Digital Systems - Mini AVR 1

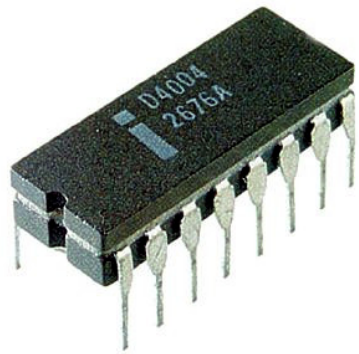
Pere Palà - Alexis López

iTIC http://itic.cat

March 2016

Processor Basics

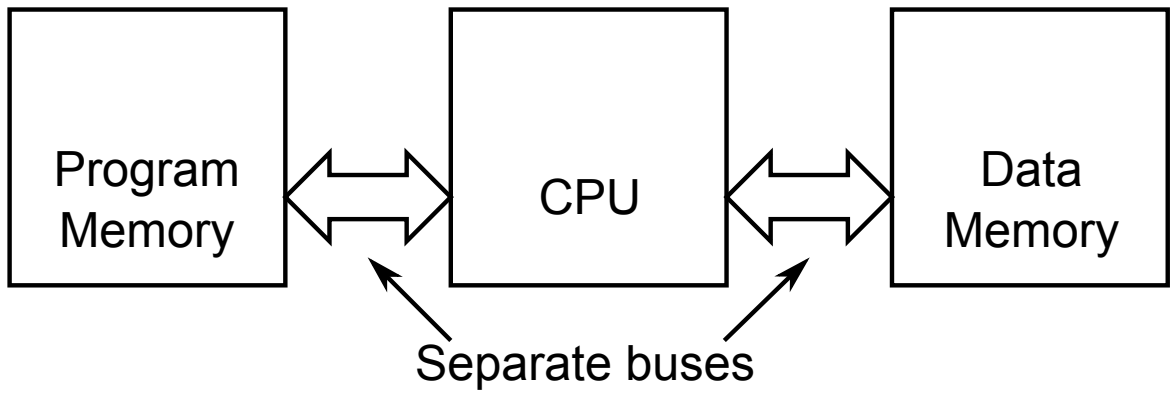
- Processor: A specific (and interesting) (and complex) kind of digital system
- A processor inside a digital system
  - Specialized tasks
  - Multiple small processors
  - Other digital circuitry complements the system
- Processor basics
  - A list of instructions
  - Space to store data
  - Core that acts according to instructions
  - Input and output ports to interact with the real world
- Architectures
  - Von Neumann
  - Harvard



