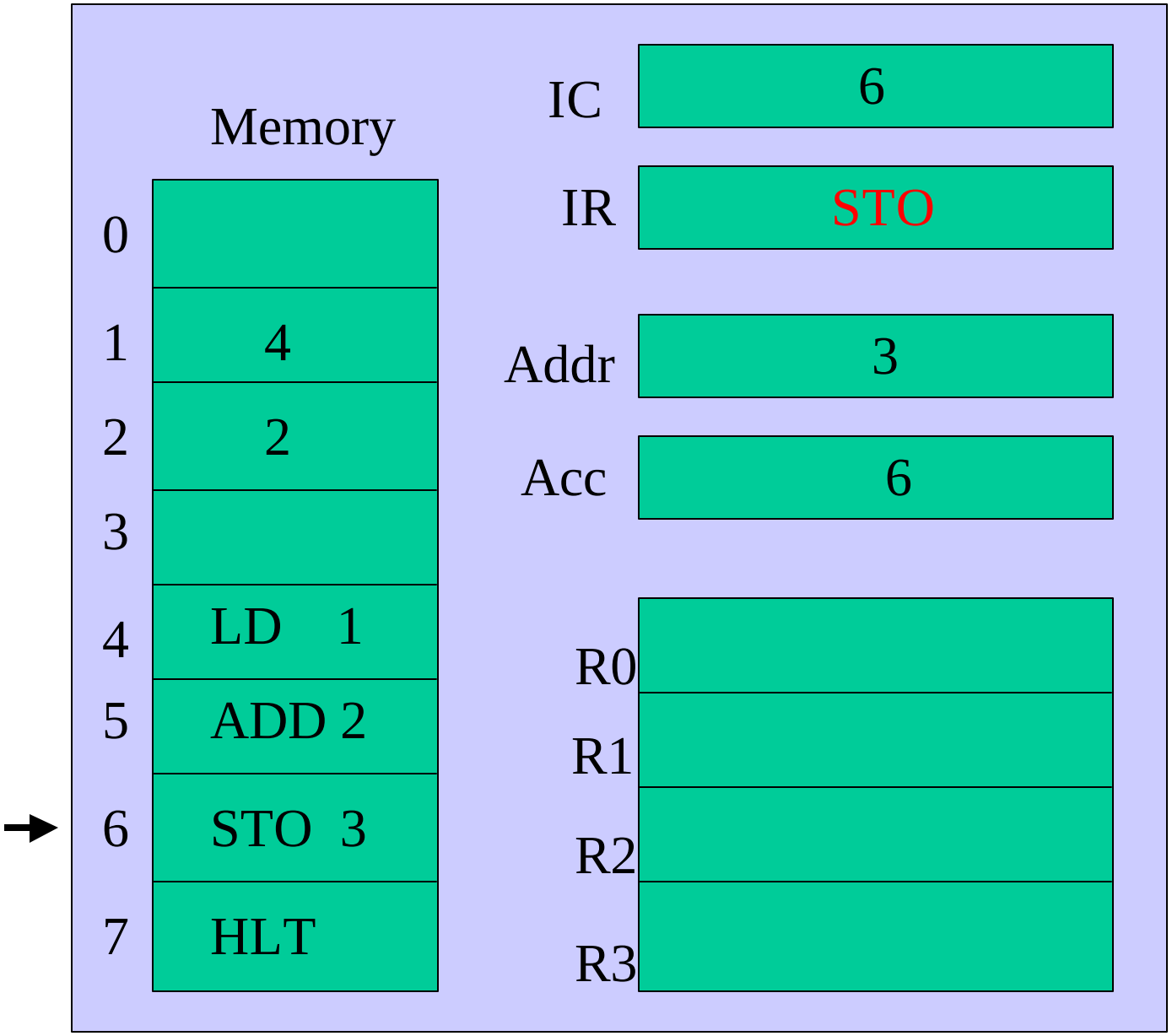

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Add contents of location 2 to the Accumulator.



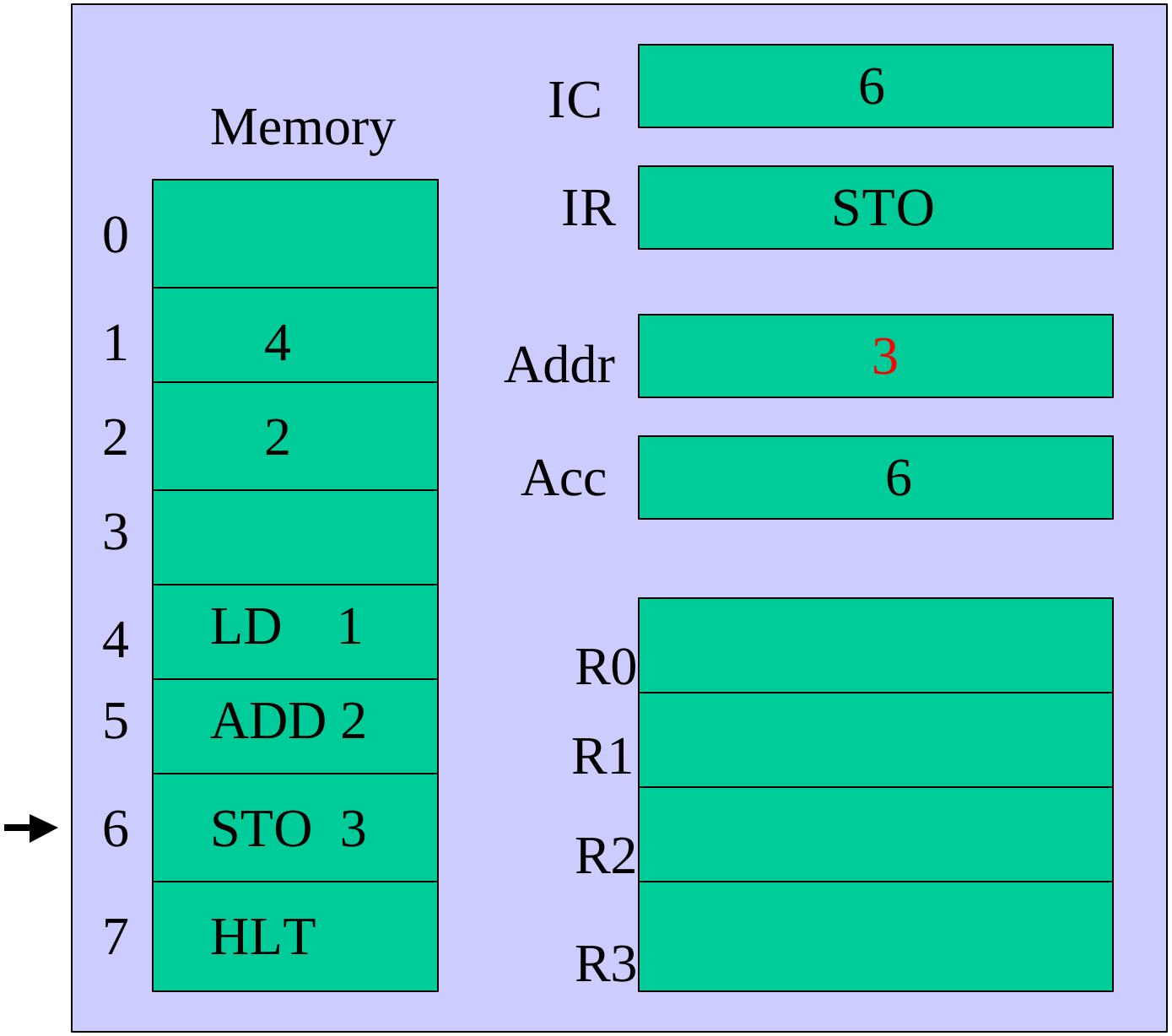

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Fetch memory location 6.



