

# Digital Systems - Mini AVR 2

Implementing RJMP and Conditional Branches  
Adding Logic Function Instructions to the ALU

Pere Palà Schönwälder

iTIC <http://itic.cat>

April 2025

## Enhancing the processor

- ▶ Up to now, we have a linear program flow
  - ▶ Suitable for a pre-planned task
  - ▶ Unable to react to anything
- ▶ Need to jump to specific locations of the program
- ▶ Depending on inputs or computed results.
- ▶ RJMP (relative jump)
- ▶ BRANCH: BREQ / BRNE

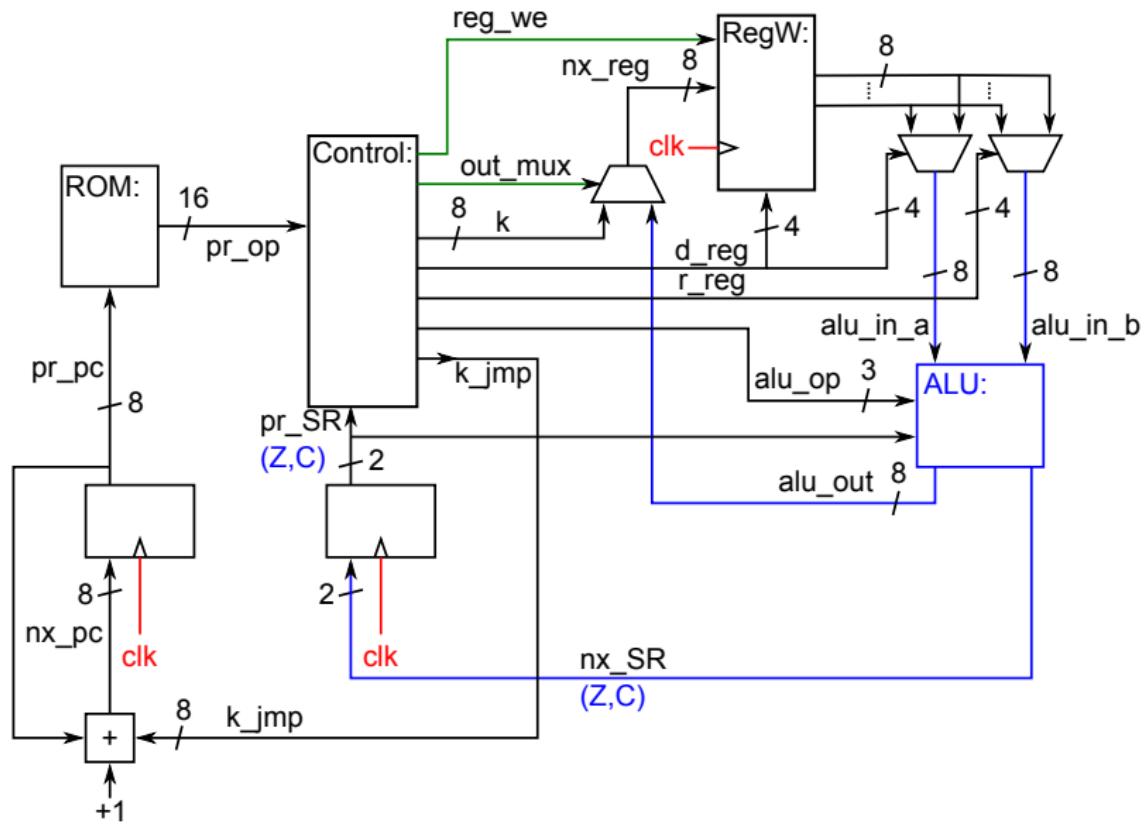
## Documentation of AVR instructions : RJMP

- ▶ RJMP – Relative Jump
- ▶ Description:
  - ▶ Relative jump to an address within  $PC - 2K + 1$  and  $PC + 2K$  (words). For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location. See also JMP .
- ▶ Operation:  $PC \leftarrow PC + k + 1$
- ▶ Syntax: RJMP k
- ▶ Operands:  $-2K \leq k < 2K$
- ▶ Program Counter:  $PC \leftarrow PC + k + 1$
- ▶ 16-bit Opcode: 1100 kkkk kkkk kkkk
- ▶ Status Register (SREG) and Boolean Formula:

I T H S V N Z C

-----

# The architecture with relative jumps



## Documentation of AVR instructions : BREQ

- ▶ BREQ - Branch if Equal
- ▶ Description:
  - ▶ Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is set. This instruction branches relatively to PC in either direction ( $PC - 63 \leq \text{destination} \leq PC + 64$ ). The parameter k is the offset from PC and is represented in two's complement form.
- ▶ Operation: If  $Rd = Rr$  ( $Z = 1$ ) then  $PC \leftarrow PC + k + 1$ , else  $PC \leftarrow PC + 1$
- ▶ Syntax: BREQ k
- ▶ Operands:  $-64 \leq k \leq +63$
- ▶ Program Counter:  $PC \leftarrow PC + k + 1$  or  $PC \leftarrow PC + 1$ , if condition is false
- ▶ 16-bit Opcode: 1111 00kk kkkk k001
- ▶ Status Register (SREG) and Boolean Formula:

I T H S V N Z C

-----

## Documentation of AVR instructions : BRNE

- ▶ BRNE - Branch if Not Equal
- ▶ Description:
  - ▶ Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is zero. This instruction branches relatively to PC in either direction ( $PC - 63 \leq$  destination  $\leq PC + 64$ ). The parameter k is the offset from PC and is represented in two's complement form.
- ▶ Operation: If  $Rd \neq Rr$  ( $Z = 0$ ) then  $PC \leftarrow PC + k + 1$ , else  $PC \leftarrow PC + 1$
- ▶ Syntax: BRNE k
- ▶ Operands:  $-64 \leq k \leq +63$
- ▶ Program Counter:  $PC \leftarrow PC + k + 1$  or  $PC \leftarrow PC + 1$ , if condition is false
- ▶ 16-bit Opcode: 1111 01kk kkkk k001
- ▶ Status Register (SREG) and Boolean Formula:

I T H S V N Z C

-----

# VHDL implementation of program counter logic

```
NEXT_PC : process(pr_pc,k_jmp)
  variable tmp_pc : std_logic_vector(8 downto 0);
begin
  tmp_pc := std_logic_vector(signed(pr_pc & '1') +
                           signed(k_jmp & '1'));
  nx_pc  <= tmp_pc(8 downto 1);
end process;
```

# VHDL implementation of Control Unit

```
constant BRANCH    : std_logic_vector(3 downto 0) := "1111";
...
CONTROL : process(pr_op,pr_pc,pr_SR)
begin
    out_mux <= mux_alu;
    r_reg   <= pr_op(3 downto 0);      -- Defaults
    d_reg   <= pr_op(7 downto 4);
    reg_we  <= '0';
    k_jmp   <= (others => '0');
...
    case pr_op(15 downto 12) is
...
        when BRANCH => ----- BRANCH Instruction
            if pr_op(10) = '0' then -- BREQ Instruction
                if pr_SR.Z = '1' then
                    k_jmp(6 downto 0) <= pr_op(9 downto 3);
                    k_jmp(7) <= pr_op(9);
                end if;
                else -- BRNE (Complete this instruction)
                end if;
            when RJMP => -- RJMP (Complete this instruction)
            when others =>
...
    end process;
```

# Documentation of AVR instructions : AND

- ▶ AND – Logical AND
- ▶ Description:
  - ▶ Performs the logical AND between the contents of register Rd and register Rr and places the result in the destination register Rd.
- ▶ Operation:  $Rd \leftarrow Rd \bullet Rr$
- ▶ Syntax: AND Rd,Rr
- ▶ Operands:  $0 \leq d \leq 31, 0 \leq r \leq 31$
- ▶ Program Counter:  $PC \leftarrow PC + 1$
- ▶ 16-bit Opcode: 0010 00rd dddd rrrr
- ▶ Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	$\Leftrightarrow$	0	$\Leftrightarrow$	$\Leftrightarrow$	-
- ▶ Z: Set if the result is x"00"; cleared otherwise.

# Documentation of AVR instructions : OR

- ▶ OR – Logical OR
- ▶ Description:
  - ▶ Performs the logical OR between the contents of register Rd and register Rr and places the result in the destination register Rd.
- ▶ Operation:  $Rd \leftarrow Rd \vee Rr$
- ▶ Syntax: OR Rd,Rr
- ▶ Operands:  $0 \leq d \leq 31, 0 \leq r \leq 31$
- ▶ Program Counter:  $PC \leftarrow PC + 1$
- ▶ 16-bit Opcode: 0010 10rd dddd rrrr
- ▶ Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	$\Leftrightarrow$	0	$\Leftrightarrow$	$\Leftrightarrow$	-
- ▶ Z: Set if the result is x"00"; cleared otherwise.

# Documentation of AVR instructions : EOR

- ▶ EOR – Exclusive OR
- ▶ Description:
  - ▶ Performs the logical EOR between the contents of register Rd and register Rr and places the result in the destination register Rd.
- ▶ Operation:  $Rd \leftarrow Rd \oplus Rr$
- ▶ Syntax: EOR Rd,Rr
- ▶ Operands:  $0 \leq d \leq 31, 0 \leq r \leq 31$
- ▶ Program Counter:  $PC \leftarrow PC + 1$
- ▶ 16-bit Opcode: 0010 01rd dddd rrrr
- ▶ Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
-	-	-	$\Leftrightarrow$	0	$\Leftrightarrow$	$\Leftrightarrow$	-
- ▶ Z: Set if the result is x"00"; cleared otherwise.

