Last Time

- Flip-flops
- Combining flip-flops with combinatorial logic
- Counter design

Today

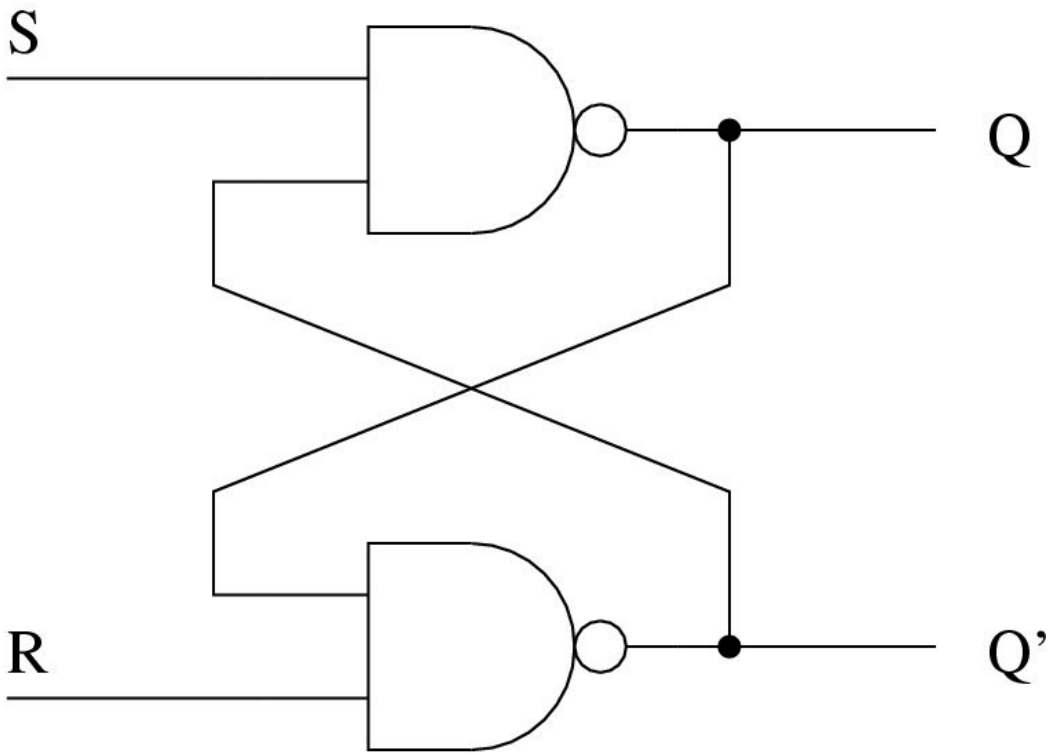
- A bit more on flip-flops
- Memory
- Microcontroller essentials

Administrivia

- Homework 3 due on Tuesday
- Lab 1 due in one week
  - Demonstration
  - Project report (as a group)
  - Personal reports
- Reading for today and Tuesday:
  - Parts from Chapter 4 (see the schedule)
  - J-K Flip-flop web page: “play” with the circuit

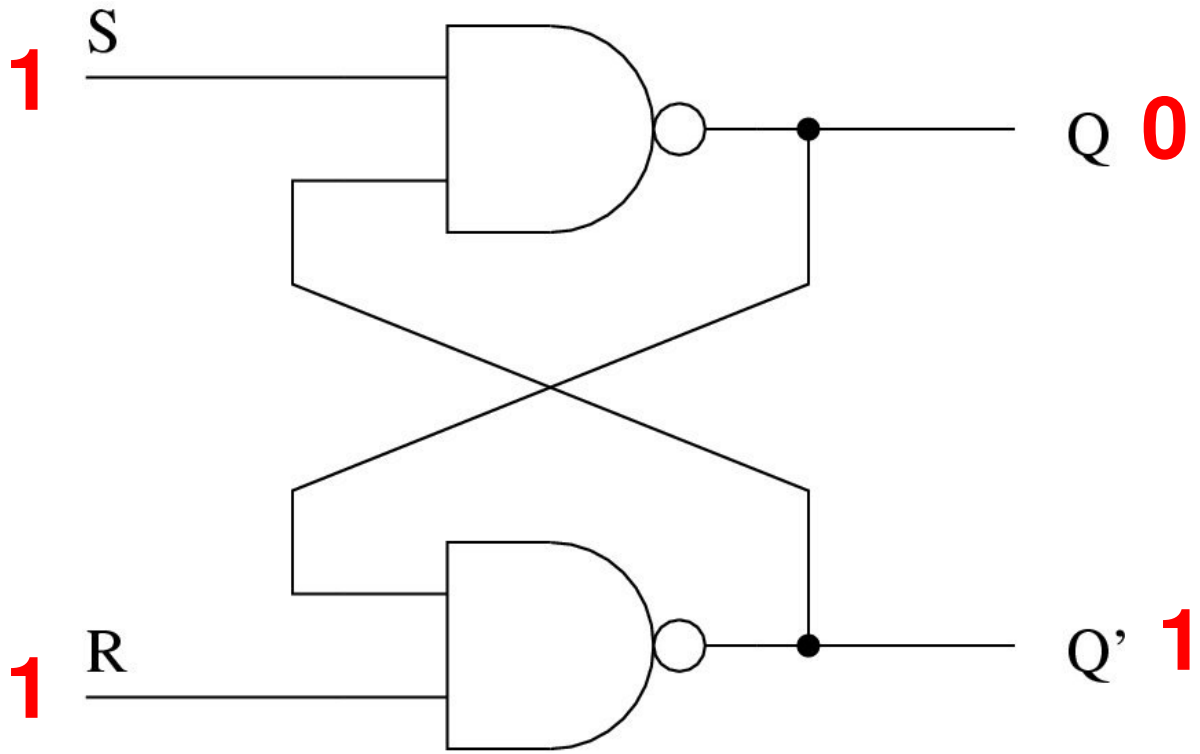
NAND Latch

What does this circuit do?



NAND Latch

Consider this initial state

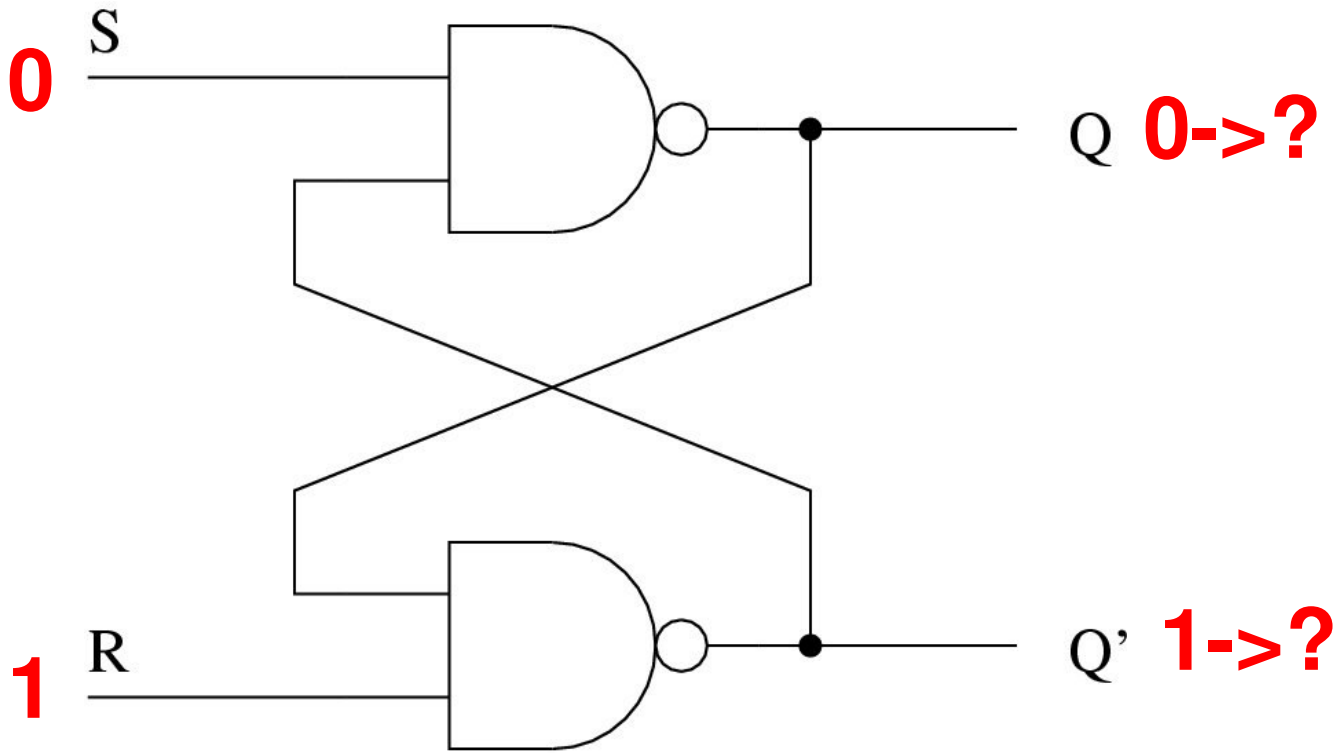


Is this a stable state?

Yes!

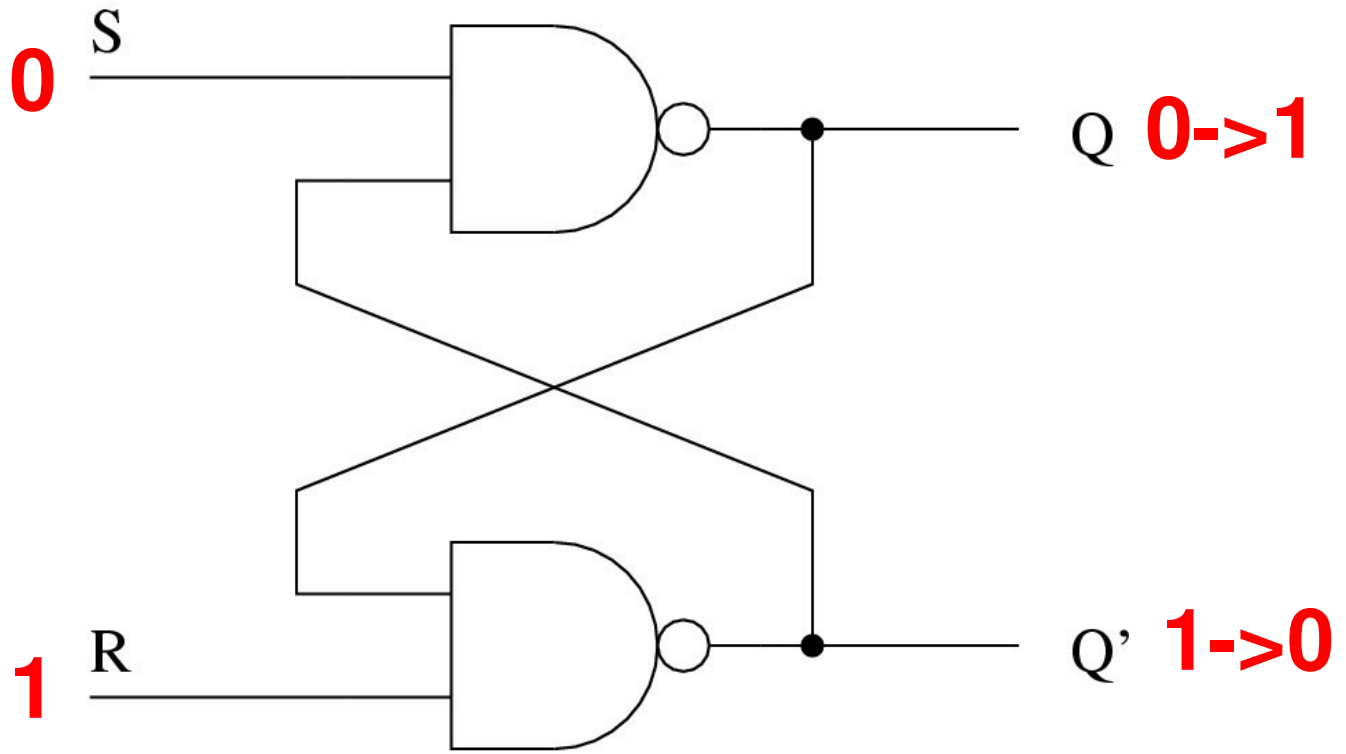
NAND Latch

What happens with S is set to 0?



NAND Latch

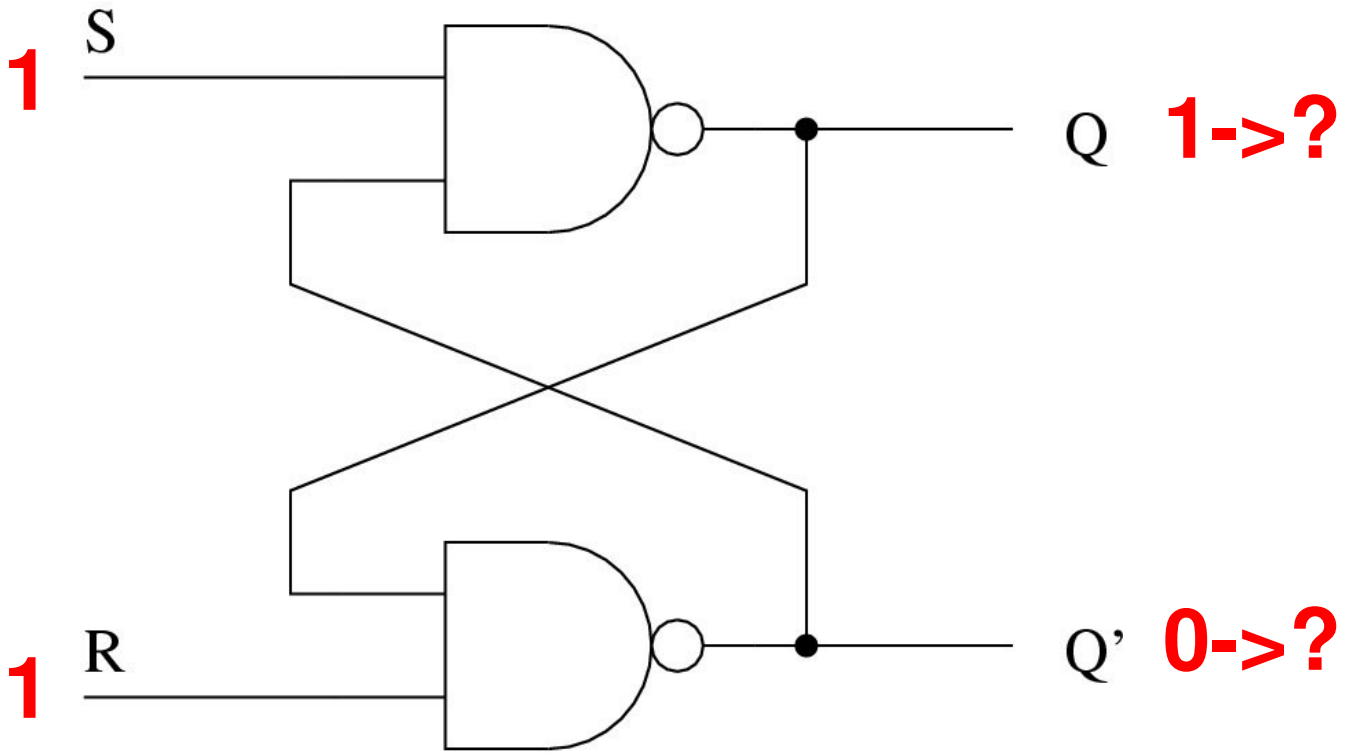
What happens with S is set to 0?



Q becomes 1 (thus S ‘sets’ Q)

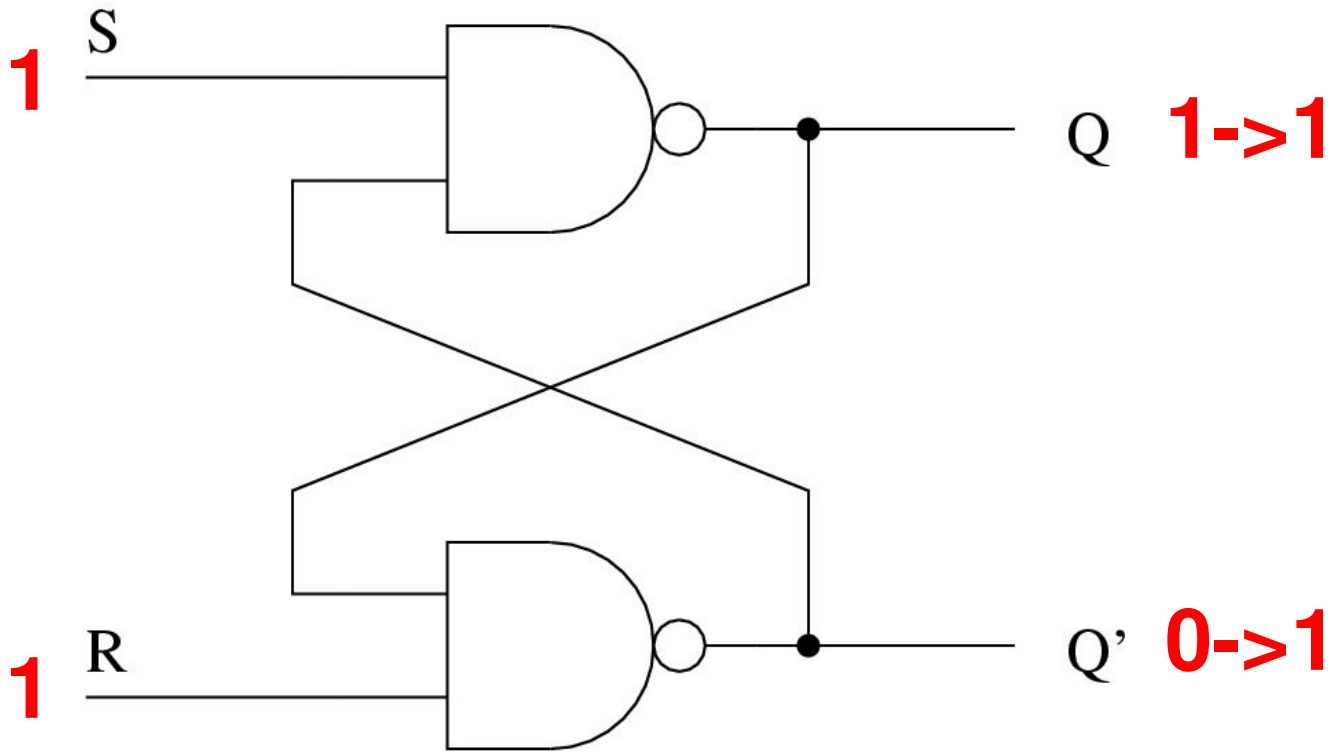
NAND Latch

Now S is set 1 – what happens?



NAND Latch

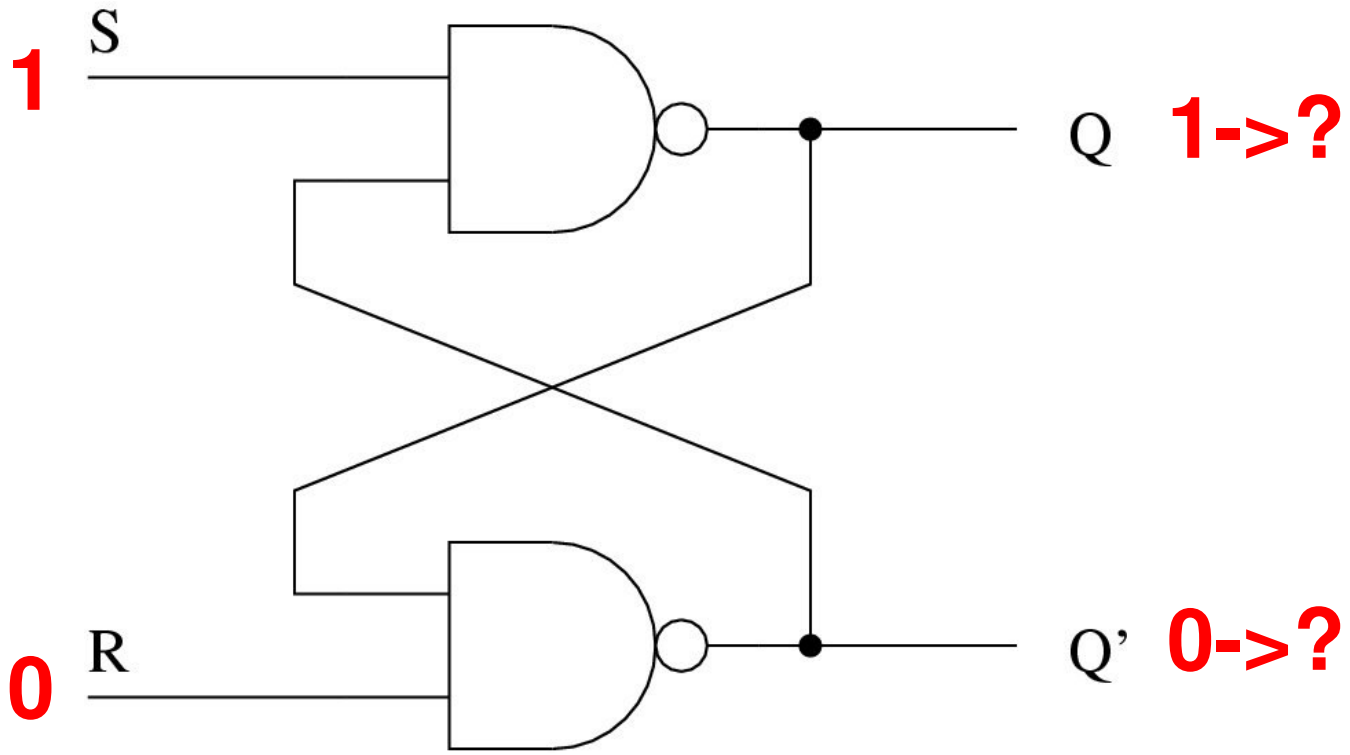
Q and Q’ remain the same!



So Q and Q’ retain a memory of old state!

