AVR Architecture: RISC vs. CISC

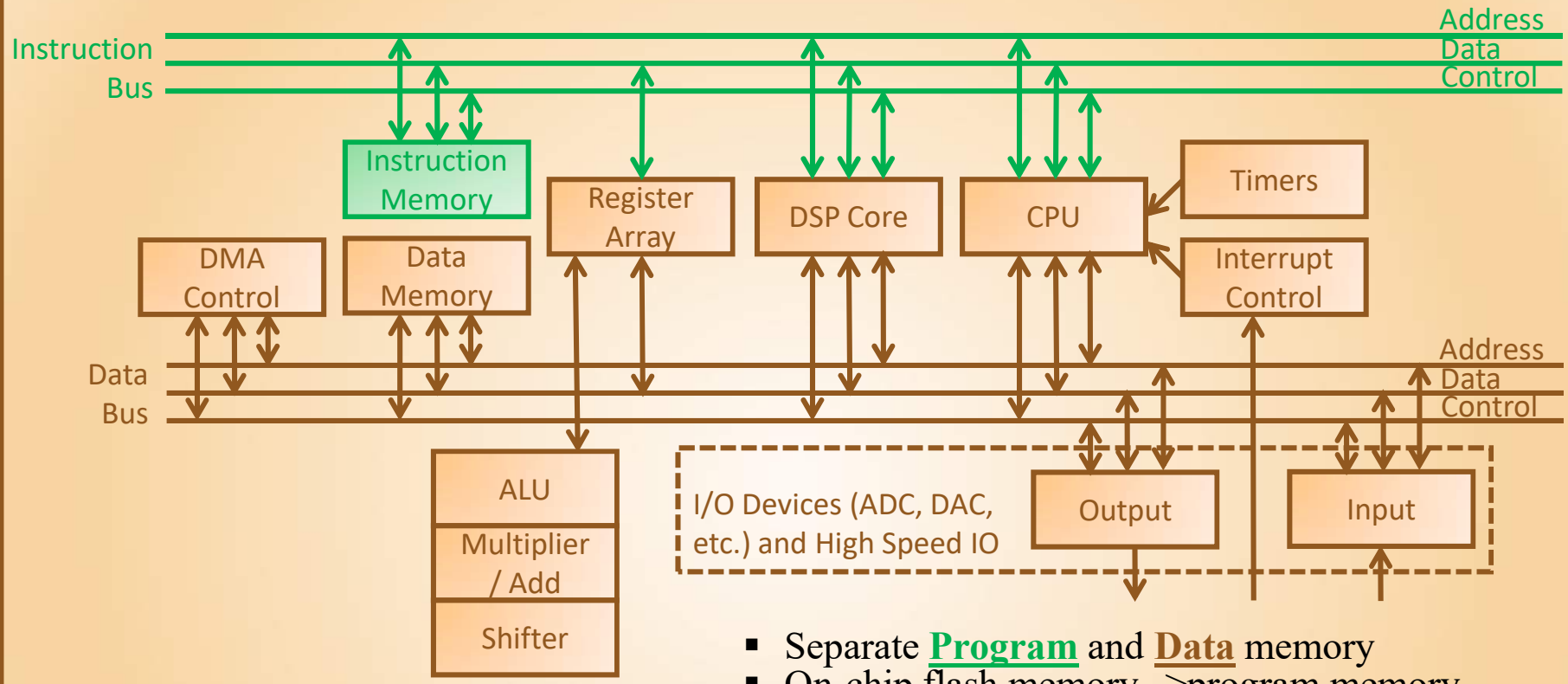
CISC

- Richer instruction set, some simple, some very complex
- Instructions generally take more than 1 clock to execute
- Instructions of variable size
- Instructions interface with memory in multiple mechanisms with complex addressing modes
- No pipelining
- Upward compatibility within a family
- Microcode control
- Work well with simpler compiler

RISC

- Simple, primitive instructions and addressing modes.
- Instructions typically execute in one clock cycle.
- Uniformed length instructions and fixed instruction format.
- Instructions interface with memory via fixed mechanisms.
- Pipelining
- Instruction set is orthogonal (little overlapping of instruction functionality.
- Hardwired control
- Complexity pushed to the compiler.

