AVR ISA & AVR Programming (I)

Lecturer: Sri Parameswaran
Notes by: Annie Guo
Lecture Overview

- AVR ISA
- AVR Instructions & Programming (I)
  - Basic construct implementation
Atmel AVR

- 8-bit RISC architecture
  - Most instructions have 16-bit fixed length
  - Most instructions take 1 clock cycle to execute.
- Load-store memory access architecture
  - All calculations are on registers
- Two-stage instruction pipelining
- Internal program memory and data memory
- Wide variety of on-chip peripherals (digital I/O, ADC, EEPROM, UART, pulse width modulator (PWM) etc).
AVR Registers

- General purpose registers
  - 32 8-bit registers, R0 ~ R31 or r0 ~ r31
  - Can be further divided into two groups
    - First half group: R0 ~ R15 and second half group: R16 ~ R31
    - Some instructions work only on the second half group R16~R31
      - Due to the limitation of instruction encoding bits
      - Will be covered later
      - E.g. ldi rd, #number ; rd ∈ R16~R31
AVR Registers (cont.)

- General purpose registers
  - The following register pairs can work as address indexes
    - X, R26:R27
    - Y, R28:R29
    - Z, R31:R31
  - The following registers can be applied for specific use
    - R1:R0 store the result of multiplication instruction
    - R0 stores the data loaded from the program memory
AVR Registers (cont.)

- **I/O registers**
  - 64 8-bit registers
    - Their names are defined in the m64def.inc file
  - Used in input/output instructions
    - Mainly storing data/addresses and control signal bits
  - Some instructions work only with I/O registers, others with general purpose registers – don’t confuse them
    - E.g. in rd, port ; port must be an I/O register
  - Will be covered in detail later

- **Status register (SREG)**
  - A special I/O register
The Status Register in AVR

- The Status Register (SREG) contains information about the result of the most recently executed arithmetic instruction. This information can be used for altering program flow in order to perform conditional operations.
- SREG is updated after any of ALU operations by hardware.
- SREG is not automatically stored when entering an interrupt routine and restored when returning from an interrupt. This must be handled by software.
  - Using in/out instruction to store/restore SREG
The Status Register in AVR (cont.)

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 7 – I: Global Interrupt Enable**
  - Used to enable and disable interrupts.
  - 1: enabled. 0: disabled.
  - The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts.
The Status Register in AVR (cont.)

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- Bit 6 – T: Bit Copy Storage
  - The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit. A bit from a register in the Register File can be copied into T by the BST instruction, and a bit in T can be copied into a bit in a register in the Register File by the BLD instruction.
The Status Register in AVR (cont.)

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- Bit 5 – H: Half Carry Flag
  - The Half Carry Flag H indicates a Half Carry (carry from bit 4) in some arithmetic operations.
  - Half Carry is useful in BCD arithmetic.
The Status Register in AVR (cont.)

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit 7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- **Bit 4 – S: Sign Bit**
  - Exclusive OR between the Negative Flag N and the Two’s Complement Overflow Flag V (\( S = N \oplus V \)).

- **Bit 3 – V: Two’s Complement Overflow Flag**
  - The Two’s Complement Overflow Flag V supports two’s complement arithmetic.
The Status Register in AVR (cont.)

<table>
<thead>
<tr>
<th>I</th>
<th>T</th>
<th>H</th>
<th>S</th>
<th>V</th>
<th>N</th>
<th>Z</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>7</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

- **Bit 2 – N: Negative Flag**
  - N is the most significant bit of the result.
- **Bit 1 – Z: Zero Flag**
  - Z indicates a zero result in an arithmetic or logic operation. 1: zero. 0: Non-zero.
The Status Register in AVR (cont.)

