ATmega1284P Assembly I

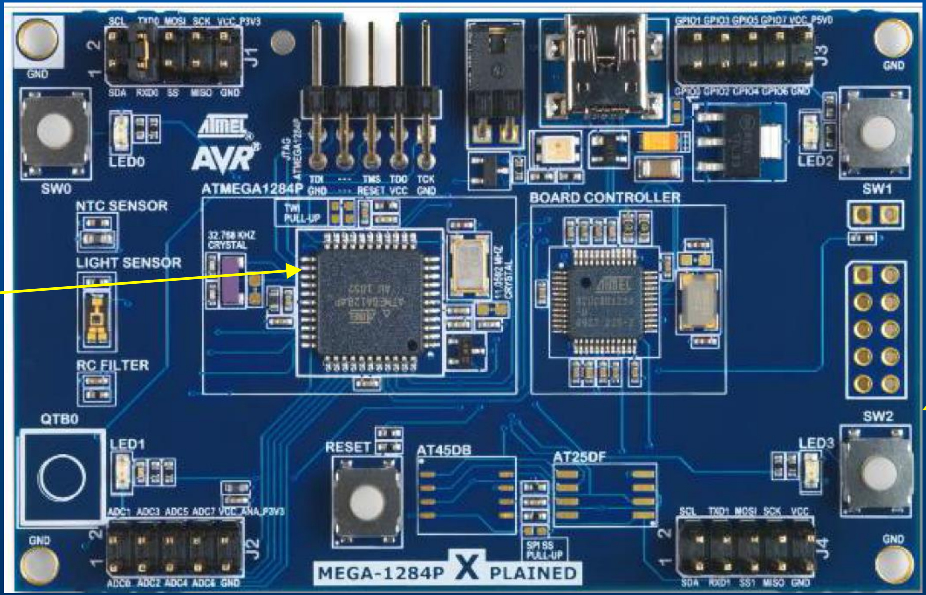
Outline:

- ATmega1284P architecture
- AVR assembly language
- Elementary example program
- AVR Assembler
- Using the STUDIO7 simulator

The ATmega1284P Microprocessor

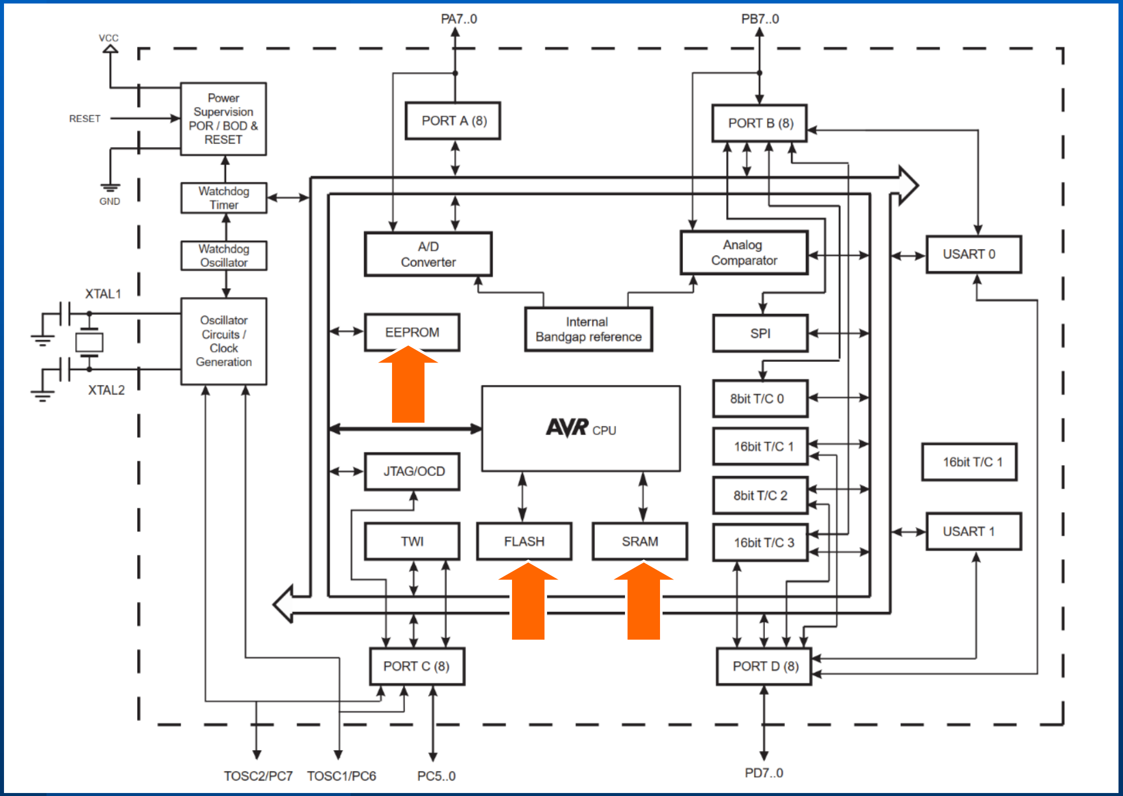
- *In this course you will be using the ATmega1284P microprocessor mounted on the MEGA-1284P Xplained Board*

