Content of F5

- Encoding negative numbers
  – the ”2-complement concept”
- Systems engineering/Software engineering
  – How to think
  – How to implement systems with menus
- The ”HIGH” and ”LOW” keywords and setting the Stack pointer.
- Looking at timing calculations to create a ”fair” dice…

2's Complement Arithmetic….

…and negative numbers.

Addition, Subtraction and Negative numbers…

- We place binary numbers In registers and memory locations.
- However, we need to decide how to interpret those numbers….
- Simple binary numbers work well for addition, but:
  - What happens when numbers are too big?
  - How can we represent negative numbers?
  - Do we need a subtractor circuit?
- Note: The AVR processor do have a SUB instruction (applies when unsigned numbers are used)….

2's Complement Arithmetic

This presentation will demonstrate

- That subtracting one number from another is the same as making one number negative and adding.
- How to create negative numbers in the binary number system.
- The 2's Complement Process.
- How the 2's complement process can be used to add (and subtract) binary numbers.

Negative Numbers?

- Digital electronics requires frequent addition and subtraction of numbers. You know how to design an adder, but what about a subtract-er?
- A subtract-er is not needed with the 2's complement process. The 2's complement process allows you to easily convert a positive number into its negative equivalent.
- Since subtracting one number from another is the same as making one number negative and adding, the need for a subtractor circuit has been eliminated.

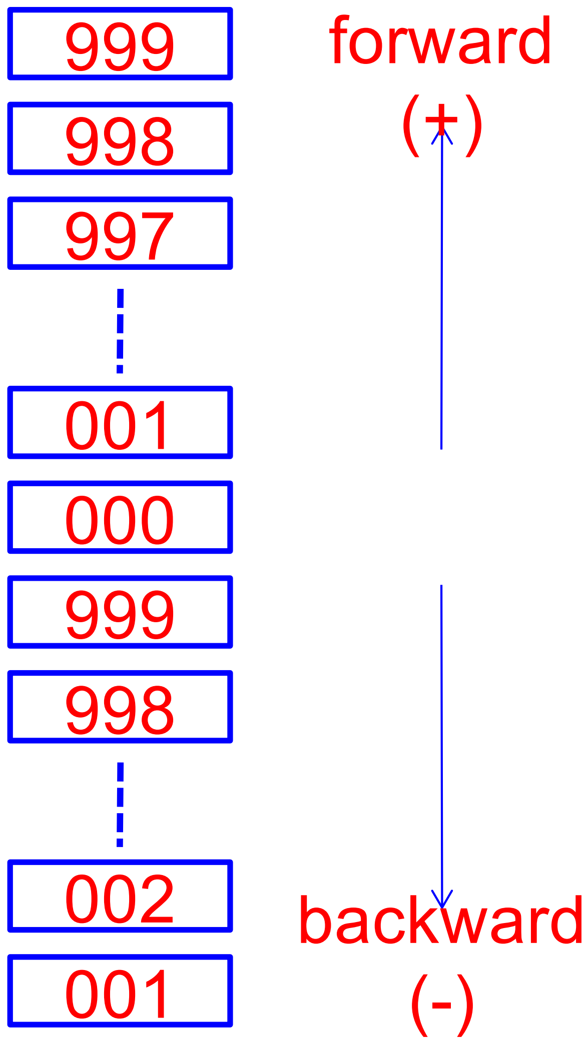
How To Create A Negative Number

- In digital electronics you cannot simply put a minus sign in front of a number to make it negative.
- You must represent a negative number in a *fixed-length* binary number system. All signed arithmetic must be performed in a *fixed-length* number system.
- A physical *fixed-length* device (usually memory) contains a fixed number of bits (usually 4-bits, 8-bits, 16-bits) to hold the number.

3-Digit Decimal Number System

A bicycle odometer with only three digits is an example of a fixed-length decimal number system.

The problem is that without a negative sign, you cannot tell a +998 from a -2 (also a 998). Did you ride forward for 998 miles or backward for 2 miles?



Note: Car odometers do not work this way.

Negative Decimal

How do we represent negative numbers in this 3-digit decimal number system without using a sign?

→Cut the number system in half.

→Use 001 – 499 to indicate positive numbers.

→Use 500 – 999 to indicate negative numbers.

→Notice that 000 is not positive or negative.

