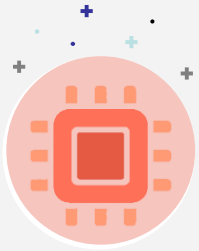
1402/11/22



Microcontrollers

Lecture 2:

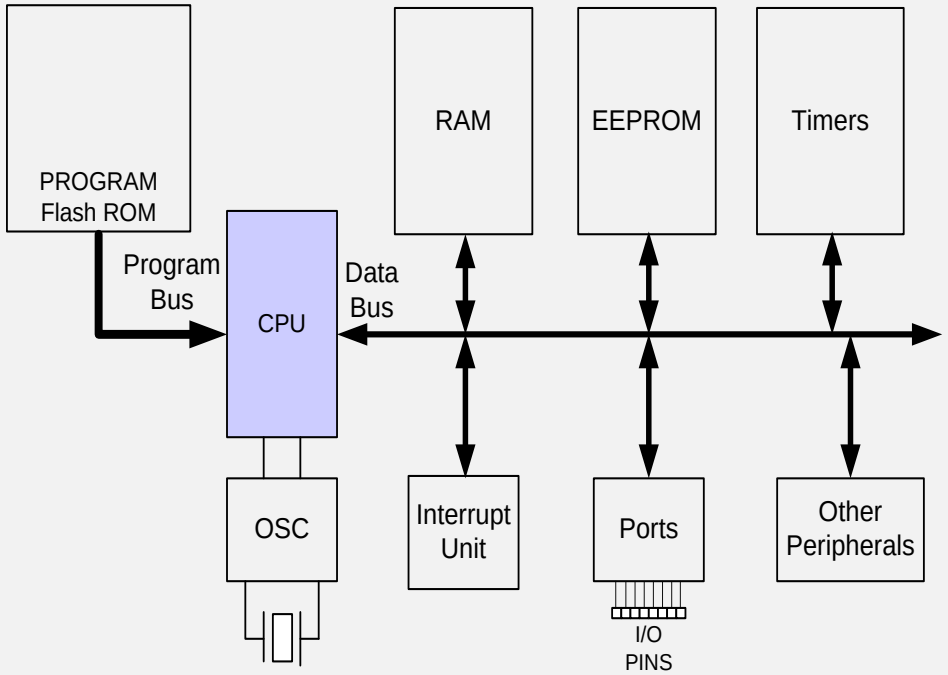
Introduction to AVR Registers and Architecture

By: M.Razeghizadeh

Start now!

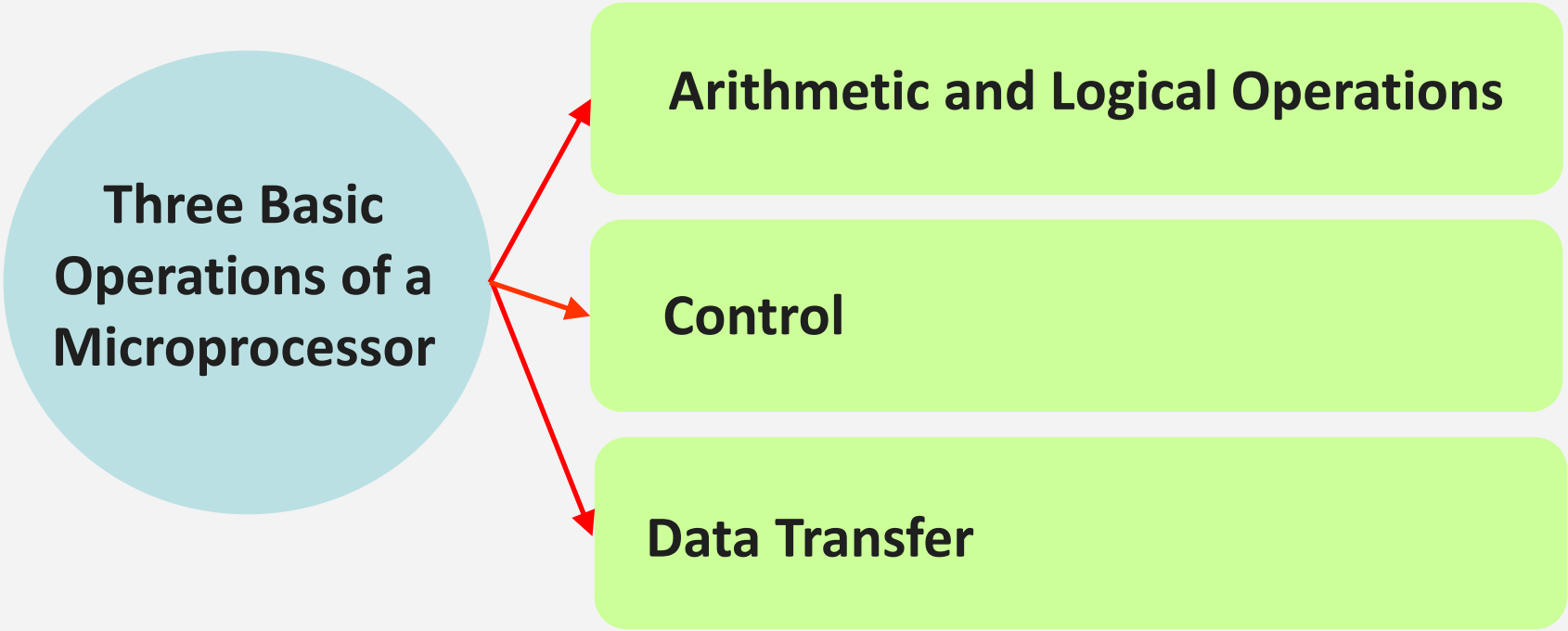
Contents of This Slide

- AVR's CPU
  - Its architecture
  - Some simple programs
- Data Memory access
- Program memory
- RISC architecture

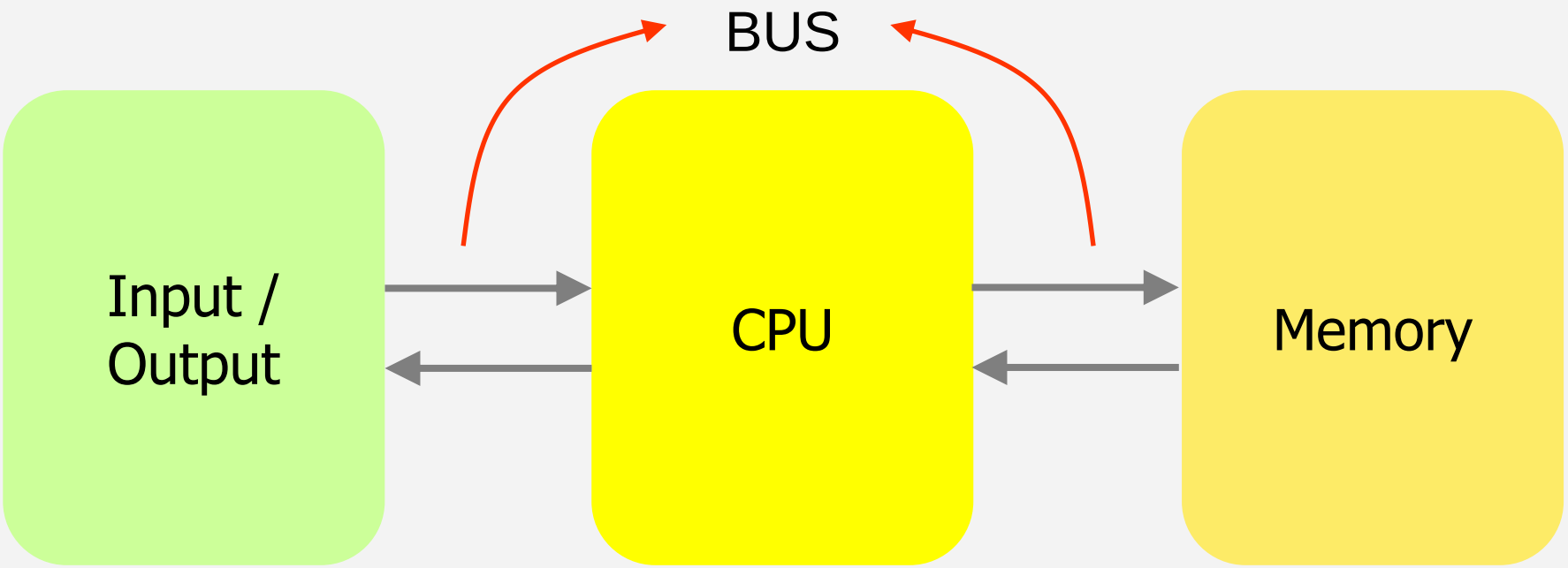


What is a microprocessor?

A microprocessor is a device that can be programmed to perform various operations; it is a programmable controller.



a microcontroller system



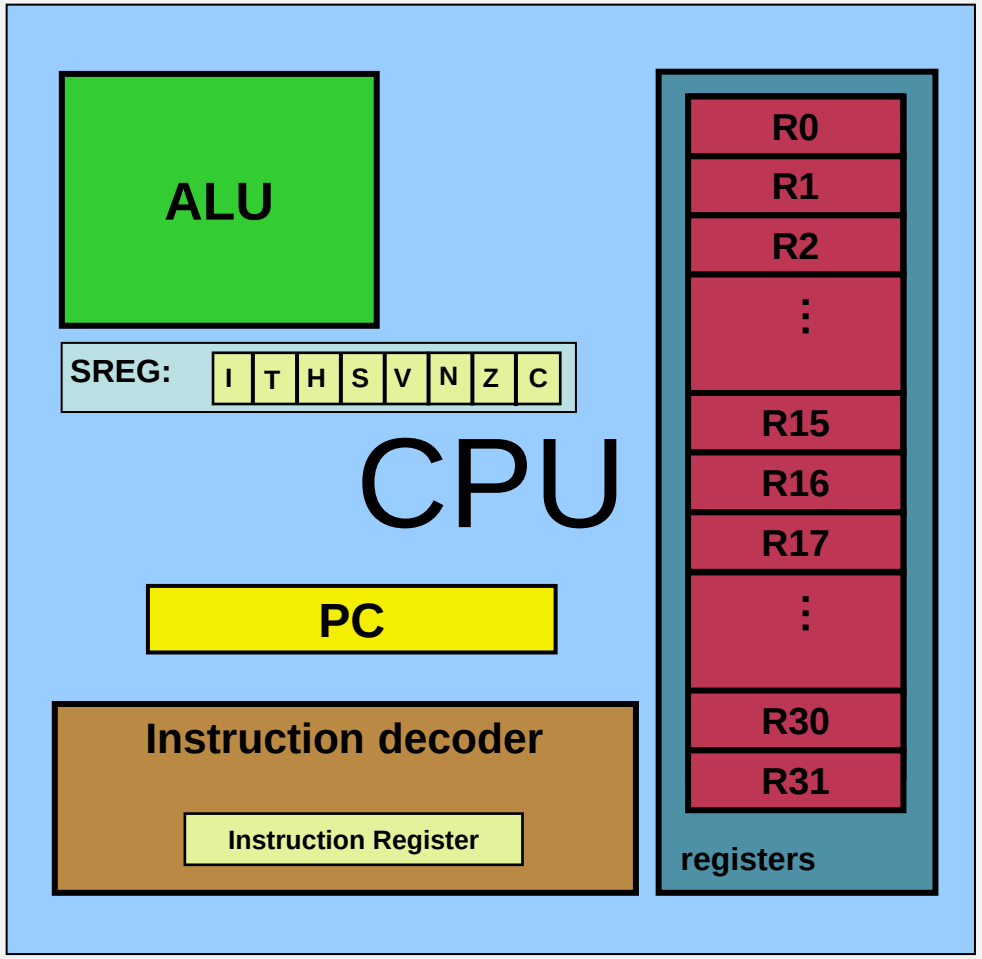
- CPUs use registers to store data temporarily.
- To program CPU, we must understand the registers and architecture of a given CPU and the role they play in processing data.

AVR's CPU

- AVR's CPU
  - ALU
  - 32 General Purpose registers (R0 to R31)
  - PC register
  - Instruction decoder

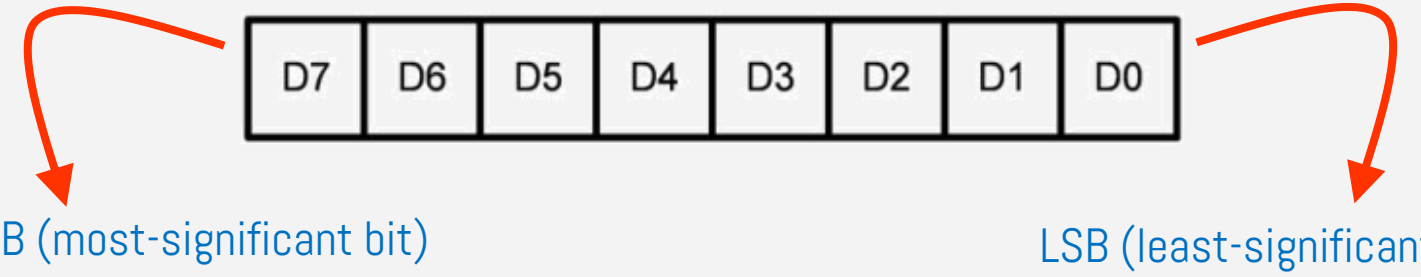
`

AVR's CPU



General Purpose Registers (GPRs)

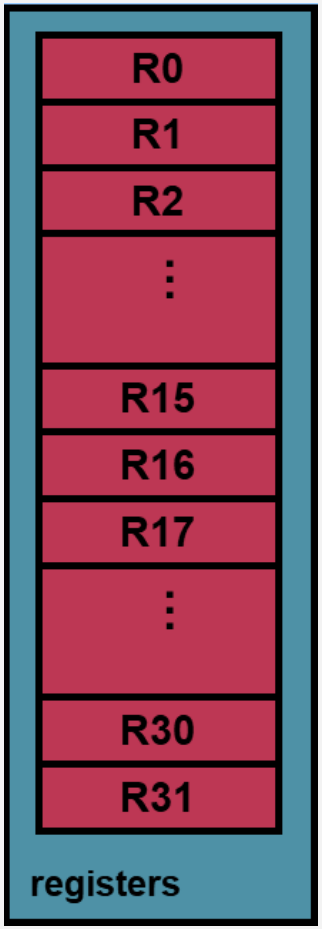
- AVR microcontrollers have many registers for arithmetic and logic operations.
- In the CPU, registers are used to store information temporarily.
- That information could be a byte of data to be processed, or an address pointing to the data to be fetched.
- The vast majority of AVR registers are 8-bit registers.
- In the AVR there is only one data type: 8-bit.



- any data larger than 8 bits must be broken into 8-bit chunks before it is processed.

General Purpose Registers (GPRs)

- In AVR there are 32 general purpose registers (R0–R31).
- located in the lowest location of memory address.
- All of these registers are 8 bits.
- They can be used by all arithmetic and logic instructions.



Command Template

1. Loading values into the general purpose registers:

LDI (Load Immediate)

- LDI Rd, k
  - Its equivalent in high level languages:
    Rd = k

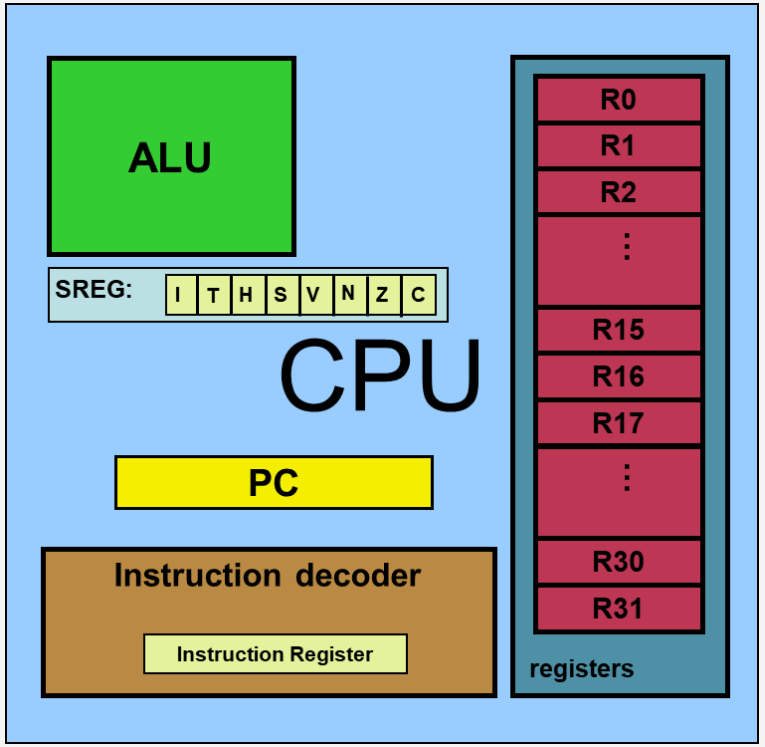
K: an 8-bit = 0–255 in decimal, or 00–FF in hex

Example

- **LDI R16,53**
  - **R16 = 53**
- LDI R19,$27
- LDI R23,0x27
  - R23 = 0x27
- LDI R23,0b11101100

Some simple instructions

To understand the use of the general purpose registers, show it in the context of simple instructions.



Command Template

2. Arithmetic calculation

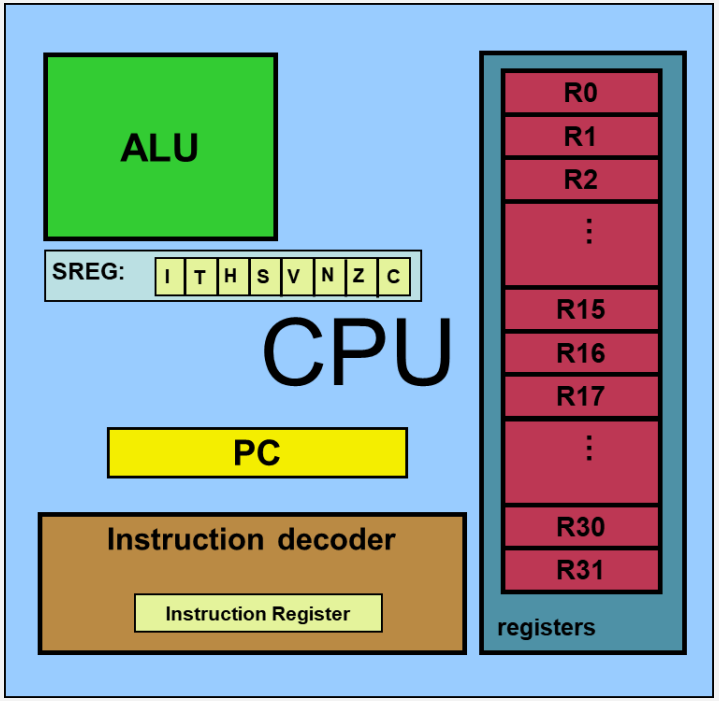
There are some instructions for doing Arithmetic and logic operations; such as:

ADD, SUB, MUL, AND, etc.

Example

- **ADD Rd,Rs**
  - **Rd = Rd + Rs**
  - ADD R25, R9
    - R25 = R25 + R9
  - ADD R17,R30
    - R17 = R17 + R30

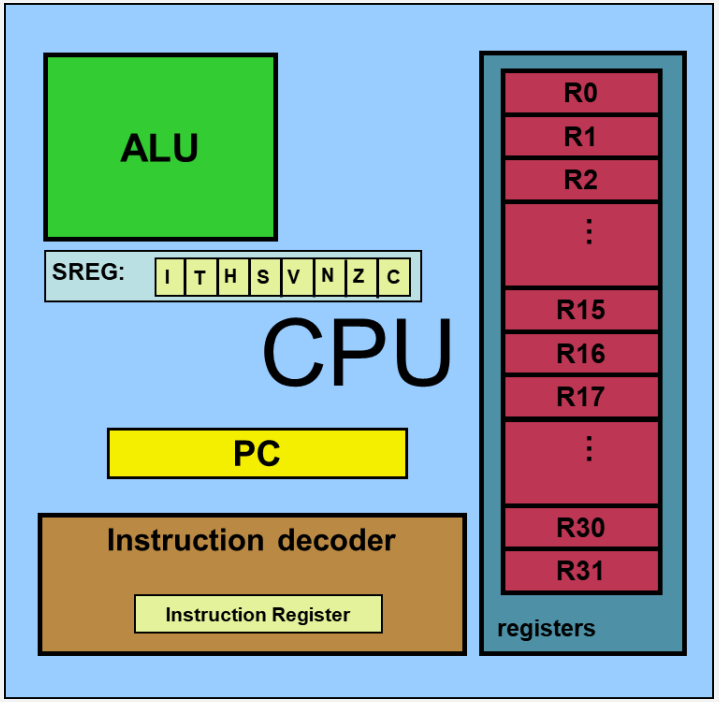
Some simple instructions



Simple Program

Write a program that calculates 19 + 95 ?

Some simple instructions



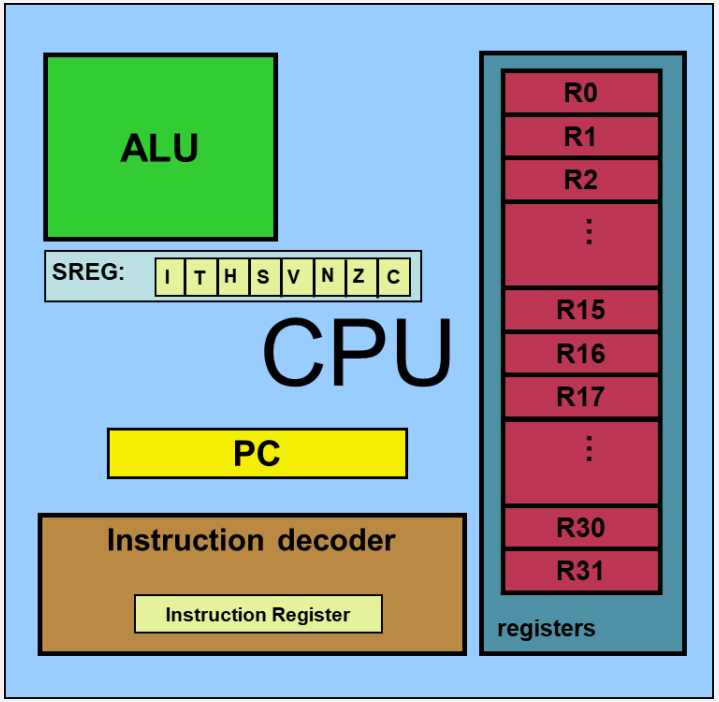
Simple Program

Write a program that calculates 19 + 95 ?

Answer

```
LDI R16, 19        ;R16 = 19
LDI R20, 95        ;R20 = 95
ADD R16, R20       ;R16 = R16 + R20
```

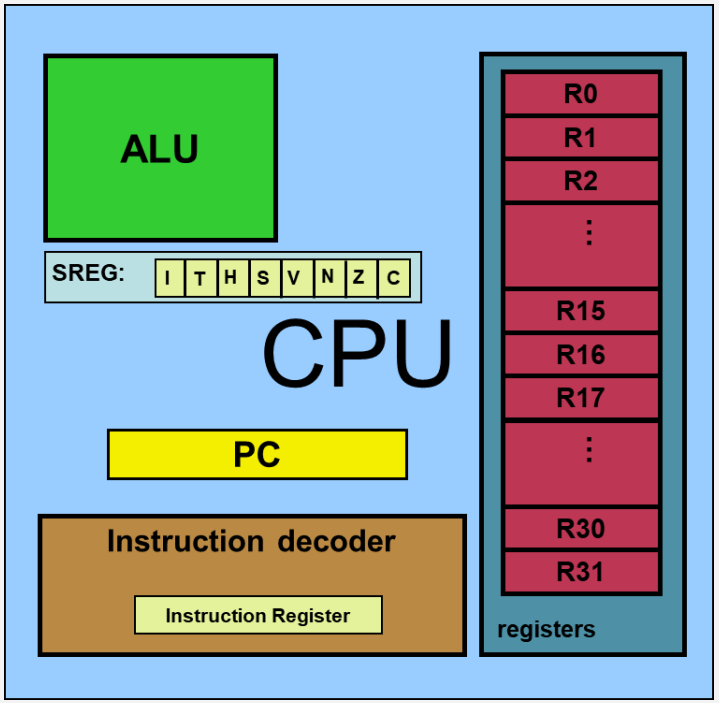
Some simple instructions



Simple Program

Write a program that calculates 19 + 95+ 5?

Some simple instructions



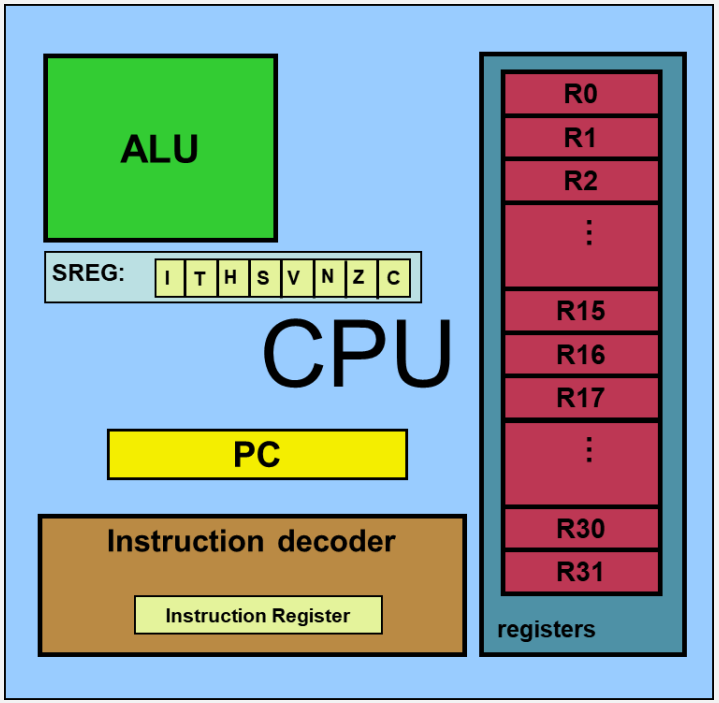
Simple Program

Write a program that calculates 19 + 95+ 5?

Answer 1

```
LDI     R16, 19     ;R16 = 19
LDI     R20, 95     ;R20 = 95
LDI     R21, 5      ;R21 = 5
ADD     R16, R20    ;R16 = R16 + R20
ADD     R16, R21    ;R16 = R16 + R21
```

Some simple instructions



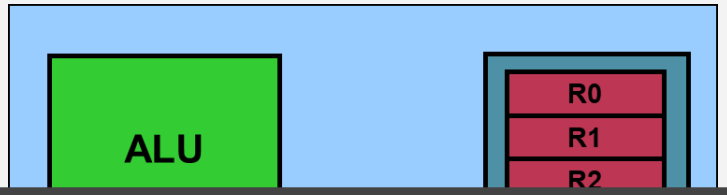
Simple Program

Write a program that calculates 19 + 95+ 5?

Answer 1

```
LDI     R16, 19     ;R16 = 19
LDI     R20, 95     ;R20 = 95
LDI     R21, 5      ;R21 = 5
ADD     R16, R20    ;R16 = R16 + R20
ADD     R16, R21    ;R16 = R16 + R21
```

Some simple instructions



Answer 2

```
LDI     R16, 19     ;R16 = 19
LDI     R20, 95     ;R20 = 95
ADD     R16, R20    ;R16 = R16 + R20
LDI     R20, 5      ;R20 = 5
ADD     R16, R20    ;R16 = R16 + R20
```

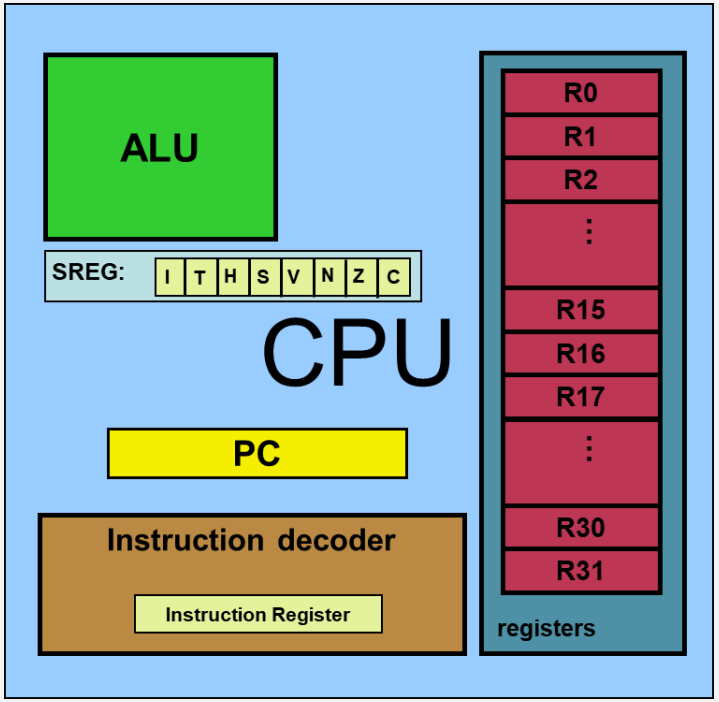
Command Template

- **SUB Rd,Rs**
  - **Rd = Rd - Rs**
- **INC Rd**
  - **Rd = Rd + 1**
- **DEC Rd**
  - **Rd = Rd - 1**

Example

- SUB R17,R30
  - R17 = R17 – R30
- INC R25
  - R25 = R25 + 1
- DEC R23
  - R23 = R23 - 1

Some simple instructions



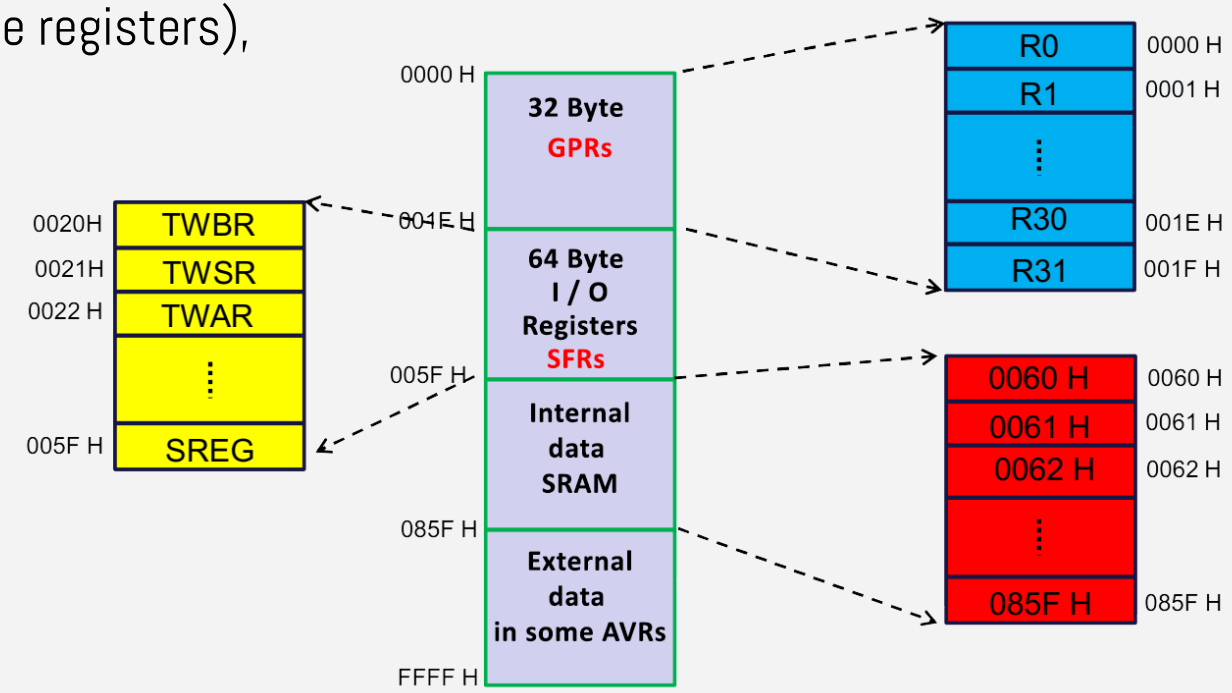
AVR Memory

In AVR microcontrollers:

1. **code memory space**
   - **program is stored here**
2. **data memory space.**
   - **data is stored here**

data memory space

1. GPRs (general purpose registers),
2. I/O memory,
3. internal data SRAM



GPRs

- GPRs (general purpose registers):
  - GPRs use 32 bytes of data memory space.
  - take the address location $00–$1F in the data memory space, regardless of the AVR chip number.

SFRs (special function registers)

- I/O memory:
  - dedicated to specific functions such as status register, timers, serial communication, I/O ports, ADC, and so on.
  - The function of each I/O memory location is fixed by the CPU designer at the time of design
  - The AVR I/O memory is made of 8-bit registers.
  - The first 64-byte section is called standard I/O memory.
  - In AVRs with more than 32 I/O pins (e.g., ATmega64, ATmega128, and ATmega256) there is also an extended I/O memory, called SFRs (special function registers)

GPRs

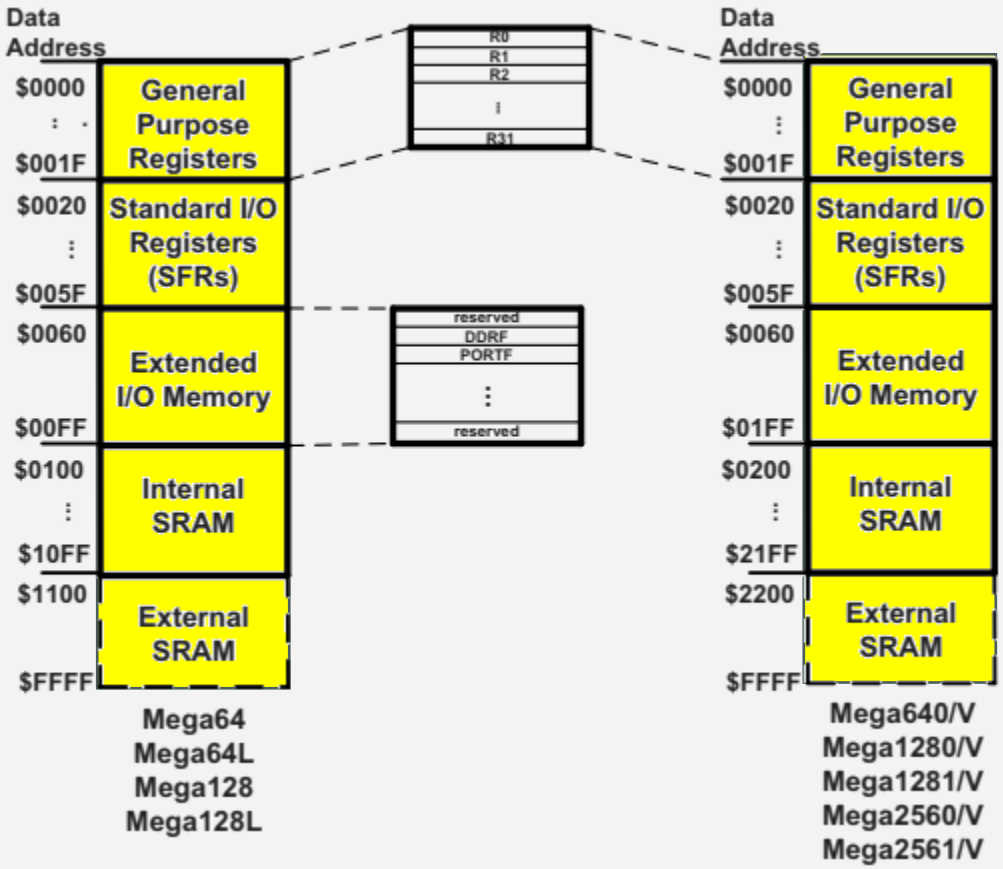
- GPRs (general purpose registers):
  - GPRs use 32 bytes of data memory space.
  - take the address location $00–$1F in the data memory space, regardless of the AVR chip number.

SFRs (special function registers)

- I/O memory:
  - dedicated to specific functions such as status register, timers, serial communication, I/O ports, ADC, and so on.
  - The function of each I/O memory location is fixed by the CPU designer at the time of design
  - The AVR I/O memory is made of 8-bit registers.
  - The first 64-byte section is called standard I/O memory.
  - In AVRs with more than 32 I/O pins (e.g., ATmega64, ATmega128, and ATmega256) there is also an extended I/O memory, called SFRs (special function registers)

GPRs

SFRs (special function registers)



function registers)

SRAM

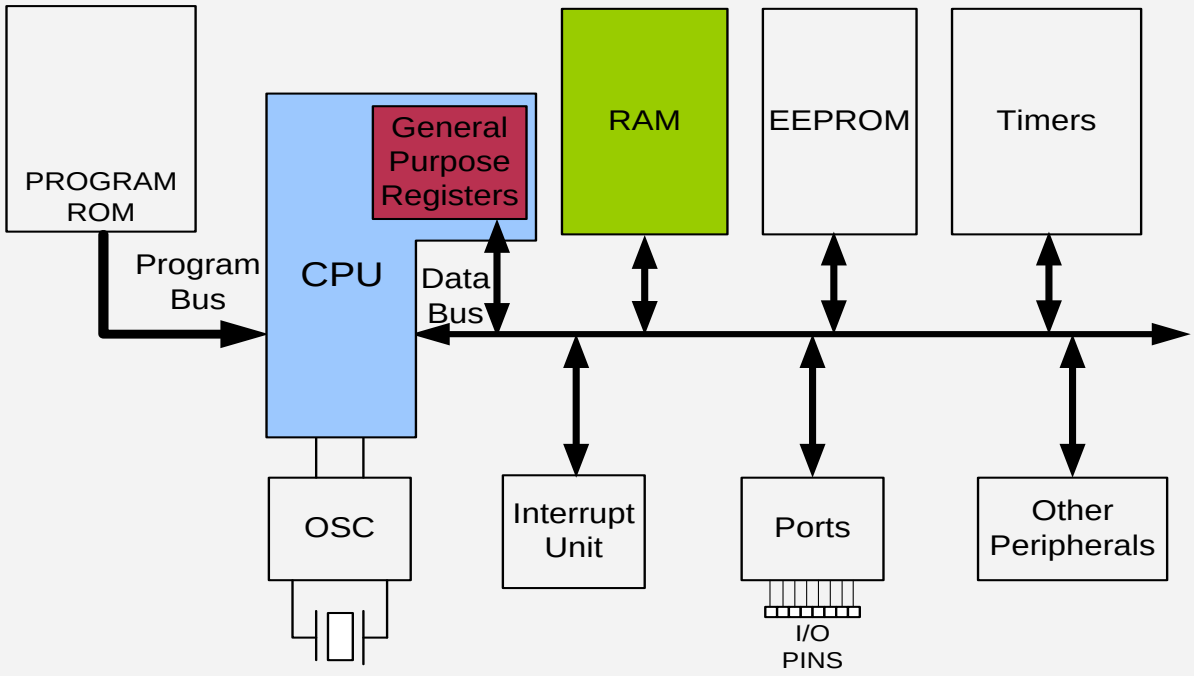
- internal data SRAM :
  - widely used for storing data and parameters by AVR programmers and C compilers.
  - Each location of the SRAM can be accessed directly by its address.
  - Each location is 8 bits wide and can be used to store any data
  - The size of SRAM can vary from chip to chip, even among members of the same family.

Memory in AVRs

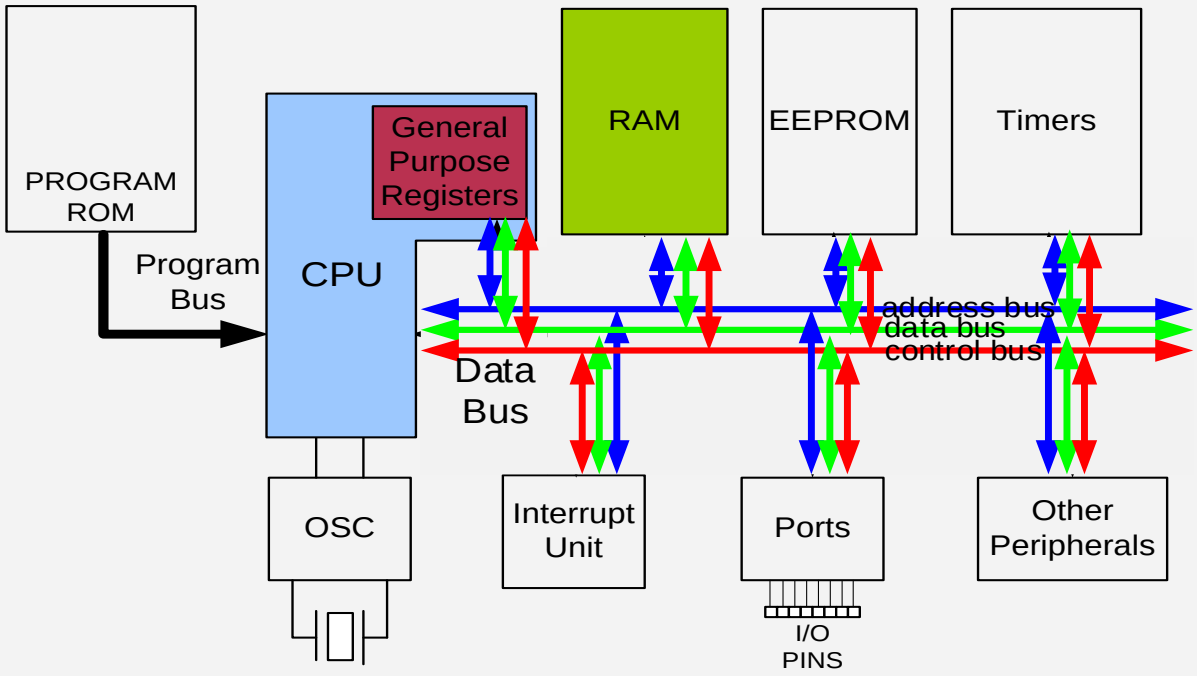
	Data Memory (Bytes)	=	I/O Registers (Bytes)	+	SRAM (Bytes)	+	General Purpose Register
ATtiny25	224		64		128		32
ATtiny85	608		64		512		32
ATmega8	1120		64		1024		32
ATmega16	1120		64		1024		32
ATmega32	2144		64		2048		32
ATmega128	4352		64+160		4096		32
ATmega2560	8704		64+416		8192		32

Extracted from http://www.atmel.com

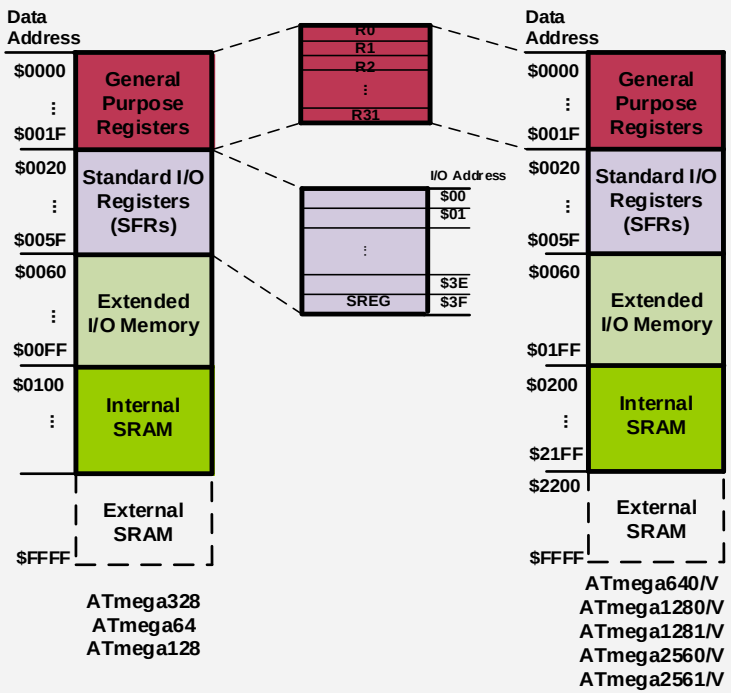
Data Address Space



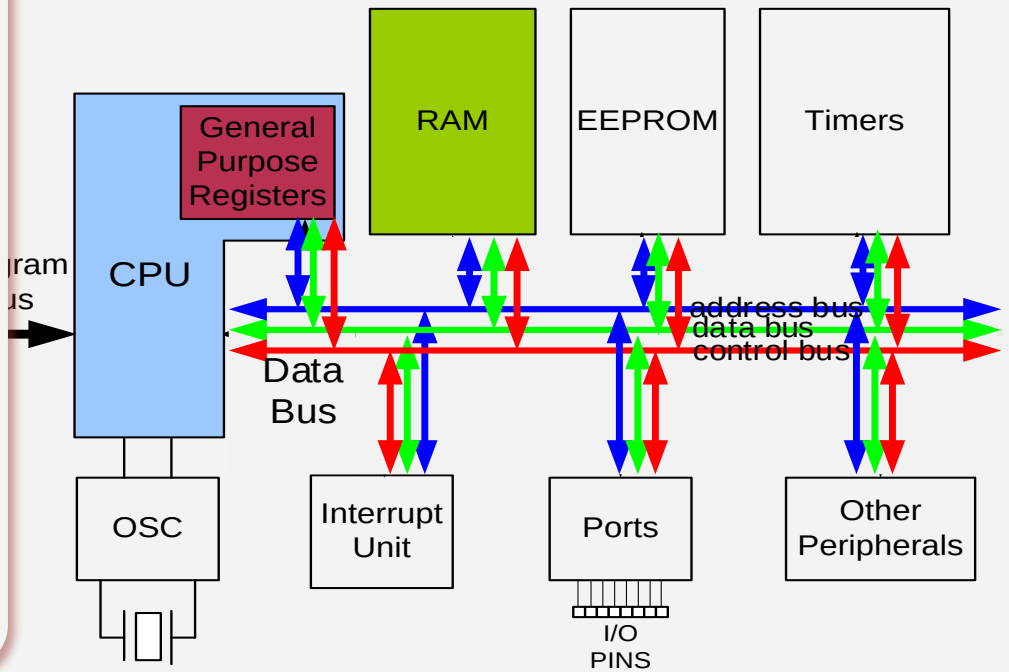
Data Address Space



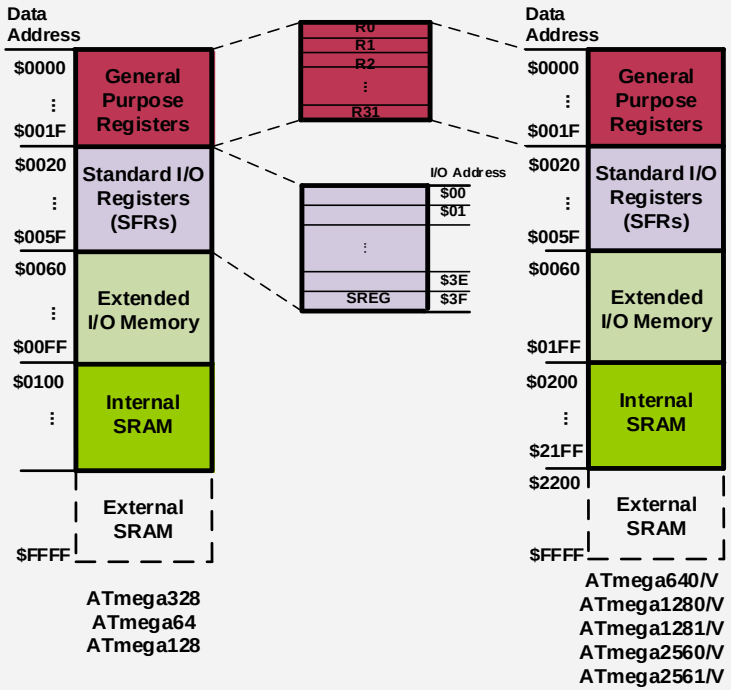
The Data Memory for the AVRs



Data Address Space



The Data Memory for the AVRs



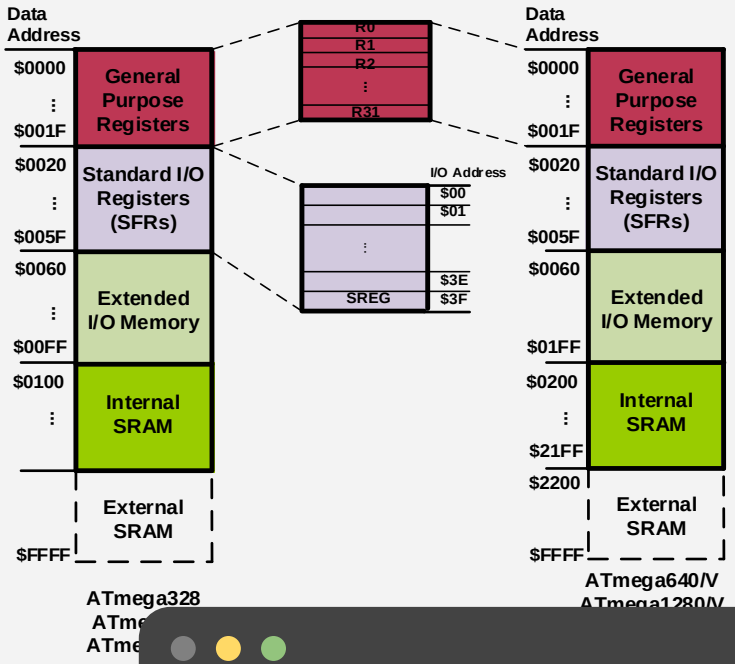
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

Example

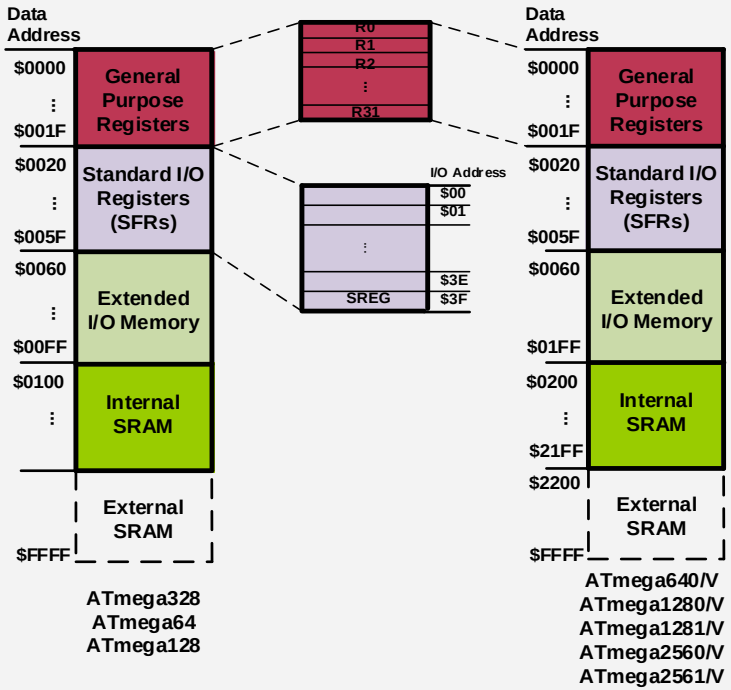
LDS (Load direct from data space)

LDS Rd, addr ;Rd = [addr]

Example:

```
LDS R1, 0x60
```

The Data Memory for the AVRs



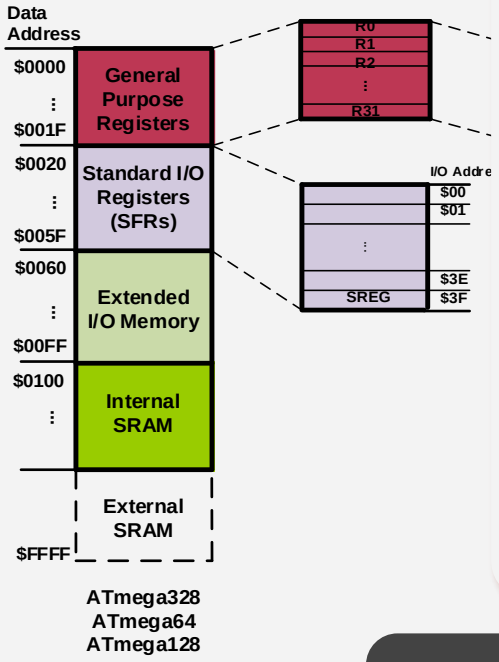
Data Address Space

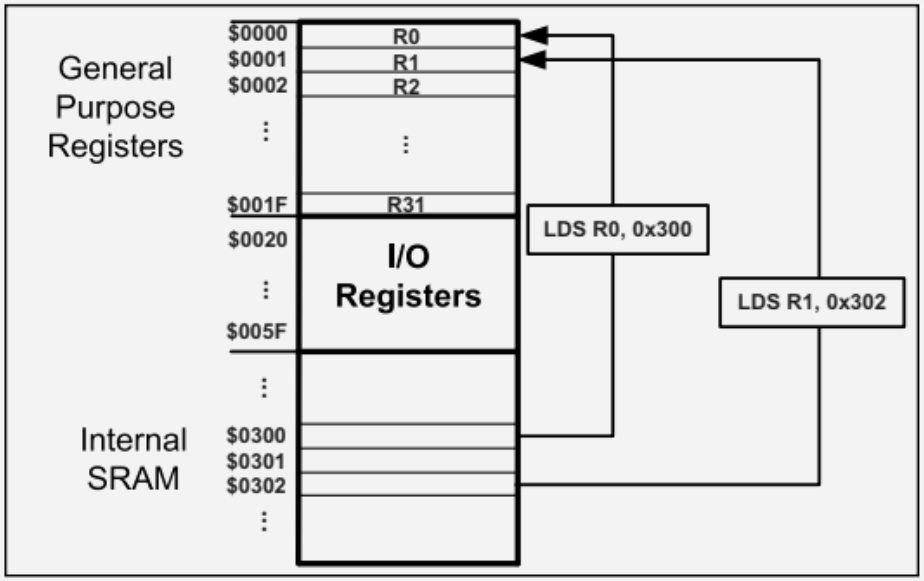
Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Mei … Address Space





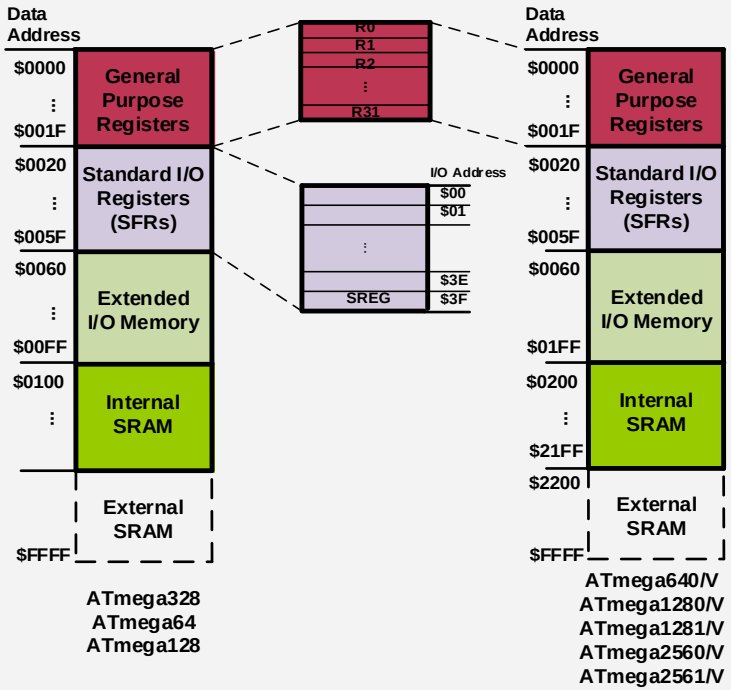
Address Mem.	Address I/O	Name
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

ATmega1280/V
ATmega1281/V

	$33	$13	-	$49	$29	-

	R0	R1	Loc $300	Loc $302
Before LDS R0,0x300	?	?	α	β
After LDS R0,0x300	α	?	α	β
After LDS R1,0x302	α	β	α	β
After ADD R0, R1	α + β	β	α	β

The Data Memory for the AVRs



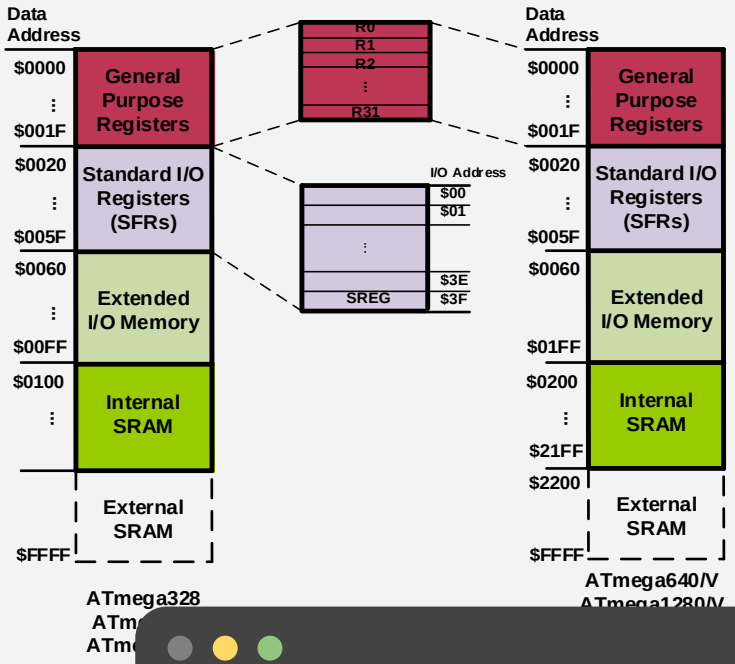
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

Example

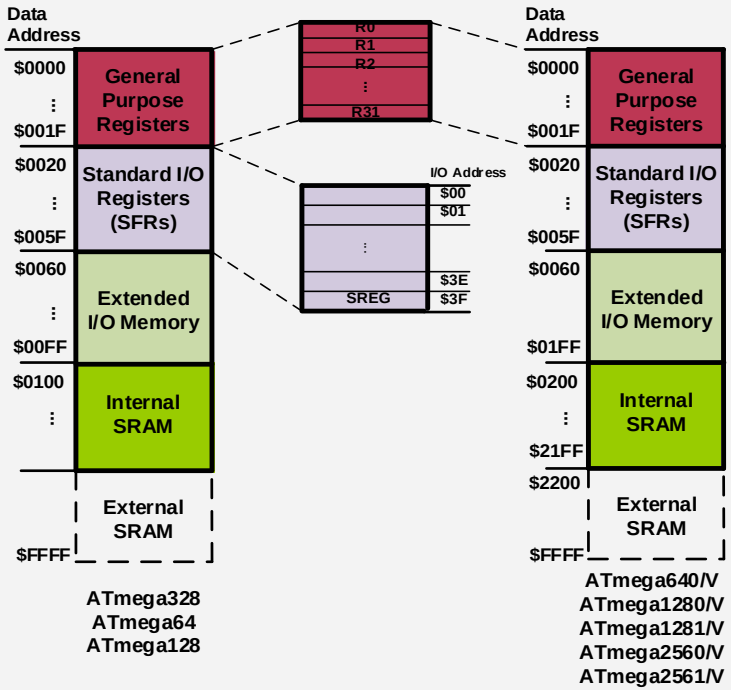
STS (Store direct to data space)

STS addr,Rd ;[addr]=Rd

Example:

```
STS 0x60,R15 ; [0x60] = R15
```

The Data Memory for the AVRs



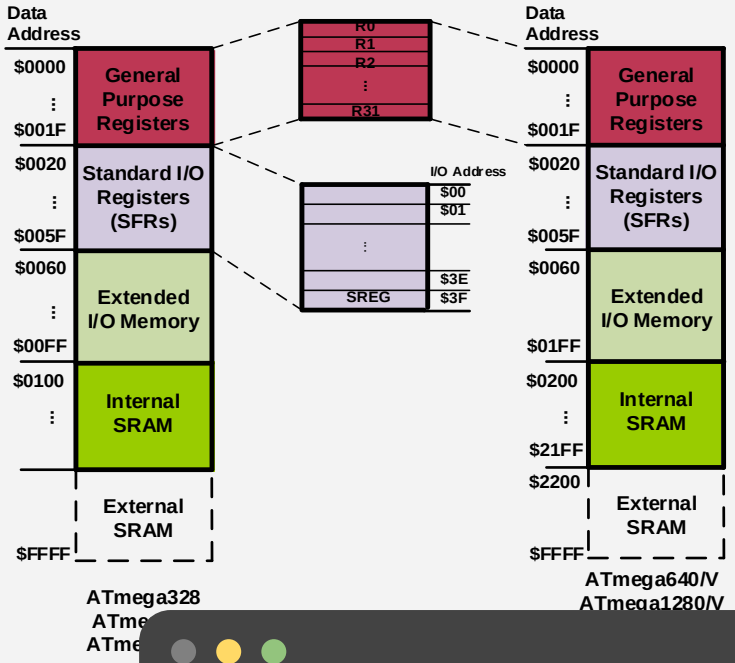
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-

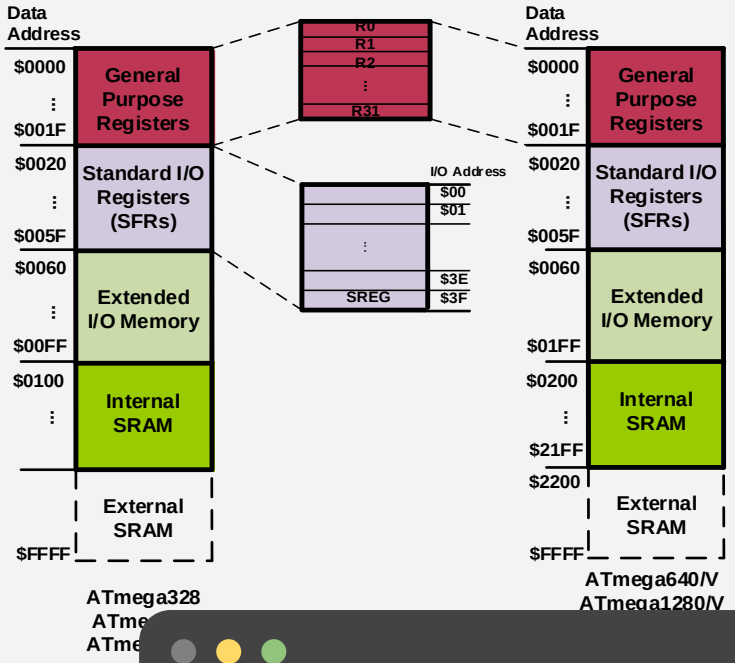
Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

Example

Write a program that stores 55 into location 0x80 of RAM.

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

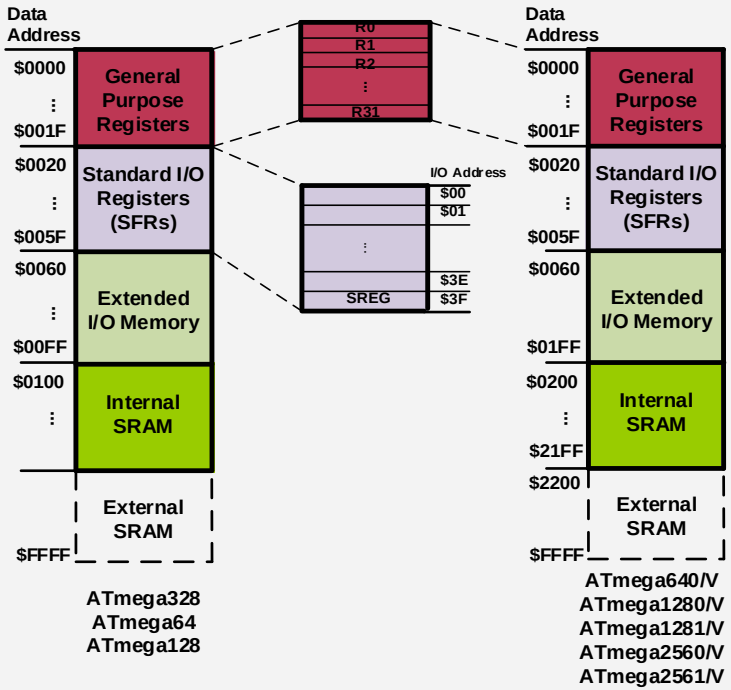
Example

Write a program that stores 55 into location 0x80 of RAM.

Solution:

```
LDI   R20, 55     ;R20 = 55
STS   0x80, R20   ;[0x80] = R20 = 55
```

The Data Memory for the AVRs



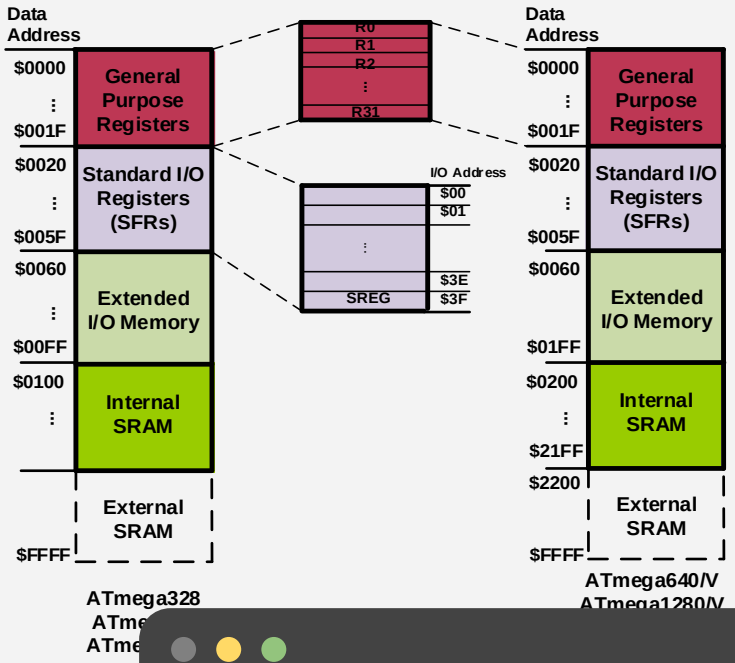
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-

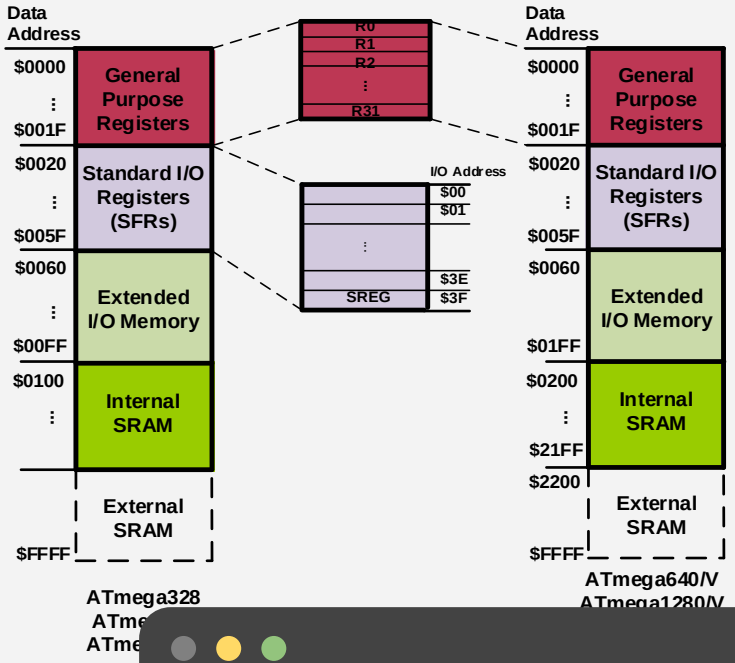
Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

Example

Write a program that copies the contents of location 0x80 of RAM into location 0x81.

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
		SREG

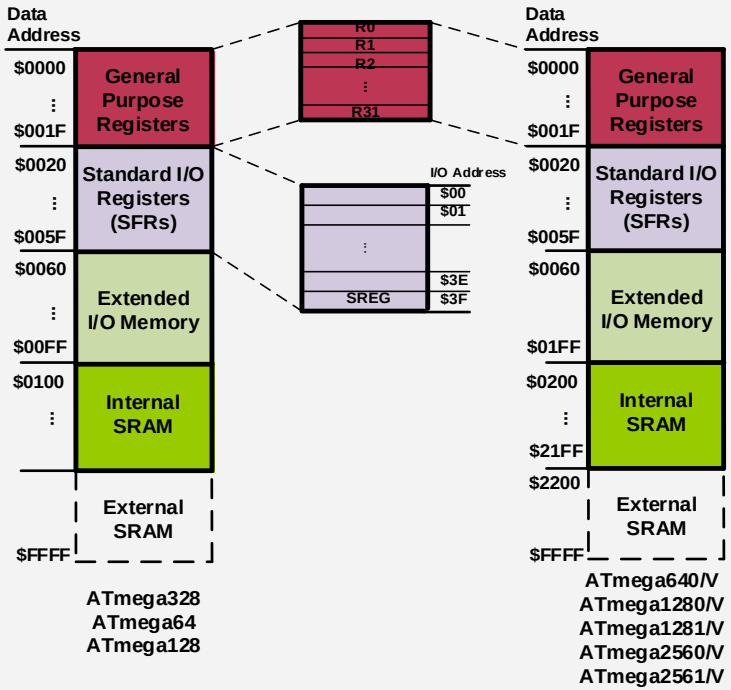
Example

Write a program that copies the contents of location 0x80 of RAM into location 0x81.

Solution:

```
LDS   R20, 0x80          ;R20 = [0x80]

STS   0x81, R20          ;[0x81] = R20 = [0x80]
```

The Data Memory for the AVRs



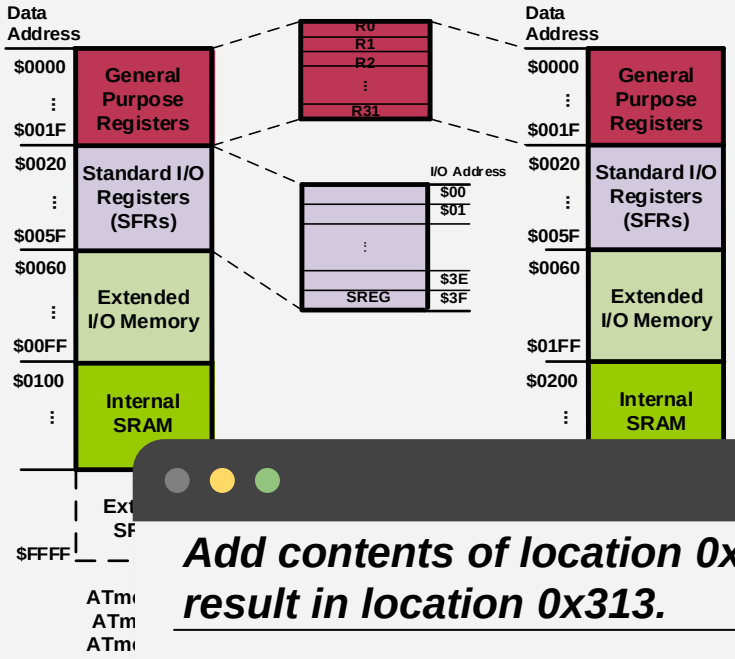
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-

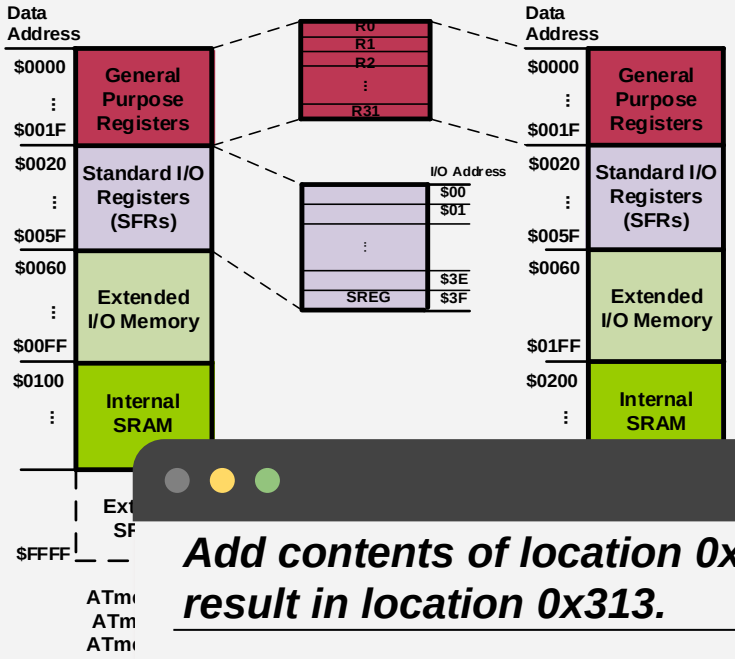
Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
		-
		-
		-
		-
		SPL
		SPH
		SREG

Example

Add contents of location 0x90 to contents of location 0x95 and store the result in location 0x313.

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
		-
		-
		-
		-
		SPL
		SPH
		SREG

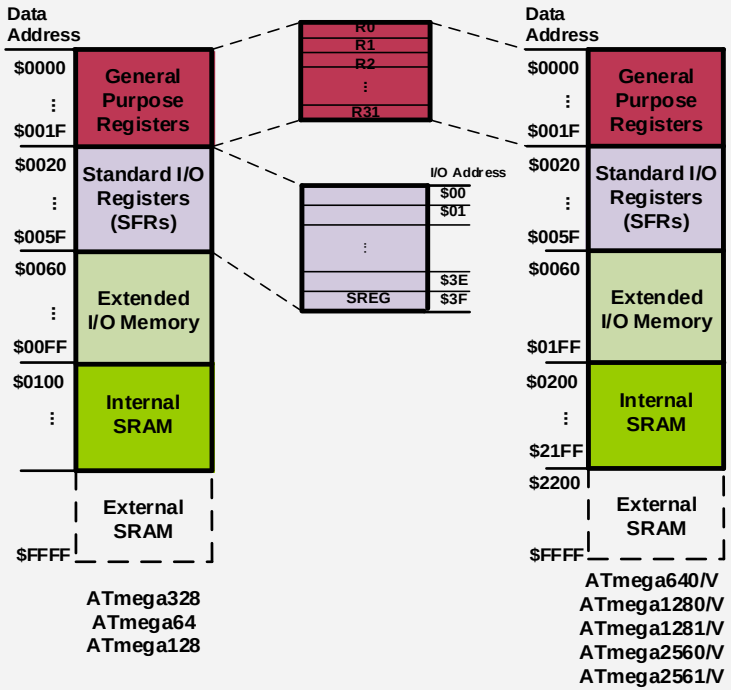
Example

Add contents of location 0x90 to contents of location 0x95 and store the result in location 0x313.

Solution:

```
LDS   R20, 0x90        ;R20 = [0x90]
LDS   R21, 0x95        ;R21 = [0x95]
ADD   R20, R21         ;R20 = R20 + R21
STS   0x313, R20       ;[0x313] = R20
```

The Data Memory for the AVRs



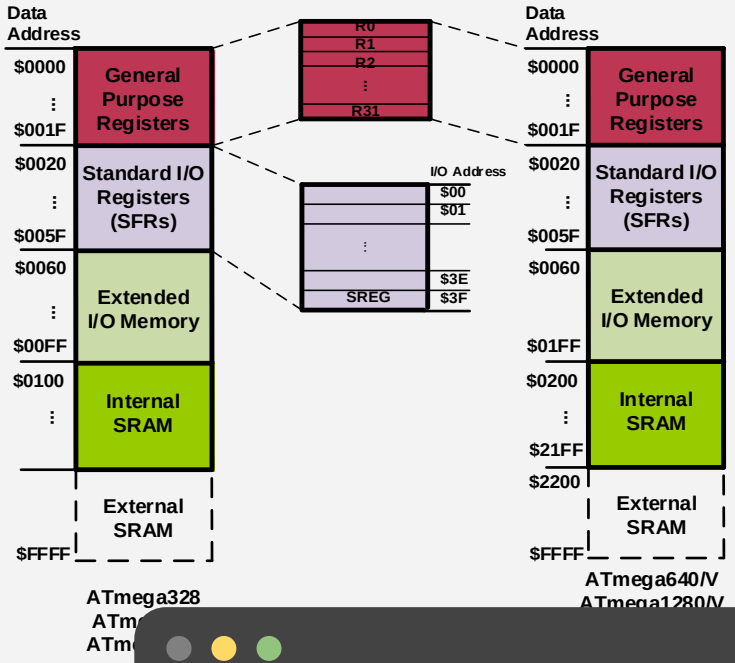
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-

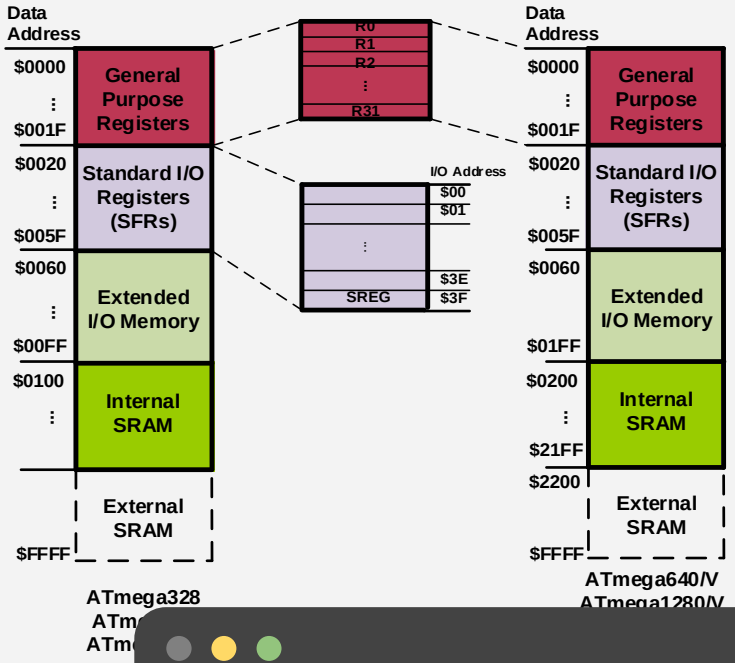
Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

Example

What does the following instruction do? `LDS R20,2`

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

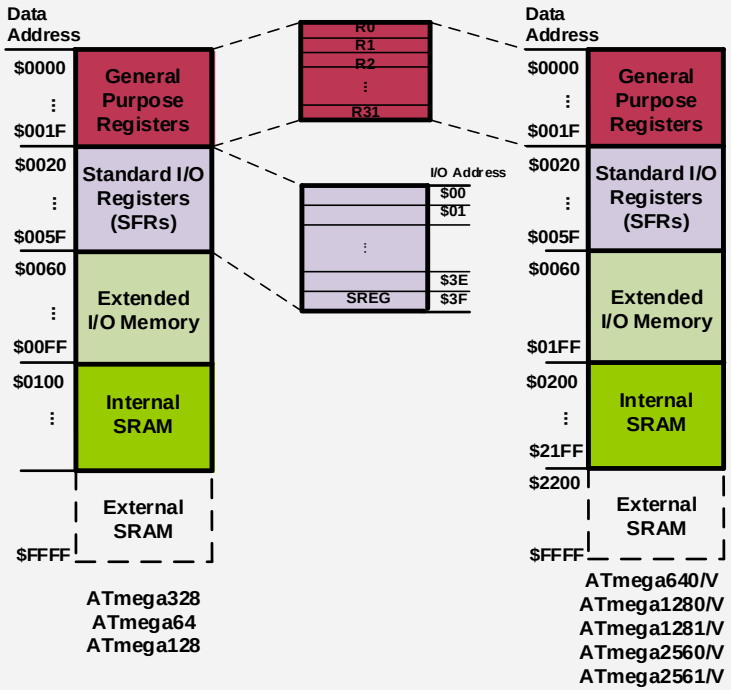
Example

What does the following instruction do? `LDS R20,2`

Answer:

It copies the contents of R2 into R20; as 2 is the address of R2.

