Microprocessors & Interfacing

AVR ISA &
AVR Programming (I)

Lecturer : Annie Guo

Lecture Overview

- AVR ISA and instructions
  - A brief overview
- AVR programming (I)
  - Implementation of basic programming constructs

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 AL calculations are on registers
- Internal program memory and data memory
- Wide variety of on-chip peripherals (digital I/O, ADC, EEPROM, UART, pulse width modulator (PWM) …).

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 together as address registers (or address pointers)
    - X, R27:R26
    - Y, R29:R28
    - Z, R31:R30
  - The following registers can be applied for specific purpose
    - 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 m2560def.inc file
  - Used in input/output operations
    - Mainly storing data/addresses and control signal bits
  - Will be covered in detail later
- Status register (SREG)
  - A special I/O register

SREG

- The Status Register (SREG) contains information about the result of the most recently executed AL instruction. This information can be used for altering program flow in order to perform conditional operations.
- SREG is updated by hardware after an AL operation.
  - Some instructions such as load do not affect SREG.
- SREG is not automatically saved 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
  - To be covered later

SREG (cont.)

I	T	H	S	V	N	Z	C
Bit	7	6	5	4	3	2	1	0

- 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 whether x<y or not. If x<y, C=1; otherwise, C=0.

SREG (cont.)

I	T	H	S	V	N	Z	C
Bit	7	6	5	4	3	2	1	0

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

SREG (cont.)

I	T	H	S	V	N	Z	C
Bit	7	6	5	4	3	2	1	0

- Bit 3 – V: Two's Complement Overflow Flag
  - The Two's Complement Overflow Flag V supports two's complement arithmetic.
- Bit 4 – S: Sign Bit
  - Exclusive OR between the Negative Flag N and the Two's Complement Overflow Flag V (S = N ⊕V).

SREG (cont.)

I	T	H	S	V	N	Z	C
Bit	7	6	5	4	3	2	1	0

- Bit 5 – H: Half Carry Flag
  - The Half Carry Flag H indicates a Half Carry (carry from bit 3) in some arithmetic operations.
  - Half Carry is useful in BCD arithmetic.

SREG (cont.)

I	T	H	S	V	N	Z	C
Bit	7	6	5	4	3	2	1	0

- Bit 6 – T: Bit Copy Storage
  - Used for copying a bit from one register to another register
    - The Bit Copy instructions BLD (Bit LoaD) and BST (Bit STore) use the T-bit as source or destination for the operated bit.

SREG (cont.)

Bit	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C

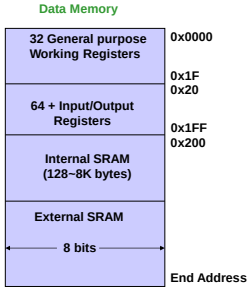
- Bit 7 – I: Global Interrupt Enable
  - Used to enable and disable interrupts.
  - 1: enable. 0: disable.
  - The I-bit is cleared by hardware after an interrupt has occurred, and is set by the RETI instruction to enable subsequent interrupts.
    - Will be covered later

AVR Address Spaces

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

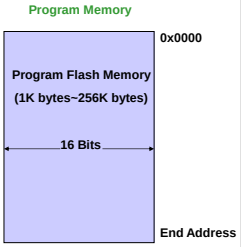
Data Memory Space

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



Program Memory Space

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



AVR Instruction Format

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

Instruction Examples (1) - 16 bits long

- Clear register

  Syntax: *clr Rd*

  Operand: 0 ≤ d ≤ 31

  Operation: Rd ← 0
- Instruction format

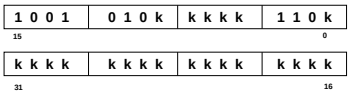
15															0
0	0	1	0	0	1	d	d	d	d	d	d	d	d	d	d

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

  1 clock cycle

Instruction Examples (2) - 32 bits long

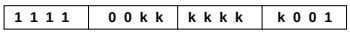
- Unconditional branch
  Syntax: *jmp k*
  Operand: 0 ≤ k < 4M
  Operation: PC ← k
- Instruction format



- Execution time
  3 clock cycles

Instruction Examples (3) - with variable exec. time

- Conditional branch
  Syntax: *breq k*
  Operand: -64 ≤ k < +63
  Operation: If Z=1(Rd=Rr) then PC ← PC+k+1, else PC ←PC+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 is 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
      - E.g. BREQ dst
        - » test ALU flag and jump to specified address if the condition is true
    - Skip
      - E.g. SBIC A, k
        - » test a bit in a register or an IO register and skip the next instruction if the condition is true.
  - Unconditional
    - Jump to the specified address
      - E.g. RJMP dst
- 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
  - Get a general view of the instruction set
  - Learn each instruction when use it.