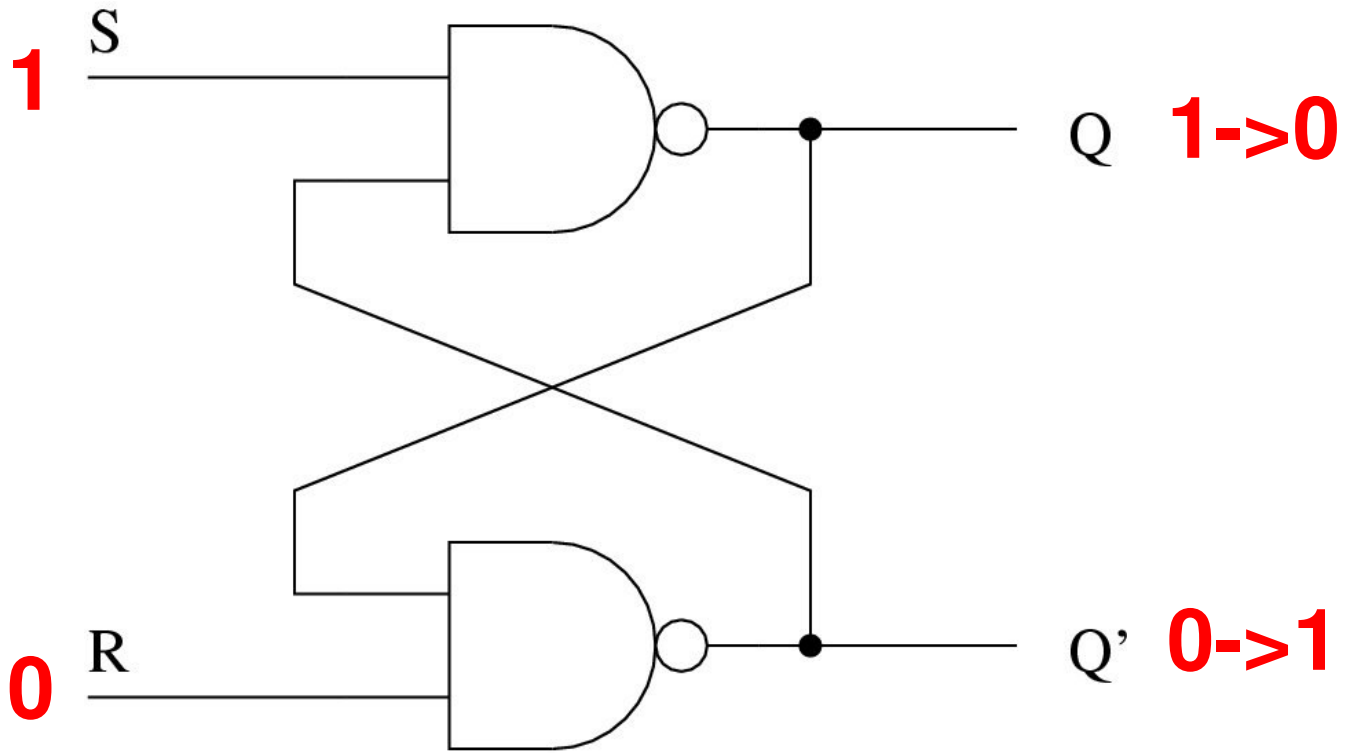
NAND Latch

Now set R to 0 – what happens?



NAND Latch

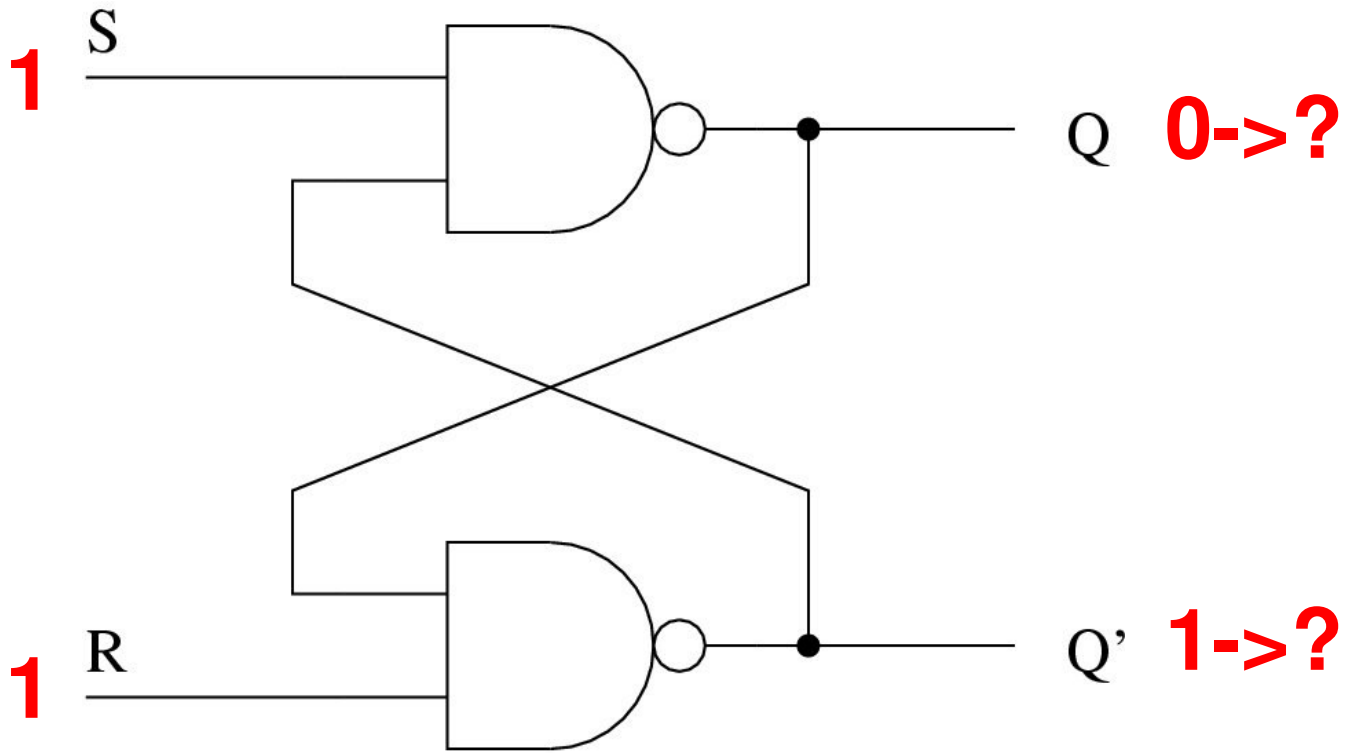
Now set R to 0 – what happens?



The state flips back (Q is ‘reset’)

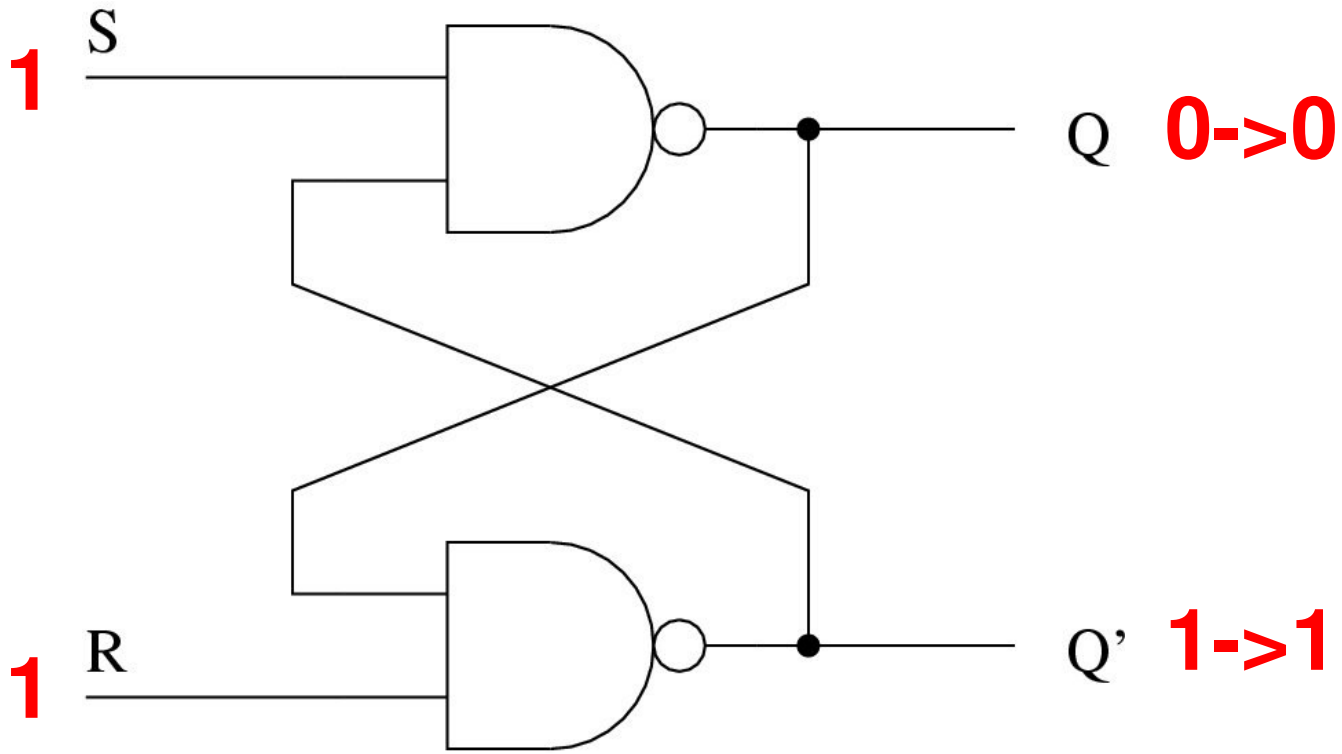
NAND Latch

Finally: set R to 1 – what happens?



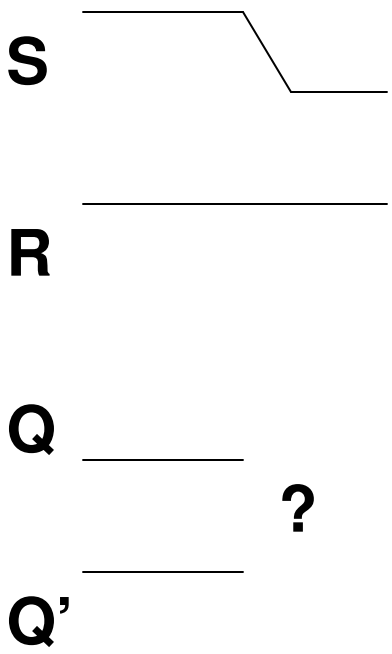
NAND Latch

Finally: set R to 1 – what happens?

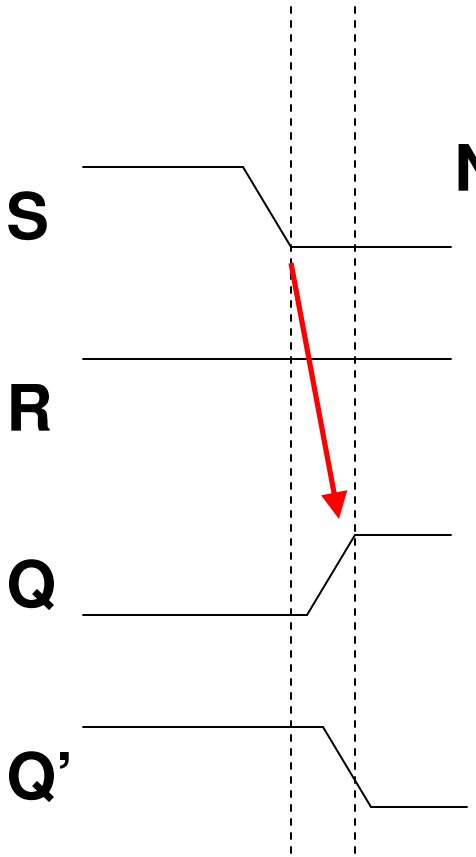


Q and Q' do not change state

Timing Diagram Representation

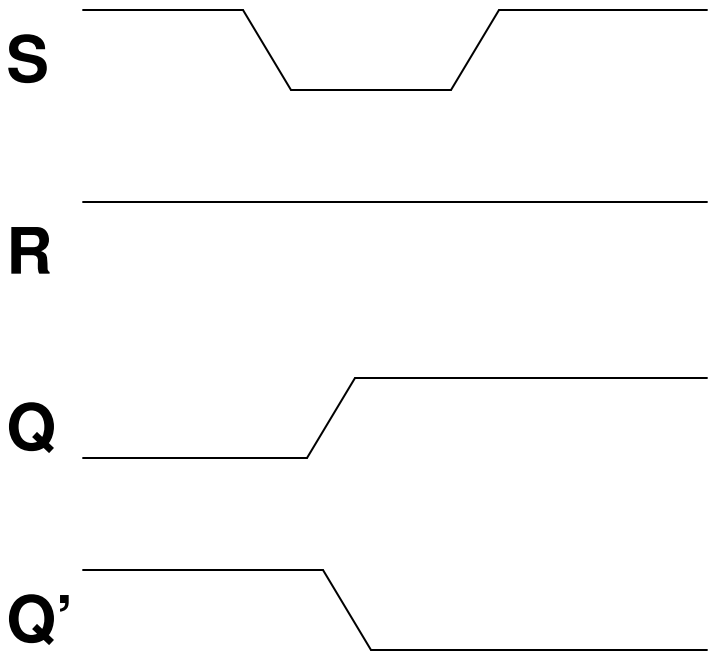


Timing Diagram Representation



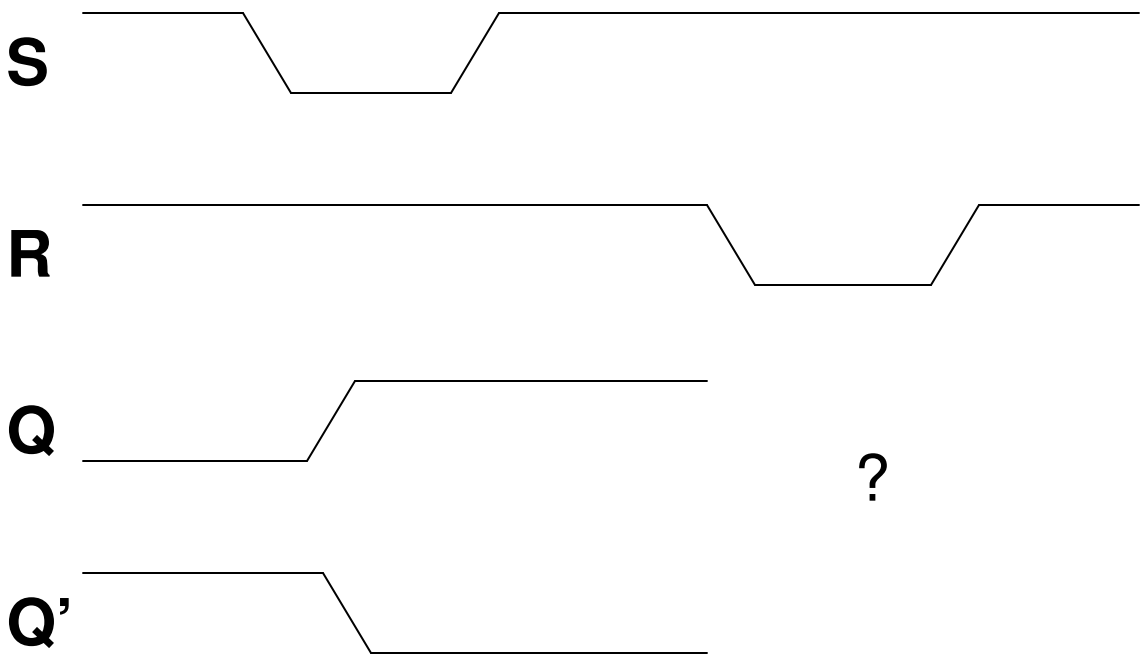
Note small delay in response in Q and Q'

Timing Diagram Representation

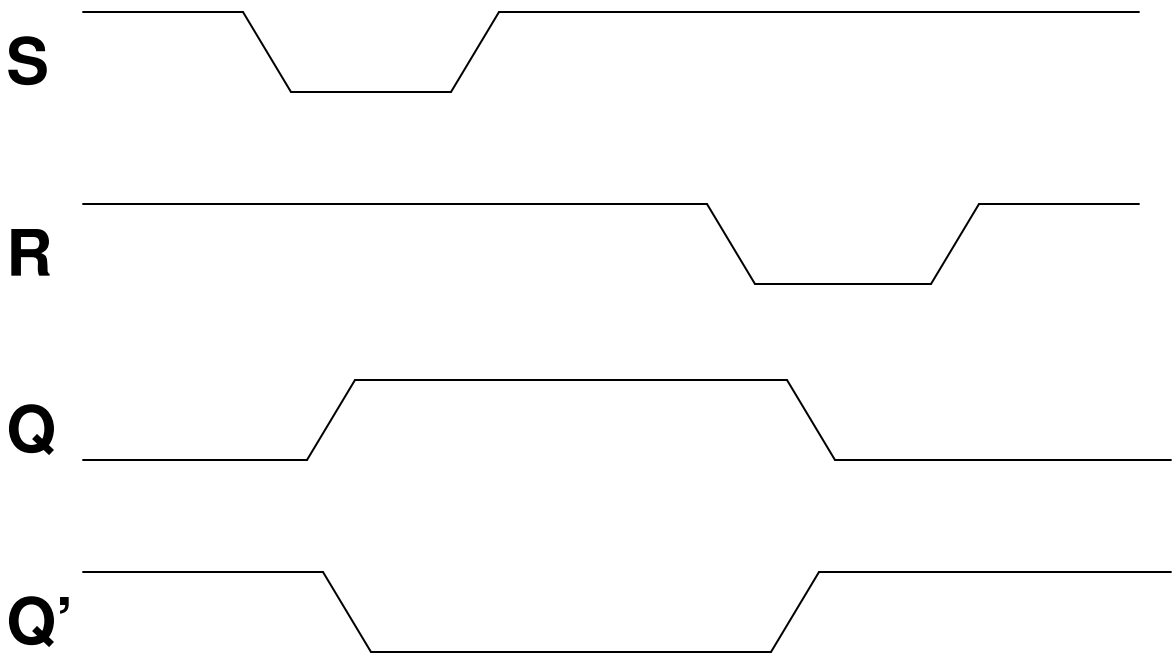


When S returns to high – both Q and Q' remain in the same state

Timing Diagram Representation

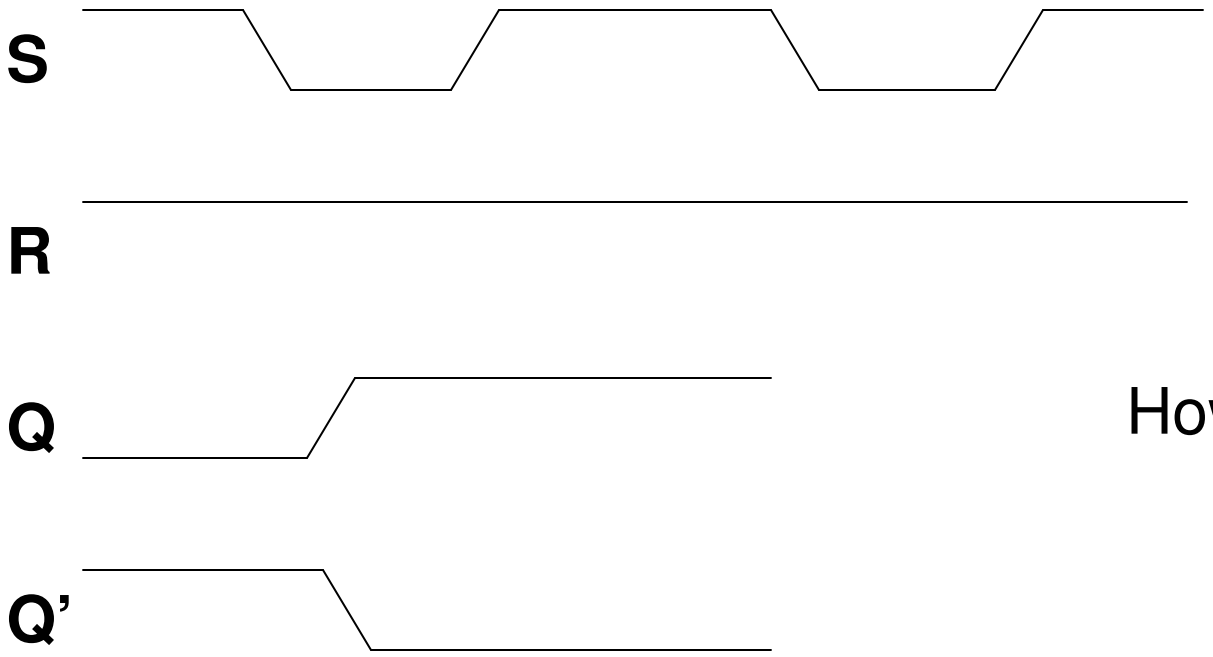


Timing Diagram Representation



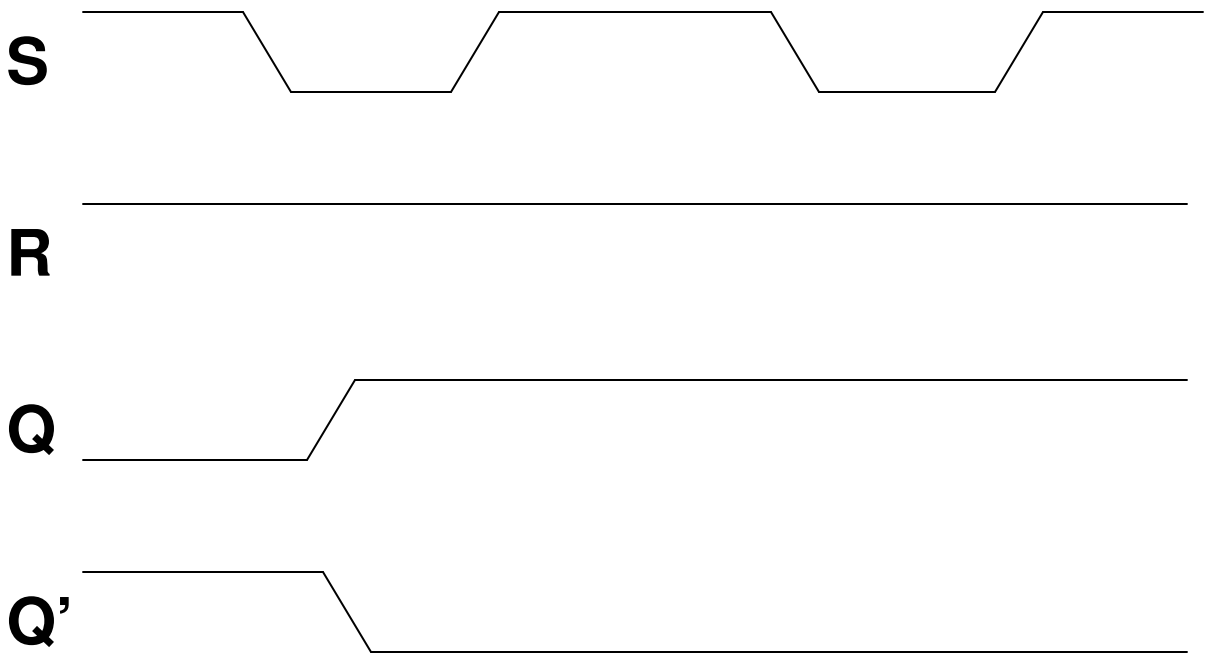
Q and Q’ flip state

Timing Diagram Representation



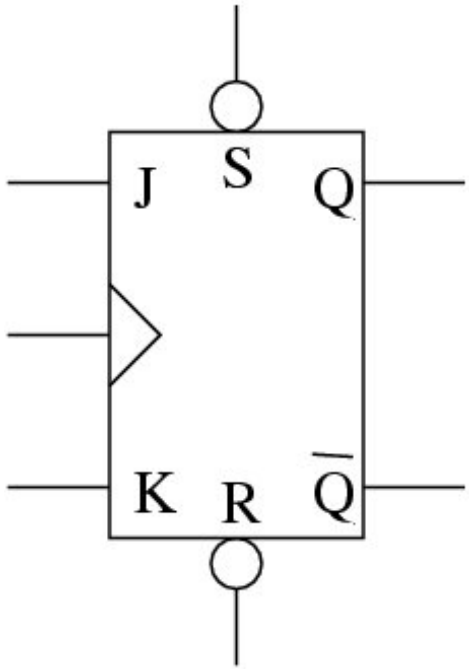
How about this case?

