Micro Controllers Summary

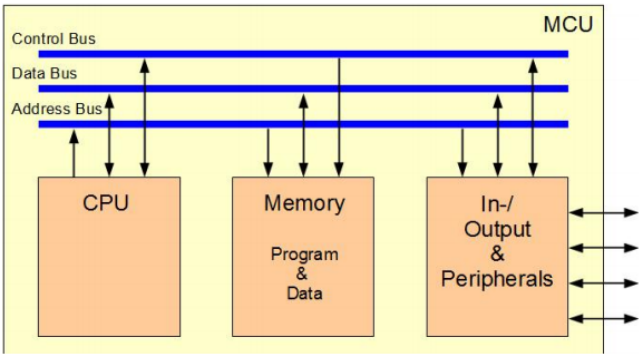
Lucien Zürcher

June 16, 2019

Contents

1 System Components

1.1 Von Neumann Architecture



Components:

- **CPU**, Central Processing Unit
- **Memory**, Program and Data
- **In-/Output**-Unit, Peripherals
- **Bus**-System: Communication

One ***shared bus and memory*** *for program and data.*

1.2 Harvard-Architecture

basically same as Von Neumann, with the difference, that there are ***two separate bus systems*** *for program and data*

1.3 Numerical Systems

Numerical value Z_B of a n-digit, integer number with base B ($B \geq 2$):

$$Z_B = \sum_{i=0}^{n-1} x_i \cdot B^i$$

Decimal	**Dual / Binary**	**Hexadecimal**
197	0b1100'0101	0xC5
$B = 10$	$B = 2$	$B = 16$
$= 1 \cdot 10^2 +$ $9 \cdot 10^1 +$ $7 \cdot 10^0$	$= 1 \cdot 2^7 + 1 \cdot 2^6 +$ $0 \cdot 2^5 + 0 \cdot 2^4 +$ $0 \cdot 2^3 + 1 \cdot 2^2 +$ $0 \cdot 2^1 + 1 \cdot 2^0$	$= C \cdot 16^1 + 5 \cdot 16^0$ $= 12 \cdot 16^1 + 5 \cdot 16^0$

The amount of presentable numbers is B^n The highest presentable number is $B^n - 1$. Calculated from $x_i = B - 1$ for $n - 1 \geq i \geq 0$

1.4 hex / binary

H	D	B	Dec	Bin	
0	0	0000	16	2^5	(max 31)
1	1	0001	32	2^6	(max 63)
2	2	0010	64	2^7	(max 127)
3	3	0100	128	2^8	(max 255)
4	4	0101	256	2^9	(max 511)
5	5	0110	512	2^{10}	(max 1'023)
6	6	0111	1'024	2^{11}	(max 2'047)
7	7	1000	2'048	2^{12}	(max 4'095)
9	9	1001	4'096	2^{13}	(max 8'191)
A	10	1010	8'192	2^{14}	(max 16'383)
B	11	1011	16'384	2^{15}	(max 31'767)
C	12	1110	32'768	2^{16}	(max 65'535)
D	13	1011			
E	14	1011			
F	15	1011			

1.5 Signed numbers

two's compliment is beeing used

$$Z_{signed} = -x_{n-1} \cdot 2^{n-1} + \sum_{i=0}^{n-2} x_i \cdot 2^i$$

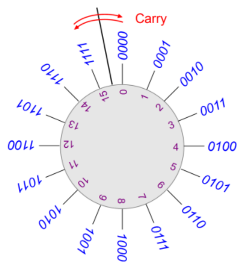
most significant bit is negative

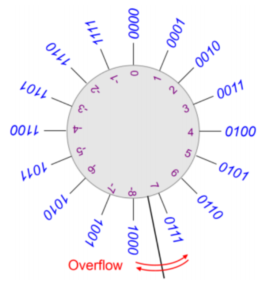
Example: -1 as 16-bit Hex = $0xFFFF$

Conversion:

1. *Invert binary :* $-6 \rightarrow 0110 \rightarrow 1001$
2. *increment by 1 :* $1001 + 0001 \rightarrow 1010$

