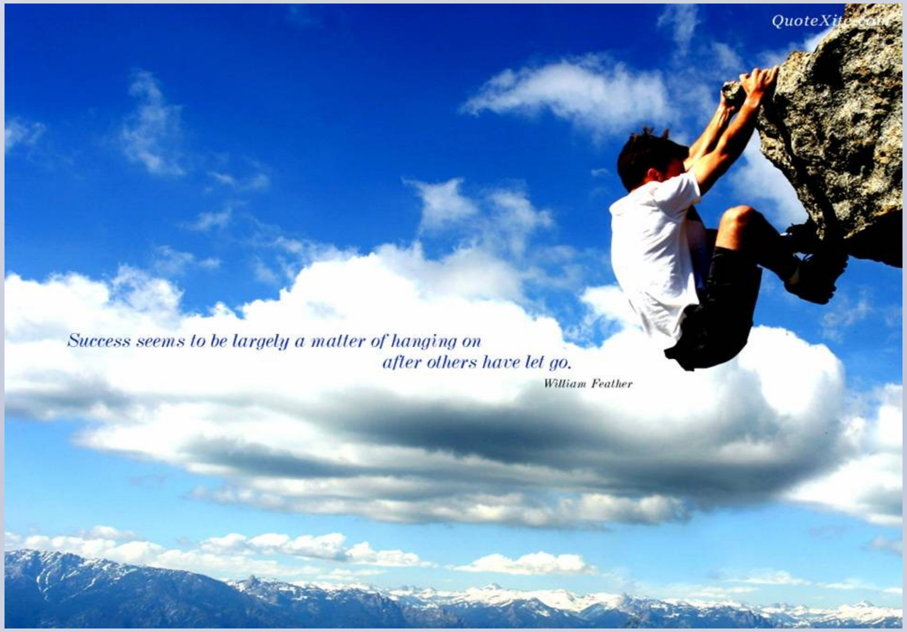
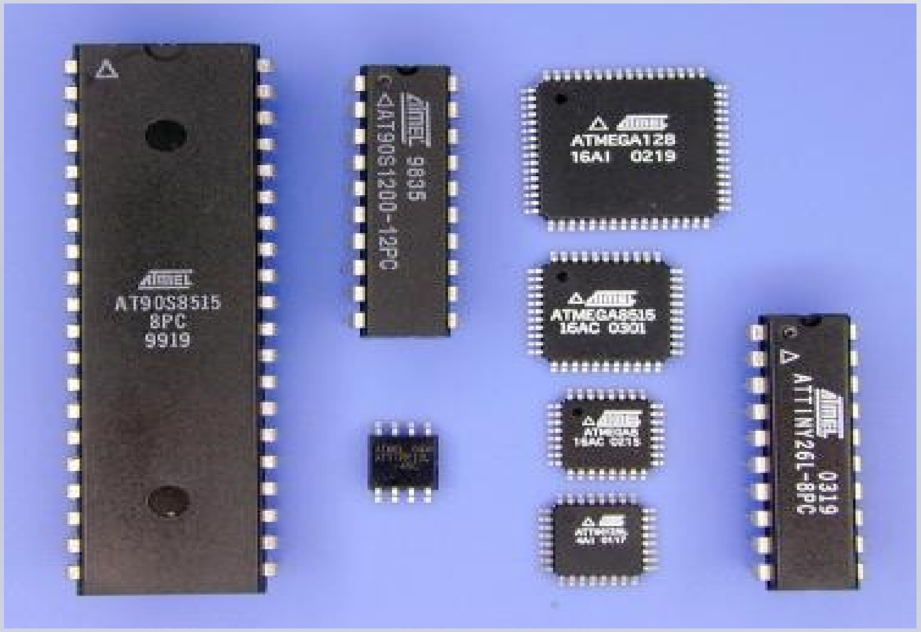
Atmel Avr


QuoteXite.com
Success seems to be largely a matter of hanging on after others have let go.
William Feather



By Vivek Nainwal

C-DAC Hyderabad

Brief History

- AVR basic architecture was conceived by two students at the Norwegian Institute of Technology (NTH) Alf-Egil Bogen and Vegard Wollan.
- The acronym AVR has been reported to stand for Advanced Virtual RISC, but it has also been rumored to stand for the initials of the chip's designers: Alf and Vegard [RISC].
- Atmel says that the name AVR is not an acronym and does not stand for anything in particular.

Device Overview

The AVR is a Harvard architecture machine with programs and data stored separately.