MEGA-1284P Microcontroller



MEGA-1284P Xplained Evaluation Board

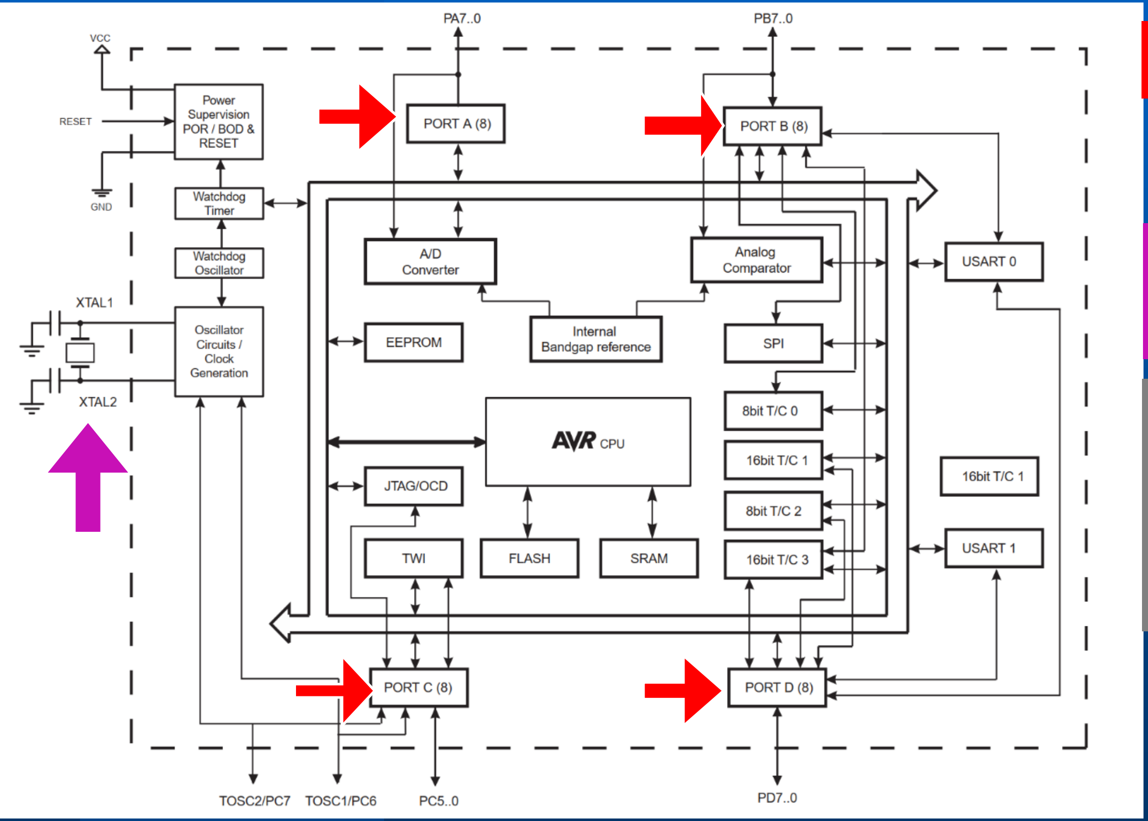
ATmega1284P diagram



- RISC Architecture
- 131 Instructions
- 32 x 8 Registers
- 20 MIPS @ 20 MHz

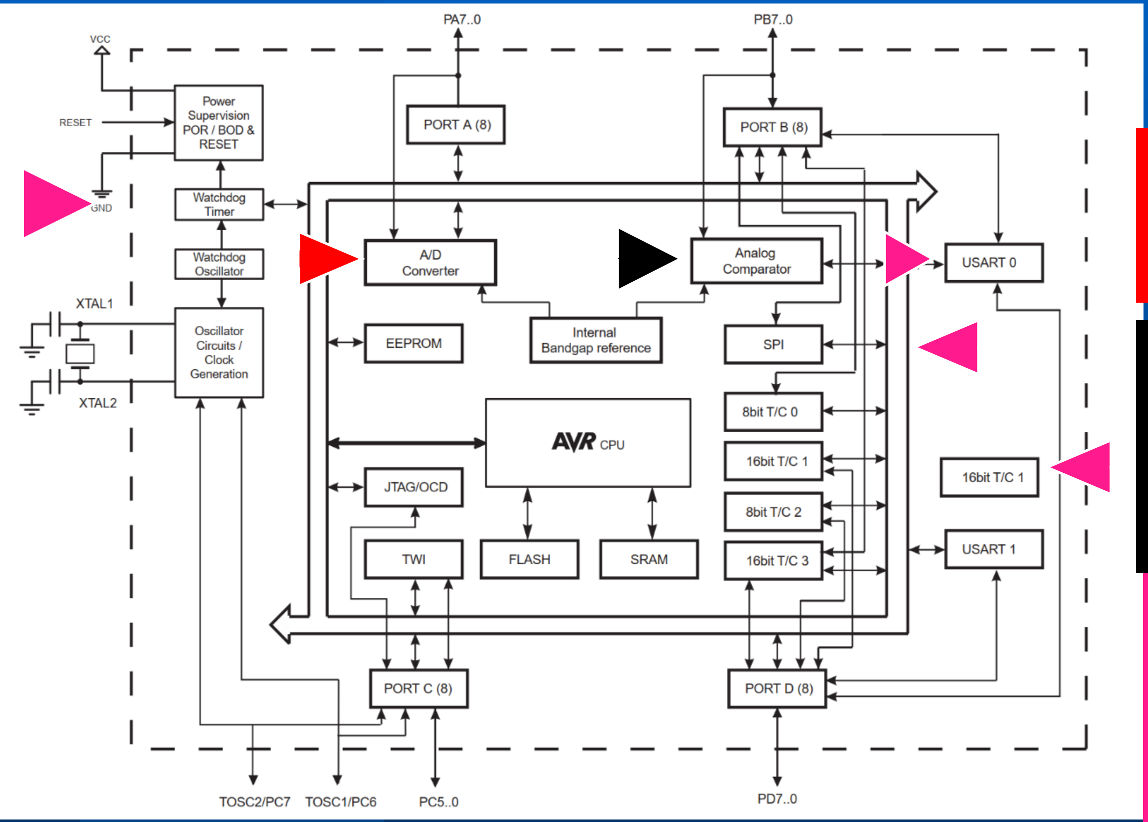
- 128 Kbytes In-System Prog. Flash Memory
- 16K Kbytes SRAM
- 4 Kbytes In-System EEPROM

ATmega1284P Peripherals I



- 4 I/O PORTS: A, B, C, D
- XTAL provides the 20 MHz clock
- Port Pins Usually have other functions except I/O

