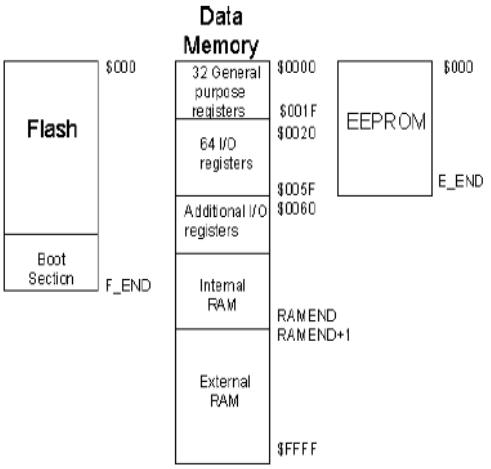
AVR address spaces

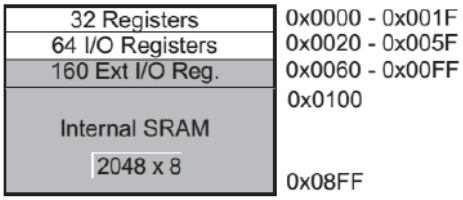


ATmega 328P / 2560

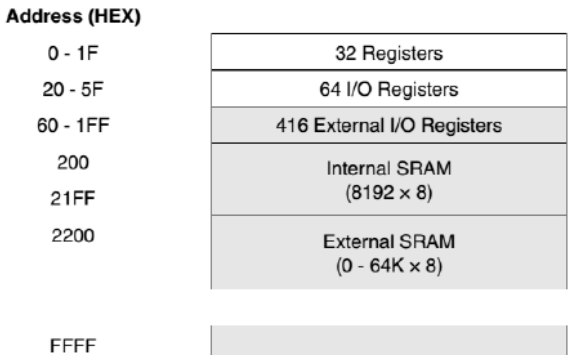
Flash: 0x0000 – 0x3FFF / 0x1FFF
- Program Counter (PC) - 14 / 17 bits
- Program memory is addressable on words (2 bytes): 16K x 16 biti (32KB) / 128Kx16 bits (256 KB)
- Program memory cannot be extended

Data memory: is addressable on bytes
Registers: 32 GPR, 64 I/O, 160/416 extended I/O
(0x0000 – 0x001F): General purpose registers (GPR):
(0x0020 – 0x005F): direct I/O access or as memory
(0x0060 – 0x00FF / 0x01FF): Extended I/O in SRAM, access with ST/STS/STD si LD/LDS/LDD
(0x0100 – 0x08FF / 0x0200 – 0x21FF): internal SRAM

Atmega 328P (Data Memory)



ATmega 2560 (Data Memory)



32 GPR

- 32 x 8-bit GPR (R0 – R31);
- 1 cycle read/write

7 0	Addr.	
R0	0x00	
R1	0x01	
R2	0x02	
...		
R13	0x0D	
R14	0x0E	
R15	0x0F	
R16	0x10	
R17	0x11	
...		
R26	0x1A	X-register Low Byte
R27	0x1B	X-register High Byte
R28	0x1C	Y-register Low Byte
R29	0x1D	Y-register High Byte
R30	0x1E	Z-register Low Byte
R31	0x1F	Z-register High Byte

Groups of registers:
R(0:15), R(16:31), R(26:31) – various possible usage
X, Y, Z, (16 bits); indirect addressing of instruction and data memory

Requirements for the register block (1 cycle operations) - input/output schemes:
•One 8-bit output operand and one 8-bit result input.
•Two 8-bit output operands and one 8-bit result input.
•Two 8-bit output operands and one 16-bit result input.
•One 16-bit output operand and one 16-bit result input.

