



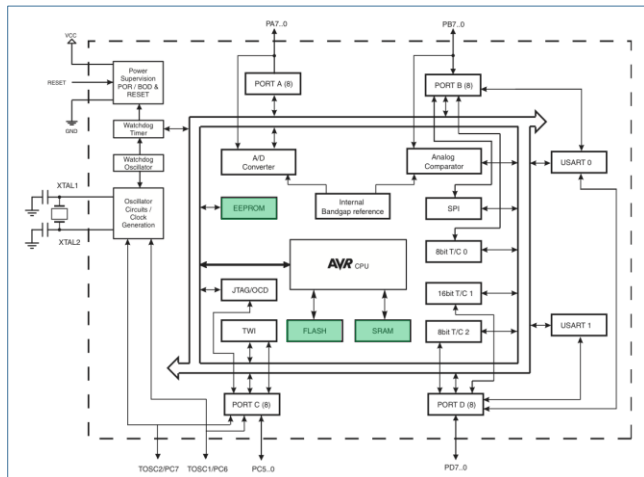
Sensors & Microsystem Electronics: microcontrollers

PART 2: MEMORY, TIMERS & INTERRUPTS

Memory

Memory overview (Volatile vs Non-volatile)

- FLASH
 - 32 kilobytes
 - Program storage
 - non-volatile
 - 16-bit addresses
- SRAM
 - 2 kilobytes
 - Data memory
 - volatile
 - 16-bit addresses
- EEPROM
 - 1 kilobyte
 - Data memory
 - non-volatile
 - 16-bit addresses



The microcontroller has three different memories: FLASH, SRAM and EEPROM.

Both FLASH and EEPROM are non-volatile, meaning they keep their content between power cycles. This is not the case with the SRAM which is volatile.

The flash memory is also called the program memory. This is because your compiled program is written here, and loaded when the microcontroller is running. It is not possible to write to this memory from your running program. Writing can only be done by the programmer. As a programmer you can store blocks of readonly data here to be read by your program.

The SRAM is your “working memory” this part of the memory can be used to temporarily store values. The SRAM has very fast access times and has a virtually unlimited number of read/write cycles.

The EEPROM can be written from your code as opposed to the FLASH. It is used to store values that need to be kept between power cycles. Some examples of this are: settings, highscores, ... The EEPROM has slower access times, as well as a limited number of write-cycles (10k-100k)

SRAM / Data Memory

Data Memory	
32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Reg.	0x0060 - 0x00FF
Internal SRAM (512/1024/1024/2048 x 8)	0x0100
	0x02FF/0x04FF/0x4FF/0x08FF

ATmega328 VOLATILE DATA MEMORY (SRAM)															
	0	1	2	3	4	5	6	7							
0x0000	R0	R1	R2	R3	R4	R5	R6	R7							
0x0008	R8	R9	R10	R11	R12	R13	R14	R15							
0x0010	R16	R17	R18	R19	R20	R21	R22	R23							
0x0018	R24	R25	R26XL	R27XH	R28YL	R29YH	R30ZL	R31ZH							
0x0020					PNIB	DDRB	PORTB	PINC	DDRC						
0x0028	PORTC				PORTD										
0x0030					PCFR	EPR	ENSK	SPSR	ECSR						
0x0038					GTCCR	TCCRBA	TCCRB	TCNT0	OCR1A						
0x0040	FEOR	EEARL	EEARH			TCRNB	TCNT1	OCR1A							
0x0048	OCR0B	ACSR		SPCRH	SPCR	SPSR	SPDR								
0x0050					MCUSR	MCUSR	SPI	SPI	SPICSR						
0x0058									SPR	SREG					
0x0060	WDTCSR	CLKPR			PRR		OSCCAL								
0x0068	PCSR	EICRA			PCMSK0	PCMSK1	PCMSK2	TIMSK0	TIMSK1						
0x0070					TIMSK2										
0x0078	ADCL	ADCH	ADCSRA		ADCSRB		ADMX	DDRB		DDRH					
0x0080	TICHA		TCRNB		TCRRC		TCNT1	TCNT1H	ICR1	ICR1H					
0x0088	OCR1AL		OCR1AL		OCR1BL		OCR1BH								
0x0090															
0x0098															
0x00A0															
0x00A8															
0x00B0	TCCR2A		TCCR2B		TCNT2		OCR2A		OCR2B		ASR				
0x00B8	TWBR		TWSR		TWAR		TWDR		TWCR		TWAMR				
0x00C0	UCSRA		UCSRB		UCSRRC				UDR0H		UDR0H				
0x00C8															
0x00D0															
0x00D8															
0x00E0															
0x00E8															
0x00F0															
0x00F8															
0x0100															
...															
...															
0x08F8															
	0	1	2	3	4	5	6	7							
									INTERNAL DATA SRAM (usable by user) (see LDDSR153)						

