CO321: EMBEDDED SYSTEMS

Introduction to AVR
Mostly from Chapter 1

The AVR microcontroller and embedded systems

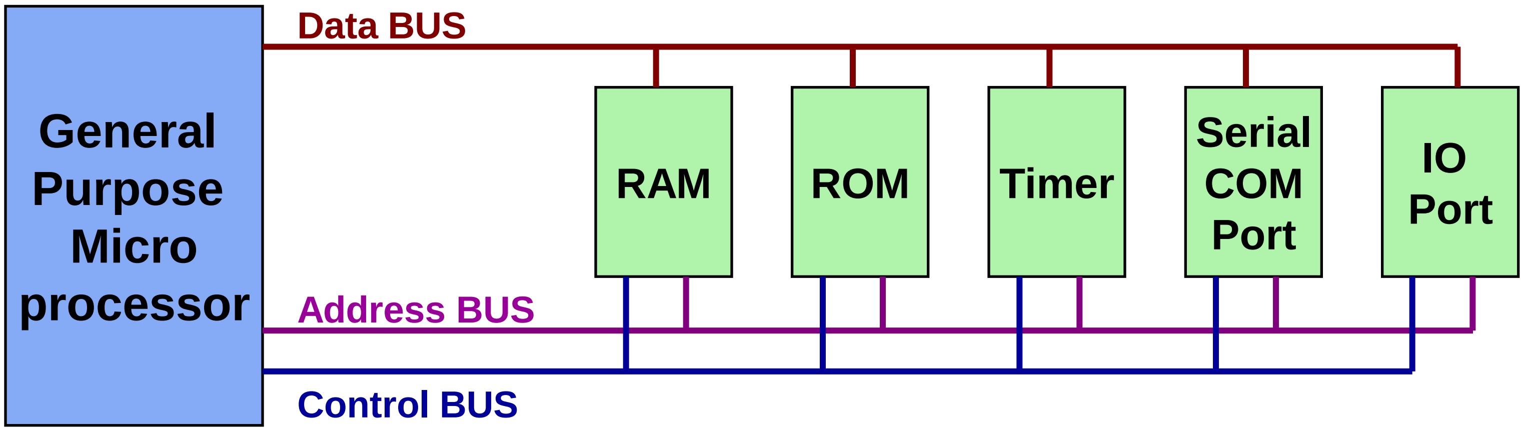
using assembly and c

Topics

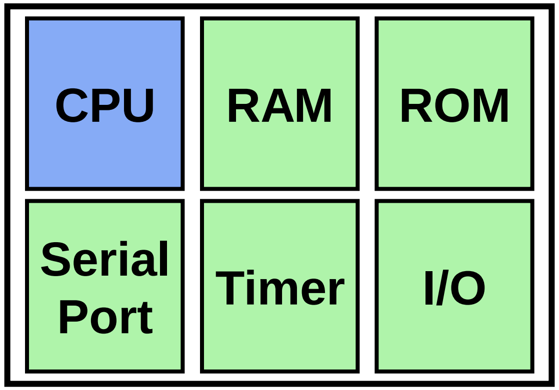
- Microcontrollers vs. Microprocessors
- Most common microcontrollers
- AVR Features
- AVR members

General Purpose Microprocessors vs. Microcontrollers

- General Purpose Microprocessors



- Microcontrollers



Most common microcontrollers

- 8-bit microcontrollers
  - AVR
  - PIC
  - HCS12
  - 8051
- 32-bit microcontrollers
  - ARM
  - PIC32

AVR History

- The **Atmel AVR**™ is a family of 8-bit RISC microcontrollers produced by Atmel.
- The AVR architecture was conceived by two students at the **Norwegian Institute of Technology (NTH)** and further refined and developed at **Atmel Norway**, the Atmel daughter company founded by the two chip architects.

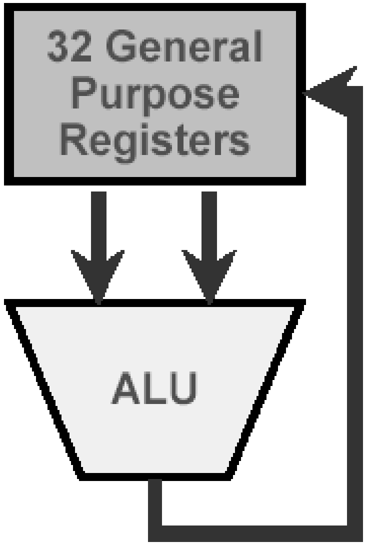
AVR Architecture

- What are the features of RISC?
  - 1 instruction per clock cycle (pipelined)
  - Lots of registers: 32 GP registers
  - Register-to-register operation

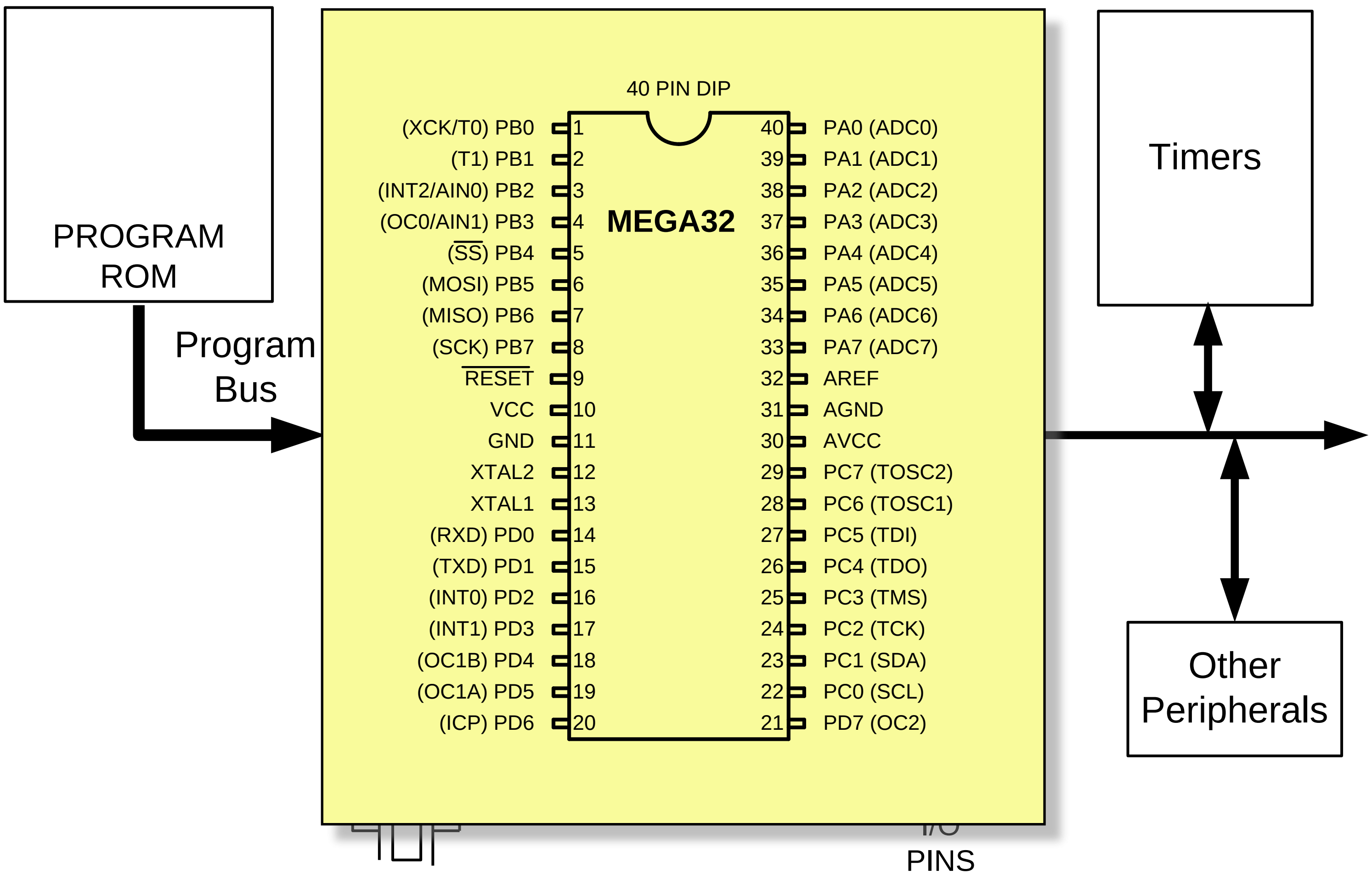
- Variations in the parts:
  - TINY to MEGA
  - ATtiny10
    - Processor has only 8 pins
  - ATmega128 (128K bytes flash)
    - Processor has 64 pins

AVR Architecture

- **RISC architecture with CISC instruction set**
  - Easy to learn and powerful instruction set for C and Assembly
- **Single cycle execution**
  - One instruction per external clock
  - Low power consumption
- **32 Working Registers**
  - All registers are directly connected to ALU!
- **Very efficient core**
  - New design using new technology
  - Fully scalable for future products