The Data Memory for the AVRs



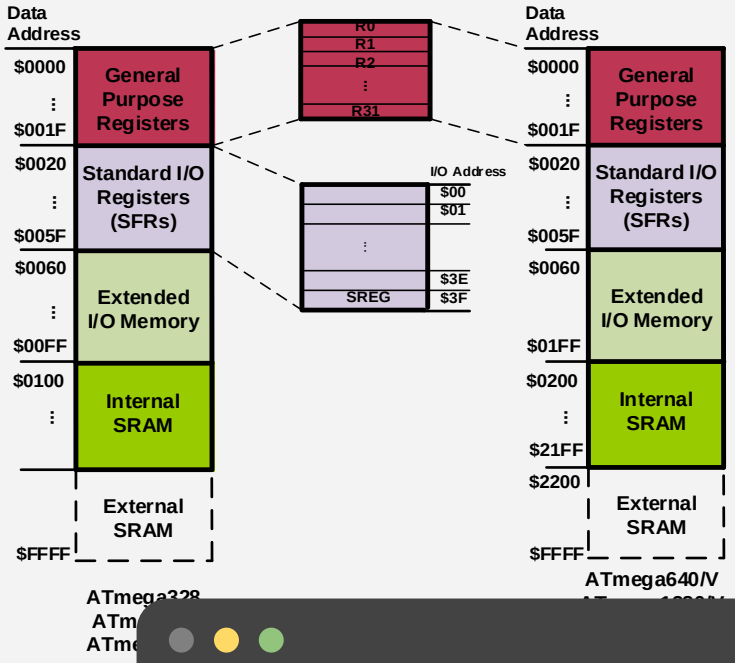
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-

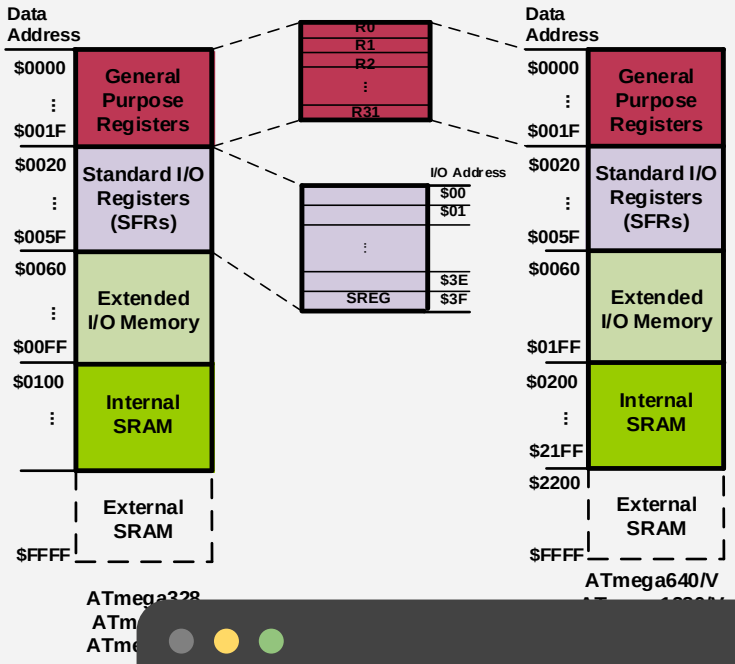
Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
		SREG

Example

Store 0x53 into the SPH register. The address of SPH is 0x5E

The Data Memory for the AVRs



Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
		SREG

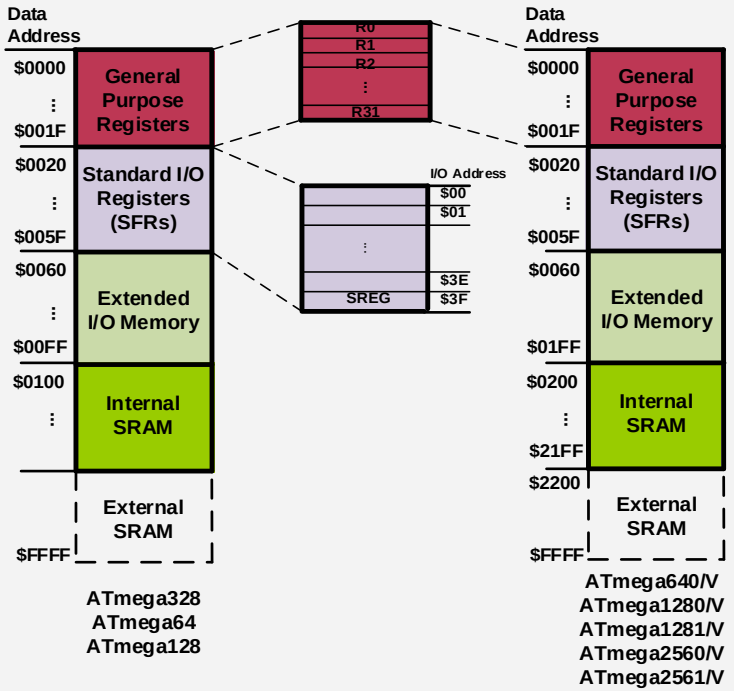
Example

Store 0x53 into the SPH register. The address of SPH is 0x5E

Solution:

```
LDI    R20, 0x53          ;R20 = 0x53
STS    0x5E, R20          ;SPH = R20
```

The Data Memory for the AVRs



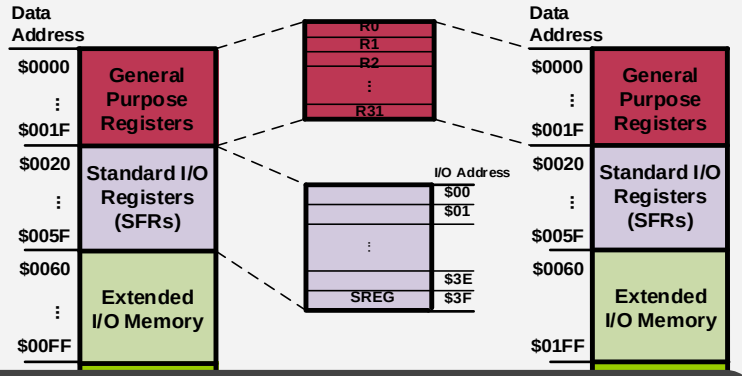
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Command Template

IN (IN from IO location)

```
IN  Rd,IOaddress        ;Rd = [addr]
```

Example:

```
IN  R1, 0x3F            ;R1 = SREG
IN  R17,0x3E            ;R17 = SPH
```

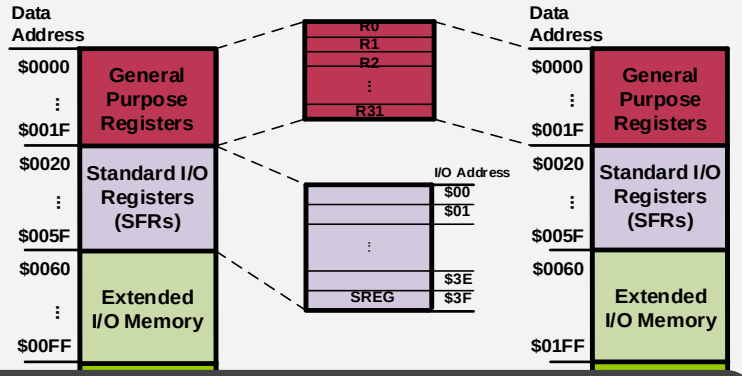
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Command Template

IN (IN from IO location)

```
IN  Rd,IOaddress        ;Rd = [addr]
```

Example:

```
IN  R1, 0x3F            ;R1 = SREG
IN  R17,0x3E            ;R17 = SPH
```

OUT (OUT to I/O location)

```
OUT IOaddress,Rd        ;[addr]=Rd
```

Example:

```
OUT  0x3F,R12           ;SREG = R12
OUT  0x3E,R15           ;SPH = R15
```

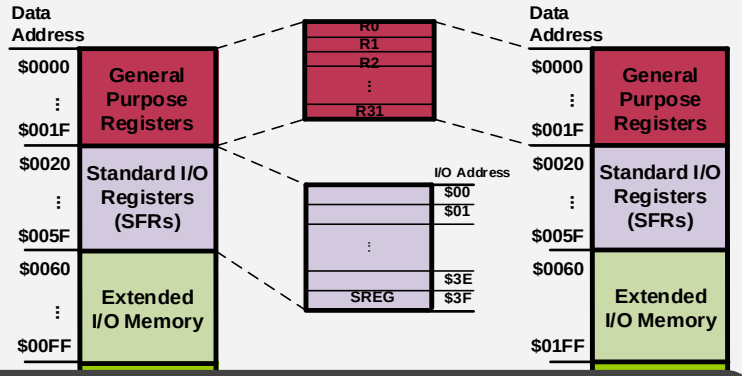
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Command Template

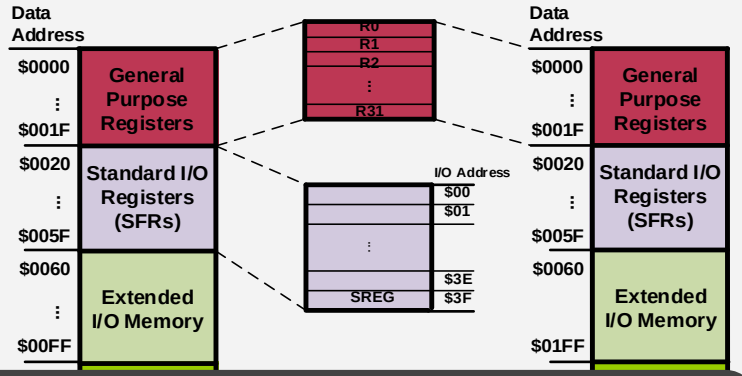
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Command Template

Using Names of I/O registers

Example:

```
OUT   SPH,R12      ;OUT   0x3E,R12
IN    R15,SREG     ;IN    R15,0x3F
```

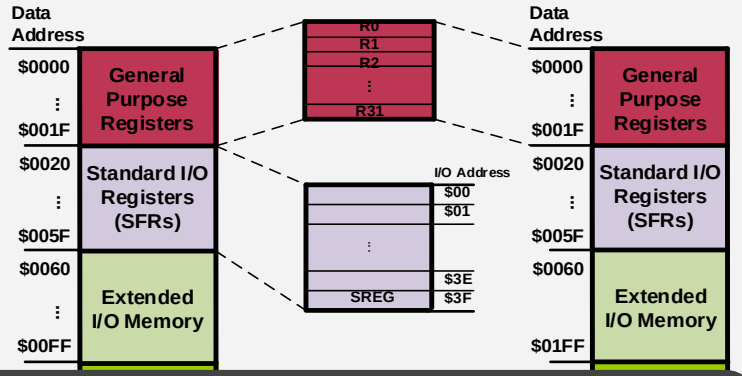
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Command Template

Example: Write a program that adds the contents of the PINC I/O register to the contents of PIND and stores the result in location 0x90 of the SRAM

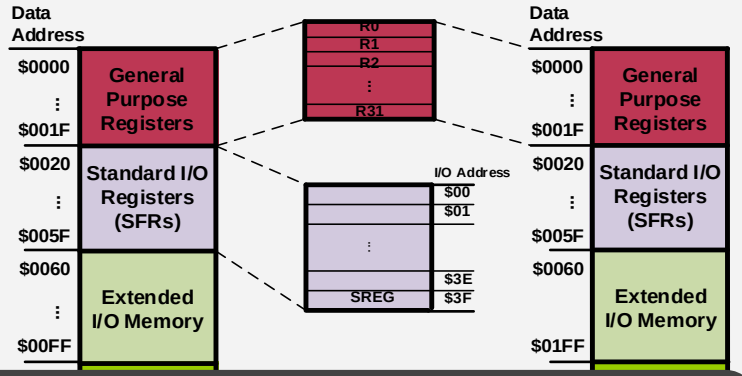
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

The Data Memory for the AVRs



Command Template

Example: Write a program that adds the contents of the PINC I/O register to the contents of PIND and stores the result in location 0x90 of the SRAM

Solution:

```
IN    R20,PINC   ;R20 = PINC
 IN   R21,PIND   ;R21 = PIND
ADD   R20,R21    ;R20 = R20 + R21
STS   0x90,R20   ;[0x90] = R20
```

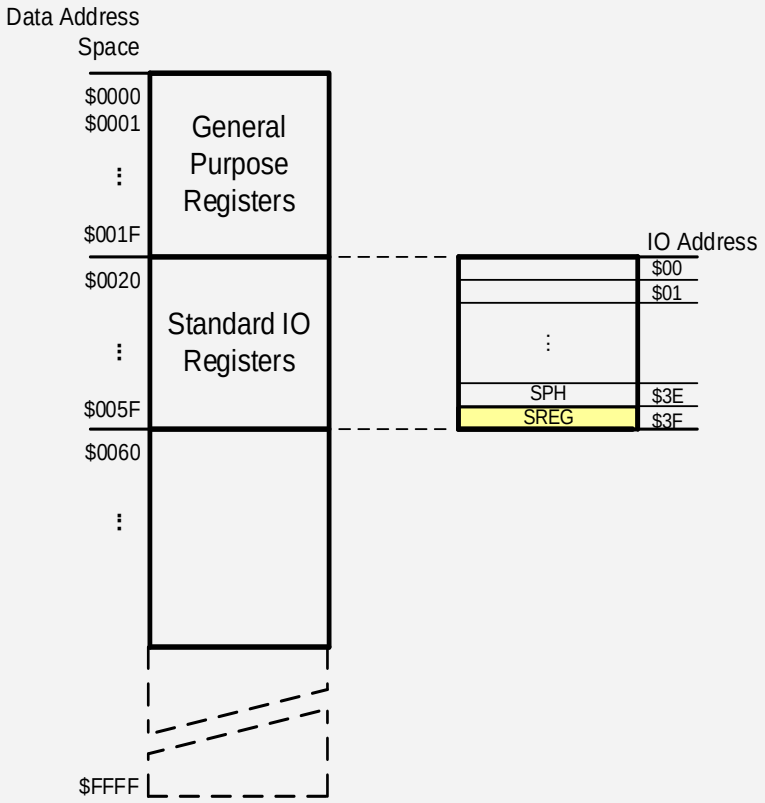
Data Address Space

Address		Name
Mem.	I/O	
$20	$00	-
$21	$01	-
$22	$02	-
$23	$03	PINB
$24	$04	DDRB
$25	$05	PORTB
$26	$06	PINC
$27	$07	DDRC
$28	$08	PORTC
$29	$09	PIND
$2A	$0A	DDRD
$2B	$0B	PORTD
$2C	$0C	-
$2D	$0D	-
$2E	$0E	-
$2F	$0F	-
$30	$10	-
$31	$11	-
$32	$12	-
$33	$13	-
$34	$14	-
$35	$15	TIFR0

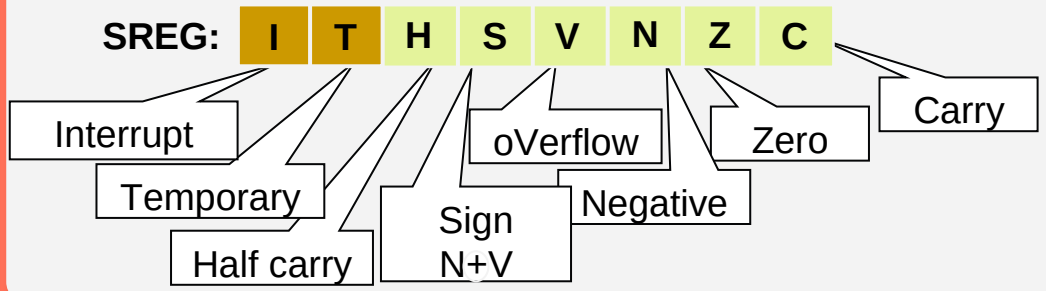
Address		Name
Mem.	I/O	
$36	$16	TIFR1
$37	$17	TIFR2
$38	$18	-
$39	$19	-
$3A	$1A	-
$3B	$1B	PCIFR
$3C	$1C	EIFR
$3D	$1D	EIMSK
$3E	$1E	GPIOR0
$3F	$1F	EECR
$40	$20	EEDR
$41	$21	EEARL
$42	$22	EEARH
$43	$23	GTCCR
$44	$24	TCCR0A
$45	$25	TCCR0B
$46	$26	TCNT0
$47	$27	OCR0A
$48	$28	OCR0B
$49	$29	-
$4A	$2A	GPIOR1
$4A	$2A	GPIOR2

Address		Name
Mem.	I/O	
$4C	$2C	SPCR0
$4D	$2D	SPSR0
$4E	$2E	SPDR0
$4F	$2F	-
$50	$30	ACSR
$51	$31	DWDR
$52	$32	-
$53	$33	SMCR
$54	$34	MCUSR
$55	$35	MCUCR
$56	$36	-
$57	$37	SPMCSR
$58	$38	-
$59	$39	-
$5A	$3A	-
$5B	$3B	-
$5C	$3C	-
$5D	$3D	SPL
$5E	$3E	SPH
$5F	$3F	SREG

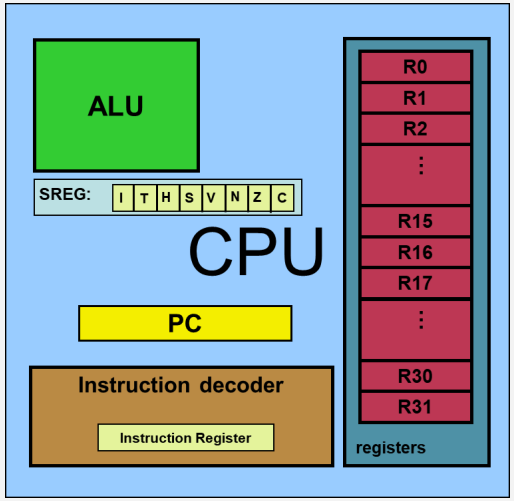
Status Register (SREG)



Status Register (SREG)



AVR's CPU



Example

Example: Show the status of the C, H, and Z flags after the addition of 0x38 and 0x2F in the following instructions:

```
LDI   R16, 0x38      ;R16 = 0x38
LDI   R17, 0x2F      ;R17 = 0x2F
ADD   R16, R17       ;add R17 to R16
```

Example

Example: Show the status of the C, H, and Z flags after the addition of 0x38 and 0x2F in the following instructions:

```
LDI   R16, 0x38      ;R16 = 0x38
LDI   R17, 0x2F      ;R17 = 0x2F
ADD   R16, R17       ;add R17 to R16
```

Solution:

	1	
	$38	0011 1000
+	$2F	0010 1111
	$67	0110 0111

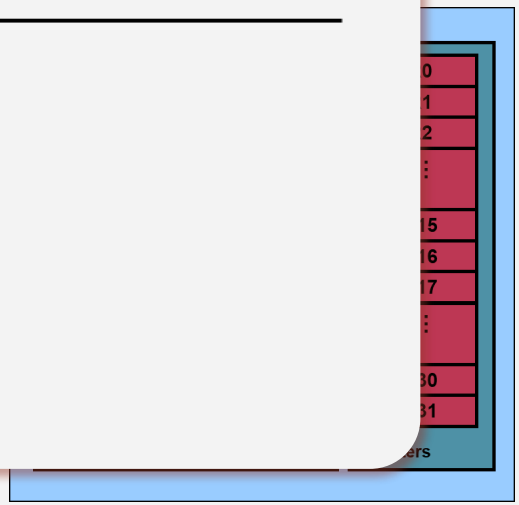
R16 = 0x67

C = 0 because there is no carry beyond the D7 bit.
H = 1 because there is a carry from the D3 to the D4 bit.
Z = 0 because the R16 (the result) has a value other than 0 after the addition.

Example

Example: Show the status of the C, H, and Z flags after the addition of 0x9C and 0x64 in the following instructions:

```
LDI     R20, 0x9C
LDI     R21, 0x64
ADD     R20, R21        ;add R21 to R20
```



Example

Example: Show the status of the C, H, and Z flags after the addition of 0x9C and 0x64 in the following instructions:

```
LDI     R20, 0x9C
LDI     R21, 0x64
ADD     R20, R21        ;add R21 to R20
```

Solution:

```
                 1
    $9C    1001 1100
+   $64    0110 0100
  $100   1 0000 0000      R20 = 00
```

C = 1 because there is a carry beyond the D7 bit.
H = 1 because there is a carry from the D3 to the D4 bit.
Z = 1 because the R20 (the result) has a value 0 in it after the addition.

Example

Example: Show the status of the C, H, and Z flags after the subtraction of 0x23 from 0xA5 in the following instructions:

```
LDI     R20, 0xA5
LDI     R21, 0x23
SUB     R20, R21        ;subtract R21 from R20
```

Example

Example: Show the status of the C, H, and Z flags after the subtraction of 0x23 from 0xA5 in the following instructions:

```
LDI     R20, 0xA5
LDI     R21, 0x23
SUB     R20, R21        ;subtract R21 from R20
```

Solution:

```
      $A5     1010 0101
    - $23     0010 0011
      $82     1000 0010      R20 = $82
```

C = 0 because R21 is not bigger than R20 and there is no borrow from D8 bit.
Z = 0 because the R20 has a value other than 0 after the subtraction.
H = 0 because there is no borrow from D4 to D3.

Example

Example: Show the status of the C, H, and Z flags after the subtraction of 0x73 from 0x52 in the following instructions:

```
LDI     R20, 0x52
LDI     R21, 0x73
SUB     R20, R21        ;subtract R21 from R20
```

Example

Example: Show the status of the C, H, and Z flags after the subtraction of 0x73 from 0x52 in the following instructions:

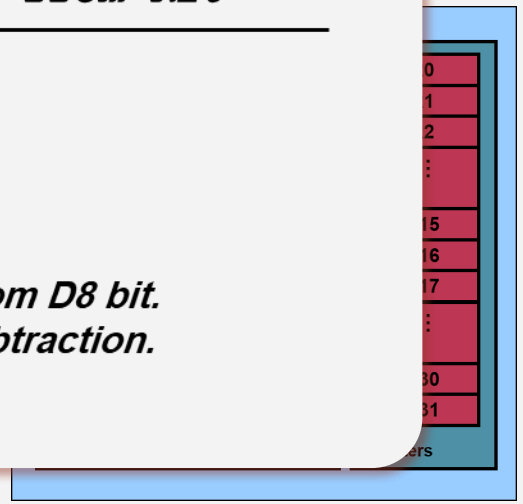
```
LDI     R20, 0x52
LDI     R21, 0x73
SUB     R20, R21        ;subtract R21 from R20
```

Solution:

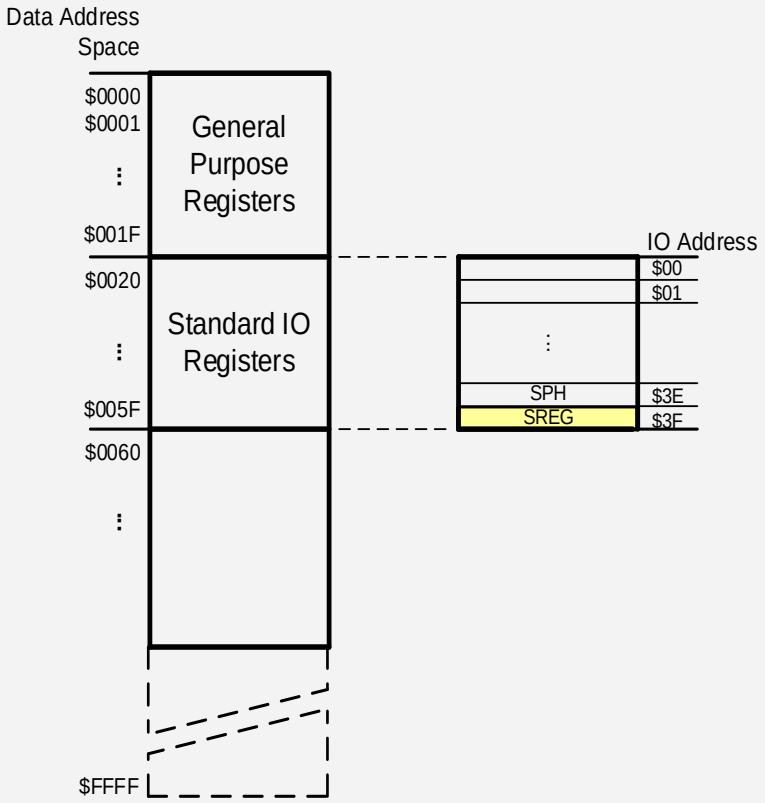
	$52	0101 0010
-	$73	0111 0011
	$DF	1101 1111

R20 = $DF

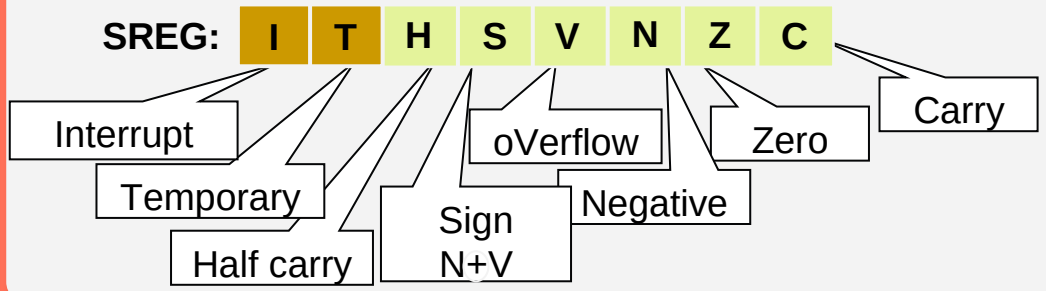
C = 1 because R21 is bigger than R20 and there is a borrow from D8 bit.
Z = 0 because the R20 has a value other than zero after the subtraction.
H = 1 because there is a borrow from D4 to D3.



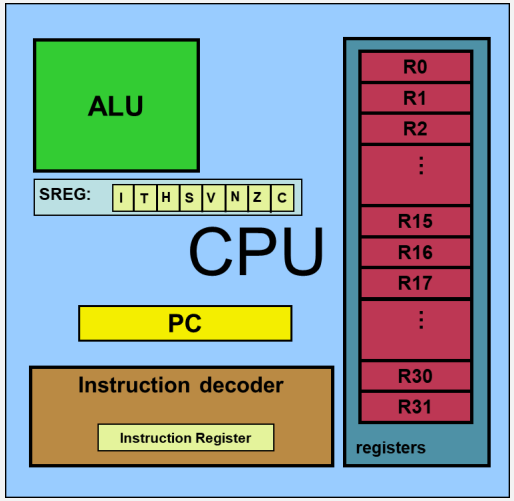
Status Register (SREG)



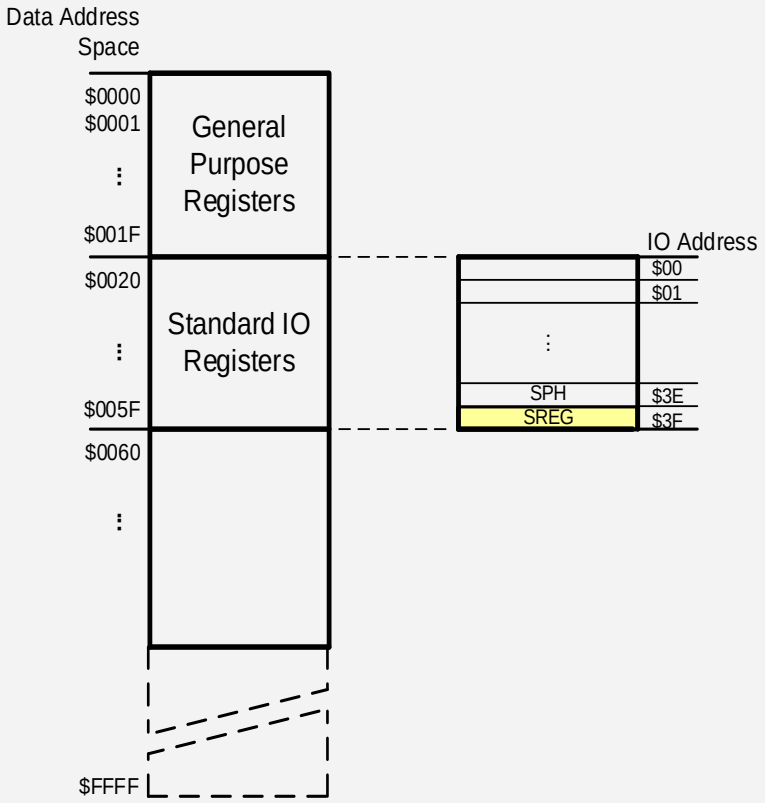
Status Register (SREG)



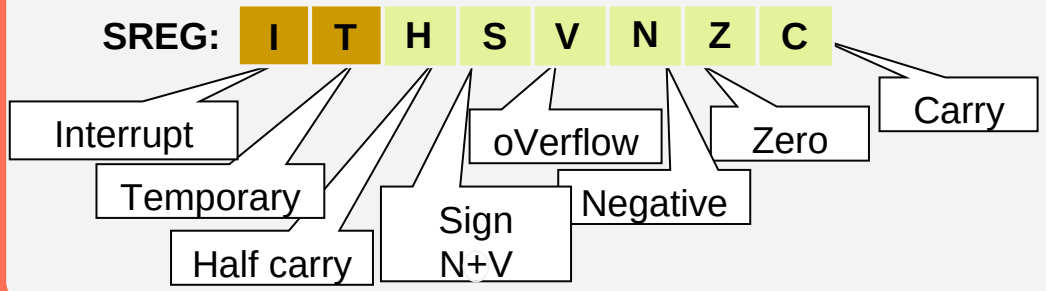
AVR's CPU



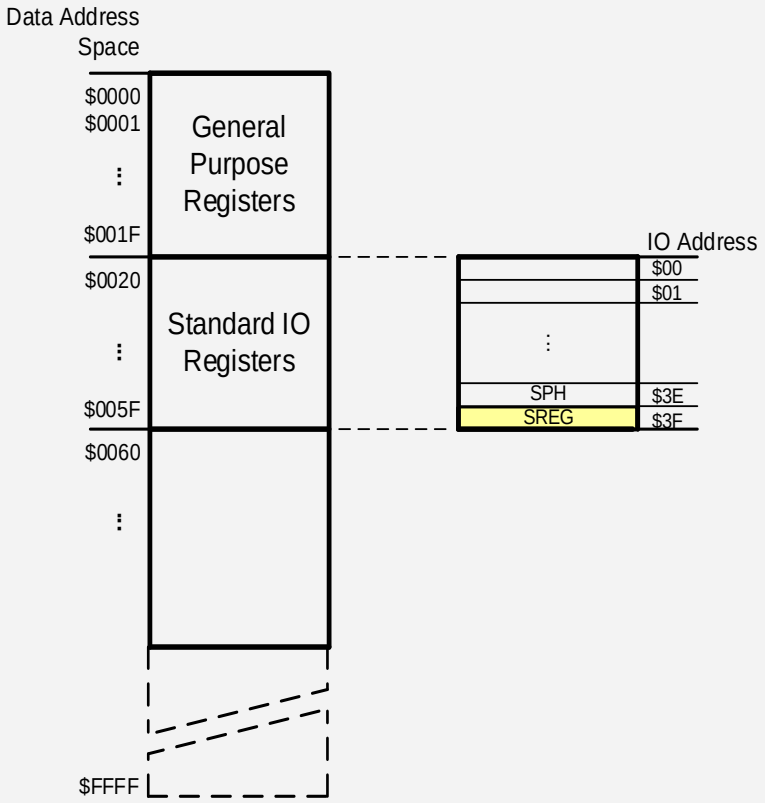
Status Register (SREG)



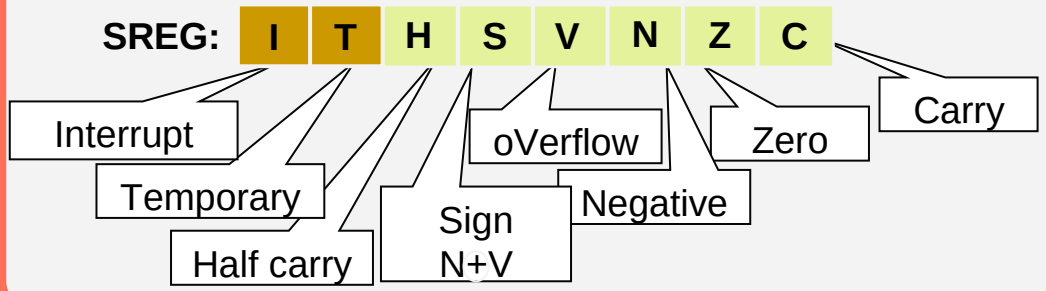
Status Register (SREG)



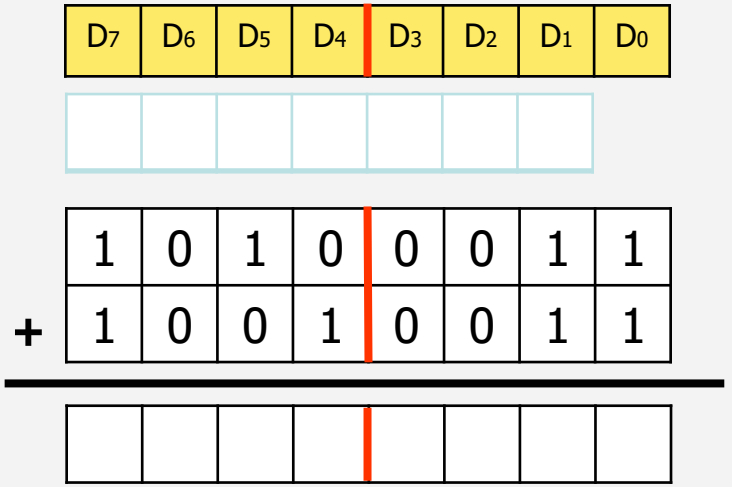
Status Register (SREG)



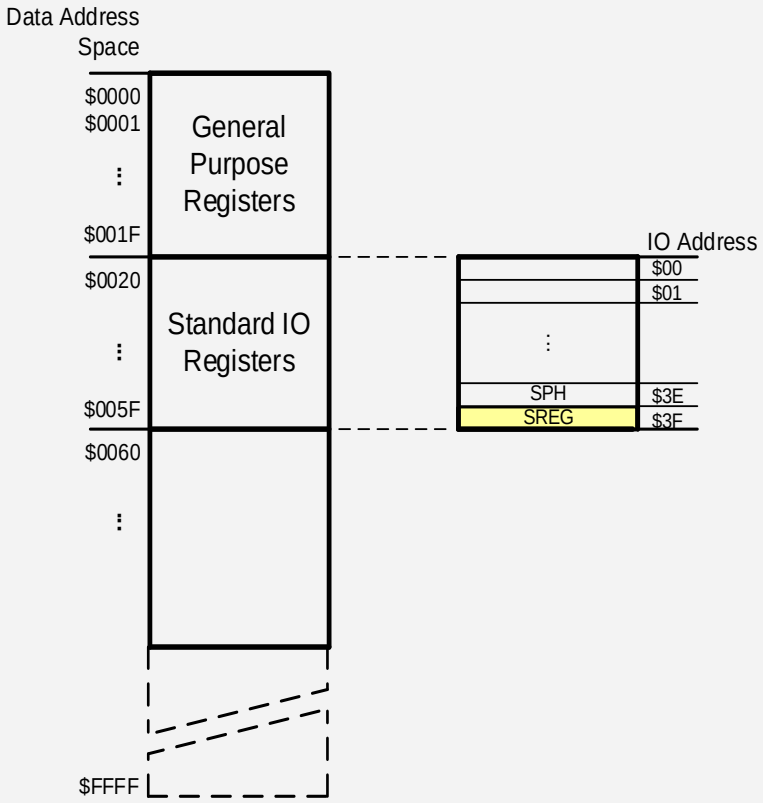
Status Register (SREG)



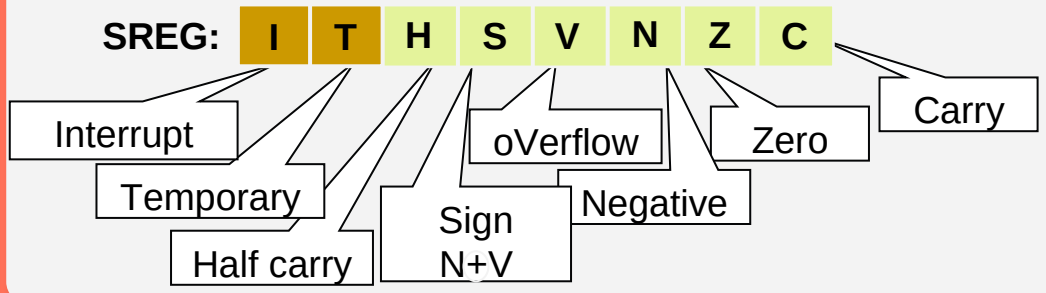
Carry Flag



Status Register (SREG)



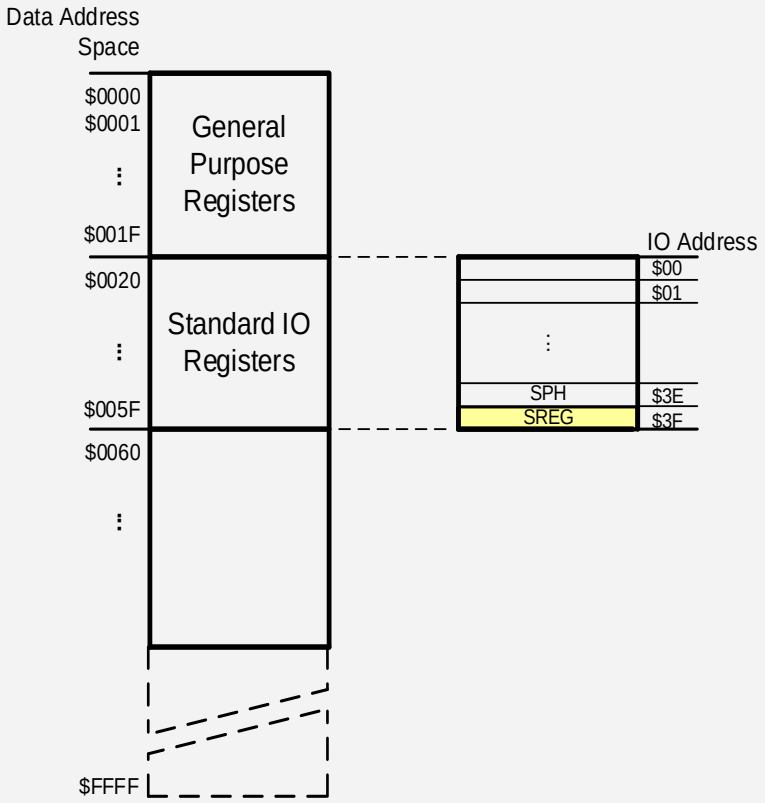
Status Register (SREG)



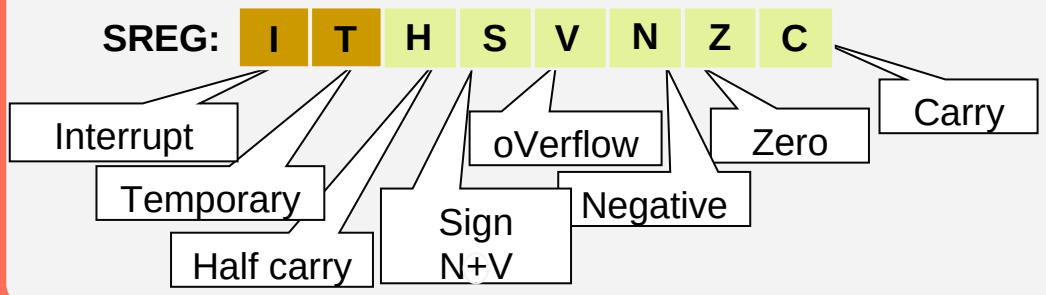
Carry Flag

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
							1	
	1	0	1	0	0	0	1	1
+	1	0	0	1	0	0	1	1
								0

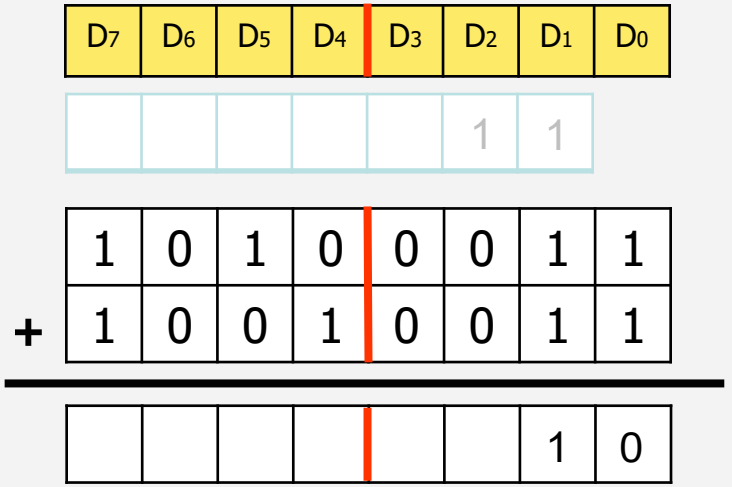
Status Register (SREG)



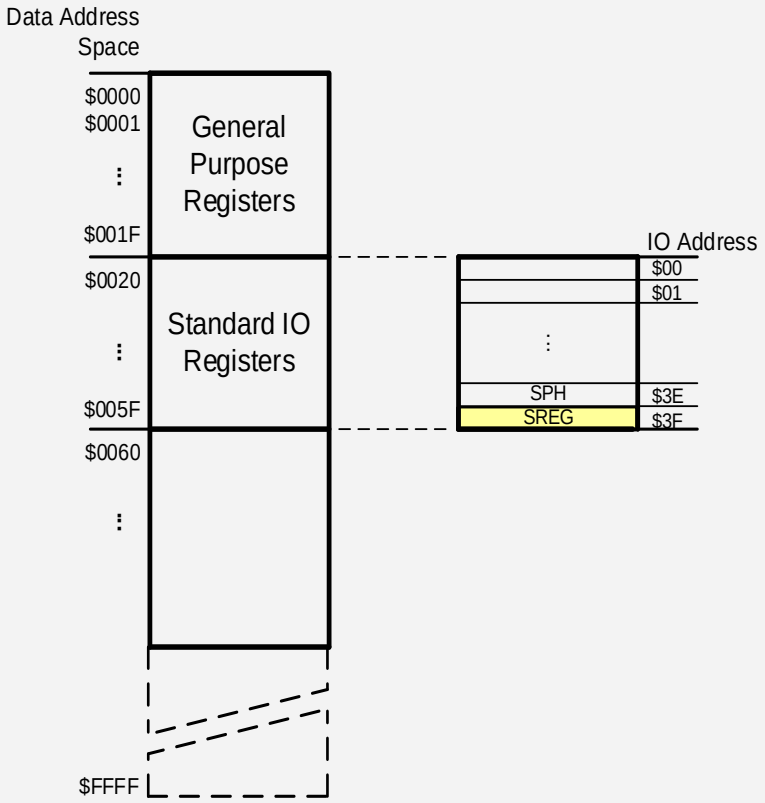
Status Register (SREG)



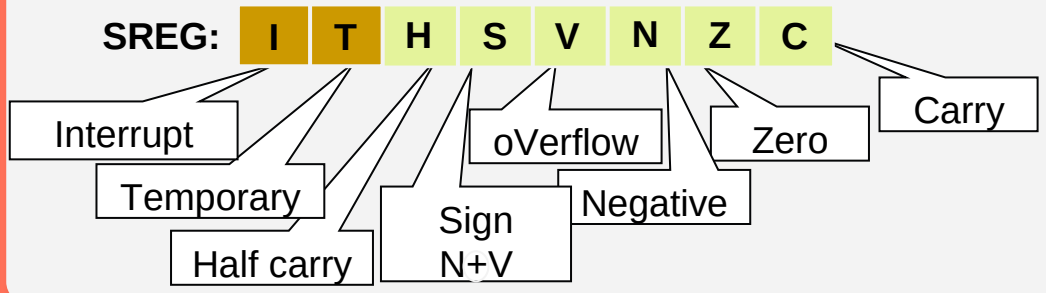
Carry Flag



Status Register (SREG)



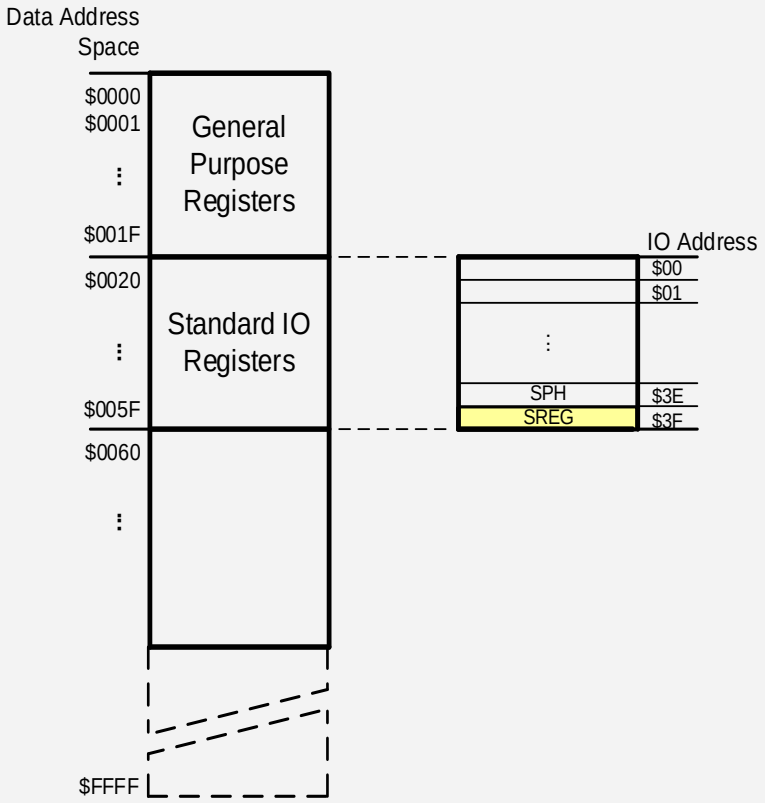
Status Register (SREG)



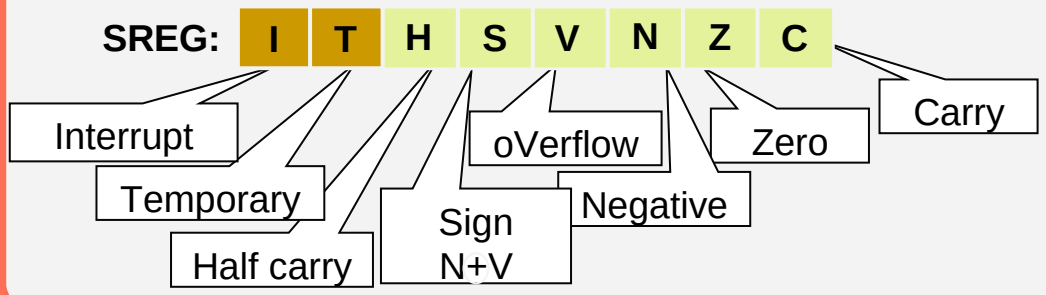
Carry Flag

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
					0	1	1	
	1	0	1	0	0	0	1	1
+	1	0	0	1	0	0	1	1
						1	1	0

Status Register (SREG)



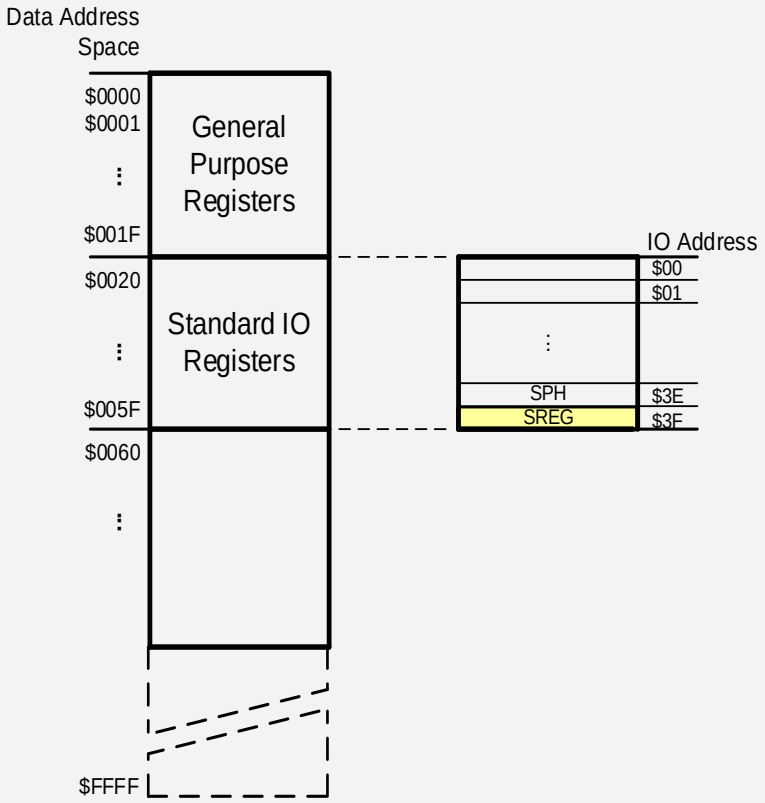
Status Register (SREG)



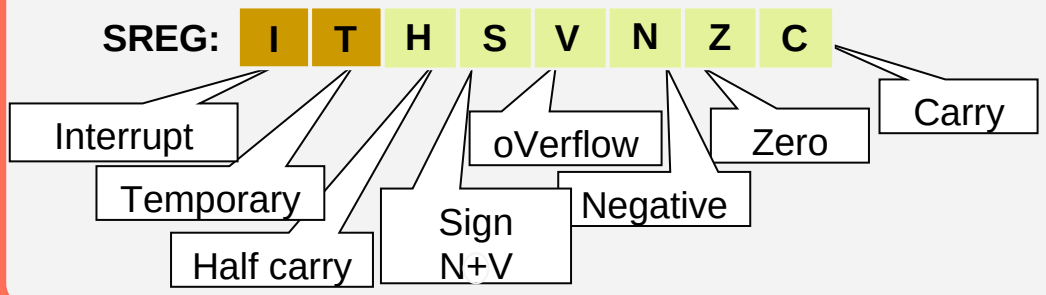
Carry Flag

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
				0	0	1	1	
	1	0	1	0	0	0	1	1
+	1	0	0	1	0	0	1	1
					0	1	1	0

Status Register (SREG)



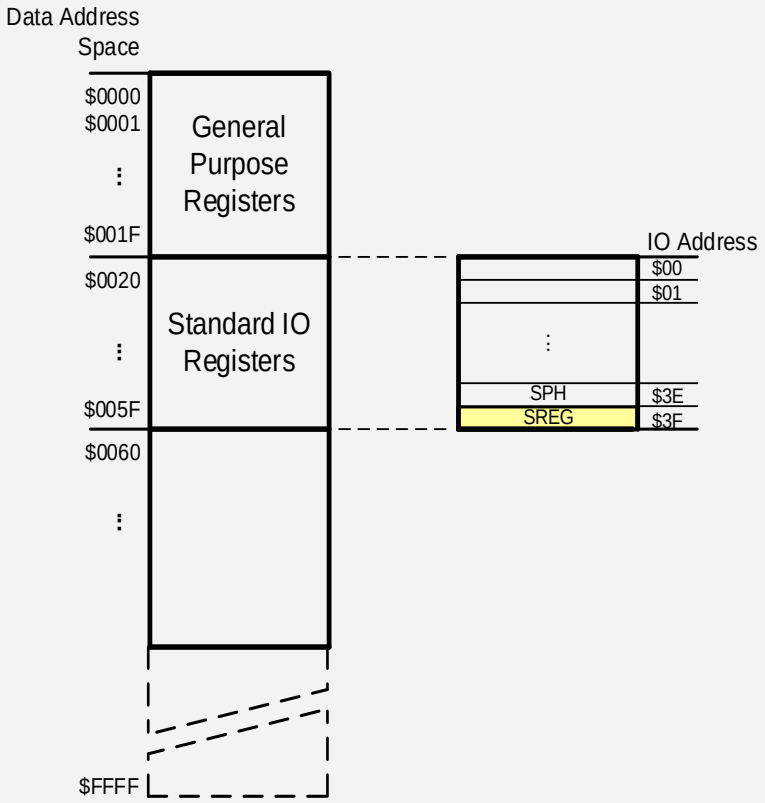
Status Register (SREG)



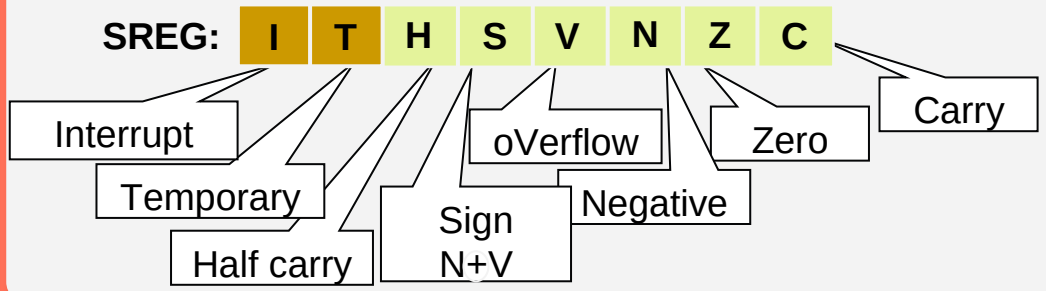
Carry Flag

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
			0	0	0	1	1	
	1	0	1	0	0	0	1	1
+	1	0	0	1	0	0	1	1
				1	0	1	1	0

Status Register (SREG)



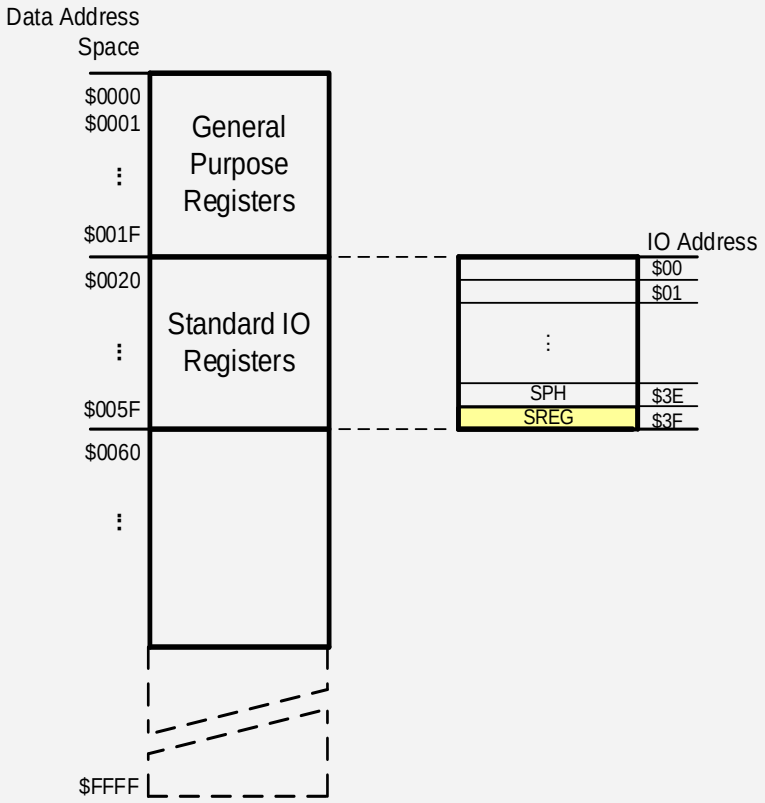
Status Register (SREG)



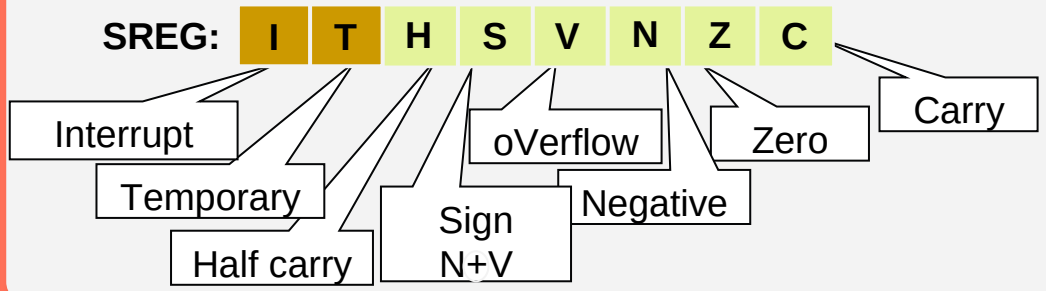
Carry Flag

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
		0	0	0	0	1	1	
	1	0	1	0	0	0	1	1
+	1	0	0	1	0	0	1	1
			1	1	0	1	1	0

Status Register (SREG)



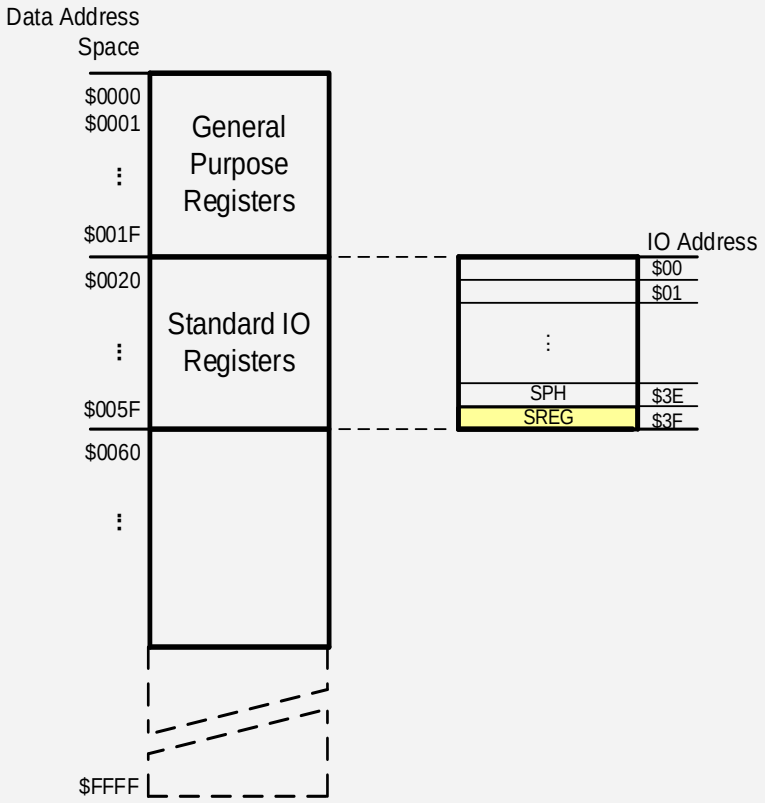
Status Register (SREG)



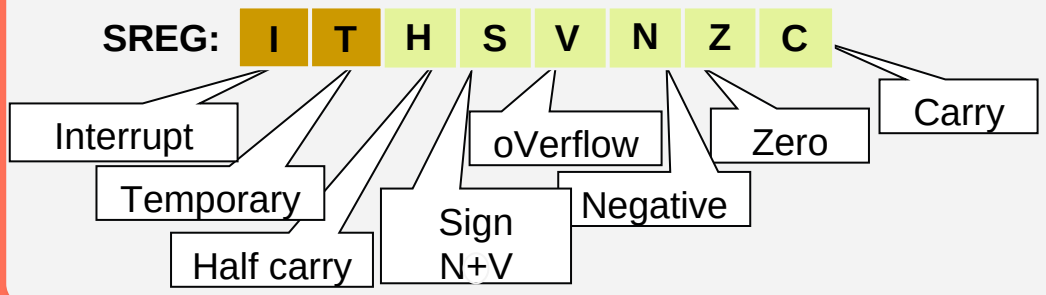
Carry Flag

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	0	0	0	0	0	1	1	
	1	0	1	0	0	0	1	1
+	1	0	0	1	0	0	1	1
		0	1	1	0	1	1	0

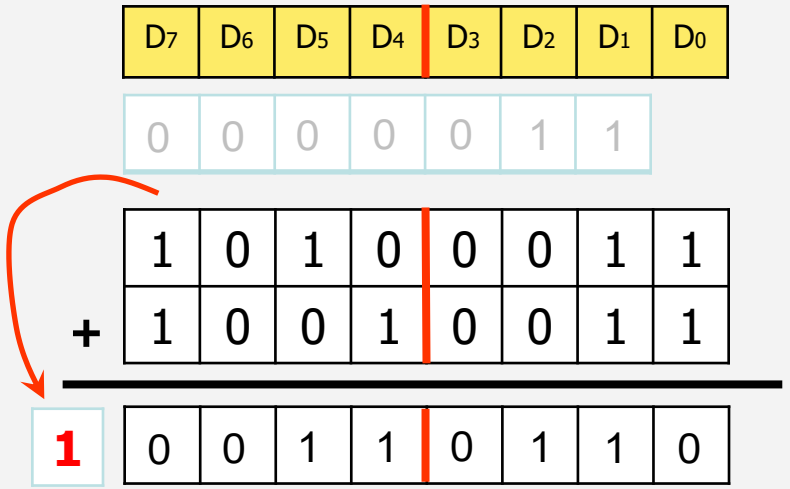
Status Register (SREG)



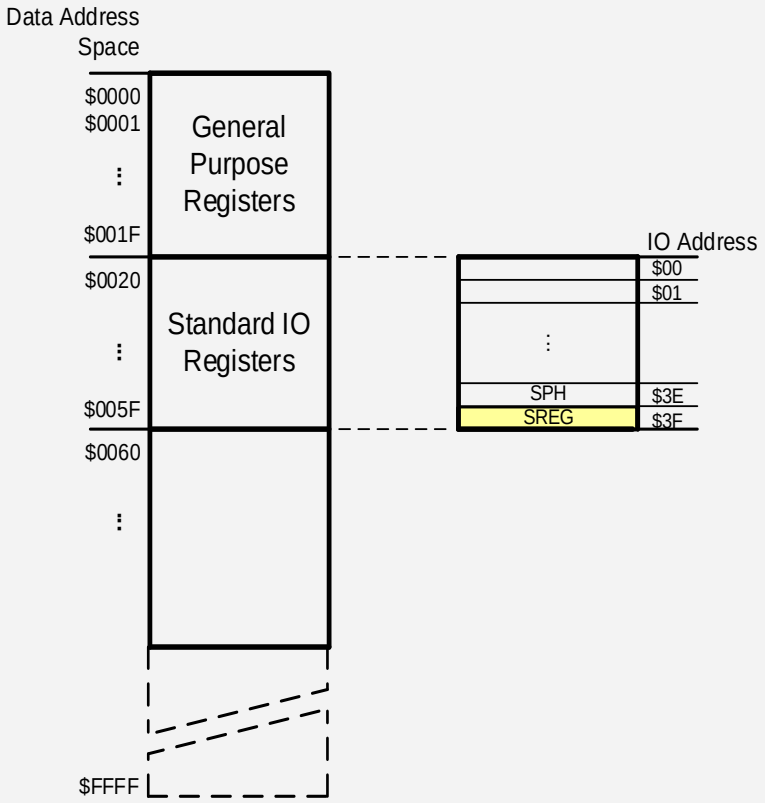
Status Register (SREG)



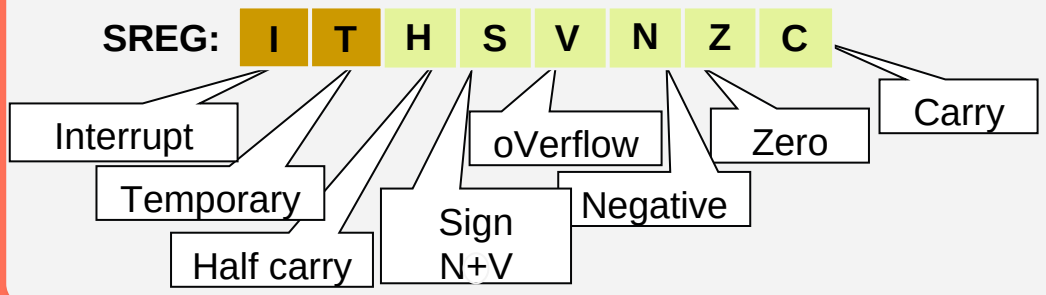
Carry Flag



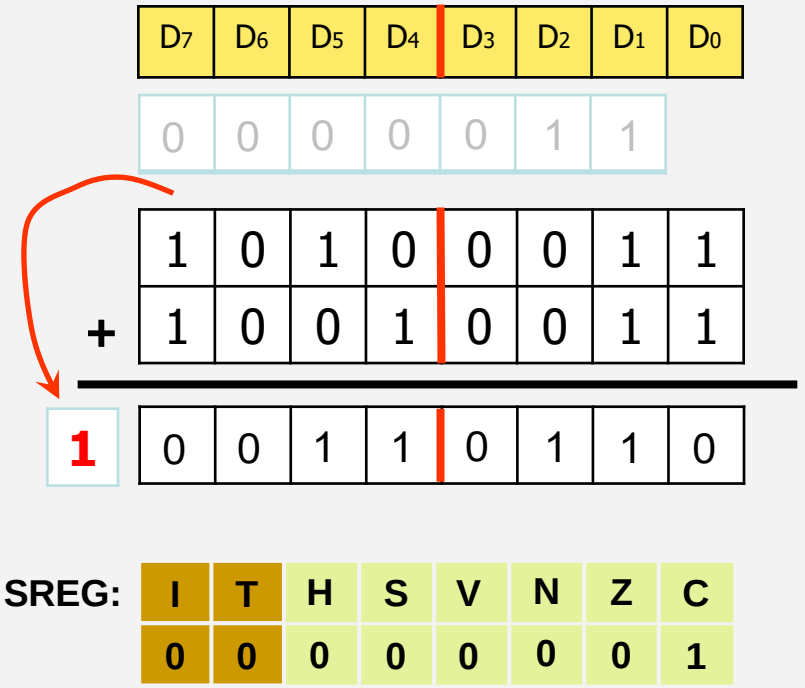
Status Register (SREG)



Status Register (SREG)

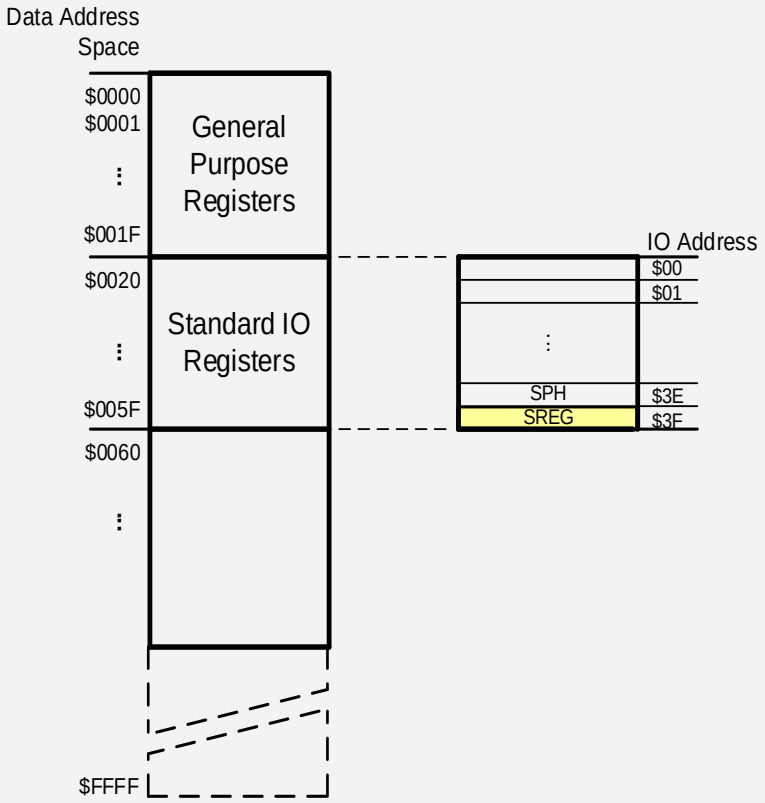


Carry Flag

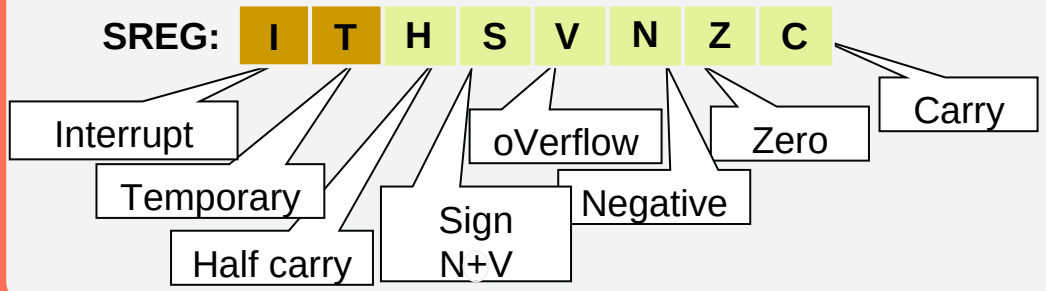


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	1

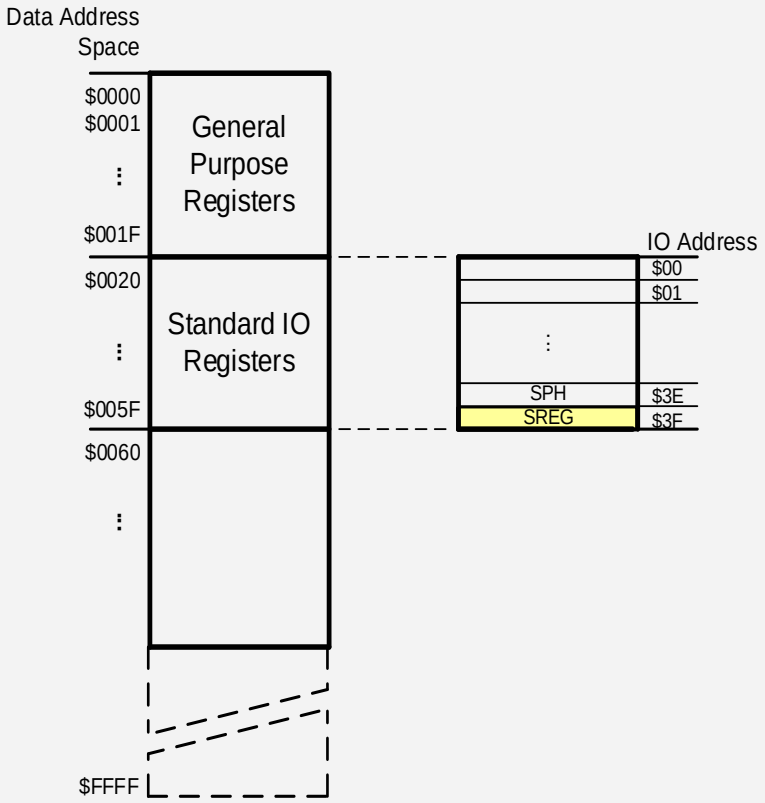
Status Register (SREG)



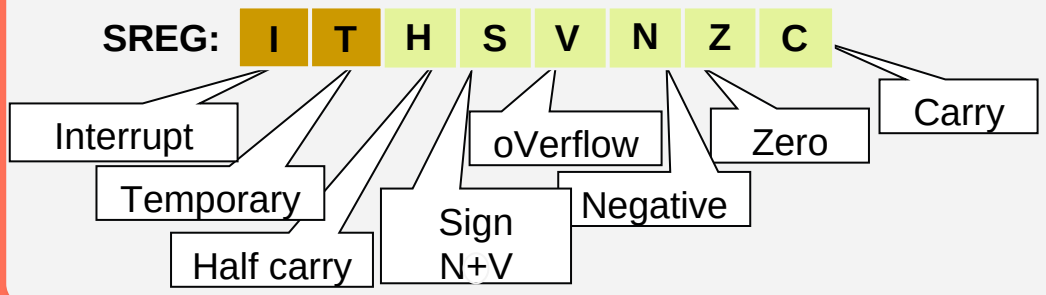
Status Register (SREG)



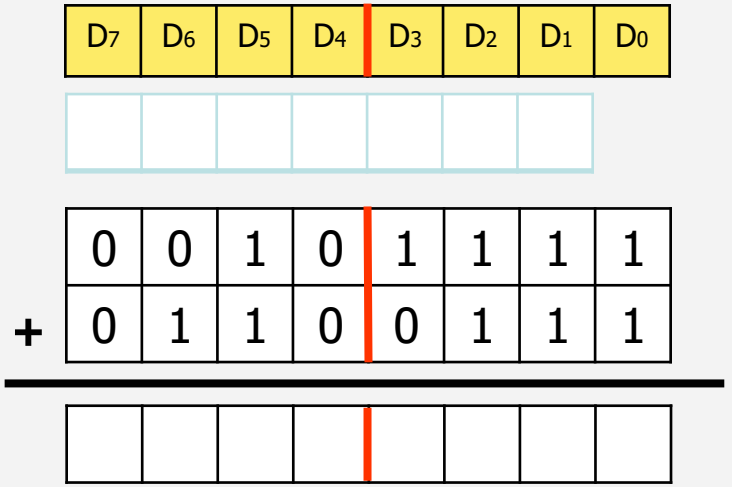
Status Register (SREG)



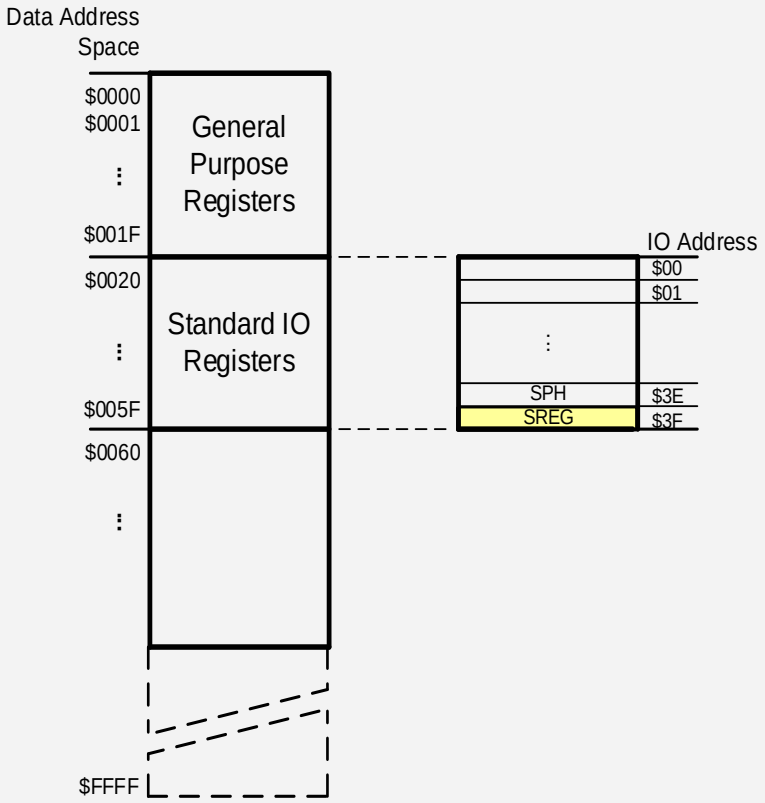
Status Register (SREG)



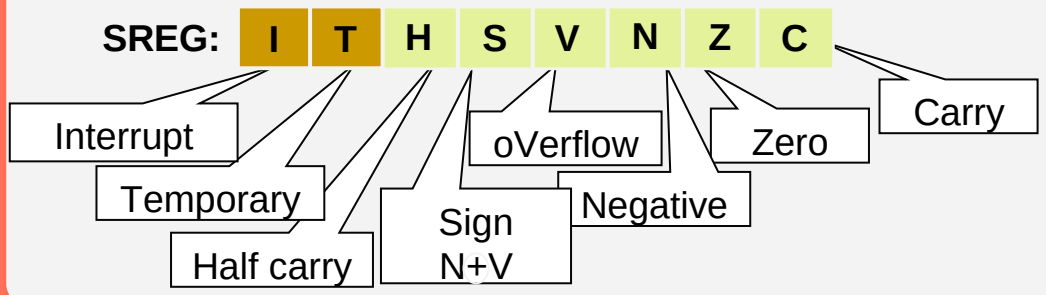
Half Cary Flag



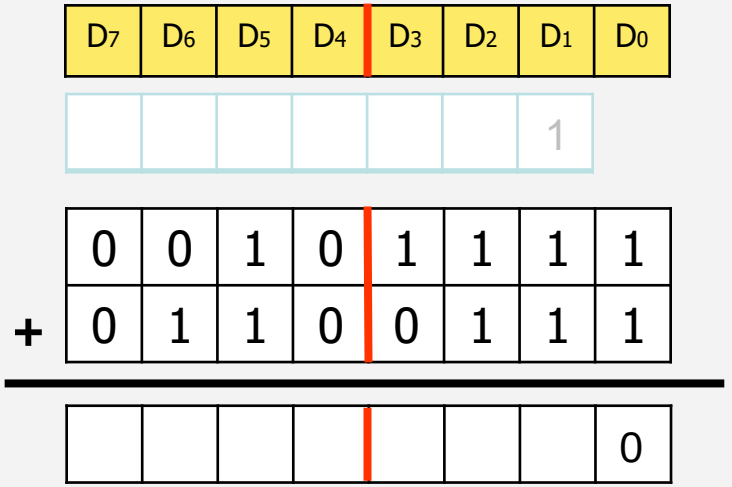
Status Register (SREG)



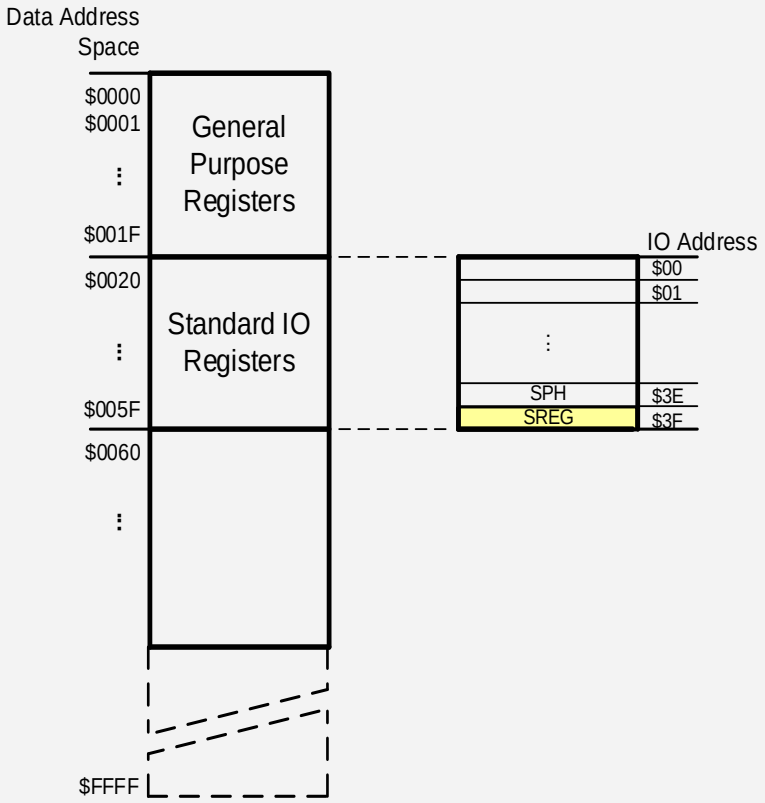
Status Register (SREG)



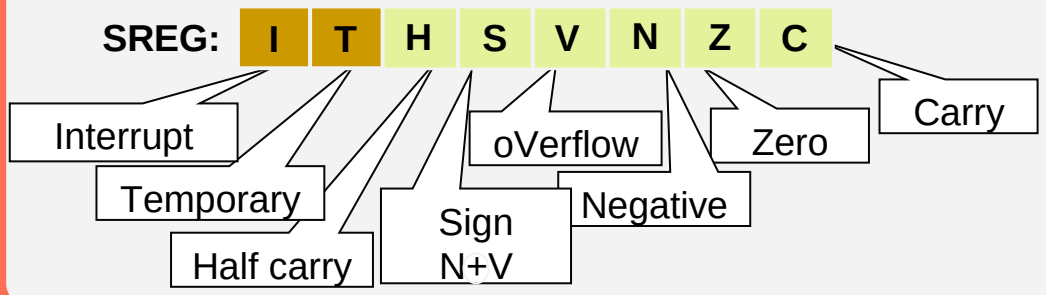
Half Cary Flag



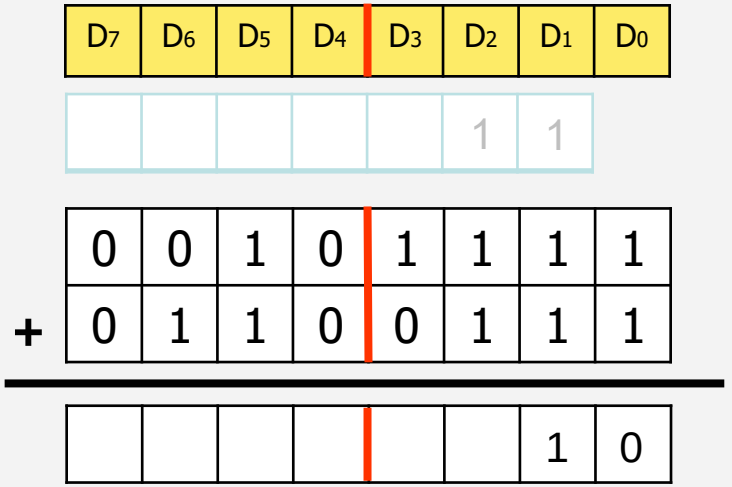
Status Register (SREG)



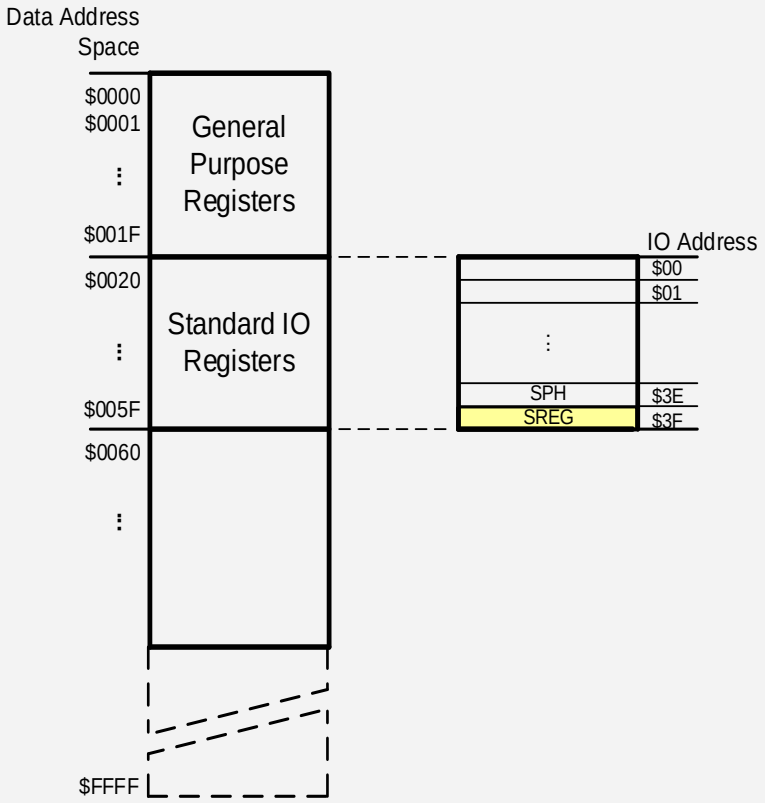
Status Register (SREG)



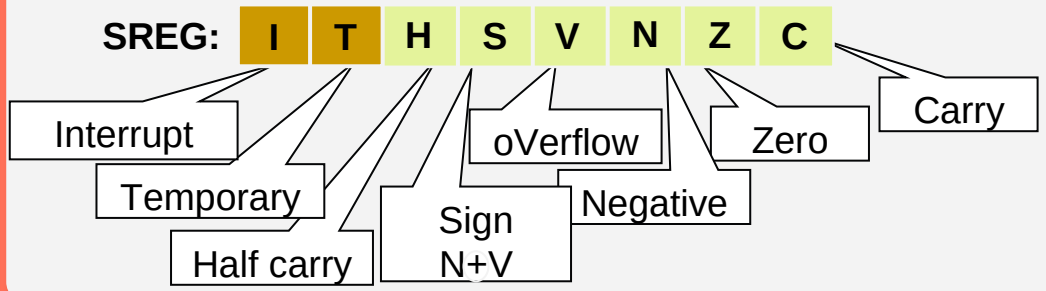
Half Cary Flag



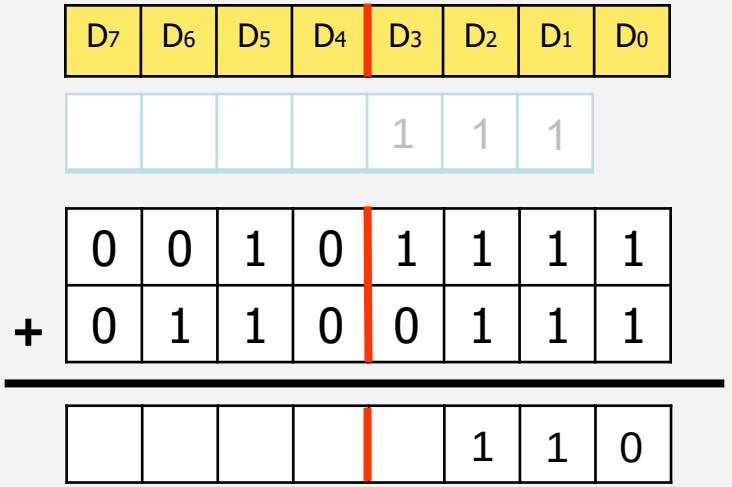
Status Register (SREG)



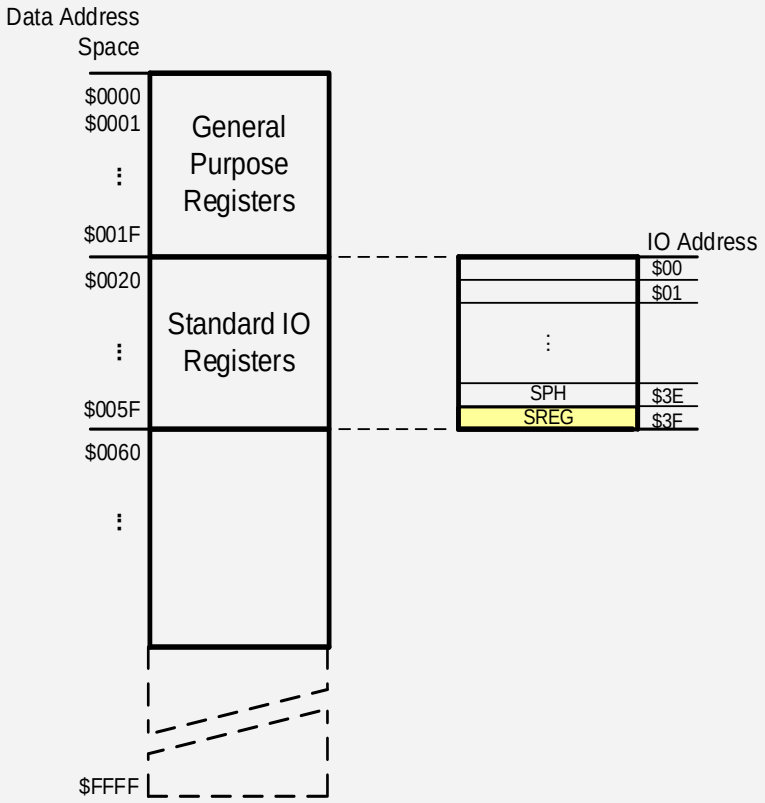
Status Register (SREG)