+499	499	pos(+)
+498	498	
+497	497	
⋮	⋮	
+001	001	
000	000	
-001	999	
-002	998	
⋮	⋮	
-499	501	
-500	500	neg(-)

“Odometer” Math Examples

$$\begin{array}{r} 3 \\ +\ \ 2 \\ \hline 5 \end{array} \qquad \begin{array}{r} 003 \\ +\ \ 002 \\ \hline 005 \end{array} \qquad \begin{array}{r} 6 \\ +\ (-3) \\ \hline 3 \end{array} \qquad \begin{array}{r} 006 \\ +\ \ 997 \\ \hline 1]003 \end{array}$$

Disregard Overflow

$$\begin{array}{r} (-5) \\ +\ \ 2 \\ \hline (-3) \end{array} \qquad \begin{array}{r} 995 \\ +\ \ 002 \\ \hline 997 \end{array} \qquad \begin{array}{r} (-2) \\ +\ (-3) \\ \hline (-5) \end{array} \qquad \begin{array}{r} 998 \\ +\ \ 997 \\ \hline 1]995 \end{array}$$

Disregard Overflow

It Works!

Complex Problems

- The previous examples demonstrate that this process works, but how do we *easily* convert a number into its negative equivalent?
- In the examples, converting the negative numbers into the 3-digit decimal number system was fairly easy. To convert the (-3), you simply counted backward from 1000 (i.e., 999, 998, 997).
- This process is not as easy for large numbers (e.g., -214 is 786). How did we determine this?
- To convert a large negative number, you can use the 10's Complement Process.

10's Complement Process

The **10's Complement** process uses base-10 (decimal) numbers. Later, when we're working with base-2 (binary) numbers, you will see that the **2's Complement** process works in the same way.

First, complement all of the digits in a number.

– A digit's complement is the number you add to the digit to make it equal to the largest digit in the base (i.e., 9 for decimal). The complement of 0 is 9, 1 is 8, 2 is 7, etc.

Second, add 1.

– Without this step, our number system would have two zeroes (+0 & -0), which no number system has.

10's Complement Examples

Example #1

$$\begin{array}{r} -003 \\ \downarrow\downarrow\downarrow \\ 996 \\ +1 \\ \hline 997 \end{array}$$

Complement Digits

Add 1

Example #2

$$\begin{array}{r} -214 \\ \downarrow\downarrow\downarrow \\ 785 \\ +1 \\ \hline 786 \end{array}$$

Complement Digits

Add 1

8-Bit Binary Number System

Apply what you have learned to the binary number systems. How do you represent negative numbers in this 8-bit binary system?

→Cut the number system in half.

→Use 00000001 – 01111111 to indicate positive numbers.

→Use 10000000 – 11111111 to indicate negative numbers.

→Notice that 00000000 is not positive or negative.

+127	01111111	pos(+)
+126	01111110	
+125	01111101	
⋮	⋮	
+1	00000001	
0	00000000	
-1	11111111	
-2	11111110	
⋮	⋮	
-127	10000001	
-128	10000000	neg(-)

Sign Bit

- What did do you notice about the most significant bit of the binary numbers?
- The MSB is (0) for all positive numbers.
- The MSB is (1) for all negative numbers.
- The MSB is called the sign bit.
- In a signed number system, this allows you to instantly determine whether a number is positive or negative.

+127	01111111	pos(+)
+126	01111110	
+125	01111101	
⋮	⋮	
+1	00000001	
0	00000000	
-1	11111111	
-2	11111110	
⋮	⋮	
-127	10000001	
-128	10000000	neg(-)

2' S Complement Process

The steps in the **2' s Complement** process are similar to the 10' s Complement process. However, you will now use the base two.

1. **First, complement all of the digits in a number**.
   – A digit' s complement is the number you add to the digit to make it equal to the largest digit in the base (i.e., 1 for binary). In binary language, the complement of 0 is 1, and the complement of 1 is 0.
2. **Second, add 1.**
   – Without this step, our number system would have two zeroes (+0 & -0), which no number system has.