Bit 0 – C: Carry Flag
- Its meaning depends on the operation.
  - For addition X+Y, it is the carry from the most significant bit
  - For subtraction x-y, where x and y are unsigned integers, it indicates if x<y. If x<y, C=1; otherwise, C=0.
AVR Address Spaces

- Three address spaces
  - Data memory
    - Storing data to be processed
  - Program memory
    - Storing program and sometimes constants
  - EEPROM memory
    - Large permanent data storage
Data Memory Space

- Covers
  - Register file
    - I.e. registers in the register file also have memory address
  - I/O registers
    - I.e. I/O registers have two versions of addresses
      - I/O addresses
      - Memory addresses
  - SRAM data memory
    - The highest memory location is defined as RAMEND

![Data Memory Diagram]

- 32 General purpose Working Registers
- 64 Input/Output Registers
- Internal SRAM (128~4K bytes)
- External SRAM
- 8 bits
- End Address
Program Memory Space

- Covers
  - 16 bit Flash Memory
    - Read only
      - Instructions are retained when power off
  - Can be accessed with special instructions
    - LPM
    - SPM

Program Memory

- 0x0000

Program Flash Memory
(1K bytes~128K bytes)

16 Bits

End Address
EEPROM Memory Space

- Covers
  - 8-bit EEPROM memory
    - Use to permanently store large set of data
  - Can be accessed using load and store instructions with special control bit settings
- Not covered in this course
AVR Instruction Format

- For AVR, almost all instructions are 16 bits long
  - add Rd, Rr
  - sub Rd, Rr
  - mul Rd, Rr
  - brge k
- Few instructions are 32 bits long
  - lds Rd, k (0 ≤ k ≤ 65535)
    - loads 1 byte from the SRAM to a register.
Examples (1) -- 16 bits long

- Clear register instruction
  Syntax: \texttt{clr} \textit{Rd}
  Operand: \(0 \leq d \leq 31\)
  Operation: \(Rd \leftarrow 0\)

- Instruction format

  \begin{array}{cccccccccc}
  0 & 0 & 1 & 0 & 0 & 1 & d & d & d & d & d & d & 0 \\
  \end{array}

  - OpCode uses 6 bits (bit 9 to bit 15).
  - The only operand uses the remaining 10 bits (only 5 bits (bit 0 to bit 4) are actually needed).

- Execution time
  1 clock cycle
Examples (2)  
-- 32 bit long

- Unconditional branch
  Syntax: $jmp\ k$
  Operand: $0 \leq k < 4M$
  Operation: $PC \leftarrow K$

- Instruction format

- Execution time
  3 clock cycles
Examples (3) -- with variable cycles

- Conditional branch
  Syntax: `breq k`
  Operand: `-64 \leq k < +63`
  Operation: If `Rd=Rr(Z=1)` then `PC \leftarrow PC+k+1`, else `PC \leftarrow PC+1`

| 1 1 1 1 | 0 0 k k | k k k k | k 0 0 1 |

- Instruction format
- Execution time
  1 clock cycle if condition is false
  2 clock cycles if condition is true
AVR Instructions

- AVR has the following classes of instructions:
  - Arithmetic and Logic
  - Data transfer
  - Program control
  - Bit and Others
    - Bit and Bit test
    - MCU Control
- An overview of the instructions are given in the next slides.
AL Instructions

- Arithmetic
  - addition
    - E.g. ADD Rd, Rr
  - Subtraction
    - E.g. SUB Rd, Rr
  - Increment/decrement
    - E.g. INC Rd
  - Multiplication
    - E.g. MUL Rd, Rr

- Logic
  - E.g. AND Rd, Rr

- Shift
  - E.g. LSL Rd
Transfer Instructions

- GP register
  - E.g. MOV Rd, Rr

- I/O registers
  - E.g. IN Rd, PORTA
    - OUT PORTB, Rr

- Stack
  - PUSH Rr
  - POP Rd

- Immediate values
  - E.g. LDI Rd, K8

- Memory
  - Data memory
    - E.g. LD Rd, X ST X, Rr
  - Program memory
    - E.g. LPM
  - EEPROM memory
    - Not covered in this course
Program Control Instructions

- **Branch**
  - **Conditional**
    - Jump to address
      - BREQ des
        - test ALU flag and jump to specified address if the test was true
  - skips
    - SBIC k
      - test a bit in a register or an IO register and skip the next instruction if the test was true.