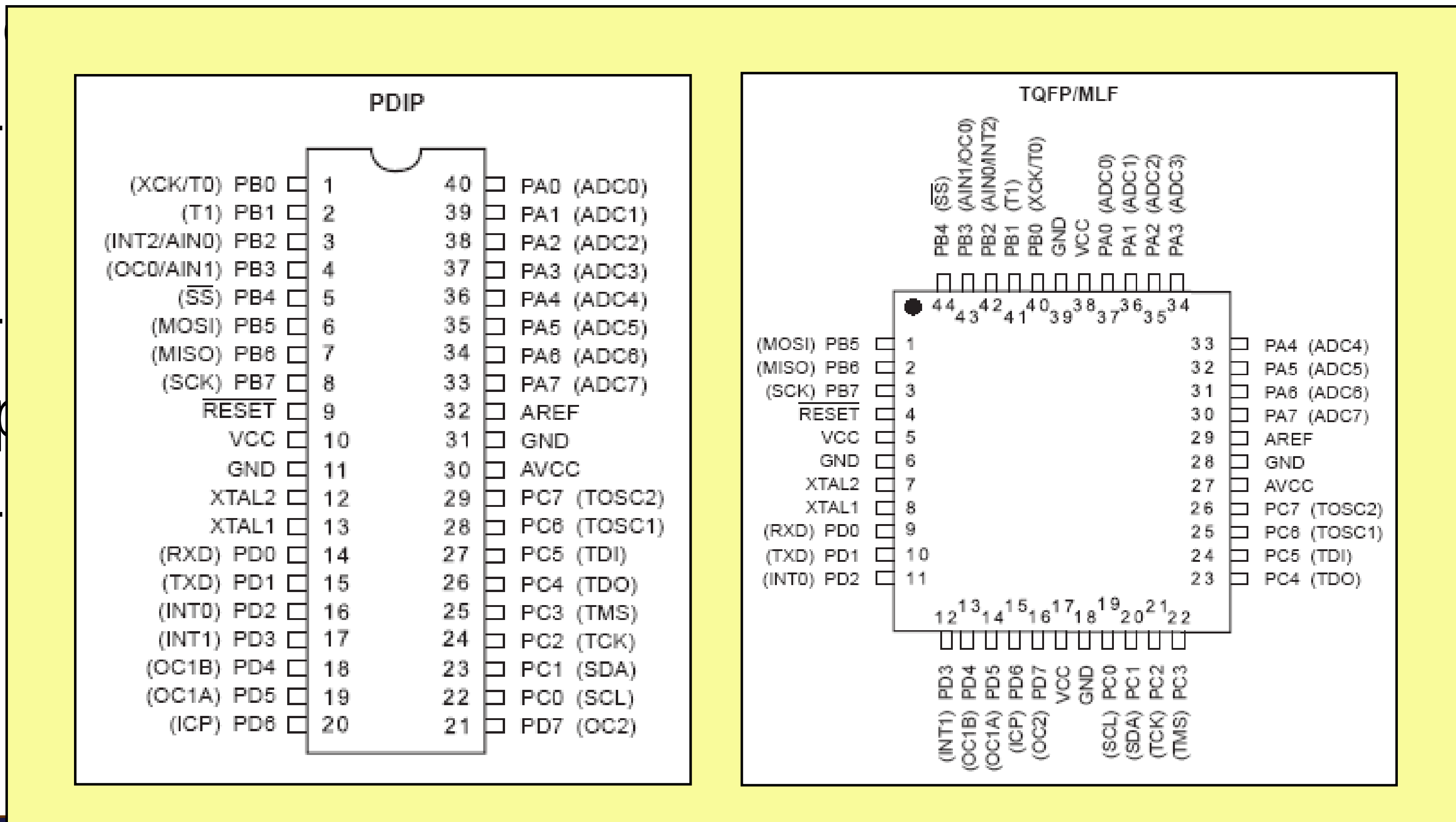


AVR Internal Architecture

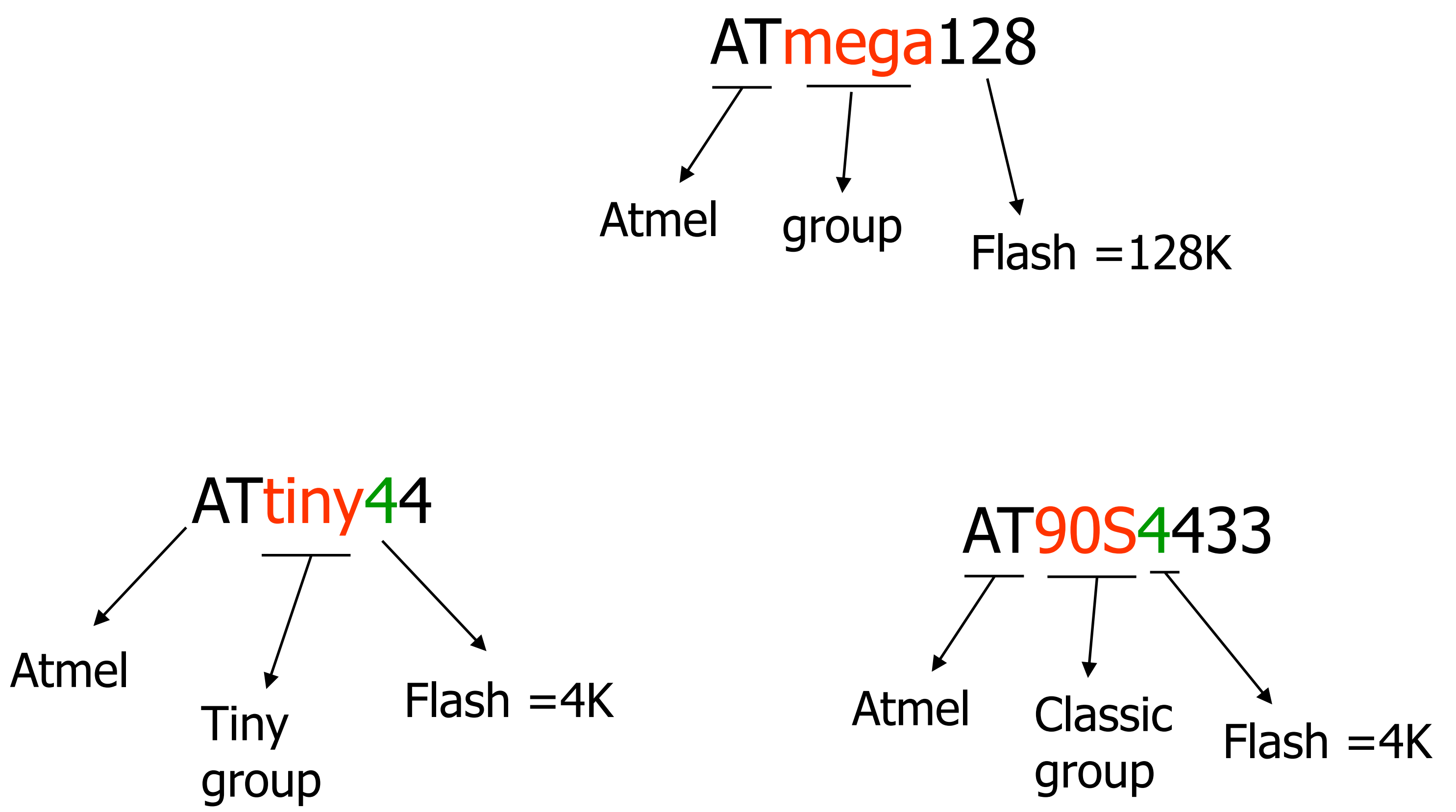


AVR Different Groups

- Classic AVR
  - e.g. AT90S2313, AT90S4433
- M
  -
- Ti
  -
- Sp
  -



Let's get familiar with the AVR part numbers



AVR Architecture

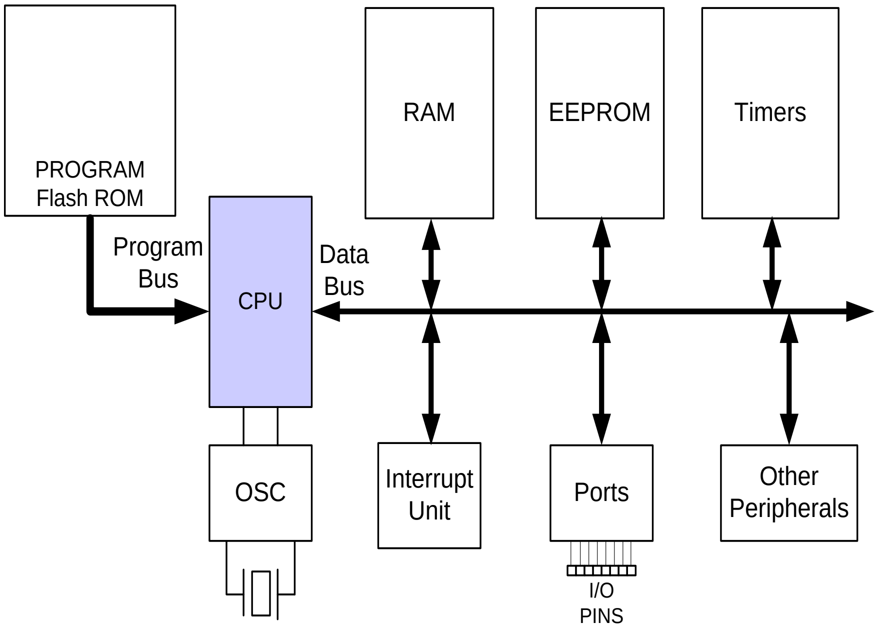
Mostly from Chapter 2 (pp 55 - 75)