Timing Diagram Representation



No change in Q and Q’

J-K Flip-Flops

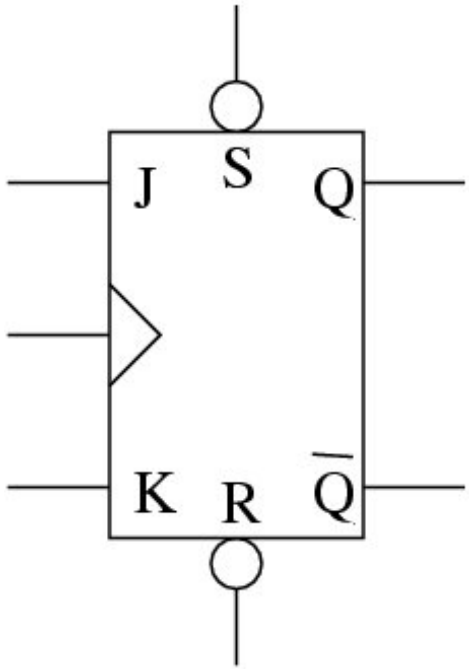


J-K Flip-Flops

No matter what the clock state is:

- S (set): when low, forces Q to 1
- R (reset): when low, forces Q to 0

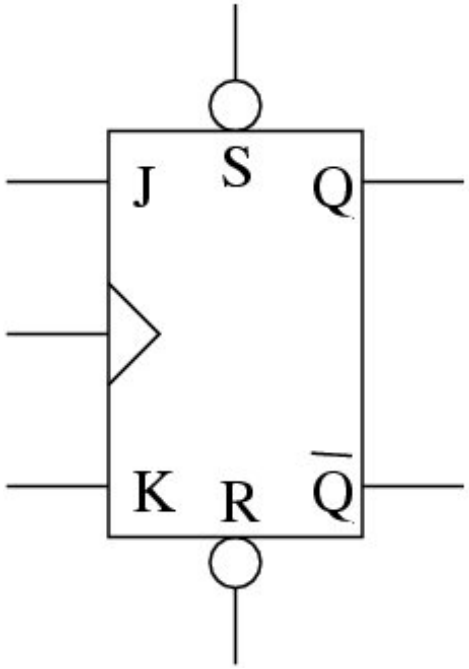
In general, these are both high



J-K Flip-Flops

When clock transitions from high to low:

J	K		Q
0	0		No change
0	1		0
1	0		1
1	1		Flip state



Collections of Bits

- 8 bits: a “byte”
- 4 bits: a “nybble”

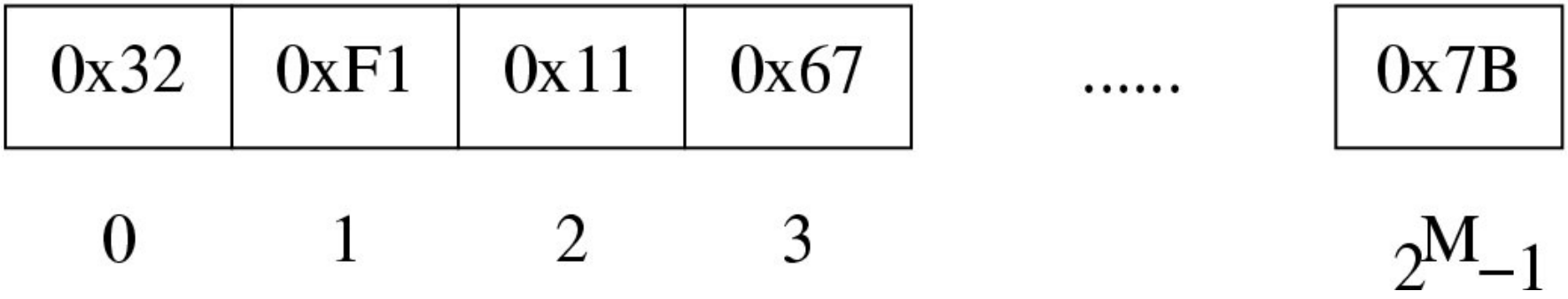
- “words”: can be 8, 16, or 32 bits (depending on the processor)

Memory

What are the essential components of a memory?

A Memory Abstraction

- We think of memory as an array of elements – each with its own address
- Each element contains a value
- It is most common for the values to by 8-bits wide (so a byte)



Memory Operations

Read

```
foo(A+5);
```

reads the value from the memory location referenced by ‘A’ and adds the value to 5. The result is handed to a function called `foo();`

Memory Operations

Write

```
A = 5;
```

writes the value 5 into the memory location referenced by ‘A’

Types of Memory

Random Access Memory (RAM)

- Computer can change state of this memory at any time
- Once power is lost, we lose the contents of the memory

- This will be our data storage on our microcontrollers

Types of Memory

- Read Only Memory (ROM)
  - Computer **cannot** arbitrarily change state of this memory
  - When power is lost, the contents are maintained

Types of Memory

Erasable/Programmable ROM (EPROM)

- State can be changed under very specific conditions (usually not when connected to a comptuer)

- Our microcontrollers have an Electrically Erasable/Programmable ROM (EEPROM) for program storage

Last Time

- R-S Latch (the heart of our flip flops)
- J-K flip flops
- Memory: abstraction and types

Today

- Memory (RAM) Behavior
- Essential microprocessor components

Administrivia

- Homework 3 due @5:00
- Homework 4 out tonight (due in 1 week)
- Project 1 due Thursday @5:00
  - Demonstration
  - Group report
  - Personal reports

Project Grading

- Personal report: assign percent effort to each group member (including yourself)
- The group receives a grade of *G*
- Person *i* gives person *k* an effort score of $g_{i,k}$ (percent)
- Person *k* receives the following grade:

$$G * \sum_{i} g_{i,k}$$

Example: A Read/Write Memory Module

Inputs:

- 2 Address bits: A0 and A1
- 1 “chip select” (CS) bit
- 1 read/write bit (1 = read; 0 = write)
- 1 clock signal (CLK)

Input or Output:

- Data bit (connected to the “data bus”)

Implementing A Read/Write Memory Module

With 2 address bits, how many memory elements can we address?

How could we implement each memory element?

Implementing A Read/Write Memory Module

With 2 address bits, how many memory elements can we address?

- 4 1-bit elements

How could we implement each memory element?

- With a D flip-flop
  - (more about this later)

Memory Module Specification

“chip select” signal:

- Allows us to have multiple devices (e.g., memory modules) that can write to the bus
- But: only one device will ever be selected at one time

Memory Module Specification

When chip select is low:

- No memory elements change state
- The memory does not drive the data bus

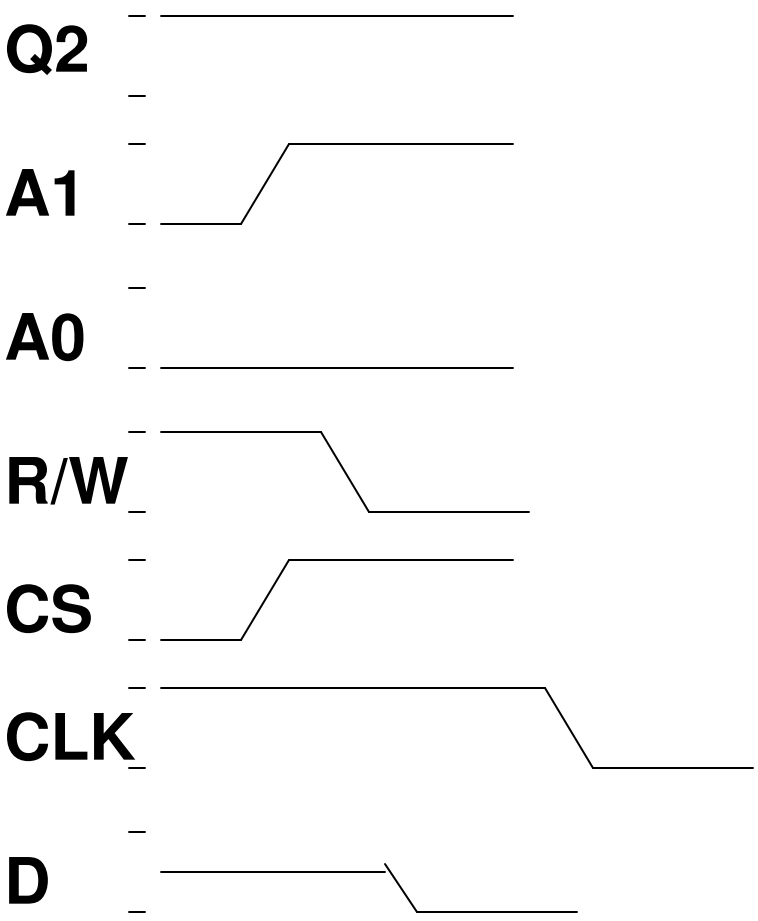
Memory Module Specification

When chip select is high:

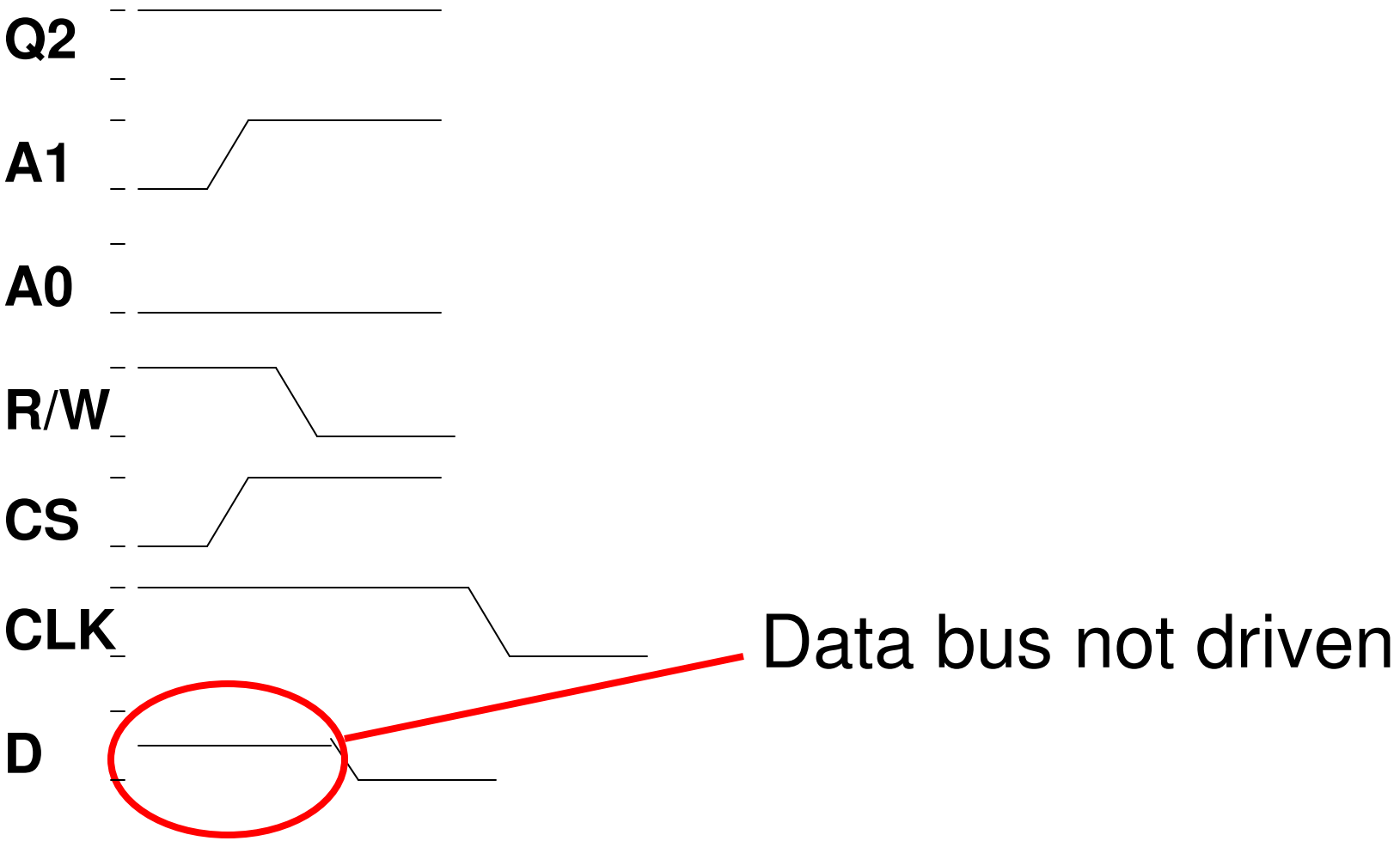
- If R/W is high:
  - Drive the data bus with the value that is stored in the element specified by A1, A0

- If R/W is low:
  - Store the value that is on the data bus in the element specified by A1, A0

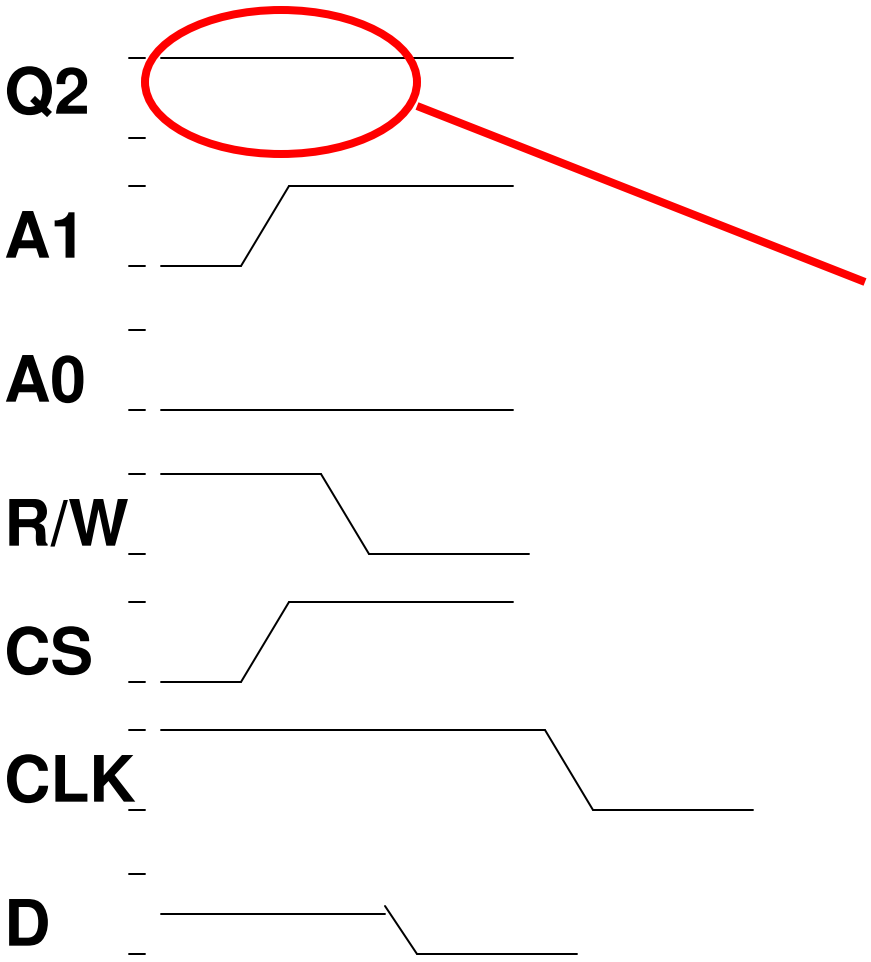
Memory Timing Diagram



Memory Timing Diagram

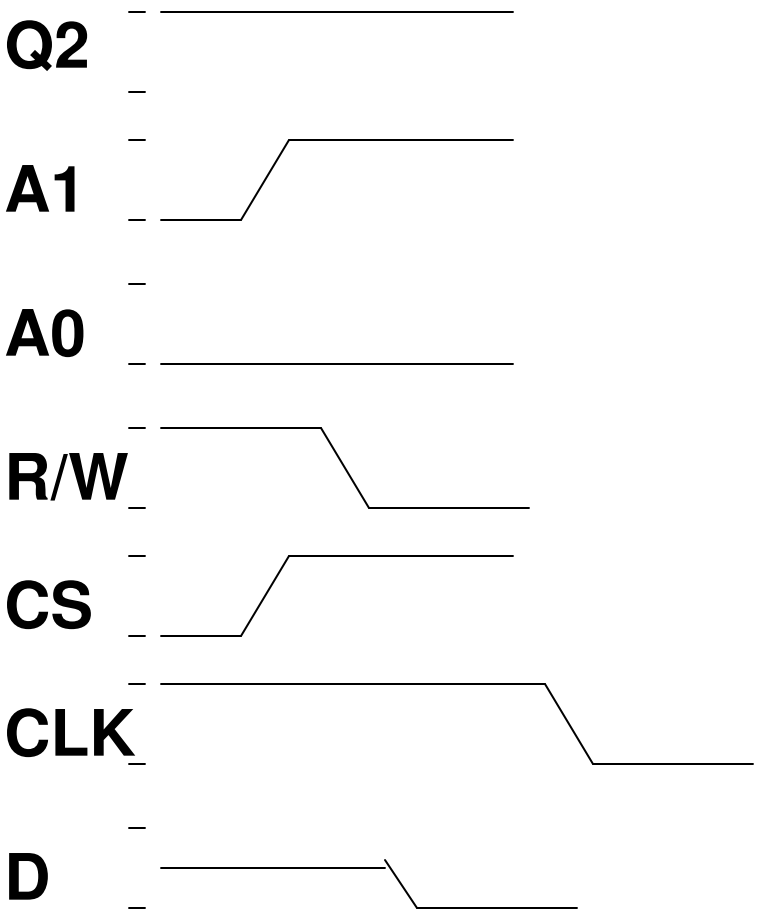


Memory Timing Diagram



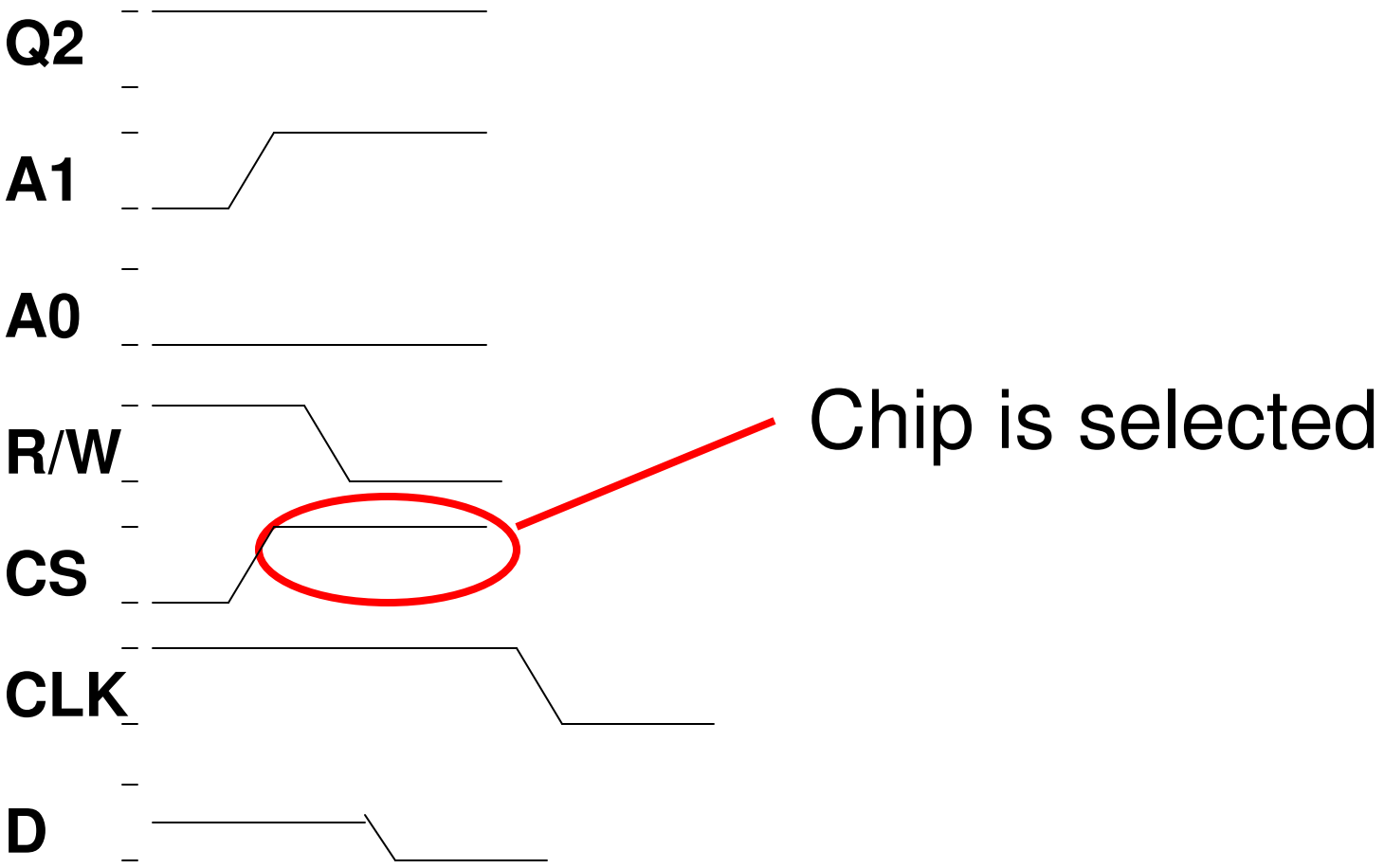
Memory element 2 is initially in a high state

Memory Timing Diagram

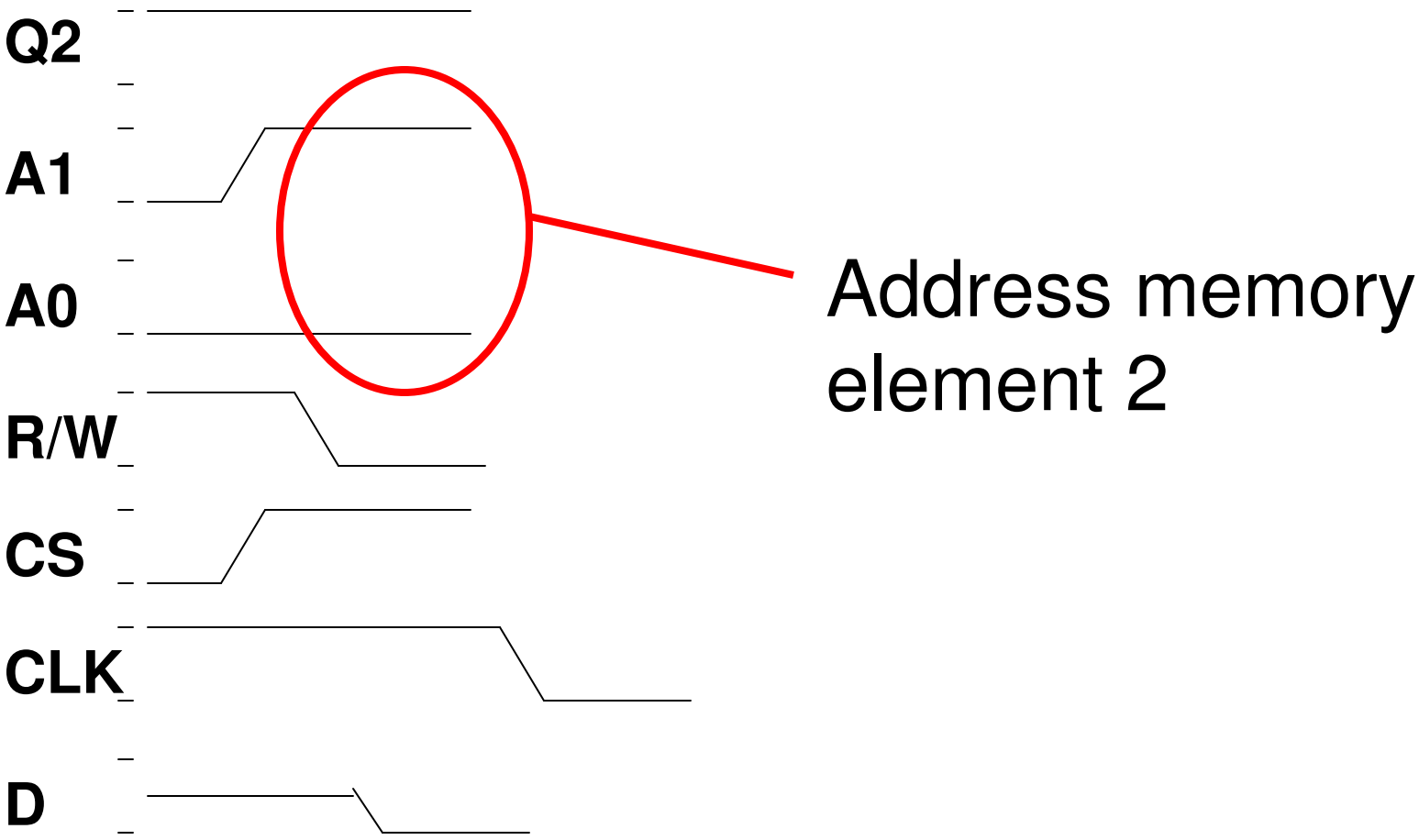


What happens next?