2's Complement Examples

Example #1

```
  5 = 00000101
      11111010
            +1
      --------
 -5 = 11111011
```

Complement Digits

Add 1

Example #2

```
-13 = 11110011
      00001100
            +1
      --------
 13 = 00001101
```

Complement Digits

Add 1

Using The 2's Complement Process

Use the 2's complement process to add together the following numbers. 4 combinations…

$$\begin{array}{r} \text{POS} \\ +\ \text{POS} \\ \hline \text{POS} \end{array} \Rightarrow \begin{array}{r} 9 \\ +\ 5 \\ \hline 14 \end{array} \qquad \begin{array}{r} \text{NEG} \\ +\ \text{POS} \\ \hline \text{NEG} \end{array} \Rightarrow \begin{array}{r} (-9) \\ +\ 5 \\ \hline -\ 4 \end{array}$$

$$\begin{array}{r} \text{POS} \\ +\ \text{NEG} \\ \hline \text{POS} \end{array} \Rightarrow \begin{array}{r} 9 \\ +\ (-5) \\ \hline 4 \end{array} \qquad \begin{array}{r} \text{NEG} \\ +\ \text{NEG} \\ \hline \text{NEG} \end{array} \Rightarrow \begin{array}{r} (-9) \\ +\ (-5) \\ \hline -\ 14 \end{array}$$

1/4 POS + POS → POS Answer

If no 2's complement is needed, use regular binary addition.

$$\begin{array}{rcr} 9 & \longrightarrow & 00001001 \\ +\ \ 5 & \longrightarrow & +\ 00000101 \\ \hline 14 & \longleftarrow & 00001110 \end{array}$$

2/4 POS + NEG → POS Answer

Take the 2’s complement of the negative number and use regular binary addition.

$$\begin{array}{r} 9 \\ +\ (-5) \\ \hline 4 \end{array} \qquad \begin{array}{r} 00001001 \\ +\ 11111011 \\ \hline \cancel{1}]\,00000100 \end{array}$$

8th Bit = 0: Answer is Positive
Disregard 9th Bit

$$\left.\begin{array}{r} 00000101 \\ \downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow \\ 11111010 \\ +1 \\ \hline 11111011 \end{array}\right\}$$

2’s Complement Process

3/4 NEG + POS → NEG Answer

Take the 2's complement of the negative number and use regular binary addition.

```
  (-9)          11110111
+   5         + 00000101
------        ----------
  -4            11111100
```

8th Bit = 1: Answer is Negative

To Check: Perform 2's Complement On Answer

```
 11111100
 ↓↓↓↓↓↓↓↓
 00000011
       +1
---------
 00000100
```

```
 00001001
 ↓↓↓↓↓↓↓↓
 11110110
       +1
---------
 11110111
```

2's Complement Process

4/4 NEG + NEG → NEG Answer

Take the 2’s complement of both negative numbers and use regular binary addition.

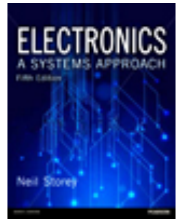
(-9)	→	11110111
+ (-5)	→	+ 11111011
$\boxed{-14}$	←	~~1~~] 11110010

2’s Complement Numbers, See Conversion Process In Previous Slides

8^{th} Bit = 1: Answer is Negative
Disregard 9^{th} Bit

To Check: Perform 2’s Complement On Answer

11110010
↓↓↓↓↓↓↓↓
00001101
+1
00001110

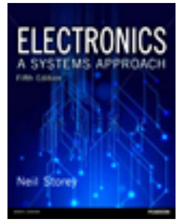


A Systems Approach to Engineering

Video 11A 11.2

- Engineering systems are often very complex
- One approach is to adopt a **systematic approach**
  - complex systems divided into a number of elements
  - a top-down approach
  - a ‘reductionist’ view
  - assumes that a system is no more than the sum of its parts
  - however, some system features relate to the interaction of many system elements
    - e.g. the ‘ride’ or ‘feel’ of a car is not determined by one part

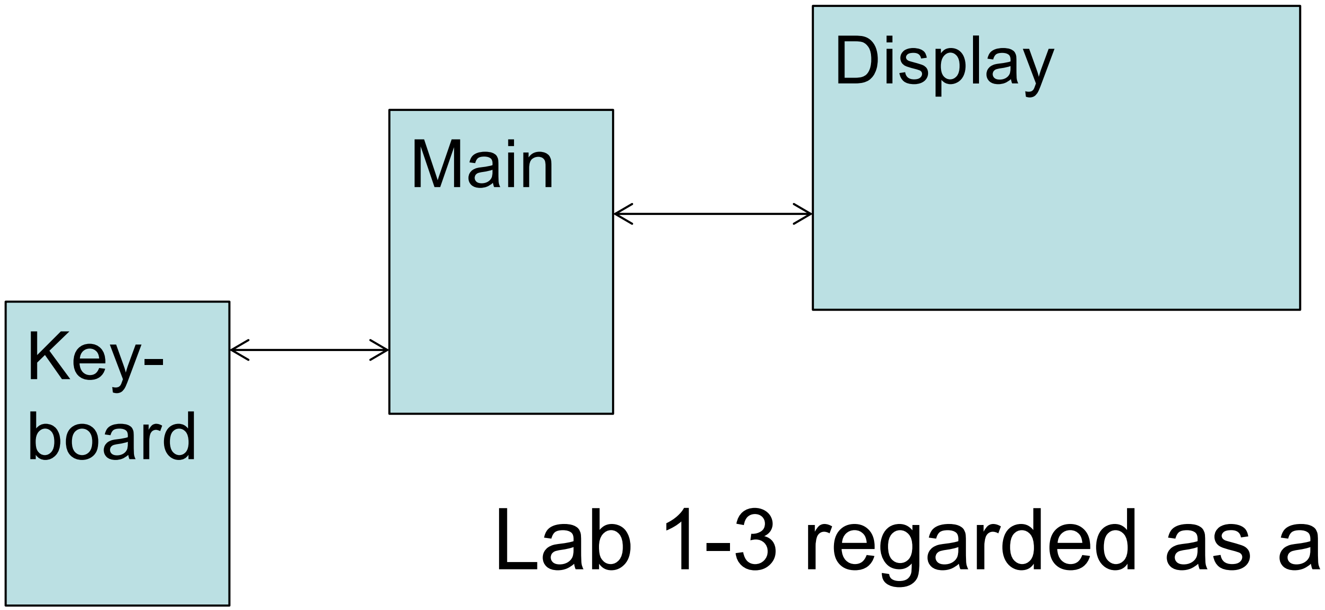
IS4.22



11.6

System Block Diagrams

- It is often convenient to represent complex arrangements by a simplified block diagram



Lab 1-3 regarded as a system

Subsystem and Interfaces

- We can se the Keyboard HW and subroutine (from Lab1) as one subsystem.
- The **Interface** is the subroutine call (no parameters in, but a return value in R24 – RVAL)
- The Display routines created in Lab 2 can also be regarded as a subsystem.
- Lab 3 will tie all these together…

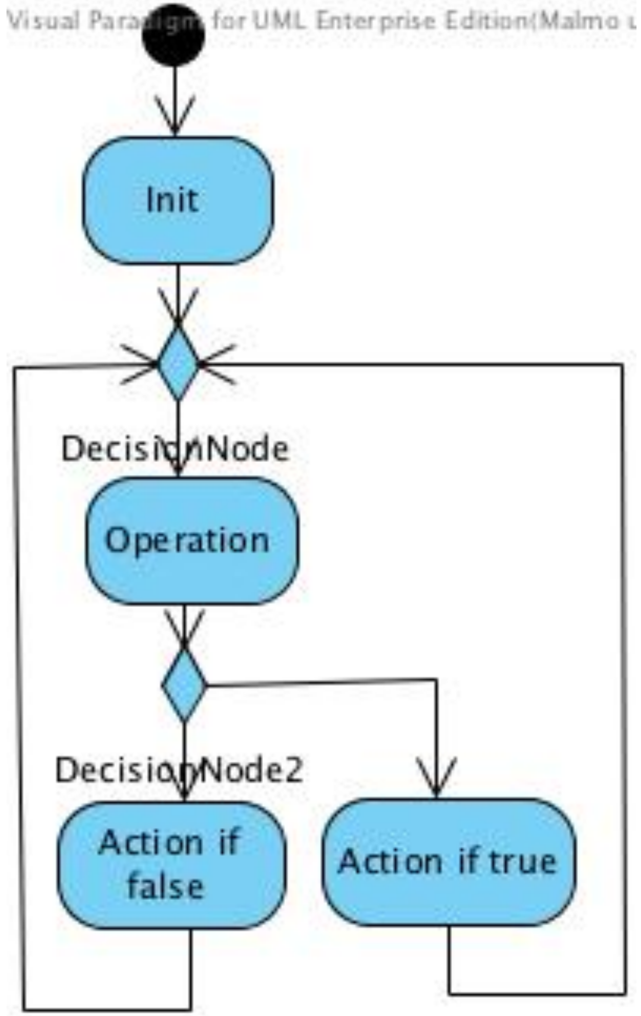
Methodology – a way to think…

- The “wise-guy” optimises every piece of code that he/she produces
- He/She is praised in the department for always producing effective code
- However, it takes many hours….

