FAKULTÄT
FÜR !NFORMATIK

Faculty of Informatics

# FAKULTÄT FÜR !NFORMATIK
# 182.694 Microcontroller VU

Martin Perner

SS 2014

Featuring Today:
A Deep Look into the Processor Core
Getting Code onto the Microcontroller Chip

# Organizational Stuff

- Make your slot suggestions in myTI.
- Use the lab!

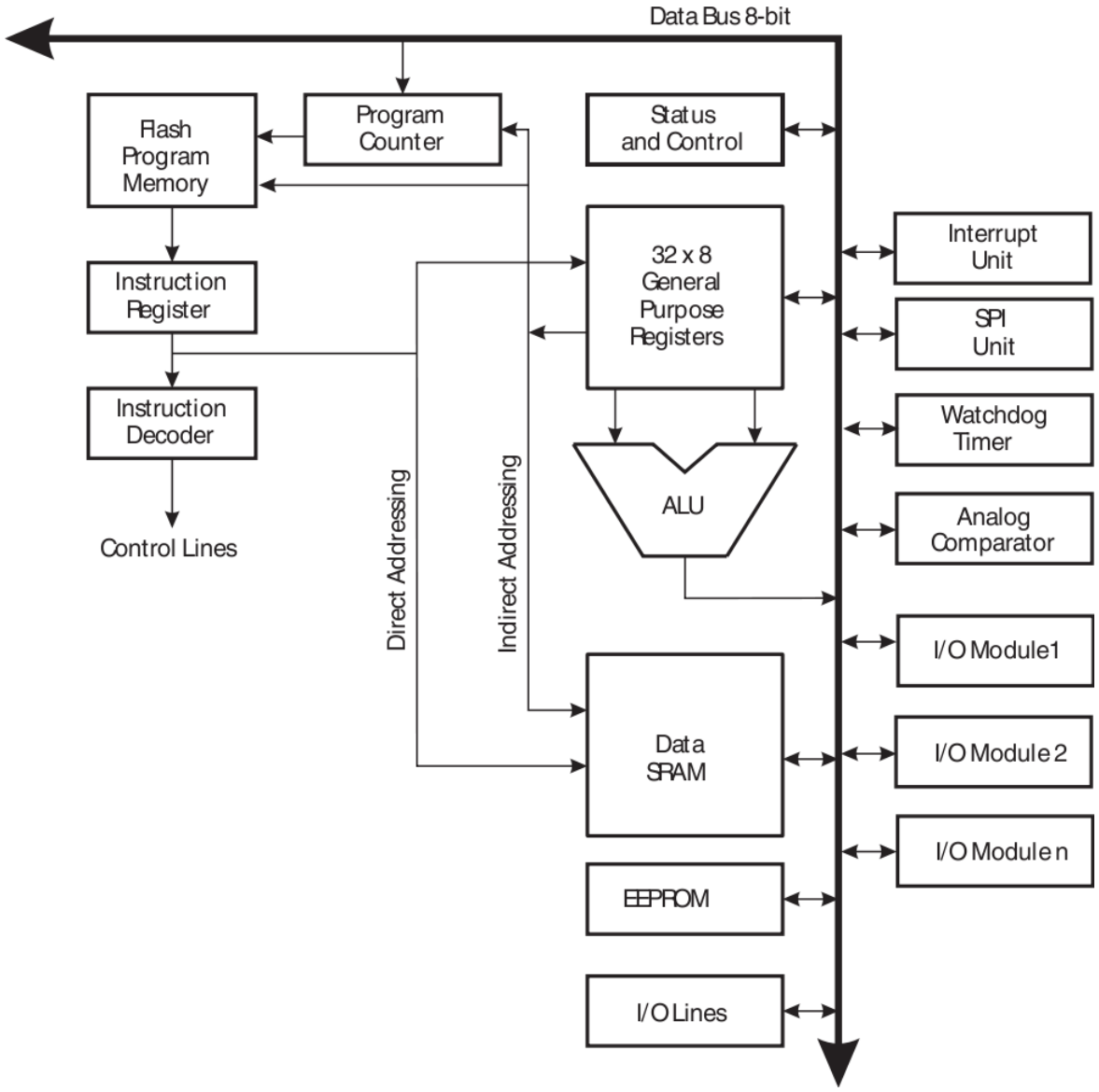
# Weekly Training Objective

- This week
  - 1.2 Board test
  - 2.1.1 Assembler demo program
  - 2.1.2 Makefile
  - 2.2.1 Logical operations
- Next week
  - 2.2.2 Input with floating pins
  - 2.2.4 Monoflop buttons
  - 2.2.5 Digital I/O
  - 2.4.1 precompiled LCD
- Until Exam (in 2.5 weeks)
  - 2.2.8 LED curtain
  - 2.4.2 Calling conventions I
  - 2.4.3 Calling conventions II
  - 2.4.4 Fibonacci Numbers

# A Deep Look into the Processor Core

- We are using Atmel AVR microcontroller in lab
- Not restricted to AVR only
- Every microcontroller works nearly 'the same'

# AVR Core


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

# A Microcontroller Executable

## Motorola S-Record File: (*.srec)

```
S00C000064656D6F2E7372656373
S11300000C940C002F93202F015618F0095108F07E
S11300102052022F2F9108950FEF0DBF01E20EBF62
S113002000E002B90FEF01B904E40EBB00E00FBB1E
S11300300591003029F00E94020002B90C941800C6
S11300400C94200048656C6C6F20576F726C6421AF
S105005030007A
S9030000FC
```

Is a printable form of an 'exe' file for a microcontroller.

# Motorola S-Record Format

S11300000C940C002F93202F015618F0095108F07E

- Record Type (S1)

| | |
|---|---|
| S0 | Block Header |
| S1-3 | Data Records |
| S5 | Record Count |
| S7-9 | End of Block |

- Byte Count (0x13) Address + Data + Checksum
- Address (0x0000)
- Data (only for S0-3)
- Checksum (0x7E)
  (ones' complement of the sum of: byte count, address bytes, and data bytes) & 0xFF

# A Microcontroller Executable

## Intel Hex File: (*.hex)

```
:100000000C940C002F93202F015618F0095108F082
:10001000205202 2F2F9108950FEF0DBF01E20EBF66
:1000200000E002B90FEF01B904E40EBB00E00FBB22
:10003000059100 3029F00E94020002B90C941800CA
:100040000C94200048656C6C6F20576F726C6421B3
:020050003000 7E
:00000001FF
```

Same data, other format.

# Intel Hex Format

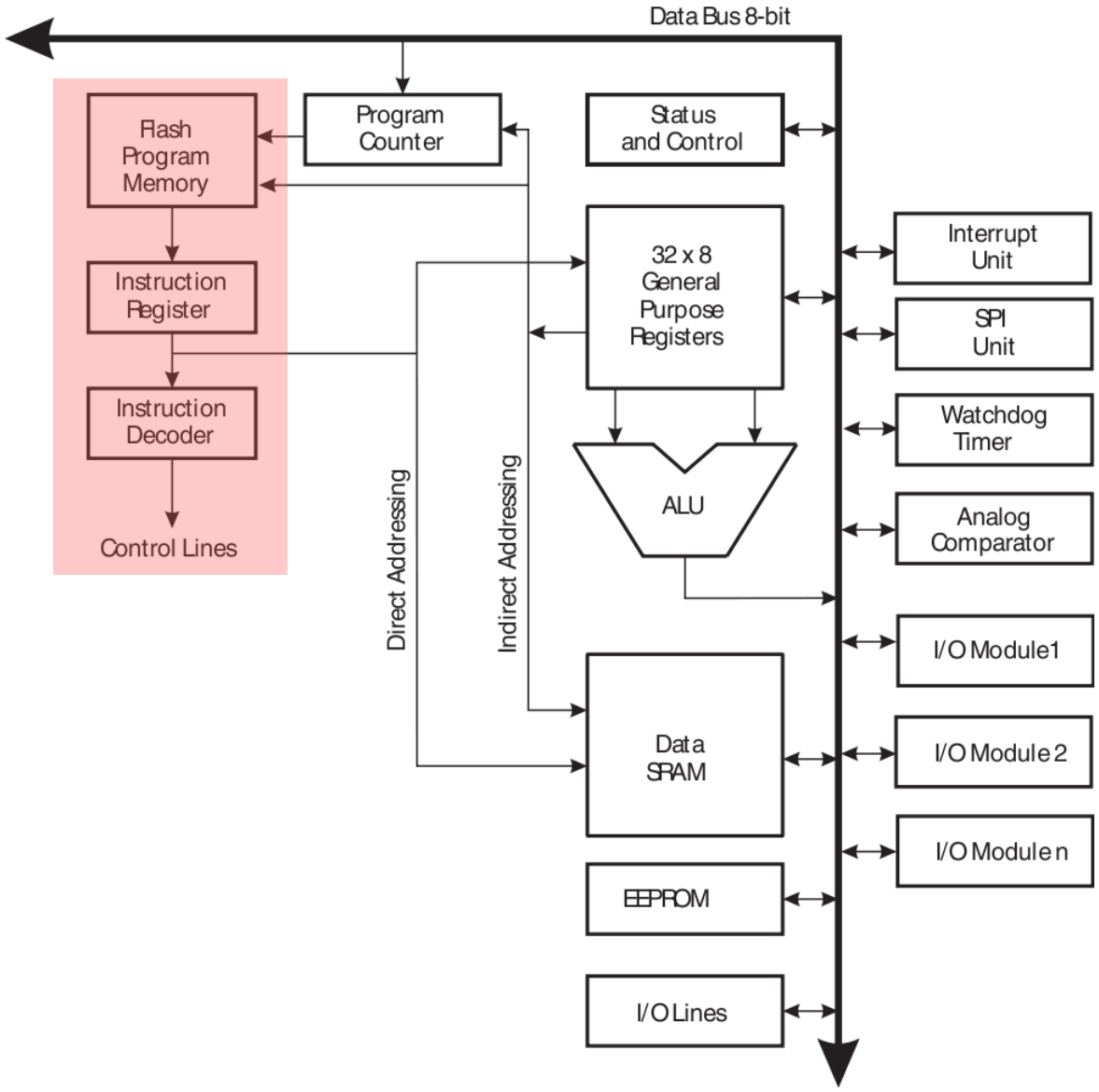
:100000000C940C002F93202F015618F0095108F082

