The EuroOne v0.005 Manual

Maximilian Noppel

7. Februar 2020

The following document is WORK IN PROGRESS!

It describes the EuroOne-CPU. The CPU is currently simulated via logisim but should be realized in a much smaller and simpler version in some years. This document is about the entire project. Beside the logical and hardware parts, it is also containing a bunch of information about the suggested machinecode, a suggested assembler slang and also about a higher language implementation called EuroLang. I also may implement a C or C++ compiler some times. This will also be included in this document.

Most of the tables and list are generated from one big JSON file, from which also the microcode and the assemblerparser is generated. This means, the tables should reflect a specific JSON file from which one can generate microcode for the CPU. The purpose of this document is mainly for to look things up when developing the CPU but it also gives everybody the chance to implement her own microcode or assemblerparser. The CPU is designed to be super flexibel. So feel free to write your own instruction set from this. Especially the router is a very powerful tool for the instruction set because you can route nearly everything to everywhere.

Inhaltsverzeichnis

1 Features

- 8-Bit Data and 16-Bit Addressbus
- 4x General Purpose 8-Bit-Registers (Combine any two to get 16-Bit-Registers)
- 32kB of RAM
- StackPointer Register
- BasePointer Register
- ProgramCounter Register
- Peripherie via Memory-Mapped-IO:
  - TTY Interface
  - 8x8 LED-Matrix with double framebuffer
  - 1x 16-Bit-Timer
  - 1x 4-Bit-GPIO
- Conditional Jumps: `jz`,`jnz`,`jg`,`jge`,`jl`,`jle`,`je`
- Stackframeoperations: `call`,`ret`,`pop`,`push`
- Dynamic Segmented Adressing: `storerrao`, `loadraor`
- Microprogramming by DiodeMatrix
- 16-Bit muliplication result
- 16-Bit division input
- Seperate ProgramCounter, StackPointer and BasePointer for the interrupt mode
- Arithmetics: `add`,`sub`,`imul`,`idiv`,`and`,`or`,`xor`,`not`,`neg`

2 Introduction

3 Logic

3.1 Registers

The cpu is using four 8-bit general purpose register which can be used as 16-bit registers in any combination. So the memory of register `ra` and `rab` overlaps. Also it got some special purpose register `rpc`, `rsp` and `rbp`. Which are used as ProgramCounter, StackPointer and BasePointer. These are 16-bit registers as the address bus is 16 bit. You could use them as normal register, but as they serve some special function you value may get changed arbitrarily. You may want to set the StackPointer to the beginning of your stack for normal mode and also for the interrupt mode. In the interrupt mode those three special purpose register are mirrored. The general purpose register `a-b` will get save automatically to the stack of the normal mode if an interrupt occurs.

Also there are the register containing const values. These are mostly for internal use, but you could use them as constants to.

Shortname	Name	Type	Width	Id
`r00`	Constant Zero Register	for internal use	8bit	`0x00`
`r01`	Constant One Register	for internal use	8bit	`0x01`
`r02`	Constant Two Register	for internal use	8bit	`0x02`
`r03`	Constant Three Register	for internal use	8bit	`0x03`
`ra`	A Register	GeneralPurpose	8bit	`0x04`
`rb`	B Register	GeneralPurpose	8bit	`0x05`
`rc`	C Register	GeneralPurpose	8bit	`0x06`
`rd`	D Register	GeneralPurpose	8bit	`0x07`
`rpc`	ProgrammCounter	SpecialPurpose	16bit	`0x98`
`rsp`	StackPointer	SpecialPurpose	16bit	`0xba`

Continued on next page

Shortname	Name	Type	Width	Id
rbp	BasePointer	SpecialPurpose	16bit	0xdc
rab	AB Register	GeneralPurpose	16bit	0x45
rcd	CD Register	GeneralPurpose	16bit	0x67
rbc	BC Register	GeneralPurpose	16bit	0x56
rda	DA Register	GeneralPurpose	16bit	0x74
rba	BA Register	GeneralPurpose	16bit	0x54
rcb	CB Register	GeneralPurpose	16bit	0x65
rdc	DC Register	GeneralPurpose	16bit	0x74
rad	AD Register	GeneralPurpose	16bit	0x47

3.2 ALU

Bit	Name	Function
0	CF	Carry Flag
1	ZF	Zero Flag
2	SF	Sign Flag
3	OF	Overflow Flag

3.3 Control Unit

3.4 Microprogramming

3.4.1 Signals

The cpu is controlled by a bunch of ca. 90 signals. There are flexibal and powerful router implement through which you can route nearly every signal everywhere. For every bus there are 5 select lines to select what should be routed there. Table 3 is listing all possible things to route.

Name	Bit
databus_select_0	0
databus_select_1	1
databus_select_2	2
databus_select_3	3
databus_select_4	4
addressbus_low_select_0	5
addressbus_low_select_1	6
addressbus_low_select_2	7
addressbus_low_select_3	8
addressbus_low_select_4	9
store_select_0	10
store_select_1	11
store_select_2	12
store_select_3	13
store_select_4	14
alu_in0_select_0	15
alu_in0_select_1	16
alu_in0_select_2	17
alu_in0_select_3	18
alu_in0_select_4	19
alu_in1_select_0	20
alu_in1_select_1	21
alu_in1_select_2	22
alu_in1_select_3	23
alu_in1_select_4	24
alu_carry_select_0	25

Continued on next page

Name	Bit
alu_carry_select_1	26
alu_carry_select_2	27
alu_carry_select_3	28
alu_carry_select_4	29
none0	30
none1	31
alu_signed	32
alu_invB	33
none11	34
none12	35
none13	36
alu_none2	37
none2	38
none3	39
none4	40
addressbus_high_select_0	41
addressbus_high_select_1	42
addressbus_high_select_2	43
addressbus_high_select_3	44
addressbus_high_select_4	45
halt	46
reset	47
jmp_if_carry	48
jmp_if_z	49
jmp_if_nz	50
jmp_if_e	51
jmp_if_g	52
jmp_if_l	53
jmp_if_ge	54
jmp_if_le	55
jmp_if_ne	56
alu_op0	57
alu_op1	58
alu_op2	59
alu_op3	60
load_op0	61
load_op1	62
fin	63
addressline0	64
addressline1	65
addressline2	66
addressline3	67
addressline4	68
addressline5	69
addressline6	70
addressline7	71
jumpline0	72
jumpline1	73
jumpline2	74
jumpline3	75
jumpline4	76
jumpline5	77
jumpline6	78
jumpline7	79
addressline8	80
addressline9	81
jumpline8	82