Three Basic Families

- **TinyAvr**
  - ✓ 1-8 kB program memory
  - ✓ 8-32-pin package
  - ✓ Limited peripheral set

- **megaAVRs**
  - ✓ 4-256 kB program memory
  - ✓ 28-100-pin package
  - ✓ Extended instruction set
  - ✓ Extensive peripheral set

- **Application specific AVRs**
  - megaAVRs with special features such as LCD controller, USB controller

Features in atmega8

Advanced RISC Architecture

- 133 Powerful Instructions – Most Single Clock Cycle Execution
- 32 x 8 General Purpose Working Registers + Peripheral Control Registers
- Up to 16 MIPS Throughput at 16 MHz
- On-chip 2-cycle Multiplier

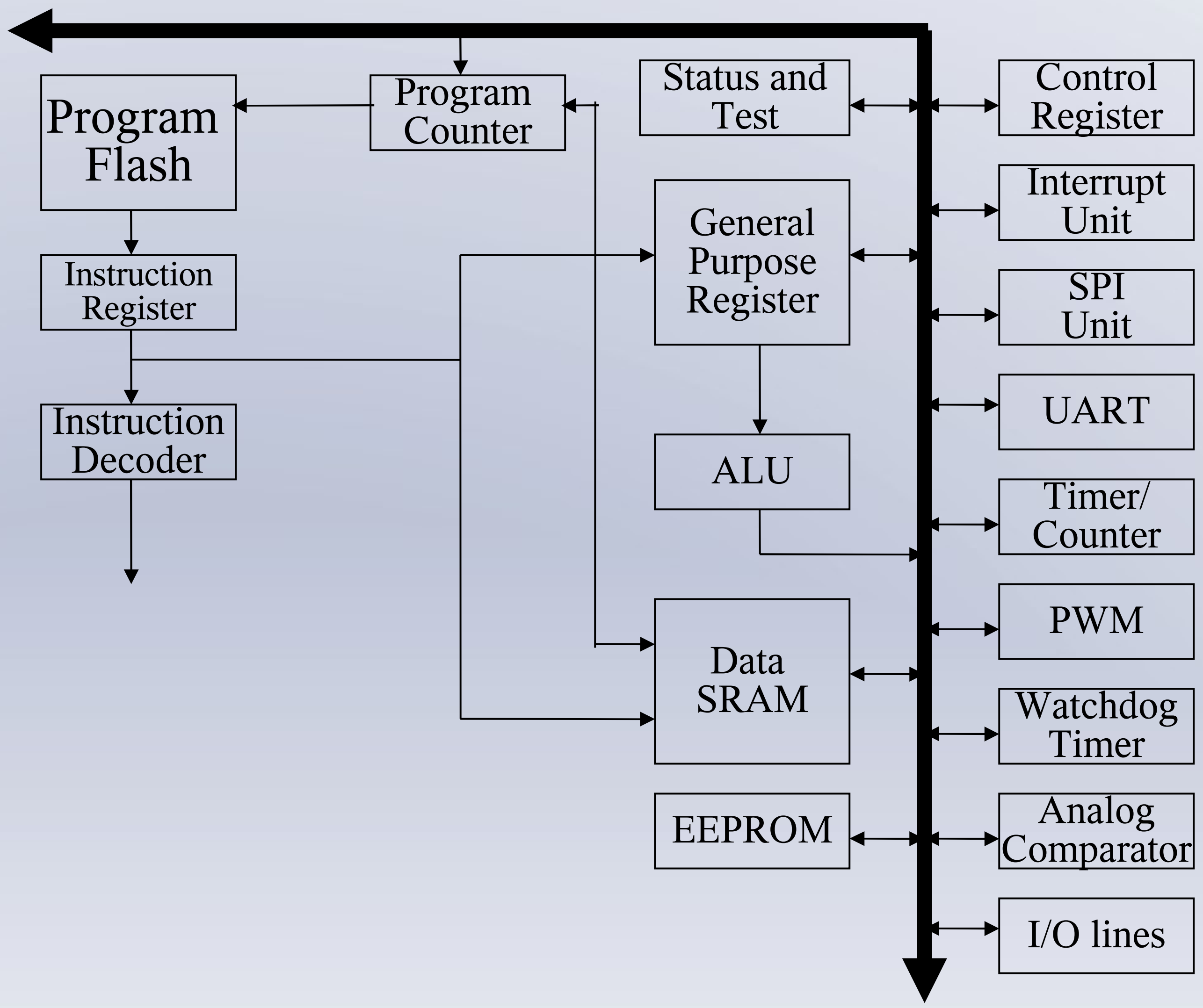
Nonvolatile Program and Data Memories

- 8K Bytes of In-System Reprogrammable Flash
- Endurance: 10,000 Write/Erase Cycles
- 512K Bytes EEPROM
- Endurance: 100,000 Write/Erase Cycles
- 1K Bytes Internal SRAM

- **Peripheral Features**
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counters with Separate Prescaler, Compare Mode and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Three PWM Channels
  - Output Compare Modulator
  - 6-channel, 10-bit ADC
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USARTs
  - On-chip Analog Comparator
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog– On-chip Analog Comparator

