Example 2.58 Suppose the following variables are allocated in the specified order. Let `r` be a pointer that points to `l`, `q` be a pointer that points to `s`, and `p` be a pointer that points to `c`. Also, let `s` be a 16-bit `short`, `l` be a 32-bit `long`, and `c` be an 8-bit `char`. Note that in this hypothetical example, pointers are 32-bit variables meaning the processor has 32-bit addressable space.

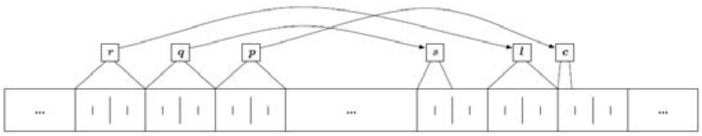


Figure 2.6: A schematic view of memory. The smallest division represents one 8-bit memory location. The height of the division line represents the byte-, short-, and long-address boundaries.

As indicated by Ex. 2.58, the amount of space reserved for any pointer is the same, no matter to what type the pointer is pointing. In the example, 32-bits are assumed for the addressable space; this is based on the architecture of the CPU. One way to make the pointer point to some variable is via the unary operator `&`, which gives the address of the label on its right-hand-side.

Example 2.59

```
p = &c;
q = &s;
r = &l;
```

Note that `&p` is the address of the pointer variable.

One way to access the contents of a variable using a pointer is via the unary operator `*`, which is called the *dereferencing* operator.

Example 2.60 At the end of these three statements, variable `c` is loaded with the value 10 via the dereference of its address in pointer `p`.

```
p = &c;
c = 0;
*p = 10;

/* now it is true that (c == 10) */
```

To declare a pointer, just add the `*` symbol to the left of the variable name.

Example 2.61

```
char *p;
short *q;
long *r;
```

This syntax is intended as a mnemonic. Using Ex. 2.61, the notation implies that the expression `*p` is a `char`, `*q` is a `short` and `*r` is a `long`. Note that as seen before, the space allocated to hold `p`, `q` and `r` is all the same (usually 32-bits on modern microprocessors), but what they point to is different. This matters to the compiler when pointer indexing is used.

The unary operators `*` and `&` have a very high precedence. However, the unary operators `++` and `--` have the same level of operator precedence. When the compiler parses a line of source code, it resolves operators with the same precedence from right-to-left. Thus, the statement `*p++;` will have a very different result compared to `(*p)++;`. The former case will increment the address stored in `p` first, then dereference the result. The latter case will read the dereferenced address first and increment the resulting value without changing the address stored in `p`. Several examples of using pointer indexing are listed in Table 2.13, assuming that `char c = 5;`, `char *p;`, and `p = &c;`. Notice that all but the final statement are equivalent.

Table 2.13: Pointer Indexing Operations

	Before			After			
Instruction	**&c = 100**	**101**	**p**	**&c = 100**	**101**	**p**	***p**
`c = *p + 1;`	5	0	100	6	0	100	6
`*p += 1;`	5	0	100	6	0	100	6
`++*p;`	5	0	100	6	0	100	6
`(*p)++;`	5	0	100	6	0	100	6
`*p++;`	5	0	100	5	0	101	0

Table 2.9: Bitwise Operators

Operator	Operation
&	AND (boolean intersection)
|	OR (boolean union)
^	XOR (boolean exclusive-or)
<<	left shift
>>	right shift
~	NOT (boolean negation, i.e., ones' complement)

Statement	c	mask	d	Embedded usefulness
`d = (c & mask);`	0x55	0x0F	0x05	Clear bits that are 0 in the mask
`d = (c | mask);`	0x55	0x0F	0x5F	Set bits that are 1 in the mask
`d = (c ^ mask);`	0x55	0x0F	0x5A	Invert bits that are 1 in the mask
`d = (c << 3);`	0x55		0xA8	Multiply by a power of 2
`d = (c >> 2);`	0x55		0x15	Divide by a power of 2
`d = ~c;`	0x55		0xAA	Invert all bits

Table 2.10: Assignment Operators

Operator	Syntax	Equivalent Operation
+=	`i += j;`	`i = (i + j);`
-=	`i -= j;`	`i = (i - j);`
*=	`i *= j;`	`i = (i * j);`
/=	`i /= j;`	`i = (i / j);`
%=	`i %= j;`	`i = (i % j);`
&=	`i &= j;`	`i = (i & j);`
|=	`i |= j;`	`i = (i | j);`
^=	`i ^= j;`	`i = (i ^ j);`
<<=	`i <<= j;`	`i = (i << j);`
>>=	`i >>= j;`	`i = (i >> j);`

CPE 301 - ATmega328P and ATmega2560 GPIO ports B, C, and D from textbook.