AVR Architecture (fig 1.7): Harvard vs Von-Neumann

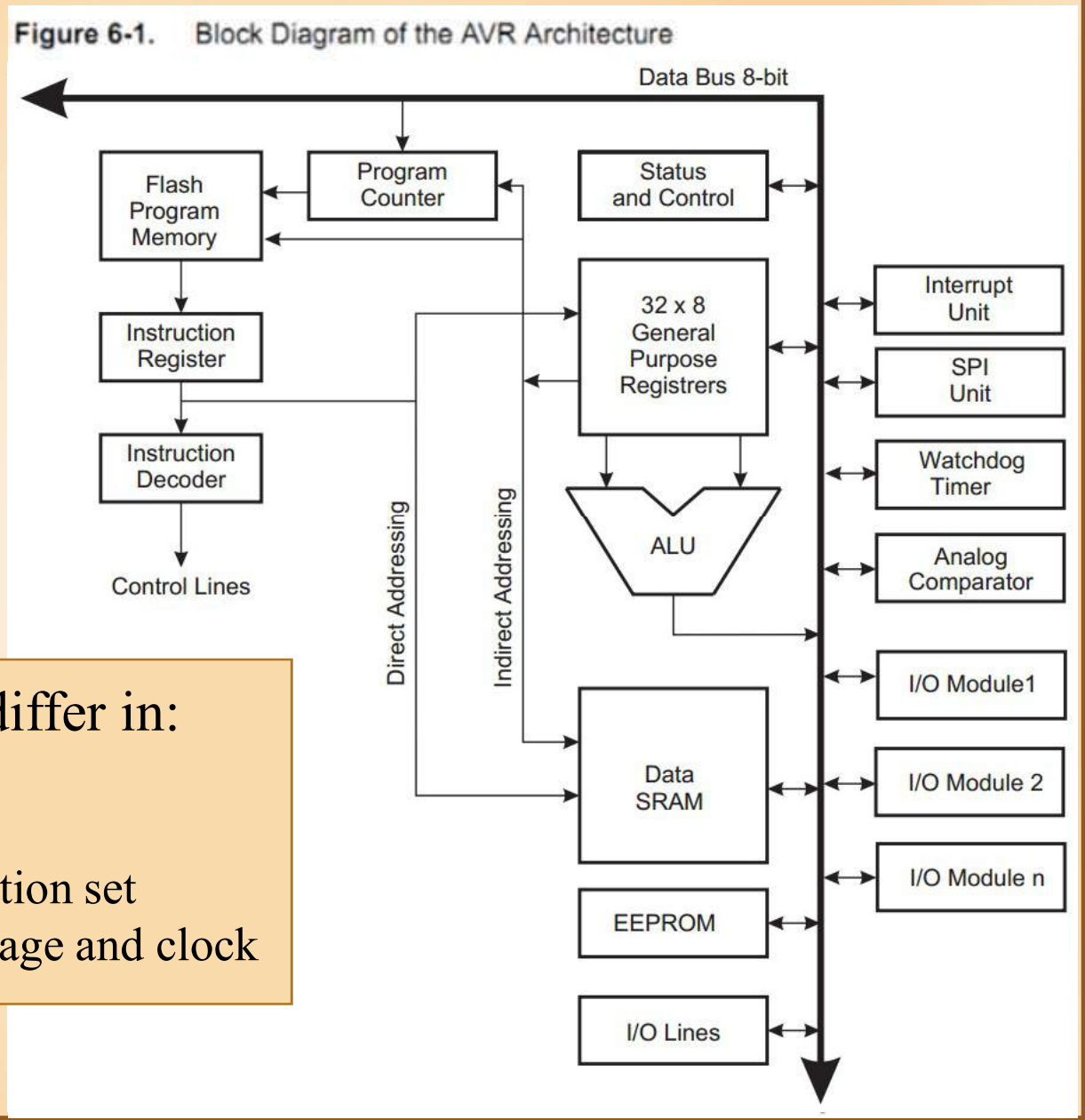


- Separate **Program** and **Data** memory
- On-chip flash memory-->program memory
- On-chip data memory (RAM and EEPROM)
- 32 x 8 General purpose registers
- On-chip programmable timers
- Internal and external interrupt sources
- Programmable watchdog timer
- On-chip RC clock oscillator (more on clock later)
- Variety of I/O, Programmable I/O lines

AVR Architecture: - summary

- RISC vs. CISC
- Harvard vs. Von Neumann
- Separate Program and Data memory
- On-chip flash memory-->program memory
- On-chip data memory (RAM and EEPROM)
- 32 x 8 General purpose registers
- On-chip programmable timers
- Internal and external interrupt sources
- Programmable watchdog timer
- On-chip RC clock oscillator (more on clock later)
- Variety of I/O, Programmable I/O lines

Architecture - General 8Bit AVR Architecture



- Different Models differ in:
  - Memory size
  - I/Os
  - Slightly in instruction set
  - Power usage, voltage and clock

Architecture - Pipeline

- AVR has a 2-stage pipeline (Instruction Fetch [IF] and Execute [EX]) – Harvard architecture…
  - Decode happens in IF cycle.
- Many Instructions execute in a single clock cycle, and a few take 2 or more clock cycles.

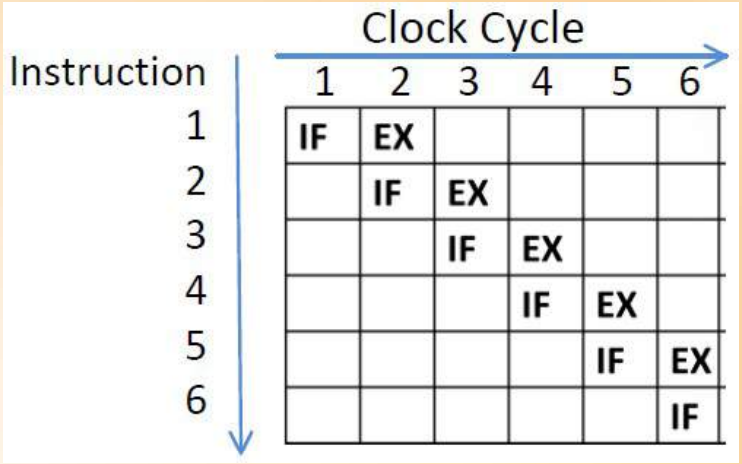
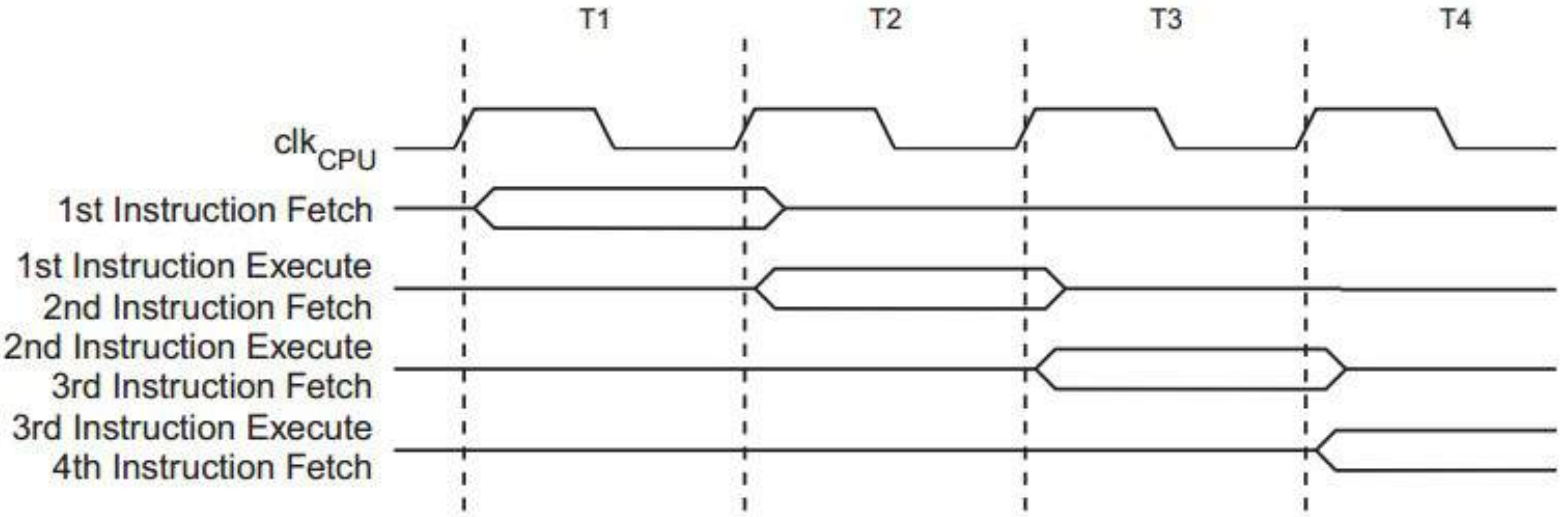