Ex: ATmega 2560 (MEGA): http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf

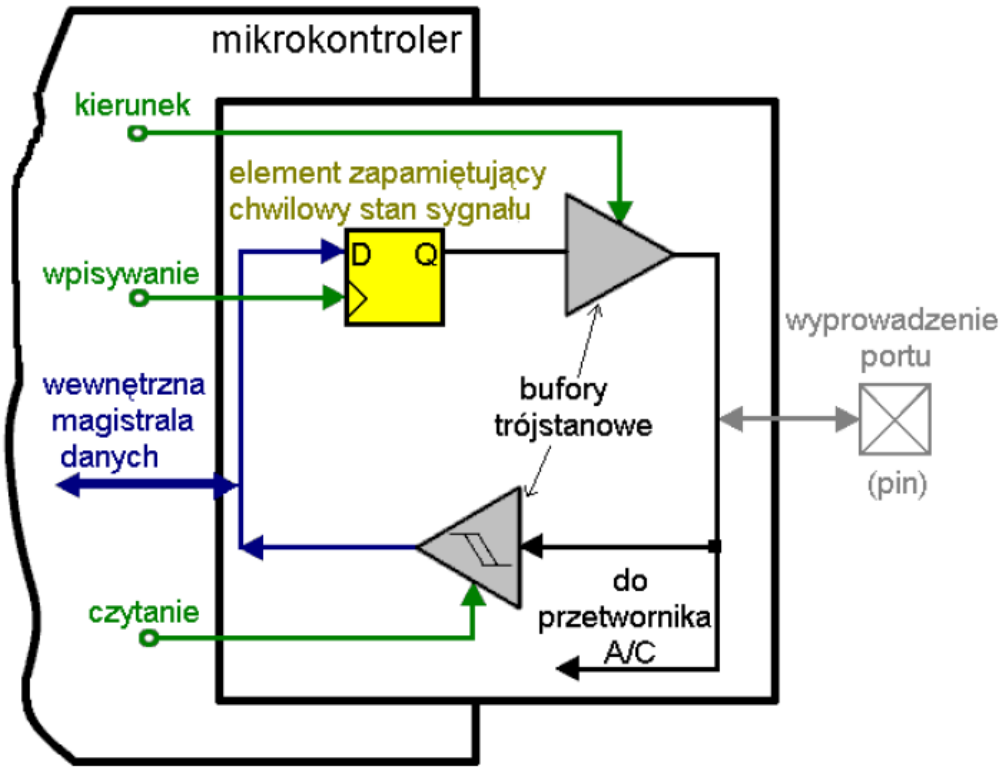
0x17 (0x37)	TIFR2	-	-	-	-	-	OCF2B	OCF2A	TOV2
0x16 (0x36)	TIFR1	-	-	ICF1	-	OCF1C	OCF1B	OCF1A	TOV1
0x15 (0x35)	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0
0x14 (0x34)	PORTG	-	-	PORTG5	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0
0x13 (0x33)	DDRG	-	-	DDG5	DDG4	DDG3	DDG2	DDG1	DDG0
0x12 (0x32)	PING	-	-	PING5	PING4	PING3	PING2	PING1	PING0
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0
0x0E (0x2E)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0
0x0D (0x2D)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0
0x0C (0x2C)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
0x02 (0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0
0x01 (0x21)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0
0x00 (0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0

64 Porty wejścia-wyjścia

Porty wejścia-wyjścia

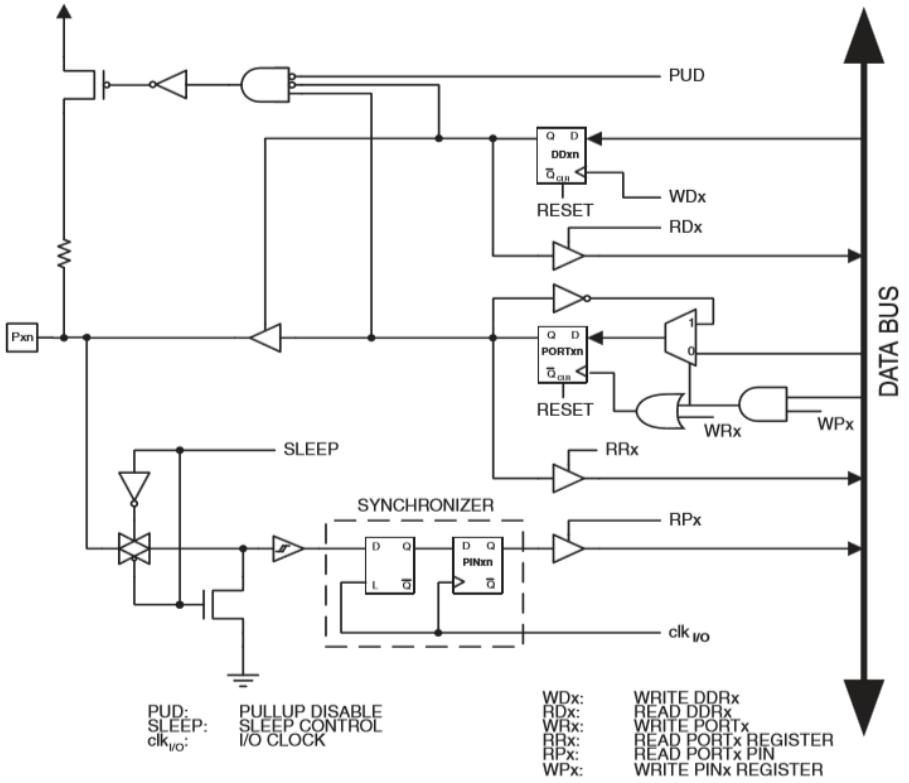
Zazwyczaj z dwukierunkowym portem we/wy związane są trzy rejestry:

- rejestr **danych wyjściowych** (PORTx dla mk ATmega16),
- rejestr **kierunku przepływu danych** (DDRx),
- rejestr **danych wejściowych** (PINx).



Rys. 2.37. Schematyczna budowa pojedynczej linii portu równoległego

Figure 13-2. General Digital I/O[1]



Note: 1. WRx, WPx, WDx, RRx, RPx, and RDx are common to all pins within the same port. $clk_{I/O}$, SLEEP, and PUD are common to all ports.

Control and status registers

SREG - Status Register

Bit	7	6	5	4	3	2	1	0	
0x3F (0x5F)	I	T	H	S	V	N	Z	C	SREG
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Bit 7 – I: Global Interrupt Enable
I = SREG(7) ← 1/0; SEI/CLI; global validation/invalidation of interrupts

Bit 6 – T: Bit Copy Storage
Bit Copy: BLD (Bit LoaD) RegFile (Register(i)) ← SREG(T)
BST (Bit STore) SREG(T) ← RegFile (Register(i))

Bit 5 – H: Half Carry Flag; BCD arithmetic.

Bit 4 – S: Sign Bit, S = N xor V

Bit 3 – V: Two's Complement Overflow Flag

Bit 2 – N: Negative Flag; negative result in an arithmetic or logic operation.

Bit 1 – Z: Zero Flag; zero result in an arithmetic or logic operation.

Bit 0 – C: Carry Flag; carry in an arithmetic or logic operation.

SREG register can be accessed as follows:

Assembly:
read: in ri, SREG
write: out SREG, ri

C language)
byte sr;
sr=SREG;
SREG=sr;

Control and status registers

MCUCR – MCU Control Register

MCUCR – MCU Control Register

Bit	7	6	5	4	3	2	1	0	
0x35 (0x55)	JTD	–	–	PUD	–	–	IVSEL	IVCE	MCUCR
Read/Write	R/W	R	R	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- **Bit 4 – PUD: Pull-up Disable**

When this bit is written to one, the I/O ports pull-up resistors are disabled even if the DDxn and PORTxn Registers are configured to enable the pull-up resistor ({DDxn, PORTxn} = 0b01). See "Configuring the Pin" on page 68 for more details about this feature.

Rejestr Stanu (statusowy - SREG, 0x5F)

Rejestr stanu zawiera informacje o rezultacie ostatniej operacji arytmetycznej. Ta informacja może zostać wykorzystana poprzez różne programy do wykonania operacji warunkowych. Należy zwrócić uwagę, że rejestr stanu jest aktualizowany po każdej operacji na ALU. TO w wielu przypadkach sprawia, że niepotrzebne jest używanie wyspecjalizowanych instrukcji porównywania, co wpływa na przyspieszenie działania kodu i bardziej zwięzły kod.

Rejestr stanu nie jest automatycznie zapamiętywany kiedy procesor przechodzi do obsługi przerwania i ustawiany podczas powrotu z przerwania. To musi zostać obsłużone przez oprogramowanie.