Continued on next page

Name	Bit
jumpline9	83
none14	84
none15	85
interrupt_mode_off	86
interrupt_mode_on	87

3.5 Selectors

Tabelle 3: Selectors

Databus and addressbus, ALU Ins and CarryIn, Store	SelectCode
const 00	0x00
const 01	0x01
const 02	0x02
const 03	0x03
A	0x04
B	0x05
C	0x06
D	0x07
PC_low	0x08
PC_high	0x09
SP_low	0x0A
SP_high	0x0B
BP_low	0x0C
BP_hight	0x0D
CARRY	0x0E
unused	0x0F
reg0l	0x10
reg0h	0x11
reg1l	0x12
reg1h	0x13
reg2l	0x14
reg2h	0x15
unused	0x16
unused	0x17
op0	0x18
op1	0x19
op2	0x1A
RAM	0x1B
alu_result	0x1C
alu_mulcarry	0x1D
alu_flags	0x1E
FLAGS	0x1F

4 Mechnical aspects

5 Hardware aspects

5.1 Order and assembly

6 Machinecode

We implemented a suggested machinecode instruction set.

Commandname	op0	op1	op2	Description	Binary
HLT				Stops the clock.	0x00
MOVRR	reg8	reg8		Copies reg to reg	0x01
MOVCR	const8	reg8		Writes const to reg	0x02
MOVCR16	const16h	const16l	reg16	Writes const to reg	0x80
LOADAR	addr16h	addr16l	reg8	Loads 8 bit from the address to reg	0x03
LOADRAR	reg16	reg8			0x04
LOADRAOR	reg16	const8	reg8		0x05
STORERA	reg8	addr16h	addr16l		0x06
STORERRA	reg8	reg16			0x07
STORERRAO	reg8	reg16	const8		0x08
PUSH	reg8				0x09
POP	reg8				0x0A
CALL	addr16h	addr16l			0x0B
RET					0x0C
INT					0x8D
EINT					0x8E
JMP	addr16h	addr16l			0x0F
JRA	reg16				0x10
JZ	addr16h	addr16l			0x11
JNZ	addr16h	addr16l			0x12
JE	addr16h	addr16l			0x13
JNE	addr16h	addr16l			0x14
JG	addr16h	addr16l			0x15
JGE	addr16h	addr16l			0x16
JL	addr16h	addr16l			0x17
JLE	addr16h	addr16l			0x18
RCL	reg8				0x19
RCR	reg8				0x1A
ROL	reg8				0x1B
ROR	reg8				0x1C
SAL	reg8				0x1D
SAR	reg8				0x1E
SHL	reg8				0x1F
SHR	reg8				0x20
ADD	reg8	reg8			0x21
ADD16	reg16	reg16			0xA0
ADDC	reg8	const8			0x22
ADDC16	reg16	const16h	const16l		0xA1
SUB	reg8	reg8			0x23
SUB16	reg16	reg16			0xA2
SUBC	reg8	const8			0x24
SUBC16	reg16	const16h	const16l		0xA3
IMUL	reg8	reg8			0x25
IMULC	reg8	const8			0x26
IDIV	reg8	reg8			0x27
IDIVC	reg8	const8			0x28
INC	reg8				0x29
INC16	reg16				0xA8
DEC	reg8				0x2A
DEC16	reg16				0xA9
AND	reg8	reg8			0x30
AND16	reg16	reg16			0xAA
ANDC	reg8	const8			0x31
ANDC16	reg16	const16h	const16l		0xAB
OR	reg8	reg8			0x32
OR16	reg16	reg16			0xAC
ORC	reg8	const8			0x33

Continued on next page

Commandname	op0	op1	op2	Description	Binary
ORC16	reg16	const16h	const16l		0xAD
XOR	reg8	reg8			0x34
XORC	reg8	const8			0x35
NEG	reg8	reg8			0x36
NEGC	reg8	const8			0x37
NOT	reg8	reg8			0x38
NOTC	reg8	const8			0x39
RST					0x41
NOP					0x42
MOD	reg8	reg8			0x43
MODC	reg8	const8			0x44
CMP	reg8	reg8			0x45
CMPC	reg8	const8			0x46
STORECA	const8	addr16h	addr16l		0x47
STORECRA	const8	reg16			0x48
STORECRAO	const8	reg16	const8		0x49

6.1 Machinecode HLT

Usage: `HLT`
Suggested assembler command: `HLT`
description:
This instruction is stopping the clock. It can only be reset manually, by pressing a button.

6.2 Machinecode MOVRR

Usage: `MOVRR reg8 reg8`
Suggested assembler command: `MOV reg8 reg8`
description:
Copies the value from first register to the second register.

6.3 Machinecode MOVCR

Usage: `MOVCR const8 reg8`
Suggested assembler command: `MOV const8 reg8`
description:
Write the 8 bit constant value to the given 8 bit register.

6.4 Machinecode MOVCR16

Usage: `MOVCR16 const16h const16l reg16`
Suggested assembler command: `MOV const16 reg16`
description:
Write the given 16 bit constant to the 16-bit register

6.5 Machinecode LOADAR

Usage: `LOADAR addr16h addr16l reg8`
Suggested assembler command: `LOAD const16 reg8`
description:

6.6 Machinecode LOADRAR

Usage: `LOADRAR reg16 reg8`
Suggested assembler command: `LOAD regaddr16 reg8`
description:

6.7 Machinecode LOADRAOR

Usage: `LOADRAOR reg16 const8 reg8`
Suggested assembler command: `LOAD regaddr16o8 reg8`
description:

6.8 Machinecode STORERA

Usage: `STORERA reg8 addr16h addr16l`
Suggested assembler command: `STORE reg8 const16`
description:

6.9 Machinecode STORERRA

Usage: `STORERRA reg8 reg16`
Suggested assembler command: `STORE reg8 regaddr16`
description:

6.10 Machinecode STORERRAO

Usage: `STORERRAO reg8 reg16 const8`
Suggested assembler command: `STORE reg8 regaddr16o8`
description:

6.11 Machinecode PUSH

Usage: `PUSH reg8`
Suggested assembler command: `PUSH reg8`
description:

6.12 Machinecode POP

Usage: `POP reg8`
Suggested assembler command: `POP reg8`
description:

6.13 Machinecode CALL

Usage: `CALL addr16h addr16l`
Suggested assembler command: `CALL const16`
description:

6.14 Machinecode RET

Usage: `RET`
Suggested assembler command: `RET`
description:

6.15 Machinecode INT

Usage: `INT`
Suggested assembler command: `INT`
description:

6.16 Machinecode EINT

Usage: `EINT`
Suggested assembler command: `EINT`
description:

6.17 Machinecode JMP

Usage: `JMP addr16h addr16l`
Suggested assembler command: `JMP const16`
description:

6.18 Machinecode JRA

Usage: `JRA reg16`
Suggested assembler command: `JMP regaddr16`
description:

6.19 Machinecode JZ

Usage: `JZ addr16h addr16l`
Suggested assembler command: `JZ const16`
description:

6.20 Machinecode JNZ

Usage: `JNZ addr16h addr16l`
Suggested assembler command: `JNZ const16`
description:

6.21 Machinecode JE

Usage: `JE addr16h addr16l`
Suggested assembler command: `JE const16`
description:

6.22 Machinecode JNE

Usage: `JNE addr16h addr16l`
Suggested assembler command: `JNE const16`
description:

6.23 Machinecode JG

Usage: `JG addr16h addr16l`
Suggested assembler command: `JG const16`
description:

6.24 Machinecode JGE

Usage: `JGE addr16h addr16l`
Suggested assembler command: `JGE const16`
description:

6.25 Machinecode JL

Usage: `JL addr16h addr16l`
Suggested assembler command: `JL const16`
description:

6.26 Machinecode JLE

Usage: `JLE addr16h addr16l`
Suggested assembler command: `JLE const16`
description:

6.27 Machinecode RCL

Usage: `RCL reg8`
Suggested assembler command: `RCL reg8`
description:

6.28 Machinecode RCR

Usage: `RCR reg8`
Suggested assembler command: `RCR reg8`
description:

6.29 Machinecode ROL

Usage: `ROL reg8`
Suggested assembler command: `ROL reg8`
description:

6.30 Machinecode ROR

Usage: `ROR reg8`
Suggested assembler command: `ROR reg8`
description:

6.31 Machinecode SAL

Usage: `SAL reg8`
Suggested assembler command: `SAL reg8`
description:

6.32 Machinecode SAR

Usage: `SAR reg8`
Suggested assembler command: `SAR reg8`
description:

6.33 Machinecode SHL

Usage: `SHL reg8`
Suggested assembler command: `SHL reg8`
description:

6.34 Machinecode SHR

Usage: `SHR reg8`
Suggested assembler command: `SHR reg8`
description:

6.35 Machinecode ADD

Usage: `ADD reg8 reg8`
Suggested assembler command: `ADD reg8`
description:

6.36 Machinecode ADD16

Usage: `ADD16 reg16 reg16`
Suggested assembler command: `ADD reg8`
description:

6.37 Machinecode ADDC

Usage: `ADDC reg8 const8`
Suggested assembler command: `ADD reg8`
description:

6.38 Machinecode ADDC16

Usage: `ADDC16 reg16 const16h const16l`
Suggested assembler command: `ADD reg8`
description:

6.39 Machinecode SUB

Usage: `SUB reg8 reg8`
Suggested assembler command: `SUB reg8 reg8`
description:

6.40 Machinecode SUB16

Usage: `SUB16 reg16 reg16`
Suggested assembler command: `SUB reg16 reg16`
description:

6.41 Machinecode SUBC

Usage: `SUBC reg8 const8`
Suggested assembler command: `SUB reg8 const8`
description:

6.42 Machinecode SUBC16

Usage: `SUBC16 reg16 const16h const16l`
Suggested assembler command: `SUB reg16 const16`
description:

6.43 Machinecode IMUL

Usage: `IMUL reg8 reg8`
Suggested assembler command: `IMUL reg8 reg8`
description:

6.44 Machinecode IMULC

Usage: `IMULC reg8 const8`
Suggested assembler command: `IMUL reg8 const8`
description:

6.45 Machinecode IDIV

Usage: `IDIV reg8 reg8`
Suggested assembler command: `IDIV reg8 reg8`
description:

6.46 Machinecode IDIVC

Usage: `IDIVC reg8 const8`
Suggested assembler command: `IDIV reg8 const8`
description:

6.47 Machinecode INC

Usage: `INC reg8`
Suggested assembler command: `INC reg8`
description:

6.48 Machinecode INC16

Usage: `INC16 reg16`
Suggested assembler command: `INC reg16`
description:

6.49 Machinecode DEC

Usage: `DEC reg8`
Suggested assembler command: `DEC reg8`
description:

6.50 Machinecode DEC16

Usage: `DEC16 reg16`
Suggested assembler command: `DEC reg16`
description:

6.51 Machinecode AND

Usage: `AND reg8 reg8`
Suggested assembler command: `AND reg8 reg8`
description:

6.52 Machinecode AND16

Usage: `AND16 reg16 reg16`
Suggested assembler command: `AND reg16 reg16`
description:

6.53 Machinecode ANDC

Usage: `ANDC reg8 const8`
Suggested assembler command: `AND reg8 const8`
description:

6.54 Machinecode ANDC16

Usage: `ANDC16 reg16 const16h const16l`
Suggested assembler command: `AND reg16 const16`
description:

6.55 Machinecode OR

Usage: `OR reg8 reg8`
Suggested assembler command: `OR reg8 reg8`
description:

6.56 Machinecode OR16

Usage: `OR16 reg16 reg16`
Suggested assembler command: `OR reg16 reg16`
description:

6.57 Machinecode ORC

Usage: `ORC reg8 const8`
Suggested assembler command: `OR reg8 const8`
description:

6.58 Machinecode ORC16

Usage: `ORC16 reg16 const16h const16l`
Suggested assembler command: `OR reg16 const16`
description:

6.59 Machinecode XOR

Usage: `XOR reg8 reg8`
Suggested assembler command: `XOR reg8 reg8`
description:

6.60 Machinecode XORC

Usage: `XORC reg8 const8`
Suggested assembler command: `XOR reg8 const8`
description:

6.61 Machinecode NEG

Usage: `NEG reg8 reg8`
Suggested assembler command: `NEG reg8 reg8`
description:

6.62 Machinecode NEGC

Usage: `NEGC reg8 const8`
Suggested assembler command: `NEG reg8 const8`
description:

6.63 Machinecode NOT

Usage: `NOT reg8 reg8`
Suggested assembler command: `NOT reg8 reg8`
description:

6.64 Machinecode NOTC

Usage: `NOTC reg8 const8`
Suggested assembler command: `NOT reg8 const8`
description:

6.65 Machinecode RST

Usage: `RST`
Suggested assembler command: `RST`
description:

6.66 Machinecode NOP

Usage: `NOP`
Suggested assembler command: `NOP`
description:

6.67 Machinecode MOD

Usage: `MOD reg8 reg8`
Suggested assembler command: `MOD reg8 reg8`
description:

6.68 Machinecode MODC

Usage: `MODC reg8 const8`
Suggested assembler command: `MOD reg8 const8`
description:

6.69 Machinecode CMP

Usage: `CMP reg8 reg8`
Suggested assembler command: `CMP reg8 reg8`
description:

6.70 Machinecode CMPC

Usage: `CMPC reg8 const8`
Suggested assembler command: `CMP reg8 const8`
description:

6.71 Machinecode STORECA

Usage: `STORECA const8 addr16h addr16l`
Suggested assembler command: `STORE const8 const16`
description:

6.72 Machinecode STORECRA

Usage: `STORECRA const8 reg16`
Suggested assembler command: `STORE const8 regaddr16`
description:

6.73 Machinecode STORECRAO

Usage: `STORECRAO const8 reg16 const8`
Suggested assembler command: `STORE const8 regaddr16o8`
description:

6.74 Accessing the memory

There are a bunch of options to access a bigger memoryspace as the wordwidth indicates. In my case the Addressbus is 16-bit width, while the Databus only got 8-bit.

The possible options i thought about were:

- using multiple register for addressing
- using one special 16-bit register for addressing
- using one additional 8-bit register for the upper 8-bits of a address
- using some kind of memory banking

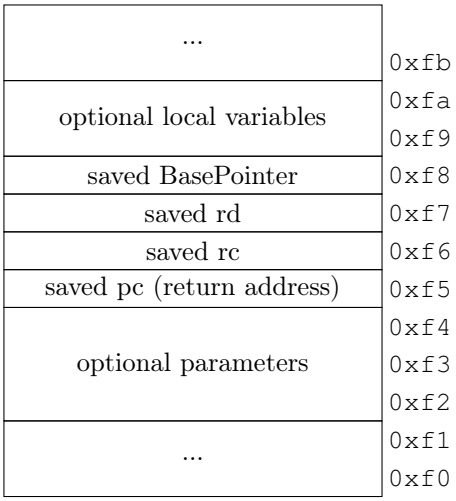
[1]

6.75 Segmented Adressing

[2]

6.76 Framehandling

Bei einem Aufruf von `call` werden die Register `rc` und `rb` gesichert und beim Verlassen des Stackframes wiederhergestellt. Es wird empfohlen die beiden Register `ra` und `rb` zur Übergabe von Parametern zu verwenden. Ansonsten können Parameter natürlich wie üblich über den RAM übergeben werden. Zur Offsetberechnung ist folgender Stackframeaufbau zu beachten:



7 Assembler

I implemented also a reference assembler version. The suggested instructions are directly given at each machinecode instruction. So I my assembler parser is mainly reading in the cpu description file and can generate a binary. This make the development of the assembler much easier. As you though multiple machinecode instructions are treated by one assembler instruction. For example all `MOV*` machinecode instructions are just `MOV` in assembler. The parse knowns from the given parameters which machinecode to choose.

Of cause there are some more things for the assembler, like different section, labels, aliases and directives. All those things are currently implement. In the next section we introduce our suggested assembler style for the EuroOne processor.

7.1 Register

You can use all the registers mentioned in section **??**.

7.2 Constants

7.3 Adresses

7.4 Labels

7.5 Directives

7.6 Aliases

7.7 Variables

7.8 Assembler Instruction Set

Instruction	op0	op1	op2	Description
HLT				
MOV	reg8 const8 const16	reg8 reg8 reg16		
LOAD	const16 regaddr16 regaddr16o8	reg8 reg8 reg8		
STORE	reg8 reg8 reg8 const8 const8 const8	const16 regaddr16 regaddr16o8 const16 regaddr16 regaddr16o8		
PUSH	reg8			
POP	reg8			
CALL	const16			
RET				
INT				
EINT				
JMP	const16 regaddr16			
JZ	const16			
JNZ	const16			
JE	const16			
JNE	const16			
JG	const16			
JGE	const16			
JL	const16			
JLE	const16			
RCL	reg8			
RCR	reg8			
ROL	reg8			
ROR	reg8			
SAL	reg8			
SAR	reg8			
SHL	reg8			
SHR	reg8			
ADD	reg8 reg8 reg8 reg8			
SUB	reg8 reg16	reg8 reg16		

Continued on next page

Instruction	op0	op1	op2	Description
	reg8 reg16	const8 const16		
IMUL	reg8 reg8	reg8 const8		
IDIV	reg8 reg8	reg8 const8		
INC	reg8 reg16			
DEC	reg8 reg16			
AND	reg8 reg16 reg8 reg16	reg8 reg16 const8 const16		
OR	reg8 reg16 reg8 reg16	reg8 reg16 const8 const16		
XOR	reg8 reg8	reg8 const8		
NEG	reg8 reg8	reg8 const8		
NOT	reg8 reg8	reg8 const8		
RST				
NOP				
MOD	reg8 reg8	reg8 const8		
CMP	reg8 reg8	reg8 const8		

7.9 Comments

8 Peripherie

The memory mapped io is starting at 0x7F00. The registers are as follows:

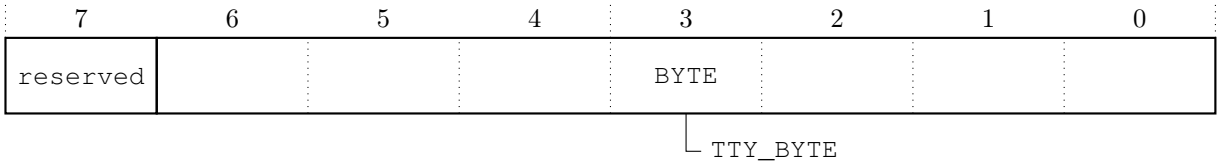
Registername	Description	Address
TTY	Writes to this register will be printed in ascii to the terminal	0x7F00
INT0_STATE	States which interrupts have occured. Write triggers interrupts.	0x7F01
INT0_MASK	Show which interrupts are enabled.	0x7F02
INT0_CLEAR	Write to this register to clear an occured interrupt.	0x7F03
TIMER0L	The low byte of the TIMER0	0x7F04
TIMER0H	The high byte of the TIMER0	0x7F05
MODE	Mode register to handle multiple controller	0x7F0A
FB00	Framebufferbyte	0x7F10
FB01	Framebufferbyte	0x7F11
FB02	Framebufferbyte	0x7F12
FB03	Framebufferbyte	0x7F13
FB04	Framebufferbyte	0x7F14
FB05	Framebufferbyte	0x7F15
FB06	Framebufferbyte	0x7F16
FB07	Framebufferbyte	0x7F17
FB10	Framebufferbyte	0x7F18
FB11	Framebufferbyte	0x7F19
FB12	Framebufferbyte	0x7F1A
FB13	Framebufferbyte	0x7F1B

Continued on next page

Registername	Description	Address
FB14	Framebufferbyte	0x7F1C
FB15	Framebufferbyte	0x7F1D
FB16	Framebufferbyte	0x7F1E
FB17	Framebufferbyte	0x7F1F

8.1 TTY

address: 0x7F00
access: write only

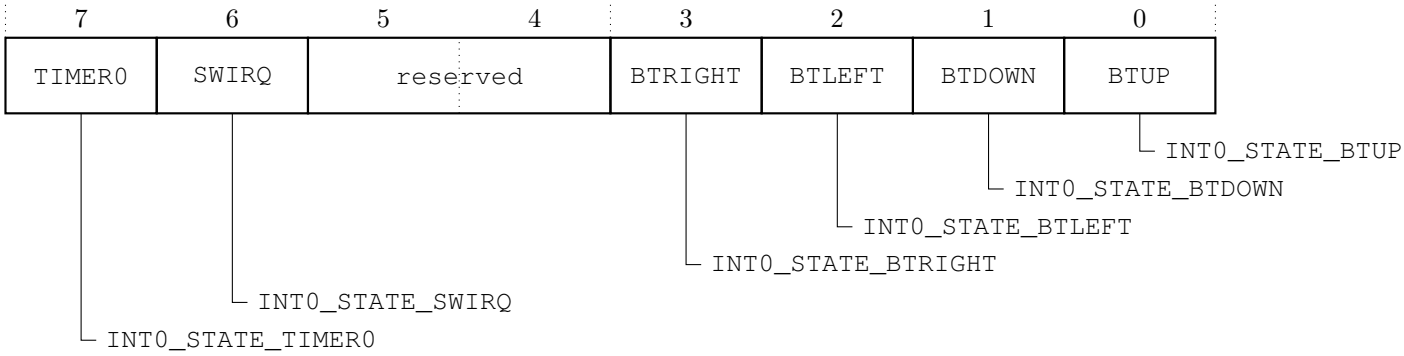


Writes to this register will be printed in ascii to the terminal

Bit	Name	Description
[0:6]	BYTE	0-127 are interpreted as ASCII and plotted to the TTY terminal.
[7]	reserved	ignored

8.2 INT0_STATE

address: 0x7F01
access: readwrite

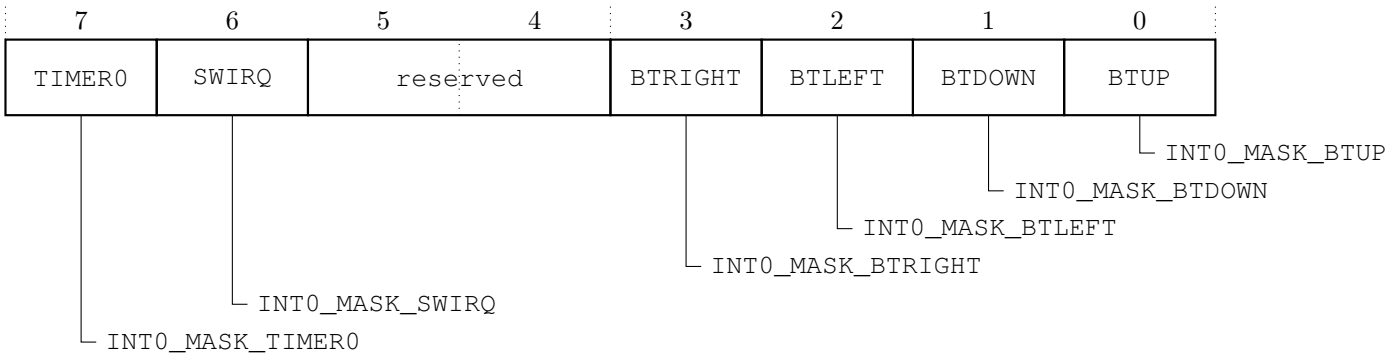


Read this register to see which interrupt has occured. You can also write to it. This will trigger an interrupt. To clear an interrupt use the INT0_CLEAR register.

Bit	Name	Description
[0]	BTUP	Button UP interrupt
[1]	BTDOWN	Button DOWN interrupt
[2]	BTLEFT	Button LEFT interrupt
[3]	BTRIGHT	Button RIGHT interrupt
[4:5]	reserved	
[6]	SWIRQ	Softwareinterrupt
[7]	TIMER0	Timerinterrupt