- The “wise” guy starts with simple/”brute force” solutions
- He/She then **measures** in order to find the places where optimisation is needed and optimises there.
- Less hours in total are spent and most of the code is easy to read

UML documentation

- Assembler programs can be difficult to read and understand….
- Use UML diagrams to document the code, e.g.:
- Or use Pseudo Code (as in Lab manuals…)



The use of RJMP vs RCALL

- When implementing the main program, or a sub-menu, a loop can be created, using RJMP.
- Each menu selection should correspond to a subroutine call RCALL.
  - The functionality can be placed in a separate file (using .include)
- Returning from the subroutines are done through RET, not RJMP!

HIGH and LOW

- Access High and Low part of operand

```
LDI     R20, LOW(0x1234)          LDI     R20, $34
LDI     R21, HIGH(0x1234)         LDI     R21, $12
```

$1234

HIGH LOW

```
LDI     R20, LOW(-200)            LDI     R20, $38
LDI     R21, HIGH(-200)           LDI     R21, $FF
```

-200 = $FF38

HIGH LOW

Animation: slide 23

Initializing the Stack-pointer

Internal SRAM	Size	ISRAM size	2,5K bytes
	Start Address	ISRAM start	0x100
	End Address	ISRAM end	0x0AFF

```
LDI     R16,   HIGH(RAMEND)
OUT     SPH,   R16
LDI     R16,   LOW(RAMEND)
OUT     SPL,   R16
```

“The first 2,816 Data Memory locations address both the Register File, the I/O Memory, Extended I/O Memory, and the internal data SRAM. The first 32 locations address the Register file, the next 64 location the standard I/O Memory, then 160 locations of Extended I/O memory and the next 2,560 locations address the internal data SRAM”.

Recall Timing calculations…

<prev section removed>

```
        LDI     R16,  50    ;this is in the main program
        RCALL   wait
.
.
wait:   NOP
        DEC   R16
        BRNE wait
        RET
```

RCALL	takes 4 cycles
NOP	takes 1 cycle
DEC	takes 1 cycle
BRNE	takes 2 cycles if branching, 1 cycle if not
RET	takes 4 cycles
RJMP	takes 2 cycles

Example roll_dice (part of lab 3)

```
; Tarning.inc
; R16 contains the dice value on return
roll_dice:
            LDI    R16, 6     ;dice have 6 values
test:       NOP
            NOP
            RCALL read_keyboard      ;key-value in RVAL
            CPI RVAL, ROLL_KEY
            BREQ roll         ;yes, key 1 is still pressed
            RET               ;no, key is released
roll:       DEC    R16        ;start cycle count here
            BREQ roll_dice ;R16 is zero?, start agn at 6
            RJMP test         ;no, keep rolling
```

Example Dice

- RCALL to read_keyboard takes a large number of cycles, but the same for all iterations of the dice!
- ‘BREQ roll’ will branch as long as the key is pressed - 2 cycles and code then continues at ‘roll:’
- So, each round (“number”) takes 8 cycles (9 with DEC):
  - 6: 6 cycles before DEC (assume BREQ roll jumps…)
  - 5: 8 cycles before DEC (assume BREQ roll jumps…)
  - 4: 8 cycles before DEC (assume BREQ roll jumps…)
  - 3: 8 cycles before DEC (assume BREQ roll jumps…)
  - 2: 8 cycles before DEC (assume BREQ roll jumps…)
  - 1: 8 cycles before DEC (assume BREQ roll jumps…)
  - 0: 2 cycles before it becomes 6 and is then tested as 6….

Using Registers – can be difficult in ASM…

- Subroutines uses Registers (some use many, some use few)
- Either, the comments (before the subroutine), describe which registers are used (read as input, used for output and used (previous content = destroyed) internally
- Or, registers used internally are saved on the stack (using PUSH) when going into the subroutine and restored (using POP) before returning from the subroutine.

ADC and Addition of 16 bits numbers

- When adding two 16 bit operands, we need to be concerned with the propagation of carry from the lower byte to the higher byte
- This is called multi-byte addition to distinguish it from addition of individual bytes
- The instruction ADC (ADD with Carry) is used on such occasions
- For example, let us see the addition of 0x3CE7+0x3B8D

$$\begin{array}{r} 1 \\ 3C\ E7 \\ \underline{3B\ 8D} \\ 78\ 74 \end{array}$$

- Assume that **R1** = 8D, **R2** = 3B, **R3** = E7 and **R4** = 3C,

  ADD R3, R1 ; R3 = R3 + R1 = E7 + 8D = 74 and C = 1

  ADC R4, R2 ; R4 = R4 + R2 + Carry = 3C + 3B + 1 = 78