The SRAM or data memory consists of 4 parts:

The 32 registers on which one can do operations.

64 I/O registers. And 160 extended I/O registers. These registers are used to configure the peripherals of the microcontroller.

The internal SRAM: this is free memory space to be freely used by the programmer

Although the microcontroller is 8 bit and as such all registers are 8 bit, the memory is addressed using 16bit addresses. Otherwise the amount of memory space would be limited to 255 bytes. The 16bit addressing is done through two 8 bit registers, one contains the upper 8 bit, and the other the lower 8 bit.

Registers

- Bitaddressable
- R16-R31 support LDI (load constant)
- Certain registers for specific instructions
 - See instruction set
- R27 – R31 double as pointer registers
- Not zeroed on reset

Data Memory	
32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Reg.	0x0060 - 0x00FF
Internal SRAM (512/1024/1024/2048 x 8)	0x0100
	0x02FF/0x04FF/0x4FF/0x08FF

ATmega328 VOLATILE DATA MEMORY (SRAM)															
0x0000	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30
0x0008	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30
0x0010	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30
0x0018	R16	R17	R18	R19	R20	R21	R22	R23	R24	R25	R26	R27	R28	R29	R30
0x0020	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0028	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0030	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0038	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0040	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0048	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0050	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0058	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0060	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0068	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0070	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0078	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0080	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0088	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0090	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0098	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00A0	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00A8	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00B0	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00B8	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00C0	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00C8	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00D0	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00D8	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00E0	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00E8	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00F0	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x00F8	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0100	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
...	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD
0x0BFF	PORTC	PIND	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD	PORTD	DDRD

The SRAM or data memory consists of 4 parts:

The 32 registers on which one can do operations.

64 I/O registers. And 160 extended I/O registers. These registers are used to configure the peripherals of the microcontroller.

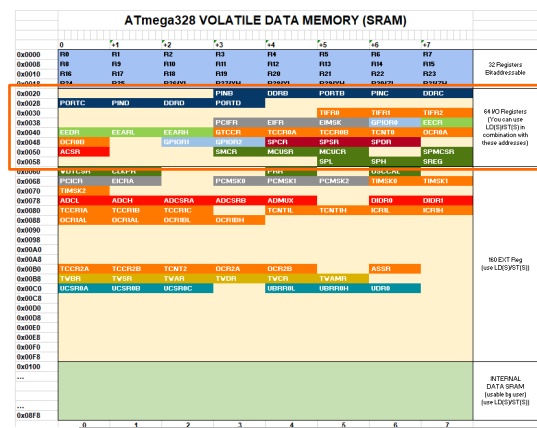
The internal SRAM: this is free memory space to be freely used by the programmer

Although the microcontroller is 8 bit and as such all registers are 8 bit, the memory is addressed using 16bit addresses. Otherwise the amount of memory space would be limited to 255 bytes. The 16bit addressing is done through two 8 bit registers, one contains the upper 8 bit, and the other the lower 8 bit.

I/O registers

- Only first 32 are bitaddressable
- To be used with **IN** and **OUT**
- Specific functions & peripheral config.
- Set to initial values on reset
 - See relevant datasheet sections

Data Memory	
32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Reg.	0x0060 - 0x00FF
Internal SRAM (512/1024/1024/2048 x 8)	0x0100
	0x02FF/0x04FF/0x4FF/0x08FF



The SRAM or data memory consists of 4 parts:

The 32 registers on which one can do operations.