Bit	7	6	5	4	3	2	1	0
0x25	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- PORTB7-0: GPIO data value stored in bit *n*.

Bit	7	6	5	4	3	2	1	0
0x24	DDRB7	DDRB6	DDRB5	DDRB4	DDRB3	DDRB2	DDRB1	DDRB0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- DDRB7-0: selects the direction of pin *n*. If DDRB*n* is written '1', then PORTB*n* is configured as an output pin. If DDRB*n* is written '0', then PORTB*n* is configured as an input pin.

Bit	7	6	5	4	3	2	1	0
0x23	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0
Read/Write	R	R	R	R	R	R	R	R
Default	-	-	-	-	-	-	-	-

- PINB7-0: logic value present on external pin *n*.

Bit	7	6	5	4	3	2	1	0
0x28	-	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- PORTC6-0: GPIO data value stored in bit *n*.

Bit	7	6	5	4	3	2	1	0
0x27	-	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- DDRC6-0: selects the direction of pin *n*. If DDRC*n* is written '1', then PORTC*n* is configured as an output pin. If DDRC*n* is written '0', then PORTC*n* is configured as an input pin.

Bit	7	6	5	4	3	2	1	0
0x26	-	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0
Read/Write	R	R	R	R	R	R	R	R
Default	0	-	-	-	-	-	-	-

- PINC6-0: logic value present on external pin *n*.

Bit	7	6	5	4	3	2	1	0
0x2B	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- PORTD7-0: GPIO data value stored in bit *n*.

Bit	7	6	5	4	3	2	1	0
0x2A	DDRD7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Default	0	0	0	0	0	0	0	0

- DDRD7-0: selects the direction of pin *n*. If DDRD*n* is written '1', then PORTD*n* is configured as an output pin. If DDRD*n* is written '0', then PORTD*n* is configured as an input pin.

Bit	7	6	5	4	3	2	1	0
0x29	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0
Read/Write	R	R	R	R	R	R	R	R
Default	-	-	-	-	-	-	-	-

- PIND7-0: logic value present on external pin *n*.

APPENDIX II :

PROGRAM INSTRUCTIONS

Operation Code	*Mnemonic*	*Explanation*
1	LDAxx	Load the accumulator with the value in location xx.
2	STAxx	Store the value of the accumulator into location xx.
3	ADDxx	Add the value in location xx to the accumulator.
4	SUBxx	Subtract the value in location xx from the accumulator.
5	MULxx	Multiply the accumulator by the value in location xx.
6	DIVxx	Divide the accumulator by the value in location xx. This rounds off the answer down to the nearest one.
7	INPxx	Input to location xx.
8	OUTxx	Output from location xx.
9	JMPxx	Jump to location xx.
000	STP	Stop.
001	SKP01	Skip the next instruction if the accumulator is less than 0.
002	SKP02	Skip the next instruction if the accumulator is greater than 0.
003	SKP03	Skip the next instruction if the accumulator is 0.
004	SKP04	Skip the next instruction if the accumulator is either less than or equal to 0.
005	SKP05	Skip the next instruction if the accumulator is either greater than or equal to 0.
006	SKP06	Skip the next instruction if the accumulator is not 0.

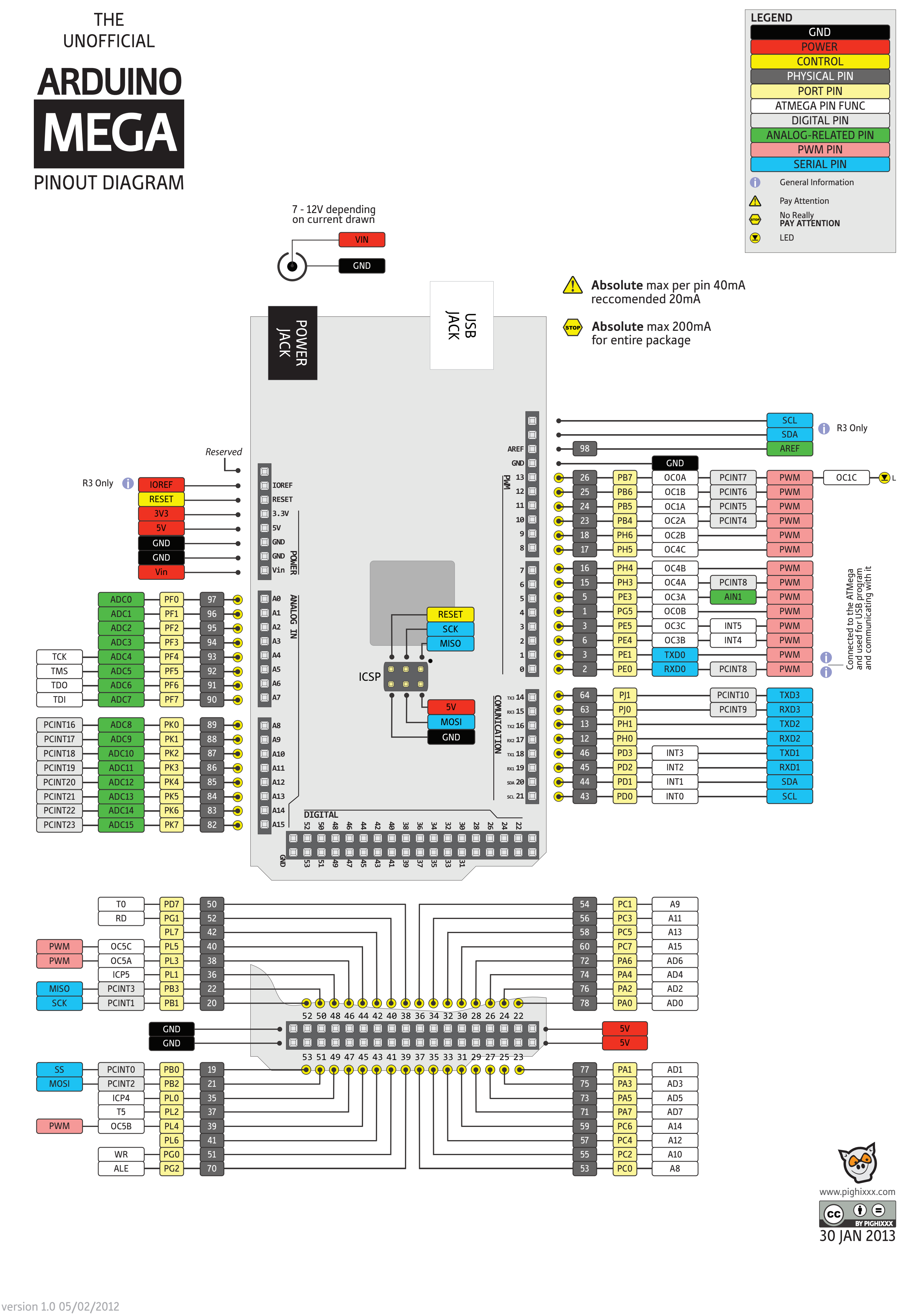
APPENDIX III : *COMMANDS*

Command	*Explanation*
RUN	Run the program starting in location 00.
RUNxx	Run the program starting in location xx.
RUNSPEDx	Run the program using a speed of x. x can be a value from 0 to 9, with 0 being the slowest and 9 the fastest.
B	Break. The computer program will stop running.
CONT	Continue at preset speed.
CONSPEDx	Continue at speed x.
CONSTP	Continue, one step at a time.
NEW	Clears memory.
LOADxx	Get ready to load information beginning at location xx.

JOYSTICK CONTROL

The joystick can be used to perform the following functions:

BREAK	Press the red button to stop a program.
SPEED	After a RUN command, move the joystick forward to increase run speed, and backward to decrease it.
STEP	Push the joystick forward after a RUNSTEP command. This does the same thing as pressing the spacebar.



THE UNOFFICIAL
ARDUINO
MEGA
PINOUT DIAGRAM
LEGEND
GND
POWER
CONTROL
PHYSICAL PIN
PORT PIN
ATMEGA PIN FUNC
DIGITAL PIN
ANALOG-RELATED PIN
PWM PIN
SERIAL PIN
General Information
Pay Attention
No Really PAY ATTENTION
LED
7 - 12V depending on current drawn
VIN
GND
POWER JACK
USB JACK
Absolute max per pin 40mA reccomended 20mA
Absolute max 200mA for entire package
R3 Only
Reserved
IOREF
RESET
3V3
5V
GND
GND
Vin
ICSP
RESET
SCK
MISO
5V
MOSI
GND
SCL
SDA
AREF
98
Connected to the ATMega and used for USB program and communicating with it
www.pighixxx.com
30 JAN 2013
version 1.0 05/02/2012