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: jmp k
- Operands: 0 ≤ k < 4M
- Operation: PC←k
- Flag affected: None
- Encoding: 0001 010k kkkk 110k
- Words: 2
- Cycles: 3
- Example:
  
  mov r1, r0 ; Copy r0 to r1
  jmp farplc ; Unconditional jump
  ...
  farplc: inc r20 ; Jump destination

Relative Jump

- Syntax: rjmp k
- Operands: -2K ≤ k < 2K
- Operation: PC←PC+k+1
- Flag affected: None
- Encoding: 1100 kkkk kkkk kkkk
- Words: 1
- Cycles: 2
- Example:
  
  cpi r16, $42 ; Compare r16 to $42
  brne error ; Branch to error if r16 ≠ $42
  rjmp ok ; Jump to ok
  ...
  error: add r16, r17 ; Add r17 to r16
  inc r16 ; Increment r16
  ok: mov r2, r20 ; Jump destination

Indirect Jump

- Syntax: ijmp
- Operation:
  (i) PC←Z(15:0) Devices with 16 bits PC, 128K bytes program memory maximum.
  (ii) PC(15:0)←Z(15:0) Devices with 22 bits PC, 8M bytes program memory maximum.
  PC(21:16) ← 0
- Flag affected: None
- Encoding: 1001 0100 0000 1001
- Words: 1
- Cycles: 2
- Example:
  
  ldi r20, 2 ; Load jump table offset
  ldi r30, low(Lab<<1) ; High byte of the starting address (base) of jump table
  ldi r31, high(Lab<<1) ; Low byte of the starting address (base) of jump table
  add r30, r20 ; Base + offset is the address of the jump table entry
  lpm r0, Z+ ; Load low byte of the jump table entry
  lpm r1, Z ; Load high byte of the jump table entry
  movw r31:r30, r1:r0 ; Set the pointer register Z to point the target instruction
  ijmp ; Jump to the target instruction
  ...
  Lab: .dw jt_l0
  .dw jt_l1
  ...
  jt_l0: nop
  jt_l1: nop
  ...

Indirect Jump (Cont.)

- Example:
  
  ldi r20, 2 ; Load jump table offset
  ldi r30, low(Lab<<1) ; High byte of the starting address (base) of jump table
  ldi r31, high(Lab<<1) ; Low byte of the starting address (base) of jump table
  add r30, r20 ; Base + offset is the address of the jump table entry
  lpm r0, Z+ ; Load low byte of the jump table entry
  lpm r1, Z ; Load high byte of the jump table entry
  movw r31:r30, r1:r0 ; Set the pointer register Z to point the target instruction
  ijmp ; Jump to the target instruction
  ...
  Lab: .dw jt_l0
  .dw jt_l1
  ...
  jt_l0: nop
  jt_l1: nop
  ...
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
    - SPH (high byte) (IO register $3E)
    - 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)
out spl, temp
ldi temp, high(RAMEND)
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)

Relative Call to Subroutine (Cont.)

• 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)

Example:
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 ; 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 010k kkkk 111k
  kkkk kkkk kkkk kkkk
- Words: 2
- Cycles: 3 (Devices with 16-bit PC)
  4 (Devices with 22-bit PC)

