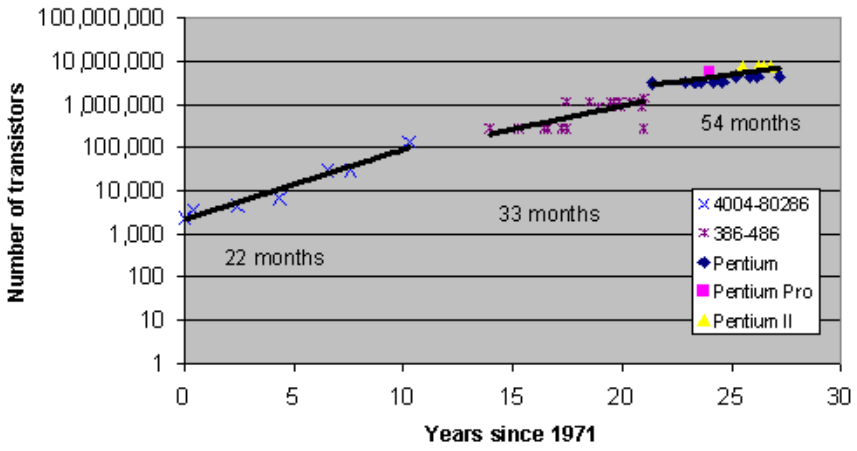
Microprocessor & Programming

School of Mechanical and Aero. Eng.
Seoul National University
Suk Won Cha

Microprocessors

- A group of millions of “transistors”
  - Moore’s Law: the number of transistors doubles every 2 years

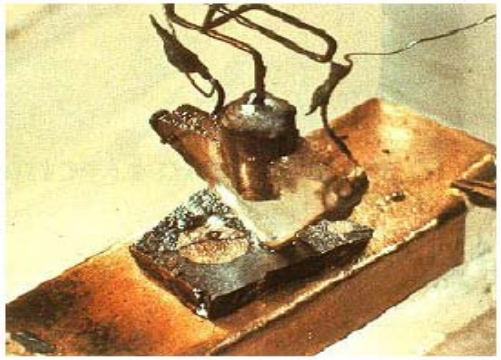


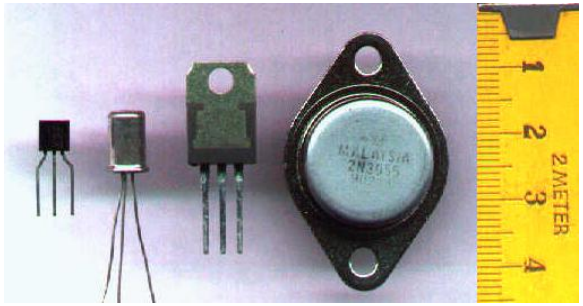
Intel 4004(1971), 4 bit processor, has 2108 transistors

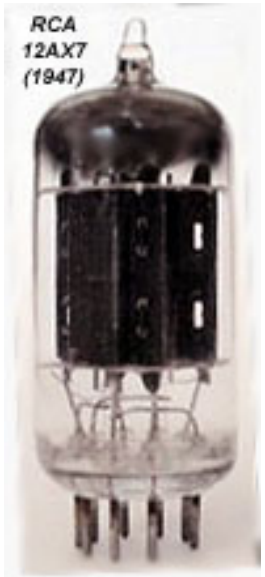
- Move data
- Arithmatic & logical
- Program flow

Transistors

- In 1947, Bardeen, Brattain and Shockley at Bell Lab invented the point contact transistor from polycrystalline Ge.(Nobel Prize 1956)
- Simple operation
  - Signal amplification
  - *Electric switch*
- Much smaller than vacuum tubes
  - ENIAC: 30 tons, 3,000 ft^3, 18000 vacuum tubes,



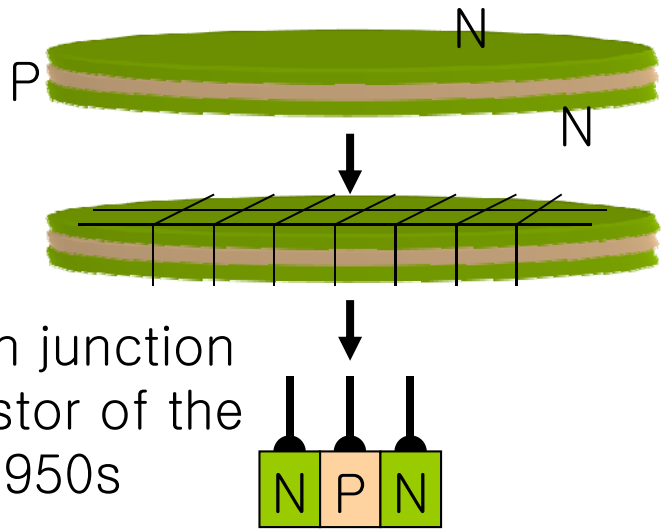




PREMA
PRECISION ENGINEERING & MANUFACTURING

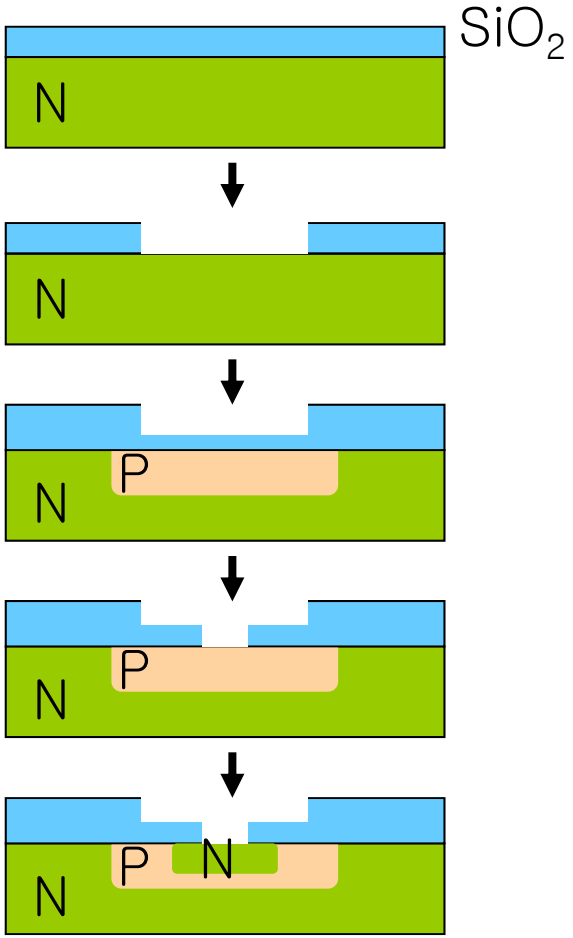
Integrated Circuit

- Transistors are NOT small enough.



Grown junction transistor of the 1950s

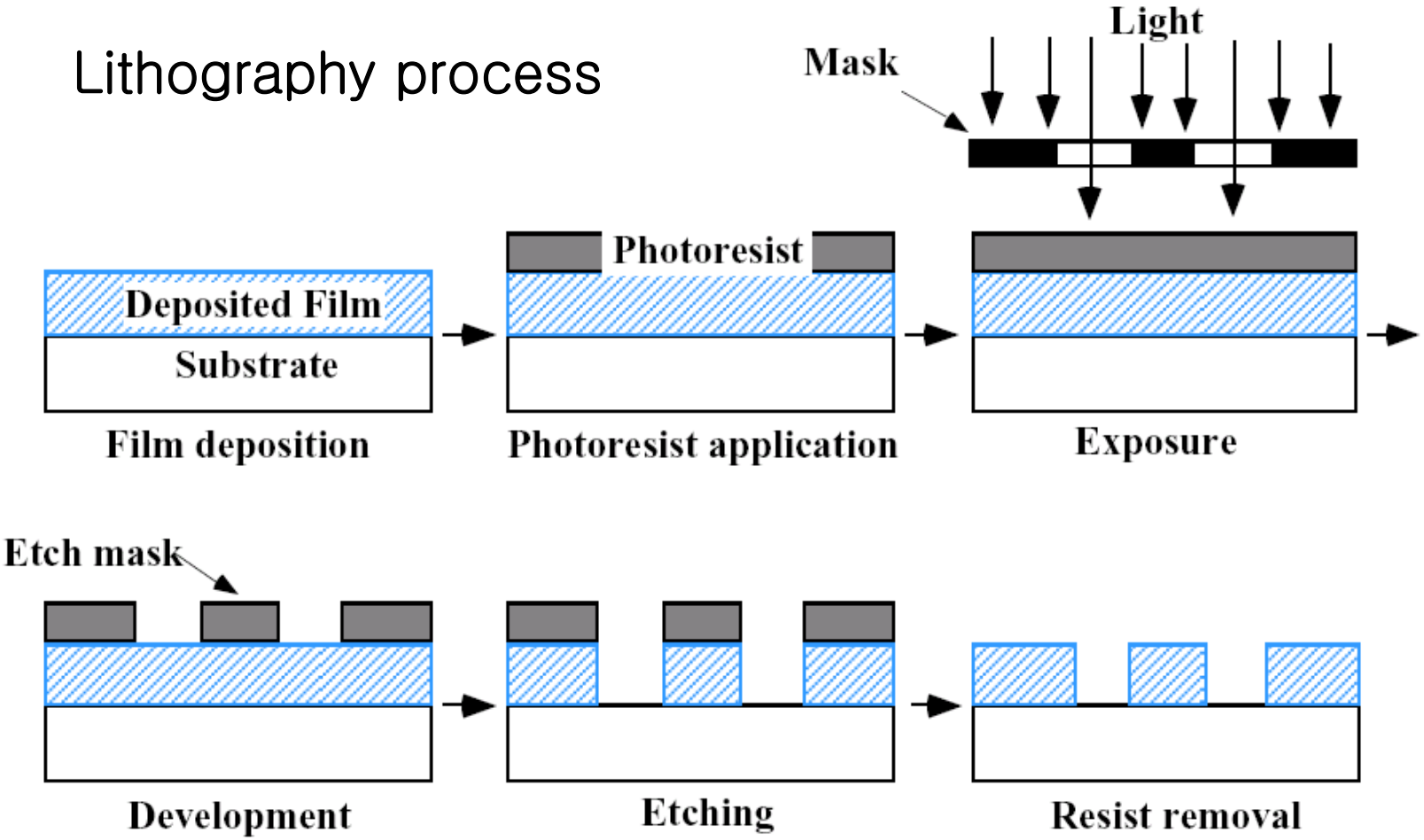
Texas Instruments' first IC



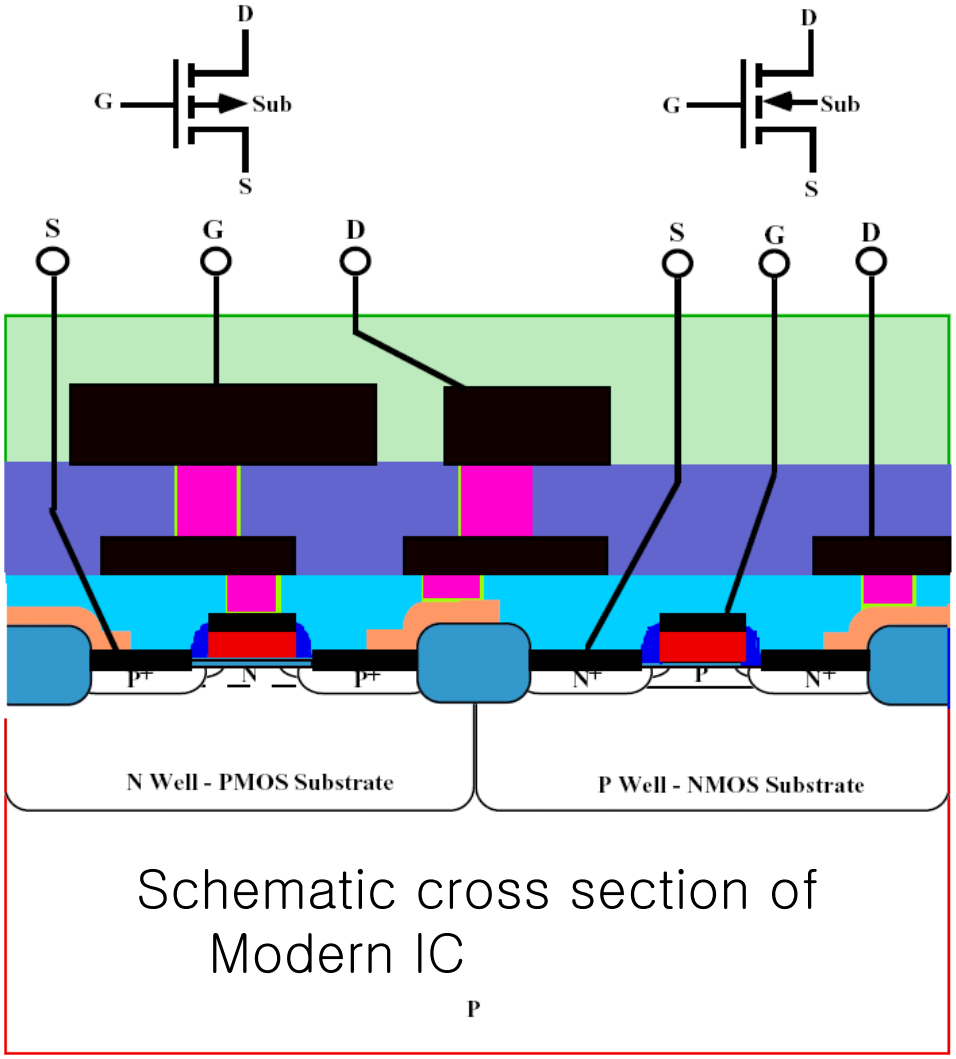
The Planar process by J. Hoerni of Fairchild in the late 1950's