- **Unconditional**
  - Jump to the specified address
    - RJMP des

- **Call subroutine**
  - E.g. RCALL k

- **Return from subroutine**
  - E.g. RET
Bit & Other Instructions

- **Bit**
  - Set bit
    - E.g. SBI PORTA, b
  - Clear bit
    - E.g CBI PORTA, b
  - Bit copy
    - E.g. BST Rd, b

- **Others**
  - NOP
  - BREAK
  - SLEEP
  - WDR
AVR Instructions (cont.)

- Not all instructions are implemented in all AVR controllers.
- Refer to the data sheet of a specific microcontroller.
- Refer to online AVR instruction document for the detail description of each instruction.
AVR Addressing Modes

- Immediate
- Register direct
- Memory related addressing mode
  - Data memory
    - Direct
    - Indirect
    - Indirect with Displacement
    - Indirect with Pre-decrement
    - Indirect with Post-increment
  - Program memory
  - EPROM memory
    - Not covered in this course
Immediate Addressing

- The operands come from the instructions
- For example
  
  \[
  \text{andi r16, 0F}
  \]

- Bitwise logic AND operation
  - Clear upper nibble of register r16
Register Direct Addressing

- The operands come from general purpose registers
- For example
  
  ```
  and r16, r0
  ```

- r16 ← r16 +r0
  - Clear upper nibble of register r16
Register Direct Addressing

- The operands come from I/O registers
- For example

\[ \text{in } r25, \ PINA \]
\[ \text{-- } r25 \leftarrow \ PINA \]
Data Memory Addressing
Data Direct Addressing

- The data memory address is given directly from the instruction
- For example

```
lds r5, $F123  
-- r5 ← Mem($F123), or r5 ← ($F123)
```
Indirect Addressing

- The address of memory data is from an address pointer \((X, Y, Z)\)
- For example

  ```
  ld r11, X
  -- r11 ← Mem(X), or r11 ← (X)
  ```
Indirect Addressing with displacement

- The address of memory data is from \((Y,Z)+q\)
- For example

\[
\text{std } Y+10, r14 \\
-- (Y+10) \leftarrow r14
\]
Indirect Addressing with Pre-decrement

- The address of memory data is from an address pointer \((X, Y, Z)\) and the value of the pointer is auto-decreased \textbf{before} each memory access.
- For example
  \[
  \text{std } -Y, r14 \\
  \quad \text{-- } Y \leftarrow Y-1, (Y) \leftarrow r14
  \]
Indirect Addressing with Post-increment

- The address of memory data is from an address pointer (X, Y, Z) and the value of the pointer is auto-increased after each memory access.

- For example

\[
\text{std } Y^+, r14 \\
\text{-- } (Y) \leftarrow r14, Y \leftarrow Y+1
\]
Program Memory Addressing
Direct Program Addressing

- The instruction address is from instruction
- For example

\[ jmp \, k \]
\[ -- \, (PC) \leftarrow k \]
Relative Program Addressing

- The instruction address is PC+k+1
- For example, \textit{rjmp} k
  \[
  \text{(PC)} \leftarrow (\text{PC})+k+1
  \]
Indirect Memory Addressing

- The instruction address is implicitly stored in Z register

\[ i\text{call} \]

\[
\text{PC}(15:0) \leftarrow (Z), \text{PC}(21:16) \leftarrow 0
\]
Program Memory Constant Addressing

- The address of the constant is stored in Z register
  - The address is a byte address.
- For example:

  ```
  lpm
  -- r0 ← (Z)
  ```
Program Memory Addressing with Post-increment

- For example

\[
\text{\textit{lpm} } r16, Z+ \\
\quad \rightarrow r16 \leftarrow (Z), Z \leftarrow Z+1
\]
AVR Programming

- Refer to the online AVR Instruction Set documentation for the complete list of AVR instructions

- The rest of the lecture covers
  - Programming to implement some basic constructs with examples
Arithmetic Calculation (1/4) -- example

- Expressions \[ z = 2x - xy - x^2 \]

- where all data including products from multiplications are 8-bit unsigned numbers; and \( x, \ y, \ z \) are stored in registers r2, r3, and r4.
What instructions do you need?

- sub
- mul
- ldi
- mov
Subtract without Carry

- Syntax: \textit{sub Rd, Rr}
- Operands: \(Rd, Rr \in \{r0, r1, \ldots, r31\}\)
- Operation: \(Rd \leftarrow Rd - Rr\)
- Flags affected: H, S, V, N, Z, C
- Words: 1
- Cycles: 1
Multiply Unsigned

- Syntax: \textit{mul Rd, Rr}
- Operands: Rd, Rr \in \{r0, r1, \ldots, r31\}
- Operation: \texttt{r1:r0} \leftarrow \texttt{Rr*Rd}
  - (unsigned \leftarrow \texttt{unsigned} \times \texttt{unsigned})
- Flags affected: Z, C
  - C is set if bit 15 of the result is set; cleared otherwise.
- Words: 1
- Cycles: 2
Load Immediate

- Syntax: \textit{ldi} \textit{Rd, k}
- Operands: \textit{Rd} \in \{r16, ..., r31\}, \ 0 \leq k \leq 255
- Operation: \textit{Rd} \leftarrow k
- Flag affected: None
- Words: 1
- Cycles: 1
- Encoding: 1110 kkkk dddd kkkk
- Example:
  \textit{ldi r16, $42} ; \text{Load $42$ to r16}
Copy Register

- Syntax: \textit{mov Rd, Rr}
- Operands: Rd, Rr ∈\{r0,r1,\ldots,r31\}
- Operation: Rd ← Rr
- Flag affected: None
- Words: 1
- Cycles: 1
Arithmetic calculation (2/4)

- AVR code for $z = 2x - xy - x^2$
- where all data including products from multiplications are 8-bit unsigned numbers;
  and $x$, $y$, $z$ are stored in registers r2, r3, and r4.

```
ldi r16, 2 ; r16 ← 2
mul r16, r2 ; r1:r0 ← 2x
mov r5, r0 ; r5 ← 2x
mul r2, r3 ; r1:r0 ← xy
sub r5, r0 ; r16 < 2x-xy
mul r2, r2 ; r1:r0 ← x^2
sub r5, r0 ; r5 ← 2x-xy- x^2
mov r4, r5 ; r4 ← z
```

- 8 instructions and 11 cycles
Arithmetic calculation (3/4)

- Expressions
  \[ z = 2x - xy - x^2 \]
  \[ z = x(2 - (x + y)) \]

- where all data including products from multiplications are 8-bit unsigned numbers; and x, y, z are stored in registers r2, r3, and r4.
What instructions do you need?

- sub
- mul
- ldi
- mov
- add
Add without Carry

- Syntax: \textit{add Rd, Rr}
- Operands: Rd, Rr \( \in \{r0, r1, \ldots, r31\} \)
- Operation: Rd\( \leftarrow \)Rd + Rr
- Flags affected: H, S, V, N, Z, C
- Words: 1
- Cycles: 1
Arithmetic calculation (4/4)

- AVR code for
  \[ z = 2x - xy - x^2 \]
  \[ z = x(2 - (x + y)) \]

- where all data including products from multiplications are 8-bit unsigned numbers; and x, y, z are stored in registers r2, r3, and r4.

```
  mov  r5, r2 ; r5 ← x
  add  r5, r3 ; r5 ← x+y
  ldi  r16, 2 ; r16 ← 2
  sub  r16, r5 ; r16 < 2-(x+y)
  mul  r2, r16 ; r1:r0 ← x(2-(x+y))
  mov  r4, r0 ; r4 ← z
```

- 6 instructions and 7 cycles
Control Structure (1/2) -- example

- IF-THEN-ELSE control structure

\[
\begin{align*}
\text{if}(x < 0) & \quad z = 1; \\
\text{else} & \quad z = -1;
\end{align*}
\]

- Numbers x, z are 8-bit signed integers and stored in registers. You need to decide which registers to use.