The AVR microcontroller and embedded systems

using assembly and c

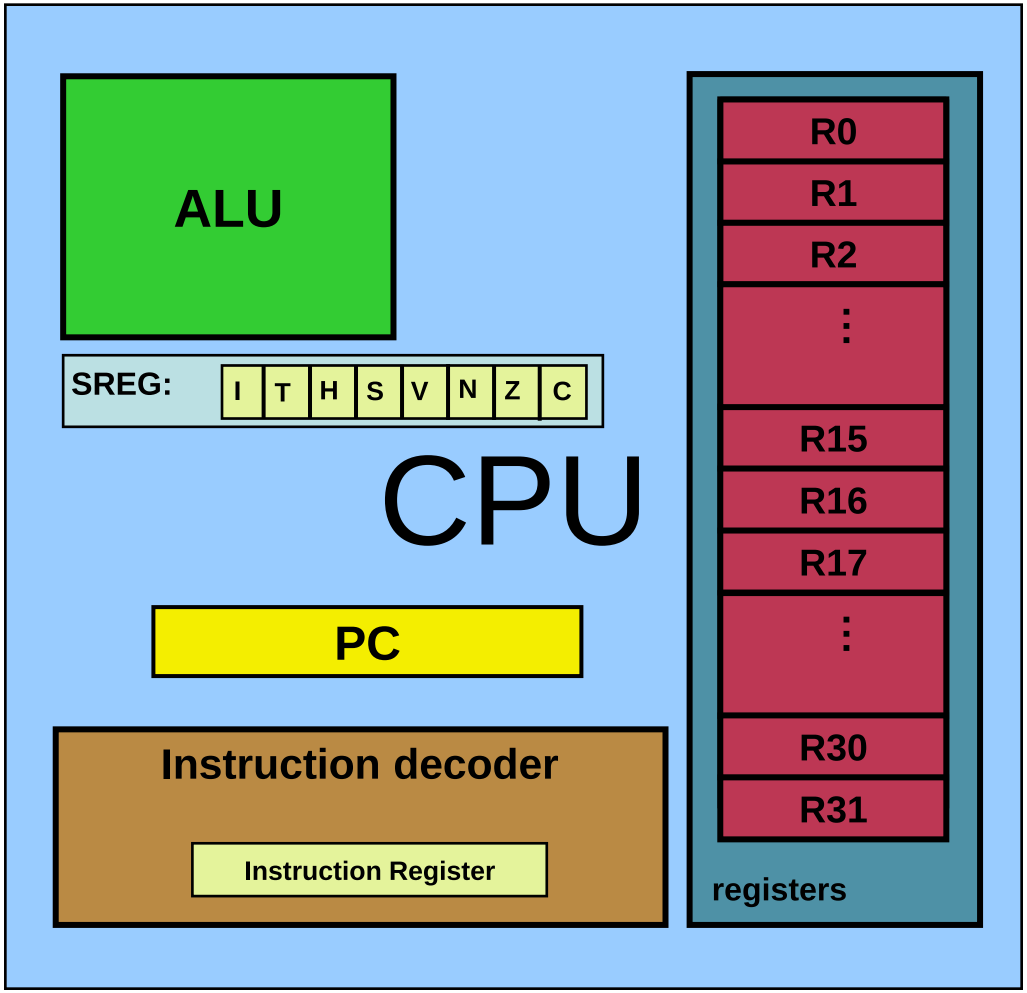
Topics

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



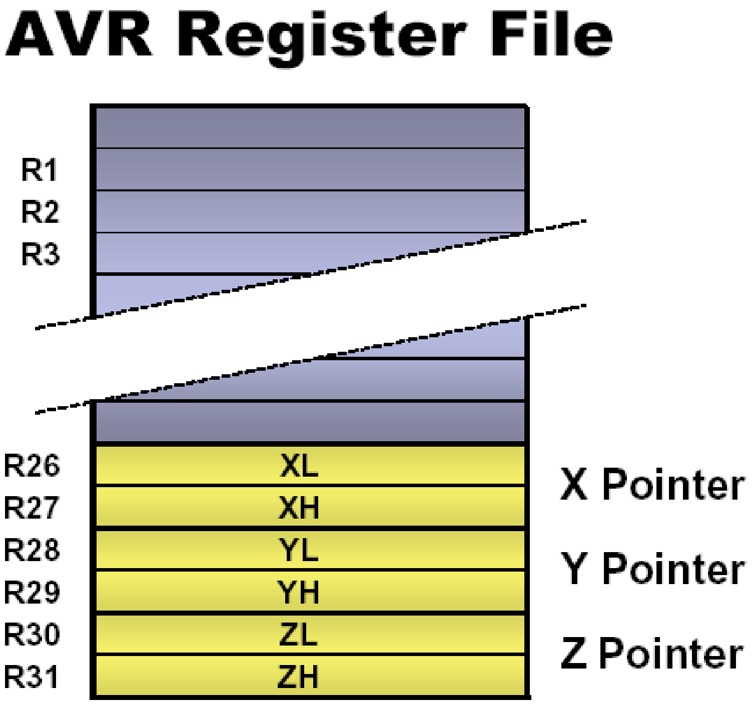
AVR's CPU

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



AVR Memory Space

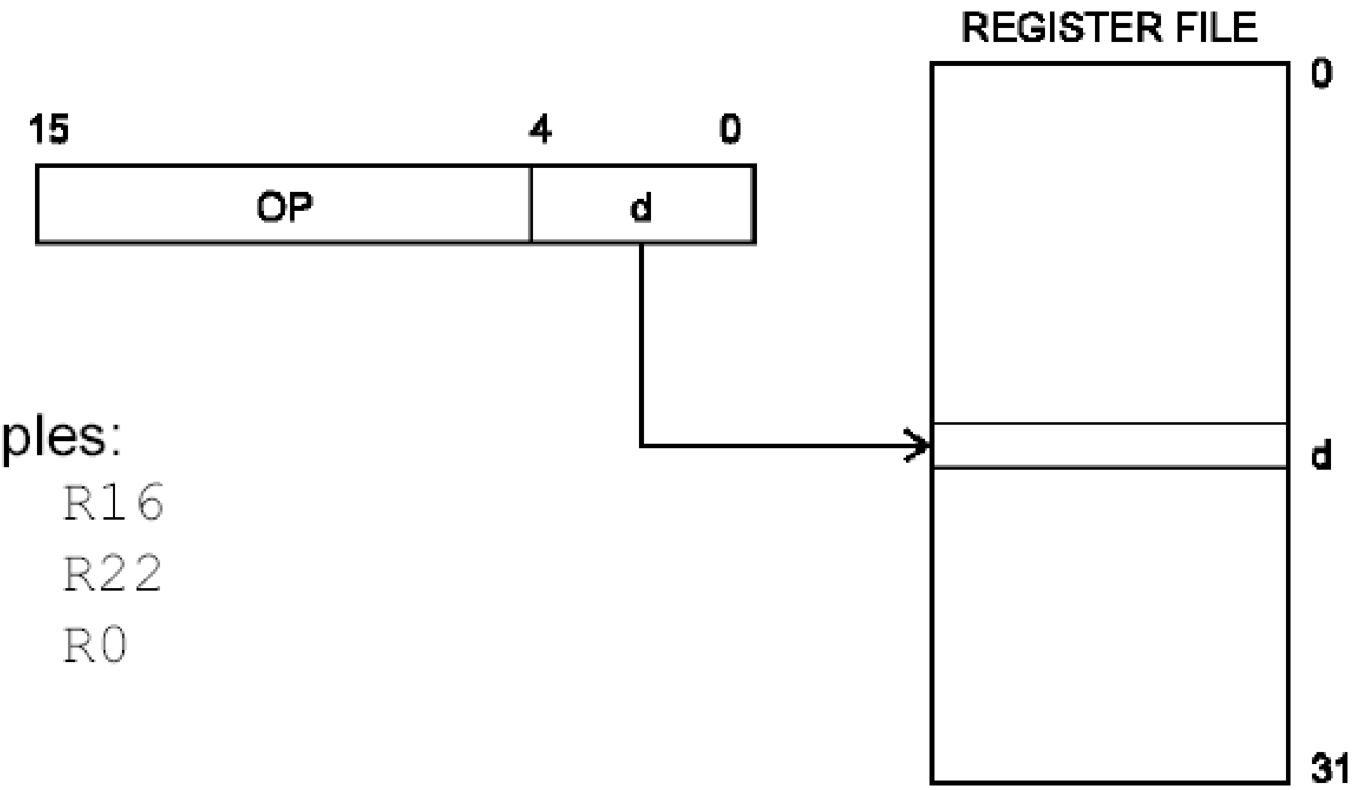
- Program Flash
  - Vectors, Code, and (Unchangeable) Constant Data
- Working Registers
  - Includes X, Y, and Z registers.
- I/O Register Space
  - Includes "named" registers
- SRAM – Data Space
  - Runtime Variables and Data
  - Stack space
- EEPROM space
  - For non-volatile but alterable data



