# MC404

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

2010

Prof. Paulo Cesar Centoducatte
Prof. Mario Lúcio Côrtes
Prof. Ricardo Pannain

# MC404

# ORGANIZAÇÃO BÁSICA DE COMPUTADORES E LINGUAGEM DE MONTAGEM

## "Organização de Memória

## e

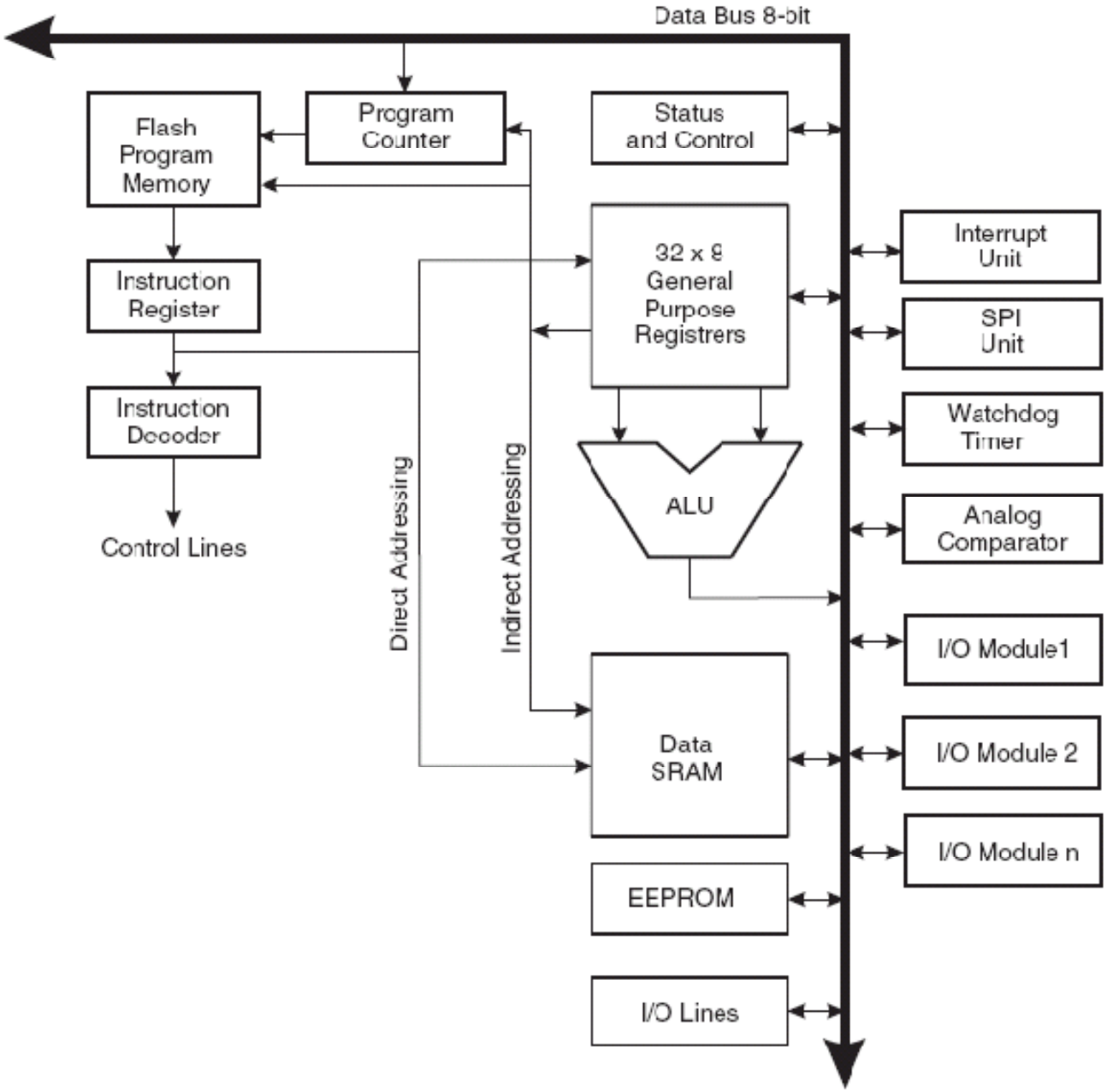
## Modos de Endereçamento"

# Organização de Memória e Modos de Endereçamento
## Sumário

- Organização de Memória do Atmega88
  - Memória de Programa – Flash
  - Memória de Dados
    - SRAM
    - EEPROM
- Modos de Endereçamento

**Table 2-1.** Memory Size Summary

| Device | Flash | EEPROM | RAM | Interrupt Vector Size |
|---|---|---|---|---|
| ATmega48 | 4K Bytes | 256 Bytes | 512 Bytes | 1 instruction word/vector |
| ATmega88 | 8K Bytes | 512 Bytes | 1K Bytes | 1 instruction word/vector |
| ATmega168 | 16K Bytes | 512 Bytes | 1K Bytes | 2 instruction words/vector |


Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Instruction Register
32 x 8 General Purpose Registrers
Instruction Decoder
ALU
Control Lines
Direct Addressing
Indirect Addressing
Interrupt Unit
SPI Unit
Watchdog Timer
Analog Comparator
I/O Module1
I/O Module 2
I/O Module n
Data SRAM
EEPROM
I/O Lines

# The AVR Status Register

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | 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:**
  - Habilita a ocorrência de interrupções. O controle de cada interrupção é realizado por um grupo diferente de registradores, mas se este bit estiver desabilitado, todas as interrupções estarão desabilitadas.
- **Bit 6 - T: Bit Copy Storage**:
  - Operações de cópia de bits (BLD, BST) utilizam este bit como fonte ou destino.
- **Bit 5 - H: Half Carry Flag**:
  - Indica ocorrência de Half Carry (Carry do bit 3 para o 4) em algumas operações aritméticas ( útil em aritmética BCD).
- **Bit 4 - S: Sign Bit**:
  - Sempre é um "ou exclusivo' entre o Bit 2 e Bit 3.
- **Bit 3 - V: Two's Complement Overow Flag**:
  - Indica overflow em operações aritméticas com complemento a 2.
- **Bit 2 - N: Negative Flag**:
  - Indica um resultado negativo em uma operação lógica/aritmética.
- **Bit 1 - Z: Zero Flag**:
  - Indica um resultado igual a zero em uma operação lógica/aritmética.
- **Bit 0 - C: Carry Flag**:
  - Indica ocorrência de carry em uma operação lógica/aritmética.

**Figure 4-2.** AVR CPU General Purpose Working Registers

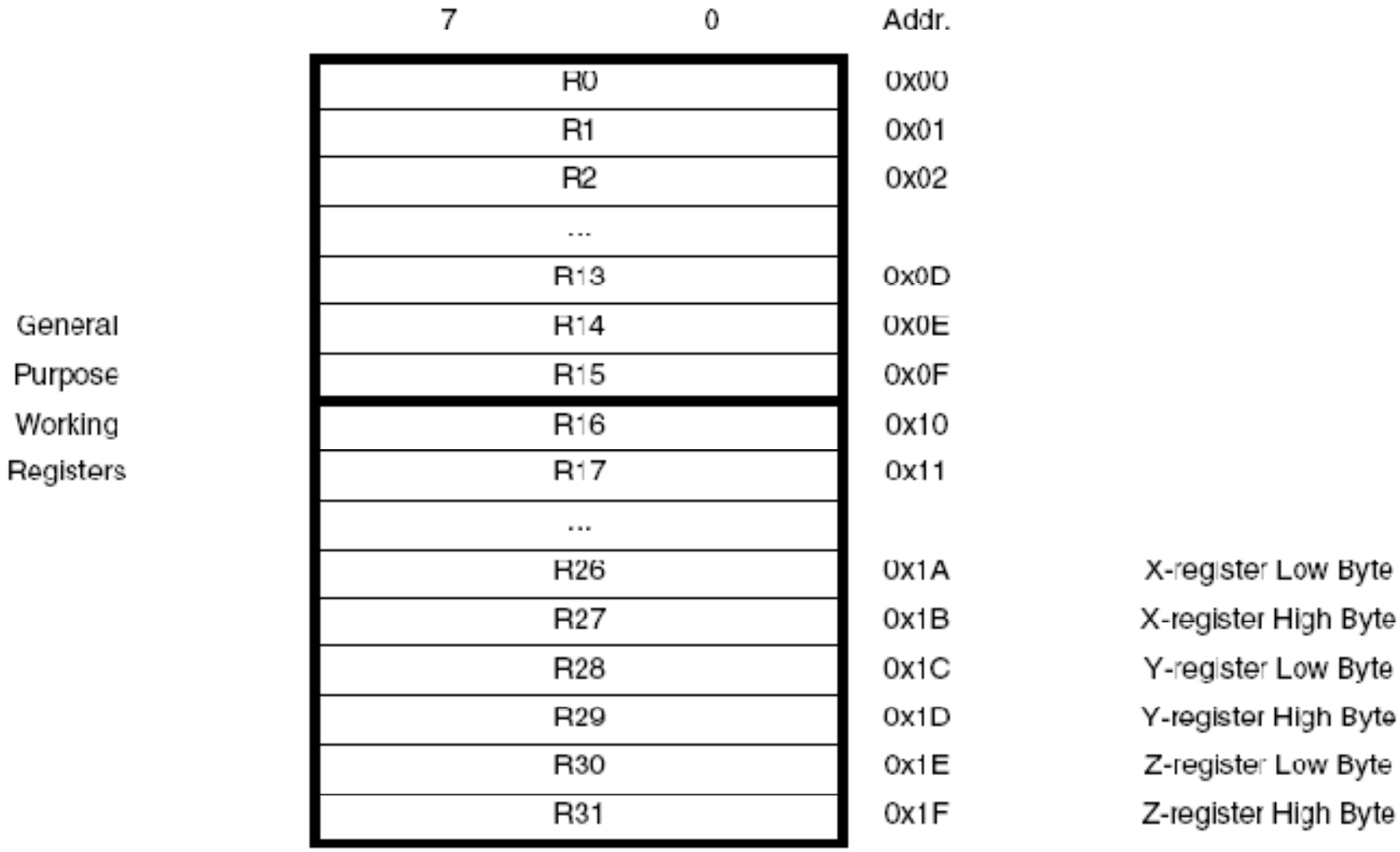
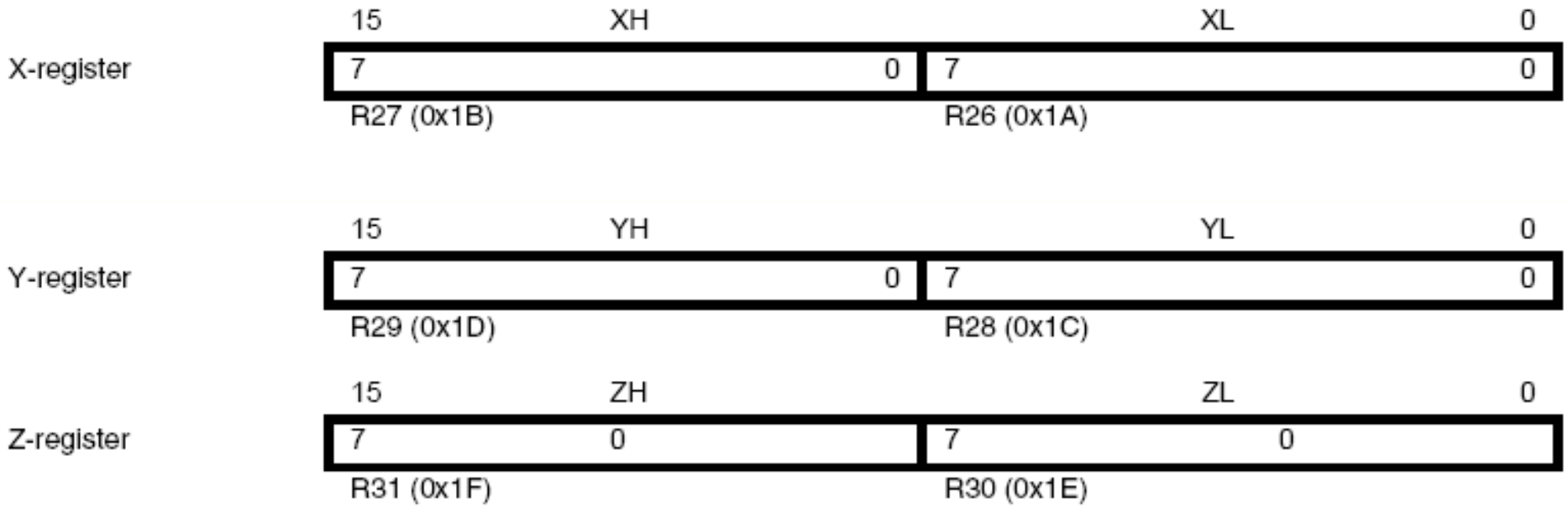