Rejestr stanu – SREG – jest zdefiniowany jako:

bit 7	6	5	4	3	2	1	0
I	**T**	**H**	**S**	**V**	**N**	**Z**	**C**

Bit 7 – I: Globalne odblokowanie przerwań (global Interrupt enable)

Bit ten musi być ustawiony na 1 dla zezwolenia na przyjmowanie przerwań. Indywidualna kontrola przerwań jest wtedy wykonywana w oddzielnym rejestrze kontrolnym. Jeżeli globalny rejestr przerwań jest wyzerowany, żadne z przerwań nie może zostać obsłużone niezależnie od indywidualnych ustawień przerwań. I-bit jest ustawiany w stan 1 rozkazem SEI i zerowany rozkazem CLI. Jest też wyzerowany sprzętowo po zajściu przerwania i jest ustawiany przez rozkaz RETI, aby można było obsługiwać następne przerwania.

Bit 6 – T: rejestr bitowy (bit copy sTorage)

Rozkazy kopiowania bitu BLD i BST korzystają z bitu T jako źródło lub bit docelowy dla zmienianego bitu. Rozkaz BST kopiuje bit z rejestru do bitu T, natomiast rozkaz BLD kopiuje bit T do rejestru w zestawie rejestrów.

Bit 5 – H: znacznik przeniesienia połówkowego (Half carry flag)

Flaga ta wskazuje przeniesienie w niektórych operacjach arytmetycznych gdy pojawi się przeniesienie z bitu 3 na 4. Flaga ta jest wykorzystywana przy arytmetyce BCD.

Bit 4 – S: bit znaku S= N ⊕ V (Exclusive-OR)

Bit znaku przyjmuje wartość logiczną 1gdy wynik operacji jest jawnie ujemny i gdy nie nastąpiło przepełnienie. Dzięki temu znacznik ten przyjmuje wartość poprawną nawet wówczas, gdy znacznik wartości ujemnej N został ustawiony wskutek przepełnienia.

Bit 3 – V: znacznik przepełnienia uzupełnienia do dwóch
Flaga ta wspiera arytmetykę dwójkową w kodzie U2.

Bit 2 – N: znacznik wartości ujemnej
Równy najstarszemu bitowi wyniku. Flaga ta wskazuje na ujemny rezultat w operacjach logicznych lub arytmetycznych.

Bit 1 – Z: znacznik wartości zerowej
Ustawiany jest na wartość 1 gdy wszystkie bity wyniku przyjmują wartość zerową.
Flaga ta wskazuje zerowy rezultat w operacjach arytmetycznych i logicznych.

Bit 0 – C: znacznik przeniesienia
Flaga przeniesienia wskazuje na przeniesienie podczas operacji logicznej lub arytmetycznej.

Stos

Stos jest głównie używany do przechowywania tymczasowych danych, lokalnych zmiennych i adresów powrotów z przerwań lub wywołań podprogramów. Rejestr wskaźnika stosu zawsze pokazuje na górę stosu. Stos jest zaimplementowany rosnąco w pamięci od wyższych numerów do niższych. Wpływa to na to, że instrukcja PUSH zmniejsza wartość wskaźnika stosu.
Wskaźnik stosu wskazuje na dane w przestrzeni stosu w SRAM, gdzie są ulokowane stosy Przerwań i Wywołań podprogramów. Rozmiar stosu musi być zdefiniowany zanim może dojść do wykonania podprogramów lub przerwań. Wskaźnik stosu musi zostać ustawiony tak aby wskazywał ponad adres 0x0200. Początkowa wartość wskaźnika stosu wskazuje ostatni adres wewnętrznej pamięci SRAM. Wskaźnik stosu jest dekrementowany o jeden kiedy dane są odkładane na stos przez instrukcję PUSH i jest dekrementowany o dwa (lub 3 dla procesora ATMega256) kiedy odkładany jest adres powrotu z przerwania lub podprogramu. Wskaźnik stosu jest inkrementowany o jeden kiedy dane pobierane są przez instrukcję POP a o dwa (lub 3 dla procesora ATMega256) przy powrotach z podprogramów - RET i przerwań – RETI.
Wskaźnik stosu w AVR jest zaimplementowany jako dwa 8-bitowe rejestry w przestrzeni I/O.