1.6 carry / overflow





Carry *is set on crossover between lowest and highest number*

Overflow *happens on crossover between highest absolut values*

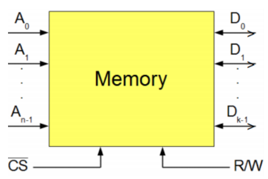
1.7 Bit groups

Nibble/Tetrade *has the size of 4 bits*

Byte *has the size of 8 bits*

Word *is MC9S08JM60 specific, it has 16 bits*

1.8 Quantity of address lines



n *Quantity of address lines*
$\mathbf{2^n}$ *Quant. of storage locations*
$\mathbf{2^n \cdot k}$ *Quant. of bits in memory*

1 K	=	2^{10}	=	1024 Bit	≙	10 Adresslines
64 K	=	2^{16}	=	65536 Bit	≙	16 Adresslines

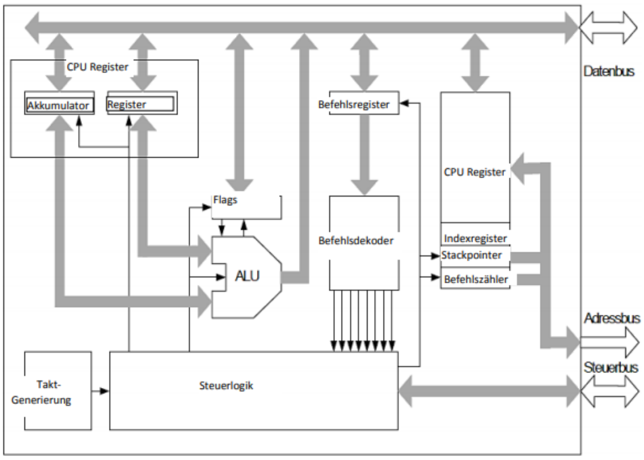
example, $32K \times 8$ memory storage space:

bits storage: $32 \cdot 2^10 \cdot 8 = 2^5 \cdot 2^10 \cdot 2^3 = 2^18 \rightarrow 18$ *Bits*
number address lines: $32 \cdot 2^10 = 2^15 = 32\,768$
highest address: $2^{18} - 1 = 0x7FFFF = 262'143$

1.9 Microprocessor vs Mircocontroller

Mircocontroller *contains CPU (Processor), Peripherals (I/O) and Memory (RAM / ROM). Basically a small computer.*
Mircoprocessor *has only CPU and som integrated Circuits.*

1.10 CPU components



ALU (Aritmetic Unit), AKKU (Accumulator), PC (Programming Counter), Busses, Instruction-Register, Address-Register, Operand-Register, Control Unit, ..

1.11 Instruction Cycle Steps

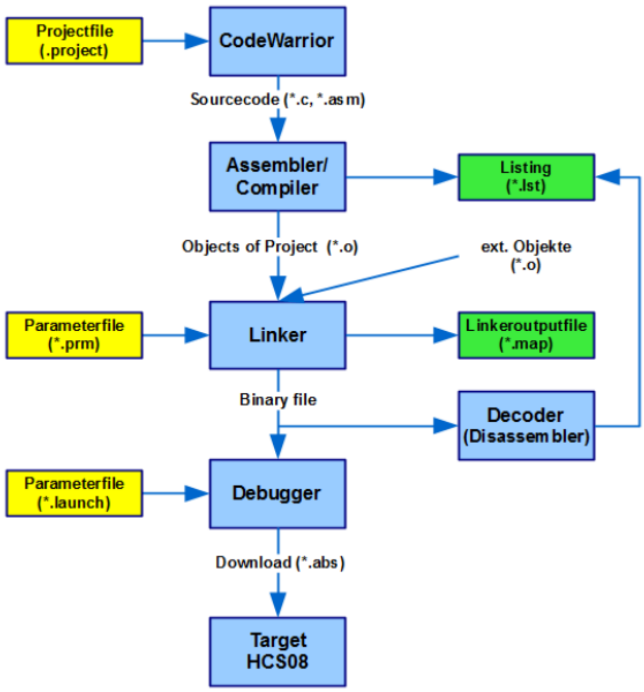
1. instruction fetch
2. instruction decode
3. (operand fetch)
4. instruction execute
5. next address and inc PC