AVR Addressing Modes

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

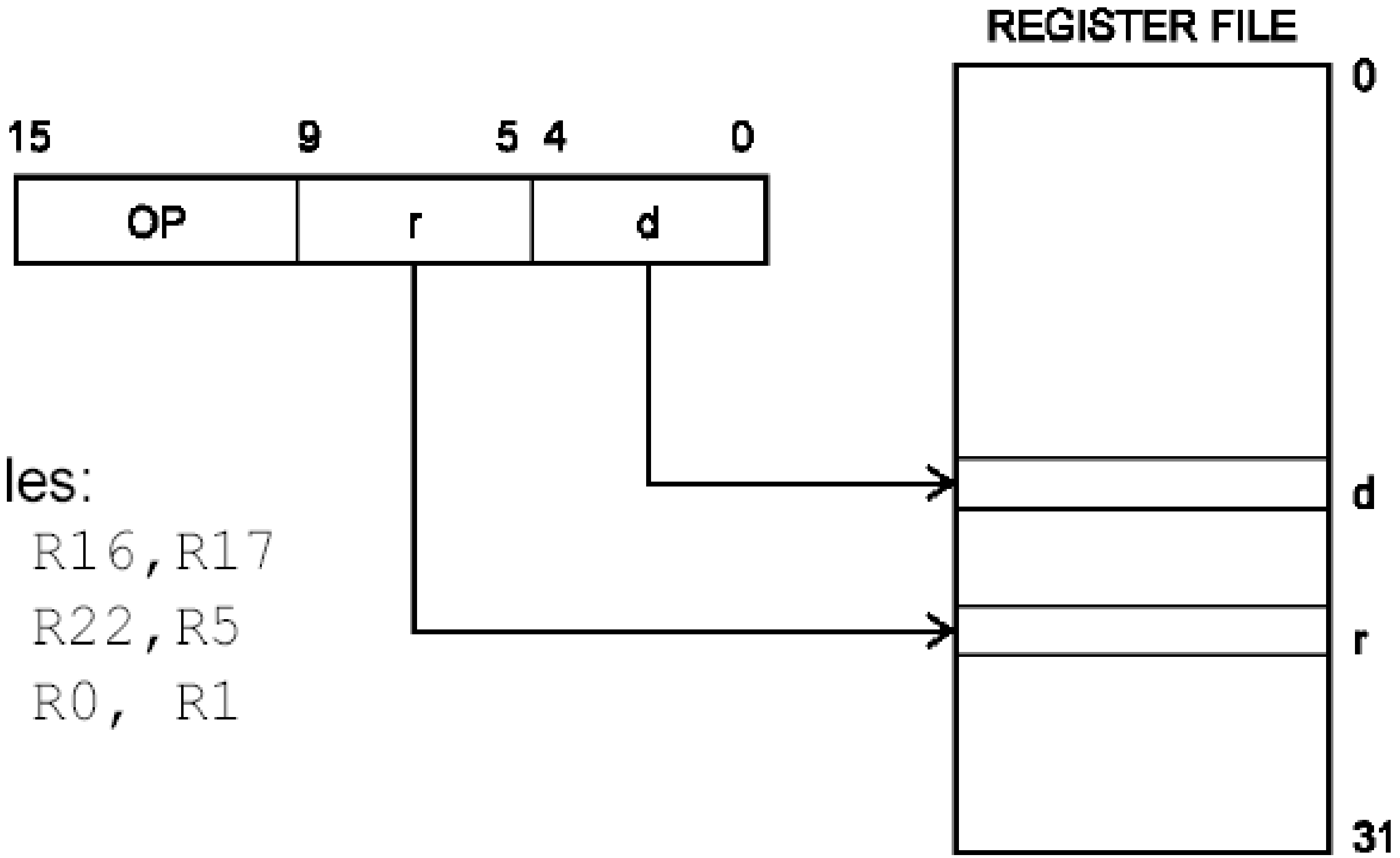
Register Direct: 1 Register



Examples:

```
INC    R16
CLR    R22
EOR    R0
```

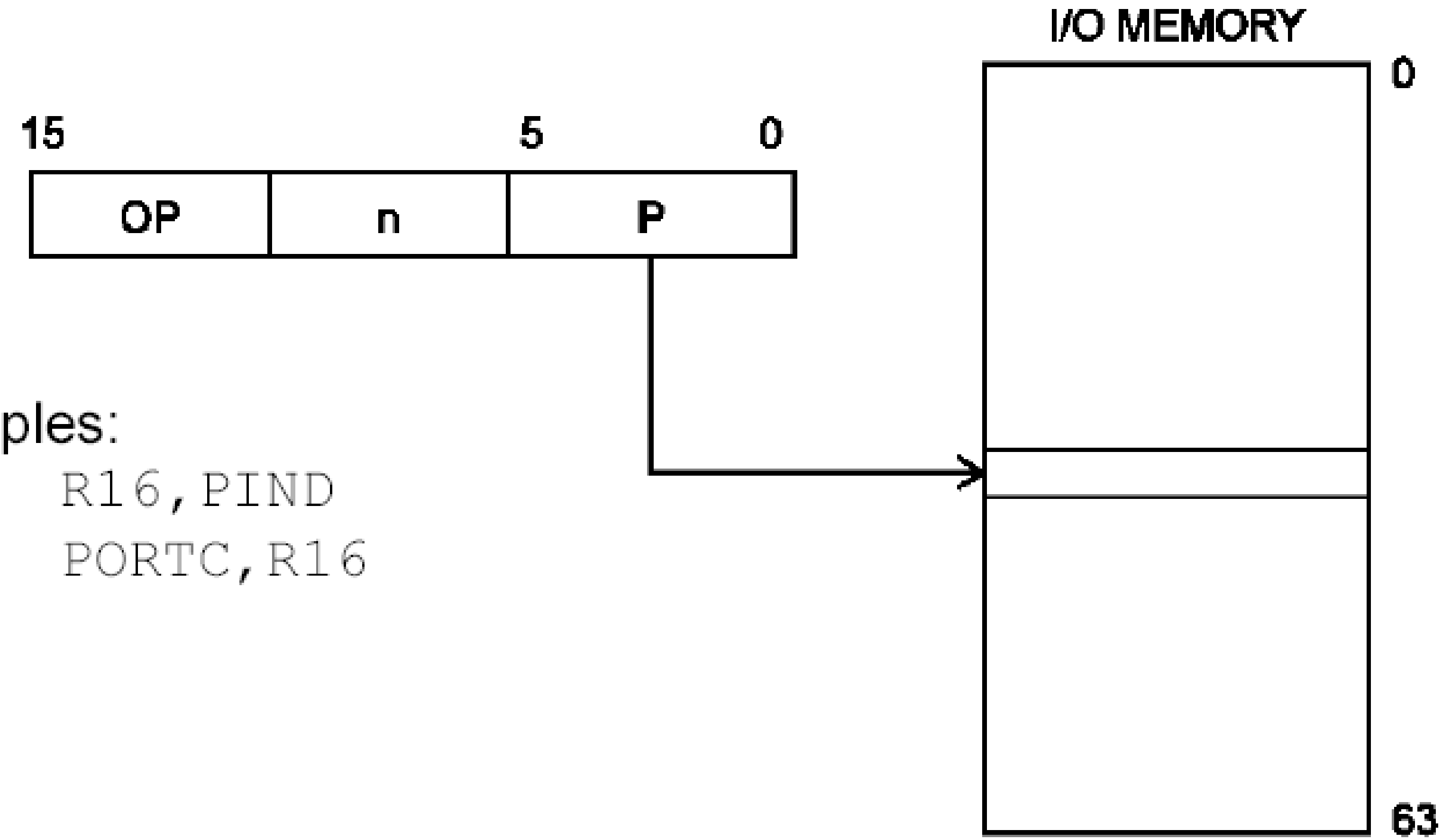
Register Direct: 2 Registers



Examples:

```
ADD    R16,R17
CP     R22,R5
MOV    R0, R1
```

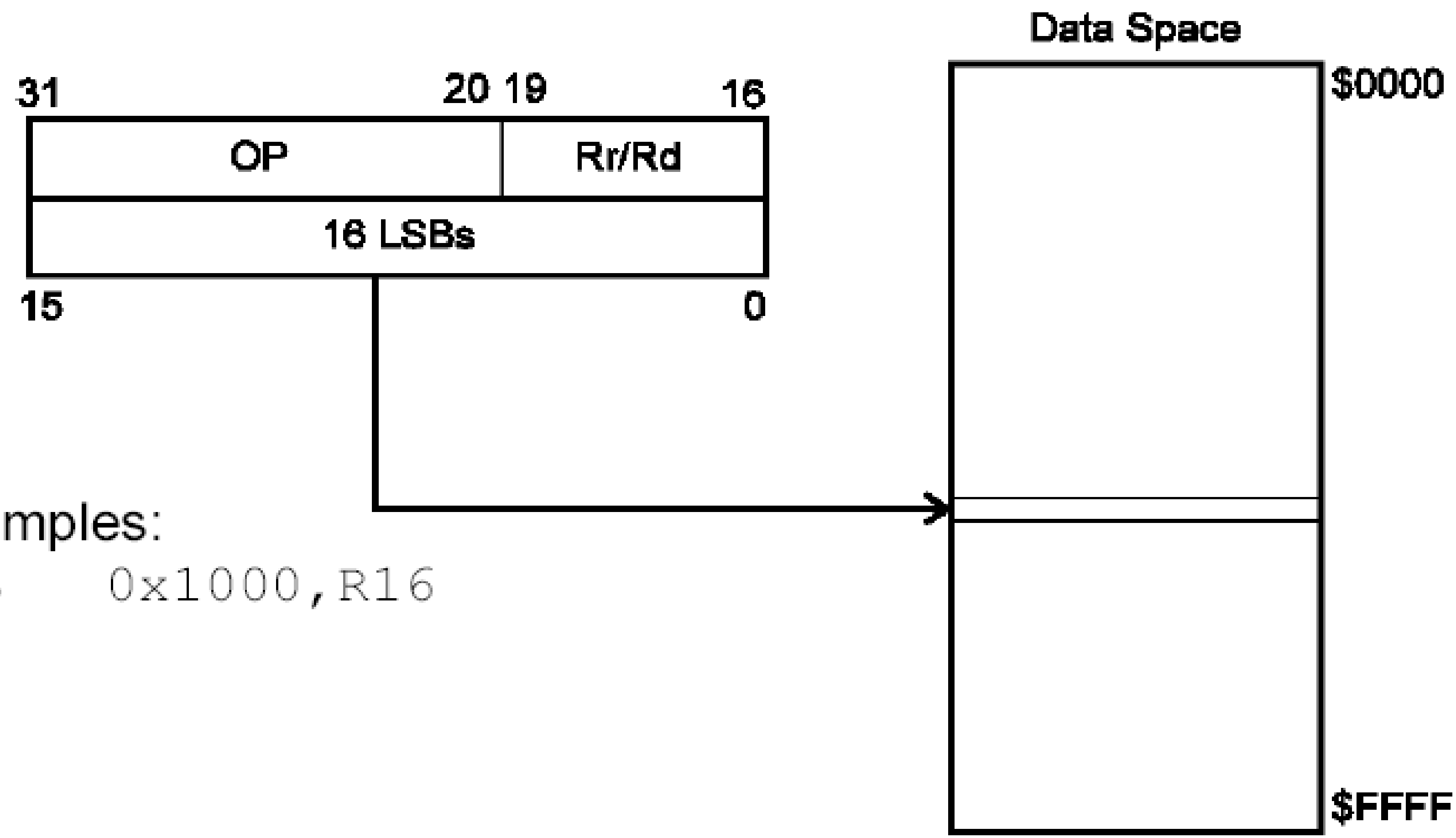
I/O Direct



Examples:

```
IN      R16,PIND
OUT     PORTC,R16
```