ATmega1284 Peripherals II

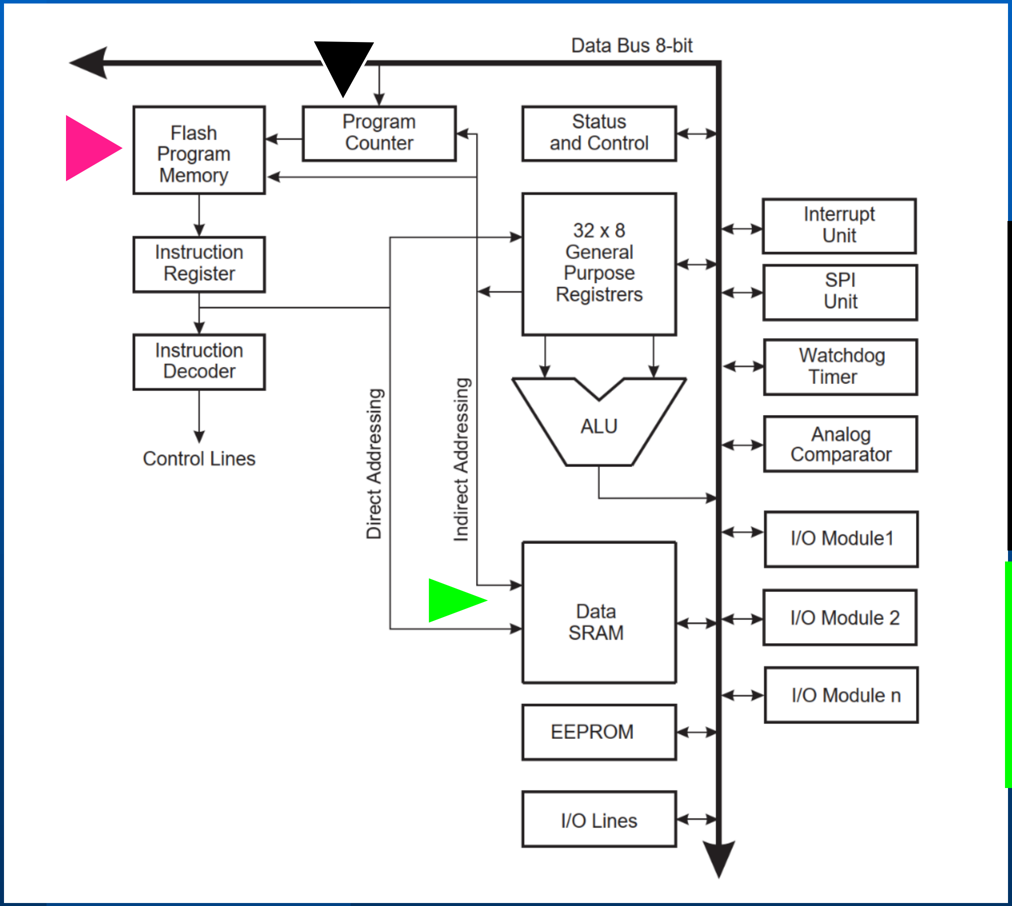


8-Channel 10 Bit ADC with interrupt

On Chip analog Comparator with Interrupt.

- **Prog. Watchdog Timer**
- **Real Time Counters**
- **SPI Interface**
- **UART**

The ATmega1284P Architecture I

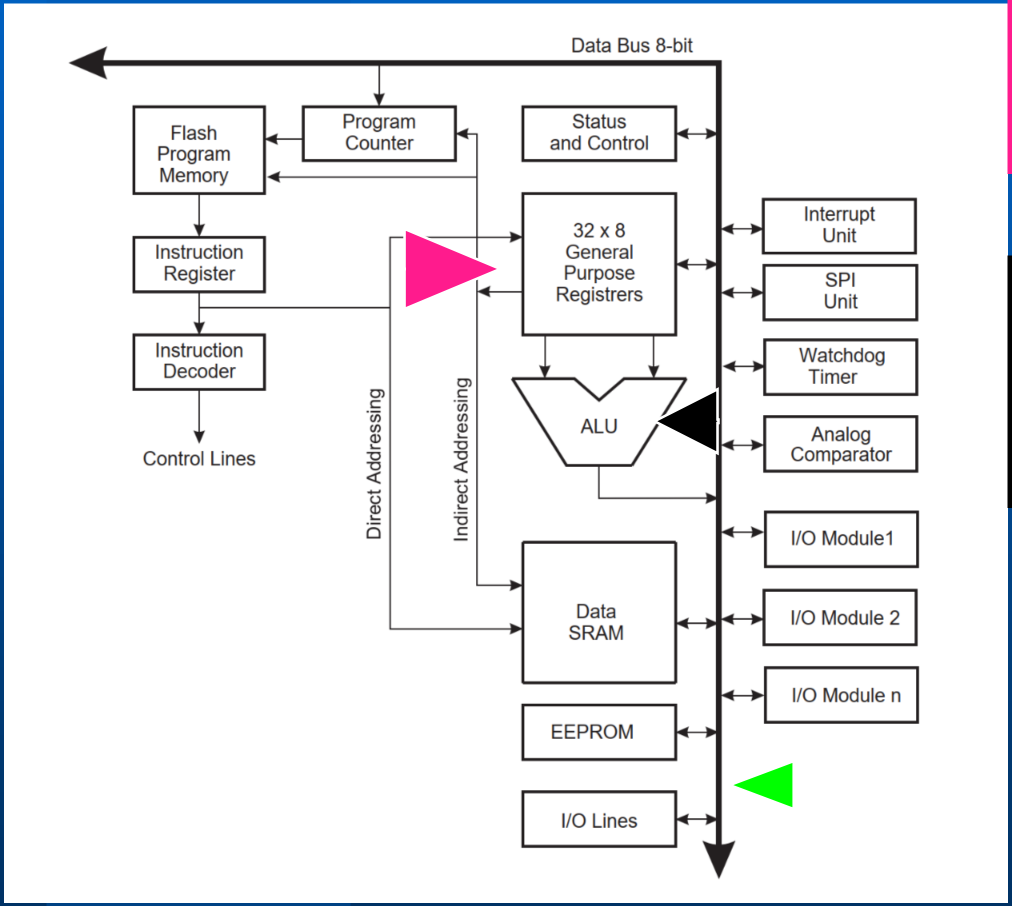


- **Your Program resides in this 128 Kbytes memory in .hex Intel format (hex numbers)**

The Program Counter Keeps track which instruction is to be executed next

The PC value at a subroutine call is stored in SRAM and after the subroutine execution (return) it is increased by 1 and loaded back.