SPH (0x5E)	SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8
SPL (0x5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0

Ich zawartość musi być ustawiona na początku programu zanim procesor zacznie korzystać z procedur lub zezwoli na zgłaszanie przerwań. Typowy przykład inicjalizacji wskaźnika stosu wygląda następująco:

```
ldi  R16, high(RAMEND)    ; ładuj rejestr roboczy MSB adresu końca pamięci SRAM
out  SPH, R16             ; inicjalizuj MSB wskaźnika stosu
ldi  R16, low(RAMEND)     ; ładuj rejestr roboczy LSB adresu końca pamięci SRAM
out  SPL, R16             ; inicjalizuj LSB wskaźnika stosu
```

Instruction set

Instructions on 16 or 32 bits
The position of the registers addresses can vary in different instructions

Instruction types
- ARITHMETIC AND LOGIC INSTRUCTIONS
- BRANCH INSTRUCTIONS
- DATA TRANSFER INSTRUCTIONS
- BIT AND BIT-TEST INSTRUCTIONS
- MCU CONTROL INSTRUCTIONS

Registers and Operands (notations):
Rd: Destination (and source) register in the Register File
Rr: Source register in the Register File
R: Result after instruction is executed
K: Constant data
k: Constant address
b: Bit in the Register File or I/O Register (3-bit)
s: Bit in the Status Register (3-bit)
X,Y,Z: Indirect Address Register
(X=R27:R26, Y=R29:R28 and Z=R31:R30)
A: I/O location address
q: Displacement for direct addressing (6-bit)

Data Transfer Instructions

DATA TRANSFER INSTRUCTIONS			
MOV	Rd, Rr	Move Between Registers	Rd ← Rr
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr
LDI	Rd, K	Load Immediate	Rd ← K
LD	Rd, X	Load Indirect	Rd ← (X)
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)
LD	Rd, Y	Load Indirect	Rd ← (Y)
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)
LD	Rd, Z	Load Indirect	Rd ← (Z)
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)
ST	X, Rr	Store Indirect	(X) ← Rr
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr
ST	Y, Rr	Store Indirect	(Y) ← Rr
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr
ST	Z, Rr	Store Indirect	(Z) ← Rr
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr
STS	k, Rr	Store Direct to SRAM	(k) ← Rr
LPM		Load Program Memory	R0 ← (Z)
LPM	Rd, Z	Load Program Memory	Rd ← (Z)
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1
SPM		Store Program Memory	(Z) ← R1:R0
IN	Rd, P	In Port	Rd ← P
OUT	P, Rr	Out Port	P ← Rr
PUSH	Rr	Push Register on Stack	STACK ← Rr
POP	Rd	Pop Register from Stack	Rd ← STACK

Data Transfer Instructions

Instruction formats and restrictions

LDI Rd,K 16 ≤ d ≤ 31, 0 ≤ K ≤ 255 PC ← PC + 1

1110	KKKK	dddd	KKKK

(i) LD Rd, X 0 ≤ d ≤ 31 PC ← PC + 1
(ii) LD Rd, X+ 0 ≤ d ≤ 31 PC ← PC + 1
(iii) LD Rd, -X 0 ≤ d ≤ 31 PC ← PC + 1

(i)	1001	000d	dddd	1100
(ii)	1001	000d	dddd	1101
(iii)	1001	000d	dddd	1110

LD

(i) LPM None, R0 implied PC ← PC + 1
(ii) LPM Rd, Z 0 ≤ d ≤ 31 PC ← PC + 1
(iii) LPM Rd, Z+ 0 ≤ d ≤ 31 PC ← PC + 1

(i)	1001	0101	1100	1000
(ii)	1001	000d	dddd	0100
(iii)	1001	000d	dddd	0101

LPM

IN Rd,A 0 ≤ d ≤ 31, 0 ≤ A ≤ 63 PC ← PC + 1

1011	0AAd	dddd	AAAA

IN

OUT A,Rr 0 ≤ r ≤ 31, 0 ≤ A ≤ 63 PC ← PC + 1

1011	1AAr	rrrr	AAAA

OUT

Data Transfer Instructions - examples

Load Immediate

```
char a;
…
a = 0x10;
==============================
ldi r24, 0x10      ; Load imm 10
sts a, r24         ; Store to a
------------------------------------------
int a;
…
a = 0x2030;
==============================
???
```