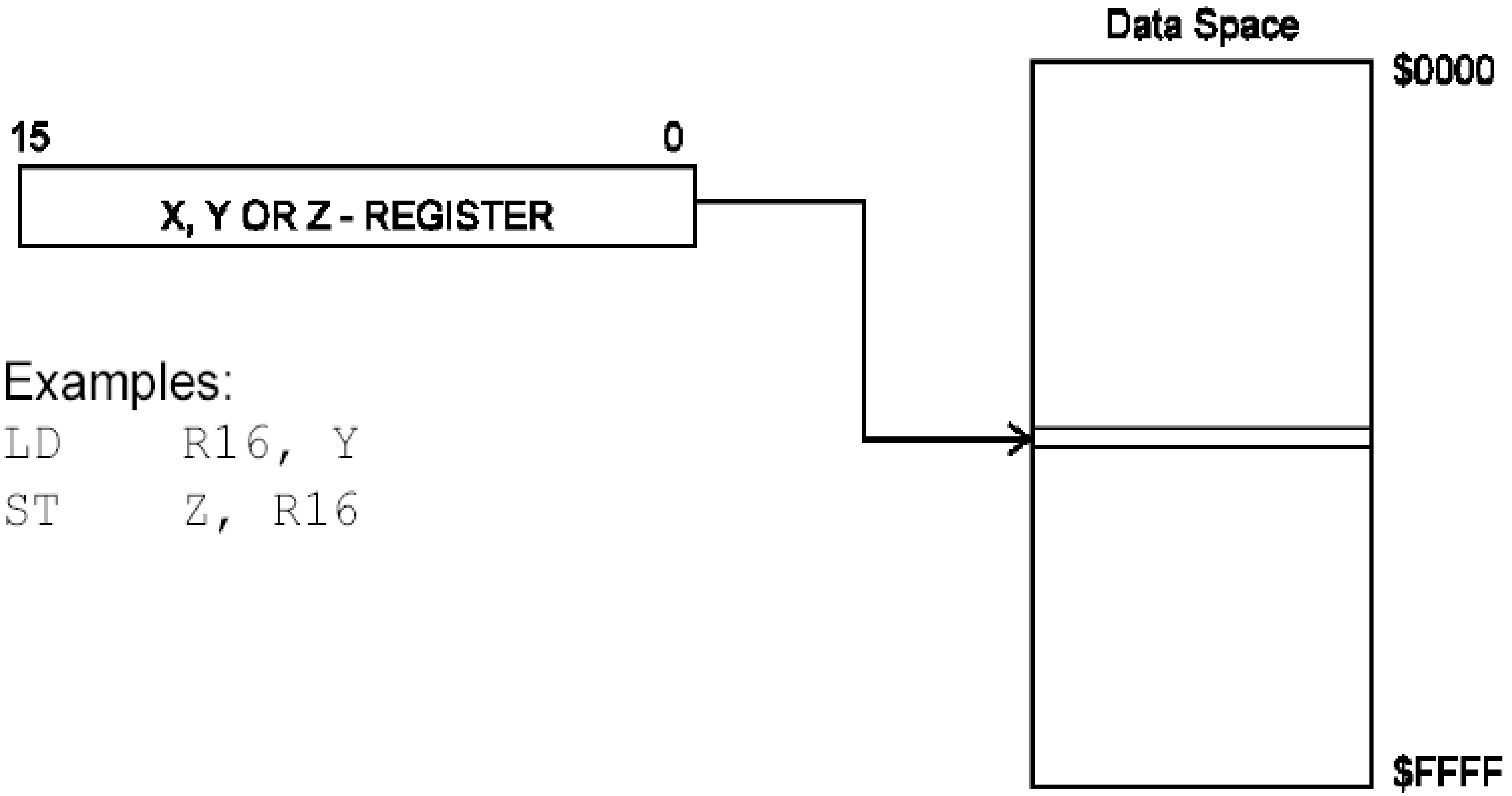
Data Direct



Examples:

```
STS    0x1000,R16
```

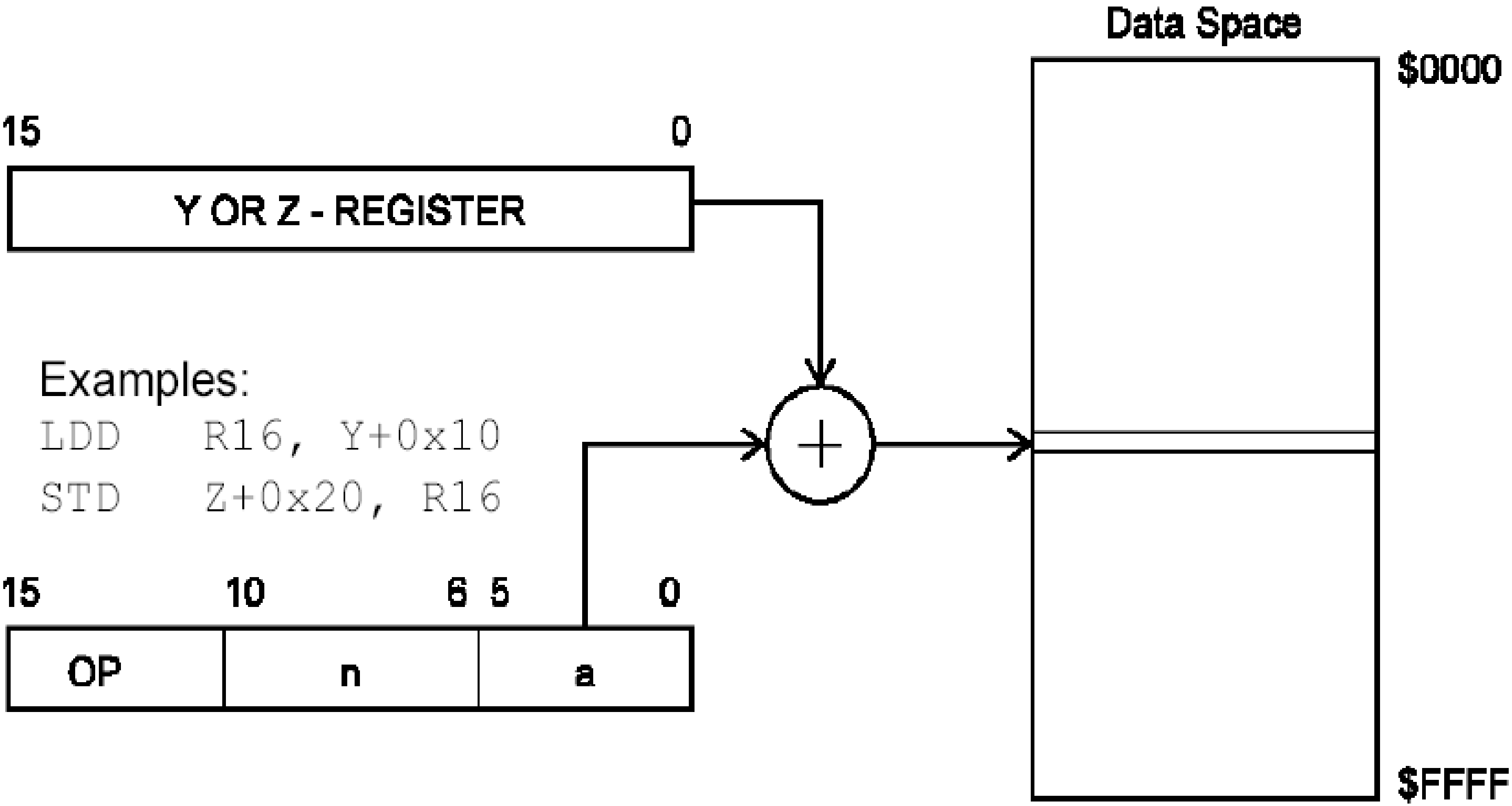
Data Indirect



Examples:

```
LD     R16, Y
ST     Z, R16
```

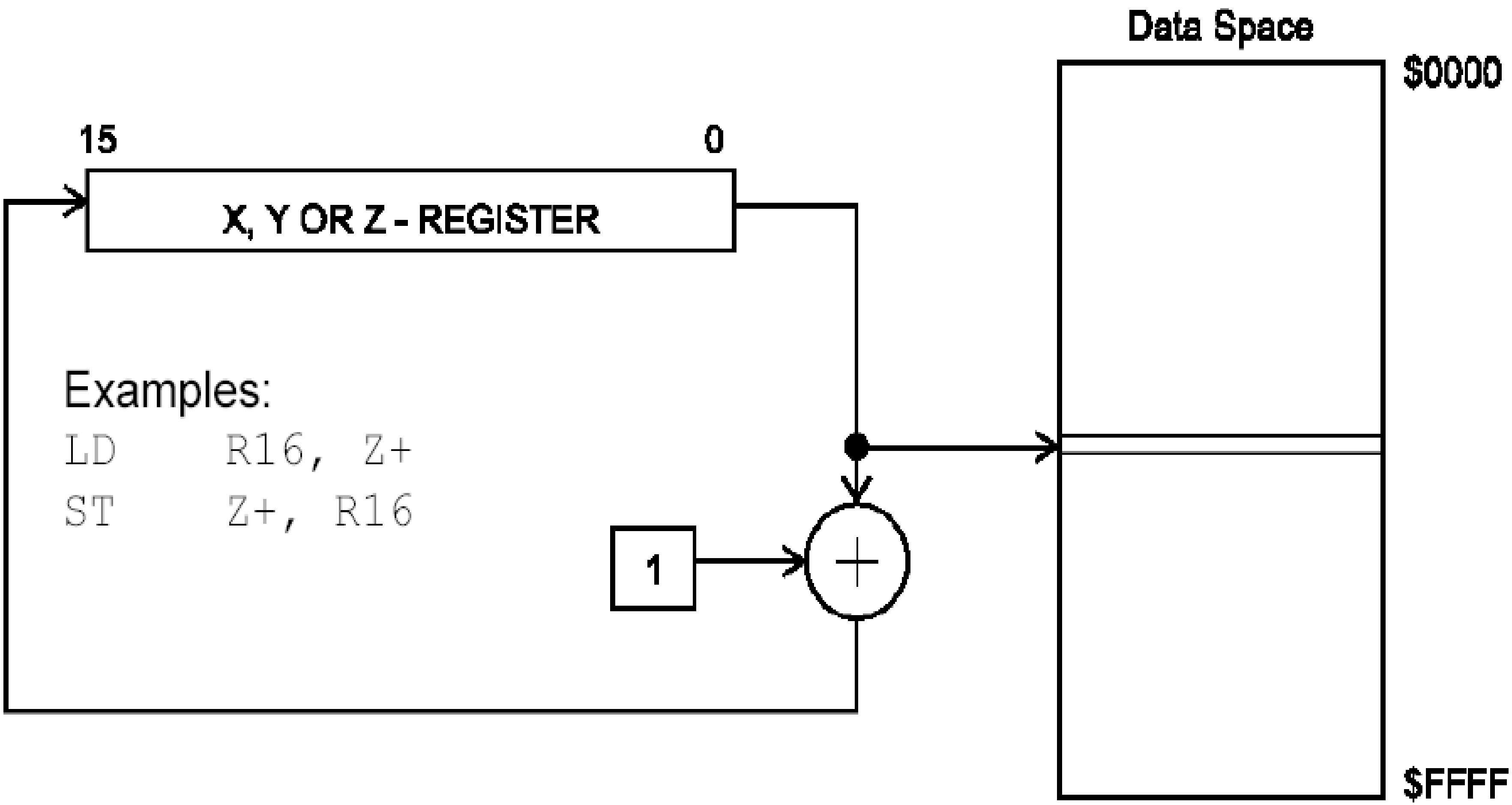
Data Indirect w/ Displacement



Examples:

```
LDD    R16, Y+0x10
STD    Z+0x20, R16
```

Data Indirect: Post-Increment



Examples:

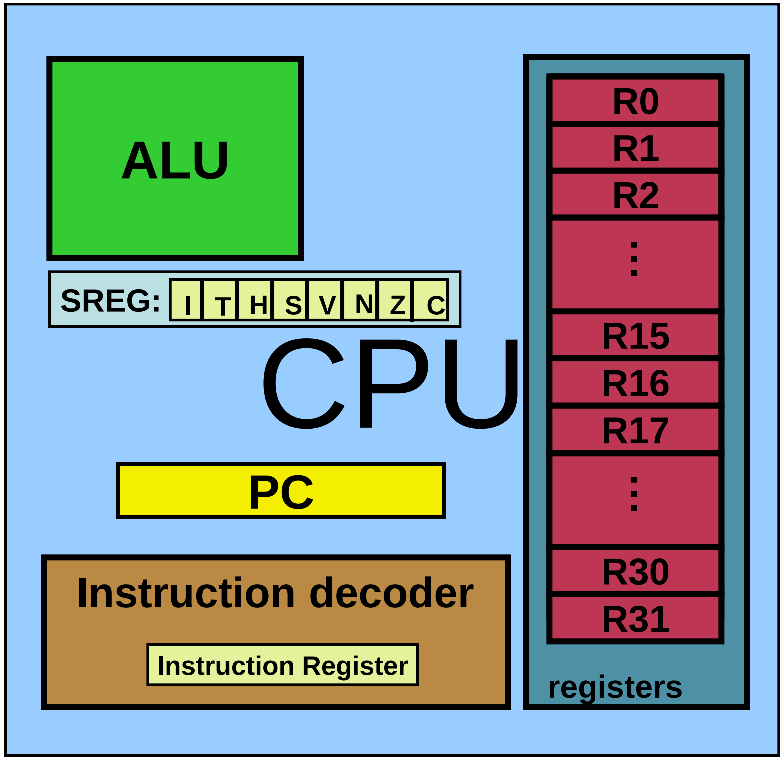
```
LD     R16, Z+
ST     Z+, R16