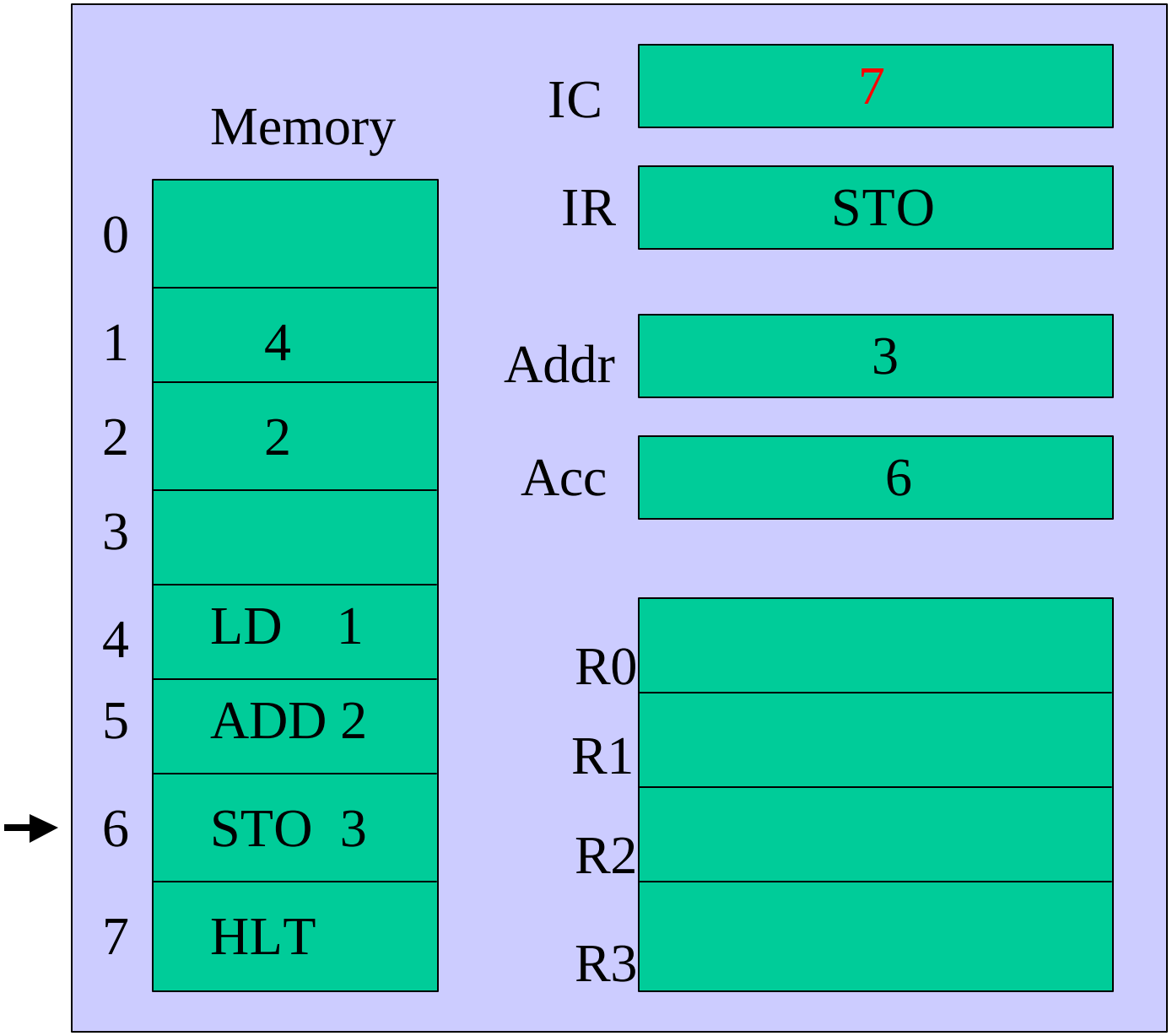

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Place the operation code into the Instruction Register.



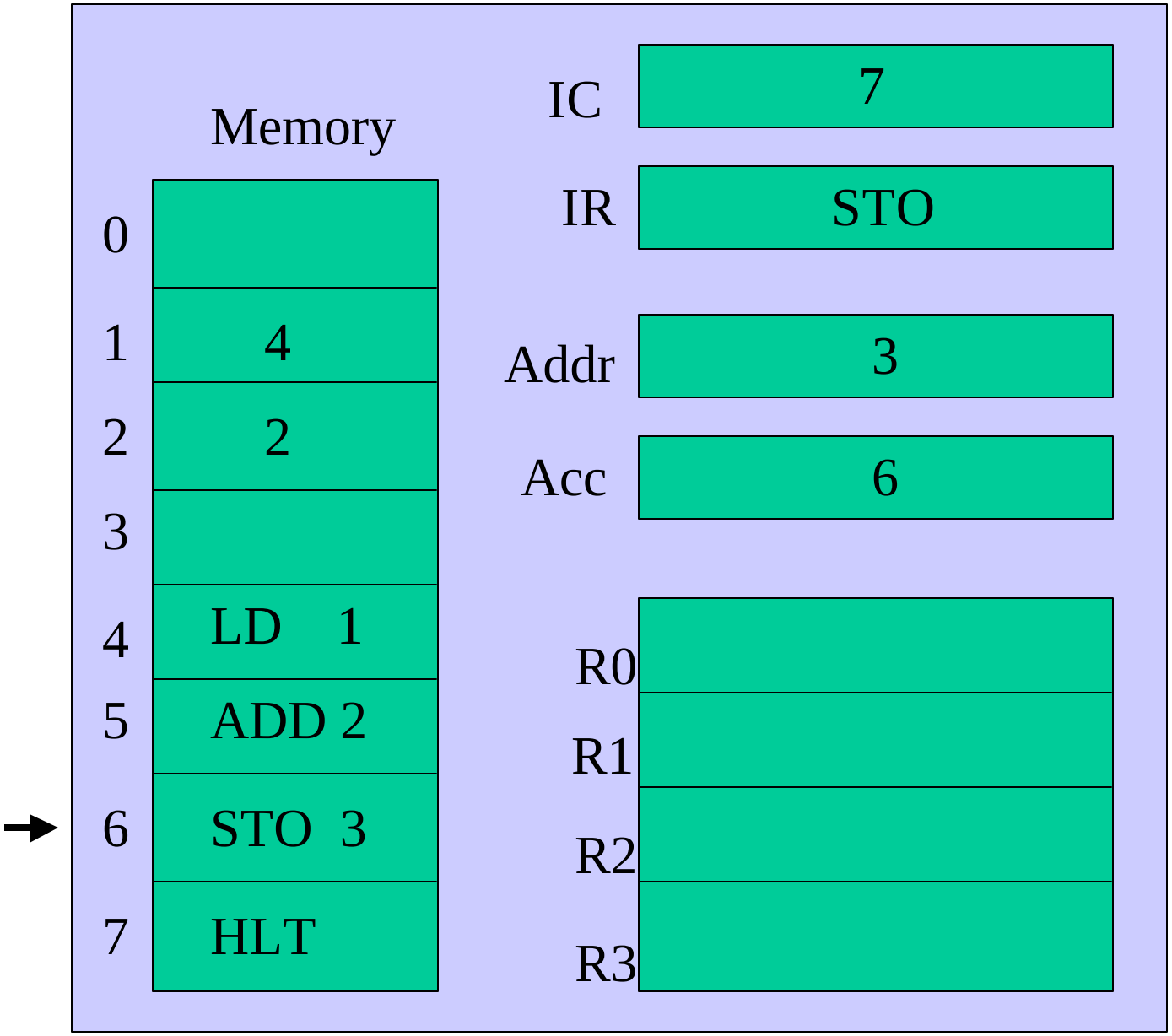

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Place the argument field into the Address Register.