- Start Code (:)
- Byte Count (0x10) Data only
- Address (0x0000)
- Record Type (00)

  00 Data Records
  01 EOF Record
  ...

- Data
- Checksum (0x82)
  (two's complement of the sum of: byte count, address bytes, record type, and data bytes) & 0xFF

# AVR Core


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

# Instructions

```
:100000000C940C002F93202F015618F0095108F082
:10001000205202 2F2F9108950FEF0DBF01E20EBF66
:1000200000E002B90FEF01B904E40EBB00E00FBB22
:10003000059100 3029F00E94020002B90C941800CA
:100040000C94200048656C6C6F20576F726C6421B3
:0200500030007E
:00000001FF
```

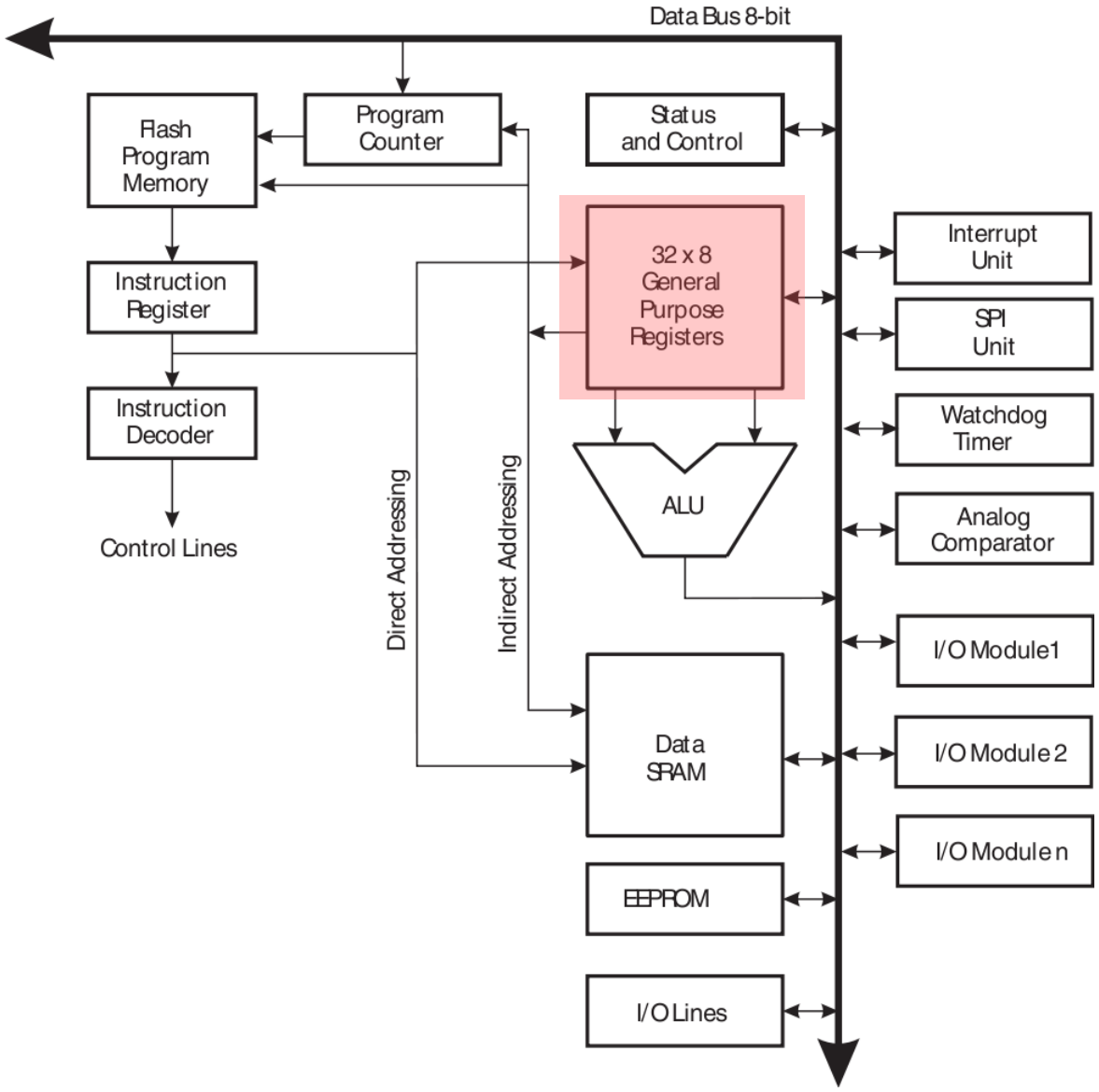
- AVR Instructions 16/32 bit opcodes
- Instructions directly readable from hex file (little endian)

# Instructions

```
:100000000C940C002F93202F015618F0095108F082
:10001000205202 2F2F9108950FEF0DBF01E20EBF66
:1000200000E002B90FEF01B904E40EBB00E00FBB22
:10003000059100 3029F00E94020002B90C941800CA
:100040000C94200048656C6C6F20576F726C6421B3
:0200500030007E
:00000001FF
```

- AVR Instructions 16/32 bit opcodes
- Instructions directly readable from hex file (little endian)
- e.g., 01E2 → opcode 0xE201 → 1110 0010 0000 $0001_b$ → ldi r16, 0x21 (loads value 0x21 into register r16)

# AVR Core


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

# Register

```
:100000000C940C002F93202F015618F0095108F082
:10001000205202 2F2F9108950FEF0DBF01E20EBF66
:1000200000E002B90FEF01B904E40EBB00E00FBB22
:10003000059100 3029F00E94020002B90C941800CA
:100040000C94200048656C6C6F20576F726C6421B3
:0200500030007E
:00000001FF
```