Load direct

```
char a, b;
…
a = b;
================
lds r24, b         ; Load from b
sts a, r24         ; Store to a
------------------------------------------
int a, b;
…
a = b;
================
???
```

AVR Instructions: restrictions

About Code Examples – Datasheet: I/O Regs in extended I/O map:
For I/O Registers located in extended I/O map, “IN”, “OUT”, “SBIS”, “SBIC”, “CBI”, and “SBI” instructions must be replaced with instructions that allow access to extended I/O. Typically “LDS” and “STS” combined with “SBRS”, “SBRC”, “SBR”, and “CBR”.

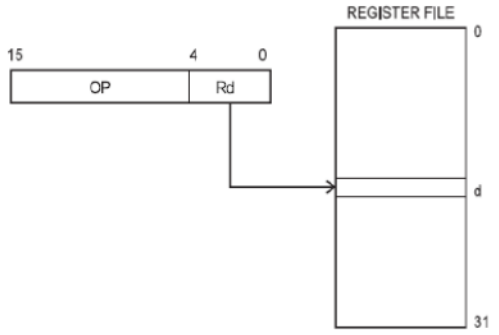
AVR - addressing modes

AVR, addressing modes

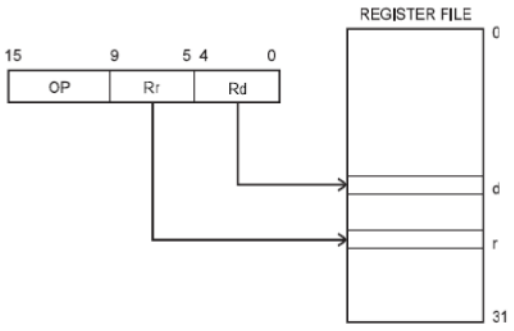
- Register Direct, with 1 and 2 registers
- I/O Direct
- Data Direct
- Data Indirect
  - with pre-decrement
  - with post-increment
- Addressing instruction memory