Integrated Circuit

Lithography process

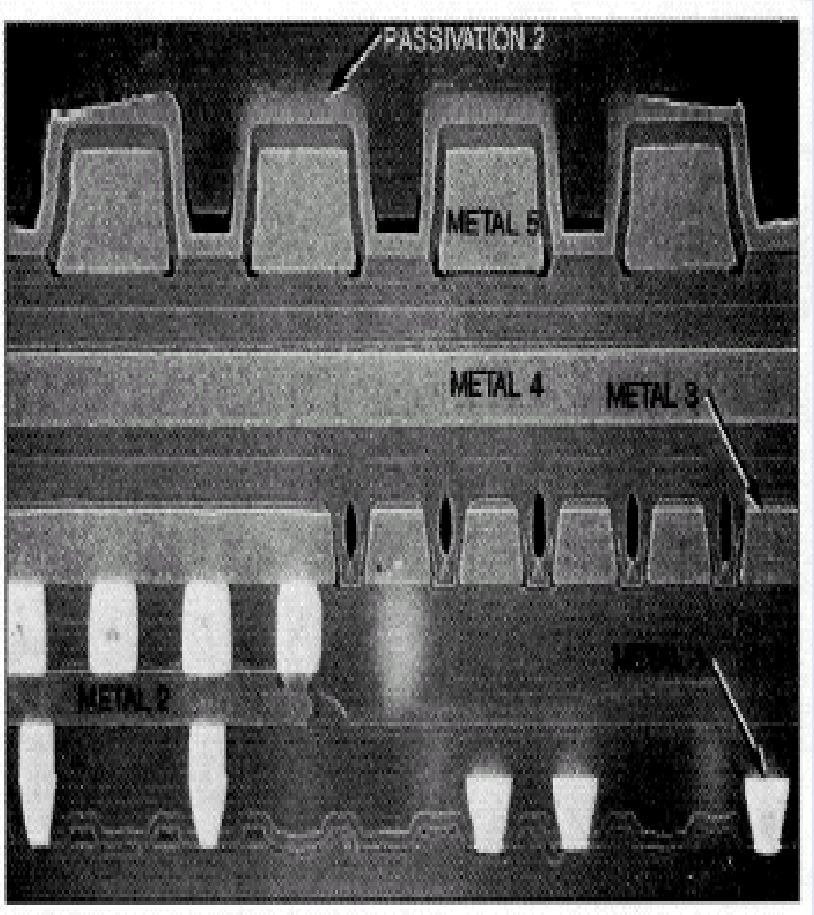


Integrated Circuit



Schematic cross section of Modern IC

P



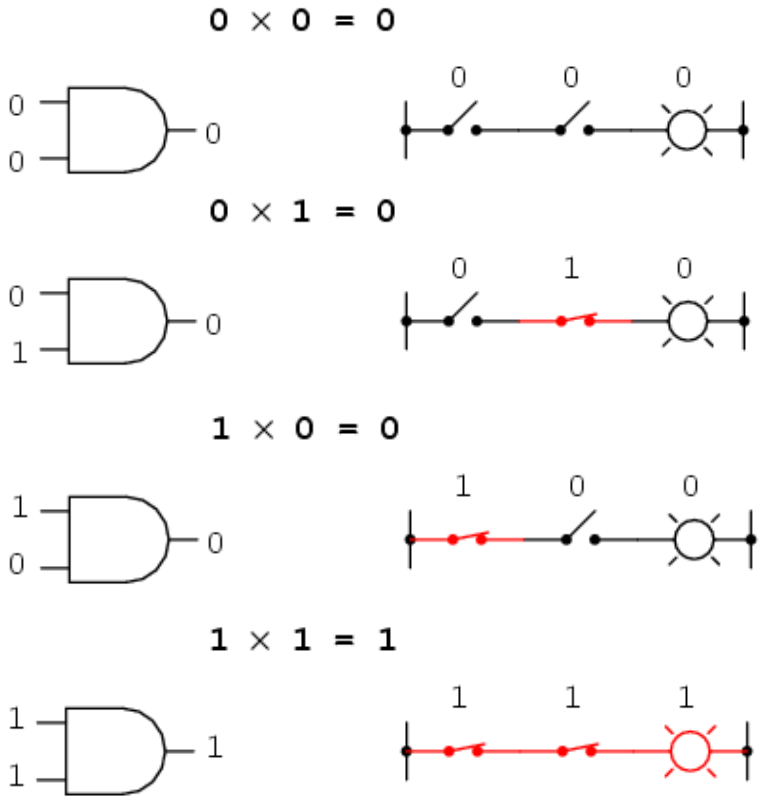
Fab Machines SNU

- Ion implanter: Varian(U)
- E-Beam: Leica(G)
- Stepper: Ultratech(U), Hitachi(J)
- Aligner: Karlsuss(G), EV(Austria)
- Etcher: STS(G), Applied Materials(U), Drytek(U)
- Bonder & Saw: Disco(J), Lapmaster(UK), EV(Austria), Karlsuss(G), Mikrotechnik(G), Kulicke & Soffa(U)...

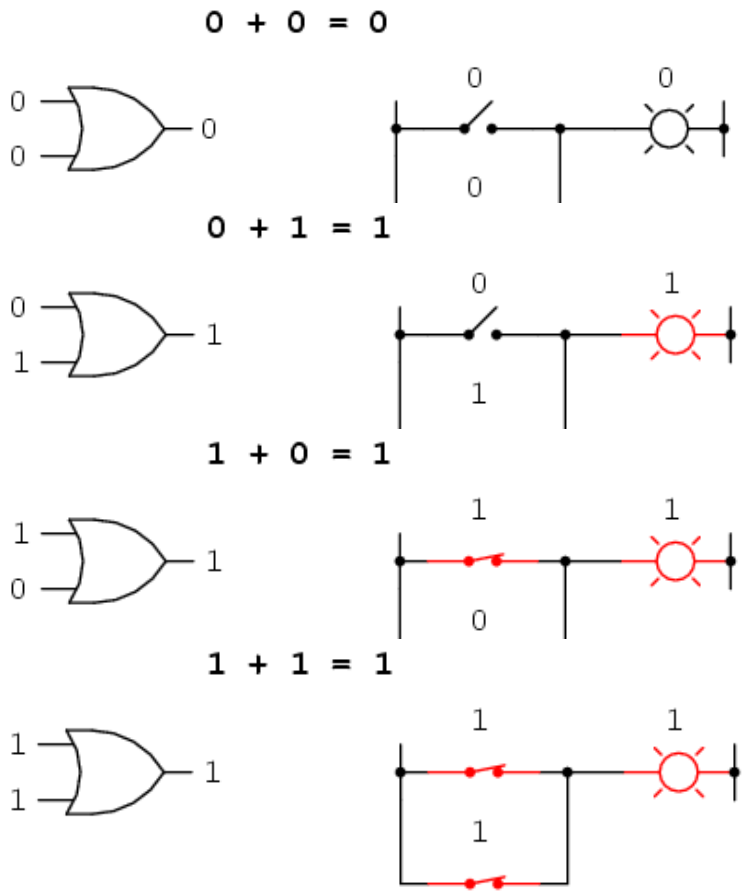
- PVD: Varian, NEVA(J), 서울진공, 코리아 Vacuum Tech...
- CVD, MOCVD: ICTEC(K), 서울일렉트론...
- Furnace: ICT(K)...

Boolean Operation

AND Gate

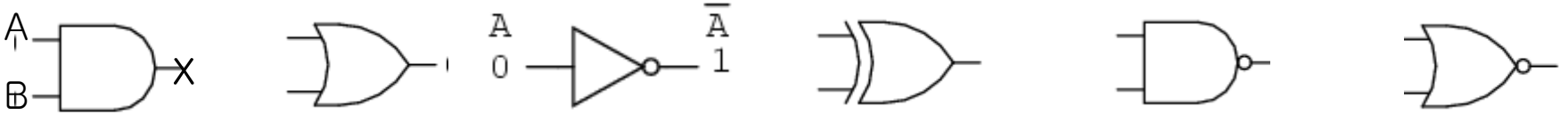


OR Gate



Boolean Operation

AND OR NOT XOR NAND NOR



A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

A	X
0	1
1	0

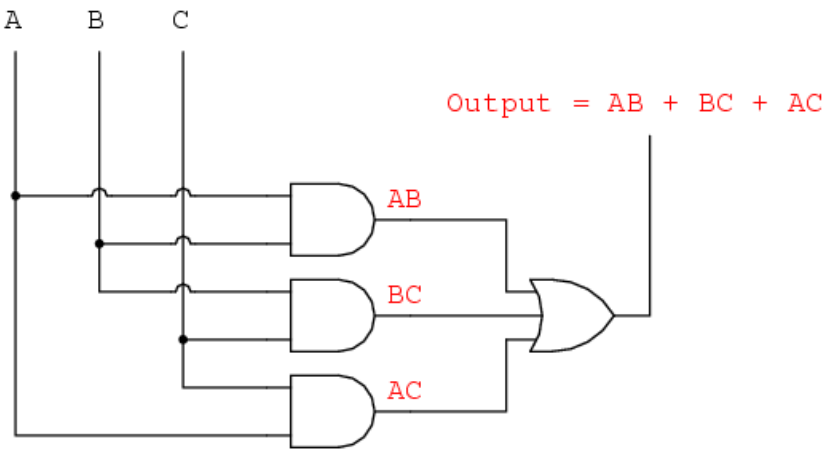
A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

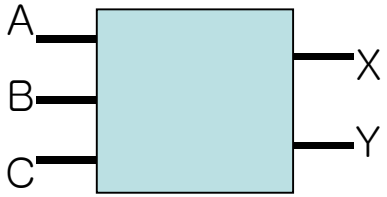
A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Example

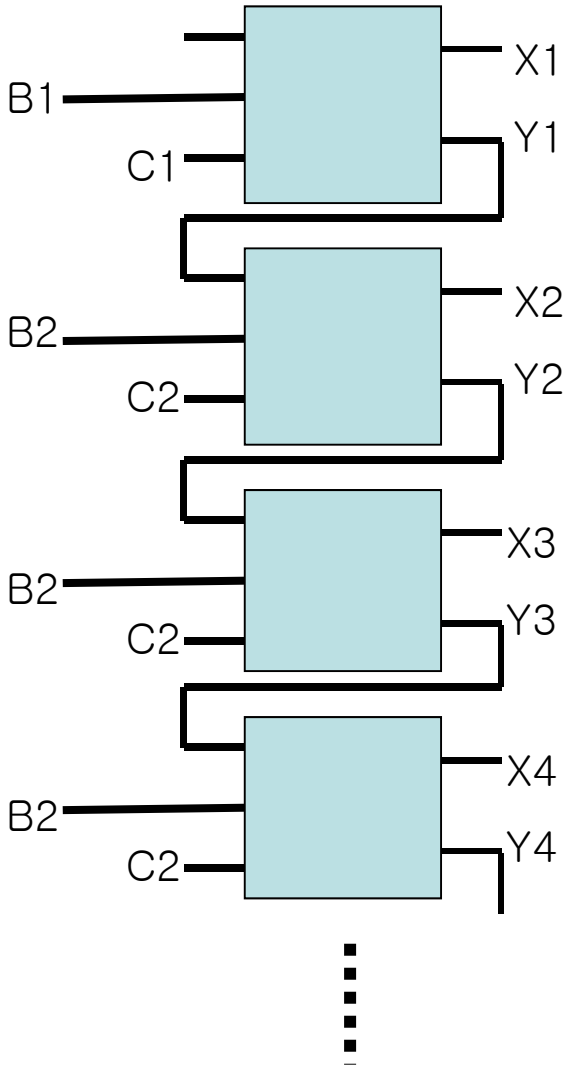
A	B	C	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



Simple Calculator



A	B	C	X	Y
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
0	1	1	0	1
1	1	1	1	1

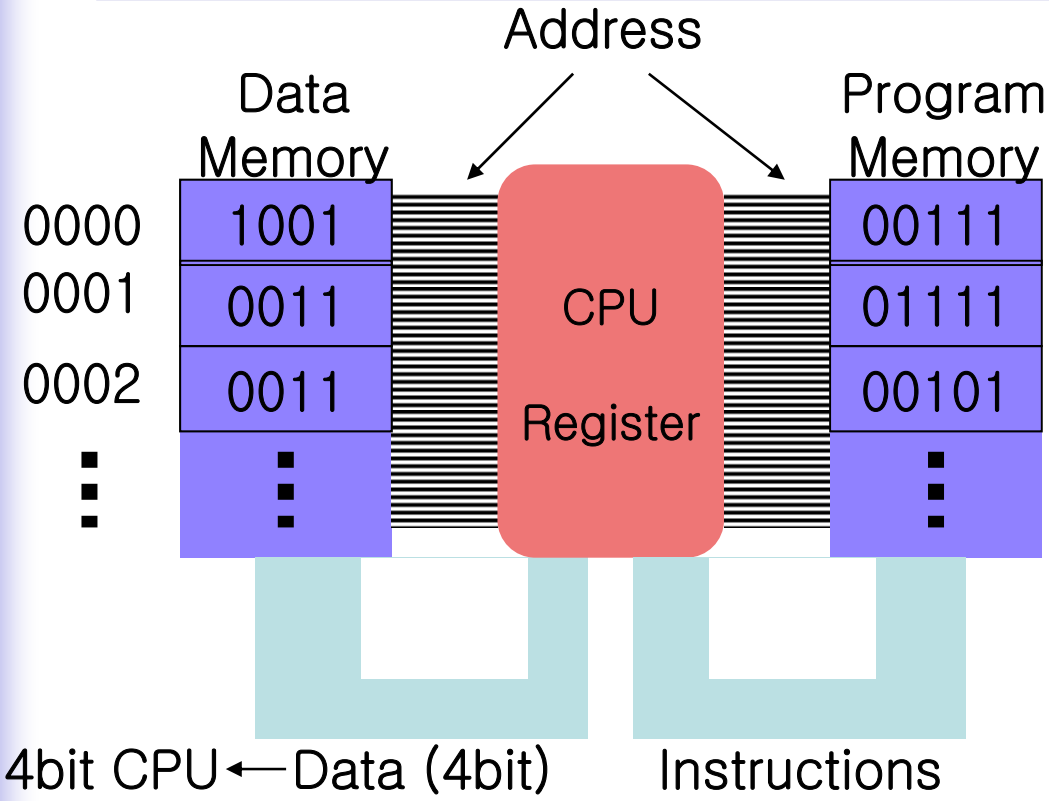


B4 B3 B2 B1= 1011 = 11

C4 C3 C2 C1= 1100 = 12

Y4 X4 X3 X2 X1 =10111 = 23

Harvard Architecture



- Separate program & data memory
- Normal instruction:

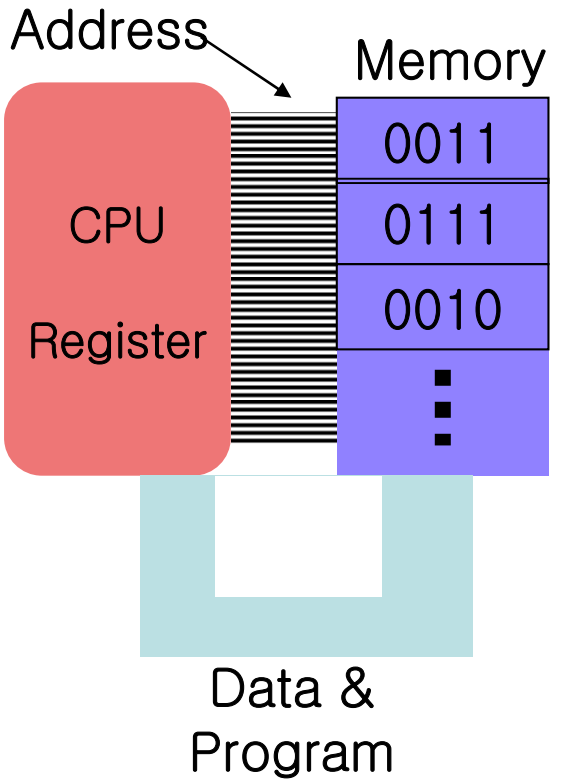
  Access data
- Special instruction:

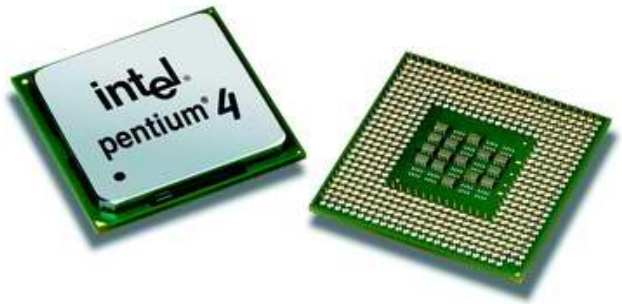
  Access program
- Retrieve program and data at the same time

AVR 8bit RISC from ATMEL Corp.