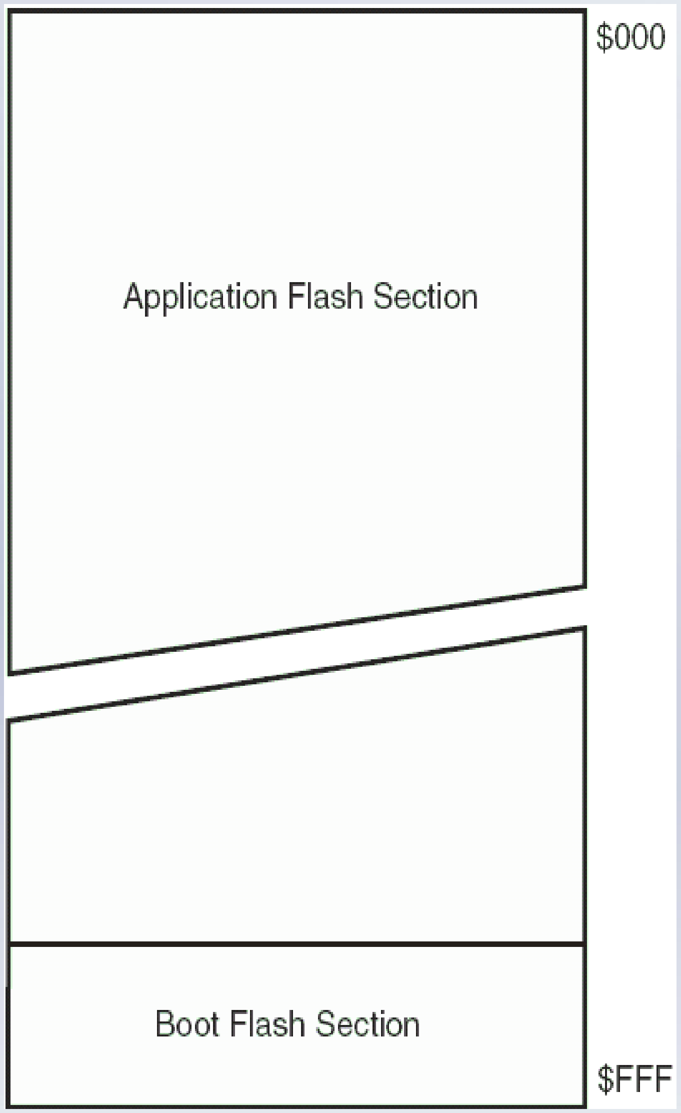
AVR Family Architecture

- RISC Processor
- Harvard Architecture
- 32 X 8 general purpose registers
- On-chip programmable timer
- SLEEP and POWER DOWN modes