1.12 Types of MCU Registers

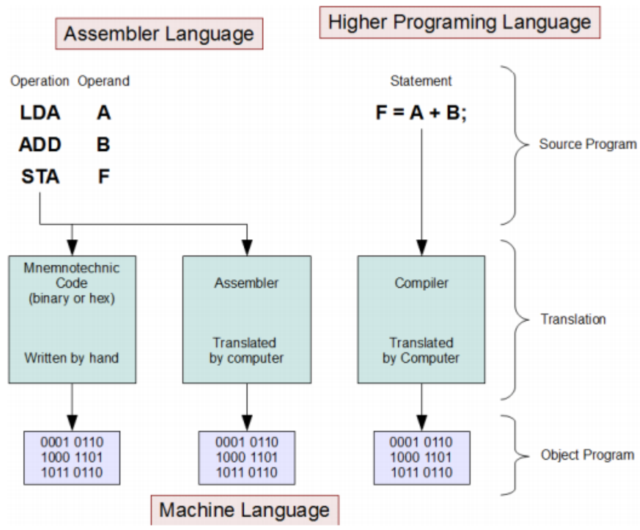
AKKU, PC, Instruction-Register (decoder), Operand-Register

2 Compiling

2.1 Codewarrior Designflow



2.2 Programming Language



High level programming languages are:

- portable
- efficient (normaly)
- Better readable
- easier to maintain

High level programming languages are usually prefered, if enough computational power and memory is available. Assembler is often used, if the application:

- is time critical and needs exact timing
- timing of the high level programming language to unpredictible is

2.3 Assembler Code-Format

	Label	Instruction	Operands	comment
Ex1	Limit:	EQU	$CD	; define limit
Ex2	Start:	LDA	#Limit	; load limit

***Instruction**: is a command for the processor*
***Directive**: are instructions that direct the assembler / compiler to do something*

	Type	Directed to	Results in program code
Ex1	**Instruction**	Target CPU	Yes
Ex2	**Directive**	Assembler	Only indirect
	Comment	Programmer	No

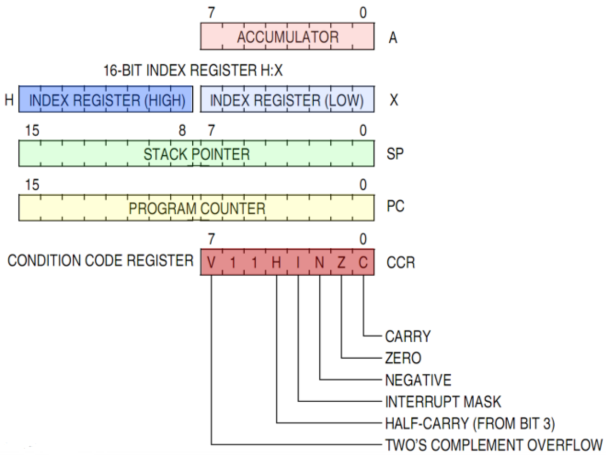
2.4 Parameter file

The Parameter file (.prm) is used for by the Linker. It takes the machine code and defines the location on the controller. It is important, so that jumps work correctly. It contains:*

- Memory-Map of the Prozessor (Location and size of Flash, RAM, ..)
- Extra definitions, where which parts of the code on the Controller should be located

3 Assembler & HCS08

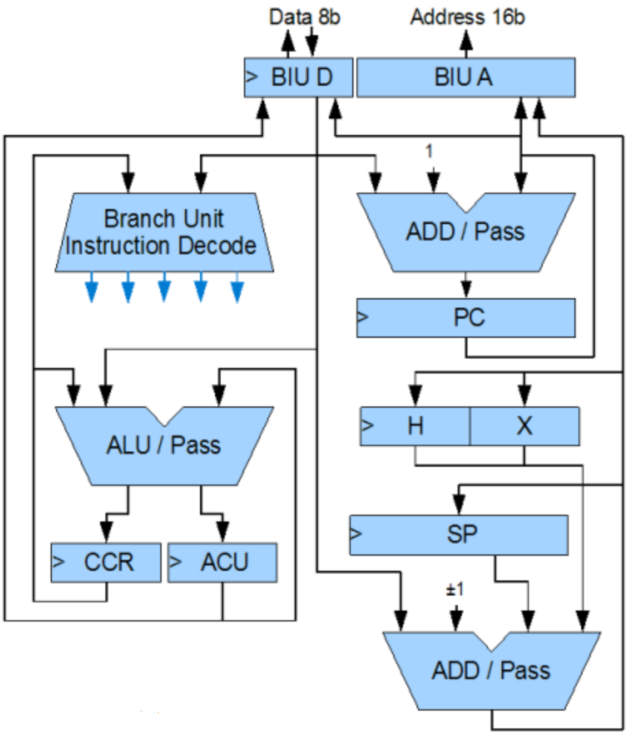
3.1 HCS08 CPU Registers



Registers the HCS08 contains:

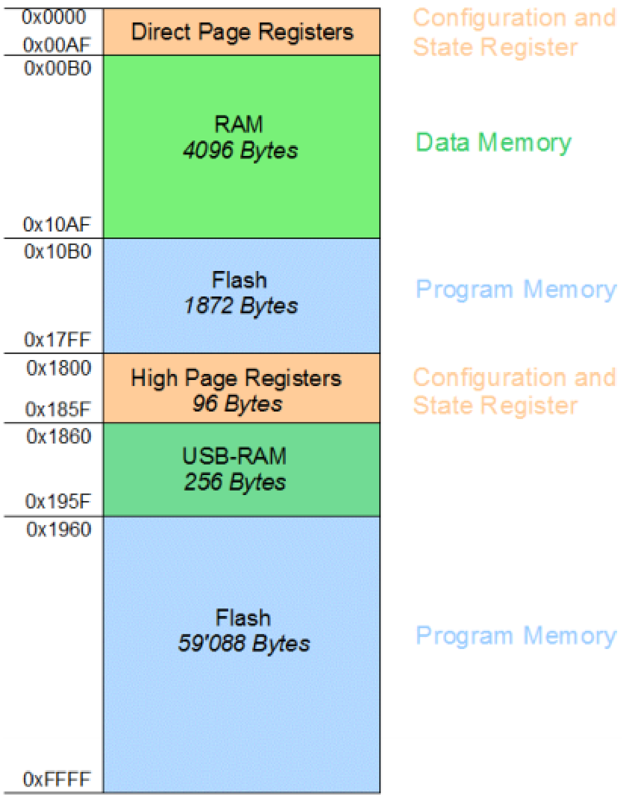
- HX Register
- PC
- Akku
- Stack Pointer
- CCR

3.2 HCS08 Processor



- 8 Bit, Von Neumann archidecture
- **BIU** Bus Interface Unit
- **PC** Program Counter
- **ACU** Accumulator
- **ALU** Arithmetic Logic Unit
- **CCR** Condition Code Register (Collection of status flags)
- **SP** Stack (LI-FO, Pointer for Context and Parameter)
- **H:X** Index Register

3.3 Memory Mapping



Access to the directpage (0x0000 - 0x0AF) needs less cycles, since the address is only 1 Bytes long.

3.4 Register configuration HCS08

```
// define the dataflow direction input = 0 |
    output = 1
PTADD = 0x04;

// set output value
PTAD = 0x04;
// read value
uint_8 val = PTAD;

// set pullup enable port
PTADD = 0x00;
PTAPE = 0x04;
```

Reg. Name	Description
PTxDD	*Data Direction of Port x*
PTxD	*Data value of Port x*
PTxPE	*Set Pullup Enable of Port x (PTxDD needs to be 0)*

***Pullup Enable** is used to pullup the value of the output to 1. This is usually used on a bus system to prevent a short circuit.*

3.5 Differences of Operations

Comparing different operations, following should be taken in consideration:

- number of cycles
- memory usage, 8bit (directpage) / 16bit
- Set CCR bits / flags
- Used registers
- Address modes

4 Assembler Directives & Addressing Modes

4.1 Directives

Directive	Description
SECTION	*Defines the beginning of a relocatable section*
EQU	*Assigns an expression to a name. Not redefinable*
DC	*Defines one or more constants and their names. Will be stored at the set location*
DS	*Allocates memory(RAM) for variables*

The Assembler-Directive SECTION *defines program- and data section. Those section can be moved freely within the memory (relocative assembling), **after** the **assembly** process is finished.*
*The final memory area location happens after the linking process. The locations of those sections can therefor be defined in the **Linker-Parameterfile**.*