PIC 16F877-20/P from Microchip Technology Inc.

Von Neumann Architecture





- Memory store both program and data
- Read instruction or data one at a time
- No distinction between instruction and data
- Good for general applications
- Von Neumann bottle neck
- Modern computers use both Harvard and Von Neumann Architecture
  - external memory vs. cache

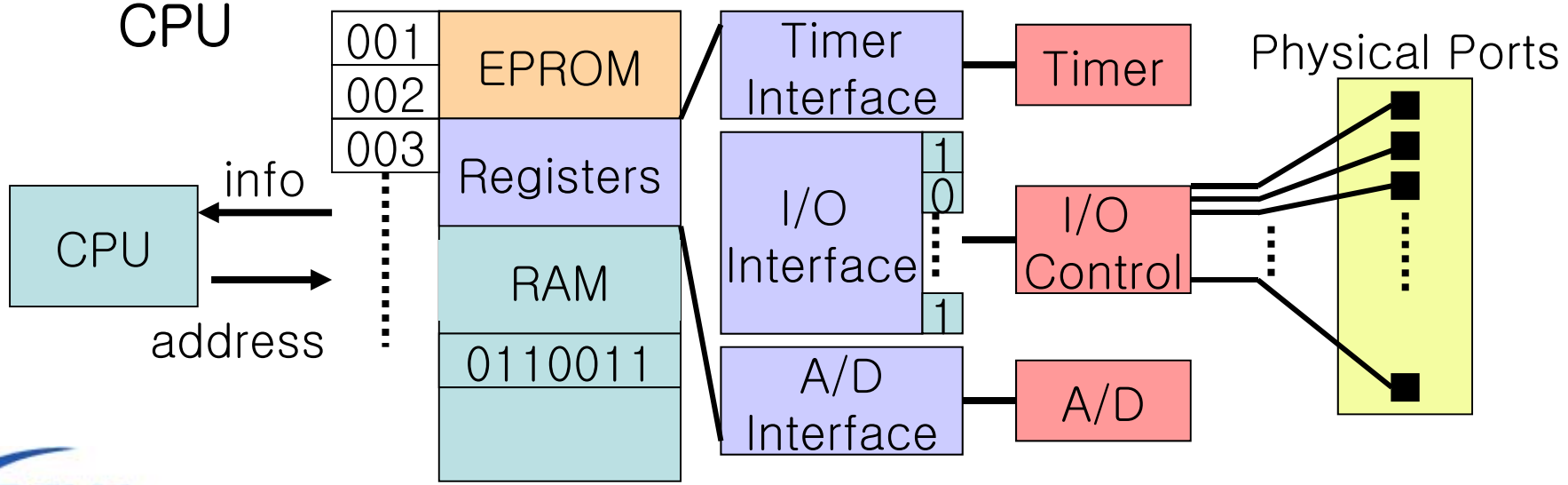
Registers & Counters

- Memory slots in CPU
- Closely related to CPU operation and status
- Don't need address (can be accessed by instruction)
- Each register dedicated to a special function
  - Accumulators: arithmetic calculation
  - Index (pointer) registers : addressing of memory
  - Stack pointer: first-in last-out memory, usually automatic (programmer doesn't worry)
  - Program counters: holds the address of the next instruction
  - Status (Condition code) register: stores most recent arithmetic result
    - Each bit represents the status of interrupt request, overflow, MSB, zero, carry or borrow...
  - I/O registers: read & write to I/O ports

.....

Memory

- Measured in Byte – 512K byte, 1M byte
- 1 byte = 1 1 1 0 1 0 1 1 (= EA_{16})
- No type information
- RAM vs ROM
- Conveniently represented by Hexadecimal
- Each memory slot has“Address” to be access by CPU



RISC vs CISC

- Reduced Instruction Set Computer
  - Many registers
  - Single cycle execution
  - Fixed width instruction
  - Prefer large, fast memory (large code)

  Apple's iMac, Power Mac (switching to Intel?), SUN, HP...

Apple's G5 RISC Processor

- Complex Instruction Set Computer
  - Many address mode
  - Complicated circuit (hardware orientated)
  - Multi cycle execution
  - Short programs

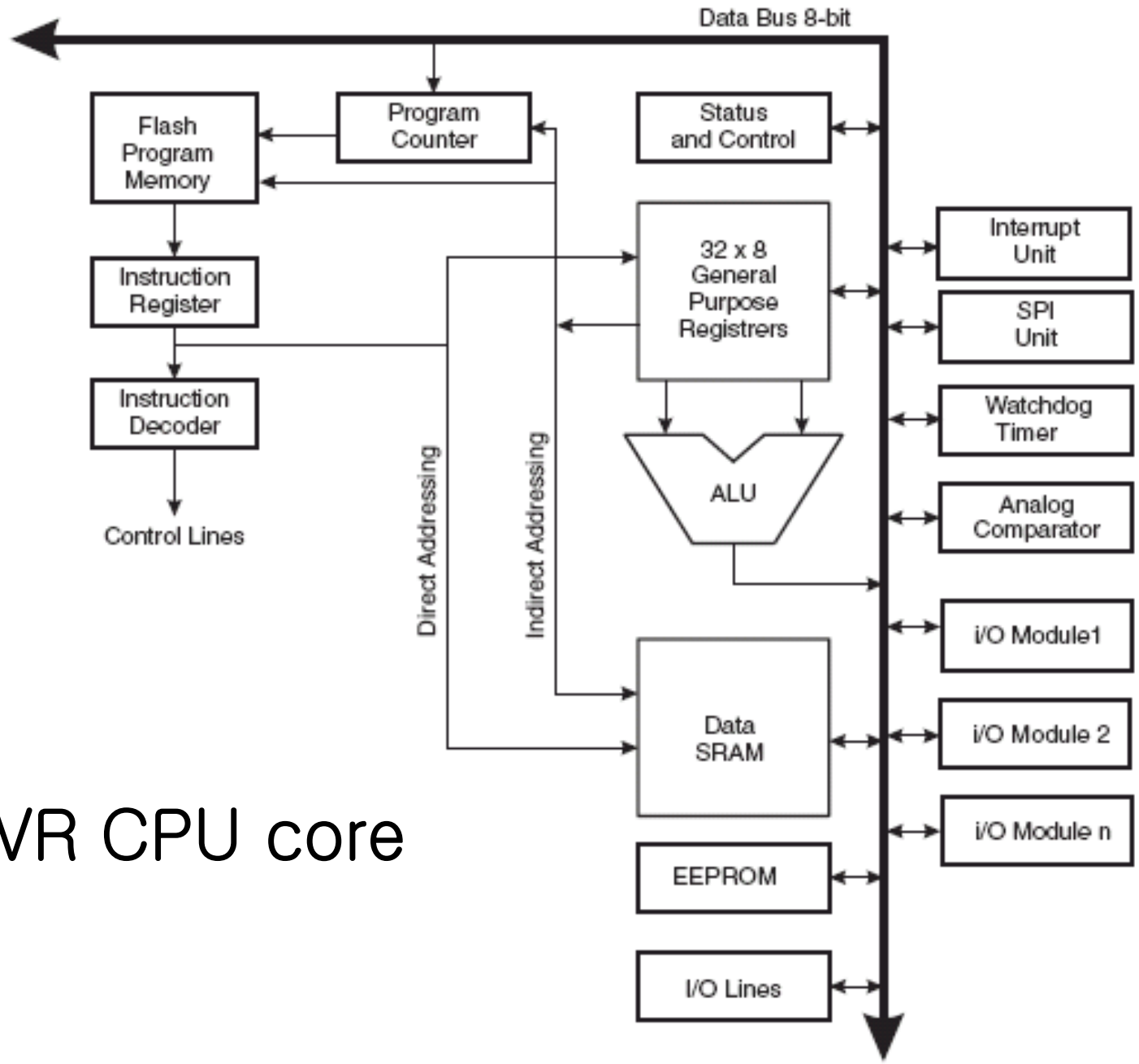
  Ex. Intel's Pentium processors (more than 200 instruction sets)


Intel Pentium 4 CISC Processor

AVR CPU Spec

RISC Architecture
Up to 16 MIPS Throughput at 16MHZ
131 Powerful Instructions
32X8 General Purpose Working Registers
16K Bytes of In-System Self-Programmable Flash
Optional Bott Code Section with Independent Lock Bits
512 Bytes EEPROM
100,000 Write/Erase Cycles
1K Byte Internal SRAM
Two 8-bit Timer/Counters
One 16-bit Timer/Counter
Real Time Counter with Separate Oscillator
Four PWM Channels
8-channel 10-bit ADC
Two-wire Serial Interface(I2C)
Programmable Serial USART
Master/Slave SPI Serial Interface
Programmable Watchdog Timer

AVR CPU core


Data Bus 8-bit
Flash Program Memory
Program Counter
Status and Control
Instruction Register
Instruction Decoder
Control Lines
32 x 8 General Purpose Registrers
ALU
Direct Addressing
Indirect Addressing
Data SRAM
EEPROM
I/O Lines
Interrupt Unit
SPI Unit
Watchdog Timer
Analog Comparator
i/O Module1
i/O Module 2
i/O Module n

AVR Registers Summary

Status

Stack

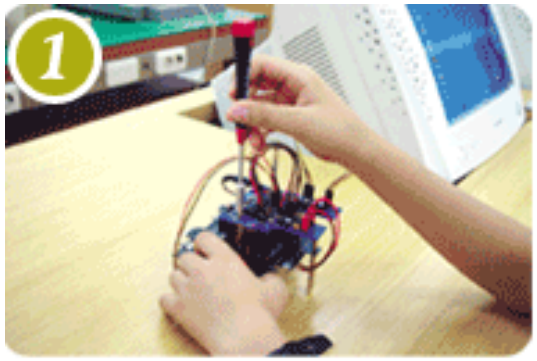
Interrupt

Timer

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0x3F (0x5F)	SREG	I	T	H	S	V	N	Z	C
0x3E (0x5E)	SPH	–	–	–	–	–	SP10	SP9	SP8
0x3D (0x5D)	SPL	SP7	SP6	SP5	SP4	SP3	SP2	SP1	SP0
0x3C (0x5C)	Reserved								
0x3B (0x5B)	GICR	INT1	INT0	–	–	–	–	IVSEL	IVCE
0x3A (0x5A)	GIFR	INTF1	INTF0	–	–	–	–	–	–
0x39 (0x59)	TIMSK	OCIE2	TOIE2	TICIE1	OCIE1A	OCIE1B	TOIE1	–	TOIE0
0x38 (0x58)	TIFR	OCF2	TOV2	ICF1	OCF1A	OCF1B	TOV1	–	TOV0
0x37 (0x57)	SPMCR	SPMIE	RWWSB	–	RWWSRE	BLBSET	PGWRT	PGERS	SPMEN
0x36 (0x56)	TWCR	TWINT	TWEA	TWSTA	TWSTO	TWWC	TWEN	–	TWIE
0x35 (0x55)	MCUCR	SE	SM2	SM1	SM0	ISC11	ISC10	ISC01	ISC00
0x34 (0x54)	MCUCSR	–	–	–	–	WDRF	BORF	EXTRF	PORF
0x33 (0x53)	TCCR0	–	–	–	–	–	CS02	CS01	CS00
0x32 (0x52)	TCNT0	Timer/Counter0 (8 Bits)							
0x31 (0x51)	OSCCAL	Oscillator Calibration Register							
0x30 (0x50)	SFIOR	–	–	–	–	ACME	PUD	PSR2	PSR10
0x2F (0x4F)	TCCR1A	COM1A1	COM1A0	COM1B1	COM1B0	FOC1A	FOC1B	WGM11	WGM10
0x2E (0x4E)	TCCR1B	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
0x2D (0x4D)	TCNT1H	Timer/Counter1 – Counter Register High byte							
0x2C (0x4C)	TCNT1L	Timer/Counter1 – Counter Register Low byte							
0x2B (0x4B)	OCR1AH	Timer/Counter1 – Output Compare Register A High byte							
0x2A (0x4A)	OCR1AL	Timer/Counter1 – Output Compare Register A Low byte							
0x29 (0x49)	OCR1BH	Timer/Counter1 – Output Compare Register B High byte							
0x28 (0x48)	OCR1BL	Timer/Counter1 – Output Compare Register B Low byte							
0x27 (0x47)	ICR1H	Timer/Counter1 – Input Capture Register High byte							
0x26 (0x46)	ICR1L	Timer/Counter1 – Input Capture Register Low byte							