The Program and Data Addressing Modes

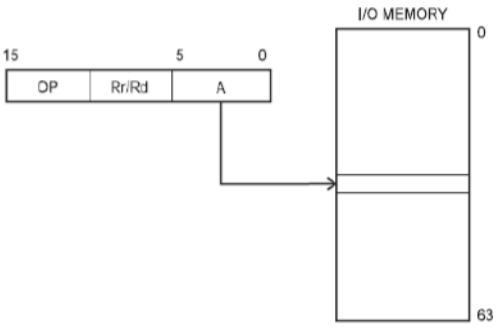
Direct Register Addressing



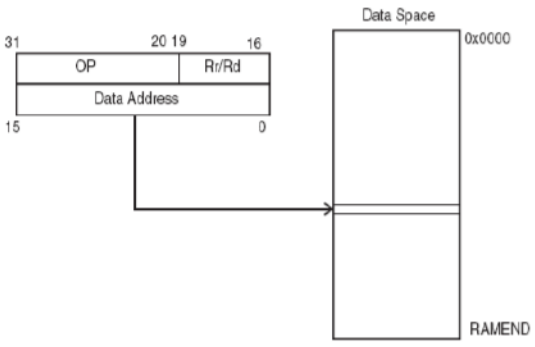
Direct Single Register Addressing



Direct Register Addressing, Two Registers



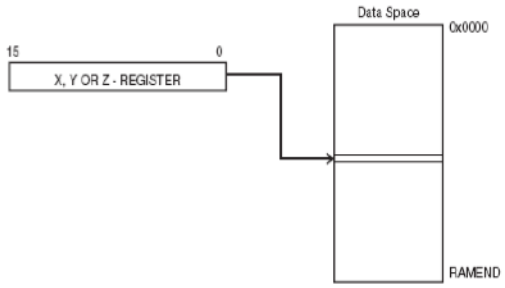
I/O Direct Addressing



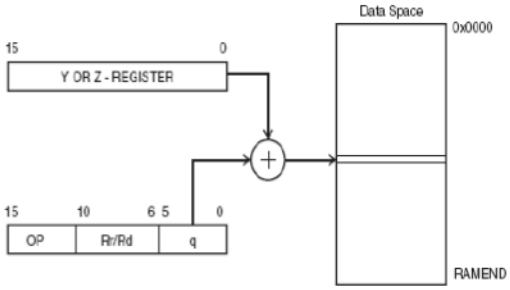
Direct Data Addressing

The Program and Data Addressing Modes

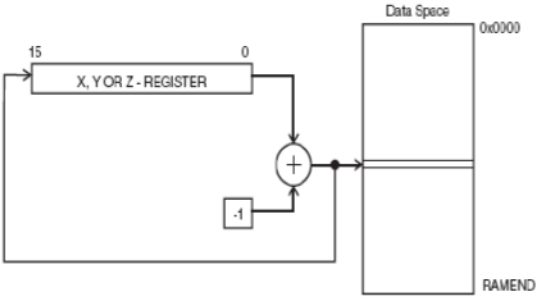
Indirect Data Addressing



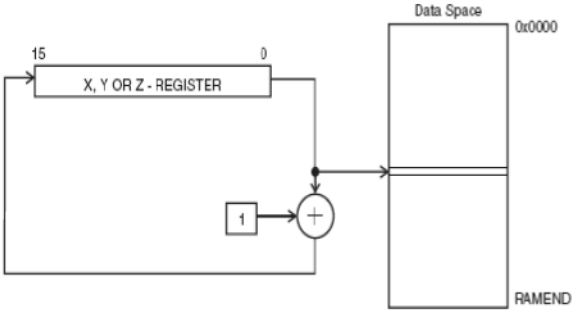
Indirect Data Addressing



Indirect Data addressing with Displacement



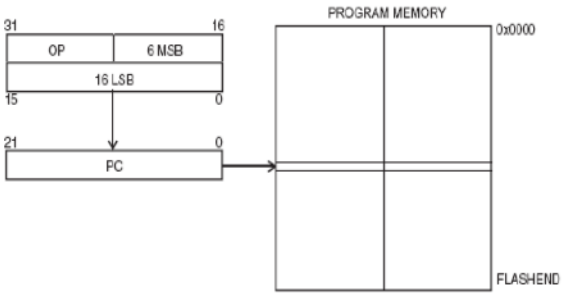
Indirect Data Addressing with Pre-decrement



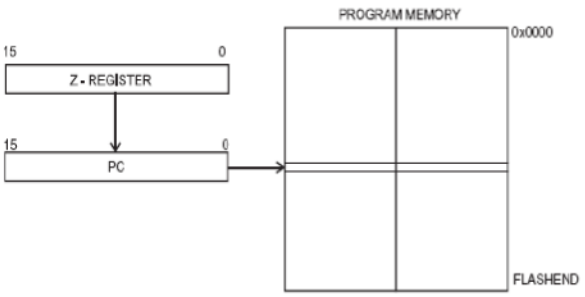
and Post-increment

The Program and Data Addressing Modes

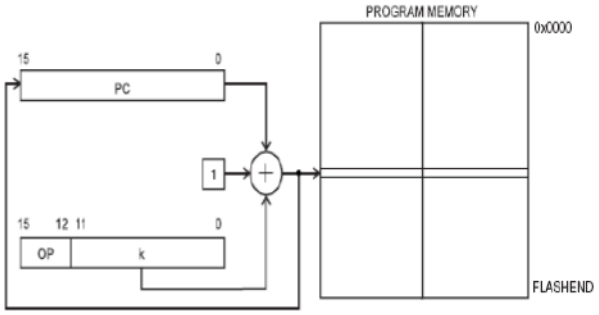
Program Addressing



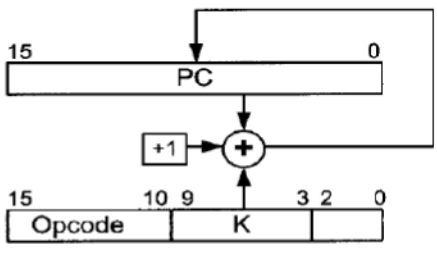
Direct Program Addressing, JMP, CALL



Indirect Program Addressing, IJMP, ICALL



Relative Program Addressing, RJMP, RCALL



Conditional Branch – short relative jump

Data Transfer Instructions - examples

Load Indirect

```
int a = *pInt;
==============================
; Use the Z register (R31:R30)
lds R30, pInt      ; Load from pInt
lds R31, pInt+1    ;
ld r24, Z          ; load from (*pInt)
ldd r25, Z+1       ;
sts a, r24         ; store to a
sts a+1, r25       ;
-----------------------------------------------
```