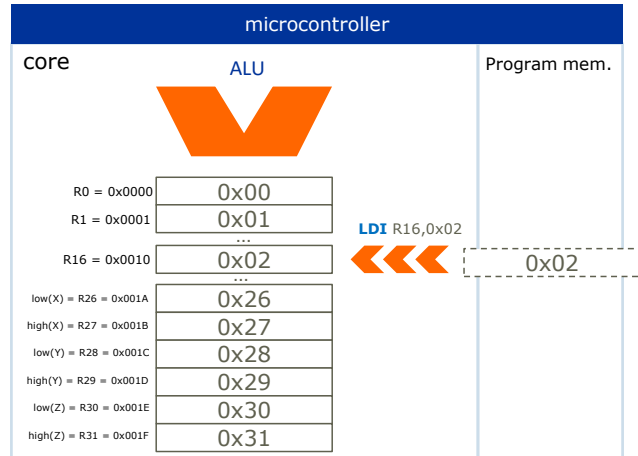
64 I/O registers. And 160 extended I/O registers. These registers are used to configure the peripherals of the microcontroller.

The internal SRAM: this is free memory space to be freely used by the programmer

Although the microcontroller is 8 bit and as such all registers are 8 bit, the memory is addressed using 16bit addresses. Otherwise the amount of memory space would be limited to 255 bytes. The 16bit addressing is done through two 8 bit registers, one contains the upper 8 bit, and the other the lower 8 bit.

Data memory

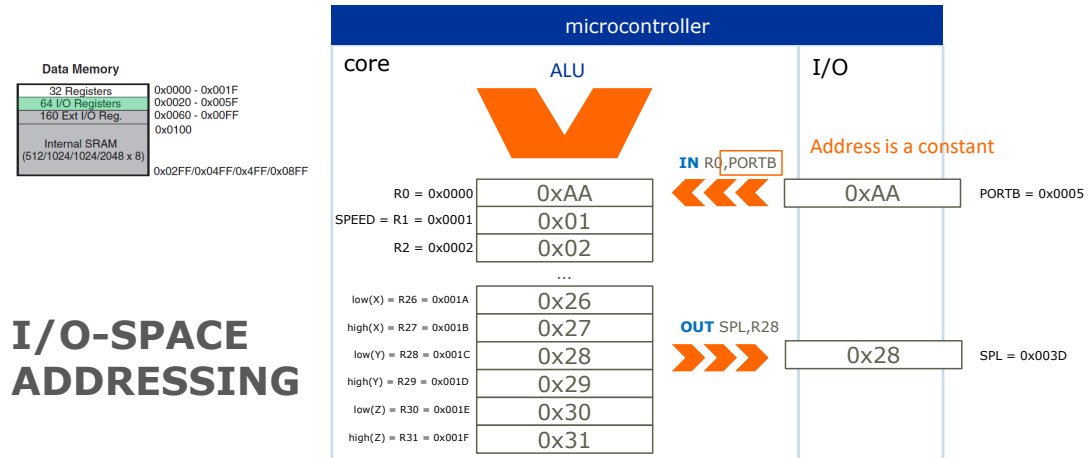
Data Memory	
32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Reg.	0x0060 - 0x00FF
Internal SRAM (512/1024/1024/2048 x 8)	0x0100
	0x02FF/0x04FF/0x4FF/0x08FF



Only the first 32 registers are usable with instructions. Furthermore there can be some limitations on the instructions, e.g. Multiply stores the result in specific registers. As well as registers with additional functions such as the addressing registers.

The LDI instruction can load a constant in a registers. Like LDI R16,0x02 loads 0x02 in register 16. Note: the LDI instruction can only be used on R16-R31. To load a constant in any of the registers below 16, one needs to free up a register in the upper half, load it there with LDI, then copy it to the intended register, (and restore the temporary register again if necessary)

Data memory: I/O-space ADDRESSING



PART 2: MEMORY

VUB - SENSORS & MICROSYSTEM ELECTRONICS

8

The IN and OUT instructions can move data from one of the 32 registers to the I/O registers.

