Overview

- Program control instructions in AVR
- Stacks
- Sample AVR assembly programs using program control instructions

Motivations

- Arithmetic and logic instructions cannot change the program control flow.
- How to implement “if some condition holds then do task A else do task B”?
- How to call a subroutine?
- How to return to the caller from a function (subroutine)?
- How to return from an interrupt handler?

Selected Program Control Instructions

- Unconditional jump: jmp, rjmp, ijmp
- Subroutine call: rcall, icall, call
- Subroutine and interrupt return: ret, reti
- Conditional branching: breq, brne, brsh, brlo, brge, brlt, brvs, brvc, brie, brid
- Refer to the main textbook and AVR Instruction Set for a complete list.
### Jump

- **Syntax:** \texttt{jmp k}
- **Operands:** \(0 \leq k < 4M\)
- **Operation:** \(\text{PC} \leftarrow k\)
- **Flag affected:** None
- **Encoding:** 1001 010k kkkk 110k
- **Words:** 2
- **Cycles:** 3
- **Example:**
  - \texttt{mov r1, r0} ; Copy r0 to r1
  - \texttt{jmp farplc} ; Unconditional jump
  - \texttt{... farplc: inc r20} ; Jump destination

### Relative Jump

- **Syntax:** \texttt{rjmp k}
- **Operands:** \(-2K \leq k < 2K\)
- **Operation:** \(\text{PC} \leftarrow \text{PC} + k + 1\)
- **Flag affected:** None
- **Encoding:** 1100 kkkk kkkk kkkk
- **Words:** 1
- **Cycles:** 2
- **Example:**
  - \texttt{cmp r16, $42} ; Compare r16 to $42
  - \texttt{brne error} ; Branch to error if r16 \(\neq\) $42
  - \texttt{rjmp ok} ; Jump to ok
  - \texttt{error: add r16, r17} ; Add r17 to r16
  - \texttt{inc r16} ; Increment r16
  - \texttt{ok: mov r2, r20} ; Jump destination

### Indirect Jump

- **Syntax:** \texttt{ijmp}
- **Operation:**
  1. \(\text{PC} \leftarrow Z(15:0)\) Devices with 16 bits PC, 128K bytes program memory maximum.
  2. \(\text{PC}(15:0) \leftarrow Z(15:0)\) Devices with 22 bits PC, 8M bytes program memory maximum.
- **Flag affected:** None
- **Encoding:** 1001 0100 0000 1001
- **Words:** 1
- **Cycles:** 2

### Indirect Jump (Cont.)

- **Example:**
  - \texttt{clr r10} ; Clear r10
  - \texttt{ldi r20, 2} ; Load jump table offset
  - \texttt{ldi r30, low(Lab<<1)} ; High byte of the starting address (base) of jump table
  - \texttt{ldi r31, high(Lab<<1)} ; Low byte of the starting address (base) of jump table
  - \texttt{add r30, r20} ; Add r30, r20
  - \texttt{adc r31, r10} ; Base + offset is the address of the jump table entry
  - \texttt{ldm r0, Z+} ; Load low byte of the the jump table entry
  - \texttt{ldm r1, Z} ; Load high byte of the jump table entry
  - \texttt{movw r31:r30, r1:r0} ; Set the pointer register Z to point the target instruction
  - \texttt{ijmp} ; Jump to the target instruction
  - \texttt{... Lab: .dw jt_10} ; The first entry of the jump table
  - \texttt{... .dw jt_11} ; The second entry of the jump table
  - \texttt{... jt_10: nop} ;
  - \texttt{jt_11: nop} ;
  - \texttt{...
Stacks

- A stack is an area of memory that supports two operations
  - push – put something on the top of the stack
  - pop – take something off the top of the stack
- (LIFO – last in, first out)
- Every processor has a stack of some kind
  - Used for procedure calls (or subroutines) and interrupts
  - Used to store local variables in C
- Special register called a Stack Pointer (SP) stores the address of the top of the stack

AVR and Stacks

- Stacks are part of SRAM space.
- Stacks grow downwards (from a higher address to a lower address).
- SP needs to hold addresses (therefore 16 bits wide).
  - Made up of two 8 bit registers
    o SPH (high byte) (IO register $3E)
    o SPL (low byte) (IO register $3D)
- First thing to do in any program is to initialize the stack pointer.
  - Typically stacks use the top of SRAM space.

