Overview

- Data transfer instructions in AVR
- Sample AVR assembly programs using data transfer instructions

Motivations

- How to transfer data between two registers?
- How to transfer data between memory and a register?
- How to transfer a constant to a register?

Selected Data Transfer Instructions

- mov, movw
- ldi, ld, ldd, lds
- st, sts
- lpm
- in, out
- Push, pop
- Refer to the main textbook and AVR Instruction Set for a complete list.
Copy Register

- Syntax: \texttt{mov Rd, Rr}
- Operands: \(Rd, Rr \in \{r0, r1, \ldots, r31\}\)
- Operation: \(Rd \leftarrow Rr\)
- Flag affected: None
- Encoding: 0010 11rd dddd rrrr
- Words: 1
- Cycles: 1
- Example:

  \texttt{mov r1, r0} ; Copy r0 to r1

Copy Register Pair

- Syntax: \texttt{movw Rd+1:Rd, Rr+1:Rr}
- Operands: \(d, r \in \{0, 2, \ldots, 28, 30\}\)
- Operation: \(Rd+1:Rd \leftarrow Rr+1:Rr\)
- Flag affected: None
- Encoding: 0000 0001 dddd rrrr
- Words: 1
- Cycles: 1
- Example:

  \texttt{movw r21:r20, r1:r0} ; Copy r1:r0 to r21:r20

Load Immediate

- Syntax: \texttt{ldi Rd, k}
- Operands: \(Rd \in \{r16, r17, \ldots, r31\}\) and \(0 \leq k \leq 255\)
- Operation: \(Rd \leftarrow k\)
- Flag affected: None
- Encoding: 1110 kkkk dddd kkkk
- Words: 1
- Cycles: 1
- Example:

  \texttt{ldi r16, $42} ; Load $42 to r16

Load Indirect

- Syntax: \texttt{ld Rd, v}
- Operands: \(Rd \in \{r0, r1, \ldots, r31\}\) and \(v \in \{x, x+, -x, y, y+, -y, z, z+, -z\}\)
- Operation:
  
  (i) \(Rd \leftarrow (v)\) if \(v \in \{x, y, z\}\)
  
  (ii) \(x \leftarrow x-1\) and \(Rd \leftarrow (x)\) if \(v = x\)
  
  \(y \leftarrow y-1\) and \(Rd \leftarrow (y)\) if \(v = y\)
  
  \(z \leftarrow z-1\) and \(Rd \leftarrow (z)\) if \(v = z\)
  
  (iii) \(Rd \leftarrow (x)\) and \(x \leftarrow x+1\) if \(v = x+\)
  
  \(Rd \leftarrow (y)\) and \(y \leftarrow y+1\) if \(v = y+\)
  
  \(Rd \leftarrow (z)\) and \(z \leftarrow z+1\) if \(v = z+\)
- Flag affected: None
- Encoding: Depends on \(v\). Refer to AVR Instruction Set for details
- Words: 1
- Cycles: 2
- Comments: Post-inc and pre-dec are used to load contiguous data.
Load Indirect (Cont.)

- Example: 4-byte integer addition

```assembly
.def loop_counter = r20
.equ loop_bound = 4
.dseg
    int1: .byte 4   ; Allocate 4 bytes to the first integer;
    int2: .byte 4   ; Allocate 4 byte to the second integer;
    int3: .byte 4   ; Allocate 4 byte to store the result
.cseg
    ldi r26, low(int1) ; Load low byte of the address of the 1st int
    ldi r27, high(int1) ; Load high byte of the address of the 2nd int
    ldi r28, low(int2)  
    ldi r29, high(int2) 
    ldi r30, low(int3)  
    ldi r31, high(int3) 
    clr loop_counter  ; loop_counter=0
    ld r0, x+          ; Load byte 0 of the 1st int
    ld r1, y+          ; Load the byte 0 of the 2nd int
    add r1, r0         ; Add bytes 0
    st z+, r1          ; Store byte 0 of the result
    ld r0, x+          ; Load byte 1 of the 1st int
    ld r1, y+          ; Load the byte 1 of the 2nd int
    adc r1, r0         ; Add bytes 1 with carry
    st z+, r1          ; Store byte 1 of the result
```

Load Indirect (Cont.)

```assembly
ld r0, x+ ; Load byte 0 of the 1st int
ld r1, y+ ; Load the byte 0 of the 2nd int
add r1, r0 ; Add bytes 0
st z+, r1 ; Store byte 0 of the result
ld r0, x+ ; Load byte 1 of the 1st int
ld r1, y+ ; Load the byte 1 of the 2nd int
adc r1, r0 ; Add bytes 1 with carry
st z+, r1 ; Store byte 1 of the result
```

Load Indirect (Cont.)

```assembly
ld r1, y+ ; Load the byte 2 of the 2nd int
adc r1, r0 ; Add bytes 2 with carry
st z+, r1 ; Store byte 2 of the result
ld r0, x+ ; Load byte 3 of the 1st int
ld r1, y+ ; Load the byte 3 of the 2nd int
adc r1, r0 ; Add bytes 3 with carry
st z+, r1 ; Store byte 3 of the result
```