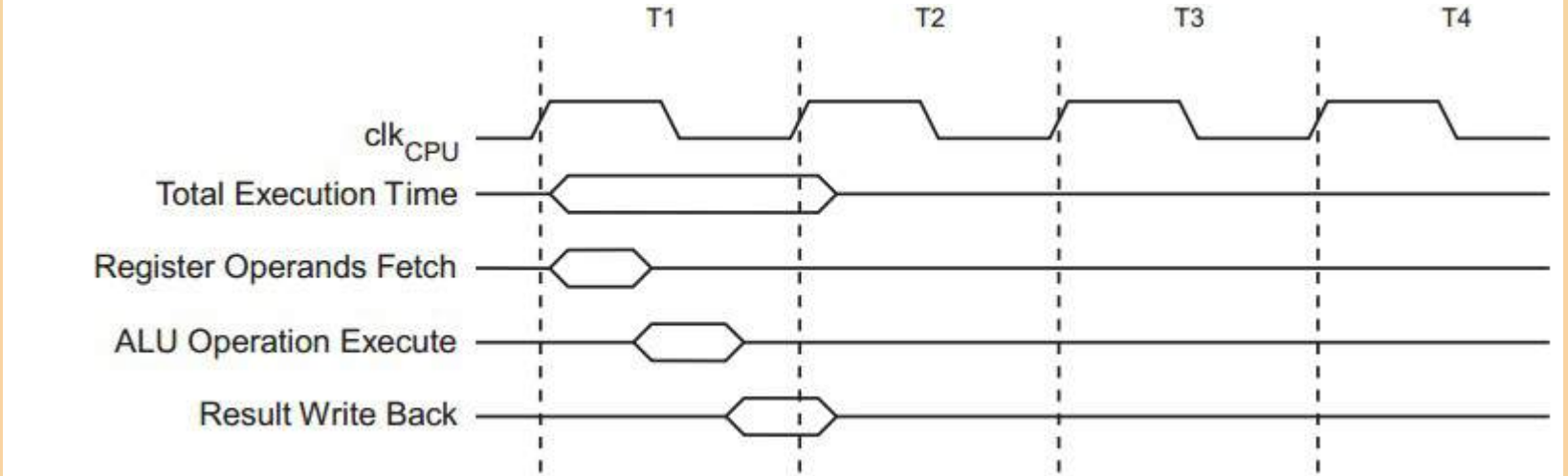
Figure 6-2. The Parallel Instruction Fetches and Instruction Executions



Architecture - Execution Timing:

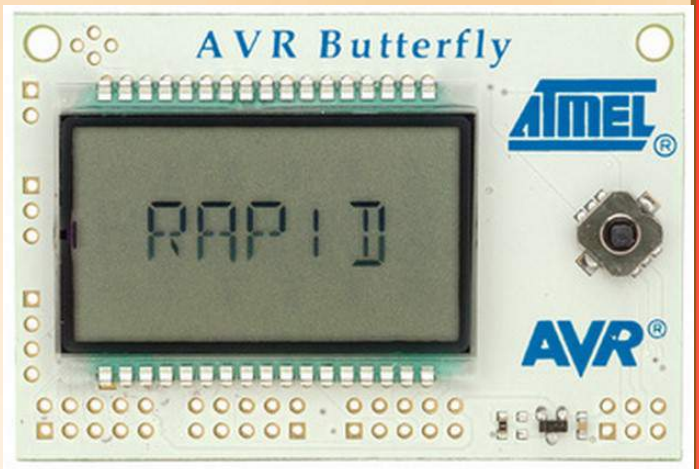
- Note that operations take less than a full clock cycle – a PLL that provides a phase shift and/or accelerated internal clock may facilitate this.

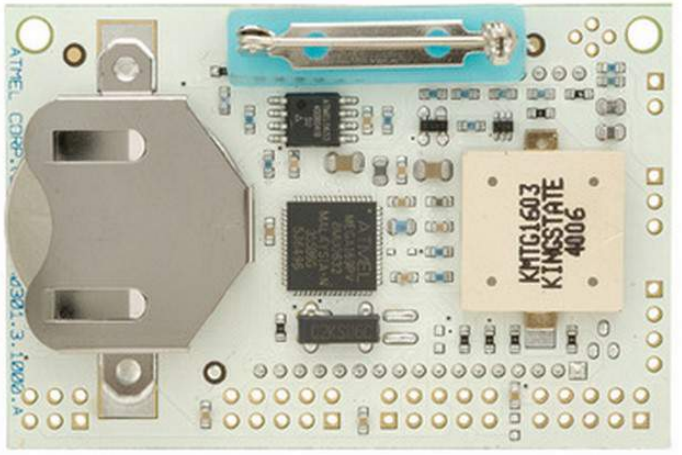
Figure 6-3. Single Cycle ALU Operation



Architecture - ATmega169P (AVR Butterfly):

- ATmega169P is our case study
- Full datasheet can be found at: http://www.atmel.com/dyn/resources/prod_documents/doc8018.pdf
- From megaAVRfamily: (VS tinyAVR, XMEGA, 32bit AVR)
- "P"at 169P means "Low Power"
- "16"in 169P --> 16K program memory
- "9"shows design revision

