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.

Overview

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

- Syntax: `mov Rd, Rr`
- Operands: `Rd, Rr ∈ {r0, r1, ..., r31}`
- Operation: `Rd ← Rr`
- Flag affected: None
- Encoding: `0010 11rd dddd rrr`
- Words: 1
- Cycles: 1
- Example: `mov r1, r0 ; Copy r0 to r1`

Copy Register Pair

- Syntax: `movw Rd+1, Rr+1`
- Operands: `d, r ∈ {0, 2, ..., 28, 30}`
- Operation: `Rd+1 ← Rr+1`
- Flag affected: None
- Encoding: `0000 0001 dddd rrr`
- Words: 1
- Cycles: 1
- Example: `movw r21:20, r1:0 ; Copy r1:0 to r21:20`

Load Immediate

- Syntax: `ldi Rd, k`
- Operands: `Rd ∈ {r16, r17, ..., r31}` and `0 ≤ k ≤ 255`
- Operation: `Rd ← k`
- Flag affected: None
- Encoding: `1110 kkkk dddd kkkk`
- Words: 1
- Cycles: 1
- Example: `ldi r16, $42 ; Load $42 to r16`

Load Indirect

- Syntax: `ld Rd, v`
- Operands: `Rd ∈ {r0, r1, ..., r31}` and `v ∈ {x, x+, -x, y, y+, -y, z, z+, -z}`
- Operation: 
  1. `Rd ← (v)` if `v ∈ {x, y, z}`
  2. `x ← x - 1` and `Rd ← (x)` if `v = -x`
  3. `Rd ← (y)` if `v = -y`
  4. `z ← z - 1` and `Rd ← (z)` if `v = -z`
  5. `Rd ← (x)` and `x ← x + 1` if `v = x +`
  6. `Rd ← (y)` and `y ← y + 1` if `v = y +`
  7. `Rd ← (z)` and `z ← 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 bytes 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
```

Load Indirect (Cont.)

• Example: 4-byte integer addition.

```assembly
loop:
  ld r0, x+ ; Load the next byte of the 1st int
  ld r1, y+ ; Load the next byte of the 2nd int
  inc loop_counter
  cpi loop_counter, 1 ; Least significant byte?
  breq first_byte ; Add two bytes with carry
  ade r1, r0
  jmp store
  first_byte:
  add r1, r0 ; Add two least significant bytes
  store:
  st z+, r1 ; Store the result
  cpi loop_counter, loop_bound ; End of loop?
  bret
  first_byte:
  adcr1, r0 ; Add two least significant bytes
  jmp store
  store:
  st z+, r1 ; Store the result
  cpi loop_counter, loop_bound ; End of loop?
  bret
```

Load Indirect with Displacement

• Syntax: ldd Rd, v
• Operands: Rd∈{r0, r1, ..., r31} and v∈{y+s, z+s}
• Operation: Rd←(v)
• Flag affected: None
• Encoding: Depends on v. Refer to AVR Instruction Set for details
• Words: 1
• Cycles: 2
• Example: ldd r31, $60 ; 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
  ldi 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: st v, Rr
• Operands: Rr∈{r0, r1, ..., r31} and v∈{x, x+, -x, y, y+, -y, z, z+, -z}
• Operation: (i) v←Rr if v∈{x, y, z}
  (ii) x←x-1 and (y)←Rr if v=x
       y←y-1 and (y)←Rr if v=y
       z←z-1 and (z)←Rr if v=z
  (iii) (x)←Rr and x←x+1 if v=x+
       (y)←Rr and y←y+1 if v=y+
       (z)←Rr and z←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.
Store Indirect with Displacement

- Syntax: \( \text{std } v, Rr \)
- Operands: \( Rd \in \{r0, r1, \ldots, r31\} \) 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:
  
  \[
  \begin{align*}
  \text{clr r29} & ; \text{Clear Y high byte} \\
  \text{ldi r28, } & S60 \ ; \text{Set Y low byte to } S60 \\
  \text{st} & Y+, r0 \ ; \text{Store r0 in data space loc. } 60(\text{Y post inc}) \\
  \text{st} & Y, r1 \ ; \text{Store r1 in data space loc. } S61 \\
  \text{ldi r28, } & S63 \ ; \text{Set Y low byte to } S63 \\
  \text{st} & Y, r2 \ ; \text{Store r2 in data space loc. } S63 \\
  \text{st} & -Y, r3 \ ; \text{Store r3 in data space loc. } S62 \ (\text{Y pre dec}) \\
  \text{std} & Y+2, r4 \ ; \text{Store r4 in data space loc. } S64 \\
  \end{align*}
  \]
- Comments: std is used to store an element of a structure.

Load Program Memory

- Syntax:
  
  \[
  (i) \text{ LPM} \\
  (ii) \text{ LPM Rd, Z} \\
  (iii) \text{ LPM Rd, Z+}
  \]
- Operands:
  
  \[
  \begin{align*}
  \text{None, R0 implied} & \quad R0\leftarrow (Z) \\
  0 \leq d \leq 31 & \quad Rd\leftarrow (Z) \\
  0 \leq d \leq 31 & \quad Rd\leftarrow (Z)
  \end{align*}
  \]
- Operations:
  
  \[
  (i) 1001 0101 1100 1000 \\
  (ii) 1001 000d dddd 0100 \\
  (iii) 1001 000d dddd 0101
  \]
- Flag affected: None
- Encoding:
  
  \[
  \begin{align*}
  \text{(i) } & 1001 0101 1100 1000 \\
  \text{(ii) } & 1001 000d dddd 0100 \\
  \text{(iii) } & 1001 000d dddd 0101
  \end{align*}
  \]
- 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.

Load Program Memory (Cont.)

- Example
  
  \[
  \begin{align*}
  \text{ldi zh, high(Table_1<<<1)} & \ ; \text{Initialize Z pointer} \\
  \text{ldi zl, low(Table_1<<<1)} & \\
  \text{lpm r16, z+} & \ ; r16=0x76 \\
  \text{lpm r17, z} & \ ; r17=0x58 \\
  \ldots \\
  \text{Table_1: .dw 0x5876} \\
  \ldots
  \end{align*}
  \]
- Comments: Table_1<<<1 converts word address into byte address

Load Program Memory (Cont.)

- Example
  
  \[
  \begin{align*}
  \text{ldi zh, high(Table_1<<<1)} & \ ; \text{Initialize Z pointer} \\
  \text{ldi zl, low(Table_1<<<1)} \\
  \text{lpm r16, z+} & \ ; r16=0x76 \\
  \text{lpm r17, z} & \ ; r17=0x58 \\
  \ldots \\
  \text{Table_1: .dw 0x5876} \\
  \ldots
  \end{align*}
  \]
- 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: Rd←I/O (A)
  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 0AAAd ddd dAAAA
- Words: 1
- Cycles: 1
- Example:
  
  in r25, S16 ; 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 1AAAr rrrr AAAAA
- Words: 1
- Cycles: 1
- Example:
  
  clr r16 ; Clear r16
sr 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)←Rr

  SP←SP−1
- Flag affected: None
- Encoding: 1001 001d ddd d1111
- 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

Pop Register from Stack

- Syntax: pop Rr
- Operands: Rr∈{r0, r1, ..., r31}
- Operation: Rr←(SP)

  SP←SP+1
- Flag affected: None
- Encoding: 1000 000d ddd d1111
- 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