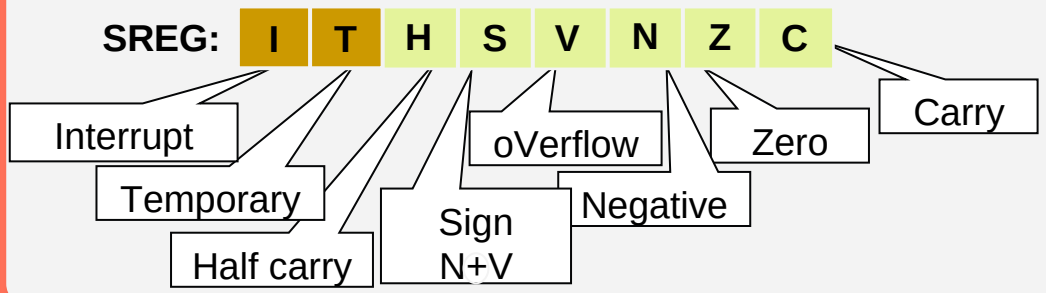
Half Cary Flag



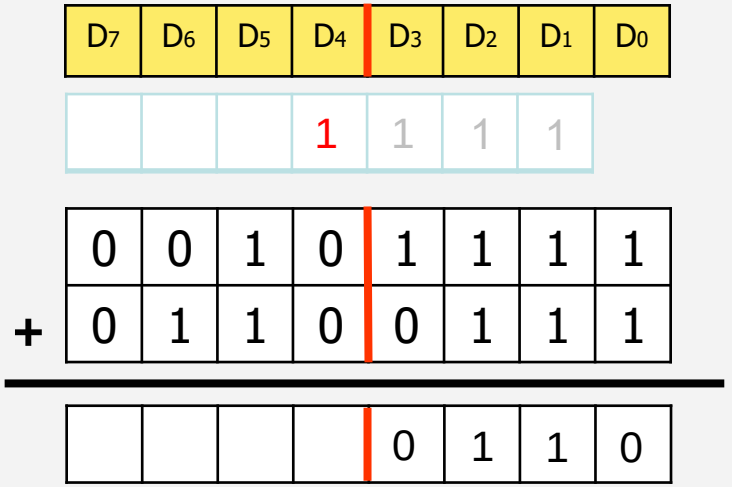
Status Register (SREG)



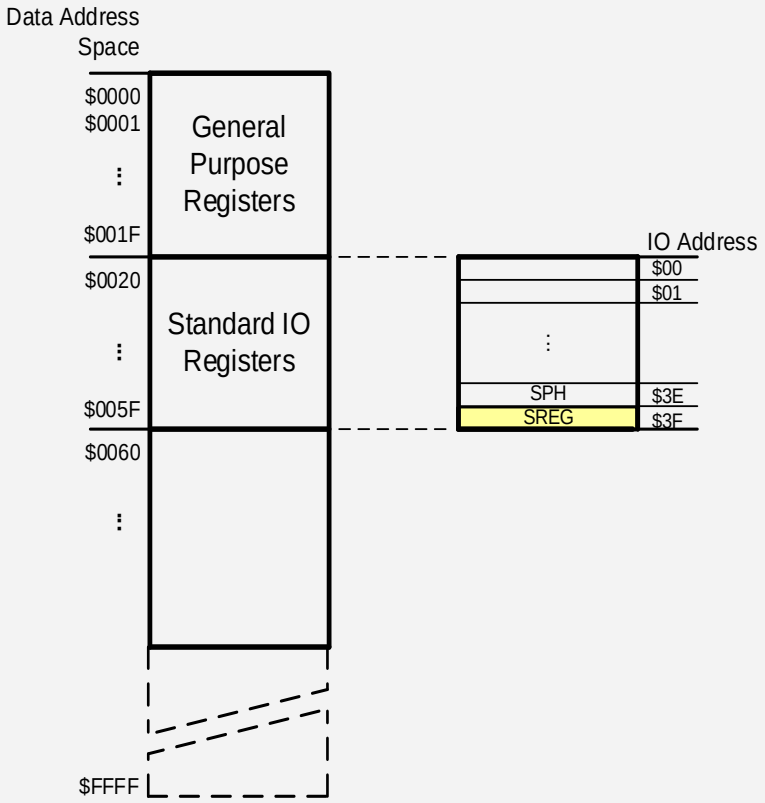
Status Register (SREG)



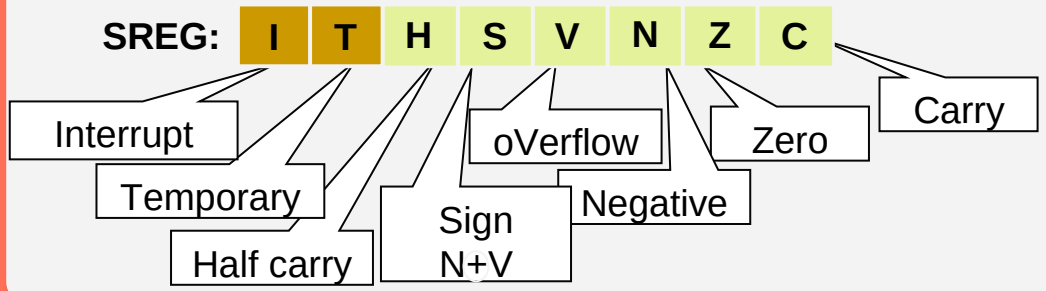
Half Cary Flag



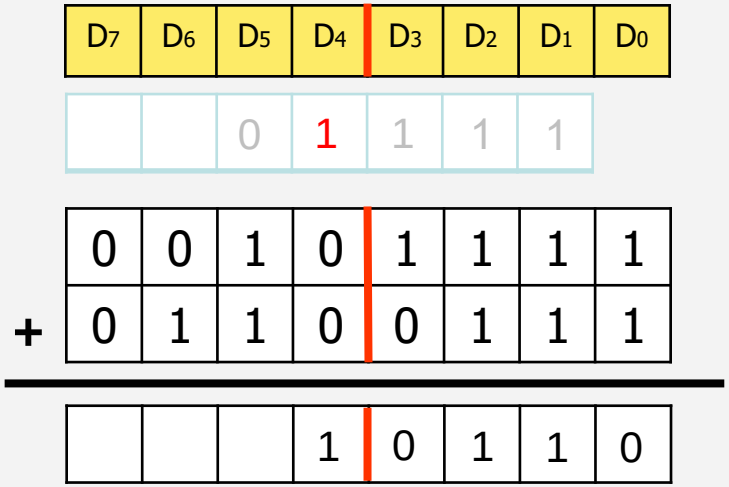
Status Register (SREG)



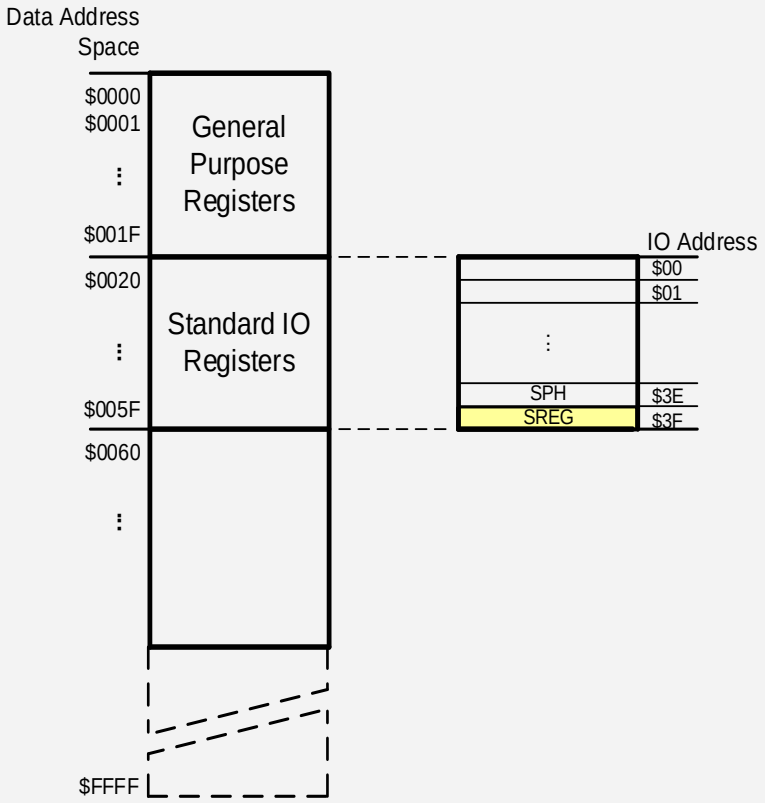
Status Register (SREG)



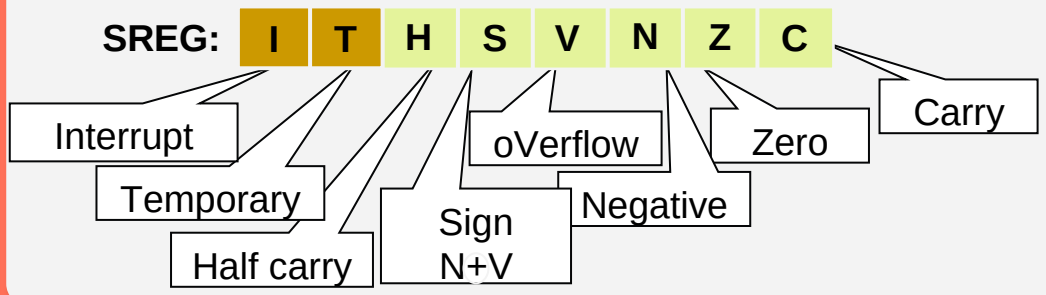
Half Cary Flag



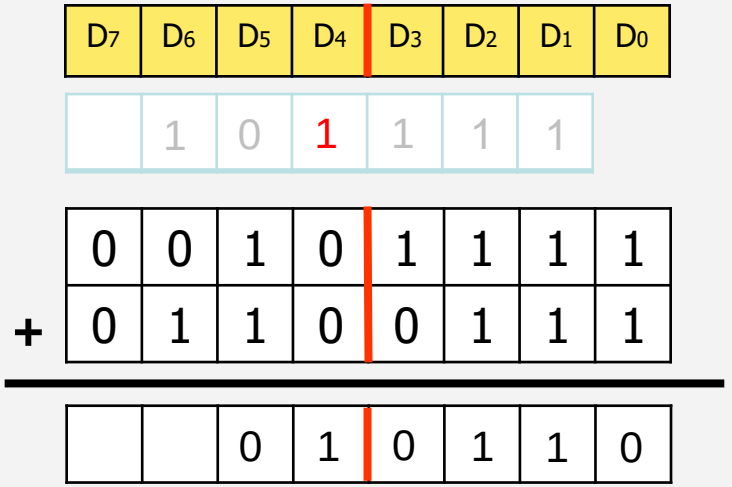
Status Register (SREG)



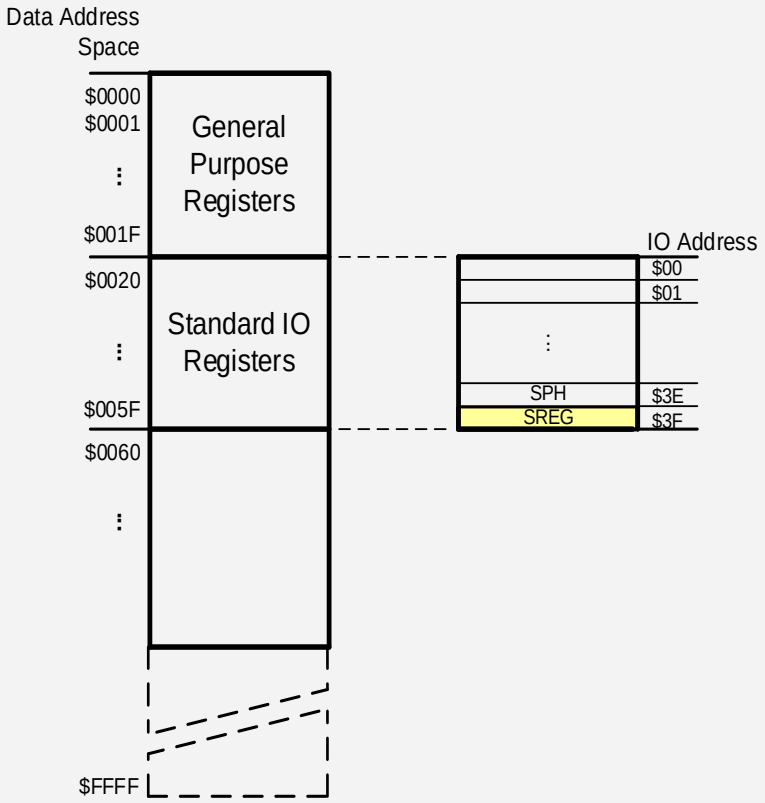
Status Register (SREG)



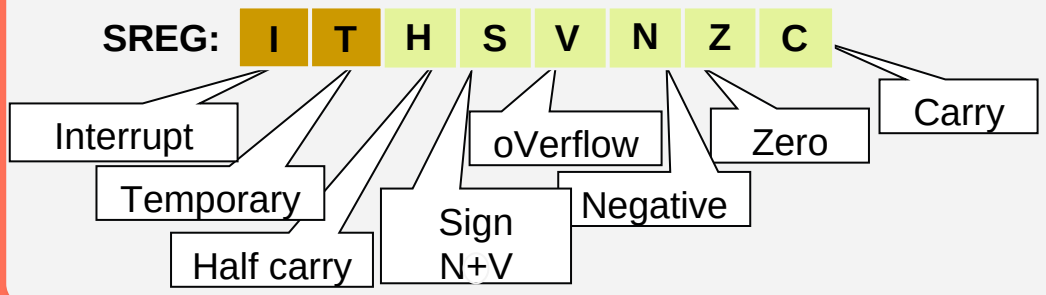
Half Cary Flag



Status Register (SREG)



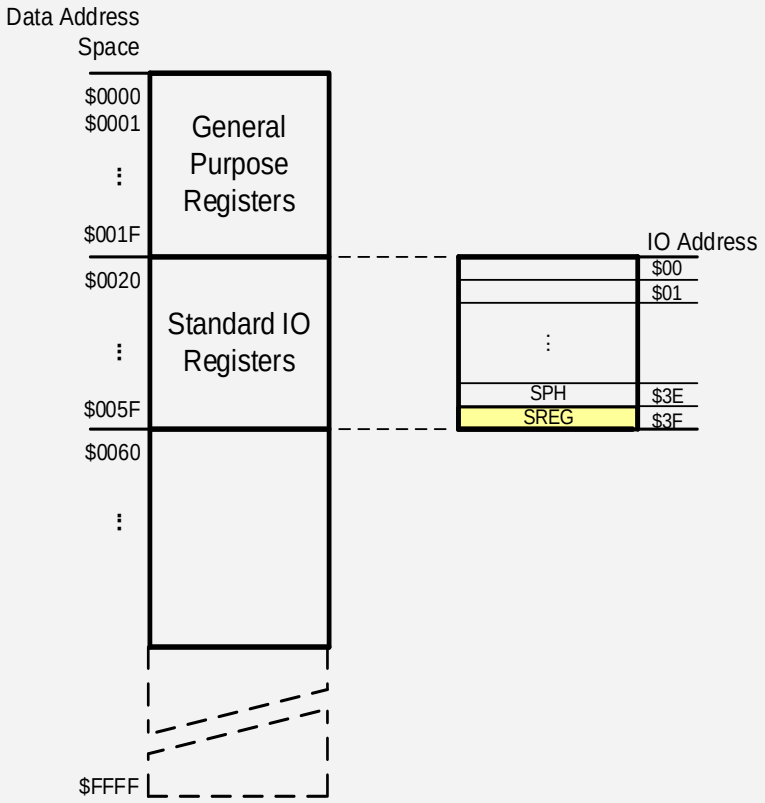
Status Register (SREG)



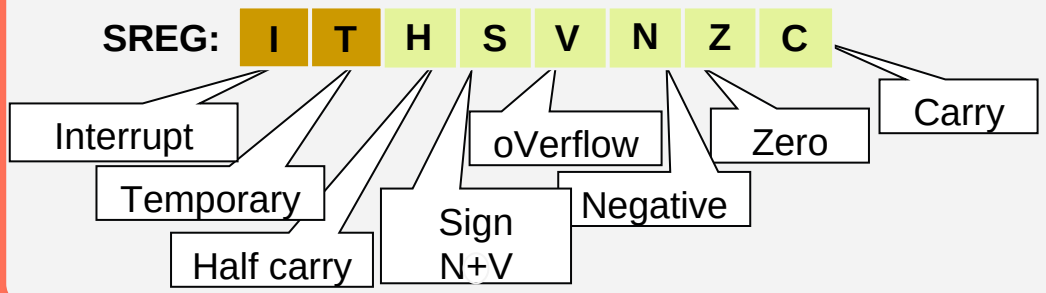
Half Cary Flag

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	0	1	1	1	1	
0	0	1	0	1	1	1	1
+ 0	1	1	0	0	1	1	1
	0	0	1	0	1	1	0

Status Register (SREG)



Status Register (SREG)

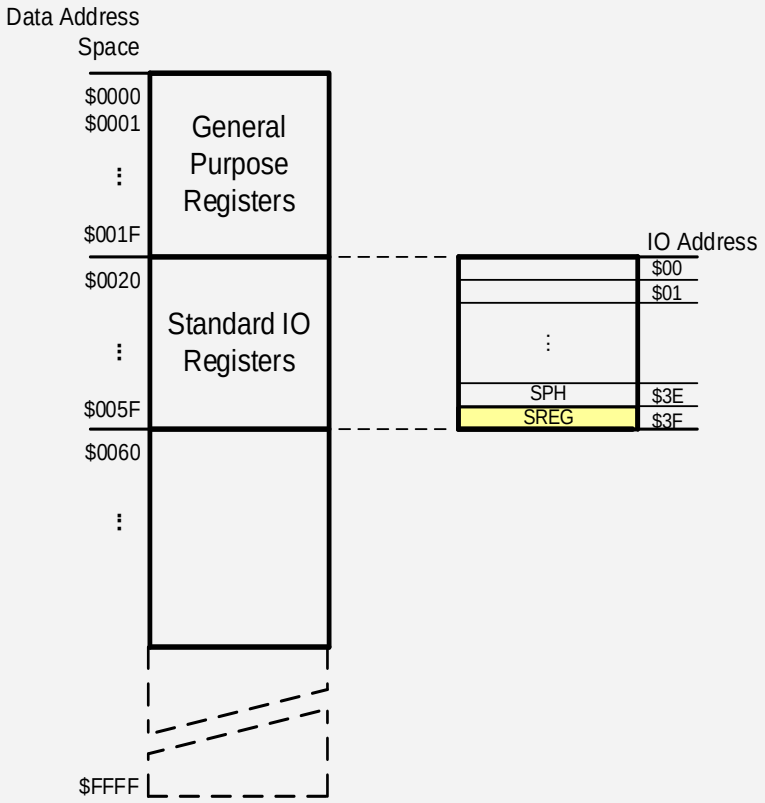


Half Cary Flag

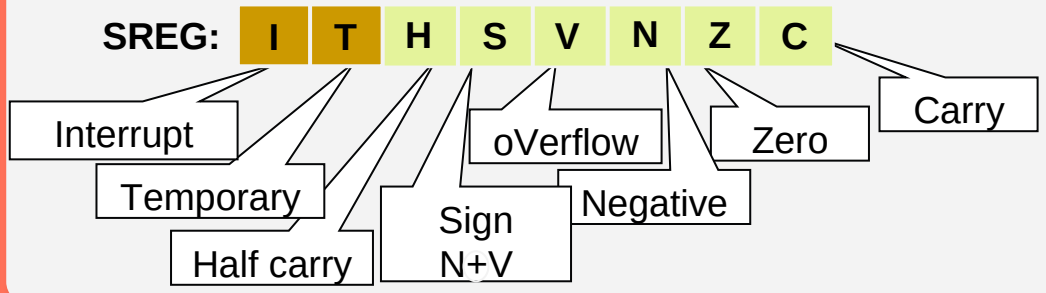
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	0	1	1	1	1	

	0	0	1	0	1	1	1	1
+	0	1	1	0	0	1	1	1
	1	0	0	1	0	1	1	0

Status Register (SREG)



Status Register (SREG)



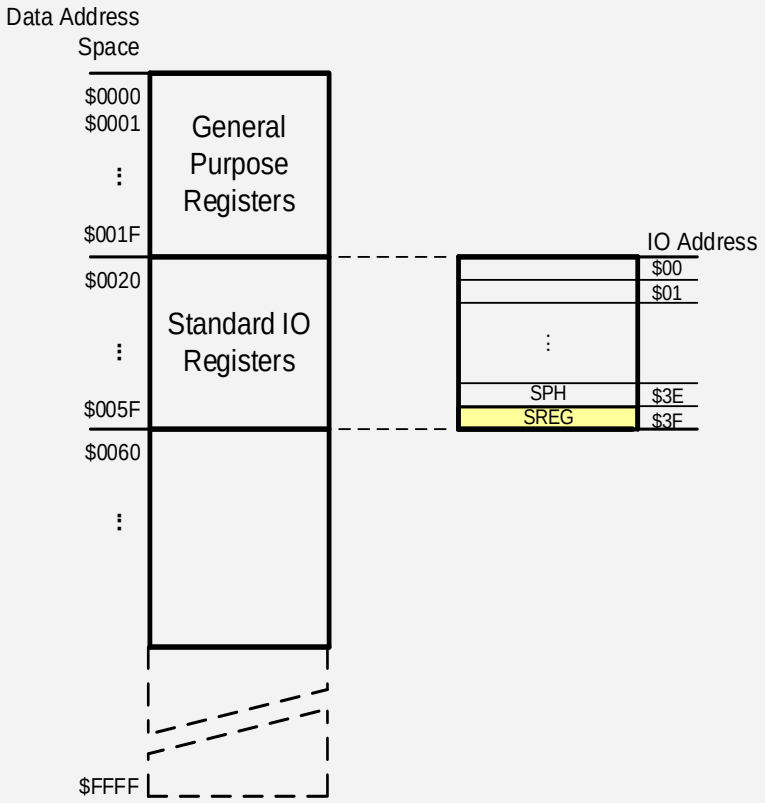
Half Cary Flag

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	0	1	1	1	1	

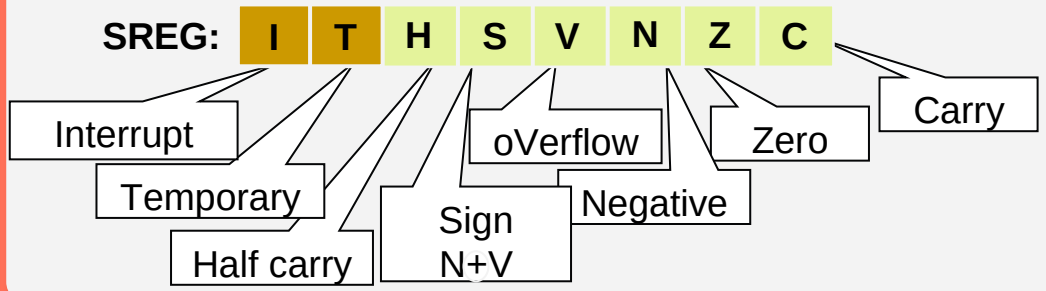
	0	0	1	0	1	1	1	1
+	0	1	1	0	0	1	1	1
	1	0	0	1	0	1	1	0

SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

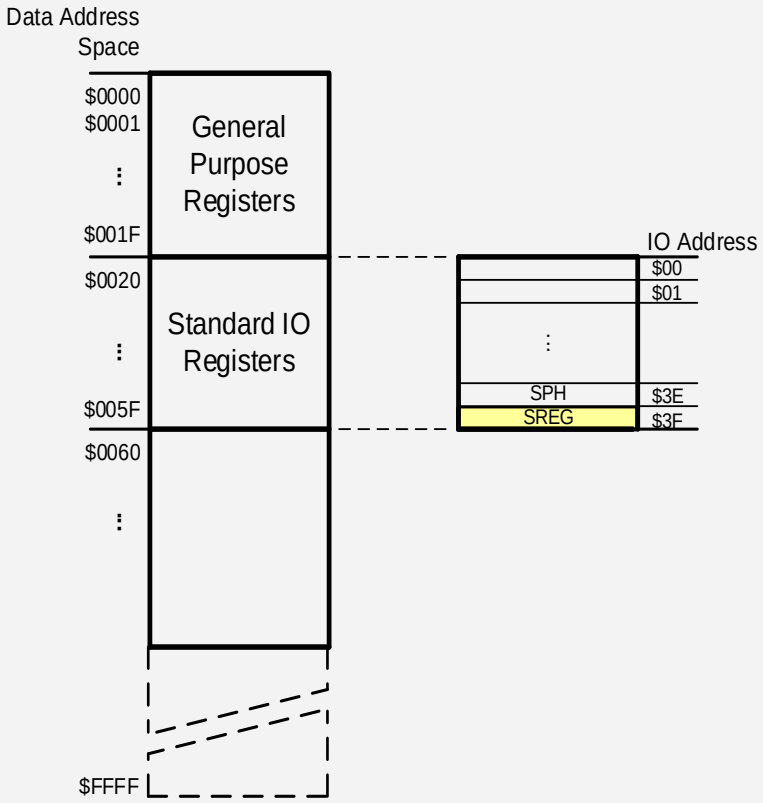
Status Register (SREG)



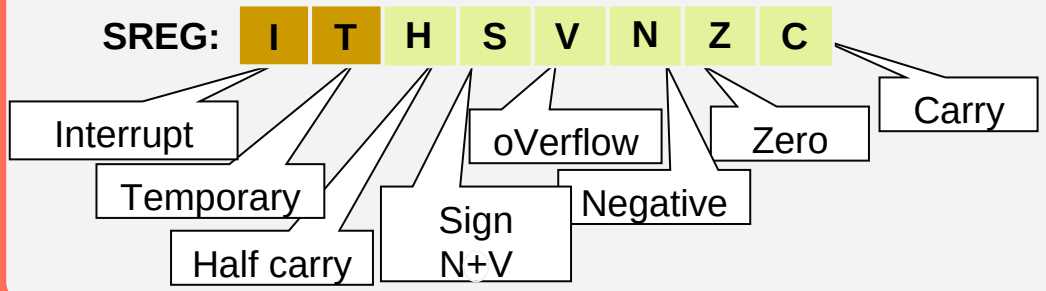
Status Register (SREG)



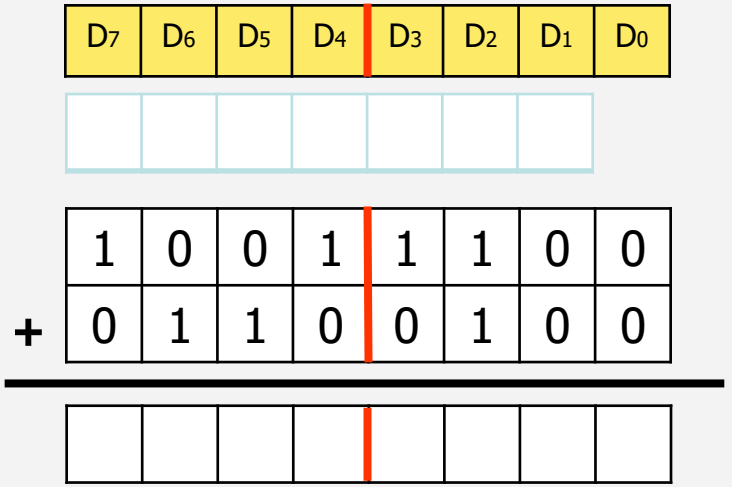
Status Register (SREG)



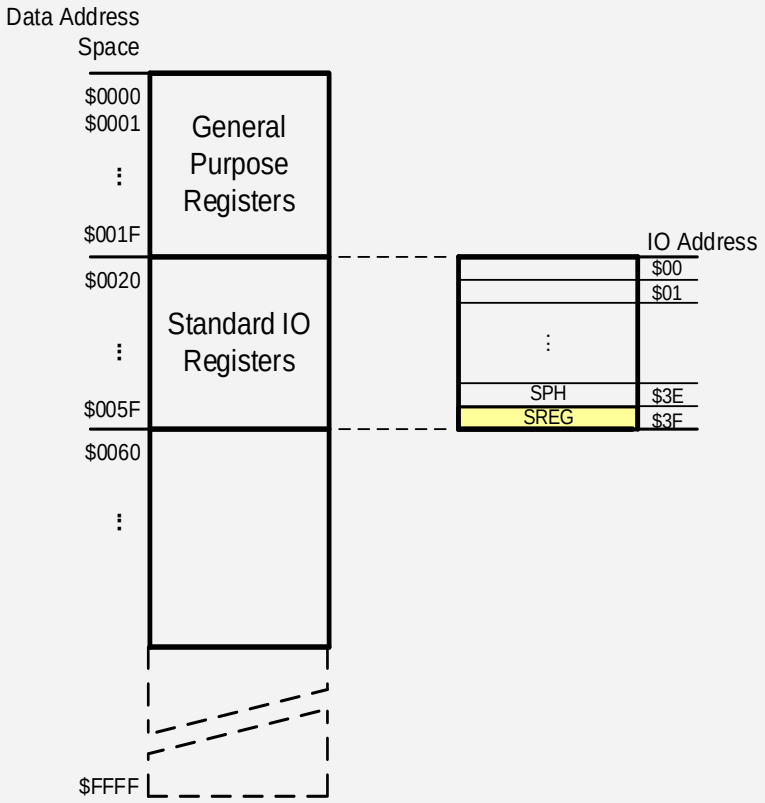
Status Register (SREG)



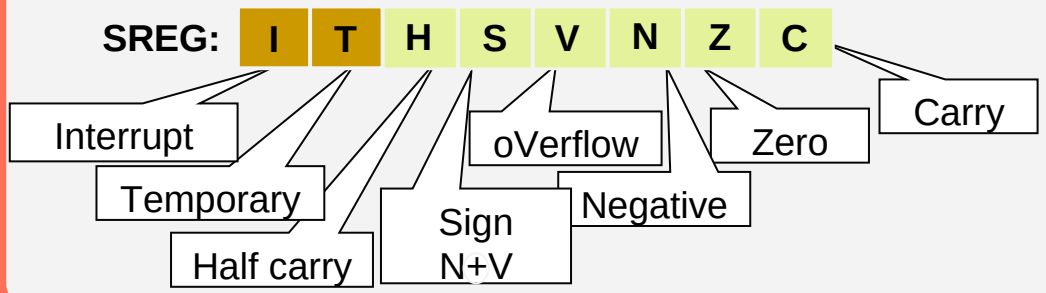
Zero Flag



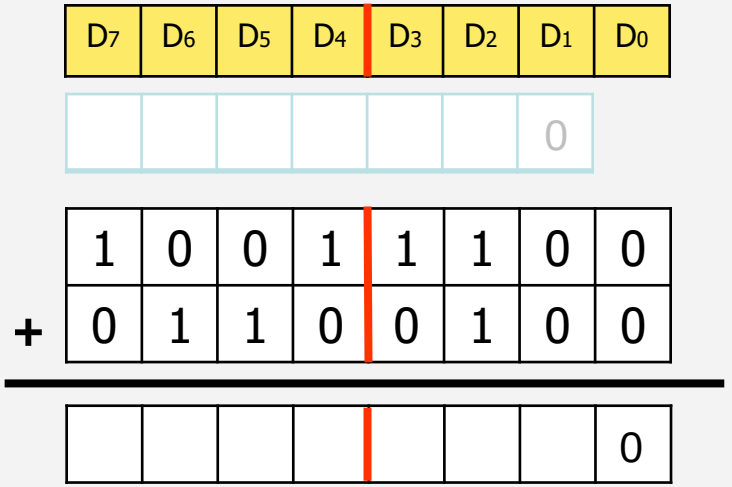
Status Register (SREG)



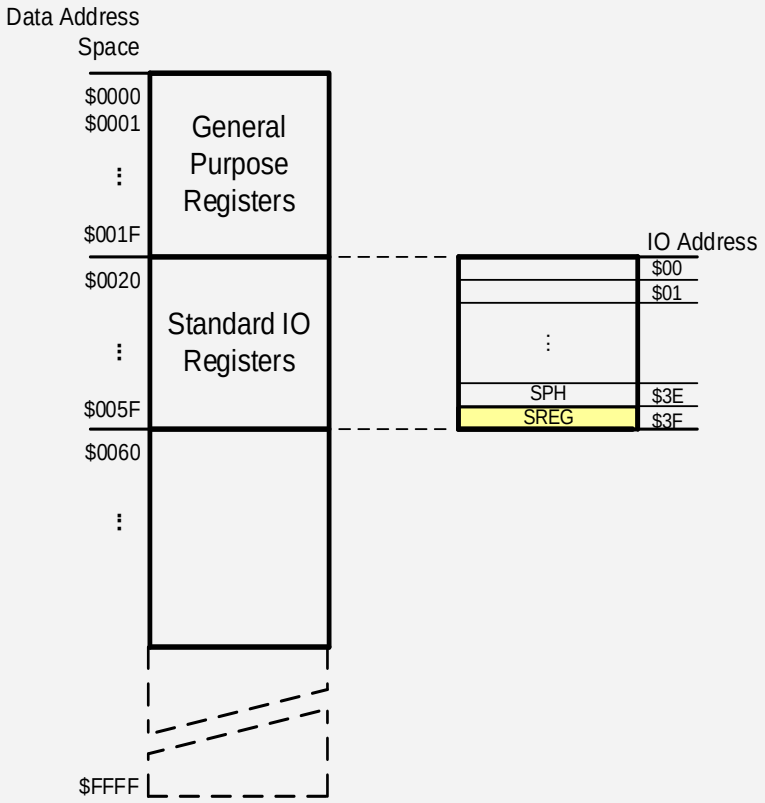
Status Register (SREG)



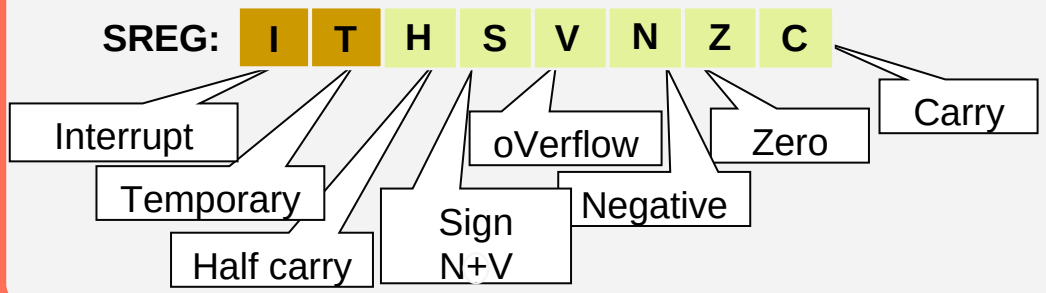
Zero Flag



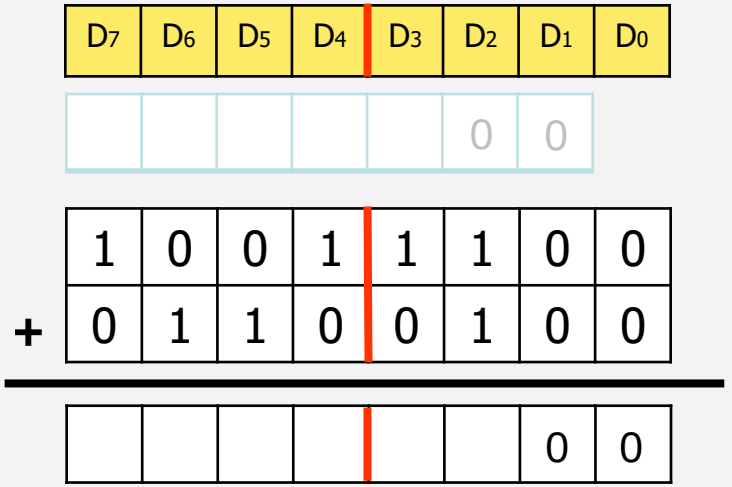
Status Register (SREG)



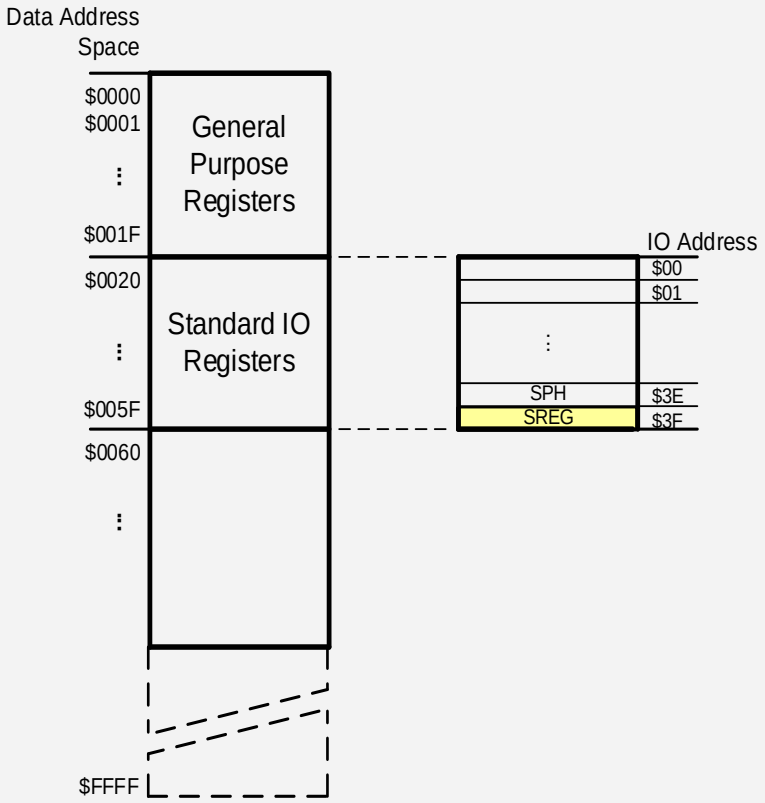
Status Register (SREG)



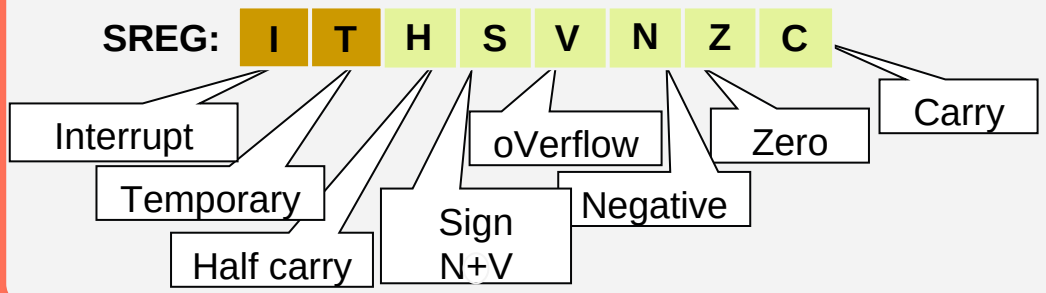
Zero Flag



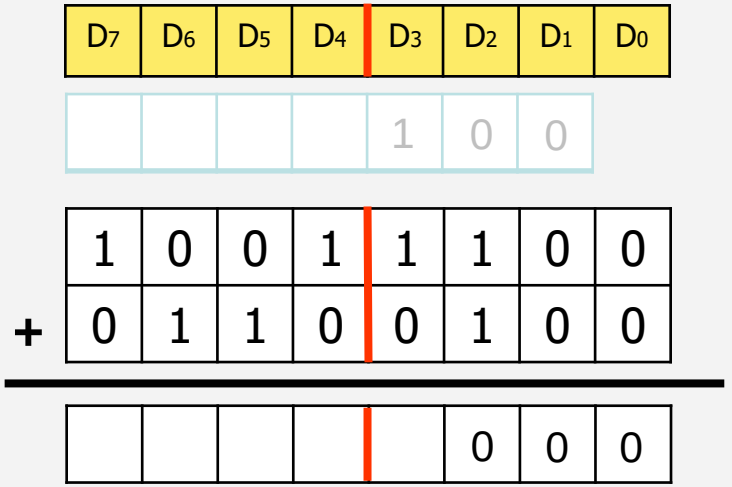
Status Register (SREG)



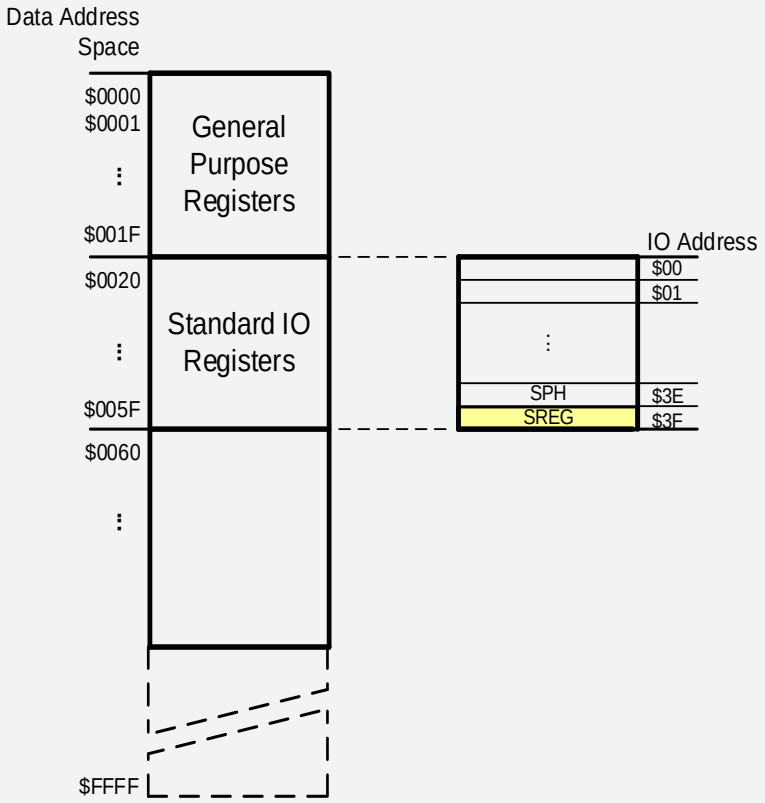
Status Register (SREG)



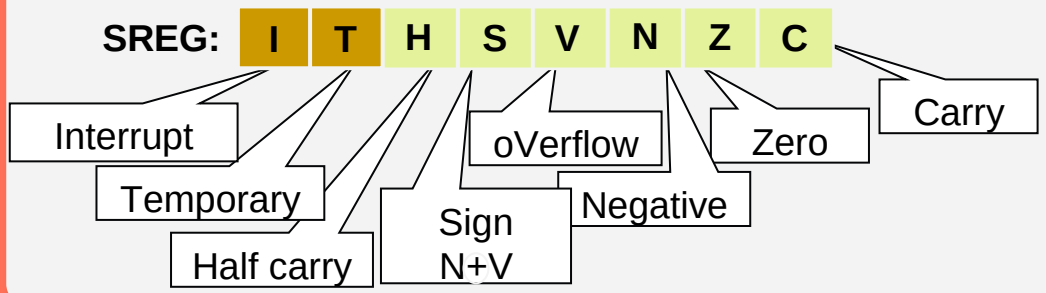
Zero Flag



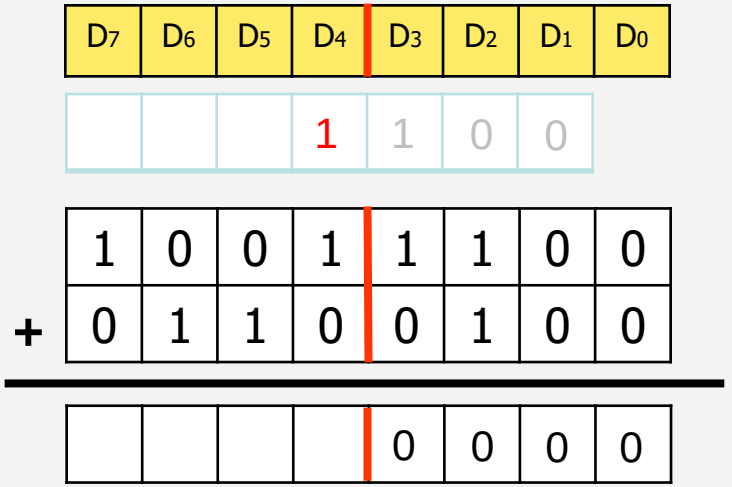
Status Register (SREG)



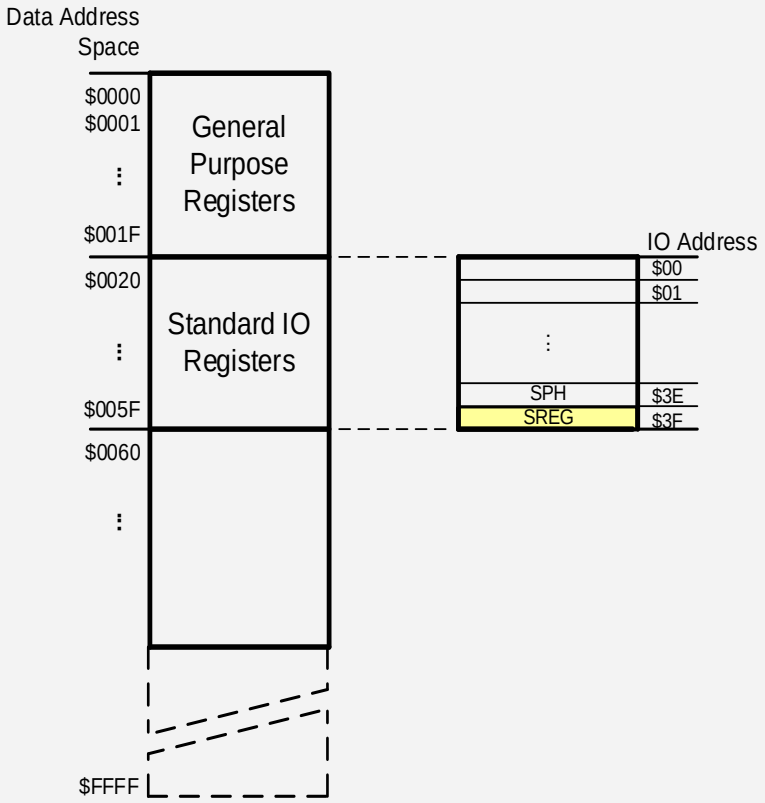
Status Register (SREG)



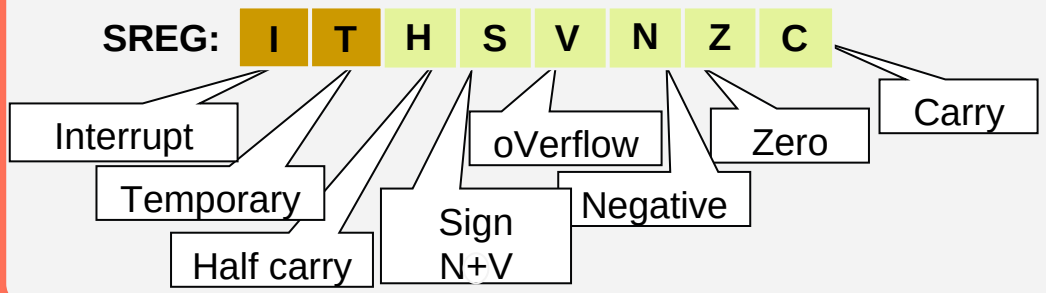
Zero Flag



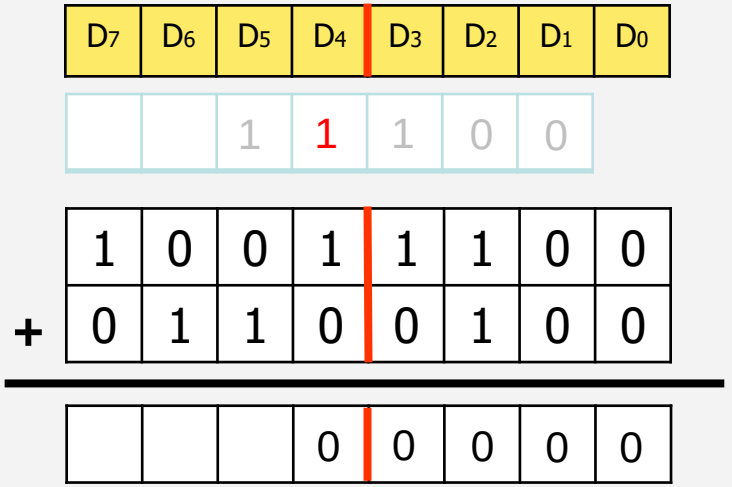
Status Register (SREG)



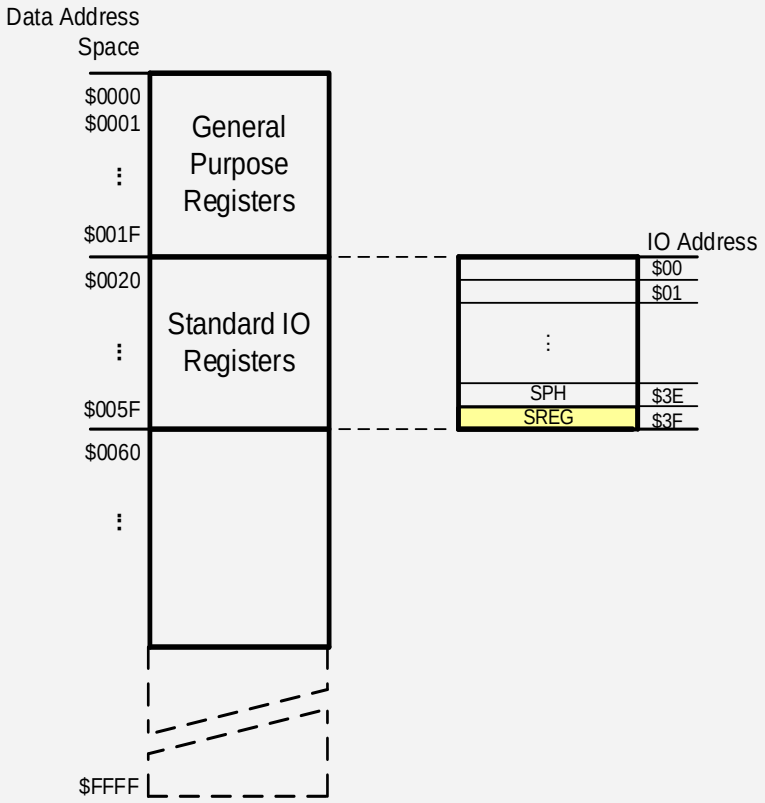
Status Register (SREG)



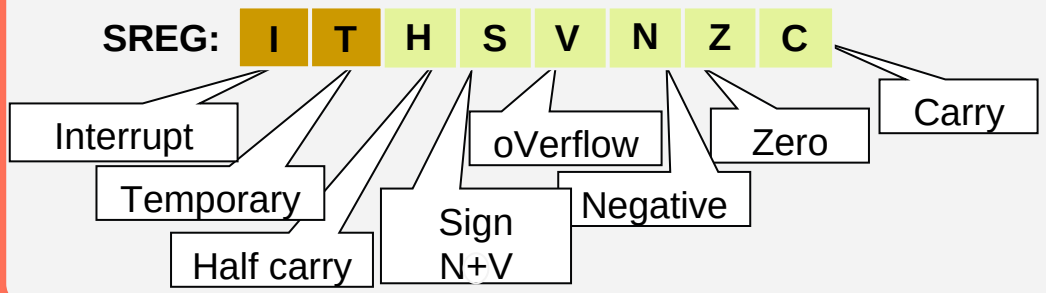
Zero Flag



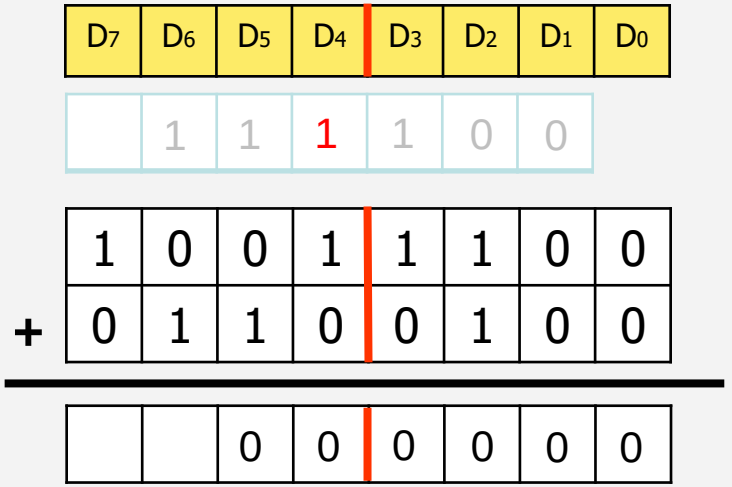
Status Register (SREG)



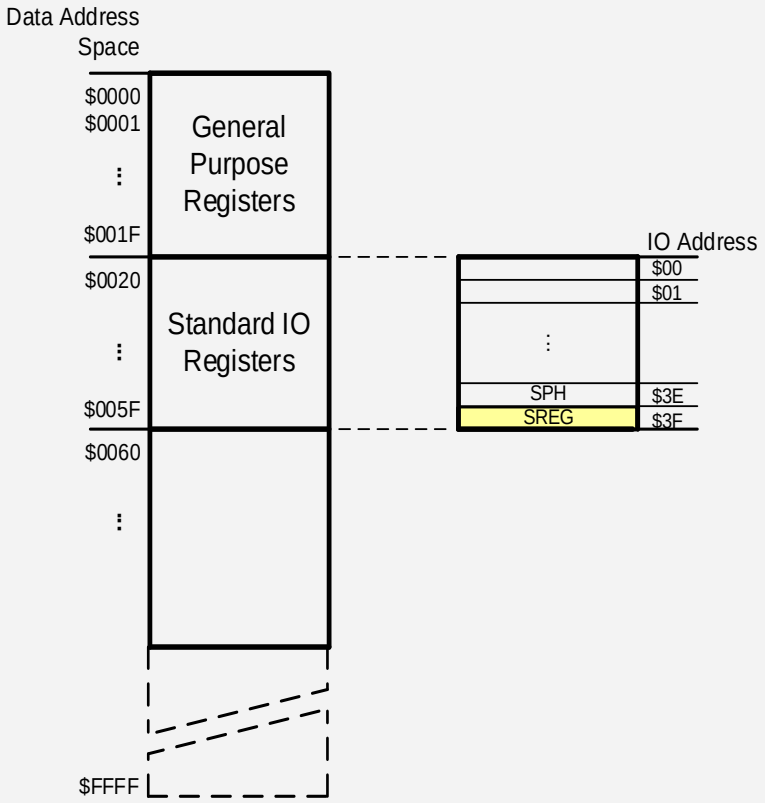
Status Register (SREG)



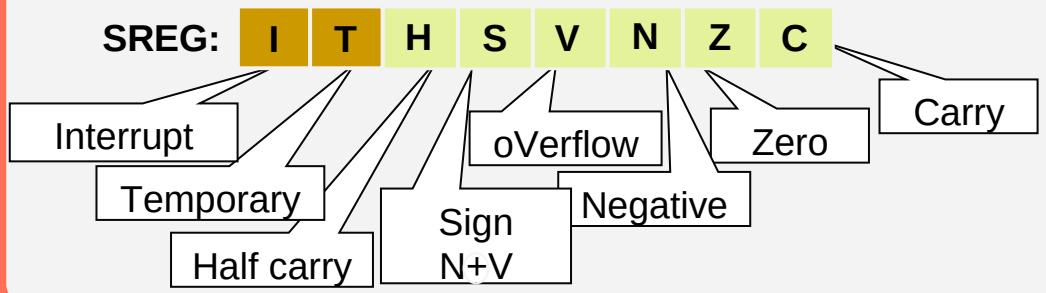
Zero Flag



Status Register (SREG)



Status Register (SREG)

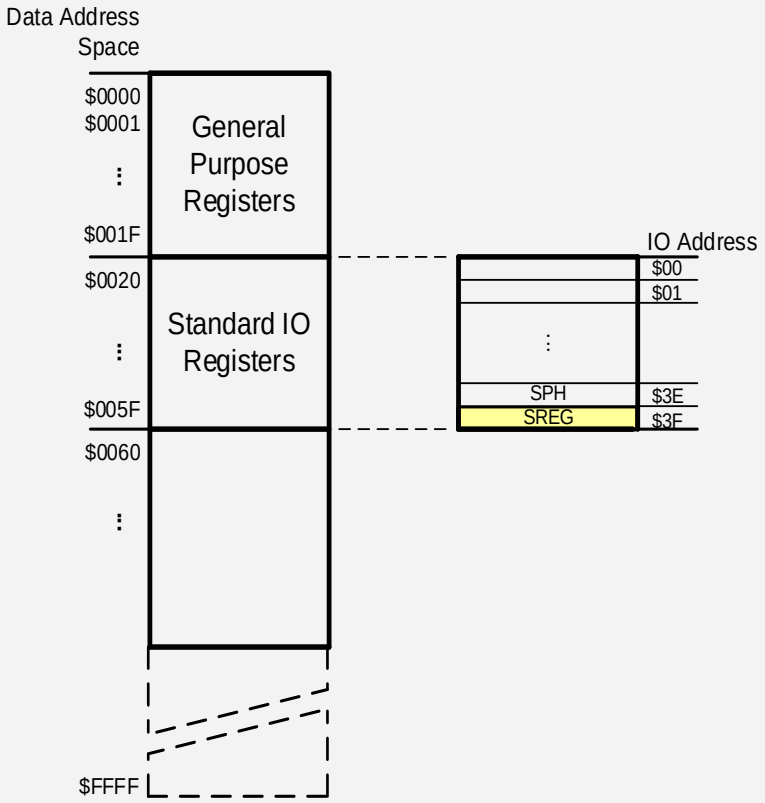


Zero Flag

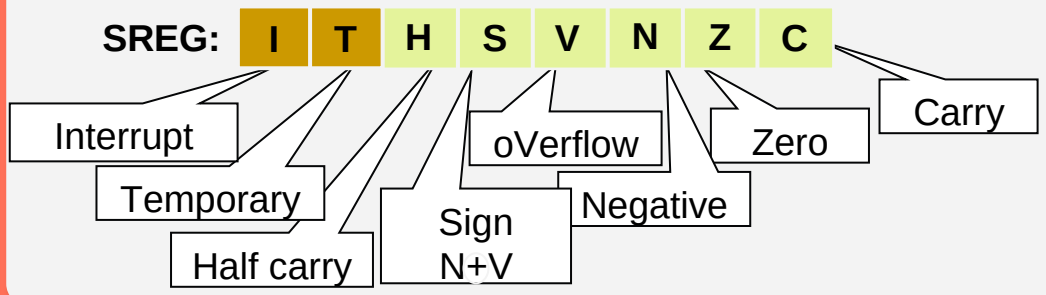
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
1	1	1	1	1	0	0	

	1	0	0	1	1	1	0	0
+	0	1	1	0	0	1	0	0
		0	0	0	0	0	0	0

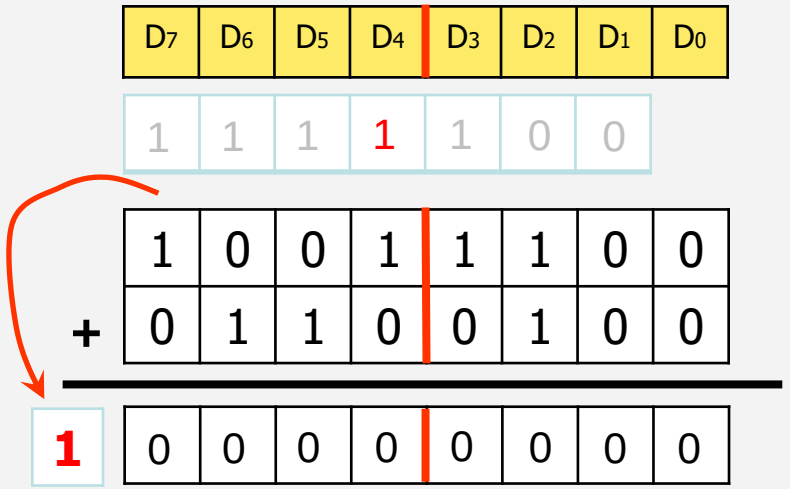
Status Register (SREG)



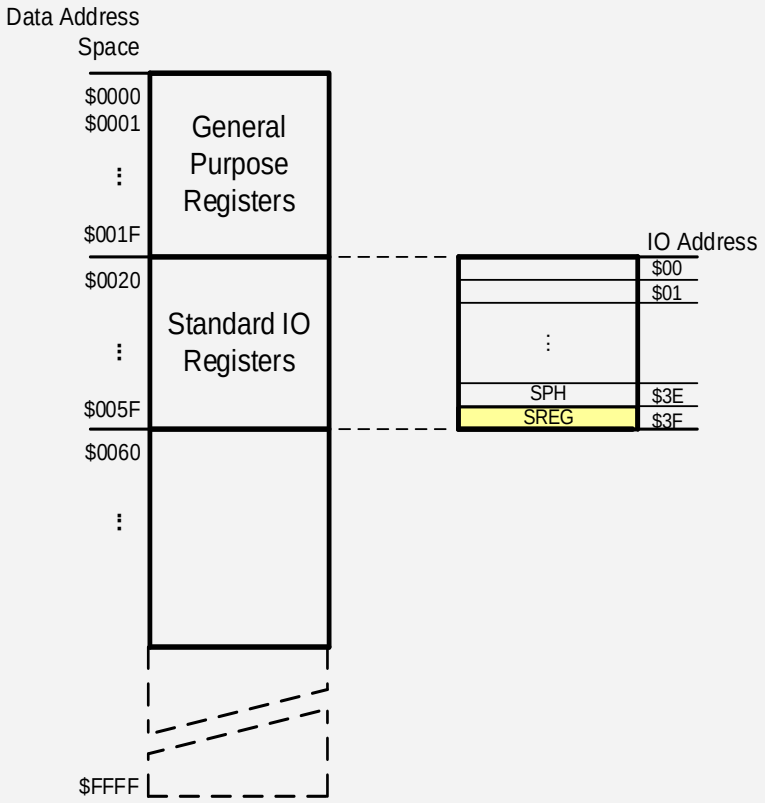
Status Register (SREG)



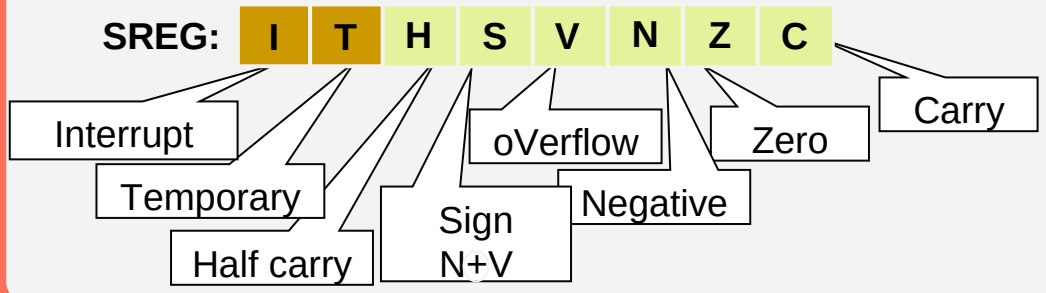
Zero Flag



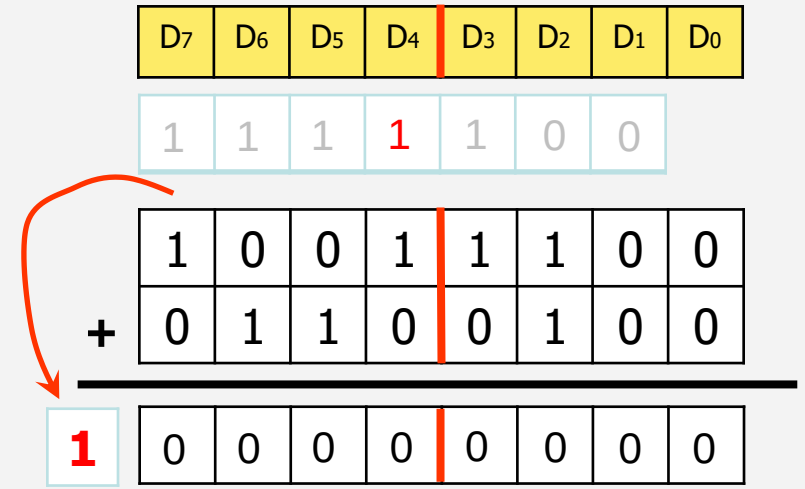
Status Register (SREG)



Status Register (SREG)



Zero Flag



SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	1	1

Representation of Negative Binary Numbers

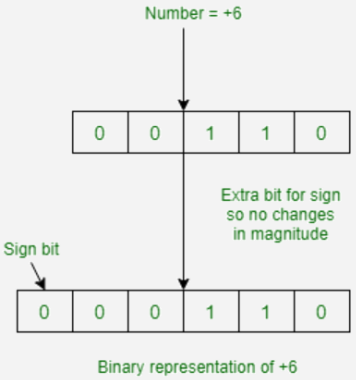
1. Signed Magnitude Method

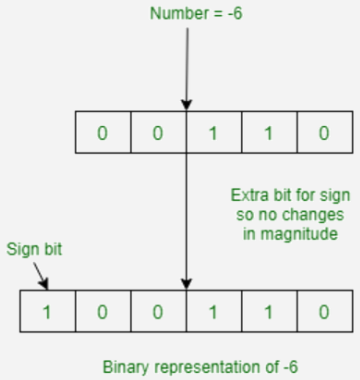
Range of Numbers:

$-(2^{(n-1)}-1)$ to $+(2^{(n-1)}-1)$

Example (for 8-bit registers):

-127 to +127





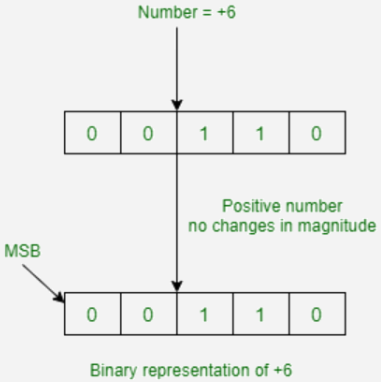
2. 1's Complement Method

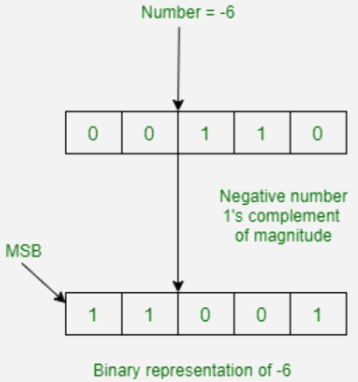
Range of Numbers:

$-(2^{(n-1)}-1)$ to $+(2^{(n-1)}-1)$

Example (for 8-bit registers):

-127 to +127





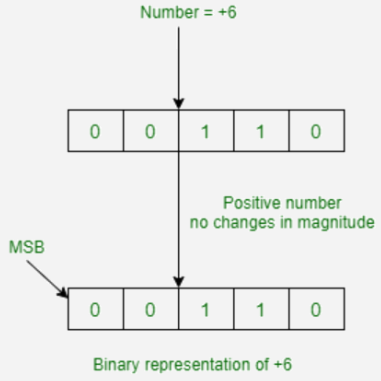
3. 2's Complement Method

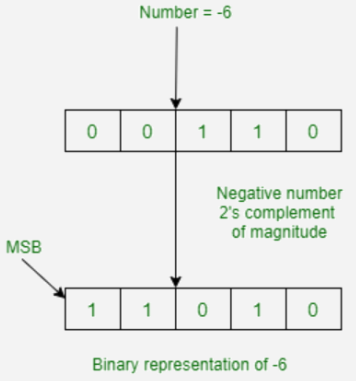
Range of Numbers:

$-(2^{(n-1)})$ to $+(2^{(n-1)}-1)$

Example (for 8-bit registers):

-128 to +127

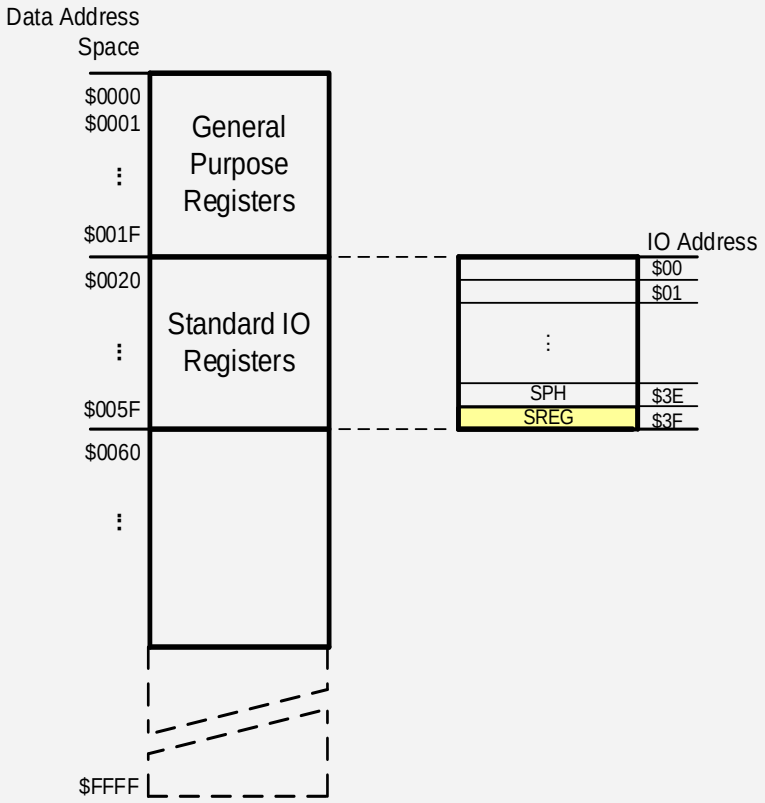




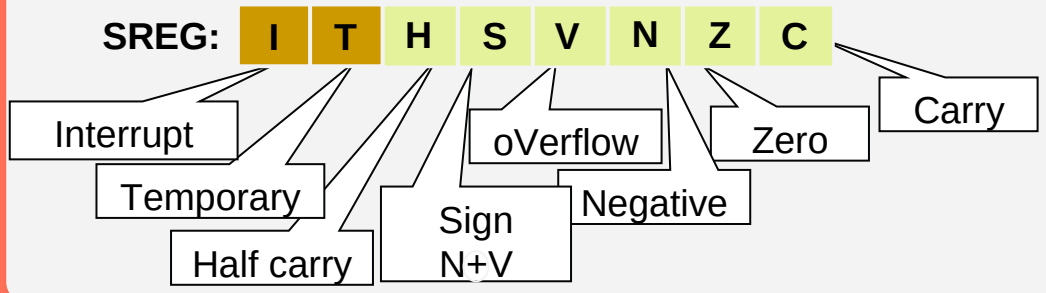
Representation of Negative Binary Numbers

decimal	Signed Magnitude	1's Complement	2's Complement
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
-0	1000	1111	-
-1	1001	1110	1111
-2	1010	1101	1110
-3	1011	1100	1101
-4	1100	1011	1100
-5	1101	1010	1011
-6	1110	1001	1010
-7	1111	1000	1001

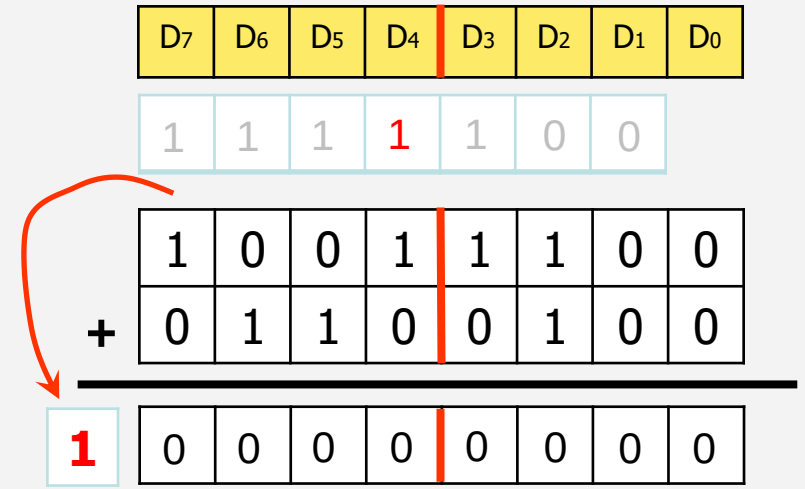
Status Register (SREG)



Status Register (SREG)



Zero Flag



SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	1	1

Representation of Negative Binary Numbers

Data Address Space

$0000
$0001
⋮
$001F
$0020
⋮
$005F
$0060
⋮
$FFFF

Remember 2's Complement Method

-5 in 2's Complement Representation?

1) $5_{(10)} = 00000101_{(2)}$

2) $\sim 5_{(10)} = 11111010_{(2)}$

3)

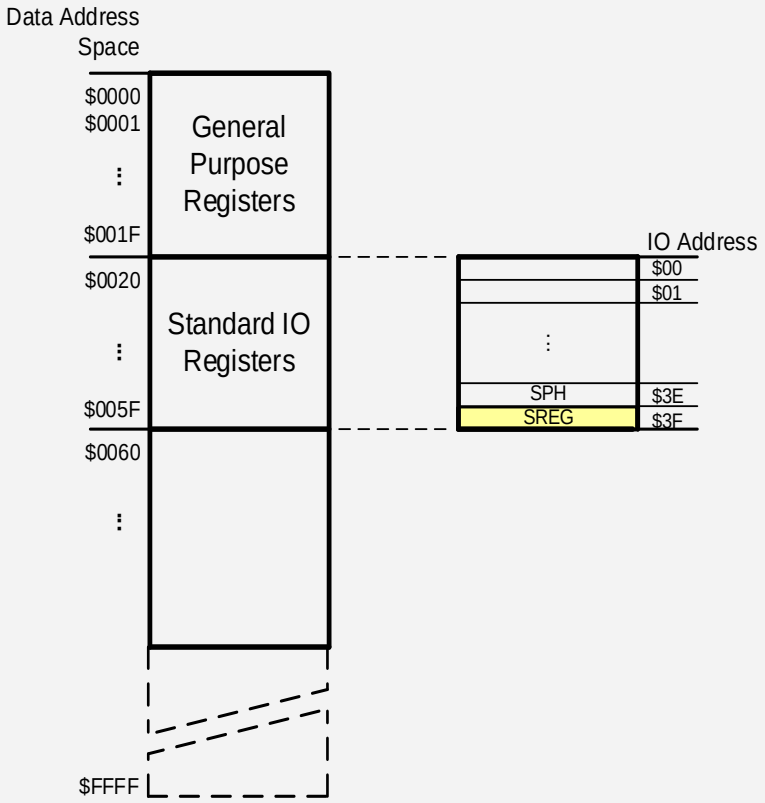
```
  11111010
+ 00000001
----------
  11111011
```

Answer Evaluation:

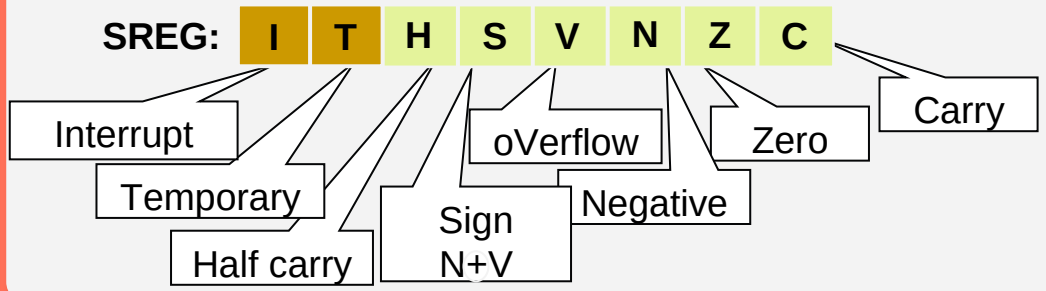
1	1	1	1	1	0	1	1
128	64	32	16	8	4	2	1

-128 + 64 + 32 + 16 + 8 + 2 + 1 = -5

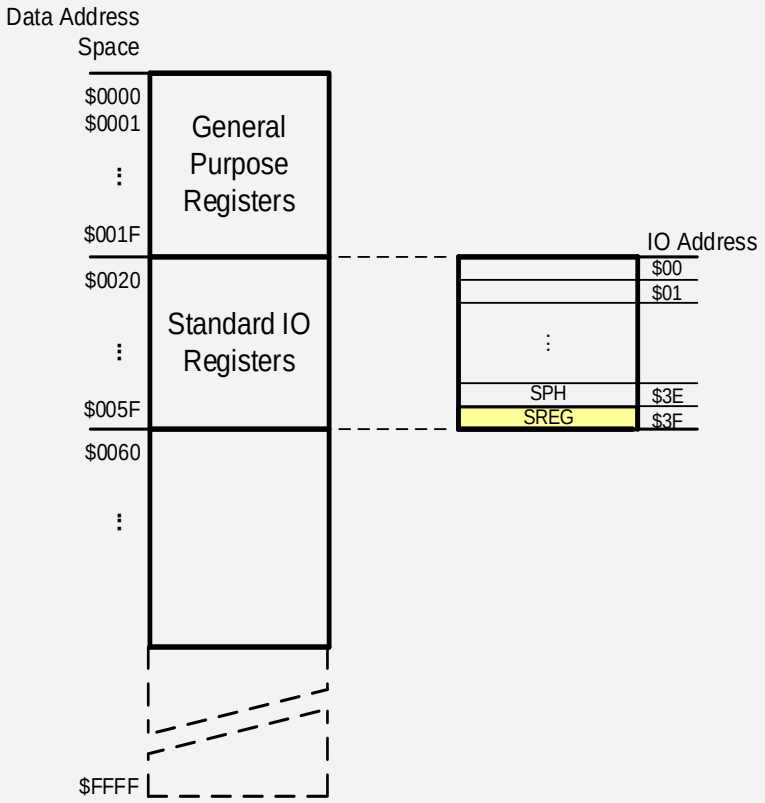
Status Register (SREG)



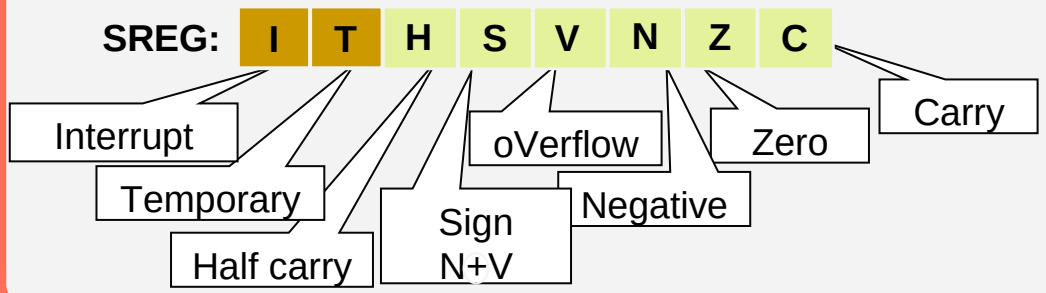
Status Register (SREG)



Status Register (SREG)



Status Register (SREG)



note

Addition of two numbers with sign bits OFF may result in a number with sign bit ON

When is the V flag set?

In 8-bit signed number operations, V is set to 1 if either of the following two conditions occurs:

1. There is a carry from D6 to D7 but no carry out of D7 (C = 0).

2. There is a carry from D7 out (C = 1) but no carry from D6 to D7.

Exception:

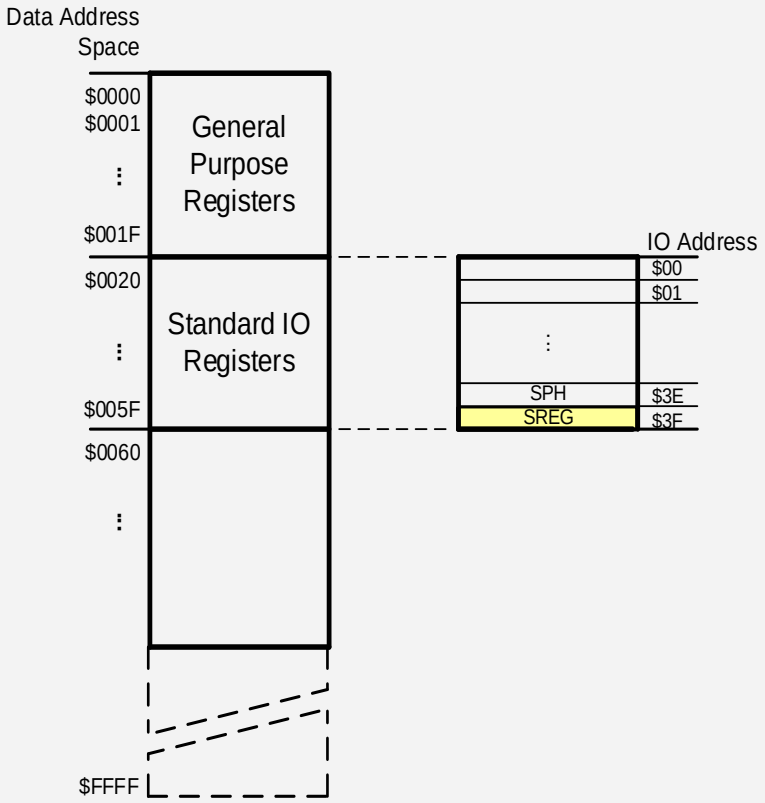
If we ADD two numbers with the same sign, the results should have the same sign too. If we add two numbers with the same sign and the result sign is different, we know that overflow has occurred.