AVR Stack Initialization

```
.include "m64def.inc"
.def temp=r20
.cseg
ldi temp, low(RAMEND)  ; 0 1
out spl, temp
ldi temp, high(RAMEND)  ; 0 1
out sph, temp
```

AVR Stack Operations

.include "m64def.inc"
.def temp=r20
.cseg
ldi temp, low(RAMEND)
out spl, temp
ldi temp, high(RAMEND)
out sph, temp
ldi r1, 0xff
push r1

Relative Call to Subroutine

- Syntax: rcall k
- Operands: -2K ≤ k < 2K
- Operation: (i) STACK ← PC + 1 (Store return address)
  (ii) SP ← SP − 2 (2 bytes, 16 bits) for devices with 16 bits PC
       SP ← SP − 3 (3 bytes, 22 bits) for devices with 22 bits PC
  (iii) PC ← PC + k + 1
- Flag affected: None.
- Encoding: 1101 kkkk kkkk kkkk
- Words: 1
- Cycles: 3 (Devices with 16-bit PC)
  4 (Devices with 22-bit PC)

Example:
rcall routine ; Call subroutine
... routine:
push r14 ; Save r14 on the stack
push r15 ; Save r15 on the stack
... ; Put the code for the subroutine here.
pop r15 ; Restore r15
pop r14 ; Restore r14
ret ; Return from subroutine
Indirect Call to Subroutine

- Syntax: icall
- Operation: (i) STACK ← PC + 1 (Store return address)
  (ii) SP ← SP – 2 (2 bytes, 16 bits) for devices with 16 bits PC
       SP ← SP – 3 (3 bytes, 22 bits) for devices with 22 bits PC
  (iii) PC(15:0) ← Z(15:0) for devices with 16 bits PC
       PC(15:0) ← Z(15:0) and PC(21:16) ← 0 for devices with
              22 bits PC
- Flag affected: None.
- Encoding: 1001 0101 0000 1001
- Words: 1
- Cycles: 3 (Devices with 16-bit PC)
  4 (Devices with 22-bit PC)

Indirect Call to Subroutine (Cont.)

- Example:
  clr r10 ; Clear r10
  ldi r20, 2 ; Load call table offset
  ldi r30, low(Lab<<1) ; High byte of the starting address (base) of call table
  ldi r31, high(Lab<<1) ; Low byte of the starting address (base) of call table
  add r30, r20
  adc r31, r10 ; Base + offset is the address of the call table entry
  lpm r0, Z+ ; Load low byte of the the call table entry
  lpm r1, Z ; Load high byte of the call table entry
  movw r31:r30, r1:r0 ; Set the pointer register Z to point the target function
  icall ; Call the target function

  Lab: .dw ct_l0 ; The first entry of the call table
       .dw ct_l1 ; The second entry of the call table
       ...
       ct_l0: nop
       ct_l1: nop
       ...

Long Call to Subroutine

- Syntax: call k
- Operands: 0 ≤ k < 64K
- Operation: (i) STACK ← PC + 1 (Store return address)
  (ii) SP ← SP – 2 (2 bytes, 16 bits) for devices with 16 bits PC
       SP ← SP – 3 (3 bytes, 22 bits) for devices with 22 bits PC
  (iii) PC ← k
- Flag affected: None.
- Encoding: 1001 0101 kkkk 111k
       kkkk kkkk kkkk kkkk
- Words: 2
- Cycles: 4 (Devices with 16-bit PC)
  5 (Devices with 22-bit PC)

Long Call to Subroutine (Cont.)

- Example:
  mov r16, r0 ; Copy r0 to r16
  call check ; Call subroutine
  nop ; Continue (do nothing)
  ...
  check: cpi r16, $42 ; Check if r16 has a special value
      breq error ; Branch if equal
      ...
  error: ldi r1, 1
  ...
  ; put the code for handling the error here
  ret ; Return from subroutine
Return from Subroutine

- Syntax: ret
- Operation: (i) SP ← SP + 2 (2 bytes, 16 bits) for devices with 16 bits PC
  SP ← SP + 3 (3 bytes, 22 bits) for devices with 22 bits PC
(ii) PC(15:0) ← STACK for devices with 16 bits PC
  PC(21:0) ← STACK Devices with 22 bits PC
- Flag affected: None
- Encoding: 1001 0101 0000 1000
- Words: 1
- Cycles: 4 (Devices with 16-bit PC)
  5 (Devices with 22-bit PC)
- Example: routine: push r14 ; Save r14 on the stack
  ... ; Put the code for the subroutine here.
  pop r14 ; Restore r14
  ret ; Return from subroutine

Return from Interrupt

- Syntax: reti
- Operation: (i) SP ← SP + 2 (2 bytes, 16 bits) for devices with 16 bits PC
  SP ← SP + 3 (3 bytes, 22 bits) for devices with 22 bits PC
(ii) PC(15:0) ← STACK for devices with 16 bits PC
  PC(21:0) ← STACK Devices with 22 bits PC
- Flag affected: 1 ← 1
- Encoding: 1001 0101 0001 1000
- Words: 1
- Cycles: 4 (Devices with 16-bit PC)
  5 (Devices with 22-bit PC)
- Example:

Return from Interrupt (Cont.)