Store Indirect

```
* (int*) (0x0100) = b;
============================
; Use Z register (R31:R30)
ldi r30, 0         ; load imm 0x0100
ldi r31, 0x01      ;
lds r24, b         ; load b
lds r25, b+1       ;
st Z, r24          ; store to 0x0100
std Z+1, r25       ;
-----------------------------------------------
```

String Copy

```
void strcpy (char *dst, char *src)
{ char ch;
do {
ch = *src++;
*dst++ = ch;
} while (ch);
}
```

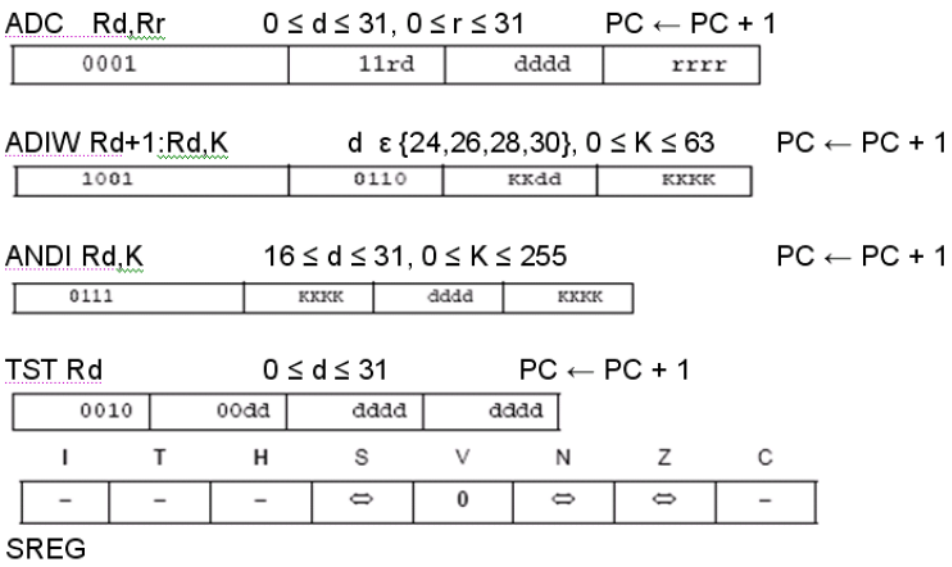
```
; dst in R25:R24, src in R23:R22
strcpy:
movw r30, r24    ; Z<=dst
movw r26, r22    ; X<=src
loop:
  ld r20, X+     ; ch=*src++
  st Z+, r20     ; *dst++=ch
  tst r20        ; ch==0?
  brne loop      ; loop if not
ret
```

Arithmetic and Logic Instructions

Mnemonics	Operands	Description	Operation
ARITHMETIC AND LOGIC INSTRUCTIONS			
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr
COM	Rd	One's Complement	Rd ← 0xFF – Rd
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)
INC	Rd	Increment	Rd ← Rd + 1
DEC	Rd	Decrement	Rd ← Rd – 1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd
SER	Rd	Set Register	Rd ← 0xFF
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1

Arithmetic and Logic Instructions

Instruction formats and restrictions



A&L instructions - examples

C language

```
int a, b;
…
a = a + b;
```

Assambly language

```
lds r18, a          ; load a
lds r19, a+1        ;
lds r24, b          ; load b
lds r25, b+1        ;
add r24, r18        ; add lower half
adc r25, r19        ; add higher half
sts a+1, r25        ; store a.byte1
sts a, r24          ; store a.byte0
```

```
int a;
…
a -= 0x4321;
```

```
lds r24, a          ; load from a
lds r25, a+1        ;
subi r24, 0x21      ; sub imm 0x21
sbci r25, 0x43      ; sub imm 0x43 with
                      carry
sts a+1, r25        ; store a
sts a, r24          ;
```