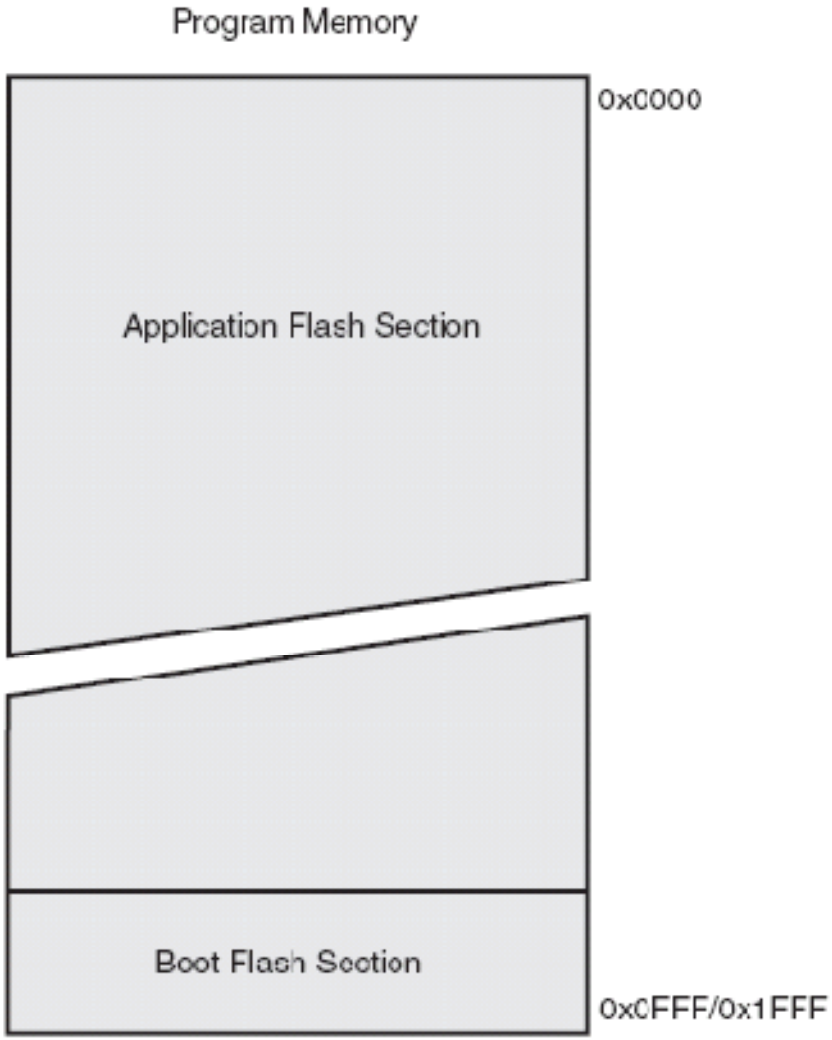
**Figure 4-3.** The X-, Y-, and Z-registers



# AVR ATmega48/88/168 Memories

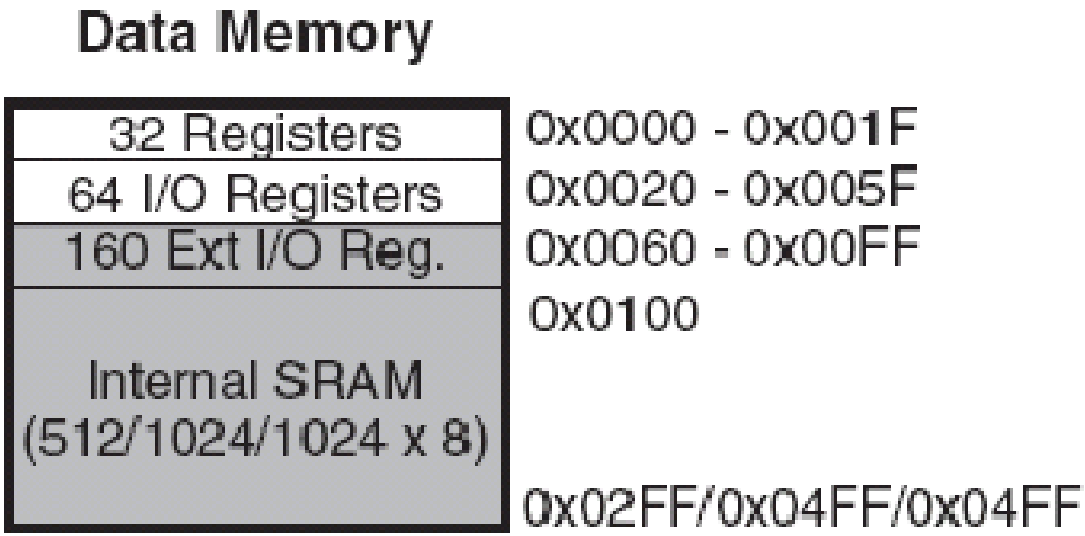
## In-System Reprogrammable Flash Program Memory

**Figure 5-2.** Program Memory Map, ATmega88 and ATmega168



# SRAM Data Memory

Figure 5-3. Data Memory Map



# EEPROM Data Memory

## The EEPROM Address Register – EEARH and EEARL

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | – | – | – | – | – | EEAR8 | EEARH |
| | EEAR7 | EEAR6 | EEAR5 | EEAR4 | EEAR3 | EEAR2 | EEAR1 | EEAR0 | EEARL |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R | R | R | 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 | X | |
| | X | X | X | X | X | X | X | X | |

## The EEPROM Data Register – EEDR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | MSB | | | | | | | LSB | EEDR |
| 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 | |

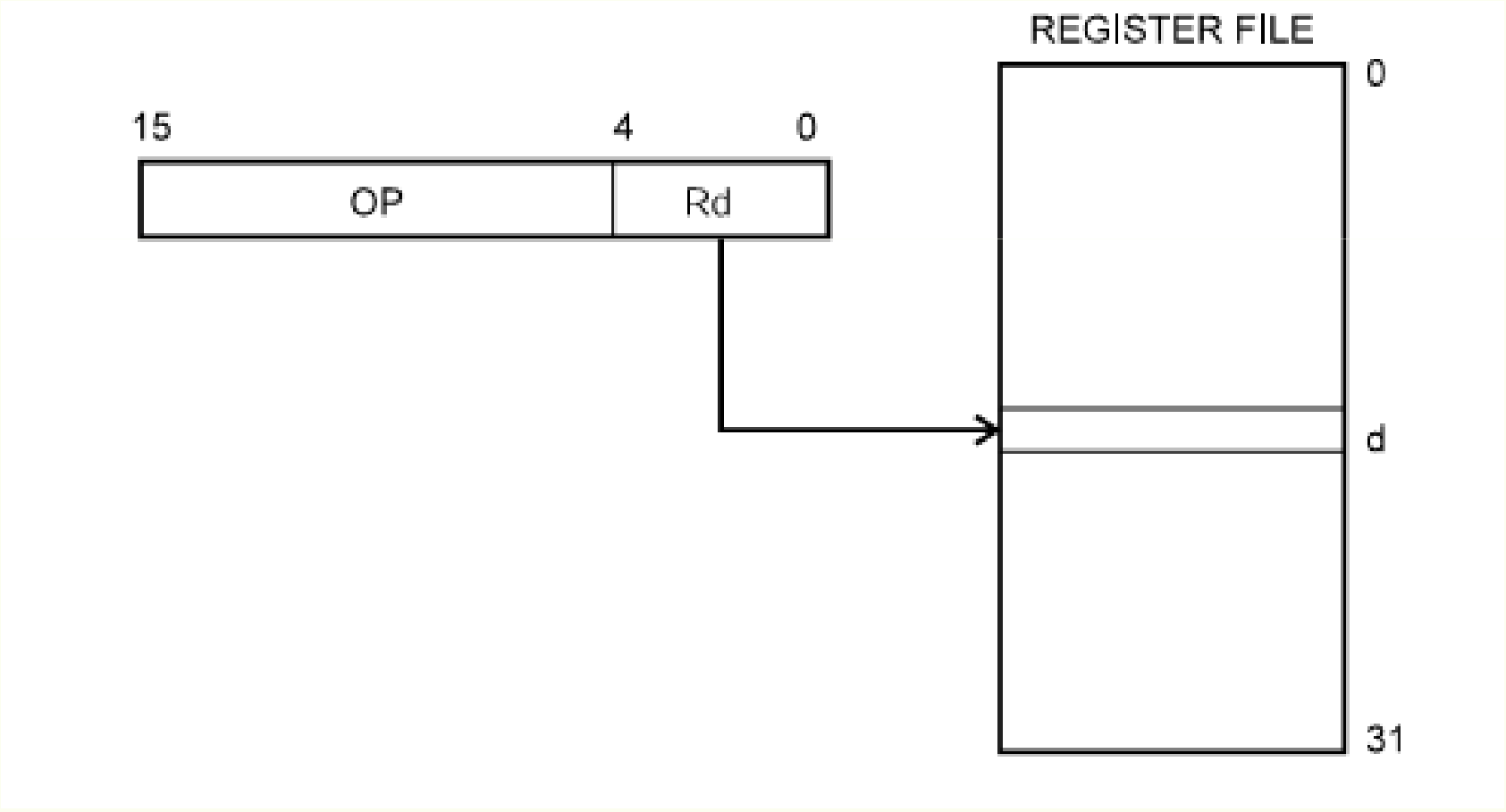
## The EEPROM Control Register – EECR

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | – | – | EEPM1 | EEPM0 | EERIE | EEMPE | EEPE | EERE | EECR |
| Read/Write | R | R | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | X | X | 0 | 0 | X | 0 | |