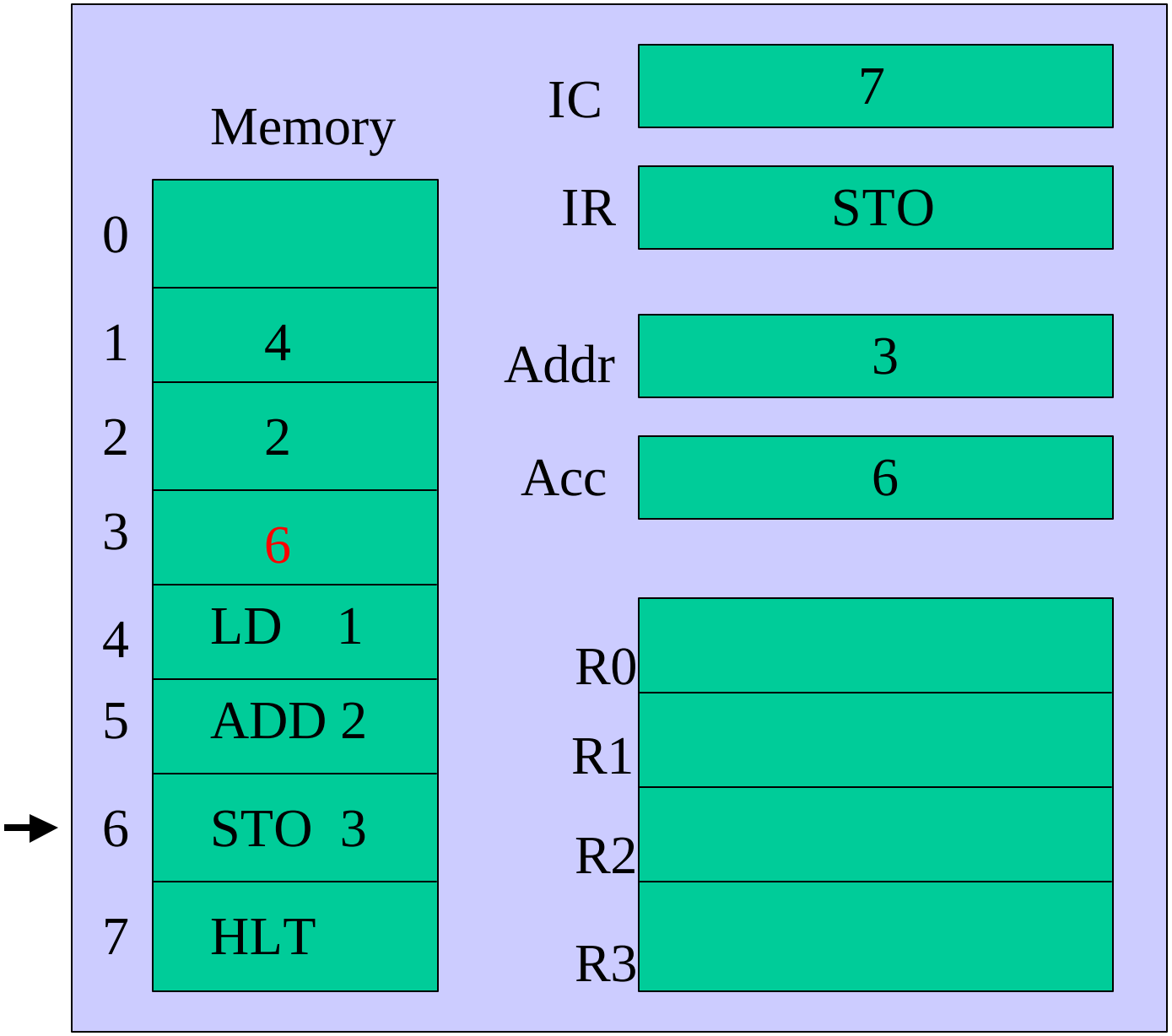

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Increment the Instruction Counter.