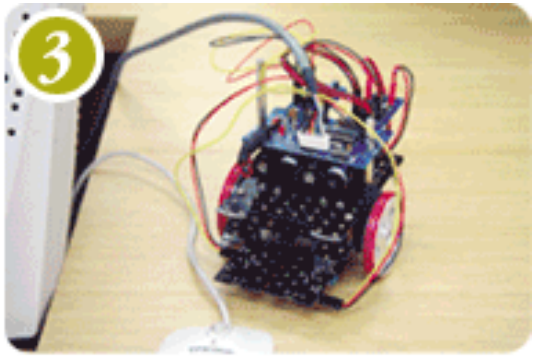
PREMA
PRECISION ENGINEERING & MANUFACTURING

Process Flow

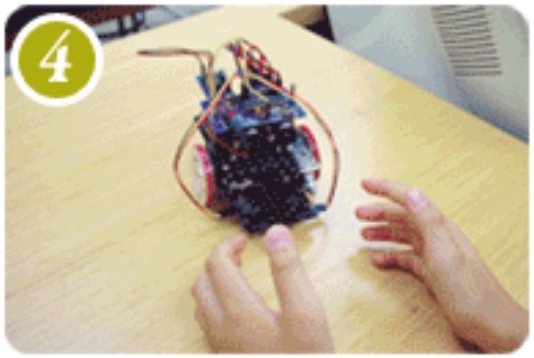


Design / Manufacturing

Coding Algorithm

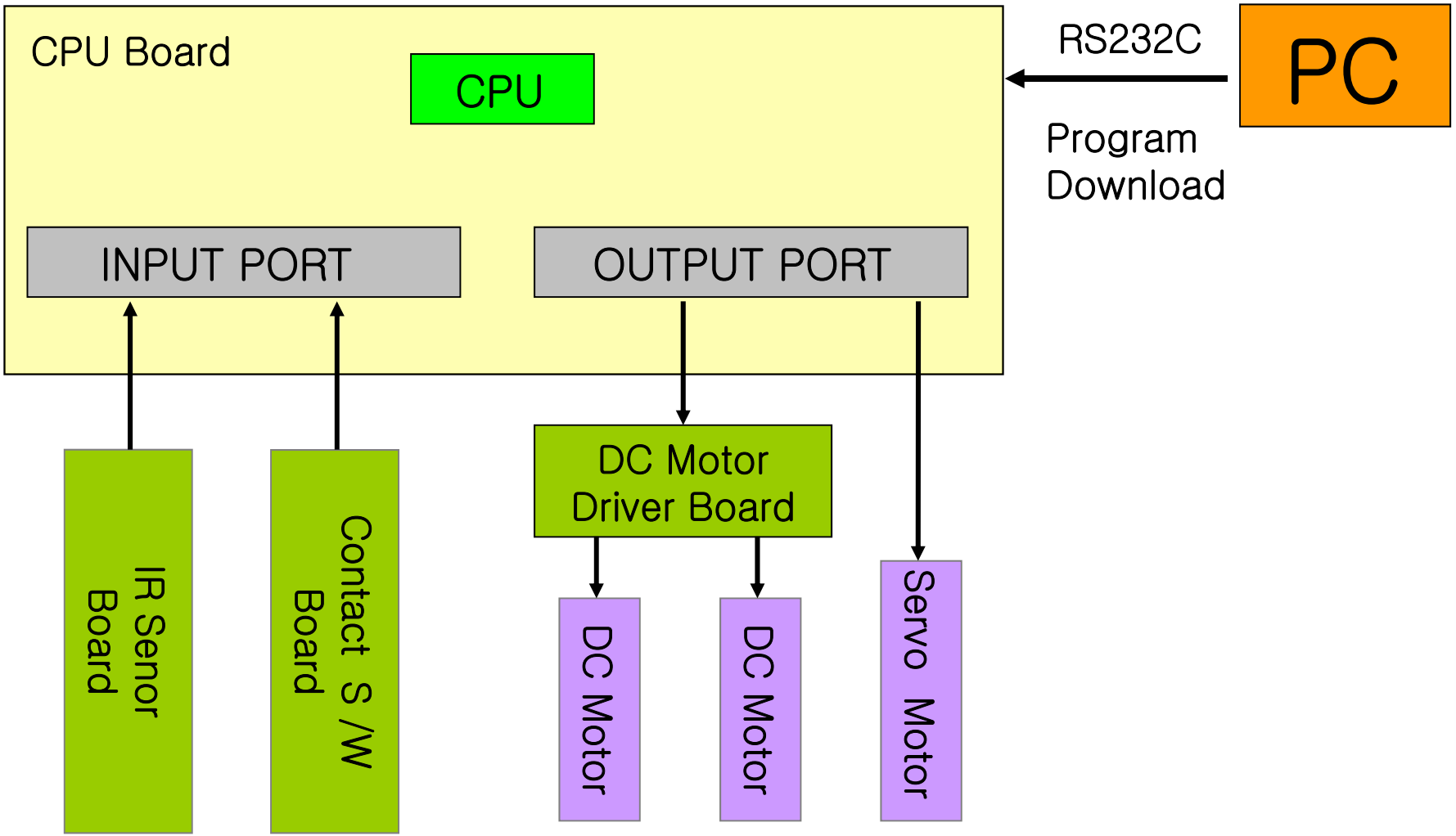


Downloading Program

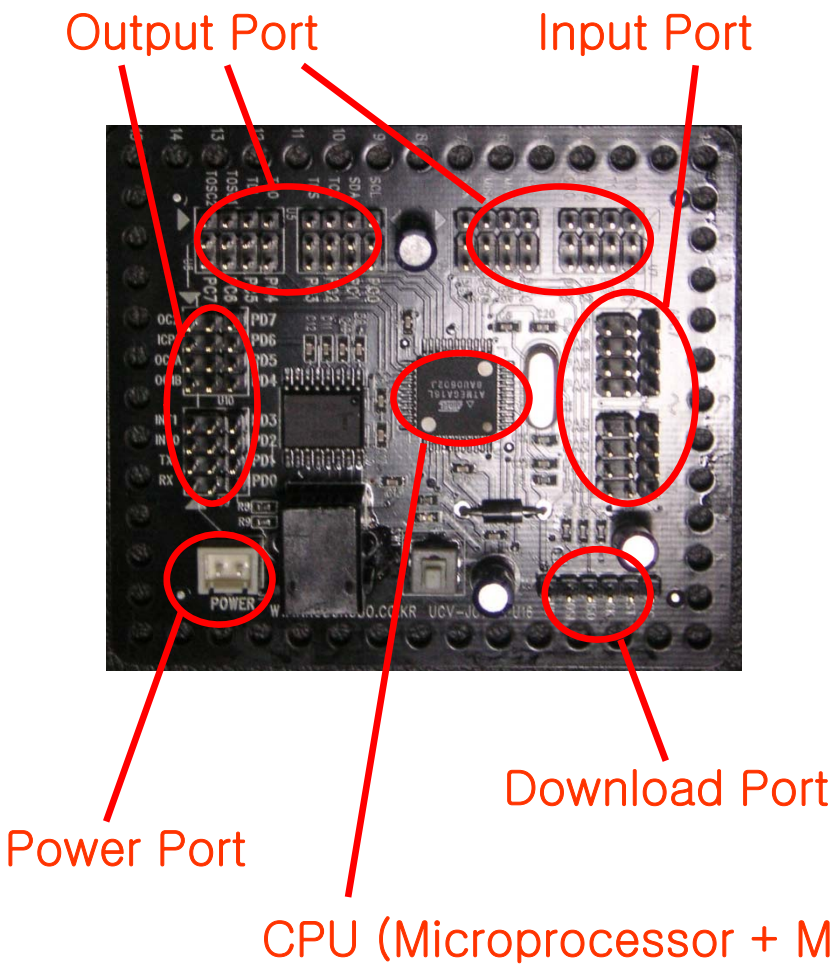


Execution

Schematic of system



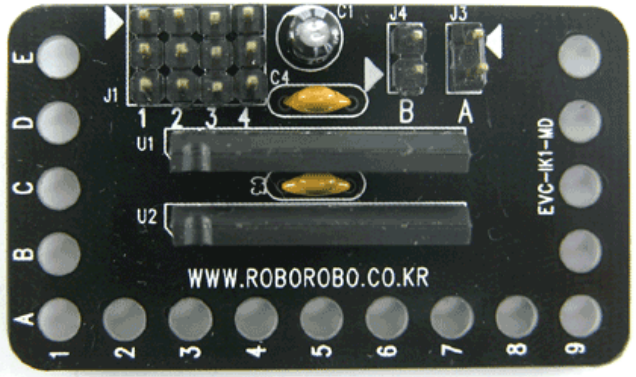
CPU Board



- 24 Output ports
  - 12 DC Motor control
  - 24 RC Servo motor control
- 8 Input ports
  - 8 sensors
- Storing of algorithm
- Execution of algorithm logic
- 16KB Programmable Memory

DC Motor & Driver

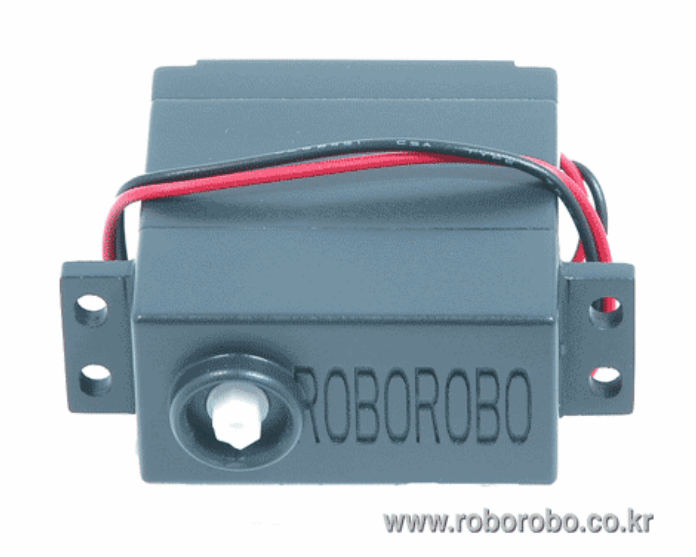
www.roborobo.co.kr



www.roborobo.co.kr

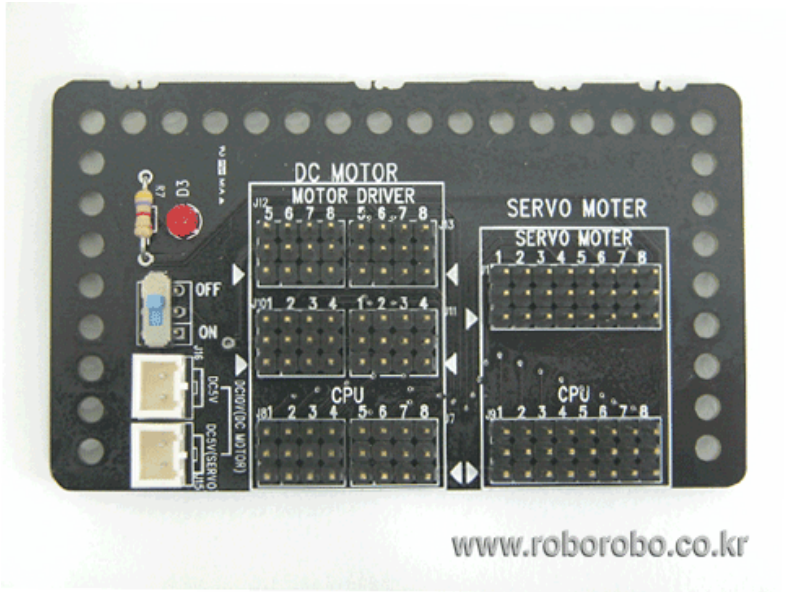
- Rotational velocity : 95.15 (12,000) rpm with gear (124.8:1)
- Rated Load Torque = 636.5 g·cm (5.1g·cm)
- Stall Load Torque = 3.7kg·cm (30 g·cm)
- Directional control (Forward/Backward)
- 2 DC motor(5V) control

Servo Motor



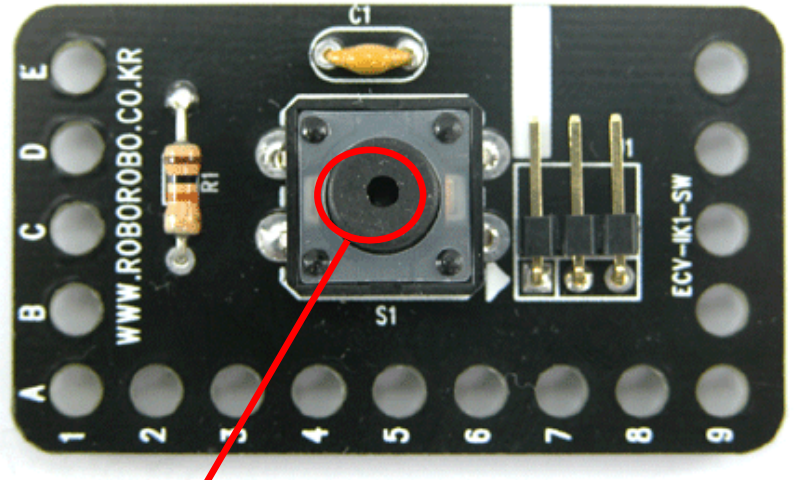
- Position Control
- Travel Range : 180°
- Absolute Encoder
  - Resolution : 11.25 ° (16 steps)
- Torque : 2.5 kg·cm

Power Interface Board



- POWER : 5V(CPU POWER) , 5V + 5V(MOTOR POWER)