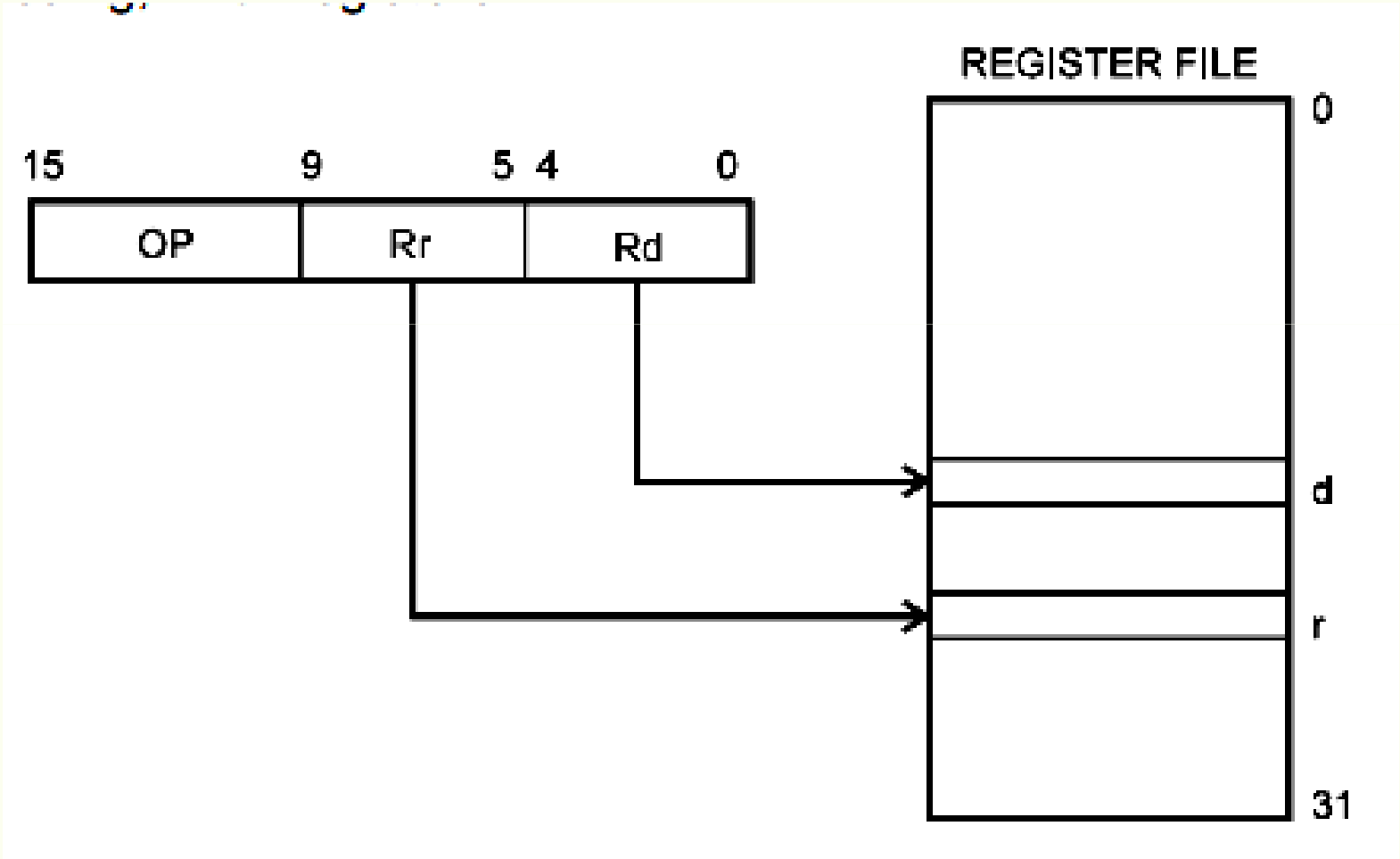
**Table 5-1.** EEPROM Mode Bits

| EEPM1 | EEPM0 | Programming Time | Operation |
|---|---|---|---|
| 0 | 0 | 3.4 ms | Erase and Write in one operation (Atomic Operation) |
| 0 | 1 | 1.8 ms | Erase Only |
| 1 | 0 | 1.8 ms | Write Only |
| 1 | 1 | – | Reserved for future use |

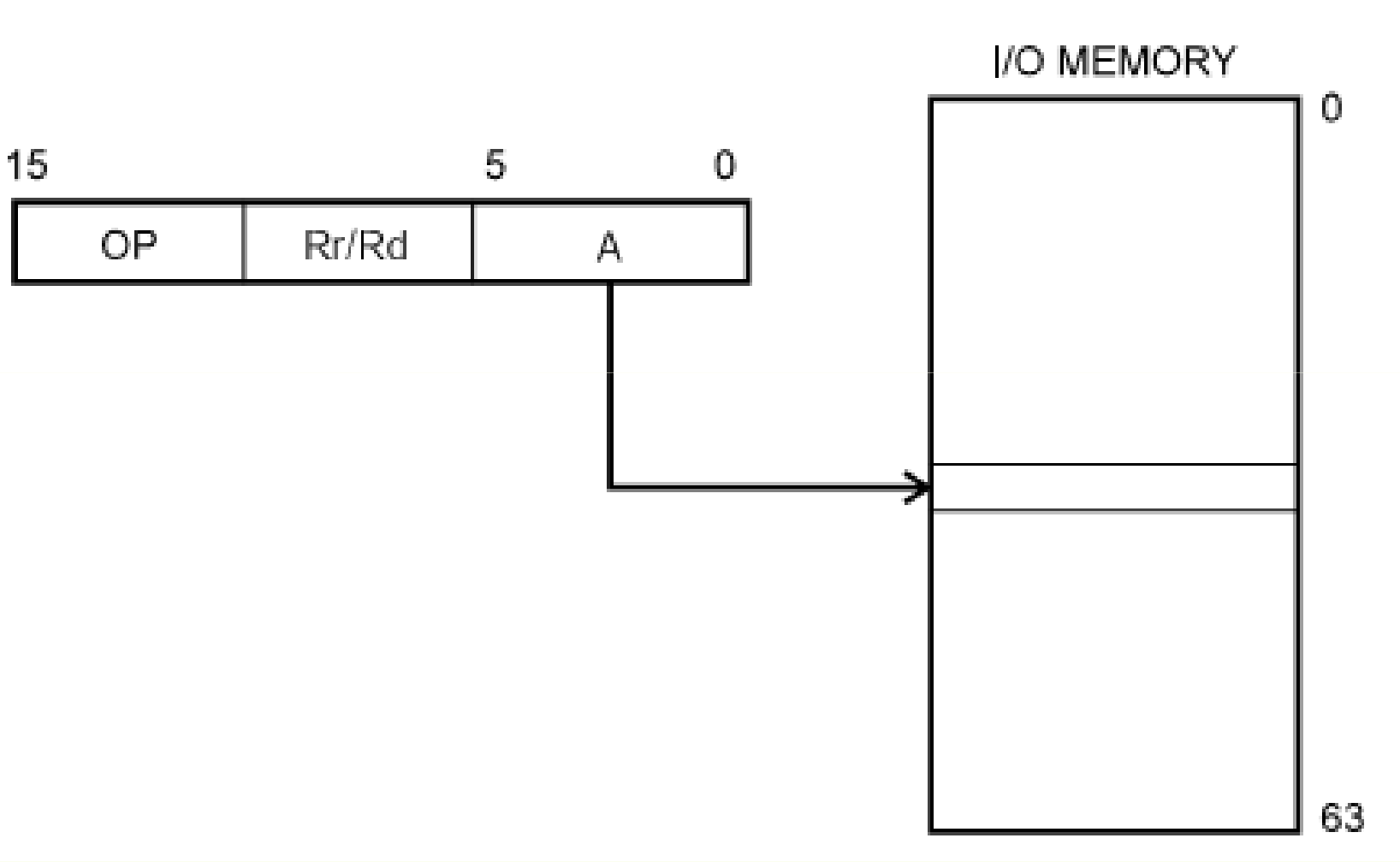
# Direct Single Register Addressing



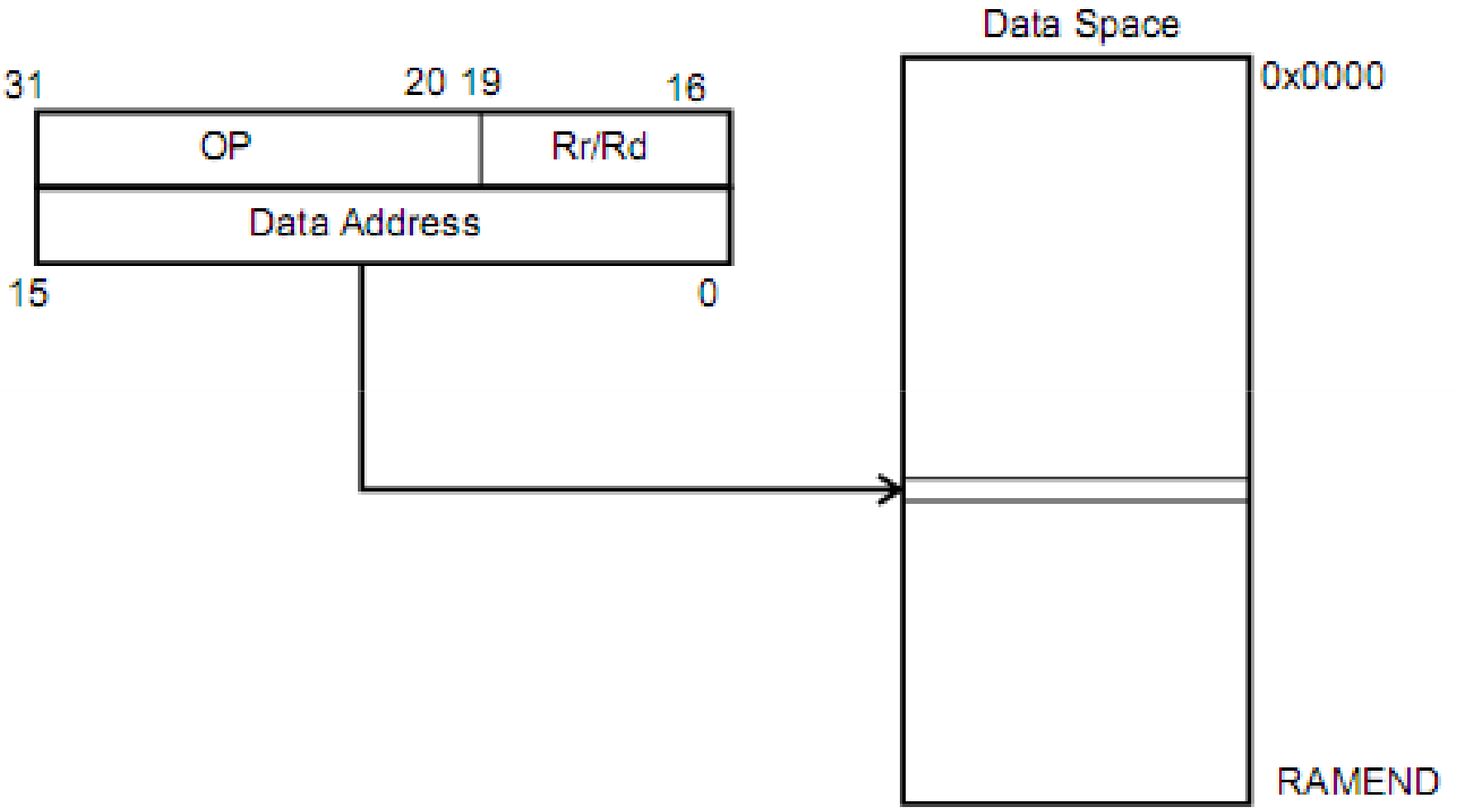
# Direct Register Addressing, Two Registers