Intel 4004

Harvard Architecture

- Separate memories
  - Program (instruction) memory
  - Data memory

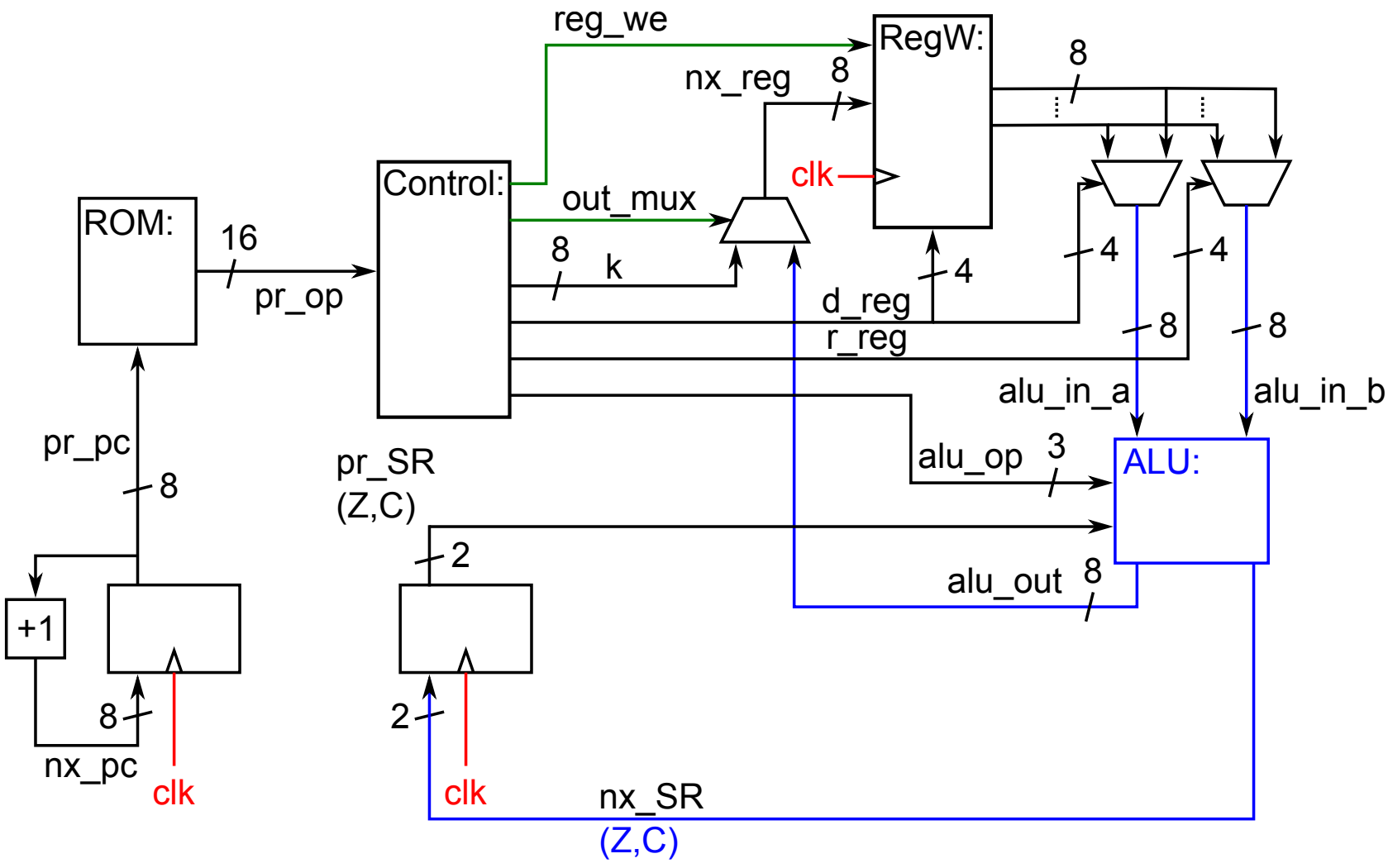


- Allows accessing instructions and data simultaneously
- Eliminates bottleneck of common bus
- Common in small micro-controllers, such as the Atmel AVR

The Mini AVR

- Objective
  - To build a small AVR-compatible micro-controller
  - With a reduced number of instructions
- Why?
  - Just for fun!
  - To be able to port an AVR program to our micro-controller
- Instructions
  - NOP : No operation
  - LDI : Load immediate
  - ADC : Add with carry
  - MOV : Move
  - AND, OR, EOR : And, or, exclusive-or operations
  - RJMP : Relative jump
  - BREQ : Branch if equal
- Some flags of the Status Register (SREG)
  - C: Carry Flag
  - Z: Zero Flag

Overview (don't panic yet!)



Documentation of AVR instructions : NOP

- NOP – No Operation
- Description
  - This instruction performs a single cycle No Operation.
- Operation: No
- Syntax: NOP
- Operands: None
- Program Counter: PC $\leftarrow$ PC + 1
- 16-bit Opcode: `0000 0000 0000 0000`
- Status Register (SREG) and Boolean Formula:

```
I T H S V N Z C
- - - - - - - -
```

Documentation of AVR instructions : LDI

- LDI – Load Immediate
- Description:
  - Loads an 8 bit constant directly to register 16 to 31.
- Operation: Rd $\leftarrow$ K
- Syntax: LDI Rd,K
- Operands: $16 \leq d \leq 31$, $0 \leq K \leq 255$
- Program Counter: PC $\leftarrow$ PC + 1
- 16-bit Opcode: `1110 KKKK dddd KKKK`
- Status Register (SREG) and Boolean Formula:

```
I T H S V N Z C
- - - - - - - -
```