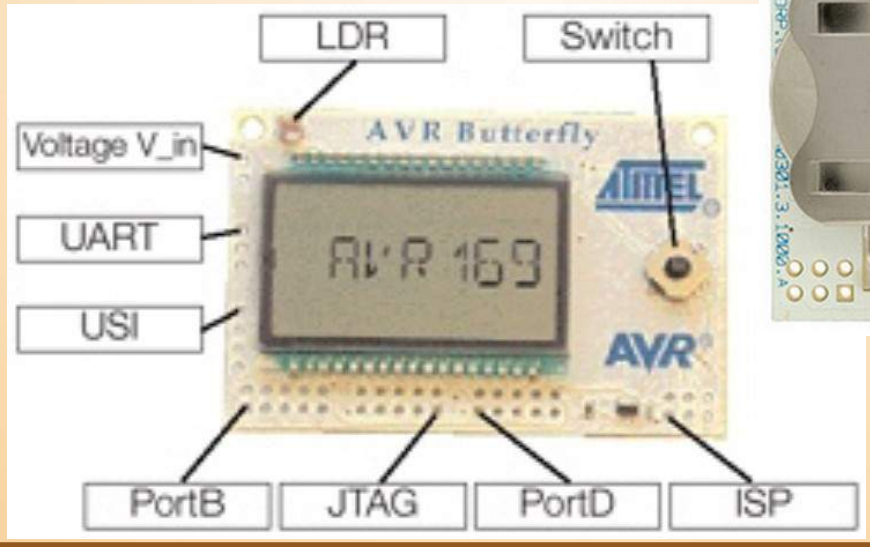




ATMEL

8-bit AVR Microcontroller with 16K Bytes In-System Programmable Flash

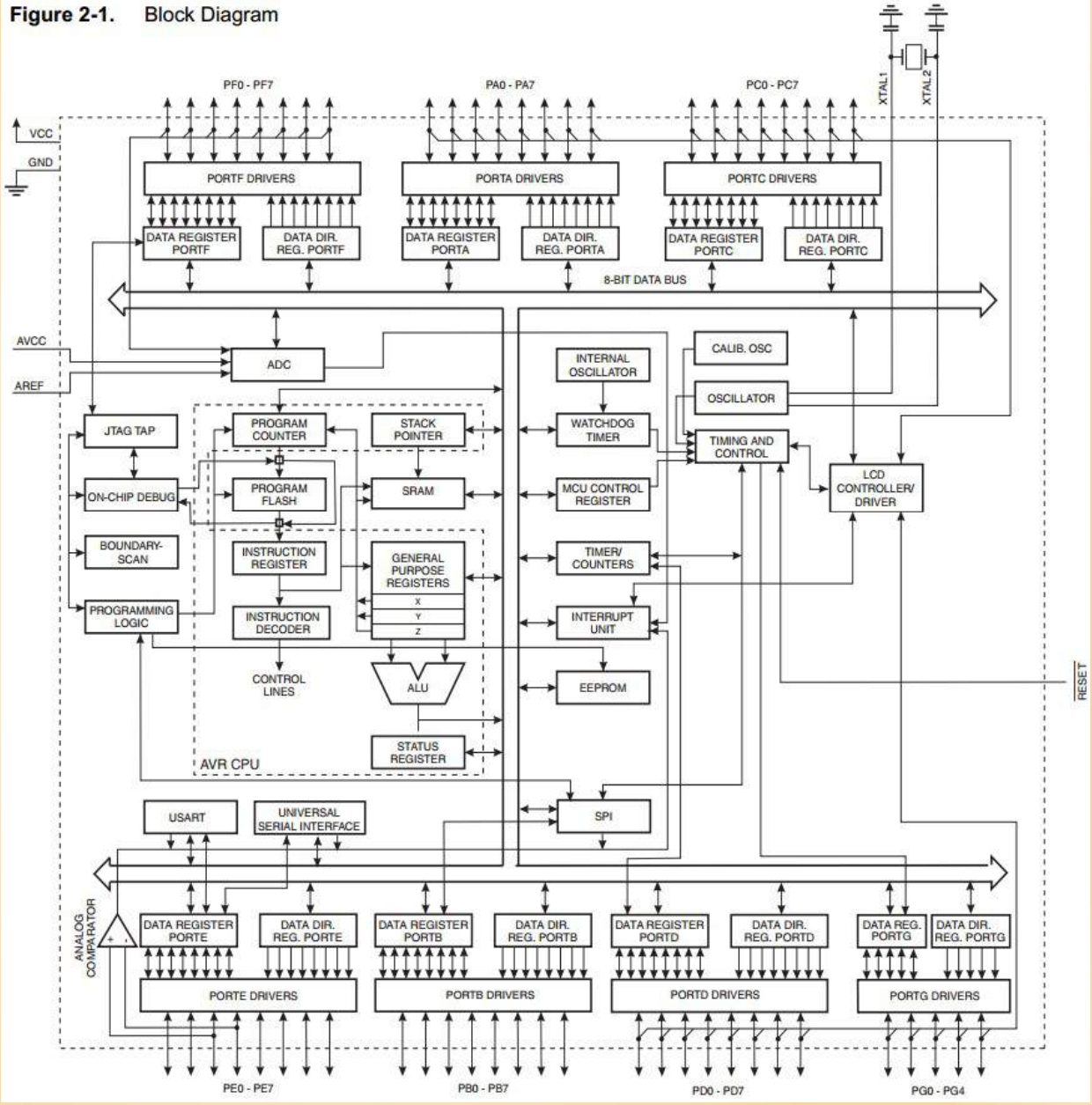
ATmega169P



Architecture - ATmega169P Architecture:

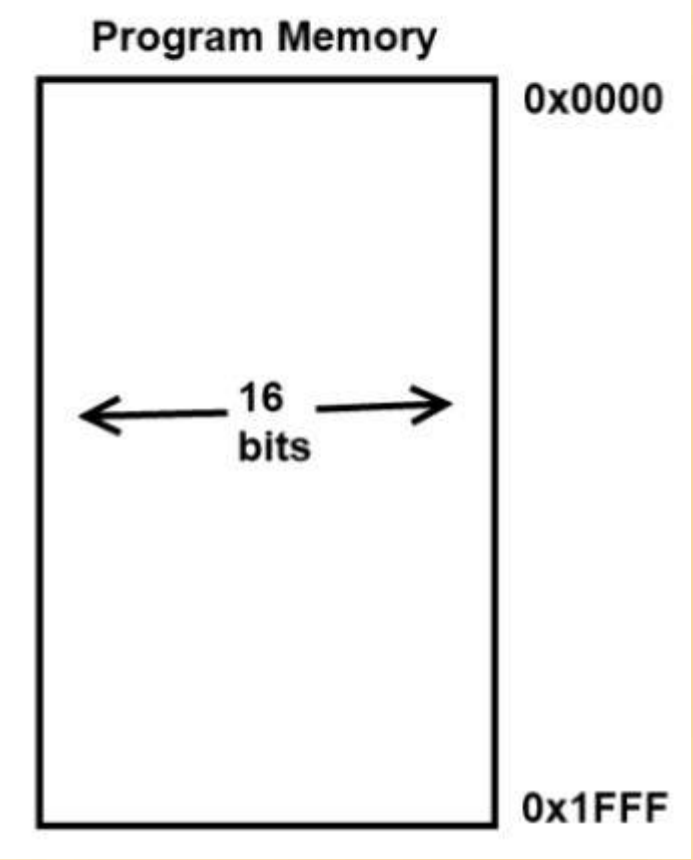
Source: Data Sheet

Figure 2-1. Block Diagram



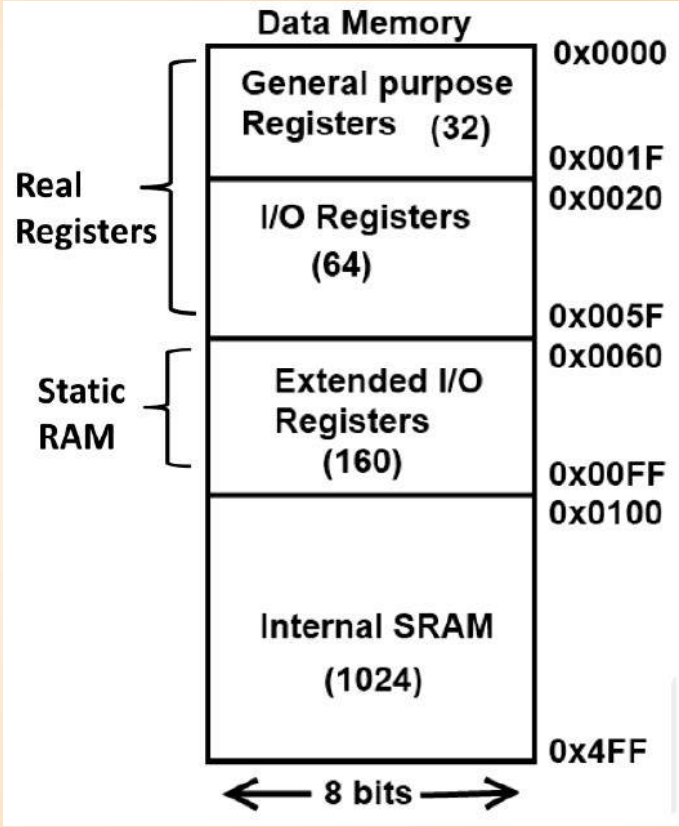
Architecture - Program (Instruction) Memory Map:

- The ATmega169P contains 16 Kbytes Flash memory for program storage.
- All AVR instructions are 16 or 32 bits wide, Therefore the Flash is organized as 8K × 16.
- 8K = 2^{13} --> 13 bits are required for addressing the program memory
- Program Counter (PC) = 13 bits
- *How many hex bits are required to represent 0x1FFF?*



Architecture - Data Memory Map (DMP):

- The entire Data Memory Space shown here can be accessed as memory, but using register and I/O register related instructions is faster on the real registers..
- General Purpose Registers :
  - 32 General Registers
  - Labeled R0 –R31
  - Called Register File
- Example: ADD R2, R3
- Make sure to look at "Section 30. Register Summary" of Data sheet

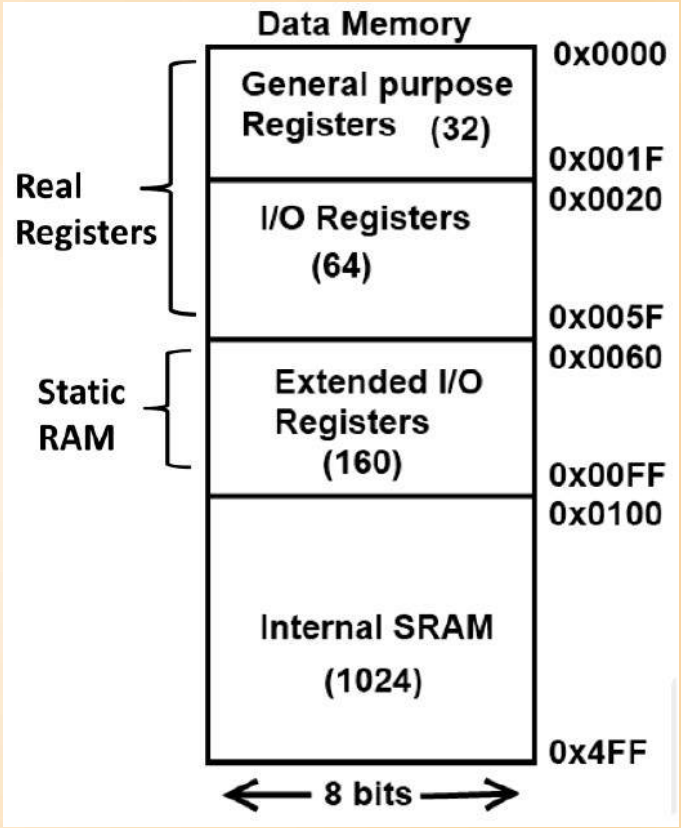


Architecture - DMP – I/O Registers (cont):

- All I/O locations may be accessed by memory instructions **LD/LDS/LDD**and **ST/STS/STD**instructions, transferring data between the 32 general purpose working registers and the I/O space.

- **64 I/O Registers**:
  - Different Memory address than I/O address (e.g. I/O Register zero(0x0) is data memory address (0x20))
  - 0x00 (0x20)–0x1F(0x3F) are bit-accessible using "SBI" and "CBI" instructions.
  - 0x00(0x20)-0x3F(0x5F) can be accessed using "IN" and "OUT" instructions.
  - When addressing I/O Registers as data space using LD and ST instructions, Data memory address must be used (e.g. 0x20 instead of 0x0)

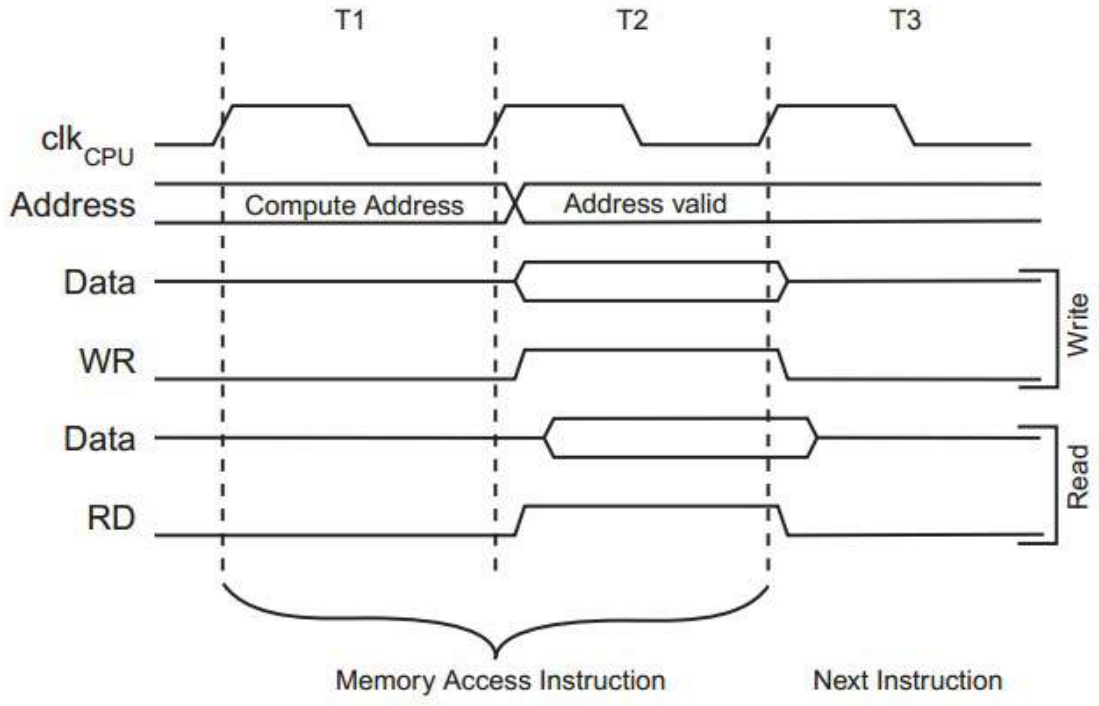
- **160 Extended I/O Registers**
  - 0x60 -0xFF in SRAM
  - Only ST/STS/STD and LD/LDS/LDD instructions can be used.