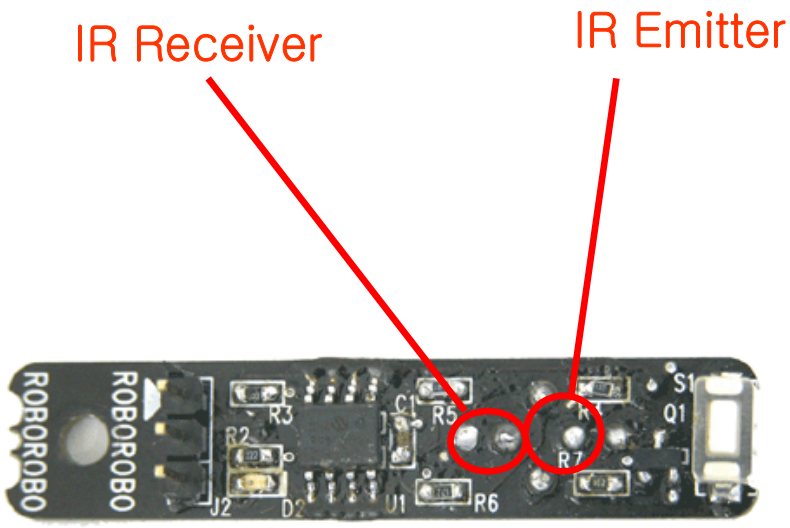
Contact Switch



Push Button

- Signal (Button state) sending to CPU
  - ex. start button of autonomous robot

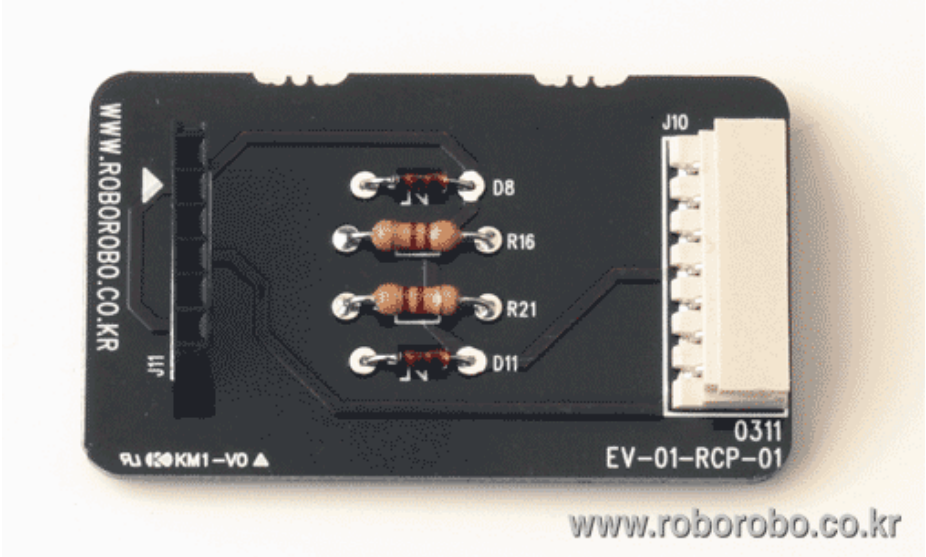
IR Sensor Board



- Sensing of reflected IR (Infrared Ray) from object
- Automatic tuning

www.roborobo.co.kr

ISP Board



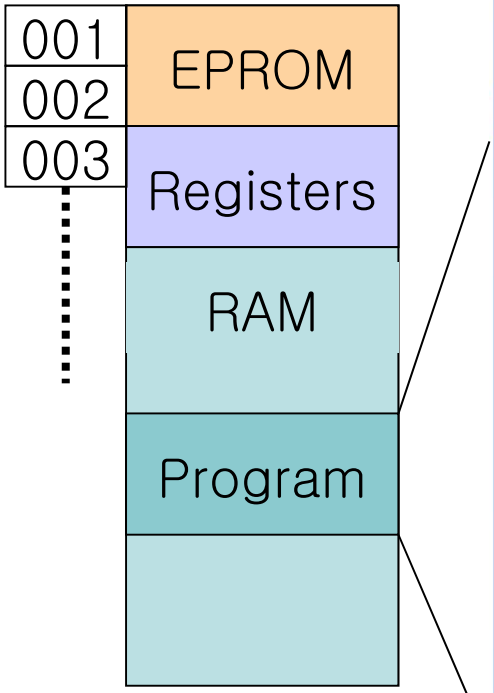
- Sending program from PC to CPU Board
- RS232C Communication (Serial port in PC)

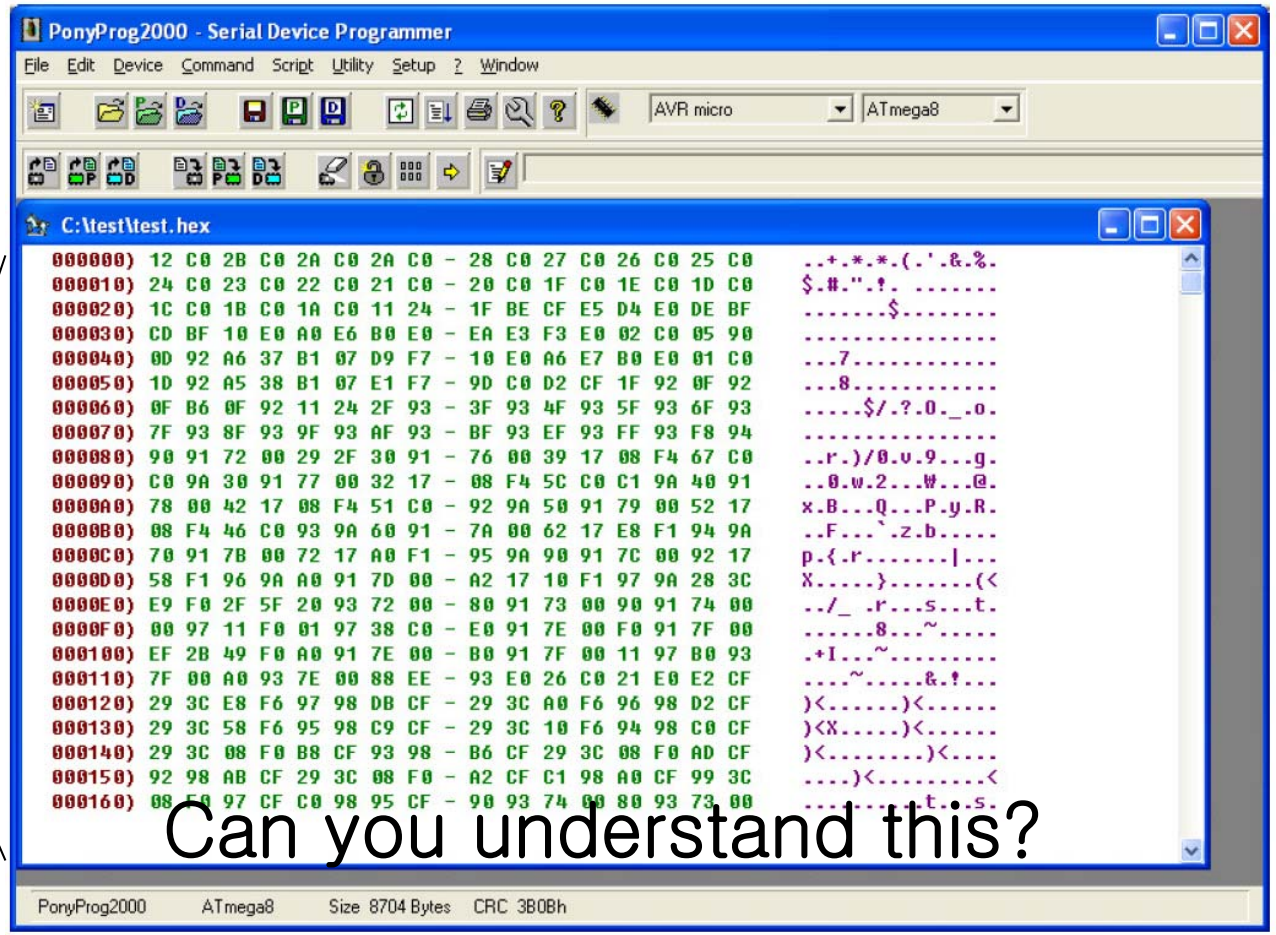
LED Board

- Useful during debugging

Programming

- CPU's understand only binary code (= machine language) as memory stores only binary information





Can you understand this?

Programming

- "Assembly Language" is a readable translation of Machine Language.

  ex) Binary (M.L.) Hex Mnemonic

  0100111 4F CLRA (clear the accumulator)

  ex)

```
0000002C A1[00000000]          mov     eax, [input1]
00000031 0305[04000000]        add     eax, [input2]
00000037 89C3                  mov     ebx, eax
```

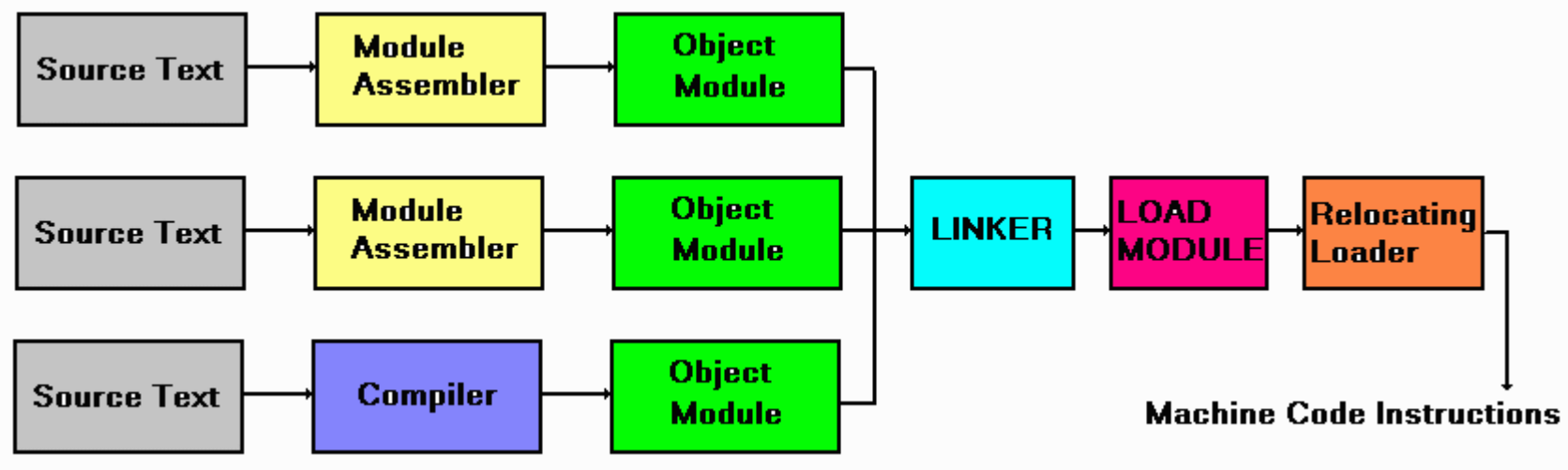
- But, still not easy to program

  ex) X = (Y + 4) * 3

```
mov eax, Y      ; move Y to the EAX register
add eax, 4      ; add 4 to the EAX register
mov ebx, 3      ; move 3 to the EBX register
imul ebx        ; multiply EAX by EBX
mov X, eax      ; move EAX to X
```

Programming

- Compiler compiles your program written in high level language such as C, C++ into machine language



"Why would you want more than machine language?"

–von Neumann's response to the concept of FORTRAN in 1954

C Language

- A programmer's language
- Terse expressive style over safety or readability
- A professional tool for optimized programming
- Highly unforgiving on mistakes!
- Low-level bit manipulation possible

```
    static signed char a[] = {0x69,
  110, 118, 97, 108, 105,  0x64, 1-1,
0x6d, 111, 118,  101, 1<<1<<1<<1<<1<<1,
114,  105, 0x6e,  103,  32, 'o'/3, 100,
32, 102, 114, 111,  0x6d, 32, 115, 116,
97, 100-001, 107, 32,37, 2*'2', '@'>>1,
116, '%' + '%' + '%','w'-'W',115, 0x74,
97, 3*'!', 107, 'q' - 'Q', 37, 10*'Wn',
10, 0}, * b = a + (1<<1<<1<<1), * w, x,
*q, c, r; int main(int d, char *e []) {
return q = (signed char *)(e+1+1), (r =
e[0] && e[1] ? 0 : 0 * puts (a) + 1) ||
(r = e[1<<1] && d != 1 <<1 && 0 * puts(
a) + 1) || e[1- -1] ||  (r = atoi(e[1])
< -0200 || atoi (e [1]) > 0x7f || ( x =
atoi( e[1] ) ) == 0 ? 0 * puts(a) + 1 :
0) || e [1- -1] || (x- -x > 1-1 ? (q[0]
  = x, q[1] = q[3] = 1, q[2] = 2) : (
    memset ( w = ( signed  char * )
```

```
    malloc(-x), '1', -x), puts (w),
  q[0] = x, q[1] = '0', q[2] = q[3] =
0)), r || (q[3] ? (c = 6 - q[1] - q[2],
(q[0] != 1) ? q[0]-- , d = q[2], q[2] =
c, main(2, e), c = q[2], q[2] = d, q[0]
++ : 0, printf(b, q[0], q[1], q[2]), (q
[0] != 1) ? q[0]--, d = q[1],q[1] = c ,
main(2, e), c = q[1], q[1] = d, q[0] ++
: 0) : - 1 - q[0] - 1 == 0 ? (w[- x - 1
- (q[1] & 1 ^ 1)] = q[1], puts (w), w [
- x - 1 - (q[1] & 1)] = q[1], puts(w) )
: - 1 - q[0] == 0 ?  (w[- x - 1] = q[ 1
  ], puts(w)) : (q[0] += 1 + ( q[1] & 1
      ^ 1), main(2, e), q[0] -= 1 + ( q
            [1] & 1 ^ 1), q[1] & 1 ? (q
               [0]+=1+1,  q[1]^=1, main
               (2, e), q[1]^=1, q[0]-=1
               +1) : 0, w[q[0] - x] = q
              [1], puts(w), q[1] & 1 ?
            0 : (q[0]+=1+1, q[1]^=1,
          main (2, e), q[1]^=1, q
        [0]-=1+1), q[0] += 1 +
    (q[1] & 1),main(2,e)
  , q[0] -= 1 + (q[1]
& 1) ) ), r; }
```

1993 winner of the "obfuscated C code writing contest", E. Berkenbilt

C Language

- Data types
  - Integer : char(1 byte), short(2), int(2 or 4), long(4)
  - Floating : float(4), double(8)
  - No Boolean: 0 (false) or anything else (true)
  - Variables: i = 0; /* end of command marked by semicolon */
  - Composite data types: structure

    ex) struct fraction { int num; int denom; };
    struct fraction f1, f2;
    f1.num=22; f1.denom=7; f2=f1;
- Arrays

  ex) int scores[100]; int board [10][10][10];
  board[0][0][0] = 13; board [9][9][9]=13;
  struct fraction numbers[1000];