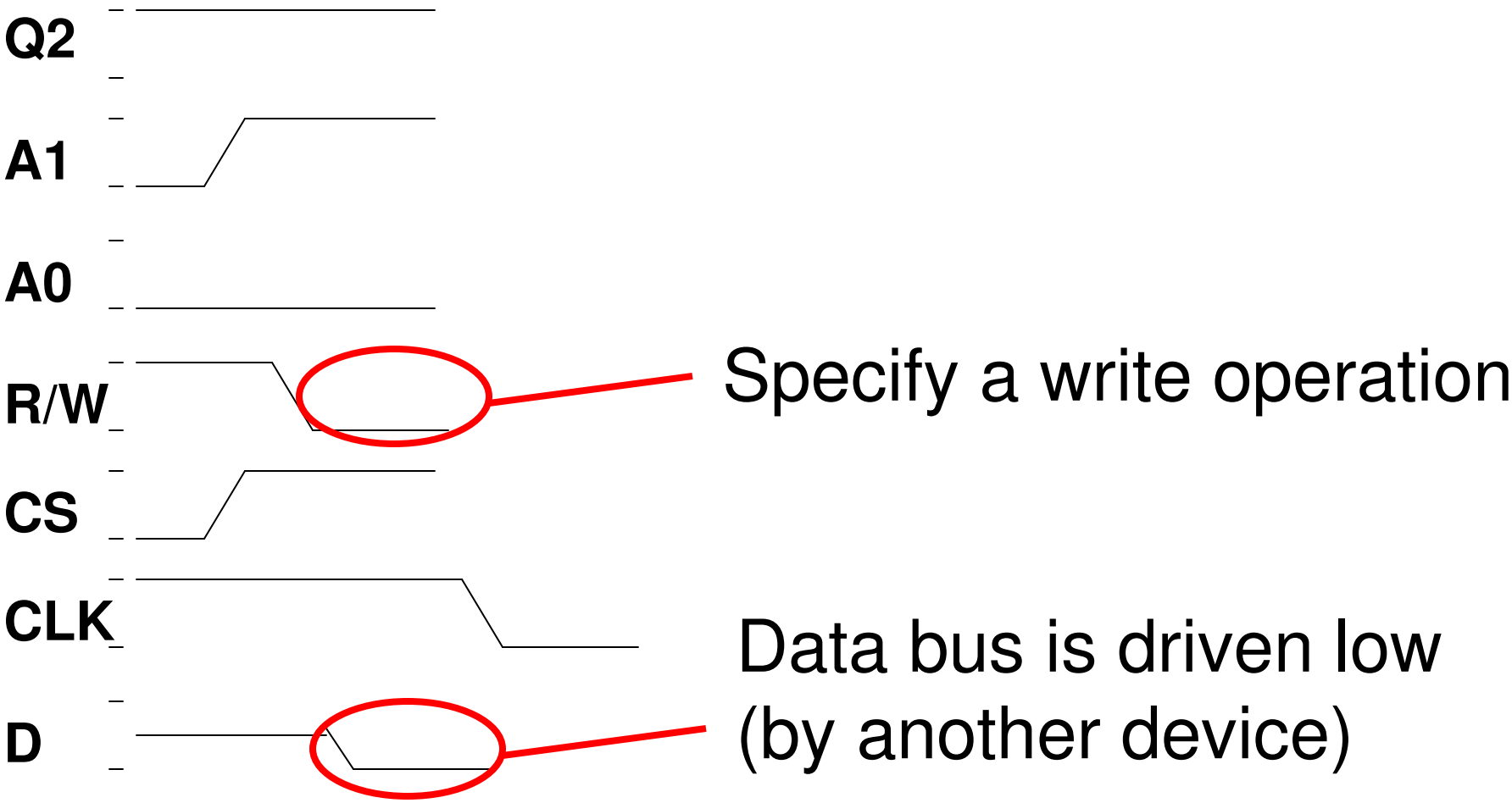
Memory Timing Diagram



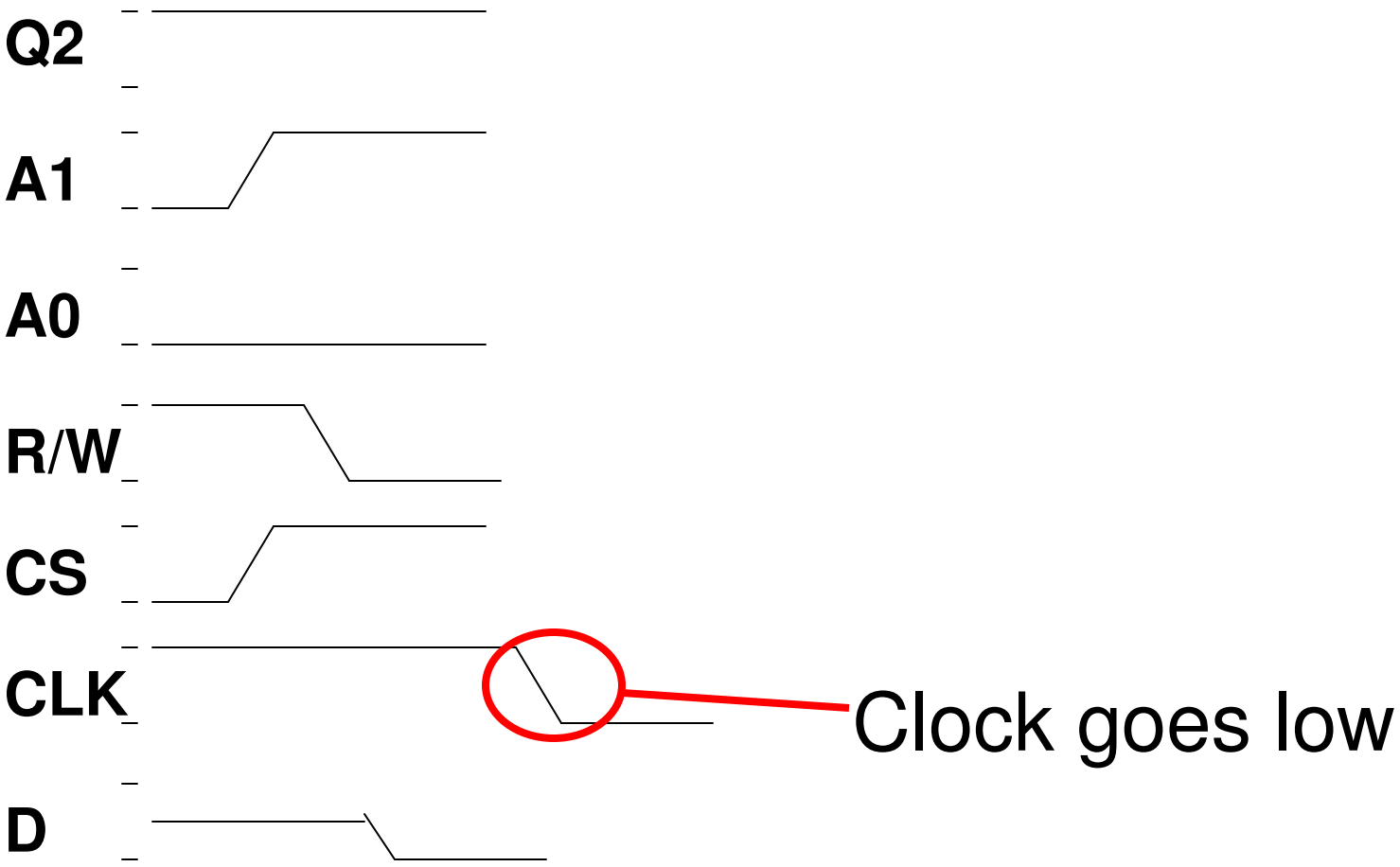
Memory Timing Diagram



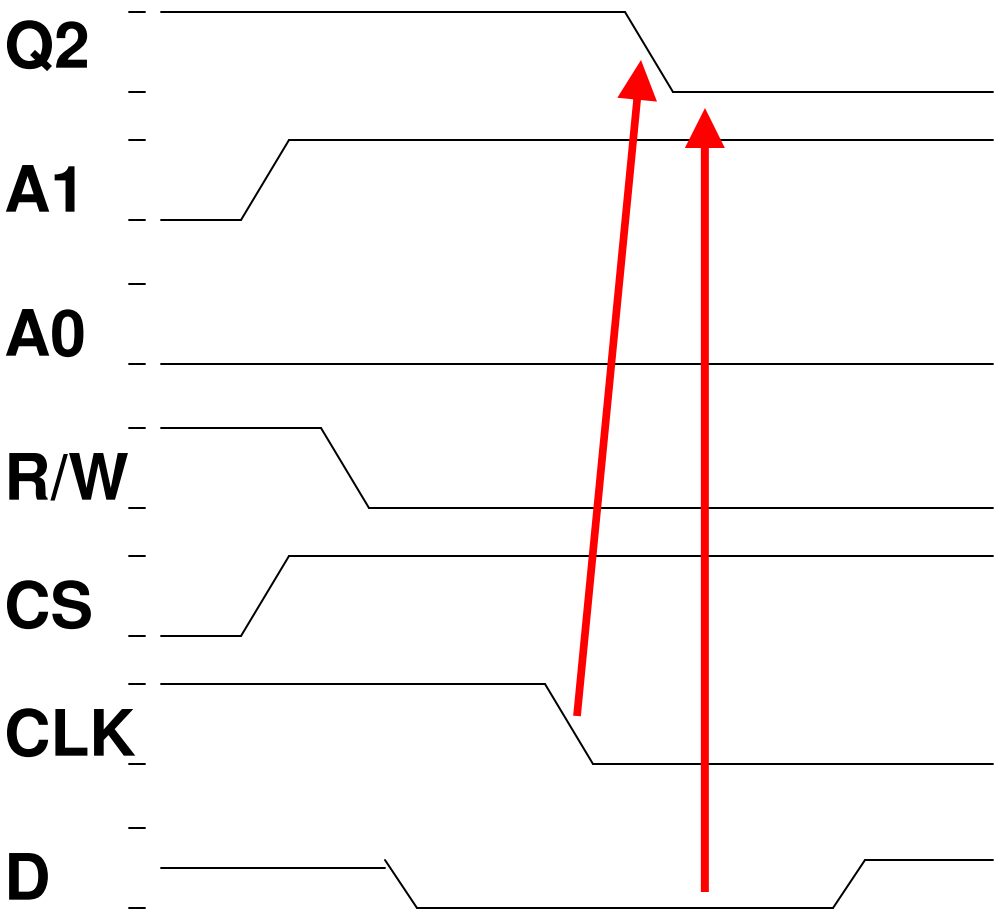
Memory Timing Diagram



Memory Timing Diagram

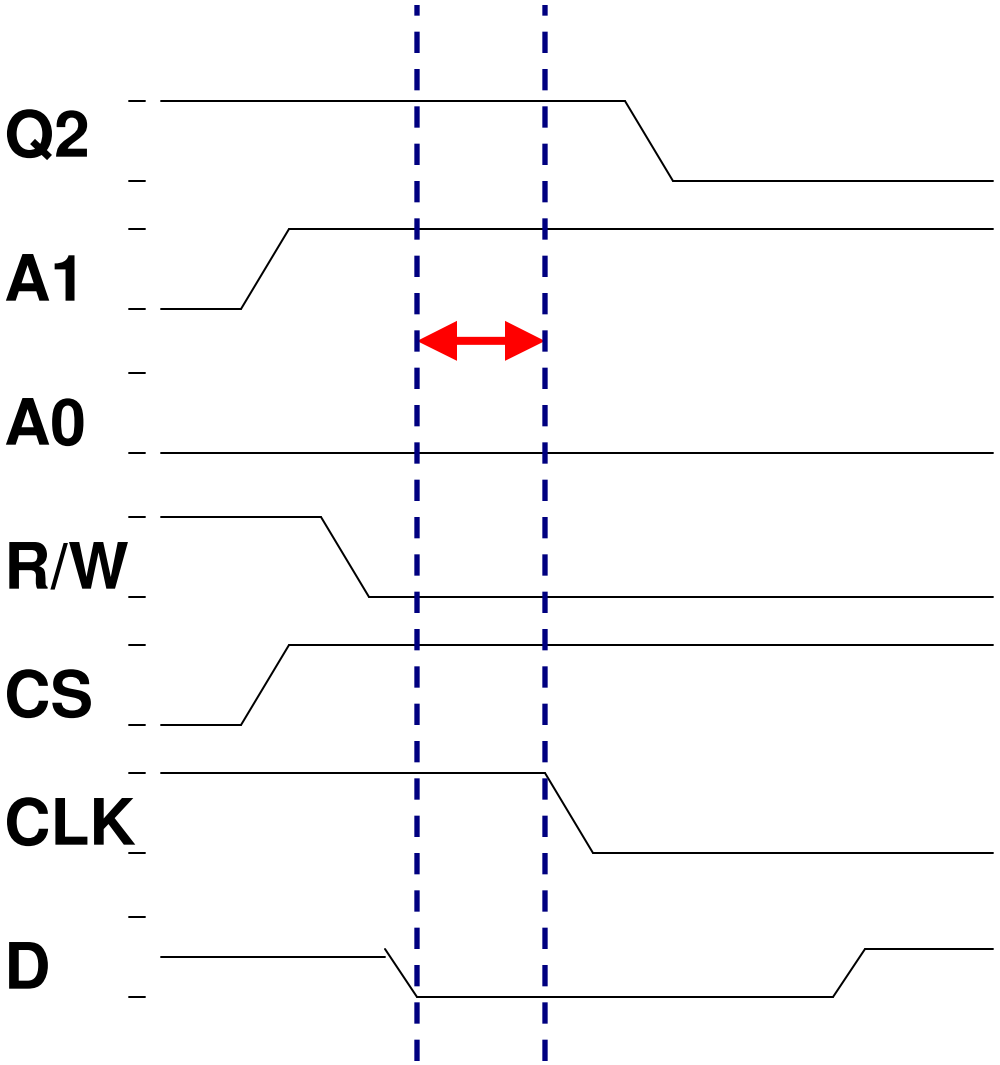


Memory Timing Diagram



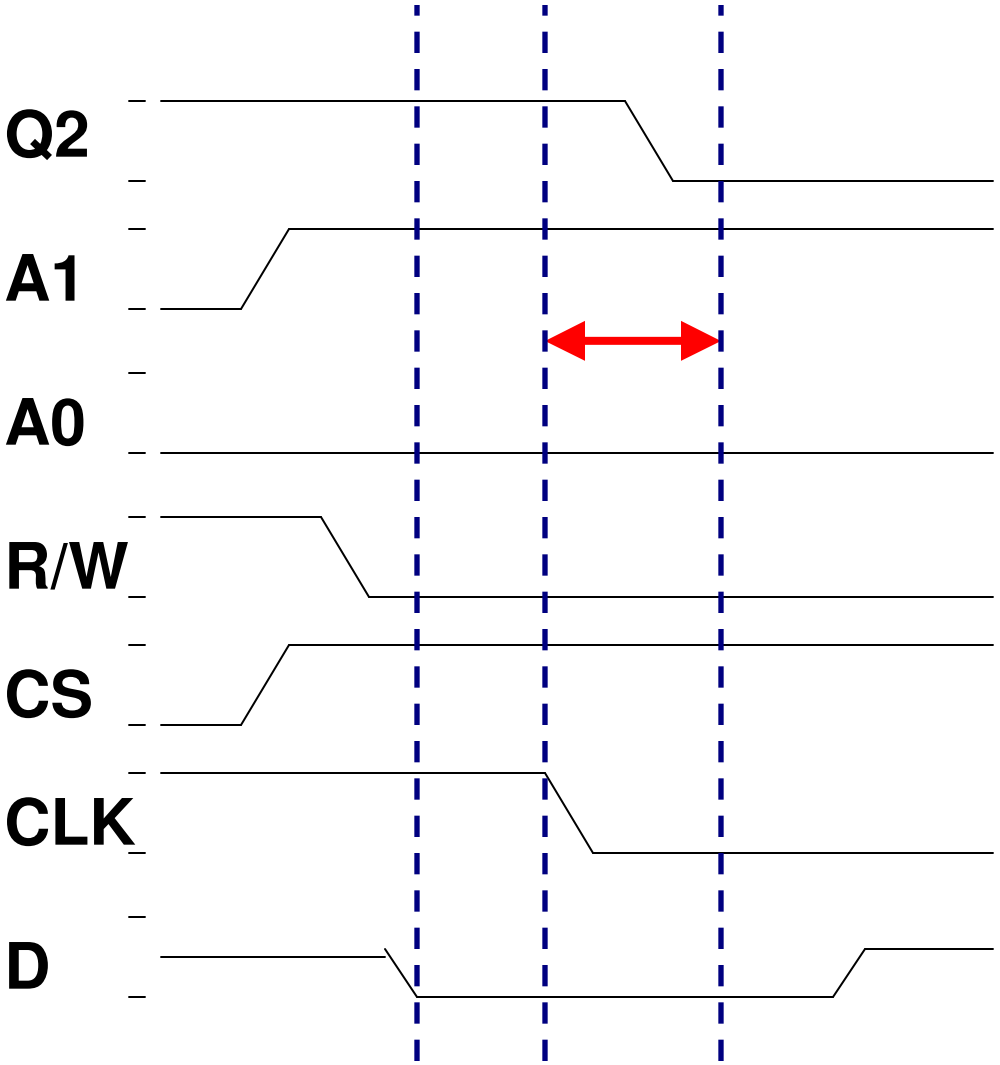
Memory element 2 changes state to low

Memory Timing Diagram



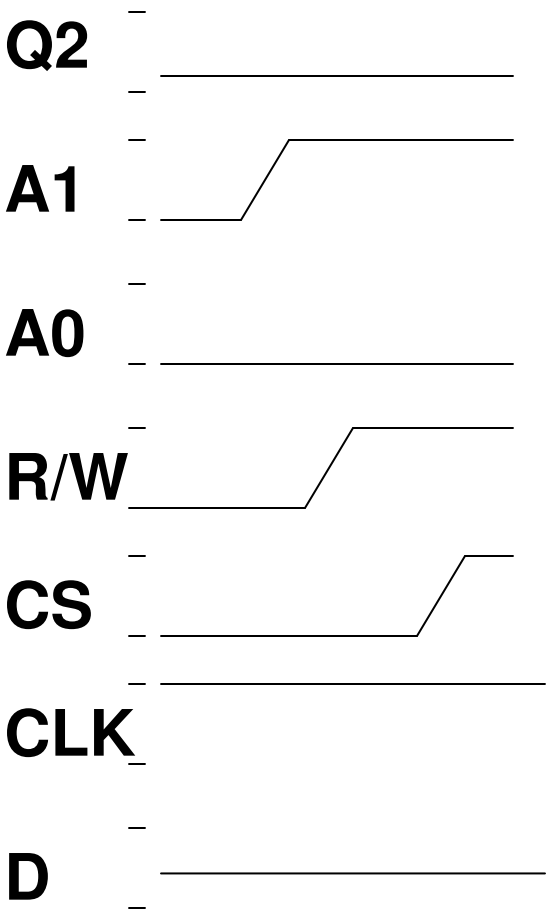
Setup time: all inputs must be valid during this time