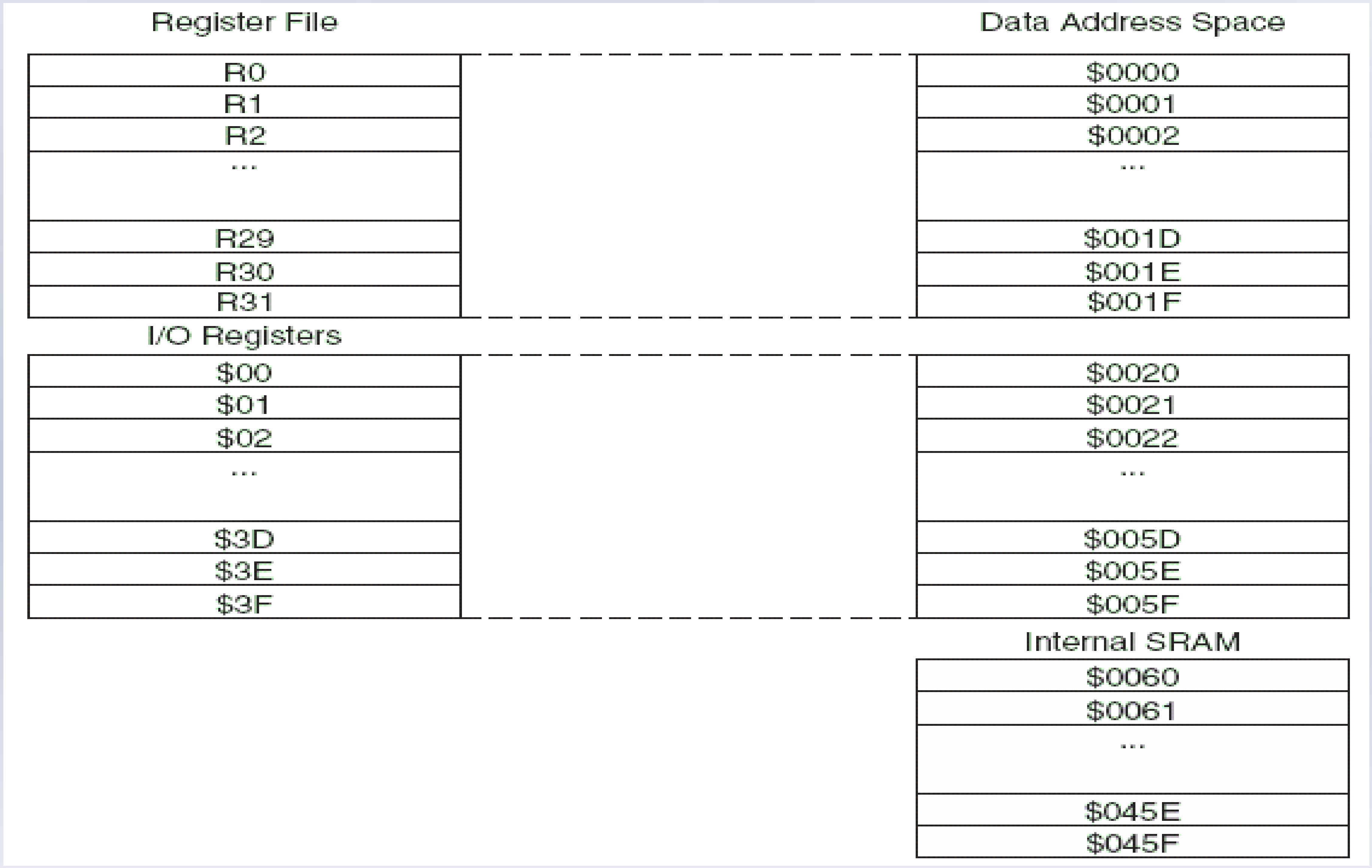


Flash in Atmega8

- Since all AVR instructions are 16 or 32 bits wide, the Flash is organized as 4K x 16.
- For software security, the Flash Program memory space is divided into two sections, Boot Program section and Application Program section.
- Constant tables can be allocated within the entire program memory address space (LPM instruction).



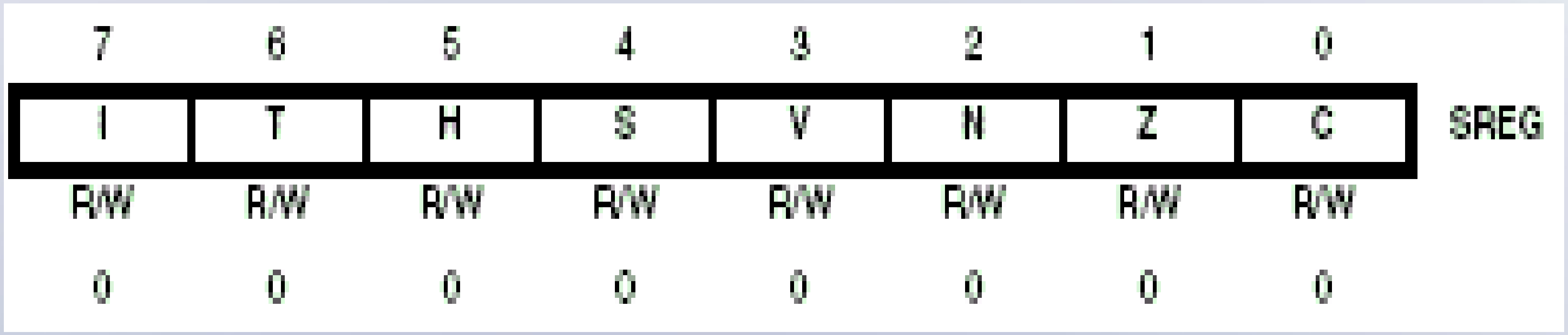
SRAM-1K


Register File
R0
R1
R2
...
R29
R30
R31
I/O Registers
$00
$01
$02
...
$3D
$3E
$3F
Data Address Space
$0000
$0001
$0002
...
$001D
$001E
$001F
$0020
$0021
$0022
...
$005D
$005E
$005F
Internal SRAM
$0060
$0061
...
$045E
$045F