4.2 Basic Assembler Program

```
; include definitions
include 'MC9S08JM60.inc'

; -- globals
GLOBAL _Startup ; define start of programm
GLOBAL main
GLOBAL dummy     ; Dummy Interrupt Service
    Routine

; -- equations
StackSize: EQU   $60   ; stack size
pi:        EQU   31416 ; example of random equ

; -- stack
DATA_STACK: SECTION
TofStack:  DS     StackSize-1 ; definiton of "
    Top of Stack"
BofStack:  DS     1           ; definition of "
    Bottom of Stack"

; -- create space for data
DATA:   SECTION
var1:    DS     1    ; Example of a 1 Byte
    Variable
Array1: DS    $20 ; Example of an Array of $20
    Bytes

; -- setup constants
CONST:      SECTION
Maske1:       DC.B     %00000001
Parameter1:  DC.B     $3A     ; DC with a point
Parameter2:  DC.W     57100   ; word with int
    value
Reserve_Par: DS       16      ; reserve empty 16
    Bytes
VarArray:     DS.W     3       ; reserve 3 Words
STRING1:      DC.B     10,"Hello",$0D

; -- program start (initialisation)
PROGRAMM:   SECTION ; Code Segment
```

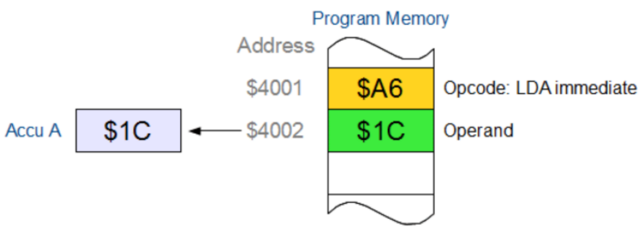
```
_Startup:           ; Resetvektor points to
    this
Stackinit:  LDHX   #(BofStack+1)
            TXS          ; decrement TXS, thats
                 why +1 BofStack
            LDA    #$00
            STA    SOPT1  ; Disable Watchdog

; -- actual program
main:
    ; turn on backligths of the car
    BSET     PTDD_PTDD2, PTDD
    BSET     PTDDD_PTDDD2, PTDDD

    CLR      RamLoc

    BCLR     PTGDD_PTGDD0, PTGDD
    BCLR     PTGDD_PTGDD1, PTGDD
    BCLR     PTGDD_PTGDD2, PTGDD
EndlessLoop:
    ; load joystick values
    MOV      RamLoc, PTGD
    JMP      EndlessLoop

; (=ensure program end if endlessloop is
    missing)
EndLoop:    BRA      *

; catch any unexpected interrupts
dummy:          BGND
                BRA      dummy
```

4.3 Addressing Modes

- ***Immediate***: *1 Byte operand in instruction (LDA #$01)*
- ***Inherent***: *no operand required (e.g. NOP, INCA..)*
- ***Direct***: *onlu direct page, 1 address Byte*
- ***Extended***: *whole 64k area, 2 address Bytes*
- ***Indexed***: *with SP (Stack pointer) or HX (7 sub modes)*
- ***Relative***: *for branches, PC=PC+2+two's compl.*

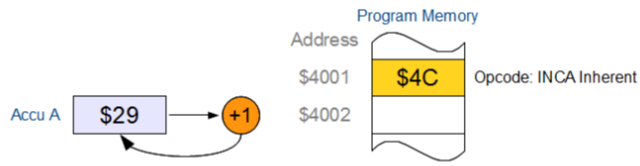
Different addressing modes of the same instruction type use differnet operation codes (e.g. LDA-MM: A6; LDA-DIR: B6).

4.3.1 Immediate (IMM)



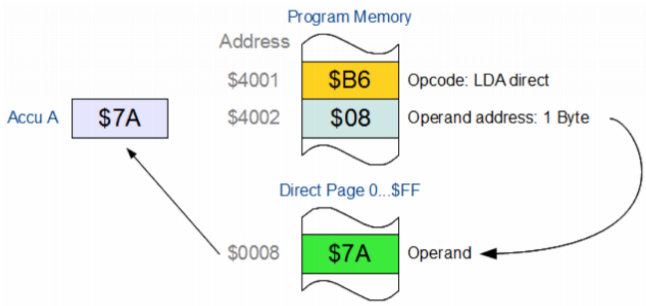
Immediat adressing *mode: the following Byte of the operation code is immediately used as the operand.*
Example: **LDA#$1C**