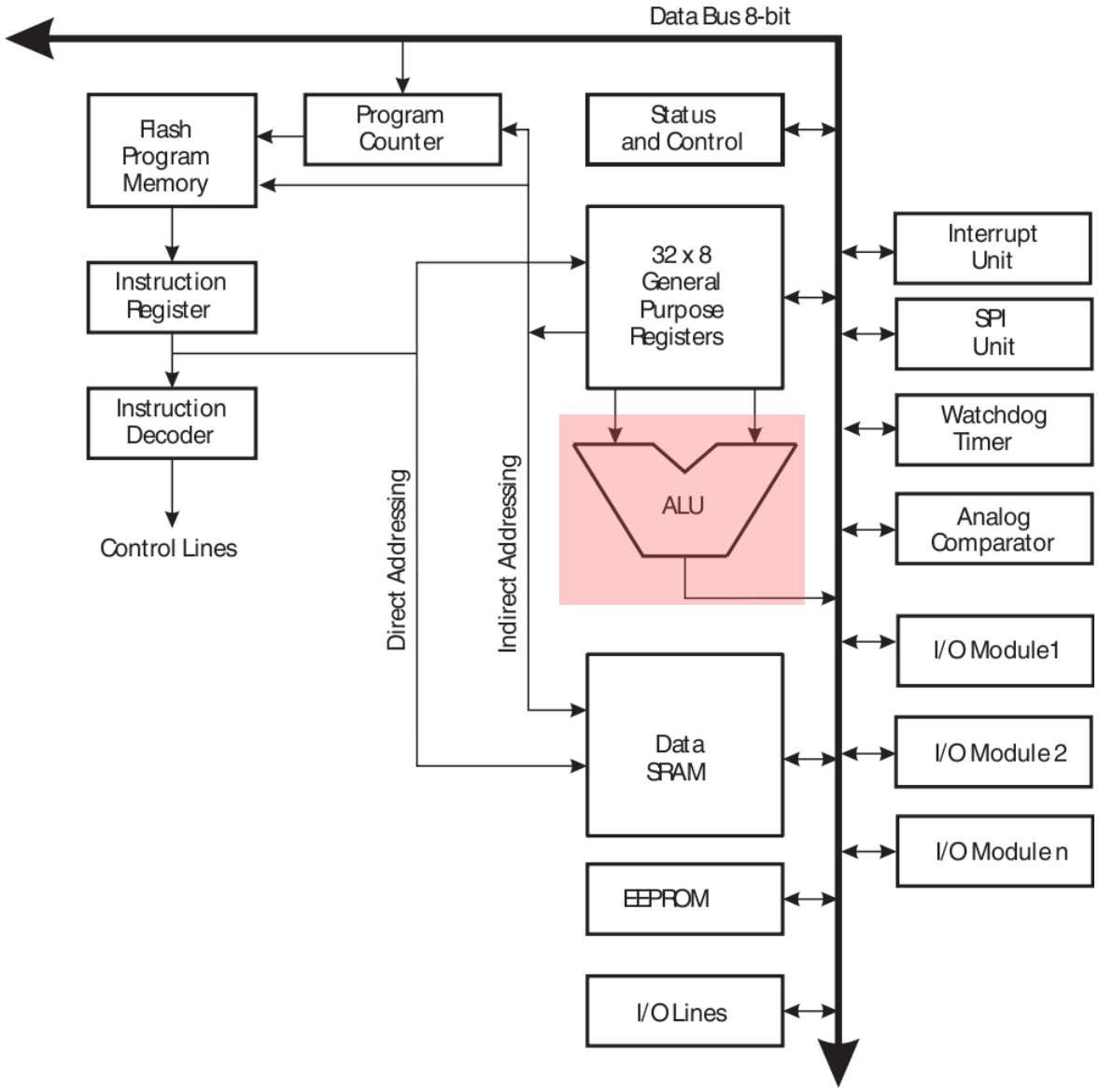
- Used to perform operations
  - ldi r16, 0x21 (loads value 0x21 into register r16)
  - subi r16, 0x61 (subtract 0x61 from value stored in register r16)
- Store the result of operations

# Register @ AVR

- 32 GP Register à 8 bit
- Mapped into SRAM at first 32 addresses
- Some operations have restrictions on the use of registers, e.g., subi only works on registers $\geq 16$
- R26-31 can be used as 16 Bit address pointers X, Y, Z (see addressing modes) and are also accessible via XL, XH, ...
- IO-Register to control peripherals

| 7 … 0 | Addr. | |
|---|---|---|
| R0 | 0x00 | |
| R1 | 0x01 | |
| R2 | 0x02 | |
| ... | | |
| R13 | 0x0D | |
| R14 | 0x0E | |
| R15 | 0x0F | |
| R16 | 0x10 | |
| R17 | 0x11 | |
| ... | | |
| R26 | 0x1A | X-register Low Byte |
| R27 | 0x1B | X-register High Byte |
| R28 | 0x1C | Y-register Low Byte |
| R29 | 0x1D | Y-register High Byte |
| R30 | 0x1E | Z-register Low Byte |
| R31 | 0x1F | Z-register High Byte |

# AVR Core


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

# Status Register

- Special register containing information about result of the last instruction

| | |
|---|---|
| I | Global Interrupt Enable |
| T | T-bit (can be used for bit copy) |
| H | Half Carry Flag |
| S | Sign Bit (N XOR V) |
| V | Two's Complement Overflow Flag |
| N | Negative Flag |
| Z | Zero Flag |
| C | Carry Flag |

# Status Register

- Affected flags can be seen in the Instruction Set

| Mnemonics | Operands | Description | Operation | Flags |
|---|---|---|---|---|
| SUBI | Rd, K | Subtract Immediate | Rd ← Rd - K | Z,C,N,V,S,H |

- Flags can be used to determine program flow

| 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 |

# Flag Example

## 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 |
|---|---|---|---|---|---|---|---|
| – | – | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ | ⇔ |

H: Rd3

S: $N \oplus V$, For signed tests.

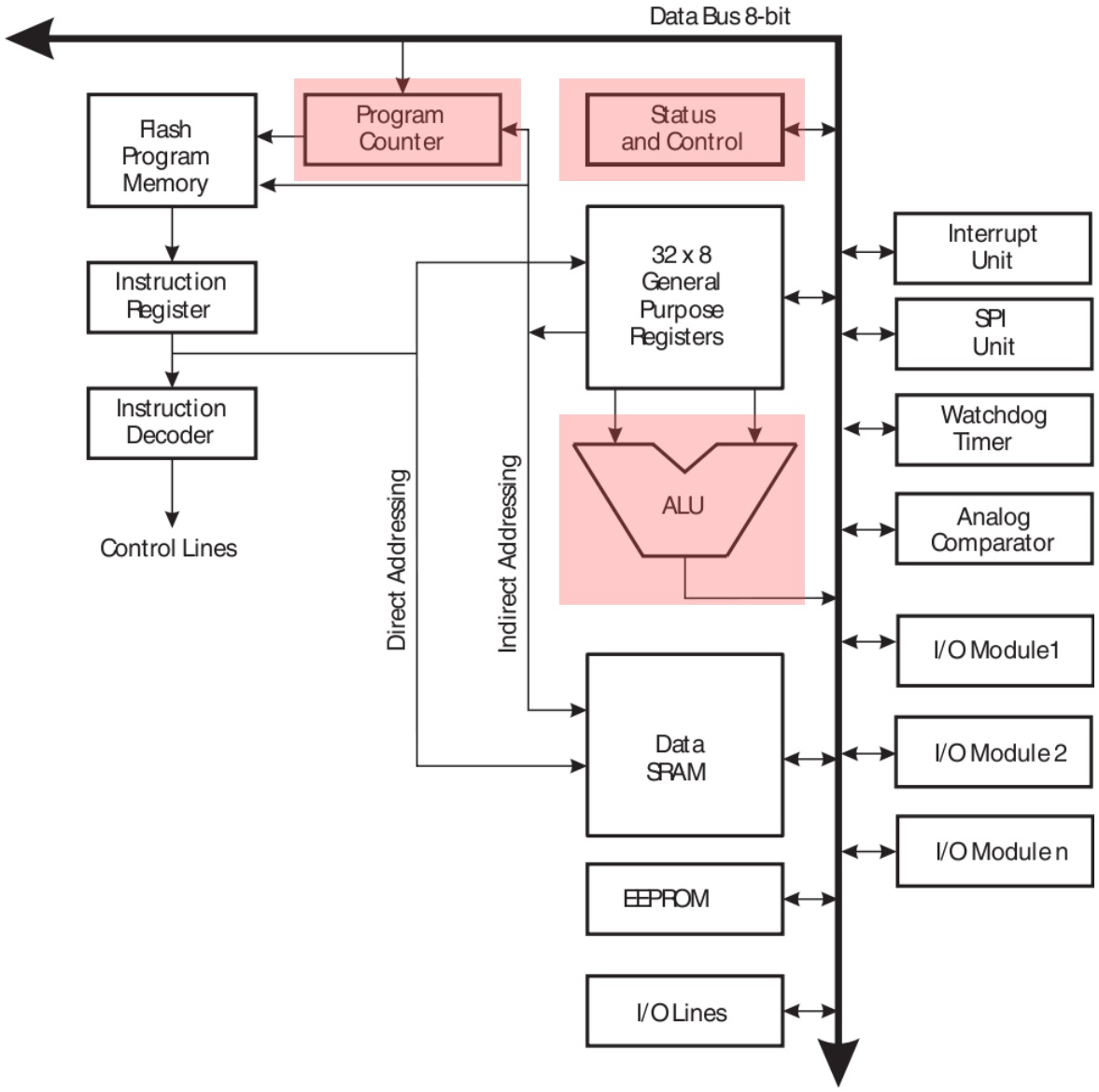
V: $N \oplus C$ (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