```

Some Simple Instructions
1. Loading values into the general purpose registers

LDI (**L**oad **I**mmediate)

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

    Rd = k
- Example:
  - LDI R16,53
    - R16 = 53
  - LDI R19,132
  - LDI R23,0x27
    - R23 = 0x27



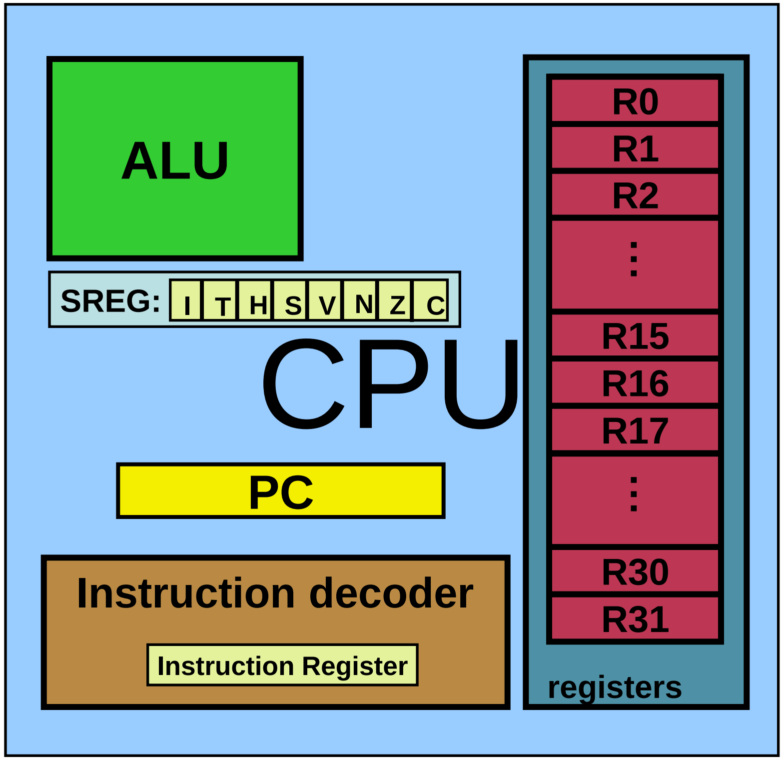
Some Simple Instructions

2. Arithmetic calculation

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

  ADD, SUB, MUL, AND, etc.