Architecture - SRAM Access Time:

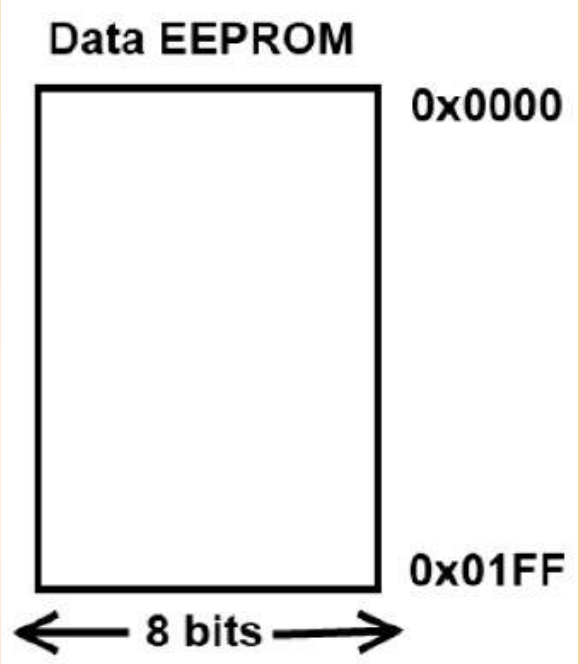
- 2 cycles
- Address pointers (Registers X, Y, Z) are used for addressing
- At first cycle, register file is accessed. At the end of the first cycle, the ALU performs address calculation.
- At second cycle, calculated address is used to access the SRAM location (to read or write)

Figure 7-3. On-chip Data SRAM Access Cycles



Architecture - EEPROM Space:

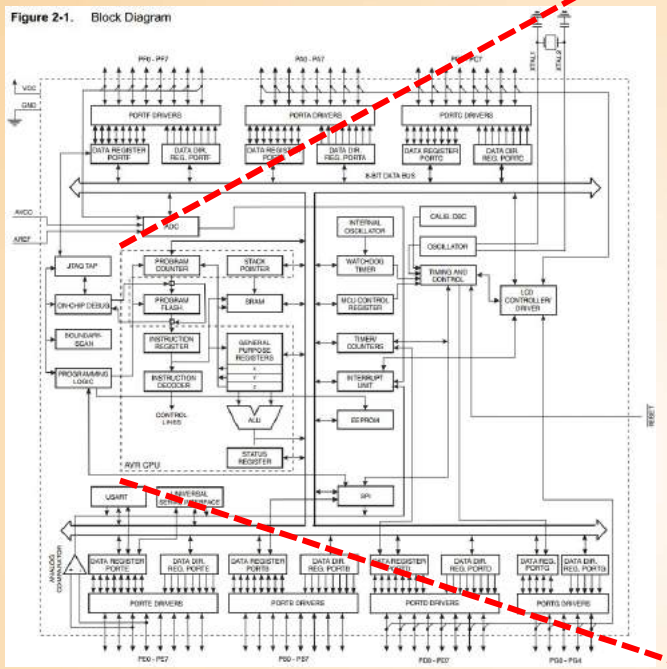
- 100,000 write/erase cycles
- Read operation halts the CPU for four clock cycles. (after 4 cycles next instruction is executed)
- Write operation halts the CPU for two clock cycles.

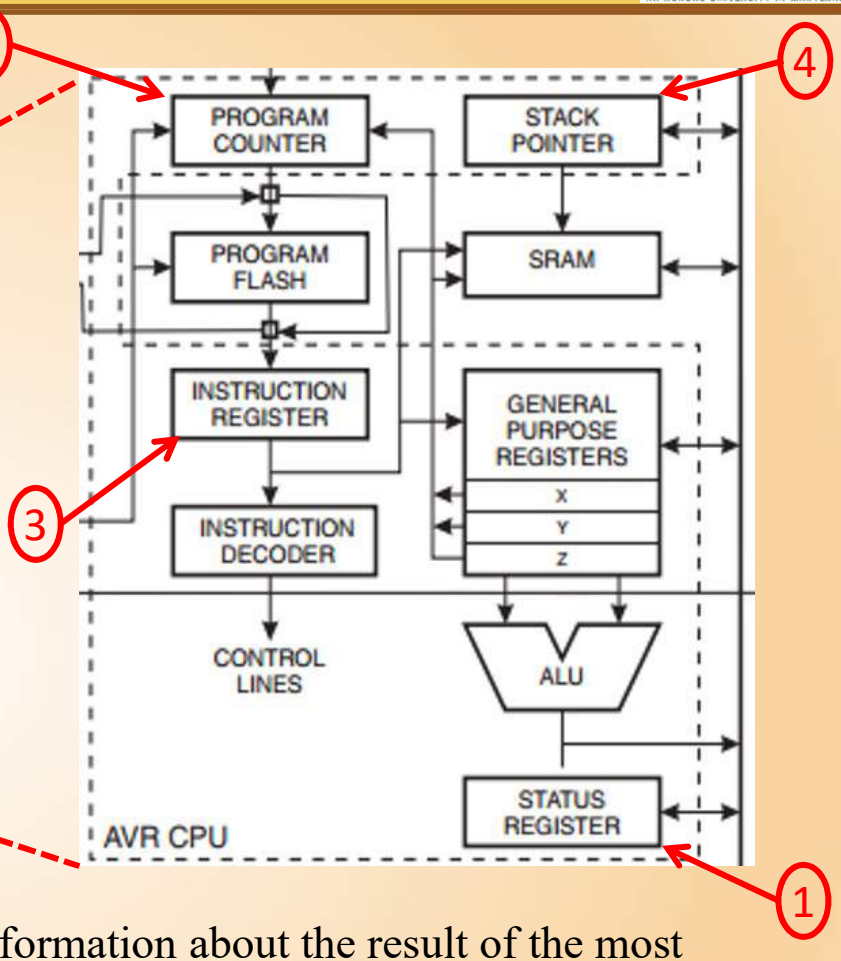


- Can be accessed through EEPROM register manipulations (I/O registers)
  - EEPROM Address Register (0x41) & (0x42)
  - EEPROM Data Register (0x40)
  - EEPROM Control Register (0x3F)