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

```
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

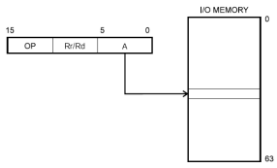
```
and r16, r0
```

  - r16 ← r16 *and* r0
    - Clear upper nibble of register r16 if r0 = 0x0F

Register Direct Addressing

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

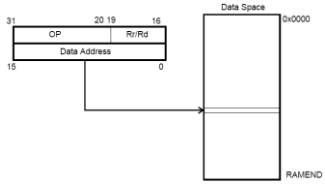
```
in r25, PINA
   -- r25 ← PIN A
```



Data Direct Addressing

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

lds r5, $F123
-- r5 ← Mem($F123), or r5 ←($F123)

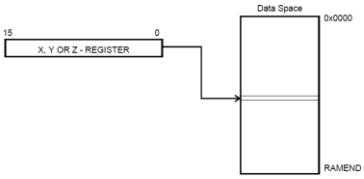


31

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)

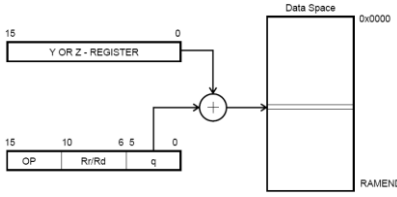


32

Indirect Addressing with Displacement

- The address of memory data is from (Y,Z)+q
  - Offset q >=0
- For example

std Y+10, r14
-- (Y+10) ← r14

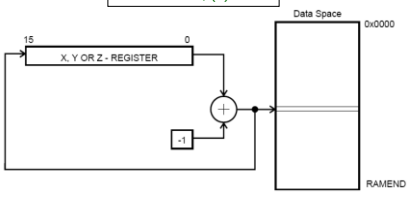


33

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 **before** each memory access.
- For example

std -Y, r14
-- Y←Y-1, (Y) ← r14

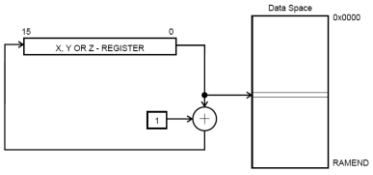


34

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

std Y+, r14
-- (Y) ← r14, Y ← Y+1



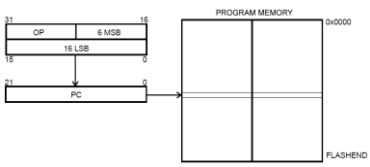
COMP9032 Week2

35

Direct Program Addressing

- The instruction address is from instruction
- For example

jmp k
-- PC ← k

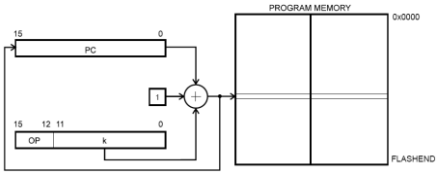


36

Relative Program Addressing

- The instruction address is PC+k+1
- For example

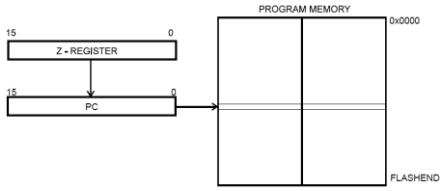
```
rjmp k
    -- PC ← PC+k+1
```



Indirect Memory Addressing

- The instruction address is implicitly stored in Z register

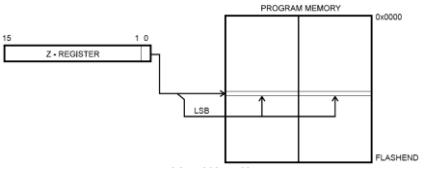
```
icall
    -- PC(15:0) ← (Z), PC(21:16) ←0
```



Program Memory Constant Addressing

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

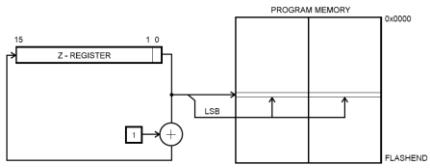
```
lpm
    -- r0 ← (Z)
```



Program Memory Addressing with Post-increment

- For example

```
lpm r16, Z+
    -- r16 ← (Z), Z←Z+1
```



AVR Programming

- Refer to the AVR Instruction Set document for the complete list of instructions
  - http://www.cse.unsw.edu.au/~cs9032, follow the link: References → Documents →AVR-Instruction-Set.pdf
- The rest of the lecture demonstrates
  - AVR assembly programming to implement some basic constructs with examples
    - Sequence
    - Selection
    - Iteration

Sequence (1/5) - example

- Expressions

$$z = 2x - xy - x^2$$

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

What instructions do you need?

- sub
- mul

Subtract without Carry

- Syntax: ***sub Rd, Rr***
- Operands: Rd, Rr ∈{r0, r1, …, r31}
- Operation: Rd←Rd–Rr
- Flags affected: H, S, V, N, Z, C
- Words: 1
- Cycles: 1

Multiply Unsigned

- Syntax: ***mul Rd, Rr***
- Operands: Rd, Rr ∈{r0, r1, …, r31}
- Operation: **r1:r0**←Rr*Rd
  - (unsigned←unsigned * unsigned)
- Flags affected: Z, C
  - C is set if bit 15 of the result is set; cleared otherwise.
- Words: 1
- Cycles: 2

What instructions do you need?

- sub
- mul
- ldi
- mov

Load Immediate

- Syntax: ***ldi Rd, k***
- Operands: Rd∈{**r16, …, r31**}, 0 ≤ k ≤ 255
- Operation: Rd←k
- Flag affected: None
- Words: 1
- Cycles: 1
- Encoding: 1110 kkkk dddd kkkk
- Example:

```
ldi  r16, $42        ; Load $42 to r16
```


Recall: 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

Copy Register

- Syntax: ***mov Rd, Rr***
- Operands: Rd, Rr ∈{r0,r1,…,r31}
- Operation: Rd←Rr
- Flag affected: None
- Words: 1
- Cycles: 1

Sequence (2/5)

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

```
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      ; r5 ← 2x-xy
mul   r2, r2      ; r1:r0 ← x2
sub   r5, r0      ; r5 ← 2x-xy- x2
mov   r4, r5      ; r4 ← z
```

  - 8 instructions and 11 cycles

Sequence (3/5)

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

```
ldi   r16, 2      ; r16 ← 2
mul   r16, r2     ; r1:r0 ← 2x
mov   r4, r0      ; r4 ← 2x
mul   r2, r3      ; r1:r0 ← xy
sub   r4, r0      ; r4 ← 2x-xy
mul   r2, r2      ; r1:r0 ← x2
sub   r4, r0      ; r4 ← 2x-xy- x2
```

  - 7 instructions and 10 cycles

Sequence (4/5)

- 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, respectively.

What instructions do you need?

- sub
- mul
- ldi
- mov
- add

Add without Carry

- Syntax: ***add Rd, Rr***
- Operands: Rd, Rr ∈{r0, r1, …, r31}
- Operation: Rd←Rd + Rr
- Flags affected: H, S, V, N, Z, C
- Words: 1
- Cycles: 1

Sequence (5/5)

- 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, respectively.

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

  - 6 instructions and 7 cycles

Selection (1/2) - example

- IF-THEN-ELSE control structure

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

  - Assume numbers a, b are 8-bit **signed** integers and stored in registers. You need to decide which registers to use.

- Instructions involved:
  - Compare
  - Conditional branch
  - Unconditional jump

Compare

- Syntax: ***cp Rd, Rr***
- Operands: Rd ∈{r0, r1, …, r31}
- Operation: Rd–Rr (Rd is not changed)
- Flags affected: H, S, V, N, Z, C
- Words: 1
- Cycles: 1
- Example:

```
        cp r4, r5          ; Compare r4 with r5
        brne noteq         ; Branch if r4 ≠ r5
        ...
noteq: nop                 ; Branch destination (do nothing)
```


Compare with Immediate

- Syntax: ***cpi Rd, k***
- Operands: Rd ∈{r16, r17, …, r31} and 0≤ k ≤ 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 ≤ k < 64
- Operation: If Rd≥Rr (N⊕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: ***rjmp k***
- Operands: -2K ≤ k < 2K
- Operation: PC←PC+k+1
- Flag affected: None
- Words: 1
- Cycles: 2

Selection (2/2)

- IF-THEN-ELSE control structure

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

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

```
.def    a=r16
.def    b=r17
        cpi     a, 0            ;a-0
        brge    ELSE            ;if a≥0, go to ELSE
        ldi     b, 1            ;b=1
        rjmp    END             ;end of IF statement
ELSE:   ldi     b, -1           ;b=-1
END:    ...
```


Iteration (1/2)

- WHILE loop

```
sum =0;
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.

Iteration (2/2)

- WHILE loop

```
sum =0;
i=1;
while (i<=n){
        sum += i*i;
        i++;
}
```

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

        ldi i, 1                ;initialization
        clr sum
loop:
        cp n, i
        brlo end
        mul i, i
        add sum, r0
        inc i
        rjmp loop
end:
        rjmp end
```


Reading Material

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

Homework

1. Refer to the AVR Instruction Set document (available at http://www.cse.unsw.edu.au/~cs9032, under the link References → Documents → AVR-Instruction-Set.pdf).

   Study the following instructions:
   - Arithmetic and logic instructions
     - add, adc, adiw, sub, subi, sbc, sbci, sbiw, 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. Write the assembly code for the following functions
   1) 2-byte addition (i.e, addition on 16-bit numbers)
   2) 2-byte signed subtraction
   3) 2-byte signed multiplication
3. Inverse a string of ten characters that is stored in the registers r0~r9; and store the inversed string in registers r10~r19