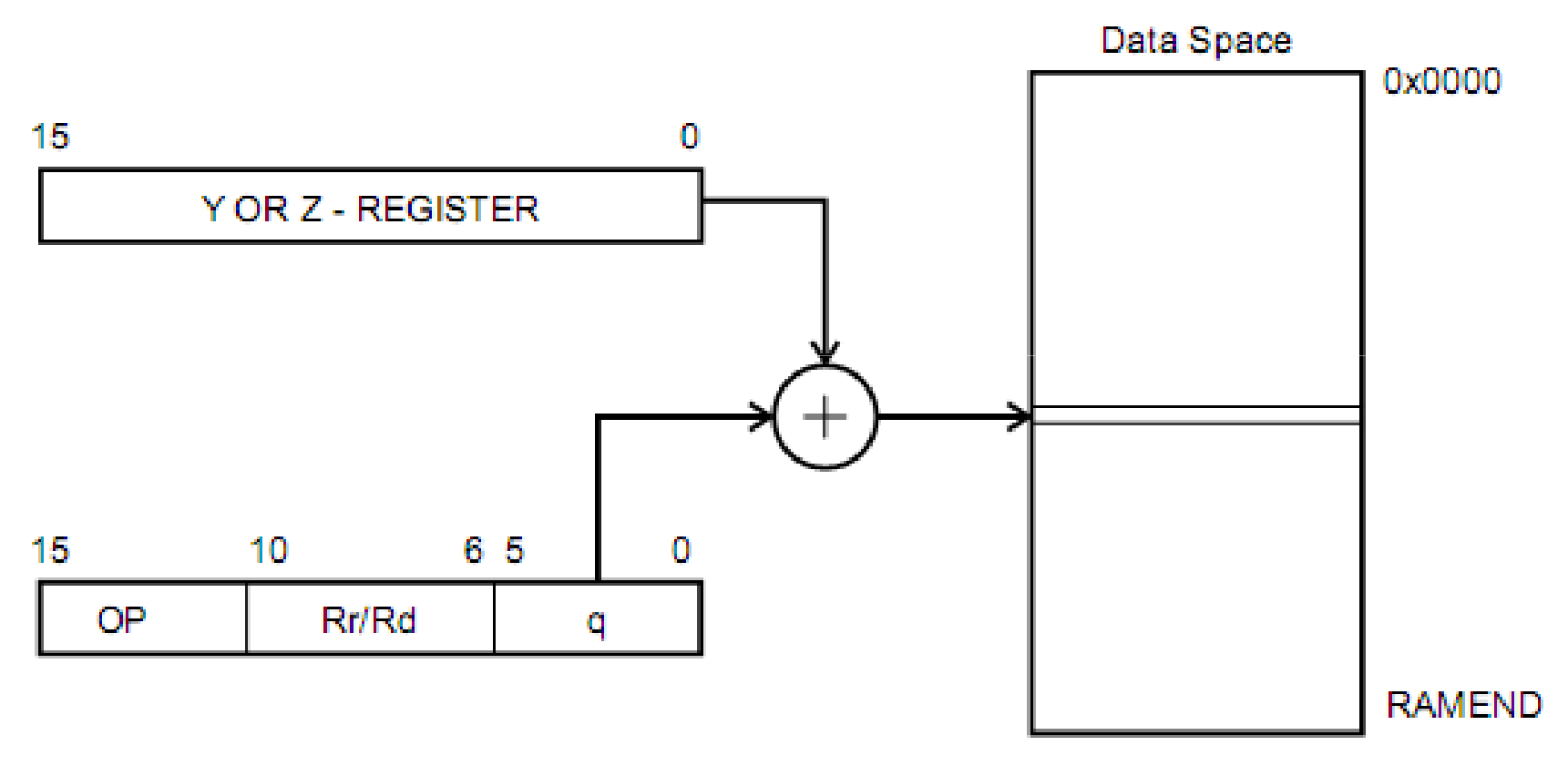
# I/O Direct Addressing



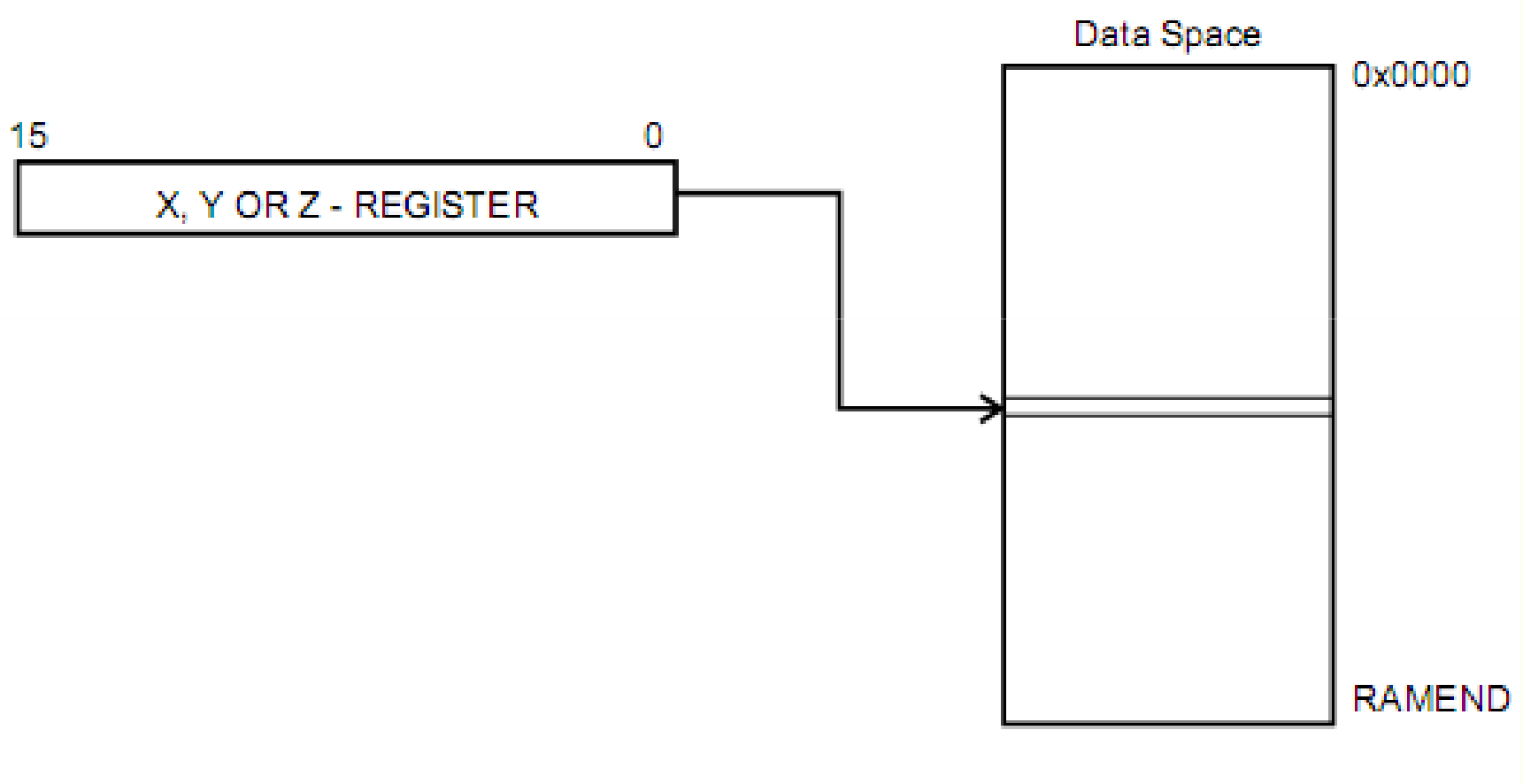
# Direct Data Addressing



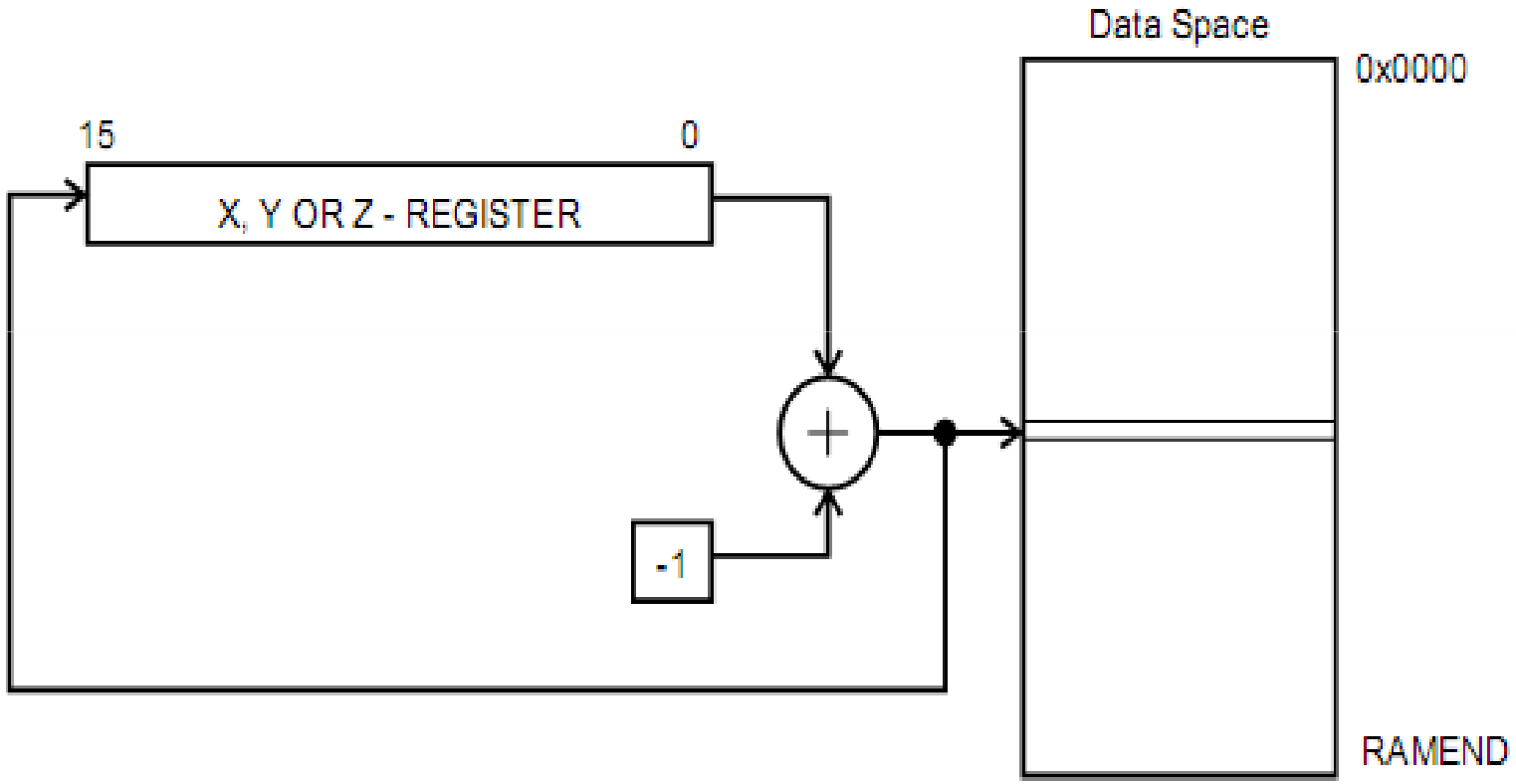