The include file (atmega328.inc) defines a number of identifiers for certain registers. For any register in the 64 I/O registers, these identifiers are only valid to be used with IN and OUT. These registers can be accessed using LDS and STS, but then the identifiers are not valid.

Extended I/O registers

- Access with **LD(S)** and **ST(S)**
- Specific functions & peripheral config.
- Set to initial values on reset
 - See relevant datasheet sections

Data Memory	
32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Reg.	0x0060 - 0x00FF
Internal SRAM (512/1024/1024/2048 x 8)	0x0100 0x02FF/0x04FF/0x4FF/0x08FF

ATmega328 VOLATILE DATA MEMORY (SRAM)																
0	1	2	3	4	5	6	7									
0x0000	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								32 Registers (End address 0x001F)
0x0001	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0002	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0003	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0004	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0005	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0006	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0007	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0008	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0009	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x000A	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x000B	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x000C	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x000D	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x000E	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x000F	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0010	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0011	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0012	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0013	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0014	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0015	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0016	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0017	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0018	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0019	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x001A	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x001B	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x001C	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x001D	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x001E	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x001F	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0020	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0021	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0022	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0023	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0024	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0025	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0026	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0027	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0028	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0029	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x002A	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x002B	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x002C	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x002D	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x002E	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x002F	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0030	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0031	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0032	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0033	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0034	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0035	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0036	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0037	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0038	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0039	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x003A	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x003B	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x003C	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x003D	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x003E	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x003F	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0040	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0041	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0042	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0043	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0044	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0045	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0046	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0047	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0048	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0049	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x004A	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x004B	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x004C	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x004D	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x004E	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x004F	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0050	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0051	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0052	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0053	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0054	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0055	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0056	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0057	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0058	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0059	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x005A	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x005B	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x005C	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x005D	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x005E	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x005F	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0060	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0061	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0062	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0063	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0064	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0065	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0066	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0067	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0068	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0069	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x006A	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x006B	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x006C	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x006D	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x006E	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x006F	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0070	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0071	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0072	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0073	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0074	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0075	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0076	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0077	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x0078	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x0079	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x007A	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x007B	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x007C	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								64 I/O Registers (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF) (I/O address 0x0040 to 0x00FF)
0x007D	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x007E	R0R	R1R	R2R	R3R	R4R	R5R	R6R	R7R								
0x007F	R0R	R1R	R2R	R3R</												

The SRAM or data memory consists of 4 parts:

The 32 registers on which one can do operations.

64 I/O registers. And 160 extended I/O registers. These registers are used to configure the peripherals of the microcontroller.

The internal SRAM: this is free memory space to be freely used by the programmer

Although the microcontroller is 8 bit and as such all registers are 8 bit, the memory is addressed using 16bit addresses. Otherwise the amount of memory space would be limited to 255 bytes. The 16bit addressing is done through two 8 bit registers, one contains the upper 8 bit, and the other the lower 8 bit.

Internal SRAM

- Access with **LD(S)** and **ST(S)**
- Not zeroed on reset
- Completely usable by user
- Careful: stack is located at the end and counts backwards

Data Memory	
32 Registers	0x0000 - 0x001F
64 I/O Registers	0x0020 - 0x005F
160 Ext I/O Reg.	0x0060 - 0x00FF
Internal SRAM (512/1024/1024/2048 x 8)	0x0100
	0x02FF/0x04FF/0x4FF/0x08FF

ATmega328 VOLATILE DATA MEMORY (SRAM)															
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0x0000	R0	R1	R2	R3	R4	R5	R6	R7							
0x0008	R8	R9	R10	R11	R12	R13	R14	R15							
0x0010	R16	R17	R18	R19	R20	R21	R22	R23							
0x0018	R24	R25	R26	R27	R28	R29	R30	R31							
0x0020	PORTC	PIND	DODR0		PINB	DDRB	PORTD	PINC	DDRC						
0x0028															
0x0030					PCSR0	CSR0	TCR0	TCR1	TCR2						
0x0038					PCSR1	CSR1	TCR3	TCR4	TCR5						
0x0040	EECR	EEARL	EEARH	EECR	EECR	EECR	EECR	EECR	EECR						
0x0048	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT						
0x0050	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT						
0x0058	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT	AD_CONVERT						
0x0060	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x0068	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x0070	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x0078	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x0080	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x0088	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x0090	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x0098	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00A0	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00A8	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00B0	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00B8	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00C0	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00C8	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00D0	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00D8	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00E0	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00E8	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00F0	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x00F8	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR	WDR						
0x0100															
...															
...															
0x08FF															