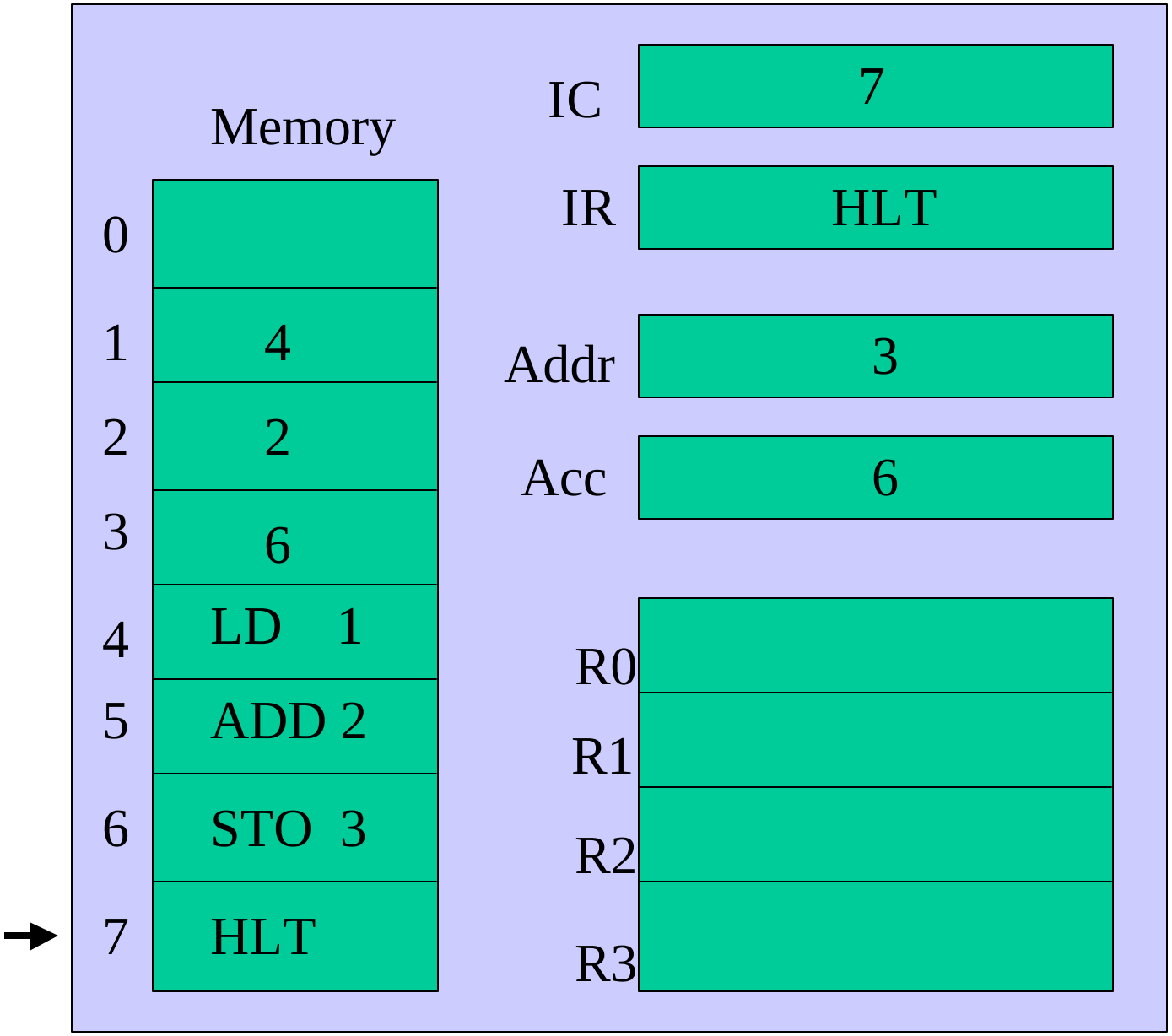

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Execute the instruction.



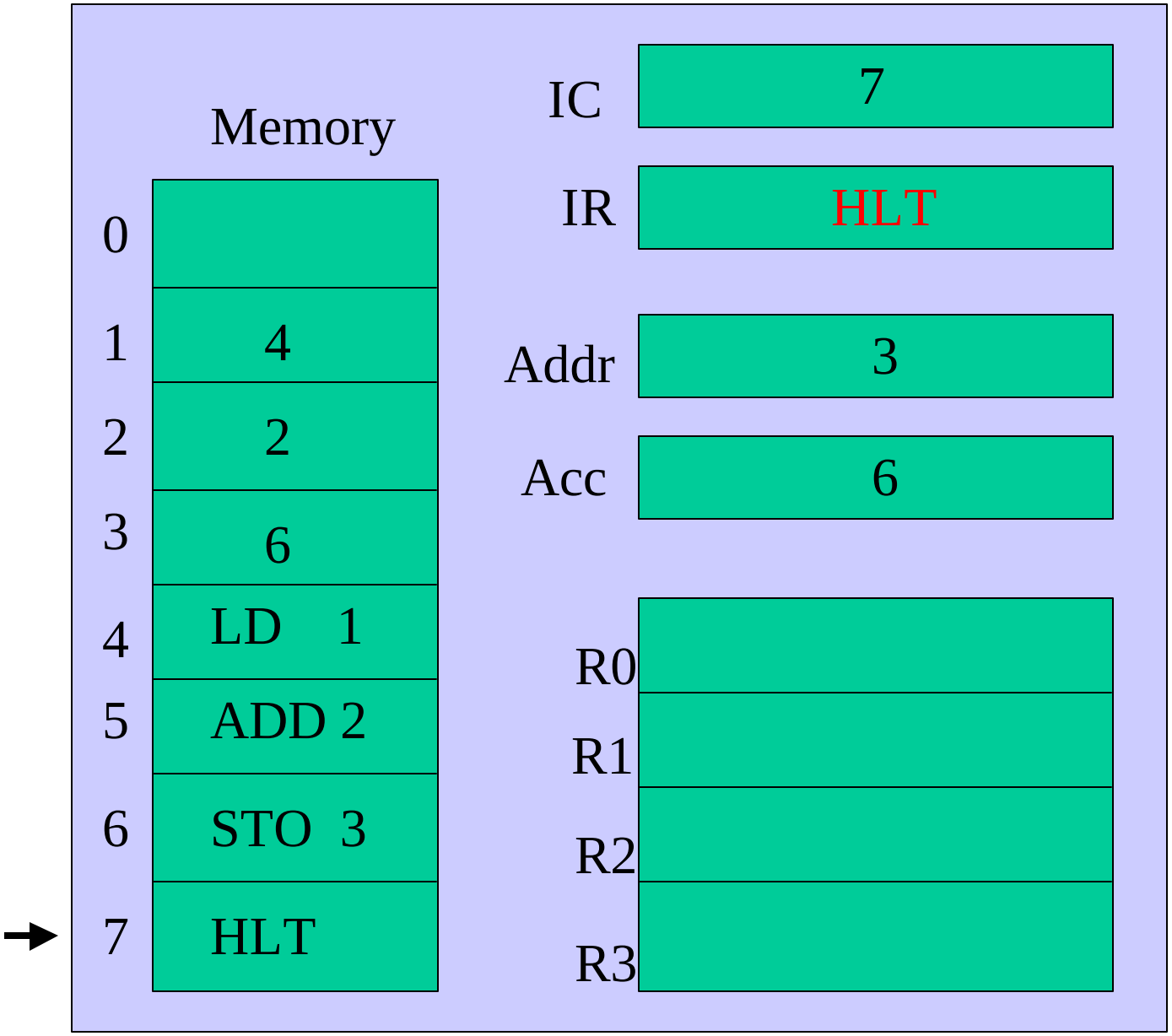

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Store the contents of the Accumulator into memory location 3.