Memory Timing Diagram

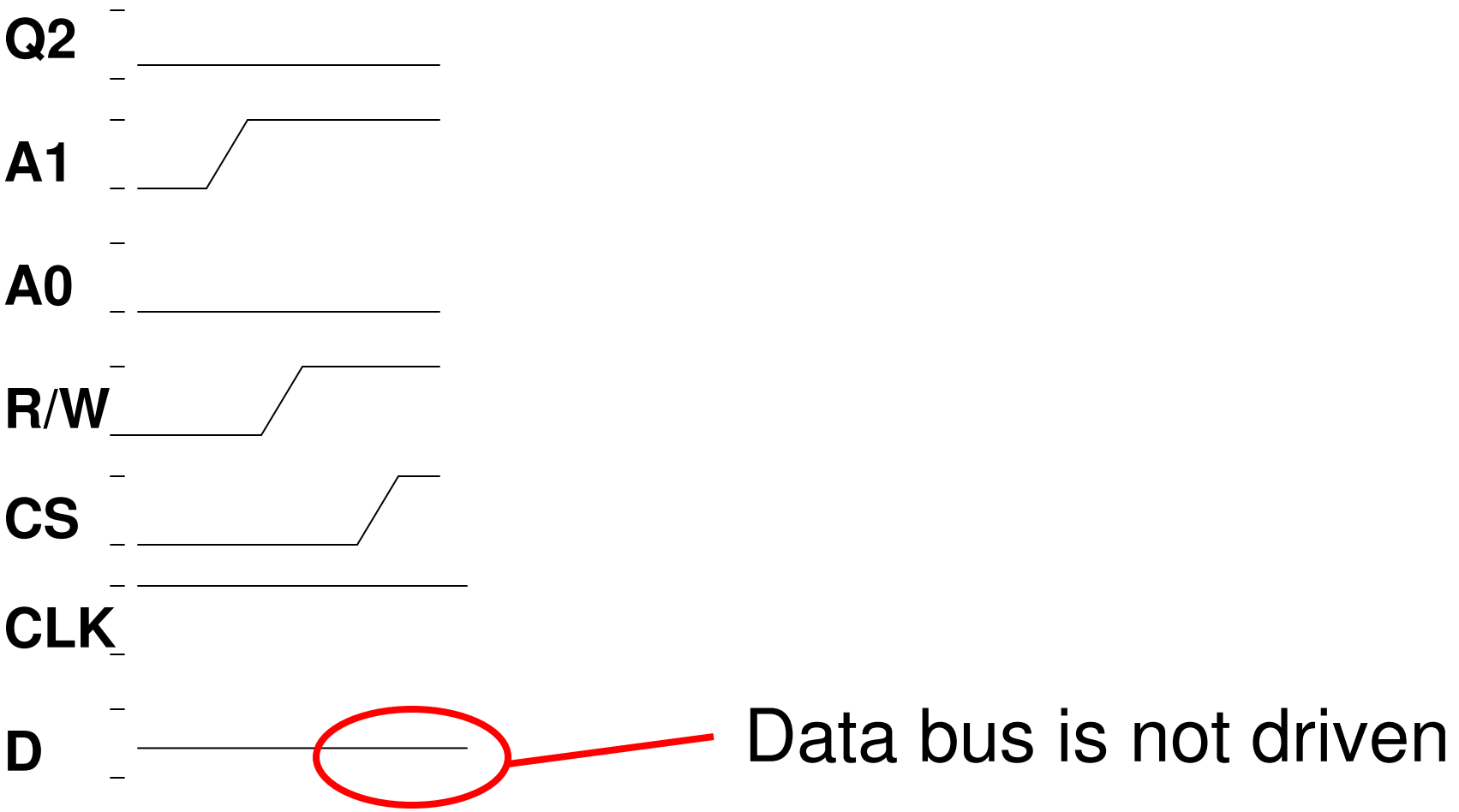


Hold time: all inputs must continue to be valid

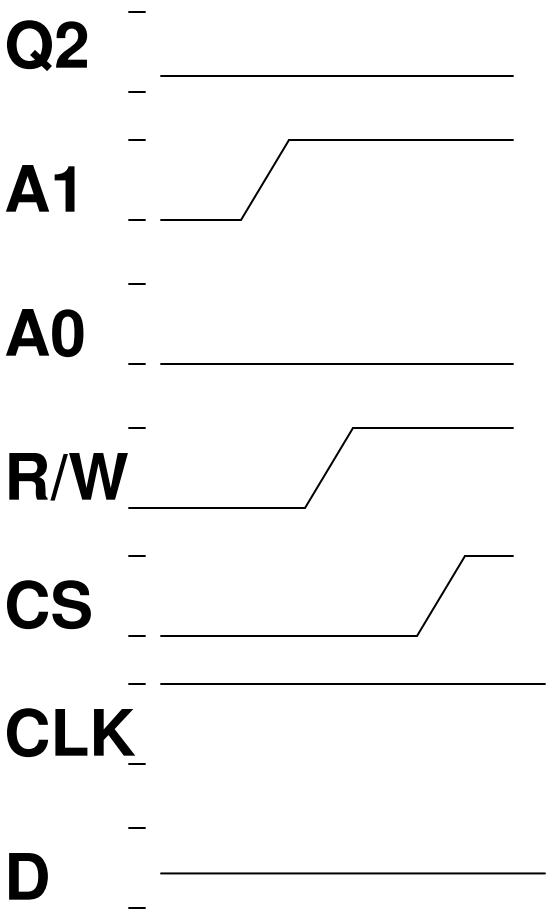
Memory Timing Diagram II



Memory Timing Diagram II

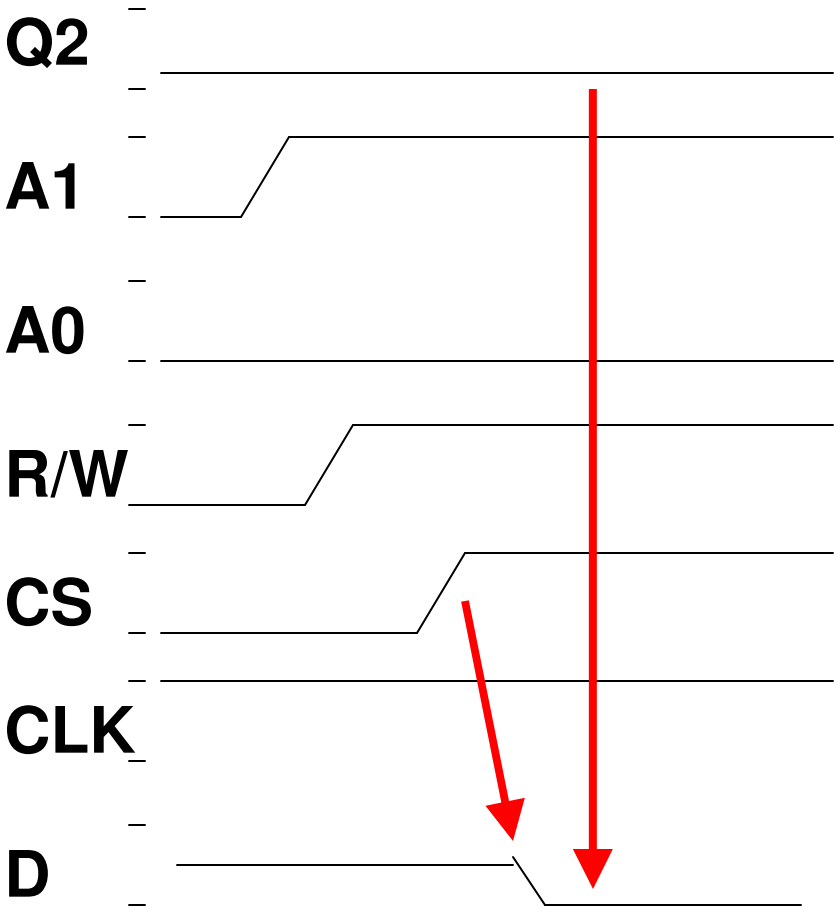


Memory Timing Diagram II



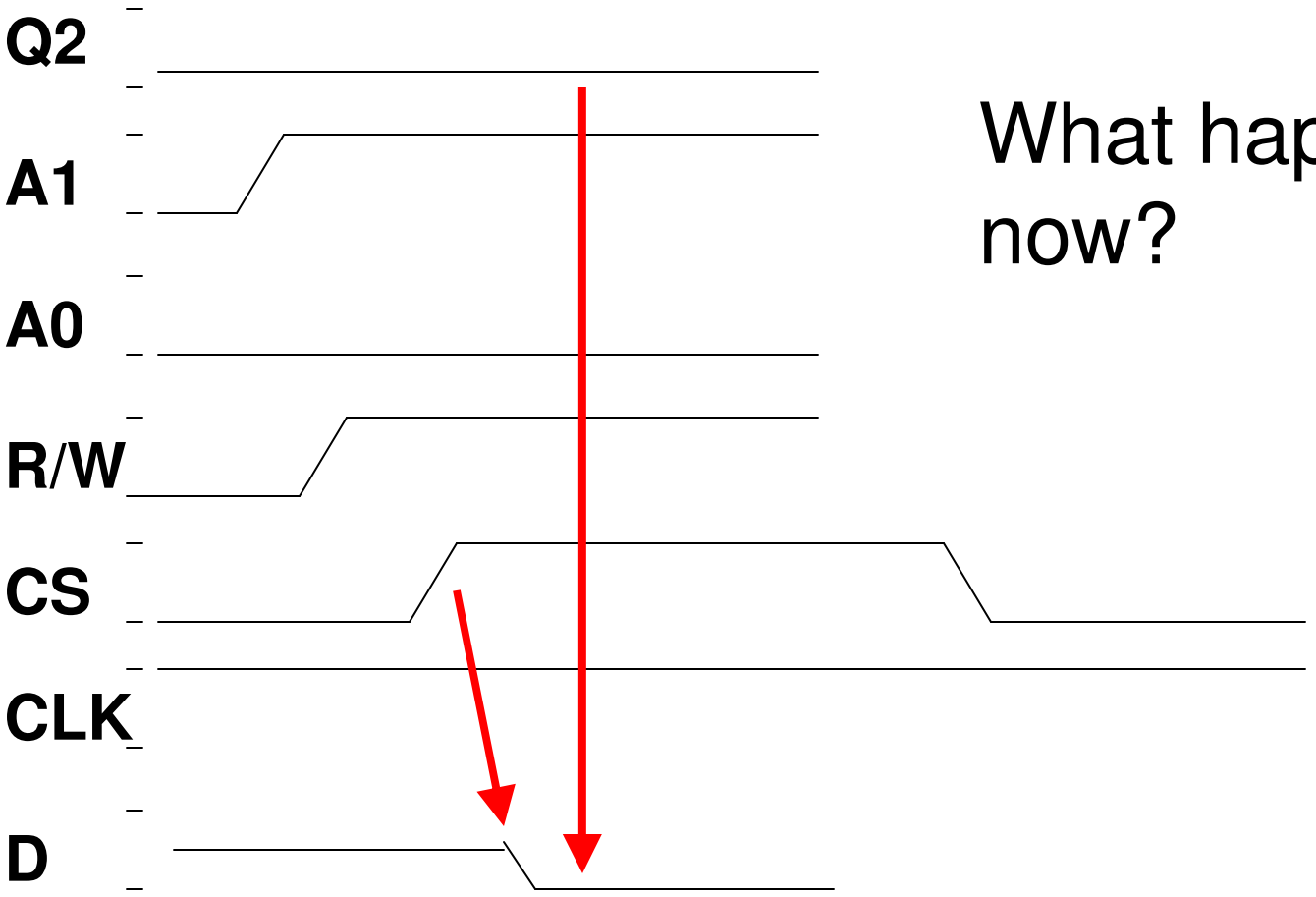
What happens next?

Memory Timing Diagram II



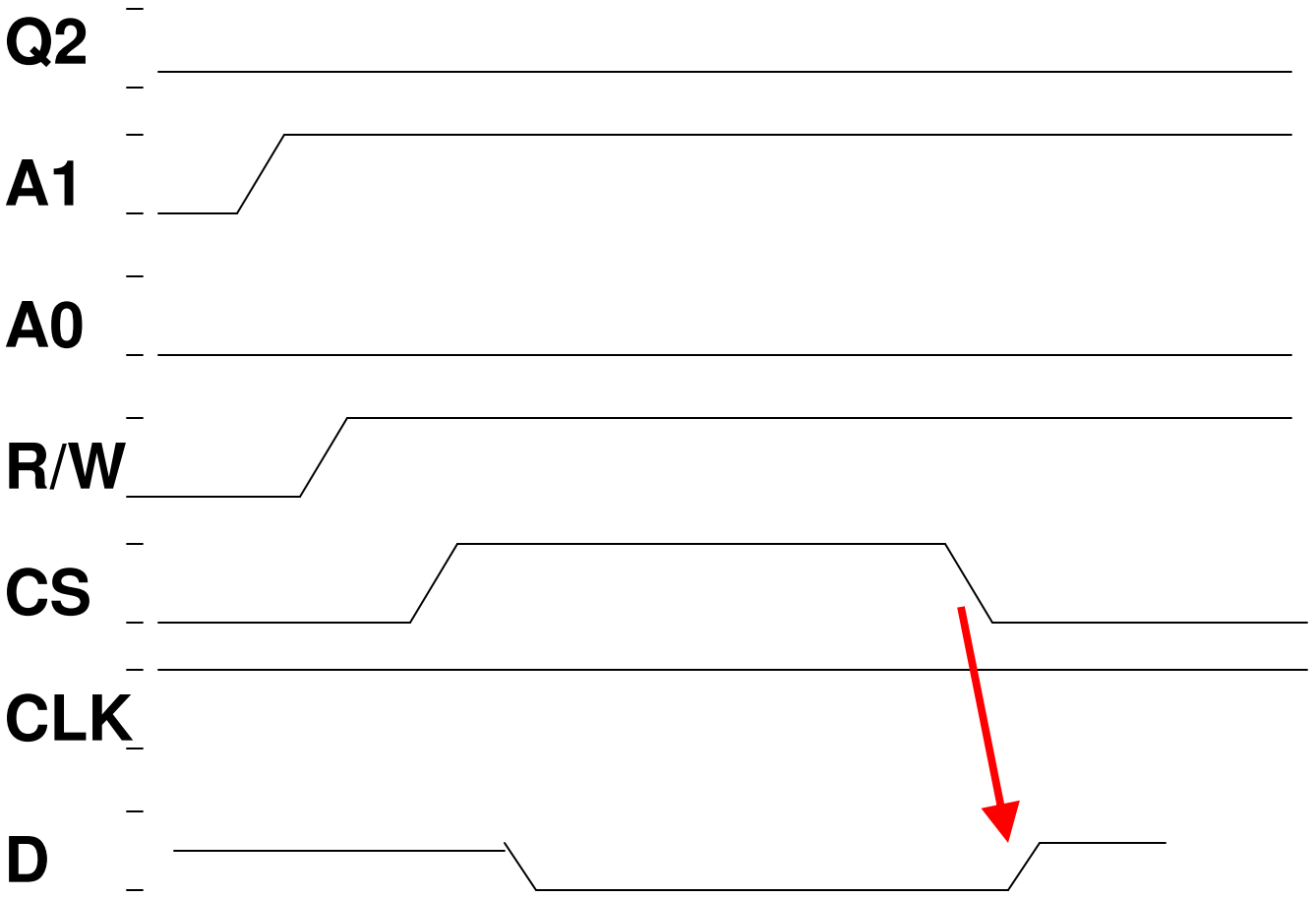
On chip select – drive data bus from Q2

Memory Timing Diagram II



What happens now?

Memory Timing Diagram II



Data bus returns to a non-driven state

Memory (cont)

- Memory is typically organized in groups of bits (8 is most common)
- For example, an entire byte may be stored at a particular address
- This means that the data bus is also “8 bits wide” (8 parallel lines)

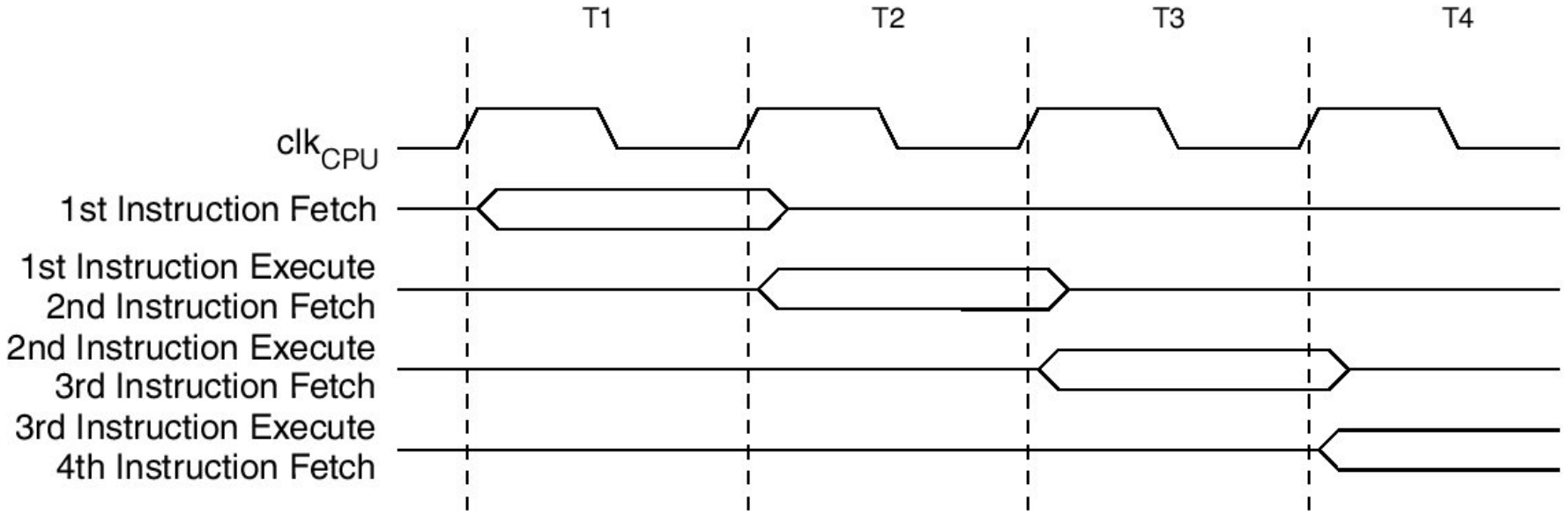
Components of a Microprocessor

- Registers (fast-access memory)
  - General purpose: used for data storage
  - Special purpose: used to control the behavior of the microprocessor and/or the devices connected to it
- Instruction decoder
  - Instructions are the primitive “actions” that the microprocessor can perform
  - Load/store to/from memory, AND, ADD, JUMP, TEST, …

Components of a Microprocessor

- Arithmetic Logical Unit (ALU)
- Memory control logic
- Timers
  - Including timing mechanisms for instruction fetch and execution
- Interrupt processor

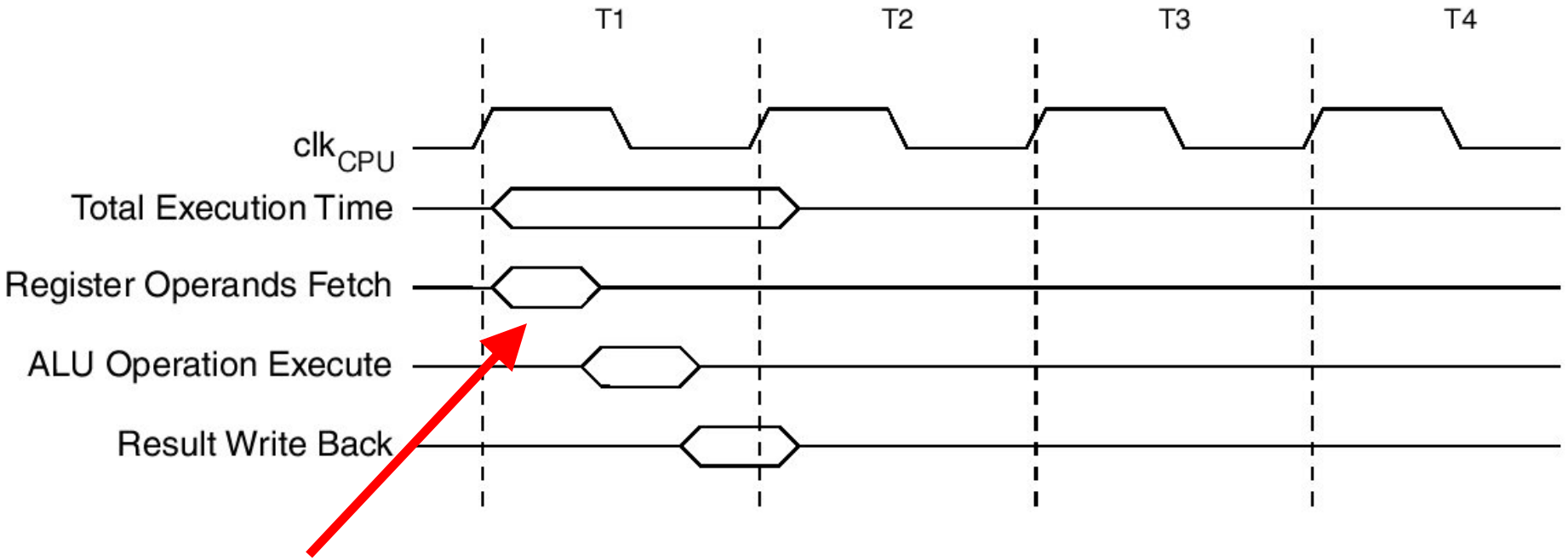
Instruction Fetch/Execution Cycle



From Atmel Mega8 spec

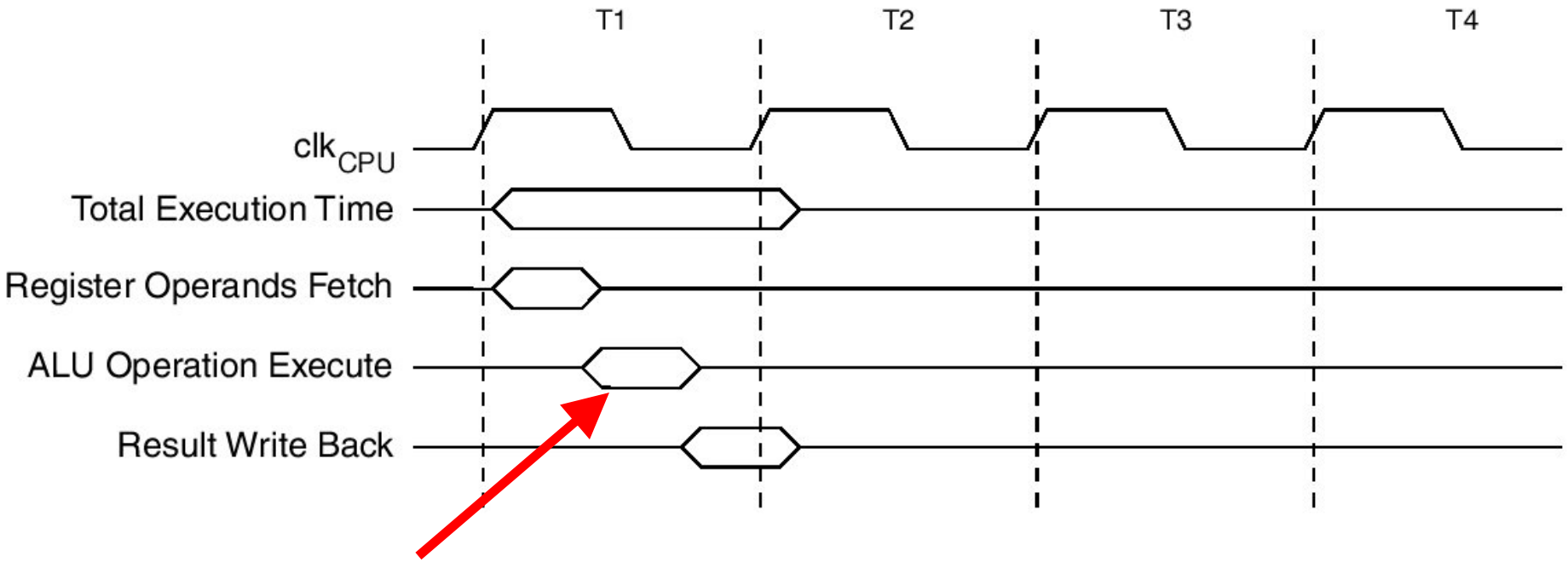
- While one instruction is being executed, the next is already being fetched from memory
- In many cases: each step happens on a single clock cycle