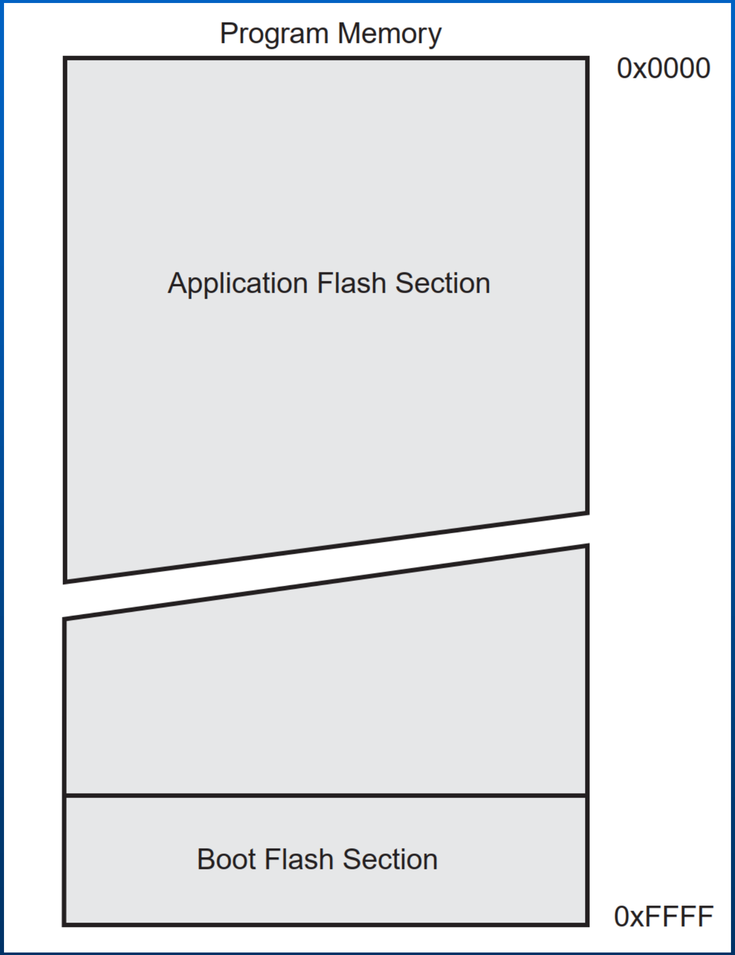
The ATmega1284P Architecture II



- 32 General Purpose Registers
- The Arithmetic Logic Unit operates on the 32 registers
- 8-bit Data Bus

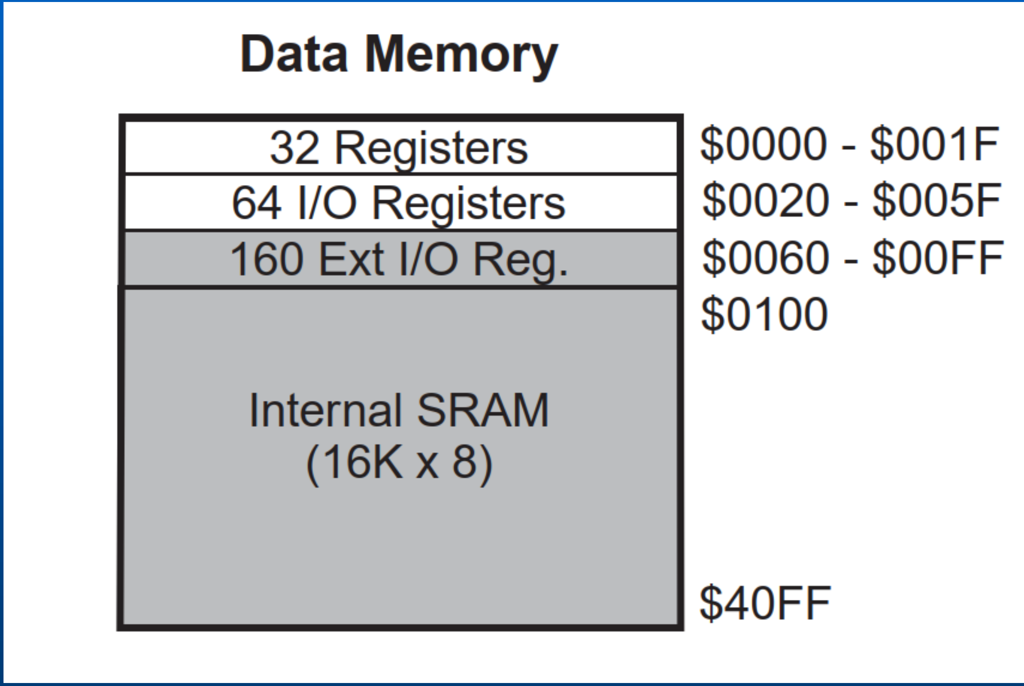
The ATmega1284P Program Memory

- This is the flash memory where the program is storred and can be written and erased reliable for at least 10000 times.
- The program memory is 128 Kbytes deep.
- The AVR instructions are 16 or 32 but wide and it is organized in 64 K locations each with 16 bits.
- The program counter has 16 bits.
- The user code (your programs) are written starting from the top of the program memory (0x0000)
- The bottom of the program memory contains the Boot Flash Section.



The ATmega1284P SRAM

- 16 Kbyte SRAM.
- The first 32 locations of the SRAM are reserved for the 32 Registers.
- The next 64 locations are the register used for I/O.
- The subsequent 160 locations can be accessed only by certain commands [ST, STS, STD, LD, LDS, LDD] .
- Will discuss later how does one access the SRAM……



AVR Assembly Language

Why Assembly ?