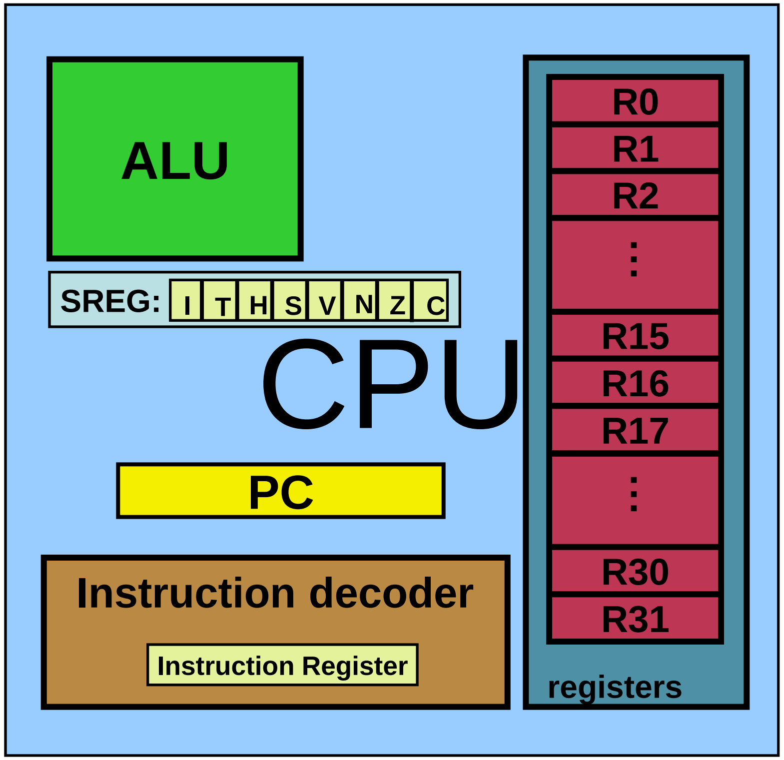
- ADD Rd,Rs
  - Rd = Rd + Rs
  - Example:
  - ADD R25, R9
    - R25 = R25 + R9
  - ADD R17,R30
    - R17 = R17 + R30



A simple program

- Write a program that calculates 19 + 95

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



A simple program

- Write a program that calculates 19 + 95 + 5

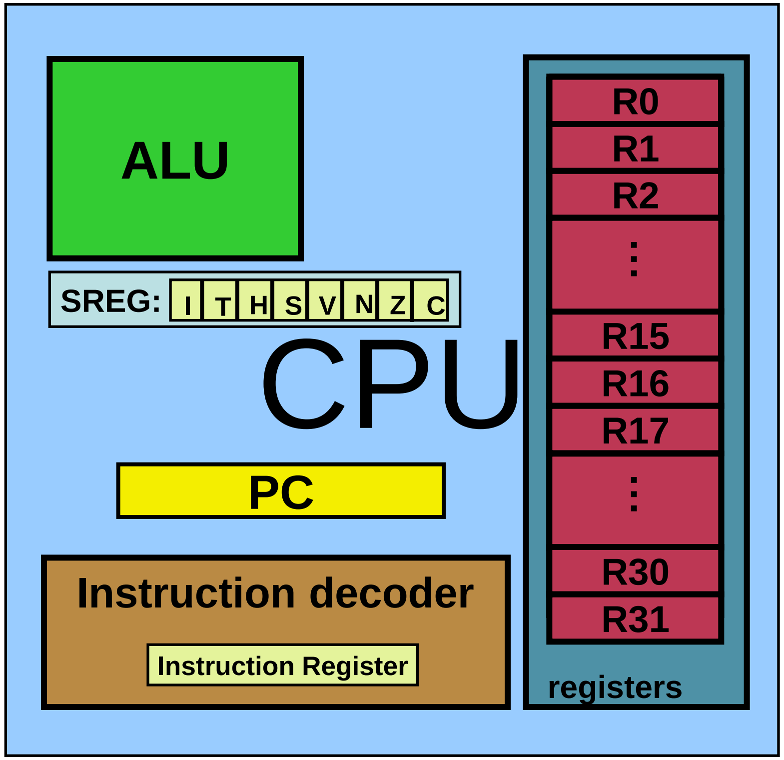
```
LDI     R16, 19         ;R16 = 19
LDI     R20, 95         ;R20 = 95
LDI     R21, 5          ;R21 = 5
ADD     R16, R20        ;R16 = R16 + R20
ADD     R16, R21        ;R16 = R16 + R21
```

```
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
```

Some Simple Instructions

2. Arithmetic calculation

- SUB Rd,Rs
  - Rd = Rd - Rs
- Example:
  - SUB R25, R9
    - R25 = R25 - R9
  - SUB R17,R30
    - R17 = R17 - R30



R0 thru R15

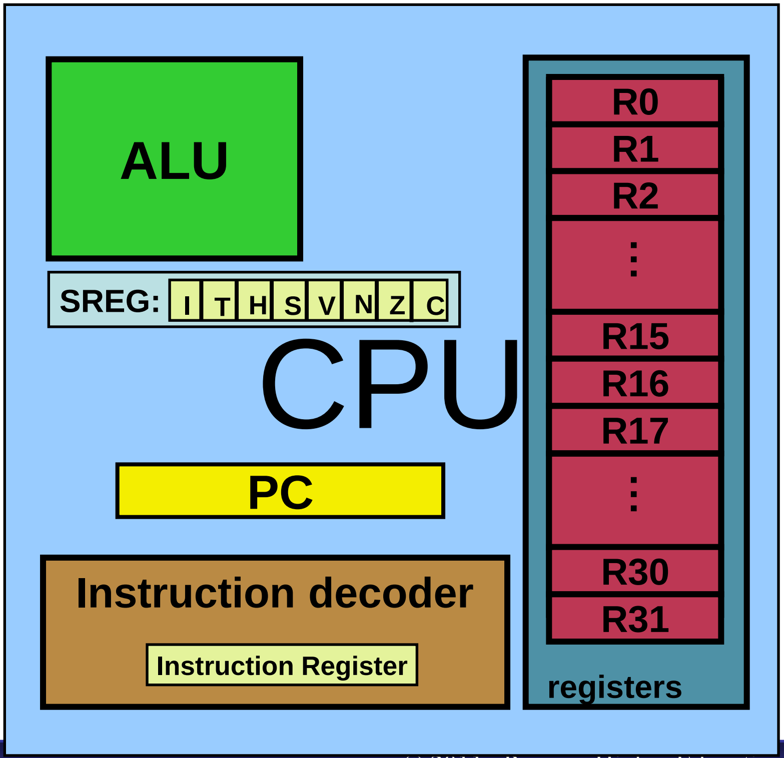
- Only registers in the range R16 to R31 can be loaded immediate. We cannot load a constant into the registers R0 to R15 directly. It would have to be loaded into a valid register first then copied. To load the value of 10 into register zero (R0):
  Code:
-

```
LDI R16, 10        ;Set R16 to value of 10
MOV R0, R16        ;Copy contents of R16 to R0
```

Some simple instructions

2. Arithmetic calculation

- INC Rd
  - Rd = Rd + 1
- Example:
  - INC R25
    - R25 = R25 + 1
- DEC Rd
  - Rd = Rd - 1
- Example:
  - DEC R23
    - R23 = R23 - 1

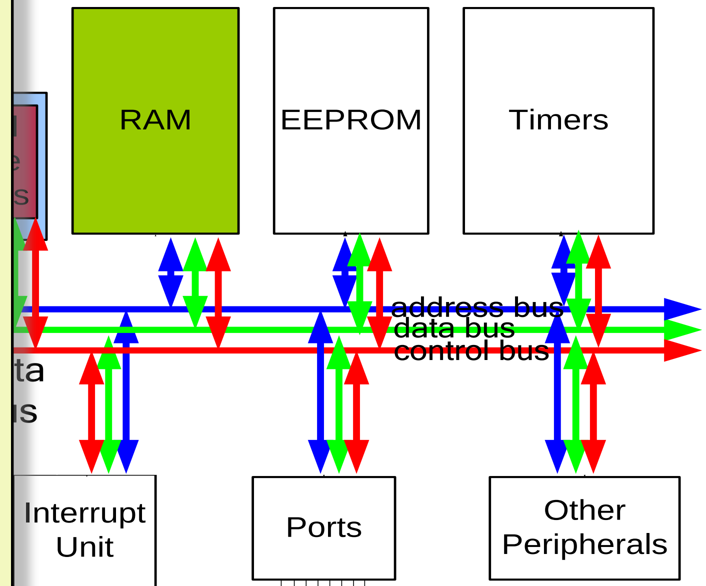


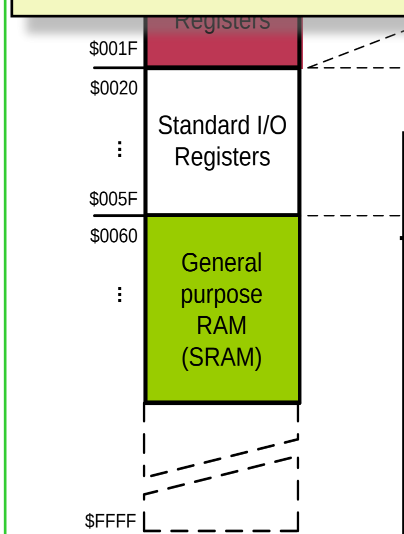
Space

Address I/O	Address Mem.	Name
$00	$20	TWBR
$01	$21	TWSR
$02	$22	TWAR
$03	$23	TWDR
$04	$24	ADCL
$05	$25	ADCH
$06	$26	ADCSRA
$07	$27	ADMUX
$08	$28	ACSR
$09	$29	UBRRL
$0A	$2A	UCSRB
$0B	$2B	UCSRA
$0C	$2C	UDR
$0D	$2D	SPCR
$0E	$2E	SPSR
$0F	$2F	SPDR
$10	$30	PIND
$11	$31	DDRD
$12	$32	PORTD
$13	$33	PINC
$14	$34	DDRC
$15	$35	PORTC

Address I/O	Address Mem.	Name
$16	$36	PINB
$17	$37	DDRB
$18	$38	PORTB
$19	$39	PINA
$1A	$3A	DDRA
$1B	$3B	PORTA
$1C	$3C	EECR
$1D	$3D	EEDR
$1E	$3E	EEARL
$1F	$3F	EEARH
$20	$40	UBRRC UBRRH
$21	$41	WDTCR
$22	$42	ASSR
$23	$43	OCR2
$24	$44	TCNT2
$25	$45	TCCR2
$26	$46	ICR1L
$27	$47	ICR1H
$28	$48	OCR1BL
$29	$49	OCR1BH
$2A	$4A	OCR1AL