Load Indirect with Displacement

- Syntax: ldd Rd, v
- Operands: Rd∈{r0, r1, …, r31} and v∈{y+q, z+q}
- Operation: Rd←(v)
- Flag affected: None
- Encoding: Depends on v. Refer to AVR Instruction Set for details
- Words: 1
- Cycles: 2
- Example: clr r31 ; Clear Z high byte
  ldi r30, $60 ; Set Z low byte to $60
  ld r0, Z+ ; Load r0 with data space loc. $60(Z post inc)
  ld r1, Z ; Load r1 with data space loc. $61
  ldi r30, $63 ; Set Z low byte to $63
  ld r2, Z ; Load r2 with data space loc. $63
  ld r3, -Z ; Load r3 with data space loc. $62(Z pre dec)
  ldd r4, Z+2 ; Load r4 with data space loc. $64
- Comments: ldd is used to load an element of a structure.
Store Indirect

- **Syntax:**  
  \[ \text{st} \ v, \ Rr \]
- **Operands:**  
  \[ Rr \in \{r0, r1, \ldots, r31\} \text{ and } v \in \{x, x+, -x, y, y+, -y, z, z+, -z\} \]
- **Operation:**  
  (i) \( (v) \leftarrow Rr \) if \( v \in \{x, y, z\} \)
  
  (ii) \( x \leftarrow x-1 \) and \( (x) \leftarrow Rr \) if \( v = -x \)
  
  \( y \leftarrow y-1 \) and \( (y) \leftarrow Rr \) if \( v = -y \)
  
  \( z \leftarrow z-1 \) and \( (z) \leftarrow Rr \) if \( v = -z \)
  
  (iii) \( (x) \leftarrow Rr \) and \( x \leftarrow x+1 \) if \( v = x+ \)
  
  \( (y) \leftarrow Rr \) and \( y \leftarrow y+1 \) if \( v = y+ \)
  
  \( (z) \leftarrow Rr \) and \( z \leftarrow z+1 \) if \( v = z+ \)

- **Flag affected:** None
- **Encoding:** Depends on \( v \). Refer to AVR Instruction Set for details
- **Words:** 1
- **Cycles:** 2
- **Comments:** Post-inc and pre-dec are used to store contiguous data.

**Example:**

```
clr r29 ; Clear Y high byte
ldi r28, $60 ; Set Y low byte to $60
st Y+, r0 ; Store r0 in data space loc. $60(Y post inc)
st Y, r1 ; Store r1 in data space loc. $61
ldi r28, $63 ; Set Y low byte to $63
st Y, r2 ; Store r2 in data space loc. $63
ldi r28, r3 ; Store r3 in data space loc. $62 (Y pre dec)
std Y+2, r4 ; Store r4 in data space loc. $64
```

Load Program Memory

- **Syntax:**  
  \[ \text{lpm} \]
- **Operands:**  
  (i) \( \text{lpm} \) None, \( R0 \) implied
  
  (ii) \( \text{lpm} \ Rd, Z \) \( 0 \leq d \leq 31 \)
  
  (iii) \( \text{lpm} \ Rd, Z+ \) \( 0 \leq d \leq 31 \)
- **Operations:**  
  (i) \( R0 \leftarrow (Z) \)
  
  (ii) \( Rd \leftarrow (Z) \)
  
  (iii) \( Rd \leftarrow (Z) \)
- **Flag affected:** None
- **Encoding:**
  (i) \( 1001 \ 0101 \ 1100 \ 1000 \)
  
  (ii) \( 1001 \ 000d \ dddd \ 0100 \)
  
  (iii) \( 1001 \ 000d \ dddd \ 0101 \)
- **Words:** 1
- **Cycles:** 3
- **Comments:** \( Z \) contains the byte address while the flash memory uses word addressing. Therefore, the word address must be converted into byte address before having access to data on flash memory.

**Example:**

```
ldi zh, high(Table_1<<1) ; Initialize Z pointer
ldi zl, low(Table_1<<1)
```

```
... Table_1: .dw 0x5876 ...
```

**Comments:** Table_1<<1 converts word address into byte address.

Store Indirect with Displacement

- **Syntax:**  
  \[ \text{std} \ v, \ Rr \]
- **Operands:**  
  (i) \( \text{std} \) \( Rd \in \{r0, r1, \ldots, r31\} \text{ and } v \in \{y+q, z+q\} \)
- **Operation:**  
  \( (v) \leftarrow Rr \)
- **Flag affected:** None
- **Encoding:** Depends on \( v \). Refer to AVR Instruction Set for details
- **Words:** 1
- **Cycles:** 2
- **Example:**

```
clr r29 ; Clear Y high byte
ldi r28, $60 ; Set Y low byte to $60
st Y+, r0 ; Store r0 in data space loc. $60(Y post inc)
st Y, r1 ; Store r1 in data space loc. $61
ldi r28, $63 ; Set Y low byte to $63
st Y, r2 ; Store r2 in data space loc. $63
ldi r28, r3 ; Store r3 in data space loc. $62 (Y pre dec)
std Y+2, r4 ; Store r4 in data space loc. $64
```