# Data Indirect with Displacement



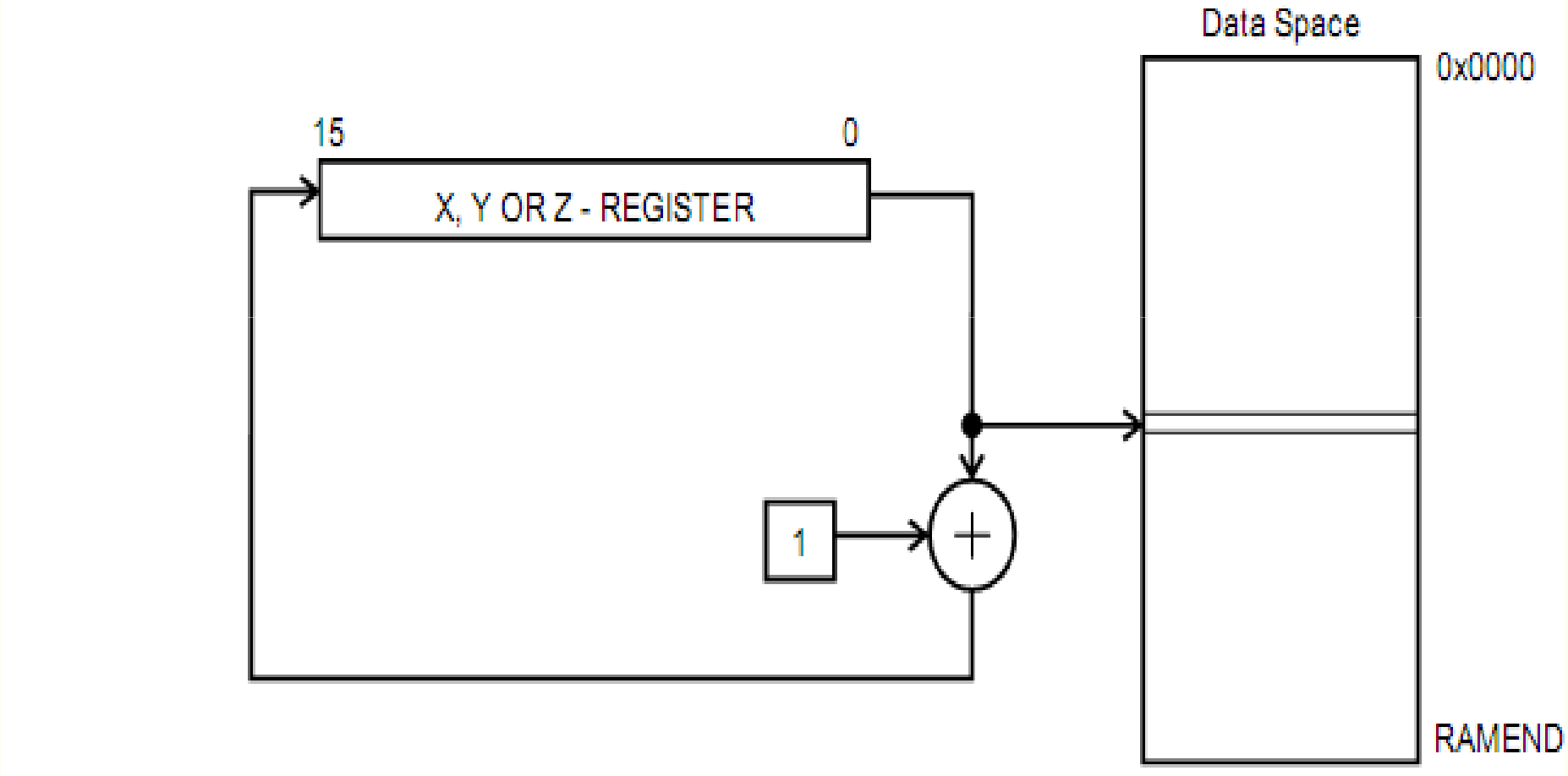
# Data Indirect Addressing



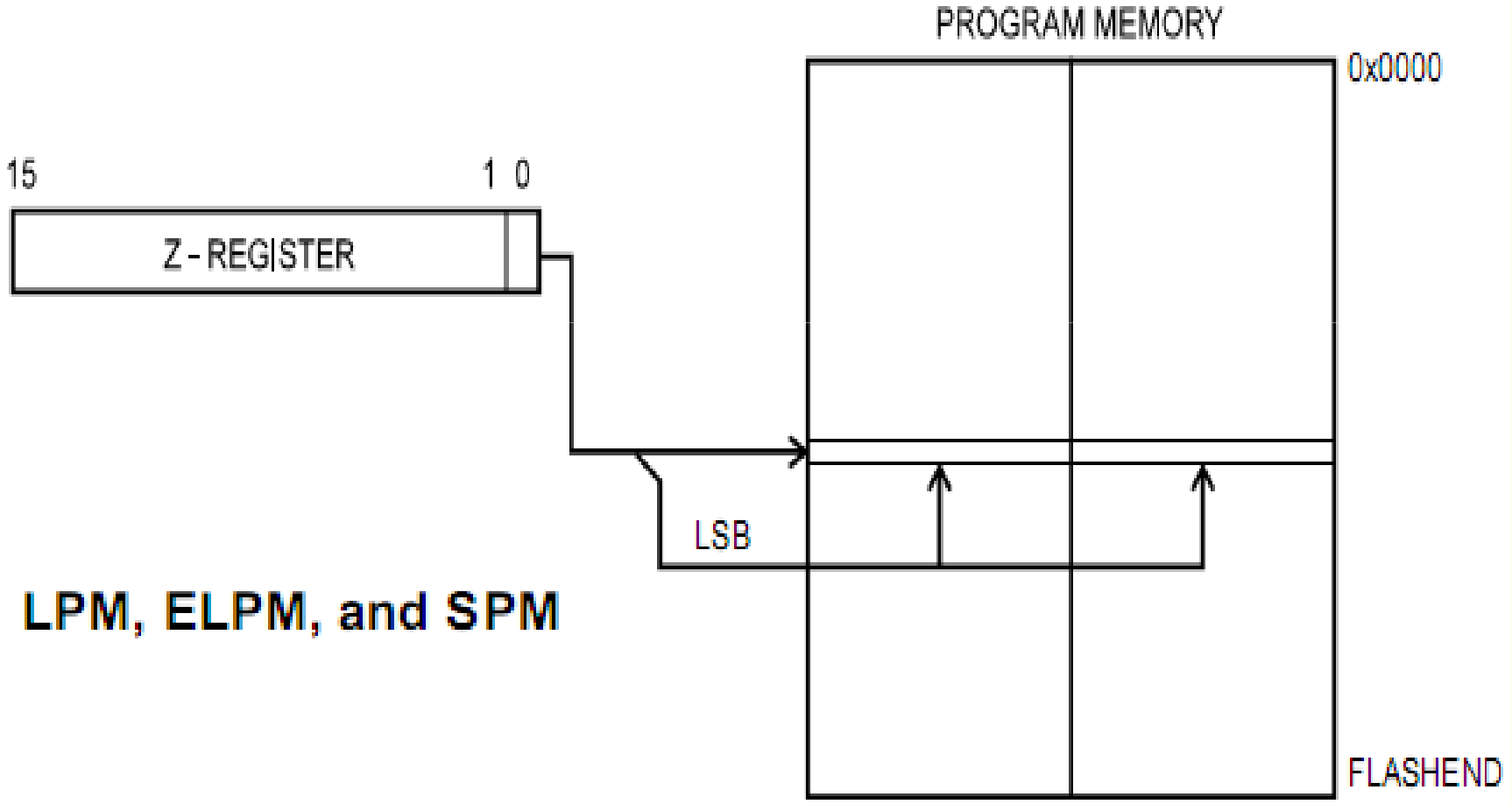
## Data Indirect Addressing with Pre-decrement