8.3 INT0_MASK

address: 0x7F02
access: readwrite

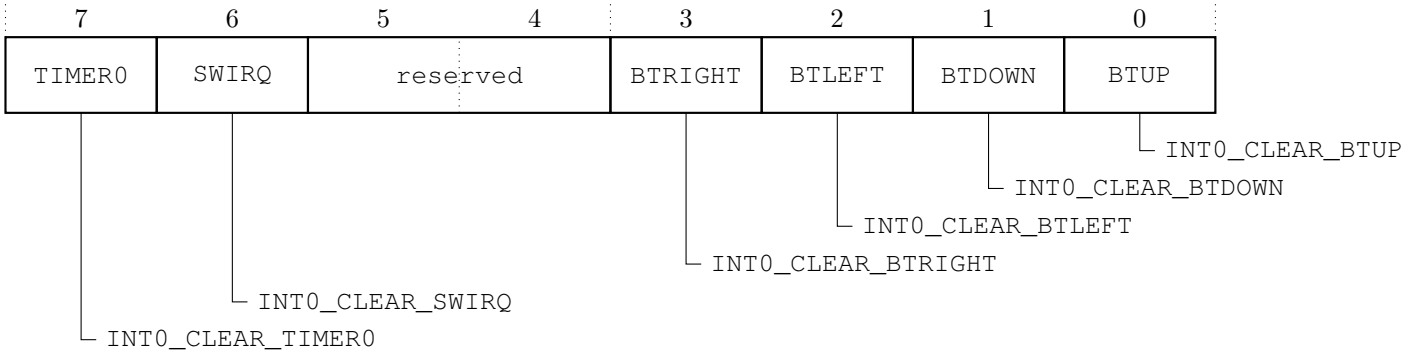


Write this register to enable interrupts. 1 means enabled while 0 means disabled. Disabled interrupts will still be seen in the INT0_MASK register. You may want to AND INT0_MASK and INT0_STATE to get only enabled interrupts.

Bit	Name	Description
[0]	BTUP	Button UP interrupt
[1]	BTDOWN	Button DOWN interrupt
[2]	BTLEFT	Button LEFT interrupt
[3]	BTRIGHT	Button RIGHT interrupt
[4:5]	reserved	reserved
[6]	SWIRQ	Softwareinterrupt
[7]	TIMER0	Timerinterrupt

8.4 INT0_CLEAR

address: 0x7F03
access: read only

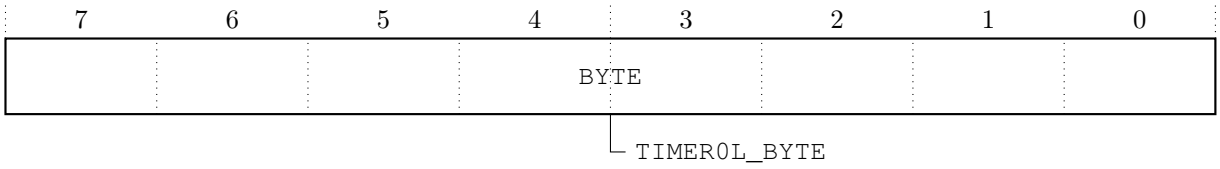


Write to this register to clear an occured interrupt. The interrupt is set to 0 in INT0_STATE.

Bit	Name	Description
[0]	BTUP	Button UP interrupt
[1]	BTDOWN	Button DOWN interrupt
[2]	BTLEFT	Button LEFT interrupt
[3]	BTRIGHT	Button RIGHT interrupt
[4:5]	reserved	reserved
[6]	SWIRQ	Softwareinterrupt
[7]	TIMER0	Timerinterrupt

8.5 TIMER0L

address: 0x7F04
access: readwrite

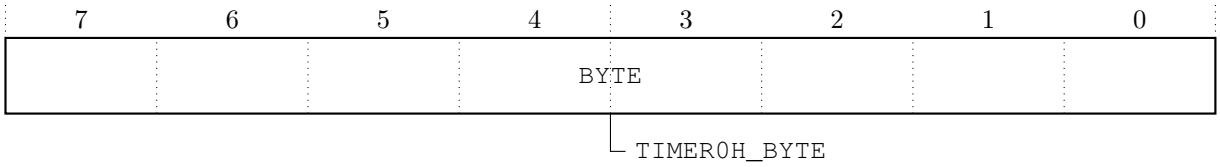


The low byte of the TIMER0. The timer running downwards with one step per cycle. The timer is only running if the MODE_TIMER0_ENABLED bit in the MODE register is set to 1. Also the interrupt is only triggered if MODE_TIMER0_ENABLED is set.

Bit	Name	Description
[0:7]	BYTE	The low byte of the TIMER0

8.6 TIMER0H

address: 0x7F05
access: readwrite

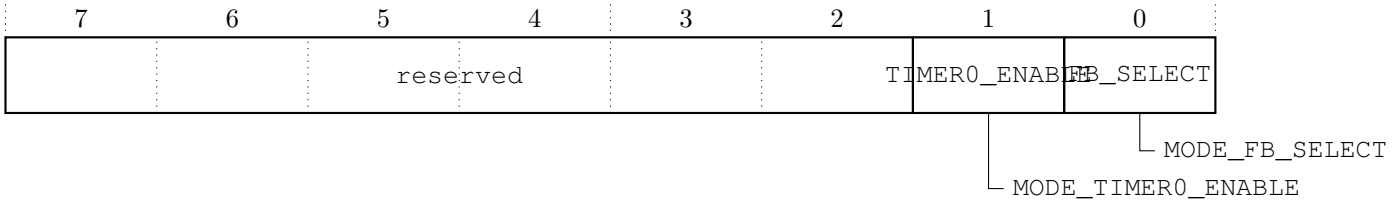


The high byte of the TIMER0. The timer running downwards with one step per cycle. The timer is only running if the MODE_TIMER0_ENABLED bit in the MODE register is set to 1. Also the interrupt is only triggered if MODE_TIMER0_ENABLED is set.

Bit	Name	Description
[0:7]	BYTE	The high byte of the TIMER0

8.7 MODE

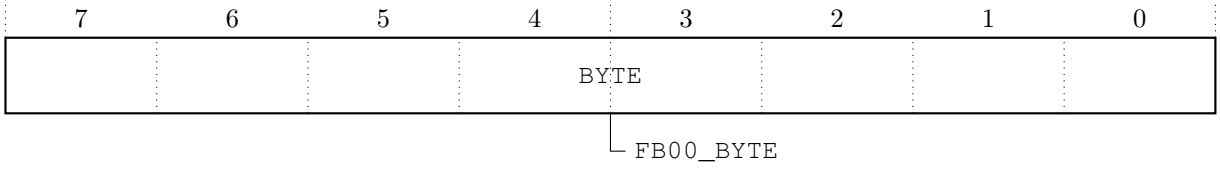
address: 0x7F0A
access: readwrite



Bit	Name	Description
[0]	FB_SELECT	If 0 framebuffer 0 is selected, otherwise framebuffer 1 is selected.
[1]	TIMER0_ENABLE	If 1 TIMER0 is enabled otherwise it is disabled.
[2:7]	reserved	

