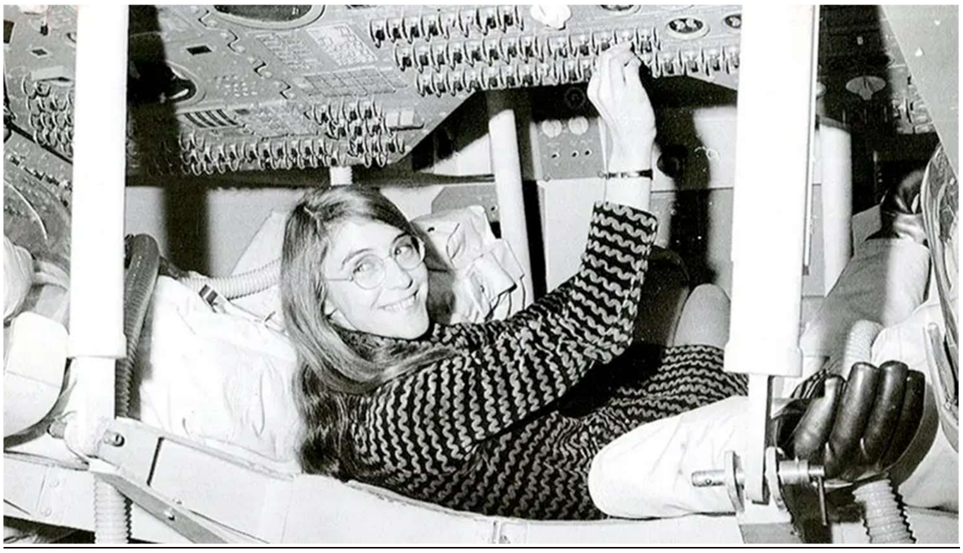
4 Addressing modes



4.1 What have we learned so far

4.1.1 Error with dll registration

STVD Connection error (usb://usb): gdi-error [40201]: can't access configuration database STVD DAO PROBLEM

Run following commands as administrator:

```
Regsvr32 /u "C:\Program Files (x86)\Common Files\Microsoft Shared\DAO\DAO350.DLL"

Regsvr32 "C:\Program Files (x86)\Common Files\Microsoft Shared\DAO\DAO350.DLL"
```

4.1.2 If then else and loops

- Draw flowcharts: they might help to get the logic right.
- Almost every instruction affect the Condition Code (CC) flags:
  - CP sets the Z flag if the operands are equal
  - INC and DEC set the Z flag if the result is 0
  - Check the programmers manual to find out what your code does with the CC flags.
  - The state of a flag is only valid till the next instruction is executed.

- The JRxx instructions check the appropriate CC flag. For instance JREQ jumps if the Z bit is set. JRNE jumps if the Z bit is cleared.

4.1.3 Digital I/O

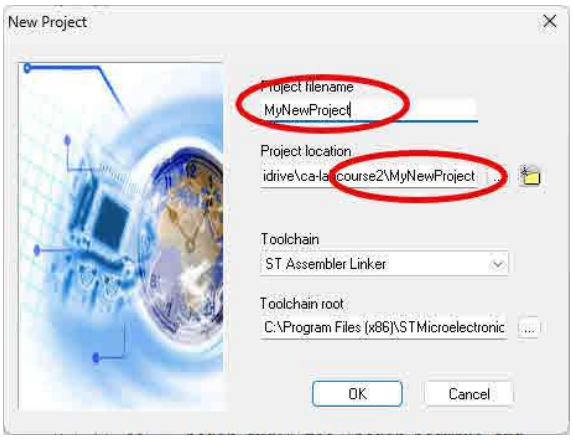
- 5 registers:
  - DDR
  - IDR (for inputs)
  - ODR (for outputs)
  - CR1
  - CR2

4.1.4 Set up a workspace

When you set up your own workspaces and projects, STVD is not very helpful in choosing the right directories.

So

- Create your workspace somewhere in a place where Windows allows you to. This means: not somewhere in c:\program files
- New projects must go in a new folder



Project filename: MyNewProject

Project location: idrive\ca-la course2\MyNewProject

Toolchain: ST Assembler Linker

Toolchain root: C:\Program Files (x86)\STMicroelectronic

4.1.5 TIM2

We always use following config for TIM2, although it has much more possibilities:

```
MOV  TIM2_CR1,#%00000001 ; counter enable ON
MOV  TIM2_IER,#$00           ; no interrupts
MOV  TIM2_CCMR1,#%01100000    ; PWM mode 1 + CC1 as output

MOV  TIM2_CCER1,#%00000001    ; enable CC1 output
```

In this configuration, TIM2 works as a PWM modulator. If CC1 is enabled, it is directly connected to PD4.

We can turn the timer on or off with bit 0 in TIM2_CR1.

The frequency is configured in TIM2_ARR:

```
ldw x, #12000
ld a, xh
ld TIM2_ARRH, a
ld a, xl
ld TIM2_ARRL, a
```

The dutycyle is configured in TIM2_CCR1

```
ldw x, #3000
ld a, xh
ld TIM2_CCR1H, a
ld a, xl
ld TIM2_CCR1L, a
```

4.1.6 TIM3

TIM3 is configured to generate a period interrupt.

```
MOV  TIM3_CR1,#%00000001 ; timer on
MOV  TIM3_PSCR,#$07 ; prescaler x128
BSET TIM3_EGR,#0     ; force UEV to update prescaler
MOV  TIM3_IER,#$01   ; TIM3 interrupt on update
enabled
```

The period can be configured by supplying values to TIM3_ARRH and TIM3_ARRL. Eventually the value of the prescaler must be changed as well.