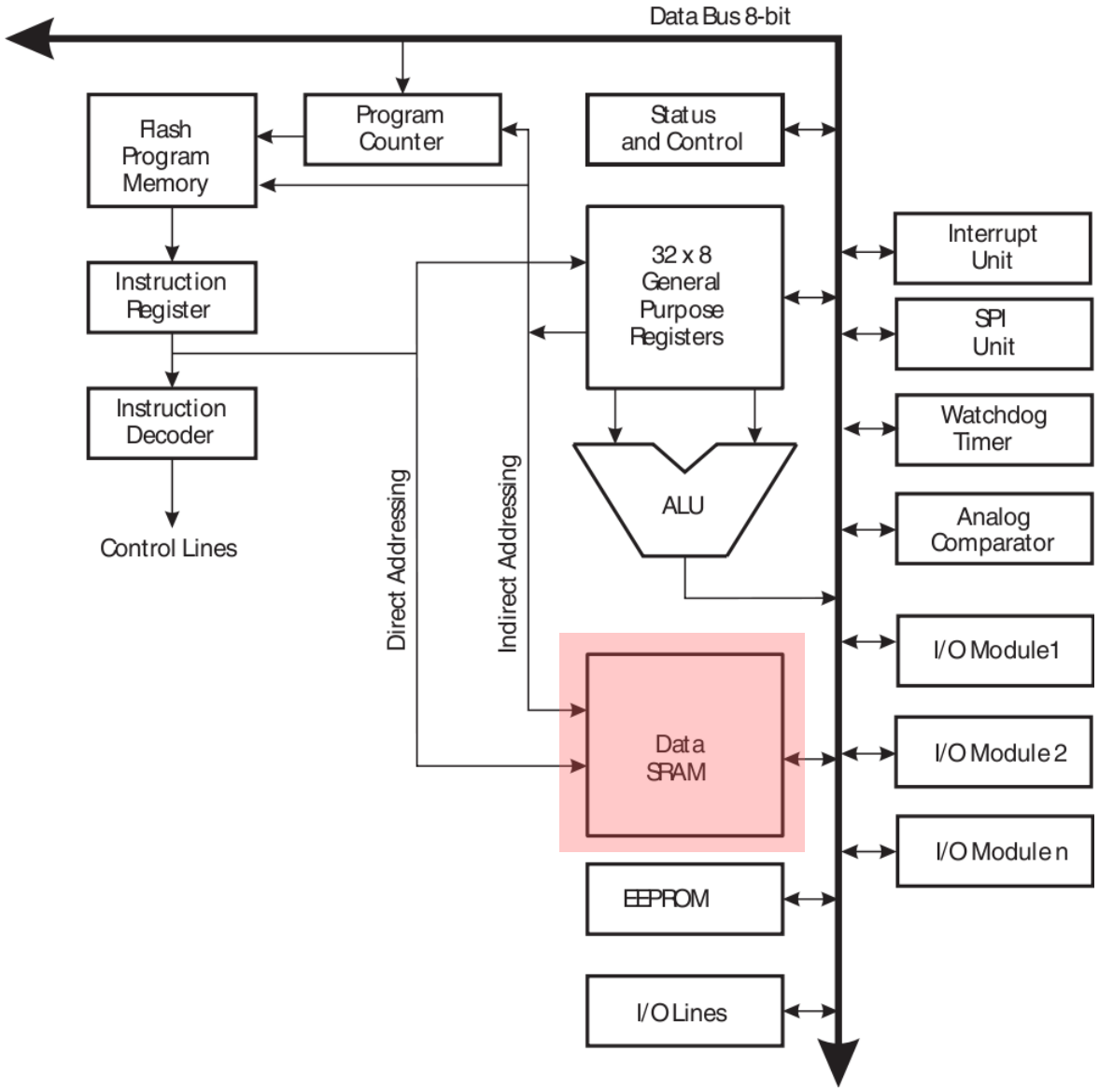
# AVR Core


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

# Branches

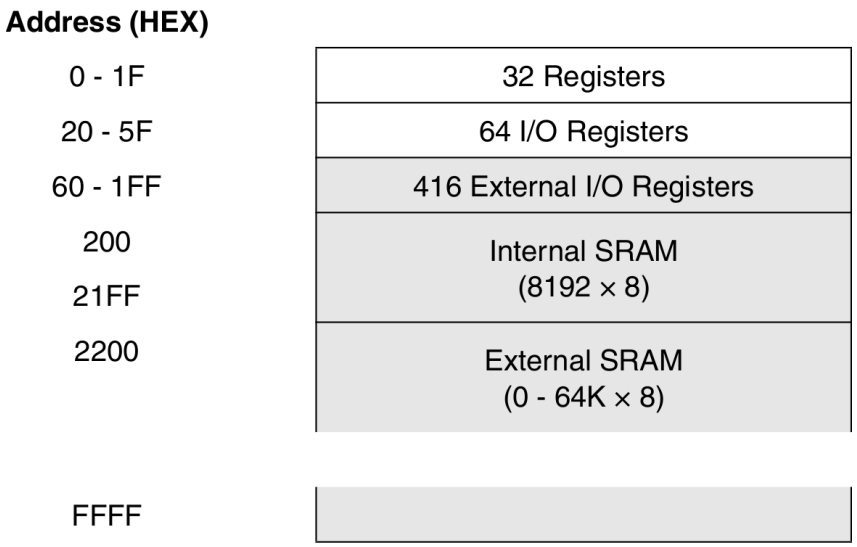
- Flags in the status register can also be used for conditional branch (jump-if-statements) [very limited range!]
  - BREQ label1 ; jumps to label1 if Z=1
  - BRLO label2 ; jumps to label2 if C=1
  - ...
- Unconditional Jumps
  - JMP (reaches complete memory range)
  - RJMP (fast but smaller range)

# AVR Core

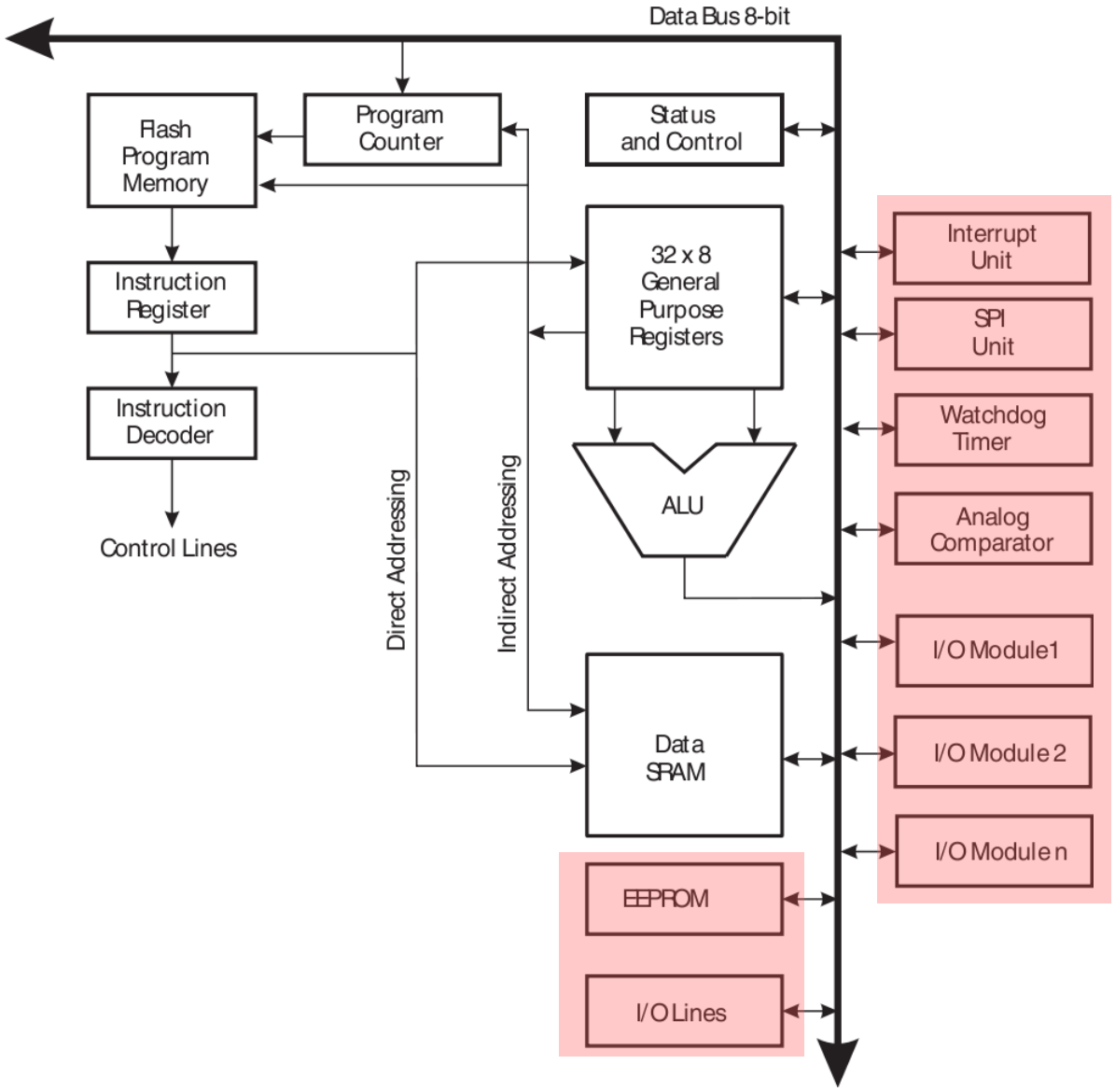

Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Instruction Register
32 x 8 General Purpose Registers
Interrupt Unit
SPI Unit
Instruction Decoder
Watchdog Timer
ALU
Analog Comparator
Control Lines
Direct Addressing
Indirect Addressing
I/O Module1
Data SRAM
I/O Module 2
I/O Module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

# RAM

- General Purpose Register and IO-Register mapped into SRAM address range
- ATmega1280 . . . 8k SRAM
- Multiple addressing modes
  - Register Direct
  - Data Direct
  - Data Indirect (Pointer)
  - Powerful displacement, pre-decrement, and post-increment modes!