- ***Direct access to the architecture of the processor***
- ***Direct use of the machine registers memory and stack***
- ***Full control of the processor***
- ***Faster***

Registers AVR Assembly

- There are 32 registers R0-R31 on ATmega1284P. You may name them in a way that you can remember:

```
Example:  .def  VoltageRegister  =  r16
```

- You can zero (clear) them by:

```
Example:  clr  r16   ; This would load $00 on r16
```

Setting Registers I

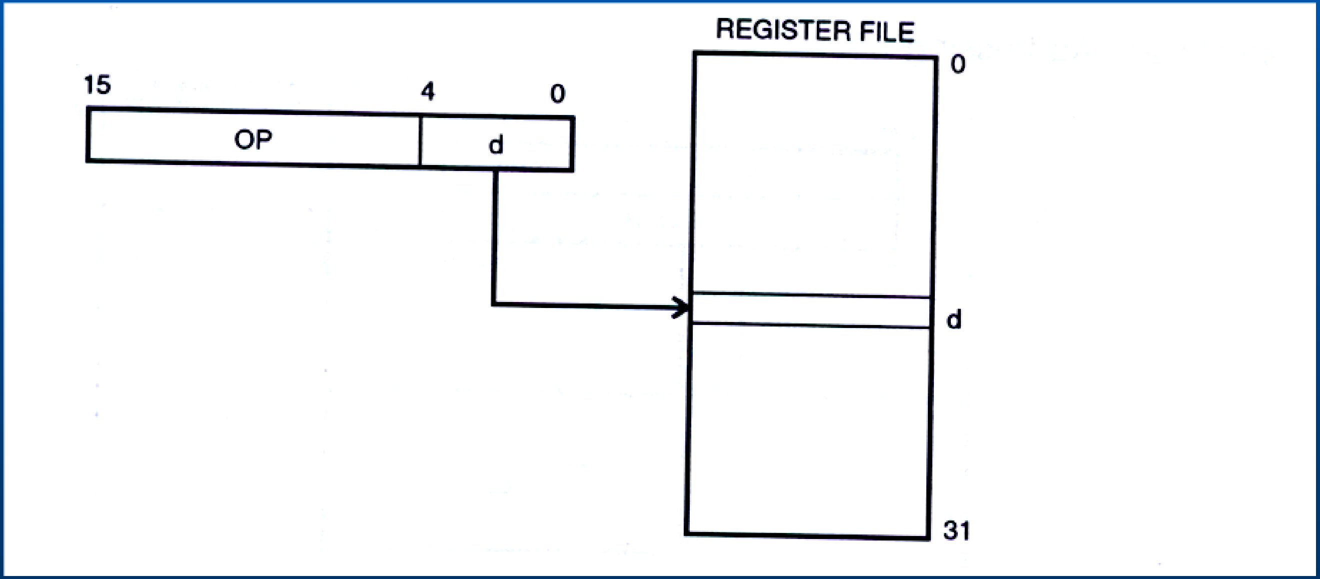
- You can set them to ones by:

Example: **ser r16 ; *This would load $FF on r16***

Remember the D-Flip-Flop ??