C Language

- Address operators
  - * Dereference (pointer)
  - & Address of something

  ex) int *intPtr; char *charPtr; struct fraction *f1, *f2

  ex) void foo() {

```
        int *p; int i;
        i = 0;
        p = &i;  /* Make p point to i */
        *p = 1;   /* Change what p points to, i, to 1  */
    }
```

- Conversion

  ex) float pi; int i; pi = 3.14; i = pi; /* i = 3 */

C Language

- Assignment

```
b= ((c = (a+2))+3);  x = y = z = 10;
If (x = 3) {}          /* always true */
x += 10;               /*  x = x + 10  */
```

=	Assign	<<=	Left shift by
+=	Increment by	&=	And with
−=	Decrement by	^=	XOr with
*=	Multiply by	|=	Or with
/=	Divide by		
%=	Mod by		
>>=	Right shift by		

C Language

- Mathematical operator

+	Addition	*var*++	Post-increment
−	Subtraction	++*var*	Pre-increment
/	Division	*var*−−	Post-decrement
*	Multiplication	−−*var*	Pre-decrement
%	Remainder		

```
ex) x = 6/4;  y=6/4.0          /* x = 1, y = 1.5*/
      x = 1; y = 1;
      printf("%d₩n", x++);     /* output:1 */
      printf("%d₩n", ++y);     /* output:2 */
      printf("%d₩n", x);       /* output:2 */
      printf("%d₩n", y);       /* output:2 */
```

C Language

- Logical operators

!	Logical Not
&&	Logical And
||	Logical Or
*	Multiplication
%	Remainder

Bitwise operators

~	Bitwise Negation
&	Bitwise And
|	Bitwise Or
^	Bitwise Ex-Or
>>	Right shift (divide by 2)
<<	Left shift (multiply by 2)

- Relational operator

>	Less than	<=	Less or equal
<	Greater than	==	equal
>=	Greater or equal	!=	Not equal

ex) if (x==3) {}

C Language

- Control structure
  - If statement

```
if (<expression>) {
  <statement>
}
else {                  /* optional */
  <statement>
}
```

ex)

```
if (x<y) {
   min = x;
}
else {
   min = y;
}
```

C Language

- Control structure
  - Switch statement

```
switch (<expression>) {
  case <expression 1>:
  case <expression 2>:
  ...
  case <expression n>:
       <statement>
       break;  /* not required but a good idea*/
  default:
       <statement>
}
```

C Language

Note: switch enables only “Jumps”.

ex) if response is ‘Y’, what is the output of following two codes?

```
switch (response) {
    case ‘Y’:
    case ‘y’:
      printf(“Yes₩n”);
    case ‘N’:
    case ‘n’:
      printf(“No₩n”);
    default:
      printf(Huh?₩n”);
}
```

```
switch (response) {
    case ‘Y’:
    case ‘y’:
      printf(“Yes₩n”);
      break;
    case ‘N’:
    case ‘n’:
      printf(“No₩n”);
      break;
    default:
      printf(Huh?₩n”);
}
```

C Language

- Control structure
  - While loop & Do-while loop

```
while (<expression>) {
  <statement>
}
```

```
do {
  <statement>
} while (<expression>)
```

* check condition before (while) or after (do-while) the loop execution

  - For loop

```
for (<initialization>; <continuation condition>; <action>) {
  <statements>
}
```

C Language

```
ex) for (i = 1; i <= 10; i++) {
            printf("%d₩n", i);     /* output: 1 to 10 */
    }
```

- Break & Continue

```
While(<expression>) {
   ...
   if (somethingAwful)
     break;
   ...
}
...  /* break ends up here */
```

```
While(<expression>) {
   ...
   if (somethingAwful)
     continue;
   ...
    /* continue ends up here */
}
...
```

C Language

- Functions
  - In C, everything is a function.
  - Nesting function is illegal.
  - “void” is useful for no return or no parameter functions, such as main procedure.
  - *Local variables* are defined in a function
  - *Global variables* are defined outside of any function
  - *Static variables* are defined in a function, but the values are retained for the next execution of the function.
  - “ return” command returns values.

C Language

```
#include <stdio.h>   /* to use a library function printf() */
void other( void );
int main() {
    extern int i;           /* Reference to i, defined below */
    static int a;           /* Initial value is zero; a is visible only within main */
    register int b = 0;     /* b is stored in a register, if possible */
    int c = 0;              /* Default storage class is auto: */
    printf( "%d₩n%d₩n%d₩n%d₩n", i, a, b, c ); /* output: 1, 0, 0, 0 */
    other();
    return; }
int i = 1;
void other( void ) {
    static int *external_i = &i; /* Address of global i assigned to pointer variable: */
    int i = 16;                  /* i is redefined; global i no longer visible: */
    static int a = 2;            /* This a is visible only within the other function: */
    a += 2;                      /* a = 2 now, but for the next call a = 4 */
    printf( "%d₩n%d₩n%d₩n", i, a, *external_i );        /* output: 16, 4, 1 */
}
```

PREMA
PRECISION ENGINEERING & MANUFACTURING

C Language

- Call by reference

```
/* Wrong */

int a = 1;
int b = 2;
swap(a, b);

void swap(int x, int y)  {
   int temp;
   temp = x;
   x = y;
   y = temp;
}
```

```
/* Correct */

int a = 1;
Int b = 2;
swap(&a, &b);

void swap(int *x, int *y)  {
   int temp;
   temp = *x;
   *x = *y;
   *y = temp;
}
```

Library functions

void motorX(signed char velocity)

- velocity = −15 ~ 15
- motor1 : PC0, PC1
- motor2 : PC2, PC3
- motor3 : PC4, PC5
- motor4 : PC6, PC7

```
Ex: motor1(15);//1번 모터 최대 속도로 정회전
    motor2(0);       //2번 모터 정지
    motor3(-15);     //3번 모터 최대 속도로 역회전
    motor4(7);       //4번 모터 7속도로 정회전
```

```
void delay(unsigned long sec);
void delay_ms(unsigned int count);
void delay_us(unsigned int count);

void on(unsigned char output_port);
void off(unsigned char output_port);
```

Library functions

Input Method : 8 input port, IN0 ~ IN7

Ex

```
while(1)
{
    if(IN0)//PA0 눌렀을 경우
    {
        motor1(15);       //모터1 정회전 최대
        motor2(15);       //모터2 정회전 최대
    }
    else if(IN1)//PA1 눌렀을 경우
    {
        motor1(-15);      //모터1 역회전 최대
        motor2(-15);      //모터2 역회전 최대
    }
    else      //아무것도 누르지 않았을 경우
    {
      motor1(0);          //모터1 정지
      motor2(0);          //모터2 정지
    }
    return 0;
}
```

Library functions

void servo(unsigned char port_number,unsigned char position);

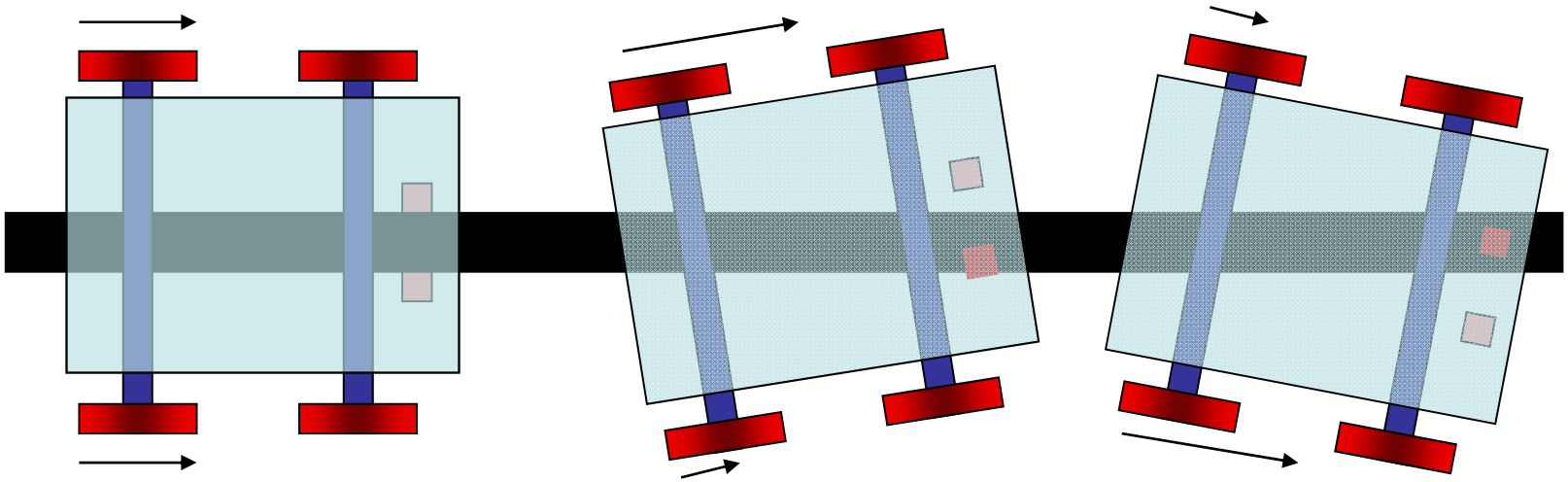
- port_number: PB0 ~ PB7,
- position: 0~205 , 모터축을 보면서 ->0(왼쪽)~205(오른쪽)
- Ex

```
servo(PB0,0);          //PB0 0위치로 이동
servo(PB1,30);         //PB1 30위치로 이동
servo(PB7,200);        //PB7 200위치로 이동
```

Line Tracer Example

```
#include “roborobo.h”
int main(void) {
    while(1)  {
        if(!IN0)  {    //PA0 sensor (left sensor) on
                motor1(5); //모터1 정회전 최대
                motor2(15);          //모터2 정회전 최대
        }
        else if(!IN1)  {  //PA1 sensor (right sensor) on
                motor1(15);          //모터1 역회전 최대
                motor2(5); //모터2 역회전 최대
        }
        else  {      //no sensor input
                motor1(15);          //모터1 정지
                motor2(15);          //모터2 정지
        }
        delay(1);
    }
    return 0;
}
```