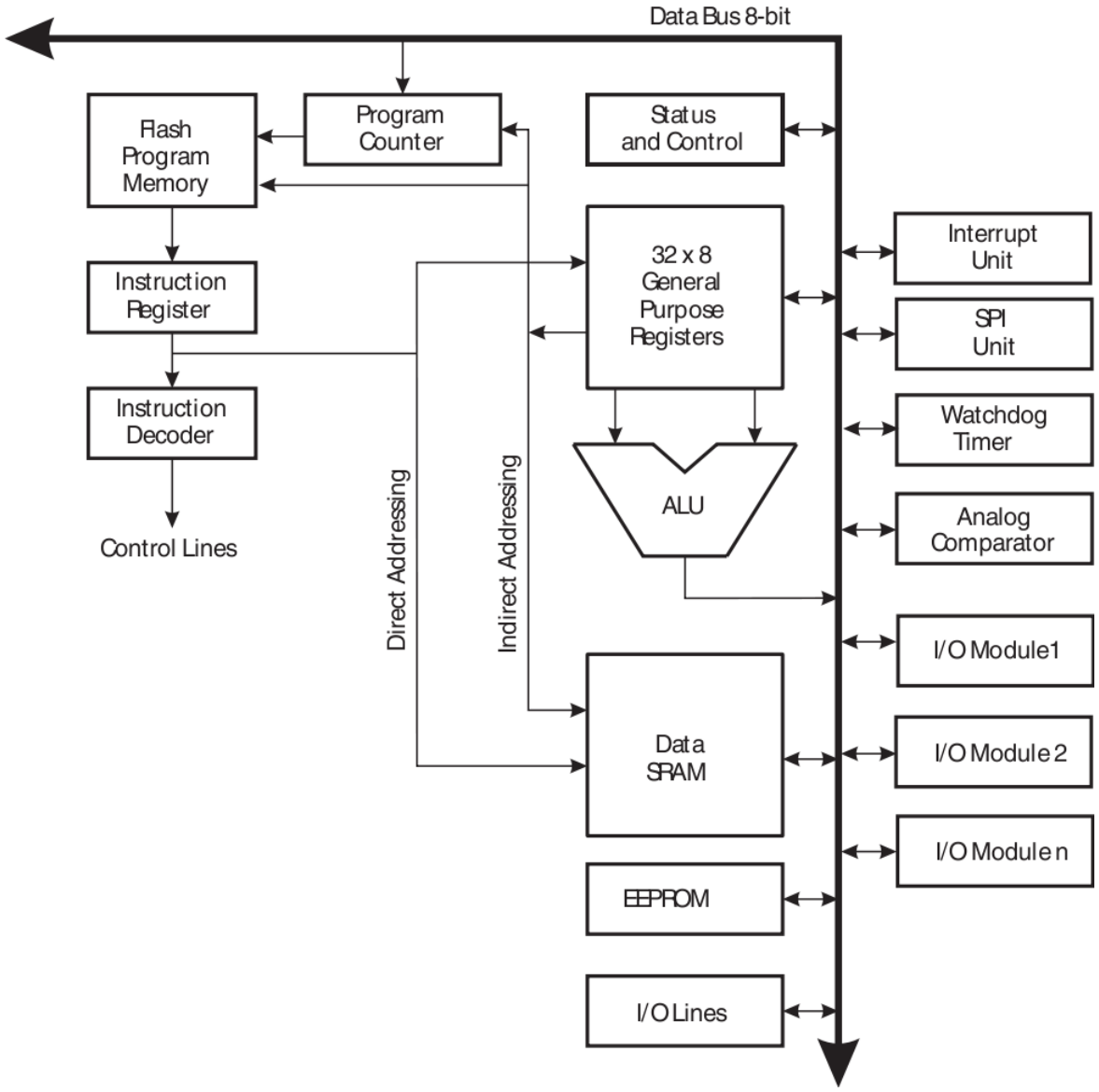
# AVR Core


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

# IO-Modules

- Configured by IO-Registers
  - Digital IO Ports: DDRx, PORTx, PINx
  - EEPROM: EEARH/L, EEDR, EECR
  - Ext. Interrupts: EICRA, EIMSK, EIFR
  - Timer
  - UART
  - ...
- Detailed explanations in the manual
- Depend on used microcontroller
- Beware @ ATmega1280: Opcodes IN and OUT only work on the 64 I/O Registers, not on the External ones!
- PORT{H,J,K,L}, TIMSK*, ADC, ... are all in the External range

# AVR Core


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

# Example Program

```
.NOLIST
.INCLUDE "m1280def.inc"
.LIST
; Name the registers you are using.
;***************************
.equ temp,  0x10;r16
.equ temp1, 0x11;r17
.equ temp2, 0x12;r18
;***************************
.section .text

.global main

.org 0x0000
  jmp main

toupper:
  push temp2

  mov temp2, temp2     ;backup x in temp2
  subi temp, 'a'       ;subtract 'a' from x
  brlo end_of_sub      ;if x < 'a' do nothing
                       ;else
  subi temp, 'z'-'a'   ;subtract 'z'-'a' from x
  brlo end_of_sub      ;if x-'a' >= 'z'-'a'+1
                       ;do nothing. else
  subi temp2, 0x20     ;toupper backup
end_of_sub:
  mov temp, temp2      ;store back to temp
  pop temp2
  ret

; This is is the main program.
main:
  ;initialize stack pointer
  ldi temp, lo8(RAMEND)
  out SPL,  temp
  ldi temp, hi8(RAMEND)
  out SPH,  temp
  ;initialize PORTA as output and zero
  ldi temp,  0x00
  out PORTA, temp
  ldi temp,  0xFF
  out DDRA,  temp
  ;initialize Z to point at data
  ldi temp, lo8(mydata)
  out ZL,   temp
  ldi temp, hi8(mydata)
  out ZH,   temp

loop:
  lpm temp, Z+       ;read data from flash
                     ;using post increment
  cpi temp, 0x00
  breq end           ;got termination
  call toupper       ;call toupper(temp)
  out PORTA, temp    ;output
  jmp loop
end:
  jmp end
mydata:
.byte 'H', 'e', 'l', 'l', 'o', ' ',
      'W', 'o', 'r', 'l', 'd', '!', '\0'
```

# Example Program

```
.NOLIST
.INCLUDE "m1280def.inc"
.LIST
; Name the registers you are using.
;***************************
.equ temp,  0x10;r16
.equ temp1, 0x11;r17
.equ temp2, 0x12;r18
;***************************
.section .text

.global main

.org 0x0000
  jmp main
```

Unconditional Jump

```
toupper:
  push temp2

  mov temp2, temp2     ;backup x in temp2
  subi temp, 'a'       ;subtract 'a' from x
  brlo end_of_sub      ;if x < 'a' do nothing
                       ;else
  subi temp, 'z'-'a'   ;subtract 'z'-'a' from x
  brlo end_of_sub      ;if x-'a' >= 'z'-'a'+1
                       ;do nothing. else
  subi temp2, 0x20     ;toupper backup
end_of_sub:
  mov temp, temp2      ;store back to temp
  pop temp2
  ret

; This is is the main program.
main:
  ;initialize stack pointer
  ldi temp, lo8(RAMEND)
  out SPL,  temp
  ldi temp, hi8(RAMEND)
  out SPH,  temp
  ;initialize PORTA as output and zero
  ldi temp,  0x00
  out PORTA, temp
  ldi temp,  0xFF
  out DDRA,  temp
  ;initialize Z to point at data
  ldi temp, lo8(mydata)
  out ZL,   temp
  ldi temp, hi8(mydata)
  out ZH,   temp

loop:
  lpm temp, Z+       ;read data from flash
                     ;using post increment
  cpi temp, 0x00
  breq end           ;got termination
  call toupper       ;call toupper(temp)
  out PORTA, temp    ;output
  jmp loop
end:
  jmp end
mydata:
.byte 'H', 'e', 'l', 'l', 'o', ' ',
      'W', 'o', 'r', 'l', 'd', '!', '\0'
```