Address I/O	Address Mem.	Name
$2B	$4B	OCR1AH
$2C	$4C	TCNT1L
$2D	$4D	TCNT1H
$2E	$4E	TCCR1B
$2F	$4F	TCCR1A
$30	$50	SFIOR
$31	$51	OCDR OSCCAL
$32	$52	TCNT0
$33	$53	TCCR0
$34	$54	MCUCSR
$35	$55	MCUCR
$36	$56	TWCR
$37	$57	SPMCR
$38	$58	TIFR
$39	$59	TIMSK
$3A	$5A	GIFR
$3B	$5B	GICR
$3C	$5C	OCR0
$3D	$5D	SPL
$3E	$5E	SPH
$3E	$5E	SREG





Example: Add conten

Example: What does t

```
LDS   R20,2
```

Answer:

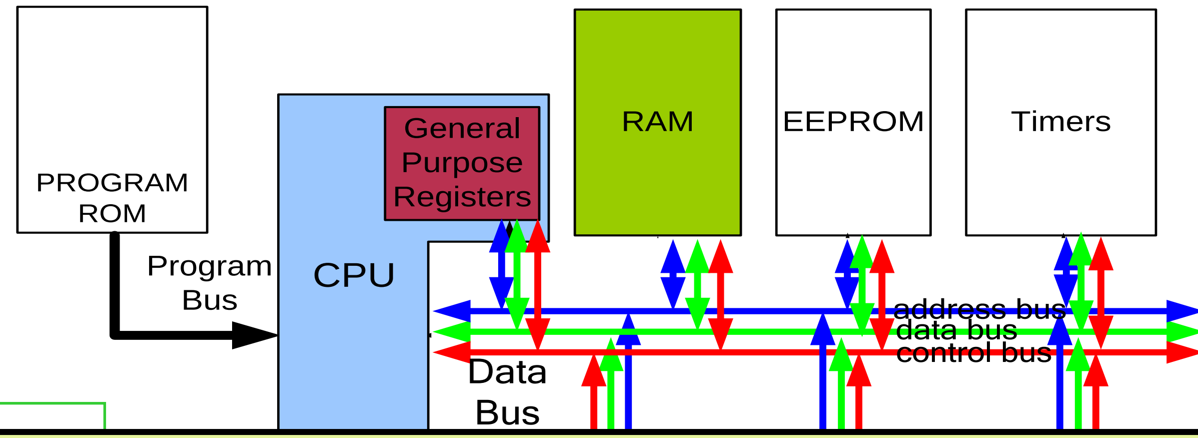
It copies the conte

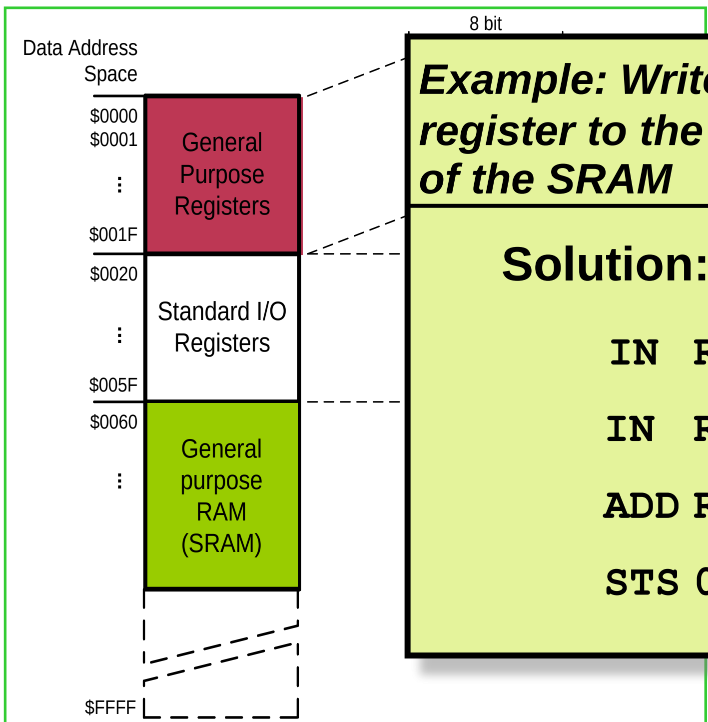
Example: Store 0x53 into the SPH register. The address of SPH is 0x5E

Solution:

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

Data Address Space





Example: Write a program that adds the contents of the PINC IO 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
```

Status Register (SREG)

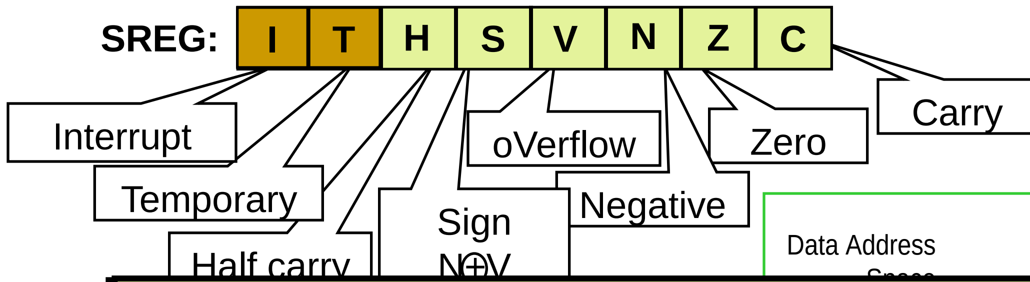


Table 2-5: AVR Branch (Jump) Instructions Using Flag Bits

Instruction	Action
BRLO	Branch if C = 1
BRSH	Branch if C = 0
BREQ	Branch if Z = 1
BRNE	Branch if Z = 0
BRMI	Branch if N = 1
BRPL	Branch if N = 0
BRVS	Branch if V = 1
BRVC	Branch if V = 0

Example: Show the status of the C, H, and Z fla subtraction of 0x9C from 0x9C in the following

```
LDI    R20, 0x9C
LDI    R21, 0x9C
SUB    R20, R21        ;subtract R21 from R20
```

Solution:

	$9C	**1001 1100**	
-	**$9C**	**1001 1100**	
	$00	**0000 0000**	**R20 = $00**

C = 0 because R21 is not bigger than R20 and there is no borrow from D8 bit.
Z = 1 because the R20 is zero after the subtraction.
H = 0 because there is no borrow from D4 to D3.

Writing Immediate Value to SRAM

- You cannot copy immediate value directly into SRAM location in AVR.
- This must be done via GPRs
- Example: The following program adds content of location 0x220 to location 0x221

```
LDS    R30, 0x220   ;load R30 with the contents of location 0x220
LDS    R31, 0x221   ;load R31 with the contents of location 0x221
ADD    R31, R30     ;add R30 to R31
STS    0x221, R31   ;store R31 to data space location 0x221
```

Example 2-1

State the contents of RAM locations $212 to $216 after the following program is executed:

```
LDI   R16, 0x99    ;load R16 with value 0x99
STS   0x212, R16
LDI   R16, 0x85    ;load R16 with value 0x85
STS   0x213, R16
LDI   R16, 0x3F    ;load R16 with value 0x3F
STS   0x214, R16
LDI   R16, 0x63    ;load R16 with value 0x63
STS   0x215, R16
LDI   R16, 0x12    ;load R16 with value 0x12
STS   0x216, R16
```

Write the program in AVR Studio to verify that

After the execution of `STS 0x212, R16` data memory location $212 has value 0x99;
after the execution of `STS 0x213, R16` data memory location $213 has value 0x85;
after the execution of `STS 0x214, R16` data memory location $214 has value 0x3F;
after the execution of `STS 0x215, R16` data memory location $215 has value 0x63;

Note: do not forget to add iat the beginning of the program:

.include "M32DEF.inc"

Example 2- 2

State the contents of R20, R21, and data memory location 0x120 after the following program:

```
LDI   R20, 5        ;load R20 with 5
LDI   R21, 2        ;load R21 with 2
ADD   R20, R21      ;add R21 to R20
ADD   R20, R21      ;add R21 to R20
STS   0x120, R20    ;store in location 0x120 the contents of R20
```

Verify using AVR Studio

The program loads R20 with value 5. Then it loads R21 with value 2. Then it adds the R21 register to R20 twice. At the end, it stores the result in location 0x120 of data memory.

Location	Data
R20	5
R21	
0x120	

After LDI R20, 5

Location	Data
R20	5
R21	2
0x120	

After LDI R21, 2

Location	Data
R20	7
R21	2
0x120	

After ADD R20, R21

Location	Data
R20	9
R21	2
0x120	

After ADD R20, R21

Location	Data
R20	9
R21	2
0x120	9

After STS 0x120, R20

Example 2- 3

Write a program to get data from the PINB and send it to the I/O register of PORT C continuously.

Solution:

```
AGAIN:IN    R16, PINB   ;bring data from PortB into R16
      OUT   PORTC,R16   ;send it to Port C
      JMP   AGAIN       ;keep doing it forever
```

Example 2- 4

Write a simple program to toggle the I/O register of PORT B continuously forever.

Solution:

```
      LDI    R20, 0x55    ;R20 = 0x55
      OUT    PORTB, R20   ;move R20 to Port B SFR (PB = 0x55)
L1:   COM    R20          ;complement R20
      OUT    PORTB, R20   ;move R20 to Port B SFR
      JMP    L1           ;repeat forever (see Chapter 3 for JMP)
```

Exercises

- Try all the exercises from the book