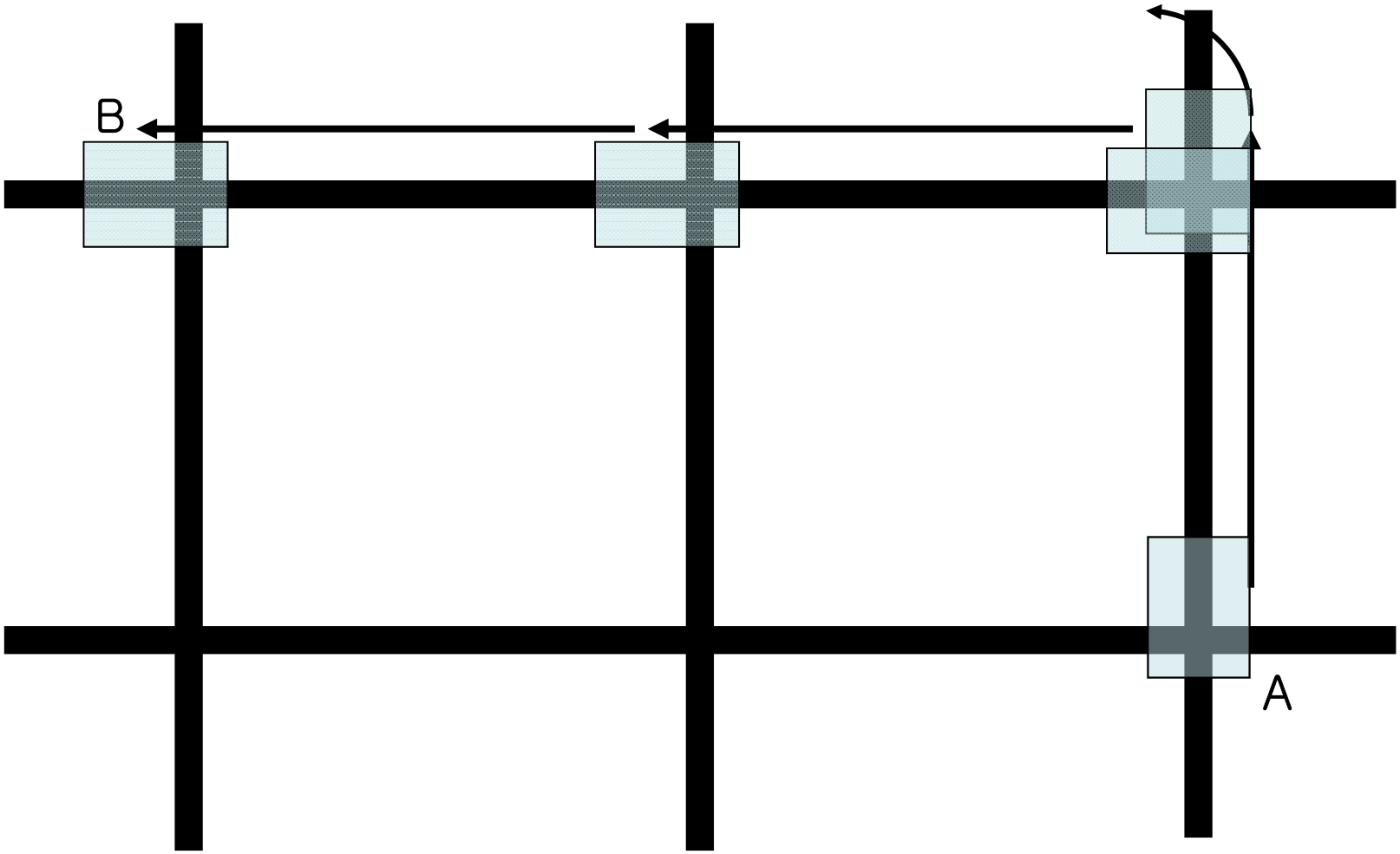
Line Tracing



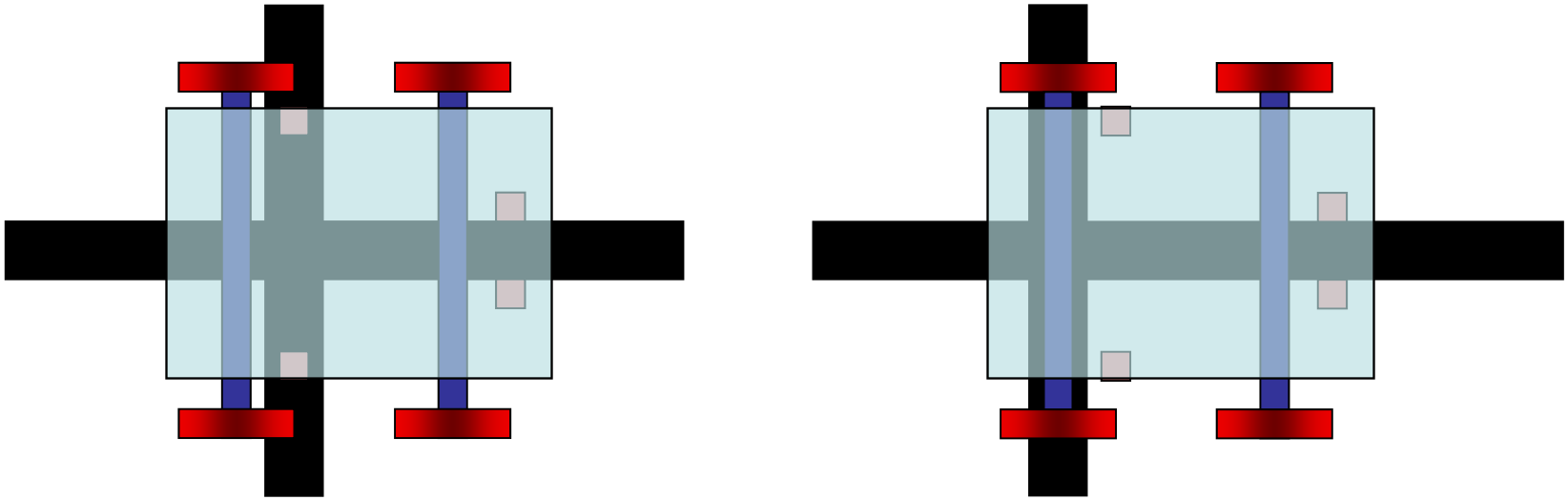
void dc_motor(L, 1, V_L); void dc_motor(R, 1, V_R)
void wait(Delay);

- Determining V & Delay: trial & error method
- Source of error: battery, loading, wheel, floor, motor speed...

Reference Position

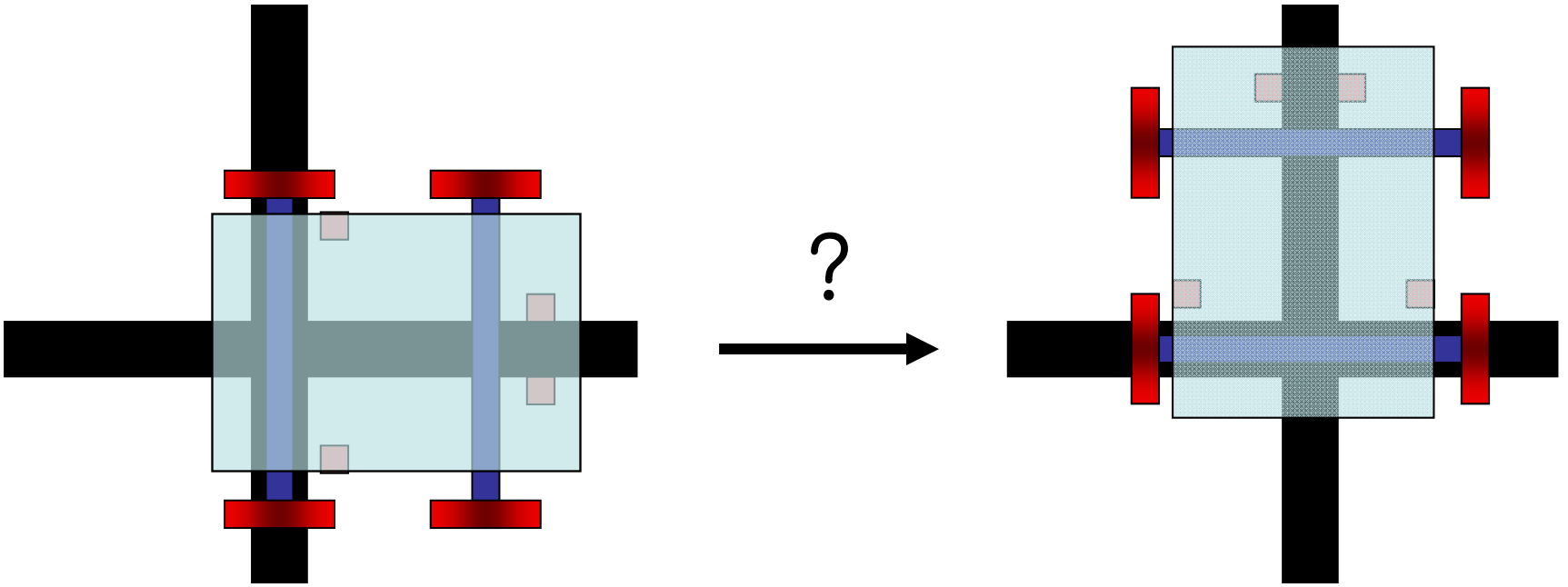


Line Counting



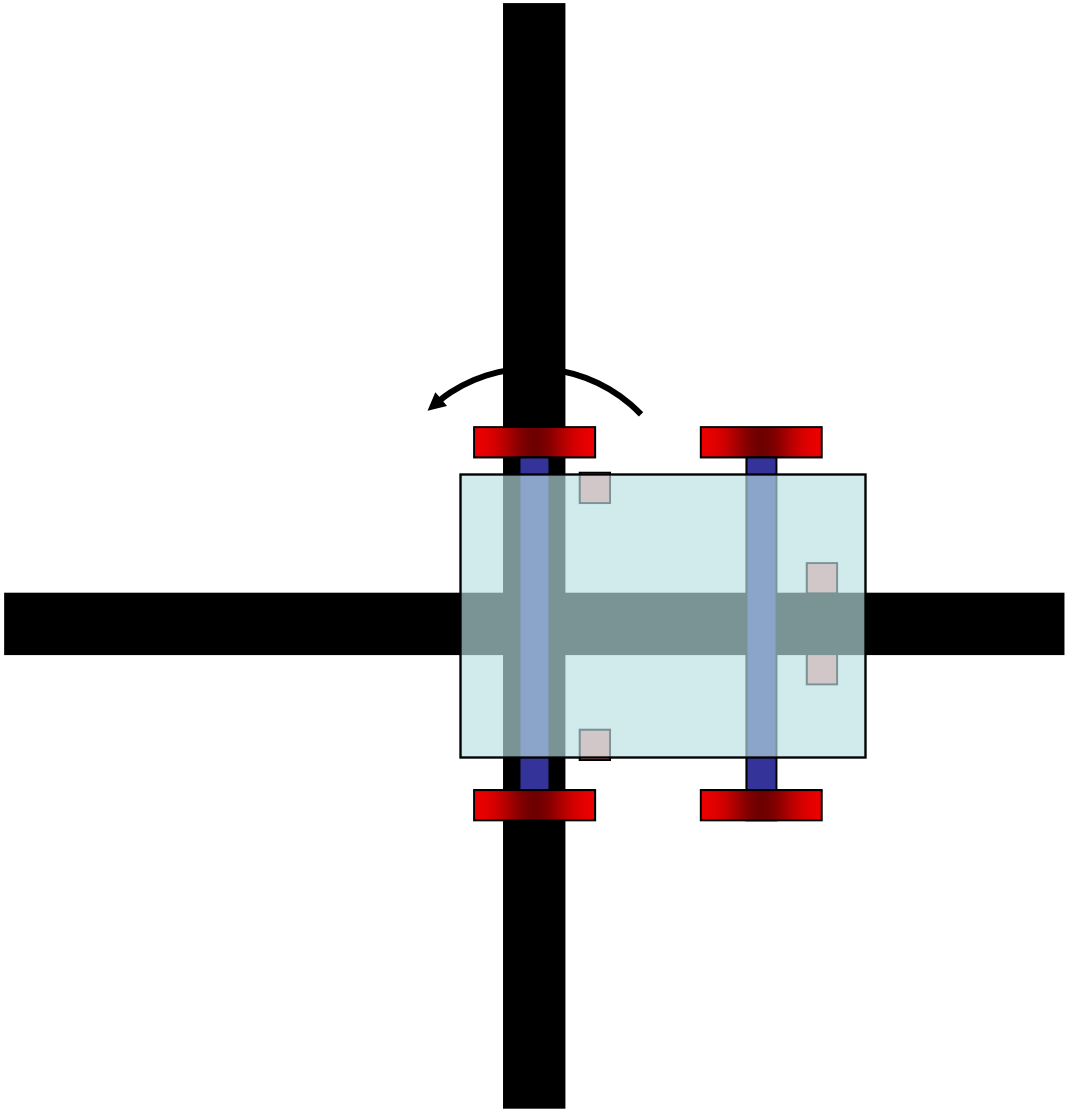
- One sensor or two sensor
- Most critical for position calculation
- Careful not to miss counting during line tracing
- May use LED board while debugging

Turning

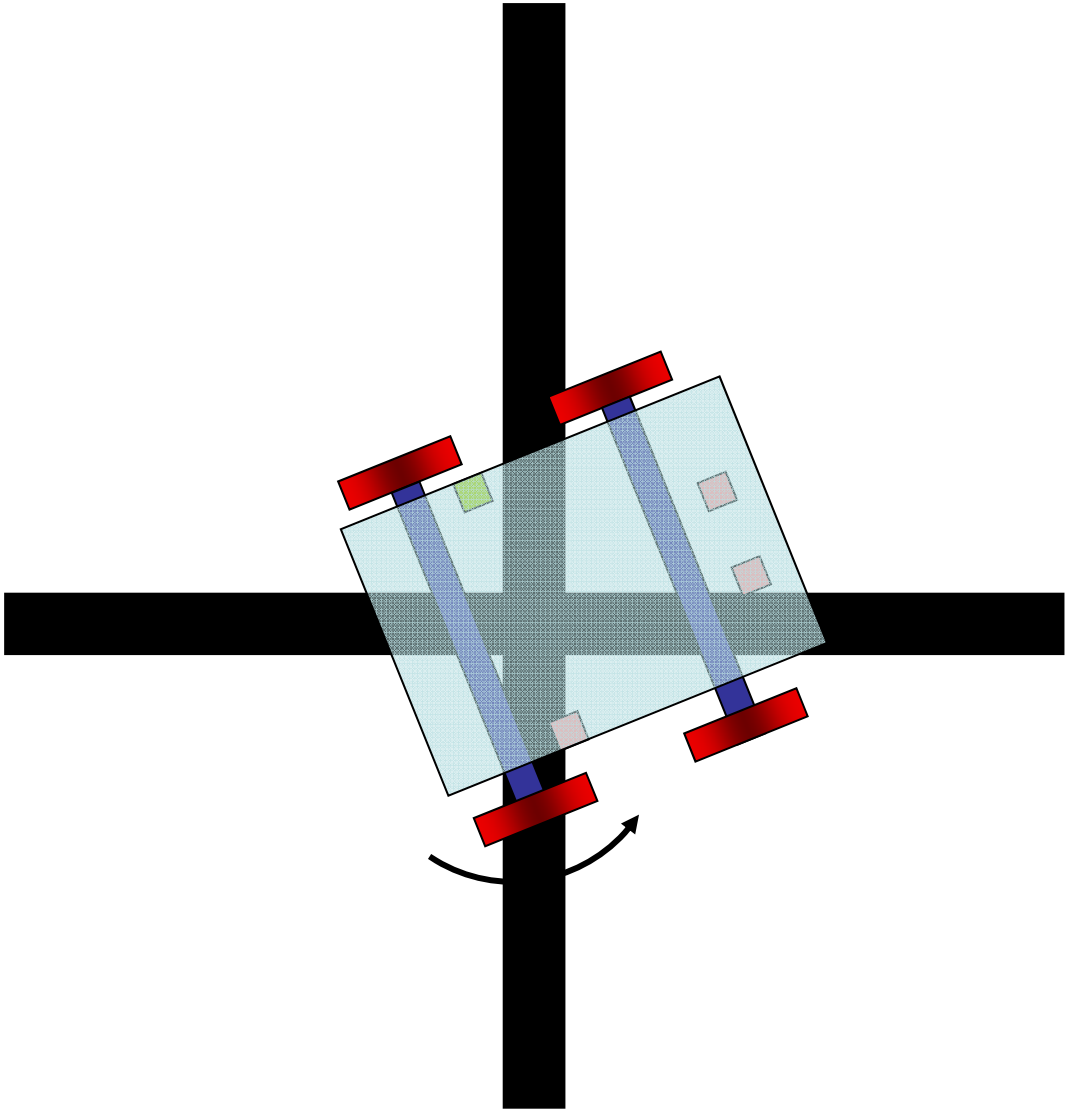


- How to position the robot into YOUR reference position
- Two counting sensor may be helpful for left & right turn

