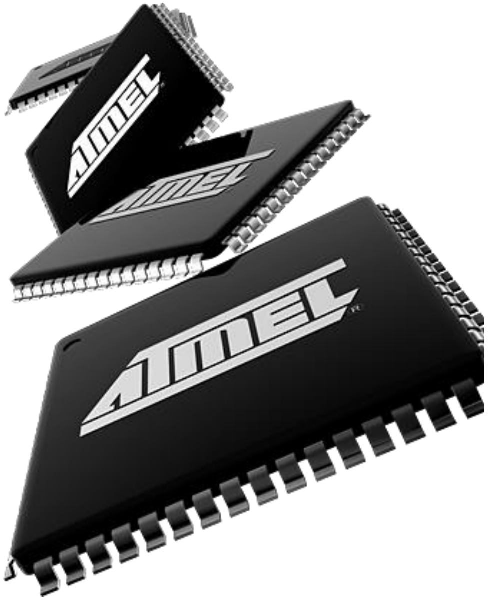
Principles and Applications of Microcontrollers

Yan-Fu Kuo
Dept. of Biomechatronics Engineering
National Taiwan University

Today:
- Bit manipulating
- Advanced arithmetic



Outline

- Bit manipulating
  - Set and clear
  - Bitwise conditional jump
  - Bitwise logic
  - Rotate and shift
- Arithmetic
  - 16-bit or wider arithmetic
  - Multiplication and division
  - Compare
- Getting started

Set and Clear I/O – SBI & CBI

- SBI – set bit in I/O register
  - Syntax: `SBI ioReg, bit` ioReg: | | | | | | | | |
  - Examples:
    - `SBI PORTD, 4 ;PORTD 4 = 1`
    - `SBI DDRC, 5 ;DDRC 5 = 1`
- CBI – clear bit in I/O register
  - Syntax: `CBI ioReg, bit` ioReg: | | | | | | | | |
  - Examples:
    - `CBI PORTD, 6 ;PORTD 6 = 0`
    - `CBI DDRC, 4 ;DDRC 4 = 0`

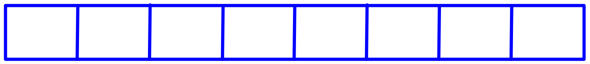
Example: Turning on LED

```
        LDI     R20, 0xFF
        OUT     DDRD, R20        ;make PORTD an output port
        SBI     PORTD, 0         ;turn on the LED of PD0
        CALL    DELAY            ;delay
        SBI     PORTD, 1         ;turn on the LED of PD1
        CALL    DELAY            ;delay
        SBI     PORTD, 2         ;turn on the LED of PD2
        CALL    DELAY
        SBI     PORTD, 3
        CALL    DELAY
        SBI     PORTD, 4
        CALL    DELAY
        SBI     PORTD, 5
        CALL    DELAY
        SBI     PORTD, 6
        CALL    DELAY
        SBI     PORTD, 7
        CALL    DELAY
```

Skip Instruction – SBIS & SBIC

- SBIS – skip the next instruction if bit in I/O register **set**
  - Syntax: `SBIS ioReg, bit`
  - Example:

ioReg:



```
SBIS PIND, 5
INC  R20
LDI  R19, 0x23
```

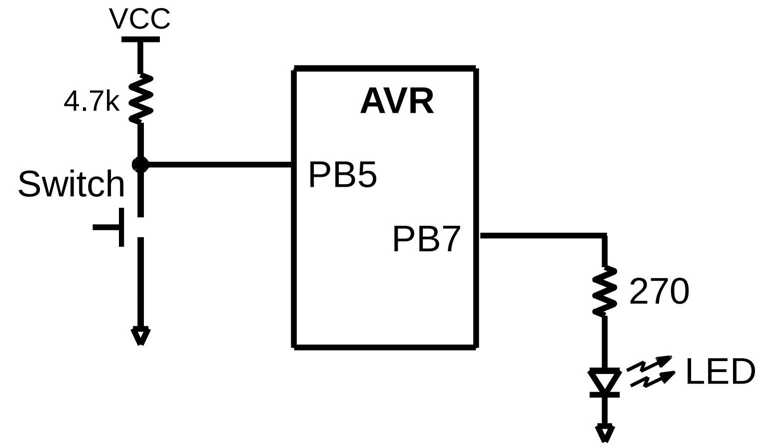
PD5==1

- SBIC – skip the next instruction if bit in I/O register **cleared**
  - Syntax: `SBIC ioReg, bit`
  - Example:

```
SBIC PIND, 5
INC  R20
LDI  R19, 0x23
```

PD5==0

Example: LED Switch



- A switch is connected to pin 5 of Port B (PB5) and an LED to pin 7 of Port B (PB7)
- Turn on the LED if PB5 is HIGH; otherwise, turn off the LED
- Check PB5 indefinitely

```
        CBI   DDRB, 5          ;make PB5 an input
        SBI   DDRB, 7          ;make PB7 an output
AGAIN:  SBIC  PINB, 5          ;skip next if PB5 is clear
        RJMP  TURNON       PB5==0
        CBI   PORTB, 7         ;PB7 output LOW
        RJMP  AGAIN
TURNON: SBI   PORTB, 7         ;PB7 output HIGH
        RJMP  AGAIN
```

http://www.youtube.com/watch?v=9TWJFkUwLsM

Example: SBIS

- Set pin 2 of Port B (PB2) as input and enable the pull-up
- Set Port C and pin 3 of Port D (PD3) as outputs
- Keep monitoring the PB2 until it becomes HIGH
- When PB2 becomes HIGH, write value $45 to Port C, and also send a HIGH-to-LOW pulse to PD3

```
        CBI   DDRB, 2       ;make PB2 an input
        SBI   PORTB, 2      ;turn on pull-up
        LDI   R16, 0xFF
        OUT   DDRC, R16     ;make Port C an output port
        SBI   DDRD, 3       ;make PD3 an output
AGAIN:  SBIS  PINB, 2       ;Skip if Bit PB2 is HIGH
        RJMP  AGAIN         ;keep checking if LOW
        LDI   R16, 0x45
        OUT   PORTC, R16    ;write 0x45 to port C
        SBI   PORTD, 3      ;set bit PD3 (H-to-L)
        CBI   PORTD, 3      ;clear bit PD3
        RJMP  AGAIN
```

Standard Binary Operation

- Suppose there are two binary variables x and y
- Standard binary operation between bits
  - AND (•)
  - OR (+)
  - Exclusive-OR (EOR)

x **AND** y

		x	
		0	1
y	0		
	1		

x **OR** y

		x	
		0	1
y	0		
	1		

x **EOR** y

		x	
		0	1
y	0		
	1		

"Bitwise" Logical Instructions

- Instruction:
  - `&    Rd, Rr` ;Bitwise **AND**
  - `|    Rd, Rr` ;Bitwise **OR**
  - `^    Rd, Rr` ;Bitwise **EOR**
  - `COM  Rd` ;Rd = 1' Complement of Rd (0xFF–Rd)
  - `NEG  Rd` ;Rd = **2' Complement** of Rd (0x100–Rd)
- Example:

```
     35H   0011 0101
     0FH   0000 1111
--------------------
&    05H   0000 0101
|    34H   0011 1111
^    2CH   0011 1010
```

- Also check **ANDI** and **ORI**

Toggling with EOR

- Toggling – change the value of a bit between 0 and 1
- Syntax: **`EOR Rd, Rr`**