Documentation of AVR instructions : ADC

- ADC – Add with Carry
- Description:
  - Description: Adds two registers and the contents of the C Flag and places the result in the destination register Rd.
- Operation: Rd $\leftarrow$ Rd + Rr + C
- Syntax: ADC Rd,Rr
- Operands: $0 \leq d \leq 31$, $0 \leq r \leq 31$
- Program Counter: PC $\leftarrow$ PC + 1
- 16-bit Opcode: `0001 11rd dddd rrrr`
- Status Register (SREG) and Boolean Formula:

  `I T H S V N Z C`

  \- - $\Leftrightarrow\Leftrightarrow\Leftrightarrow\Leftrightarrow\Leftrightarrow\Leftrightarrow$
- Z: Set if the result is x"00"; cleared otherwise.
- C: Set if there was carry from the MSB of the result; cleared otherwise

Documentation of AVR instructions : MOV

- MOV – Copy Register
- Description:
  - This instruction makes a copy of one register into another. The source register Rr is left unchanged, while the destination register Rd is loaded with a copy of Rr.
- Operation: Rd $\leftarrow$ Rr
- Syntax: MOV Rd,Rr
- Operands: $0 \leq d \leq 31$, $0 \leq r \leq 31$
- Program Counter: PC $\leftarrow$ PC + 1
- 16-bit Opcode: `0010 11rd dddd rrrr`
- Status Register (SREG) and Boolean Formula:

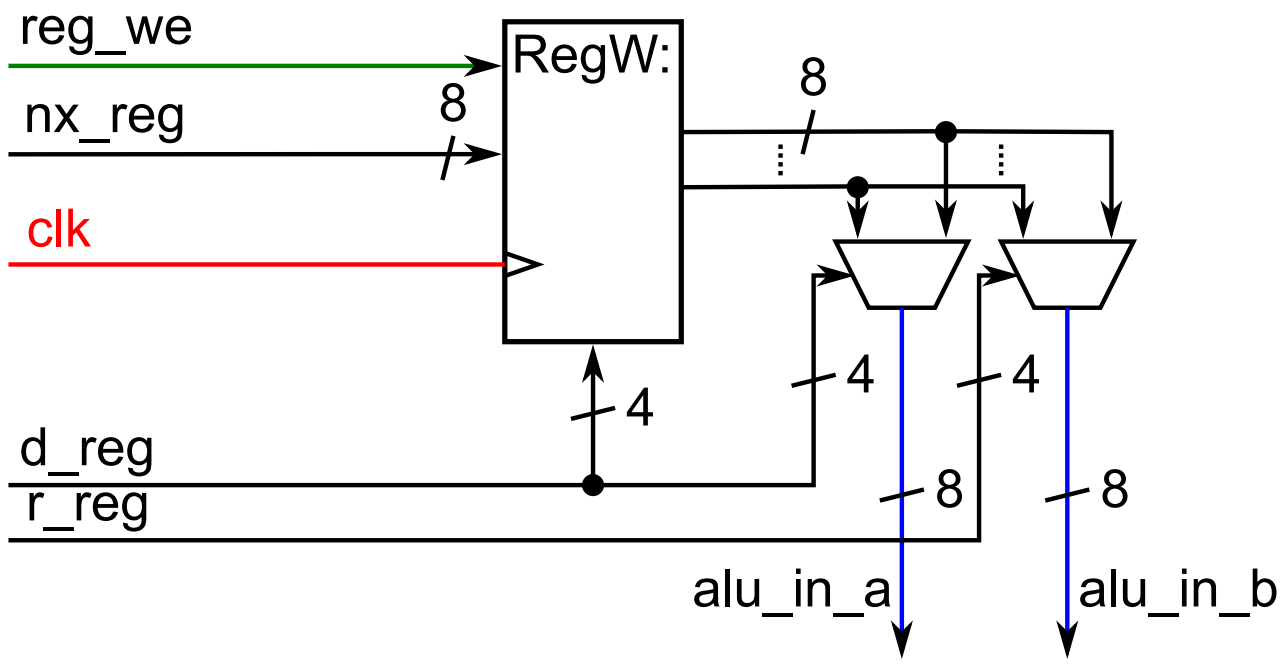
```
I T H S V N Z C
- - - - - - - -
```

References

- Further information:
  - 8-bit AVR Instruction Set
  - `http://ww1.microchip.com/downloads/en/devicedoc/atmel-0856-avr-instruction-set-manual.pdf`

