Microprocessors & Interfacing

Lecturer : Annie Guo

Notice

- Before the lecture starts, please
  - Turn off Notifications on your computer
  - Mute your mobile phone
- During the lecture,
  - If you have any questions,
    - raise your hand to ask, or
    - post them in chat (preferred)
  - Set the microphone properly in your MS Teams
    - When you are allowed to speak
      - Unmute the microphone
    - When you are not speaking
      - Mute the microphone

Lecture Overview

- Course Introduction
  - A whole picture of the course
- Basics of Computing with Microprocessor Systems

Course Organization

- Lecture:
  - Online (6-8pm, Mon. & Tue.)
- Lab:
  - Online and face-to-face (f2f)
    - f2f classroom: ELecEng119 (K-J17)
    - Social distancing should be maintained
      - See a guide posted on the course website
  - Four labs (start from Week2)
    - Week 1: Set up the simulation environment at home; Lab groups are formed (three students per group)
- Assignment (Project design):
  - Microprocessor application

Goals of the Course

- After completing the course, you should
  – Understand the basic concepts and structure of microprocessors
  – Gain assembly programming skills
  – Understand how hardware and software interact with each other
  – Know how to use microprocessors to solve problems
  – Be familiar with the development of microprocessor applications

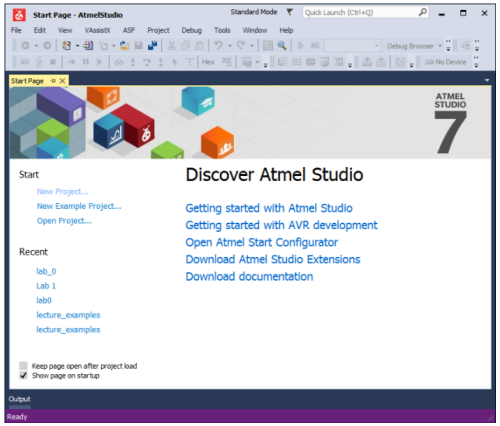
Strategies (1)

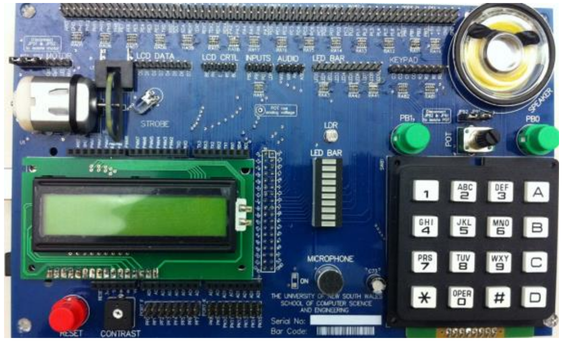
- Lectures
  - Concepts
  - Principles
  - Problem solving approaches and techniques

Key topics
- **Basics of Computing with Microprocessor Systems**
- **Instruction set architecture**
- **AVR assembly programming**
- **Input and Output**
- **Interrupt**
- **Serial communication**
- **Analog/digital and digital/analog conversion**

Strategies (2)

- Labs
  - Lab tools
    - Atmel studio development environment
      - Development, simulation and debug
    - AVR lab board
      - Devices and connections
      - Programming and testing
  - Lab exercises
    - Prepare before the lab class
    - Finish in lab
    - Marked off by the lab tutor
      - Late penalty
        - » **30%** per week





Strategies (3)

- Project design
  - Through a whole design cycle
  - Apply what have been learnt in the course
    - concepts
    - approaches and techniques
  - Collaborate with team members
  - Communicate project work
    - Oral demonstration
      - Assessed as a group
    - Written report
      - Assessed individually
    - Late penalty
      - 10% per day

Strategies (4)

- Homework
  - Readings
  - Online quizzes or polls
    - For quick feedback
  - Etc

How are labs run?

- Lab exercises are carried out in groups. Each group has three students.
- COVID-19 specific arrangement
  - Social distancing should be maintained in the f2f classes
    - Up to 12 students are allowed each time in the laboratory
  - The flexible students in a group can take turns to attend f2f class.
    - E.g.

	studentA	studentB	studentC
lab-week i	online	online	f2f
lab-week i+1	online	f2f	online
lab-week i+2	online	online	f2f

How are labs run? (cont.)

- Members in each group can work together to leverage the advantages from both online and f2f classes.
- Furthermore
  - The lab boards will be distributed through f2f classes
  - Each group is required to have at least one student be able to access the lab board
- Therefore, you will be assigned to a lab group that has at least one flexible student.

Assessment

- Lab exercises
  - 25% (of the final result)
  - Mainly carried out in group
    - Mostly being marked individually
- Project design
  - 15%
  - Carried out in group
    - Design demonstration marked as a group
    - Report submitted individually and marked individually
- Final exam
  - 60%
- To pass the course,
  - (final result >=50)&(**final_exam>=40**)

References

- Main references:
  - Fredrick M. Cady: Microcontrollers and Microcomputers — Principles of Software and Hardware Engineering
  - AVR documents (available on the course website)
    - Data Sheet
    - Instruction Set
  - Additional materials provided on the course website
- Lecture notes
  - Posted before each lecture
- Lecture recordings
  - Available after each lecture

Resources for Help

- Course website
  - www.cse.unsw.edu.au/~cs9032
- Lecturer
  - Consultation
    - Wed. 16:00—17:30
- Lab tutors
- Course forum

NOTE

- From time to time I will post notice on the course website.
- Please check the website frequently for new notices, lectures, lab exercises, and the assignment specification.

Microprocessors & Interfacing

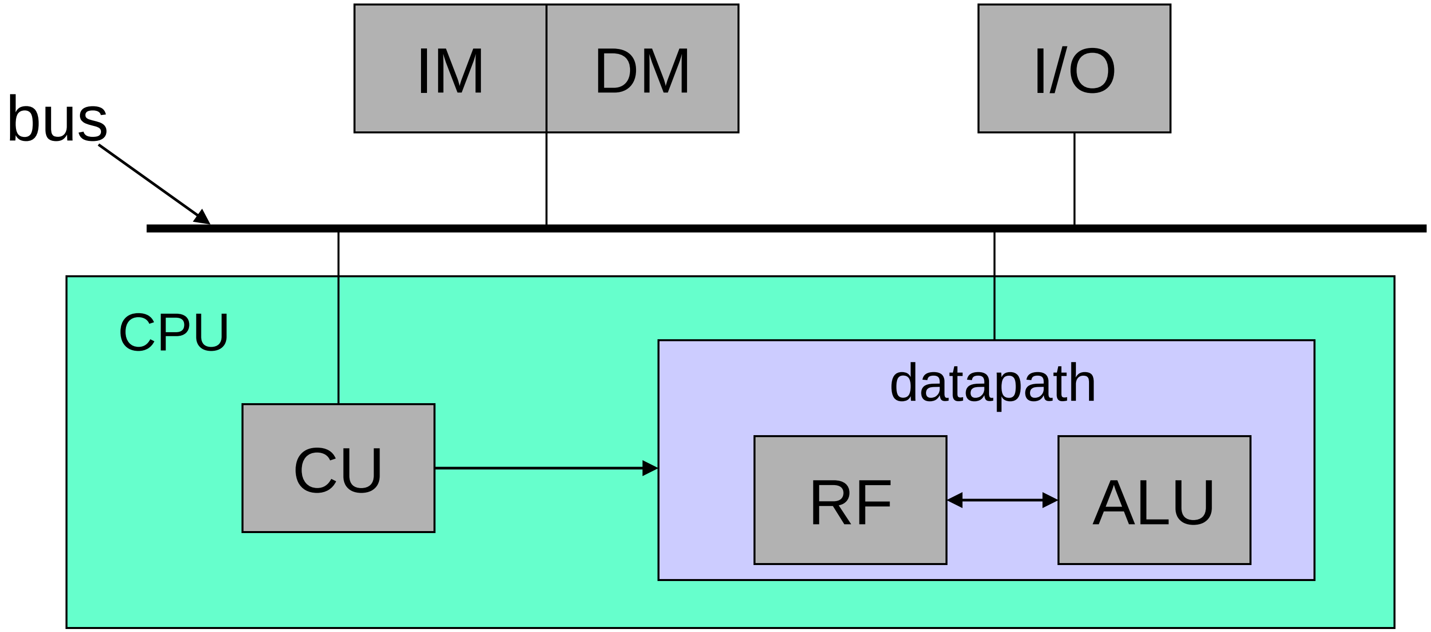
Basics of Computing with Microprocessor Systems

Lecturer: Annie Guo

Lecture Overview

- Microprocessor Hardware Structures
- Data Representation
  - Binary code
- Instruction Set Architecture

Fundamental Hardware Components in Computing System



- **ALU**: Arithmetic and Logic Unit
- **RF**: Register File (a set of registers)
- **CU**: Control Unit
- **IM/DM**: Instruction/Data Memory
- **I/O**: Input/Output Devices

Execution Cycle

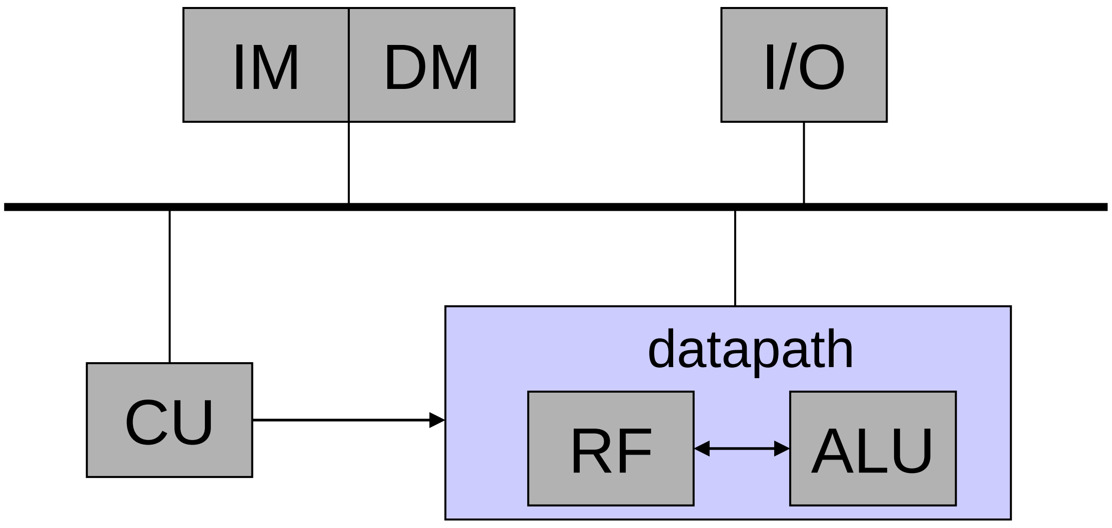
IF: Instruction Fetch

ID: Instruction decode

RR: Read Register File

EX: Execution

WR: Write result back



Note: Steps can be merged/broken down/expanded

Microprocessor

- A *microprocessor* is the datapath and control unit on a single chip.
  - Note, it often includes other components for functional and performance enhancement
- If a microprocessor, its associated support circuitry, memory and peripheral I/O components are implemented on a single chip, it is a *microcontroller*.
  - We use Atmel AVR microcontroller as the example in our course

Data Representation

- For a digital microprocessor system to be able to compute and process data, the data must be properly represented
  - How to represent numbers for calculation?
    - Binary number
    - Binary code
  - How to represent characters, symbols and other values for processing?
    - Will be covered later

Binary

- Example

$$\boxed{\begin{aligned}&(1011)_2\\&=1\times 2^3+0\times 2^2+1\times 2+1\end{aligned}}$$

  - All digits (aka bits) must be less than 2 (0~1).

Hexadecimal

- Example

$$
\begin{aligned}
&(F24B)_{16} \\
&= F \times 16^3 + 2 \times 16^2 + 4 \times 16 + B \\
&= 15 \times 16^3 + 2 \times 16^2 + 4 \times 16 + 11
\end{aligned}
$$

  - All digits must be less than 16 (0~9,**A,B,C,D,E,F)**

- Conversion between binary to hexadecimal
  - One hexadecimal digit ←→ 4 binary digits
  - E.g. 0xA1

    0b110111

Binary Arithmetic Operations

- Are similar to decimal operations
- Examples of addition and multiplication are given in the next two slides.

Binary Addition

- Example:
  - Addition of two 4-bit binary numbers. How many bits are required for holding the result?

1001+0110 = (___________)

Binary Multiplication

- Example:
  - Multiplication of two 4-bit unsigned binary numbers. How many bits are required for holding the result?

1001*0110 = (__________________________)

Binary Subtraction

- Subtraction can be defined as addition of the additive inverse (namely signed addition)

  $a - b = a + (-b)$

- We use **two's complement code/number,** b^*, to represent $-b$.

$$b^* = 2^n - b = (2^n - 1) - b + 1$$

  - $(b^*)^* = b$
  - The **MSB** (Most Significant Bit) of a 2's complement code is the **sign bit**
    - For example, for a 4-bit 2's complement number
    - (1001) → -7,   (0111) → 7

Exercise 1

- For each of the following decimal numbers, what is its 8-bit 2's complement number?
    - (a) 7
    - (b) 127
    - (c) -12

- An *n*-bit binary number can be **interpreted** in two different ways: signed or unsigned. What decimal value does the 4-bit number, *1011*, represent in each of the following two cases?
    - (a) if it is a signed number
    - (b) if it is an unsigned number

Signed Addition

- E.g. 4-bit 2's-complement additions/subtractions

(1) 0101 + 0010 (5 + 2):

```
    0101
+   0010
--------
= 00111
```

(2) 0101 - 0010 (5 - 2):

```
    0101
+   1110  (= 0010*)
--------
= 10011
```

(3) 0010 - 0101 (2 - 5):

```
    0010
+   1011  (= 0101*)
--------
=   1101  (= 0011*).
```

Result means -3.

(4) -0101 - 0010 (-5 - 2):

```
    1011  (= 0101*)
+   1110  (= 0010*)
--------
= 11001
```

Result means -7.

Overflow

- In digital computer systems, values are represented by a fixed number of bits.
- Overflow happens when the calculation result is beyond the range that can be represented with the given number size.

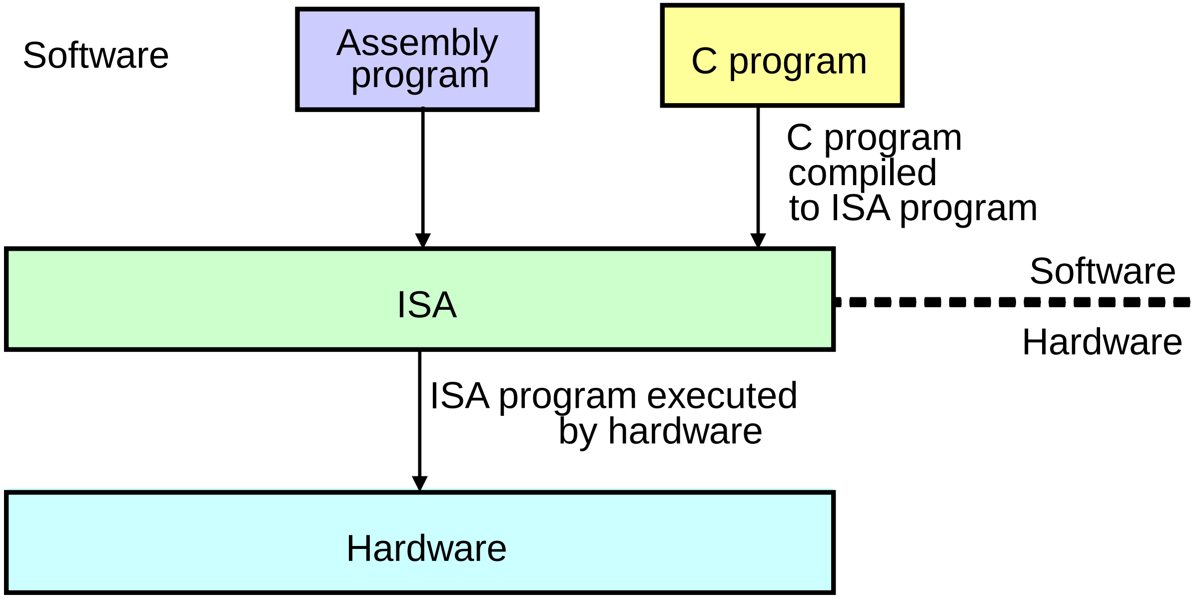
Exercise 2

For the following 4-bit **signed** calculations, check whether there are any overflows.

1) 1000-0001
2) 1000+0101
3) 0101+0110

Microprocessor Applications

- A microprocessor application system can be abstracted in a three-level structure
  – ISA (Instruction Set Architecture) is the interface between hardware and software



Instruction Set

- Instruction set provides the vocabulary and grammar for programmer/software to communicate with the hardware machine.
- It is machine oriented
  - Different type of machines have different instruction set
    - For example
      - 68K has a more comprehensive instruction set than ARM machine
  - Same operations could be represented differently in different machines
    - AVR
      - Addition: *add r2, r1* ;r2 ← r2+r1
      - Branching: *breq 6* ;branch if equal condition is true
      - Load: *ldi r30, $F0* ;r30 ← F0
    - 68K:
      - Addition: *add d1,d2* ;d2 ← d2+d1
      - Branching: *breq 6* ;branch if equal condition is true
      - Load: mov #1234, d2 ;d2 ← 1234

Instructions

- Instructions can be written in two languages
  - Machine language
    - Made of binary digits
    - Used by machines
  - Assembly language
    - Text representation of machine language
    - Easier to understand than machine language
    - Used by human being.

Machine Code vs. Assembly Code

- Basically, there is a one-to-one mapping between machine instructions and assembly instructions
  - For example, AVR instruction for incrementing register r16 by 1:
    - 1001010100000011 (machine code)
    - inc r16 (assembly code)
- Assembly language also includes **directives**
  - Directives
    - Instructions to the assembler
      - **Assembler** is a program to translate assembly code into machine code.
    - Example:
      - **.def** temp = r16
      - **.include** "m2560def.inc"

Instruction Set Architecture (ISA)

- ISA specifies all aspects of a computer architecture visible to a programmer
  - Instructions (just mentioned)
  - Native data types
  - Registers
  - Memory models
  - Addressing modes

Native Data Types

- Different machines support different data types in hardware
  - e.g. Pentium II:

Data Type	8 bits	16 bits	32 bits	64 bits	128 bits
Signed integer	✓	✓	✓		
Unsigned integer	✓	✓	✓		
BCD integer	✓				
Floating point			✓	✓	

  - e.g. Atmel AVR (we are using):

Data Type	8 bits	16 bits	32 bits	64 bits	128 bits
Signed integer	✓				
Unsigned integer	✓				
BCD integer					
Floating point					

Registers

- Two types
  - General purpose
  - Special purpose
    - e.g.
      - Program Counter (PC)
      - Status Register
      - Stack Pointer (SP)
      - Input/Output Registers
  - Stack Pointer and Input/Output Registers will be discussed in detail later.

General Purpose Registers

- A set of registers in the machine
  - Used for storing temporary data/results
  - For example
    - In (68K) instruction *add d3, d5, the operands of the operation are stored in general registers d3 and d5, and the result is stored in d5.*
- Can be structured differently in different machines
  - For example
    - Separate general-purpose registers for data and address
      - 68K
    - Different number of registers and different size of registers
      - 32 32-bit registers in MIPS
      - 16 32-bit registers in ARM

Program Counter (PC)

- Special register
  - For storing the memory address of currently executed instruction
- Can be of different size
  - E.g. 16 bits, 32 bits
- Can be auto-incremented
  - By the instruction word size
  - Hence, giving rise to the name “counter”

Status Register

- Contains a number of bits with each bit being associated with processor (CPU) operations
- Typical status bits
  - V: Overflow
  - C: Carry
  - Z: Zero
  - N: Negative
- Used for controlling the program execution flow

Memory Model

- Related to how memory is used to store data
- Issues
  - Addressable unit size
  - Address spaces
  - Endianness
  - Alignment

Addressable Unit Size

- Memory has units, each of which has an address
- Most basic unit size is 8 bits (1 byte)
  - Related addresses are called **byte-addresses**.
- Modern processors can have multiple-byte unit
  - e.g. 32-bit instruction memory in MIPS

    16-bit instruction memory in AVR
  - Related addresses are called **word-addresses**.

Address Space

- The range of addresses a processor can access.
  - A processor can have one or more address spaces. For example
    - Princeton architecture or Von Neumann architecture
      - A single linear address space for both instructions and data memory
    - Harvard architecture
      - Separate address spaces for instruction and data memories

Address Space (cont.)

- Address space is not necessarily just for "memory"
  - E.g, all general purpose registers and I/O registers can be accessed through memory addresses in AVR

Endianness

- Memory objects
  - Memory objects are basic entities that can be accessed as a function of the **address** and the **length**
    - E.g. bytes, words, longwords
- For large objects (multiple bytes), there are two byte-ordering conventions
  - **Little endian** – little end (least significant byte) stored first (at lowest address)
    - Intel microprocessors (Pentium etc)
  - **Big endian** – big end (most significant byte) stored first
    - SPARC, Motorola microprocessors

Big Endian & Little Endian

- Example: 0x12345678—a long word of 4 bytes. It is stored in the memory from a byte address 0x00000013
  - big endian:

Byte Address	data
0x00000013	0x12
0x00000014	0x34
0x00000015	0x56
0x00000016	0x78

  - little endian:

Byte Address	data
0x00000013	0x78
0x00000014	0x56
0x00000015	0x34
0x00000016	0x12

Alignment

- Modern computers read from or write to a memory address in fix-sized chunks
  - for example, word size
- Alignment improves the memory access efficiency by putting the data at a memory address that is multiple of the chunk size
  - for example, with AVR, data of the word type in the program memory are aligned with the word addresses.
    - 0x1234

Byte Address	data
0x00000013	0x12
0x00000014	0x34

not aligned

Byte Address	data
0x00000014	0x12
0x00000015	0x34

aligned

Addressing Mode

- Instructions need to specify where to get operands
- Some possible ways
  - an operand value is in the instruction
  - an operand value is in a register
    - the register number is given in the instruction
  - an operand value is in memory
    - address is given in instruction
    - address is given in a register
      - the register number is in the instruction
    - address is a register content plus some offset
      - register number is in the instruction
      - offset is in the instruction (or in a register)
- These ways of specifying the operand locations are called **addressing mode.**

Addressing Mode (cont.)

- Some examples, based on the 68K machine, are given in the next slides.
  - Using instruction: *addw a, b*
    - *addition on operands of the word size, b ← a+b*
- For each addressing mode, there are
  - a general description and
  - an example to show how the address mode is used.
    - the specified addressing mode for the first operand of instruction is highlighted in red.

Immediate Addressing

- The operand is directly from the instruction
  - i.e the operand is immediately available from the instruction

- For example, in 68K

```
addw     #99, d7
```

  - d7 ← 99 + d7; value 99 comes from the instruction
  - d7 is a register

Register Direct Addressing

- Data from a register and the register is directly given by the instruction

- For example, in 68K

addw	***d0*****,d7**

  - d7 ← d7 + d0; add value in d0 to value in d7 and store result to d7
  - d0 and d7 are registers

Memory Direct Addressing

- The data is from memory, the memory address is directly given by the instruction
- We use notion *(addr)* to represent memory value at address *addr.*

- For example, in 68K

```
addw        0x123A, d7
```

  – d7 ← d7 + (0x123A); add value in memory location 0x123A to register d7

Memory Register Indirect Addressing

- The data is from memory, the memory address is given by a register, which is directly given by the instruction

- For example, in 68K

```
addw        (a0),d7
```

  – d7 $\leftarrow$ d7 + (a0); add value in memory with the address stored in register a0, to register d7
    - For example, if a0 = 100 and (100) = 123, then this adds 123 to d7

Memory Register Indirect Auto-increment

- The data is from memory, the memory address is given by a register, which is directly given by the instruction; and the value of the register is automatically increased – to point to the next memory object.

- For example, in 68K

addw ***(a0)+*****,d7**

– d7 ← d7 + (a0); a0 ← a0 + 2

Memory Register Indirect Auto-decrement

- The data is from memory, the memory address is given by a register, which is directly given by the instruction; but the value of the register is automatically decreased before such an operation.

- For example, in 68K

```
addw        -(a0),d7
```

  – a0 ← a0 –2; d7 ← d7 + (a0);

Memory Register Indirect with Displacement

- Data is from the memory with the address given by the register plus a constant
  - Used in the access a member in a data structure

- For example, in 68K

addw **a0@(8), d7**

  - d7 $\leftarrow$ (a0+8) +d7

Address Register Indirect with Index and Displacement

- The address of the data is sum of the initial address and the index address as compared to the initial address.
  - Used in accessing element of an array

- For example, in 68K

```
addw      a0@(d3)8, d7
```

  - d7 ← (a0 + d3+8)
  - With a0 as an initial address and d3 varied to dynamically point to different elements plus a constant for a certain member of an element of an array.

Reading Material

– Cady “Microcontrollers and Microprocessors”, Chapter 1.1, Chapter 2.2-2.4
– Cady “Microcontrollers and Microprocessors”, Appendix A
– Week 1 reference: “Number Conversion”
    • available on the course website

Homework

1. Install Atmel Studio at home and complete lab0
   - Available on the **Labs** page on the course website

2. Complete Quiz 1
   - Released after the lecture
   - Available on the **Activities** page on the course website