Do not forget to put

```
BRES TIM3_SR1,#0 ; clear flag
```

as last instruction on your ISR.

4.1 Addressing modes

4.1.1 Immediate addressing

```
LD A, #1
```

4.1.2 Direct addressing

```
LD A, 1

         Or

    Segment ram0
Count ds.b
    Segment rom
…
…
    LD A, Count
```

4.1.3Indexed addressing

=> retrieve data at a **base address + an offset.**

eg: base = 0x0800, index = 0x10, indexed addressing would retrieve data at address 0x8010

example application: **arrays**.

- **Name of the array equals the address of the array.**
- **Indexed values are at the base address + index.**
- **Example:**
  - **If the array myValues is at address 0x13**
  - **myValues[0] is at address 0x13**
  - **myValues[1] is at address 0x14**
  - **myValues[5] is at address 0x18**
- **Red flag: assembler is stupid and does not know how long your array is.**

- **Red flag 2: if myValues is an array of words**
  - **myValues[0] is at address 0x13**
  - **myValues[1] is at address 0x15**
  - **myValues[5] is at address 0x1D**
  - **…**
- **Red flag 3: the C language (3th year,operating systems) is stupid and does not know how long your array is. However, C knows the difference between bytes, integer and long integer, …**

assembler:

```
          segment ‘ram0’
myValues  ds.b 10 ; allocates 10 bytes for array myValues


          LDW  X, #5
          LD A, (myValues,X)  ; loads myValues[5] in the
     accumulator

     INCW X
     LD a, #10
     LD (myValues,X), a ; loads the accumulator to myValues[6]
```

- **Notes:**
  - NO SPACES in brackets E.G. LD A, (tones,X)!
  - A strange bug is that the text editor doesn’t allow you to write the left squared bracket on a Belgian period AZERTY keyboard. An easy workaround is to copy paste it from another location/file. Alt 91 also works. (alt 93 =])
  - X is a 16 bit register
- More info can be found in the Programming Manual.

4.1.4 Exercises

Exercise 1: set up a new workspace and project

Exercise 2: arrays part 1

- Allocate 8 bytes in ram1 segment. (array1 ds.b 8)
- Write a loop that is executed **only once** that fills this ram locations with all powers of 2 under 256 (1, 2, 4, ..., 128)
    - The base address is the label that you assigned to your array
    - X or Y is your loop counter
    - Use incw and the CC flags to control your loop.
    - Carefully study the SLL instruction. How does it affect the bits in CC? Which JRxx instruction is the most beneficial for you?
    -

```
Skeleton:
          segment ‘ram0’
myArray   ds.b 8
counter   ds.b

          segment ‘rom’
main
```

```
          …
          …
          ldw x, #0 ;start at index 0
          ld a, #1       ; initialize a

          … do something with a to get the next
number
          …
          ld (myarray,x) , a    ; store the contents of
a at address myaddress + x
               in this case this is myaddress[0]
          incw x
          …
```

Exercise 3 Arrays part 2

Do the same exercise, but now with an array of words instead of bytes.

Exercise 4 Arrays part 3

- Allocate an array in the rom segment that contains all the bit patterns that occur in the Knight Rider (3, 6 12, 24 …).
    - ○ Do not forget to put a label!
    - ○ Use dc.b directive!
- Allocate a variable that holds the number of values in that array.
- Write a program that is executed repeatedly. Every 500 ms it should read a value from you array (index 0 … n where n-1 is the number of values in your array. The value read must be copied immediately to PD_ODR.
- Do not forget to configure port D as outputs, push-pull.

Exercise 5 Arrays part 3: play a scale

- We will use TIM2 as tone synthesizer. Configure D4 as push-pull output and place the jumper to connect to the speaker.
- This table holds the frequencies of a scale over 1 octave.
- Calculate the ARR value for TIM2 for each frequency.
- Your program can calculate the duty cycle (divide by 2 = shift right)
- Program the 8 values in table in the rom segment. Label this array ‘pitch’.

```
                    segment ‘rom’
        pitch dc.w 880, 988, ….
```

- Write a program that plays the scale 1 time. Every note should play for 500 ms. The scale should play A -> A’, G -> A.

tone	f	counts	hex

A	880
B	988
C	1047
D	1175
E	1319
F	1397
G	1568
A'	1760

Exercise 6: play a tune.

- Make an array in the rom segment met following values:
  2-3-4-2-4-5-6-6
- Label this array ‘song’
- Program a loop that uses these values as indexes to access the array ‘pitch’ Play that note for 500 ms.
- Play the tune once.
- Extension: after playing it once, wait 5 seconds and play it again.

Exercise 7: PWM

- Configure TIM2 for 100 Hz
- Set the jumper to D4
- Watch LED D4 for different values of the duty cycle : 10%, 20%, 50%, 100% do you notice a difference in brightness?