8.8 FB00

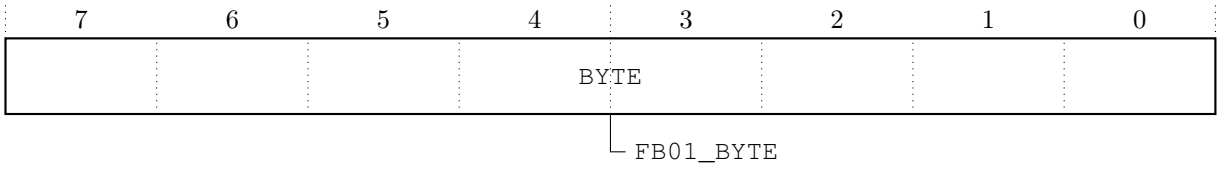
address: 0x7F10
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 0 for framebuffer 0

8.9 FB01

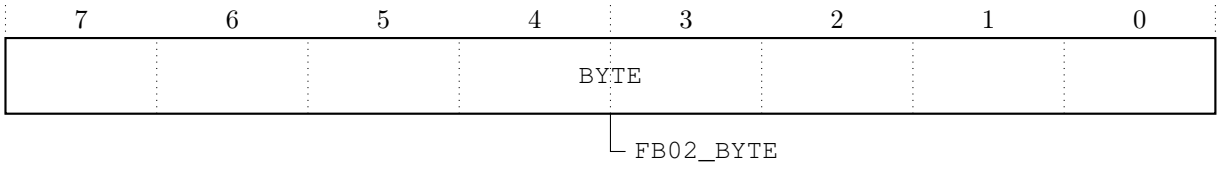
address: 0x7F11
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 1 for framebuffer 0

8.10 FB02

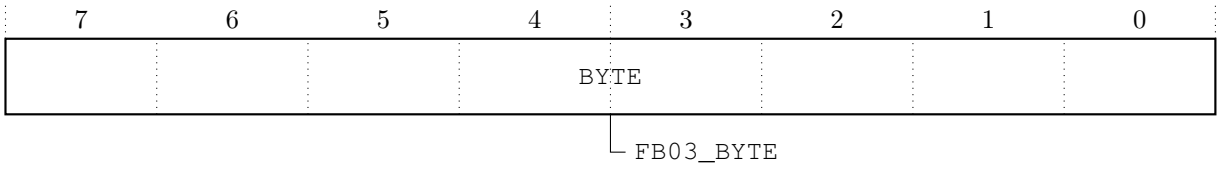
address: 0x7F12
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 2 for framebuffer 0

8.11 FB03

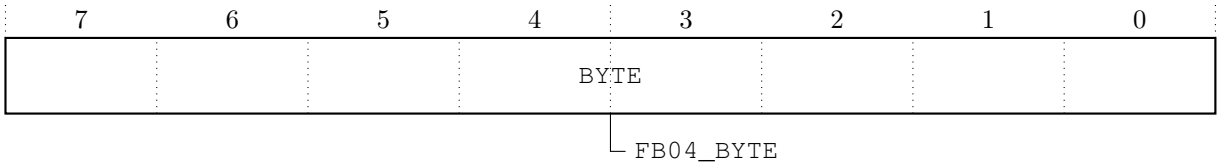
address: 0x7F13
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 3 for framebuffer 0

8.12 FB04

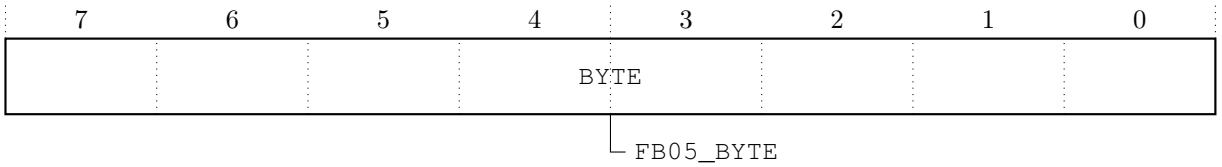
address: 0x7F14
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 4 for framebuffer 0

8.13 FB05

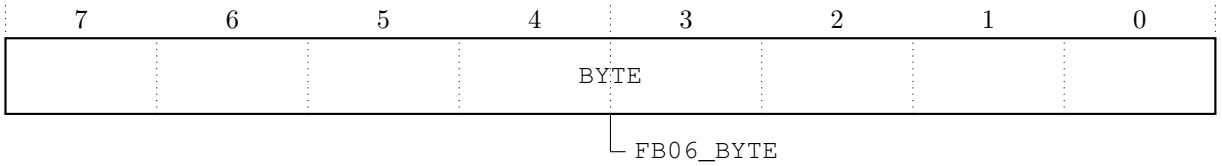
address: 0x7F15
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 5 for framebuffer 0

8.14 FB06

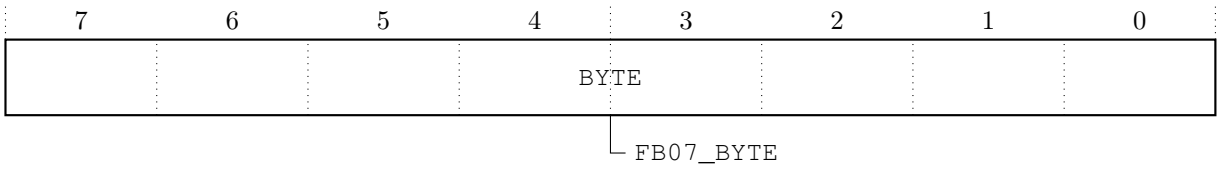
address: 0x7F16
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 6 for framebuffer 0

8.15 FB07

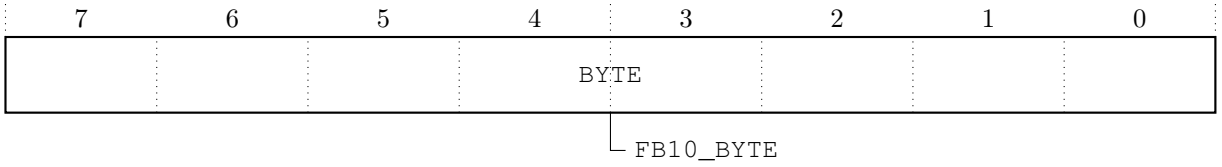
address: 0x7F17
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 7 for framebuffer 0

8.16 FB10

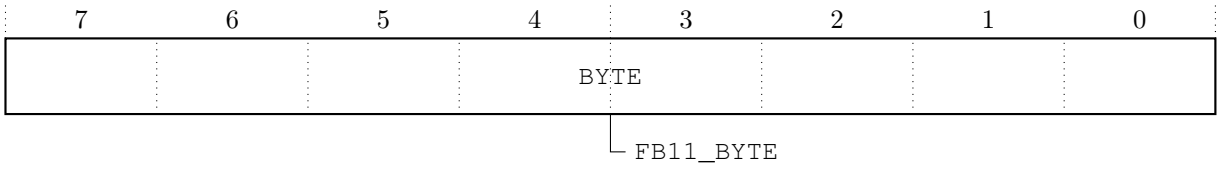
address: 0x7F18
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 0 for framebuffer 1

8.17 FB11

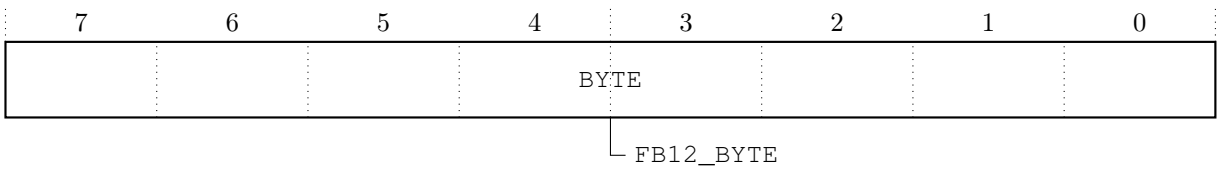
address: 0x7F19
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 1 for framebuffer 1

8.18 FB12

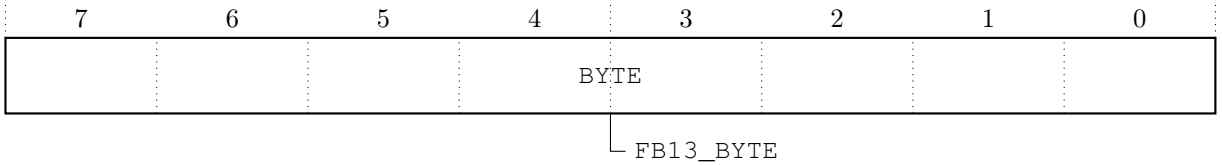
address: 0x7F1A
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 2 for framebuffer 1

8.19 FB13

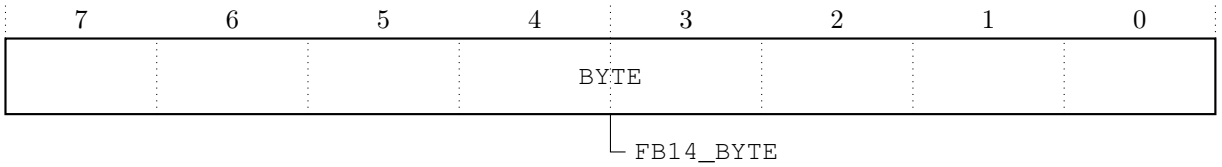
address: 0x7F1B
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 3 for framebuffer 1

8.20 FB14

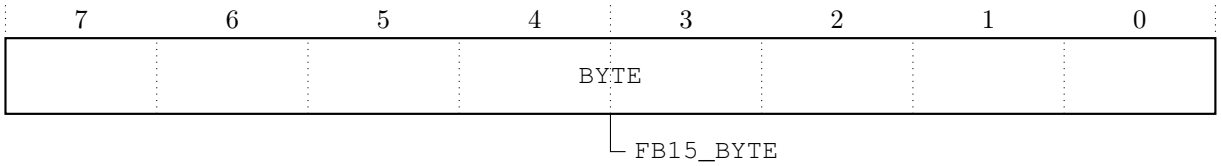
address: 0x7F1C
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 4 for framebuffer 1

8.21 FB15

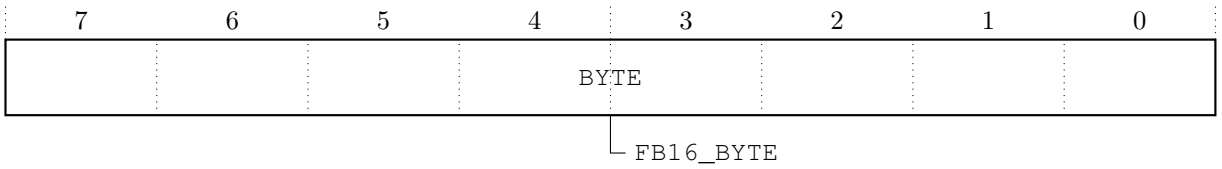
address: 0x7F1D
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 5 for framebuffer 1

8.22 FB16

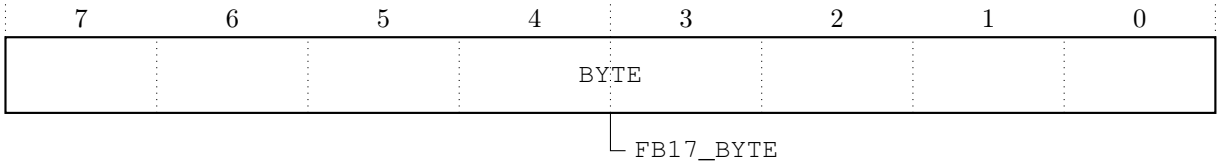
address: 0x7F1E
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 6 for framebuffer 1

8.23 FB17

address: 0x7F1F
access: write_only



Bit	Name	Description
[0:7]	BYTE	Byte 7 for framebuffer 1

9 Interrupts

There are just two vector addresses. `0x0000` is the main reset address and `0x0003` is the address for interrupts. We suggest to use two jumps from this vector addresses like this:

```
1  .text
2  JMP start
3  JMP isr
4
5  start:
6  MOV stack_start rsp
7  ...
8
9  isr:
10 MOV isrstack_start rsp
11 ...
```

10 EuroLang

11 Compiler

12 TODO

- DebuggingInterface / Breakpoints
- 24bit Adressbus/16 bit virtual
- 16-bit Operations
- Timer implementieren
- Interrupts /ISR
- Adressoffset, static, dynamic

Literatur

[1] phuclv, "How can 8-bit processor support more than 256 bytes of ram?."

[2] "x86 assembly language - wikipedia."