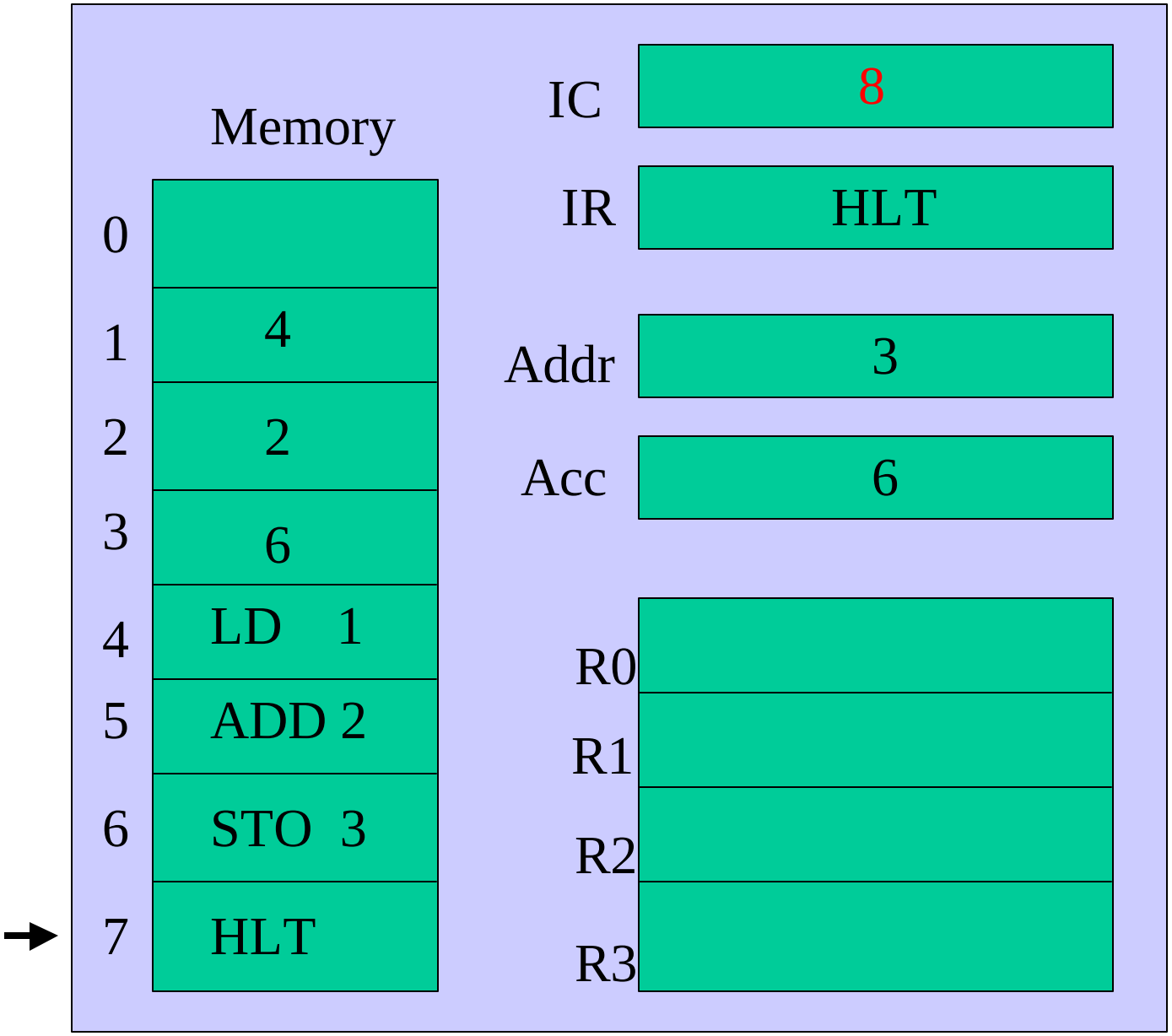

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Fetch the contents of memory location 7.



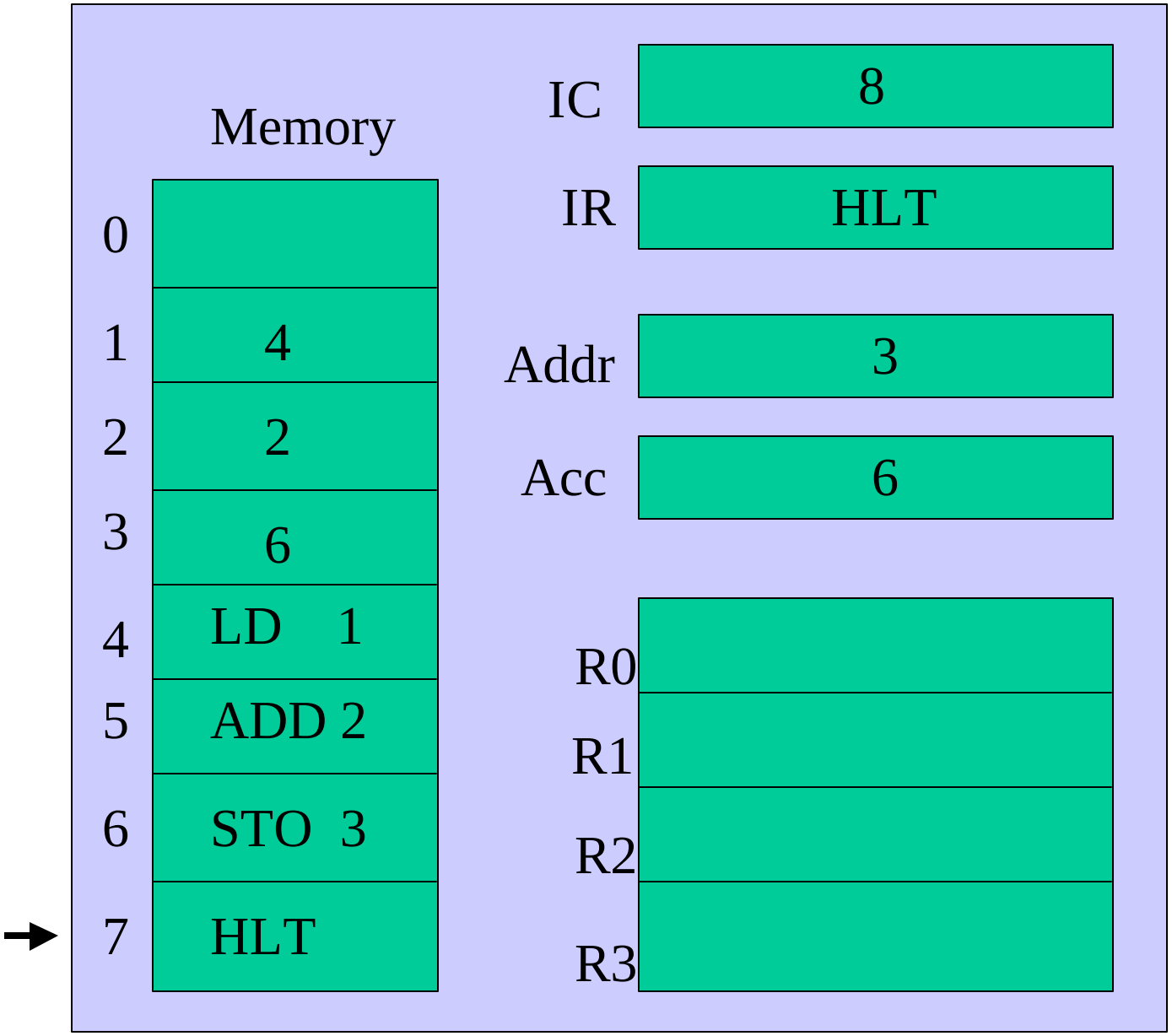

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Place operation code into the Instruction Register.




```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Increment the Instruction Counter (IC)

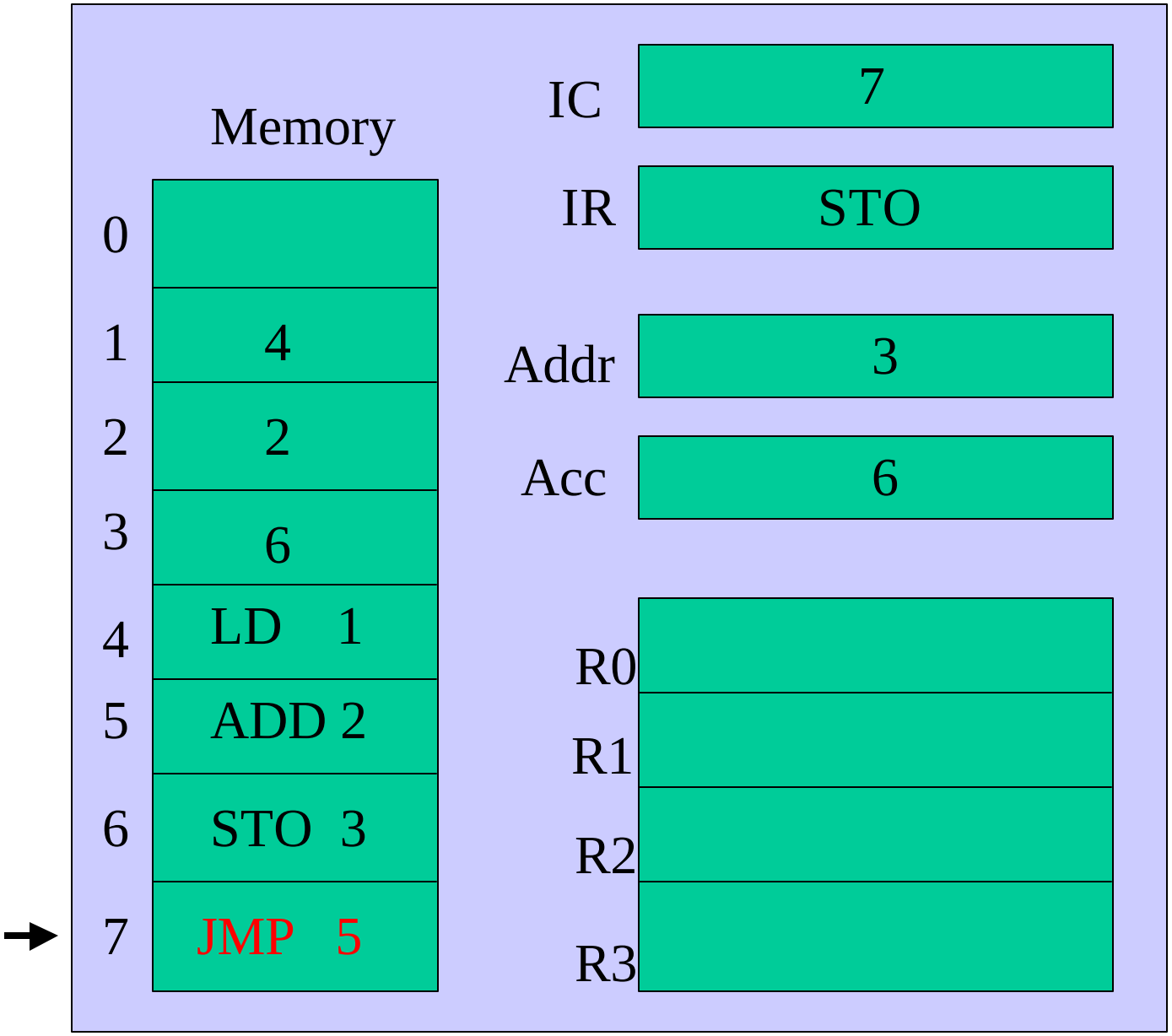