## Data Indirect Addressing with Post-increment

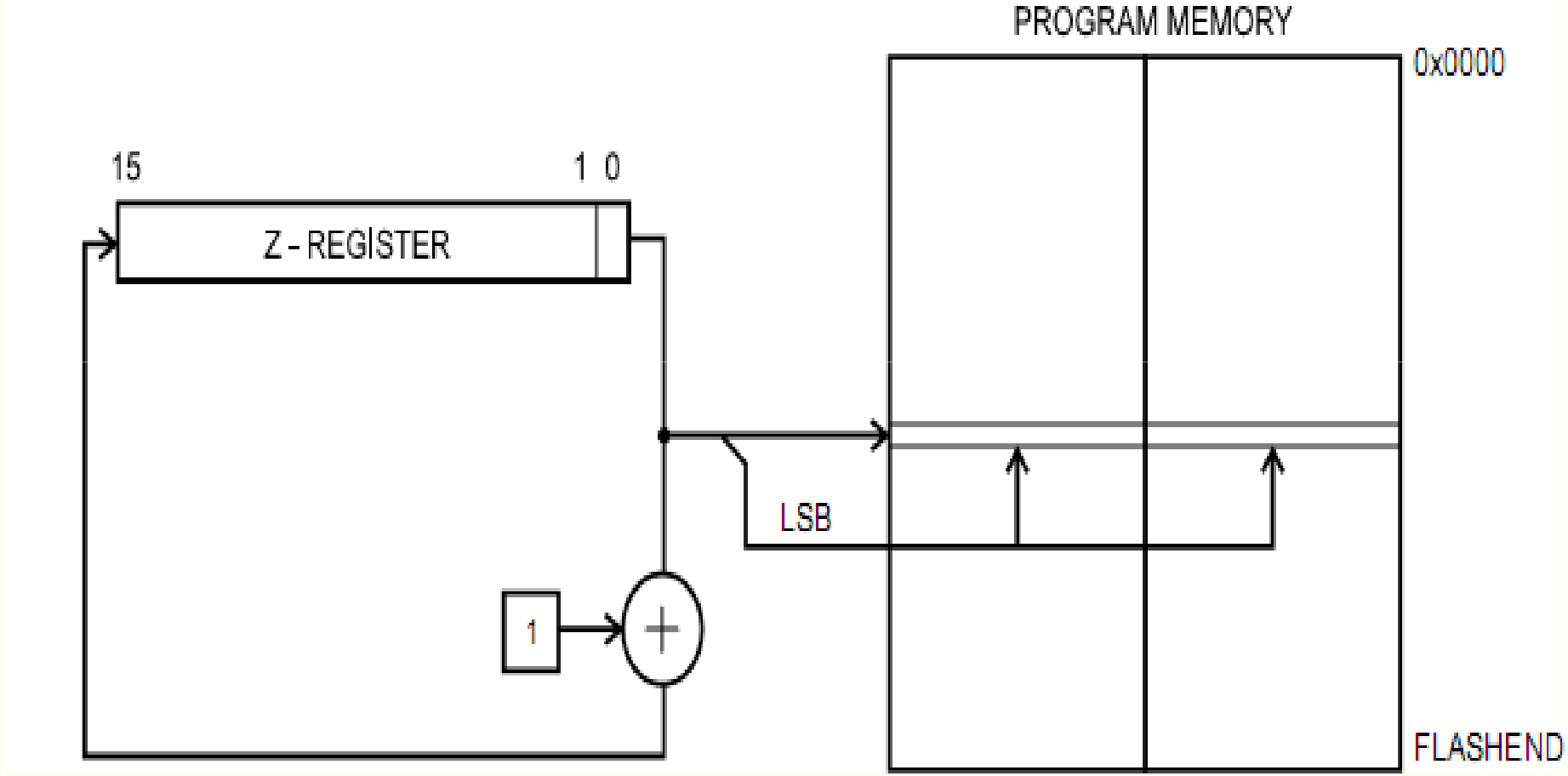


# Program Memory Constant Addressing



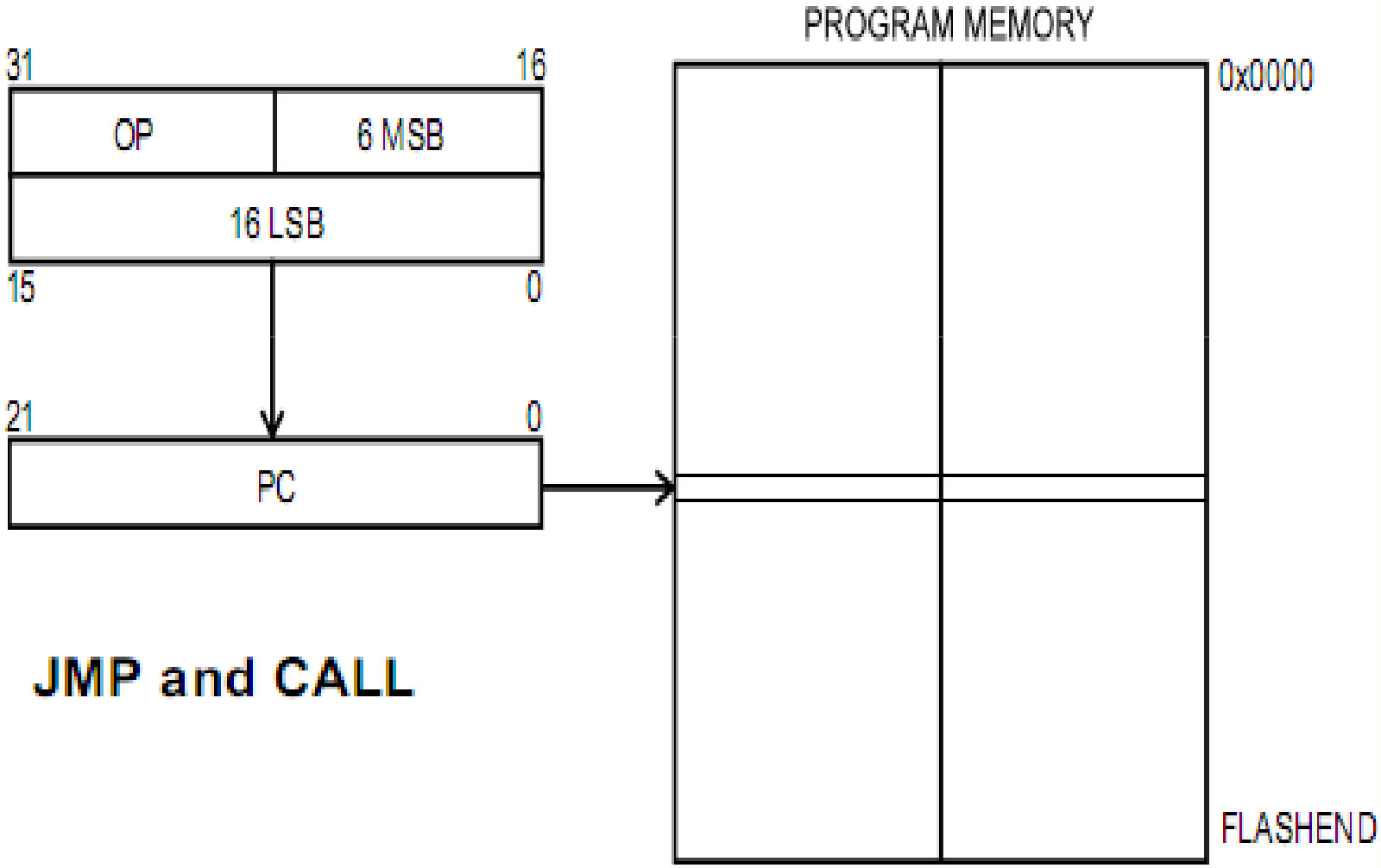

**LPM, ELPM, and SPM**

## Program Memory Addressing with Post-increment



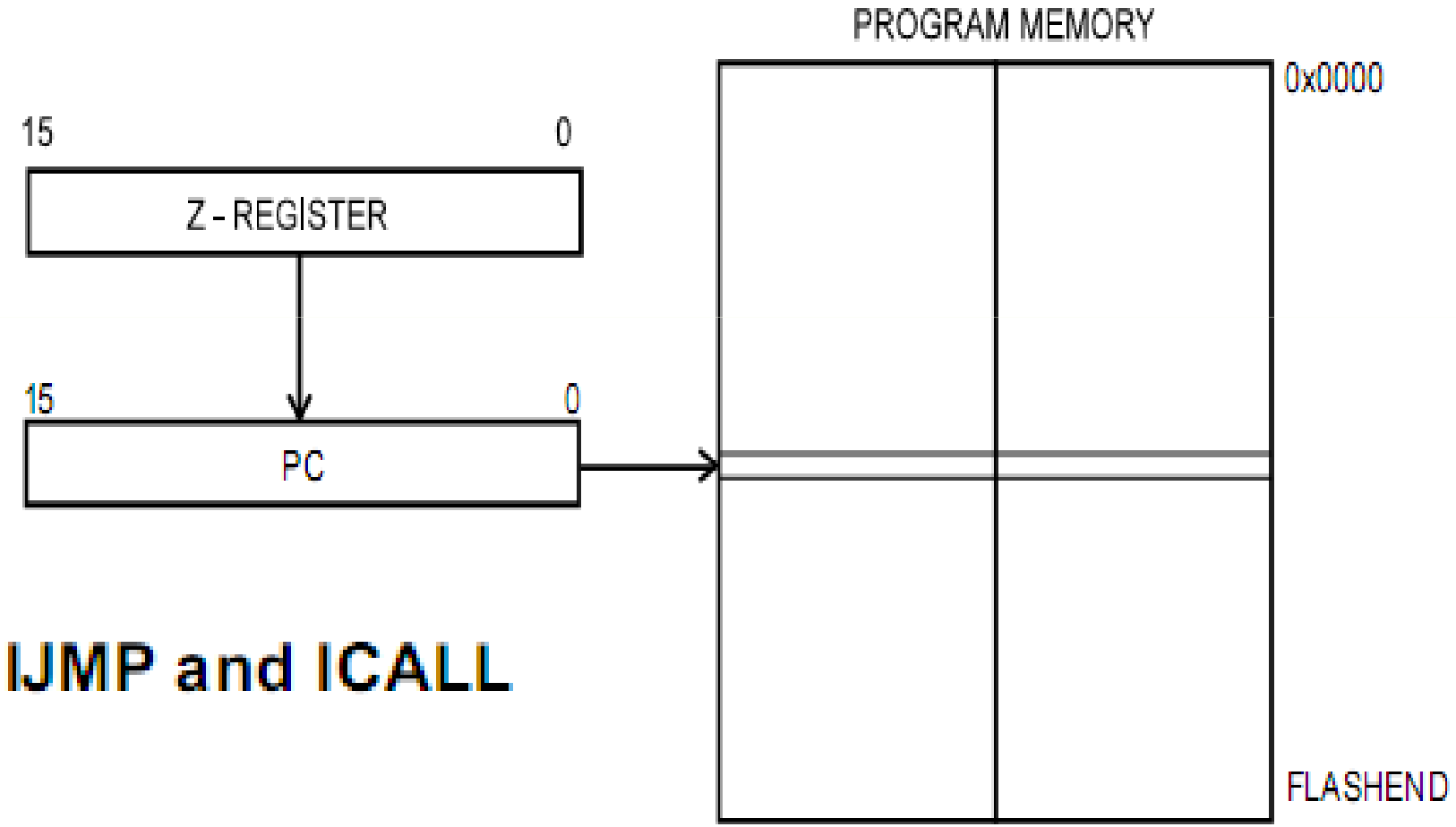

**LPM Z+ and ELPM Z+**

# Direct Program Memory Addressing



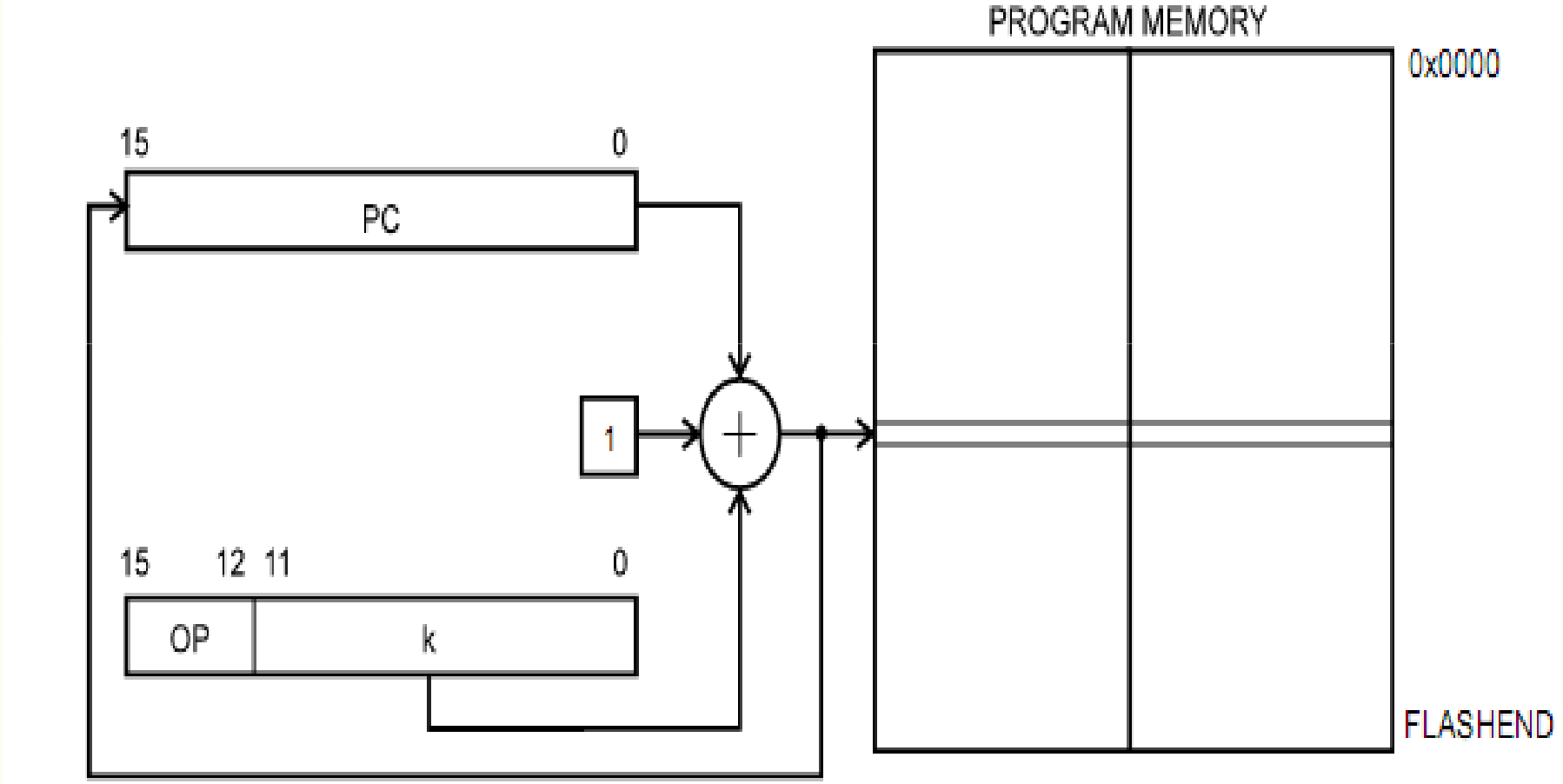

**JMP and CALL**

# Indirect Program Memory Addressing




**IJMP and ICALL**

## Relative Program Memory Addressing




**RJMP and RCALL**

# Conditional Branch Summary

| Test | Boolean | Mnemonic | Complementary | Boolean | Mnemonic | Comment |
|---|---|---|---|---|---|---|
| Rd > Rr | Z•(N ⊕ V) = 0 | BRLT(1) | Rd ≤ Rr | Z+(N ⊕ V) = 1 | BRGE* | Signed |
| Rd ≥ Rr | (N ⊕ V) = 0 | BRGE | Rd < Rr | (N ⊕ V) = 1 | BRLT | Signed |
| Rd = Rr | Z = 1 | BREQ | Rd ≠ Rr | Z = 0 | BRNE | Signed |
| Rd ≤ Rr | Z+(N ⊕ V) = 1 | BRGE(1) | Rd > Rr | Z•(N ⊕ V) = 0 | BRLT* | Signed |
| Rd < Rr | (N ⊕ V) = 1 | BRLT | Rd ≥ Rr | (N ⊕ V) = 0 | BRGE | Signed |
| Rd > Rr | C + Z = 0 | BRLO(1) | Rd ≤ Rr | C + Z = 1 | BRSH* | Unsigned |
| Rd ≥ Rr | C = 0 | BRSH/BRCC | Rd < Rr | C = 1 | BRLO/BRCS | Unsigned |
| Rd = Rr | Z = 1 | BREQ | Rd ≠ Rr | Z = 0 | BRNE | Unsigned |
| Rd ≤ Rr | C + Z = 1 | BRSH(1) | Rd > Rr | C + Z = 0 | BRLO* | Unsigned |
| Rd < Rr | C = 1 | BRLO/BRCS | Rd ≥ Rr | C = 0 | BRSH/BRCC | Unsigned |
| Carry | C = 1 | BRCS | No carry | C = 0 | BRCC | Simple |
| Negative | N = 1 | BRMI | Positive | N = 0 | BRPL | Simple |
| Overflow | V = 1 | BRVS | No overflow | V = 0 | BRVC | Simple |
| Zero | Z = 1 | BREQ | Not zero | Z = 0 | BRNE | Simple |

Note: 1. Interchange Rd and Rr in the operation before the test, i.e., CP Rd,Rr → CP Rr,Rd

## Instruction Set Summary

| Mnemonics | Operands | Description | Operation | Flags | #Clock Note |
|---|---|---|---|---|---|
| **Arithmetic and Logic Instructions** | | | | | |
| ADD | Rd, Rr | Add without Carry | Rd ← Rd + Rr | Z,C,N,V,S,H | 1 |