Registers

- Implement only registers from 16 to 31
- ADC 16-bit Opcode
  - `0001 11rd dddd rrrr`
  - `0001 1111 dddd rrrr`
- No stack, stack pointer, etc



VHDL implementation of registers

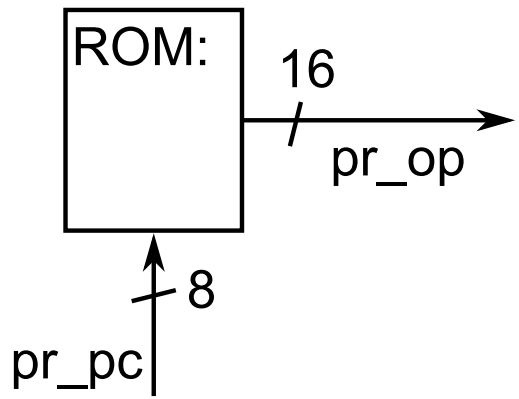
```
type register_bank  is
   array (15 downto 0) of std_logic_vector(7 downto 0);
signal regs : register_bank;              --Registers r16..r31
```

```
   RegW : process(clk) --Register write
   begin
      if rising_edge(clk) then
         if reg_we = '1' then -- write
            regs(to_integer(unsigned(d_reg))) <= nx_reg;
         end if;
      end if;
   end process RegW;
```

```
   --Register read: ALU inputs
   alu_in_a <= regs(to_integer(unsigned(d_reg)));
   alu_in_b <= regs(to_integer(unsigned(r_reg)));
```

Program ROM

- 256 positions (arbitrary limit)
  - Program Counter has to be 8 bits long
- 16-bit opcodes
- Not clocked: Purely combinational ROM
- Given `pr_pc`, the present program counter, the system outputs `pr_op`, the present opcode



Program listing

- A simple program code:

```
NOP             ; do nothing
LDI r16,x83     ; r16 ← x83
ADC r16,r16     ; x83 + x83 = x106 : r16 ← 06 and C is set!
ADC r16,r16     ; x06 + x06 + x01 = x0D and C is cleared
MOV r17,r16     ; r17 ← x0D
NOP             : do nothing
```

- NOP : `0000 0000 0000 0000`
- LDI : `1110 kkkk dddd kkkk`
- ADC : `0001 1111 dddd rrrr` (`0001 11rd dddd rrrr`)
- MOV : `0010 1111 dddd rrrr` (`0010 11rd dddd rrrr`)
- The first 4 bits are enough to determine the instruction
- VHDL definition:

```
constant NOP : std_logic_vector(3 downto 0) := "0000";
constant LDI : std_logic_vector(3 downto 0) := "1110";
constant ADC : std_logic_vector(3 downto 0) := "0001";
constant MOV : std_logic_vector(3 downto 0) := "0010";
```

VHDL implementation of the program memory

```
ROM : process(pr_pc) --Program ROM
begin
   case pr_pc is
   when X"00" =>
      pr_op <= NOP & "0000" & "--------";      -- NOP
   when X"01" =>
      pr_op <= LDI & "1000" & "0000" & "0011";-- LDI r16,x83
   when X"02" =>
      pr_op <= ADC & "1111" & "0000" & "0000";-- ADC r16,r16
   when X"03" =>
      pr_op <= ADC & "1111" & "0000" & "0000";-- ADC r16,r16
   when X"04" =>
      pr_op <= MOV & "1111" & "0001" & "0000";-- MOV r17,r16
   when X"05" =>
      pr_op <= NOP & NOP    & "--------";      -- NOP
   when others => pr_op <= (others => '-');
 end case;
end process;
```

- This works as a first try! Later we will learn to describe a ROM in a way that allows the synthesis tools to choose (much) better implementations.