Registers

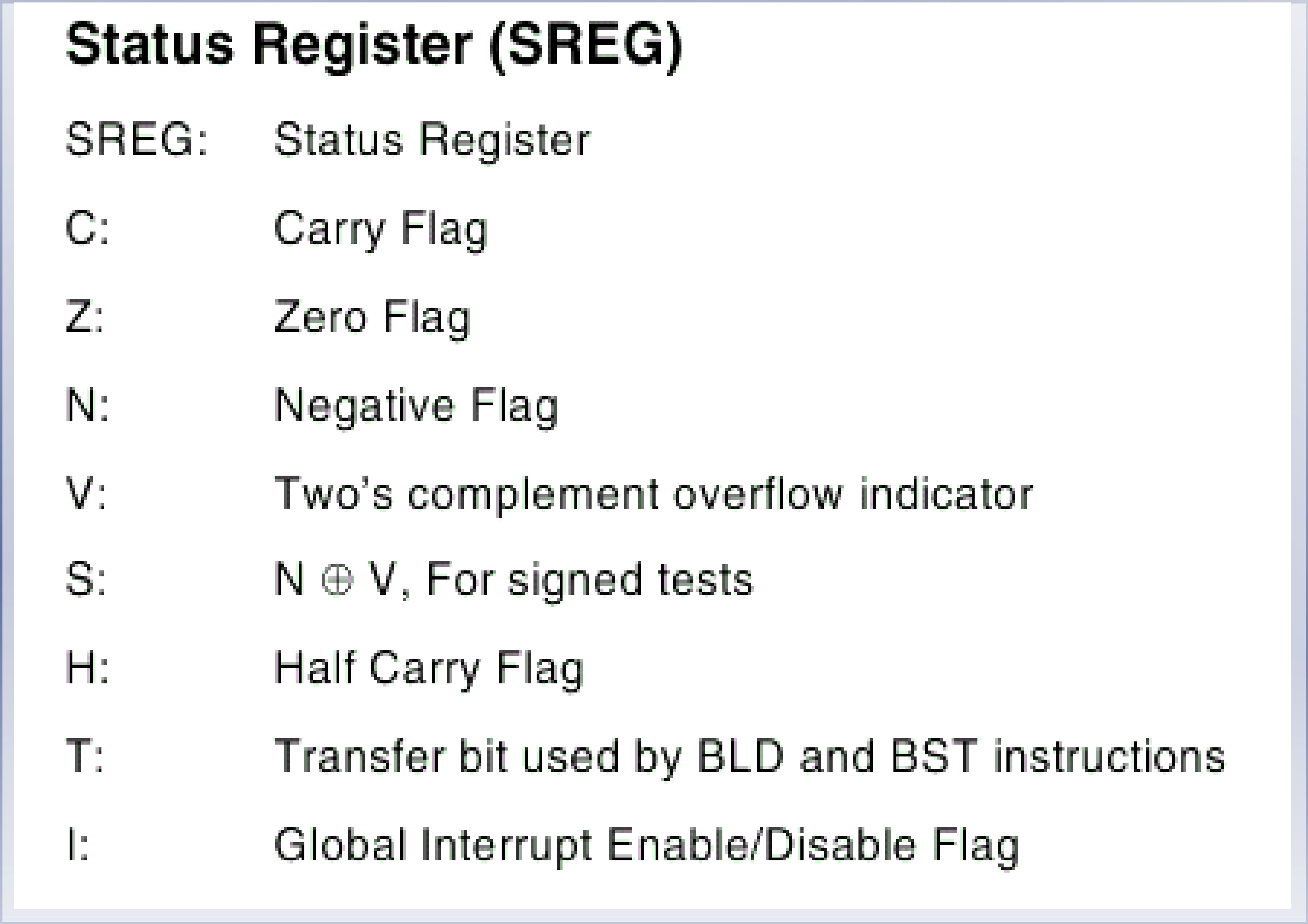
• Program Counter (PC)	[16 bit]
• Status Register (SREG)	[8 bit]
Sack Pointer (SP)	[16 bit] [SPH, SPL]
• General Purpose Register (R0 – R32)	[8 bit]
• X , Y , Z Register	[16 bit]

status register



	7	6	5	4	3	2	1	0
SREG	I	T	H	S	V	N	Z	C
	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
	0	0	0	0	0	0	0	0

- The micro controller operates based on the Status Register (SREG) and other internal registers or components. Most important is the Status Register which holds information on the last instruction and its result and Interrupt enable status.
- The SREG holds 8 Flags:



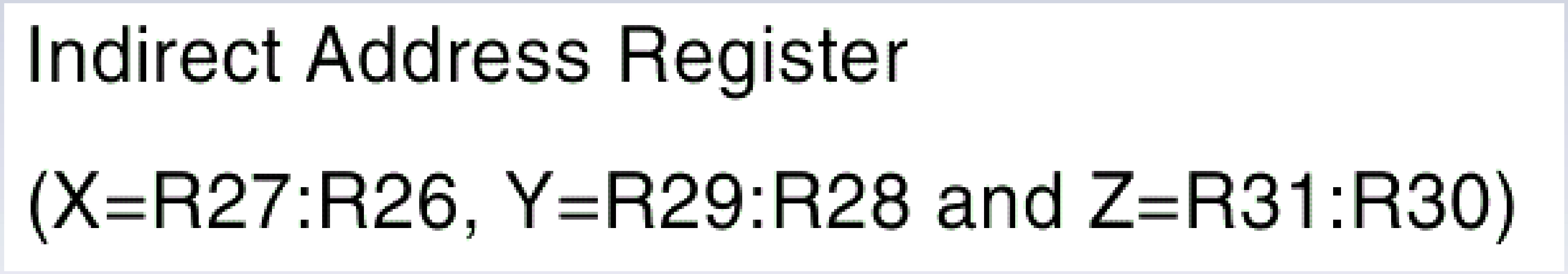
Status Register (SREG)

SREG:	Status Register
C:	Carry Flag
Z:	Zero Flag
N:	Negative Flag
V:	Two's complement overflow indicator
S:	$N \oplus V$, For signed tests
H:	Half Carry Flag
T:	Transfer bit used by BLD and BST instructions
I:	Global Interrupt Enable/Disable Flag

x, y and z registers

- The register r28 -r31 have some additional function to their general purpose usage.

