Smart Software Project

Lecture: Week 3

ATmega2560 MCU

Prof. HyungJune Lee

hyungjune.lee@ewha.ac.kr

이화여자대학교
EWHA WOMANS UNIVERSITY

Today

- Review from the last lecture
  - Embedded system
- HW #1 on Git & GitHub
  - Due: 2pm, Mon Mar 28
- Midterm: 2pm - 4:30pm, Mon Apr 11
- Collaborative programming environment
  - Source control: central vs. distributed
  - Create repository, maintain, collaborate, comment, etc.
- ATmega2560 microcontroller (MCU) architecture
  - CPU, Memory, I/O ports
- Announcement

Class Schedule

Week	Lecture Contents	Lab Contents
Week 1	Course introduction	Arduino introduction: platform & programming environment
Week 2	Embedded system overview & source management in collaborative repository (using GitHub)	Lab 1: Arduino Mega 2560 board & SmartCAR platform
Week 3	ATmega2560 Micro-controller (MCU): architecture & I/O ports, Analog vs. Digital, Pulse Width Modulation	Lab 2: SmartCAR LED control
Week 4	Analog vs. Digital & Pulse Width Modulation	Lab 3: SmartCAR motor control (Due: HW on creating project repository using GitHub)
Week 5	ATmega2560 MCU: memory, I/O ports, UART	Lab 4: SmartCAR control via Android Bluetooth
Week 6	ATmega2560 UART control & Bluetooth communication between Arduino platform and Android device	Lab 5: SmartCAR control through your own customized Android app (Due: Project proposal)
Week 7	Midterm exam	
Week 8	ATmega2560 Timer, Interrupts & Ultrasonic sensors	Lab 6: SmartCAR ultrasonic sensing
Week 9	Infrared sensors & Buzzer	Lab 7: SmartCAR infrared sensing
Week 10	Acquiring location information from Android device & line tracing	Lab 8: Implementation of line tracer
Week 11	Gyroscope, accelerometer, and compass sensors	Lab 9: Using gyroscope, accelerometer, and compass sensors
Week 12	Project	Team meeting (for progress check)
Week 13	Project	Team meeting (for progress check)
Week 14	Course wrap-up & next steps	
Week 15	Project presentation & demo I (Due: source code, presentation slides, & poster slide)	Project presentation & demo II
Week 16	Final week (no final exam)	

Today

- **Review from the last lecture**
  - **Embedded system**

- Collaborative programming environment
  - Source control: central vs. distributed
  - Create repository, maintain, collaborate, comment, etc.

- ATmega2560 microcontroller (MCU) architecture
  - CPU, Memory, I/O ports

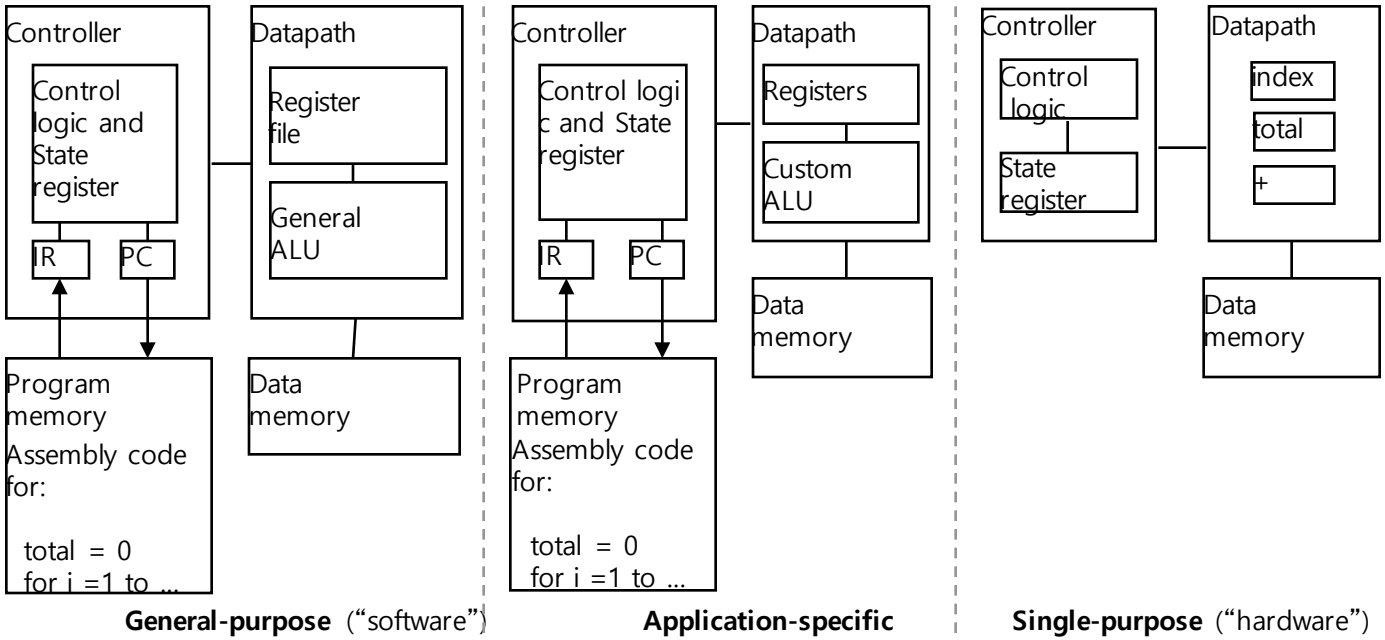
- Announcement

Some common characteristics of embedded systems

- Single-functioned
  - Executes a single program, repeatedly
- Tightly-constrained
  - Low cost, low power, small, fast, etc.
- Reactive and real-time
  - Continually reacts to changes in the system's environment
  - Must compute certain results in real-time witho ut delay

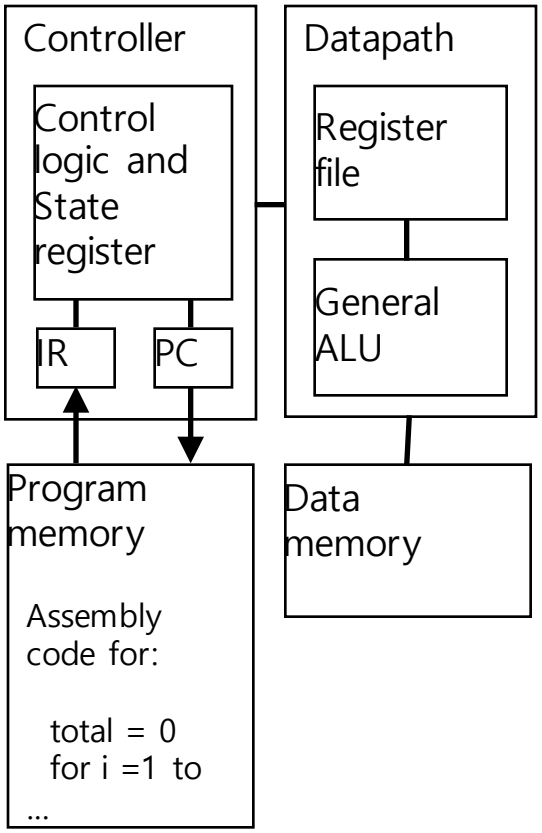
Processor technology

- The architecture of the computation engine used to im plement a system's desired functionality
- Processor does not have to be programmable
  - "Processor" *not* equal to general-purpose processor



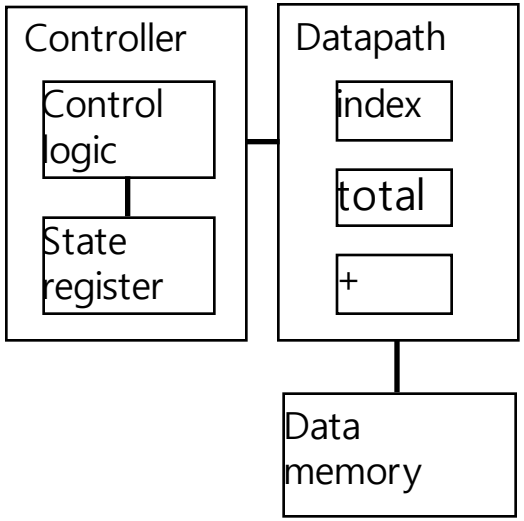
General-purpose processors

- Programmable device used in a variety of applications
  - Also known as "microprocessor"
- Features
  - Program memory
  - General datapath with large register file and general ALU
- User benefits
  - Low time-to-market and NRE costs
  - High flexibility
- "Intel Core i7" the most latest, but there are hundreds of others



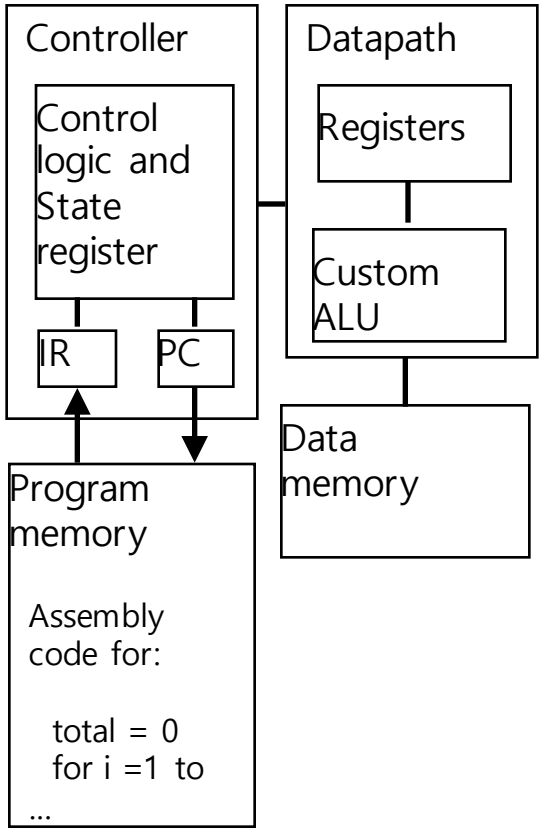
Single-purpose processors

- Digital circuit designed to execute exactly one program
  - a.k.a. coprocessor, accelerator or peripheral
- Features
  - Contains only the components needed to execute a single program
  - No program memory
- Benefits
  - Fast
  - Low power
  - Small size



Application-specific processors

- Programmable processor optimized for a particular class of applications having common characteristics
  - Compromise between general-purpose and single-purpose processors
- Features
  - Program memory
  - Optimized datapath
  - Special functional units
- Benefits
  - Some flexibility, good performance, size and power
- Example
  - Qualcomm Snapdragon AP
  - Nvidia Graphics AP



Today

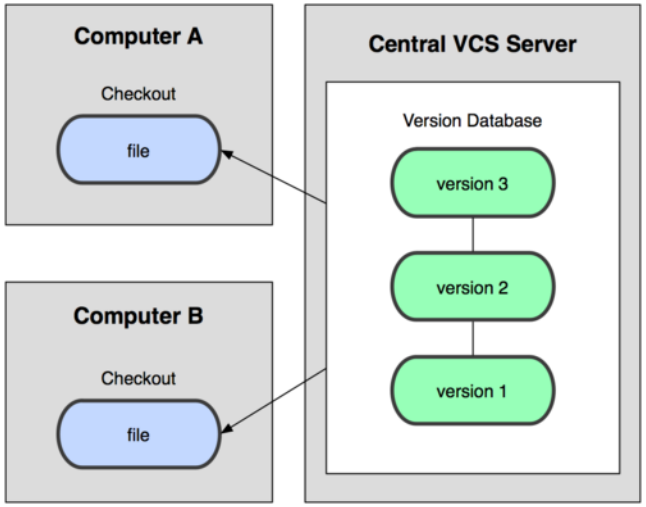
- Review from the last lecture
  - Embedded system
- **Collaborative programming environment**
  - Source control: central vs. distributed
  - Create repository, maintain, collaborate, comment, etc.
- ATmega2560 microcontroller (MCU) architecture
  - CPU, Memory, I/O ports
- Announcement

Why version control?

- For working by yourself:
  - Gives you a "time machine" for going back to earlier versions
  - Gives you great support for different versions (standalone, web app, etc.) of the same basic project
- For working with others:
  - Greatly simplifies concurrent work, merging changes
- For getting an internship or job:
  - Any company with a clue uses some kind of version control
  - Companies without a clue are bad places to work

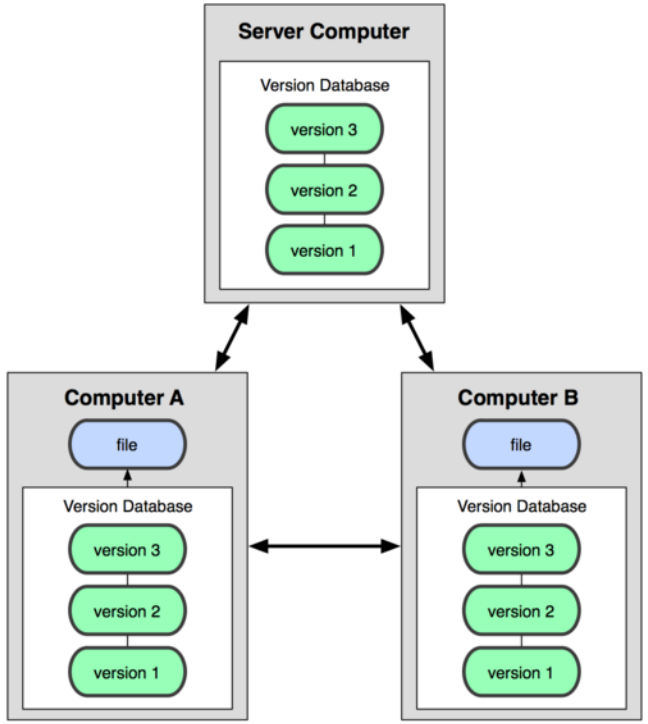
Central vs. Distributed Version Control System (VCS)

<Centralized Model>



Ex: CVS, Subversion(SVN), Perforce

<Distributed Model>



Ex: Git, Mercurial

Result: Many operations are local

What is Git?

- Formal version control system
- Developed by Linus Torvalds (creator of Linux)
  - Used to manage the source code for Linux

- Tracks any content (but mostly plain text files)
  - Source code
  - Data analysis projects
  - Manuscripts
  - Websites
  - Presentations

Why use Git?

- Git is better than CVS, SVN (and most VCSs)
  - It's fast
  - Don't need access to a server
- Distributed
- Branches
- Always in a working state
- Free

What is GitHub?

- Provide Git repositories for free
  - http://github.com
- Graphical user interface for Git
  - Exploring code and its history
  - Tracking issues
- Real open source
  - Immediate, easy access to the code
- Facilitates:
  - Learning from others
  - Seeing what people are up to
  - Contributing to others' code
- Lowers the barrier to collaboration
  - "There is a typo in your xxx.doc" vs.
    "Here is a correction that I modified for your xxx.doc"

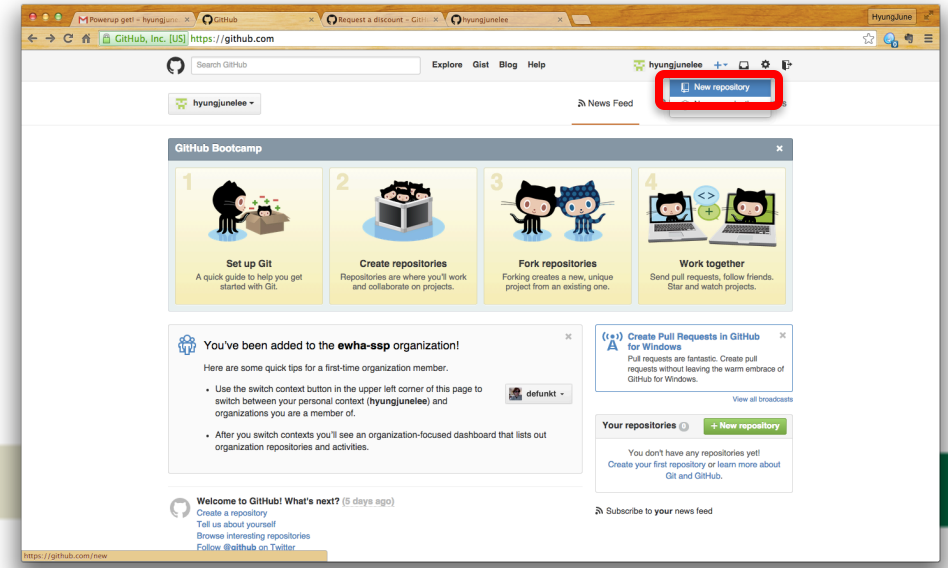
Getting started with GitHub

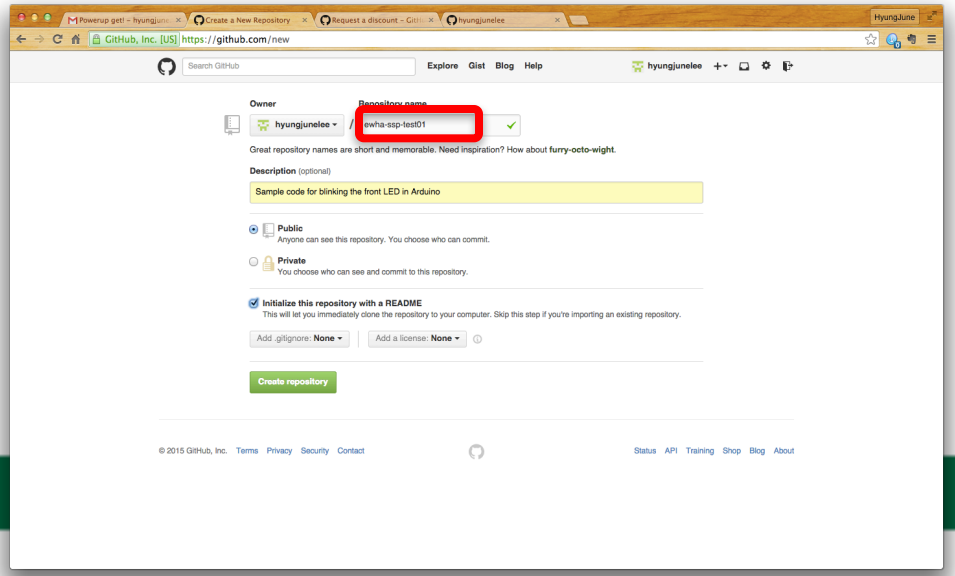
- Step 1: Get an account
  - http://github.com
- Step 2: Create a GitHub repository
  - Click the "Create a new repo" button
  - Give it a name and description
  - Click the "Create repository" button
- Step 3: Work in your computer
  - 1) Command line after installation of Git utility, or
    - Windows: http://msysgit.github.io/ (Git BASH)
    - Mac OS: no installation needed
  - 2) GitHub client for Windows or Mac
    - Windows: https://windows.github.com/
    - Mac OS: https://mac.github.com/

Q1) How to create a repository?

- Case 1: starting from nothing
  - There is no files yet
  - Just want to create a null repository
  - Will add files later on

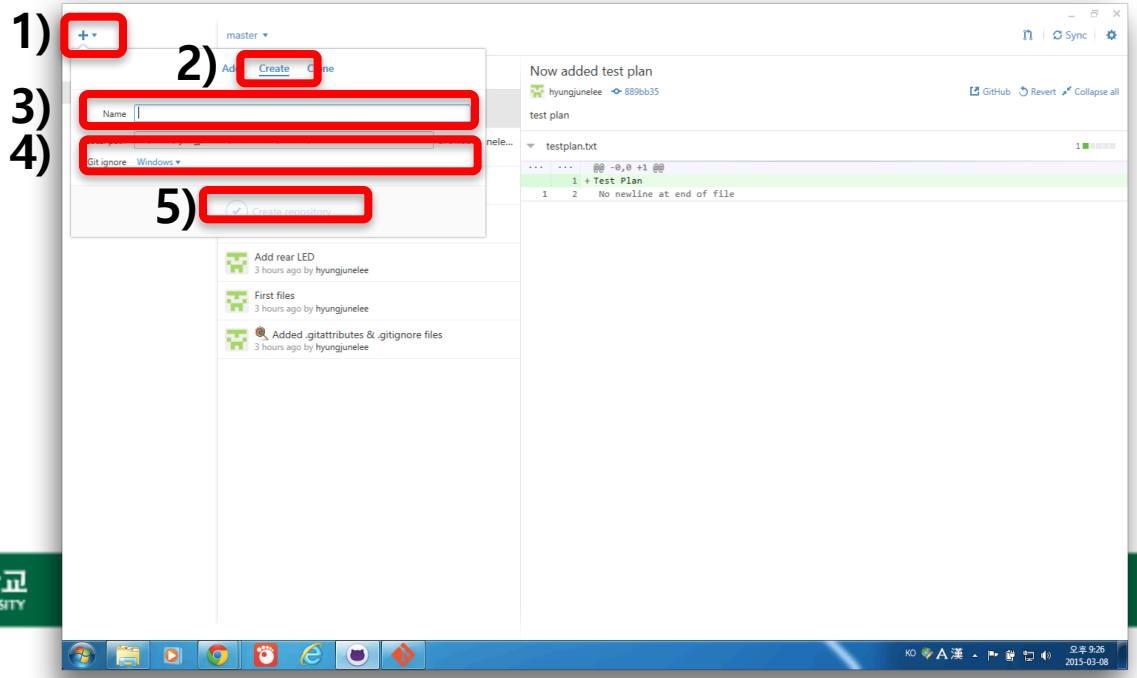
  => Create a repository directly at http://github.com
  - Step: 1) Select "new repository" ➔ 2) enter a repository name ➔3) click "create"





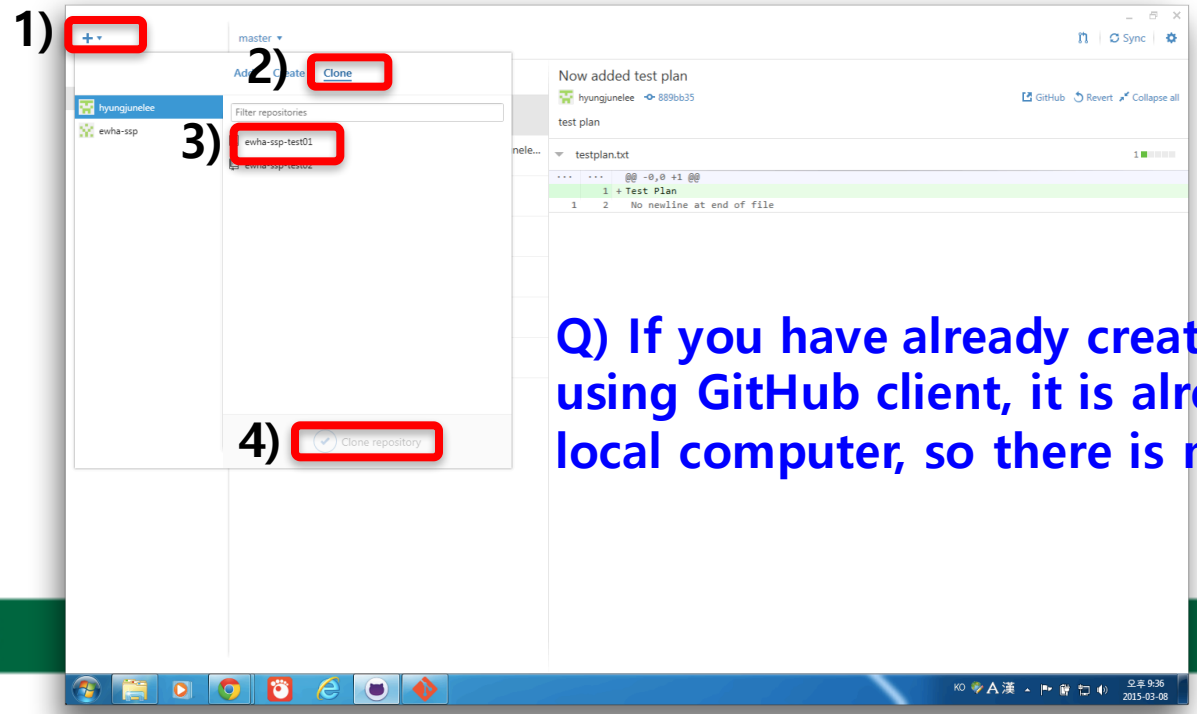
Q1) How to create a repository?

- Case 2: Some files already in your local computer
  - Want to create a repository with initial files
  - Create a repository using GitHub client
  - Step: 1) Click "+" ➔ 2) Click "Create" ➔3) enter a repository name ➔ 4) "Browse" the path for the files ➔5) "Create"



Q2) How to get a repository at your local computer from GitHub

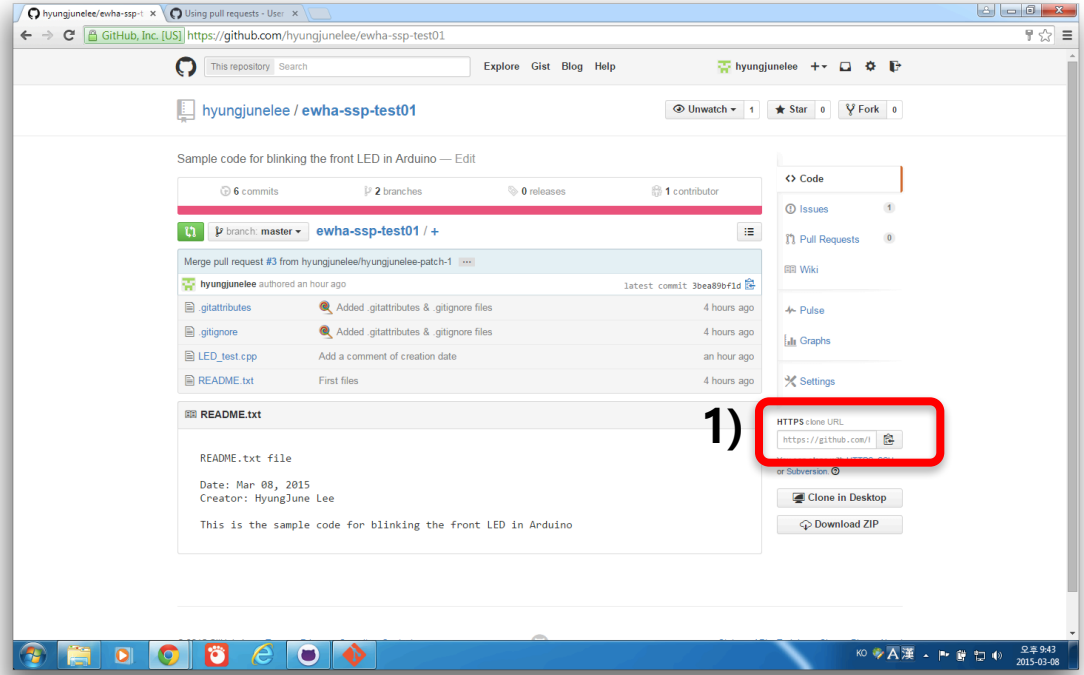
- Case 1: using GitHub client
  - Step: 1) Click "+" ➔ 2) Click "Clone" ➔3) select a repository that is already created at GitHub ➔4) Click "Clone repository" ➔5) See files



Q) If you have already created a repository using GitHub client, it is already in your local computer, so there is nothing to do!

Q2) How to get a repository at your local computer from GitHub

- Case 2: using command line
  - Step: 1) Copy your GitHub URL ➔
    2) At terminal: > git clone [GitHub URL] ➔3) See files



2)

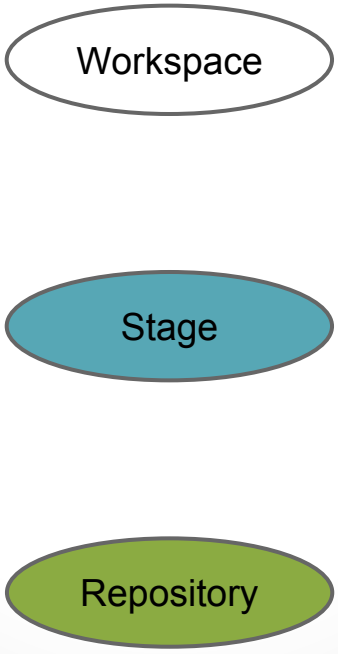
```
git-repo — bash — 93×32
Hyungjunes-MacBook-Air:git-repo hyungjunelee$ git clone https://github.com/hyungjunelee/ewha-ssp-test01.git
```

Terminal Output

```
HyungJunes-MacBook-Air:git-repo hyungjunelee$ git clone https://github.com/hyungjunelee/ewha-ssp-test01.git
Cloning into 'ewha-ssp-test01'...
remote: Counting objects: 11, done.
remote: Compressing objects: 100% (8/8), done.
remote: Total 11 (delta 3), reused 11 (delta 3), pack-reused 0
Unpacking objects: 100% (11/11), done.
HyungJunes-MacBook-Air:git-repo hyungjunelee$ ls
ewha-ssp-test01
HyungJunes-MacBook-Air:git-repo hyungjunelee$ cd ewha-ssp-test01/
HyungJunes-MacBook-Air:ewha-ssp-test01 hyungjunelee$ ls
LED_test.cpp        README.txt
```

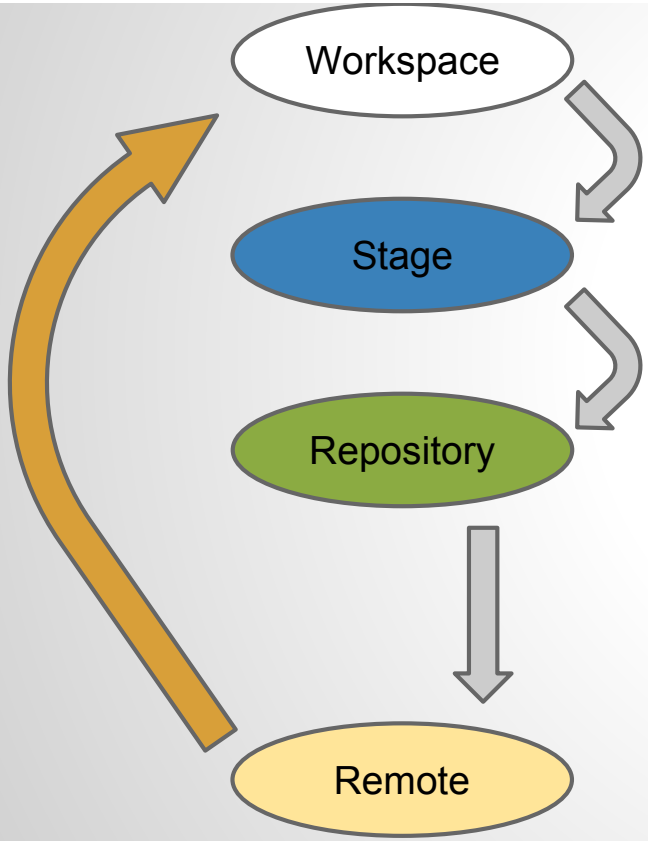
Q3) How to maintain?

Case 1: command line using Git



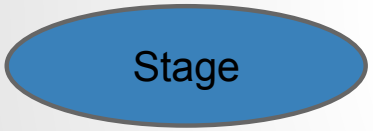
- 1) Pull changes from your collaborator
  - Before you start working
    > **git pull**
- 2) Change some files (in your "workspace")
- 3) See what you have changed
    > **git status**
    > **git diff**
    > **git log**
- 4) Indicate what changes to save (in your "stage")
- 5) Commit those changes into your local "repository"
- 6) Push the changes to your remote original "GitHub" repository
    > **git push**

git pull



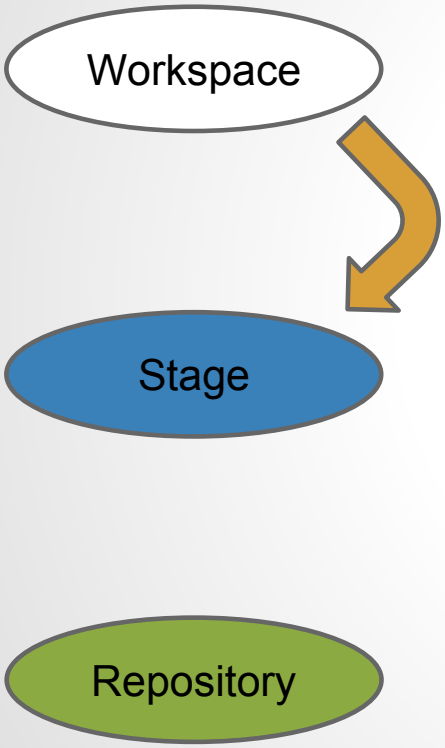

```
[~/Test]$ git pull
remote: Counting objects: 3, done
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (3/3), done.
From github.com:user/Test
   c3b8ef6..ea5dda8  master         -> origin/master
Updating c3b8ef6..ea5dda8
Fast-forward
 hello.js | 6 ++++--
 1 file changed, 4 insertions(+), 1 deletion(-)
[~/Test]$
```

git status




```
[~/Test]$ echo “console.log(‘hello world’);” > hello.js


[~/Test]$ git status
On branch master

Initial commit

Untracked files:
   (use “git add <file>...” to include in what will be committed)

        hello.js

nothing added to commit but untracked files present
[~/Test]$
```

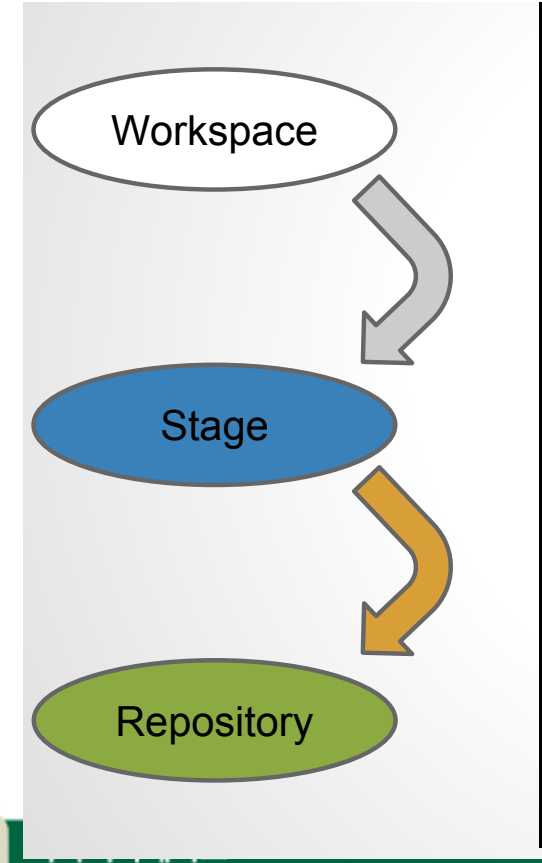
git add




```
[~/Test]$ git add .


[~/Test]$ git status
On branch master

Initial commit

Changes to be committed:
   (use “git rm --cached <file>...” to unstage)

        new file:    hello.js
[~/Test]$
```

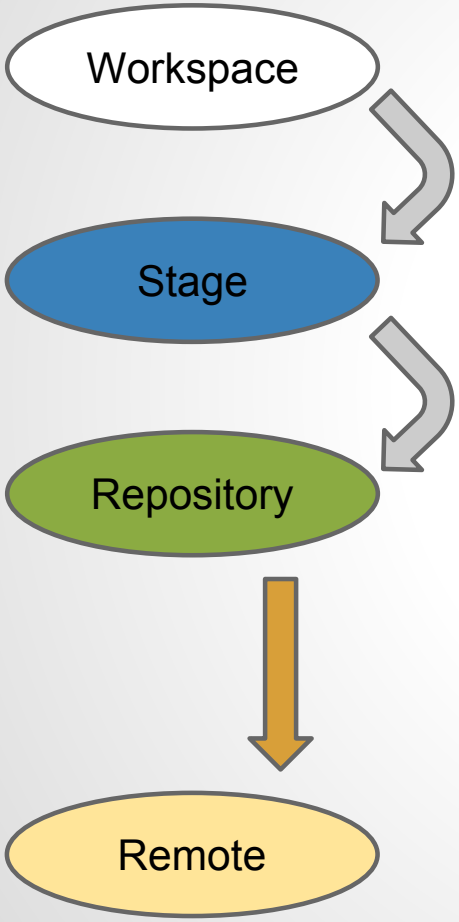
git commit

- Use –m “comment on what’s going on”




```
[~/Test]$ git commit -m “Started hello world program”
[master (root-commit) c3b8ef6] Started hello world program
 1 file changed, 1 insertion(+)
 create mode 100644 hello.js


[~/Test]$ git status
On branch master
nothing to commit, working directory clean
[~/Test]$
```

git push

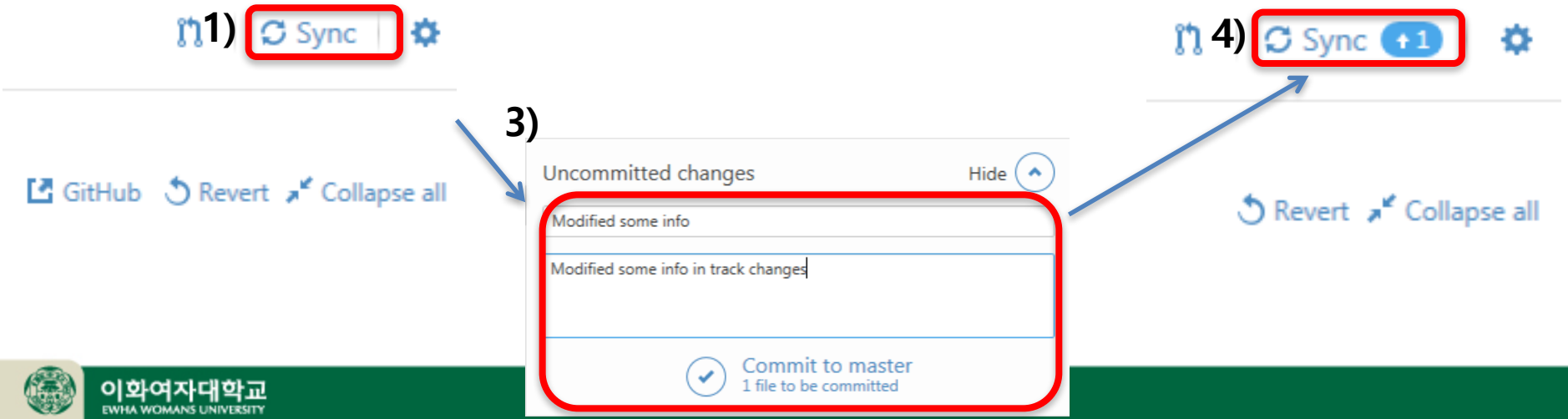



```
[~/Test]$ git push
Counting objects: 1, done.
Delta compression using up to 4 threads.
Compressing objects: 100% (1/1), done.
Writing objects: 100% (1/1), 28 bytes | 0 bytes/s, done.
Total 1 (delta 1), reused 0 (delta 0)
To git@github.com:user/Test.git
   (root-commit)..c3b8ef6    master -> master
[~/Test]$
```

Q3) How to maintain?

Case 2: using GitHub client

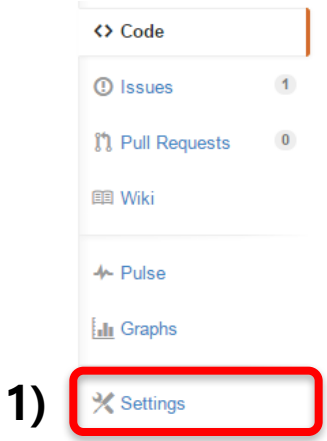
- 1) Click sync to pull (corresponding to "git pull")
  - Before you start working
- 2) Change some files (in your local computer)
- 3) Commit those changes (in your local computer)
- 4) Click sync to push (corresponding to "git push")

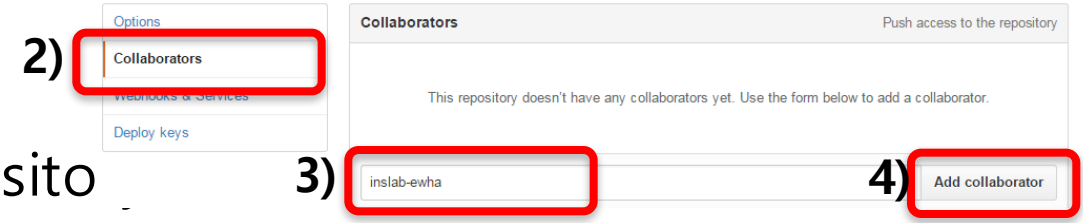


Q4) How to collaborate with other people?

Case 1: Administrator

- 1) Click "Settings" in the right panel
- 2) Click "Collaborators"
- 3) Enter a username to collaborate
- 4) Click "Add collaborator"
  - This means you give the "push" privilege to this person



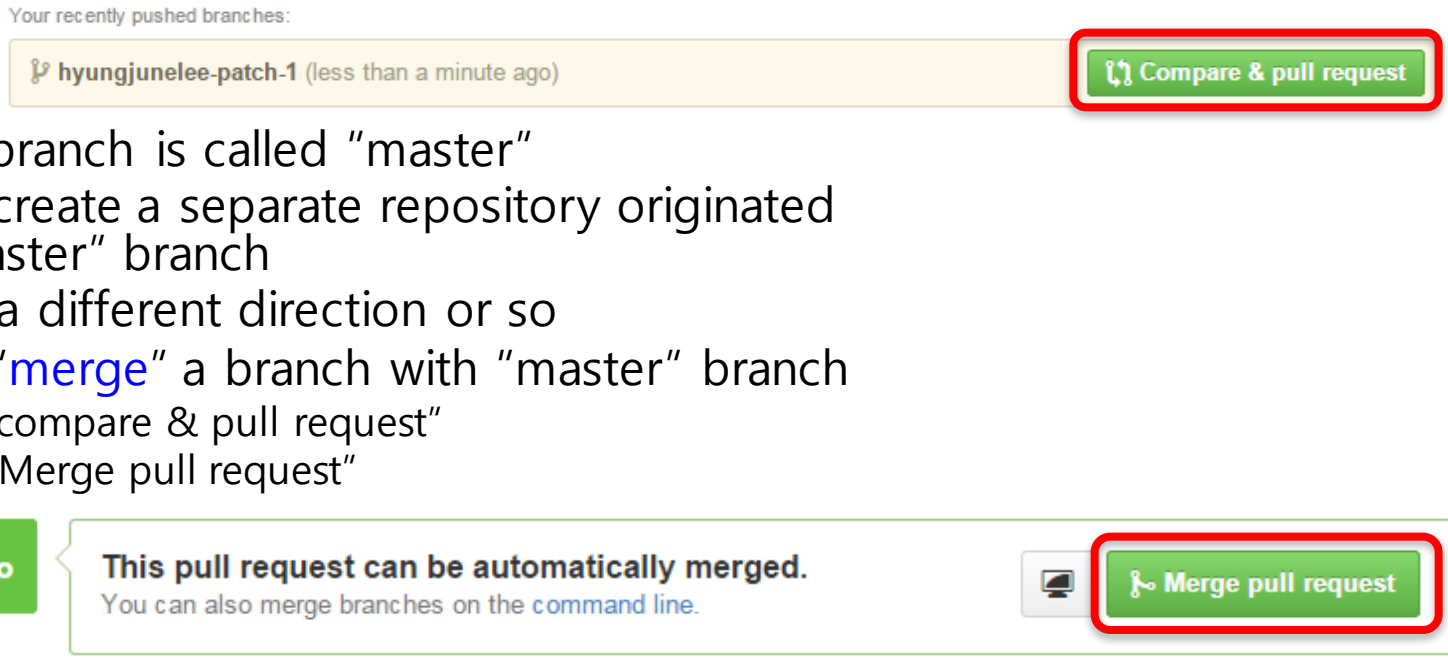


Case 2: Collaborator

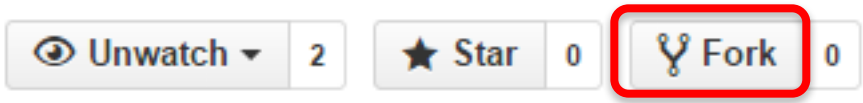
- 1) Visit the original reposito
- 2) Copy the URL
- 3) Clone it in your local computer
- 4) Work on the files
- 5) Push to the original repository

Q5) How to create a separate repository for your own purpose?

- Branch vs. Fork
- Branch
  - Original branch is called "master"
  - You can create a separate repository originated from "master" branch
  - Develop a different direction or so
  - You can "merge" a branch with "master" branch
    - Click "compare & pull request"
    - Click "Merge pull request"

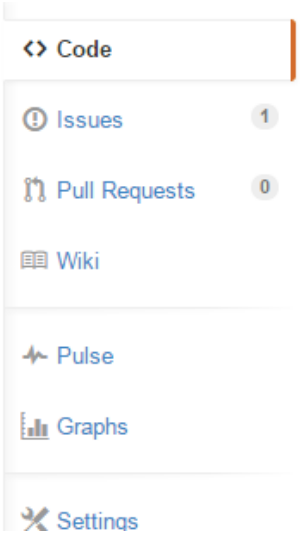


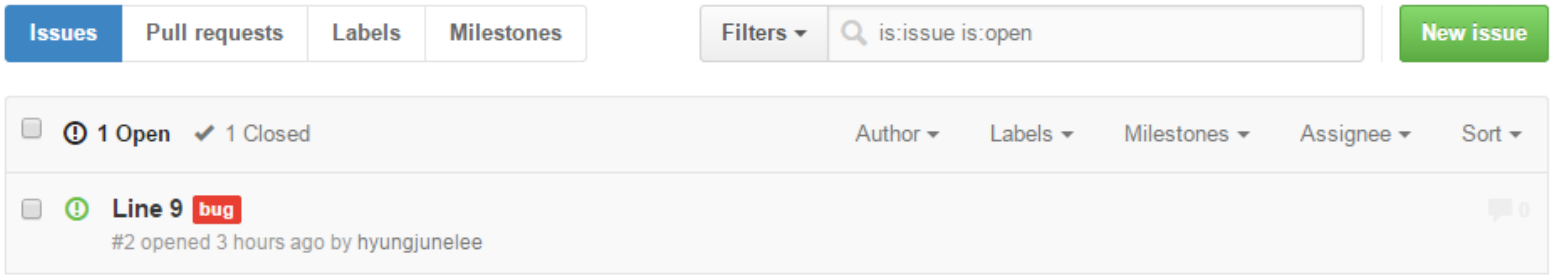
- Fork
  - You are not a collaborator, but
  - You want a personal copy of someone else's project
  - Visit a repository
  - Click "fork"



Q6) Leave some high-level comments?

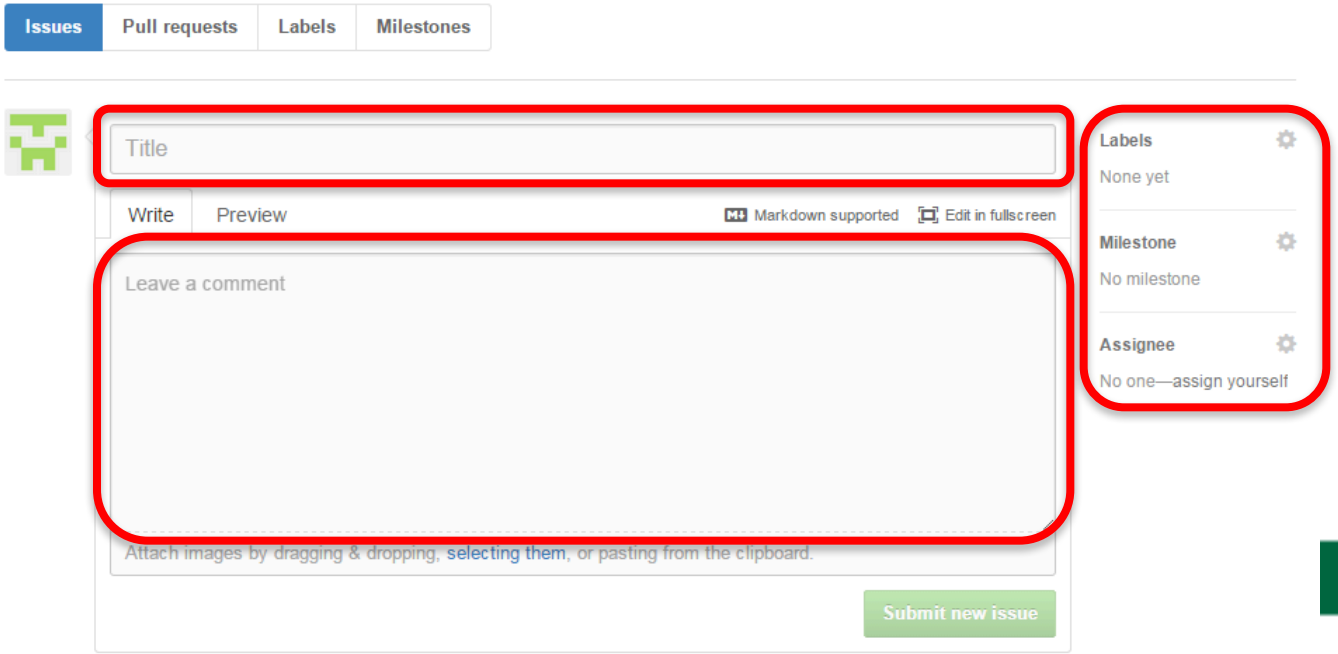
- Create an "issue"
  - Click "Issues"
  - Click "New Issue"





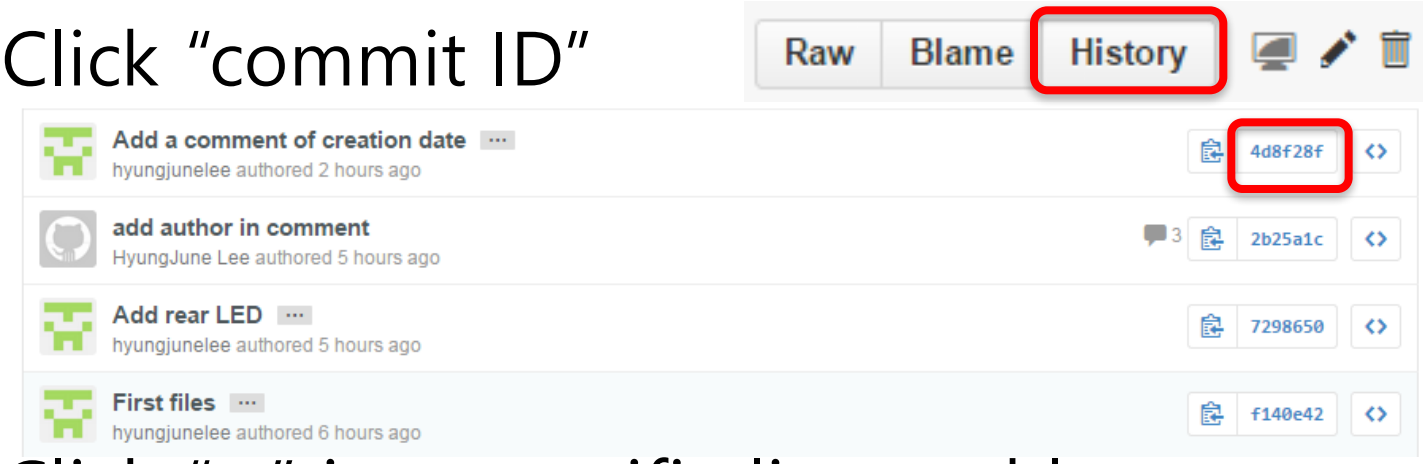
Q6) Leave some high-level comments?

- Leave comments
- Select an appropriate Label
  - Ex) bug, question, etc.
- Select Milestone
- Select Assignee: who should resolve this issue?



Q7) Leave some line-by-line comments ?

- Select a file
- Select “History”
- Click “commit ID”



- Click “+” in a specific line and leave a comment

```
2   #include "LED_test.h"
3
4   #define FRONT_LED_PIN  10
5  +#define REAR_LED_PIN   9       //Add a new code
6
7   //The setup function is called once at startup of the sketch
8   void setup()
9   {
10          pinMode(FRONT_LED_PIN, OUTPUT);
11  +       pinMode(REAR_LED_PIN, OUTPUT); //Add a new code
12  +
```

Q8) How to move to a commit or a branch using command line Git

- To a previous commit
  - > **git checkout commit-ID**
- To a certain branch
  - > **git checkout branch-name**
- To the latest revision
  - > **git checkout master**

Q9)  git vs. GitHub 

Question: Do I have to use Github to use Git?

Answer: No!

- You can use Git completely locally for your own purposes, or

- You or someone else could set up a server to share files, or

More Resources

- For a quick interactive Git tutorial
  - http://try.github.io/
- Git Cheat Sheet
  - https://training.github.com/kit/downloads/github-git-cheat-sheet.pdf
- Many Git & GitHub Learning Materials
  - https://help.github.com/articles/good-resources-for-learning-git-and-github

Today

- Review from the last lecture
  - Embedded system
- Collaborative programming environment
  - Source control: central vs. distributed
  - Create repository, maintain, collaborate, comment, etc.
- **ATmega2560 microcontroller (MCU) architecture**
  - CPU, Memory, I/O ports
- Announcement

Microprocessors

A **Microprocessor** is a processor implemented on (usually) one integrated circuit.

A processor requires:
- memory for program and data storage
- support logic
- Input and Output ports

Data is read from and written to memory
Instructions are read from (flash) memory

Microprocessor vs. Microcontroller

- Microprocessor
  - Single chip semi-conductor device
  - But not a complete computer
  - Contains ALU, PC, timing & control units, registers
- Microcontroller
  - Functional computer **system-on-chip**
  - Contains microprocessor, memory, and program mable input/output peripherals
    - RAM, EEPROM
    - Timer, parallel I/O
    - ADC, DAC

What is ATmega2560?

- One of AVR 8-bit RISC **microcontrollers** by Atmel
- The acronym **AVR** has been reported to stand for
  - Advanced Virtual RISC and also for the chip's designers: Alf-Egil Bogen and Vegard Wollan who designed the basic architecture at the Norwegian Institute of Technology

- RISC stands for **reduced instruction set computer**

  : CPU design with a reduced instruction set as well as a simpler set of instructions (like for example PIC and AVR)

A little history

- The PIC (Programmable Interrupt Controller) appeared around 1980.
  → 8 bit bus
  → executes 1 instruction in 4 clk cycles
  → Harvard architecture

- AVR (1994)
  → 8 bit bus
  → one instruction per cycle
  → Harvard architecture

RISC vs. CISC

- RISC - Reduced Instruction Set Computer
  - The instruction set is small, and most instructions complete in one cycle (100 or less instruction types, smaller range of addressing modes).
  - Multiply & Divide performed using add/subtract & shift

RISC vs. CISC

- CISC - Complex Instruction Set Computer
  - The instruction set is large, and offers great variety of instructions (100 or more instruction types, many addressing modes).
  - Few instructions complete in one cycle
  - Typically includes multiply & divide operations that m ay take many cycles to complete.

Von Neumann vs. Harvard

- Von Neumann Architecture
  - The computer follows a step-by-step program that governs its operation.
  - The program is stored as data
  - No distinction between data and instructions.

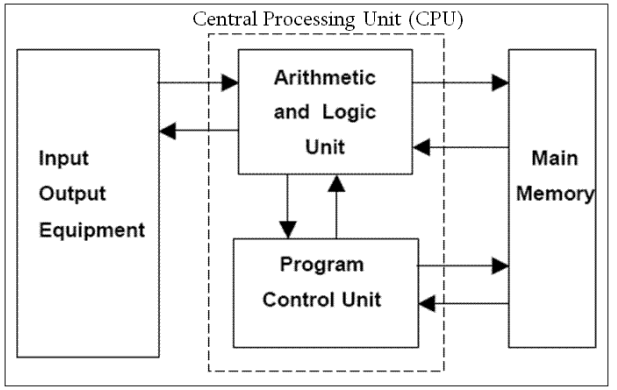
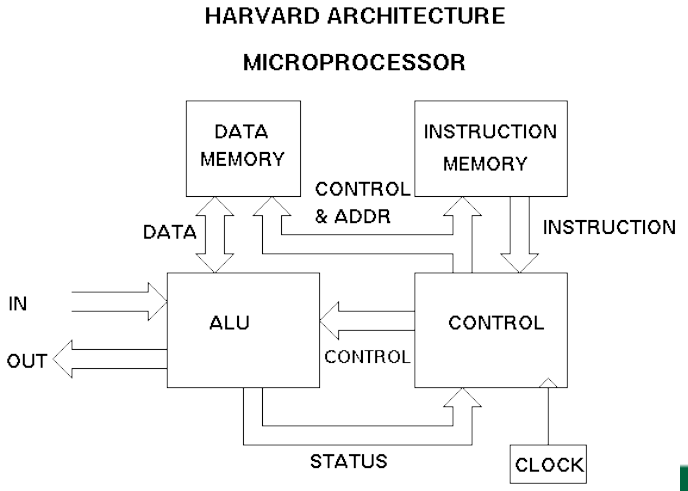


Figure : General structure of Von Neumann Architecture

- Harvard Architecture
  - Separate data and address spaces
  - The program is stored in its own address space



AVR 8-Bit RISC High Performance

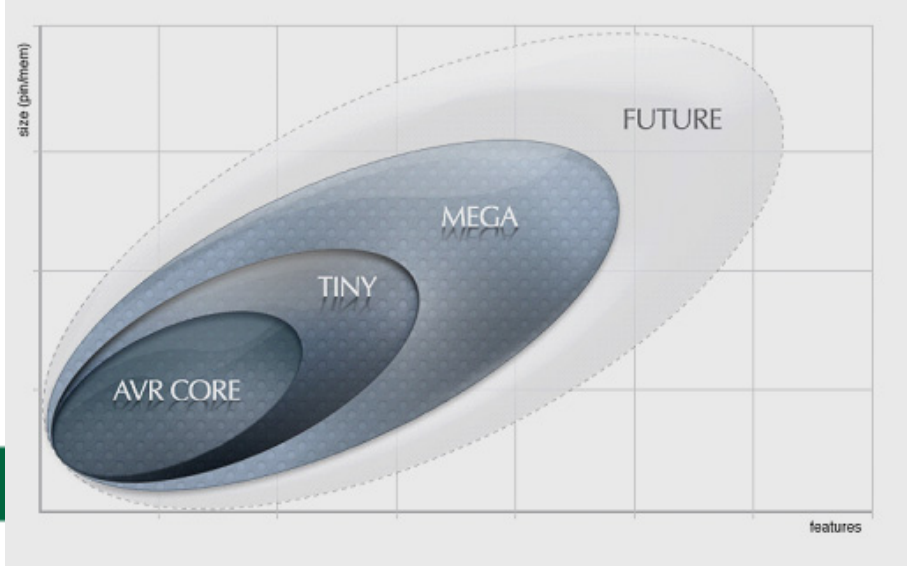
- True single cycle execution
  - single-clock-cycle-per-instruction execution
  - PIC microcontrollers take 4 clock cycles per instruction
- One MIPS (mega instructions per second) per MHz
  - up to 20 MHz clock
- 32 x 8 bit general purpose registers
  - provide flexibility and performance when using high level languages
  - prevents access to RAM
- Harvard architecture
  - separate bus for program and data memory

AVR 8-Bit RISC Low Power Consumption

- 1.8 to 5.5V operation
  - will use all the energy stored in your batteries
- A variety of sleep modes
  - AVR Flash microcontrollers have up to six different sleep modes
  - fast wake-up from sleep modes
- Software controlled frequency

AVR 8-Bit RISC Compatibility

- AVR® Flash microcontrollers share a sing le
  core architecture
  - use the same code for all families
  - 1 Kbytes to 256 Kbytes of code
  - 8 to 100 pins
  - all devices have
    Internal oscillators

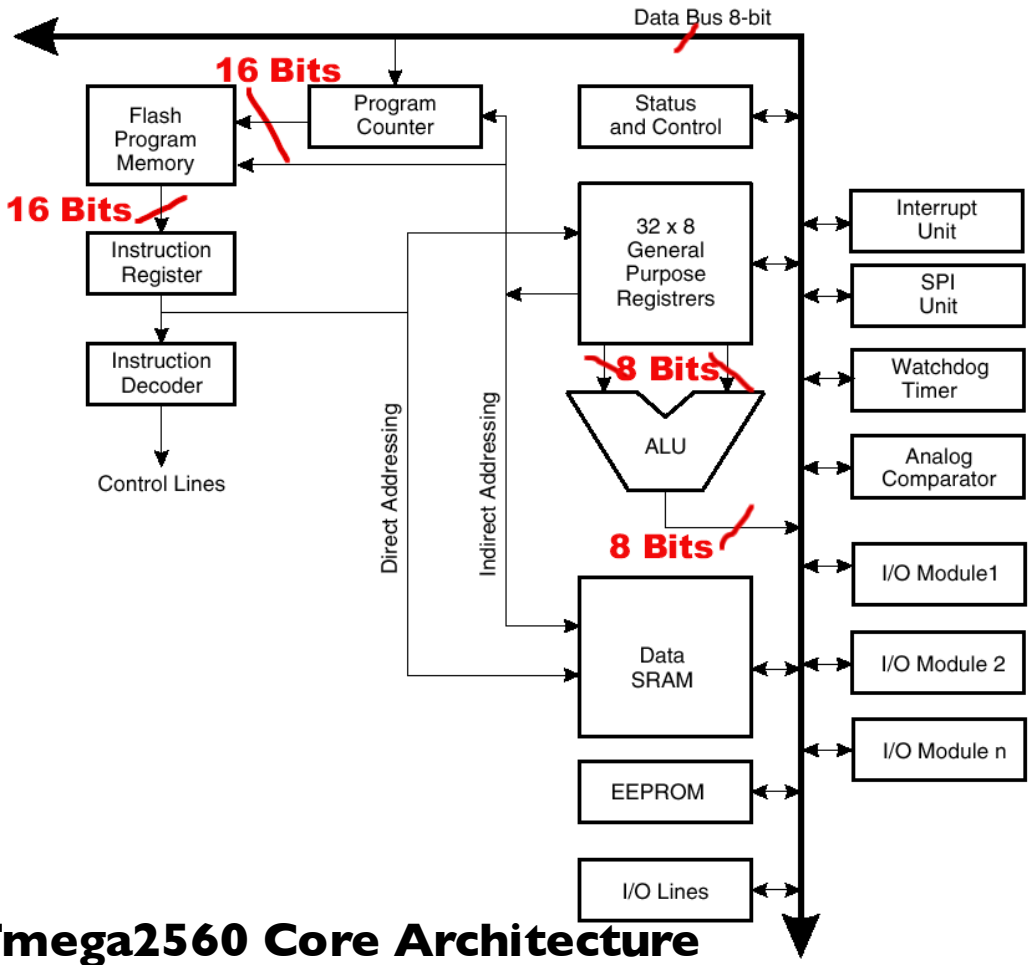


ATmega2560 CPU Architecture

Figure 3. Block Diagram of the AVR Architecture

Mega2560 CPU Core

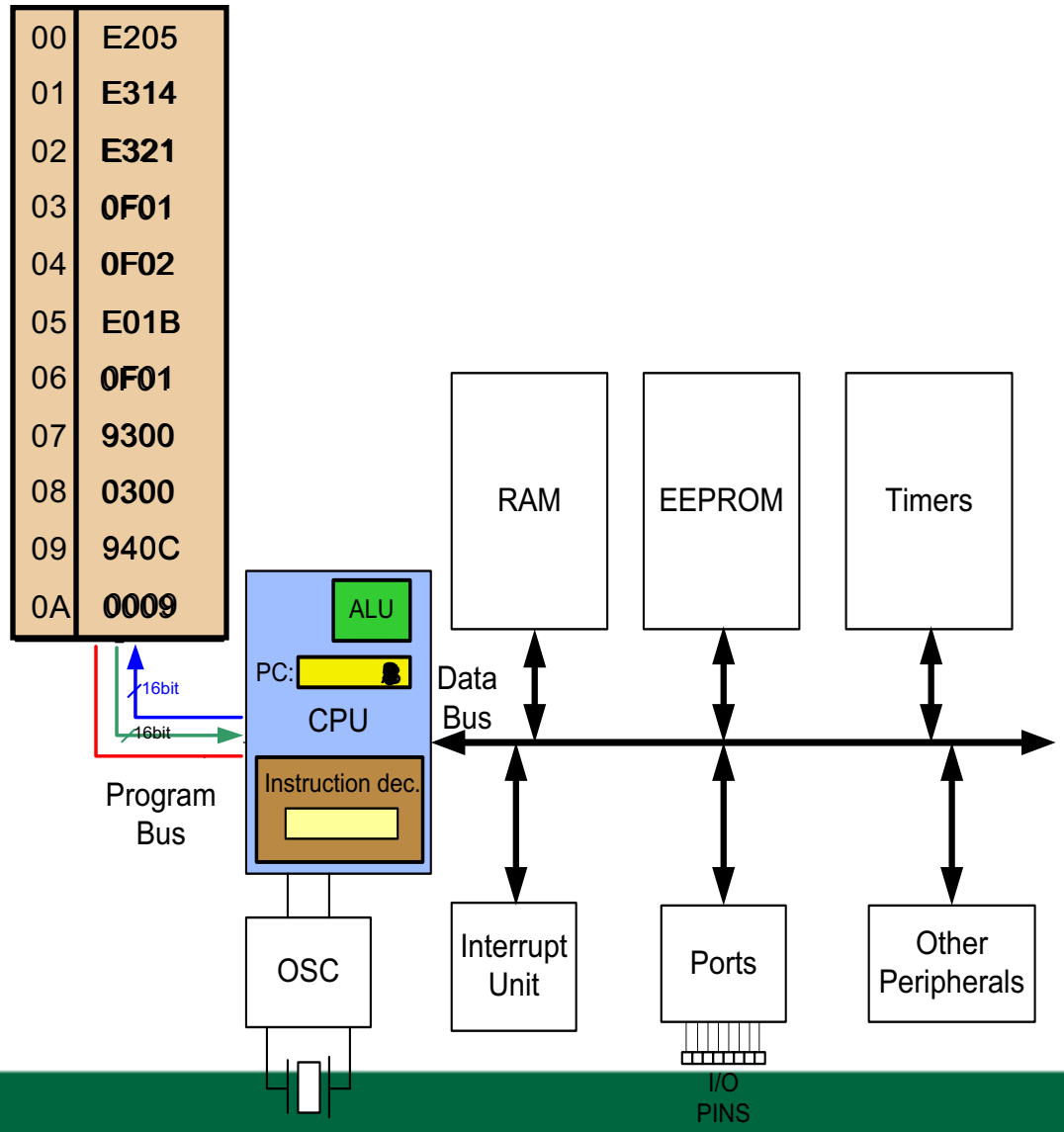
- Seperate Instruction and Data Memories (Harvard)
- all 32 General Purpose Registers connected to ALU
- I/O Modules connected to Data Bus and accessible via Special Function Registers



ATmega2560 Core Architecture

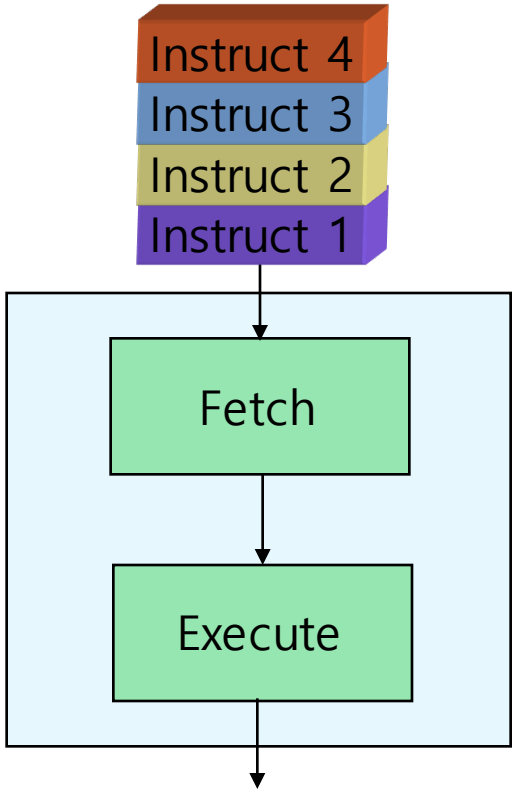
Flash memory and PC register

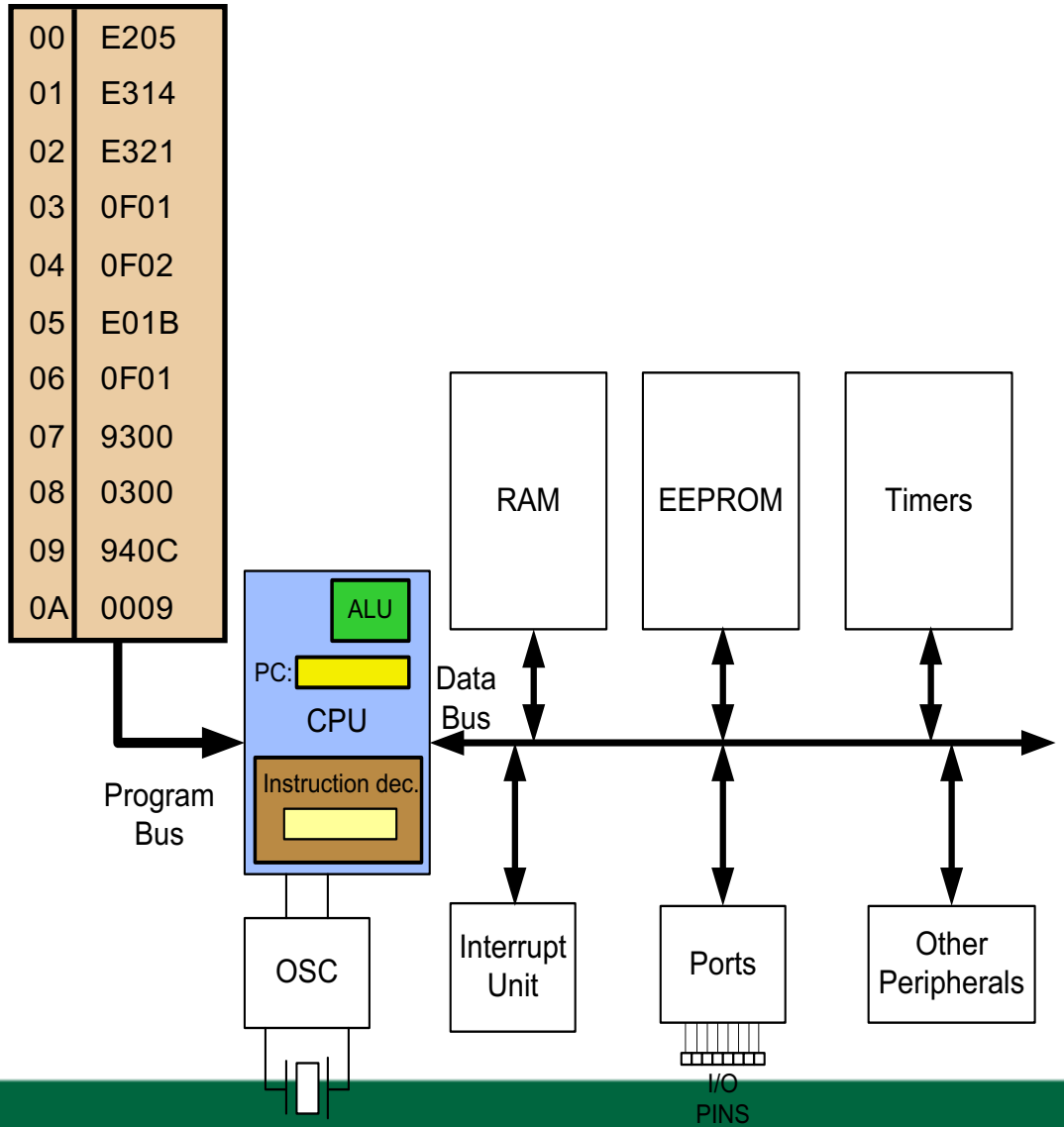
```
        LDI   R16, 0x25
        LDI   R17, $34
        LDI   R18, 0x31
        ADD   R16, R17
        ADD   R16, R18
        LDI   R17, 11
        ADD   R16, R17
        STS   SUM, R16
HERE:JMP  HERE
```



Fetch and execute

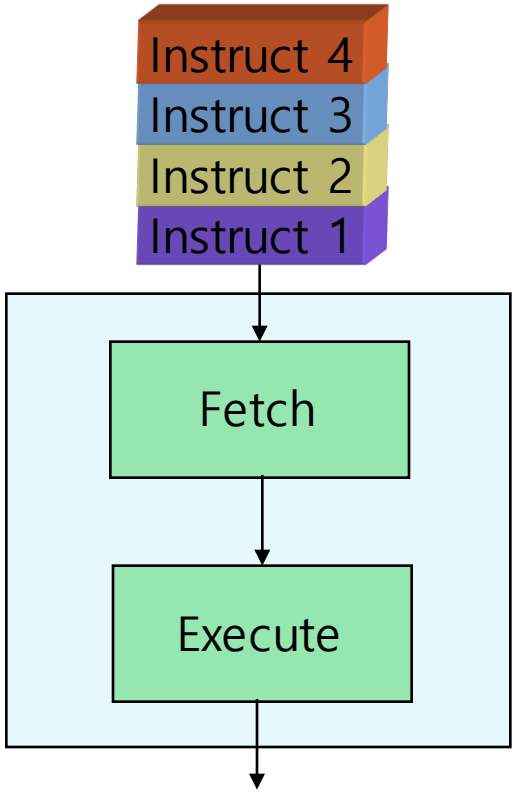
- Old Architectures

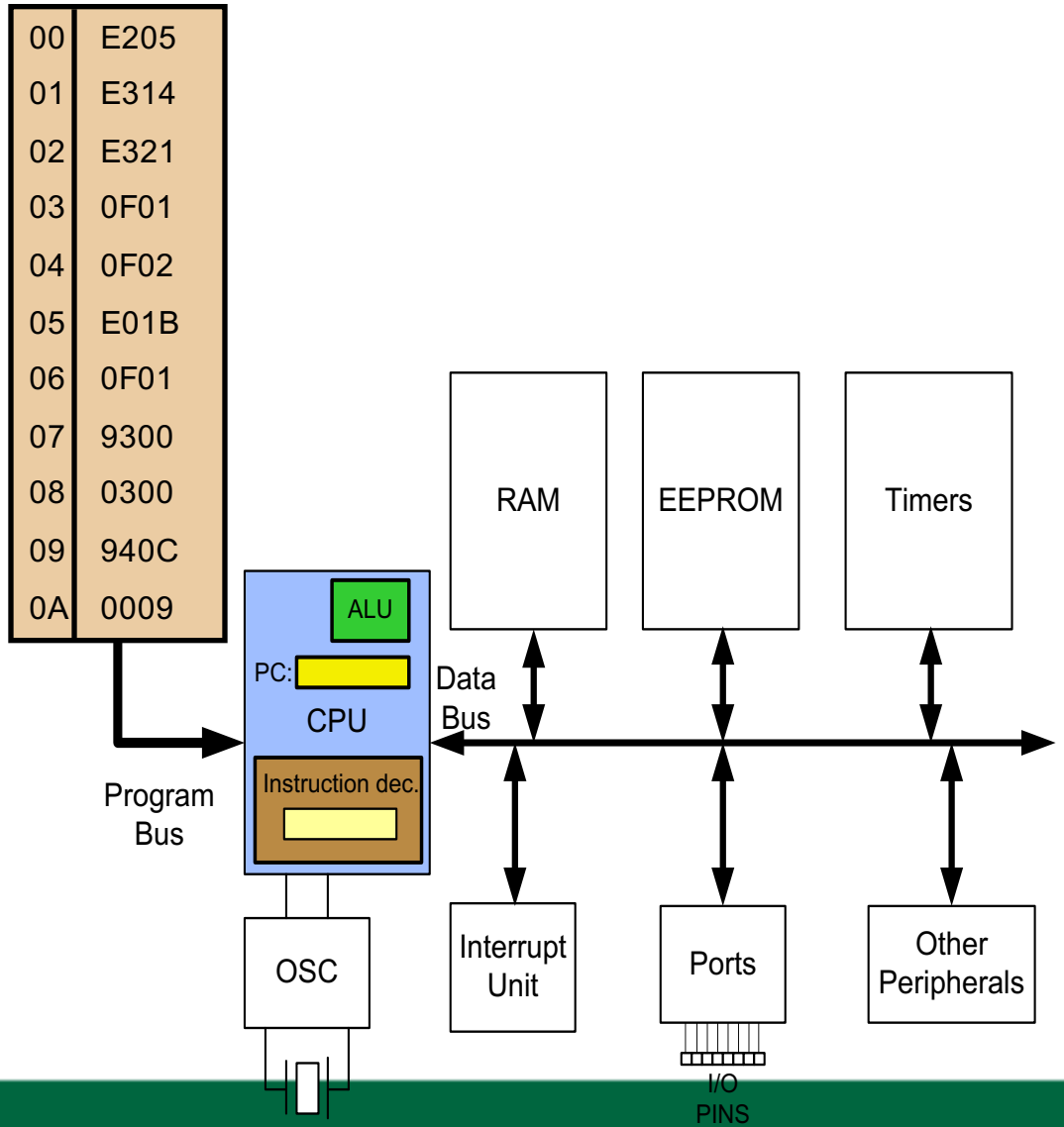




Pipelining

- Pipelining



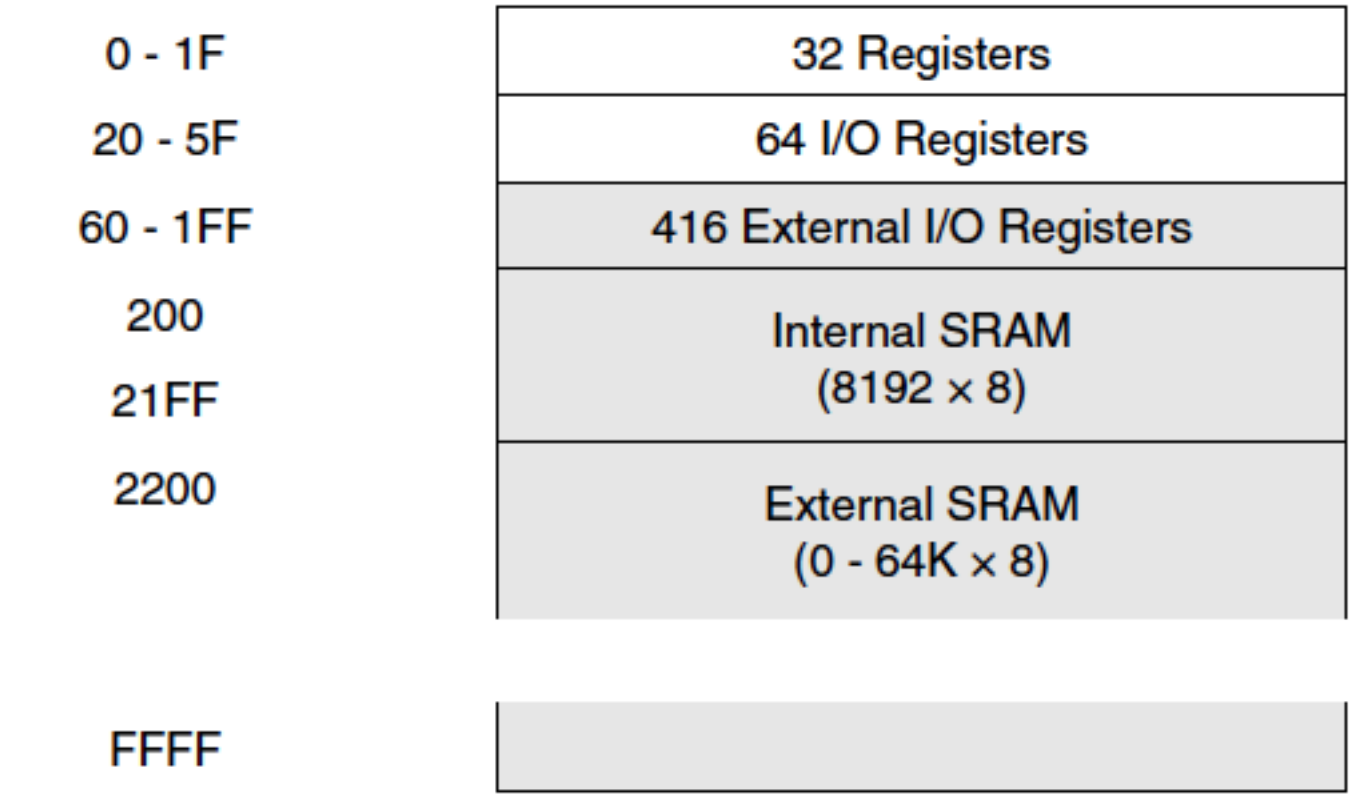


Memory

- **Flash Code Memory**
  - 16-bit words starting at 0x0000
  - Size dependent on AVR microcontroller
  - Non-volatile
  - Read-only memory (writing is external to code)
- **Data Memory**
  - General Purpose Registers
    - 32 8-bit registers
  - I/O Registers
    - Two 8-bit registers for each I/O line
  - SRAM
    - 8-bit memory with size dependent on AVR microcontroller
- **EEPROM Memory**
  - Typically reserved for variables that must retain their value in the event of a s hutdown
    (e.g., system calibration data unique to each board)
    - Slow speed writing (1 millisecond for 1 byte of memory)
    - Limited number of write cycles

Data Memory Map

Figure 8-2. Data Memory Map



AVR Register File

- The Register File
  - 32 8-bit registers

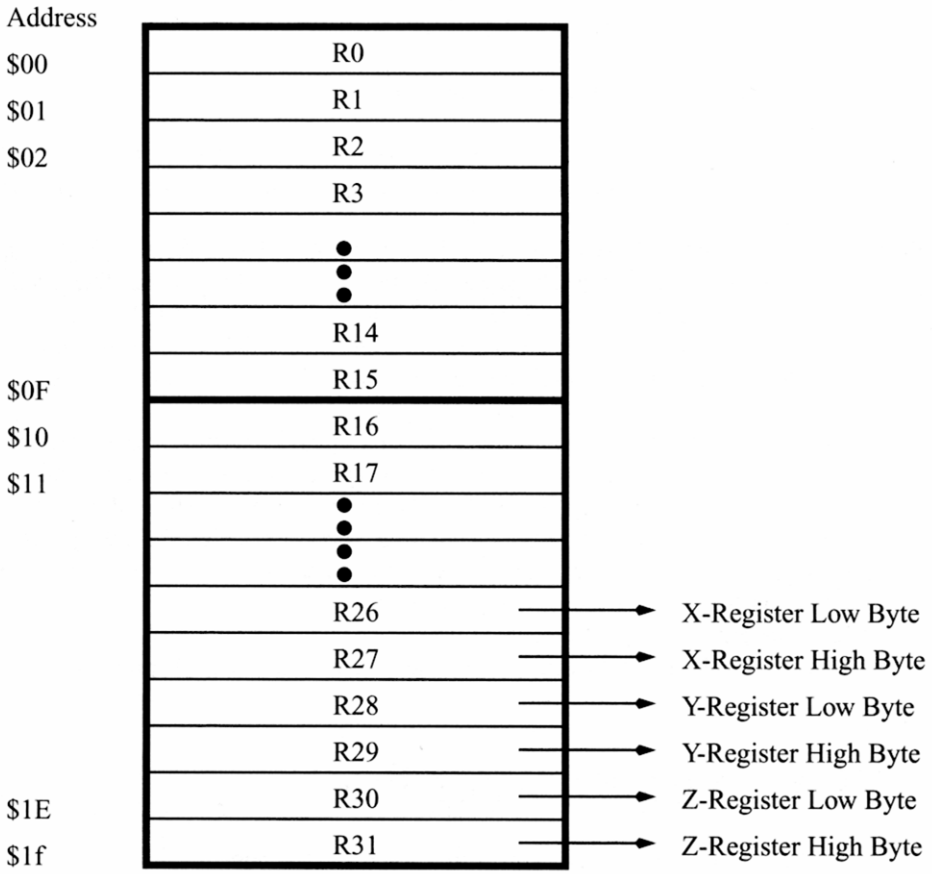


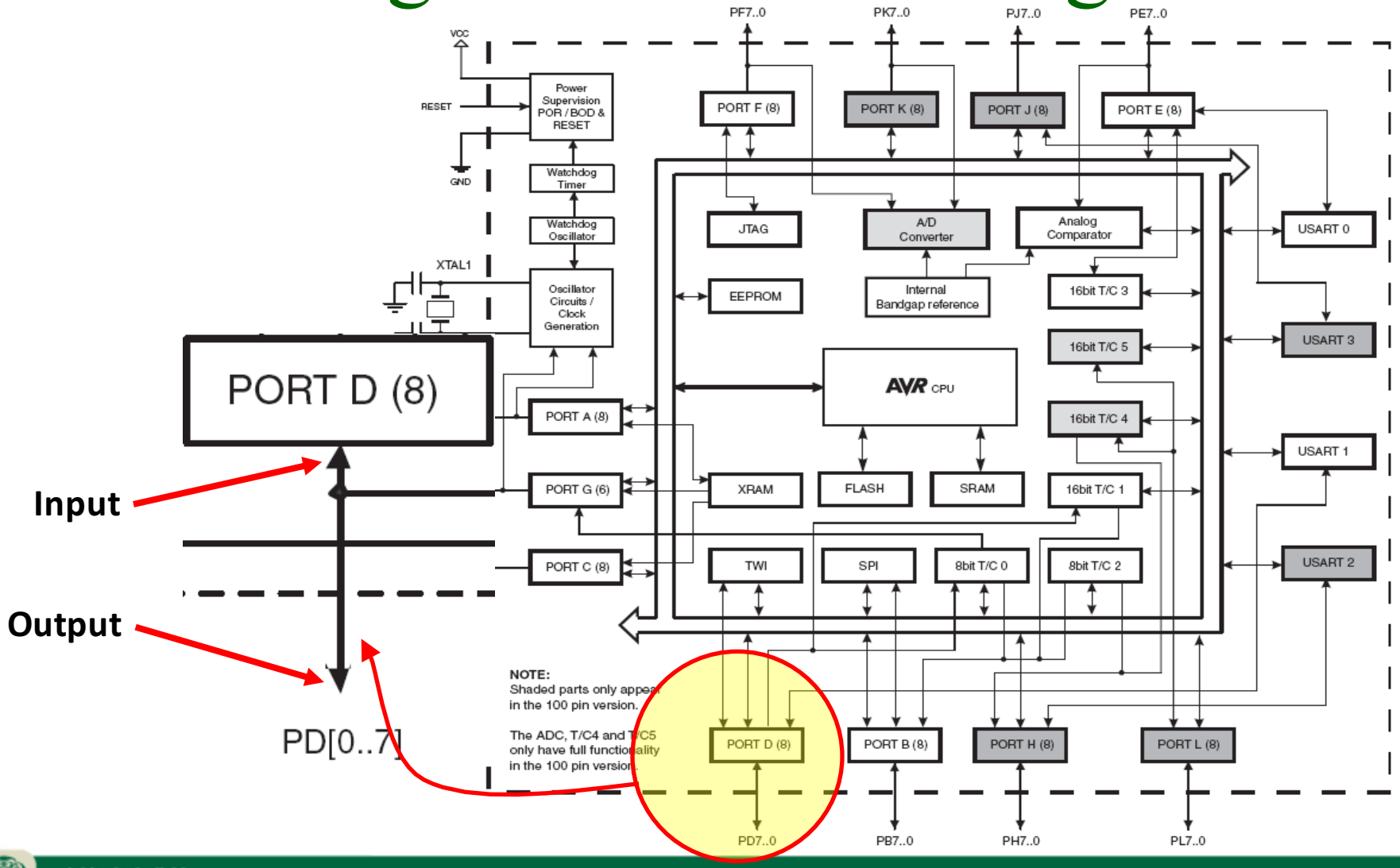
FIGURE 3.4 AVR register file.

I/O Memory Registers

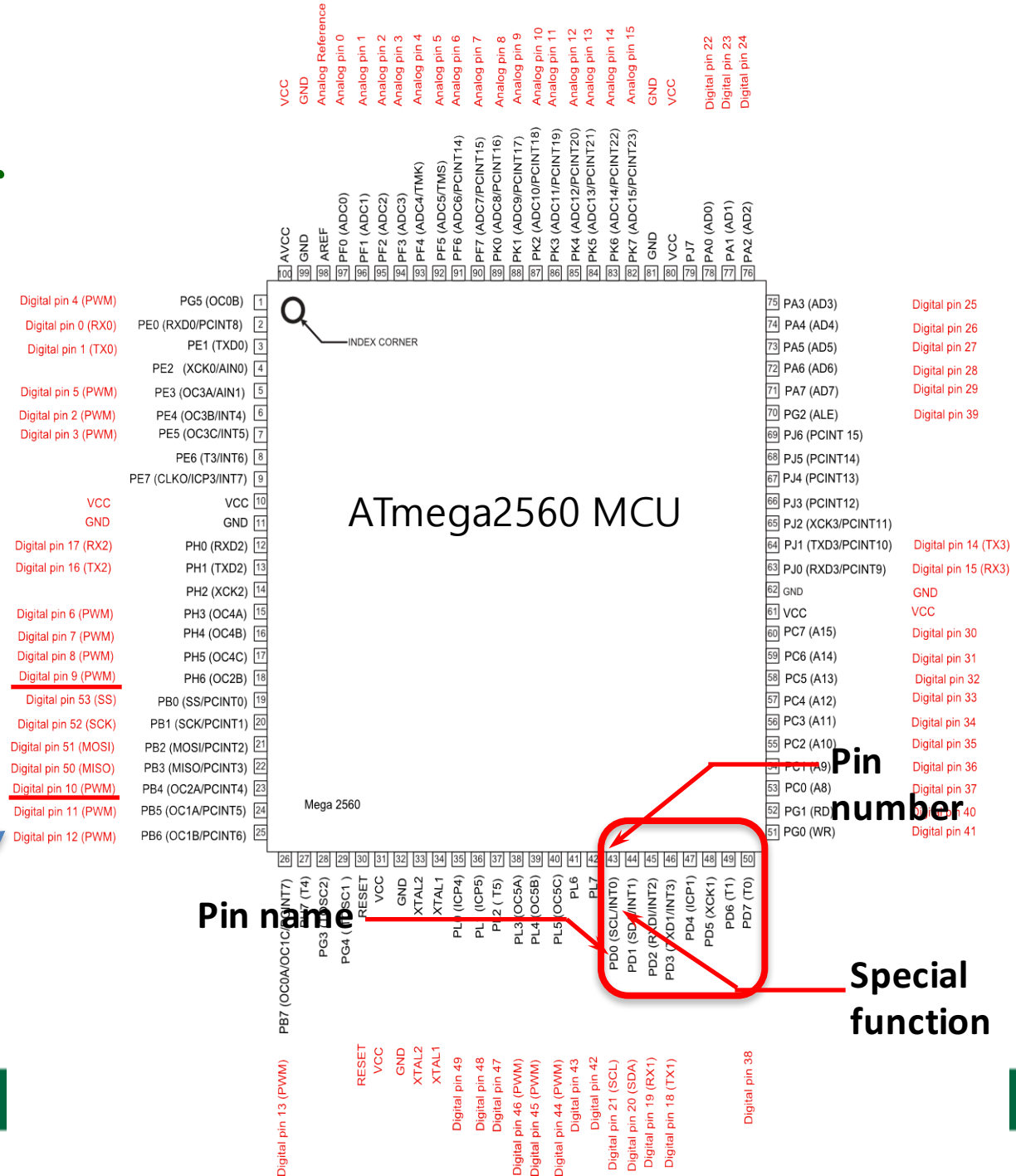
- **SREG**: Status Register
- **SP**: Stack Pointer Register
- **GIMSK**: General Interrupt Mask Register
- **GIFR**: General Interrupt Flag Register
- **MCUCR**: MCU General Control Register
- **MCUSR**: MCU Status Register
- **TCNTO**: Timer/Counter 0 Register
- **TCCR0A**: Timer/Counter 0 Control Register A
- **TCCR0B**: Timer/Counter 0 Control Register B
- **OCR0A**: Timer/Counter 0 Output Compare Register A
- **OCR0B**: Timer/Counter 0 Output Compare Register B
- **TIMSK0**: Timer/Counter 0 Interrupt Mask Register
- **TIFR0**: Timer/Counter 0 Interrupt Flag Register
- **EEAR**: EEPROM Address Register

- **EEDR**: EEPROM Data Register
- **EECR**: EEPROM Control Register
- **PORTB**: PortB Data Register
- **DDRB**: PortB Data Direction Register
- **PINB**: Input Pins on PortB
- **PORTD**: PortD Data Register
- **DDRD**: PortD Data Direction Register
- **PIND**: Input Pins on PortD
- **SPI** I/O Data Register
- **SPI** Status Register
- **SPI** Control Register
- **UART** I/O Data Register
- **UART** Status Register
- **UART** Control Register
- **UART** Baud Rate Register
- **ACSR**: Analog Comparator Control and Status Register

ATmega2560 Block Diagram



Atmega2560 Microcontroller Pin Mapping to Arduino Board



Arduino ADK Board Pin Mapping

PORT Pin and Register Details

PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

Parallel I/O Ports

- Most general-purpose I/O devices
- Each I/O Port has 3 associated registers

1. DDRx (where "x" is A, B, C…)
   - Data Direction Register Port x
   - Determines which bits of the port are input and which are output

```
DDRB = 0x02; /* sets the second lowest of port B to output" */
```

2. PORTx
   - Port Driver Register for write

```
PORTB = 0x02; /* sets the second bit of port B and clears the others */
```

3. PINx
   - Port Pins Registers for read
   - Returns the status of all 8 port B pins.

```
unsigned int x;
x = PINB; /* Places the status of port B into variable x */
```

Input/Output Ports

- All ports initially set to input
  - DDRB = 0x00; /* by default */
- Must declare all output pins using DDRx
  (Data Direction Registry Port x)
- The default for input port pins is *floating*
  Can supply a pull-up resistor by writing logic 1 to the corresponding bit of the port driver register

```
DDRA = 0xC0;  /*  upper 2 bits are output, lower 6 bits are input*/
PORTA = 0x03; /enable internal pull-ups on lowest 2 bits*/
```

- Port pins in output mode are typically capable of sinking 20 mA, but source much less

Data Direction Register (DDR)

- If the bit is *zero* -> pin will be an *input*
  - Making a bit to be zero == '**clearing** the bit'
- If the bit is *one* -> pin will be an *output*
  - Making a bit to be one == '**setting** the bit'
- To change the data direction for a set of pins belonging to PORTx at the same time:
  1. Determine which bits need to be set and cleared in DDRx
  2. Store the binary number or its equivalent (in an alte rnate base, such as hex) into DDRx

Bitwise Operations

- Treat the value as an array of bits
- Bitwise operations are performed on pairs of corresponding bits

```
X = 0b0011, Y = 0b0110
Z = X | Y = 0b0111
Z = X & Y = 0b0001
Z = X ^ Y = 0b0101
Z = ~X = 0b1100
Z = X << 1 = 0b0110
Z = x >> 1 = 0b0001
```

Bit Masks

- Need to access a subset of the bits in a variable
  - Write or read
- Masks are bit sequences which identify the important bits with a '1' value
- Ex. Set bits 3 and 5 or X, don't change other bits

  X = 01010101, mask = 0010100

  X = X | mask
- Ex. Clear bits 2 and 4

  mask = 11101011

  X = X & mask

Bit Assignment Macros

```
#define SET_BIT(p,n) ((p) |= (1 << (n)))
#define CLR_BIT(p,n) ((p) &= ~(1 << (n)))
```

- 1 << (n) and ~(1 << (n)) create the mask
  - Single 1 (0) shifted n times
- Macro doesn't require memory access (on stack)

Example 1

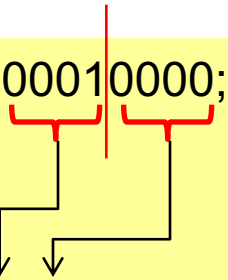
- Make Arduino pin 10 (PB4) to be output
  - Arduino pin 10 connected to SmartCAR front LED

- Arduino approach

```
pinMode(10, OUTPUT);
```

- Alternative approach

```
DDRB = 0b00010000;

or

DDRB = 0x10;

or

DDRB |= (1<<PB4);
```



Example 2

- Make pins Arduino pins 0 and 1 (PE0 and PE1) inputs, and turn on initially (enabling pull-up R)

- Arduino approach

```
pinMode(0, INPUT);
pinMode(1, INPUT);
digitalWrite(0, HIGH);
digitalWrite(1, HIGH);
```

- Alternative approach

```
DDRE = 0; // all PORTE pins inputs
PORTE = 0b00000011;
or
PORTE = 0x03;

or better yet:
  DDRE &= ~(1<<PE1 | 1<<PE0);
  PORTE |= (1<<PE1 | 1<<PE0);
```

Course Announcement

- For lab session, we will cover
  - SmartCAR LED Control

- Next week, we will study on
  - ATmega2560 microcontroller (MCU) architecture
  - Timer & Interrupts