- Instructions interested
  - Compare
  - Conditional branch
  - Unconditional jump
Compare

- Syntax: \( cp \ Rd, \ Rr \)
- Operands: \( Rd \in \{r0, r1, \ldots, r31\} \)
- Operation: \( Rd–Rr \) (\( Rd \) is not changed)
- Flags affected: \( H, S, V, N, Z, C \)
- Words: 1
- Cycles: 1
- Example:
  
  \[
  \begin{align*}
  &cp \ r4, \ r5 \quad ; \text{Compare r4 with r5} \\
  &brne \ \text{noteq} \quad ; \text{Branch if } r4 \neq r5 \\
  &\ldots \\
  &\text{noteq: nop} \quad ; \text{Branch destination (do nothing)}
  \end{align*}
  \]
Compare with Immediate

- Syntax: \textit{cpi }Rd, k
- Operands: Rd \(\in\{r16, r17, \ldots, r31\}\) and \(0 \leq k \leq 255\)
- Operation: Rd – k (Rd is not changed)
- Flags affected: H, S, V, N, Z, C
- Words: 1
- Cycles: 1
Conditional Branch

- **Syntax:** `brge k`
- **Operands:** `-64 \leq k < 64`
- **Operation:** If `Rd \geq Rr (N \oplus V=0)` then
  
  PC ← PC+k+1,

  else PC ← PC+1 if condition is false

- **Flag affected:** None
- **Words:** 1
- **Cycles:** 1 if condition is false; 2 if condition is true
Relative Jump

- Syntax: \textit{rjmp} \: k
- Operands: \(-2K \leq k < 2K\)
- Operation: \(PC \leftarrow PC + k + 1\)
- Flag affected: None
- Words: 1
- Cycles: 2
Control (2/2)

- IF-THEN-ELSE control structure

```plaintext
if(a<0)  
  b=1;  
else  
  b=-1;
```

- Numbers $x$, $z$ are 8-bit signed integers and stored in registers. You need to decide which registers to use.

```plaintext
.def a=r16
.def b=r17

cpi a, 0 ;a-0
brge ELSE ;if $a \geq 0$
ldi b, 1 ;b=1
rjmp END ;end of IF statement
ELSE: ldib, -1 ;b=-1
END: ...
```
Loop (1/2)

- WHILE loop

```
    i=1;
    while (i<=n){
        sum += i*i;
        i++;
    }
```

- Numbers i, sum are 8-bit unsigned integers and stored in registers. You need to decide which registers to use.
Loop (2/2)

* WHILE loop *

```
.def i = r16
.def n = r17
.def sum = r18

ldi i, 1 ;initialize
clr sum

loop:
    cp n, i
    brlo end
    mul i, i
    add sum, r0
    inc i
    rjmp loop

end:
    rjmp end
```
Homework


Study the following instructions:

- Arithmetic and logic instructions
  - `add`, `adc`, `adiw`, `sub`, `subi`, `sbc`, `sbc`, `subiw`, `mul`, `mul`, `muls`, `mulsu`
  - `and`, `andi`, `or`, `ori`, `eor`
  - `com`, `neg`
Homework

1. Study the following instructions (cont.)
   - Branch instructions
     - cp, cpc, cpi
     - rjmp
     - breq, brne
     - brge, brlt
     - brsh, brlo
   - Data transfer instructions
     - mov
     - ldi, ld, st
Homework

2. Implement the following functions with AVR assembly language
   1) 2-byte addition
   2) 2-byte multiplication

3. Inverse a string of ten characters that is stored in the registers r0~r9; and store the inverted string in registers r10~r19
Homework

4. Translate the following if-then-else statement, where $x$ is an 8-bit unsigned integer.

\[
\begin{align*}
\text{if}(x<0) & \quad z=1; \\
\text{else} & \quad z=255;
\end{align*}
\]
Reading Material

- AVR Instruction Set online documentation
  - Instruction Set Nomenclature
  - I/O Registers
  - The Program and Data Addressing
  - Arithmetic instructions, program control instructions