# Example Program

```
.NOLIST
.INCLUDE "m1280def.inc"
.LIST
; Name the registers you are using.
;***************************
.equ temp,  0x10;r16
.equ temp1, 0x11;r17
.equ temp2, 0x12;r18
;***************************
.section .text

.global main

.org 0x0000
  jmp main

toupper:
  push temp2

  mov temp2, temp2     ;backup x in temp2
  subi temp, 'a'       ;subtract 'a' from x
  brlo end_of_sub      ;if x < 'a' do nothing
                       ;else
  subi temp, 'z'-'a'   ;subtract 'z'-'a' from x
  brlo end_of_sub      ;if x-'a' >= 'z'-'a'+1
                       ;do nothing. else
  subi temp2, 0x20     ;toupper backup
end_of_sub:
  mov temp, temp2      ;store back to temp
  pop temp2
  ret
```

```
; This is is the main program.
main:
  ;initialize stack pointer
  ldi temp, lo8(RAMEND)
  out SPL,  temp
  ldi temp, hi8(RAMEND)
  out SPH,  temp
  ;initialize PORTA as output and zero
  ldi temp,  0x00
  out PORTA, temp
  ldi temp,  0xFF
  out DDRA,  temp
  ;initialize Z to point at data
  ldi temp, lo8(mydata)
  out ZL,   temp
  ldi temp, hi8(mydata)
  out ZH,   temp

loop:
  lpm temp, Z+      ;read data from flash
                    ;using post increment
  cpi temp, 0x00
  breq end
  call toupper      ;call toupper
  out PORTA, temp   ;output
  jmp loop
end:
  jmp end
mydata:
.byte 'H', 'e', 'l', 'l', 'o', ' ',
      'W', 'o', 'r', 'l', 'd', '!', '\0'
```

Conditional Jump

# Example Program

```
.NOLIST
.INCLUDE "m1280def.inc"
.LIST
; Name the registers you are using.
;***************************
.equ temp,  0x10;r16
.equ temp1, 0x11;r17
.equ temp2, 0x12;r18
;***************************
.section .text

.global main

.org 0x0000
  jmp main
```

Function

```
toupper:
  push temp2

  mov temp2, temp2     ;backup x in temp2
  subi temp, 'a'       ;subtract 'a' from x
  brlo end_of_sub      ;if x < 'a' do nothing
                       ;else
  subi temp, 'z'-'a'   ;subtract 'z'-'a' from x
  brlo end_of_sub      ;if x-'a' >= 'z'-'a'+1
                       ;do nothing. else
  subi temp2, 0x20     ;toupper backup
end_of_sub:
  mov temp, temp2      ;store back to temp
  pop temp2
  ret
```

```
; This is is the main program.
main:
  ;initialize stack pointer
  ldi temp, lo8(RAMEND)
  out SPL,  temp
  ldi temp, hi8(RAMEND)
  out SPH,  temp
  ;initialize PORTA as output and zero
  ldi temp,  0x00
  out PORTA, temp
  ldi temp,  0xFF
  out DDRA,  temp
  ;initialize Z to point at data
  ldi temp, lo8(mydata)
  out ZL,   temp
  ldi temp, hi8(mydata)
  out ZH,   temp

loop:
  lpm temp, Z+      ;read data from flash
                    ;using post increment
  cpi temp, 0x00
  breq end          ;got termination
  call toupper      ;call toupper(temp)
  out PORTA, temp   ;output
  jmp loop
end:
  jmp end
mydata:
.byte 'H', 'e', 'l', 'l', 'o', ' ',
      'W', 'o', 'r', 'l', 'd', '!', '\0'
```

# Example Program

```
.NOLIST
.INCLUDE "m1280def.inc"
.LIST
; Name the registers you are using.
;***************************
.equ temp,  0x10;r16
.equ temp1, 0x11;r17
.equ temp2, 0x12;r18
;***************************
.section .text

.global main

.org 0x0000
  jmp main

toupper:
  push temp2

  mov temp2, temp2     ;backup x in temp2
  subi temp, 'a'       ;subtract 'a' from x
  brlo end_of_sub      ;if x < 'a' do nothing
                       ;else
  subi temp, 'z'-'a'   ;subtract 'z'-'a' from x
  brlo end_of_sub      ;if x-'a' >= 'z'-'a'+1
                       ;do nothing. else
  subi temp2, 0x20     ;toupper backup
end_of_sub:
  mov temp, temp2      ;store back to temp
  pop temp2
  ret
```

```
; This is is the main program.
main:
  ;initialize stack pointer
  ldi temp, lo8(RAMEND)
  out SPL,  temp
  ldi temp, hi8(RAMEND)
  out SPH,  temp
  ;initialize PORTA as output and zero
  ldi temp,  0x00
  out PORTA, temp
  ldi temp,  0xFF
  out DDRA,  temp
  ;initialize Z to point at data
  ldi temp, lo8(mydata)
  out ZL,   temp
  ldi temp, hi8(mydata)
  out ZH,   temp

loop:
  lpm temp, Z+      ;read data from flash
                    ;using post increment
  cpi temp, 0x00
  breq end          ;get termination
  call toupper      ;toupper(temp)
  out PORTA, temp   ;output
  jmp loop
end:
  jmp end
mydata:
.byte 'H', 'e', 'l', 'l', 'o', ' ',
      'W', 'o', 'r', 'l', 'd', '!', '\0'
```

Function Call

# Example Program

```
.NOLIST
.INCLUDE "m1280def.inc"
.LIST
; Name the registers you are using.
;***************************
.equ temp,  0x10;r16
.equ temp1, 0x11;r17
.equ temp2, 0x12;r18
;***************************
.section .text

.global main

.org 0x0000
  jmp main

toupper:
  push temp2

  mov temp2, temp2     ;backup x in temp2
  subi temp, 'a'       ;subtract 'a' from x
  brlo end_of_sub      ;if x < 'a' do nothing
                       ;else
  subi temp, 'z'-'a'   ;subtract 'z'-'a' from x
  brlo end_of_sub      ;if x-'a' >= 'z'-'a'+1
                       ;do nothing. else
  subi temp2, 0x20     ;toupper backup
end_of_sub:
  mov temp, temp2      ;store back to temp
  pop temp2
  ret
```

```
; This is is the main program.
main:
  ;initialize stack pointer
  ldi temp, lo8(RAMEND)
  out SPL,  temp
  ldi temp, hi8(RAMEND)
  out SPH,  temp
  ;initialize PORT
  ldi temp,  0x00
  out PORTA, temp
  ldi temp,  0xFF
  out DDRA,  temp
  ;initialize Z to point at data
  ldi temp, lo8(mydata)
  out ZL,   temp
  ldi temp, hi8(mydata)
  out ZH,   temp

loop:
  lpm temp, Z+       ;read data from flash
                     ;using post increment
  cpi temp, 0x00
  breq end           ;got termination
  call toupper       ;call toupper(temp)
  out PORTA, temp    ;output
  jmp loop
end:
  jmp end
mydata:
.byte 'H', 'e', 'l', 'l', 'o', ' ',
      'W', 'o', 'r', 'l', 'd', '!', '\0'
```

Register Direct

# Example Program

```
.NOLIST
.INCLUDE "m1280def.inc"
.LIST
; Name the registers you are using.
;***************************
.equ temp,  0x10;r16
.equ temp1, 0x11;r17
.equ temp2, 0x12;r18
;***************************
.section .text

.global main

.org 0x0000
  jmp main

toupper:
  push temp2

  mov temp2, temp2     ;backup x in temp2
  subi temp, 'a'       ;subtract 'a' from x
  brlo end_of_sub      ;if x < 'a' do nothing
                       ;else
  subi temp, 'z'-'a'   ;subtract 'z'-'a' from x
  brlo end_of_sub      ;if x-'a' >= 'z'-'a'+1
                       ;do nothing. else
  subi temp2, 0x20     ;toupper backup
end_of_sub:
  mov temp, temp2      ;store back to temp
  pop temp2
  ret
```

```
; This is is the main program.
main:
  ;initialize stack pointer
  ldi temp, lo8(RAMEND)
  out SPL,  temp
  ldi temp, hi8(RAMEND)
  out SPH,  temp
  ;initialize PORTA as output and zero
  ldi temp,  0x00
  out PORTA, temp
  ldi temp,  0xFF
  out DDRA,  temp
  ;initialize Z to point at data
  ldi temp, lo8(mydata)
  out ZL,   temp
  ldi temp, hi8(mydata)
  out ZH,   temp

loop:
  lpm temp, Z+       ;read data from flash
                     ;using post increment
  cpi temp, 0x00
  breq end           ;got termination
  call toupper       ;call toupper(temp)
  out PORTA, temp    ;output
  jmp loop
end:
  jmp end
mydata:
.byte 'H', 'e', 'l', 'l', 'o', ' ',
      'W', 'o', 'r', 'l', 'd', '!', '\0'
```

IO-Register Direct

# Example Program

```
.NOLIST
.INCLUDE "m1280def.inc"
.LIST
; Name the registers you are using.
;***************************
.equ temp,  0x10;r16
.equ temp1, 0x11;r17
.equ temp2, 0x12;r18
;***************************
.section .text

.global main

.org 0x0000
  jmp main

toupper:
  push temp2

  mov temp2, temp2     ;backup x in temp2
  subi temp, 'a'       ;subtract 'a' from x
  brlo end_of_sub      ;if x < 'a' do nothing
                       ;else
  subi temp, 'z'-'a'   ;subtract 'z'-'a' from x
  brlo end_of_sub      ;if x-'a' >= 'z'-'a'+1
                       ;do nothing. else
  subi temp2, 0x20     ;toupper backup
end_of_sub:
  mov temp, temp2      ;store back to temp
  pop temp2
  ret
```

```
; This is is the main program.
main:
  ;initialize stack pointer
  ldi temp, lo8(RAMEND)
  out SPL,  temp
  ldi temp, hi8(RAMEND)
  out SPH,  temp
  ;initialize PORTA as output and zero
  ldi temp,  0x00
  out PORTA, temp
  ldi temp,  0xFF
  out DDRA,  temp
  ;initialize Z to point at data
  ldi temp, lo8(mydata)
  out ZL,   temp
  ldi temp, hi8(mydata)
  out ZH,   temp

loop:
  lpm temp, Z+
  cpi temp, 0x00
  breq end          ;got termination
  call toupper      ;call toupper(temp)
  out PORTA, temp   ;output
  jmp loop
end:
  jmp end
mydata:
.byte 'H', 'e', 'l', 'l', 'o', ' ',
      'W', 'o', 'r', 'l', 'd', '!', '\0'
```

Program Indirect – Post Increment

# Getting Code onto the Microcontroller Chip

- Starting with an assembler program
- Goal: running program on microcontroller
- Question: HOW?