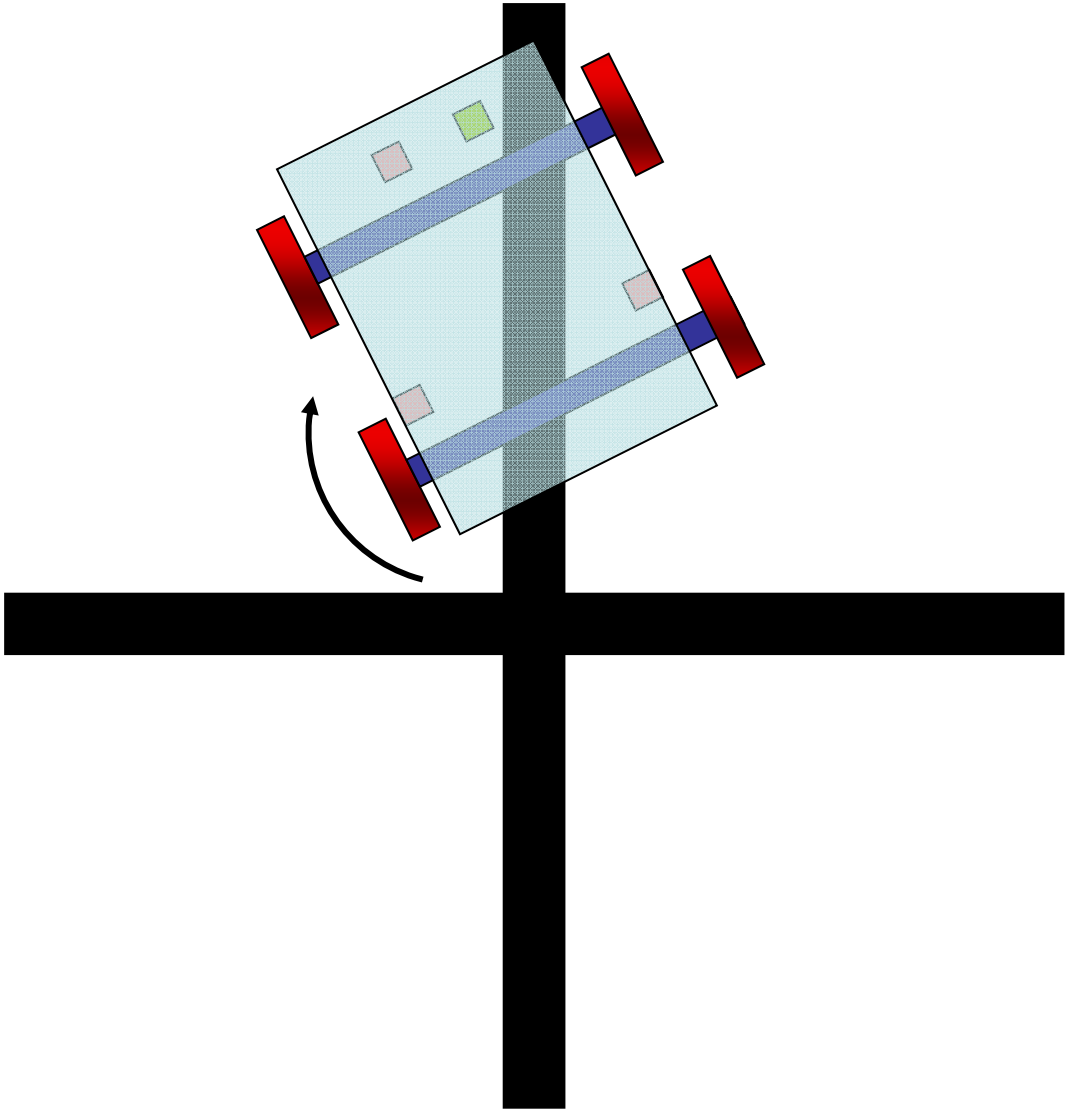
3 Step Turning



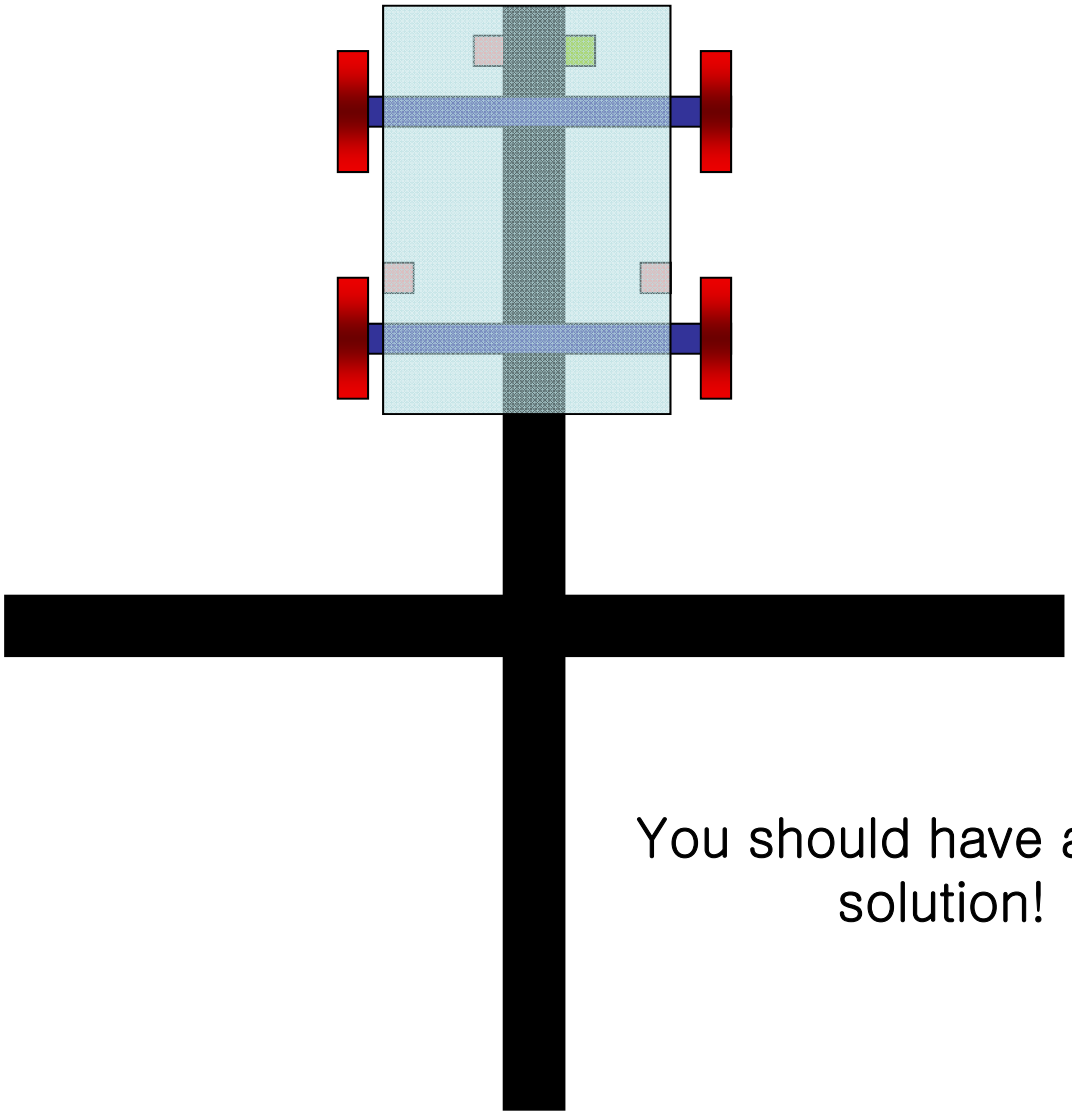
3 Step Turning



3 Step Turning

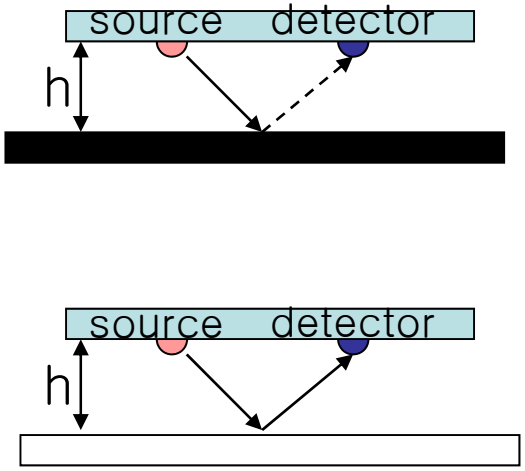


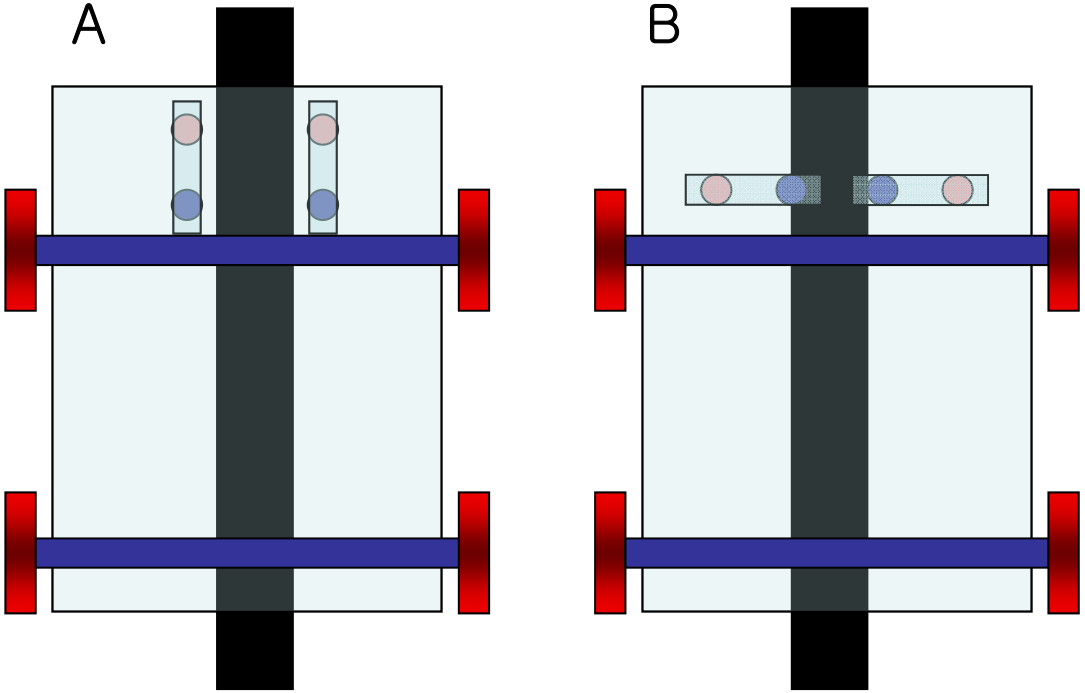
3 Step Turning



You should have a better solution!

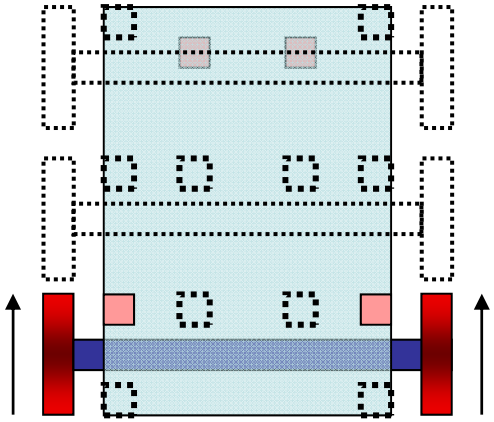
Sensor Alignment





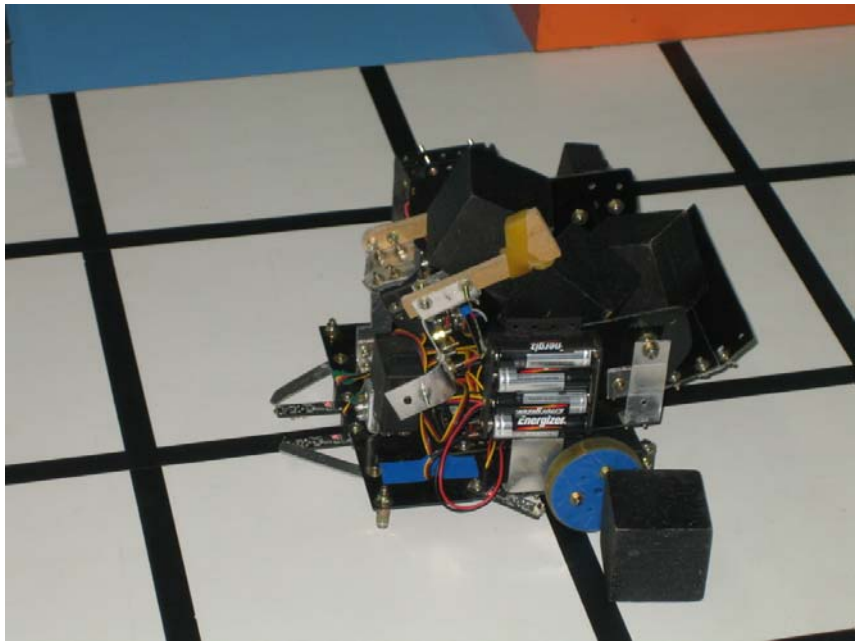
- Sensitivity decay ~ $1/h^2$
- A good physical position guarantees reliable sensing.
- Source of error: environmental illumination, floor stain...
- A or B?

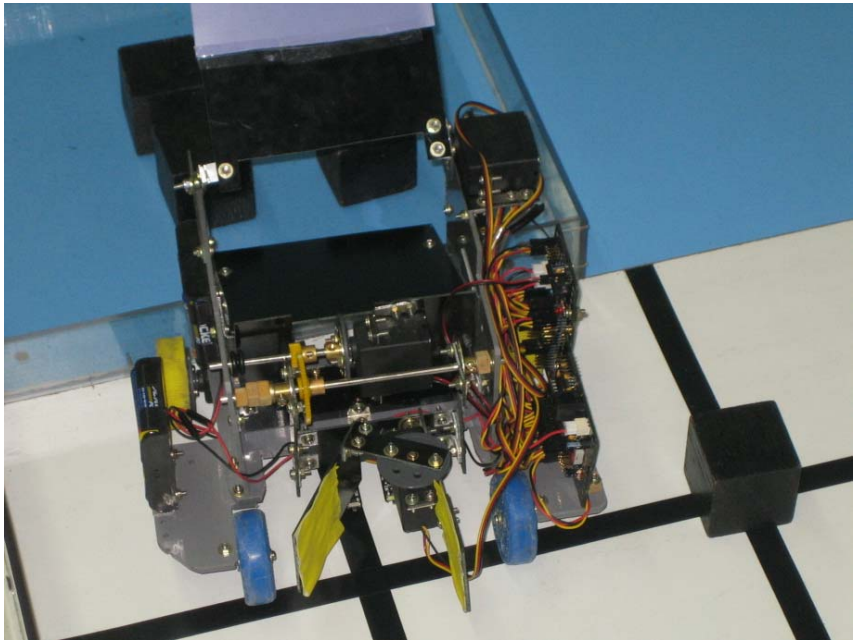
Sensor & Wheel Position



- Sensor & wheel position is critical for reliable turning, line tracing and line counting.
- You may minimize number of steps during turning via proper sensor & wheel position.

Pick-up & Dump





- You have limited number of motors
- Carefully design pick-up and dump mechanism under restrictions