**Comments:** std is used to store an element of a structure.
Load Program Memory (Cont.)

- **Example**
  
  ```asm
  ldi zh, high(Table_1<<1) ; Initialize Z pointer
  ldi zl, low(Table_1<<1)
  lpm r16, z+ ; r16=0x76
  lpm r17, z ; r17=0x58
  ...
  Table_1: .dw 0x5876
  ...
  ``
  
  - **Comments:** Table_1<<1 converts word address into byte address

Load an I/O Location to Register

- **Syntax:** `in  Rd, A`
- **Operands:** `Rd ∈ {r0, r1, ..., r31} and 0≤A≤63`
- **Operation:** `I/O (A) ← Rd`
  
  Loads one byte from the location `A` in the I/O Space (Ports, Timers, Configuration registers etc.) into register `Rd` in the register file.
  
  - **Flag affected:** None
  - **Encoding:** 1011 0AAd dddd AAAA
  - **Words:** 1
  - **Cycles:** 1
  - **Example:**
    ```asm
    in r25, $16 ; Read Port B
    cpi r25, 4 ; Compare read value to constant
    breq exit ; Branch if r25=4
    ...
    exit: nop ; Branch destination (do nothing)
    ```

Store Register to an I/O Location

- **Syntax:** `out  A, Rr`
- **Operands:** `Rr ∈ {r0, r1, ..., r31} and 0≤A≤63`
- **Operation:** `I/O (A) ← Rr`
  
  Store the byte in register `Rr` to the I/O location (register).
  
  - **Flag affected:** None
  - **Encoding:** 1011 1AAr rrrr AAAA
  - **Words:** 1
  - **Cycles:** 1
  - **Example:**
    ```asm
    clr r16 ; Clear r16
    ser r17 ; Set r17 to $ff
    out $18, r16 ; Write zeros to Port B
    nop ; Wait (do nothing)
    out $18, r17 ; Write ones to Port B
    ```

Push Register on Stack

- **Syntax:** `push Rr`
- **Operands:** `Rr ∈ {r0, r1, ..., r31}`
- **Operation:** `SP ← (SP) – 1`
  
  - **Flag affected:** None
  - **Encoding:** 1001 001d dddd 1111
  - **Words:** 1
  - **Cycles:** 2
  - **Example**
    ```asm
    call routine ; Call subroutine
    ...
    routine: push r14 ; Save r14 on the stack
              push r13 ; Save r13 on the stack
    ...
    pop r13 ; Restore r13
    pop r14 ; Restore r14
    ret ; Return from subroutine
    ```
Pop Register from Stack

- Syntax: `pop Rr`
- Operands: `Rr ∈ {r0, r1, ..., r31}`
- Operation: `Rr ← (SP)`
  `SP ← SP +1`
- Flag affected: None
- Encoding: `1000 000d dddd 1111`
- Words: 1
- Cycles: 2
- Example:
  ```
  call routine ; Call subroutine
  ...
  routine:
  push r14             ; Save r14 on the stack
  push r13 ; Save r13 on the stack
  ...
  pop r13 ; Restore r13
  pop r14 ; Restore r14
  ret ; Return from subroutine
  ```

Lower-Case to Upper-Case

The following AVR assembly program converts 5 lower-case letters in a string which stored in the program memory (FLASH memory) into the upper-case letters and stores the resulting string into the data memory (SRAM).

```
.include "m64def.inc"
.equ size =5 ; Define size to be 5
.def counter =r17 ; Define counter to be r17
dseg ; Define a data segment
.org 0x100 ; Set the starting address of data segment to 0x100
.Cap_string: .byte 5 ; Allocate 5 bytes of data memory (SRAM) to store the string of upper-case letters.
cseg ; Define a code segment
.Low_string: .db "hello" ; Define the string "hello" which is stored in the program (Flash) memory.
```

Lower-Case to Upper-Case (Cont.)

```assembly
ldi zl, low(Low_string<<1) ; Get the low byte of the address of "h"
ldi zh, high(Low_string<<1) ; Get the high byte of the address of "h"
ldi yl, low(Cap_string)
ldi yh, high(Cap_string)
clr counter ; counter=0
```

Lower-Case to Upper-Case (Cont.)

```assembly
main:
  lpm  r20, z+ ; Load a letter from flash memory
  subi r20, 32 ; Convert it to the capital letter
  st  y+,r20 ; Store the capital letter in SRAM
  inc  counter
  cpi  counter, size brlt main
loop: nop
rjmp loop
```
Reading Material

1. AVR Instruction Set.