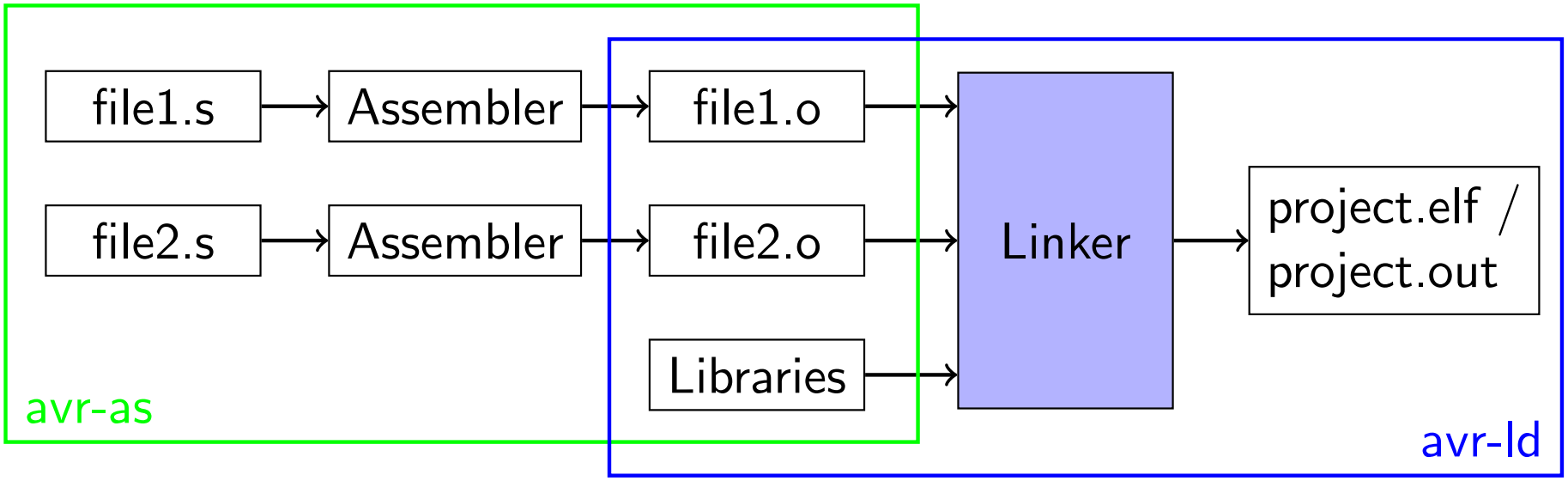
# Starting with an Assembler File

- Assembler file not directly download-able because of:
  - File is not binary
  - Includes (m1280def.inc)
  - Names of Registers (temp)
  - Calculations ('z'-'a')
  - Jumplabels (toupper)
  - Comments
  - ...
- File needs to be assembled so that:
  - Mnemonics replaced by their opcodes
  - Includes need to be included
  - Registernames translated to addresses
  - Calculations are replaced by their results
  - Jumplables replaced by addresses
  - Comments deleted
  - ...

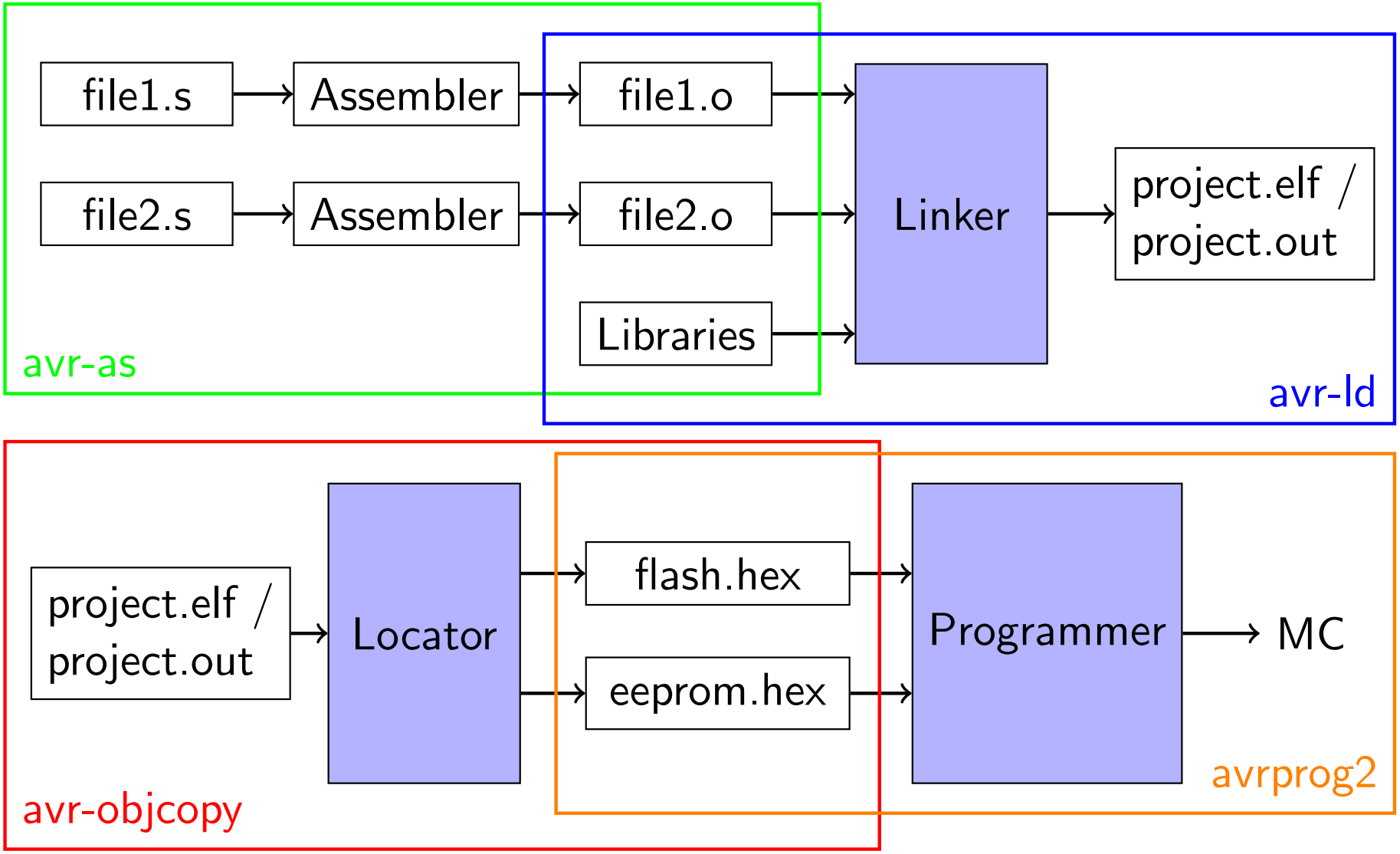
# Easy Way – Makefile

- Use Makefile and type `make` and `make install` to assemble and download program
- Problem: Makefile might not be there (e.g., new toolchain or first MC exam!)
- Solution: Get to know the toolchain and write one (Ex 2.1.2)