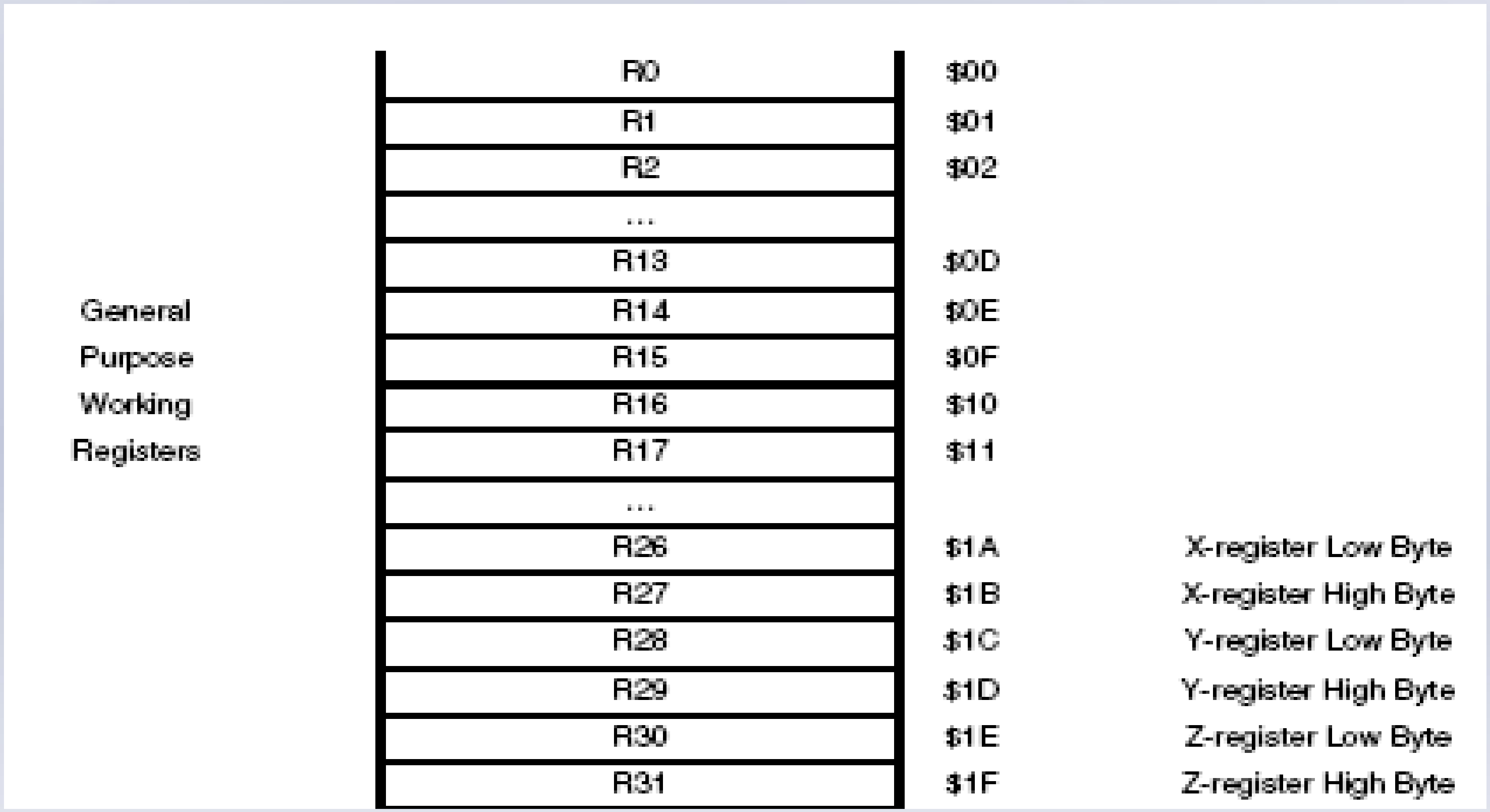
- These register are 16 bit address pointer for indirect addressing of the data space .



Indirect Address Register

(X=R27:R26, Y=R29:R28 and Z=R31:R30)

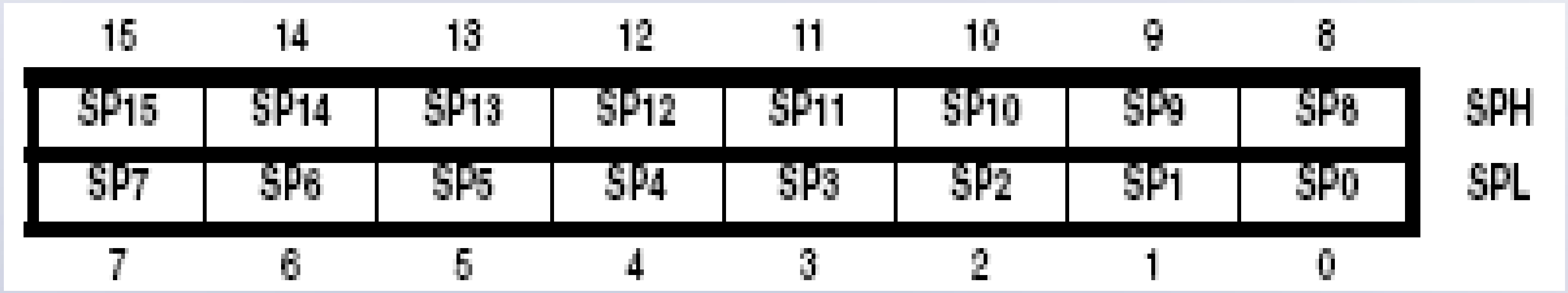
General purpose register file



General Purpose Working Registers

R0	$00	
R1	$01	
R2	$02	
...		
R13	$0D	
R14	$0E	
R15	$0F	
R16	$10	
R17	$11	
...		
R26	$1A	X-register Low Byte
R27	$1B	X-register High Byte
R28	$1C	Y-register Low Byte
R29	$1D	Y-register High Byte
R30	$1E	Z-register Low Byte
R31	$1F	Z-register High Byte