```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Execute the instruction.  Halt.

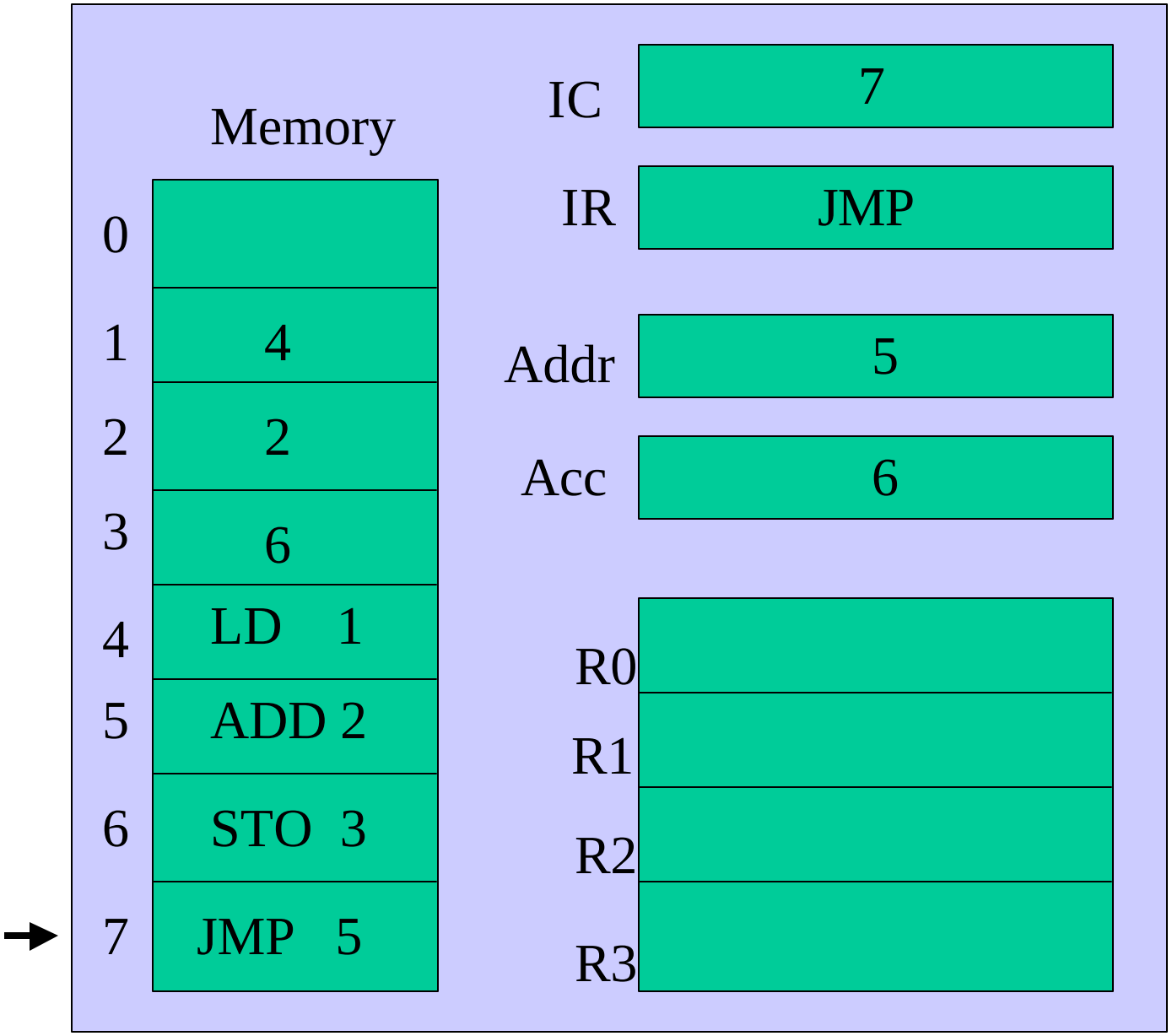
# Stored Instruction Program

- Advantage: You can change the order of execution of instructions.
- Unconditional or Conditional Jump



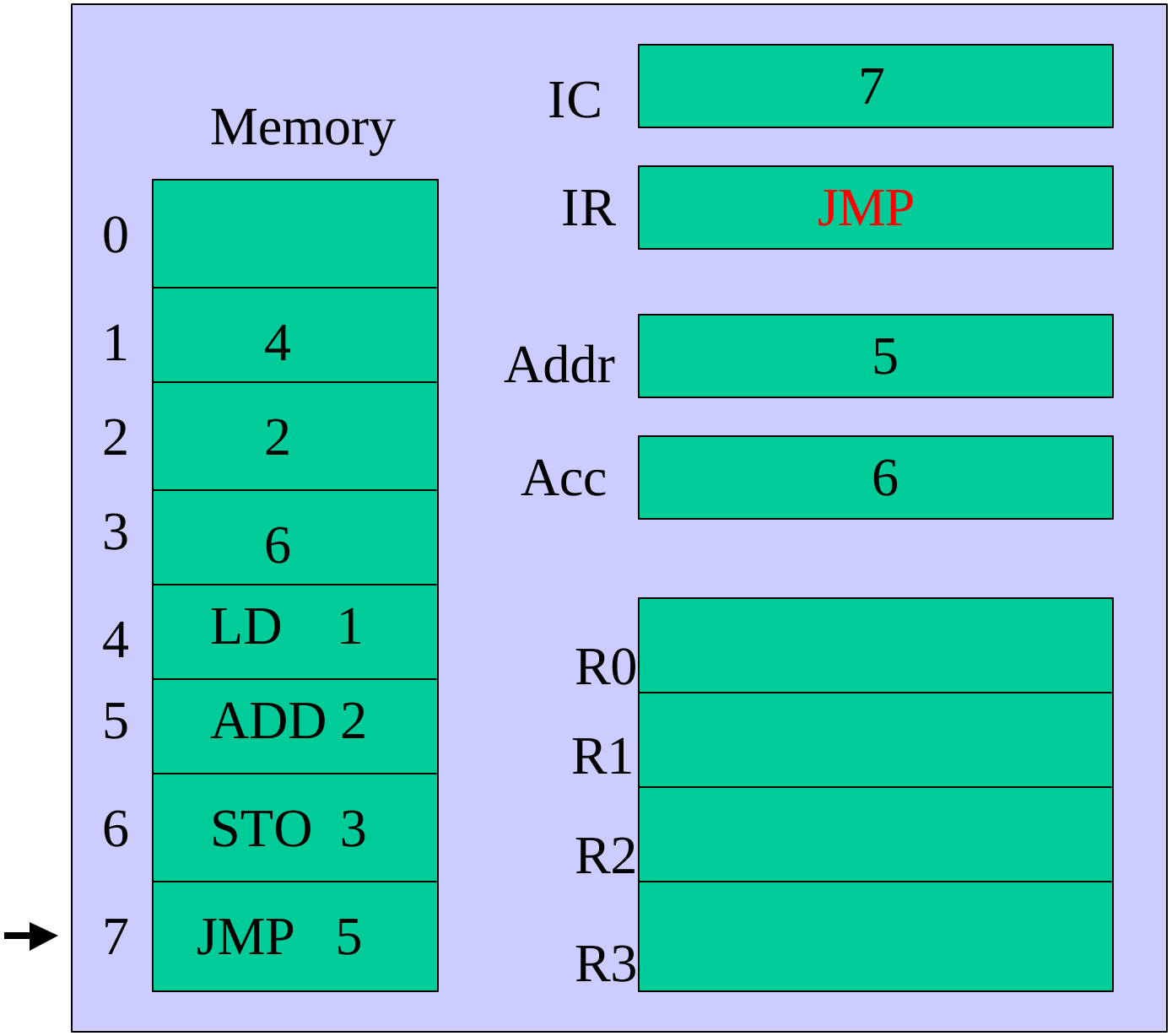

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
JMP 5
```

Suppose location 7 had the instruction JMP 5 (Jump to 5).




```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
JMP 5
```

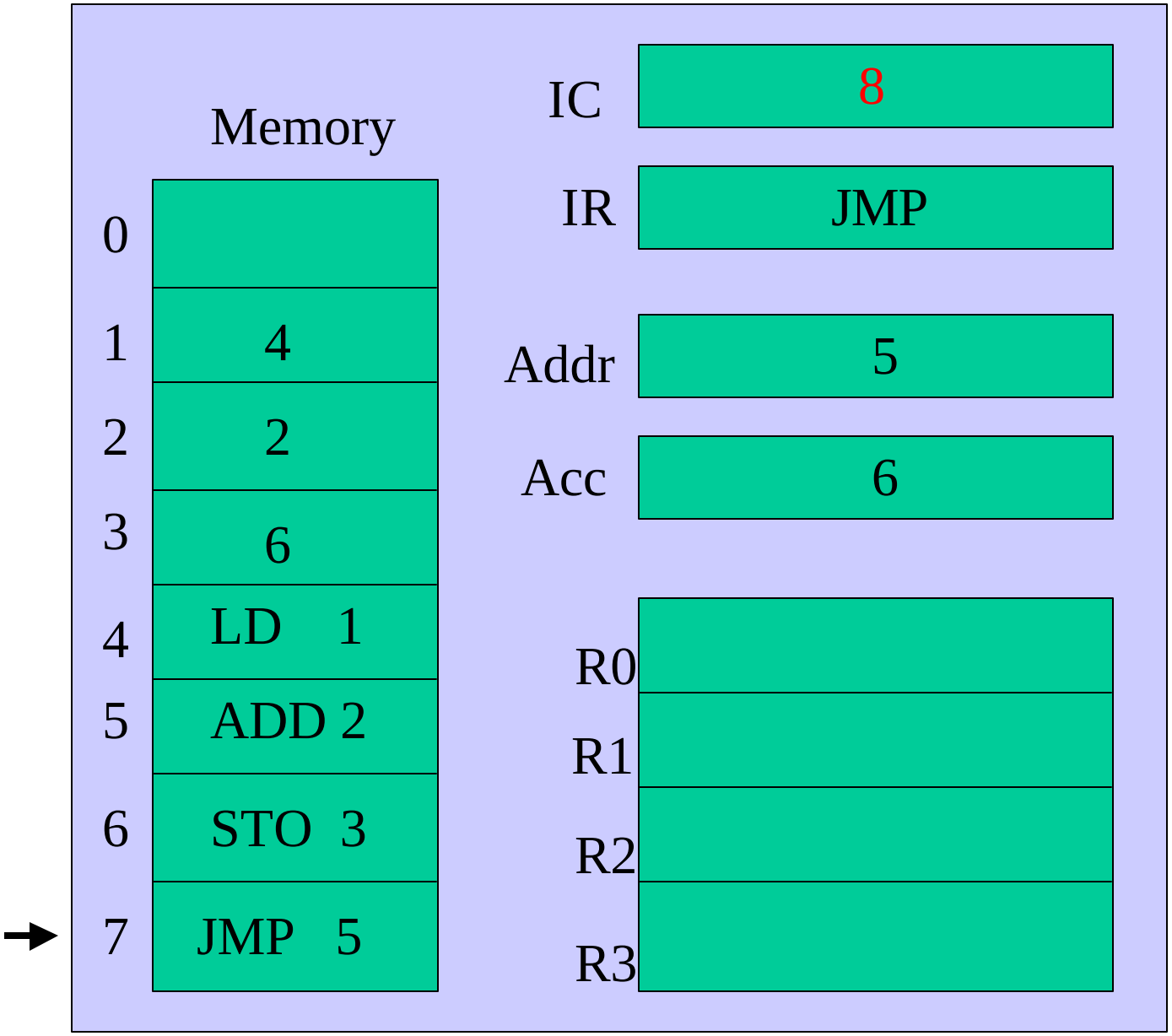
Fetch the contents of memory location 7.




```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
JMP 5
```

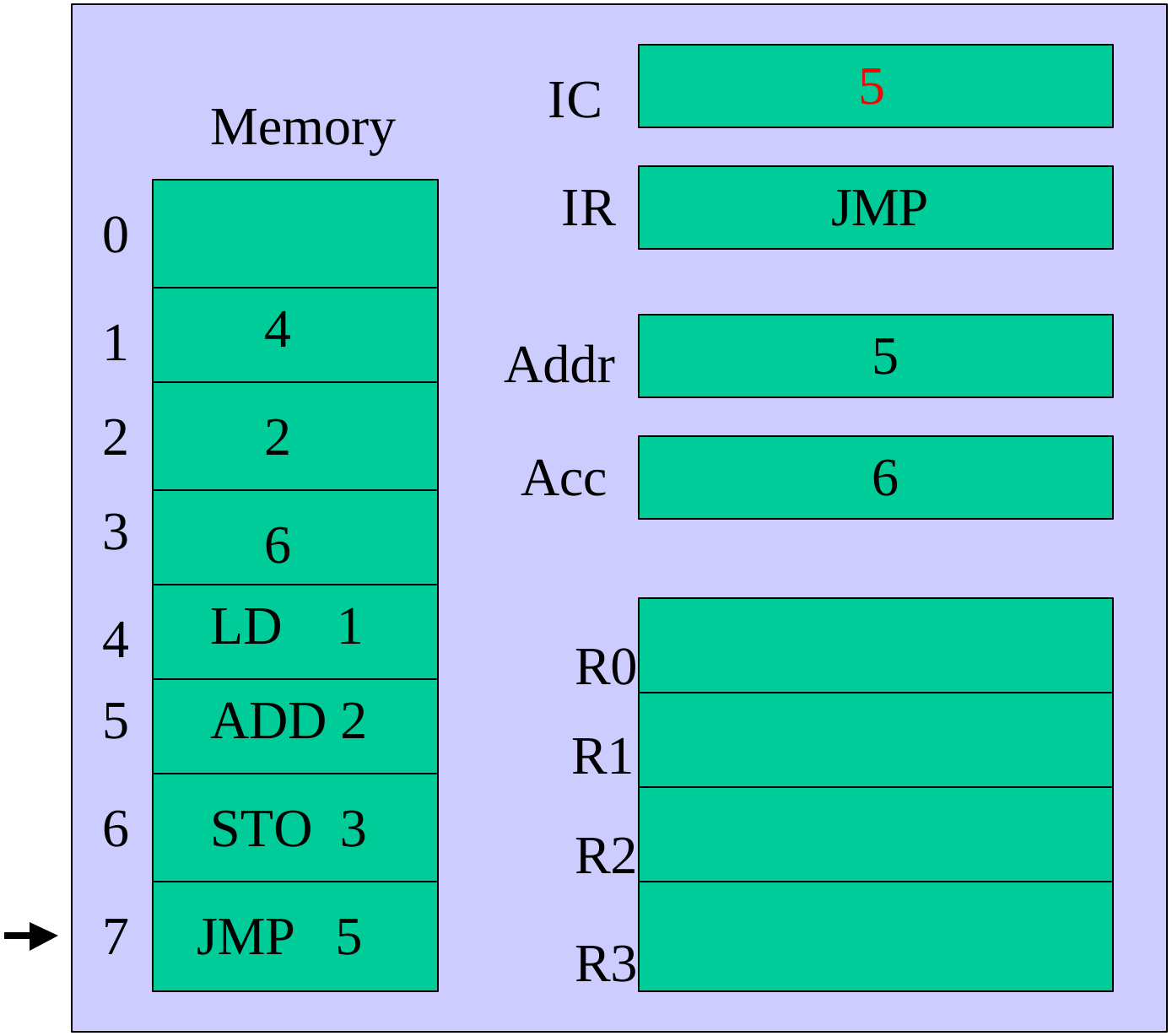
( In real computers, the JMP argument does not go into ADDR. ADDR remains unchanged. It is done here for simplicity. )

Place operation code into the Instruction Register.



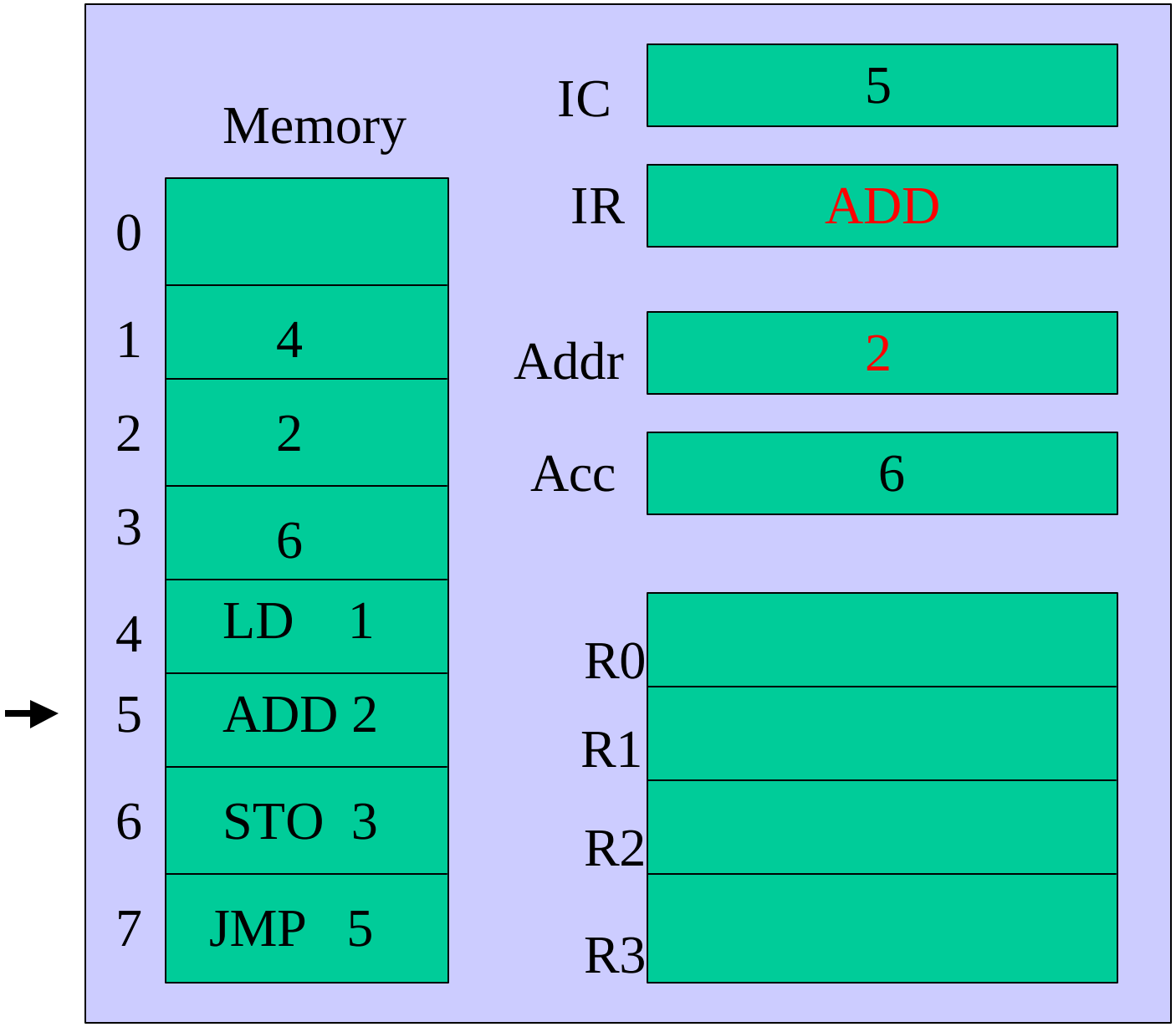

```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

Increment the Instruction Counter (IC)




```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
HLT
```

JMP: Place the argument field into the Instruction Counter.




```
ORG 1
ENTRY 4
EQU 4
EQU 2
RES 1
LD   1
ADD 2
STO  3
JMP 5
```

Fetch the next instruction from memory location 5.

# Transfer or Jump

- This short program shows an endless loop.
- A good program would have a loop termination condition.
- Ability to do conditional loops and other conditional transfers is the property that makes the computer powerful compared to a non-programmable calculator or tabulating machine.