| ADC | Rd, Rr | Add with Carry | Rd ← Rd + Rr + C | Z,C,N,V,S,H | 1 |
| ADIW | Rd, K | Add Immediate to Word | Rd+1:Rd ← Rd+1:Rd + K | Z,C,N,V,S | 2 (1) |
| SUB | Rd, Rr | Subtract without Carry | Rd ← Rd - Rr | Z,C,N,V,S,H | 1 |
| SUBI | Rd, K | Subtract Immediate | Rd ← Rd - K | Z,C,N,V,S,H | 1 |
| SBC | Rd, Rr | Subtract with Carry | Rd ← Rd - Rr - C | Z,C,N,V,S,H | 1 |
| SBCI | Rd, K | Subtract Immediate with Carry | Rd ← Rd - K - C | Z,C,N,V,S,H | 1 |
| SBIW | Rd, K | Subtract Immediate from Word | Rd+1:Rd ← Rd+1:Rd - K | Z,C,N,V,S | 2 (1) |
| AND | Rd, Rr | Logical AND | Rd ← Rd • Rr | Z,N,V,S | 1 |
| ANDI | Rd, K | Logical AND with Immediate | Rd ← Rd • K | Z,N,V,S | 1 |
| OR | Rd, Rr | Logical OR | Rd ← Rd v Rr | Z,N,V,S | 1 |
| ORI | Rd, K | Logical OR with Immediate | Rd ← Rd v K | Z,N,V,S | 1 |
| EOR | Rd, Rr | Exclusive OR | Rd ← Rd ⊕ Rr | Z,N,V,S | 1 |
| COM | Rd | One's Complement | Rd ← $FF - Rd | Z,C,N,V,S | 1 |
| NEG | Rd | Two's Complement | Rd ← $00 - Rd | Z,C,N,V,S,H | 1 |
| SBR | Rd,K | Set Bit(s) in Register | Rd ← Rd v K | Z,N,V,S | 1 |
| CBR | Rd,K | Clear Bit(s) in Register | Rd ← Rd • ($FFh - K) | Z,N,V,S | 1 |
| INC | Rd | Increment | Rd ← Rd + 1 | Z,N,V,S | 1 |
| DEC | Rd | Decrement | Rd ← Rd - 1 | Z,N,V,S | 1 |
| TST | Rd | Test for Zero or Minus | Rd ← Rd • Rd | Z,N,V,S | 1 |
| CLR | Rd | Clear Register | Rd ← Rd ⊕ Rd | Z,N,V,S | 1 |
| SER | Rd | Set Register | Rd ← $FF | None | 1 |
| MUL | Rd,Rr | Multiply Unsigned | R1:R0 ← Rd × Rr (UU) | Z,C | 2 (1) |
| MULS | Rd,Rr | Multiply Signed | R1:R0 ← Rd × Rr (SS) | Z,C | 2 (1) |
| MULSU | Rd,Rr | Multiply Signed with Unsigned | R1:R0 ← Rd × Rr (SU) | Z,C | 2 (1) |
| FMUL | Rd,Rr | Fractional Multiply Unsigned | R1:R0 ← (Rd × Rr)<<1 (UU) | Z,C | 2 (1) |
| FMULS | Rd,Rr | Fractional Multiply Signed | R1:R0 ← (Rd × Rr)<<1 (SS) | Z,C | 2 (1) |
| FMULSU | Rd,Rr | Fractional Multiply Signed with Unsigned | R1:R0 ← (Rd × Rr)<<1 (SU) | Z,C | 2 (1) |