R19':								
R20:	0	0	0	0	1	0	0	0
R19:	0	0	0	0	0	0	0	0

x **EOR** *y*

		x 0	*x* 1
y	0		
	1		

```
LDI   R20, 0x08
LDI   R19, 0
EOR   R19, R20
EOR   R19, R20
```

Bitwise Manipulation Example in C Language

- A door sensor is connected to PB1, and an LED is connected to PC7
- Write an AVR C program to monitor the door sensor and, when it opens, turn on the LED

```
#include <avr/io.h>
int main(void)
{
    DDRB &= 0b11111101;
    DDRC |= 0b10000000;
    while(1)
    {
        if(PINB & 0b00000010)
            PORTC |= 0b10000000;
        else
            PORTC &= 0b01111111;
    }
}
```

"Bytewise" Logical Instructions

- Instruction:
  - `&&   Rd, Rr ;Logic AND`
  - `||   Rd, Rr ;Logic OR`
- Example:

```
        35H   0011 0101
____00H__0000_0000_
&&      01H   0000 0000
||      34H   0000 0001
```

Logical Shift Left – LSL

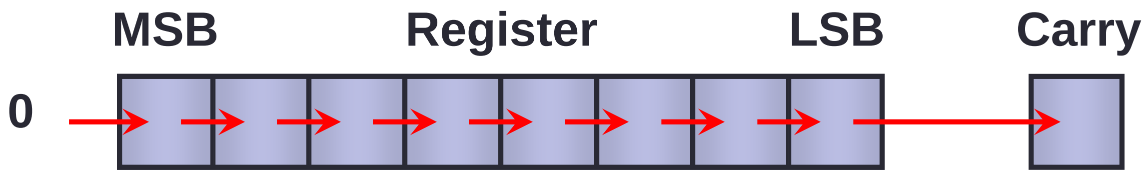
- LSL shifts bits from right to left with Carry
- 0 enters the LSB and the MSB exits to the carry flag

- This instruction multiplies content of the register by 2, assuming that after LSL the carry flag is not set
- Syntax: **LSL Rd**
- Example:

```
CLC             ;clear C flag
LDI R20, 0x26   ;R20 = 0010 0110(38)   C = 0
LSL R20         ;R20 = 0100 1100(76)   C = 0
LSL R20         ;R20 = 1001 1000(152)  C = 0
LSL R20         ;R20 = 0011 0000(48)   C = 1
```

Logical Shift Right – LSR

- LSR shifts bits from left to right with Carry
- 0 enters the MSB and the LSB exits to the carry flag

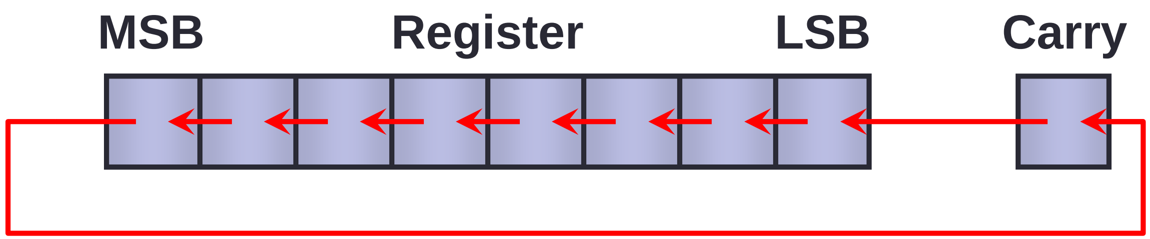


- This instruction divides content of the register by 2, and the carry flag contains the remainder of division
- Syntax: **LSR Rd**
- Example:

```
LDI R20, 0x26   ;R20 = 0010 0110 (38)
LSR R20         ;R20 = 0001 0011 (19) C = 0
LSR R20         ;R20 = 0000 1001 (9)  C = 1
LSR R20         ;R20 = 0000 0100 (4)  C = 1
```

Logical Rotate Left – ROL

- ROL rotates bits in a register with carry from right to left
- The carry flag enters the LSB and the MSB exits to the carry flag

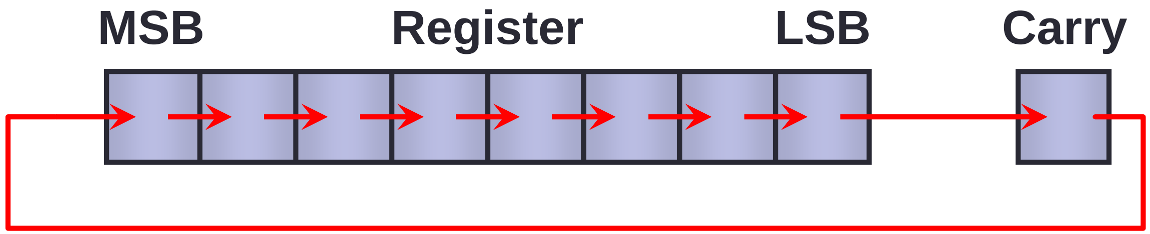


- Syntax: **ROL Rd**
- Example:

```
SEC              ;set C flag
LDI R20, 0x15    ;R20 = 0001 0101 C = 1
ROL R20          ;R20 = 0010 1011 C = 0
ROL R20          ;R20 = 0101 0110 C = 0
ROL R20          ;R20 = 1010 1100 C = 0
ROL R20          ;R20 = 0101 1000 C = 1
```

Logical Rotate Right – ROR

- ROR rotates bits in a register with carry from left to right
- The carry flag enters the MSB and the LSB exits to the carry flag

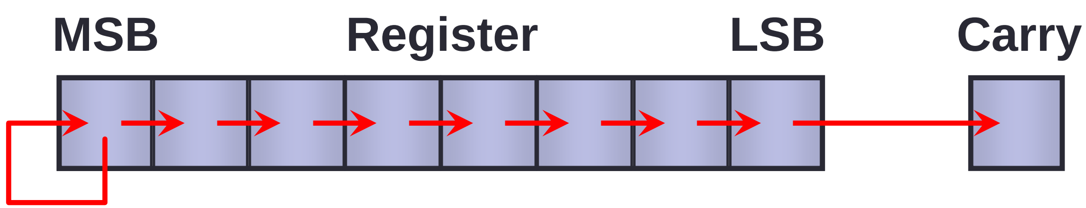


- Syntax: **ROR Rd**
- Example:

```
CLC             ;clear C flag
LDI R20, 0x26   ;R20 = 0010 0110 C = 0
ROR R20         ;R20 = 0001 0011 C = 0
ROR R20         ;R20 = 0000 1001 C = 1
ROR R20         ;R20 = 1000 0100 C = 1
```

Arithmetic Shift Right – ASR

- ASR shifts bits from left to right for signed numbers
- In ASR, the MSB is not changed but is copied to D6, D6 is moved to D5, D5 is moved to D4, and so on

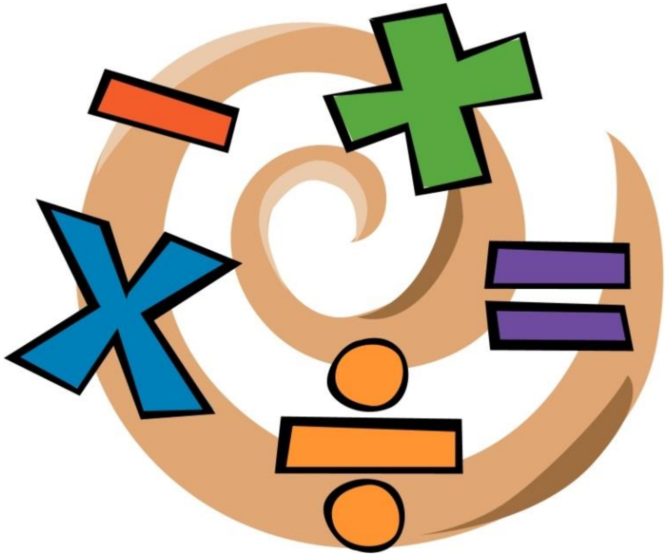


- Syntax: **ASR Rd**
- Example:

```
LDI R20, 0xB0  ;R20 = 1011 0000 (-80) C = 0
LSL R20        ;R20 = 1101 1000 (-40) C = 0
LSL R20        ;R20 = 1110 1100 (-20) C = 0
LSL R20        ;R20 = 1111 0110 (-10) C = 0
LSL R20        ;R20 = 1111 1011 ( -5) C = 0
LSL R20        ;R20 = 1111 1101 ( -3) C = 1
```

Outline

- Bit manipulating
  - Set and clear
  - Bitwise conditional jump
  - Bitwise logic
  - Rotate and shift
- Arithmetic
  - 16-bit or wider arithmetic
  - Multiplication and division
  - Compare
- Getting started



Review of ADD & SUB

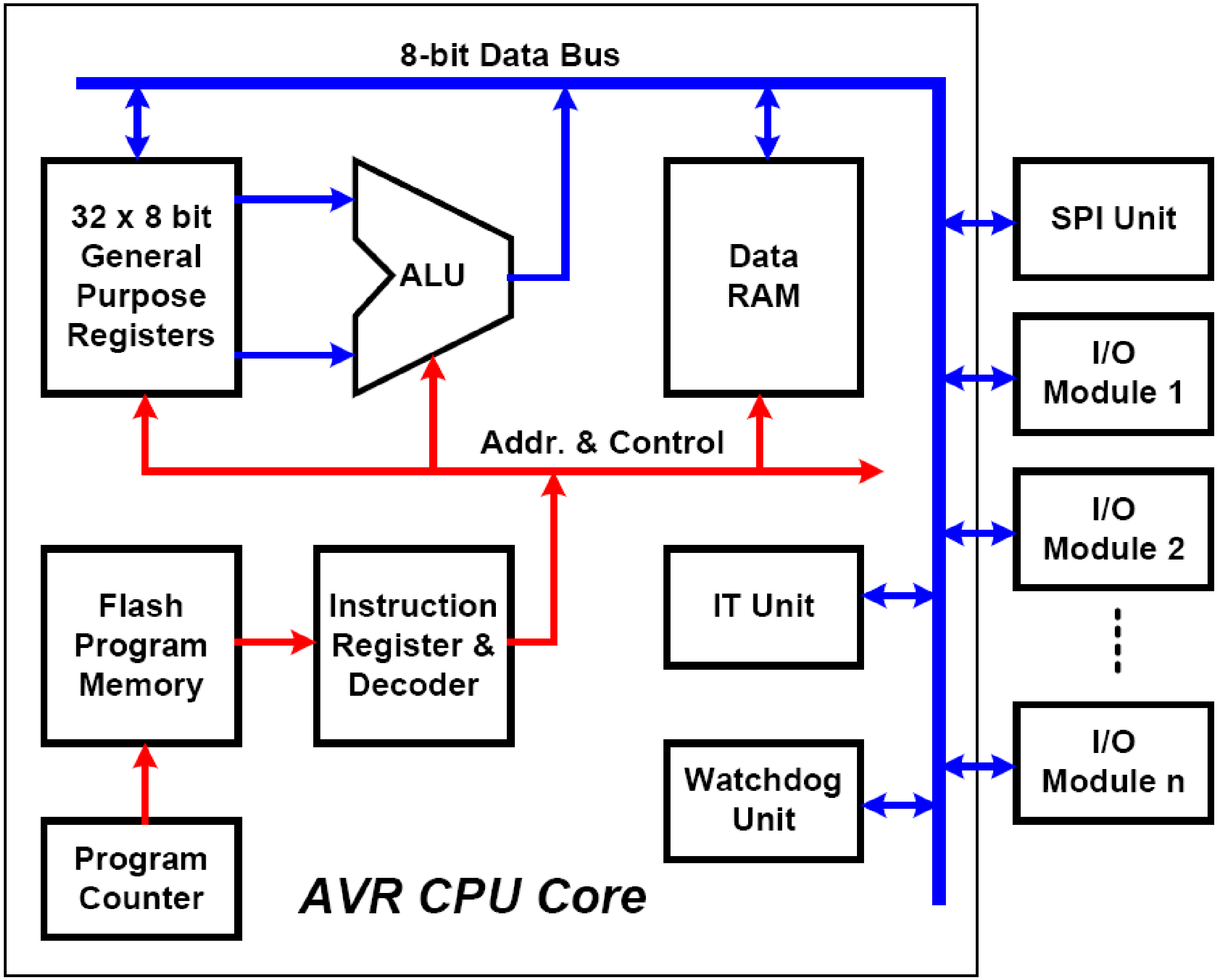
- ADD – **add** and store values in the general purpose registers
- Syntax: `ADD Rd, Rs` %Rd = Rd + Rs

where $0 \le d \le 31$, $0 \le s \le 31$

- SUB – **sub**tract and store values in the general purpose registers
- Syntax: `SUB Rd, Rs` %Rd = Rd - Rs

where $0 \le d \le 31$, $0 \le s \le 31$

8-bit Data Bus



16-bit or Wider Arithmetic

- Strategy:
  1. Several registers to hold values

Word-length	Registers
8-bit	R24
16-bit	R25:R24
32-bit	R25:R24:R23:R22
64-bit	R25:R24:R23:R22:R21:R20:R19:R18

  2. Flag to determine carry

Instruction – **ADC** & **SBC**

- ADC – **ad**d two registers with the **c**arry flag
- Syntax: `ADC Rd, Rs` %Rd = Rd + Rs + C flag

where $0 \le d \le 31$, $0 \le s \le 31$

- SBC – **s**u**b**tract two registers with the **c**arry flag
- Syntax: `SBC Rd, Rs` %Rd = Rd - Rs - C flag

where $0 \le d \le 31$, $0 \le s \le 31$

Add and Sub with Carry – ADC & SBC

- Add two 16-bit numbers $3CE7 and $3B8D

```
      1
    3C  E7  (H)
  + 3B  8D  (H)
  ------------------
    78  74  (H)
```

```
LDI R16, 8D
LDI R17, 3B
LDI R18, E7
LDI R19, 3C
ADD R18, R16
ADC R19, R17
```

- Subtract two 16-bit numbers $2762 and $1196

```
    27 62  (H)
  - 12 96  (H)
       1
  ------------------
    14 CC (H)
```

```
LDI R16, 96
LDI R17, 12
LDI R18, 62
LDI R19, 27
SUB R18, R16
SBC R19, R17
```

Multiplication (MUL) and Division

- MUL – **Mul**tiplication
- Syntax: **MUL Rd, Rr**
- The resulted high-byte is stored in R1, and low-byte in R0
- Example:

```
LDI R23, 0x25
LDI R24, 0x65
MUL R23, R24   ;25*65=0E99
```

where R1 = 0E(H)
R0 = 99(H)

- Division – no instruction available

$$\frac{Numerator}{Denominator} = Quotient$$

```
.DEF NUM = R20
.DEF DEN = R21
.DEF QUO = R22

    LDI NUM, 95
    LDI DEN, 10
    CLR QUO
L1: INC QUO
    SUB NUM, DEN
    BRCC L1
    DEC QUO
```

Compare – CP, CPC, CPI, CPSE

- In compare instruction, only flags are set

H	S	V	N	Z	C

- Instruction:
  - `CP Rd, Rr` ;Essentially Rd – Rr but…
  - `CPC Rd, Rr` ;Rd – Rr – C
  - `CPI Rd, k` ;Compare register and immediate
  - `CPSE Rd, Rr` ;Compare two registers and skip ;next instruction if equal

- Example – comparing two 32-bit numbers

```
CP      R0, R4
CPC     R1, R5
CPC     R2, R6
CPC     R3, R7
```

Signed Number Representation

- The entire 8-bit operand was used for the magnitude for unsigned numbers
- To represent a number as signed, the MSB is set aside for sign

D7	D6	D5	D4	D3	D2	D1	D0

MSB

- The sign is represented by 0 for positive numbers and 1 for negative numbers
- For an 8-bit signed integer, the range is -128 to 127 in 2’s complement representation

Representing Negative in 2’s Complement

- The steps in representing a negative number in 2’s complement
    1. Represent the absolute value of the number in 8-bit binary
    2. Invert the digits
    3. Add 1 with the inverted number

- Example: **`-76`**

`0100 1100`	(absolute value in binary)
`1011 0011`	(1’s complement)
`1011 0100`	(2’s complement)

Why 2's Complement?

① Only one form of 0
② Simple addition arithmetic

```
 0100  ( 4₁₀)
+1101  (-3₁₀)
-----
 0001  ( 1₁₀)
```

Two's Complement	Decimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

Why 2's Complement?

③ Simple subtraction arithmetic

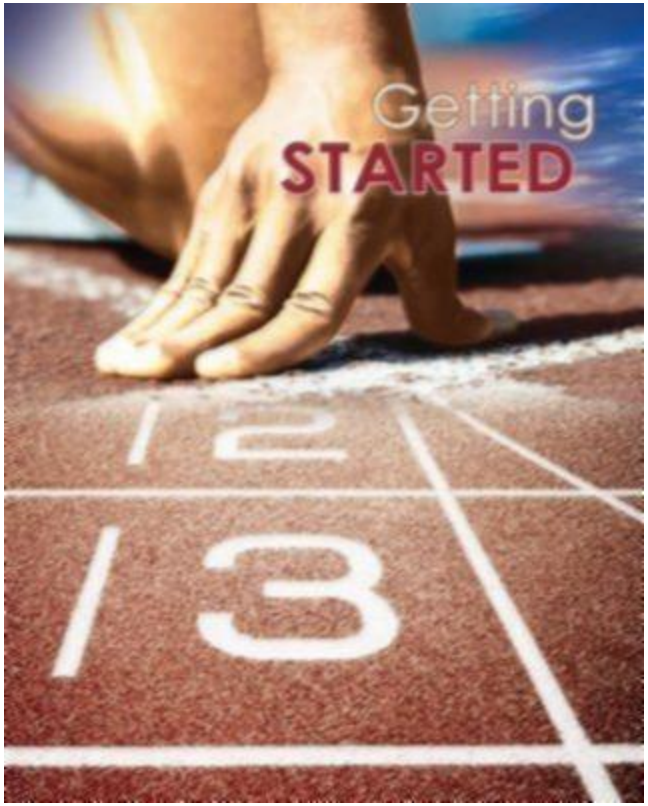
```
  1011    (-5₁₀)
 -0010  -( 2₁₀)

  1011    (-5₁₀)
 +1110  +(-2₁₀)
  ----    -----
  1001    (-7₁₀)
```

Note: ignore carry in this example

Two's Complement	Decimal
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

Outline (Cont'd)

- Bit manipulating
  - Set and clear
  - Bitwise conditional jump
  - Bitwise logic
  - Rotate and shift
- Arithmetic
  - 16-bit or wider arithmetic
  - Multiplication and division
  - Compare
- Getting started



Bitwise Manipulation Example in C Language

- An AVR C program that toggle all the bits of Port B 200 times with a delay of 1 second

```
#include <avr/io.h>
#include <util/delay.h>
#define F_CPU 1000000UL       // system clock of 1MHz

int main (void)
{
    DDRB = 0xFF;
    PORTB = 0xAA;

    for(int z=0; z<200; z++)
    {
        PORTB = ~PORTB;
        _delay_ms(1000);
    }
    return 0;
}
```

Reference

- ATmega328P data sheet
- AVR 8-bit instruction set
- M. A. Mazidi, S. Naimi, and S. Naimi, *The AVR Microcontroller and Embedded Systems: Using Assembly and C*, Prentice Hall, 2010
- AVR GCC library help http://nongnu.org/avr-libc/user-manual/modules.html