4.3.2 Inherent (INH)



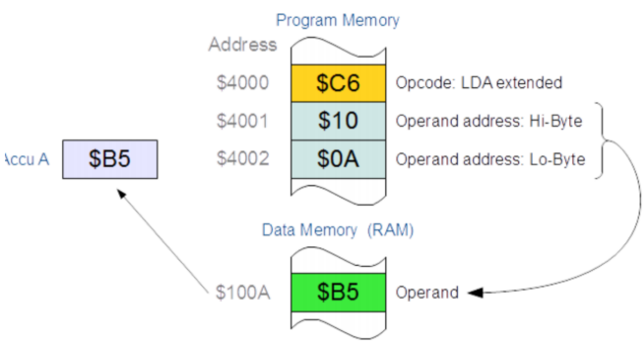
Inherent addressing *mode: no explicit operand address needed. All operands are in the CPU-registers*
Example: **INCA**

4.3.3 Direct (DIR)



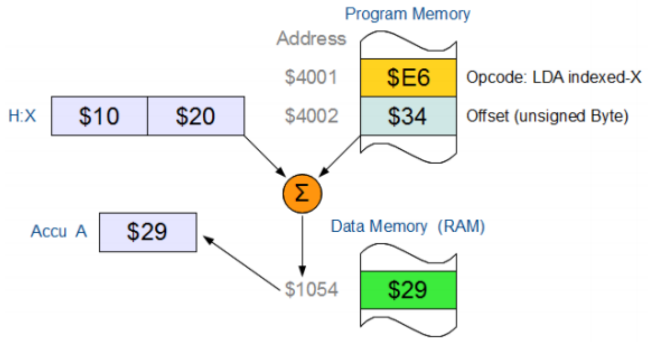
Direct addressing *mode: After the operation code, the* ***1-Byte*** *operand address follows in the program memory. Only operands in the address section between $00 and $FF are supported. (The Direct Page Registers 0x00-0xAF, Direct Page RAM 0xB0-0xFF)*
Example: **LDA$08**

4.3.4 Extended (EXT)



Extended addressing *mode: After the operation code, the* ***2-Byte*** *operand address follows in the program memory.*
Supports the whole address section between 0x0000 - 0xFFFF. But is also slower.
Example: **LDA$34, X**

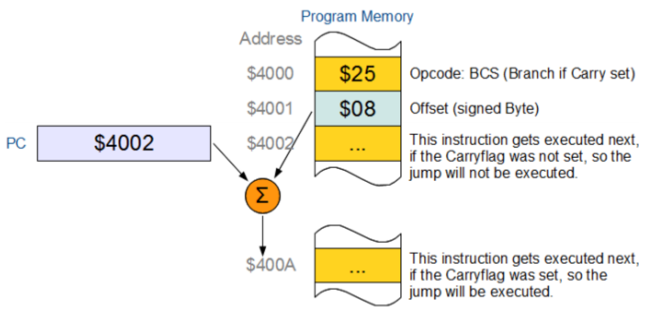
4.3.5 Indexed (IX1)



Indexed addressing *mode: uses the **HX** or **SP** register.*
Through indexed addressing the final assigned operand address is dependent from the program behaviour (address arithmetics).
Following are sub modes of the indexed addressing mode

IX	*Indexed addressing with H:X, without offset*	**LDA X**
IX1	*Indexed addressing with H:X and **8-bit offset***	**LDA $34, X**
IX2	*Indexed addressing with H:X and **16-bit offset***	**LDA $34A5, X**
IX+	*Indexed addressing with **H:X** and **H:X Increment**. Only for MOV and CBEQ (Compare Accu with value on the address that is stored in the H:X register. If values are equal, jump to Label and increment H:X) instructions*	**CBEQ X+, Label**
IX1+	*Same as IX+, with **Increment** and **8-bit offset** (Only available for instruction CBEQ)*	**CBEQ $34,X+, Label**
SP1	*Same as IX1, but with Stackpointer SP instead of H:X.*	**LDA $34, SP**
SP2	*Same as IX2, but with Stackpointer SP instead of H:X.*	**LDA $34A5, SP**

4.3.6 Relative (REL)



PC relative addressing *mode: is only used with BRANCH-Instructions.*
*The following Byte after the operand is a **two's complement** offset to the already increased program counter.*
The address range with relaive addressing is -126 to +129