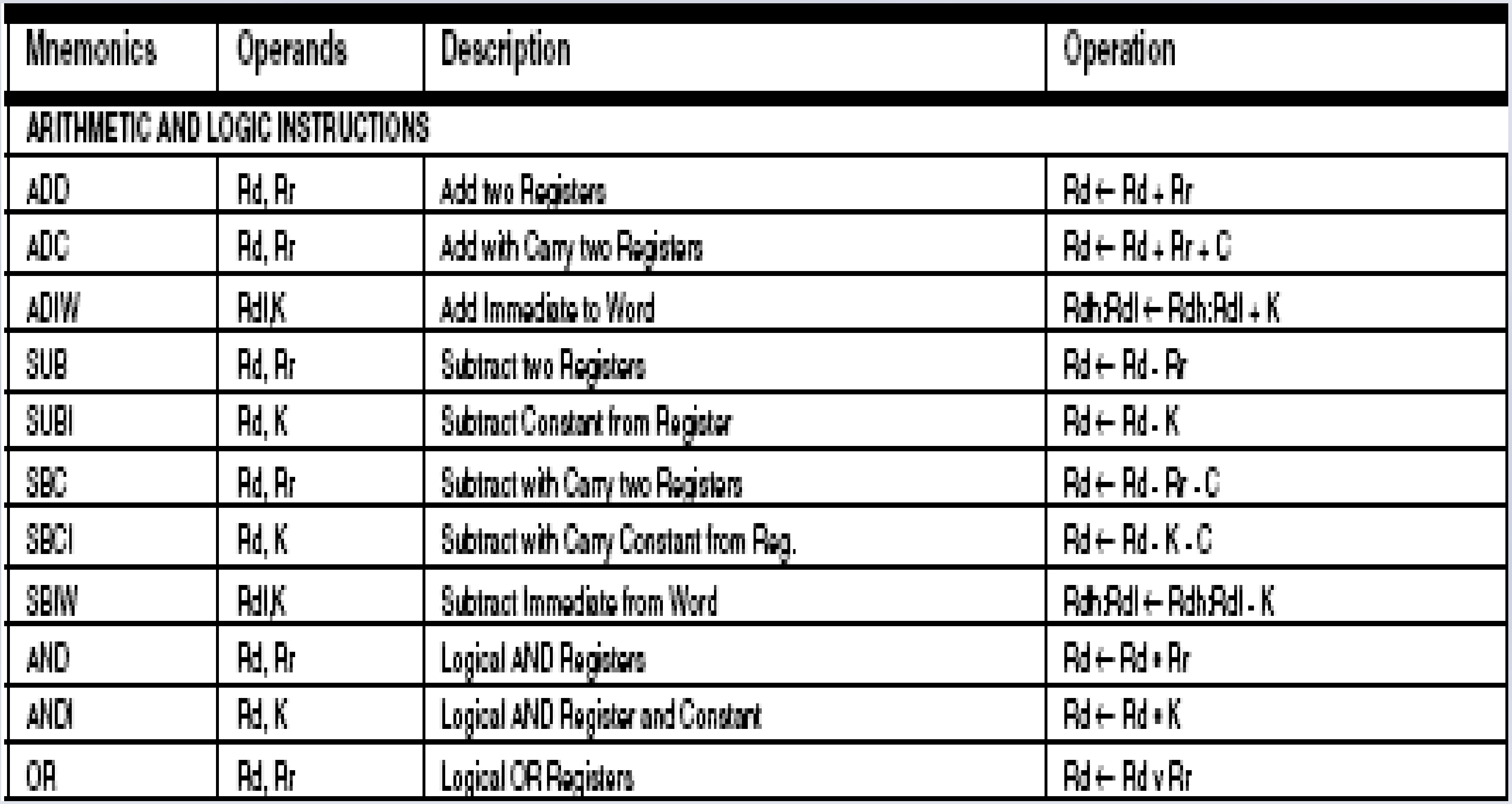
Stack Pointer



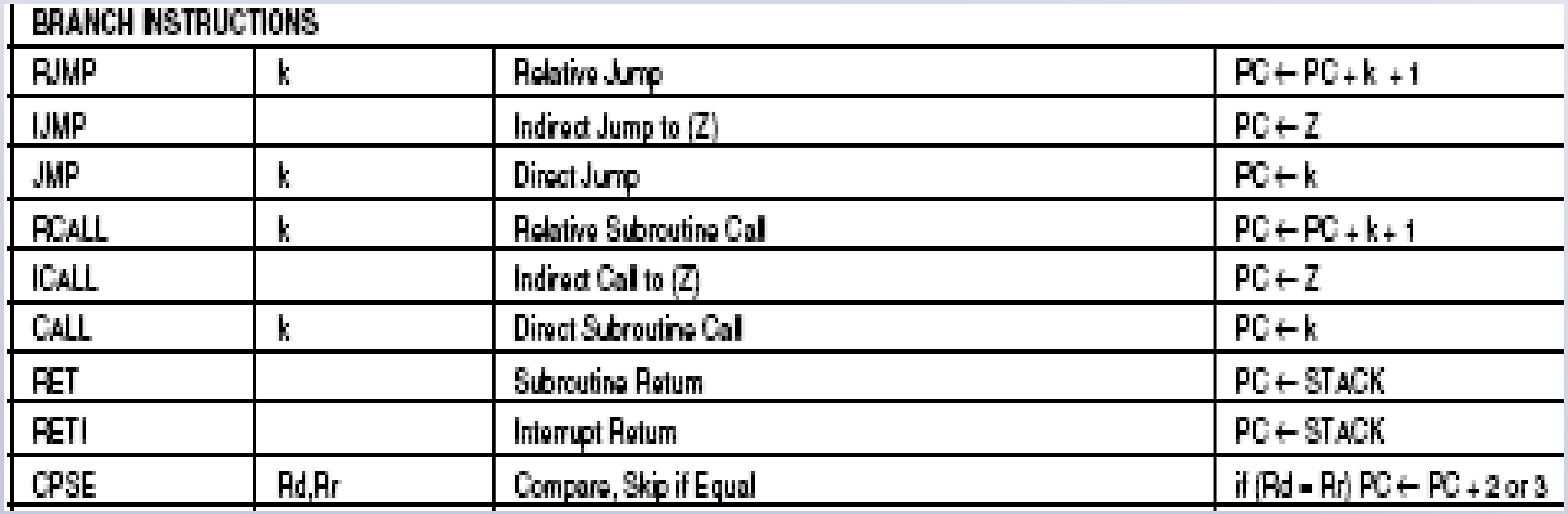
15	14	13	12	11	10	9	8	
SP15	SP14	SP13	SP12	SP11	SP10	SP9	SP8	SPH
SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0	SPL
7	6	5	4	3	2	1	0	

- The Stack is mainly used for storing temporary data, for storing local variables and for storing return addresses after interrupts and subroutine calls.
- The Stack Pointer Register always points to the top of the Stack.
- The Stack is used by the ALU to store return addresses from subroutines.

Avr instruction set



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



BRANCH INSTRUCTIONS

RJMP	k	Relative Jump	PC ← PC + k + 1
IJMP		Indirect Jump to (Z)	PC ← Z
JMP	k	Direct Jump	PC ← k
RCALL	k	Relative Subroutine Call	PC ← PC + k + 1
ICALL		Indirect Call to (Z)	PC ← Z
CALL	k	Direct Subroutine Call	PC ← k
RET		Subroutine Return	PC ← STACK
RETI		Interrupt Return	PC ← STACK
CPSE	Rd,Rr	Compare, Skip if Equal	if (Rd = Rr) PC ← PC + 2 or 3

Branch instruction.....

- The advantage of rjmp over jmp is that rjmp only needs 1 word of code space, while jmp needs 2 words. Example:rjmp go_here

- rjmp:

  "Relative Jump". This instruction performs a jump within a range of +/- 2k words. Added together, it can reach 4k words or 8k bytes of program memory

Contd…

- ijmp