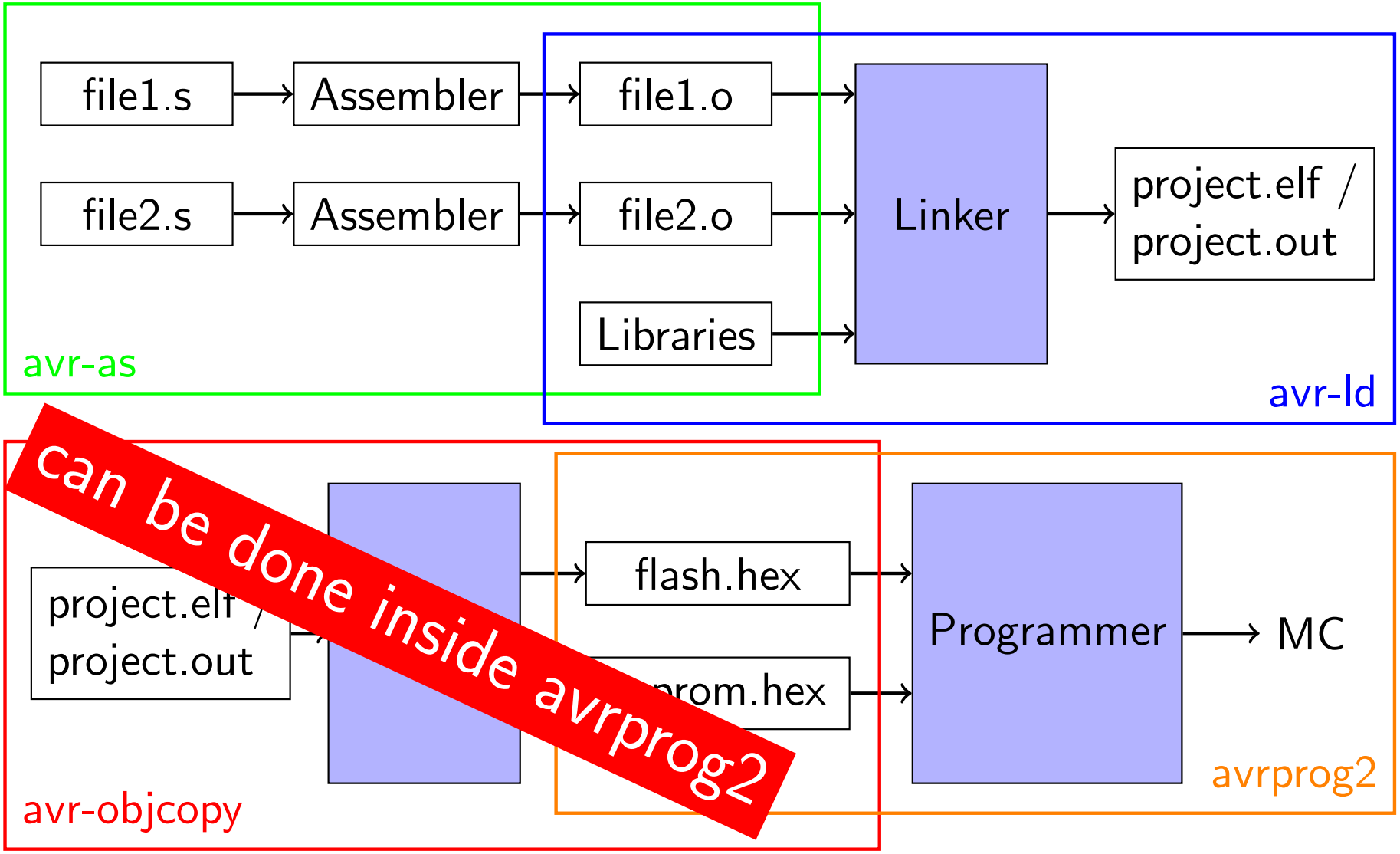
# Generating an Executable for a Microcontroller



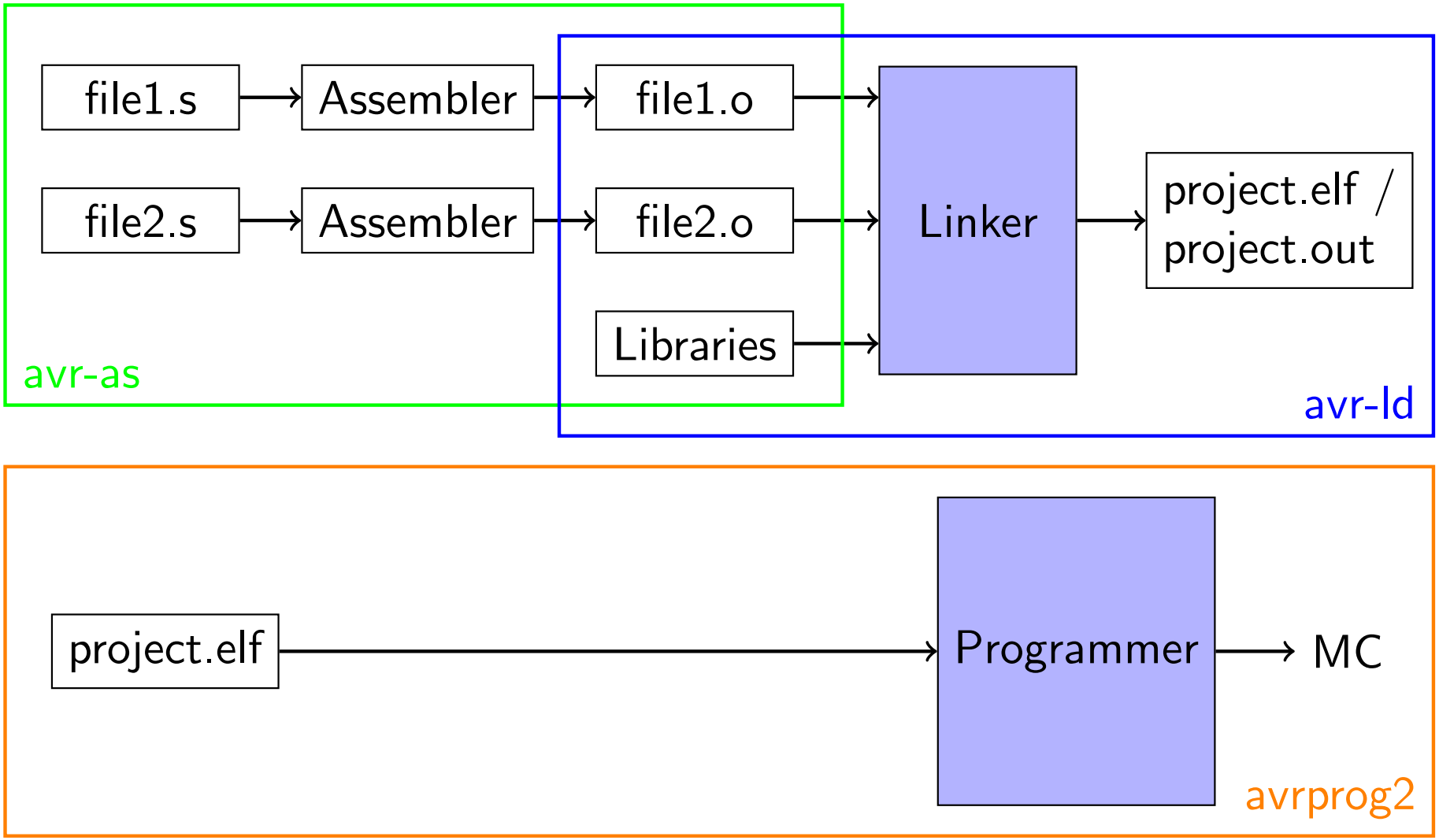
# Generating an Executable for a Microcontroller



# Generating an Executable for a Microcontroller



# Generating an Executable for a Microcontroller



# AVR - Toolchain

- Assembler
  - Transforms Mnemonics into opcodes (binary)
  - Performs inline calculations
  - Includes other files
- Linker:
  - Links multiple files
  - Calculates addresses (Jumps to ISR!)
- Need to be split to fit the physical realization of microcontroller (Flash, EEPROM, RAM)
  - .text $\rightarrow$ Flash
  - .eeprom $\rightarrow$ EEPROM
  - .data $\rightarrow$ SRAM ?!? (RAM not directly programmable !!!!! ) $\rightarrow$ in C initialized from Flash by init-code automatically linked to program
- Done by avr-objcopy (or by the programmer)

# AVR - Toolchain

- Featuring many other useful programs
  - avr-objdump (disassemble object files)
  - avr-size
  - ...
- Programs have various options

# Programming

- Several programming modes
  - Parallel
  - SPI
  - JTAG (also debugging capabilities)
  - Bootloader
- Various programmers
- @lab: avrprog2
  - USB host interface
  - SPI target interface
  - UNIX command line interface (maintained by us)
- Not only Flash and EEPROM are programmable, but also
  - Fuses (WARNING — know what you are doing — no need to change anything at the lab hardware)
  - Lock Bits

# Weekly Training Objective

- This week
  - 1.2 Board test
  - 2.1.1 Assembler demo program
  - 2.1.2 Makefile
  - 2.2.1 Logical operations
- Next week
  - 2.2.2 Input with floating pins
  - 2.2.4 Monoflop buttons
  - 2.2.5 Digital I/O
  - 2.4.1 precompiled LCD
- Until Exam (in 2.5 weeks)
  - 2.2.8 LED curtain
  - 2.4.2 Calling conventions I
  - 2.4.3 Calling conventions II
  - 2.4.4 Fibonacci Numbers

Any Questions?