Instruction Execution Cycle



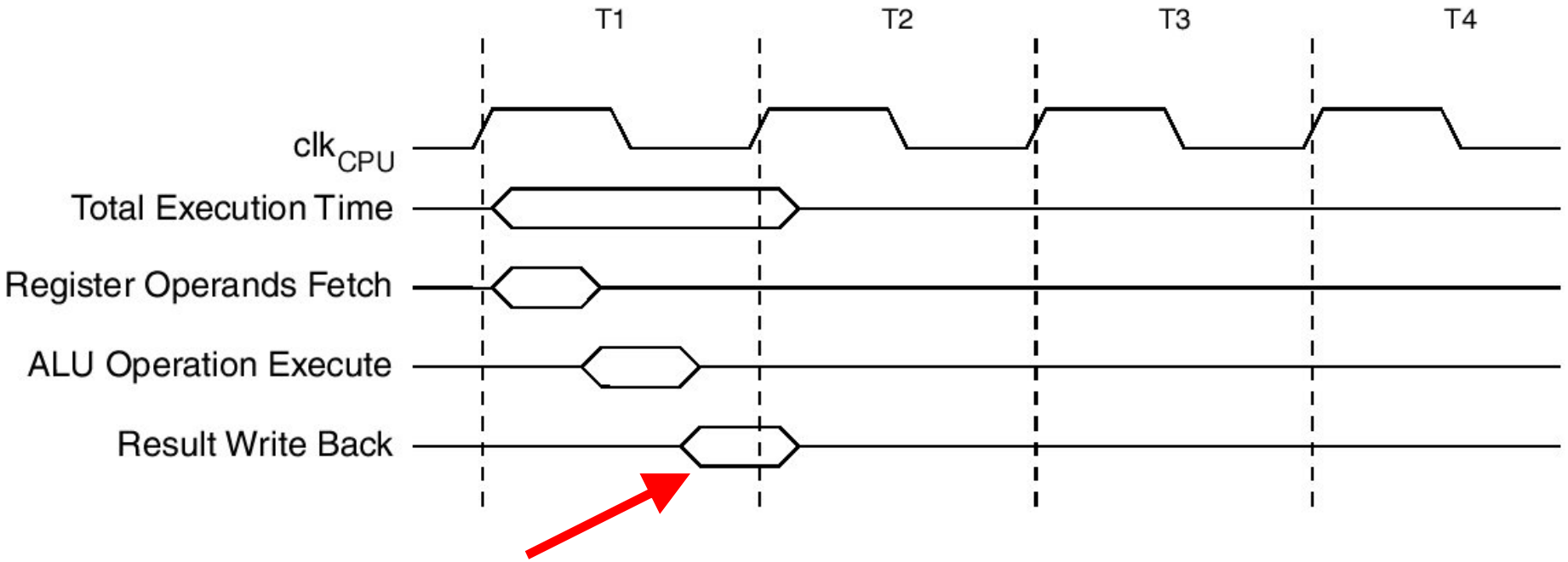
Address the registers and wait for the values to become available

Instruction Execution Cycle



Perform the operation dictated by the instruction

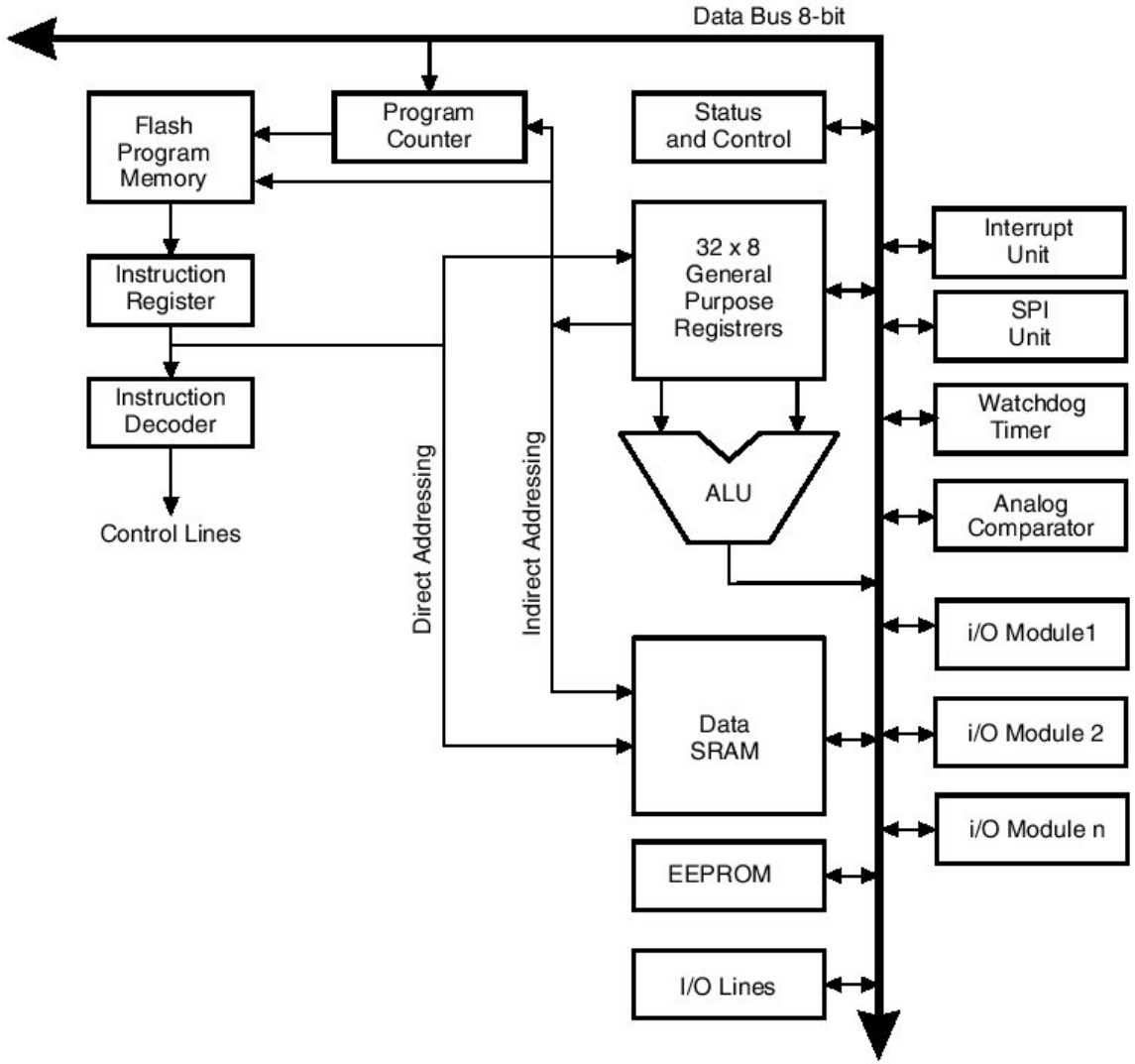
Instruction Execution Cycle



Result stored in destination register

Status register state changed

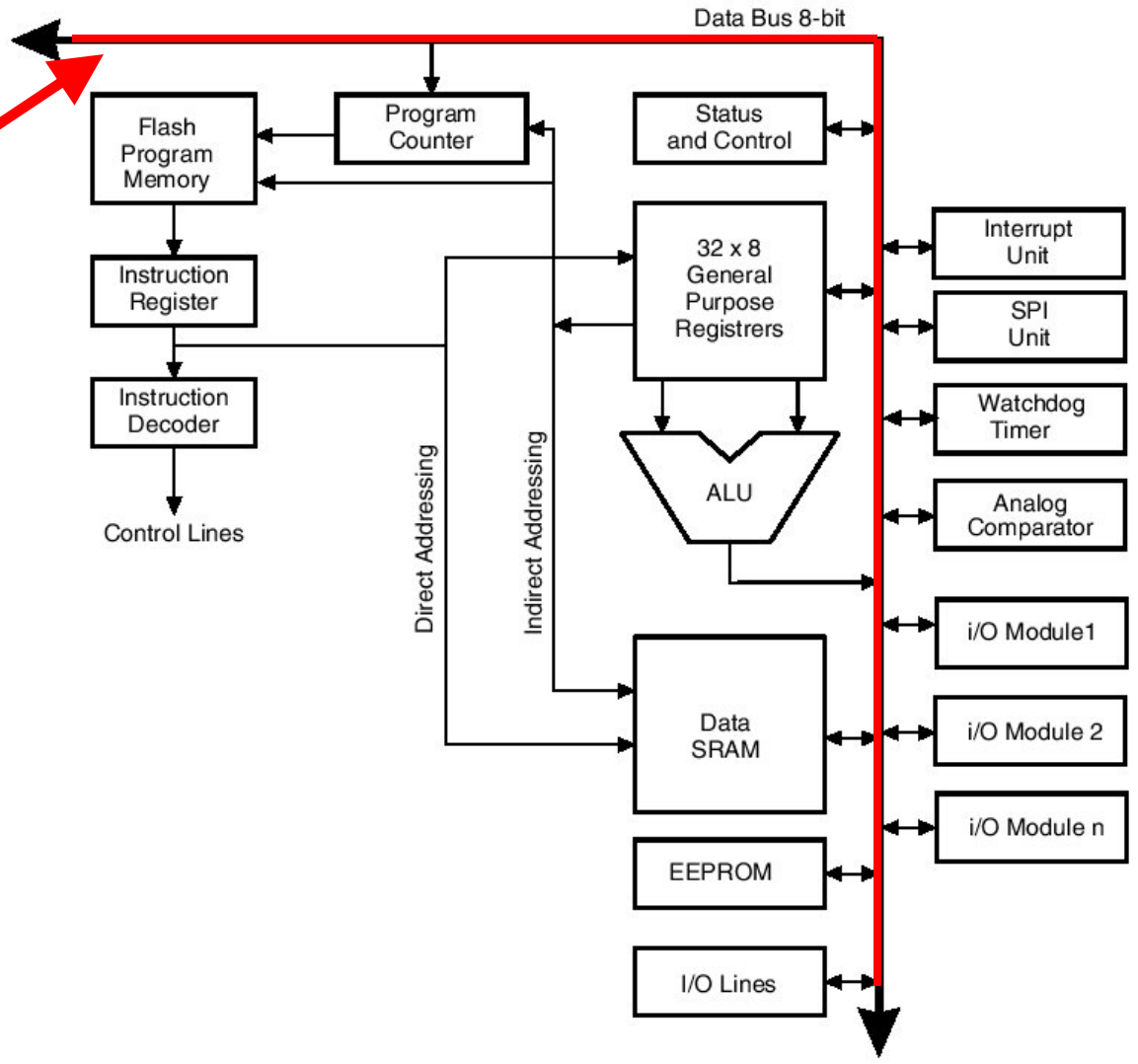
An Example: the Atmel Mega8



Atmel Mega8

8-bit data bus

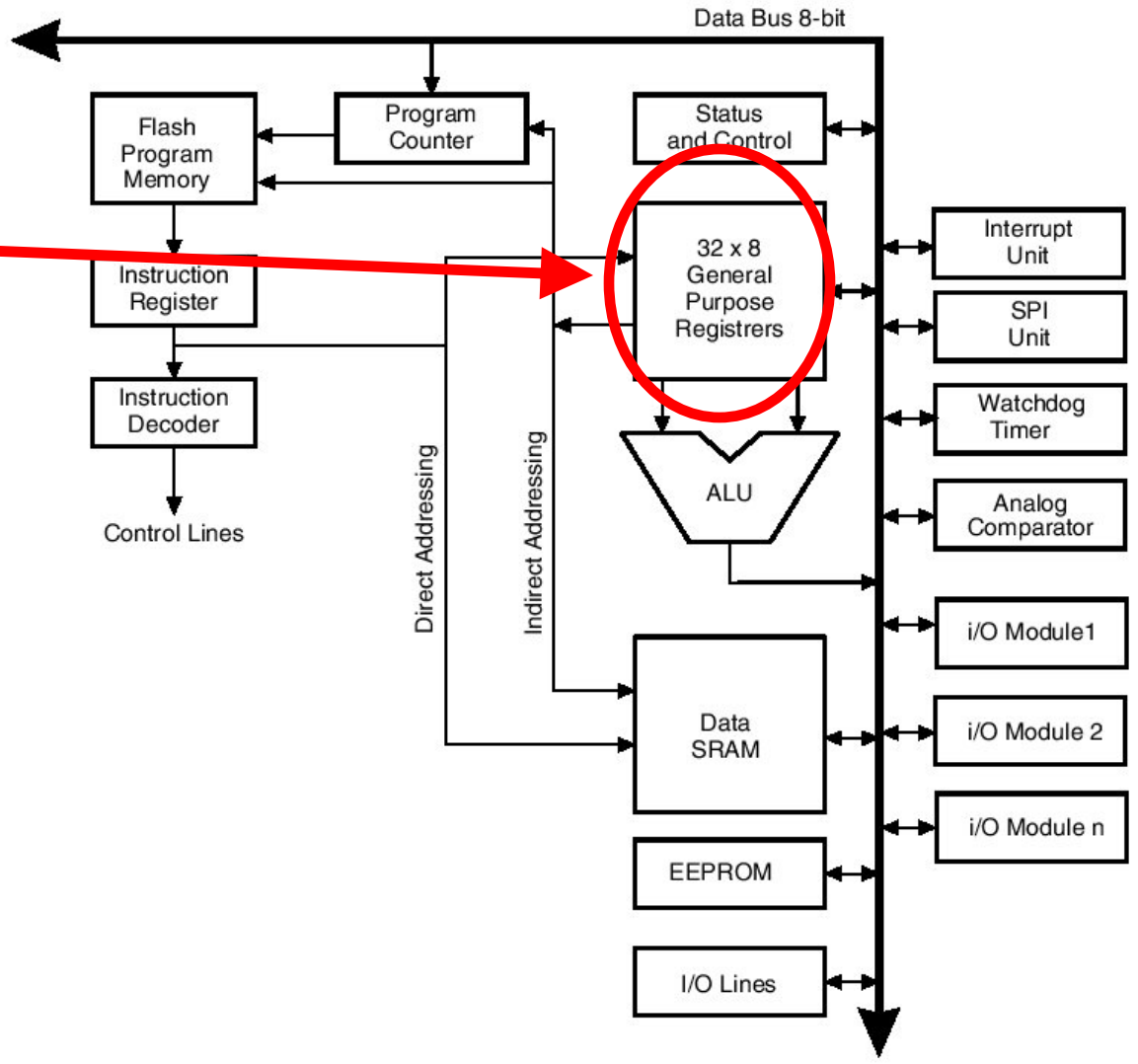
- Primary mechanism for data exchange



Atmel Mega8

32 general purpose registers

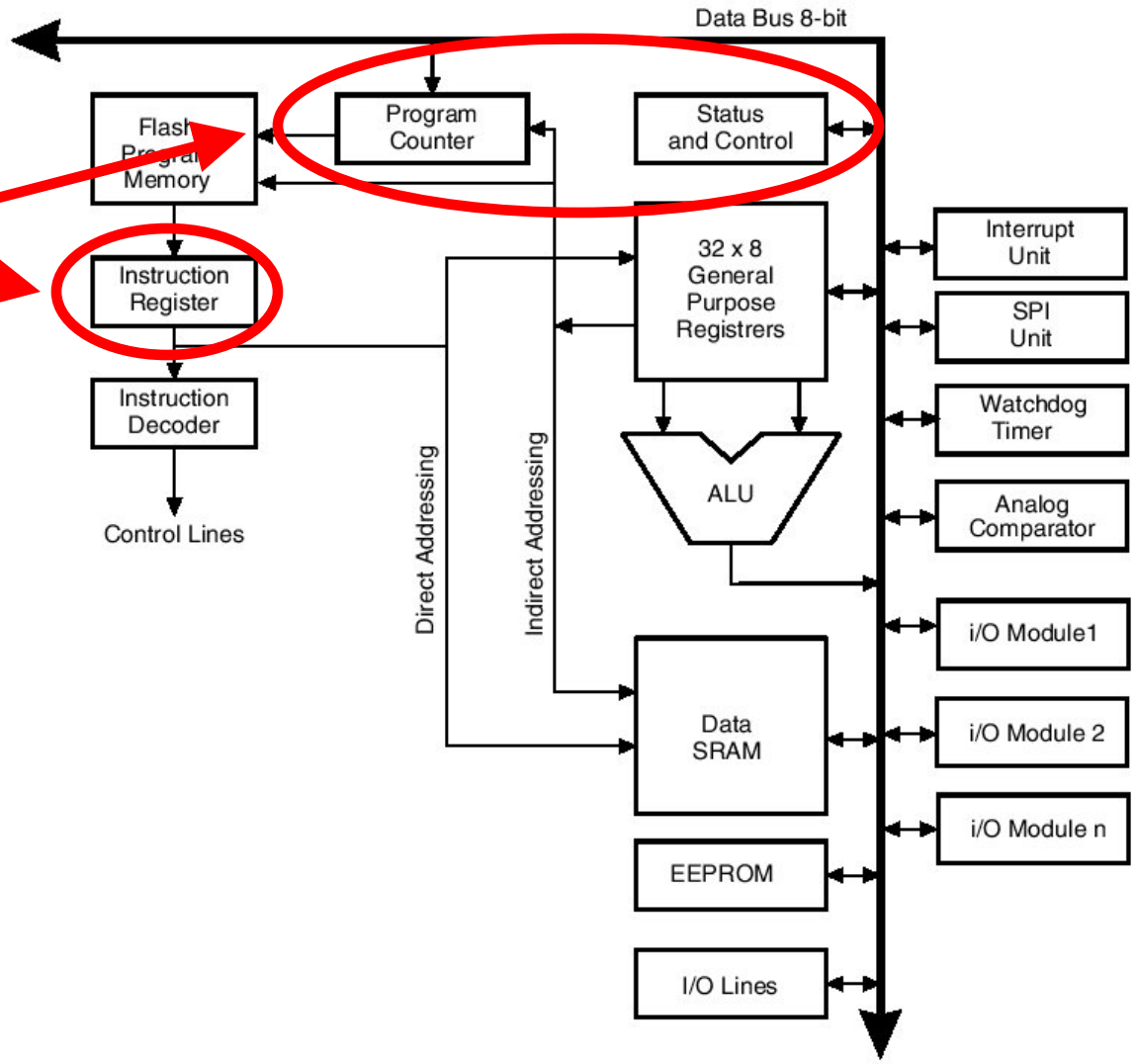
- 8 bits wide
- 3 pairs of registers can be combined to give us 16 bit registers



Atmel Mega8

Special purpose registers

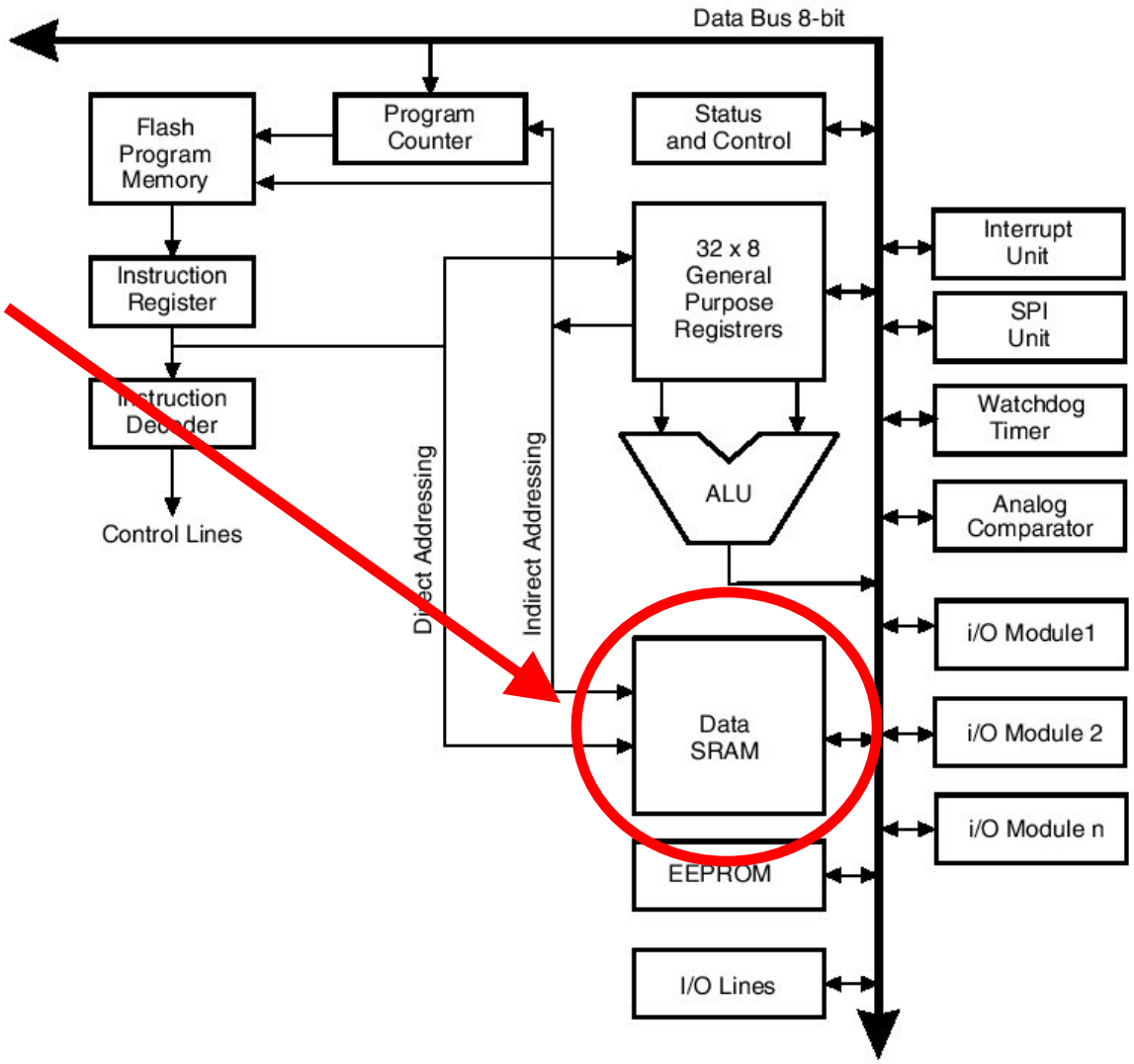
- Control of the internals of the processor



Atmel Mega8

Random Access Memory (RAM)

- 1 KByte in size

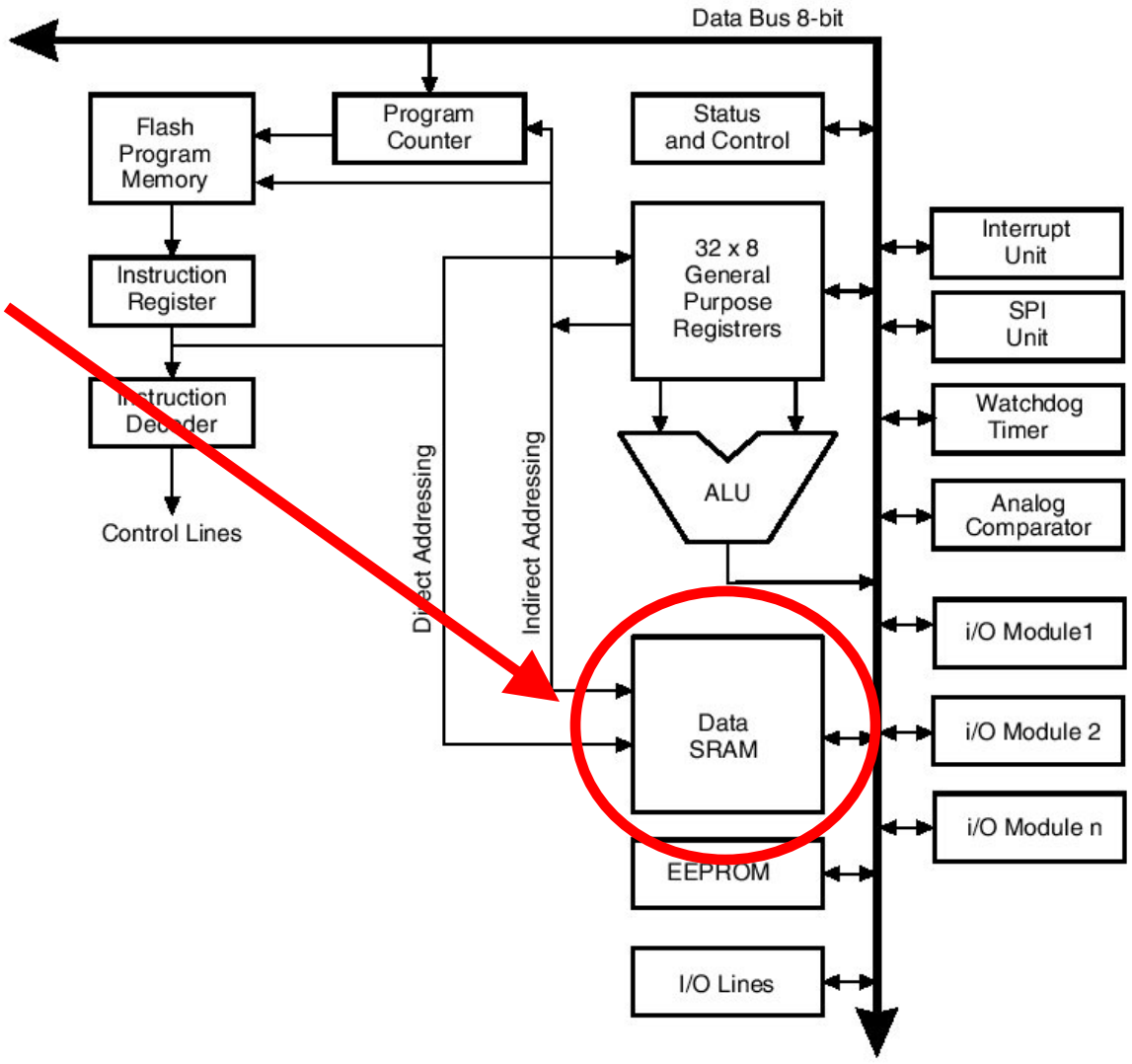


Atmel Mega8

Random Access Memory (RAM)

- 1 KByte in size

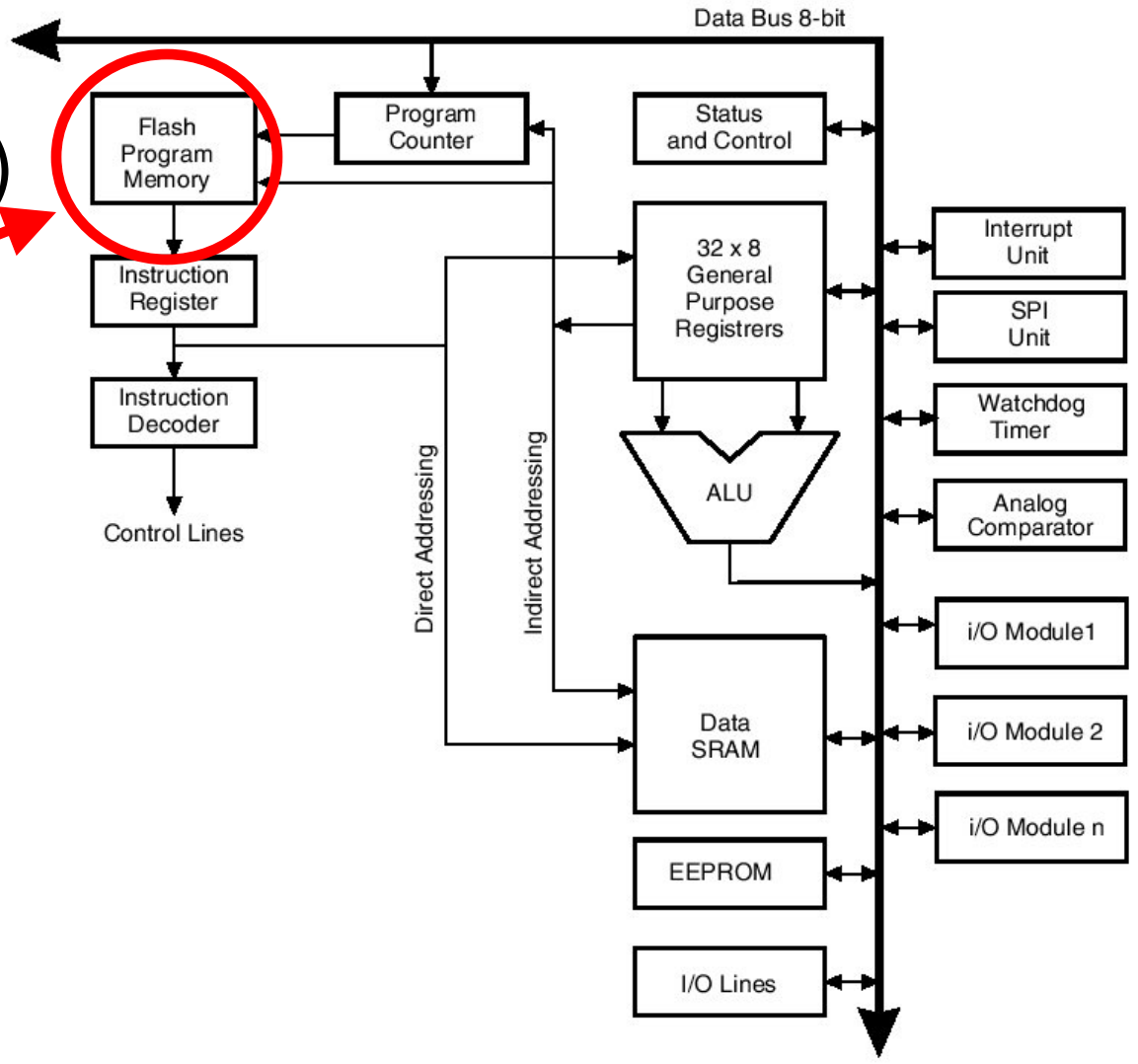
Note: in high-end processors, RAM is a separate component



Atmel Mega8

Flash (EEPROM)

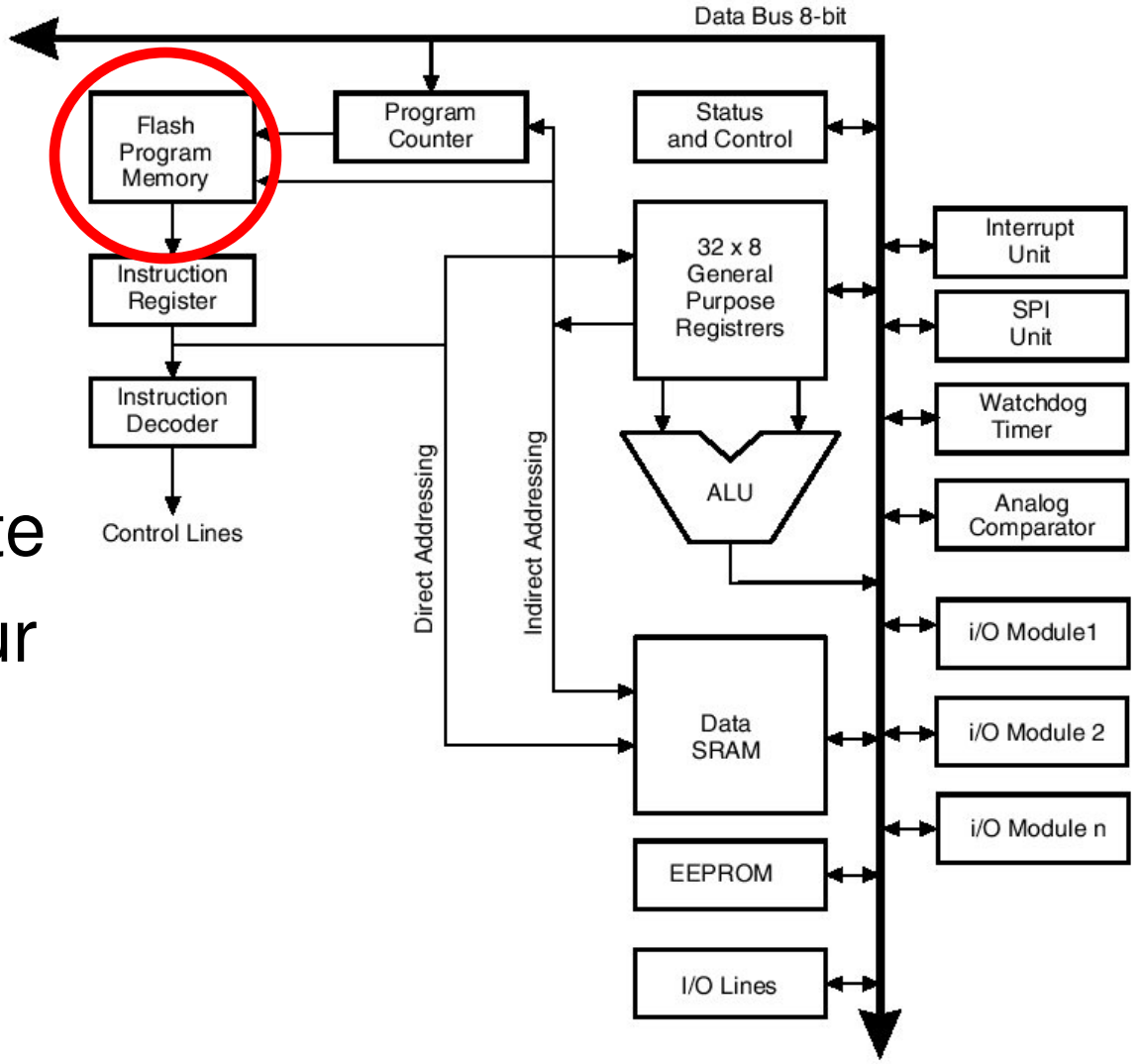
- Program storage
- 8 KByte in size



Atmel Mega8

Flash (EEPROM)

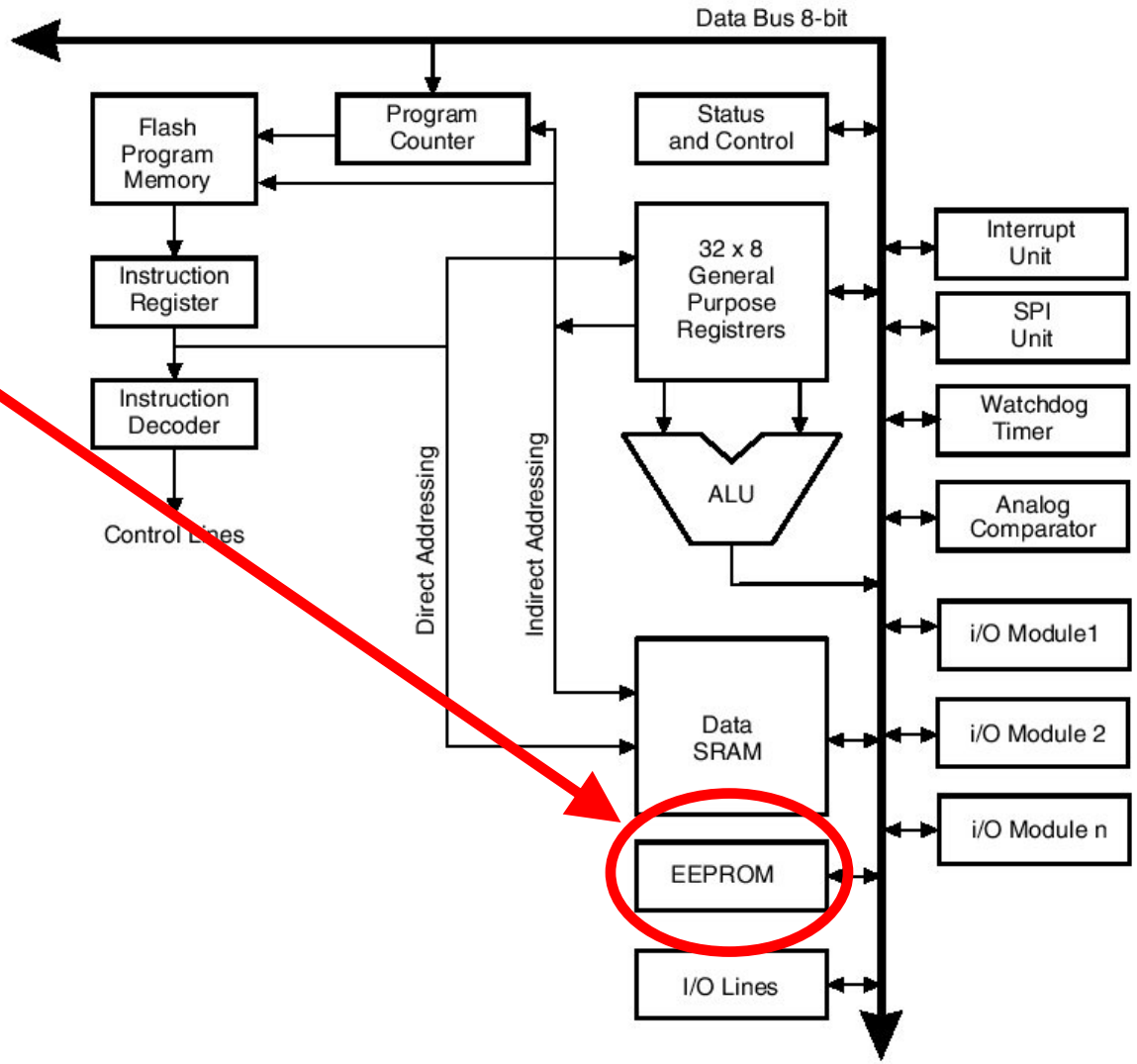
- In this and many microcontrollers, program and data storage is separate
- Not the case in our general purpose computers



Atmel Mega8

EEPROM

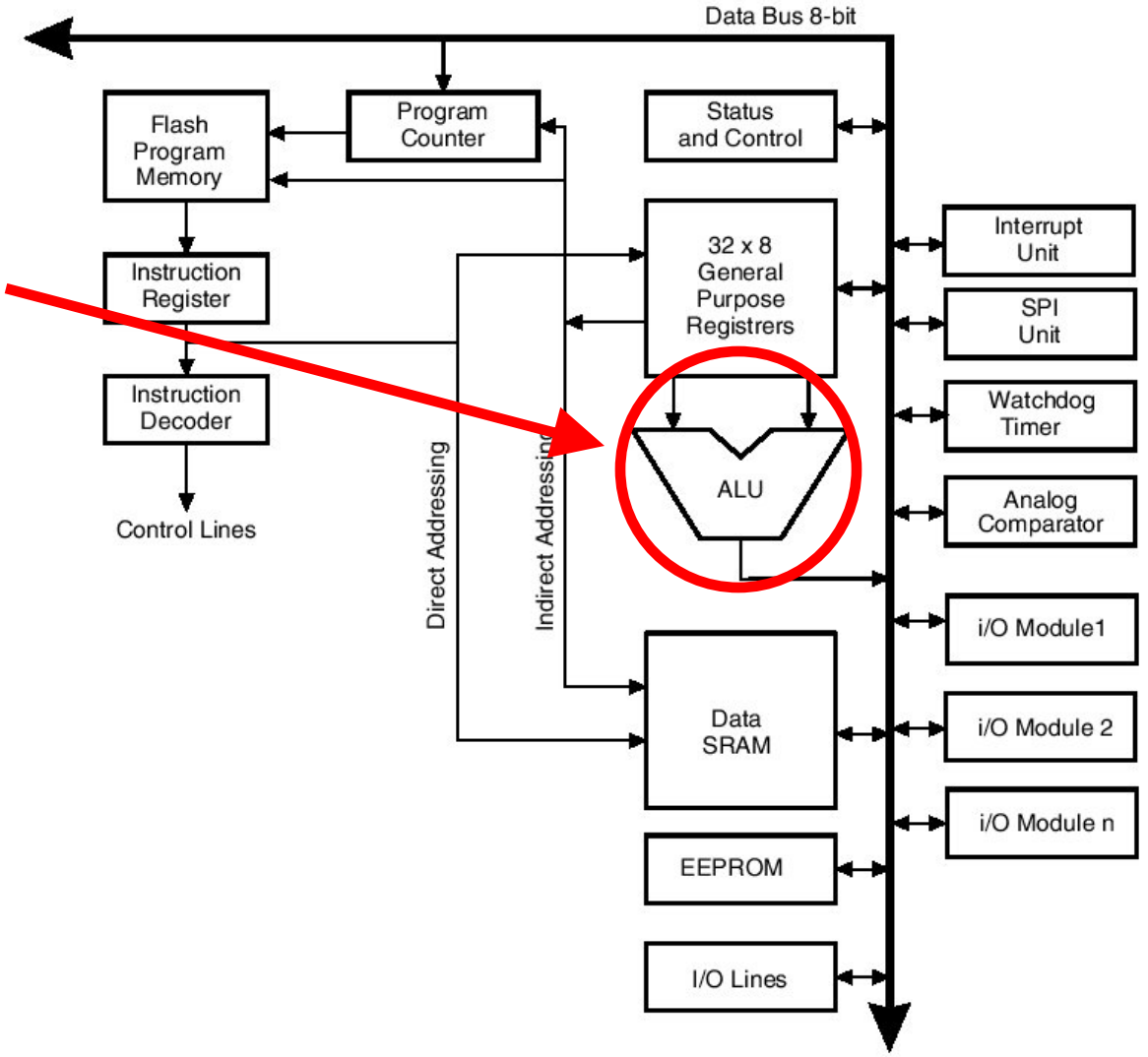
- Permanent data storage



Atmel Mega8

Arithmetic Logical Unit

- Data inputs from registers
- Control inputs not shown (derived from instruction decoder)



Machine-Level Programs

Machine-level programs are stored as sequences of machine instructions

- Stored in program memory
- Execution is generally sequential (instructions are executed in order)
- But – with occasional “jumps” to other locations in memory

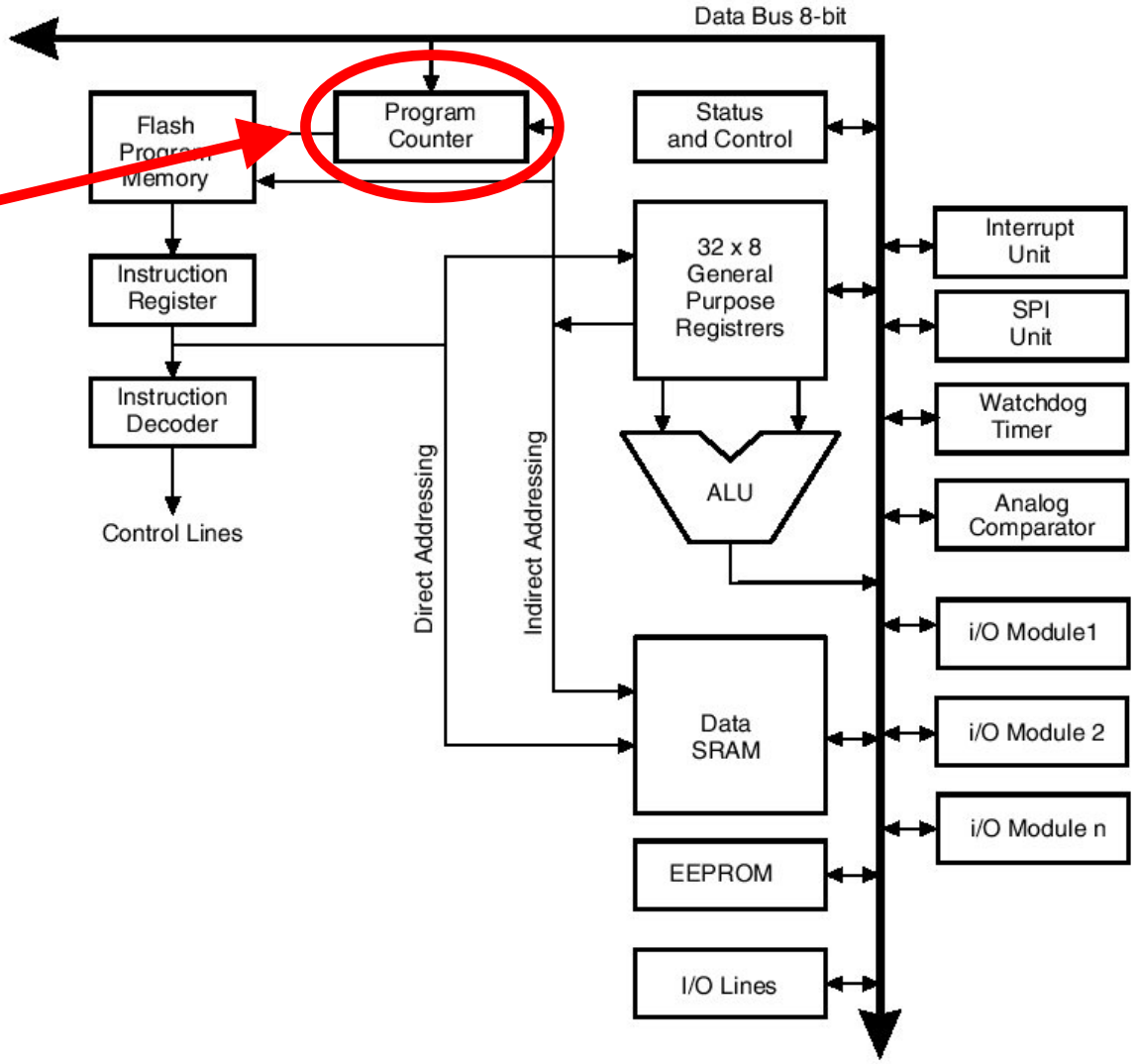
Types of Instructions

- Memory operations: transfer data values between memory and the internal registers
- Mathematical operations: ADD, SUBTRACT, MULT, AND, etc.
- Tests: value == 0, value > 0, etc.
- Program flow: jump to a new location, jump conditionally (e.g., if the last test was true)

Atmel Mega8: Decoding Instructions

Program counter

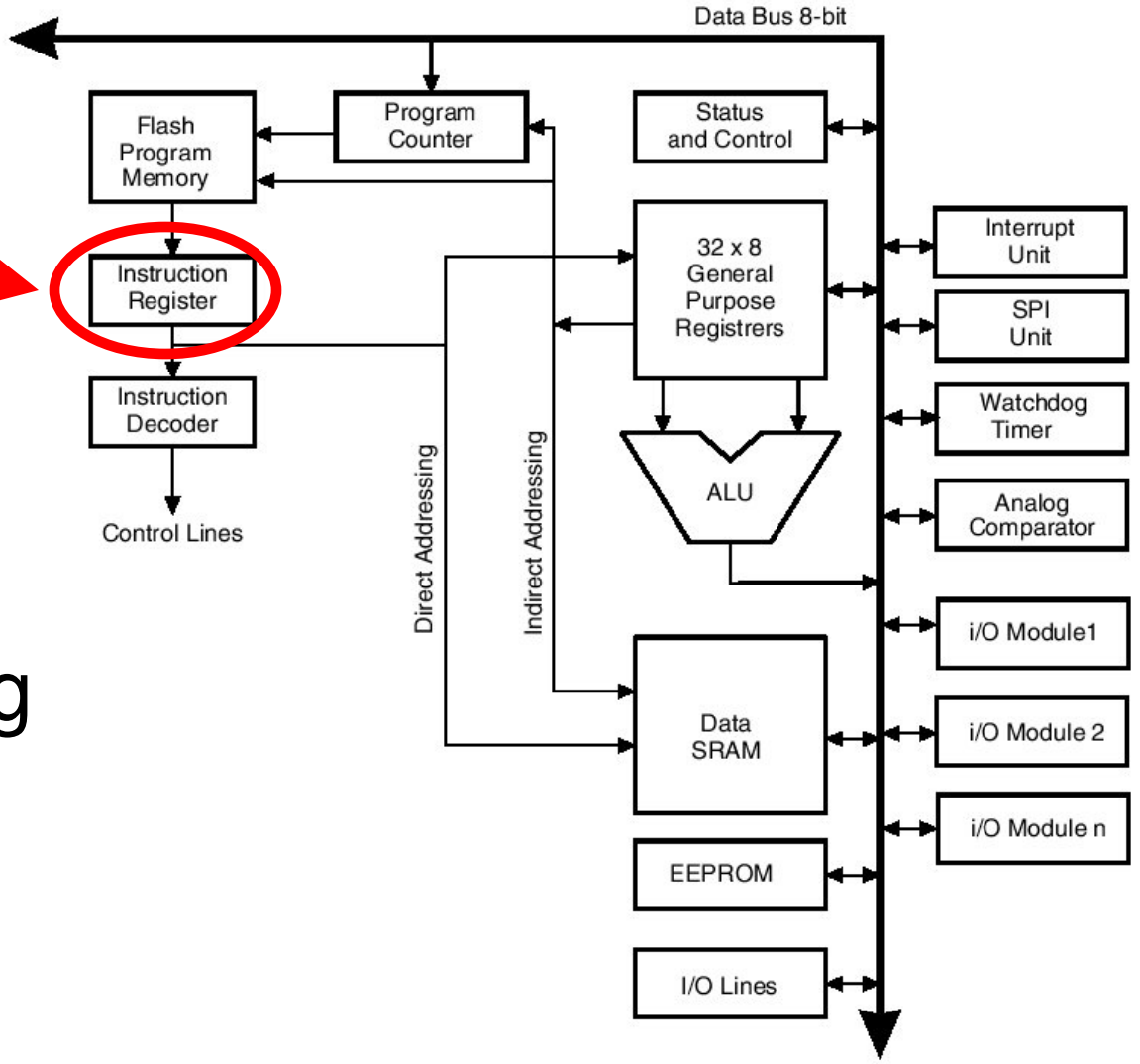
- Address of currently executing instruction



Atmel Mega8: Decoding Instructions

Instruction register

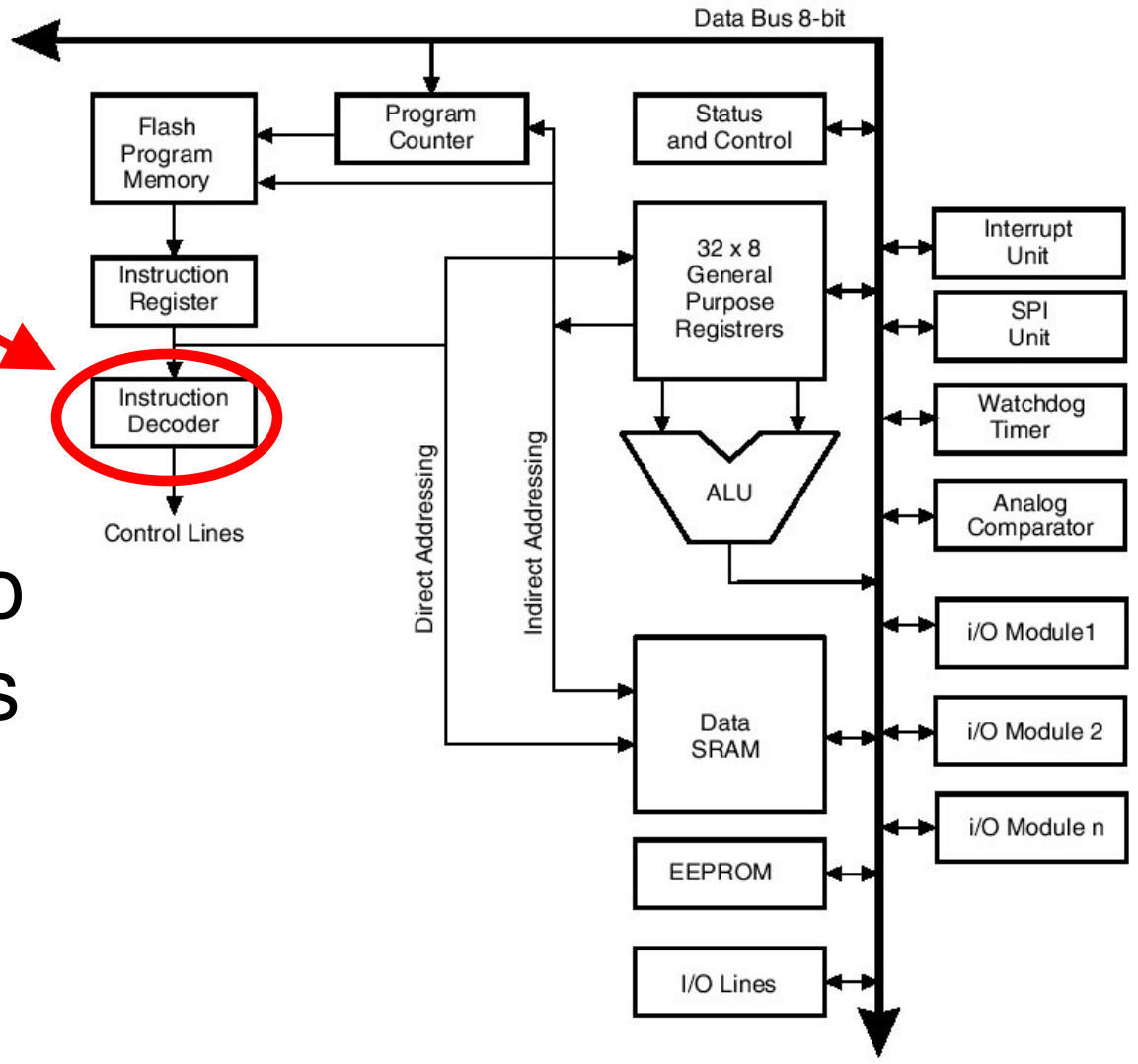
- Stores the machine-level instruction currently being executed



Atmel Mega8

Instruction decoder

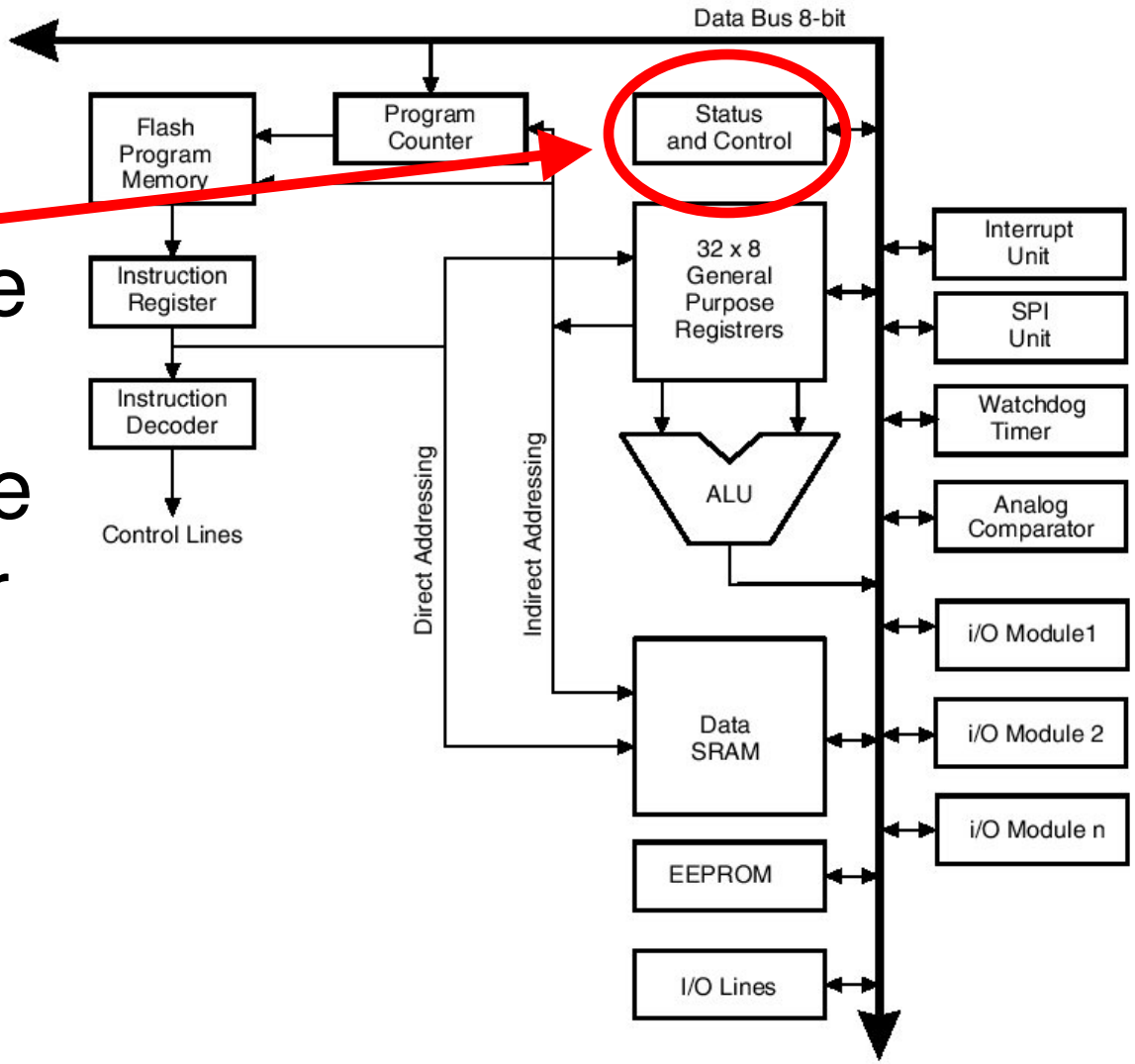
- Translates current instruction into control signals for the rest of the processor



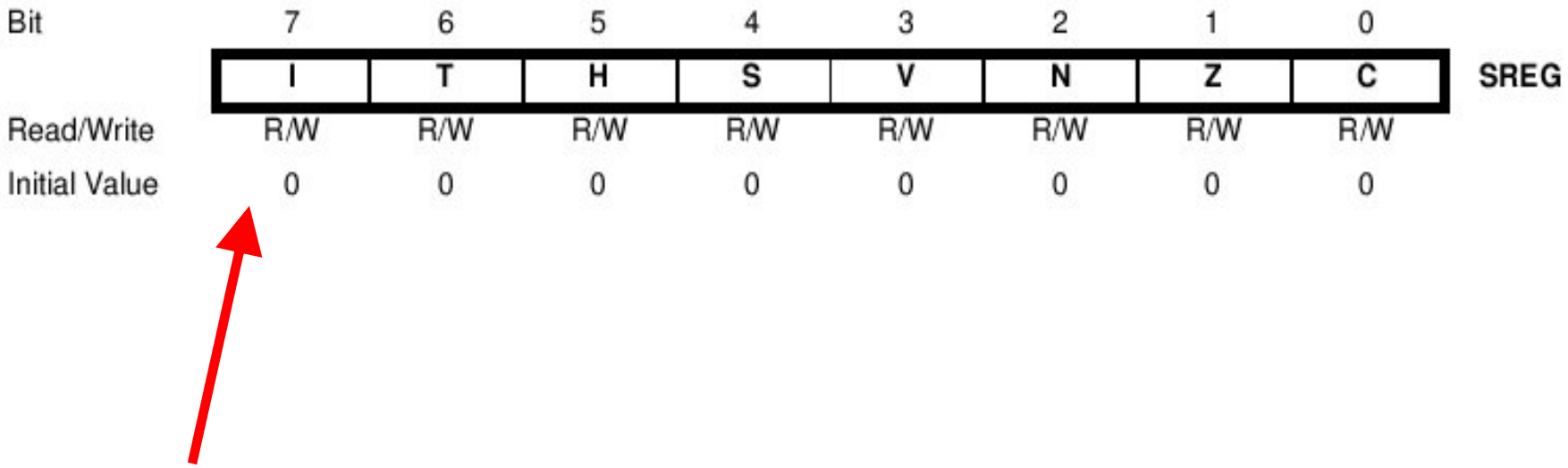
Atmel Mega8

Status register

- Many machine instructions affect the state of this register



Mega8 Status Register



Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Interrupt enable

- If ‘1’, the currently executing program can be interrupted by another event (e.g., a byte arriving through the serial port)

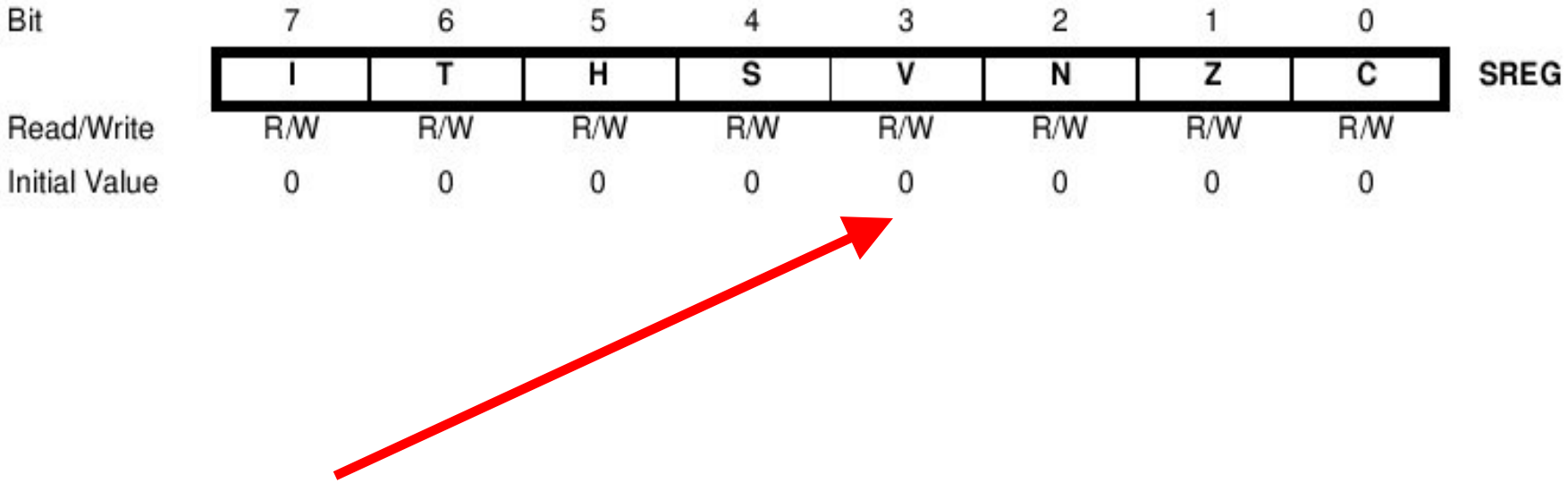
Mega8 Status Register

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Half carry flag

- Set if an arithmetic operation resulted in a carry from the first nybble to the next

Mega8 Status Register



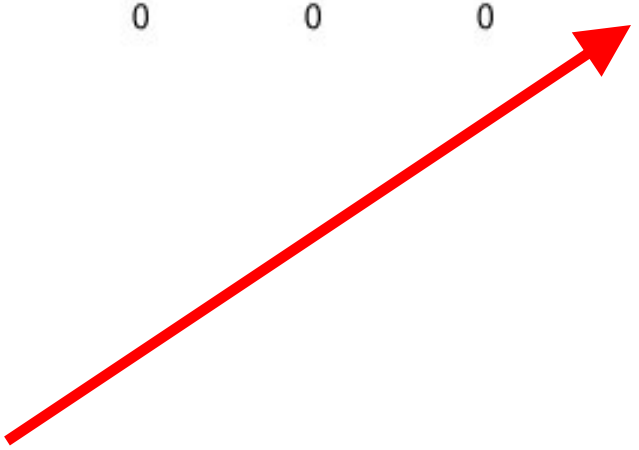
Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Two's complement overflow flag

- Set if an arithmetic operation resulted in an overflow in two's complement (e.g., incrementing an 8-bit number whose value is 127)

Mega8 Status Register

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

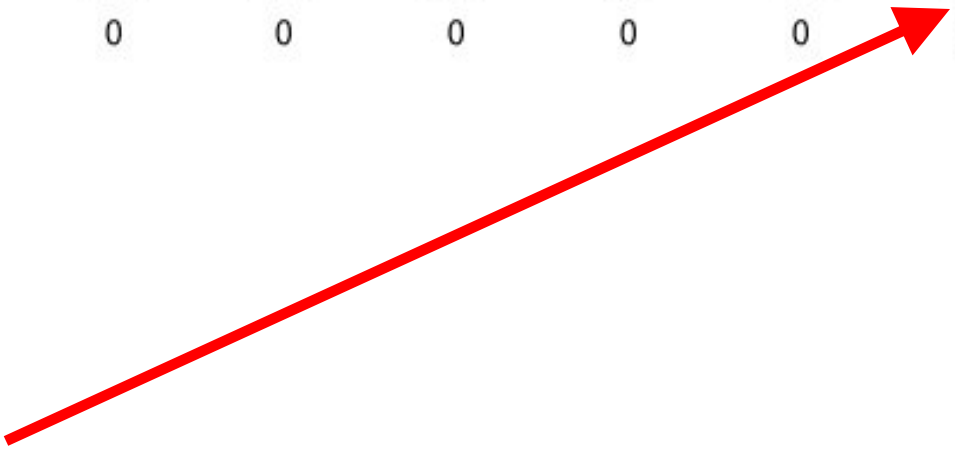


Negative flag

- Set if an arithmetic operation resulted in a negative value

Mega8 Status Register

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

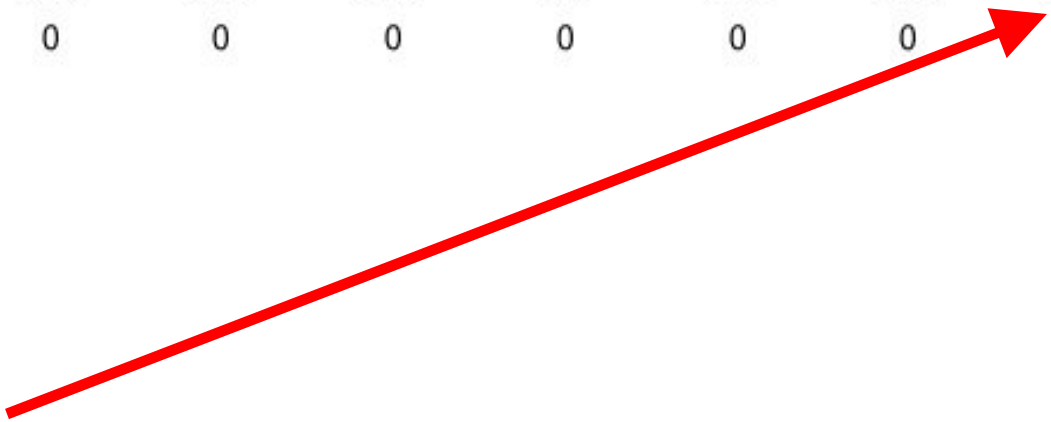


Zero flag

- Set if an arithmetic operation resulted in a value of zero

Mega8 Status Register

Bit	7	6	5	4	3	2	1	0	
	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	



Carry flag

- Set if an arithmetic operation resulted in a carry (with an unsigned value)

Next Time

- Microprocessors in practice
  - Coding
  - Digital I/O

- Readings:
  - Make sure that you are caught up on the Chapter 4 readings
  - Embedded C Programming: C Review and Architecture Sections (see schedule)