| Branch Instructions | | | | | |
|---|---|---|---|---|---|
| RJMP | k | Relative Jump | PC ← PC + k + 1 | None | 2 |
| IJMP | | Indirect Jump to (Z) | PC(15:0) ← Z, PC(21:16) ← 0 | None | 2 (1) |
| EIJMP | | Extended Indirect Jump to (Z) | PC(15:0) ← Z, PC(21:16) ← EIND | None | 2 (1) |
| JMP | k | Jump | PC ← k | None | 3 (1) |
| RCALL | k | Relative Call Subroutine | PC ← PC + k + 1 | None | 3 / 4 (4) |
| ICALL | | Indirect Call to (Z) | PC(15:0) ← Z, PC(21:16) ← 0 | None | 3 / 4 (1)(4) |
| EICALL | | Extended Indirect Call to (Z) | PC(15:0) ← Z, PC(21:16) ← EIND | None | 4 (1)(4) |
| CALL | k | Call Subroutine | PC ← k | None | 4 / 5 (1)(4) |
| RET | | Subroutine Return | PC ← STACK | None | 4 / 5 (4) |
| RETI | | Interrupt Return | PC ← STACK | I | 4 / 5 (4) |
| CPSE | Rd,Rr | Compare, Skip if Equal | if (Rd = Rr) PC ← PC + 2 or 3 | None | 1 / 2 / 3 |
| CP | Rd,Rr | Compare | Rd - Rr | Z,C,N,V,S,H | 1 |
| CPC | Rd,Rr | Compare with Carry | Rd - Rr - C | Z,C,N,V,S,H | 1 |
| CPI | Rd,K | Compare with Immediate | Rd - K | Z,C,N,V,S,H | 1 |
| SBRC | Rr, b | Skip if Bit in Register Cleared | if (Rr(b)=0) PC ← PC + 2 or 3 | None | 1 / 2 / 3 |
| SBRS | Rr, b | Skip if Bit in Register Set | if (Rr(b)=1) PC ← PC + 2 or 3 | None | 1 / 2 / 3 |
| SBIC | A, b | Skip if Bit in I/O Register Cleared | if(I/O(A,b)=0) PC ← PC + 2 or 3 | None | 1 / 2 / 3 |
| SBIS | A, b | Skip if Bit in I/O Register Set | If(I/O(A,b)=1) PC ← PC + 2 or 3 | None | 1 / 2 / 3 |
| BRBS | s, k | Branch if Status Flag Set | if (SREG(s) = 1) then PC ←PC+k + 1 | None | 1 / 2 |
| BRBC | s, k | Branch if Status Flag Cleared | if (SREG(s) = 0) then PC ←PC+k + 1 | None | 1 / 2 |
| BREQ | k | Branch if Equal | if (Z = 1) then PC ← PC + k + 1 | None | 1 / 2 |
| BRNE | k | Branch if Not Equal | if (Z = 0) then PC ← PC + k + 1 | None | 1 / 2 |
| BRCS | k | Branch if Carry Set | if (C = 1) then PC ← PC + k + 1 | None | 1 / 2 |
| BRCC | k | Branch if Carry Cleared | if (C = 0) then PC ← PC + k + 1 | None | 1 / 2 |
| BRSH | k | Branch if Same or Higher | if (C = 0) then PC ← PC + k + 1 | None | 1 / 2 |
| BRLO | k | Branch if Lower | if (C = 1) then PC ← PC + k + 1 | None | 1 / 2 |
| BRMI | k | Branch if Minus | if (N = 1) then PC ← PC + k + 1 | None | 1 / 2 |

| | | | | | |
|---|---|---|---|---|---|
| BRPL | k | Branch if Plus | if (N = 0) then PC ← PC + k + 1 | None | 1 / 2 |
| BRGE | k | Branch if Greater or Equal, Signed | if (N ⊕ V= 0) then PC ← PC + k + 1 | None | 1 / 2 |
| BRLT | k | Branch if Less Than, Signed | if (N ⊕ V= 1) then PC ← PC + k + 1 | None | 1 / 2 |
| BRHS | k | Branch if Half Carry Flag Set | if (H = 1) then PC ← PC + k + 1 | None | 1 / 2 |
| BRHC | k | Branch if Half Carry Flag Cleared | if (H = 0) then PC ← PC + k + 1 | None | 1 / 2 |
| BRTS | k | Branch if T Flag Set | if (T = 1) then PC ← PC + k + 1 | None | 1 / 2 |
| BRTC | k | Branch if T Flag Cleared | if (T = 0) then PC ← PC + k + 1 | None | 1 / 2 |
| BRVS | k | Branch if Overflow Flag is Set | if (V = 1) then PC ← PC + k + 1 | None | 1 / 2 |
| BRVC | k | Branch if Overflow Flag is Cleared | if (V = 0) then PC ← PC + k + 1 | None | 1 / 2 |
| BRIE | k | Branch if Interrupt Enabled | if ( I = 1) then PC ← PC + k + 1 | None | 1 / 2 |
| BRID | k | Branch if Interrupt Disabled | if ( I = 0) then PC ← PC + k + 1 | None | 1 / 2 |

# ADC – Add with Carry

**Description:**

Adds two registers and the contents of the C Flag and places the result in the destination register Rd.

**Operation:**

(i) Rd ← Rd + Rr + C

| | Syntax: | Operands: | Program Counter: |
|---|---|---|---|
| (i) | ADC Rd,Rr | $0 \le d \le 31, 0 \le r \le 31$ | PC ← PC + 1 |

**16-bit Opcode:**

| | | | |
|---|---|---|---|
| 0001 | 11rd | dddd | rrrr |

**Status Register (SREG) Boolean Formula:**

| I | T | H | S | V | N | Z | C |
|---|---|---|---|---|---|---|---|
| – | – | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ |

# ADD – Add without Carry

**Description:**

Adds two registers without the C Flag and places the result in the destination register Rd.

**Operation:**

(i) Rd ← Rd + Rr

| | Syntax: | Operands: | Program Counter: |
|---|---|---|---|
| (i) | ADD Rd,Rr | $0 \le d \le 31, 0 \le r \le 31$ | PC ← PC + 1 |

**16-bit Opcode:**

| | | | |
|---|---|---|---|
| 0000 | 11rd | dddd | rrrr |

**Status Register (SREG) and Boolean Formula:**

| I | T | H | S | V | N | Z | C |
|---|---|---|---|---|---|---|---|
| – | – | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ |

# ADIW – Add Immediate to Word

**Description:**

Adds an immediate value (0 - 63) to a register pair and places the result in the register pair. This instruction operates on the upper four register pairs, and is well suited for operations on the pointer registers.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

**Operation:**

(i) Rd+1:Rd ← Rd+1:Rd + K

| | Syntax: | Operands: | Program Counter: |
|---|---|---|---|
| (i) | ADIW Rd+1:Rd,K | d ∈ {24,26,28,30}, 0 ≤ K ≤ 63 | PC ← PC + 1 |

**16-bit Opcode:**

| | | | |
|---|---|---|---|
| 1001 | 0110 | KKdd | KKKK |

**Status Register (SREG) and Boolean Formula:**

| I | T | H | S | V | N | Z | C |
|---|---|---|---|---|---|---|---|
| – | – | – | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ |

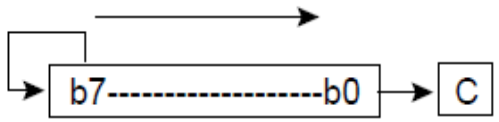
# ASR – Arithmetic Shift Right

**Description:**

Shifts all bits in Rd one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides a signed value by two without changing its sign. The Carry Flag can be used to round the result.

**Operation:**

(i)




| | Syntax: | Operands: | Program Counter: |
|---|---|---|---|
| (i) | ASR Rd | $0 \le d \le 31$ | PC ← PC + 1 |

**16-bit Opcode:**

| 1001 | 010d | dddd | 0101 |
|---|---|---|---|

**Status Register (SREG) and Boolean Formula:**

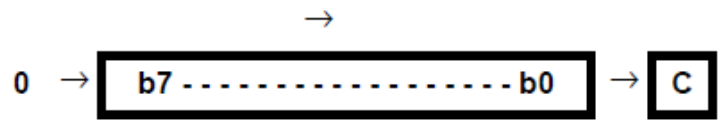
| I | T | H | S | V | N | Z | C |
|---|---|---|---|---|---|---|---|
| – | – | – | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ |

# LSR – Logical Shift Right

**Description:**

Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides an unsigned value by two. The C Flag can be used to round the result.

**Operation:**




| | Syntax: | Operands: | Program Counter: |
|---|---|---|---|
| (i) | LSR Rd | $0 \le d \le 31$ | PC ← PC + 1 |

**16-bit Opcode:**

| | | | |
|---|---|---|---|
| 1001 | 010d | dddd | 0110 |

**Status Register (SREG) and Boolean Formula:**

| I | T | H | S | V | N | Z | C |
|---|---|---|---|---|---|---|---|
| – | – | – | ⇔ | ⇔ | 0 | ⇔ | ⇔ |

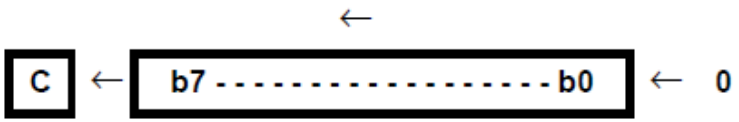
# LSL – Logical Shift Left

**Description:**

Shifts all bits in Rd one place to the left. Bit 0 is cleared. Bit 7 is loaded into the C Flag of the SREG. This operation effectively multiplies signed and unsigned values by two.

**Operation:**

(i)




| | Syntax: | Operands: | Program Counter: |
|---|---|---|---|
| (i) | LSL Rd | $0 \le d \le 31$ | PC ← PC + 1 |

**16-bit Opcode:** (see ADD Rd,Rd)

| 0000 | 11dd | dddd | dddd |
|---|---|---|---|

**Status Register (SREG) and Boolean Formula:**

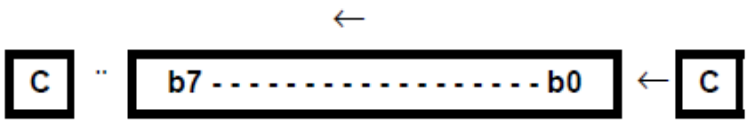
| I | T | H | S | V | N | Z | C |
|---|---|---|---|---|---|---|---|
| – | – | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ |

# ROL – Rotate Left trough Carry

**Description:**

Shifts all bits in Rd one place to the left. The C Flag is shifted into bit 0 of Rd. Bit 7 is shifted into the C Flag. This operation, combined with LSL, effectively multiplies multi-byte signed and unsigned values by two.

**Operation:**




| | Syntax: | Operands: | Program Counter: |
|---|---|---|---|
| (i) | ROL Rd | $0 \le d \le 31$ | PC ← PC + 1 |

**16-bit Opcode:** (see ADC Rd,Rd)

| 0001 | 11dd | dddd | dddd |
|---|---|---|---|

**Status Register (SREG) and Boolean Formula:**

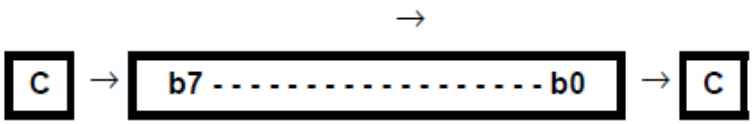
| I | T | H | S | V | N | Z | C |
|---|---|---|---|---|---|---|---|
| – | – | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ |

# ROR – Rotate Right through Carry

**Description:**

Shifts all bits in Rd one place to the right. The C Flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C Flag. This operation, combined with ASR, effectively divides multi-byte signed values by two. Combined with LSR it effectively divides multi-byte unsigned values by two. The Carry Flag can be used to round the result.

**Operation:**




| | Syntax: | Operands: | Program Counter: |
|---|---|---|---|
| (i) | ROR Rd | $0 \le d \le 31$ | PC ← PC + 1 |

**16-bit Opcode:**

| 1001 | 010d | dddd | 0111 |
|---|---|---|---|

**Status Register (SREG) and Boolean Formula:**

| I | T | H | S | V | N | Z | C |
|---|---|---|---|---|---|---|---|
| – | – | – | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ |