**32-bit CPU**

**@2018-today by Milan Vidaković**

Contents

[General information 13](#_Toc530686787)

[Details 13](#_Toc530686788)

[VGA text mode 15](#_Toc530686789)

[VGA graphics mode 320x240 15](#_Toc530686790)

[UART interface 16](#_Toc530686791)

[PS/2 keyboard interface 16](#_Toc530686792)

[Instruction set 18](#_Toc530686793)

[NOP/MOV/IN/OUT/PUSH/POP/RET/IRET/SWAP/HALT GROUP (0x00) 22](#_Toc530686794)

[nop 22](#_Toc530686795)

[mov regx, regy 22](#_Toc530686796)

[Example: 22](#_Toc530686797)

[mov reg, XX 22](#_Toc530686798)

[Example: 22](#_Toc530686799)

[in reg, [XX] 22](#_Toc530686800)

[Example: 23](#_Toc530686801)

[out [XX], reg 23](#_Toc530686802)

[Example: 23](#_Toc530686803)

[push reg 23](#_Toc530686804)

[Example: 23](#_Toc530686805)

[push XX 23](#_Toc530686806)

[Example: 23](#_Toc530686807)

[pop reg 23](#_Toc530686808)

[Example: 24](#_Toc530686809)

[ret 24](#_Toc530686810)

[Example: 24](#_Toc530686811)

[iret 24](#_Toc530686812)

[Example: 24](#_Toc530686813)

[swap 24](#_Toc530686814)

[Example: 25](#_Toc530686815)

[halt 25](#_Toc530686816)

[JUMP GROUP (0x01) 25](#_Toc530686817)

[jmp xx 25](#_Toc530686818)

[Example: 25](#_Toc530686819)

[jz xx 25](#_Toc530686820)

[Example: 25](#_Toc530686821)

[jnz xx 26](#_Toc530686822)

[Example: 26](#_Toc530686823)

[jc xx 26](#_Toc530686824)

[Example: 26](#_Toc530686825)

[jnc xx 26](#_Toc530686826)

[Example: 26](#_Toc530686827)

[jo xx 26](#_Toc530686828)

[Example: 27](#_Toc530686829)

[jno xx 27](#_Toc530686830)

[Example: 27](#_Toc530686831)

[jp xx (jge xx) 27](#_Toc530686832)

[Example: 27](#_Toc530686833)

[jnp xx (js xx) 27](#_Toc530686834)

[Example: 27](#_Toc530686835)

[jg xx 28](#_Toc530686836)

[Example: 28](#_Toc530686837)

[jse xx 28](#_Toc530686838)

[Example: 28](#_Toc530686839)

[CALL GROUP (0x02) 28](#_Toc530686840)

[call xx 28](#_Toc530686841)

[Example: 28](#_Toc530686842)

[callz xx 29](#_Toc530686843)

[Example: 29](#_Toc530686844)

[callnz xx 29](#_Toc530686845)

[Example: 29](#_Toc530686846)

[callc xx 29](#_Toc530686847)

[Example: 29](#_Toc530686848)

[callnc xx 29](#_Toc530686849)

[Example: 30](#_Toc530686850)

[callo xx 30](#_Toc530686851)

[Example: 30](#_Toc530686852)

[callno xx 30](#_Toc530686853)

[Example: 30](#_Toc530686854)

[callp xx (callge xx) 30](#_Toc530686855)

[Example: 30](#_Toc530686856)

[callnp xx (calls xx) 31](#_Toc530686857)

[Example: 31](#_Toc530686858)

[callg xx 31](#_Toc530686859)

[Example: 31](#_Toc530686860)

[callse xx 31](#_Toc530686861)

[Example: 31](#_Toc530686862)

[LOAD/STORE GROUP (0x03) 32](#_Toc530686863)

[ld regx, [regy] 32](#_Toc530686864)

[Example: 32](#_Toc530686865)

[ld reg, [XX] 32](#_Toc530686866)

[Example: 32](#_Toc530686867)

[ld regx, [regy + XX] 32](#_Toc530686868)

[Example: 32](#_Toc530686869)

[ld.b regx, [regy] 32](#_Toc530686870)

[Example: 33](#_Toc530686871)

[ld.b reg, [XX] 33](#_Toc530686872)

[Example: 33](#_Toc530686873)

[ld.b reg, [reg + XX] 33](#_Toc530686874)

[Example: 33](#_Toc530686875)

[st [regy], regx 33](#_Toc530686876)

[Example: 34](#_Toc530686877)

[st [XX], reg 34](#_Toc530686878)

[Example: 34](#_Toc530686879)

[st [reg + XX], reg 34](#_Toc530686880)

[Example: 34](#_Toc530686881)

[st.b [regy], regx 34](#_Toc530686882)

[Example: 35](#_Toc530686883)

[st.b [XX], reg 35](#_Toc530686884)

[Example: 35](#_Toc530686885)

[st.b [regy + XX], regx 35](#_Toc530686886)

[Example: 35](#_Toc530686887)

[ADD/SUB GROUP (0x0004) 35](#_Toc530686888)

[add regx, regy 35](#_Toc530686889)

[Example: 36](#_Toc530686890)

[add reg, xx 36](#_Toc530686891)

[Example: 36](#_Toc530686892)

[add regx, [regy] 36](#_Toc530686893)

[Example: 36](#_Toc530686894)

[add reg, [xx] 36](#_Toc530686895)

[Example: 37](#_Toc530686896)

[add regx, [regy + xx] 37](#_Toc530686897)

[Example: 37](#_Toc530686898)

[add.b regx, [regy] 37](#_Toc530686899)

[Example: 37](#_Toc530686900)

[add.b reg, [xx] 37](#_Toc530686901)

[Example: 38](#_Toc530686902)

[add.b regx, [regy + xx] 38](#_Toc530686903)

[Example: 38](#_Toc530686904)

[sub regx, regy 38](#_Toc530686905)

[Example: 38](#_Toc530686906)

[sub reg, xx 38](#_Toc530686907)

[Example: 39](#_Toc530686908)

[sub regx, [regy] 39](#_Toc530686909)

[Example: 39](#_Toc530686910)

[sub reg, [xx] 39](#_Toc530686911)

[Example: 39](#_Toc530686912)

[sub regx, [regy + xx] 39](#_Toc530686913)

[Example: 40](#_Toc530686914)

[sub.b regx, [regy] 40](#_Toc530686915)

[Example: 40](#_Toc530686916)

[sub.b reg, [xx] 40](#_Toc530686917)

[Example: 40](#_Toc530686918)

[sub.b regx, [regy + xx] 40](#_Toc530686919)

[Example: 41](#_Toc530686920)

[AND/OR GROUP (0x0005) 41](#_Toc530686921)

[and regx, regy 41](#_Toc530686922)

[Example: 41](#_Toc530686923)

[and reg, xx 41](#_Toc530686924)

[Example: 41](#_Toc530686925)

[and regx, [regy] 42](#_Toc530686926)

[Example: 42](#_Toc530686927)

[and reg, [xx] 42](#_Toc530686928)

[Example: 42](#_Toc530686929)

[and regx, [regy + xx] 42](#_Toc530686930)

[Example: 42](#_Toc530686931)

[and regx, [regy] 43](#_Toc530686932)

[Example: 43](#_Toc530686933)

[and.b reg, [xx] 43](#_Toc530686934)

[Example: 43](#_Toc530686935)

[and.b regx, [regy + xx] 43](#_Toc530686936)

[Example: 43](#_Toc530686937)

[or regx, regy 44](#_Toc530686938)

[Example: 44](#_Toc530686939)

[or reg, xx 44](#_Toc530686940)

[Example: 44](#_Toc530686941)

[or regx, [regy] 44](#_Toc530686942)

[Example: 44](#_Toc530686943)

[or reg, [xx] 45](#_Toc530686944)

[Example: 45](#_Toc530686945)

[or regx, [regy + xx] 45](#_Toc530686946)

[Example: 45](#_Toc530686947)

[or.b regx, [regy] 45](#_Toc530686948)

[Example: 45](#_Toc530686949)

[or.b reg, [xx] 46](#_Toc530686950)

[Example: 46](#_Toc530686951)

[or.b regx, [regy + xx] 46](#_Toc530686952)

[Example: 46](#_Toc530686953)

[XOR/NEG GROUP (0x0006) 46](#_Toc530686954)

[xor regx, regy 46](#_Toc530686955)

[Example: 46](#_Toc530686956)

[xor reg, xx 47](#_Toc530686957)

[Example: 47](#_Toc530686958)

[xor regx, [regy] 47](#_Toc530686959)

[Example: 47](#_Toc530686960)

[xor reg, [xx] 47](#_Toc530686961)

[Example: 47](#_Toc530686962)

[xor regx, [regy + xx] 48](#_Toc530686963)

[Example: 48](#_Toc530686964)

[xor.b regx, [regy] 48](#_Toc530686965)

[Example: 48](#_Toc530686966)

[xor.b reg, [xx] 48](#_Toc530686967)

[Example: 48](#_Toc530686968)

[xor.b regx, [regy + xx] 49](#_Toc530686969)

[Example: 49](#_Toc530686970)

[neg reg 49](#_Toc530686971)

[Example: 49](#_Toc530686972)

[neg [reg] 49](#_Toc530686973)

[Example: 49](#_Toc530686974)

[neg [xx] 50](#_Toc530686975)

[Example: 50](#_Toc530686976)

[neg [reg + xx] 50](#_Toc530686977)

[Example: 50](#_Toc530686978)

[neg.b [reg] 50](#_Toc530686979)

[Example: 50](#_Toc530686980)

[neg.b [xx] 51](#_Toc530686981)

[Example: 51](#_Toc530686982)

[neg.b [reg + xx] 51](#_Toc530686983)

[Example: 51](#_Toc530686984)

[SHIFT GROUP (0x0007) 51](#_Toc530686985)

[shl regx, regy 51](#_Toc530686986)

[Example: 51](#_Toc530686987)

[shl reg, xx 52](#_Toc530686988)

[Example: 52](#_Toc530686989)

[shl regx, [regy] 52](#_Toc530686990)

[Example: 52](#_Toc530686991)

[shl reg, [xx] 52](#_Toc530686992)

[Example: 52](#_Toc530686993)

[shl regx, [regy + xx] 53](#_Toc530686994)

[Example: 53](#_Toc530686995)

[shl.b regx, [regy] 53](#_Toc530686996)

[Example: 53](#_Toc530686997)

[shl.b reg, [xx] 53](#_Toc530686998)

[Example: 53](#_Toc530686999)

[shl.b regx, [regy + xx] 54](#_Toc530687000)

[Example: 54](#_Toc530687001)

[shr regx, regy 54](#_Toc530687002)

[Example: 54](#_Toc530687003)

[shr reg, xx 54](#_Toc530687004)

[Example: 54](#_Toc530687005)

[shr regx, [regy] 55](#_Toc530687006)

[Example: 55](#_Toc530687007)

[shr reg, [xx] 55](#_Toc530687008)

[Example: 55](#_Toc530687009)

[shr regx, [regy + xx] 55](#_Toc530687010)

[Example: 55](#_Toc530687011)

[shr.b regx, [regy] 56](#_Toc530687012)

[Example: 56](#_Toc530687013)

[shr.b reg, [xx] 56](#_Toc530687014)

[Example: 56](#_Toc530687015)

[shr.b regx, [regy + xx] 56](#_Toc530687016)

[Example: 56](#_Toc530687017)

[MUL/DIV GROUP (0x0008) 57](#_Toc530687018)

[mul regx, regy 57](#_Toc530687019)

[Example: 57](#_Toc530687020)

[mul reg, XX 57](#_Toc530687021)

[Example: 57](#_Toc530687022)

[mul regx, [regy] 57](#_Toc530687023)

[Example: 57](#_Toc530687024)

[mul reg, [xx] 58](#_Toc530687025)

[Example: 58](#_Toc530687026)

[mul regx, [regy + xx] 58](#_Toc530687027)

[Example: 58](#_Toc530687028)

[mul.b regx, [regy] 58](#_Toc530687029)

[Example: 58](#_Toc530687030)

[mul.b reg, [xx] 59](#_Toc530687031)

[Example: 59](#_Toc530687032)

[mul.b regx, [regy + xx] 59](#_Toc530687033)

[Example: 59](#_Toc530687034)

[div regx, regy 59](#_Toc530687035)

[Example: 59](#_Toc530687036)

[div reg, XX 60](#_Toc530687037)

[Example: 60](#_Toc530687038)

[div regx, [regy] 60](#_Toc530687039)

[Example: 60](#_Toc530687040)

[div reg, [xx] 60](#_Toc530687041)

[Example: 60](#_Toc530687042)

[div regx, [regy + xx] 61](#_Toc530687043)

[Example: 61](#_Toc530687044)

[div.b regx, [regy] 61](#_Toc530687045)

[Example: 61](#_Toc530687046)

[div reg, [xx] 61](#_Toc530687047)

[Example: 61](#_Toc530687048)

[div.b regx, [regy + xx] 62](#_Toc530687049)

[Example: 62](#_Toc530687050)

[INC/DEC GROUP (0x0009) 62](#_Toc530687051)

[inc reg 62](#_Toc530687052)

[Example: 62](#_Toc530687053)

[inc [reg] 62](#_Toc530687054)

[Example: 62](#_Toc530687055)

[inc [xx] 63](#_Toc530687056)

[Example: 63](#_Toc530687057)

[inc [reg + xx] 63](#_Toc530687058)

[Example: 63](#_Toc530687059)

[inc.b [reg] 63](#_Toc530687060)

[Example: 63](#_Toc530687061)

[inc.b [xx] 64](#_Toc530687062)

[Example: 64](#_Toc530687063)

[inc [reg + xx] 64](#_Toc530687064)

[Example: 64](#_Toc530687065)

[dec reg 64](#_Toc530687066)

[Example: 64](#_Toc530687067)

[dec [reg] 65](#_Toc530687068)

[Example: 65](#_Toc530687069)

[dec [xx] 65](#_Toc530687070)

[Example: 65](#_Toc530687071)

[dec [reg + xx] 65](#_Toc530687072)

[Example: 65](#_Toc530687073)

[dec.b [reg] 65](#_Toc530687074)

[Example: 66](#_Toc530687075)

[dec.b [xx] 66](#_Toc530687076)

[Example: 66](#_Toc530687077)

[dec.b [reg + xx] 66](#_Toc530687078)

[Example: 66](#_Toc530687079)

[CMP/INV GROUP (0x000a) 66](#_Toc530687080)

[cmp regx, regy 66](#_Toc530687081)

[Example: 67](#_Toc530687082)

[cmp reg, xx 67](#_Toc530687083)

[Example: 67](#_Toc530687084)

[cmp regx, [regy] 67](#_Toc530687085)

[Example: 67](#_Toc530687086)

[cmp reg, [xx] 67](#_Toc530687087)

[Example: 68](#_Toc530687088)

[cmp regx, [regy + xx] 68](#_Toc530687089)

[Example: 68](#_Toc530687090)

[cmp.b regx, [regy] 68](#_Toc530687091)

[Example: 68](#_Toc530687092)

[cmp.b reg, [xx] 68](#_Toc530687093)

[Example: 69](#_Toc530687094)

[cmp.b regx, [regy + xx] 69](#_Toc530687095)

[Example: 69](#_Toc530687096)

[inv reg 69](#_Toc530687097)

[Example: 69](#_Toc530687098)

[inv [reg] 69](#_Toc530687099)

[Example: 70](#_Toc530687100)

[inv [xx] 70](#_Toc530687101)

[Example: 70](#_Toc530687102)

[inv [reg + xx] 70](#_Toc530687103)

[Example: 70](#_Toc530687104)

[inv.b [reg] 70](#_Toc530687105)

[Example: 71](#_Toc530687106)

[inv.b [xx] 71](#_Toc530687107)

[Example: 71](#_Toc530687108)

[inv.b [reg + xx] 71](#_Toc530687109)

[Example: 71](#_Toc530687110)

32-bit CPU

# General information

This 32-bit CPU and computer has been developed using Verilog on the DE0 NANO FPGA board. The code is on the github at the following URL:

<https://github.com/milanvidakovic/FPGAComputer>

The assembler for this CPU is on the github at the following URL:

<https://github.com/milanvidakovic/FPGAcustomasm>

The emulator written in Java is on the github at the following URL:

<https://github.com/milanvidakovic/FPGAEmulator>

The UART loader is on the github at the following URL:

<https://github.com/milanvidakovic/FPGARaspbootin64Client>

Some photos and background story can be found at:

<https://mvidakovic.blogspot.com/2018/06/16-bit-computer-made-using-fpga.html>

# Details

This 16-bit CPU has 8 general-purpose registers (r0 – r7), pc (program counter), sp (stack pointer), ir (instruction register), mbr (memory buffer register), h (higher word when multiplying, or remainder when dividing). Each register is 16-bits wide.

The address bus is 16 bits wide, addressing 65536 addresses. Data bus is also 16 bits wide, but all the addresses are 8-bit aligned. This gives 65536 bytes, or 64KB.

By default, address mode is 16-bit. This means that if the instruction does not have the ".b" suffix, it will work with 16-bit operand. Those instructions without ".b" suffix are 16-bit wide. If those instructions deal with memory, the address must be aligned to 16-bits (even).

If the instruction has the ".b" suffix, it is byte-oriented. The address in byte-oriented instructions can be both even and odd.

For example, this instruction is word-oriented:

ld r0, [1000]

It loads the 16-bit content of the address 1000 (two bytes, one byte from the 1001 and the other from 1000) and stores that 16-bit value in the r0 register. The address must be even.

However, this instruction is byte-oriented:

ld.b r0, [1001]

It loads the 8-bit value (one byte) from the address 1001.

It the 16-bit word is stored in the memory, it is stored as big endian, having the lower byte in odd address, and the upper byte in the even address. For example, the number 0x1234 stored at the 1000 address is stored like this:

|  |  |
| --- | --- |
| address | content |
| 1000 | 0x12 |
| 1001 | 0x34 |

Video output signal is VGA, 640x480 pixels. Text mode framebuffer has 80x60 characters, each character being 8x8 pixels in dimensions. Framebuffer in text mode has 4800 16-bit words (80x60 characters with attributes). The lower byte (odd address) has the ASCII character, while the upper byte (even address) has the attributes (3 bits for the ink color, inverted, and 3 bits for the background color, remaining two bits unused).

CPU has three interrupts: IRQ0,IRQ1, and IRQ2. IRQ0 is connected to the KEY2 of the DE0-NANO, while IRQ1 is conencted to the UART. IRQ2 is connected to the PS/2 keyboard connector. Whenever a byte comes to the UART, it generates an IRQ1. Whenever a make or break code comes from the PS/2 keyboard, it generates an IRQ2.

Interrupt causes CPU to push flags to the stack, then to push PC to the stack (the address of the next instruction) and then forces CPU to jump to the location designated for the interrupt:

* for the IRQ0, it is 0x0008,
* for the IRQ1, it is 0x0010, and
* for the IRQ2, it is 0x0018.

It is up to the programmer to put the code in those locations. Usually, it is a JUMP instruction. To return from the interrupt routine, it is necessary to execute the IRET instruction. It pops the return address, and then pops the flags register, and then goes back into the original program.

KEY1 of the DE0-NANO is used as the reset key. When pressed, it forces CPU to go to the 0x0000 address. Usually there is a JUMP instruction to go to the main program.

# VGA text mode

Text mode is 80x60 characters, occupying 4800 words. The framebuffer starts at 26880 decimal. Each character needs two bytes. Lower byte is the ASCII code of a character, while the upper byte is the attributes:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | Foreground color, inverted | | |  | Background color | | |
|  | r | g | b |  | r | g | b |

The foreground color is inverted so zero values (default) would mean white color. That way, you don't need to set the foreground color to white, and by default (0, 0, 0), is white for the foreground. The default background color is black (0, 0, 0). This means that if the upper (attribute) byte is zero (0x00), the background color is black, and the foreground color is white.

VGA female connector is connected via resistors to the GPIO-0 expansion header of the DE0-NANO board:

* GPIO\_R (pin 2, GPIO\_00, PIN\_A3) -> 68Ohm -> VGA\_R,
* GPIO\_G (pin 4, GPIO\_01, PIN\_C3) -> 68Ohm -> VGA\_G,
* GPIO\_B (pin 6, GPIO\_03, PIN\_D3) -> 68Ohm -> VGA\_B,
* GPIO\_HS (pin 8, GPIO\_05, PIN\_B4) -> 470Ohm -> VGA\_HORIZONTAL\_SYNC,
* GPIO\_VS (pin 10, GPIO\_07, PIN\_B5) -> 470Ohm -> VGA\_VERTICAL\_SYNC.

# VGA graphics mode 320x240

This computer also has graphics mode of 320x240 pixels, 8 colors for each pixel. Framebuffer starts at the same address as the text mode one (26880 decimal), but it now displays pixels, instead of characters.

Each pixel can have one of eight colors. There are two pixels per byte in the framebuffer:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | Foreground color, inverted | | |  | Background color | | |
|  | r | g | b |  | r | g | b |

For example, if you want to draw two red pixels at the (0, 0) coordinates (top left corner), you need to put the following byte into the location 2400:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|  | Foreground color, inverted | | |  | Background color | | |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

Or, 0x44 in hex.

Since the default mode is text mode, there is a way to switch video modes:

mov r1, 1

out [128], r1

This code will switch to the graphics mode of 320x240. To switch back to the text mode, you need to execute the following code:

mov r1, 0

out [128], r1

Number 1 at the port 128 sets the video mode to 320x240, while 0 sets to the text mode.

# UART interface

UART interface provides TTL serial communication on 115200kbps. It uses one start bit, one stop bit, and eight data bits, no parity, no handshake.

UART is connected to the GPIO-0 expansion header of the DE0-NANO board:

* TX (pin 32, GPIO\_025, PIN\_D9) should be connected to the RX pin of the PC,
* RX (pin 34, GPIO\_027, PIN\_E10) should be connected to the TX pin of the PC.

UART is used within the CPU via IN, and OUT instructions. RX also triggers the IRQ1, which means that whenever a byte is received via UART, the IRQ1 will be triggered, forcing CPU to jump to the 0x0008 address. There you should place the JUMP instruction to your UART interrupt routine.

Inside the UART interrupt routine, you can get the received byte by using the IN instruction:

in r1, [64] ; r1 holds now received byte from the UART (address 64 decimal)

To send a byte, first you need to check if the UART TX is free. You can do it by using the in instruction:

loop:

in r5, [65] ; tx busy in r5

cmp r5, 0

jz not\_busy ; if not busy, send back the received character

j loop

not\_busy:

out [66], r1 ; send the received character to the UART

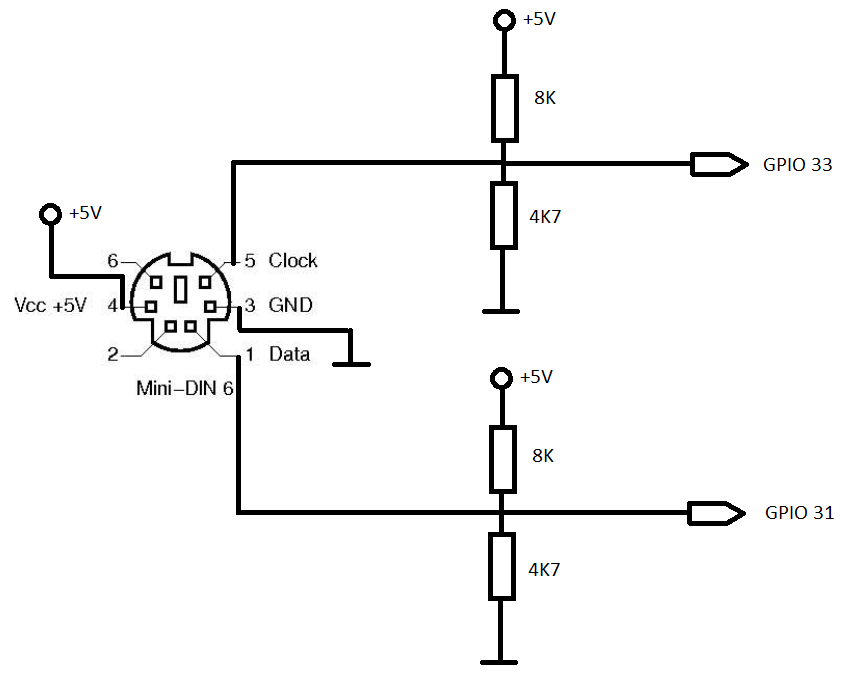
Addresses used by the UART are in the following table:

|  |  |
| --- | --- |
| Address | Description |
| 64 | Received byte from the RX part of the UART. |
| 65 | 0 if the TX part of the UART is free to send a byte, 1 if TX part is busy. |
| 66 | Used to put the byte to be sent via TX part of the UART. |

# PS/2 keyboard interface

PS/2 connector is connected to the GPIO ports of the DE0-NANO board:

* Data is connected to the GPIO31 (PIN\_D11) port
* Clock is connected to the GPIO33 (PIN\_B12) port.



Connection of the female PS/2 conenctor to the GPIO ports of the DE0 NANO board

The communication between keyboard and computer is a clocked serial. Clock pulses appear on the Clock pin, while data is on the Data pin, synchronized with the Clock on the falling edge. There is one start bit, one parity bit and one stop bit.

The IRQ2 is triggered when a byte from PS/2 keyboard arrives. The CPU then jumps to the address of 24 decimal, where the raw PS/2 keyboard handling routine should be. Actually, at that address should be one JUMP instruction which will jump to the handling routine.

In the interrupt routine, it is necessary to read the raw PS/2 byte that has arrived from the keyboard. To do so, it is necessary to use the IN assembler instruction:

in r0, [68]

Please pay attention that raw PS/2 protocol gives make and break codes when a key is pressed, so this requires quite a lot of processing.

# Instruction set

|  |  |  |  |
| --- | --- | --- | --- |
| Group number | Group name | Group members | Group description |
| 0 | [NOP/MOV/](#_NOP/MOV/IN/OUT/PUSH/POP/RET/IRET/HA)  [IN/OUT/PUSH/](#_NOP/MOV/IN/OUT/PUSH/POP/RET/IRET/HA)  [POP/RET/IRET/SWAP/HALT](#_NOP/MOV/IN/OUT/PUSH/POP/RET/IRET/HA) | nop  mov reg, xx  mov reg, reg  in reg, [xx]  out [xx], reg  push reg  push xx  pop reg  ret  iret  swap  halt | The most general group. Deals with putting values into registers, exchanging values between registers, I/O operations, stack operations, and returning from subroutines. Swapping content between registers is done using th SWAP instruction. NOP and HALT are also in this group. |
| 1 | [JUMP](#_JUMP_GROUP_(0x01)) | jmp xx  jc xx  jnc xx  jz xx  jnz xx  jo xx  jno xx  jp (jge) xx  jnp (js) xx  jg xx  jse xx | Jump to the given location. |
| 2 | [CALL](#_CALL_GROUP_(0x02)) | call xx  callc xx  callnc xx  callz xx  callnz xx  callo xx  callno xx  callp (callge) xx  callnp (calls) xx  callg xx  callse xx | Calling subroutine. Puts the return address on the stack before jumping to the subroutine. Needs to call RET when returning from the subroutine. |
| 3 | [LOAD/STORE](#_LOAD/STORE_GROUP_(0x03)) | ld reg, [xx]  ld reg, [reg]  ld reg, [reg + xx]  ld.b reg, [reg]  ld.b reg, [xx]  ld.b reg, [reg + xx]  st [xx], reg  st [reg], reg  st [reg + xx], reg  st.b [reg], reg  st.b [xx], reg  st.b [reg + xx], reg | Load from memory into the register  destination: register  source: memory address given by the number, or by the register, or by the register+number.  Store the given register into the memory location  destination: memory location given by the number, or by the register, or by the register+number. |
| 4 | [ADD/SUB](#_ADD/SUB_GROUP_(0x0004)) | add reg, reg  add reg, xx  add reg, [reg]  add reg, [xx]  add reg, [reg + xx]  add.b reg, [reg]  add.b reg, [xx]  add.b reg, [reg + xx]  sub reg, reg  sub reg, xx  sub reg, [reg]  sub reg, [xx]  sub reg, [reg + xx]  sub.b reg, [reg]  sub.b reg, [xx]  sub.b reg, [reg + xx] | ADD and SUB group. |
| 5 | [AND/OR](#_AND/OR_GROUP_(0x0005)) | and reg, reg  and reg, xx  and reg, [reg]  and reg, [xx]  and reg, [reg + xx]  and.b reg, [reg]  and.b reg, [xx]  and.b reg, [reg + xx]  or reg, reg  or reg, xx  or reg, [reg]  or reg, [xx]  or reg, [reg + xx]  or.b reg, [reg]  or.b reg, [xx]  or.b reg, [reg + xx] | AND and OR group. |
| 6 | [XOR](#_XOR_GROUP_(0x0006)) | xor reg, reg  xor reg, xx  xor reg, [reg]  xor reg, [xx]  xor reg, [reg + xx]  xor.b reg, [reg]  xor.b reg, [xx]  xor.b reg, [reg + xx]  neg reg  neg [reg]  neg [xx]  neg [reg + xx]  neg.b [reg]  neg.b [xx]  neg.b [reg + xx] | XOR and negation (-value). |
| 7 | [SHL/SHR](#_SHIFT_GROUP_(0x0007)) | shl reg, reg  shl reg, xx  shl reg, [reg]  shl reg, [xx]  shl reg, [reg + xx]  shl.b reg, [reg]  shl.b reg, [xx]  shl.b reg, [reg + xx]  shr reg, reg  shr reg, xx  shr reg, [reg]  shr reg, [xx]  shr reg, [reg + xx]  shr.b reg, [reg]  shr.b reg, [xx]  shr.b reg, [reg + xx] | Shift left and right. |
| 8 | [MUL/DIV](#_MUL/DIV_GROUP_(0x0008)) | mul reg, reg  mul reg, xx  mul reg, [reg]  mul reg, [xx]  mul reg, [reg + xx]  mul.b reg, [reg]  mul.b reg, [xx]  mul.b reg, [reg + xx]  div reg, reg  div reg, xx  div reg, [reg]  div reg, [xx]  div reg, [reg + xx]  div.b reg, [reg]  div.b reg, [xx]  div.b reg, [reg + xx] | Multiplication and division. |
| 9 | [INC/DEC](#_INC/DEC_GROUP_(0x0009)) | inc reg  inc [reg]  inc [xx]  inc [reg + xx]  inc.b [reg]  inc.b [xx]  inc.b [reg + xx]  dec reg  dec [reg]  dec [xx]  dec [reg + xx]  dec.b [reg]  dec.b [xx]  dec.b [reg + xx] | Increment and decrement. |
| 10 | [CMP/INV](#_CMP/NEG_GROUP_(0x000a)) | cmp reg, reg  cmp reg, xx  cmp reg, [reg]  cmp reg, [xx]  cmp reg, [reg + xx]  cmp.b reg, [reg]  cmp.b reg, [xx]  cmp.b reg, [reg + xx]  inv reg  inv [reg]  inv [xx]  inv [reg + xx]  inv.b [reg]  inv.b [xx]  inv.b [reg + xx] | Compare registers with registers/memory.  Inversion of bits. |

# NOP/MOV/IN/OUT/PUSH/POP/RET/IRET/SWAP/HALT GROUP (0x00)

## nop

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| 0000 | 0000 | 0000 | 0000 |

Example:

nop

binary: 0000 0000 0000 0000

hex: 00 00

## mov regx, regy

regx <-- regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8-sp  9-h | bbbb  0-7: r0-r7  8-sp  9-h | 0001  0=>mov regx, regy | 0000 |

### Example:

mov r2, r1

binary: 0001 0010 0001 0000

hex: 12 00

## mov reg, XX

reg <-- XX

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8-sp | 0010  2 =>mov reg, xx | 0000 |

### Example:

mov r1, 0x0f

binary: 0000 0001 0010 0000, 0000 0000 0000 1111

hex: 01 20, 00 0f

## in reg, [XX]

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | xbbb  0-7: r0-r7 | 0011  3 =>in reg, [xx] | 0000 |

### Example:

in r1, [0x0f]

binary: 0000 0001 0010 0001, 0000 0000 0000 1111

hex: 01 30, 00 0f

## out [XX], reg

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | xbbb  0-7: r0-r7 | 0100  4 =>out [xx], reg | 0000 |

### Example:

out r1, 0x0f

binary: 0000 0001 0100 0000, 0000 0000 0000 1111

hex: 01 40, 00 0f

## push reg

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8-sp  9-h | 0101  5=>push register | 0000 |

### Example:

push r0

binary: 0000 0000 0101 0000

hex: 00 50

## push XX

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | xxxx | 0110  6=>push number | 0000 |

### Example:

push 0x0f

binary: 0000 0000 0110 000, 0000 0000 0000 1111

hex: 00 60, 00 0f

## pop reg

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8-sp  9-h | 0111  7=>pop register | 0000 |

### Example:

pop r0

binary: 0000 0000 0111 0000

hex: 00 70

## ret

Returns from the subroutine.

|  |  |  |
| --- | --- | --- |
| unused | call/ret kind | group |
| xxxx xxxx | 1000  8=>return | 0000 |

### Example:

ret

binary: 0000 0000 1000 0000

hex: 00 80

## iret

Returns from the interrupt subroutine. First it pops flags from the stack, then it pops the return address from the stack.

|  |  |  |
| --- | --- | --- |
| unused | call/ret kind | group |
| xxxx xxxx | 10001  9=>return from interrupt | 0000 |

### Example:

iret

binary: 0000 0000 1001 0000

hex: 00 90

## swap

Swaps two registers content.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8-sp  9-h | bbbb  0-7: r0-r7  8-sp  9-h | 1010  10=>swap regx, regy | 0000 |

### Example:

swap r0, r1

binary: 0000 0001 1010 0000

hex: 00 90

## halt

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| 1111 | 1111 | 1111 | 0000 |

Example:

halt

binary: 1111 1111 1111 0000

hex: ff f0

# JUMP GROUP (0x01)

Jumps (branches).

## jmp xx

Jumps to the given location.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0000  jump without condition | 0001 |

### Example:

j 0xff

binary: 0000 0000 0000 0001, 0000 0000 1111 1111

hex: 00 01, 00 ff

## jz xx

Jumps to the given location, if Zero flag is 1.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0001  jump if zero | 0001 |

### Example:

jz 0001

binary: 0000 0000 0001 0001, 0000 0000 0000 0001

hex: 00 11, 00 01

## jnz xx

Jumps to the given location, if Zero flag is 0.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0010  jump if not carry | 0001 |

### Example:

jnz 0001

binary: 0000 0000 0010 0001, 0000 0000 0000 0001

hex: 00 21, 00 01

## jc xx

Jumps to the given location, if Carry flag is 1.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0011  jump if Carry | 0001 |

### Example:

jc 0001

binary: 0000 0000 0011 0001, 0000 0000 0000 0001

hex: 00 31, 00 01

## jnc xx

Jumps to the given location, if Carry flag is 0.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0100  jump if not carry | 0001 |

### Example:

jnc 0001

binary: 0000 0000 0100 0001, 0000 0000 0000 0001

hex: 00 41, 00 01

## jo xx

Jumps to the given location, if Overflow flag is 1.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0101  jump if overflow | 0001 |

### Example:

jo 0001

binary: 0000 0000 0101 0001, 0000 0000 0000 0001

hex: 00 51, 00 01

## jno xx

Jumps to the given location, if Overflow flag is 0.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0110  jump if not overflow | 0001 |

### Example:

jno 0001

binary: 0000 0000 0110 0001, 0000 0000 0000 0001

hex: 00 61, 00 01

## jp xx (jge xx)

Jumps to the given location, if Positive flag is 1. This is the same as jge xx (jump greater or equal).

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0111  jump if positive | 0001 |

### Example:

jp 0001

binary: 0000 0000 0111 0001, 0000 0000 0000 0001

hex: 00 71, 00 01

## jnp xx (js xx)

Jumps to the given location, if Positive flag is 0. The same as js xx (jump if smaller).

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 1000  jump if not positive | 0001 |

### Example:

jnp 0001

binary: 0000 0000 1000 0001, 0000 0000 0000 0001

hex: 00 81, 00 01

## jg xx

Jumps to the given location, if the last operation produced greater condition (Positive == 1 && Zero == 0).

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 1001  jump if greater | 0001 |

### Example:

jg 0001

binary: 0000 0000 1001 0001, 0000 0000 0000 0001

hex: 00 91, 00 01

## jse xx

Jumps to the given location, if the last operation produced smaller or equal condition (Positive == 0 && Zero == 1).

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 1010  jump if greater | 0001 |

### Example:

jse 0001

binary: 0000 0000 1010 0001, 0000 0000 0000 0001

hex: 00 91, 00 01

# CALL GROUP (0x02)

## call xx

Calls the given subroutine.

|  |  |  |
| --- | --- | --- |
| unused | call/ret kind | group |
| xxxx xxxx | 0000  call without condition | 0010 |

### Example:

call 0001

binary: 0000 0000 0000 0010, 0000 0000 0000 0001

hex: 00 02, 00 01

## callz xx

Calls the given location, if Zero flag is 1.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0001  call if zero | 0010 |

### Example:

callz 0001

binary: 0000 0000 0001 0010, 0000 0000 0000 0001

hex: 00 11, 00 01

## callnz xx

Calls the given location, if Zero flag is 0.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0010  call if not zero | 0010 |

### Example:

callnz 0001

binary: 0000 0000 0010 0010, 0000 0000 0000 0001

hex: 00 22, 00 01

## callc xx

Calls the given location, if Carry flag is 1.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0011  call if carry | 0010 |

### Example:

callc 0001

binary: 0000 0000 0011 0010, 0000 0000 0000 0001

hex: 00 32, 00 01

## callnc xx

Calls the given location, if Carry flag is 0.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0100  call if not carry | 0010 |

### Example:

callnc 0001

binary: 0000 0000 0100 0010, 0000 0000 0000 0001

hex: 00 42, 00 01

## callo xx

Calls the given location, if Overflow flag is 1.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0101  call if overflow | 0010 |

### Example:

callo 0001

binary: 0000 0000 0101 0010, 0000 0000 0000 0001

hex: 00 52, 00 01

## callno xx

Calls the given location, if Overflow flag is 0.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0110  call if not overflow | 0010 |

### Example:

callno 0001

binary: 0000 0000 0110 0010, 0000 0000 0000 0001

hex: 00 62, 00 01

## callp xx (callge xx)

Calls the given location, if Positive flag is 1. This is the same as callge xx (call if greater or equal).

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0001  call if positive | 0010 |

### Example:

callp 0001

binary: 0000 0000 0111 0010, 0000 0000 0000 0001

hex: 00 72, 00 01

## callnp xx (calls xx)

Calls the given location, if Positive flag is 0. This is the same as calls xx (call if smaller).

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 1000  jump if not positive | 0010 |

### Example:

callnp 0001

binary: 0000 0000 1000 0010, 0000 0000 0000 0001

hex: 00 82, 00 01

## callg xx

Calls the given location, if the last operation produced greater condition (Positive == 1 && Zero == 0).

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 1001  call if greater | 0010 |

### Example:

callg 0001

binary: 0000 0000 1001 0010, 0000 0000 0000 0001

hex: 00 92, 00 01

## callse xx

Calls the given location, if the last operation produced smaller or equal condition (Positive == 0 && Zero == 1).

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 1010  call if smaller or equal | 0010 |

### Example:

callse 0001

binary: 0000 0000 1010 0010, 0000 0000 0000 0001

hex: 00 a2, 00 01

# LOAD/STORE GROUP (0x03)

## ld regx, [regy]

Loads one word (two bytes) from the location to which regy points.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0000  0=>[register] | 0011 |

### Example:

ld r2, [r1]

binary: 0001 0010 0000 0011

hex: 12 03

## ld reg, [XX]

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxx | bbbb  0-7: r0-r7  8: sp  9: h | 0001  1=>[number] | 0011 |

### Example:

ld r1, [0x0f]

binary: 0000 0001 0001 0011, 0000 0000 0000 1111

hex: 01 13, 00 0f

## ld regx, [regy + XX]

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0010  2=>[reg+number] | 0011 |

### Example:

ld r4, [r3 + 0x0f]

binary: 0011 0100 0010 0011, 0000 0000 0000 1111

hex: 34 23, 00 0f

## ld.b regx, [regy]

Loads one byte from the address to which regy points

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0011  3=>[register] | 0011 |

### Example:

ld.b r2, [r1]

binary: 0001 0010 0011 0011

hex: 12 33

## ld.b reg, [XX]

Loads one byte from the address to which a given number points.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxx | bbbb  0-7: r0-r7  8: sp  9: h | 0100  4=>[number] | 0011 |

### Example:

ld.b r1, [0x0f]

binary: 0000 0001 0100 0011, 0000 0000 0000 1111

hex: 01 43, 00 0f

## ld.b reg, [reg + XX]

Loads one byte from the address to which a sum of a given register plus given number points.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0101  5=>[reg+number] | 0011 |

### Example:

ld.b r4, [r3 + 0x0f]

binary: 0011 0100 0101 0011, 0000 0000 0000 1111

hex: 34 53, 00 0f

## st [regy], regx

Stores a regx to the location to which regy points.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1000  8=>register | 0011 |

### Example:

st [r1], r2

binary: 0010 0001 1001 0011

hex: 21 83

## st [XX], reg

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xxxx | 1009  9=>number | 0011 |

### Example:

st [0x0f], r1

binary: 0001 0000 1001 0011, 0000 0000 0000 1111

hex: 10 93, 00 0f

## st [reg + XX], reg

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1010  10=>reg+number | 0011 |

### Example:

st [r3 + 0x0f], r4

binary: 0100 0011 1010 0011, 0000 0000 0000 1111

hex: 43 a3, 00 0f

## st.b [regy], regx

Stores a byte from regx to the location to which regx points.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1011  11=>register | 0011 |

### Example:

st.b [r1], r2

binary: 0010 0001 1011 0011

hex: 21 b3

## st.b [XX], reg

Stores a byte from reg to the location to which a given number points.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xxxx | 1100  12=>number | 0011 |

### Example:

st.b [0x0f], r1

binary: 0001 0000 1100 0011, 0000 0000 0000 1111

hex: 10 c3, 00 0f

## st.b [regy + XX], regx

Stores a byte from regx to the location to which sum of regx plus a given number points.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1101  13=>reg+number | 0011 |

### Example:

st.b [r3 + 0x0f], r4

binary: 0100 0011 1101 0011, 0000 0000 0000 1111

hex: 43 d3, 00 0f

# ADD/SUB GROUP (0x0004)

## add regx, regy

regx = regx + regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0000  0=>regx = regx + regy | 0100 |

### Example:

add r0, r1

binary: 0001 0000 0000 0100

hex: 10 04

## add reg, xx

regx = regx + xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0001  1=>regx = regx + xx | 0100 |

### Example:

add r0, 2

binary: 0000 0000 0001 0100, 0000 0000 0000 0010

hex: 00 14, 00 02

## add regx, [regy]

regx = regx + memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0010  2=>regx = regx + memory\_content(regy) | 0100 |

### Example:

add r0, [r1]

binary: 0001 0000 0010 0100

hex: 10 24

## add reg, [xx]

regx = regx + memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0011  3=>regx = regx + memory\_content(xx) | 0100 |

### Example:

add r0, [2]

binary: 0000 0000 0011 0100, 0000 0000 0000 0010

hex: 00 34, 00 02

## add regx, [regy + xx]

regx = regx + memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0100  4=>regx = regx + memory\_content(regy + xx) | 0100 |

### Example:

add r0, [r1 + 5]

binary: 0001 0000 0100 0100, 0000 0000 0000 0101

hex: 10 44, 00 02

## add.b regx, [regy]

regx = regx + byte\_memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0101  5=>regx = regx + byte\_memory\_content(regy) | 0100 |

### Example:

add.b r0, [r1]

binary: 0001 0000 0101 0100

hex: 10 54

## add.b reg, [xx]

regx = regx + byte\_memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0110  6=>regx = regx + byte\_memory\_content(xx) | 0100 |

### Example:

add.b r0, [2]

binary: 0000 0000 0110 0100, 0000 0000 0000 0010

hex: 00 64, 00 02

## add.b regx, [regy + xx]

regx = regx + byte\_memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0111  7=>regx = regx + byte\_memory\_content(regy + xx) | 0100 |

### Example:

add.b r0, [r1 + 5]

binary: 0001 0000 0100 0100, 0000 0000 0000 0101

hex: 10 74, 00 02

## sub regx, regy

regx = regx - regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1000  8=>regx = regx - regy | 0100 |

### Example:

sub r0, r1

binary: 0100 0000 1000 0100

hex: 40 84

## sub reg, xx

regx = regx + xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1001  9=>regx = regx + xx | 0100 |

### Example:

sub r0, 2

binary: 0000 0000 1001 0100, 0000 0000 0000 0010

hex: 00 94, 00 02

## sub regx, [regy]

regx = regx - memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1010  10=>regx = regx - memory\_content(regy) | 0100 |

### Example:

sub r0, [r1]

binary: 0001 0000 1010 0100

hex: 10 a4

## sub reg, [xx]

regx = regx - memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1011  11=>regx = regx - memory\_content(xx) | 0100 |

### Example:

sub r0, [2]

binary: 0000 0000 1011 0100, 0000 0000 0000 0010

hex: 00 b4, 00 02

## sub regx, [regy + xx]

regx = regx - memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1100  12=>regx = regx - memory\_content(regy + xx) | 0100 |

### Example:

sub r0, [r1 + 5]

binary: 0001 0000 1100 0100, 0000 0000 0000 0101

hex: 10 c4, 00 05

## sub.b regx, [regy]

regx = regx - byte\_memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1101  13=>regx = regx – byte\_memory\_content(regy) | 0100 |

### Example:

sub.b r0, [r1]

binary: 0001 0000 1101 0100

hex: 10 d4

## sub.b reg, [xx]

regx = regx - byte\_memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1110  14=>regx = byte\_regx - memory\_content(xx) | 0100 |

### Example:

sub.b r0, [2]

binary: 0000 0000 1110 0100, 0000 0000 0000 0010

hex: 00 e4, 00 02

## sub.b regx, [regy + xx]

regx = regx - byte\_memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1111  15=>regx = regx - byte\_memory\_content(regy + xx) | 0100 |

### Example:

sub.b r0, [r1 + 5]

binary: 0001 0000 1111 0100, 0000 0000 0000 0101

hex: 10 f4, 00 05

# AND/OR GROUP (0x0005)

## and regx, regy

regx = regx AND regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0000  0=>regx = regx AND regy | 0101 |

### Example:

and r0, r1

binary: 0100 0000 0000 0101

hex: 40 05

## and reg, xx

regx = regx AND xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0001  1=>regx = regx AND xx | 0101 |

### Example:

and r0, 2

binary: 0000 0000 0001 0101, 0000 0000 0000 0010

hex: 00 15, 00 02

## and regx, [regy]

regx = regx AND memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0010  2=>regx = regx AND memory\_content(regy) | 0101 |

### Example:

and r0, [r1]

binary: 0001 0000 0010 0101

hex: 10 25

## and reg, [xx]

regx = regx AND memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0011  3=>regx = regx AND memory\_content(xx) | 0101 |

### Example:

and r0, [2]

binary: 0000 0000 0011 0101, 0000 0000 0000 0010

hex: 00 35, 00 02

## and regx, [regy + xx]

regx = regx AND memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0100  4=>regx = regx AND memory\_content(regy + xx) | 0101 |

### Example:

and r0, [r1 + 5]

binary: 0001 0000 0100 0101, 0000 0000 0000 0101

hex: 10 45, 00 02

## and regx, [regy]

regx = regx AND memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0101  b=>regx = regx AND byte\_memory\_content(regy) | 0101 |

### Example:

and.b r0, [r1]

binary: 0001 0000 0101 0101

hex: 10 55

## and.b reg, [xx]

regx = regx AND byte\_memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0110  6=>regx = regx AND byte\_memory\_content(xx) | 0101 |

### Example:

and.b r0, [2]

binary: 0000 0000 0110 0101, 0000 0000 0000 0010

hex: 00 65, 00 02

## and.b regx, [regy + xx]

regx = regx AND byte\_memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0111  7=>regx = regx AND byte\_memory\_content(regy + xx) | 0101 |

### Example:

and.b r0, [r1 + 5]

binary: 0001 0000 0111 0101, 0000 0000 0000 0101

hex: 10 75, 00 02

## or regx, regy

regx = regx OR regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1000  8=>regx = regx OR regy | 0101 |

### Example:

or r0, r1

binary: 0100 0000 1000 0101

hex: 40 85

## or reg, xx

regx = regx OR xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1001  9=>regx = regx OR xx | 0101 |

### Example:

or r0, 2

binary: 0000 0000 1001 0101, 0000 0000 0000 0010

hex: 00 95, 00 02

## or regx, [regy]

regx = regx OR memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1010  10=>regx = regx OR memory\_content(regy) | 0101 |

### Example:

or r0, [r1]

binary: 0001 0000 1010 0101

hex: 10 a5

## or reg, [xx]

regx = regx OR memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1011  11=>regx = regx OR memory\_content(xx) | 0101 |

### Example:

or r0, [2]

binary: 0000 0000 1011 0101, 0000 0000 0000 0010

hex: 00 b5, 00 02

## or regx, [regy + xx]

regx = regx OR memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1100  12=>regx = regx OR memory\_content(regy + xx) | 0101 |

### Example:

or r0, [r1 + 5]

binary: 0001 0000 1100 0101, 0000 0000 0000 0101

hex: 10 c5, 00 05

## or.b regx, [regy]

regx = regx OR byte\_memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1101  13=>regx = regx OR byte\_memory\_content(regy) | 0101 |

### Example:

or.b r0, [r1]

binary: 0001 0000 1101 0101

hex: 10 d5

## or.b reg, [xx]

regx = regx OR byte\_memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1110  14=>regx = regx OR byte\_memory\_content(xx) | 0101 |

### Example:

or.b r0, [2]

binary: 0000 0000 1110 0101, 0000 0000 0000 0010

hex: 00 e5, 00 02

## or.b regx, [regy + xx]

regx = regx OR byte\_memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 11111  15=>regx = regx OR byte\_memory\_content(regy + xx) | 0101 |

### Example:

or.b r0, [r1 + 5]

binary: 0001 0000 1111 0101, 0000 0000 0000 0101

hex: 10 f5, 00 05

# XOR/NEG GROUP (0x0006)

## xor regx, regy

regx = regx XOR regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0000  0=>regx = regx XOR regy | 0110 |

### Example:

xor r0, r1

binary: 0100 0000 0000 0110

hex: 40 06

## xor reg, xx

regx = regx XOR xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1011  11=>regx = regx XOR xx | 0110 |

### Example:

xor r0, 2

binary: 0000 0000 0001 0110, 0000 0000 0000 0010

hex: 00 16, 00 02

## xor regx, [regy]

regx = regx XOR memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0010  2=>regx = regx XOR memory\_content(regy) | 0110 |

### Example:

xor r0, [r1]

binary: 0001 0000 0010 0110

hex: 10 26

## xor reg, [xx]

regx = regx XOR memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0011  3=>regx = regx XOR memory\_content(xx) | 0110 |

### Example:

xor r0, [2]

binary: 0000 0000 0011 0110, 0000 0000 0000 0010

hex: 00 36, 00 02

## xor regx, [regy + xx]

regx = regx XOR memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0100  4=>regx = regx XOR memory\_content(regy + xx) | 0110 |

### Example:

xor r0, [r1 + 5]

binary: 0001 0000 0100 0110, 0000 0000 0000 0101

hex: 10 46, 00 05

## xor.b regx, [regy]

regx = regx XOR memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0101  5=>regx = regx XOR byte\_memory\_content(regy) | 0110 |

### Example:

xor.b r0, [r1]

binary: 0001 0000 0101 0110

hex: 10 56

## xor.b reg, [xx]

regx = regx XOR byte\_memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0110  6=>regx = regx XOR byte\_memory\_content(xx) | 0110 |

### Example:

xor.b r0, [2]

binary: 0000 0000 0110 0110, 0000 0000 0000 0010

hex: 00 66, 00 02

## xor.b regx, [regy + xx]

regx = regx XOR byte\_memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0111  7=>regx = regx XOR byte\_memory\_content(regy + xx) | 0110 |

### Example:

xor.b r0, [r1 + 5]

binary: 0001 0000 0111 0110, 0000 0000 0000 0101

hex: 10 76, 00 05

## neg reg

Negates the given register.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xxxx | 1000  8=>neg reg | 0110 |

### Example:

neg r0

binary: 0000 0000 1000 0110

hex: 00 86

## neg [reg]

Negates the memory\_content(reg).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xxxx | 1010  10=> neg memory\_content(reg) | 0110 |

### Example:

neg [r1]

binary: 0001 0000 1010 0110

hex: 01 a6

## neg [xx]

Negates the memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | xxxx | 1011  11=> neg memory\_content(xx) | 0110 |

### Example:

neg [2]

binary: 0000 0000 1011 1010, 0000 0000 0000 0010

hex: 00 b6, 00 02

## neg [reg + xx]

Negates the memory\_content(reg + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xxxx | 1100  12=> neg memory\_content(reg + xx) | 0110 |

### Example:

neg [r1 + 5]

binary: 0001 0000 1100 0110, 0000 0000 0000 0101

hex: 10 c6, 00 05

## neg.b [reg]

Negates the byte\_memory\_content(reg).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xxxx | 1101  13=> neg byte\_memory\_content(reg) | 0110 |

### Example:

neg.b [r1]

binary: 0001 0000 1101 0110

hex: 10 d6

## neg.b [xx]

Negates the byte\_memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | xxxx | 1110  14=> neg byte\_memory\_content(xx) | 0110 |

### Example:

neg.b [2]

binary: 0000 0000 1110 0110, 0000 0000 0000 0010

hex: 00 e6, 00 02

## neg.b [reg + xx]

Negates the byte\_memory\_content(reg + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xxxx | 1111  15=> neg byte\_memory\_content(reg + xx) | 0110 |

### Example:

neg.b [r1 + 5]

binary: 0001 0000 1111 0110, 0000 0000 0000 0101

hex: 10 f6, 00 05

# SHIFT GROUP (0x0007)

## shl regx, regy

regx = regx << regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0000  0=>regx = regx << regy | 0111 |

### Example:

shl r0, r1

binary: 0100 0000 0000 0111

hex: 40 07

## shl reg, xx

regx = regx << xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0001  1=>regx = regx << xx | 0111 |

### Example:

shl r0, 2

binary: 0000 0000 0001 0111, 0000 0000 0000 0010

hex: 00 17, 00 02

## shl regx, [regy]

regx = regx << memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0010  2=>regx = regx << memory\_content(regy) | 0111 |

### Example:

shl r0, [r1]

binary: 0001 0000 0010 0111

hex: 10 27

## shl reg, [xx]

regx = regx << memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0011  3=>regx = regx << memory\_content(xx) | 0111 |

### Example:

shl r0, [2]

binary: 0000 0000 0011 0111, 0000 0000 0000 0010

hex: 00 37, 00 02

## shl regx, [regy + xx]

regx = regx << memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0100  4=>regx = regx << memory\_content(regy + xx) | 0111 |

### Example:

shl r0, [r1 + 5]

binary: 0001 0000 0100 0111, 0000 0000 0000 0101

hex: 10 47, 00 05

## shl.b regx, [regy]

regx = regx << byte\_memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0101  5=>regx = regx << byte\_memory\_content(regy) | 0111 |

### Example:

shl.b r0, [r1]

binary: 0001 0000 0101 0111

hex: 10 57

## shl.b reg, [xx]

regx = regx << byte\_memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0110  6=>regx = regx << byte\_memory\_content(xx) | 0111 |

### Example:

shl.b r0, [2]

binary: 0000 0000 0110 0111, 0000 0000 0000 0010

hex: 00 67, 00 02

## shl.b regx, [regy + xx]

regx = regx << byte\_memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0111  7=>regx = regx << byte\_memory\_content(regy + xx) | 0111 |

### Example:

shl.b r0, [r1 + 5]

binary: 0001 0000 0111 0111, 0000 0000 0000 0101

hex: 10 77, 00 05

## shr regx, regy

regx = regx >> regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1000  8=>regx = regx >> regy | 0111 |

### Example:

shr r0, r1

binary: 0100 0000 0110 0111

hex: 40 87

## shr reg, xx

regx = regx + xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1001  9=>regx = regx >> xx | 0111 |

### Example:

shr r0, 2

binary: 0000 0000 1001 0111, 0000 0000 0000 0010

hex: 00 97, 00 02

## shr regx, [regy]

regx = regx >> memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1010  10=>regx = regx >> memory\_content(regy) | 0111 |

### Example:

shr r0, [r1]

binary: 0001 0000 1010 0111

hex: 10 a7

## shr reg, [xx]

regx = regx << memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1011  11=>regx = regx >> memory\_content(xx) | 0111 |

### Example:

shr r0, [2]

binary: 0000 0000 1011 0111, 0000 0000 0000 0010

hex: 00 b7, 00 02

## shr regx, [regy + xx]

regx = regx >> memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1100  12=>regx = regx >> memory\_content(regy + xx) | 0111 |

### Example:

shr r0, [r1 + 5]

binary: 0001 0000 1100 0111, 0000 0000 0000 0101

hex: 10 c7, 00 05

## shr.b regx, [regy]

regx = regx >> byte\_memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1101  13=>regx = regx >> byte\_memory\_content(regy) | 0111 |

### Example:

shr.b r0, [r1]

binary: 0001 0000 1101 0111

hex: 10 d7

## shr.b reg, [xx]

regx = regx << byte\_memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1110  14=>regx = regx >> byte\_memory\_content(xx) | 0111 |

### Example:

shr.b r0, [2]

binary: 0000 0000 1110 0111, 0000 0000 0000 0010

hex: 00 e7, 00 02

## shr.b regx, [regy + xx]

regx = regx >> byte\_memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1111  15=>regx = regx >> byte\_memory\_content(regy + xx) | 0111 |

### Example:

shr.b r0, [r1 + 5]

binary: 0001 0000 1111 0111, 0000 0000 0000 0101

hex: 10 f7, 00 05

# MUL/DIV GROUP (0x0008)

## mul regx, regy

regx = regx \* regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0000  0=>regx = lower\_16\_bits(regx \* regy)  h = upper\_16\_bits(regx \* regy) | 1000 |

### Example:

mul r0, r1

binary: 0001 0000 0000 1000

hex: 10 08

## mul reg, XX

reg = reg \* XX

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0001  1 =>reg = lower\_16\_bits(reg \* number)  h = upper\_16\_bits(reg \* number) | 1010 |

### Example:

mul r1, 5

binary: 0000 0001 0001 1000, 0000 0000 0000 0101

hex: 01 18, 00 05

## mul regx, [regy]

regx = regx \* memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0010  2=>regx = regx \* memory\_content(regy)  h = upper\_16\_bits(reg \* number) | 1000 |

### Example:

mul r0, [r1]

binary: 0001 0000 0010 1000

hex: 10 28

## mul reg, [xx]

regx = regx \* memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0011  3=>regx = regx \* memory\_content(xx)  h = upper\_16\_bits(reg \* number) | 1000 |

### Example:

mul r0, [2]

binary: 0000 0000 0011 1000, 0000 0000 0000 0010

hex: 00 38, 00 02

## mul regx, [regy + xx]

regx = regx \* memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0100  4=>regx = regx \* memory\_content(regy + xx)  h = upper\_16\_bits(reg \* number) | 1000 |

### Example:

mul r0, [r1 + 5]

binary: 0001 0000 0100 1000, 0000 0000 0000 0101

hex: 10 48, 00 05

## mul.b regx, [regy]

regx = regx \* byte\_memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0101  5=>regx = regx \* byte\_memory\_content(regy)  h = upper\_16\_bits(reg \* number) | 1000 |

### Example:

mul.b r0, [r1]

binary: 0001 0000 0101 1000

hex: 10 58

## mul.b reg, [xx]

regx = regx \* byte\_memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0110  6=>regx = regx \* byte\_memory\_content(xx)  h = upper\_16\_bits(reg \* number) | 1000 |

### Example:

mul.b r0, [2]

binary: 0000 0000 0110 1000, 0000 0000 0000 0010

hex: 00 68, 00 02

## mul.b regx, [regy + xx]

regx = regx \* byte\_memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0111  7=>regx = regx \* byte\_memory\_content(regy + xx)  h = upper\_16\_bits(reg \* number) | 1000 |

### Example:

mul.b r0, [r1 + 5]

binary: 0001 0000 0111 1000, 0000 0000 0000 0101

hex: 10 78, 00 05

## div regx, regy

regx = regx / regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1000  8=>regx = regx / regy  h = regx % regy | 1000 |

### Example:

div r0, r1

binary: 0001 0000 1000 1000

hex: 10 88

## div reg, XX

reg = reg / XX

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1001  9 =>reg = reg / number  h = reg % number | 1000 |

### Example:

div r1, 5

binary: 0000 0001 1001 1000, 0000 0000 0000 0101

hex: 01 98, 00 05

## div regx, [regy]

regx = regx / memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1010  10=>regx = regx / memory\_content(regy)  h = regx % regy | 1000 |

### Example:

div r0, [r1]

binary: 0001 0000 1010 1000

hex: 10 a8

## div reg, [xx]

regx = regx / memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1011  11=>regx = regx / memory\_content(xx)  h = regx % regy | 1000 |

### Example:

div r0, [2]

binary: 0000 0000 1011 1000, 0000 0000 0000 0010

hex: 00 b8, 00 02

## div regx, [regy + xx]

regx = regx / memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1100  12=>regx = regx / memory\_content(regy + xx)  h = regx % regy | 1000 |

### Example:

div r0, [r1 + 5]

binary: 0001 0000 1100 1000, 0000 0000 0000 0101

hex: 10 c8, 00 05

## div.b regx, [regy]

regx = regx / byte\_memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1010  10=>regx = regx / byte\_memory\_content(regy)  h = regx % regy | 1000 |

### Example:

div.b r0, [r1]

binary: 0001 0000 1101 1000

hex: 10 d8

## div reg, [xx]

regx = regx / byte\_memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1110  14=>regx = regx / byte\_memory\_content(xx)  h = regx % regy | 1000 |

### Example:

div.b r0, [2]

binary: 0000 0000 1110 1000, 0000 0000 0000 0010

hex: 00 e8, 00 02

## div.b regx, [regy + xx]

regx = regx / byte\_memory\_content(regy + xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 1111  15=>regx = regx / byte\_memory\_content(regy + xx)  h = regx % regy | 1000 |

### Example:

div.b r0, [r1 + 5]

binary: 0001 0000 1111 1000, 0000 0000 0000 0101

hex: 10 f8, 00 05

# INC/DEC GROUP (0x0009)

## inc reg

Increments the given register.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0000  0=>inc reg | 1001 |

### Example:

inc r0

binary: 0000 0000 0000 1001

hex: 00 09

## inc [reg]

Increments the memory\_content(reg).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0001  1=> inc memory\_content(reg) | 1001 |

### Example:

inc [r1]

binary: 0000 0001 0001 1001

hex: 01 19

## inc [xx]

Increments the memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | xxxx | 0010  2=> inc memory\_content(xx) | 1001 |

### Example:

inc [2]

binary: 0000 0000 0010 1001, 0000 0000 0000 0010

hex: 00 29, 00 02

## inc [reg + xx]

Increments the memory\_content(reg + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0011  3=> inc memory\_content(reg + xx) | 1001 |

### Example:

inc [r1 + 5]

binary: 0000 0001 0011 1001, 0000 0000 0000 0101

hex: 01 39, 00 05

## inc.b [reg]

Increments the byte\_memory\_content(reg).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0100  4=> inc byte\_memory\_content(reg) | 1001 |

### Example:

inc.b [r1]

binary: 0000 0001 0100 1001

hex: 01 49

## inc.b [xx]

Increments the byte\_memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | xxxx | 0101  5=> inc byte\_memory\_content(xx) | 1001 |

### Example:

inc.b [2]

binary: 0000 0000 0101 1001, 0000 0000 0000 0010

hex: 00 59, 00 02

## inc [reg + xx]

Increments the memory\_content(reg + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0110  6=> inc byte\_memory\_content(reg + xx) | 1001 |

### Example:

inc.b [r1 + 5]

binary: 0000 0001 0110 1001, 0000 0000 0000 0101

hex: 01 69, 00 05

## dec reg

Decrements the given register.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1000  8=>dec reg | 1001 |

### Example:

dec r0

binary: 0000 0000 1000 1001

hex: 00 89

## dec [reg]

Decrements the memory\_content(reg).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1001  9=> dec memory\_content(reg) | 1001 |

### Example:

dec [r1]

binary: 0000 0001 1001 1001

hex: 01 99

## dec [xx]

Decrements the memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | xxxx | 1010  10=> dec memory\_content(xx) | 1001 |

### Example:

dec [2]

binary: 0000 0000 1010 1001, 0000 0000 0000 0010

hex: 00 a9, 00 02

## dec [reg + xx]

Decrements the memory\_content(reg + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1011  11=> dec memory\_content(reg + xx) | 1001 |

### Example:

dec [r1 + 5]

binary: 0000 0001 1011 1001, 0000 0000 0000 0101

hex: 01 b9, 00 05

## dec.b [reg]

Decrements the byte\_memory\_content(reg).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1100  12=> dec byte\_memory\_content(reg) | 1001 |

### Example:

dec.b [r1]

binary: 0000 0001 1100 1001

hex: 01 c9

## dec.b [xx]

Decrements the byte\_memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | xxxx | 1101  13=> dec byte\_memory\_content(xx) | 1001 |

### Example:

dec.b [2]

binary: 0000 0000 1101 1001, 0000 0000 0000 0010

hex: 00 d9, 00 02

## dec.b [reg + xx]

Decrements the byte\_memory\_content(reg + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1110  14=> dec byte\_memory\_content(reg + xx) | 1001 |

### Example:

dec.b [r1 + 5]

binary: 0000 0001 1110 1001, 0000 0000 0000 0101

hex: 01 e9, 00 05

# CMP/INV GROUP (0x000a)

## cmp regx, regy

Compares regx and regy, by subtracting them and not storing the result; just setting the flags.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0000  0=>cmp regx, regy | 1010 |

### Example:

cmp r0, r1

binary: 0001 0000 0000 1010

hex: 10 0a

## cmp reg, xx

Compares reg and number xx, by subtracting them and not storing the result; just setting the flags.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0001  1=>cmp reg, xx | 1010 |

### Example:

cmp r0, 2

binary: 0000 0000 0001 1010, 0000 0000 0000 0010

hex: 00 1a, 00 02

## cmp regx, [regy]

Compare regx and memory\_content(regy).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0010  2=> compare regx and memory\_content(regy) | 1010 |

### Example:

cmp r0, [r1]

binary: 0001 0000 0010 1010

hex: 10 2a

## cmp reg, [xx]

Compare regx and memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0011  3=> compare regx and memory\_content(xx) | 1010 |

### Example:

cmp r0, [2]

binary: 0000 0000 0011 1010, 0000 0000 0000 0010

hex: 00 3a, 00 02

## cmp regx, [regy + xx]

Compare regx and memory\_content(regy + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0100  4=> Compare regx and memory\_content(regy + xx) | 1010 |

### Example:

cmp r0, [r1 + 5]

binary: 0001 0000 0100 1010, 0000 0000 0000 0101

hex: 10 4a, 00 05

## cmp.b regx, [regy]

Compare regx and byte\_memory\_content(regy).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0101  5=> compare regx and byte\_memory\_content(regy) | 1010 |

### Example:

cmp.b r0, [r1]

binary: 0001 0000 0101 1010

hex: 10 5a

## cmp.b reg, [xx]

Compare regx and byte\_memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0110  6=> compare regx and byte\_memory\_content(xx) | 1010 |

### Example:

cmp.b r0, [2]

binary: 0000 0000 0110 1010, 0000 0000 0000 0010

hex: 00 6a, 00 02

## cmp.b regx, [regy + xx]

Compare regx and byte\_memory\_content(regy + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0111  7=> Compare regx and byte\_memory\_content(regy + xx) | 1010 |

### Example:

cmp.b r0, [r1 + 5]

binary: 0001 0000 0111 1010, 0000 0000 0000 0101

hex: 10 7a, 00 05

## inv reg

Inverts the given register.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1000  8=>neg reg | 1010 |

### Example:

neg r0

binary: 0000 0000 1000 1010

hex: 00 8a

## inv [reg]

Negates the memory\_content(reg).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1001  9=> inv memory\_content(reg) | 1010 |

### Example:

inv [r1]

binary: 0000 0001 1001 1010

hex: 01 9a

## inv [xx]

Inverts the memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | xxxx | 1010  10=> inv memory\_content(xx) | 1010 |

### Example:

inv [2]

binary: 0000 0000 1010 1010, 0000 0000 0000 0010

hex: 00 aa, 00 02

## inv [reg + xx]

Inverts the memory\_content(reg + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1011  11=> inv memory\_content(reg + xx) | 1010 |

### Example:

inv [r1 + 5]

binary: 0000 0001 1011 1010, 0000 0000 0000 0101

hex: 01 ba, 00 05

## inv.b [reg]

Negates the byte\_memory\_content(reg).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1001  12=> inv byte\_memory\_content(reg) | 1010 |

### Example:

inv.b [r1]

binary: 0000 0001 1100 1010

hex: 01 ca

## inv.b [xx]

Inverts the byte\_memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | xxxx | 1101  13=> inv byte\_memory\_content(xx) | 1010 |

### Example:

inv.b [2]

binary: 0000 0000 1101 1010, 0000 0000 0000 0010

hex: 00 da, 00 02

## inv.b [reg + xx]

Inverts the byte\_memory\_content(reg + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 1110  14=> inv byte\_memory\_content(reg + xx) | 1010 |

### Example:

inv.b [r1 + 5]

binary: 0000 0001 1110 1010, 0000 0000 0000 0101

hex: 01 ea, 00 05