The SRAM or data memory consists of 4 parts:

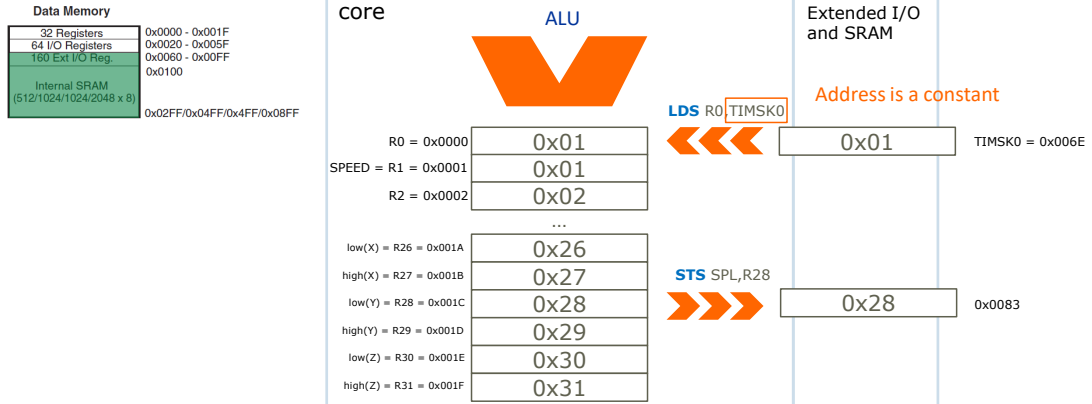
The 32 registers on which one can do operations.

64 I/O registers. And 160 extended I/O registers. These registers are used to configure the peripherals of the microcontroller.

The internal SRAM: this is free memory space to be freely used by the programmer

Although the microcontroller is 8 bit and as such all registers are 8 bit, the memory is addressed using 16bit addresses. Otherwise the amount of memory space would be limited to 255 bytes. The 16bit addressing is done through two 8 bit registers, one contains the upper 8 bit, and the other the lower 8 bit.

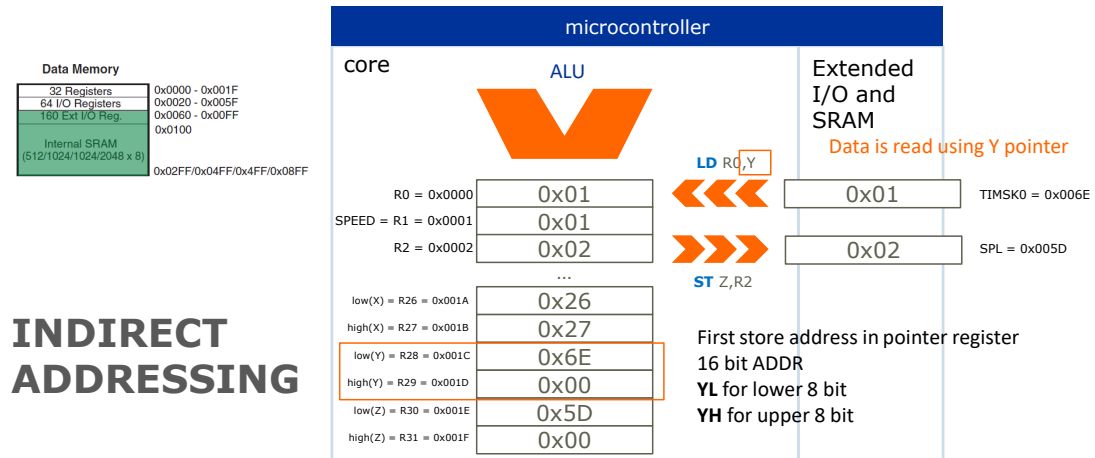
Data memory: DIRECT ADDRESSING



For the remaining part of the memory, the LDS/LD and STS/ST instructions are used. Two different methods of memory access exist: Direct and indirect.

The direct method is done through supplying the memory address to the LDS/STS instruction.

Data memory: INDIRECT ADDRESSING



PART 2: MEMORY

VUB - SENSORS & MICROSYSTEM ELECTRONICS

12

The other method is the indirect method. Here one first stores the 16-bit memory address in one of the three memory addressing registers of pointer register (each consisting of 2 separate registers to achieve 16bit addressing). After this, the memory address the register points to can be accessed using LD and ST

This method can be used to access consecutive memory locations or to calculate a certain memory address offset before using it to access that data.

Indirect addressing – Post-increment

```
;Code example: write 0xFF to
;addresses 0x108 to 0x10E in SRAM
```

```
;Set Y pointer to 0x108
```

```
LDI YH, high(0x0108);upper byte to YH (0x01)
```

```
LDI YL, low(0x0108);lower byte to YL (0x08)
```

```
LDI R20, 0xFF ;some data in a register
```

```
LDI R16, 8 ; loop iterator
```

```
Loop:
```

```
ST Y+, R20; store R20 in the current
address
```

```
DEC R16;decrement loop iterator
```

```
BRNE Loop; end of loop
```

Memory 4	
Memory: data IRAM	Address: 0x0100,data
data 0x0100	00 00 00 00 00 00 00 00
data 0x0108	ff ff ff ff ff ff ff ff
data 0x0110	00 00 00 00 00 00 00 00
data 0x0118	00 00 00 00 00 00 00 00

- First iteration (Y = 0x108):

- o ST writes R20 to 0x108
- o Y = Y + 1 after write

- 2nd iteration (Y = 0x109):

- o ST writes R20 to 0x109
- o Y = Y + 1 after write

- 3rd iteration (Y = 0x10A)

...

- 8th iteration (Y = 0x10E)

- o ST writes R20 to 0x10E
- o Y = Y + 1 after write

- End of loop (Y = 0x110)

Here we have a look to the POST-Increment feature of the indirect addressing method. This is useful when reading or writing data from or to consecutive addresses.

First the Y pointer upper and lower byte is loaded with the right part of the address with the help of the HIGH and LOW directive. Next the value 0xFF is stored in R20 and a loop of 8 iterations with R16 as iterator is made.

By adding a plus after the Y pointer in the ST instruction line. The compiler knows to use the ST with Y and post-increment bytecode when compiling.

The first iteration will write the value from R20 to the address Y is pointing to, in this iteration this is still address 0x108 which is the first address of the user SRAM. After the write, Y will be incremented by one, now pointing to 0x109 (YH = 0x01 and YL = 0x09)

The second iteration will write the value from R20 to the address Y is currently pointing to, in this iteration this has become address 0x10A. After the write, Y will be incremented by one, now pointing to 0x10A (YH = 0x01 and YL = 0x0A)

After the eight iteration, the SRAM addresses 0x108 to 0x10E will contain 0xFF as value. Y will now point to 0x110 (YH = 0x01 and YL = 0x10). The figure shows a screenshot of the SRAM from the simulator displaying that the writing was successful.

Note that when any lower pointer register overflows due to the increment, the carry will be added into the upper pointer register. This post-increment is supported by both the ST and LD instruction. The PRE-increment option does not exist in this instruction set.

Indirect addressing – Pre-decrement

;Code example: write 0xFF to
;addresses 0x108 to 0x100 in SRAM

;Set Y pointer to 0x108

LDI YH, high(0x0108);upper byte to YH (0x01)

LDI YL, low(0x0108);lower byte to YL (0x08)

LDI R20, 0xFF ;some data in a register

LDI R16, 8 ; loop iterator

Loop:

ST -Y, R20; store R20 in the current
address

DEC R16;decrement loop iterator

BRNE Loop; end of loop

Memory 4	
Memory:	data IRAM - Address: 0x0100_data
data 0x0100	ff ff ff ff ff ff ff yyyyyyyy
data 0x0108	00 00 00 00 00 00 00
data 0x0110	00 00 00 00 00 00 00
data 0x0118	00 00 00 00 00 00 00

▪ First iteration (Y = 0x108):

- Y = Y - 1 before write
- ST writes R20 to 0x107

▪ 2nd iteration (Y = 0x107):

- Y = Y - 1 before write
- ST writes R20 to 0x106

▪ 3rd iteration (Y = 0x106)

...

▪ 8th iteration (Y = 0x101)

- Y = Y - 1 before write
- ST writes R20 to 0x100

▪ End of loop (Y = 0x100)

Here we have a look to the PRE-decrement feature of the indirect addressing method. This is useful when reading or writing data from or to consecutive addresses.

First the Y pointer upper and lower byte is loaded with the right part of the address with the help of the HIGH and LOW directive. Next the value 0xFF is stored in R20 and a loop of 8 iterations with R16 as iterator is made.

By adding a minus after the Y pointer in the ST instruction line. The compiler knows to use the ST with Y and pre-decrement bytecode when compiling.

In the first iteration will first decrement Y and then write the value from R20 to the address Y is pointing to. Before the decrement Y points to 0x108. This gets decremented to 0x107 and then the write is done to that address. After the write, Y will pointing to 0x107 (YH = 0x01 and YL = 0x07)

The second iteration will decrement Y from 0x107 to 0x106 before writing the value from R20 to the address Y is currently pointing to, in this iteration this has become address 0x106. After the write, Y will pointing to 0x106 (YH = 0x01 and YL = 0x06)

After the eight iteration, the SRAM addresses 0x107 to 0x100 will contain 0xFF as value. Y will now point to 0x100 (YH = 0x01 and YL = 0x00). The figure shows a screenshot of the SRAM from the simulator displaying that the writing was successful.

Note that when any lower register underflows due to the decrement, this carries over in the upper register. For example predecrement from 0x100 (YH = 0x01 and YL = 0x00) results in Y= 0x0FF (YH = 0x00 and YL = 0xFF). The post-decrement option does not exist in this instruction set.

Keywords & addresses

- R0 – R31 are built into the compiler
- Non memory mapped (IO registers) :
 - Use keyword with **IN** and **OUT**
 - Use exact address with **IN** and **OUT**
 - compensate 0x20 offset (**IN** Rx, [keyword] - 0x20)
 - Can be used with **LDS** and **STS** (slower)
 - Keyword + 0x20
 - Exact address from memory overview excel
- Memory mapped (ext IO registers) :
 - Does NOT work with **IN** and **OUT**
 - Can be used with **LDS** and **STS** (slower)
 - Keyword
 - Exact address

m328pdef.inc

```
.equ TIMSK2= 0x70; MEMORY MAPPED
.equ TIMSK1= 0x6f; MEMORY MAPPED
.equ TIMSK0= 0x6e; MEMORY MAPPED
.equ PCMSK1= 0x6c; MEMORY MAPPED
.equ PCMSK2= 0x6d; MEMORY MAPPED
.equ PCMSK0= 0x6b; MEMORY MAPPED
.equ EICRA= 0x69; MEMORY MAPPED
.equ PCICR= 0x68; MEMORY MAPPED
.equ OSCCAL= 0x66; MEMORY MAPPED
.equ PRR= 0x64; MEMORY MAPPED
.equ CLKPR= 0x61; MEMORY MAPPED
.equ WDTCR= 0x60; MEMORY MAPPED
.equ SREG= 0x3f
.equ SPL= 0x3d
.equ SPH= 0x3e
.equ SPMCSR= 0x37
.equ MCUCR= 0x35
.equ MCUSR= 0x34
.equ SMCR= 0x33
.equ ACSR= 0x30
.equ SPDR= 0x2e
.equ SPSR= 0x2d
.equ SPCR= 0x2c
.equ GPIOR2= 0x2b
.equ GPIOR1= 0x2a
.equ OCR0B= 0x28
.equ OCR0A= 0x27
```

Memory overview

- See Section 8. AVR Memories in [AT328P_microcontroller.pdf](#)
- [MemoryOverview.xlsx](#)

ATmega328 VOLATILE DATA MEMORY (SRAM)

The diagram illustrates the memory layout of the ATmega328, showing the distribution of SRAM, EEPROM, and Flash memory across the 16 banks of the volatile data memory.

Legend:

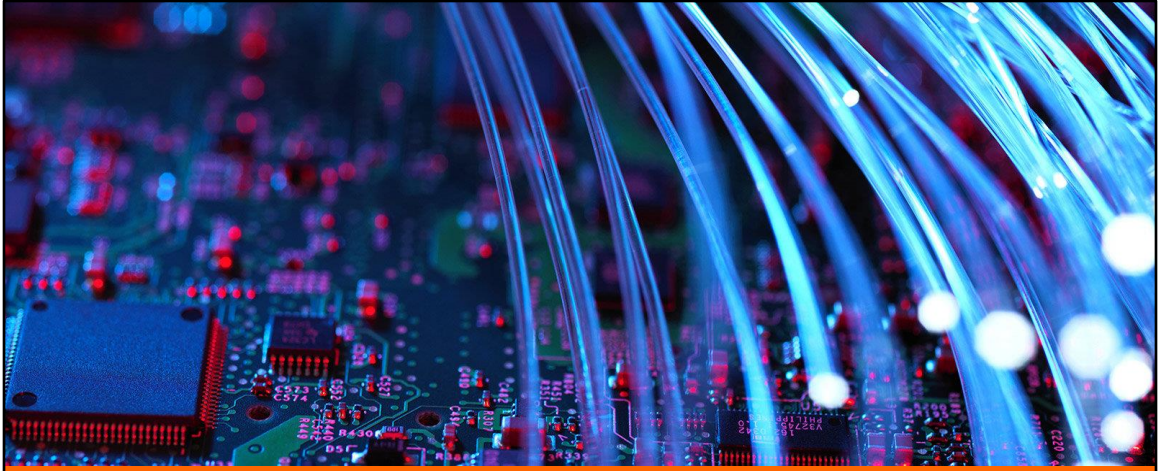
- EEPROM
- SRAM
- Flash
- EEPROM (16 bytes)
- SRAM (16 bytes)
- Flash (16 bytes)
- EEPROM (16 bytes)
- SRAM (16 bytes)
- Flash (16 bytes)
- EEPROM (16 bytes)
- SRAM (16 bytes)
- Flash (16 bytes)
- EEPROM (16 bytes)
- SRAM (16 bytes)
- Flash (16 bytes)

Memory Map:

Bank	Address Range	Memory Type	Size (bytes)
Bank 0	0x0000 - 0x000F	EEPROM	16
Bank 1	0x0010 - 0x001F	SRAM	16
Bank 2	0x0020 - 0x002F	Flash	16
Bank 3	0x0030 - 0x003F	EEPROM	16
Bank 4	0x0040 - 0x004F	SRAM	16
Bank 5	0x0050 - 0x005F	Flash	16
Bank 6	0x0060 - 0x006F	EEPROM	16
Bank 7	0x0070 - 0x007F	SRAM	16
Bank 8	0x0080 - 0x008F	Flash	16
Bank 9	0x0090 - 0x009F	EEPROM	16
Bank 10	0x00A0 - 0x00AF	SRAM	16
Bank 11	0x00B0 - 0x00BF	Flash	16
Bank 12	0x00C0 - 0x00CF	EEPROM	16
Bank 13	0x00D0 - 0x00DF	SRAM	16
Bank 14	0x00E0 - 0x00EF	Flash	16
Bank 15	0x00F0 - 0x00FF	EEPROM	16

Additional Information:

- The ATmega328 has a total of 16 banks of volatile data memory, each 16 bytes in size.
- The total size of the volatile data memory is 256 bytes (16 banks * 16 bytes/bank).
- The memory is divided into three main sections: EEPROM (16 banks), SRAM (16 banks), and Flash (16 banks).
- The diagram shows the distribution of memory types across the banks, with each bank containing a mix of EEPROM, SRAM, and Flash memory.



End of Part 2: Memory