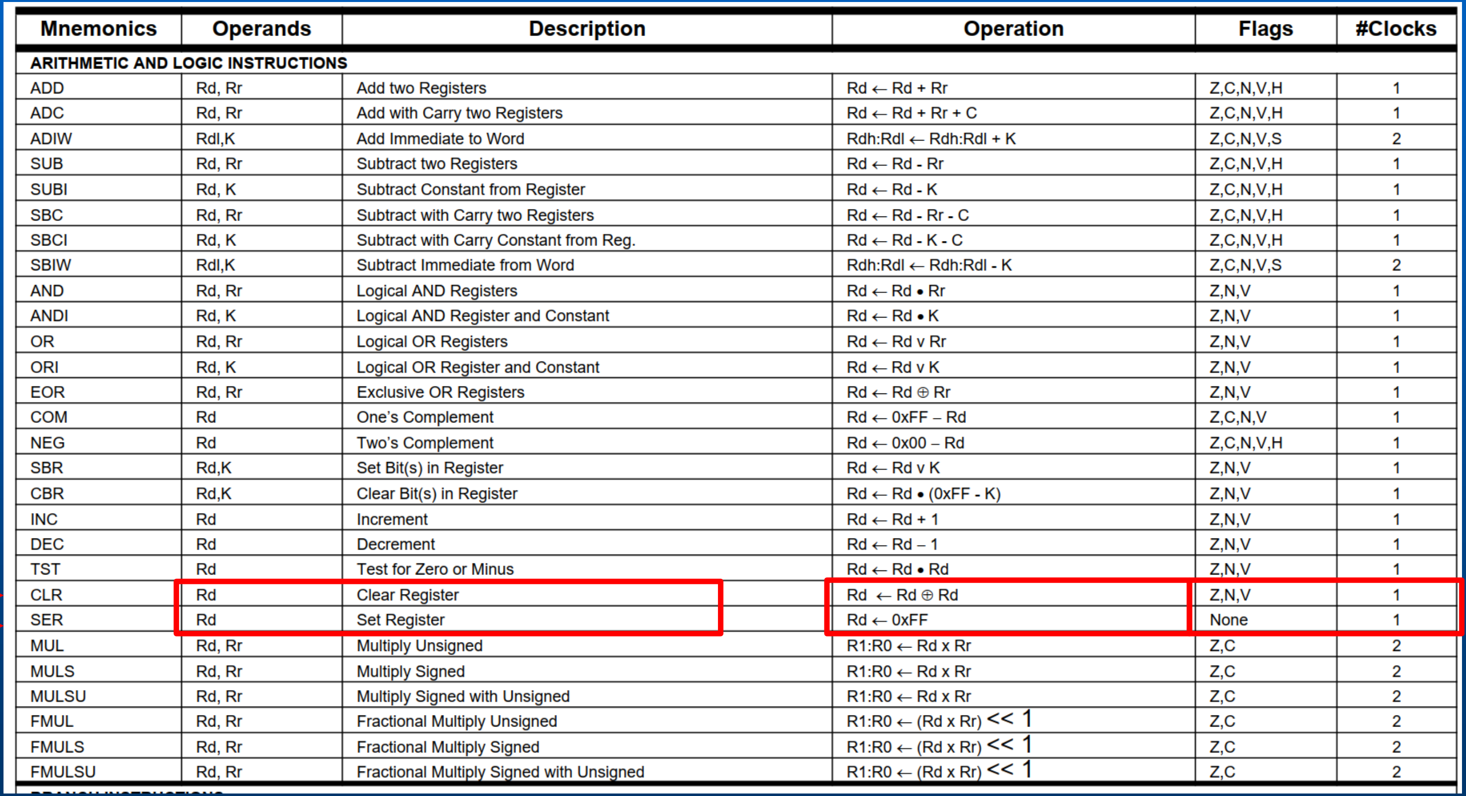
Setting Registers II

- **Both CLR and SER are *Direct Single Register Addressing Commands* because:**



Using the ATmega1284P Data Sheet Summary

Arithmetic and Logic Instructions



Mnemonics	Operands	Description	Operation	Flags	#Clocks
ARITHMETIC AND LOGIC INSTRUCTIONS					
ADD	Rd, Rr	Add two Registers	Rd ← Rd + Rr	Z,C,N,V,H	1
ADC	Rd, Rr	Add with Carry two Registers	Rd ← Rd + Rr + C	Z,C,N,V,H	1
ADIW	Rdl,K	Add Immediate to Word	Rdh:Rdl ← Rdh:Rdl + K	Z,C,N,V,S	2
SUB	Rd, Rr	Subtract two Registers	Rd ← Rd - Rr	Z,C,N,V,H	1
SUBI	Rd, K	Subtract Constant from Register	Rd ← Rd - K	Z,C,N,V,H	1
SBC	Rd, Rr	Subtract with Carry two Registers	Rd ← Rd - Rr - C	Z,C,N,V,H	1
SBCI	Rd, K	Subtract with Carry Constant from Reg.	Rd ← Rd - K - C	Z,C,N,V,H	1
SBIW	Rdl,K	Subtract Immediate from Word	Rdh:Rdl ← Rdh:Rdl - K	Z,C,N,V,S	2
AND	Rd, Rr	Logical AND Registers	Rd ← Rd • Rr	Z,N,V	1
ANDI	Rd, K	Logical AND Register and Constant	Rd ← Rd • K	Z,N,V	1
OR	Rd, Rr	Logical OR Registers	Rd ← Rd v Rr	Z,N,V	1
ORI	Rd, K	Logical OR Register and Constant	Rd ← Rd v K	Z,N,V	1
EOR	Rd, Rr	Exclusive OR Registers	Rd ← Rd ⊕ Rr	Z,N,V	1
COM	Rd	One's Complement	Rd ← 0xFF – Rd	Z,C,N,V	1
NEG	Rd	Two's Complement	Rd ← 0x00 – Rd	Z,C,N,V,H	1
SBR	Rd,K	Set Bit(s) in Register	Rd ← Rd v K	Z,N,V	1
CBR	Rd,K	Clear Bit(s) in Register	Rd ← Rd • (0xFF - K)	Z,N,V	1
INC	Rd	Increment	Rd ← Rd + 1	Z,N,V	1
DEC	Rd	Decrement	Rd ← Rd – 1	Z,N,V	1
TST	Rd	Test for Zero or Minus	Rd ← Rd • Rd	Z,N,V	1
CLR	Rd	Clear Register	Rd ← Rd ⊕ Rd	Z,N,V	1
SER	Rd	Set Register	Rd ← 0xFF	None	1
MUL	Rd, Rr	Multiply Unsigned	R1:R0 ← Rd x Rr	Z,C	2
MULS	Rd, Rr	Multiply Signed	R1:R0 ← Rd x Rr	Z,C	2
MULSU	Rd, Rr	Multiply Signed with Unsigned	R1:R0 ← Rd x Rr	Z,C	2
FMUL	Rd, Rr	Fractional Multiply Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULS	Rd, Rr	Fractional Multiply Signed	R1:R0 ← (Rd x Rr) << 1	Z,C	2
FMULSU	Rd, Rr	Fractional Multiply Signed with Unsigned	R1:R0 ← (Rd x Rr) << 1	Z,C	2

Loading a register

- **You can set the contents of r16 by:**

```
ldi   VoltageRegister,   $AA
```

The **ldi** command will load with the HEX value $AA to register **VoltageRegister** which is just **r16.**

Two register commands

- Introduce one more register r15:

```
.def RegisterTwo   =   r15
```

- The following command:

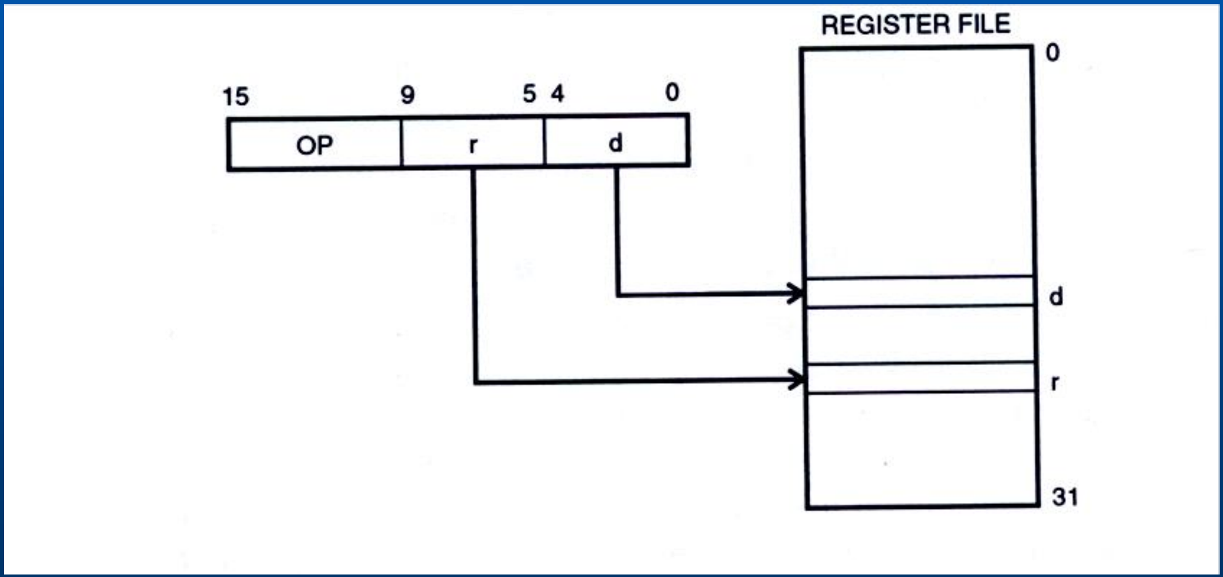
```
mov  RegisterTwo,  VoltageRegister
```

will transfer the contents of r16 ($AA) to r15

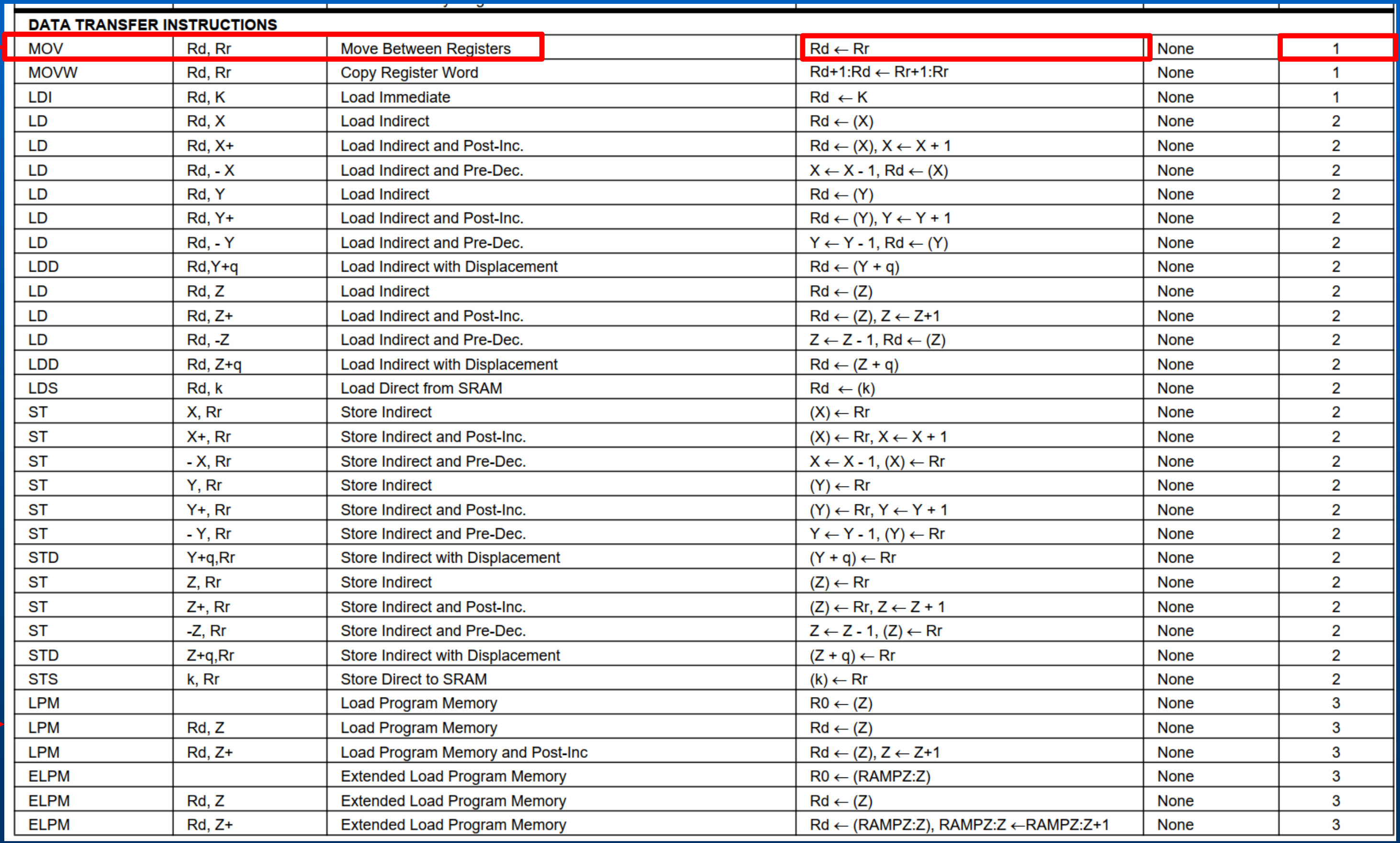
Direct two register

MOV Rd, Rr

Note: that the direction goes Against intuition:



Using the ATmega1284P Data Sheet Summary

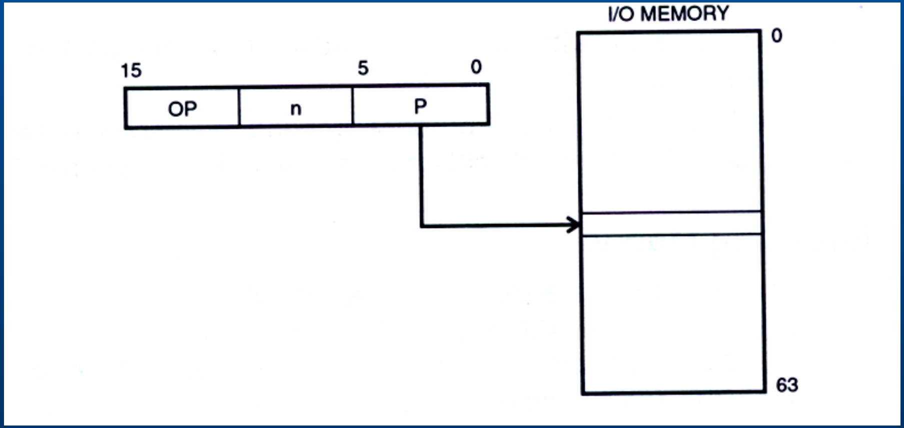


DATA TRANSFER INSTRUCTIONS					
MOV	Rd, Rr	Move Between Registers	Rd ← Rr	None	1
MOVW	Rd, Rr	Copy Register Word	Rd+1:Rd ← Rr+1:Rr	None	1
LDI	Rd, K	Load Immediate	Rd ← K	None	1
LD	Rd, X	Load Indirect	Rd ← (X)	None	2
LD	Rd, X+	Load Indirect and Post-Inc.	Rd ← (X), X ← X + 1	None	2
LD	Rd, - X	Load Indirect and Pre-Dec.	X ← X - 1, Rd ← (X)	None	2
LD	Rd, Y	Load Indirect	Rd ← (Y)	None	2
LD	Rd, Y+	Load Indirect and Post-Inc.	Rd ← (Y), Y ← Y + 1	None	2
LD	Rd, - Y	Load Indirect and Pre-Dec.	Y ← Y - 1, Rd ← (Y)	None	2
LDD	Rd,Y+q	Load Indirect with Displacement	Rd ← (Y + q)	None	2
LD	Rd, Z	Load Indirect	Rd ← (Z)	None	2
LD	Rd, Z+	Load Indirect and Post-Inc.	Rd ← (Z), Z ← Z+1	None	2
LD	Rd, -Z	Load Indirect and Pre-Dec.	Z ← Z - 1, Rd ← (Z)	None	2
LDD	Rd, Z+q	Load Indirect with Displacement	Rd ← (Z + q)	None	2
LDS	Rd, k	Load Direct from SRAM	Rd ← (k)	None	2
ST	X, Rr	Store Indirect	(X) ← Rr	None	2
ST	X+, Rr	Store Indirect and Post-Inc.	(X) ← Rr, X ← X + 1	None	2
ST	- X, Rr	Store Indirect and Pre-Dec.	X ← X - 1, (X) ← Rr	None	2
ST	Y, Rr	Store Indirect	(Y) ← Rr	None	2
ST	Y+, Rr	Store Indirect and Post-Inc.	(Y) ← Rr, Y ← Y + 1	None	2
ST	- Y, Rr	Store Indirect and Pre-Dec.	Y ← Y - 1, (Y) ← Rr	None	2
STD	Y+q,Rr	Store Indirect with Displacement	(Y + q) ← Rr	None	2
ST	Z, Rr	Store Indirect	(Z) ← Rr	None	2
ST	Z+, Rr	Store Indirect and Post-Inc.	(Z) ← Rr, Z ← Z + 1	None	2
ST	-Z, Rr	Store Indirect and Pre-Dec.	Z ← Z - 1, (Z) ← Rr	None	2
STD	Z+q,Rr	Store Indirect with Displacement	(Z + q) ← Rr	None	2
STS	k, Rr	Store Direct to SRAM	(k) ← Rr	None	2
LPM		Load Program Memory	R0 ← (Z)	None	3
LPM	Rd, Z	Load Program Memory	Rd ← (Z)	None	3
LPM	Rd, Z+	Load Program Memory and Post-Inc	Rd ← (Z), Z ← Z+1	None	3
ELPM		Extended Load Program Memory	R0 ← (RAMPZ:Z)	None	3
ELPM	Rd, Z	Extended Load Program Memory	Rd ← (Z)	None	3
ELPM	Rd, Z+	Extended Load Program Memory	Rd ← (RAMPZ:Z), RAMPZ:Z ←RAMPZ:Z+1	None	3

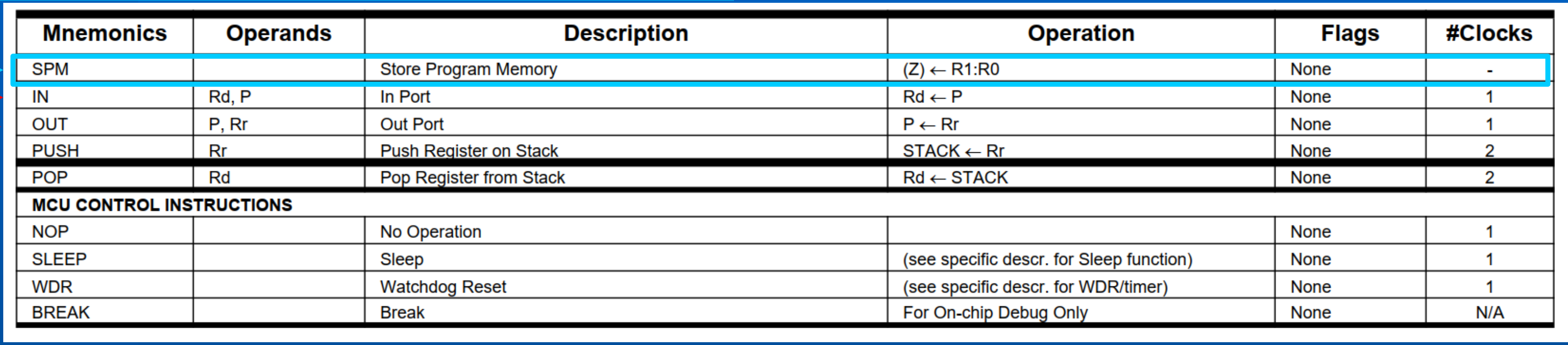
I/O Direct Addressing

To read or write to the ATmega128P ports use the commands:

```
IN    Rd,   PINX
OUT  PORTX,  Rr;  X is A-D
```



I/O and other Commands



Mnemonics	Operands	Description	Operation	Flags	#Clocks
SPM		Store Program Memory	(Z) ← R1:R0	None	-
IN	Rd, P	In Port	Rd ← P	None	1
OUT	P, Rr	Out Port	P ← Rr	None	1
PUSH	Rr	Push Register on Stack	STACK ← Rr	None	2
POP	Rd	Pop Register from Stack	Rd ← STACK	None	2
MCU CONTROL INSTRUCTIONS					
NOP		No Operation		None	1
SLEEP		Sleep	(see specific descr. for Sleep function)	None	1
WDR		Watchdog Reset	(see specific descr. for WDR/timer)	None	1
BREAK		Break	For On-chip Debug Only	None	N/A

- **SPM: It allows one to write in the program memory – Dangerous but useful**
- **IN: transfers data from the PORT PINS to Registers**
- **OUT: transfers data from a register to PORT PINS**