- "Indirect Jump" to (Z). This instruction performs a jump to the address pointed to by the Z index register pair. As Z is 16 bits wide, ijmp allows jumps within the lower 64k words range of code space (big enough for a mega128)
- Example:

```
 ldi ZL, low(go_there)
 ldi ZH, high(go_there)
 ijmp
```

Contd..

- Jmp

- "Jump". While rjmp is limited to +/- 2k words, jmp can be used to jump anywhere within the code space. The address operand of jmp can be as big as 22 bits, resulting in jumps of up to 4M words. The disadvantage over rjmp is that jmp needs 2 words of code space, while rjmp needs just one word.

- Example: jmp go_far

subroutines

- icall :"Indirect Call to (Z)". This instruction works similar to ijmp, but as a subroutine call.
- rcall :"Relative Call Subroutine". Just as rjmp, rcall can reach addresses within +/- 2k words. When rcall is executed, the return address is pushed onto the stack.
- call

conditional branches

- Conditonal branches are branches based on the micro's Status Register. If the result of a previous operation left a status (for example "Zero"), this can be used to jump to code handling this result. Loops (for, while...) make use of this.

```
        ldi r16, 5          ;load 5 inreg
    loop:dec r16                    ;
        brne loop                   ;branch if not
equal
```

Lab

- avr-as -D –gstabs -ahlms -mmcu=atmega8 -o file.o file.s
- avr-ld -Aatmega8 -o file file.o
- simulavr -d atmega8 -g &
- avr-gdb file

Lab..