- (EEPROM Details will be covered later)

AVR CPU Registers – CPU Registers:

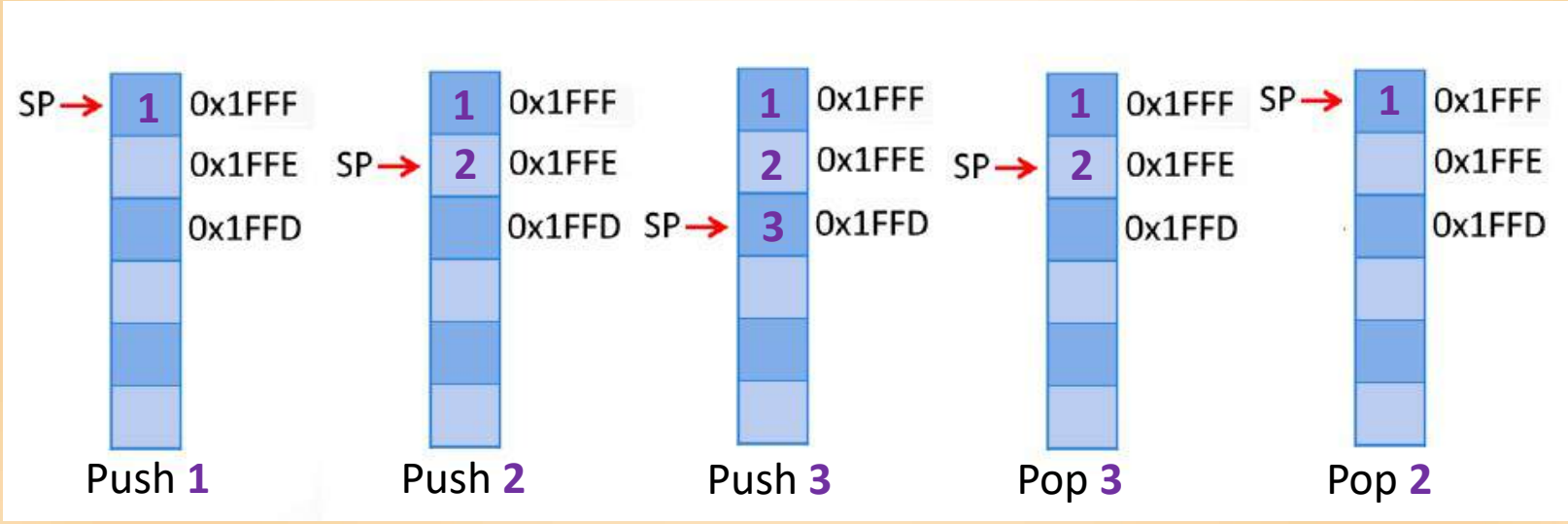




- **General Purpose Registers:**
  1) **Status Register (SREG)** contains information about the result of the most recently executed instruction
  2) **Program Counter (PC)**: Holds the address of the next instruction to be executed.
     - 13 bit to address 8K x 16bit width memory (16Kbytes)
  3) **Instruction Register (IR)**: Holds the fetched instruction from program memory.
     - What is the size of IR in bits?
  4) **Stack pointer (SP)**: points to the top of stack

AVR CPU Registers– Stack:

- The Stack is mainly used for storing temporary data, for ***storing local variables*** and for ***storing return addresses after interrupts and subroutine calls***.
- Is implemented as growing from higher to lower memory locations.
- Initial Stack Pointer is set equal to the last address of the internal SRAM
- **PUSH** command will decrease the SP.
- **POP** command will increase the SP.
- ***Table 6-1 from data sheet has all assembly Stack Pointer instructions***.



AVR CPU Registers– Stack Pointer (SP):

- Stack Pointer (SP): points to the top of stack
  - Is implemented as two 8-bit registers in the I/O space.

- What does address stored in the SP tell us?

6.4.1 SPH and SPL – Stack Pointer

Bit	15	14	13	12	11	10	9	8	
0x3E (0x5E)	–	–	–	–	–	SP10	SP9	SP8	SPH
0x3D (0x5D)	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
	7	6	5	4	3	2	1	0	
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
	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	
	0	0	0	0	0	0	0	0	

Table 6-1. Stack Pointer instructions

Instruction	Stack pointer	Description
PUSH	Decremented by 1	Data is pushed onto the stack
CALL ICALL RCALL	Decremented by 2	Return address is pushed onto the stack with a subroutine call or interrupt
POP	Incremented by 1	Data is popped from the stack
RET RETI	Incremented by 2	Return address is popped from the stack with return from subroutine or return from interrupt

AVR CPU Registers– Status Register (SREG):

- The Status Register contains information about the result of the most recently executed instruction.
- This information can be used for altering program flow in order to perform conditional operations
- The Status Register is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.

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	

AVR CPU Registers– SREG (continued):

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	

Bit7 : I : Global Interrupt Enable. Setting this bit enables all the interrupts. Resetting this disables all interrupts.

Bit6 : T : Bit Copy Storage. Used with BLD (bit load) and BST (bit store) ISTR for loading and storing bits from one register to another.

Bit5 : H : Half Carry Flag. Indicates half carry in some arithmetic ISTRs.

Bit4 : S : Sign Flag. This bit is the exclusive OR between the negative flag N and the Overflow flag V.

Bit3 : V : Overflow Flag (Two's Complement).

Bit2 : N : Negative Flag.

Bit1 : Z : Zero Flag. Indicates a zero result after an arithmetic or logical operation.

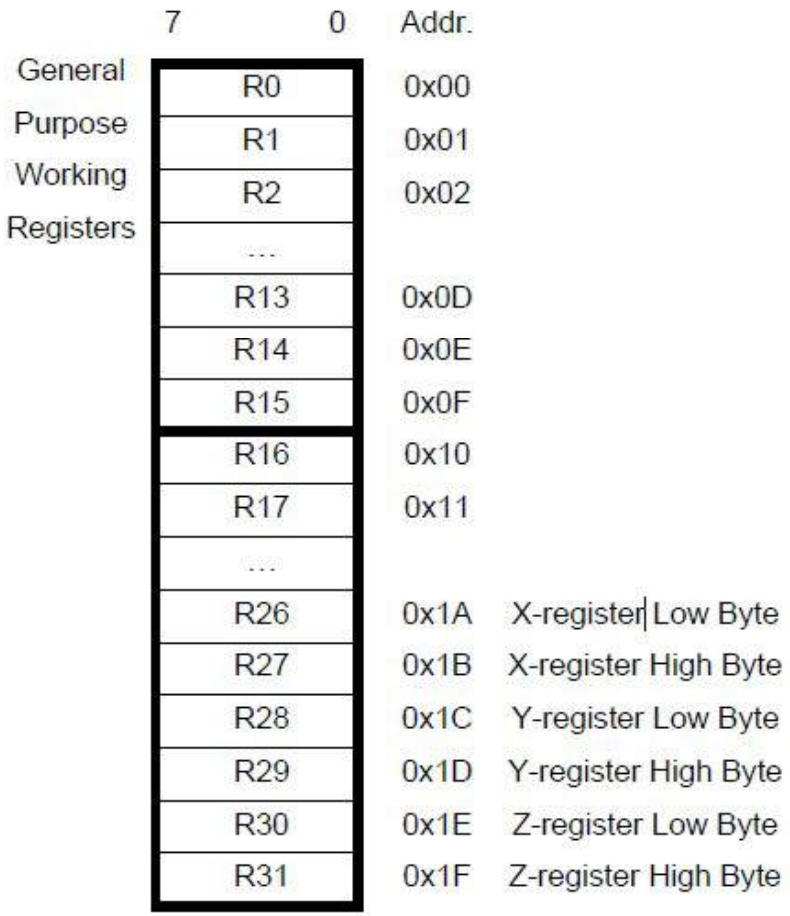
Bit0 : C : Carry Flag. Indicates a carry in arithmetic or logical operation.

AVR CPU Registers – Register File:

32 x 8 bit registers:

- Many of the instructions operating on the Register File have direct access to all registers, like **LD** (load indirect from data space) and **MUL** (multiply)
- Some only operate on **R16..R31,** like **LDI** (load data immediate), **SBR/CBR** (set/clear bits in register), ANDI, CPI, SUBI, SUBIC,MULS
- A few only operate on **R16..R23** like special multiply operations **MULSU, FMUL, FMULS, FMULSU**
- A few double-word operations (but not all) only operate on the upper four pairs of registers **R25:R24, R27:R26, R29:R28, R31:R30** like **ADIW, SBIW** (Add/Subtract Immediate from Word)
- Most single-byte, register-only or register + immediate operations are single cycle instructions. **MUL** is one exception.

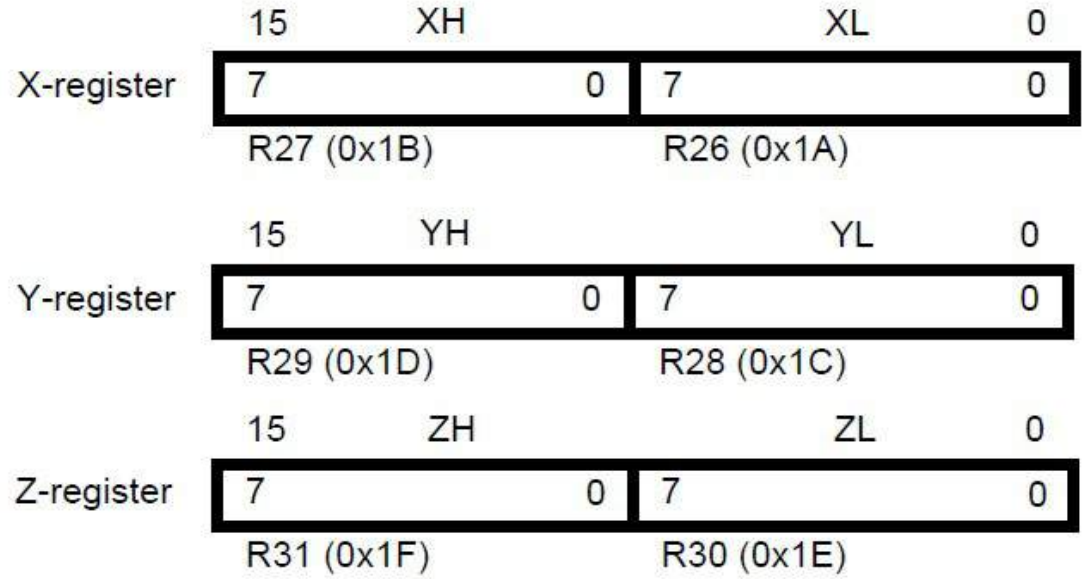
Figure 6-4. AVR CPU General Purpose Working Registers



Special Purpose Registers -X, Y , Z :

- The registers **R26..R31** have a special purpose **in addition to their general purpose usage**.
- These registers are 16-bit **address pointers** for indirect addressing of the data space.
- In different addressing modes these address registers have functions as **fixed displacement**, **automatic increment**, and **automatic decrement**. (more on addressing later!)

Figure 6-5. The X-, Y-, and Z-registers



Materials to Review for Quiz 1:

class: 1, slide: 12 → Embedded System
class: 1, slide: 15 → 3 Levels of Programming
class: 1, slide: 21 → Examples of Embedded Systems
class: 2, slide: 9 → Convert Dec to Hex to Bin to twos compliment
class: 2, slide: 6 → Harvard vs VonNeuman (and class 4 slide 2)
class: 2, slide: 16-17 → Instruction Types; Addressing Mode, Data Transfer
class: 2, slide: 18 → Execution Flow and Flow Instructions
class: 2, slide: 26 → Instruction Cycle
class: 3, slide: 2 → Microcontroller Elements
class: 3, slide: 6 → What is an ARM
class: 3, slide: 10 → When to use or not use an MCU
class: 4, slide: 1 → Diff between CISC and RISC
class: 4, slide: 9-11 → AVR Instruction and Data Memory
class: 4, slide: 17-18 → Status Reg Bits