- Example:
  ... extint: push r0 ; Save r0 on the stack
  ... pop r0 ; Restore r0
  ret ; Return and enable interrupts
- Will cover details later

Branch If Equal

- Syntax: breq k
- Operands: -64 ≤ k < 63
- Operation: If Rd = Rr (Z = 1) then PC ← PC + k + 1, else PC ← PC + 1
- Flag affected: None
- Encoding: 1111 00kk kkkk k001
- Words: 1
- Cycles: 1 if condition is false
  2 if conditional is true
- Example:
  cp r1, r0 ; Compare registers r1 and r0
  breq equal ; Branch if registers equal
  ... equal: nop ; Branch destination (do nothing)
Branch If Same or Higher (Unsigned)

- Syntax: `brsh k`
- Operands: `-64 ≤ k < 63`
- Operation: If `rd ≥ rr` (unsigned comparison) then PC ← PC + k + 1, else PC ← PC + 1
- Flag affected: none
- Encoding: `1111 01kk kkkk k000`
- Words: 1
- Cycles: 1 if condition is false
- Example:
  ```
  sbi r26, $56 ; subtract $56 from r26
  brsh test ; branch if r26 ≥ $56
  ...
  Test: nop ; branch destination
  ...
  ```

Branch If Lower (Unsigned)

- Syntax: `brlo k`
- Operands: `-64 ≤ k < 63`
- Operation: If `rd < rr` (unsigned comparison) then PC ← PC + k + 1, else PC ← PC + 1
- Flag affected: None
- Encoding: `1111 00kk kkkk k000`
- Words: 1
- Cycles: 1 if condition is false
- Example:
  ```
  eor r19, r19 ; Clear r19
  loop: inc r19 ; Increase r19
  ...
  cpi r19, $10 ; Compare r19 with $10
  brlo loop ; Branch if r19 < $10 (unsigned)
  nop ; Exit from loop (do nothing)
  ```

Branch If Less Than (Signed)

- Syntax: `brlt k`
- Operands: `-64 ≤ k < 63`
- Operation: If `rd < rr` (signed comparison) then PC ← PC + k + 1, else PC ← PC + 1
- Flag affected: None
- Encoding: `1111 00kk kkkk k100`
- Words: 1
- Cycles: 1 if condition is false
- Example:
  ```
  cp r16, r1 ; Compare r16 to r1
  brlt less ; Branch if r16 < r1 (signed)
  ...
  less: nop ; Branch destination (do nothing)
  ```

Branch If Greater or Equal (Signed)

- Syntax: `brge k`
- Operands: `-64 ≤ k < 63`
- Operation: If `rd ≥ rr` (signed comparison) then PC ← PC + k + 1, else PC ← PC + 1
- Flag affected: None
- Encoding: `1111 01kk kkkk k100`
- Words: 1
- Cycles: 1 if condition is false
- Example:
  ```
  cp r11, r12 ; Compare registers r11 and r12
  brge greateq ; Branch if r11 ≥ r12 (signed)
  ...
  greateq: nop ; Branch destination (do nothing)
  ```
Branch If Overflow Set

- Syntax: `brvs k`
- Operands: `-64 \leq k < 63`
- Operation: If `V=1` then \( \text{PC} \leftarrow \text{PC} + k + 1 \), else \( \text{PC} \leftarrow \text{PC} + 1 \)
- Flag affected: None
- Encoding: `1111 00kk kkkk k011`
- Words: 1
- Cycles: 1 if condition is false
  2 if conditional is true
- Example:
  
  ```
  add r3, r4 ; Add r4 to r3
  brvs overfl ; Branch if overflow
  ...
  overfl: nop ; Branch destination (do nothing)
  ```

Branch If Overflow Clear

- Syntax: `brvc k`
- Operands: `-64 \leq k < 63`
- Operation: If `V=0` then \( \text{PC} \leftarrow \text{PC} + k + 1 \), else \( \text{PC} \leftarrow \text{PC} + 1 \)
- Flag affected: None
- Encoding: `1111 01kk kkkk k011`
- Words: 1
- Cycles: 1 if condition is false
  2 if conditional is true
- Example:
  
  ```
  add r3, r4 ; Add r4 to r3
  brvc noover ; Branch if no overflow
  ...
  noover: nop ; Branch destination (do nothing)
  ```

Branch if Global Interrupt is Enabled

- Syntax: `brie k`
- Operands: `-64 \leq k < 63`
- Operation: If `I=1` then \( \text{PC} \leftarrow \text{PC} + k + 1 \), else \( \text{PC} \leftarrow \text{PC} + 1 \)
- Flag affected: None
- Encoding: `1111 00kk kkkk k111`
- Words: 1
- Cycles: 1 if condition is false
  2 if conditional is true
- Example:
  
  ```
  brvs inten ; Branch if the global interrupt is enabled
  ...
  inten: nop ; Branch destination (do nothing)
  ```

Branch if Global Interrupt is Disabled

- Syntax: `brid k`
- Operands: `-64 \leq k < 63`
- Operation: If `I=0` then \( \text{PC} \leftarrow \text{PC} + k + 1 \), else \( \text{PC} \leftarrow \text{PC} + 1 \)
- Flag affected: None
- Encoding: `1111 00kk kkkk k111`
- Words: 1
- Cycles: 1 if condition is false
  2 if conditional is true
- Example:
  
  ```
  brvs intdis ; Branch if the global interrupt is enabled
  ...
  intdis: nop ; Branch destination (do nothing)
  ```
Reading Material

1. AVR Instruction Set.