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) \(SP \rightarrow SP - 2\) for devices with 16 bits PC
  - \(SP \rightarrow SP - 3\) for devices with 22 bits PC
  - \(PC(15:0) \rightarrow \text{STACK}\) for devices with 16 bits PC
  - \(PC(21:0) \rightarrow \text{STACK}\) 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
  ...
  ...
  ...
  pop r14 ; Restore r14
  ret ; Return from subroutine
  ```

Return from Interrupt

- Syntax: `reti`
- Operation:
  - (i) \(SP \rightarrow SP - 2\) for devices with 16 bits PC
  - \(SP \rightarrow SP - 3\) for devices with 22 bits PC
  - \(PC(15:0) \rightarrow \text{STACK}\) for devices with 16 bits PC
  - \(PC(21:0) \rightarrow \text{STACK}\) Devices with 22 bits PC
  - (ii) Set global interrupt flag I (Bit 7 of the Program Status Register).
- Flag affected: I
- Encoding: 1001 0101 0001 1000
- Words: 1
- Cycles: 4 (Devices with 16-bit PC), 5 (Devices with 22-bit PC)
- Example:
  ```
  ... extint: push r0 ; Save r0 on the stack
  ... pop r0 ; Restore r0
  reti ; Return and enable interrupts
  ```

Return from Interrupt (Cont.)

- Will cover details later

Branch If Equal

- Syntax: `breq k`
- Operands: -64 \(\leq k < 63\)
- Operation: If Rd = Rr (Z = 1) then \(PC \leftarrow PC + k + 1\), else \(PC \leftarrow PC + 1\)
- 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: \texttt{brsh } k
• Operands: \(-64 \leq k < 63\)
• Operation: if \(rd \geq Rr\) (unsigned comparison) then \(PC \leftarrow PC + k + 1\), else \(PC \leftarrow PC + 1\)
• Flag affected: none
• Encoding: 1111 01kk kkkk k000
• Words: 1
• Cycles: 1 if condition is false
2 if conditional is true
• Example:
  sbi r26, $56 ; subtract $56 from r26
  brsh test ; branch if r26 \geq $56

  Test: nop ; branch destination

  ...

Branch If Lower (Unsigned)

• Syntax: \texttt{brlo } k
• Operands: \(-64 \leq k < 63\)
• Operation: If \(Rd < Rr\) (unsigned comparison) then \(PC \leftarrow PC + k + 1\), else \(PC \leftarrow PC + 1\)
• Flag affected: None
• Encoding: 1111 00kk kkkk k000
• Words: 1
• Cycles: 1 if condition is false
2 if conditional is true
• 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: \texttt{brlt } k
• Operands: \(-64 \leq k < 63\)
• Operation: If \(Rd < Rr\) (signed comparison) then \(PC \leftarrow PC + k + 1\), else \(PC \leftarrow PC + 1\)
• Flag affected: None
• Encoding: 1111 00kk kkkk k100
• Words: 1
• Cycles: 1 if condition is false
2 if conditional is true
• 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: \texttt{brge } k
• Operands: \(-64 \leq k < 63\)
• Operation: If \(Rd \geq Rr\) (signed comparison) then \(PC \leftarrow PC + k + 1\), else \(PC \leftarrow PC + 1\)
• Flag affected: None
• Encoding: 1111 01kk kkkk k100
• Words: 1
• Cycles: 1 if condition is false
2 if conditional is true
• Example:
  cp r11, r12 ; Compare registers r11 and r12
  brge greateq ; Branch if r11 \geq r12 (signed)

  ... greateq: nop ; Branch destination (do nothing)
Branch If Overflow Set

- Syntax: \textit{brvs} \texttt{k}
- Operands: \(-64 \leq k < 63\)
- Operation: If \(V=1\) then \(PC \leftarrow PC + k + 1\), else \(PC \leftarrow PC + 1\)
- Flag affected: None
- Encoding: 1111 00kk kkkk k011
- Words: 1
- Cycles: 1 if condition is false
- Example:

\begin{verbatim}
add r3, r4 ; Add r4 to r3
brvs overfl ; Branch if overflow
...
overfl: nop ; Branch destination (do nothing)
\end{verbatim}

Branch If Overflow Clear

- Syntax: \textit{brvc} \texttt{k}
- Operands: \(-64 \leq k < 63\)
- Operation: If \(V=0\) then \(PC \leftarrow PC + k + 1\), else \(PC \leftarrow PC + 1\)
- Flag affected: None
- Encoding: 1111 01kk kkkk k011
- Words: 1
- Cycles: 1 if condition is false
- Example:

\begin{verbatim}
add r3, r4 ; Add r4 to r3
brvs noover ; Branch if no overflow
...
noover: nop ; Branch destination (do nothing)
\end{verbatim}

Branch if Global Interrupt is Enabled

- Syntax: \textit{brie} \texttt{k}
- Operands: \(-64 \leq k < 63\)
- Operation: If \(I=1\) then \(PC \leftarrow PC + k + 1\), else \(PC \leftarrow PC + 1\)
- Flag affected: None
- Encoding: 1111 00kk kkkk k111
- Words: 1
- Cycles: 1 if condition is false
- Example:

\begin{verbatim}
brvs inten ; Branch if the global interrupt is enabled
...
inten: nop ; Branch destination (do nothing)
\end{verbatim}

Branch if Global Interrupt is Disabled

- Syntax: \textit{brid} \texttt{k}
- Operands: \(-64 \leq k < 63\)
- Operation: If \(I=0\) then \(PC \leftarrow PC + k + 1\), else \(PC \leftarrow PC + 1\)
- Flag affected: None
- Encoding: 1111 00kk kkkk k111
- Words: 1
- Cycles: 1 if condition is false
- Example:

\begin{verbatim}
brid intdis ; Branch if the global interrupt is enabled
...
intdis: nop ; Branch destination (do nothing)
\end{verbatim}