# VHDL implementation of Control Unit

```
-- ALU_B replaced the old MOV instruction
constant ALU_B : std_logic_vector(3 downto 0) := "0010";
constant ALU_B_AND : std_logic_vector(1 downto 0) := "00";
constant ALU_B_EOR : std_logic_vector(1 downto 0) := "01";
constant ALU_B_OR : std_logic_vector(1 downto 0) := "10";
constant ALU_B_MOV : std_logic_vector(1 downto 0) := "11";
...
CONTROL : process(pr_op,pr_pc,pr_SR)
begin
...
    case pr_op(15 downto 12) is
        when ALU_B => -----MOV, AND, EOR, OR Instructions
            reg_we <= '1';           --Enable register write
            case pr_op(11 downto 10) is --Decode further down
                when ALU_B_MOV => ALU_op <= ALU_MOV;
                when ALU_B_EOR => ALU_op <= ALU_EOR;
                when ALU_B_AND => ALU_op <= ALU_AND;
                when ALU_B_OR  => ALU_op <= ALU_OR;
                when others => null;
            end case;
    ...
end process;
```

# VHDL implementation of the ALU

```
ALU : process(ALU_op,alu_in_a,alu_in_b,pr_SR.C,add_temp)
begin
    nx_SR.C <= pr_SR.C;           --by default, preserve
    update_Z <= '1';             --Most operations update Z

    case ALU_op is
        when ALU_MOV => -----MOV: in_b --> out
            alu_out <= alu_in_b;
            update_Z <= '0';
        when ALU_AND => -----AND
            alu_out <= alu_in_a and alu_in_b;
        when ALU_OR  => -----OR
            alu_out <= alu_in_a or alu_in_b;
        when ALU_EOR  => -----EOR / XOR
            alu_out <= alu_in_a xor alu_in_b;
        when ALU_ADC  => -----ADC: Already known
            ...
        when others => -----Should never happen
            alu_out <= (others => '-'); --Don't care
    end case;
end process;
```

# VHDL implementation of the Z Flag

```
UPD_Z: process(update_Z,alu_out,pr_SR)
begin
    if update_Z = '1' then ----- Update Zero Flag
        if alu_out = x"00" then
            nx_SR.Z <= '1';
        else
            nx_SR.Z <= '0';
        end if;
    else
        nx_SR.Z <= pr_SR.Z;
    end if;
end process;
```

# Sample code 1

```
LDI r17,x01
LDI r16,xFE
ADC r16,r17      ;-----
BREQ +1          ;      |
RJMP -3          ; -----
RJMP -1          ; HALT
```

```
ROM : process(pr_pc) --Program ROM
begin
    case pr_pc is
        when X"00" => pr_op <= LDI & "0000" & c_r17 & "0001";
        when X"01" => pr_op <= LDI & "1111" & c_r16 & "1110";
        when X"02" => pr_op <= ADC & "1111" & c_r16 & c_r17 ;
        when X"03" => pr_op <= BREQ & "00" & "0000001" & "001";

        when X"04" => pr_op <= RJMP & "----" & "1111" & "1101";
        when X"05" => pr_op <= RJMP & "----" & "1111" & "1111";
        when others => pr_op <= (others => '-');
    end case;
end process;
```

## Sample code 2

```
LDI r17,x03
LDI r16,xFE
EOR r16,r17
ADC r16,r17
LDI r17,x03
LDI r16,xFE
OR r16,r17
LDI r17,x03
AND r16,r17
```

```
when X"00" => pr_op <= LDI      & "0000" & c_r17 & "0011";
when X"01" => pr_op <= LDI      & "1111" & c_r16 & "1110";
when X"02" => pr_op <= ALU_B    &
                  ALU_B_EOR &"00" & c_r16 & c_r17 ;
when X"03" => pr_op <= ADC      & "1111" & c_r16 & c_r17 ;
when X"04" => pr_op <= LDI      & "0000" & c_r17 & "0011";
when X"05" => pr_op <= LDI      & "1111" & c_r16 & "1110";
when X"06" => pr_op <= ALU_B    &
                  ALU_B_OR  &"00" & c_r16 & c_r17 ;
when X"07" => pr_op <= LDI      & "0000" & c_r17 & "0011";
when X"08" => pr_op <= LDI      & "1111" & c_r16 & "1110";
when X"09" => pr_op <= ALU_B    &
                  ALU_B_AND &"00" & c_r16 & c_r17 ;
when others => pr_op <= (others => '-');
```