In the AVR the equation of the V flag is as follows:

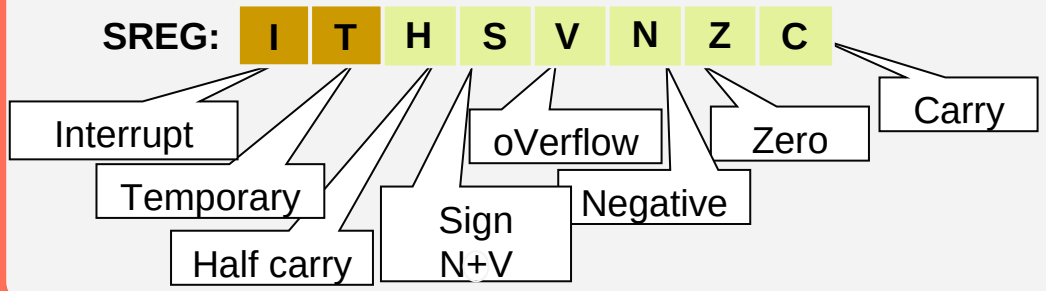
$$V = Rd7 \cdot Rr7 \cdot \overline{R7} + \overline{Rd7} \cdot \overline{Rr7} \cdot R7$$

Rd7 and Rr7 are the 7th bit of the operands
R7 is the 7th bit of the result

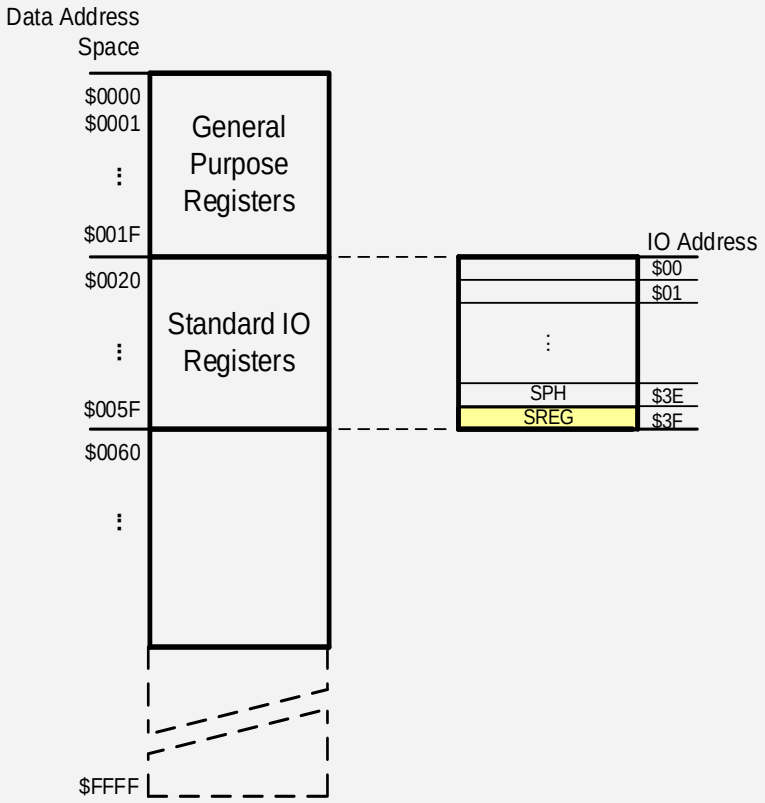
Status Register (SREG)



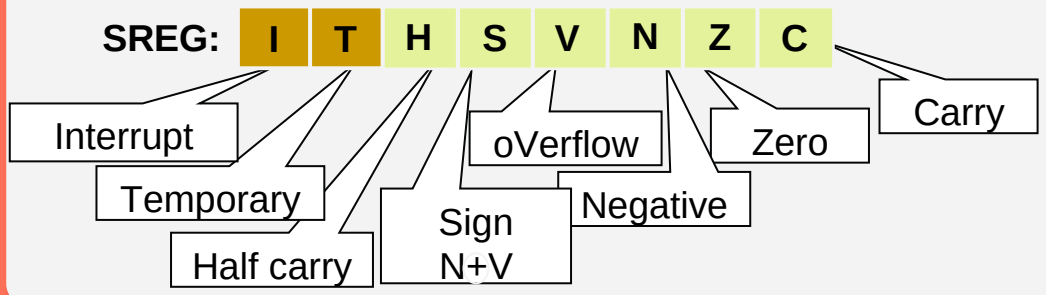
Status Register (SREG)



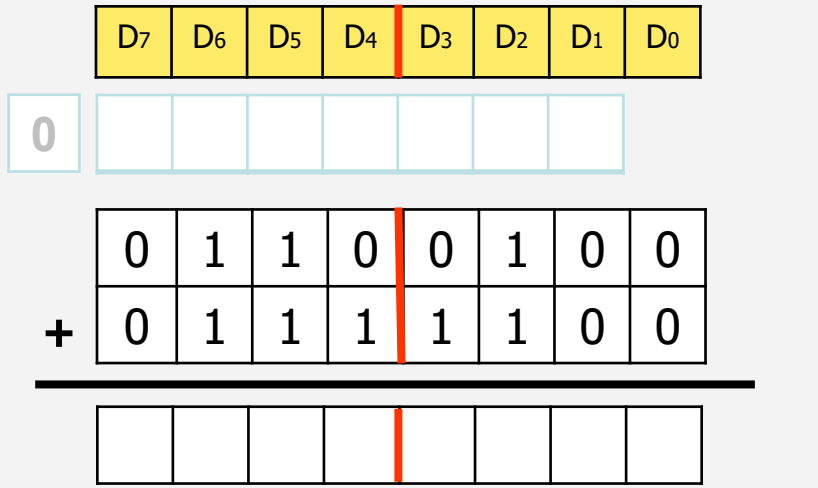
Status Register (SREG)



Status Register (SREG)

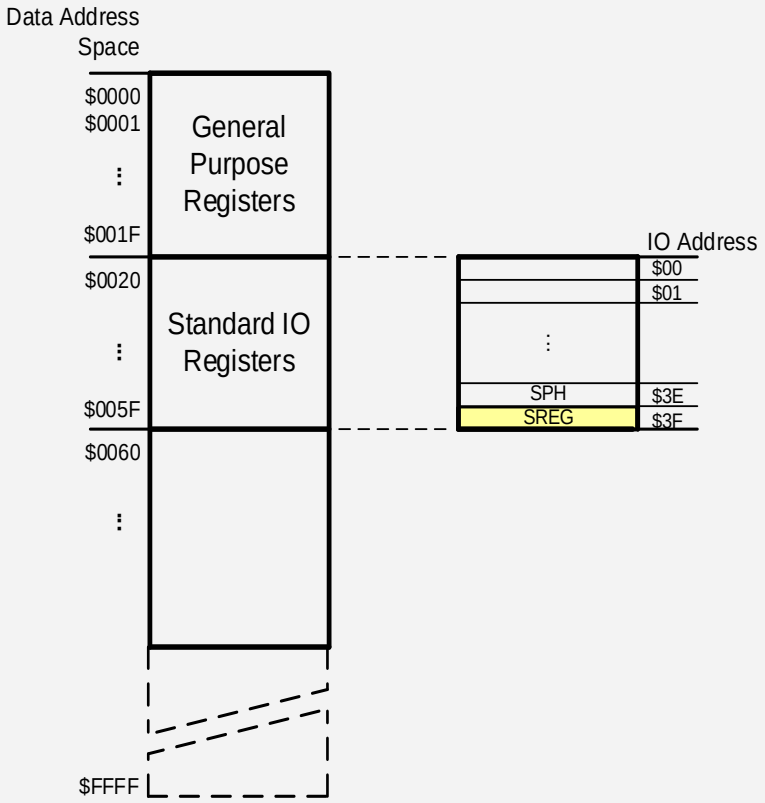


Over Flow and Negative Flag

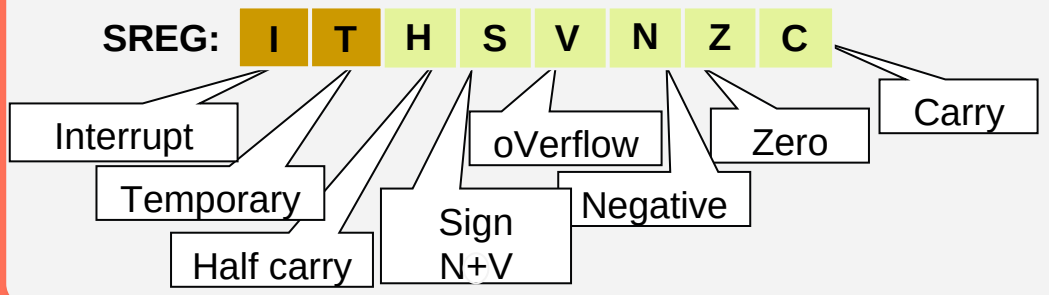


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

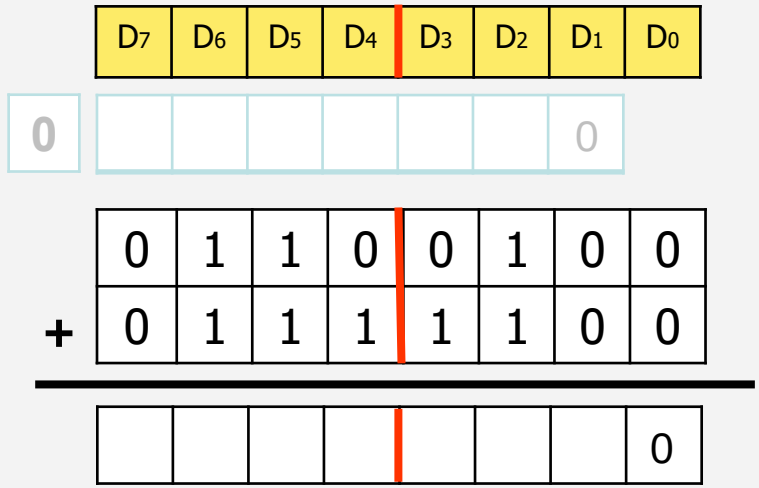
Status Register (SREG)



Status Register (SREG)

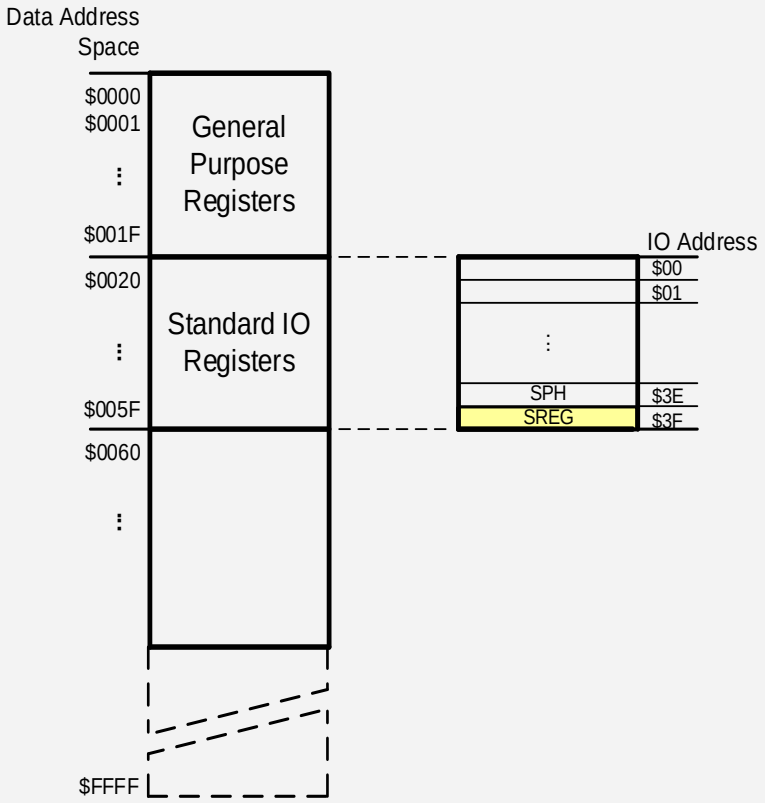


Over Flow and Negative Flag

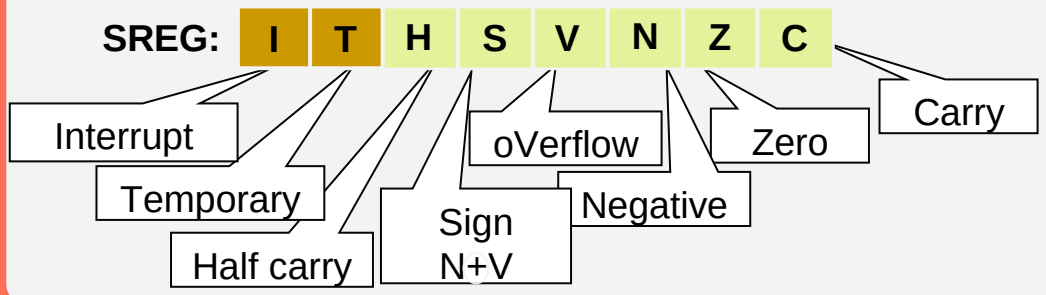


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

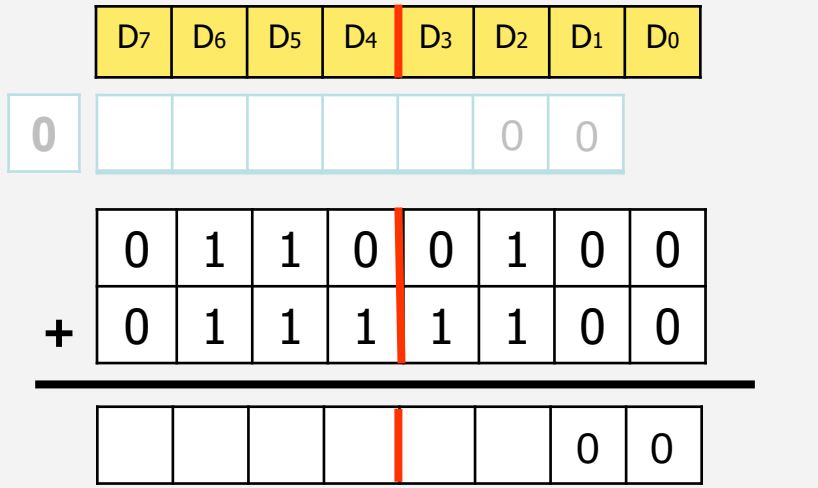
Status Register (SREG)



Status Register (SREG)

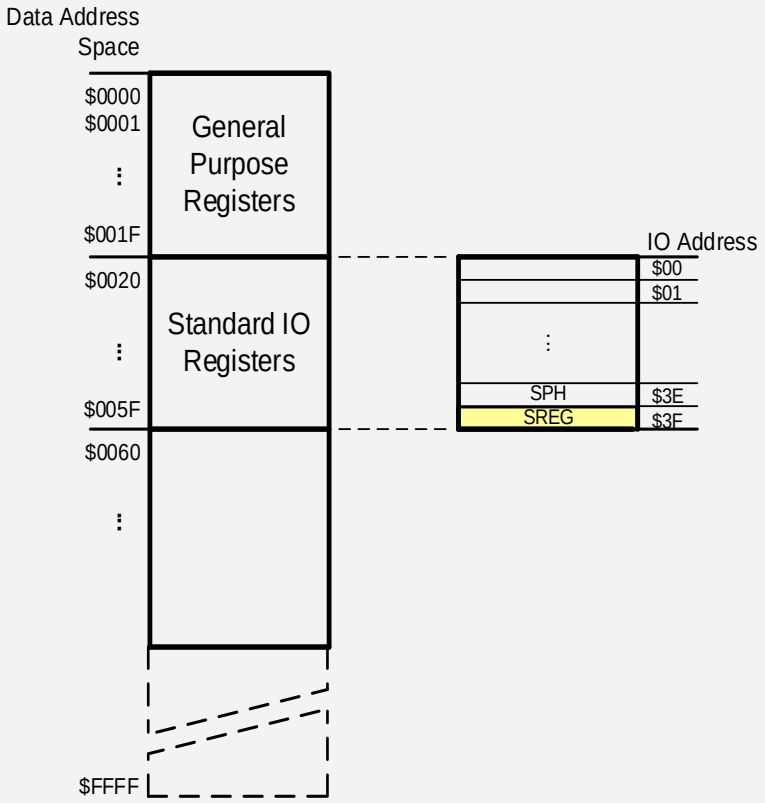


Over Flow and Negative Flag

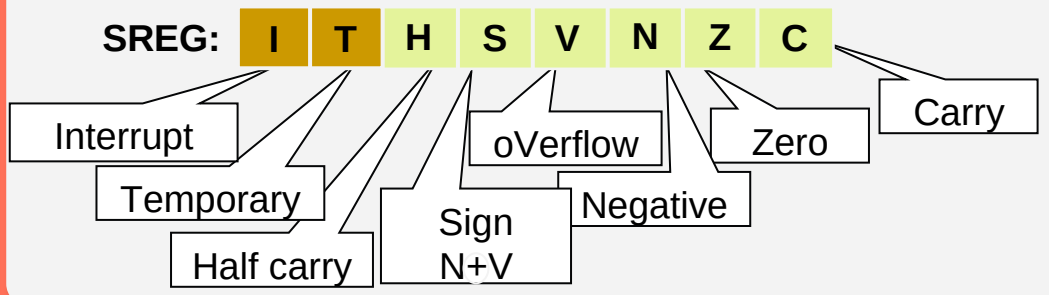


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

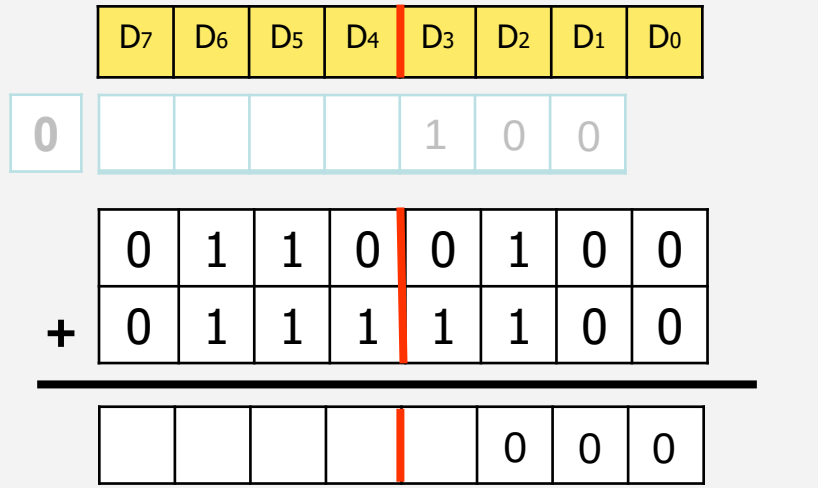
Status Register (SREG)



Status Register (SREG)

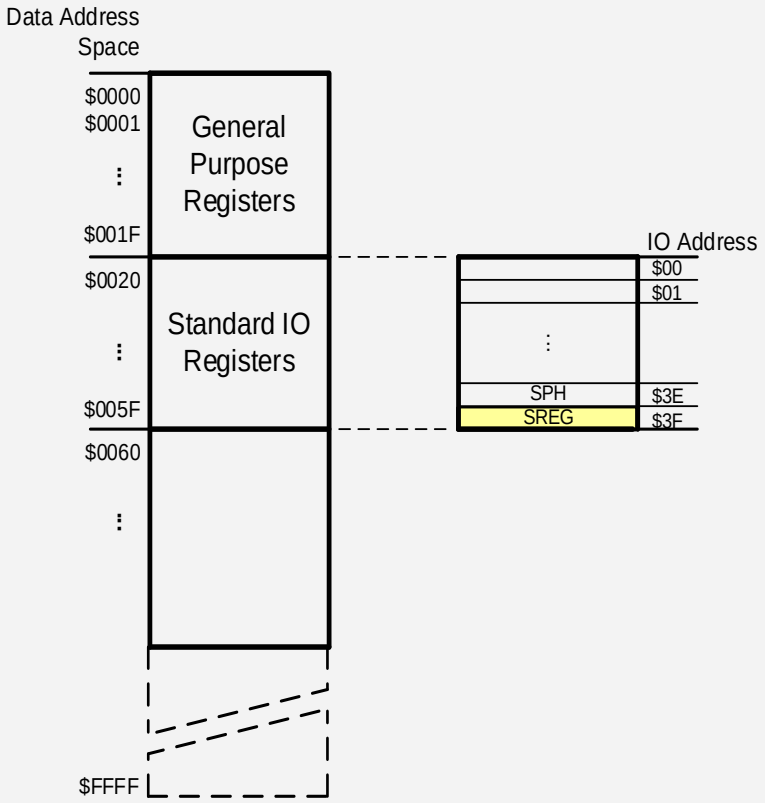


Over Flow and Negative Flag

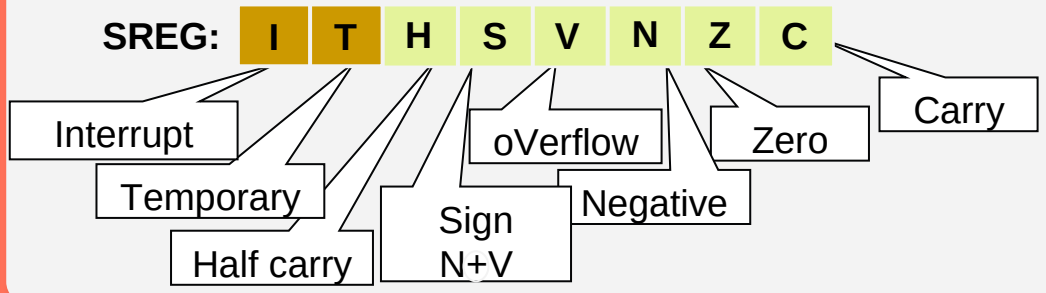


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

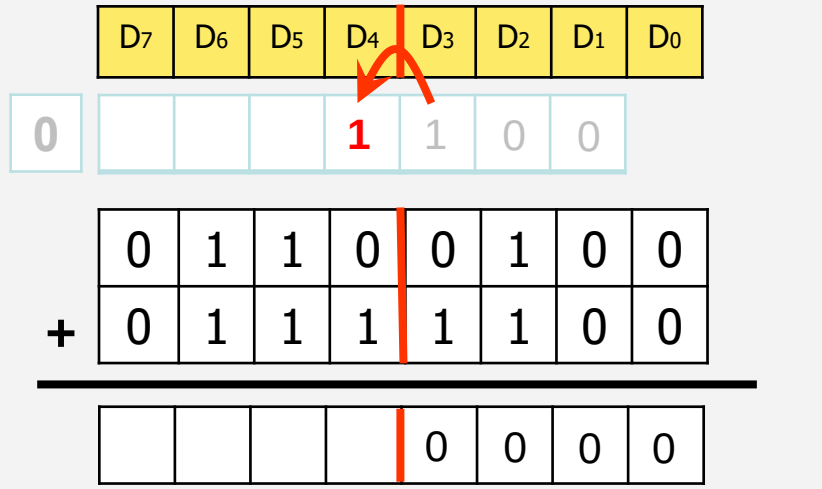
Status Register (SREG)



Status Register (SREG)

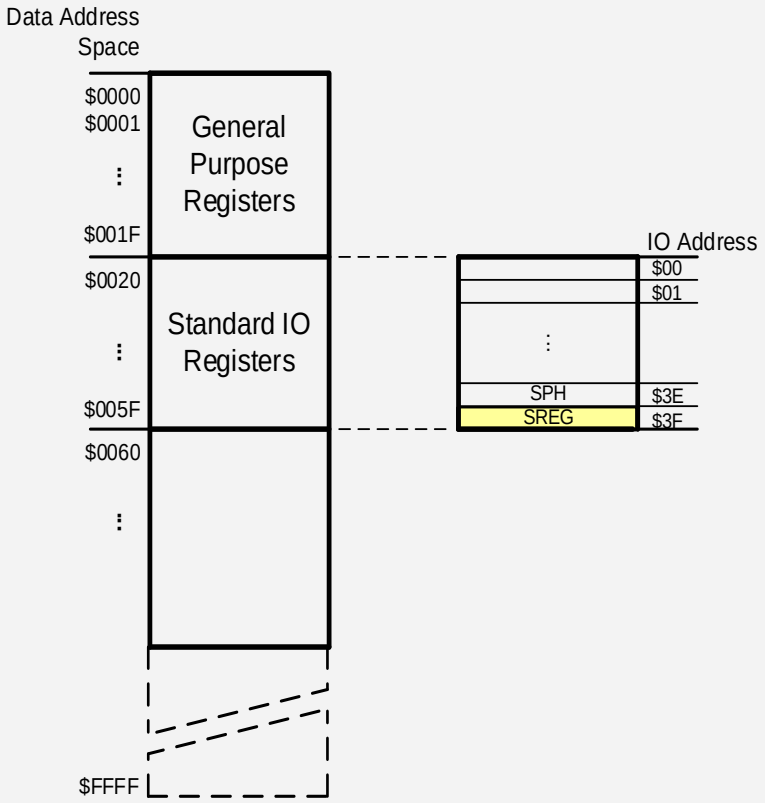


Over Flow and Negative Flag

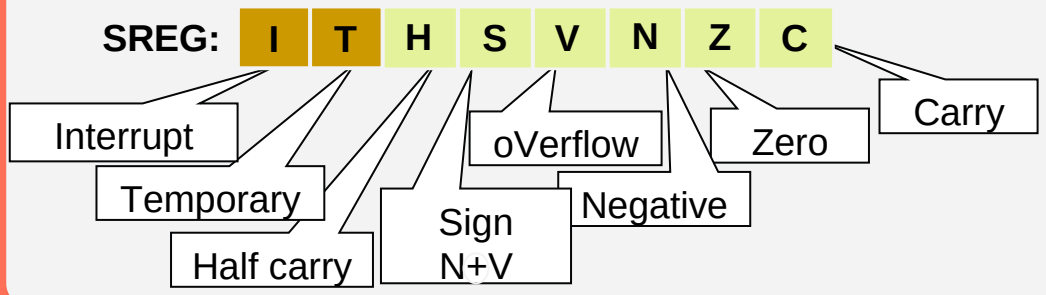


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

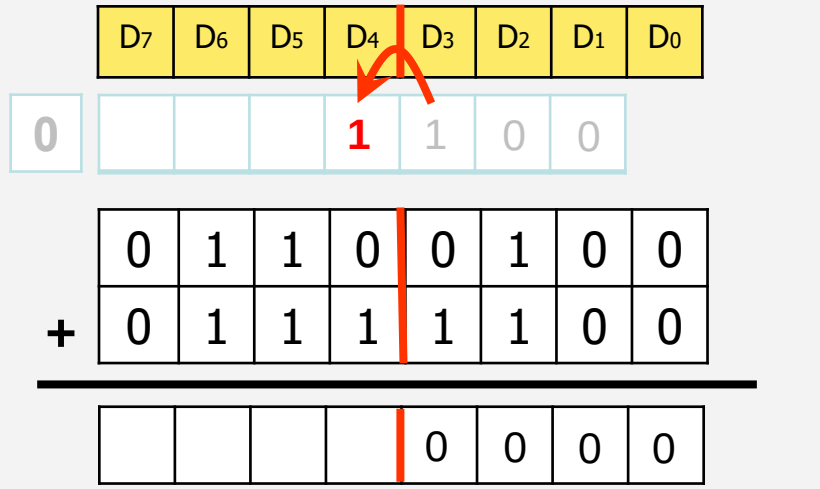
Status Register (SREG)



Status Register (SREG)

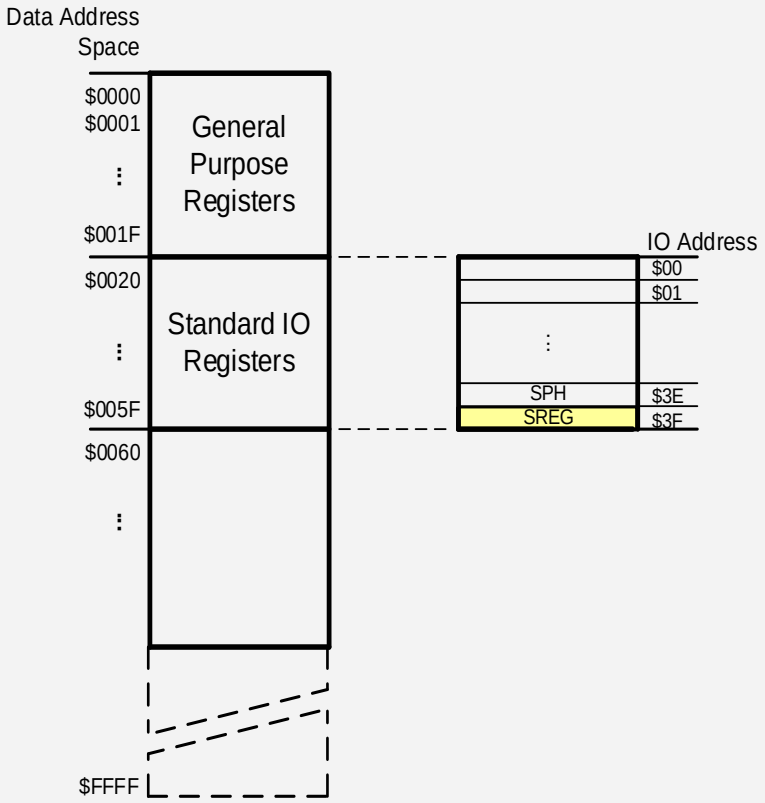


Over Flow and Negative Flag

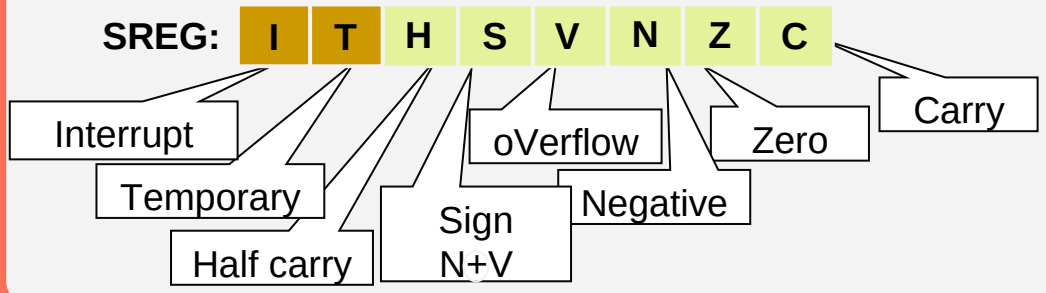


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

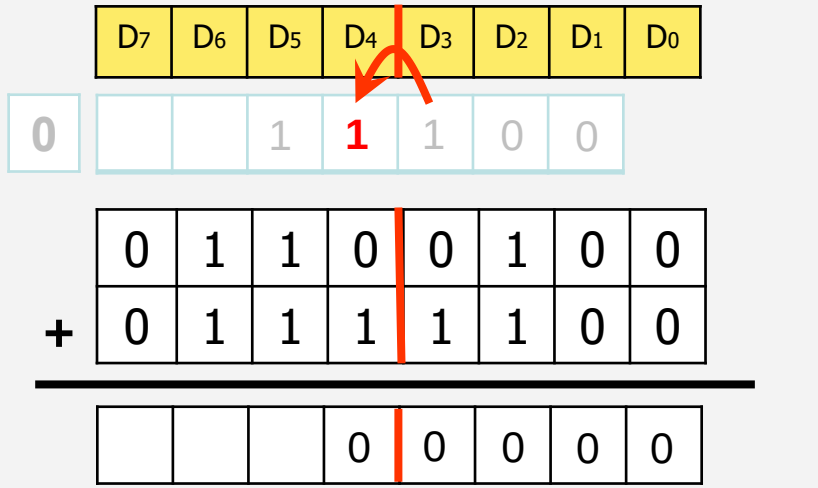
Status Register (SREG)



Status Register (SREG)

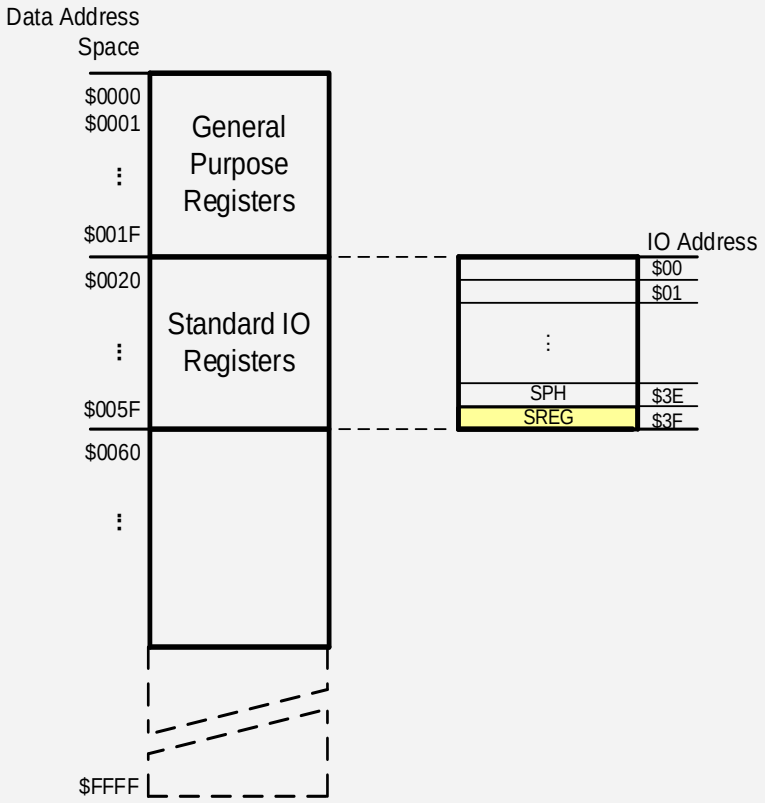


Over Flow and Negative Flag

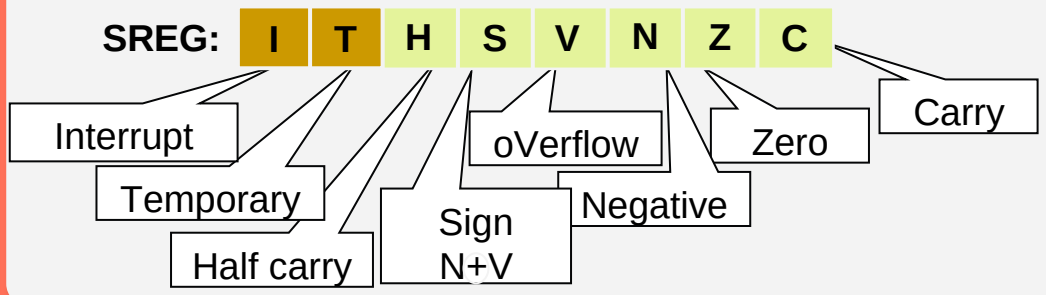


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

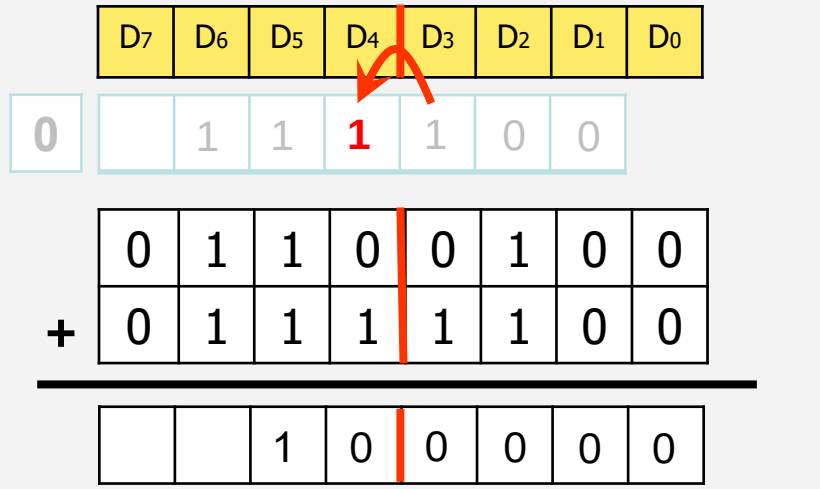
Status Register (SREG)



Status Register (SREG)

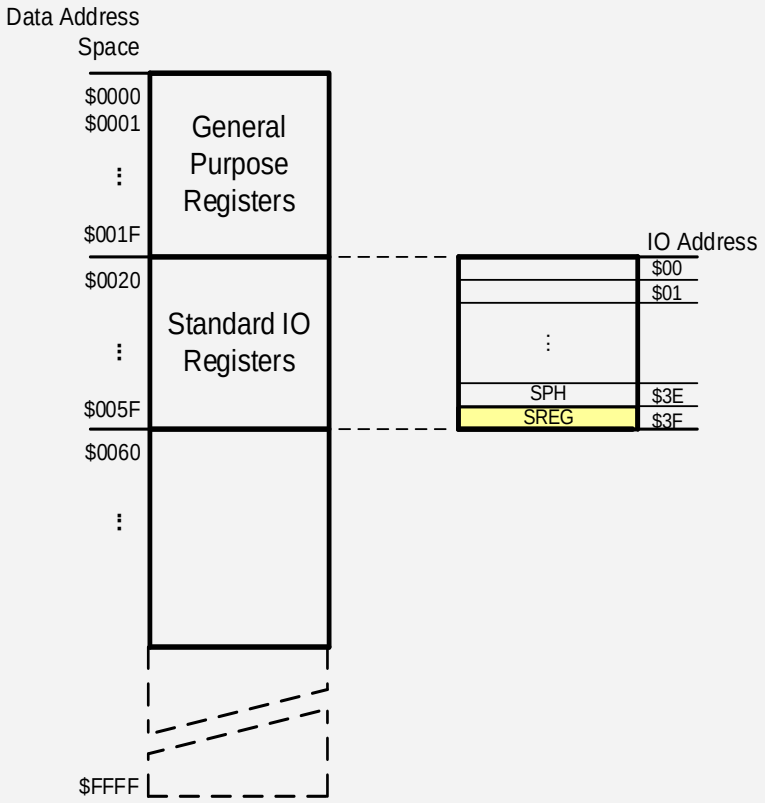


Over Flow and Negative Flag

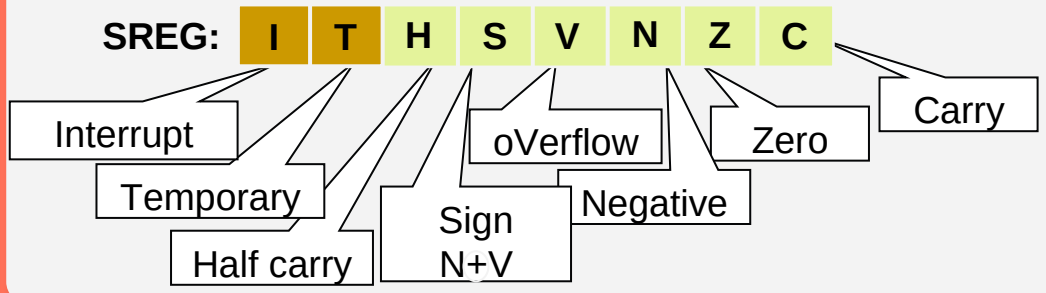


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

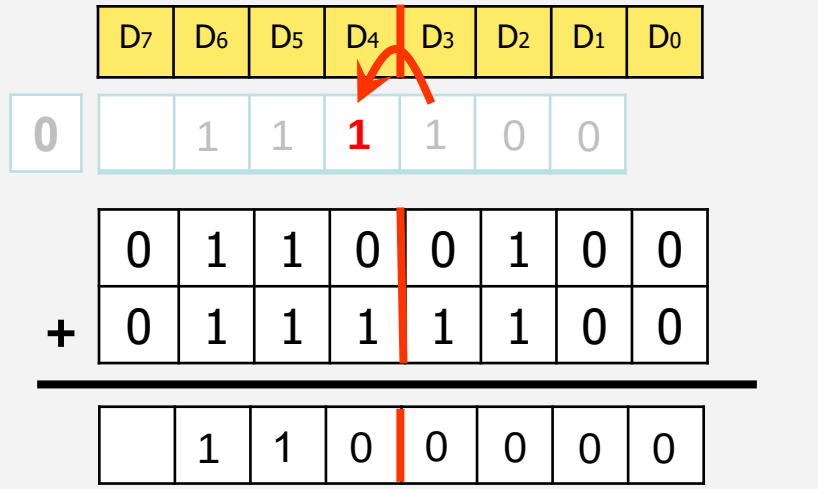
Status Register (SREG)



Status Register (SREG)

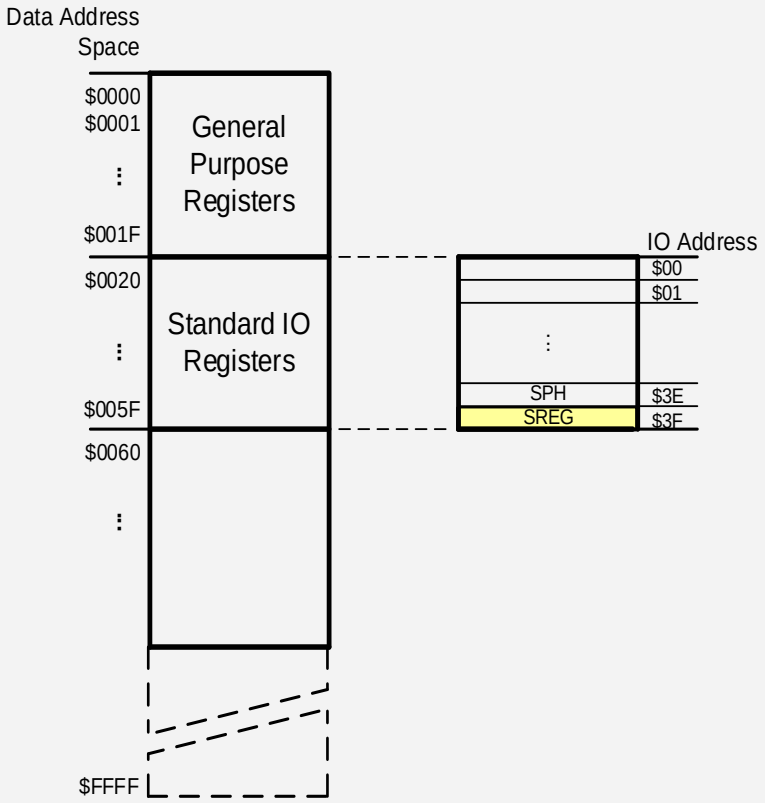


Over Flow and Negative Flag

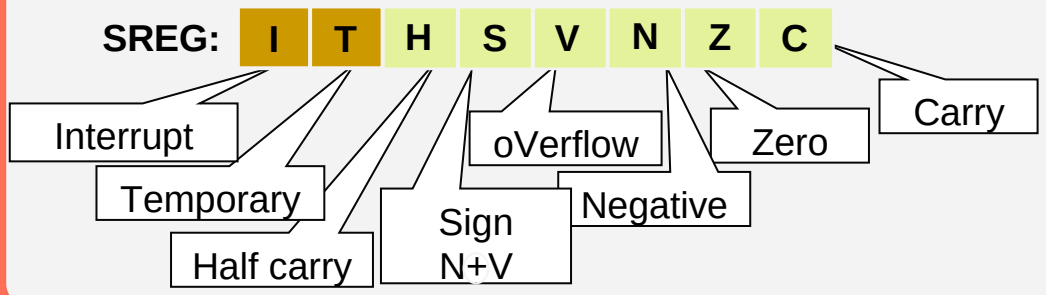


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

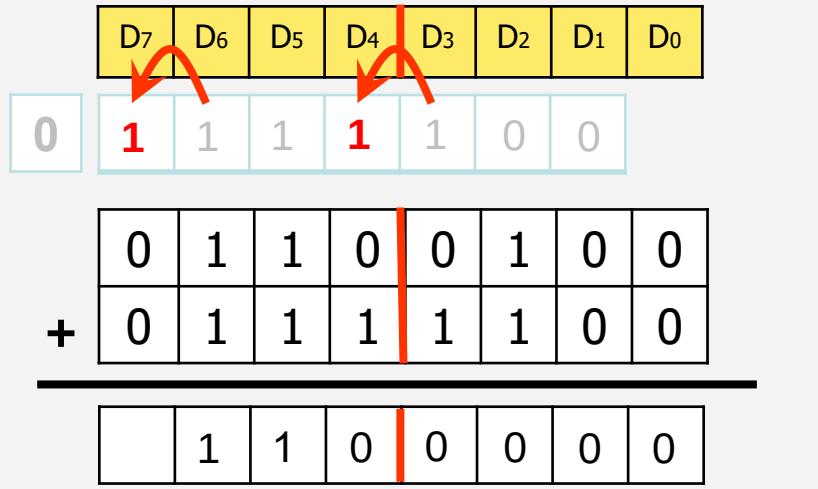
Status Register (SREG)



Status Register (SREG)

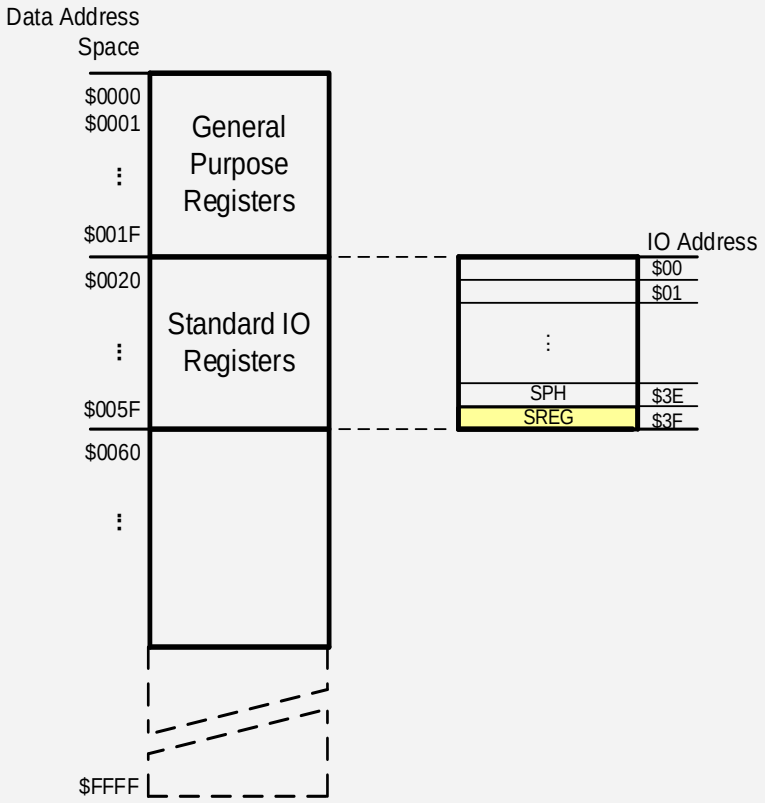


Over Flow and Negative Flag

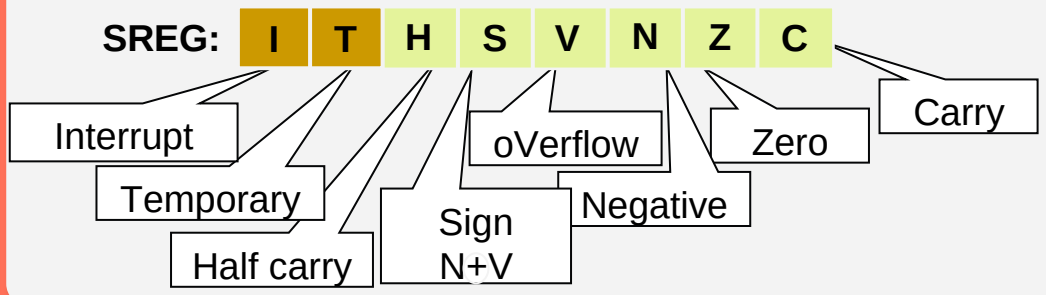


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

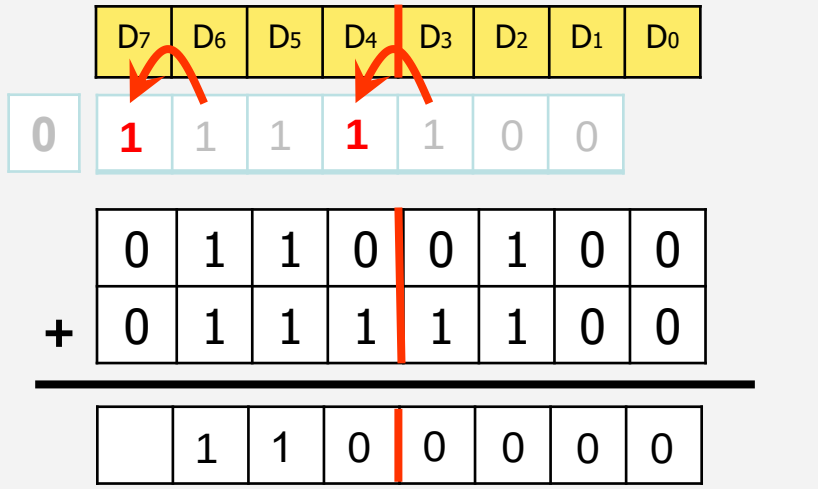
Status Register (SREG)



Status Register (SREG)

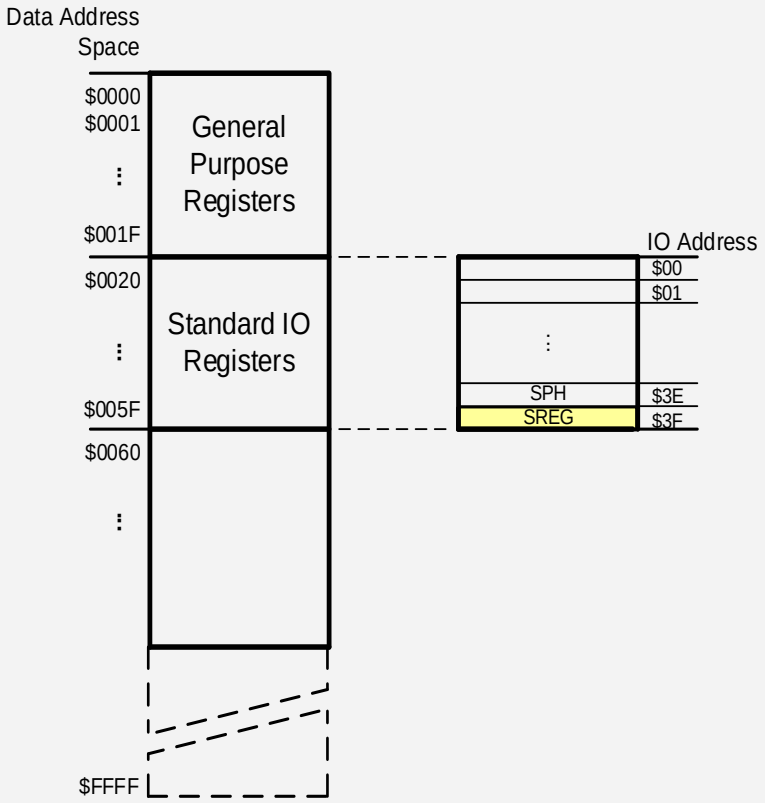


Over Flow and Negative Flag

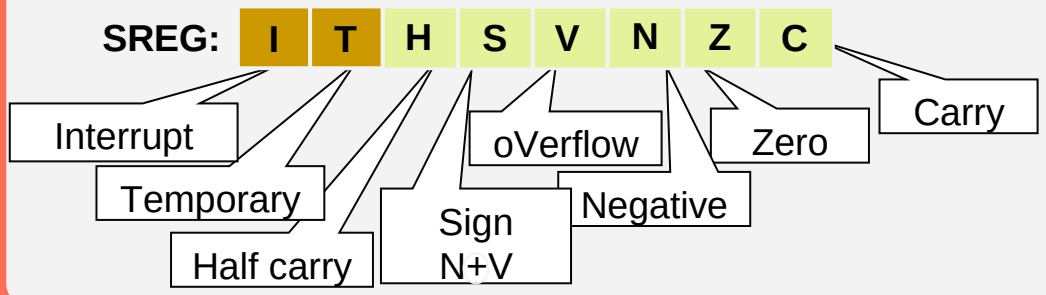


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	1	0	0	0

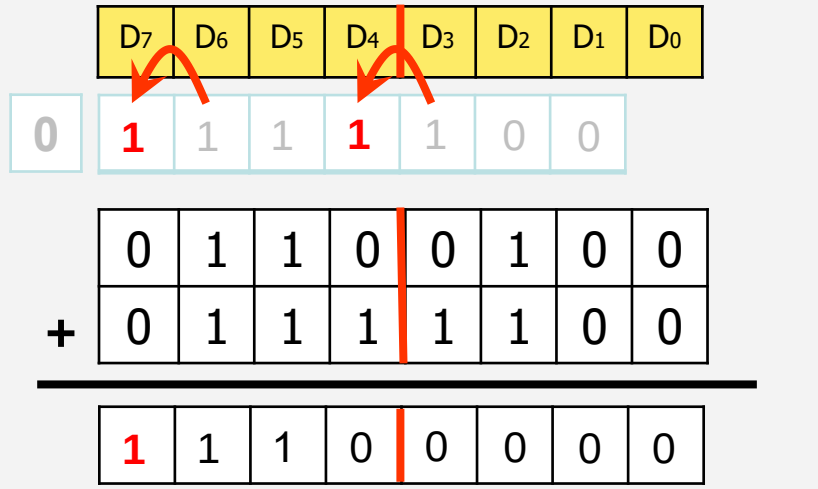
Status Register (SREG)



Status Register (SREG)

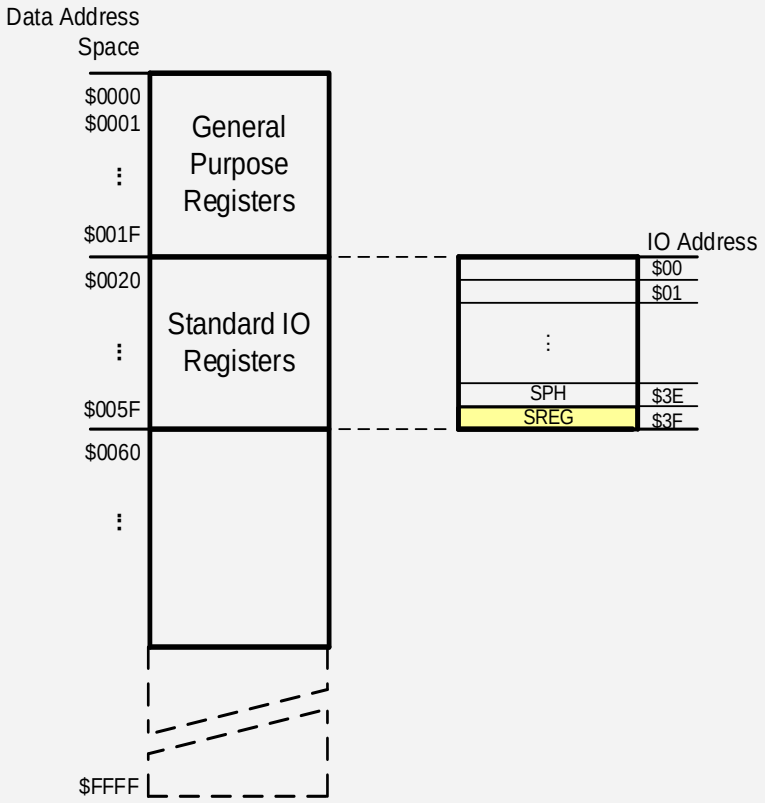


Over Flow and Negative Flag

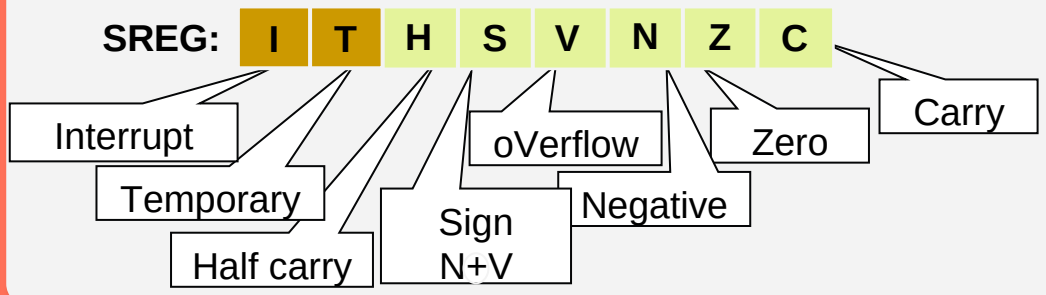


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	1	0	0	0

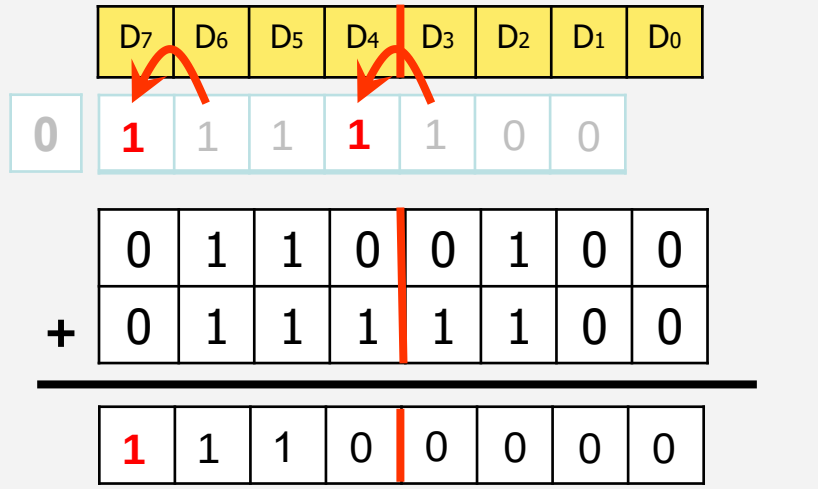
Status Register (SREG)



Status Register (SREG)

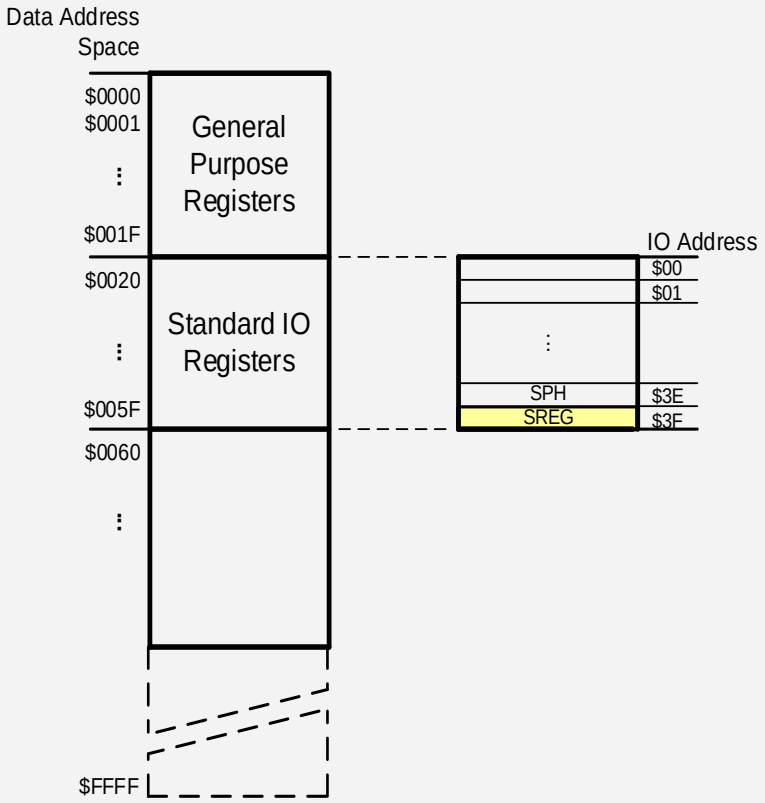


Over Flow and Negative Flag

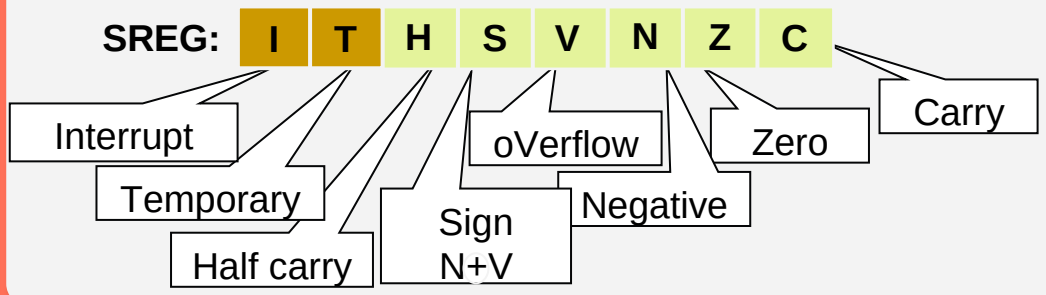


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	1	1	0	0

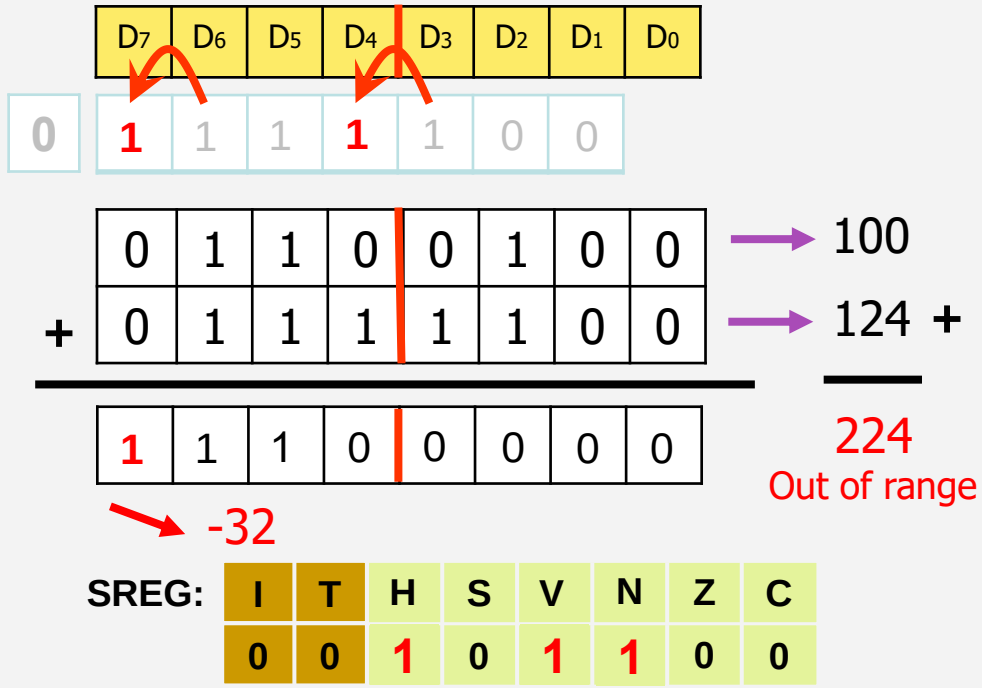
Status Register (SREG)



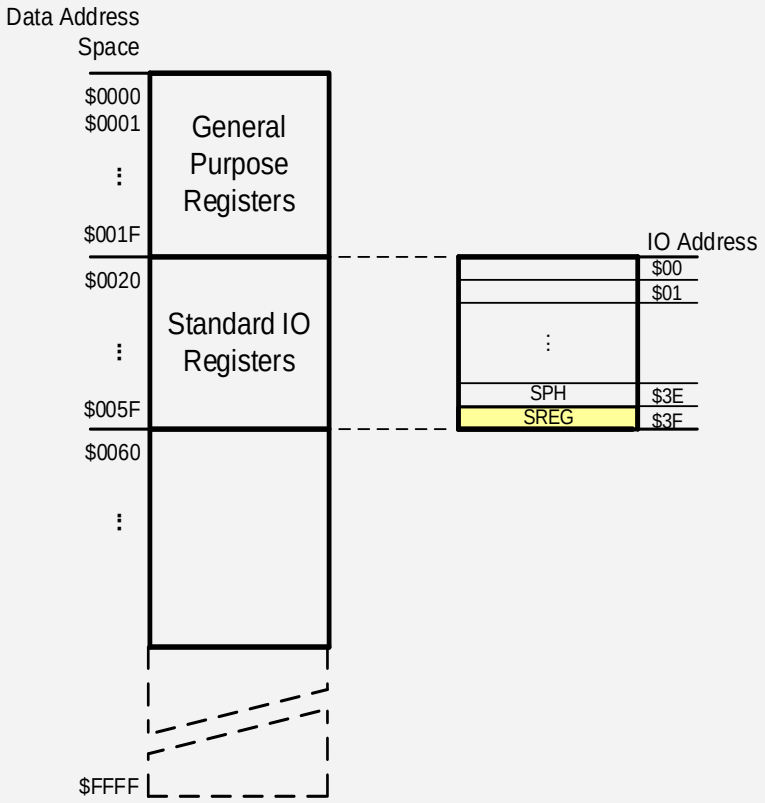
Status Register (SREG)



Over Flow and Negative Flag



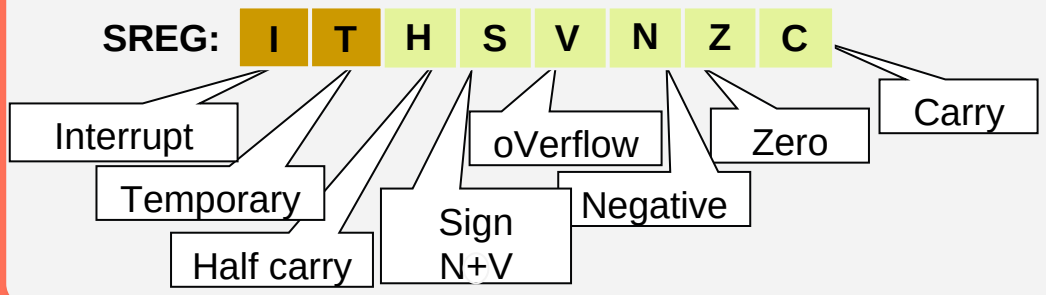
Status Register (SREG)



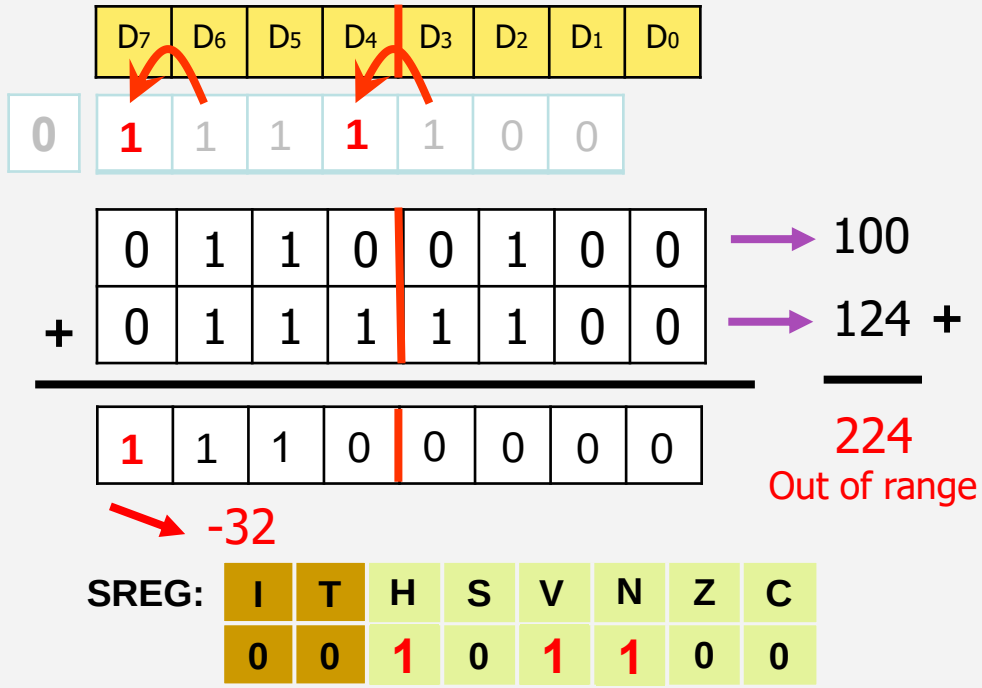
note

Addition of two numbers with sign bits OFF may result in a number with sign bit ON

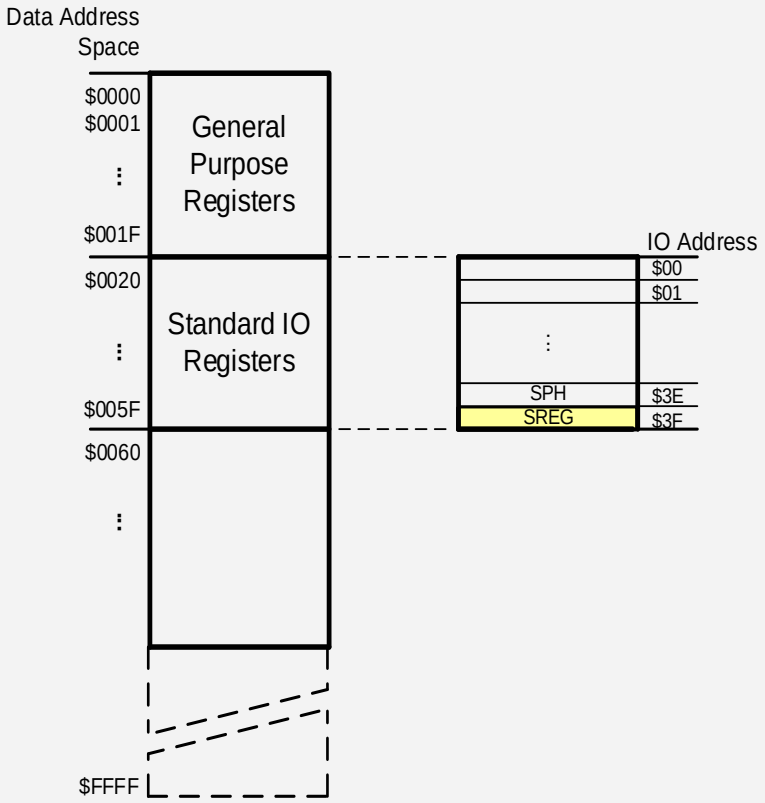
Status Register (SREG)



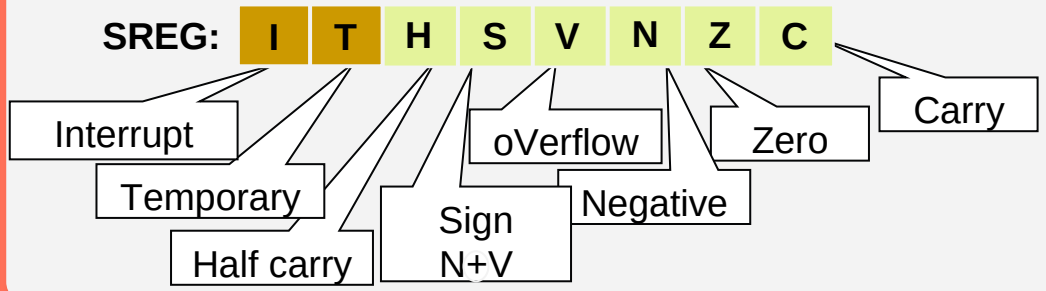
Over Flow and Negative Flag



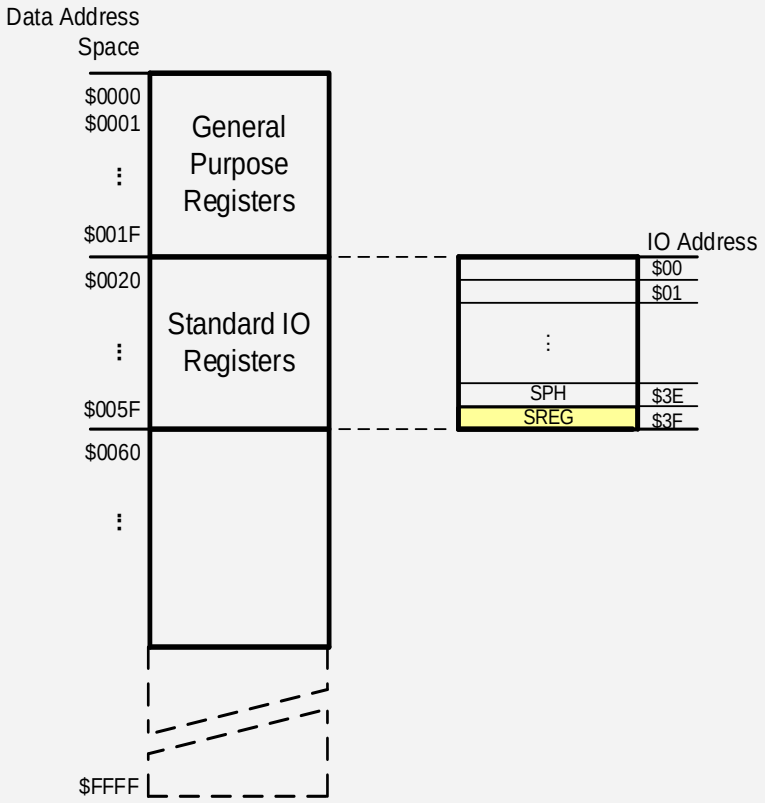
Status Register (SREG)



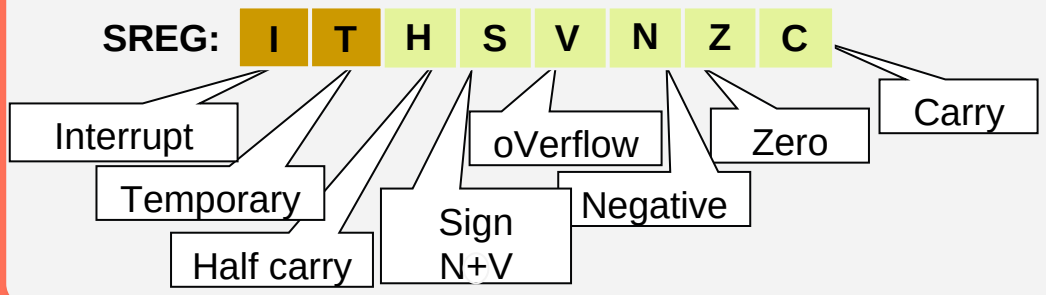
Status Register (SREG)



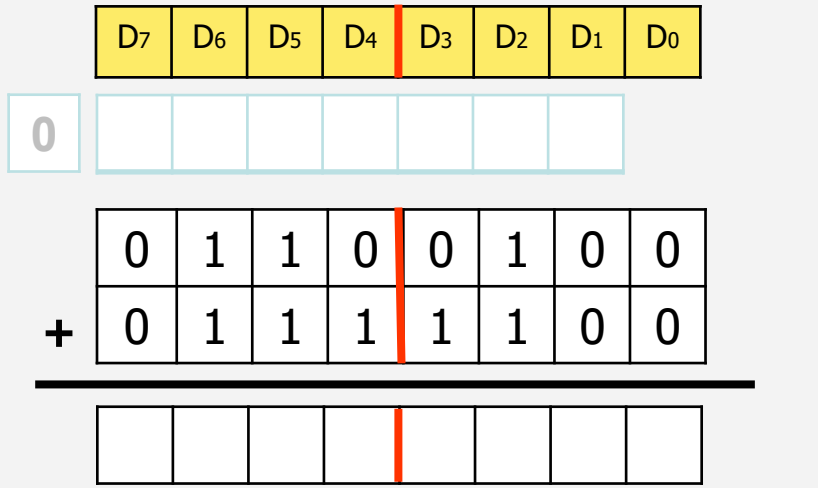
Status Register (SREG)



Status Register (SREG)

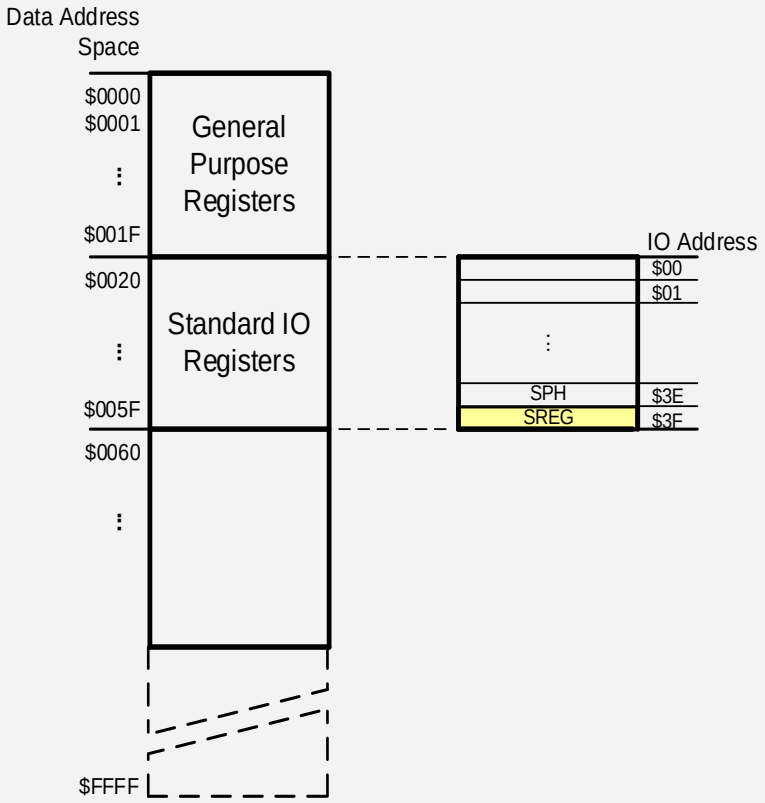


Over Flow and Negative Flag

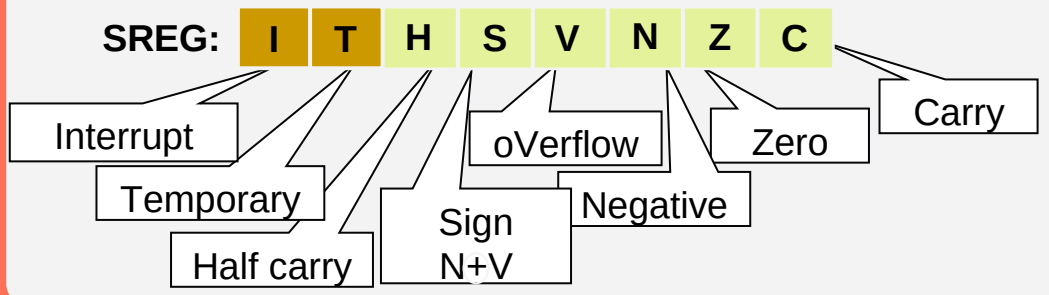


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

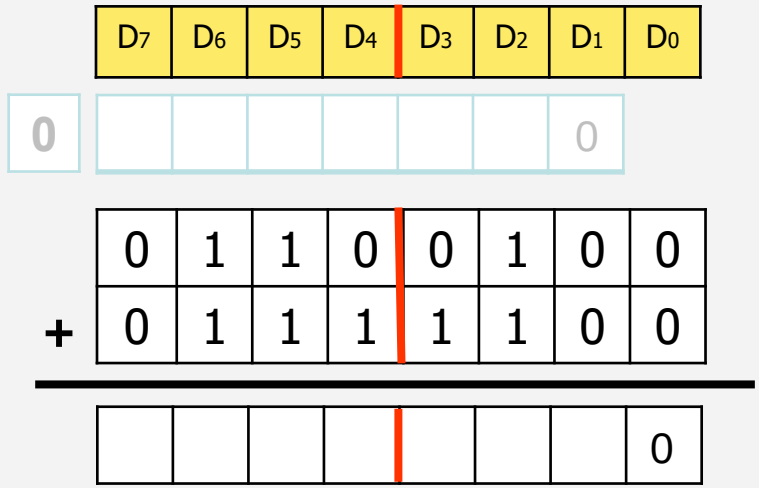
Status Register (SREG)



Status Register (SREG)

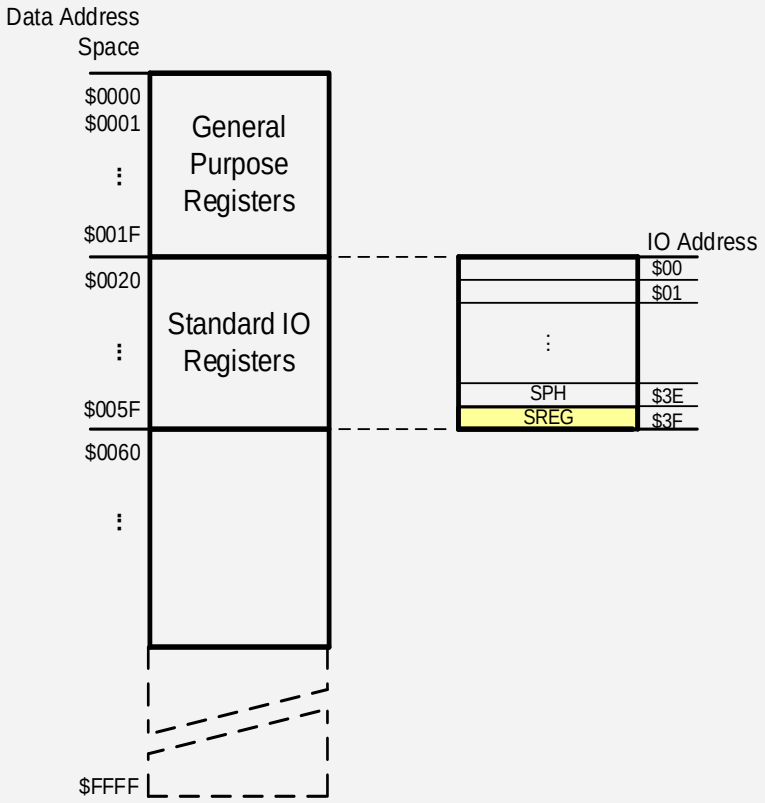


Over Flow and Negative Flag

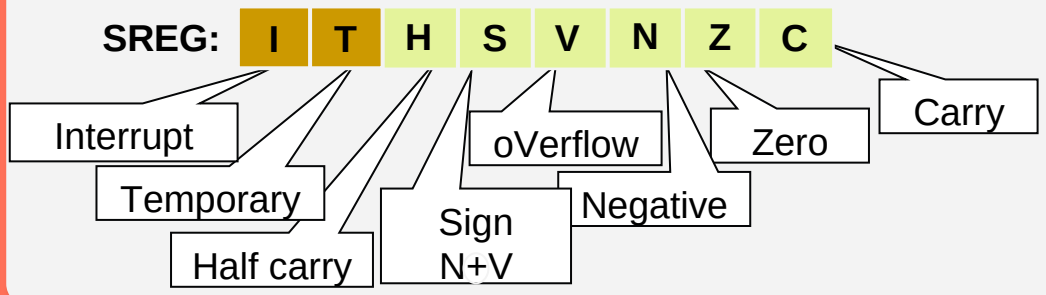


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

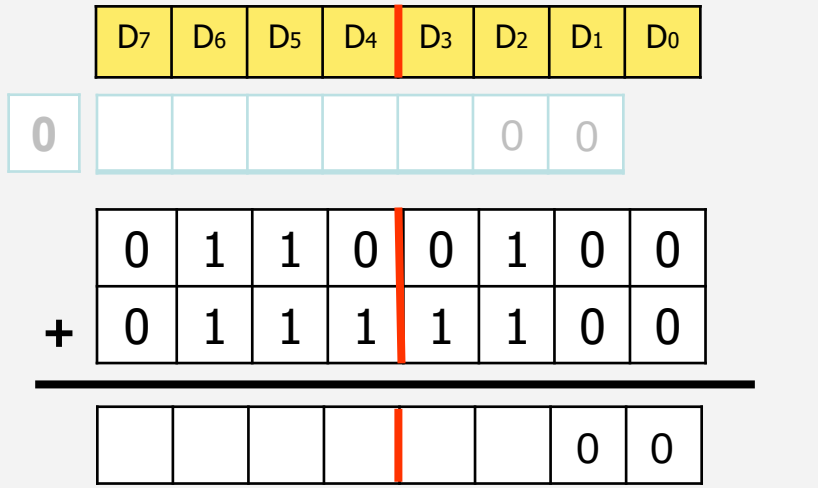
Status Register (SREG)



Status Register (SREG)

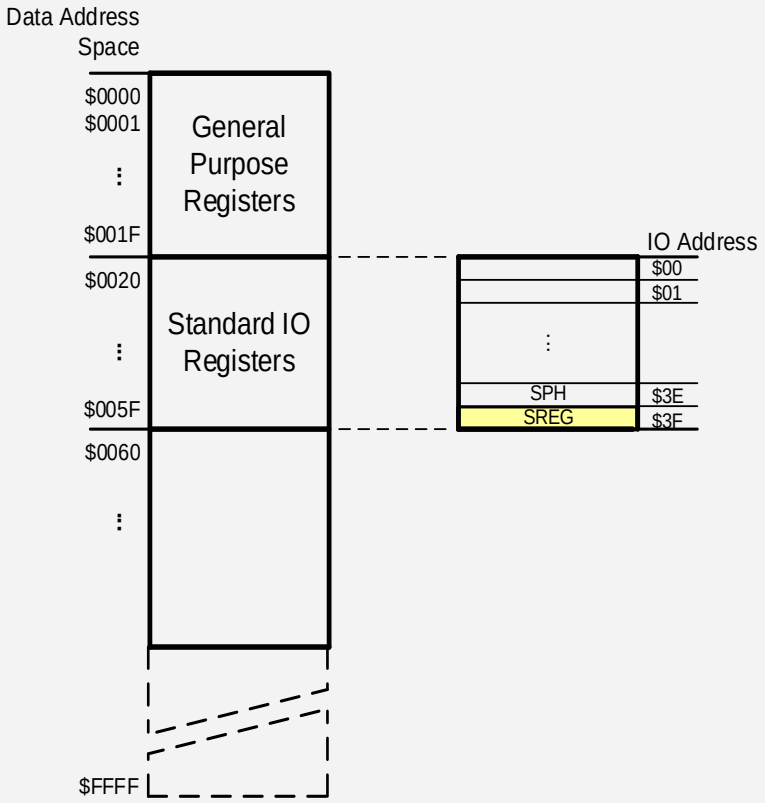


Over Flow and Negative Flag

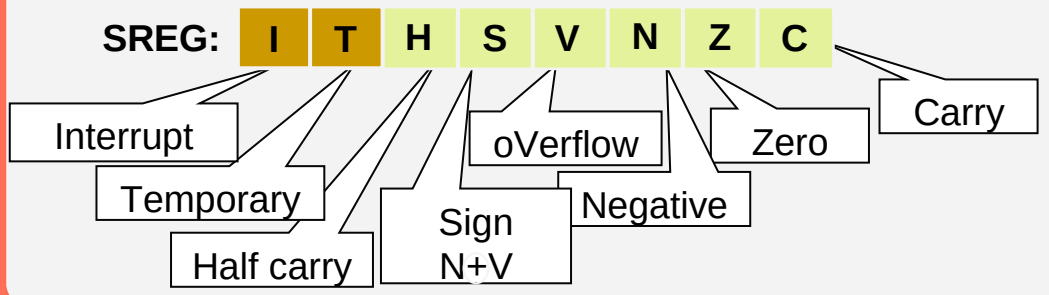


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

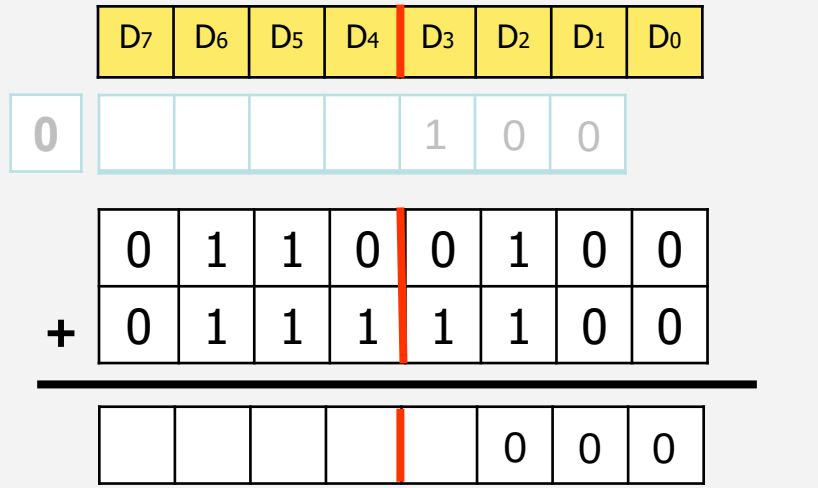
Status Register (SREG)



Status Register (SREG)

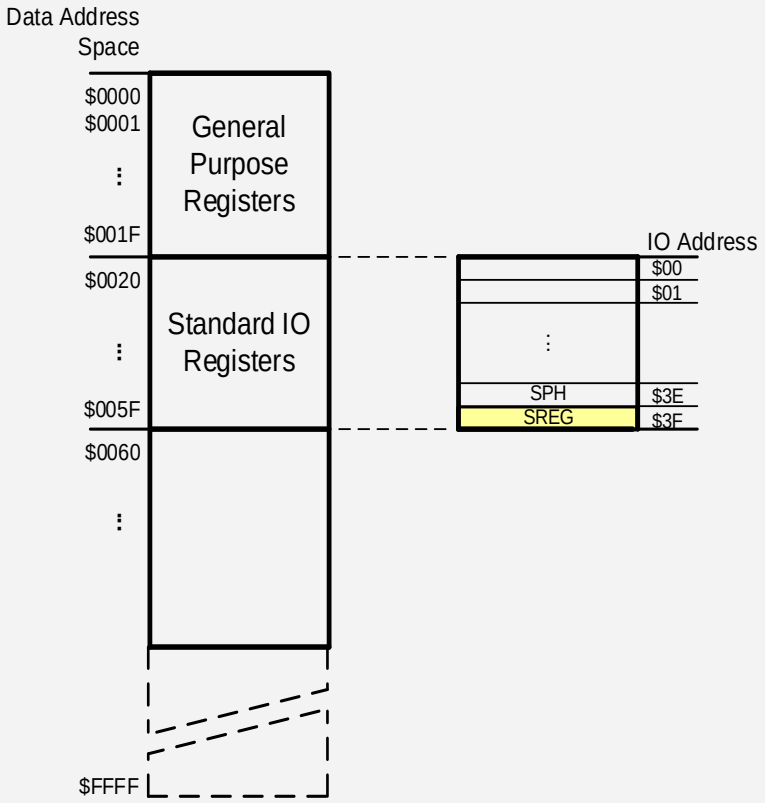


Over Flow and Negative Flag

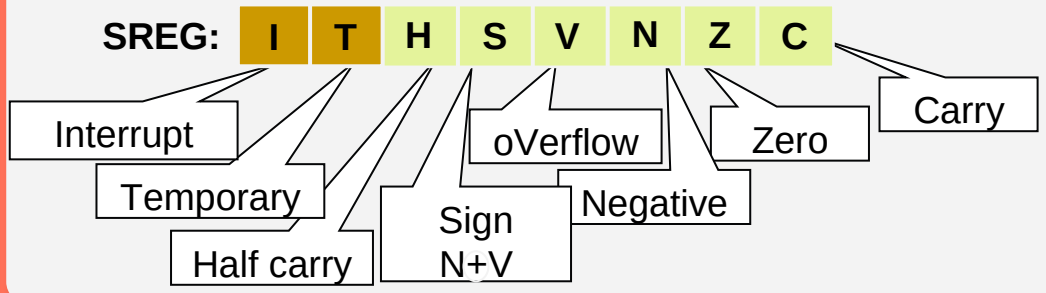


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

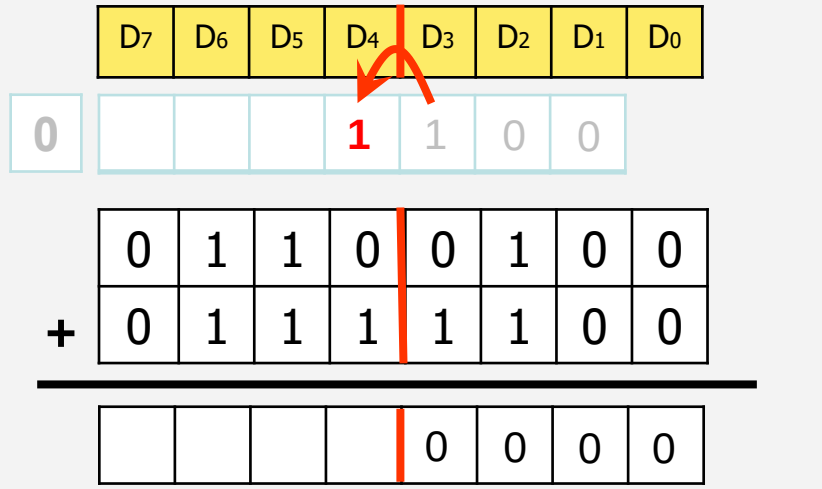
Status Register (SREG)



Status Register (SREG)

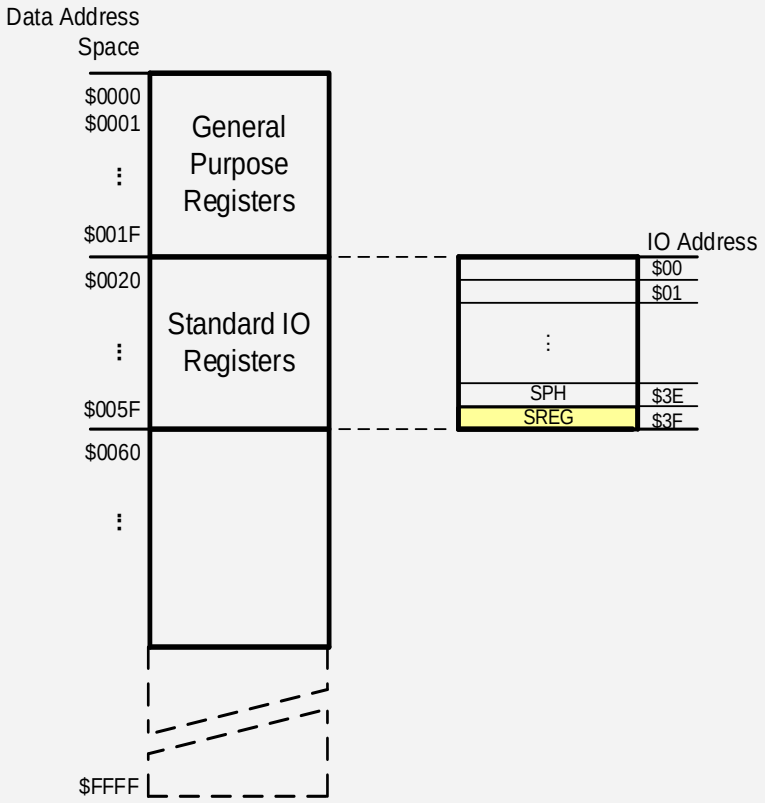


Over Flow and Negative Flag

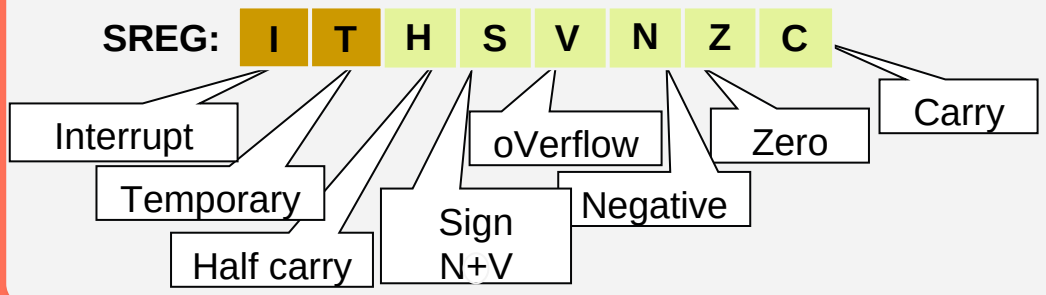


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

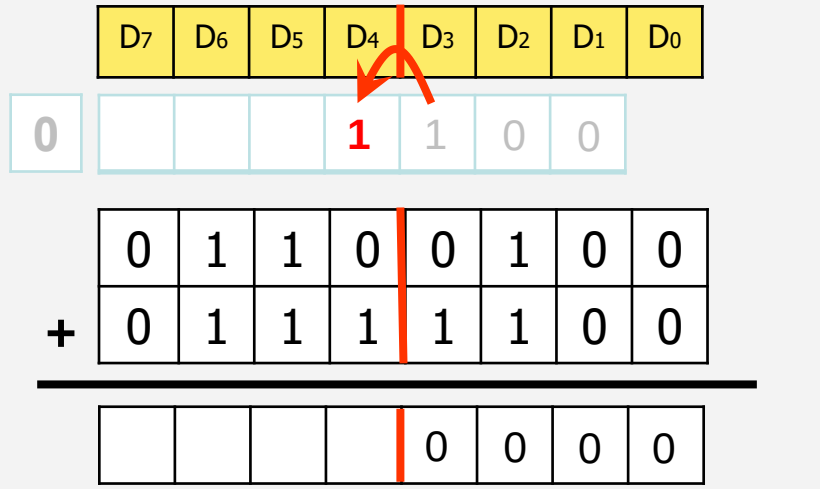
Status Register (SREG)



Status Register (SREG)

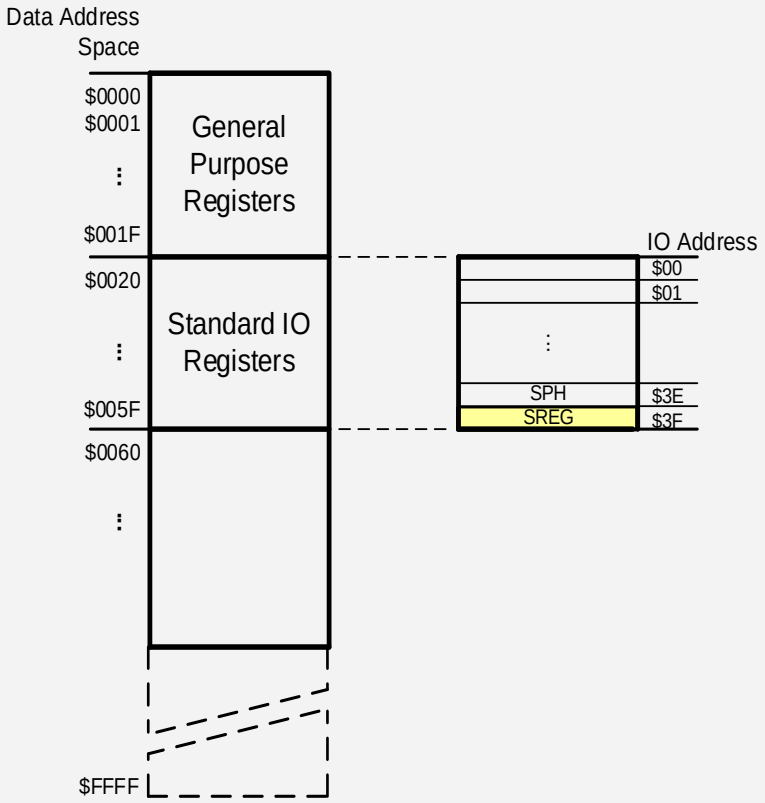


Over Flow and Negative Flag

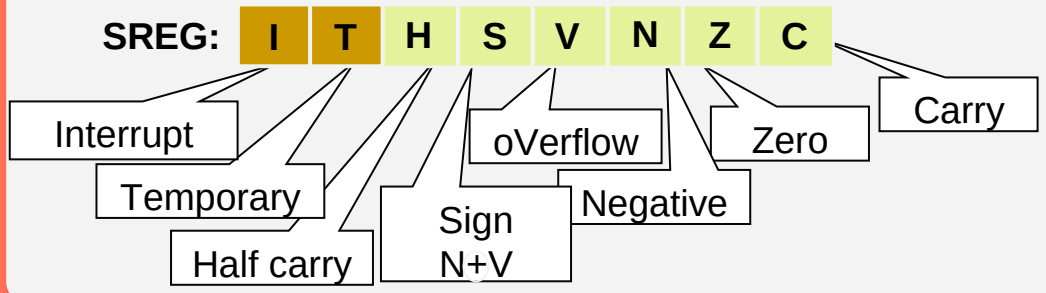


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

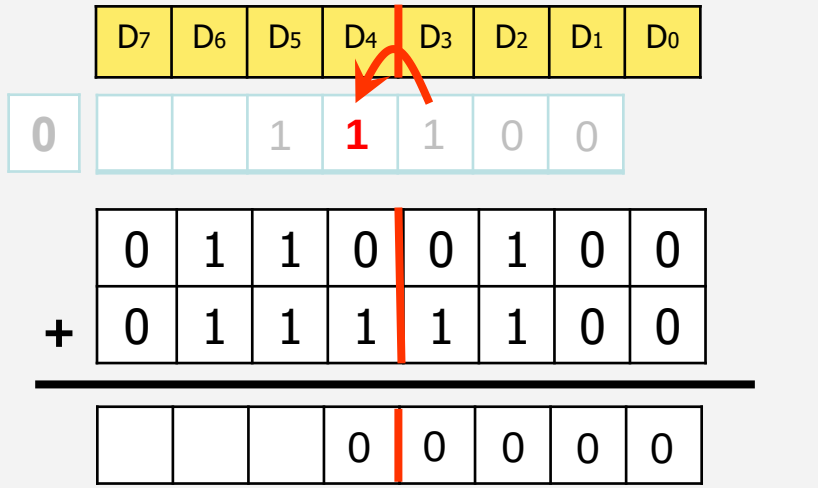
Status Register (SREG)



Status Register (SREG)

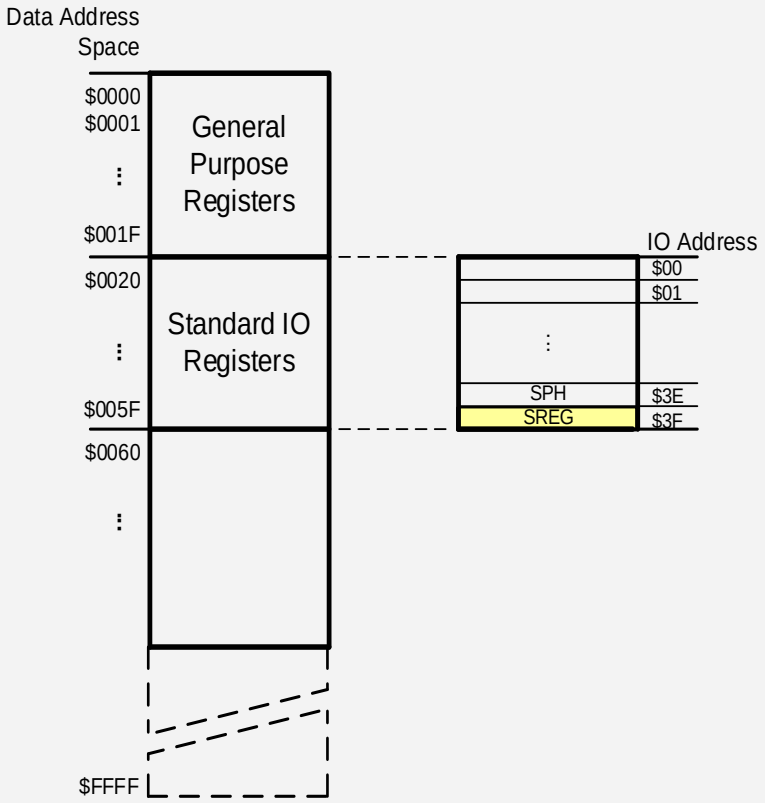


Over Flow and Negative Flag

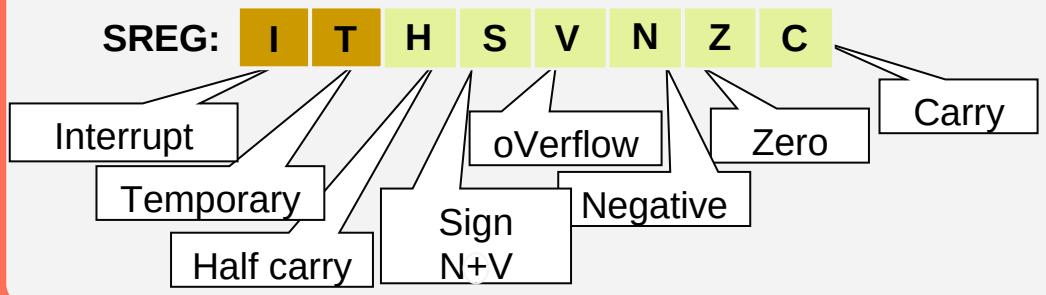


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

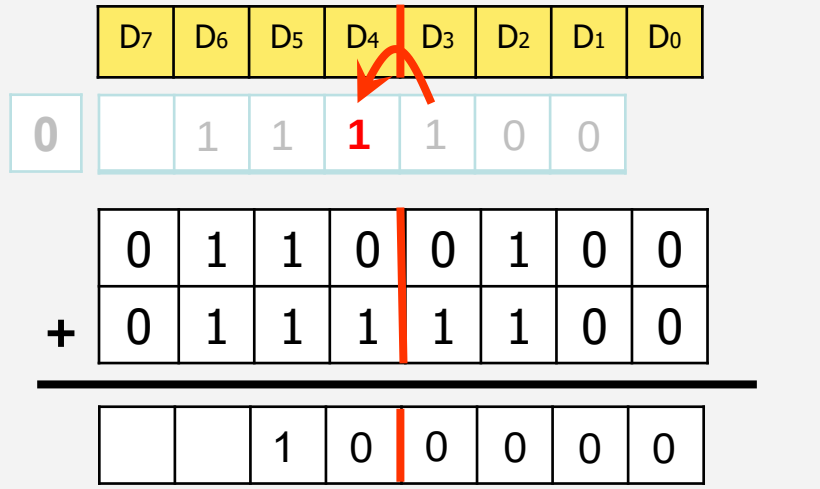
Status Register (SREG)



Status Register (SREG)

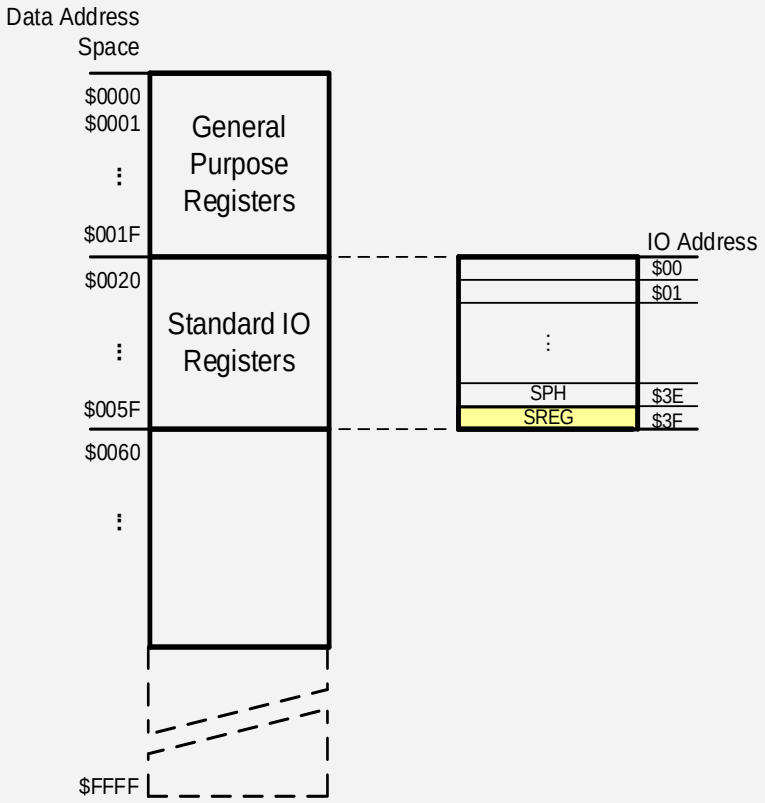


Over Flow and Negative Flag

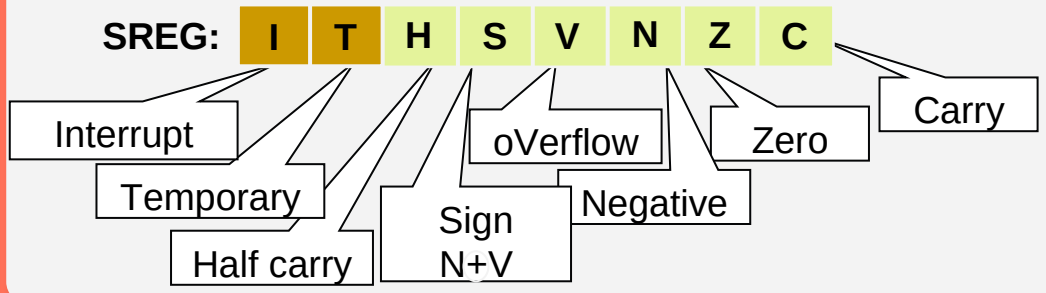


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

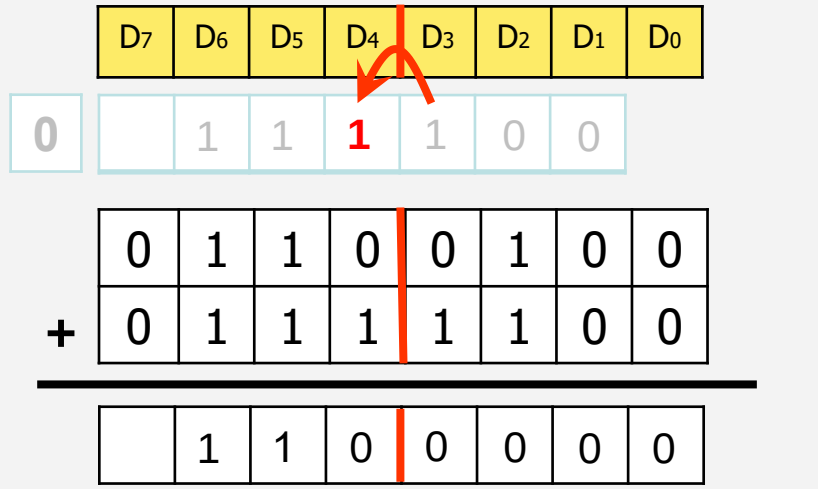
Status Register (SREG)



Status Register (SREG)

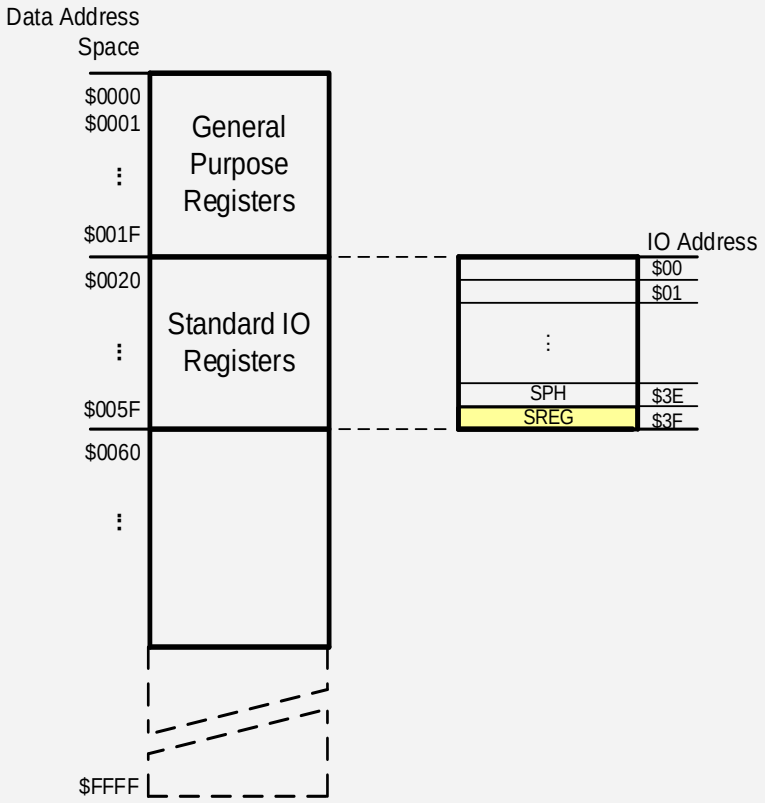


Over Flow and Negative Flag

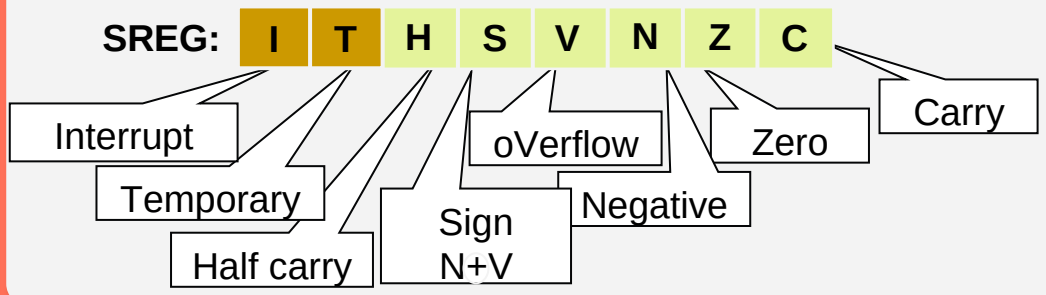


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

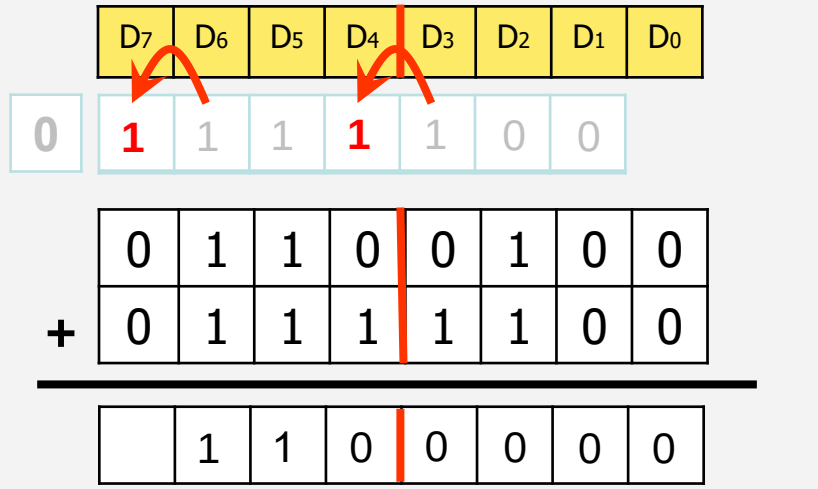
Status Register (SREG)



Status Register (SREG)

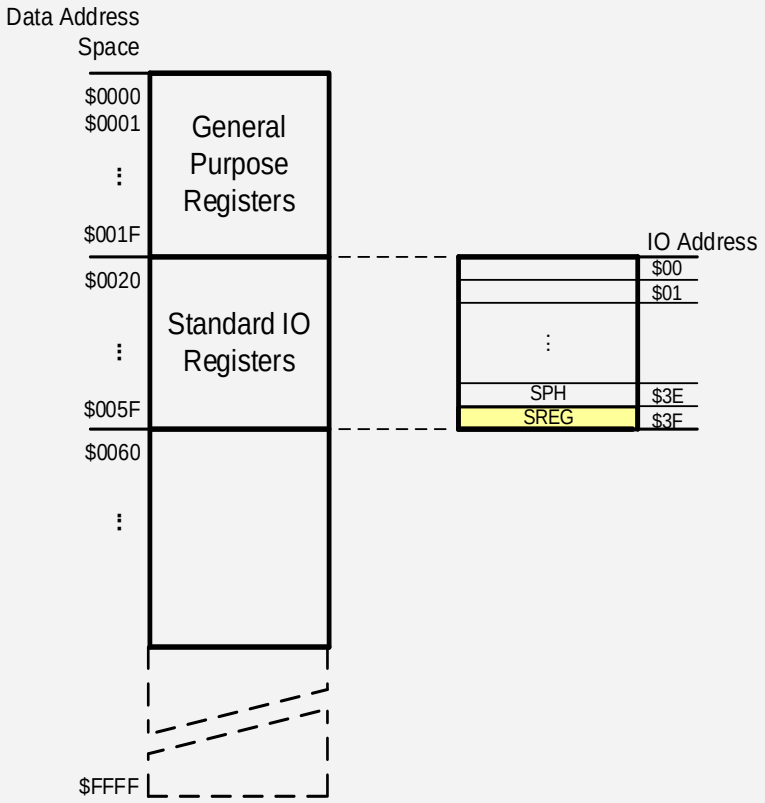


Over Flow and Negative Flag

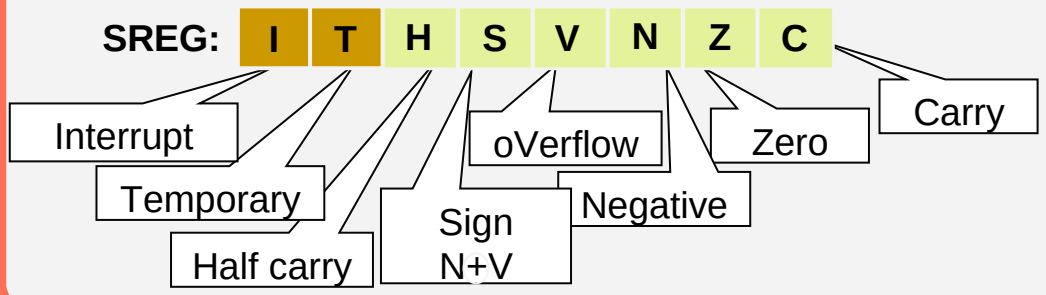


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

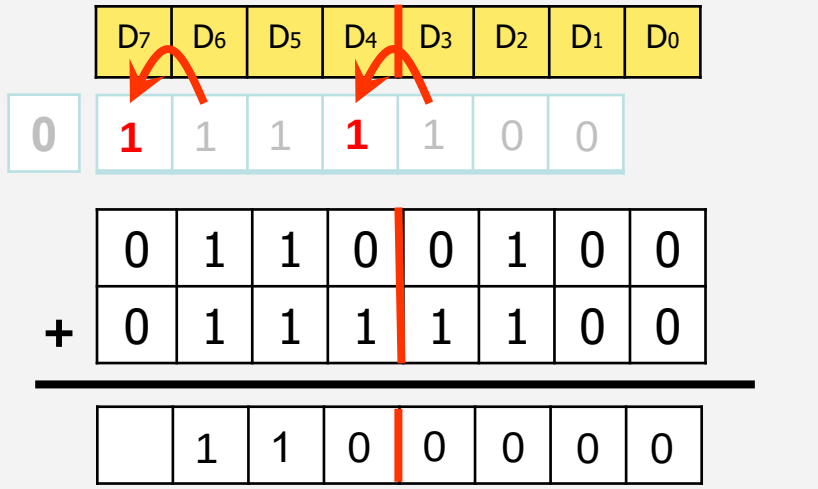
Status Register (SREG)



Status Register (SREG)

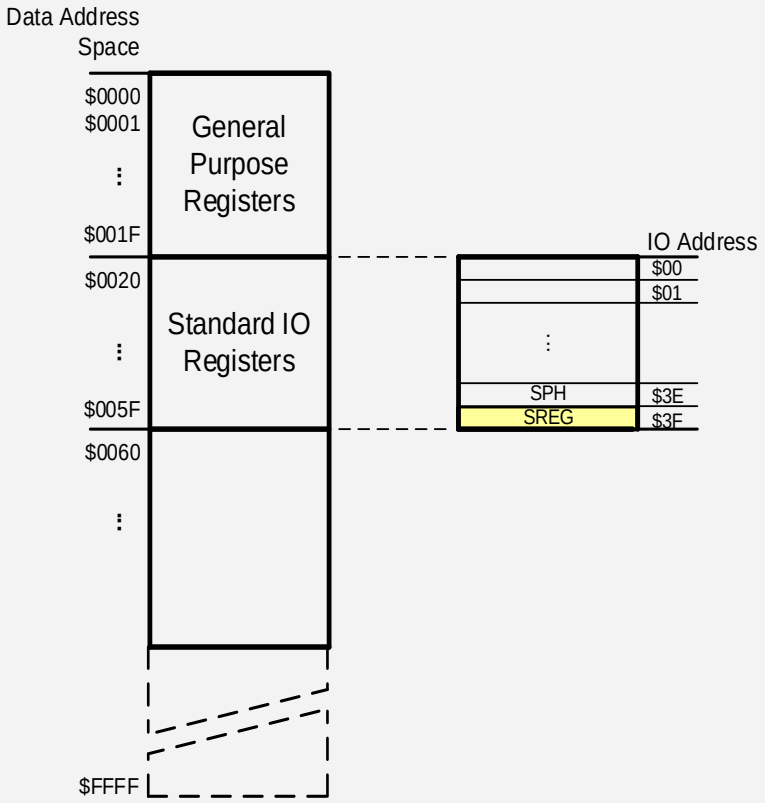


Over Flow and Negative Flag

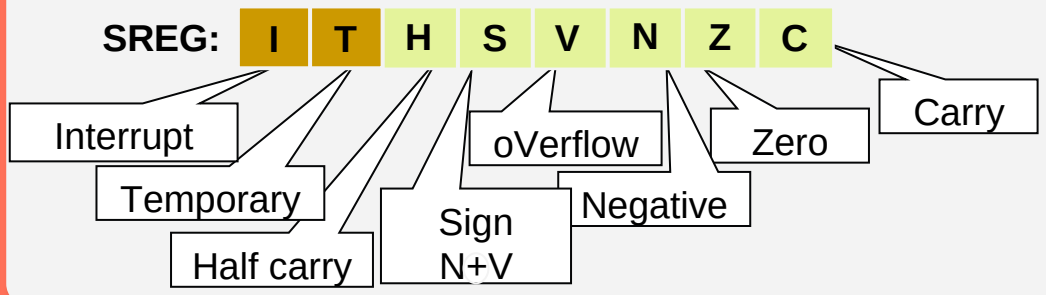


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	1	0	0	0

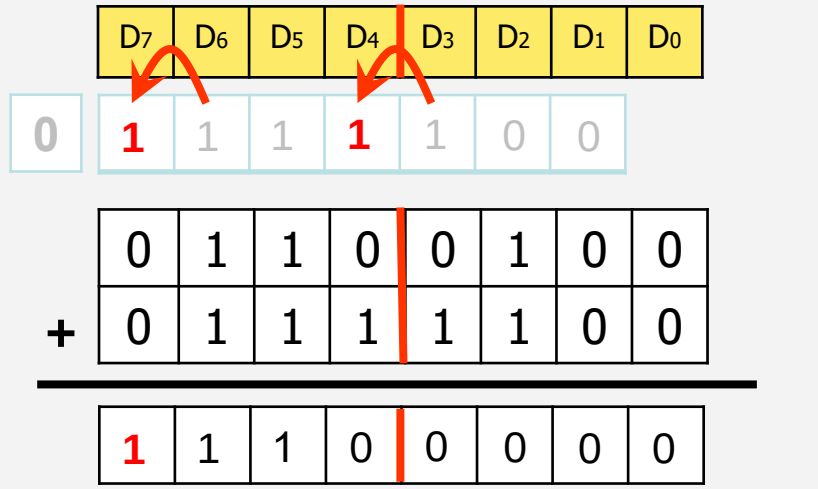
Status Register (SREG)



Status Register (SREG)

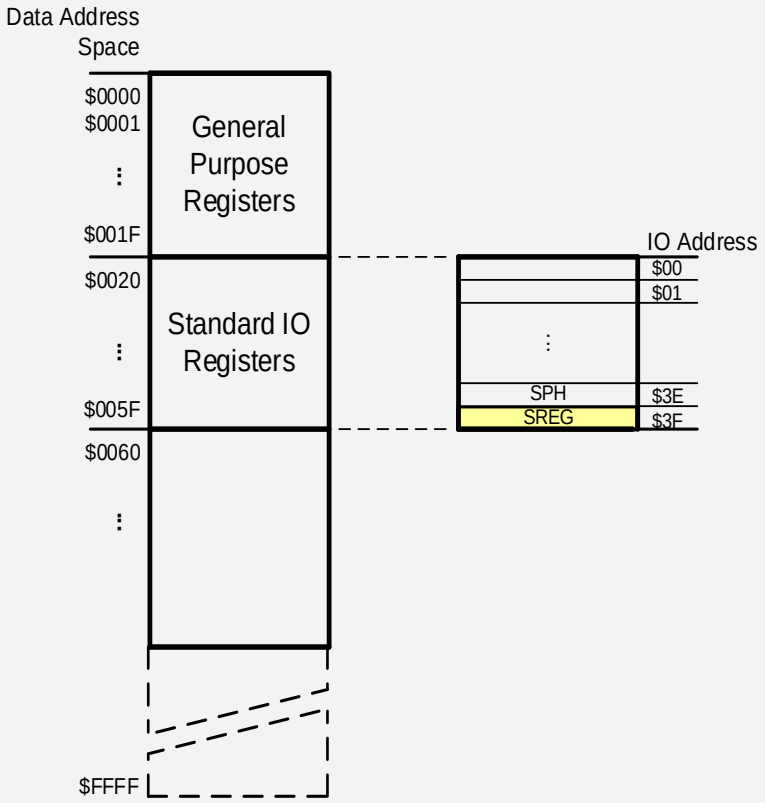


Over Flow and Negative Flag

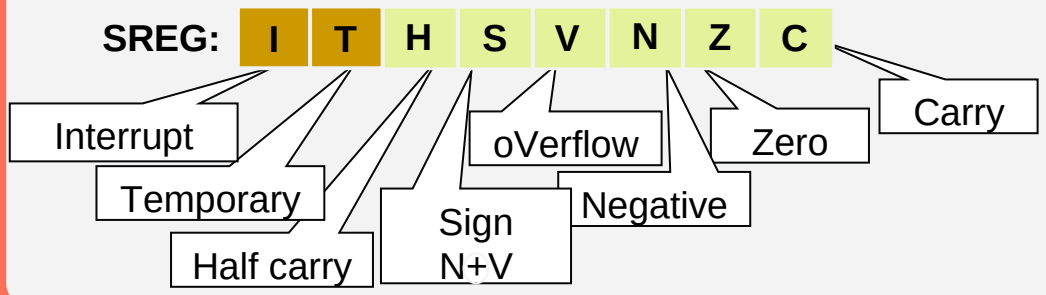


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	1	0	0	0

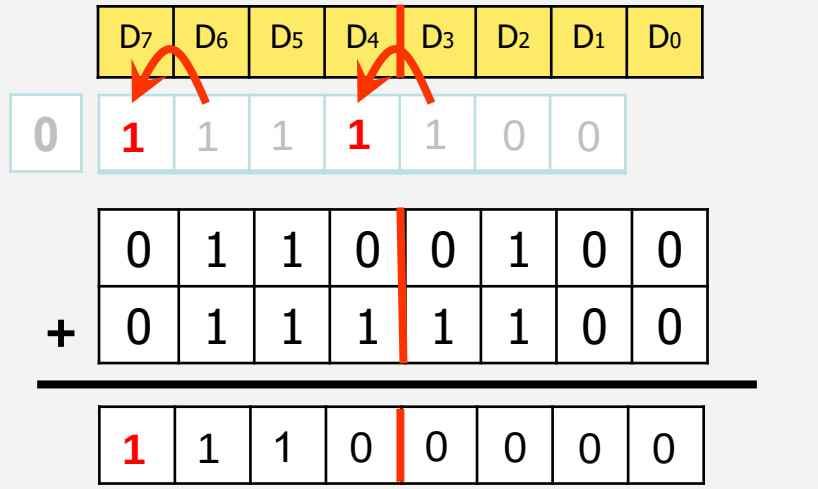
Status Register (SREG)



Status Register (SREG)

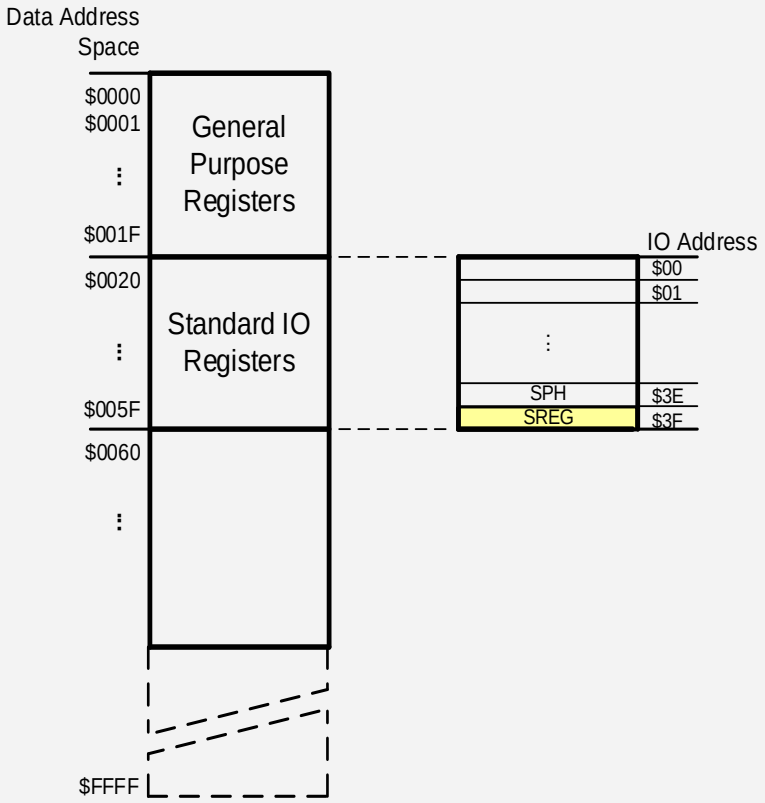


Over Flow and Negative Flag

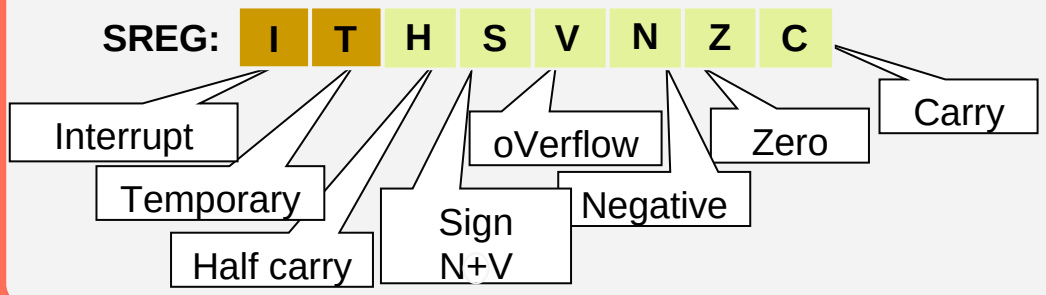


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	1	1	0	0

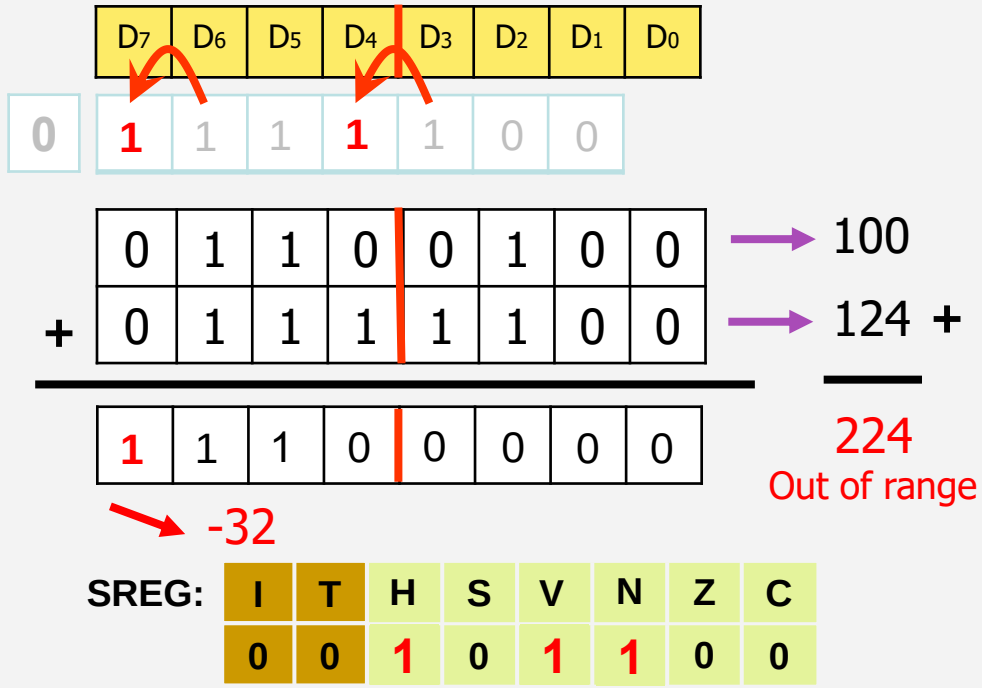
Status Register (SREG)



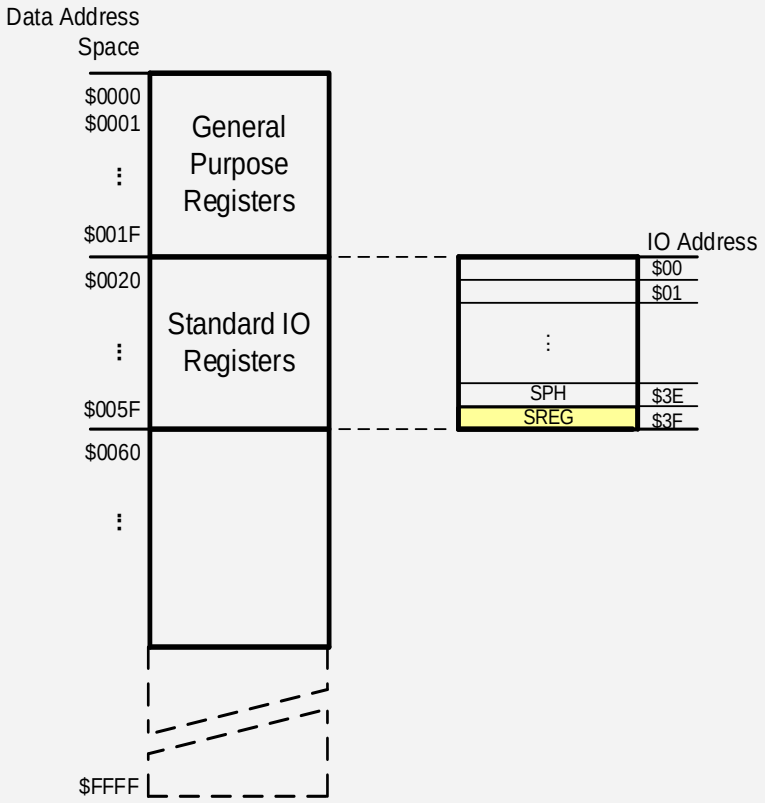
Status Register (SREG)



Over Flow and Negative Flag



Status Register (SREG)



Sign flag

$$S = V \oplus N = 1 \oplus 1 = 0$$

0	0	0	0	0	0	0	1

+256

1	1	1	0	0	0	0	0

-32

+224

Status Register (SREG)

SREG:	I	T	H	S	V	N	Z	C

Interrupt
Temporary
Half carry
Sign N+V
oVerflow
Negative
Zero
Carry

Over Flow and Negative Flag

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
	0	1	1	1	1	1	0	0

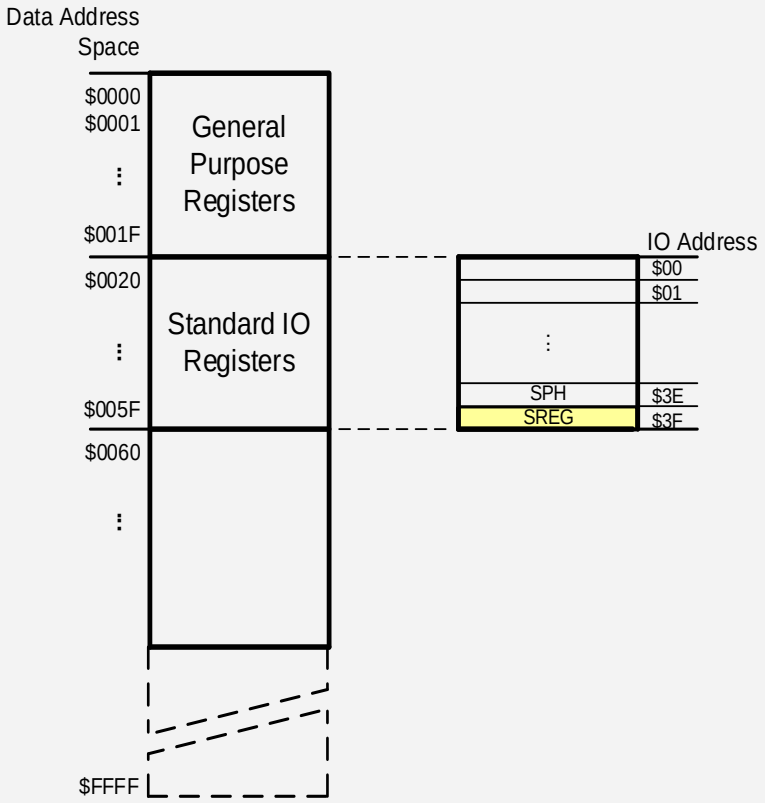
	0	1	1	0	0	1	0	0	→ 100
+	0	1	1	1	1	1	0	0	→ 124 +
	1	1	1	0	0	0	0	0	224

Out of range

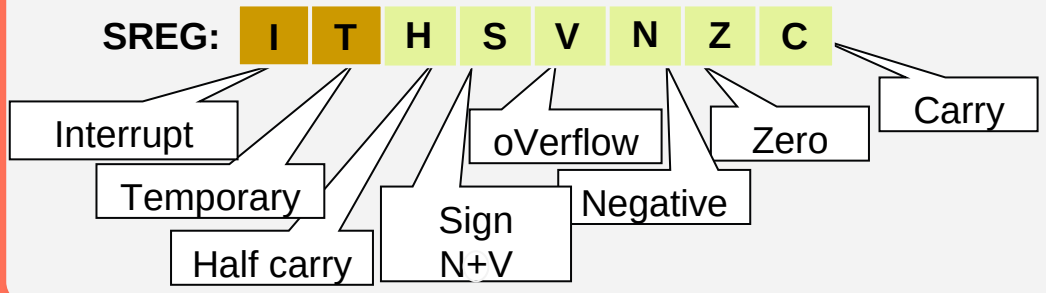
-32

SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	1	1	0	0

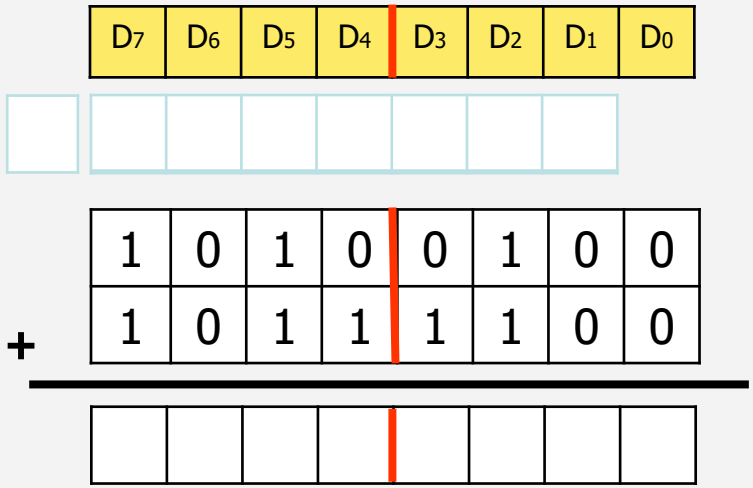
Status Register (SREG)



Status Register (SREG)

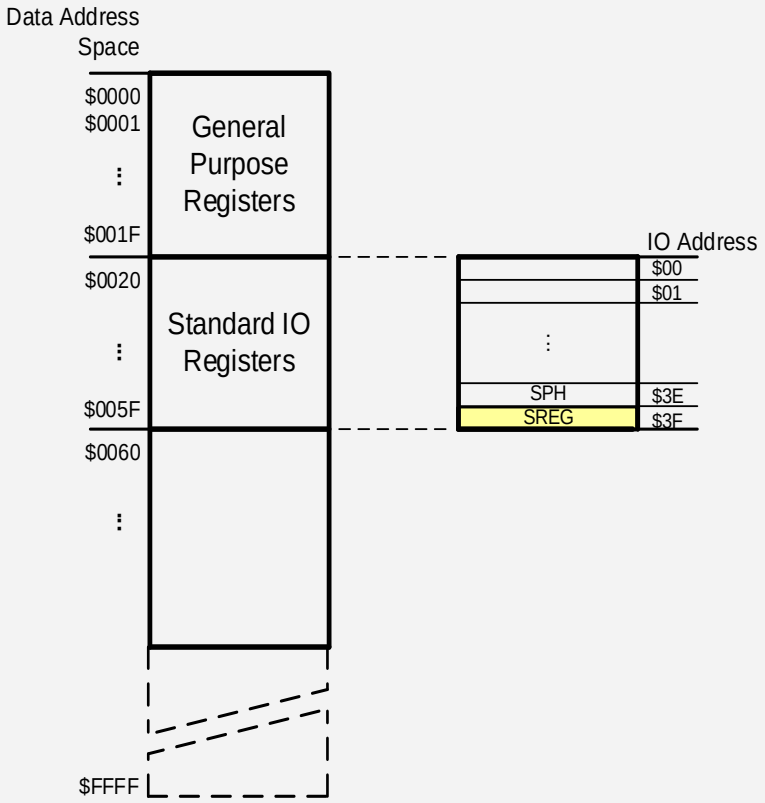


Over Flow and Negative Flag

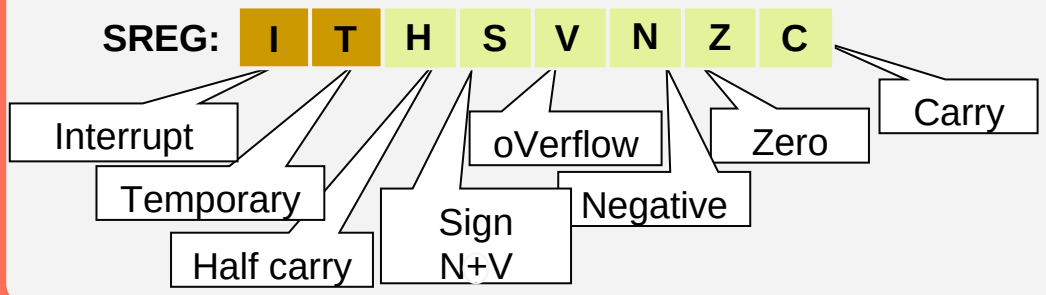


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

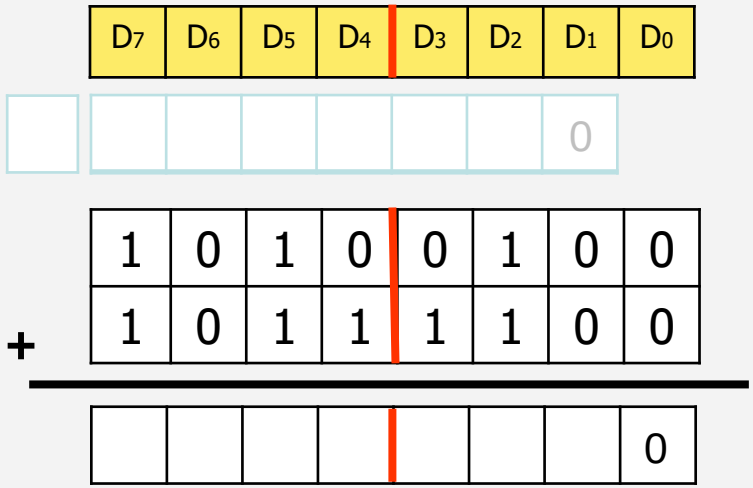
Status Register (SREG)



Status Register (SREG)

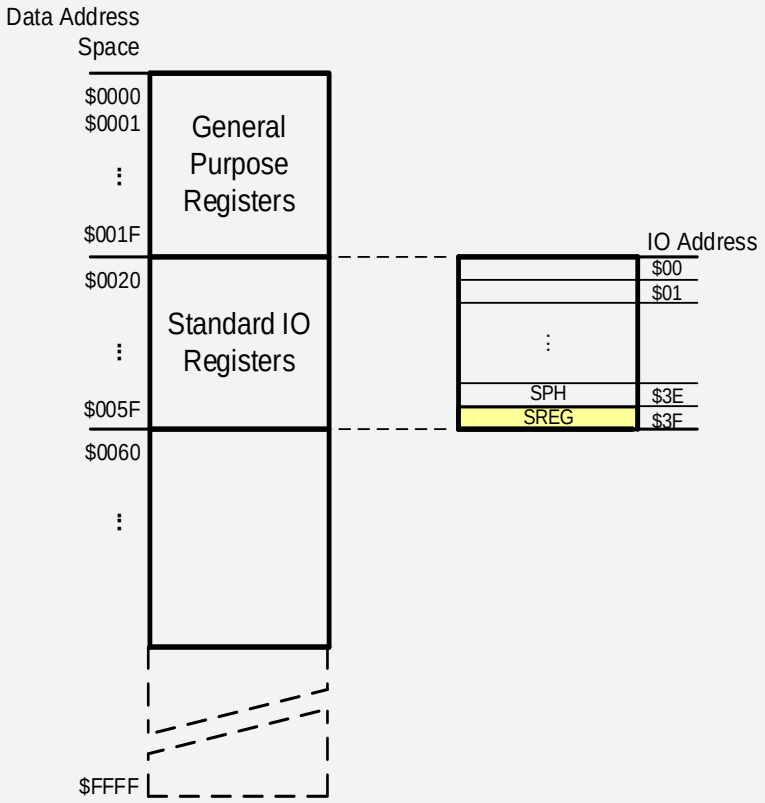


Over Flow and Negative Flag

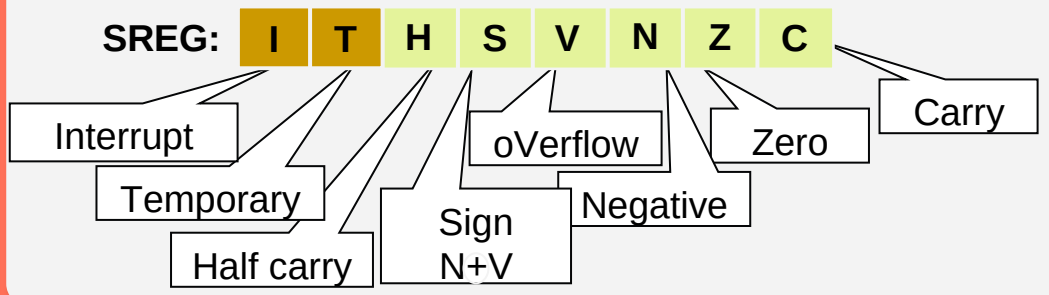


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

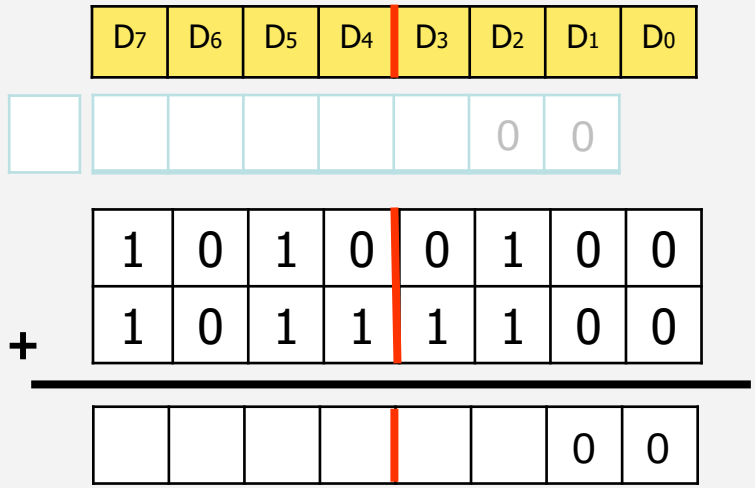
Status Register (SREG)



Status Register (SREG)

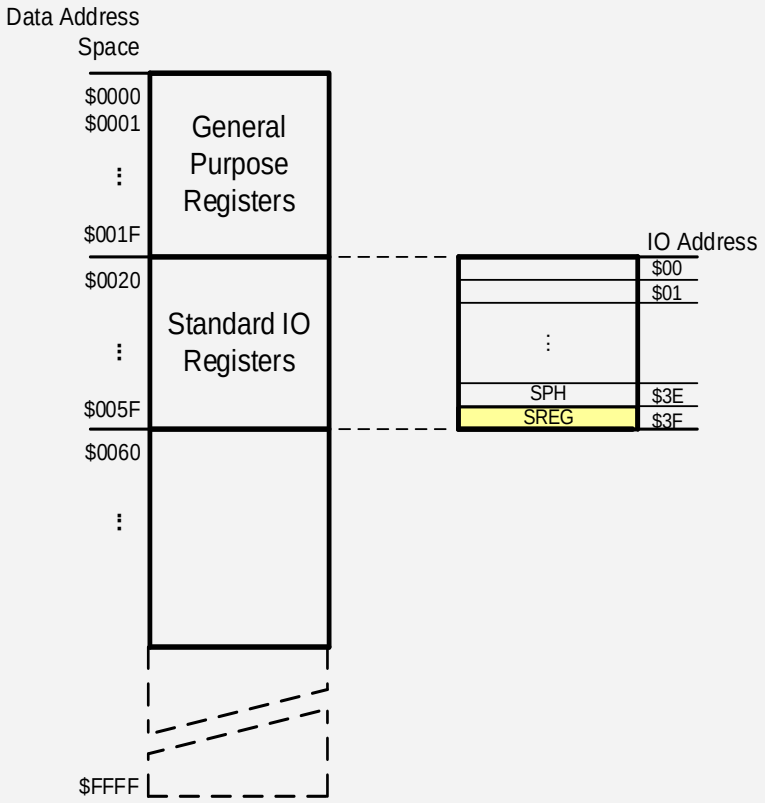


Over Flow and Negative Flag

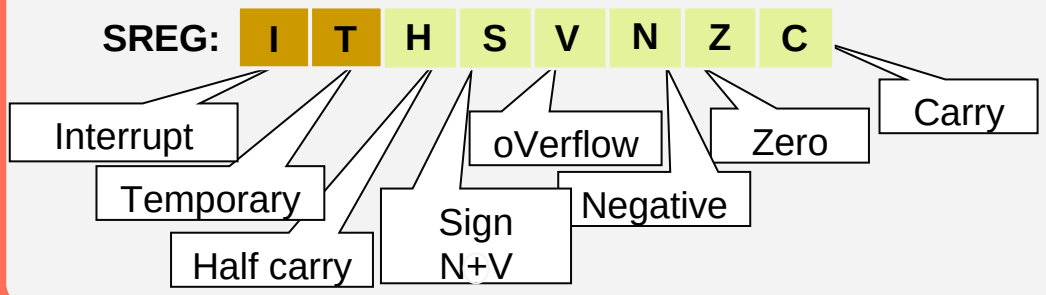


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

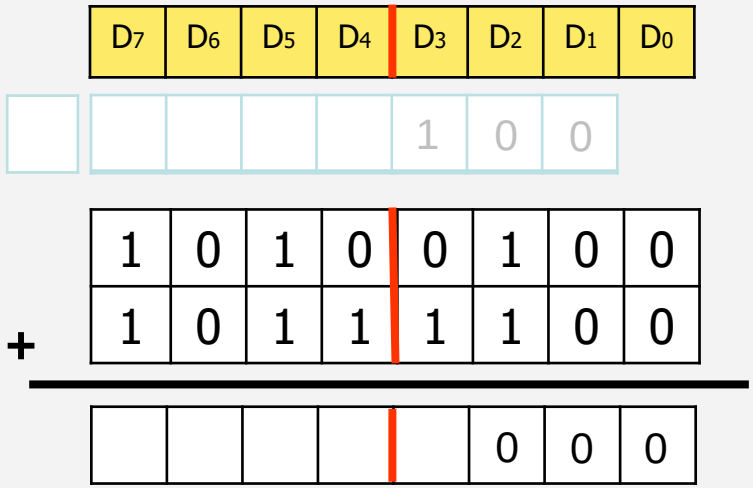
Status Register (SREG)



Status Register (SREG)

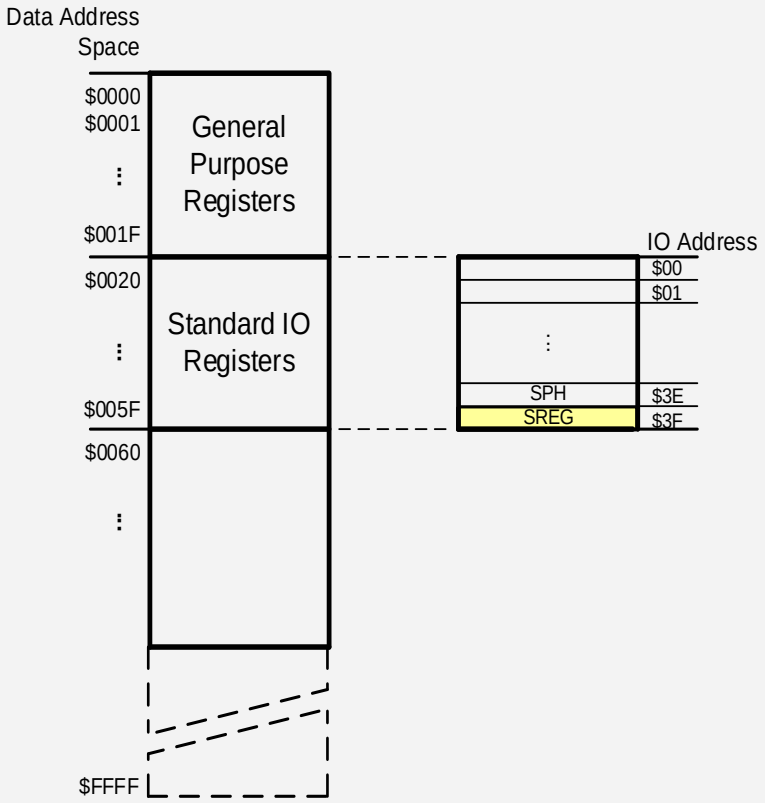


Over Flow and Negative Flag

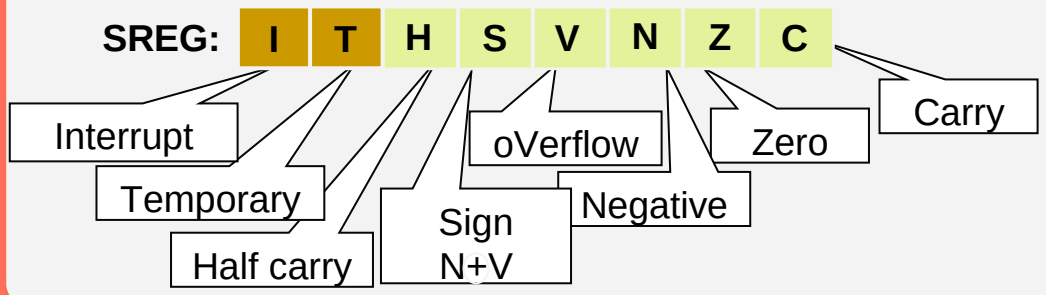


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

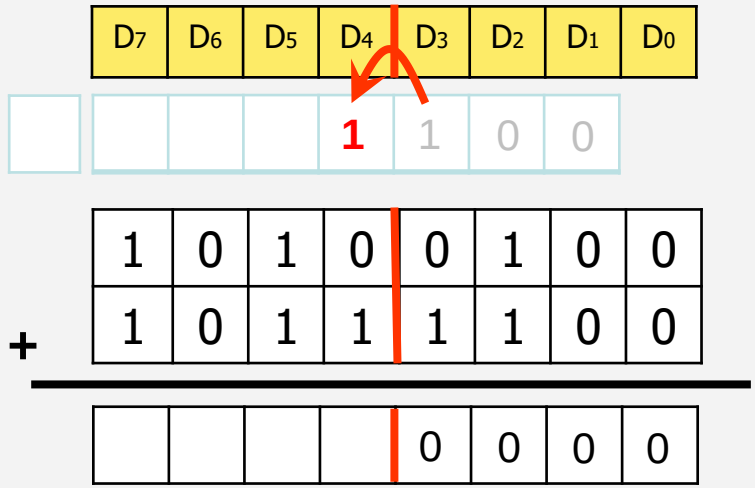
Status Register (SREG)



Status Register (SREG)

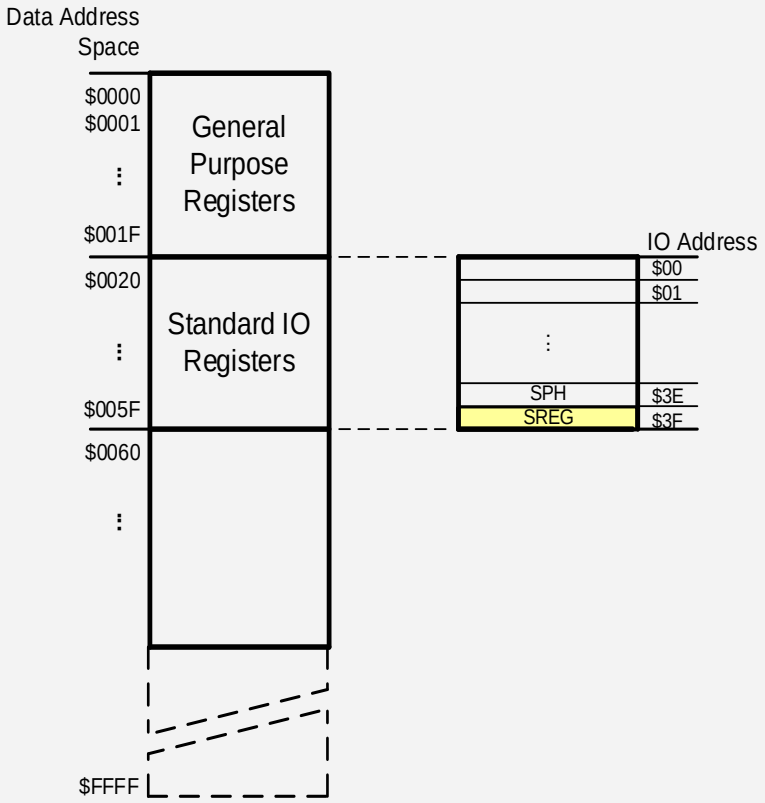


Over Flow and Negative Flag

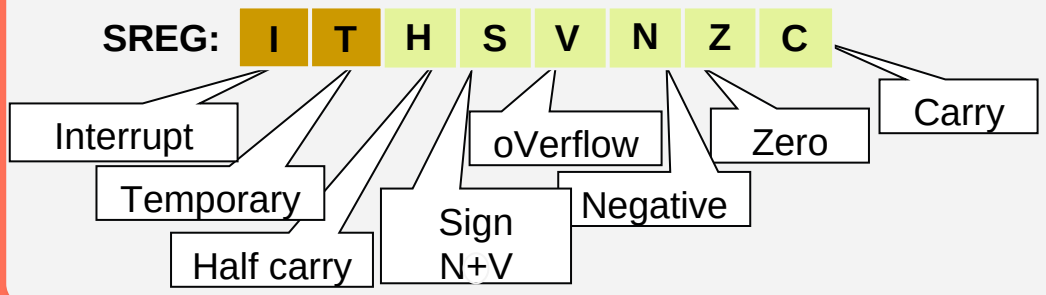


SREG:	I	T	H	S	V	N	Z	C
	0	0	0	0	0	0	0	0

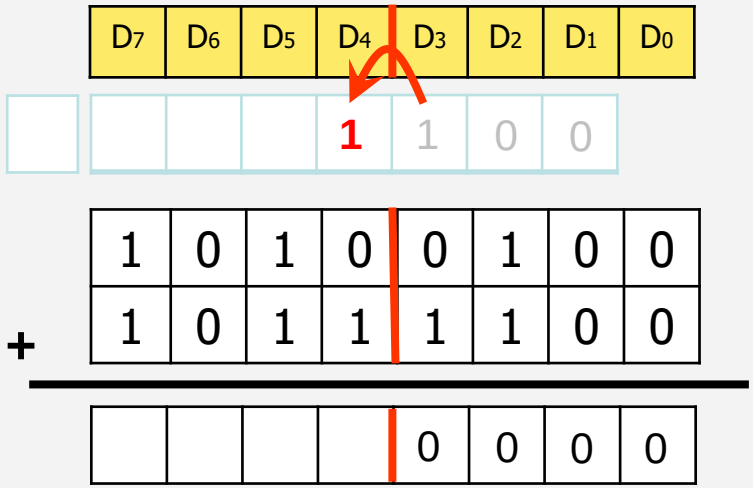
Status Register (SREG)



Status Register (SREG)

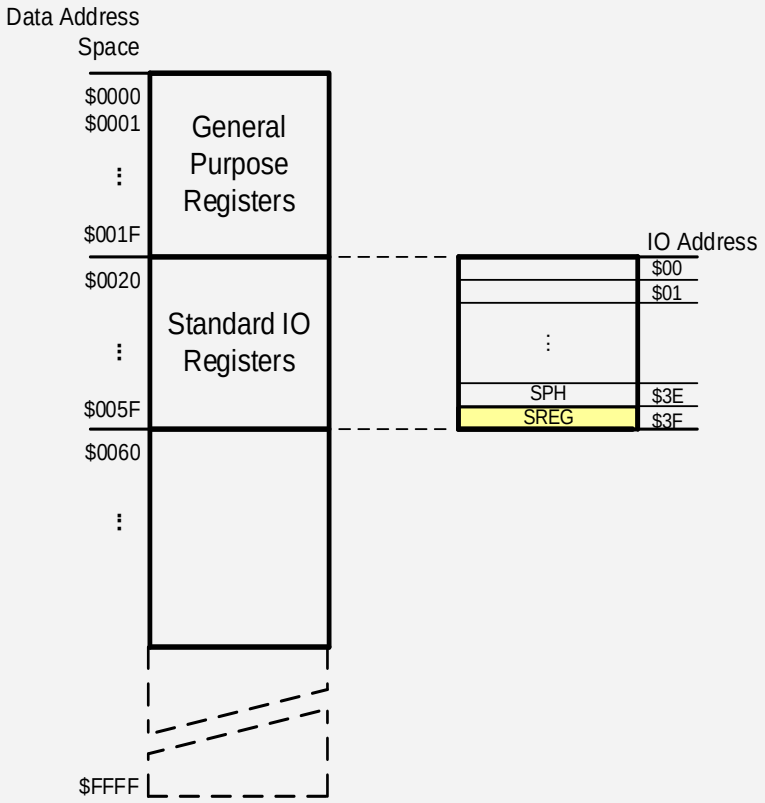


Over Flow and Negative Flag

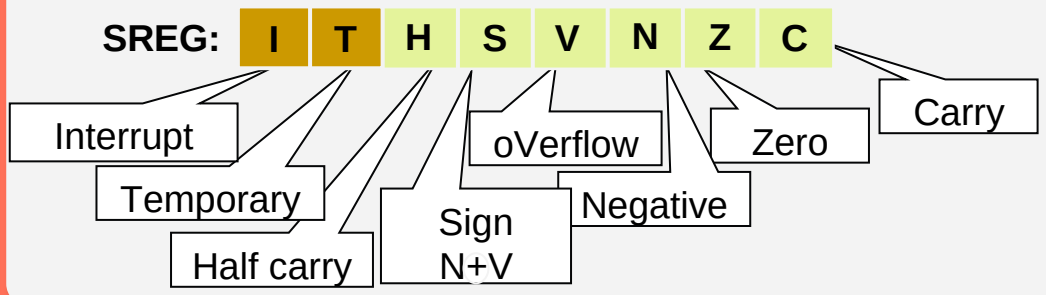


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

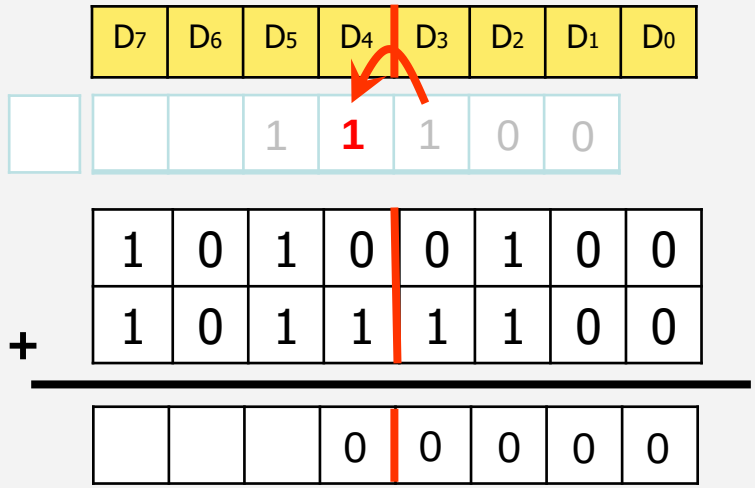
Status Register (SREG)



Status Register (SREG)

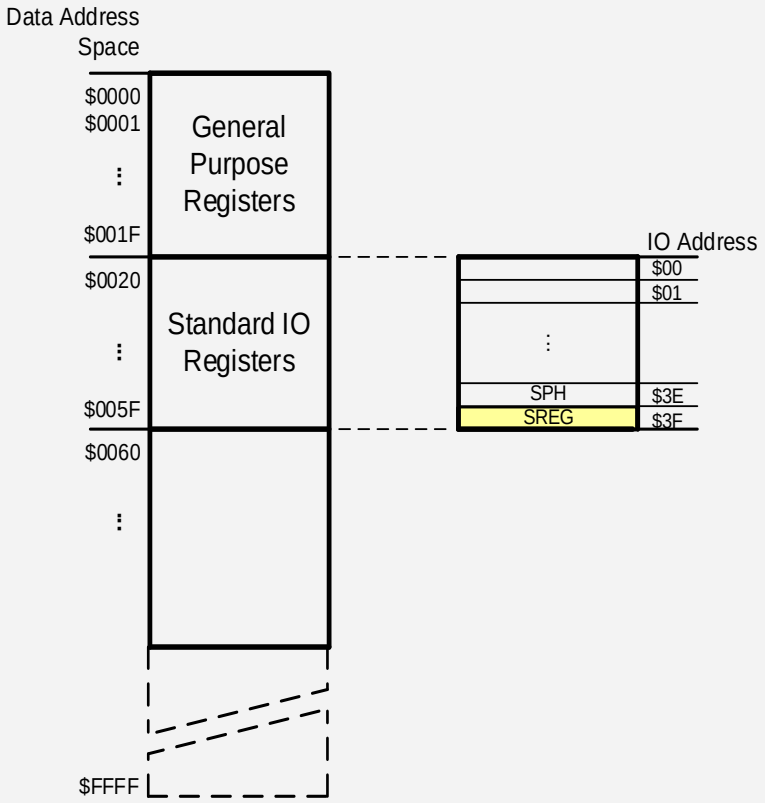


Over Flow and Negative Flag

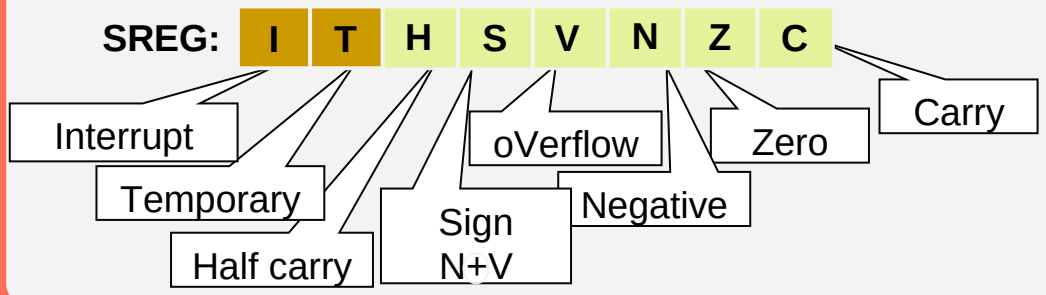


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

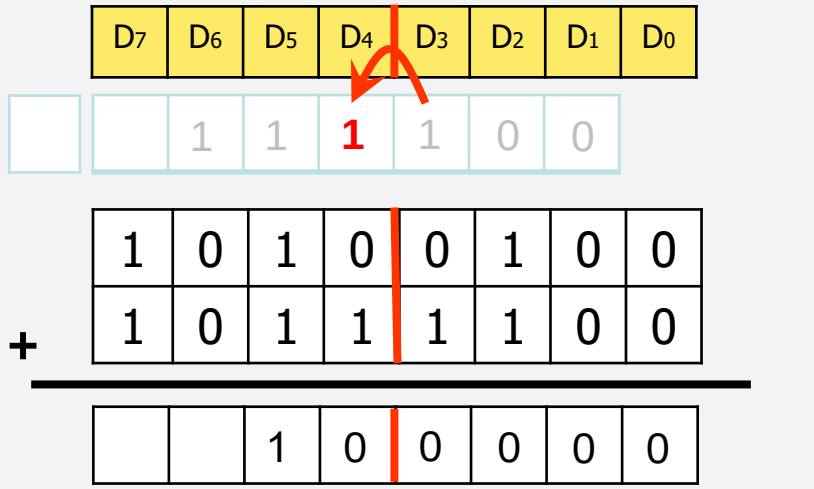
Status Register (SREG)



Status Register (SREG)

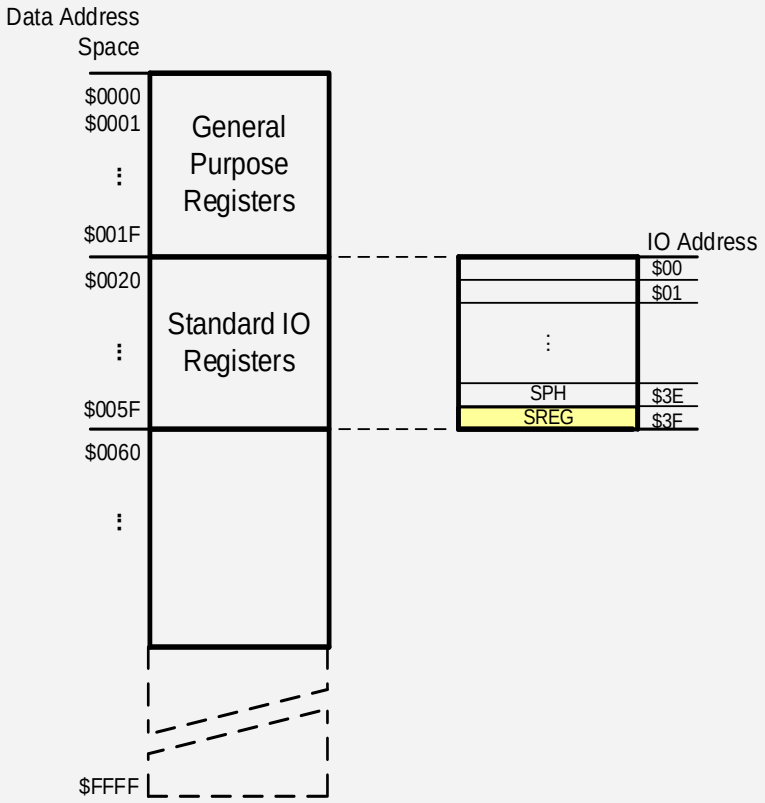


Over Flow and Negative Flag

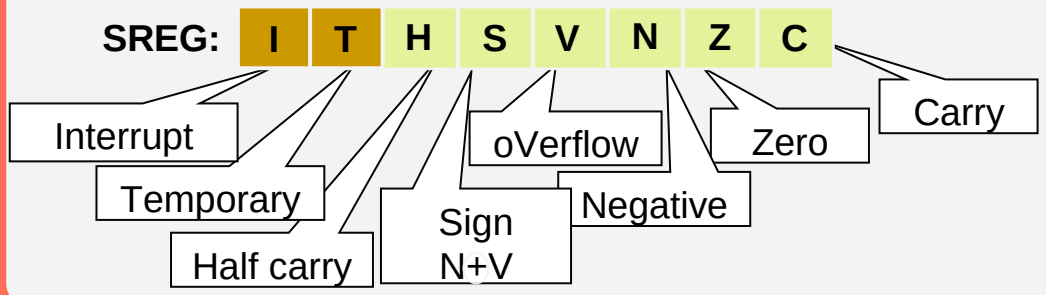


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

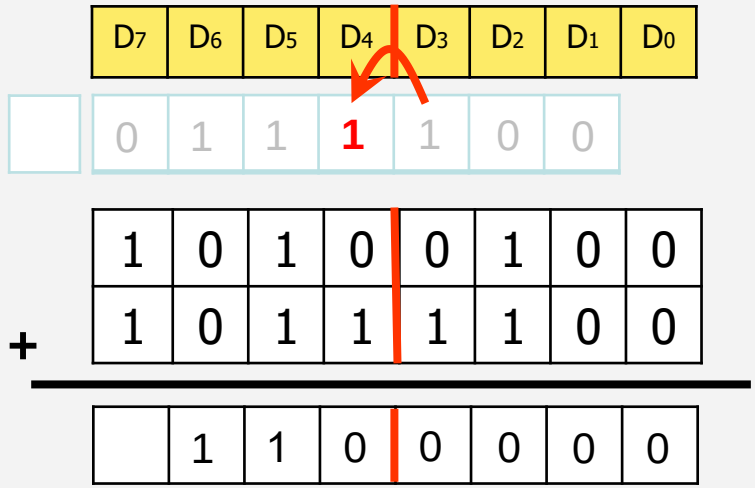
Status Register (SREG)



Status Register (SREG)

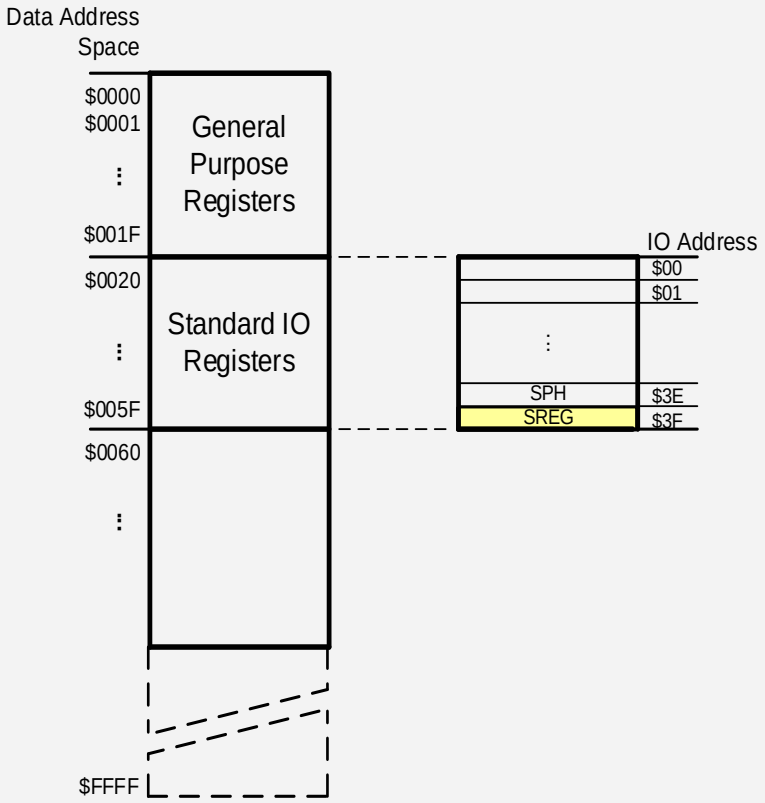


Over Flow and Negative Flag

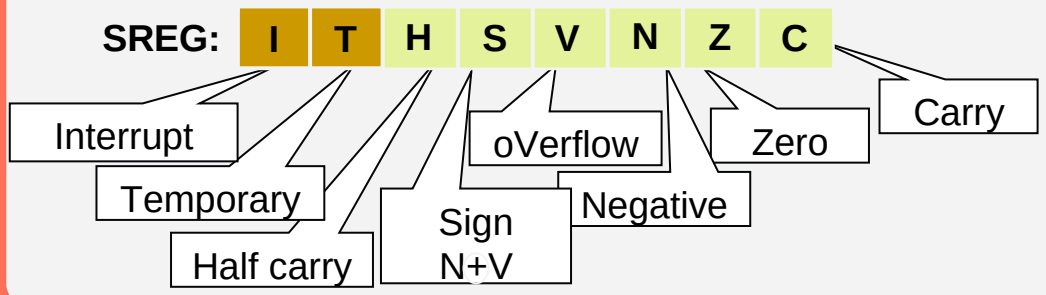


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

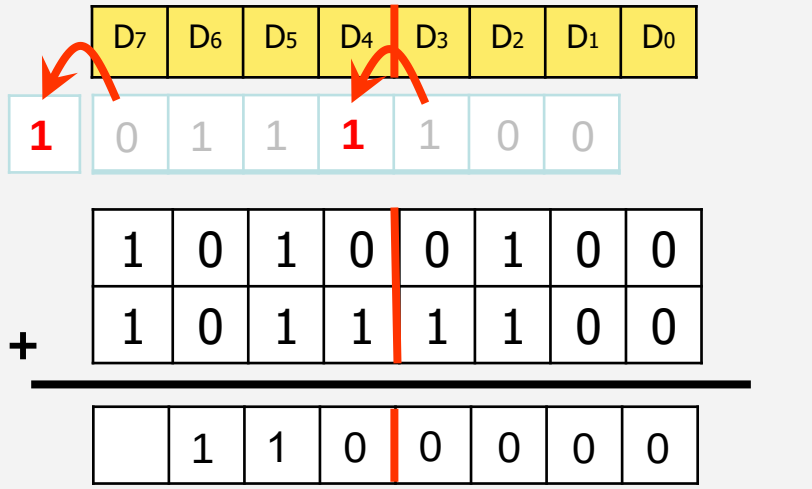
Status Register (SREG)



Status Register (SREG)

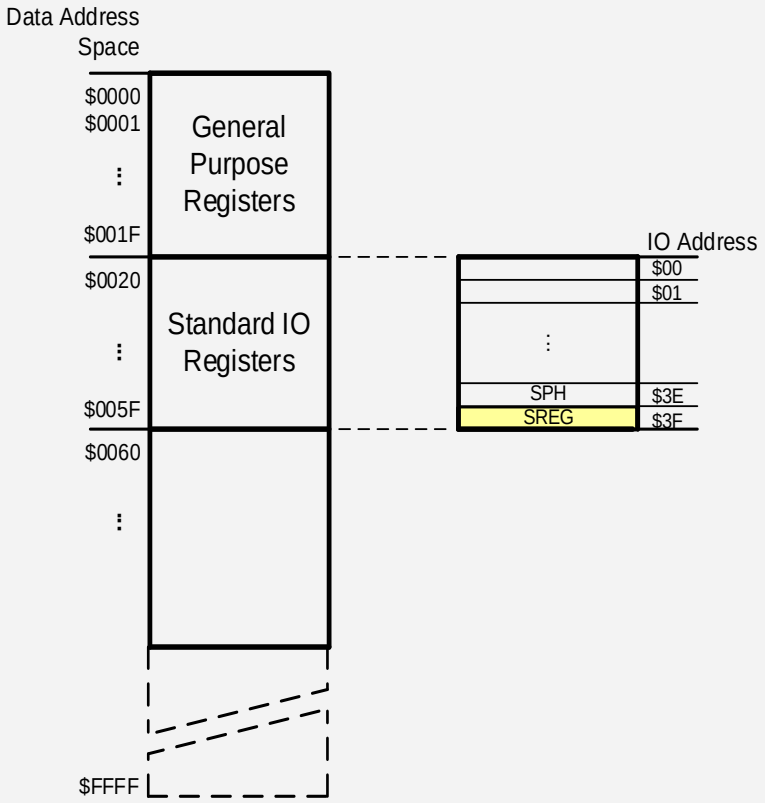


Over Flow and Negative Flag

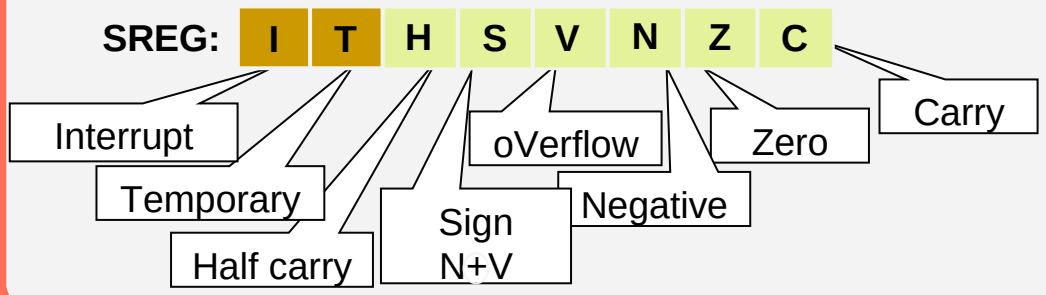


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	0	0	0	0

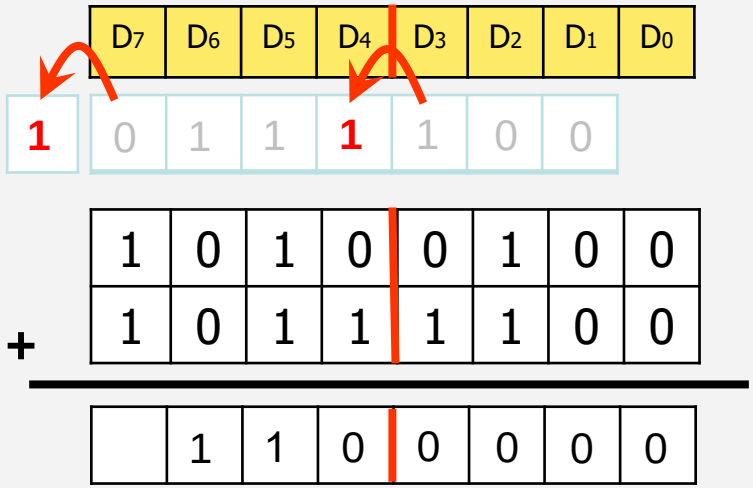
Status Register (SREG)



Status Register (SREG)

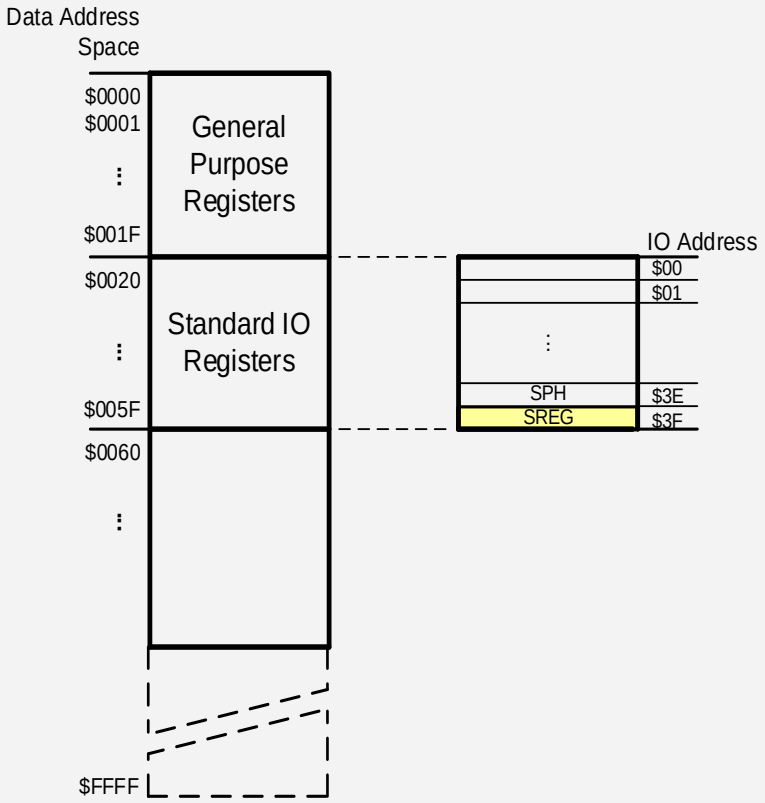


Over Flow and Negative Flag

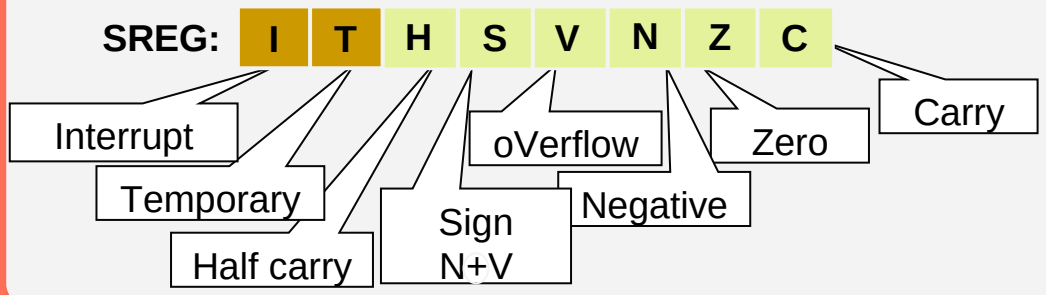


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	1	0	0	0

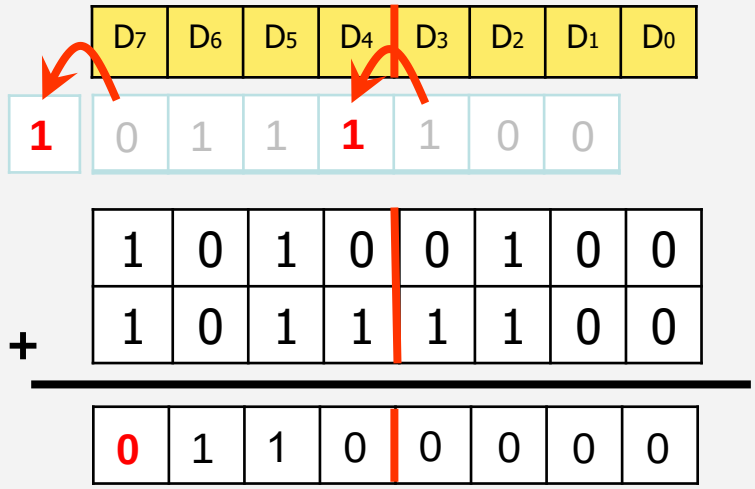
Status Register (SREG)



Status Register (SREG)

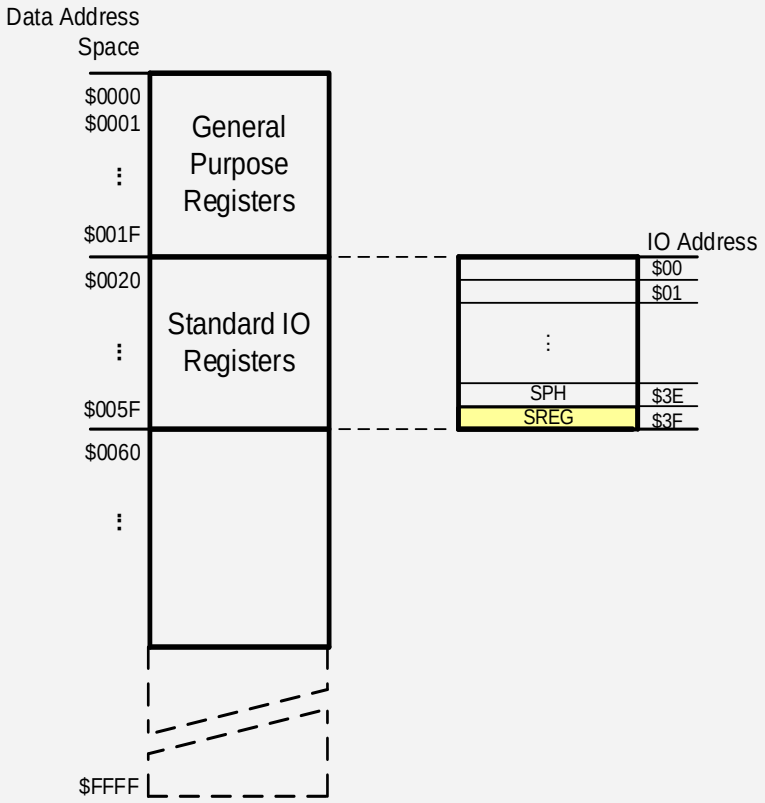


Over Flow and Negative Flag

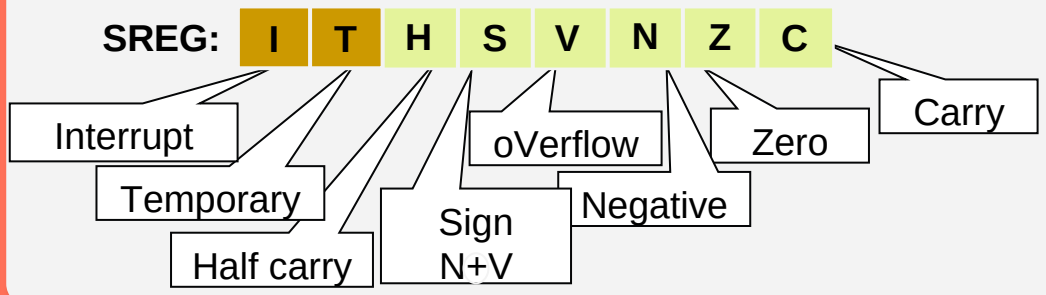


SREG:	I	T	H	S	V	N	Z	C
	0	0	1	0	1	0	0	0

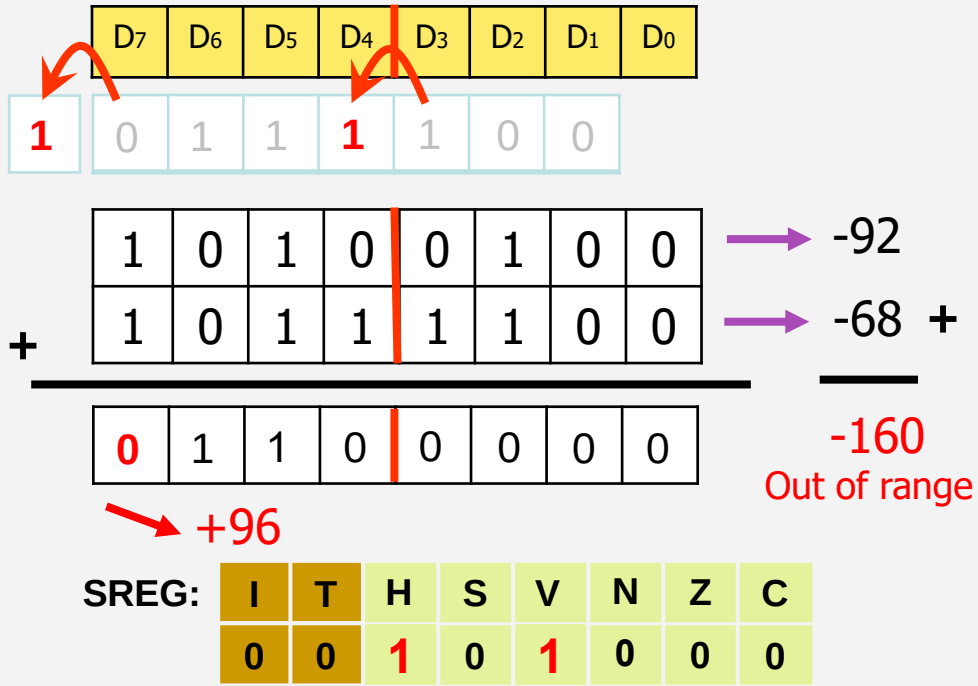
Status Register (SREG)



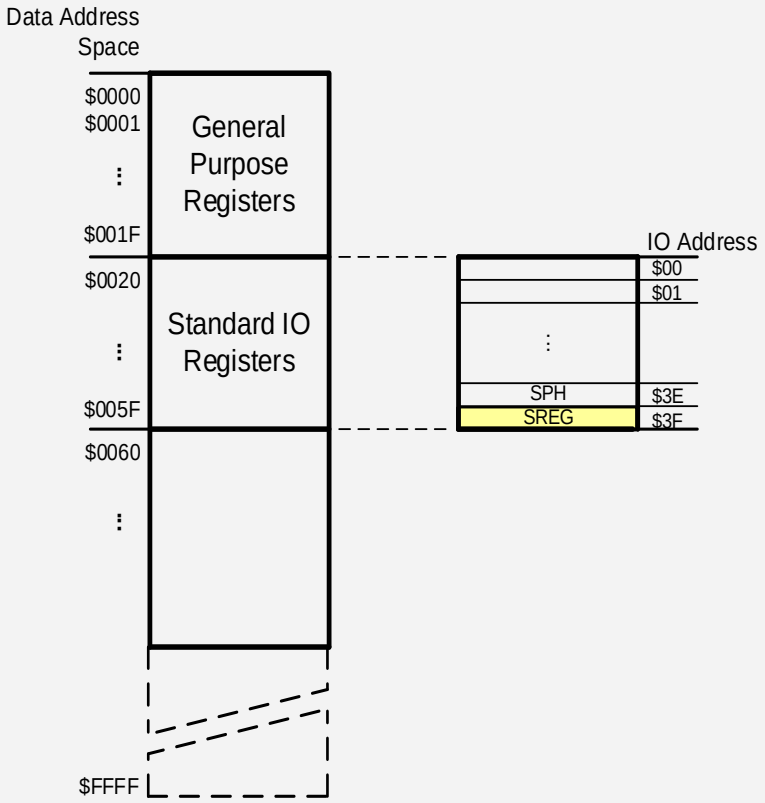
Status Register (SREG)



Over Flow and Negative Flag



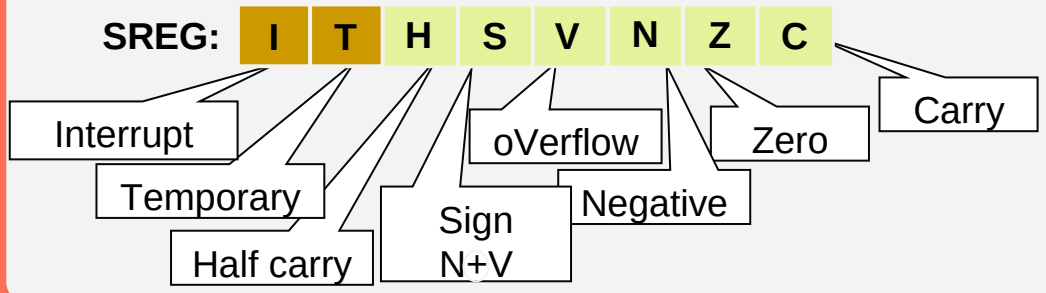
Status Register (SREG)



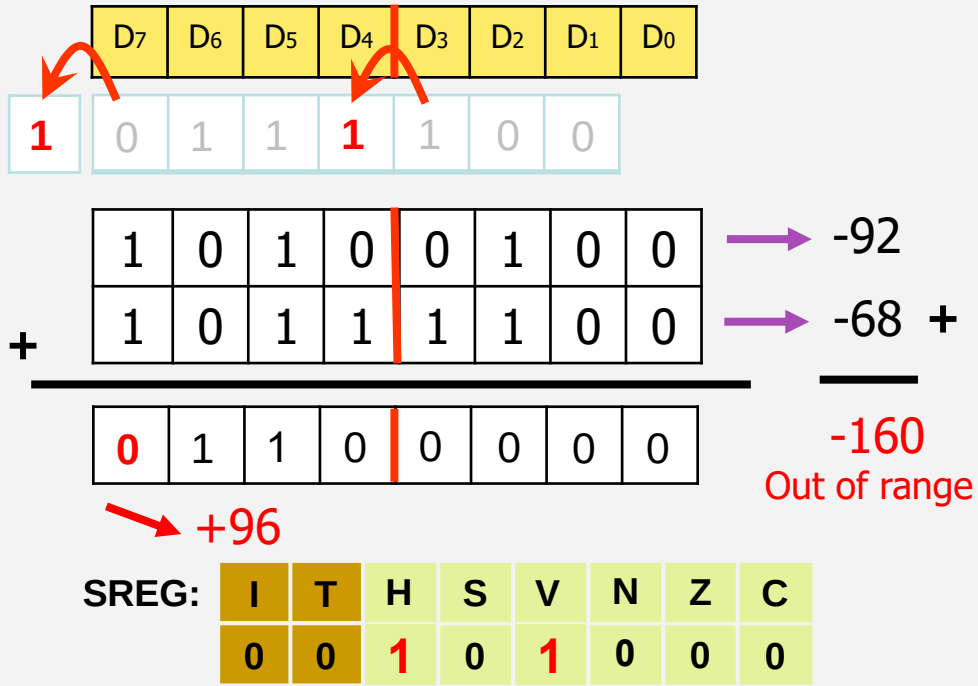
note

Addition of two numbers with sign bits OFF may result in a number with sign bit ON

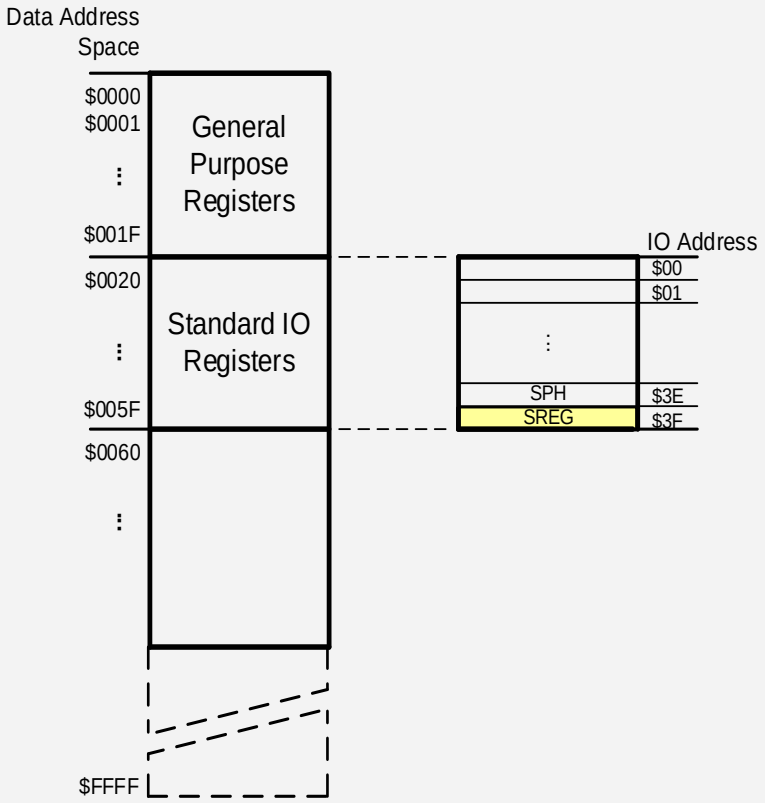
Status Register (SREG)



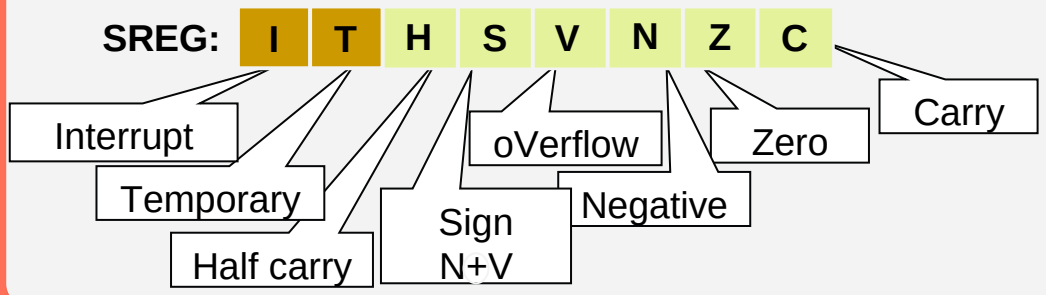
Over Flow and Negative Flag



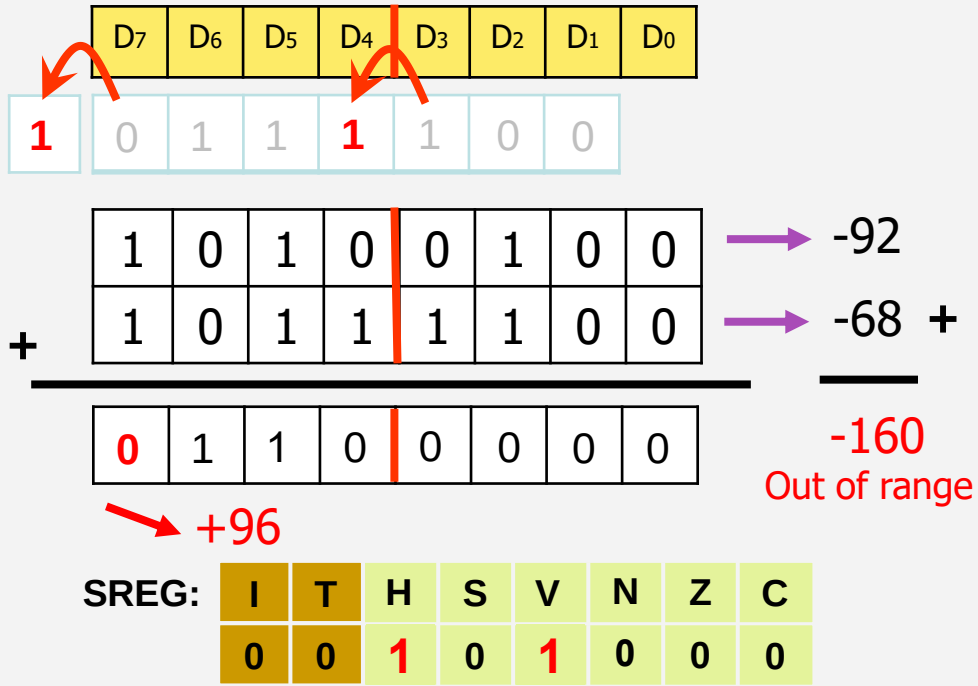
Status Register (SREG)



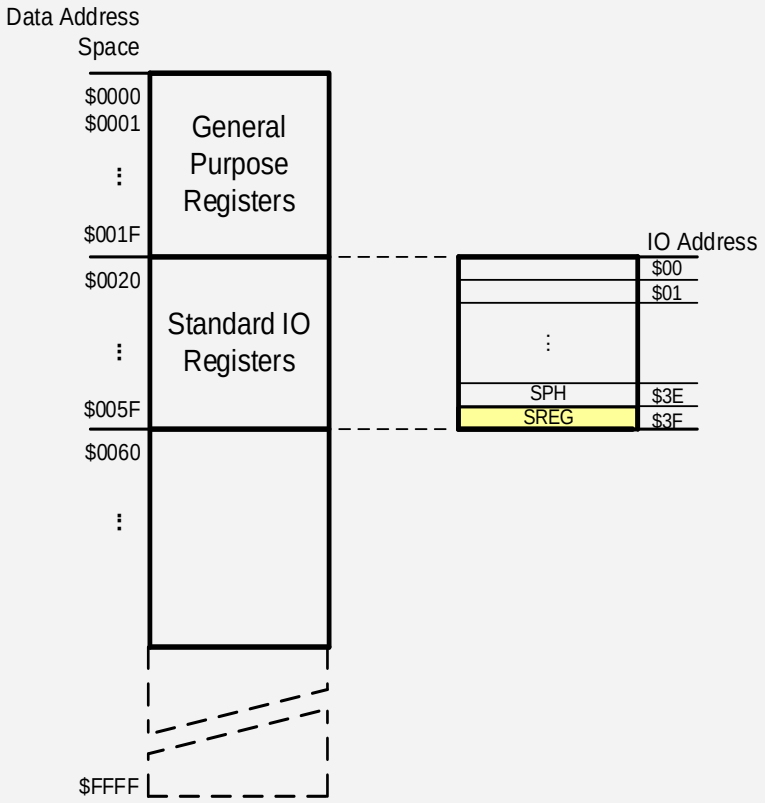
Status Register (SREG)



Over Flow and Negative Flag



Status Register (SREG)



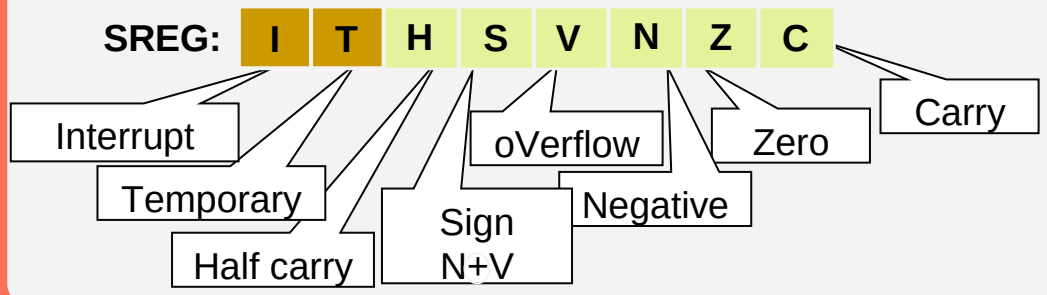
Sign flag

$$S = V \oplus N = 1 \oplus 0 = 1$$

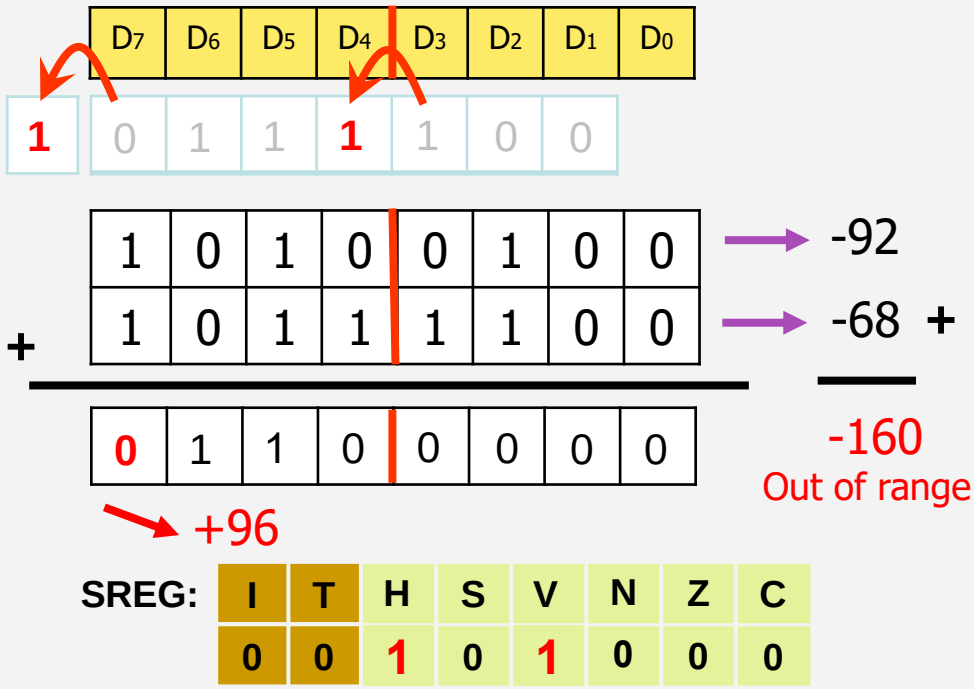
1	0	0	0	0	0	0	1		0	1	1	0	0	0	0	0
-256									+96							

-160

Status Register (SREG)



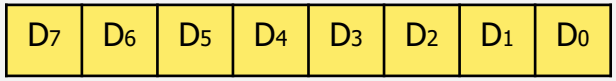
Over Flow and Negative Flag



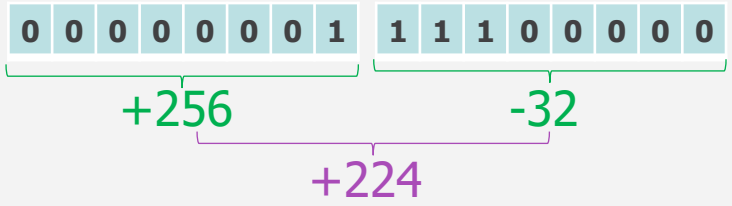
AVR DATA FORMAT AND DIRECTIVES

In AVR microcontrollers (like Atmega328):

Data types: only one type available: It is 8 bits (size of each register is also 8 bits)



- programmer should break down data larger than 8 bits (00 to 0xFF, or 0 to 255 in decimal) to be processed by the CPU

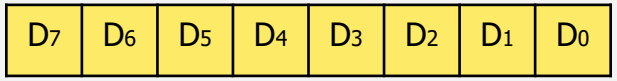


- The data types used by the AVR can be positive or negative.

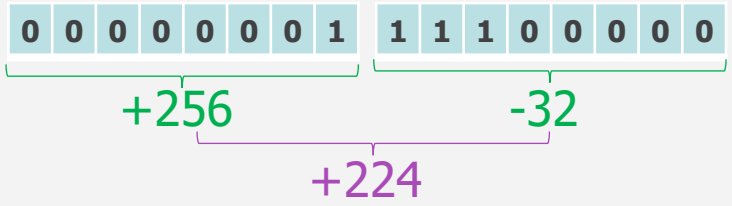
AVR DATA FORMAT AND DIRECTIVES

In AVR microcontrollers (like Atmega328):

Data types: only one type available: It is 8 bits (size of each register is also 8 bits)



- programmer should break down data larger than 8 bits (00 to 0xFF, or 0 to 255 in decimal) to be processed by the CPU



- The data types used by the AVR can be positive or negative.

Data format representation

There are four ways to represent a byte of data (8 bits) in the AVR assembler:

hex,	0x23, $23
binary,	0b00100011
decimal,	35
ASCII	'3'

Note: for defining ASCII strings (more than one character), use the .DB (define byte) directive.

AVR DIRECTIVES

instructions tell the CPU what to do

✓ the LDI and ADD instructions are commands to the CPU

directives (*pseudoinstructions*) give directions to the assembler.

✓ .EQU, .DEVICE, and .ORG are directives to the assembler.

❑ The directives help develop programs easier and make program legible (more readable).

Assembler Directives

- .EQU *name = value*

Example:

```
.EQU    COUNT = 0x25
LDI     R21, COUNT          ;R21 = 0x25
LDI     R22, COUNT + 3      ;R22 = 0x28
```

- .SET *name = value*

Example:

```
.SET    COUNT = 0x25
LDI     R21, COUNT          ;R21 = 0x25
LDI     R22, COUNT + 3      ;R22 = 0x28
.SET    COUNT = 0x19
LDI     R21, COUNT          ;R21 = 0x19
```

Using .EQU for fixed data assignment

Example:

```
.EQU    DATA1 = 0x39
.EQU    DATA2 = $39
```

Using .EQU for SFR address assignment

Example:

```
.EQU    COUNTER = 0x00          ;counter value 00
.EQU    PORTB = 0x18            ;SFR Port B address
LDI     R16, COUNTER            ;R16 = 0x00
OUT     PORTB, R16              ;Port B (loc 0x18) now has 00 too
```

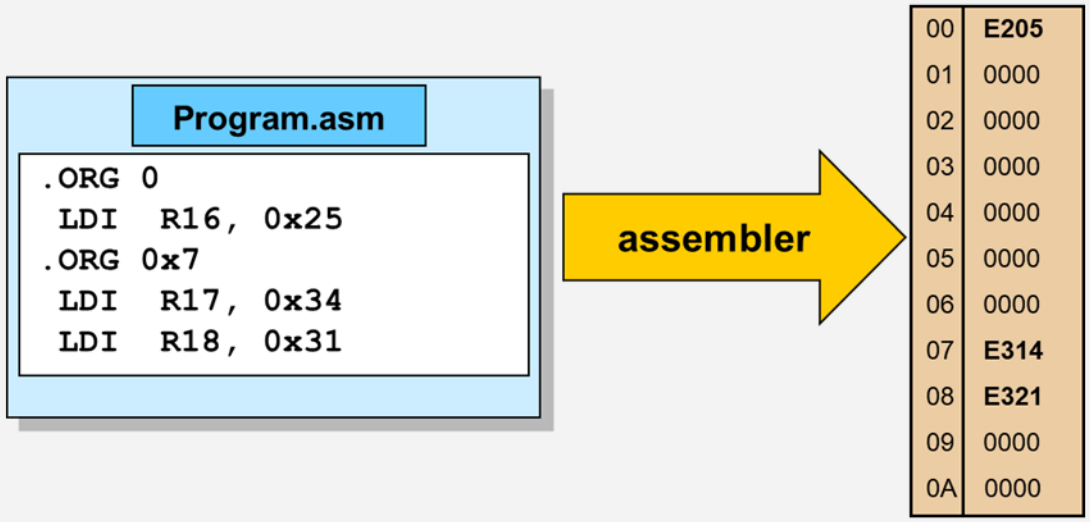
Using .EQU for RAM address assignment

Example:

```
.EQU SUM = 0x120            ;assign RAM loc to SUM
LDI R20, 5                  ;load R20 with 5
LDI R21, 2                  ;load R21 with 2
ADD R20, R21                ;R20 = R20 + R21
ADD R20, R21                ;R20 = R20 + R21
STS SUM, R20                ;store the result in loc 0x120
```

Assembler Directives

- .ORG *address*



Assembler Directives

- **.INCLUDE *"filename.ext"***

  to add the contents of a file to program (like the #include directive in C language)

Assembler Directives

- **.INCLUDE *"filename.ext"***

  to add the contents of a file to program (like the #include directive in C language)

Some of the Common AVRs and Their Include Files

ATMEGA		ATTINY		Special Purpose	
ATmega8	m8def.inc	ATtiny11	tn11def.inc	AT90CAN32	can32def.inc
ATmega16	m16def.inc	ATtiny12	tn12def.inc	AT90CAN64	can64def.inc
ATmega32	m32def.inc	ATtiny22	tn22def.inc	AT90PWM2	pwm2def.inc
ATmega328	m328def.inc	ATtiny44	tn44def.inc	AT90PWM3	pwm3def.inc
ATmega128	m128def.inc	ATtiny85	tn85def.inc	AT90USB646	usb646def.inc
ATmega2560	m2560def.inc				

➢.INCLU

ATMEGA
ATmega8
ATmega1
ATmega3
ATmega3
ATmega1
ATmega2

M328def.inc

```
.equ    SREG    = 0x3f
.equ    SPL     = 0x3d
.equ    SPH     = 0x3e
....
```

➢.INCLU

ATMEGA
ATmega8
ATmega1
ATmega3
ATmega3
ATmega1
ATmega2

M328def.inc

```
.equ    SREG    = 0x3f
.equ    SPL     = 0x3d
.equ    SPH     = 0x3e
....
```

Program.asm

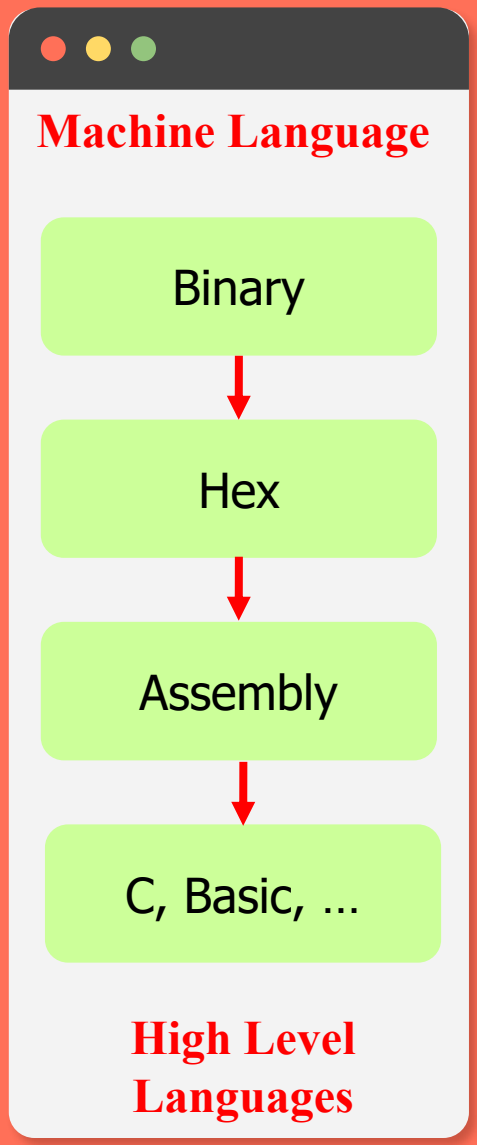
```
    LDI   R20, 10
    OUT   SPL, R20
```

INTRODUCTION TO AVR ASSEMBLY PROGRAMMING

- ✓ CPU can work only in binary at a very high speed.
- ✓ It is quite tedious and slow for humans to deal with 0s and 1s in order to program the computer.

Machine Language

- ✓ Although the **hexadecimal system** was used as a more efficient way to represent binary numbers, the process of working in machine code was still cumbersome for humans.
- ✓ **Assembly languages** were developed, which provided mnemonics for the machine code instructions, plus other features that made programming faster and less prone to error.
- ✓ Today, one can use many different programming languages, such as BASIC, Pascal, C, C++, Java, and numerous others.

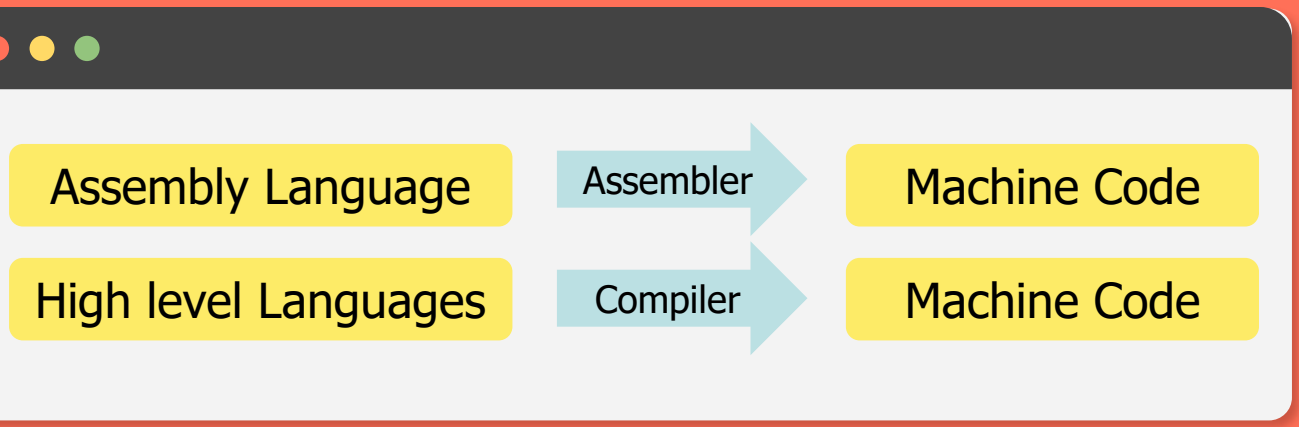


INTRODUCTION TO AVR ASSEMBLY PROGRAMMING

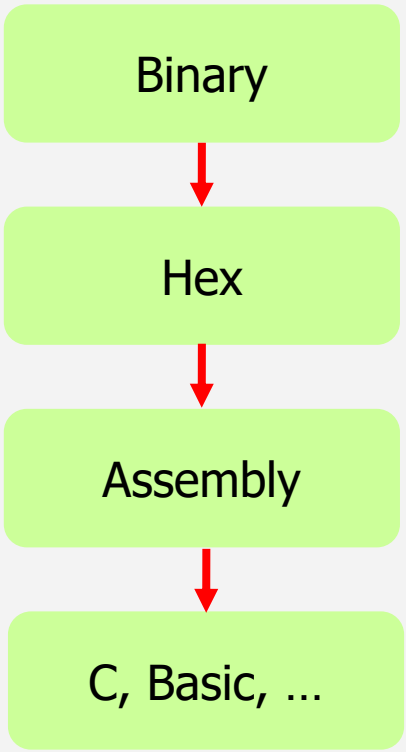
- ✓ CPU can work only in binary at a very high speed.
- ✓ It is quite tedious and slow for humans to deal with 0s and 1s in order to program the computer.

Machine Language

- ✓ Although the **hexadecimal system** was used as a more efficient way to represent binary numbers, the process of working in machine code was still cumbersome for humans.
- ✓ **Assembly languages** were developed, which provided mnemonics for the machine code instructions, plus other features that made programming faster and less prone to error.
- ✓ Today, one can use many different programming languages, such as BASIC, Pascal, C, C++, Java, and numerous others.



Machine Language



High Level Languages

Structure of Assembly language

An Assembly language program consists of:
a series of lines of Assembly language instructions.

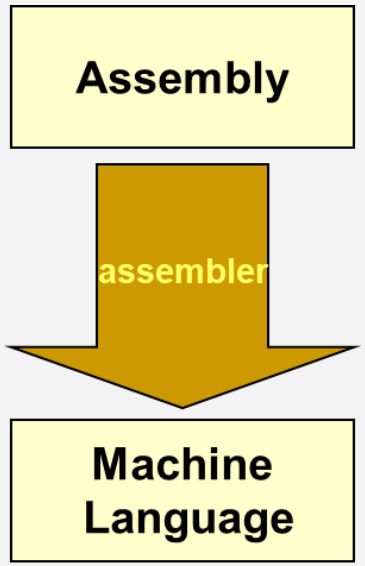
Example

```
;AVR Assembly Language Program To Add Some Data.
;store SUM in SRAM location 0x300.

       .EQU SUM = 0x300     ;SRAM loc $300 for SUM
       .ORG 00              ;start at address 0
       LDI R16, 0x25        ;R16 = 0x25
       LDI R17, $34         ;R17 = 0x34
       LDI R18, 0b00110001  ;R18 = 0x31
       ADD R16, R17         ;add R17 to R16
       ADD R16, R18         ;add R18 to R16
       LDI R17, 11          ;R17 = 0x0B
       ADD R16, R17         ;add R17 to R16
       STS SUM, R16        ;save the SUM in loc $300
HERE:  JMP HERE             ;stay here forever
```

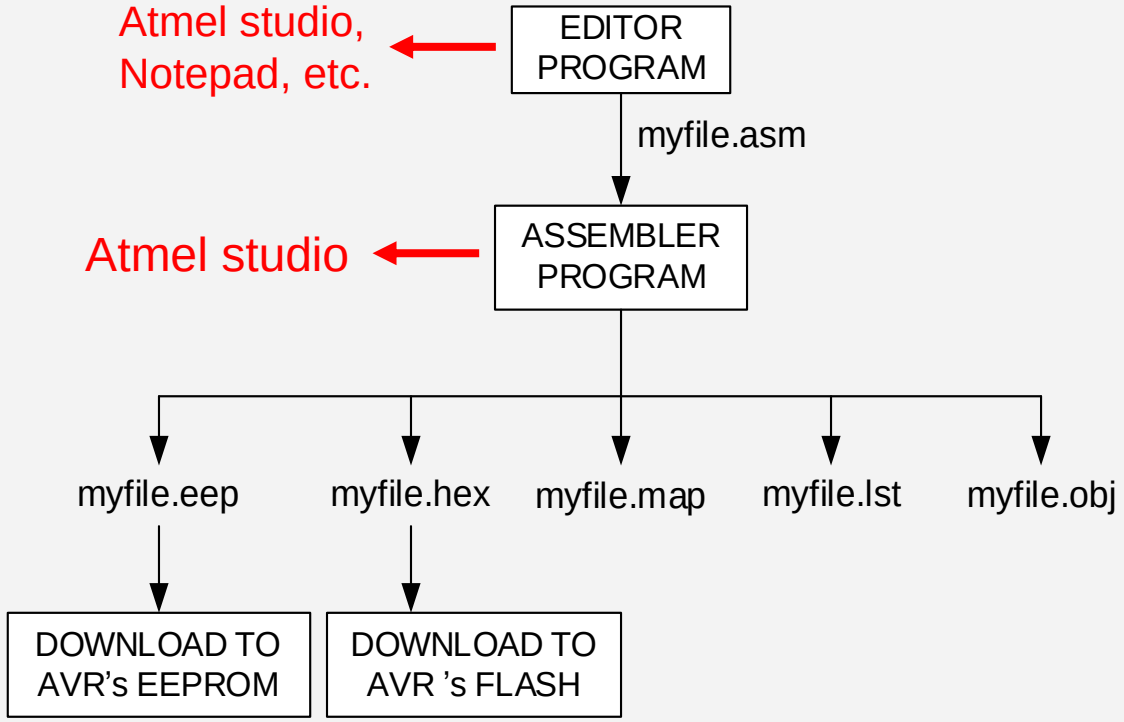
An Assembly language instruction consists of four fields:

```
[label:] mnemonic [operands] [;comment]
```

Assembler



How Assembly Program is created, assembled, and made ready to run?



Files Description

.asm: source file

.obj: is used as input to a simulator or an emulator.

.map: shows the labels defined in the program together with their values.

```
AVRASM ver. 2.1.2  F:\AVR\Sample\Sample.asm Sun Apr 06 23:39:32 2008


EQU   SUM           00000300
CSEG  HERE          00000009
```

Files Description

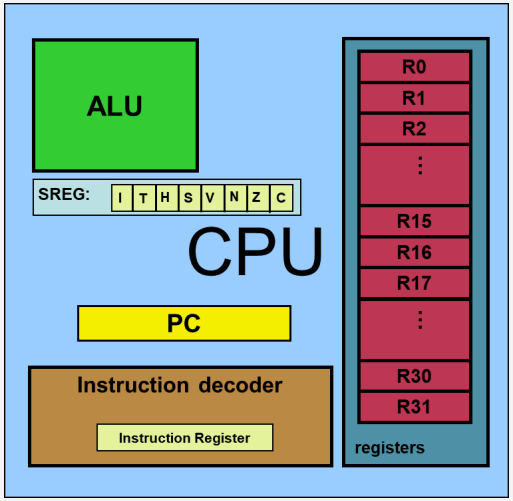
.lst: is very useful to the programmer. The list shows the binary and source code; it also shows which instructions are used in the source code, and the amount of memory the program uses.

```
AVRASM ver. 2.1.2  F:\AVR\Sample\Sample.asm Tue Mar 11 11:28:34 2008

                ;store SUM in SRAM location 0x300.
                   .DEVICE ATMega32
                   .EQU  SUM   = 0x300     ;SRAM loc $300 for SUM

                  .ORG 00                  ;start at address 0
000000 e205       LDI R16, 0x25            ;R16 = 0x25
000001 e314       LDI R17, $34             ;R17 = 0x34
000002 e321       LDI R18, 0b00110001      ;R18 = 0x31
000003 0f01       ADD R16, R17             ;add R17 to R16
000004 0f02       ADD R16, R18             ;add R18 to R16
000005 e01b       LDI R17, 11              ;R17 = 0x0B
000006 0f01       ADD R16, R17             ;add R17 to R16
000007 9300 0300  STS SUM, R16             ;save the SUM in loc $300
000009 940c 0009 HERE: JMP HERE            ;stay here forever

RESOURCE USE INFORMATION
------------------------
...
Memory use summary [bytes]:
Segment    Begin     End       Code    Data    Used     Size    Use%
---------------------------------------------------------------------
[.cseg] 0x000000 0x000016      22       0      22 unknown       -
[.dseg] 0x000060 0x000060       0       0       0 unknown       -
[.eseg] 0x000000 0x000000       0       0       0 unknown       -

Assembly complete, 0 errors, 0 warnings
```

Flash memory and PC register

- ✓ The most important register in the AVR microcontrollers
- ✓ Duty is to point to the address of the next instruction to be executed
- ✓ As the CPU fetches the opcode from the program ROM, the program counter is incremented automatically
- ✓ The wider the program counter, the more memory locations a CPU can access.
  Example: a 14-bit program counter can access a maximum of 16K (214 = 16K) program memory locations.
- ✓ In AVR microcontrollers each Flash memory location is 2 bytes wide.

AVR's CPU



Flash memory and PC register

- ✓ The most important register in the AVR microcontrollers
- ✓ Duty is to point to the address of the next instruction to be executed
- ✓ As the CPU fetches the opcode from the program ROM, the program counter is incremented automatically
- ✓ The wider the program counter, the more memory locations a CPU can access.
  Example: a 14-bit program counter can access a maximum of 16K (214 = 16K) program memory locations.
- ✓ In AVR microcontrollers each Flash memory location is 2 bytes wide.

Example

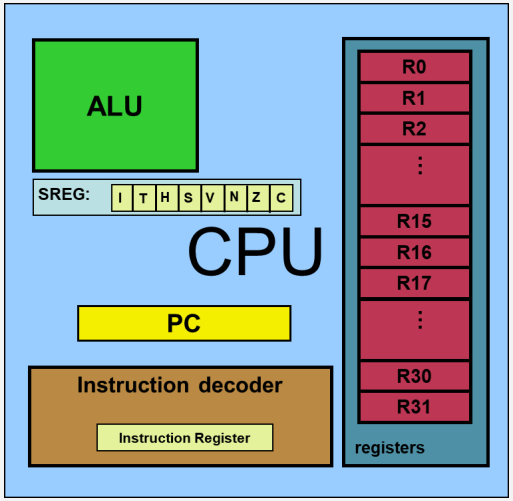
1) for Atmega32 Flash memory is 32k bytes
2) Each flash memory location is 2 bytes wide

So:
32k bytes = 16k × 2 Bytes

which mean 16k=2^{14} locations of memory each is 2 Bytes wide

so PC is 14 bits wide

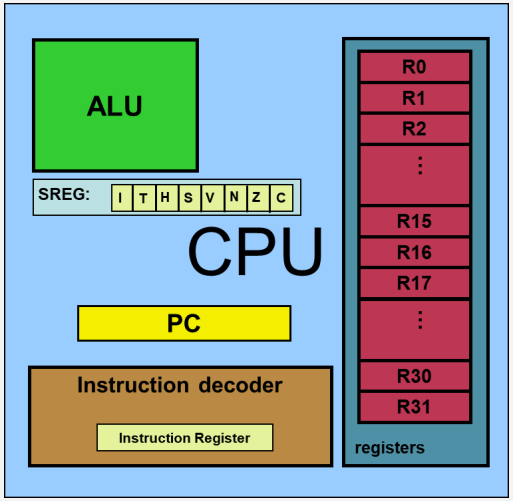
AVR's CPU



Flash memory and PC register

- ✓ The most important register in the AVR microcontrollers
- ✓ Duty is to point to the address of the next instruction to be executed
- ✓ As the CPU fetches the opcode from the program ROM, the program counter is incremented automatically
- ✓ The wider the program counter, the more memory locations a CPU can access.
  Example: a 14-bit program counter can access a maximum of 16K (214 = 16K) program memory locations.
- ✓ In AVR microcontrollers each Flash memory location is 2 bytes wide.

AVR's CPU



Flash memory and PC register

- ✓ The most important register in the AVR microcontrollers
- ✓ Duty is to point to the address of the next instruction to be executed
- ✓ As the CPU fetches the opcode from the program ROM, the program counter is incremented automatically
- ✓ The wider the program counter, the more memory locations a CPU can access.
  Example: a 14-bit program counter can access a maximum of 16K (214 = 16K) program memory locations.
- ✓ In AVR microcontrollers each Flash memory location is 2 bytes wide.

Example

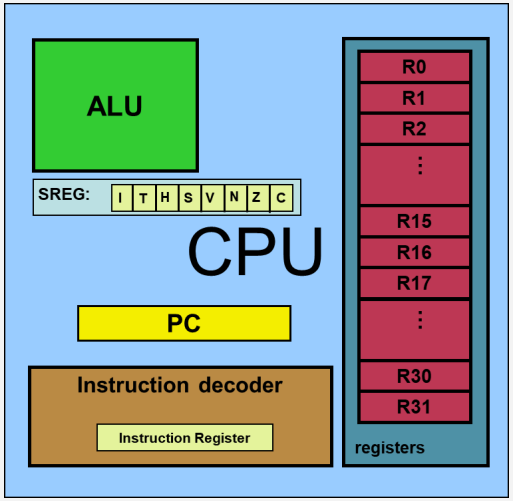
1) for Atmega64 Flash memory is 64k bytes
2) Each flash memory location is 2 bytes wide

So:
64k bytes = 32k × 2 Bytes

which mean 32k=2^{15} locations of memory each is 2 Bytes wide

so PC is 15 bits wide

AVR's CPU



Example

Find the ROM memory address of each of the following AVR chips:

(a) ATtiny25 with 2 KB
(b) ATmega16 with 16 KB
(c) ATmega64 with 64 KB

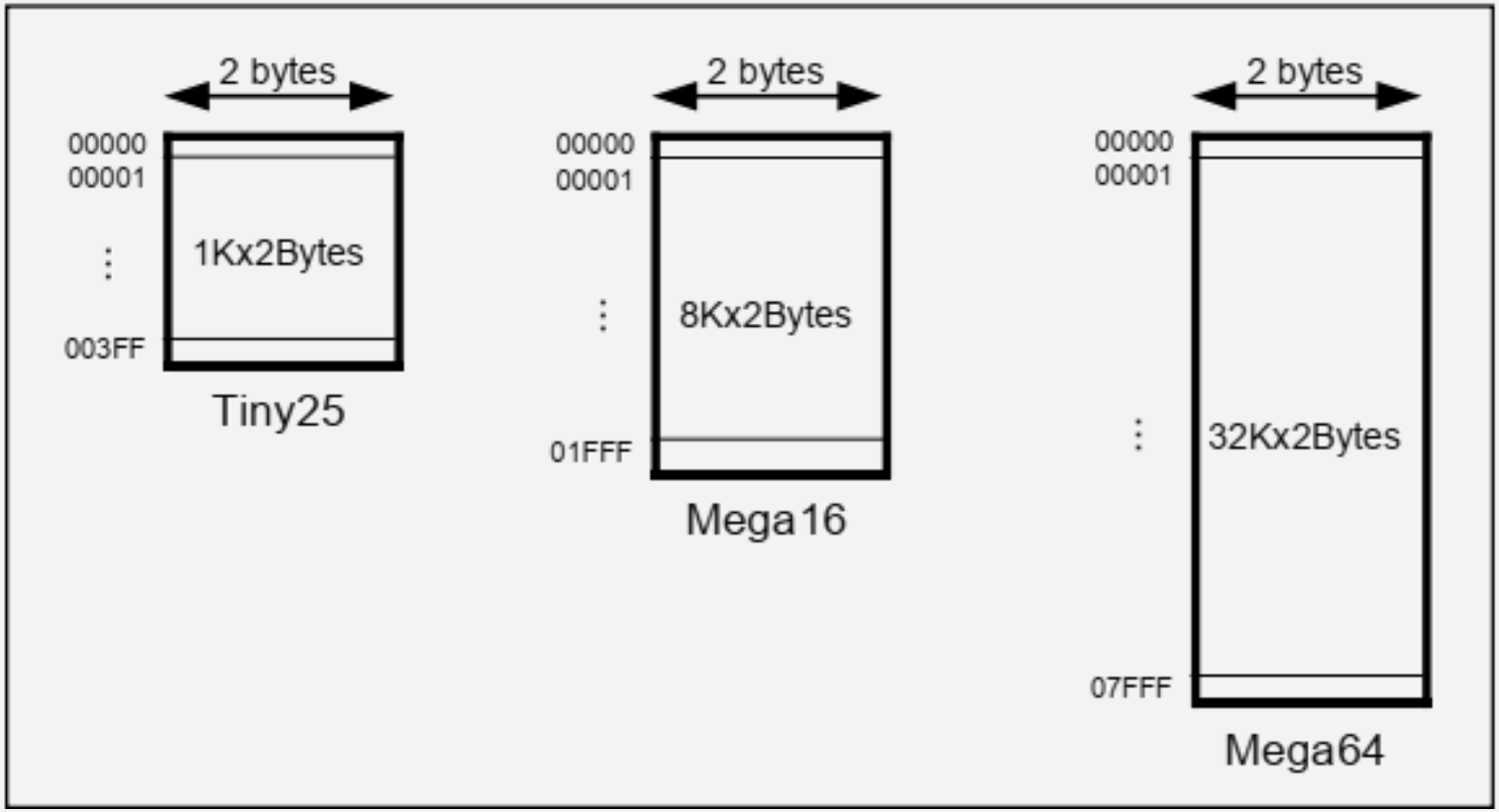
Solution:

(a) With 2K bytes of on-chip ROM memory, we have 2048 bytes (2 × 1024 = 2048). As each address location in AVR is 2 bytes, its Flash has 1024 locations (2048 / 2 = 1024). This maps to address locations of 0000 to $03FF. Notice that 0 is always the first location.

(b) With 16K bytes of on-chip ROM memory, we have 16,384 bytes (16 × 1024 = 16,384), and 8192 locations (16384 / 2 = 8192), which gives 0000–$1FFF.

(c) With 64K we have 65,535 bytes (64 × 1024 = 65,535), and 32,768 locations. Converting 32,768 to hex, we get $8000; therefore, the memory space is 0000 to $7FFF.

Example



- ✓ The first location of program ROM inside the AVR has the address of 000000,
- ✓ The last location can be different depending on the size of the ROM on the chip.
- ✓ In the case of the AVR microcontrollers (that is, all members regardless of the family and variation), the microcontroller wakes up at memory address 0000 when it is powered up.

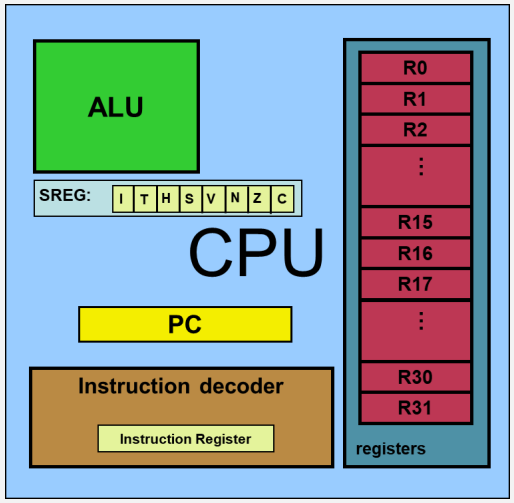
Flash memory and PC register

✓ The program counter in the AVR family can be up to 22 bits wide.

Table 7: AVR On-chip ROM Size and Address Space

	On-chip Code ROM (Bytes)	Code Address Range (Hex)	ROM Organization
ATtiny25	2K	00000–003FF	1K × 2 bytes
ATmega8	8K	00000–00FFF	4K × 2 bytes
ATmega32	32K	00000–03FFF	16K × 2 bytes
ATmega64	64K	00000–07FFF	32K × 2 bytes
ATmega128	128K	00000–0FFFF	64K × 2 bytes
ATmega256	256K	00000–1FFFF	128K × 2 bytes

AVR's CPU



Placing code in program ROM

- ADD R0,R1

  000011 00 0000 0001

  opcode *operand*

- LDI R16, 2
  LDI R17, 3
  ADD R16, R17

  1110 0000 0000 0010
  1110 0000 0001 0011
  0000 1111 0000 0001

Placing code in program ROM

LDI instruction formation

```
LDI Rd, K    ;load register Rd with value K
```

1110	KKKK	dddd	KKKK

$16 \le d \le 31, 0 \le K \le 255$

Example:

E 2	0 5

LDI R16, 0x25

E 3	1 4

LDI R17, 0x34

E 3	2 1

LDI R18, 0x31

ADD instruction formation

```
ADD Rd, Rr    ;Add Rd to Rr
```

0000	11rd	dddd	rrrr

$0 \le d \le 31, \ 0 \le r \le 31$

0 F	0 1

ADD R16,R17

0 F	0 2

ADD R16, R18

Placing code in program ROM

- ADD R0,R1

  000011 00 0000 0001

  opcode *operand*

- LDI R16, 2
  LDI R17, 3
  ADD R16, R17

  1110 0000 0000 0010
  1110 0000 0001 0011
  0000 1111 0000 0001

Placing code in program ROM

STS instruction formation

```
STS k, Rr    ;Store register Rr in memory location k
```

1001	001r	rrrr	0000
kkkk	kkkk	kkkk	kkkk

$0 \le r \le 31, 0 \le k \le 65535$

LDS instruction formation

```
LDS Rd, k    ;Load from memory location k to register Rd
```

1001	000d	dddd	0000
kkkk	kkkk	kkkk	kkkk

$0 \le d \le 31, 0 \le k \le 65535$

IN instruction formation

```
IN Rd, A   ;load from Address A of I/O memory into register Rd
```

1011	0AAd	dddd	AAAA

$0 \le d \le 31, 0 \le A \le 63$

Placing code in program ROM

- ADD R0,R1

  000011 00 0000 0001

  opcode *operand*

- LDI R16, 2
  LDI R17, 3
  ADD R16, R17

  1110 0000 0000 0010
  1110 0000 0001 0011
  0000 1111 0000 0001

Placing code in program ROM

OUT instruction formation

```
OUT A, Rr   ;Store register Rr in I/O memory location A
```

1011	1AAr	rrrr	AAAA

$0 \le d \le 31, 0 \le A \le 63$

JMP instruction formation

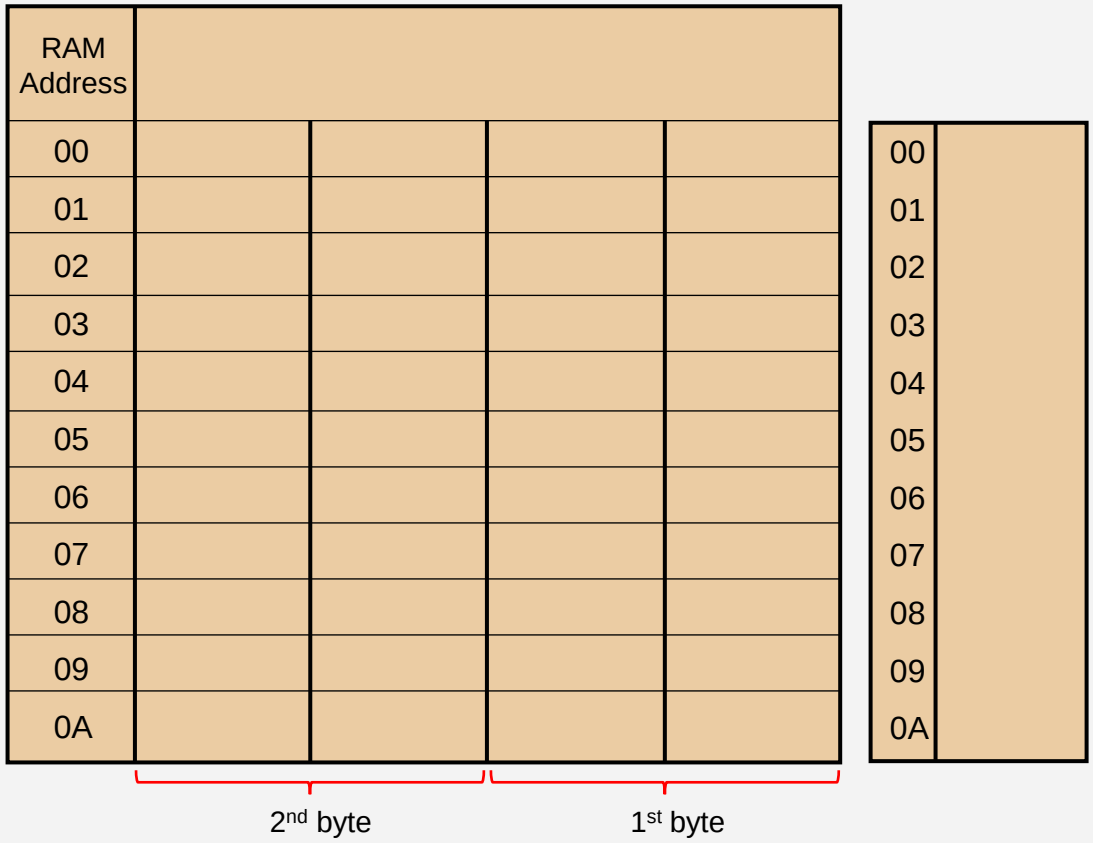
```
JMP k    ;Jump to address k
```

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

$0 \le k \le 4M$

Flash memory and PC register

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



Example:

We want to program this code to Microcontroller's RAM, So convert this Assembly Code to Computer Language (0,1 bits)

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00				
01				
02				
03				
04				
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	
01	
02	
03	
04	
05	
06	
07	
08	
09	
0A	

```
LDI Rd, K    ;load register Rd with value K
```

1110	KKKK	dddd	KKKK

$16 \le d \le 31, 0 \le K \le 255$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01				
02				
03				
04				
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	
02	
03	
04	
05	
06	
07	
08	
09	
0A	

```
LDI Rd, K    ;load register Rd with value K
```

1110	KKKK	dddd	KKKK

$$16 \le d \le 31,\ 0 \le K \le 255$$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01				
02				
03				
04				
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	
02	
03	
04	
05	
06	
07	
08	
09	
0A	

```
LDI Rd, K    ;load register Rd with value K
```

1110	KKKK	dddd	KKKK

$16 \le d \le 31, 0 \le K \le 255$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02				
03				
04				
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	E314
02	
03	
04	
05	
06	
07	
08	
09	
0A	

```
LDI Rd, K    ;load register Rd with value K
```

1110	KKKK	dddd	KKKK

$16 \le d \le 31,\ 0 \le K \le 255$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02				
03				
04				
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	**E205**
01	**E314**
02	
03	
04	
05	
06	
07	
08	
09	
0A	

```
LDI Rd, K    ;load register Rd with value K
```

1110	KKKK	dddd	KKKK

$$16 \le d \le 31,\ 0 \le K \le 255$$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03				
04				
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	E314
02	E321
03	
04	
05	
06	
07	
08	
09	
0A	

```
LDI Rd, K    ;load register Rd with value K
```

1110	KKKK	dddd	KKKK

$16 \le d \le 31,\ 0 \le K \le 255$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03				
04				
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	**E205**
01	**E314**
02	**E321**
03	
04	
05	
06	
07	
08	
09	
0A	

```
ADD Rd, Rr    ;Add Rd to Rr
```

0000	11rd	dddd	rrrr

$$0 \le d \le 31,\ 0 \le r \le 31$$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
04				
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	E314
02	E321
03	0F01
04	
05	
06	
07	
08	
09	
0A	

```
ADD Rd, Rr    ;Add Rd to Rr
```

0000	11rd	dddd	rrrr

$$0 \le d \le 31,\ 0 \le r \le 31$$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
04				
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	**E205**
01	**E314**
02	**E321**
03	**0F01**
04	
05	
06	
07	
08	
09	
0A	

```
ADD Rd, Rr    ;Add Rd to Rr
```

0000	11rd	dddd	rrrr

$$0 \le d \le 31,\ 0 \le r \le 31$$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
04	0 0 0 0	1 1 1 1	0 0 1 0	0 0 1 0
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	E314
02	E321
03	0F01
04	0F02
05	
06	
07	
08	
09	
0A	

```
ADD Rd, Rr    ;Add Rd to Rr
```

0000	11rd	dddd	rrrr

$$0 \le d \le 31,\ 0 \le r \le 31$$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
04	0 0 0 0	1 1 1 1	0 0 1 0	0 0 1 0
05				
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	E314
02	E321
03	0F01
04	0F02
05	
06	
07	
08	
09	
0A	

```
LDI Rd, K    ;load register Rd with value K
```

1110	KKKK	dddd	KKKK

$16 \le d \le 31,\ 0 \le K \le 255$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
04	0 0 0 0	1 1 1 1	0 0 1 0	0 0 1 0
05	1 1 1 0	0 0 0 0	0 0 0 1	1 1 0 1
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	E314
02	E321
03	0F01
04	0F02
05	E01B
06	
07	
08	
09	
0A	

```
LDI Rd, K    ;load register Rd with value K
```

1110	KKKK	dddd	KKKK

$16 \le d \le 31,\ 0 \le K \le 255$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
04	0 0 0 0	1 1 1 1	0 0 1 0	0 0 1 0
05	1 1 1 0	0 0 0 0	0 0 0 1	1 1 0 1
06				
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	**E205**
01	**E314**
02	**E321**
03	**0F01**
04	**0F02**
05	**E01B**
06	
07	
08	
09	
0A	

```
ADD Rd, Rr    ;Add Rd to Rr
```

0000	11rd	dddd	rrrr

$$0 \le d \le 31,\ 0 \le r \le 31$$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address					In hex	
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1	00	E205
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0	01	E314
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1	02	E321
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1	03	0F01
04	0 0 0 0	1 1 1 1	0 0 1 0	0 0 1 0	04	0F02
05	1 1 1 0	0 0 0 0	0 0 0 1	1 1 0 1	05	E01B
06	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1	06	0F01
07					07	
08					08	
09					09	
0A					0A	
	2nd byte		1st byte			

`ADD Rd, Rr  ;Add Rd to Rr`

0000	11rd	dddd	rrrr

$0 \le d \le 31,\ 0 \le r \le 31$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
04	0 0 0 0	1 1 1 1	0 0 1 0	0 0 1 0
05	1 1 1 0	0 0 0 0	0 0 0 1	1 1 0 1
06	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
07				
08				
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	E314
02	E321
03	0F01
04	0F02
05	E01B
06	0F01
07	
08	
09	
0A	

```
JMP k    ;Jump to address k
```

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

$0 \le k \le 4M$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
04	0 0 0 0	1 1 1 1	0 0 1 0	0 0 1 0
05	1 1 1 0	0 0 0 0	0 0 0 1	1 1 0 1
06	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
07	1 0 0 1	0 0 1 1	0 0 0 0	0 0 0 0
08	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	E314
02	E321
03	0F01
04	0F02
05	E01B
06	0F01
07	9300
08	0300
09	
0A	

```
JMP k    ;Jump to address k
```

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

$0 \le k \le 4M$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address				
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
04	0 0 0 0	1 1 1 1	0 0 1 0	0 0 1 0
05	1 1 1 0	0 0 0 0	0 0 0 1	1 1 0 1
06	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1
07	1 0 0 1	0 0 1 1	0 0 0 0	0 0 0 0
08	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0
09				
0A				

2nd byte 1st byte

In hex

00	E205
01	E314
02	E321
03	0F01
04	0F02
05	E01B
06	0F01
07	9300
08	0300
09	
0A	

```
JMP k    ;Jump to address k
```

1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

$0 \le k \le 4M$

Flash memory and PC register

```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```

RAM Address					In hex	
00	1 1 1 0	0 0 1 0	0 0 0 0	0 1 0 1	00	E205
01	1 1 1 0	0 0 1 1	0 0 0 1	0 1 0 0	01	E314
02	1 1 1 0	0 0 1 1	0 0 1 0	0 0 0 1	02	E321
03	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1	03	0F01
04	0 0 0 0	1 1 1 1	0 0 1 0	0 0 1 0	04	0F02
05	1 1 1 0	0 0 0 0	0 0 0 1	1 1 0 1	05	E01B
06	0 0 0 0	1 1 1 1	0 0 1 0	0 0 0 1	06	0F01
07	1 0 0 1	0 0 1 1	0 0 0 0	0 0 0 0	07	9300
08	0 0 0 0	0 0 1 1	0 0 0 0	0 0 0 0	08	0300
09	1 0 0 1	0 1 0 0	0 0 0 0	1 1 0 0	09	940C
0A	0 0 0 0	0 0 0 0	0 0 0 0	1 0 0 1	0A	0009

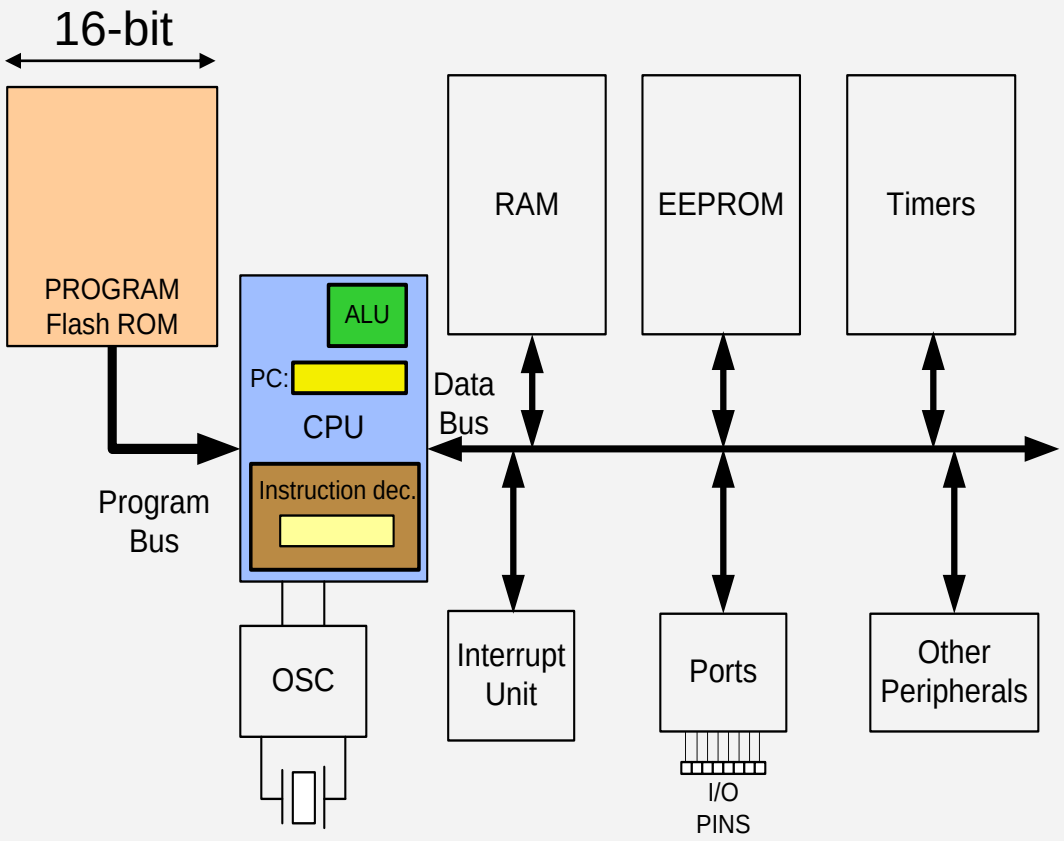
2nd byte 1st byte

```
JMP k    ;Jump to address k
```

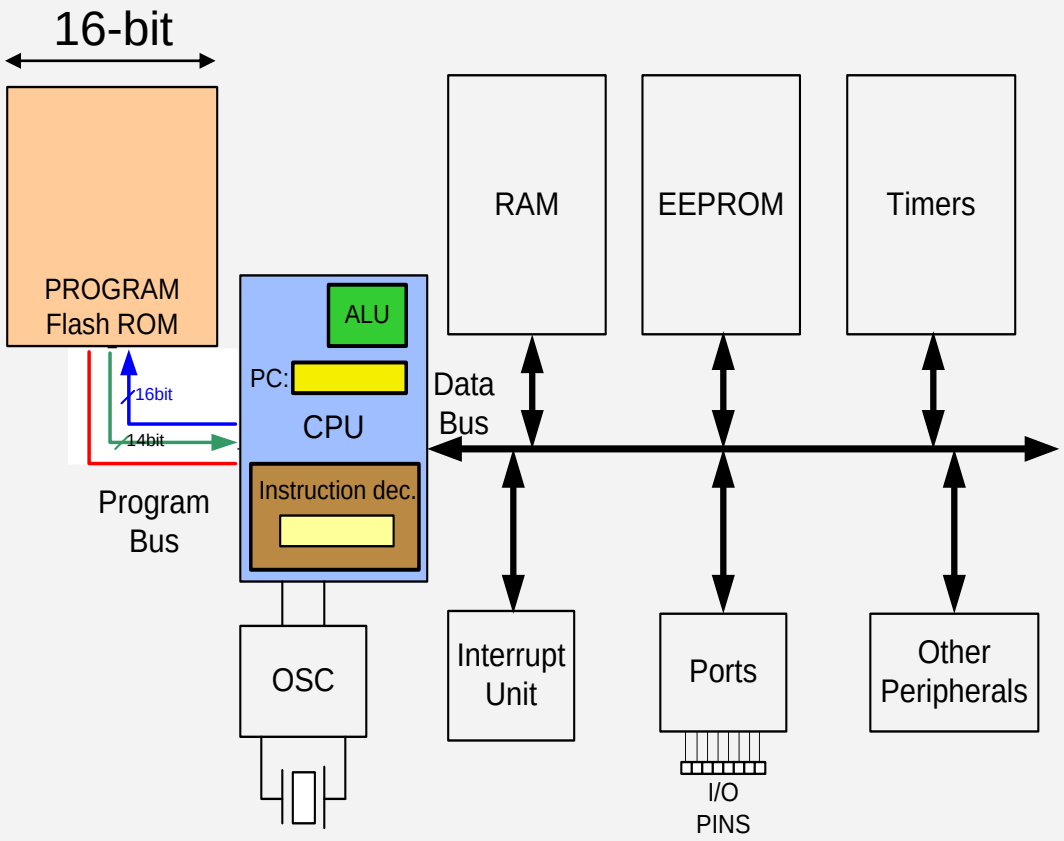
1001	010k	kkkk	110k
kkkk	kkkk	kkkk	kkkk

$0 \le k \le 4M$

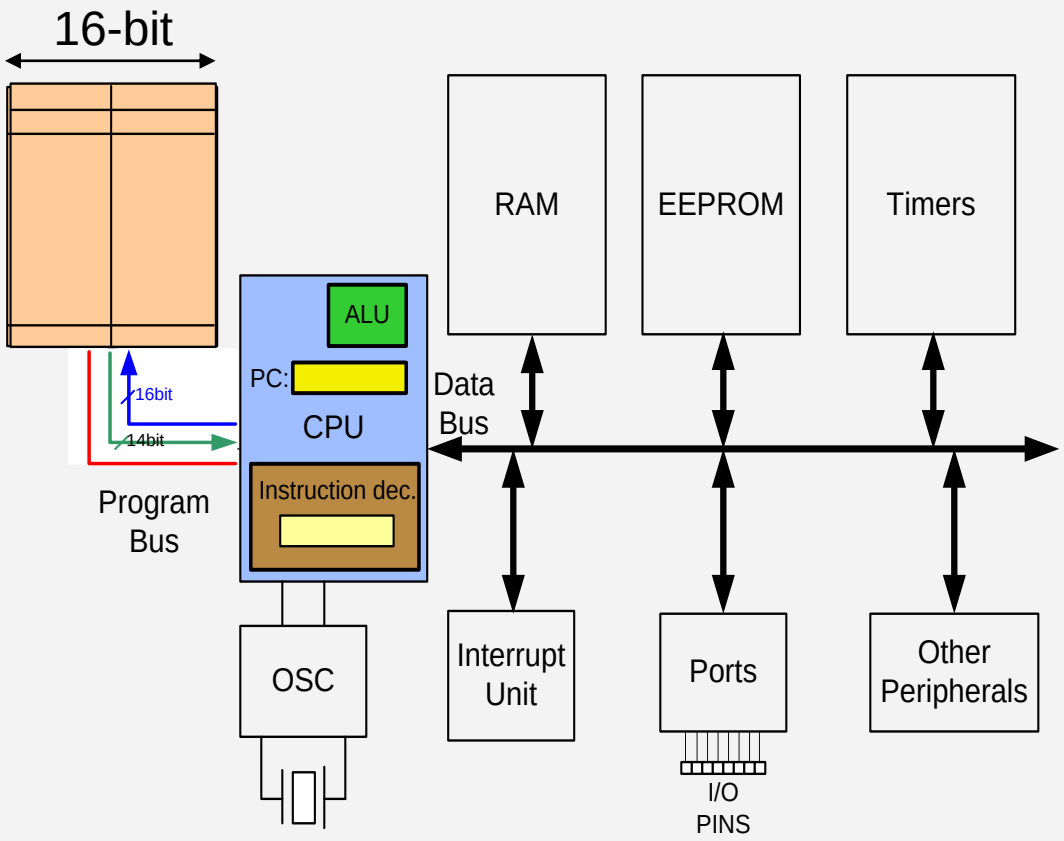
Executing a program instruction by instruction



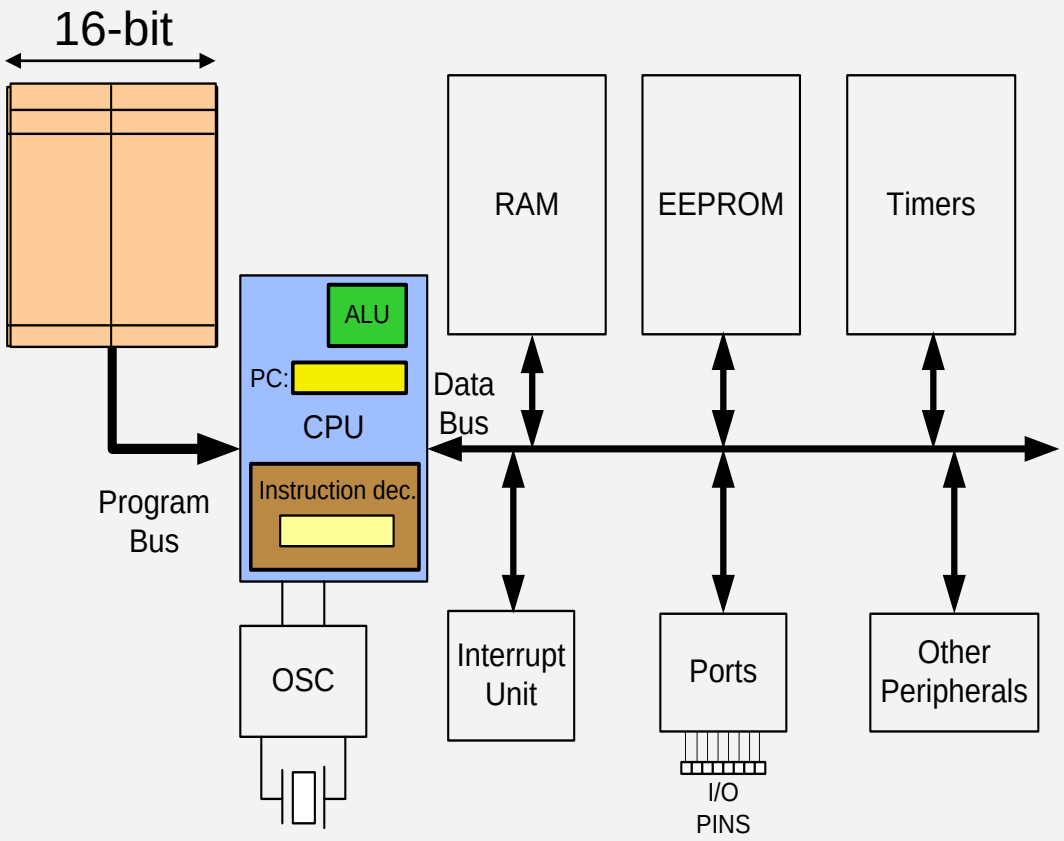
Executing a program instruction by instruction



Executing a program instruction by instruction

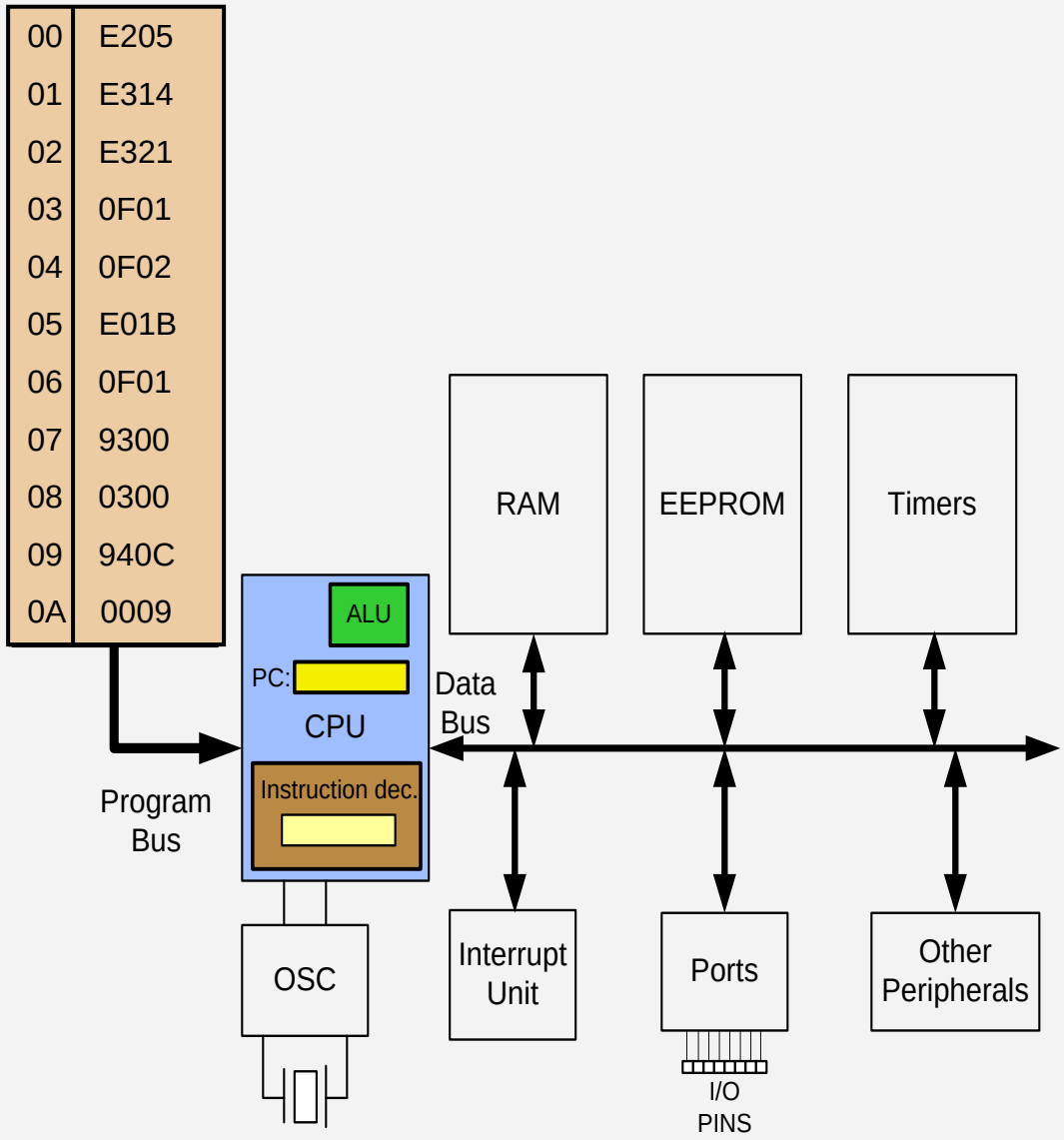


Executing a program instruction by instruction



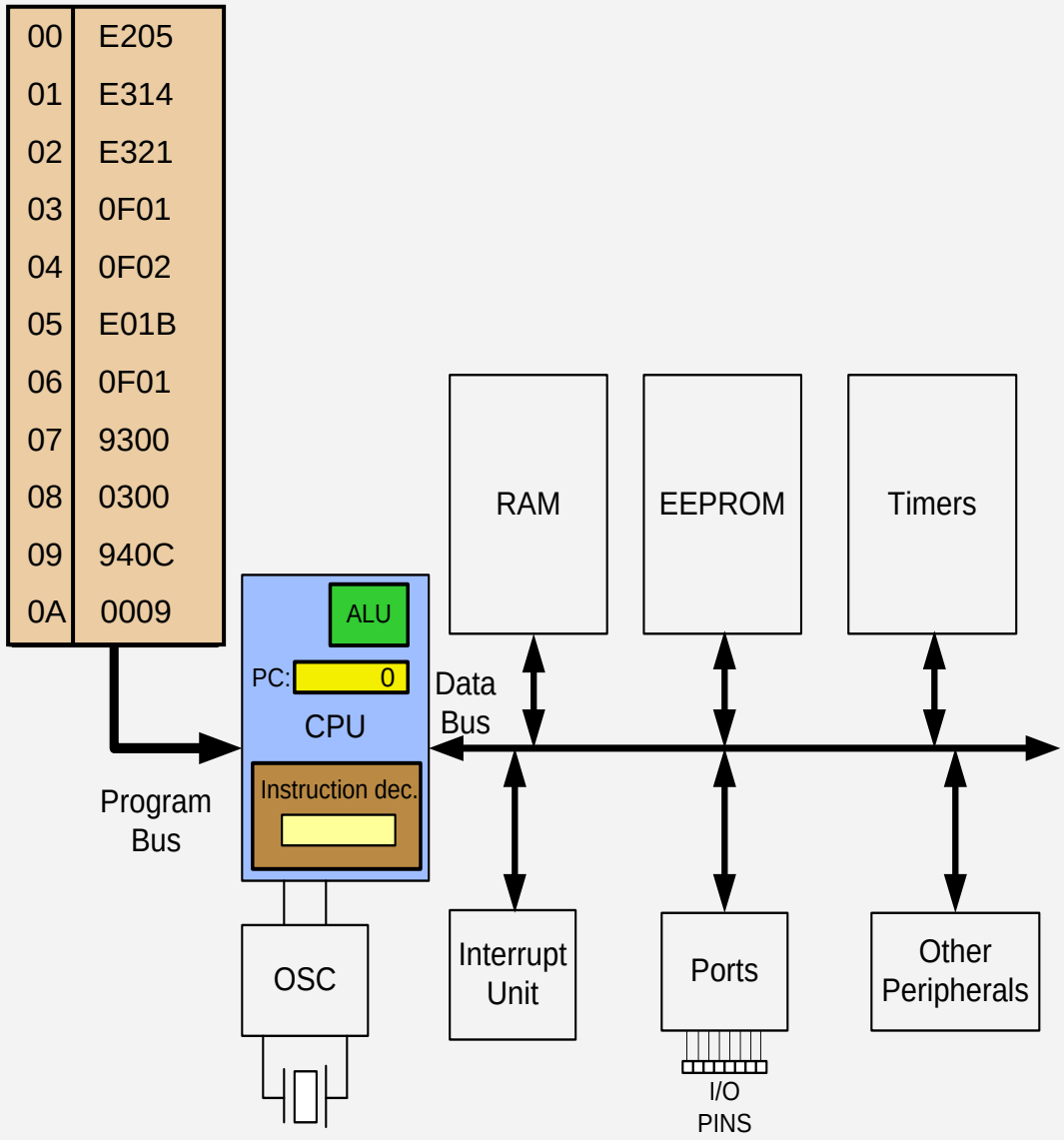
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



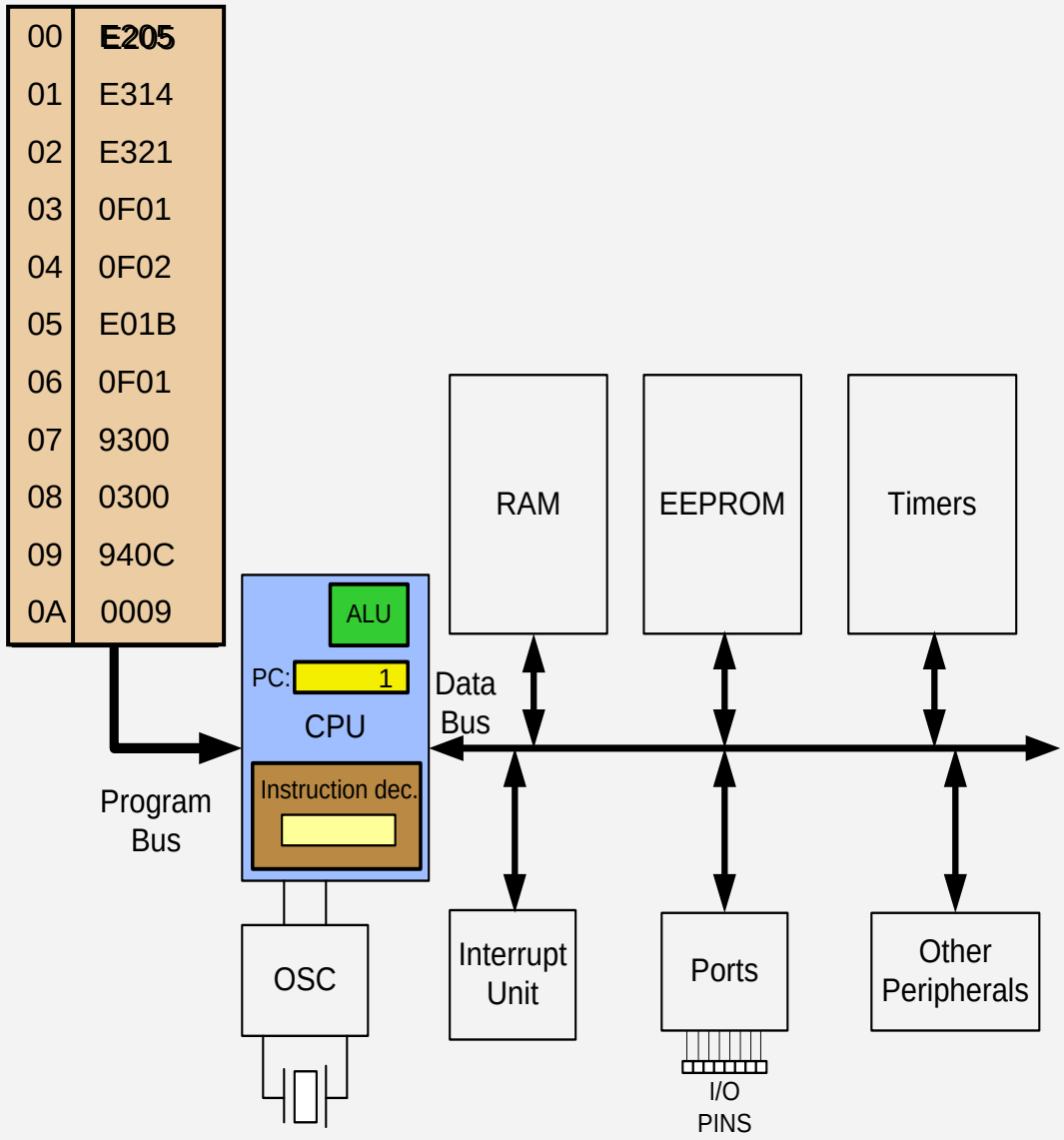
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



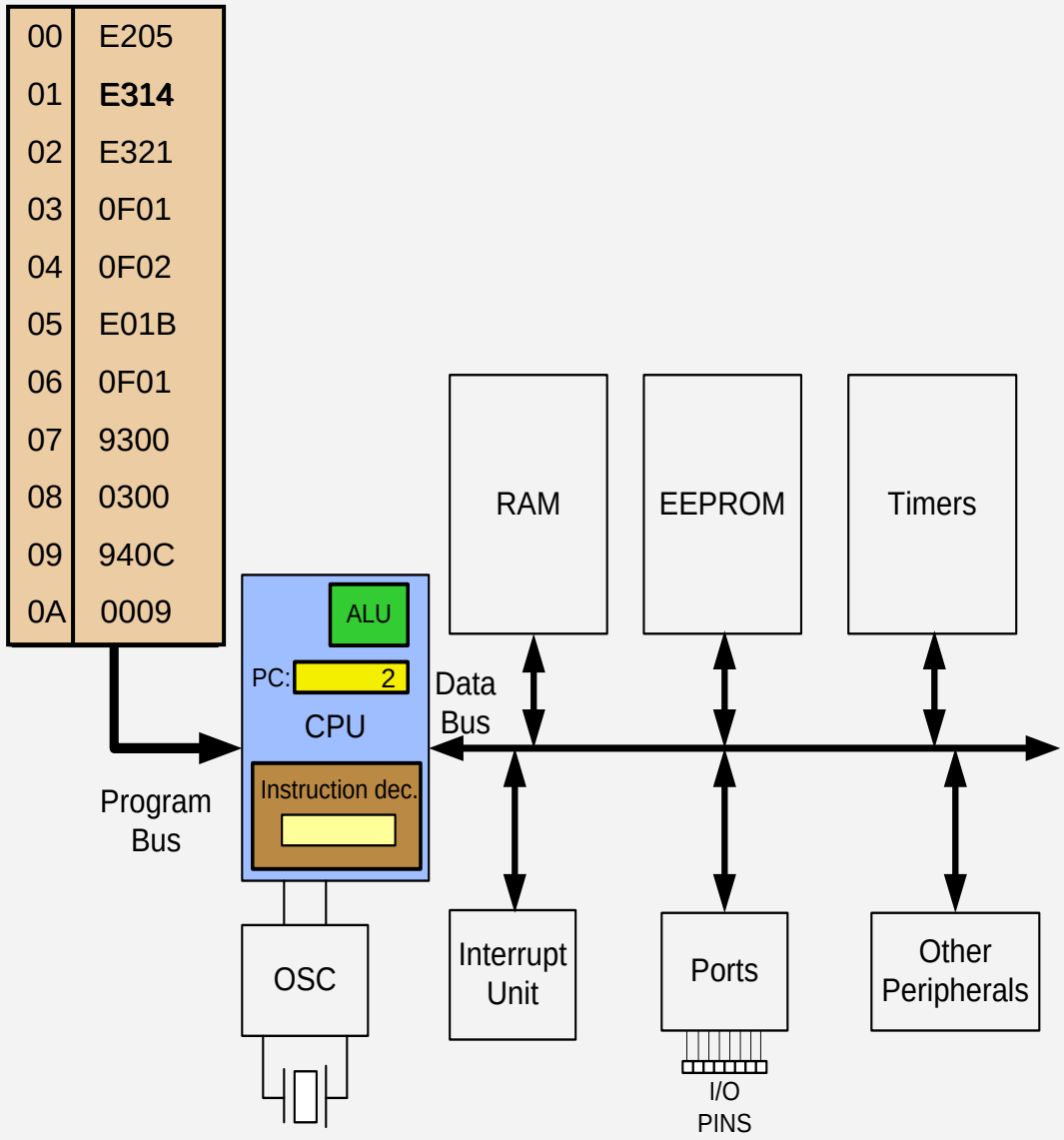
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP   HERE
```



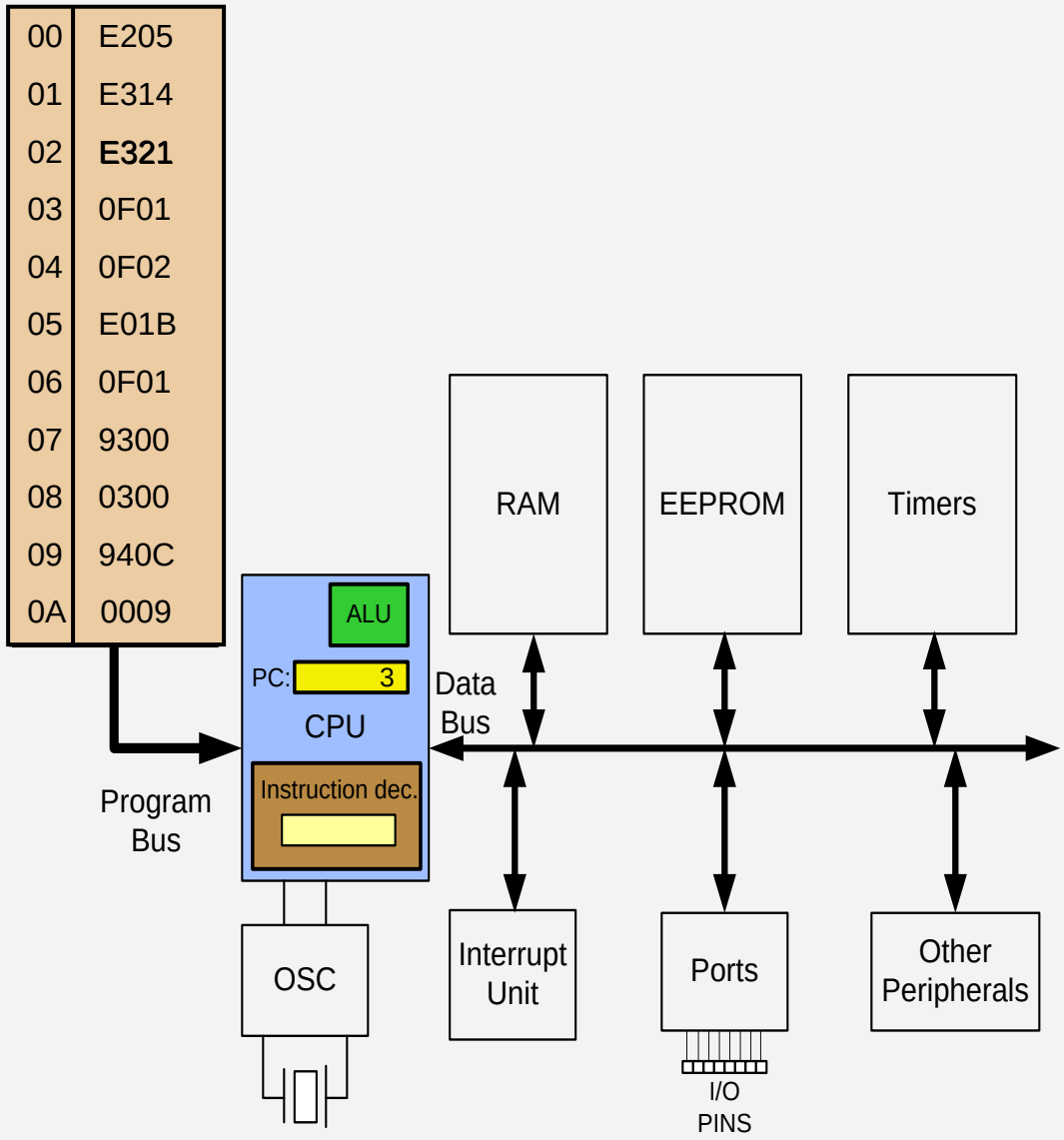
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



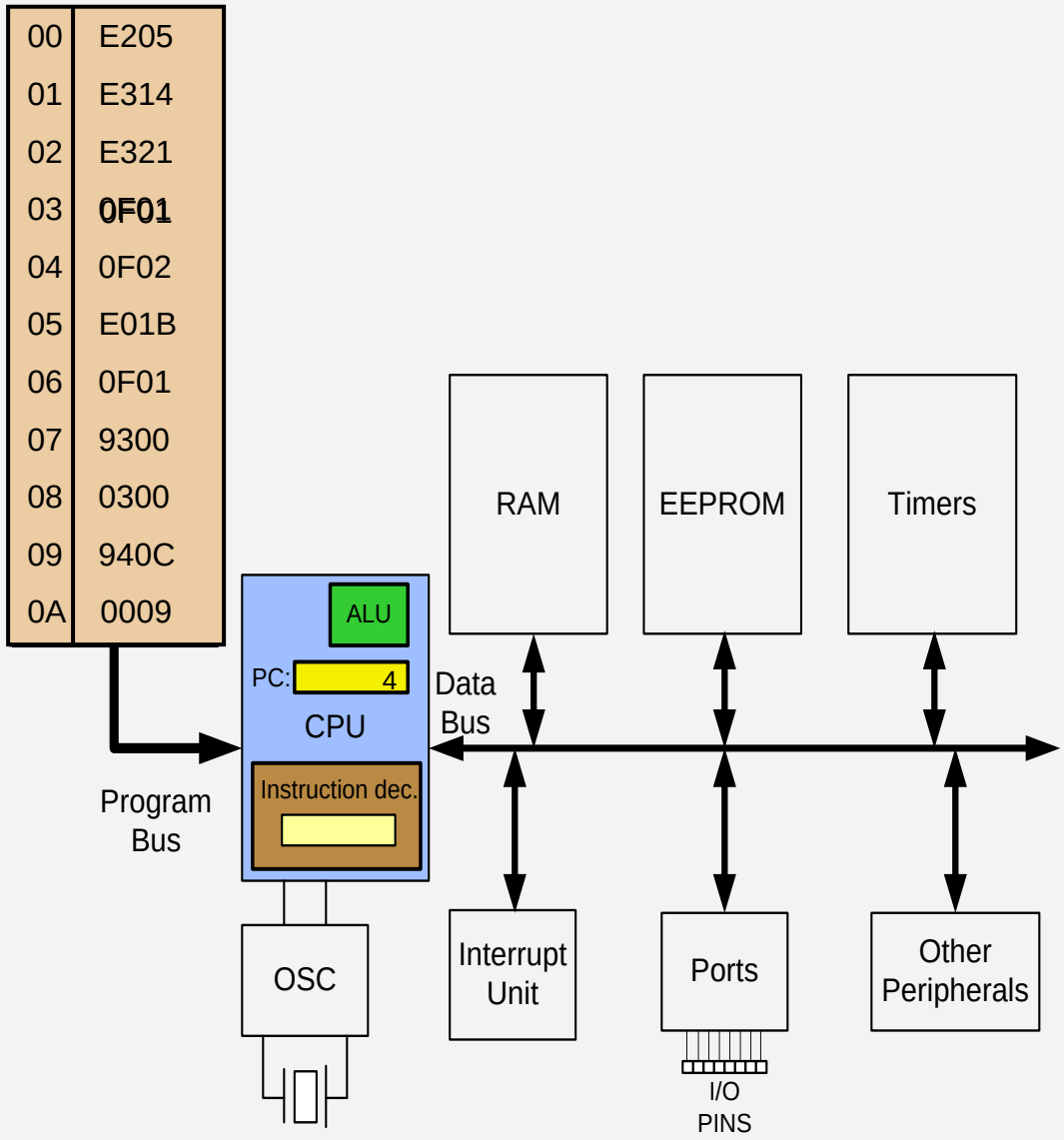
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



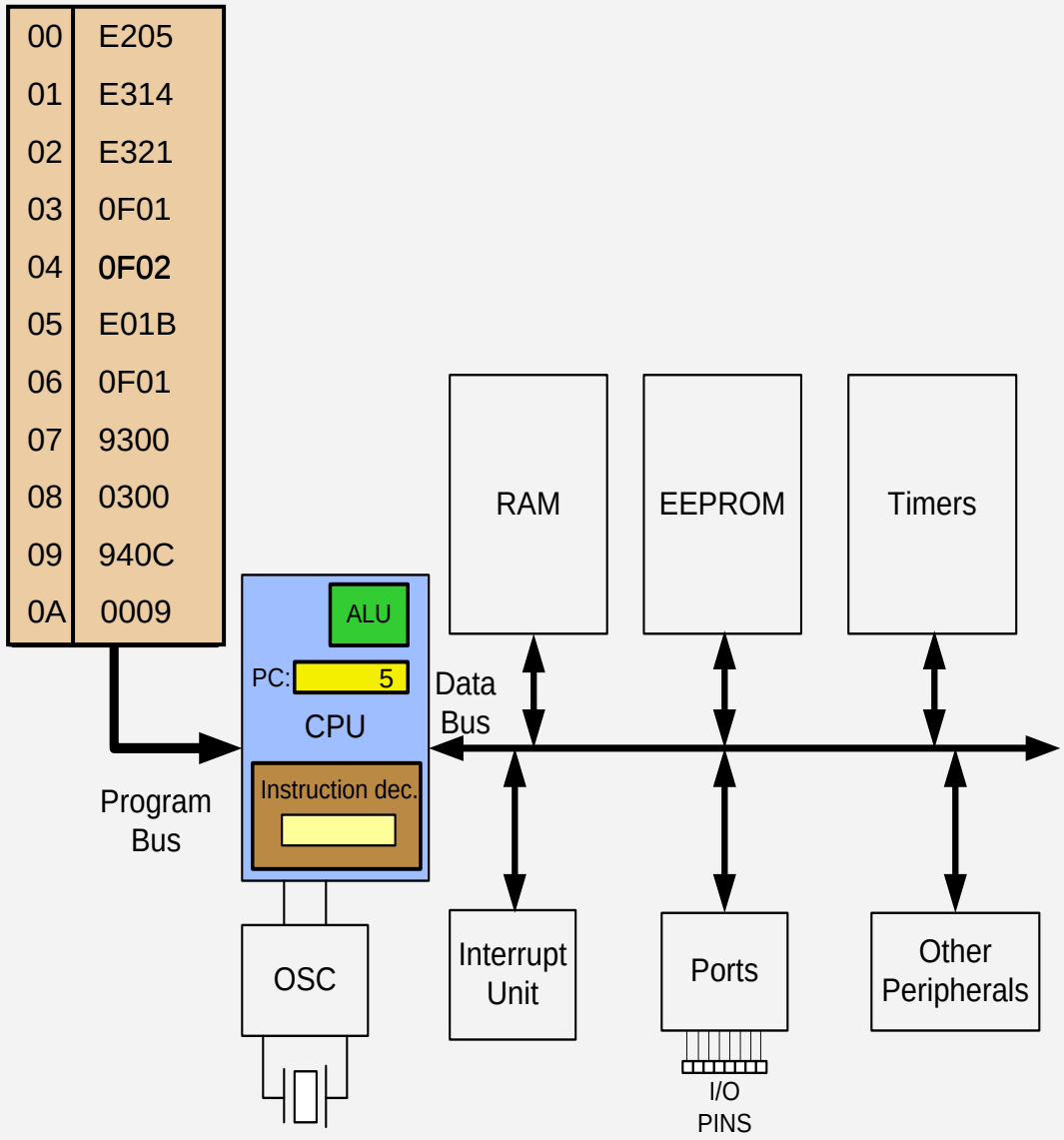
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



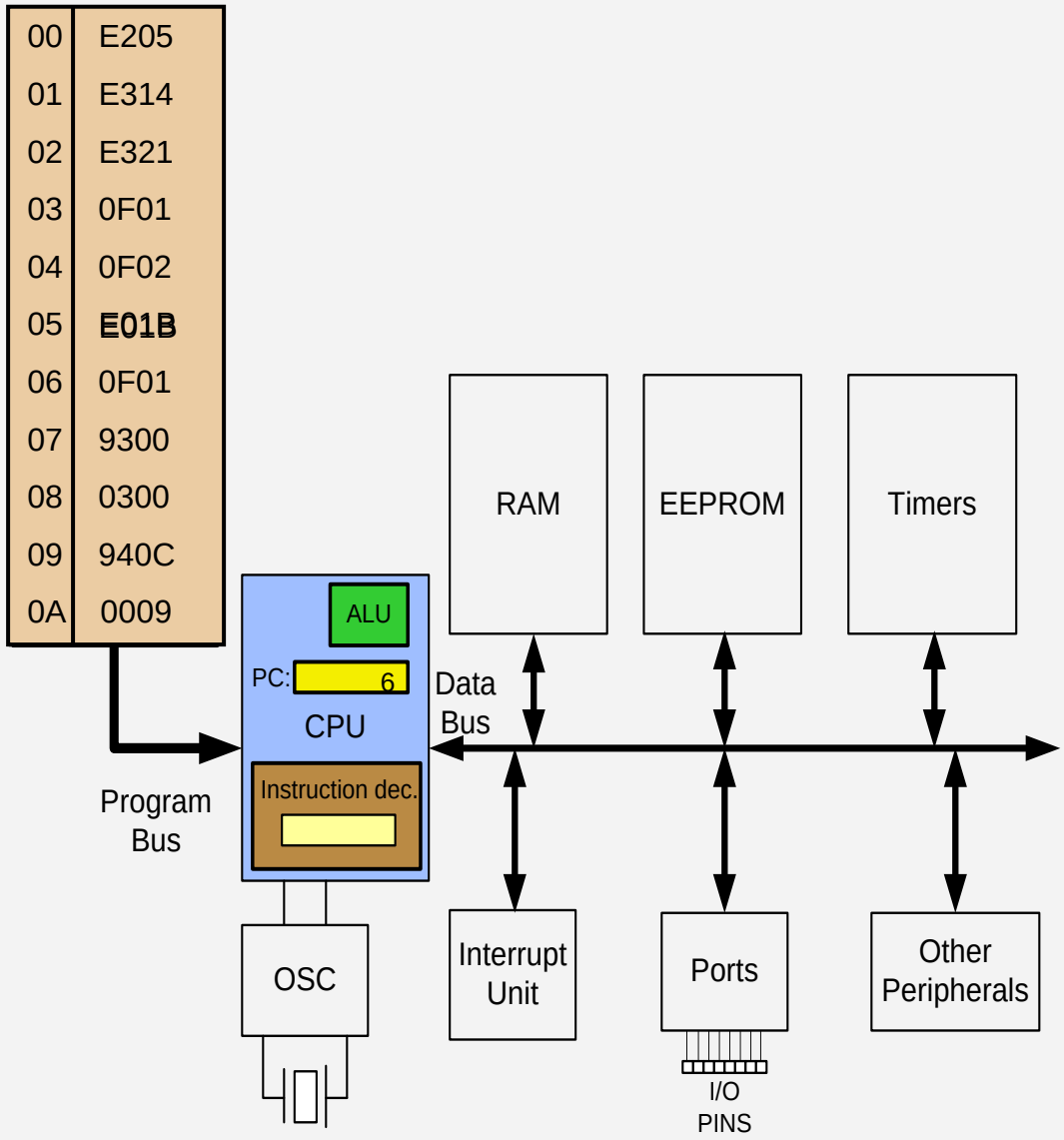
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



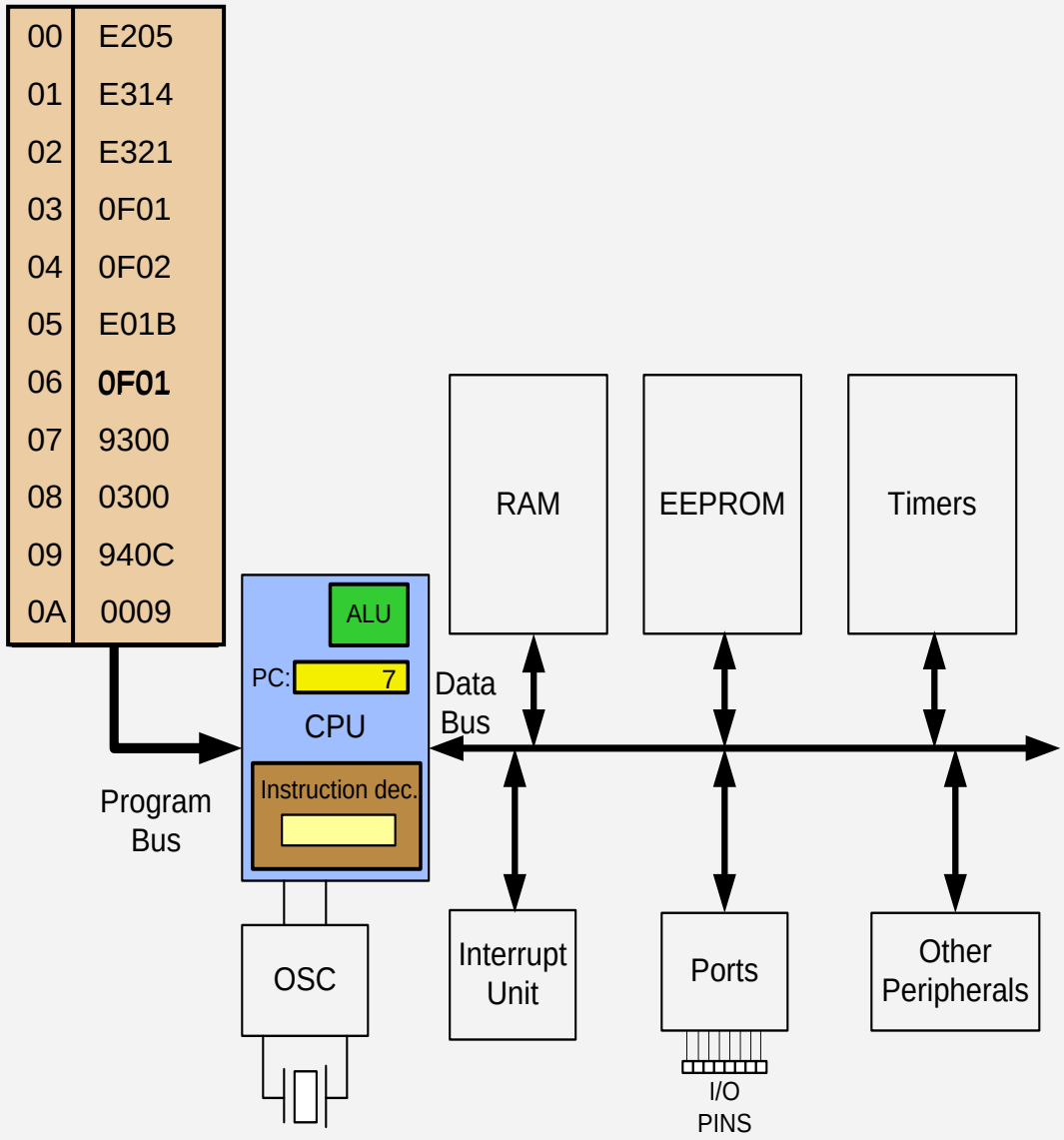
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



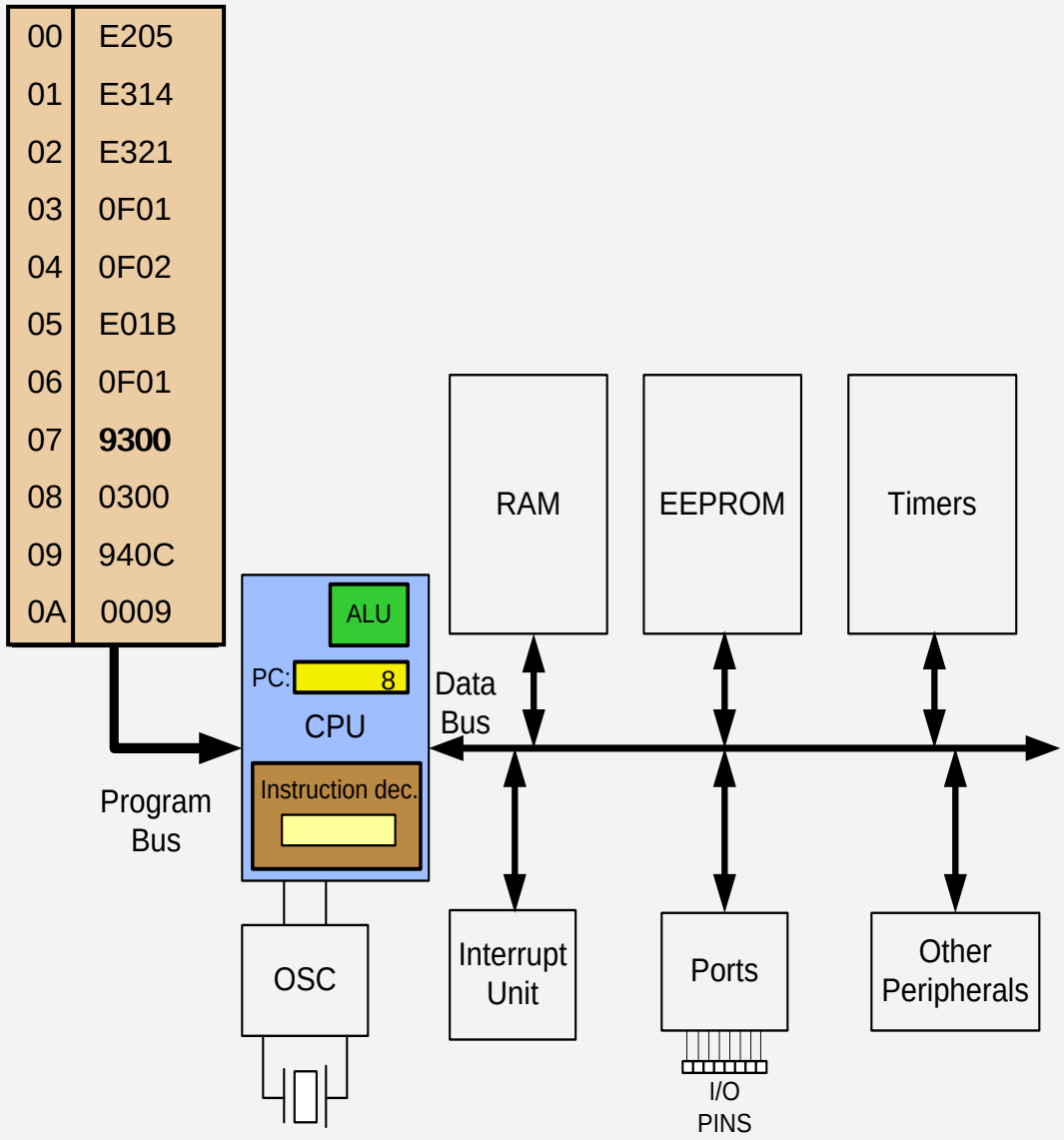
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



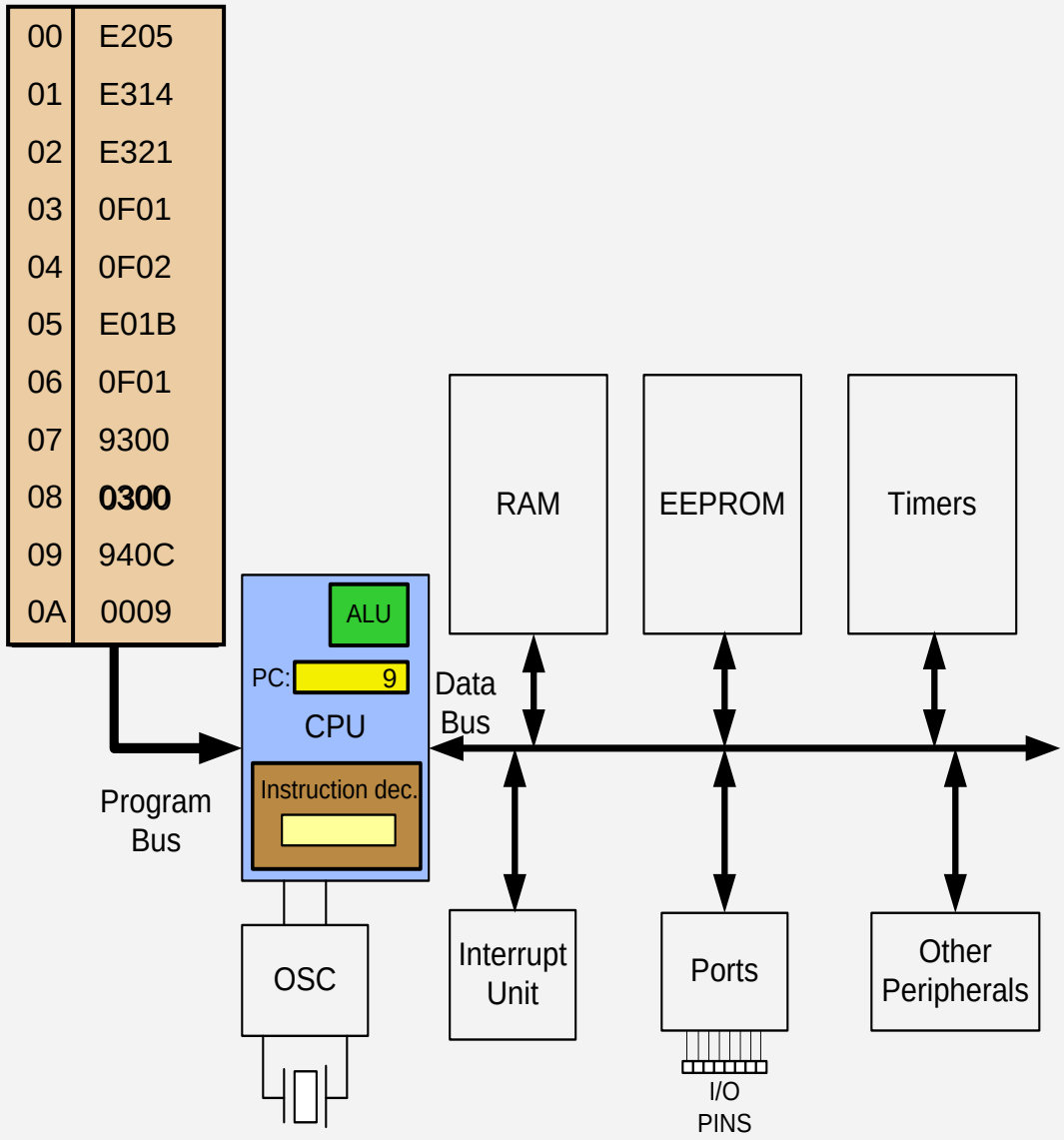
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



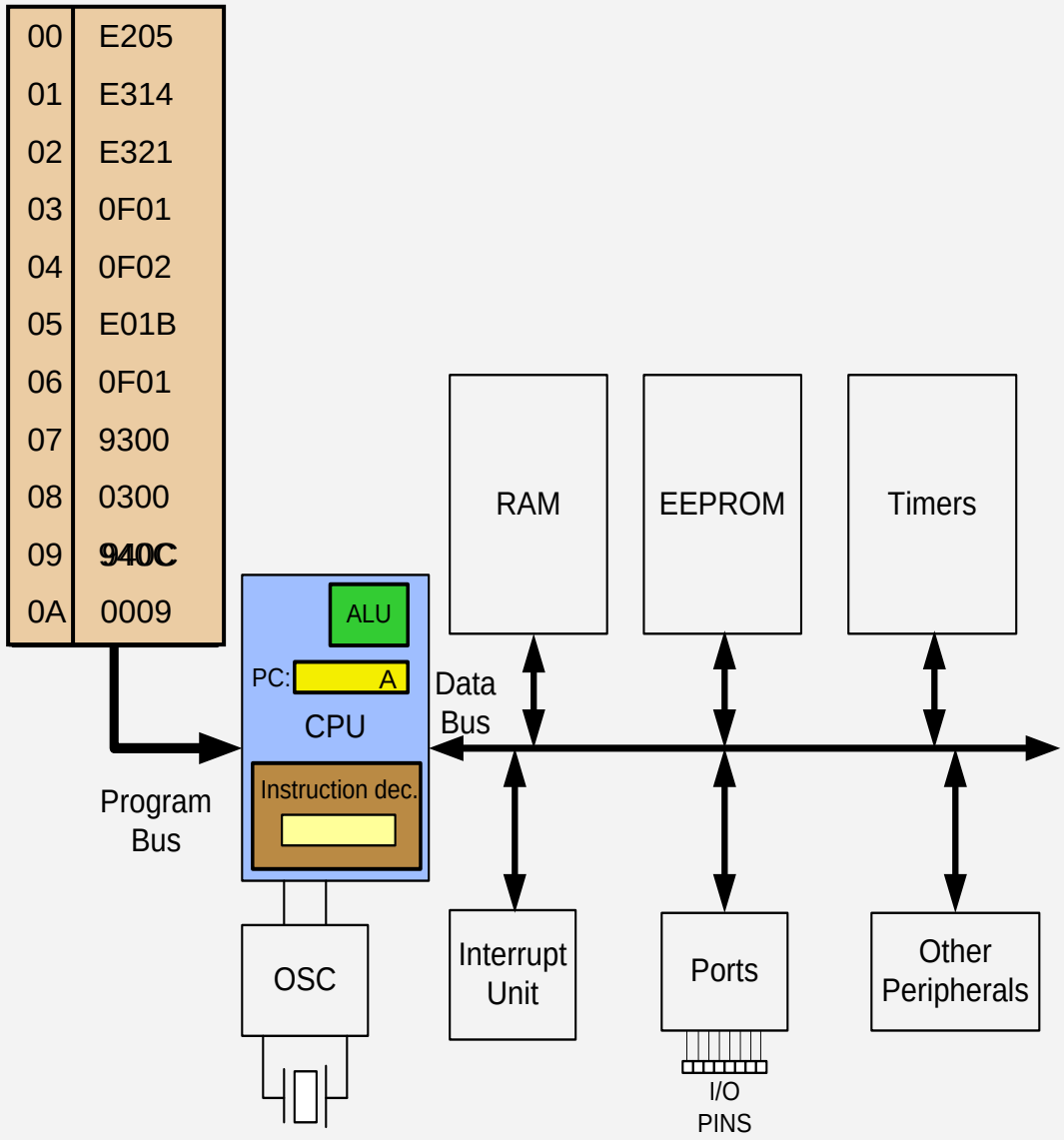
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



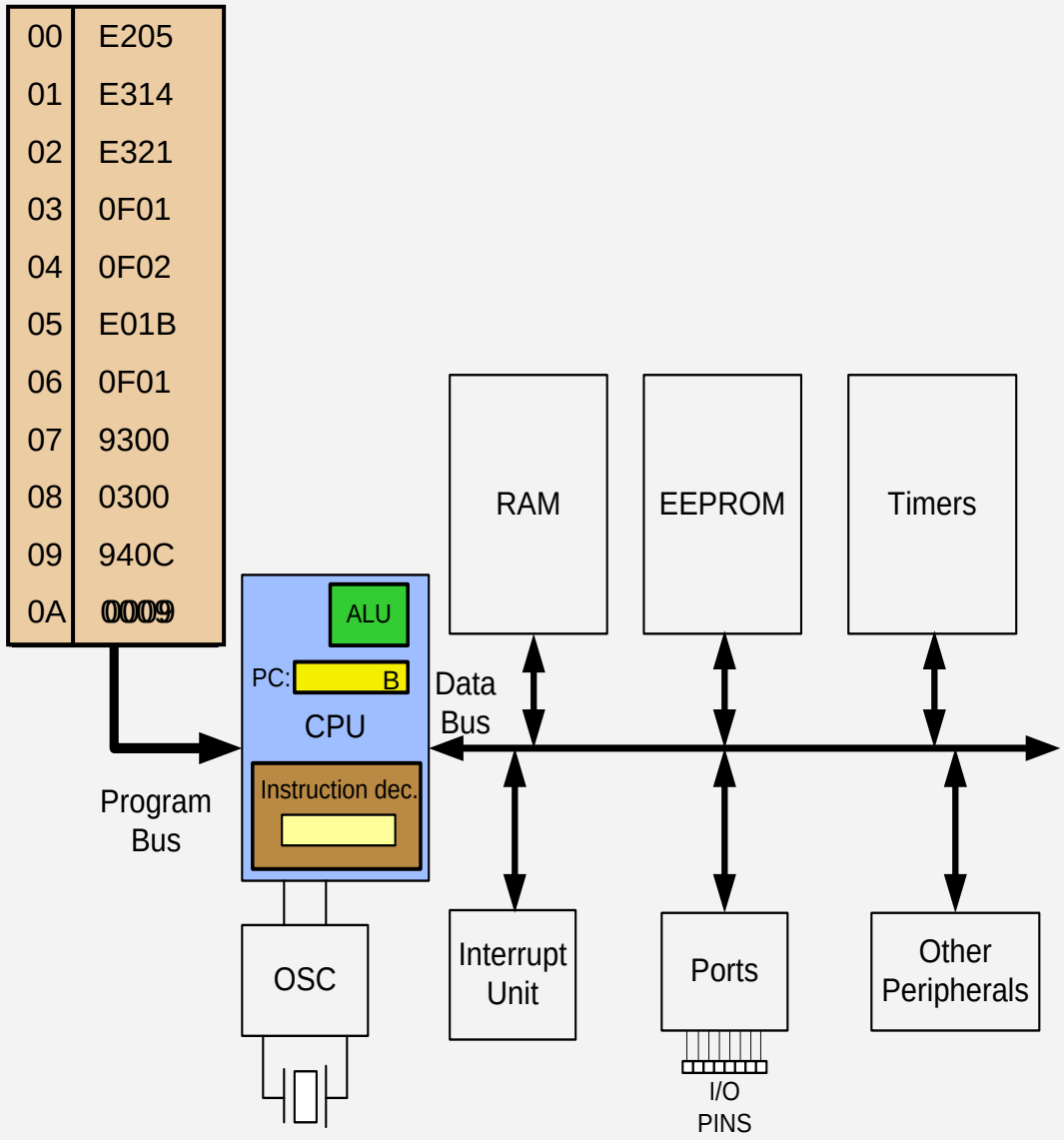
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```



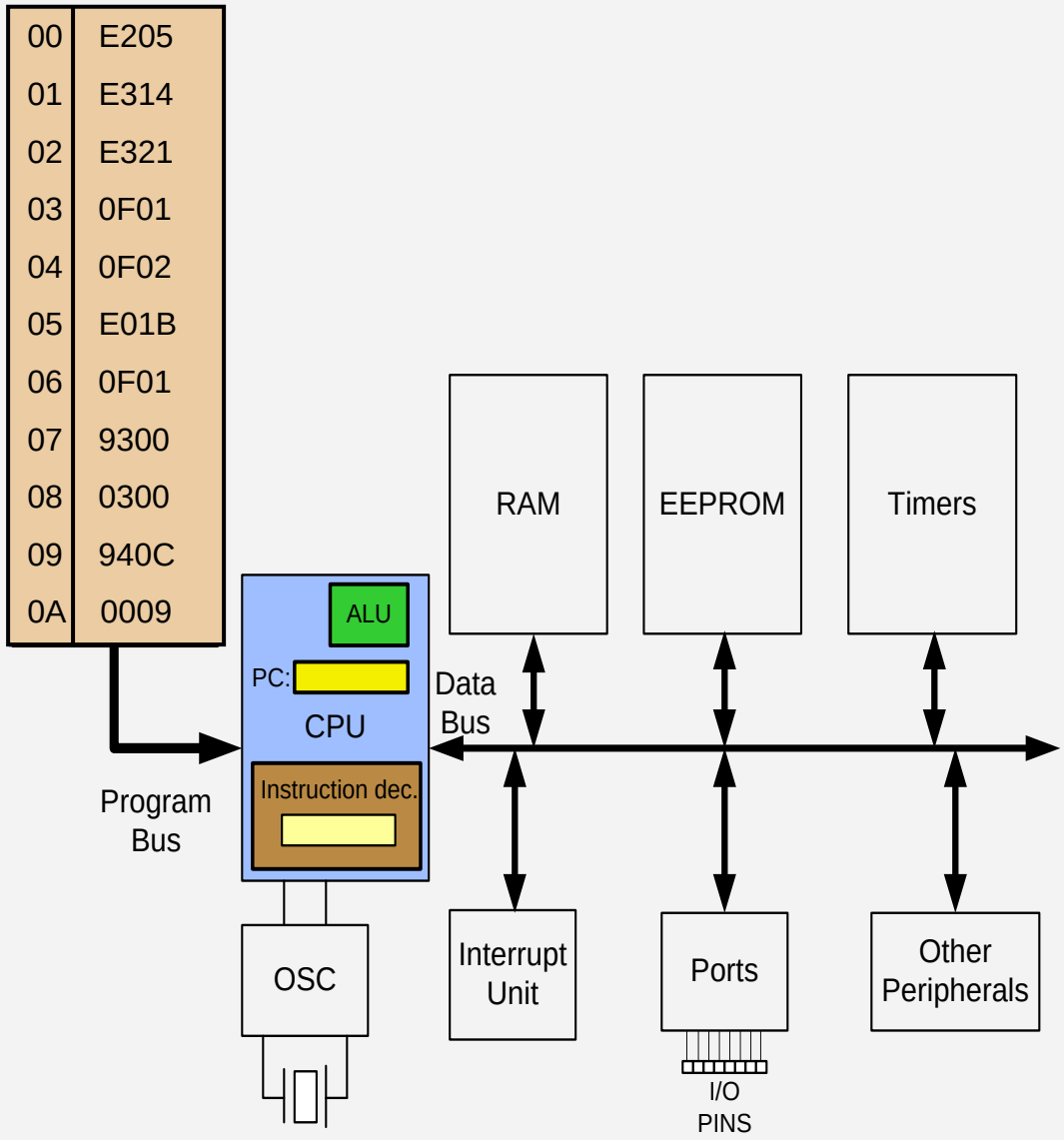
Executing a program instruction by instruction

```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```

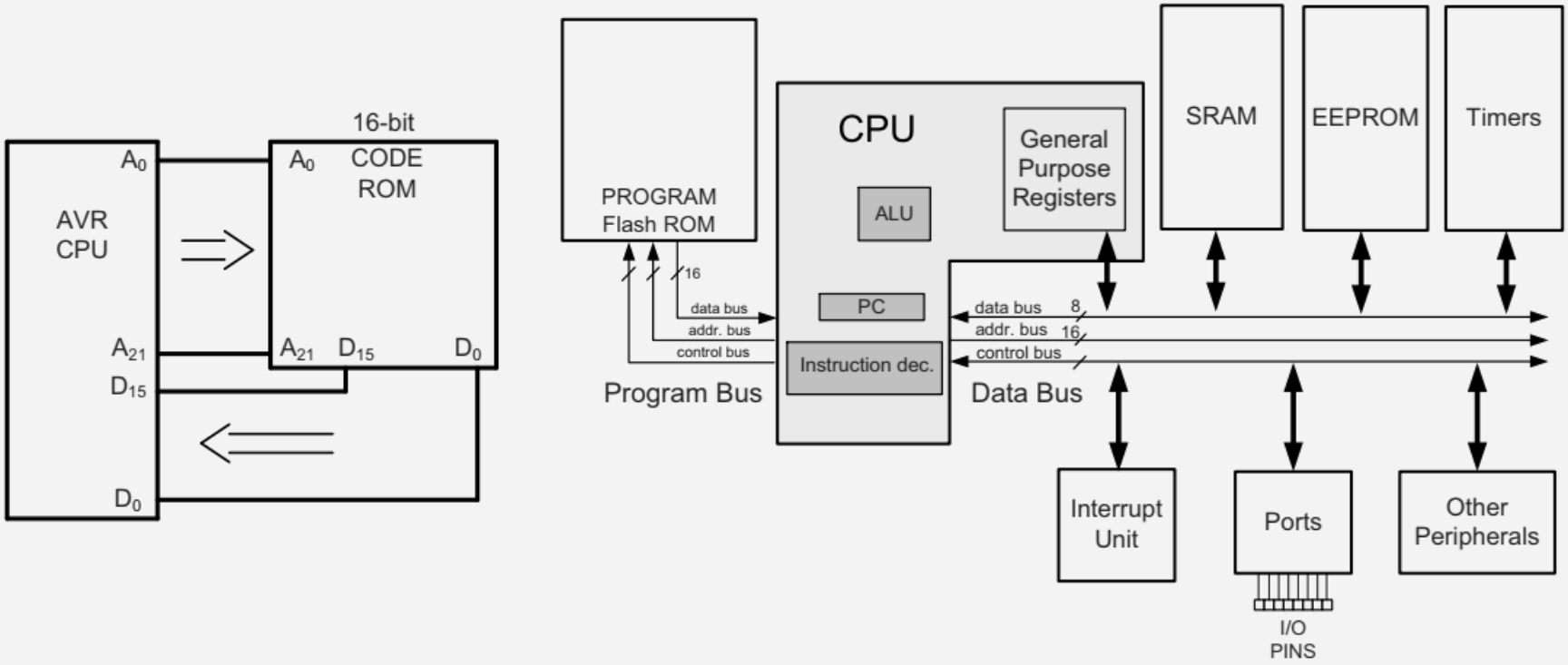


Executing a program instruction by instruction

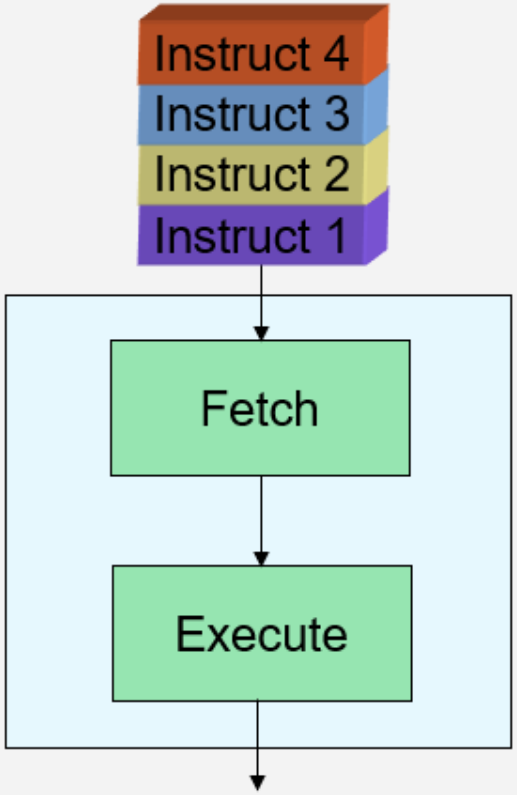
```
      LDI   R16, 0x25
      LDI   R17, $34
      LDI   R18, 0x31
      ADD   R16, R17
      ADD   R16, R18
      LDI   R17, 11
      ADD   R16, R17
      STS   SUM, R16
HERE: JMP  HERE
```

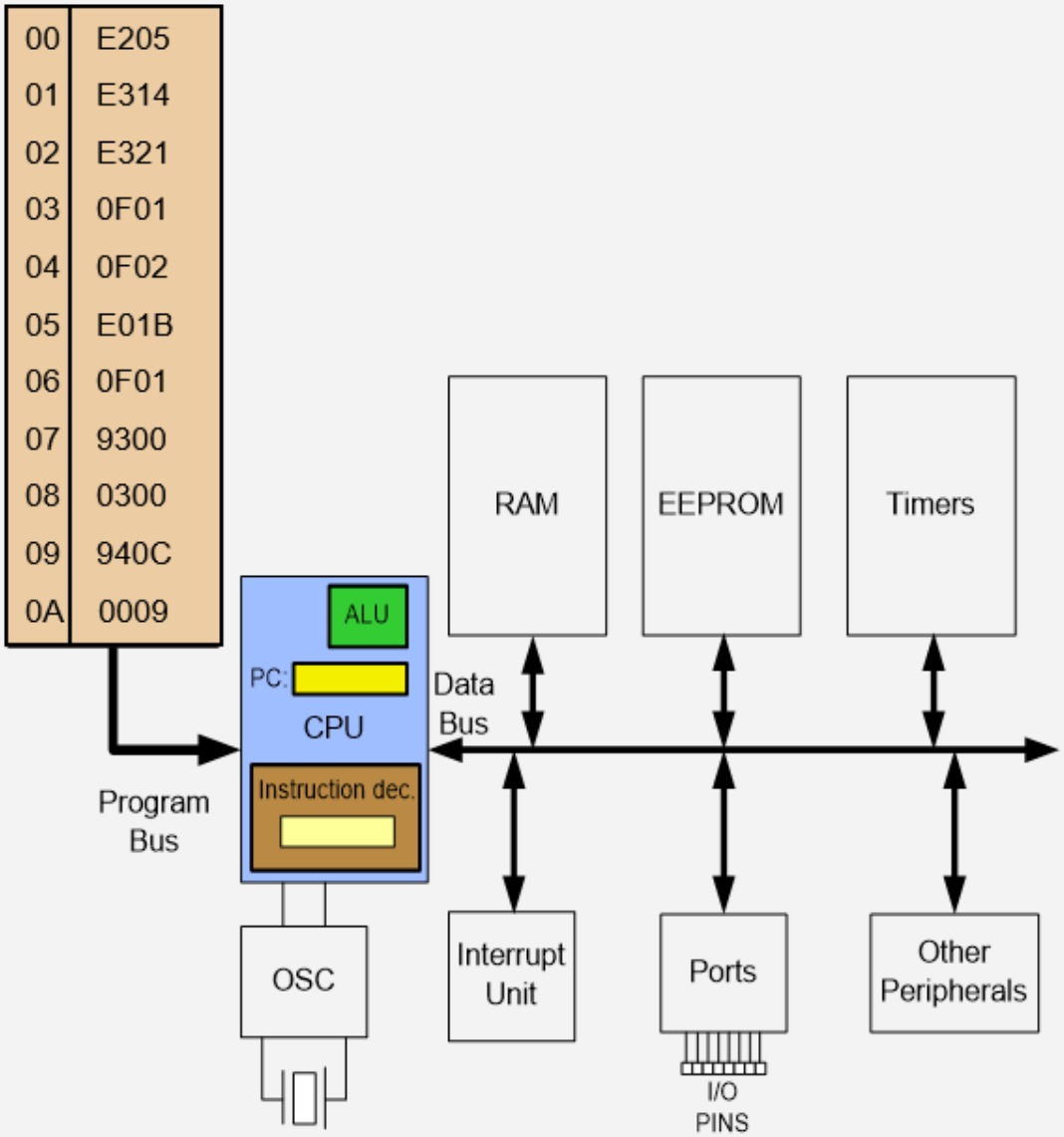


AVR Bus Architecture



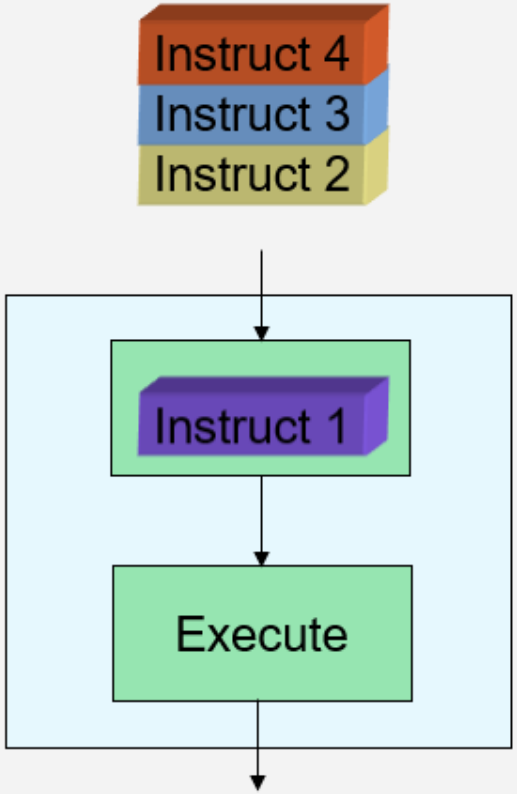
- Old Architectures

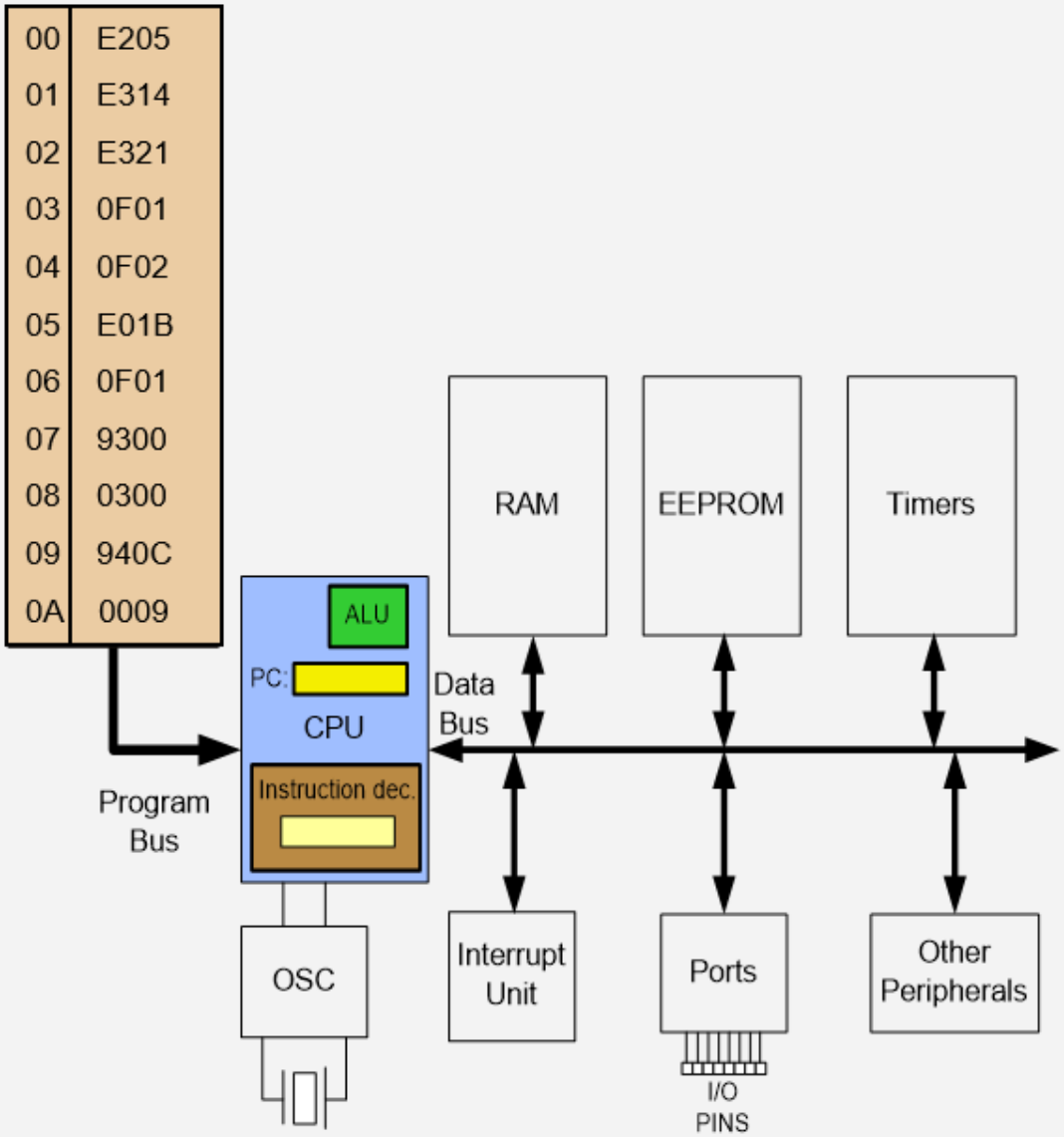




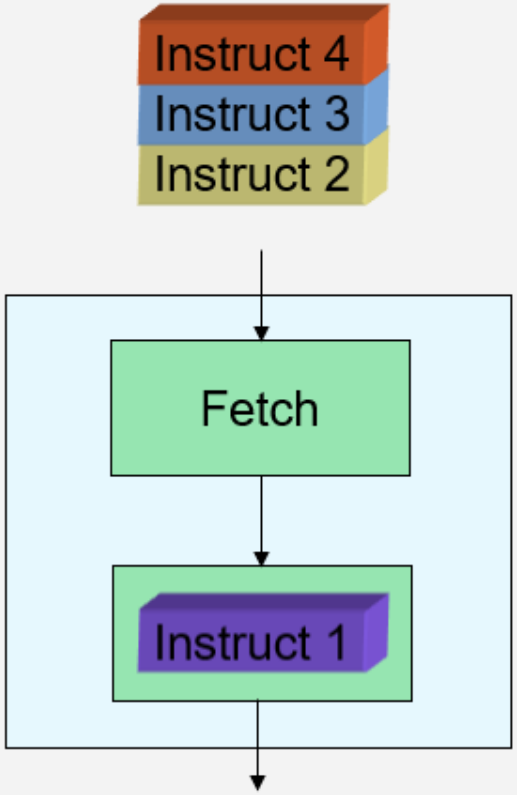
Fetch and execute

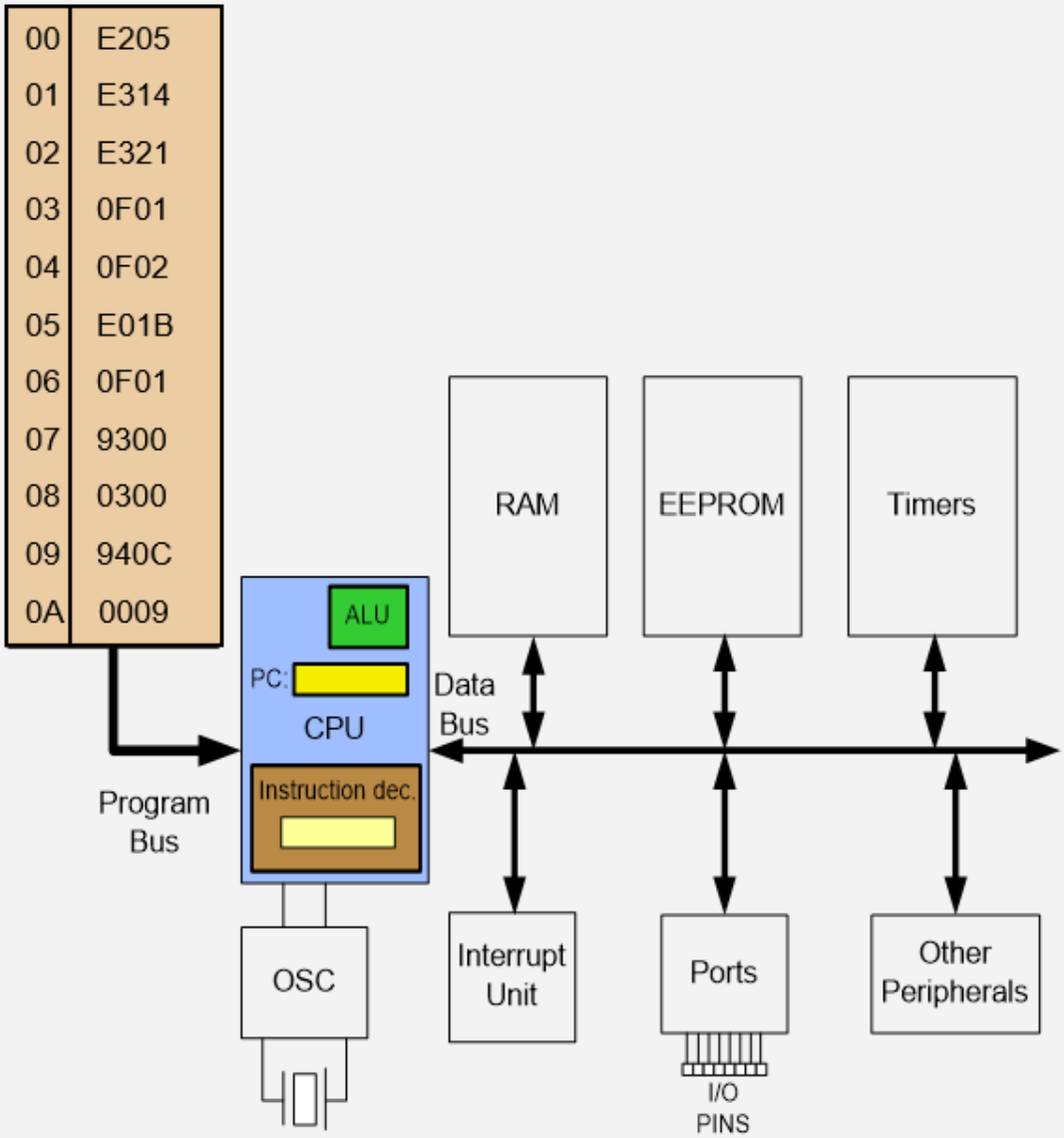
- Old Architectures





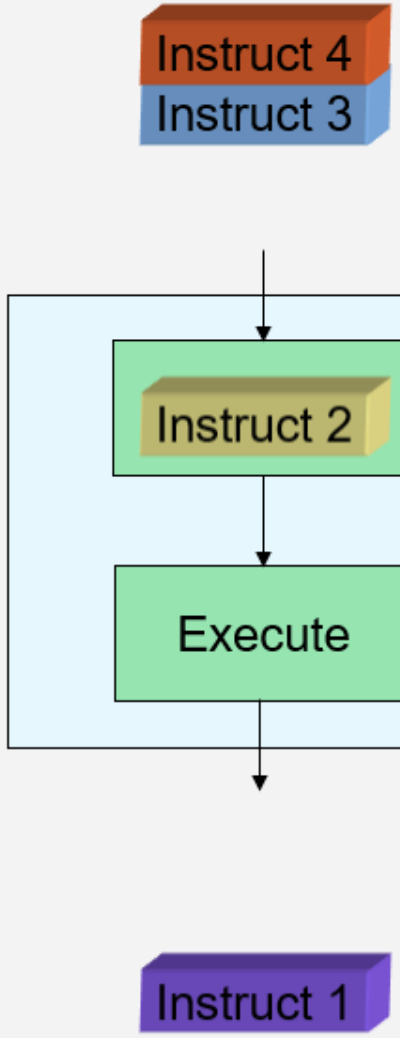
- Old Architectures

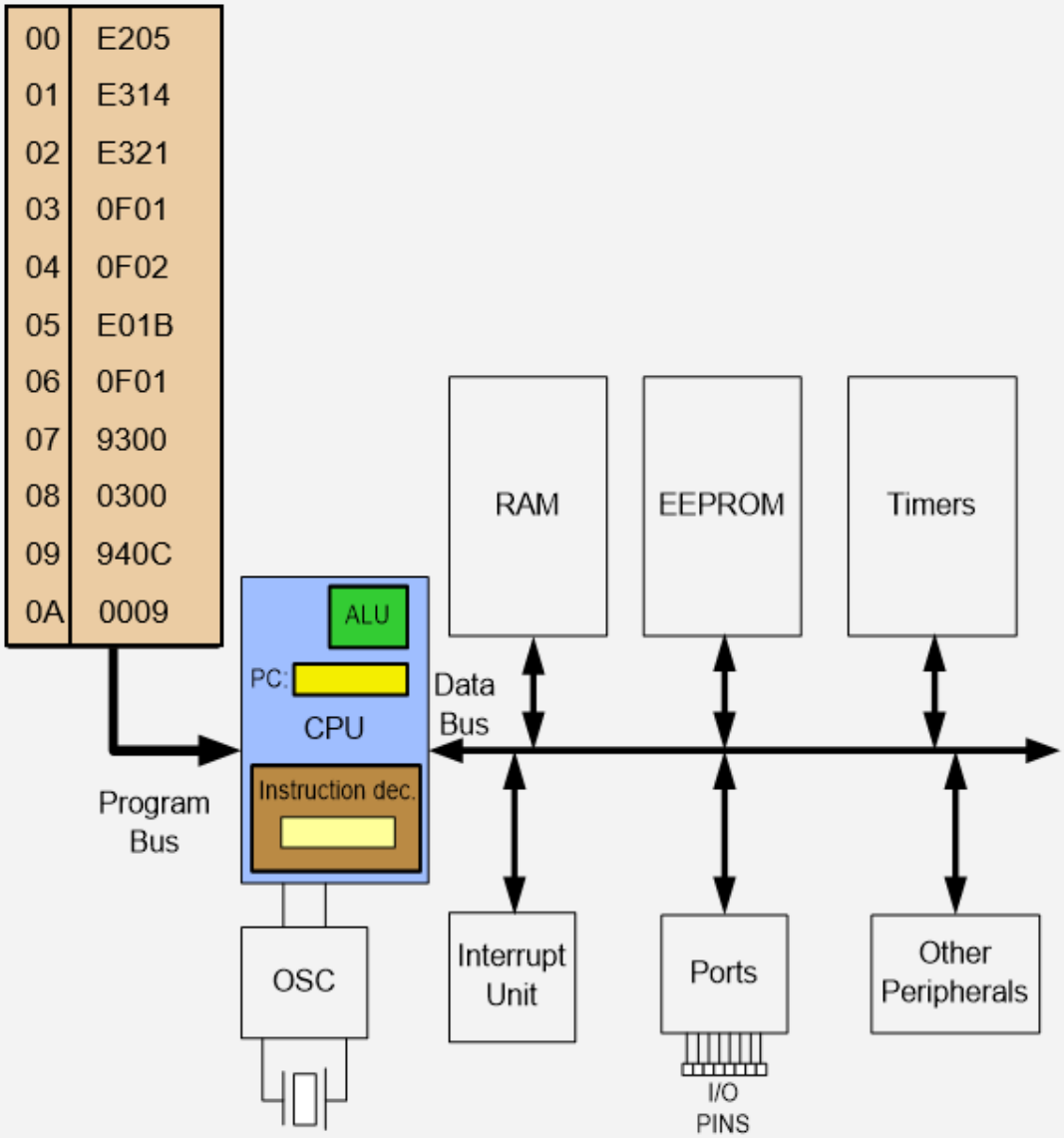




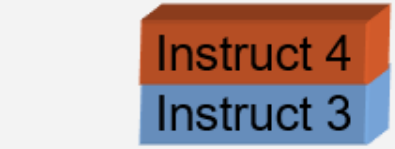
Fetch and execute

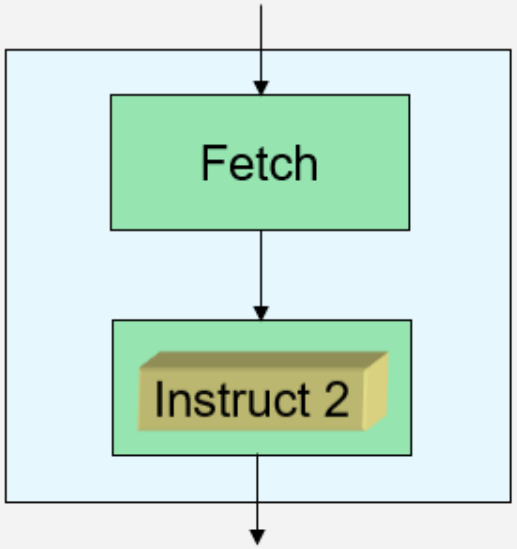
- Old Architectures

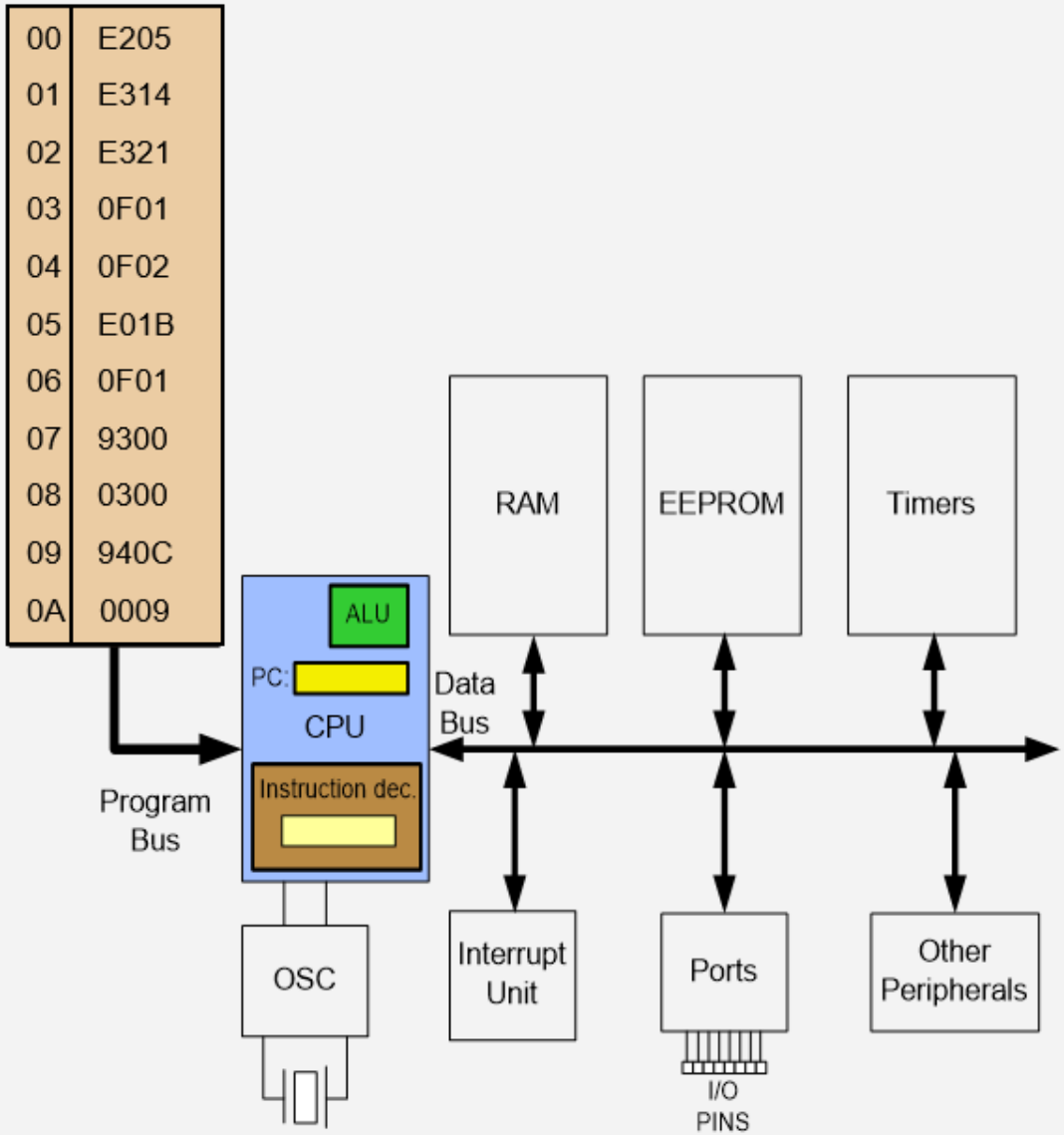




- Old Architectures

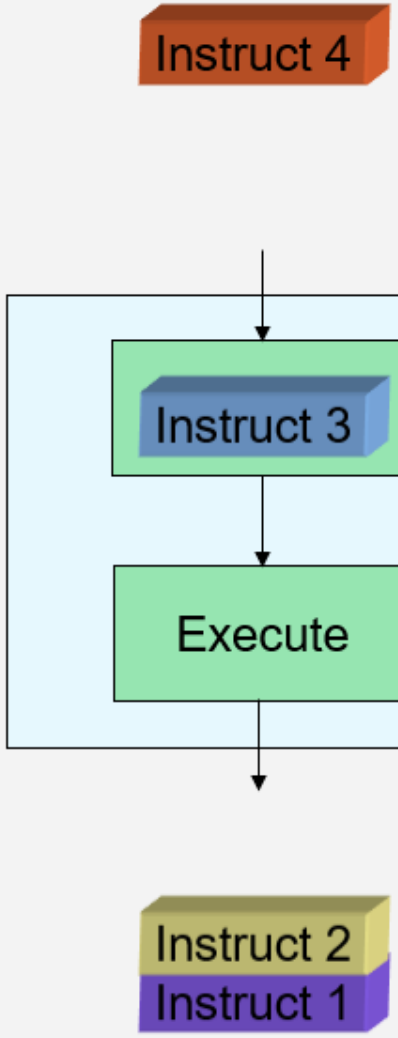


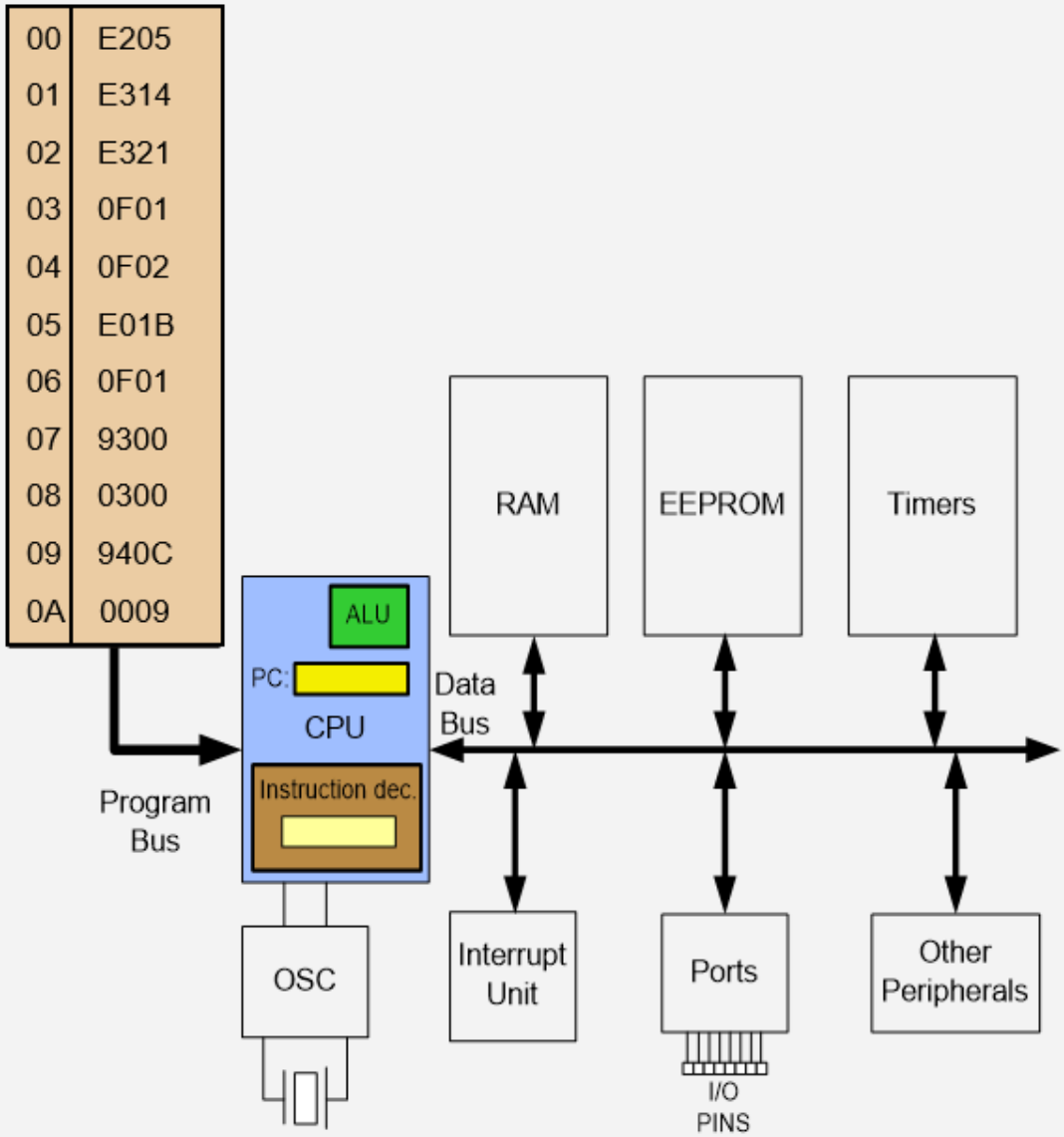




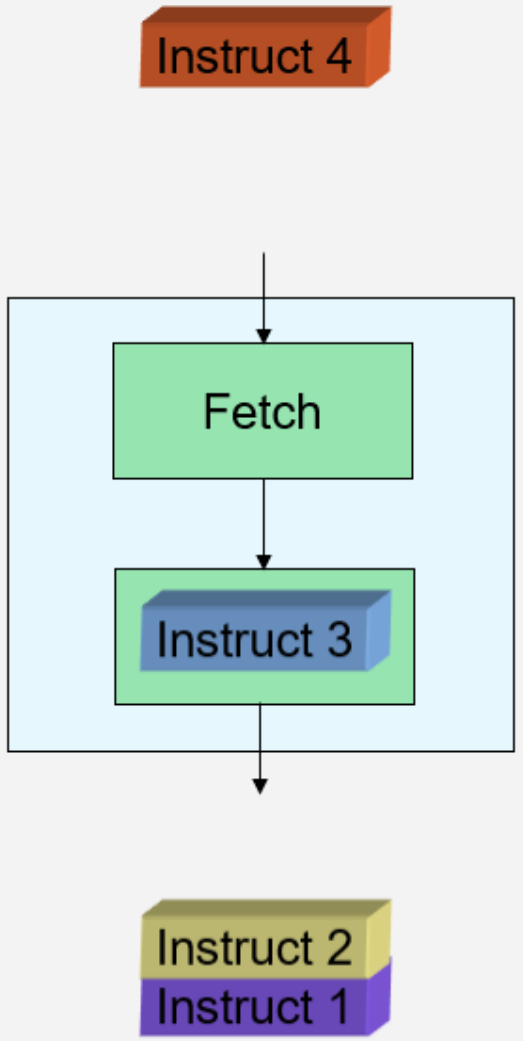
Fetch and execute

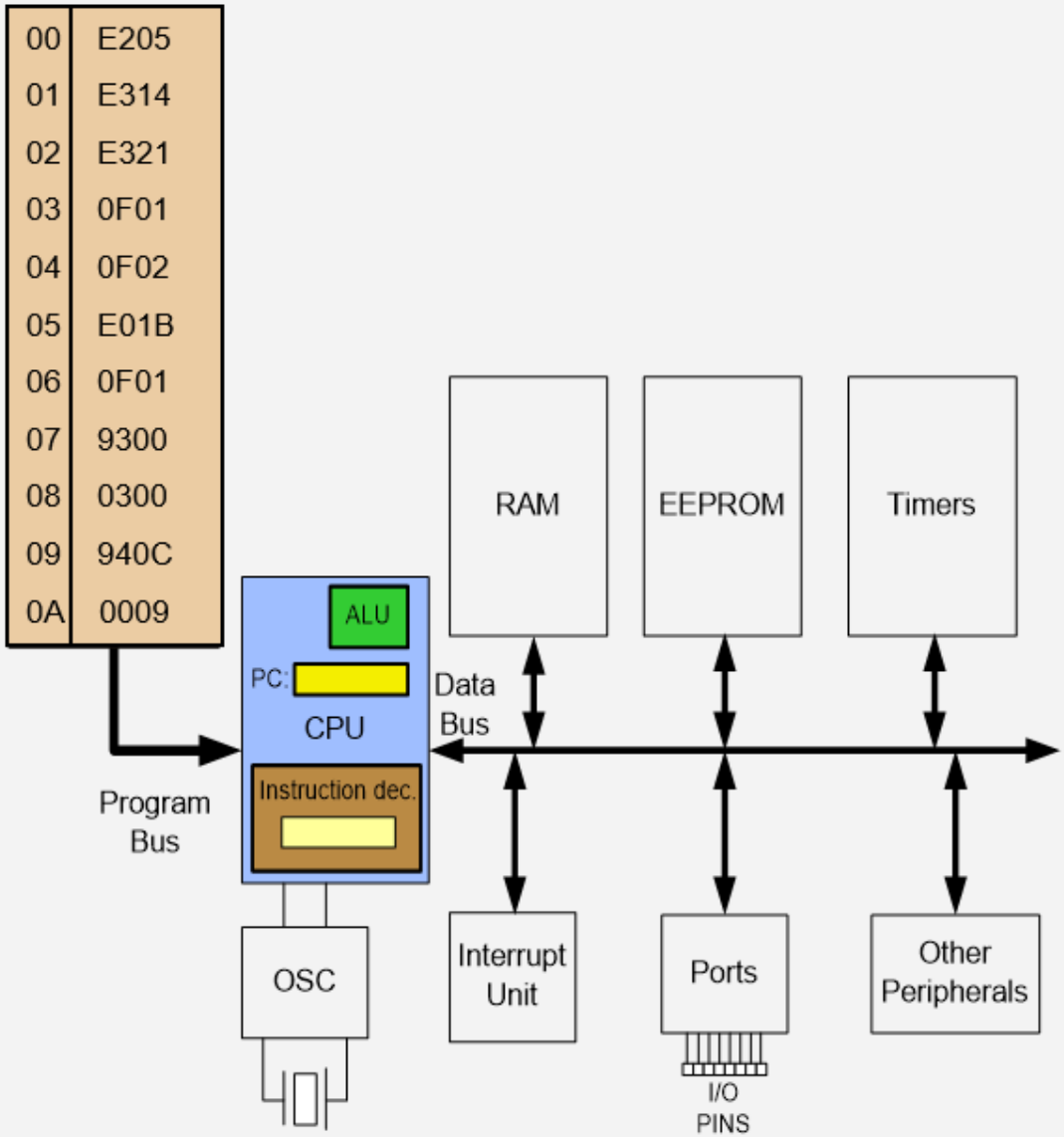
- Old Architectures



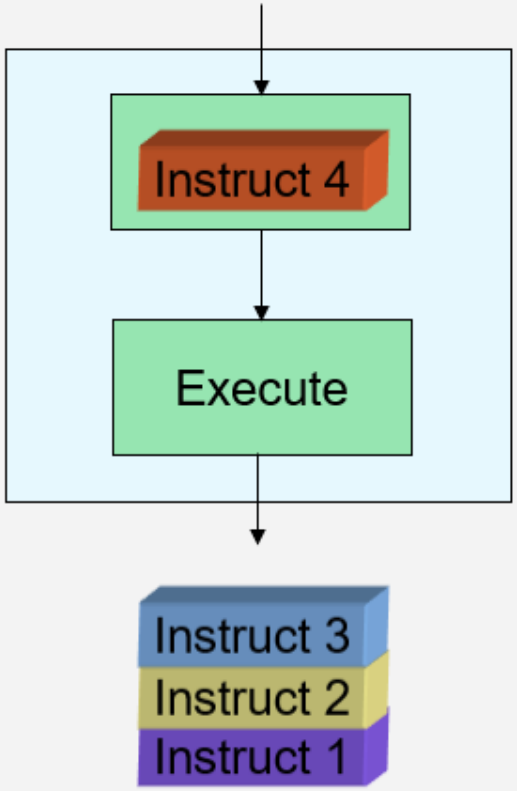


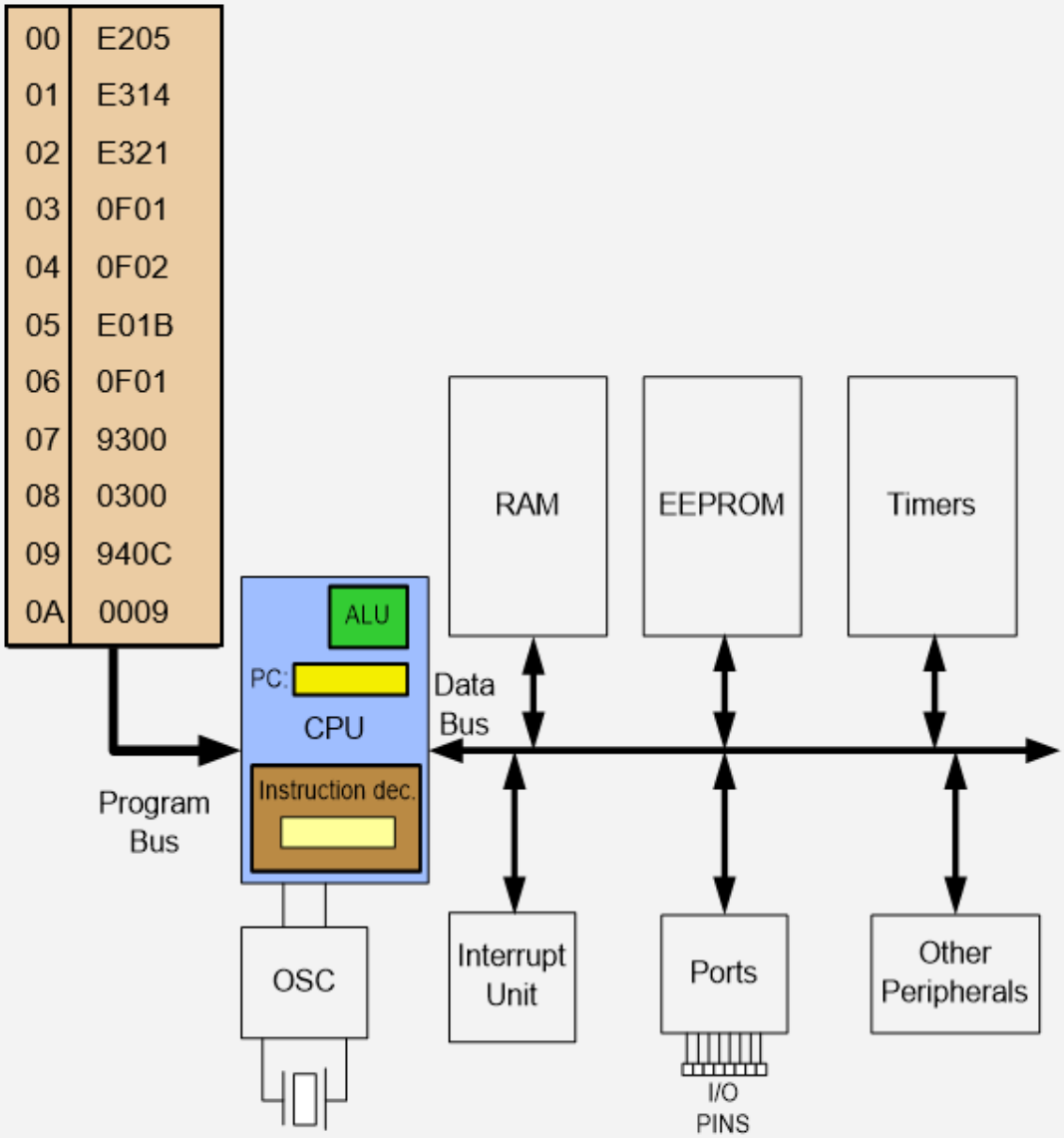
- Old Architectures



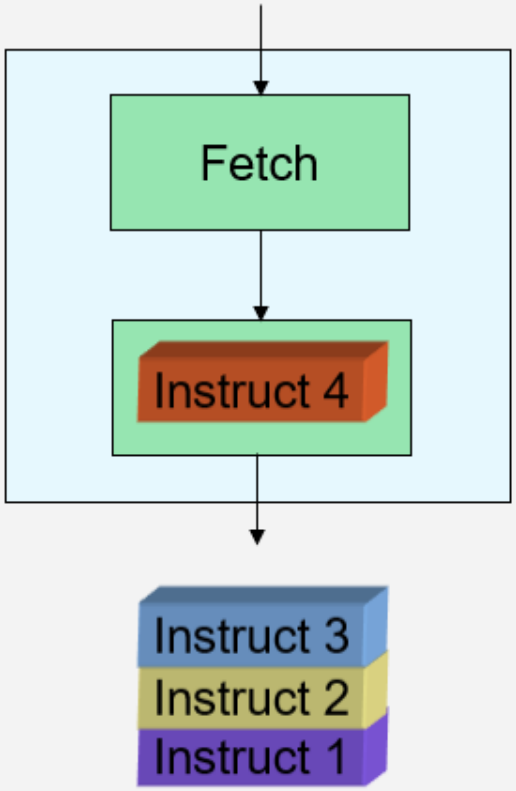


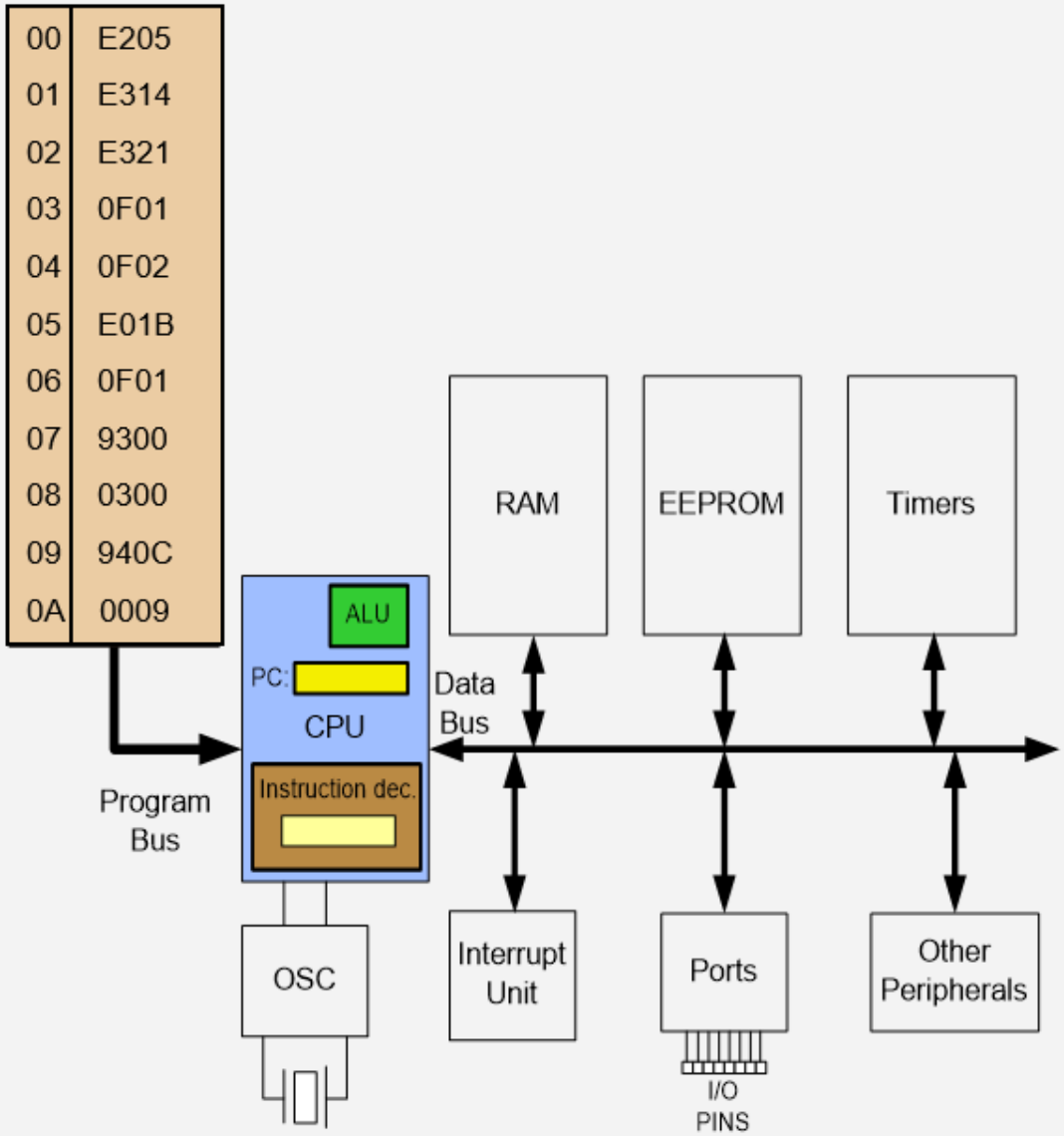
- Old Architectures



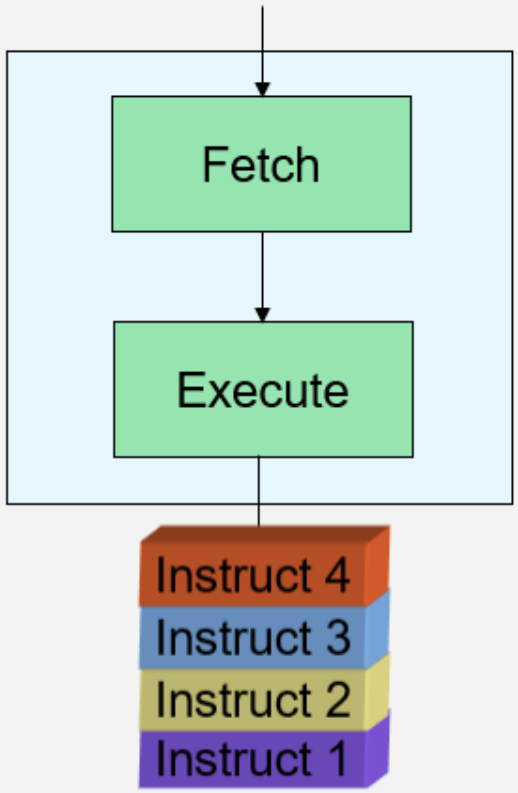


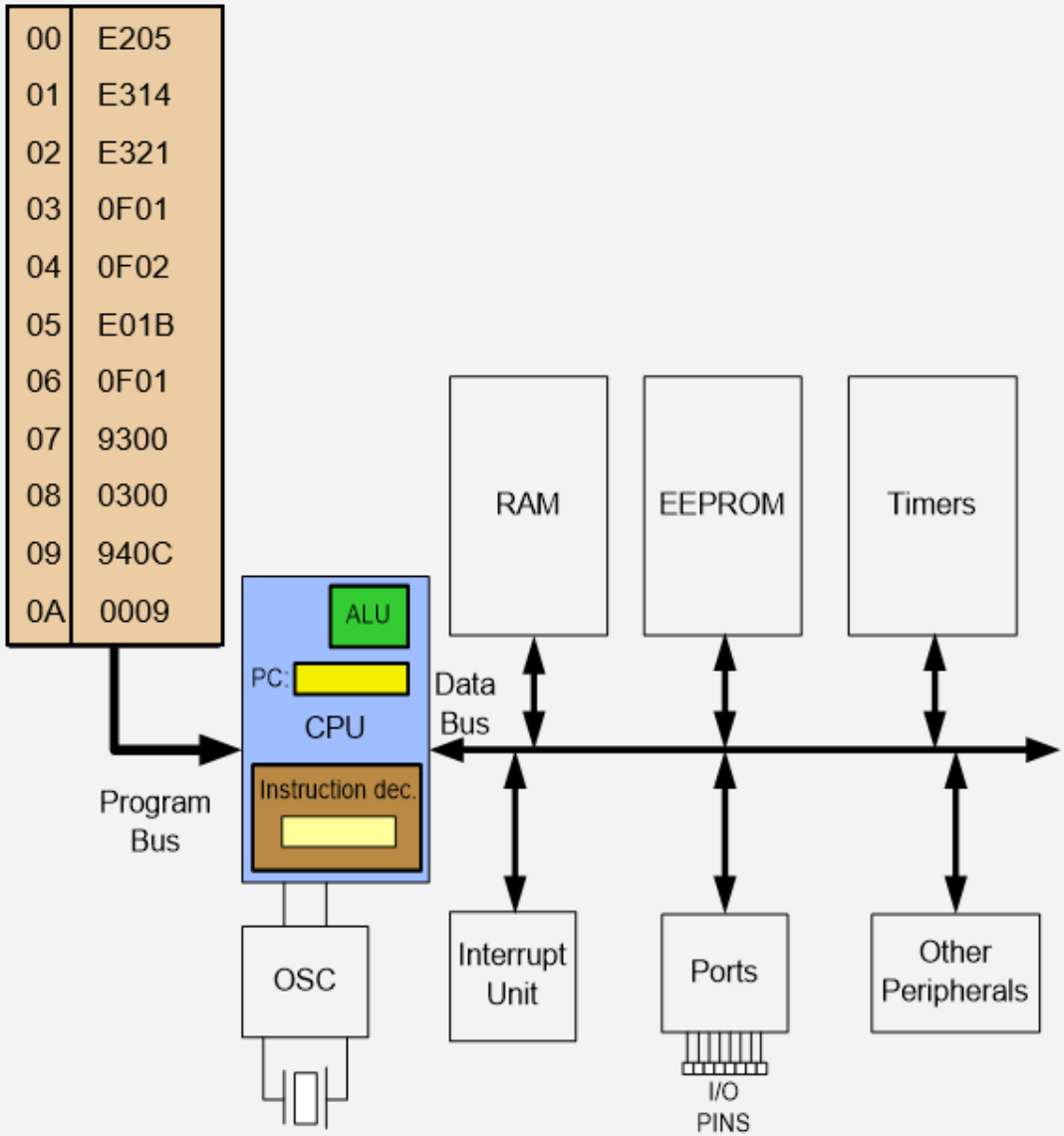
- Old Architectures





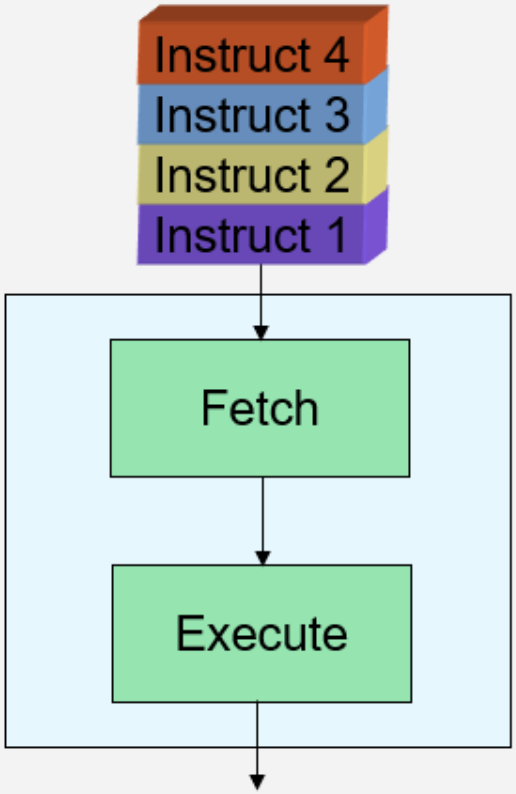
- Old Architectures

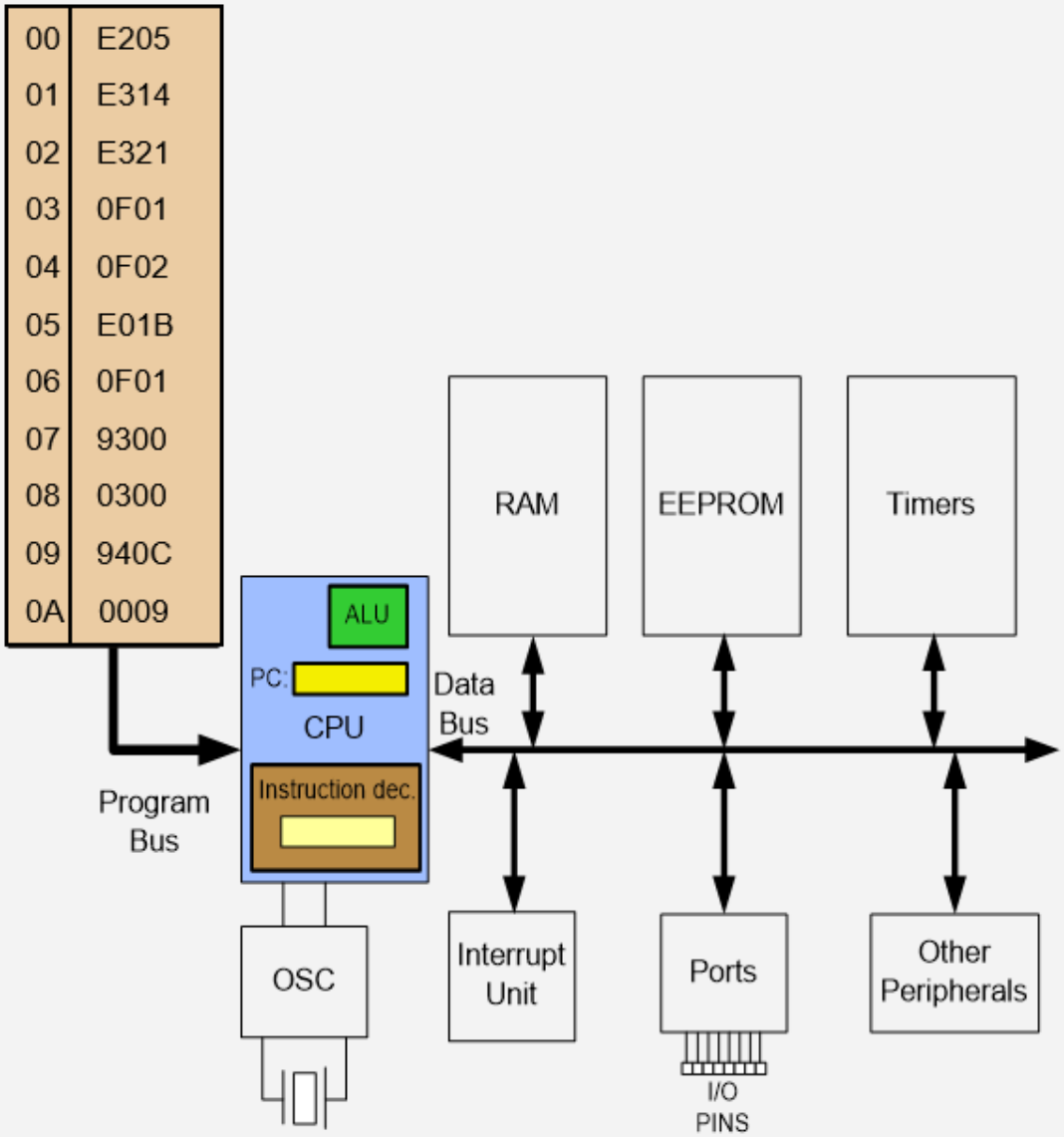




Pipelining

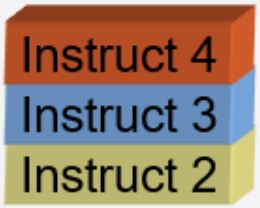
- Pipelining

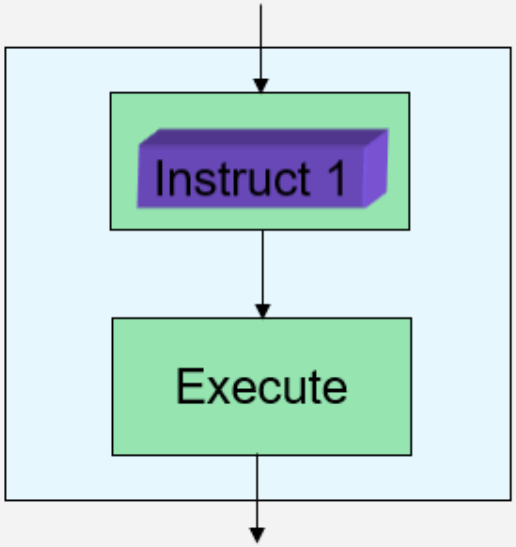


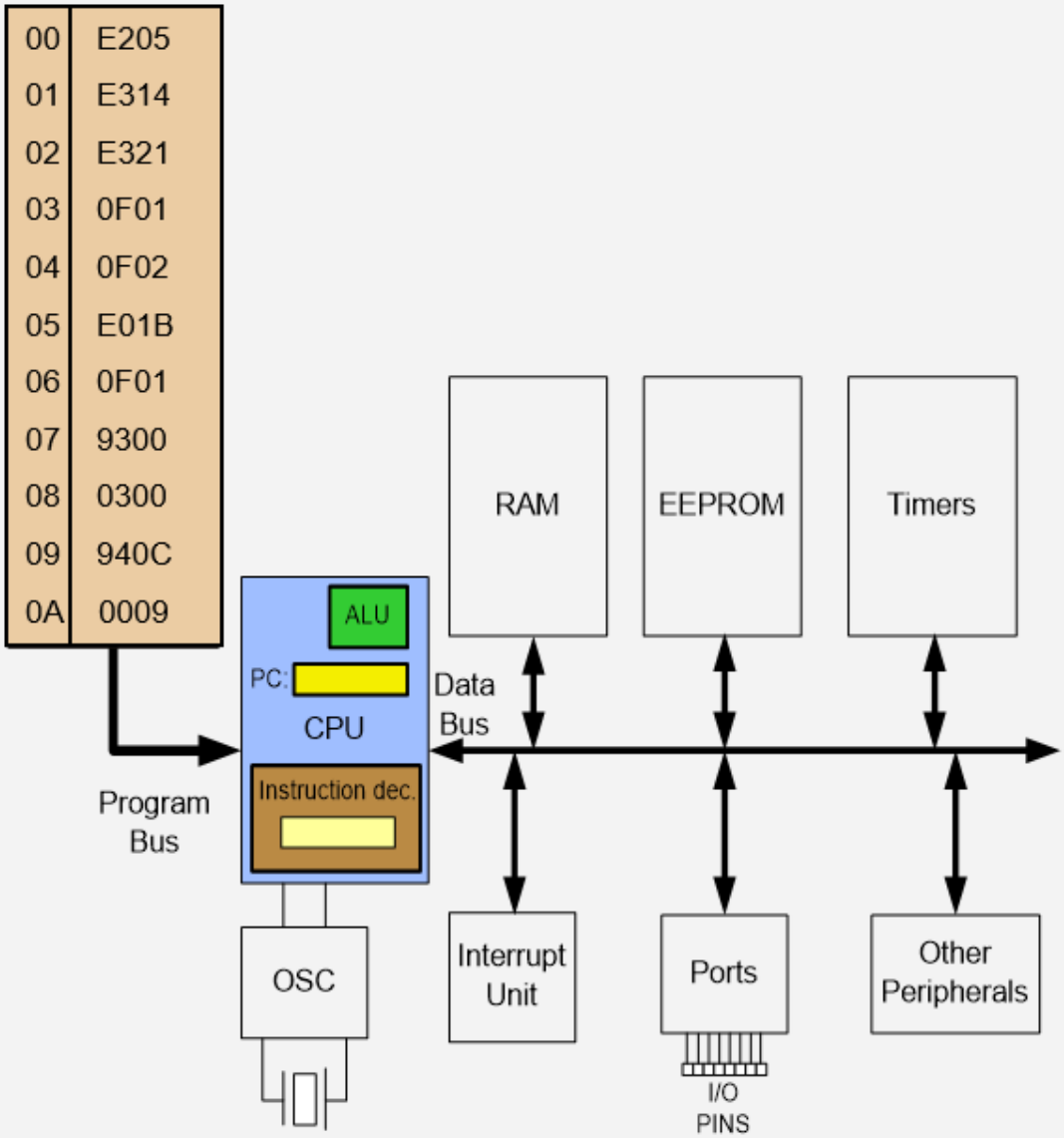


Pipelining

- Pipelining

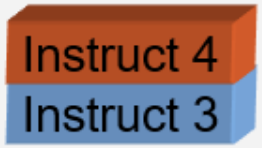


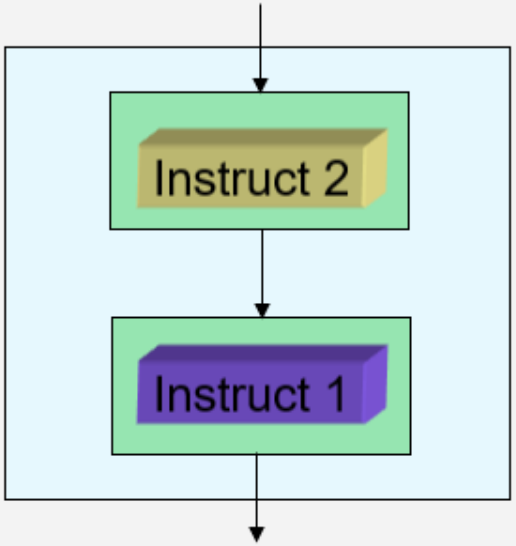


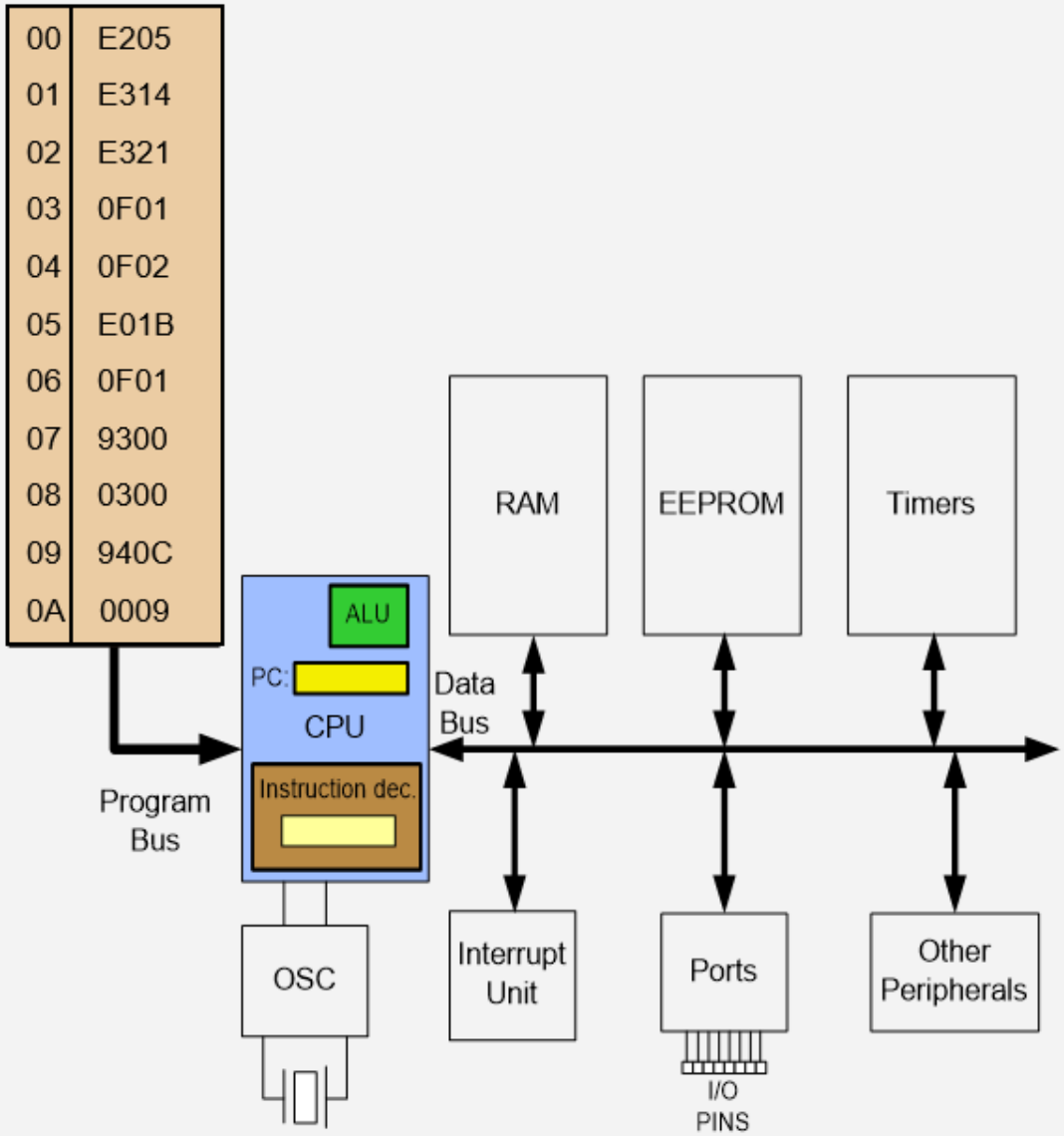


Pipelining

- Pipelining

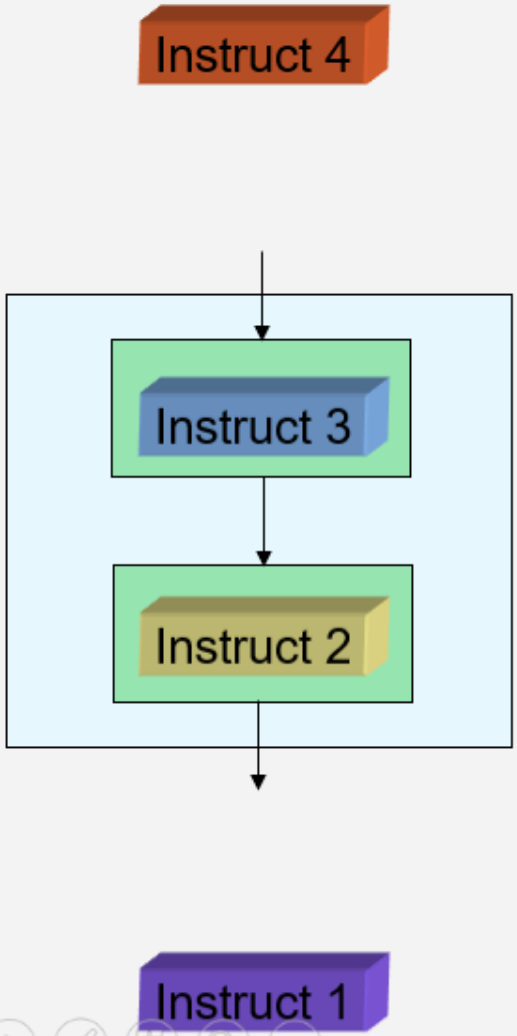


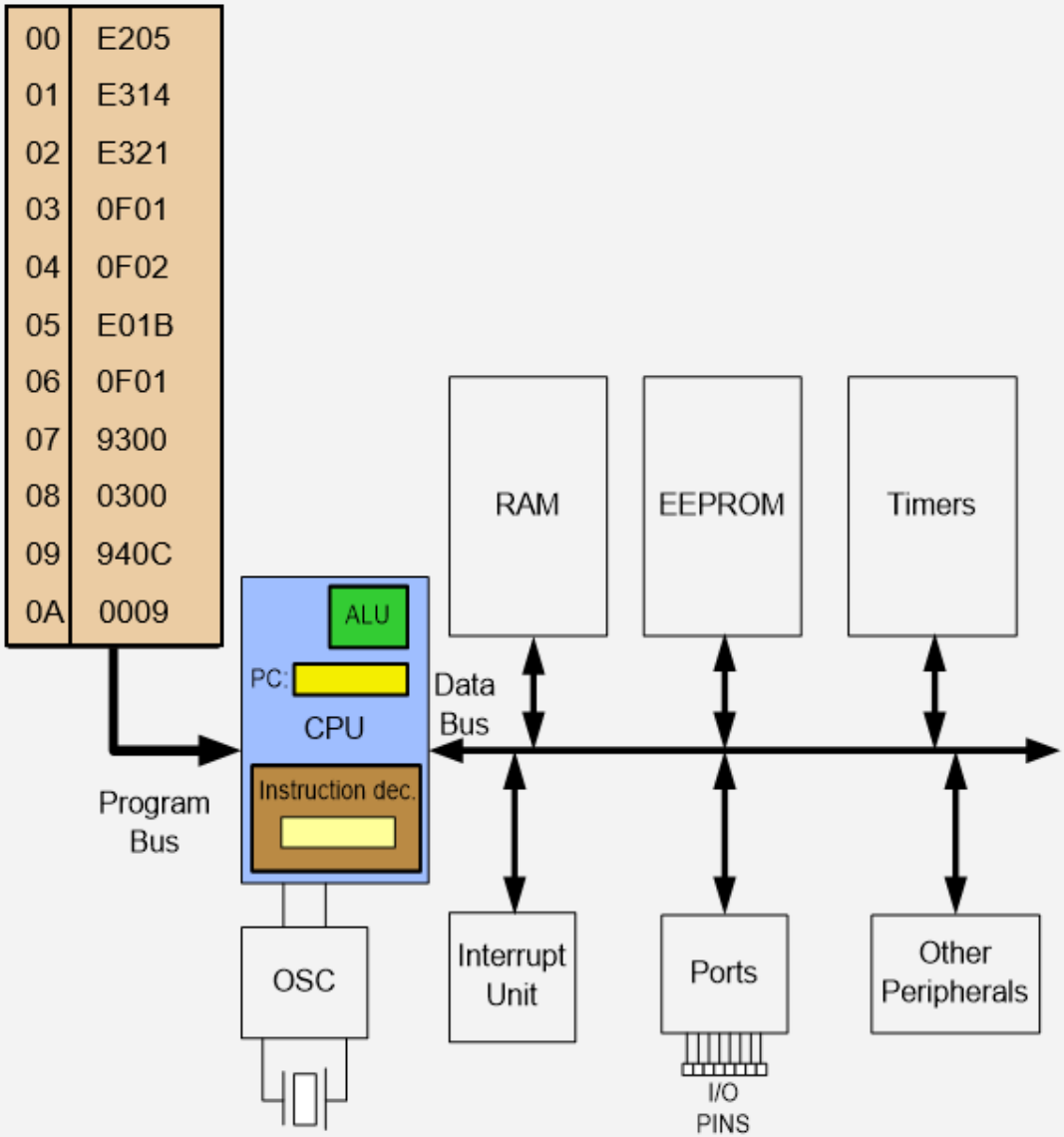




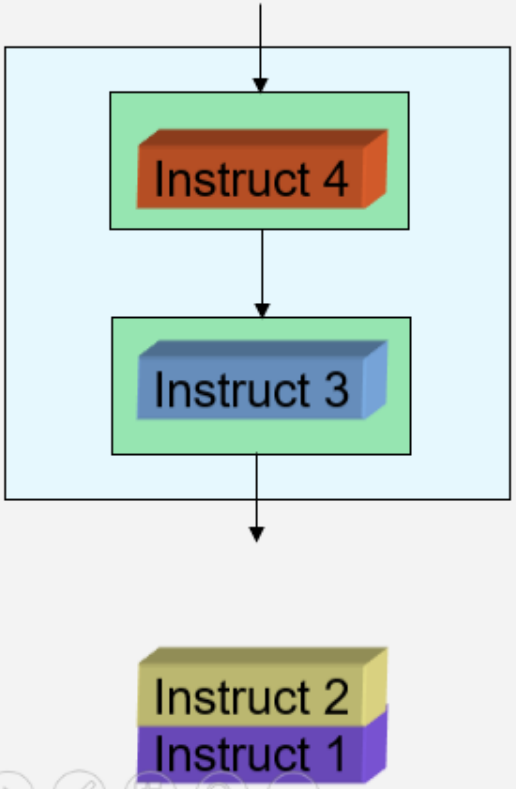
Pipelining

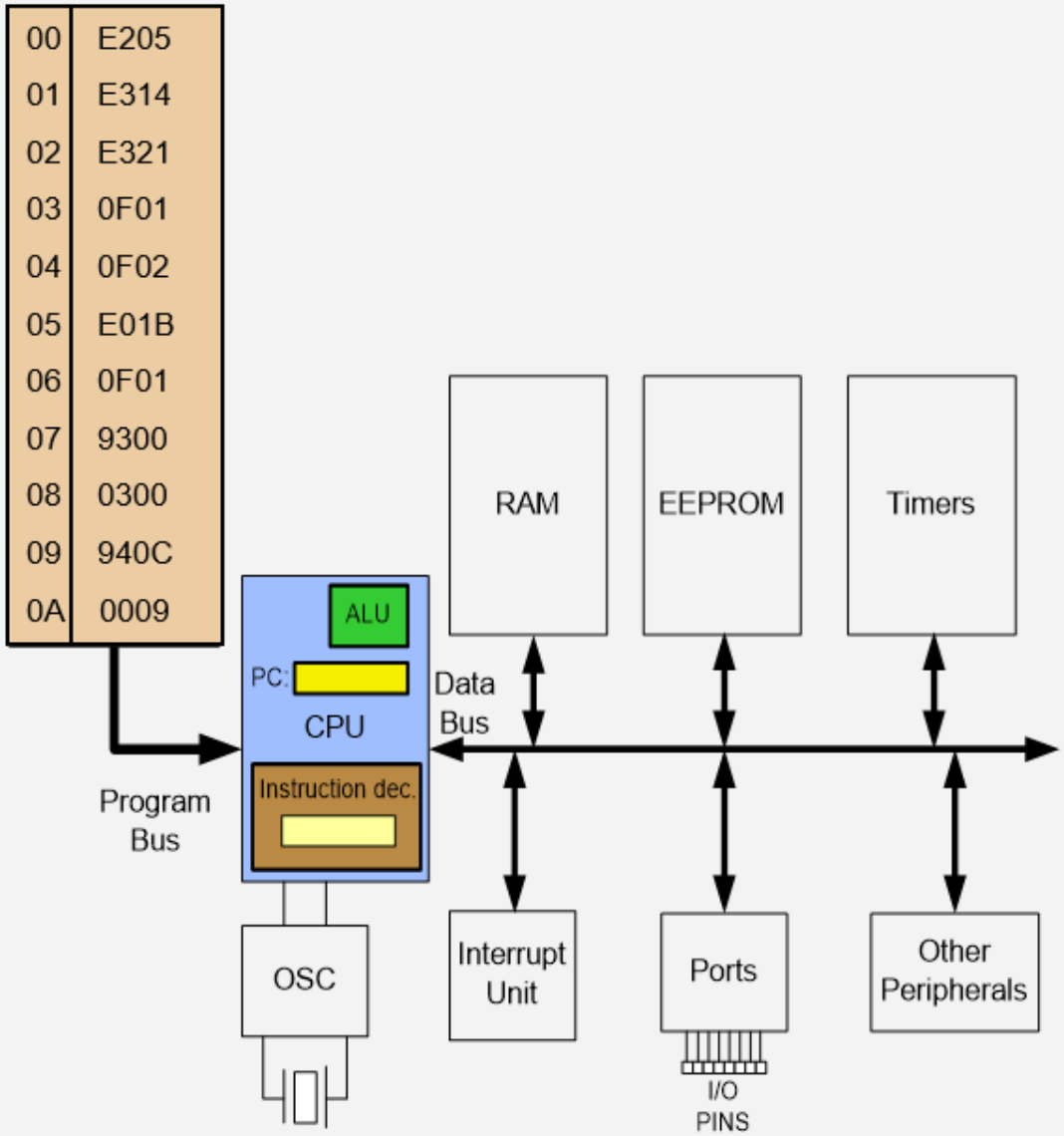
- Pipelining



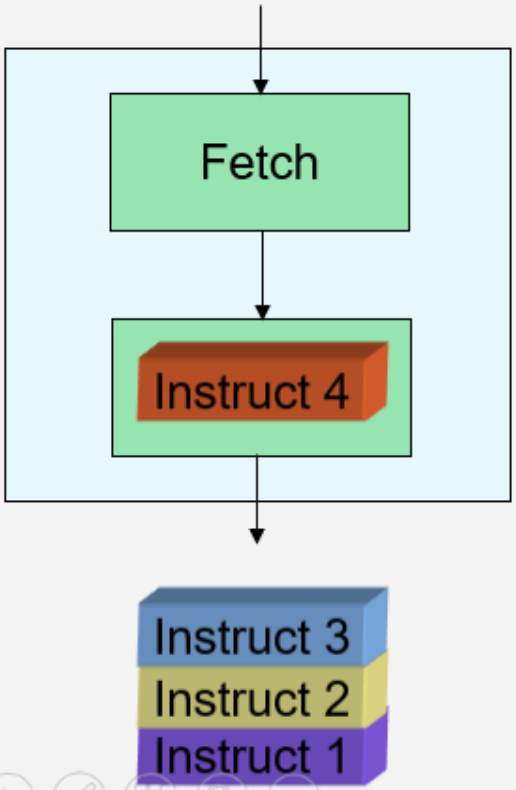


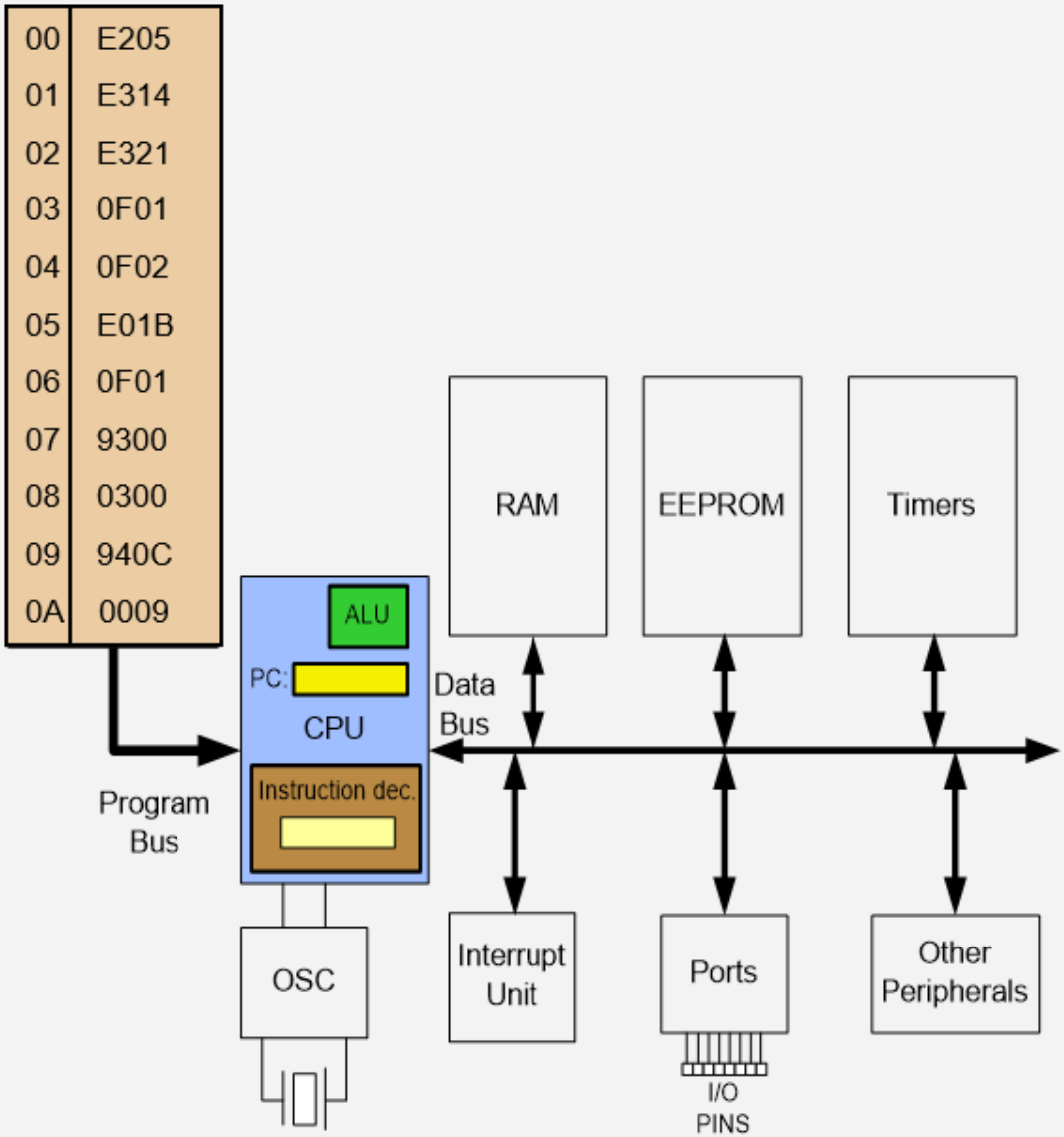
Pipelining


Instruct 4
Instruct 3
Instruct 2
Instruct 1


00 E205
01 E314
02 E321
03 0F01
04 0F02
05 E01B
06 0F01
07 9300
08 0300
09 940C
0A 0009
ALU
PC
CPU
Instruction dec.
Program Bus
Data Bus
RAM
EEPROM
Timers
OSC
Interrupt Unit
Ports
Other Peripherals
I/O PINS

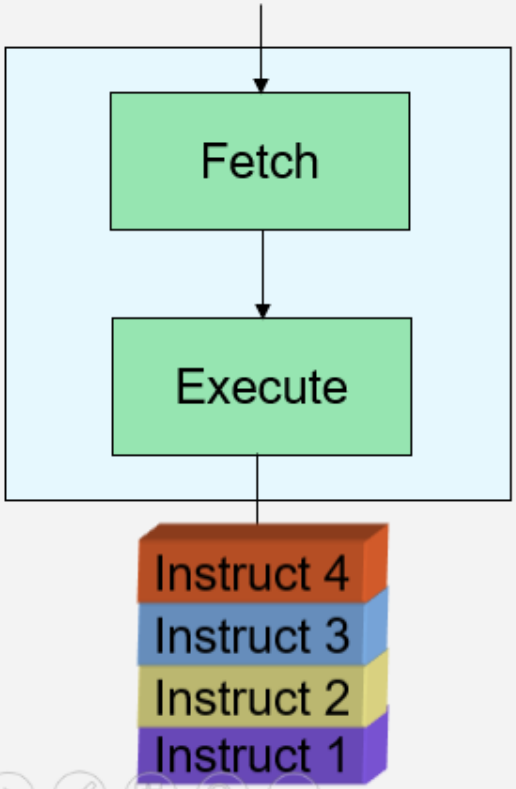
Pipelining

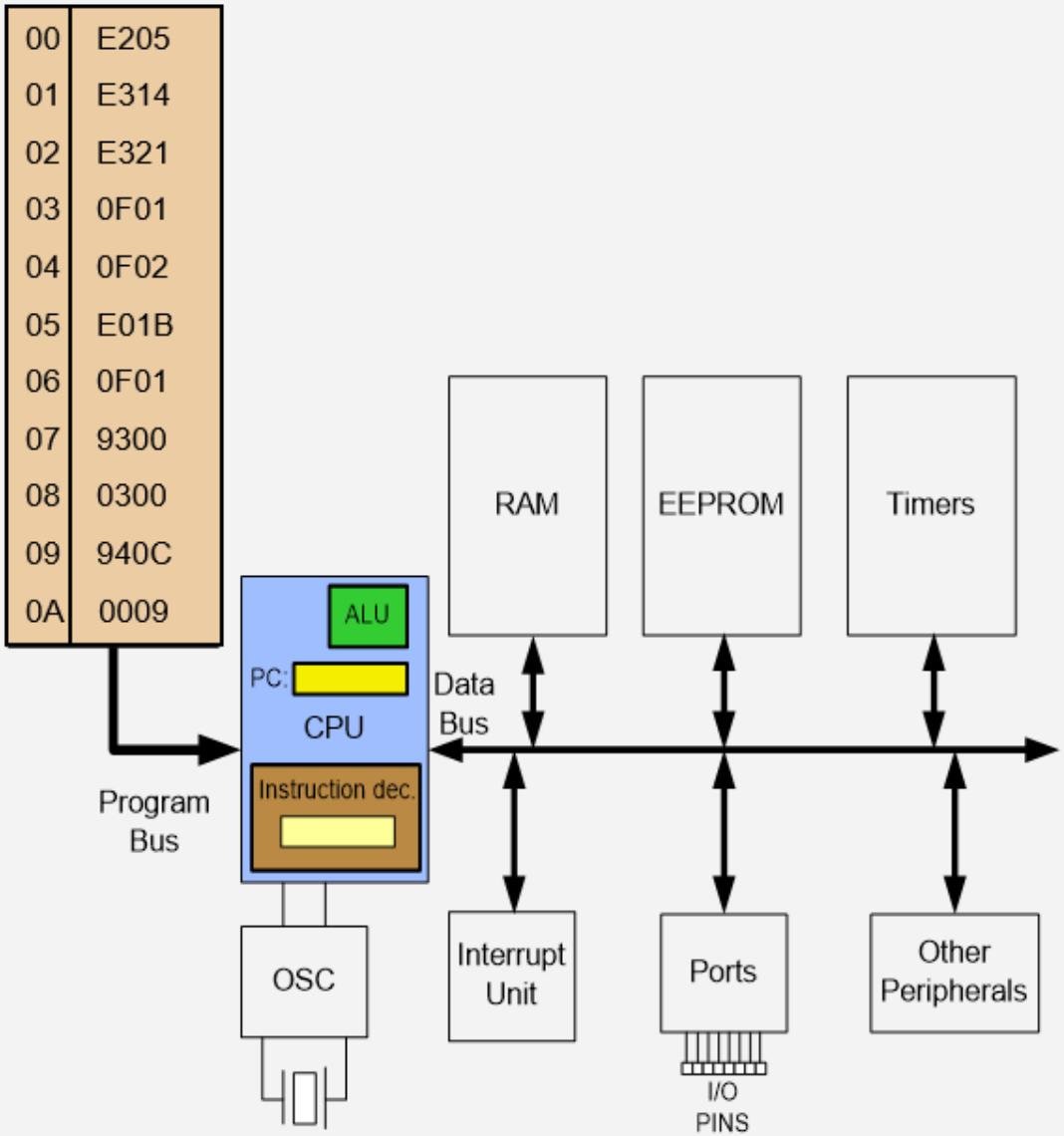

Fetch
Instruct 4
Instruct 3
Instruct 2
Instruct 1


00 E205
01 E314
02 E321
03 0F01
04 0F02
05 E01B
06 0F01
07 9300
08 0300
09 940C
0A 0009
ALU
PC
CPU
Instruction dec.
Program Bus
Data Bus
RAM
EEPROM
Timers
OSC
Interrupt Unit
Ports
I/O PINS
Other Peripherals

Pipelining

- Pipelining





RISC architecture

- **Feature 1**
  - RISC processors have a fixed instruction size. It makes the task of instruction decoder easier.
  - In AVR the instructions are 2 or 4 bytes.
  - In CISC processors instructions have different lengths
  - E.g. in 8051
  - CLR C ; a 1-byte instruction
  - ADD A, #20H ; a 2-byte instruction
  - LJMP HERE ; a 3-byte instruction

RISC architecture

- **Feature 2**

  reduce the number of instructions

  - Pros: Reduces the number of used transistors
  - Cons:
    - Can make the assembly programming more difficult
    - Can lead to using more memory

RISC architecture

- **Feature 3**

  limit the addressing mode

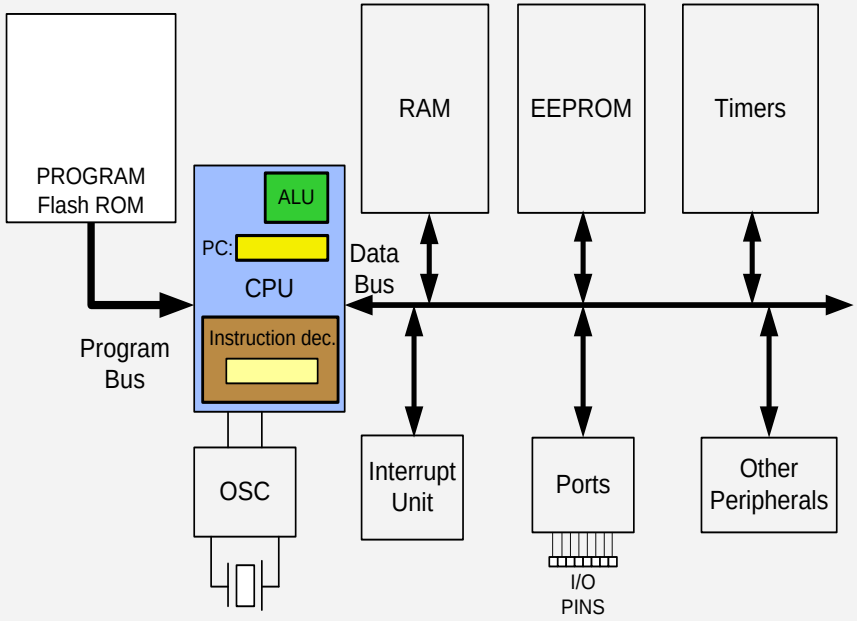
  - Advantage
    - Hard wiring
  - Disadvantage
    - Can make the assembly programming more difficult

RISC architecture

- **Feature 4**

  Load/Store

```
LDS R20, 0x200
LDS R21, 0x220
ADD R20, R21
STS 0x230, R20
```

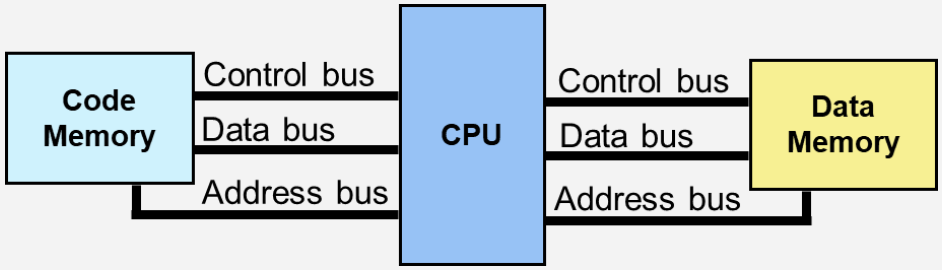


RISC architecture

- **Feature 5 (Harvard architecture)**

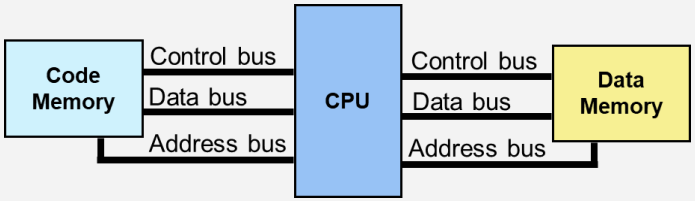
  separate buses for opcodes and operands

  - Advantage
    - opcodes and operands can go in and out of the CPU together.
  - Disadvantage
    - leads to more cost in general purpose computers.



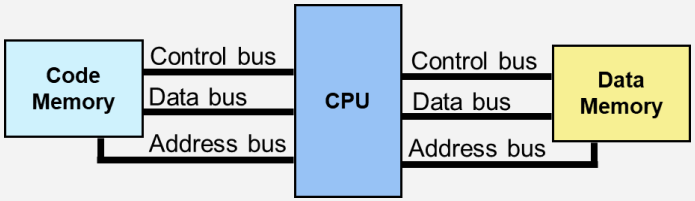
RISC architecture

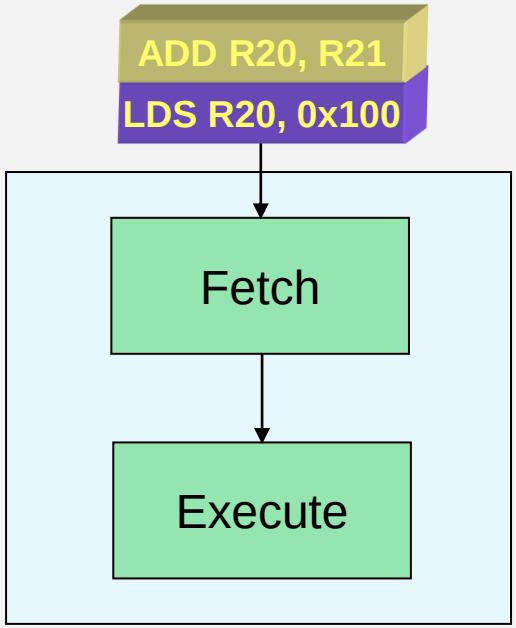
```
LDS R20, 0x100 ; R20 = [0x100]
ADD R20,R21    ; R20 = R20 + R21
```



RISC architecture

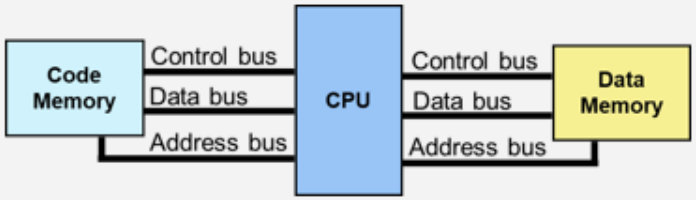
```
LDS R20, 0x100 ; R20 = [0x100]
ADD R20,R21    ; R20 = R20 + R21
```

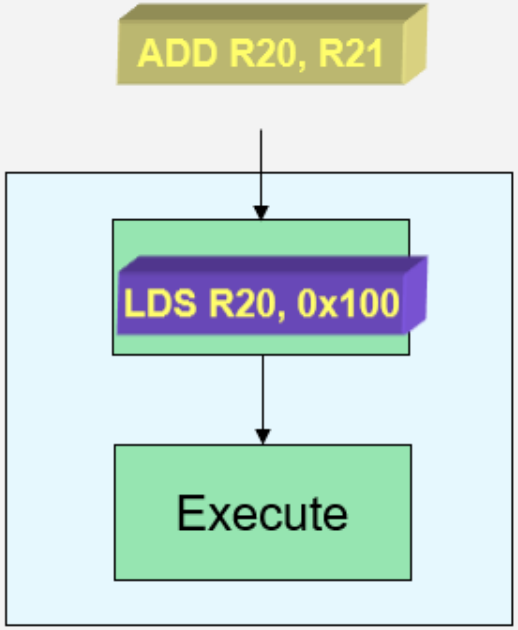




RISC architecture

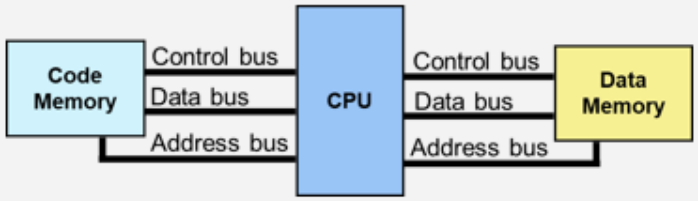
```
LDS R20, 0x100 ; R20 = [0x100]
ADD R20,R21    ; R20 = R20 + R21
```

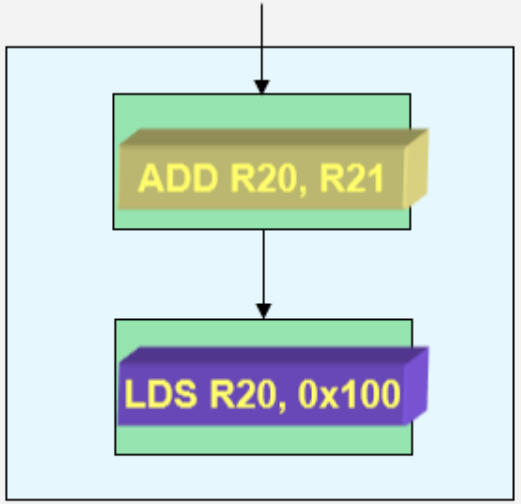




RISC architecture

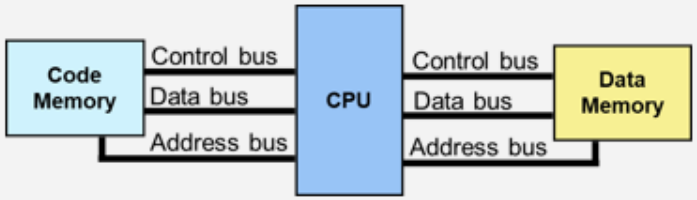
```
LDS R20, 0x100 ; R20 = [0x100]
ADD R20,R21    ; R20 = R20 + R21
```

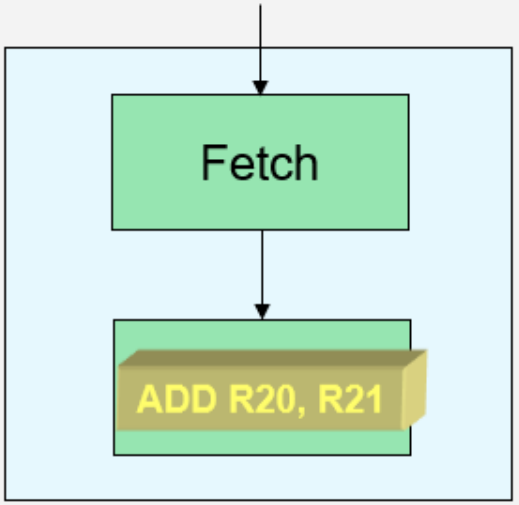




RISC architecture

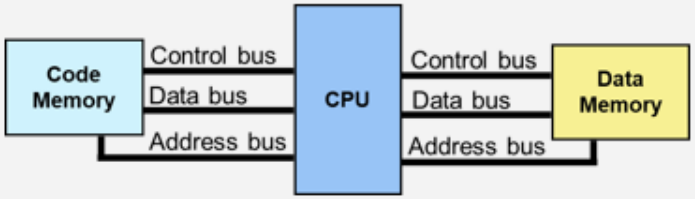
```
LDS R20, 0x100 ; R20 = [0x100]
ADD R20,R21    ; R20 = R20 + R21
```

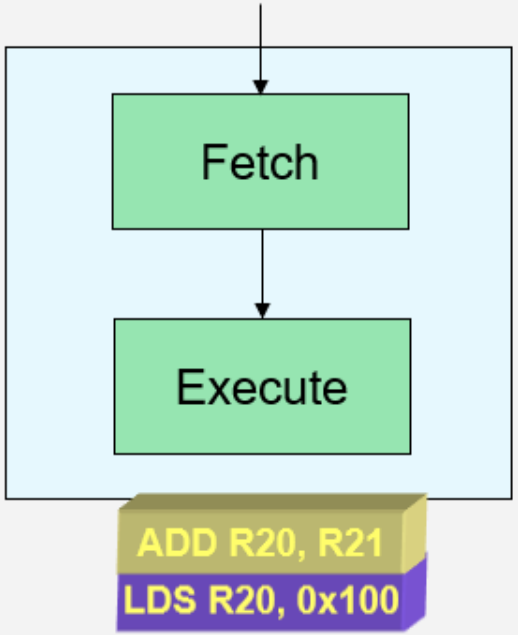




RISC architecture

```
LDS R20, 0x100 ; R20 = [0x100]
ADD R20,R21    ; R20 = R20 + R21
```





RISC architecture

- **Feature 6**

  more than 95% of instructions are executed in 1 machine cycle

RISC architecture

- **Feature 7**
  - RISC processors have at least 32 registers. Decreases the need for stack and memory usages.
    - In AVR there are 32 general purpose registers (R0 to R31)

Thanks!

Do you have any questions?

razeghizade@gmail.com

Razeghizade.pudica.ir