**16-bit CPU**

**@2018 by Milan Vidaković**

Contents

[General information 9](#_Toc517635757)

[VGA text mode 9](#_Toc517635758)

[UART interface 10](#_Toc517635759)

[Instruction set 11](#_Toc517635760)

[NOP/HALT GROUP (0x00) 14](#_Toc517635761)

[nop 14](#_Toc517635762)

[halt 14](#_Toc517635763)

[MOV GROUP (0x01) 14](#_Toc517635764)

[mov regx, regy 14](#_Toc517635765)

[Example: 14](#_Toc517635766)

[mov reg, XX 14](#_Toc517635767)

[Example: 15](#_Toc517635768)

[in reg, [XX] 15](#_Toc517635769)

[Example: 15](#_Toc517635770)

[out [XX], reg 15](#_Toc517635771)

[Example: 15](#_Toc517635772)

[LOAD GROUP (0x02) 15](#_Toc517635773)

[ld reg, [reg] 15](#_Toc517635774)

[Example: 15](#_Toc517635775)

[ld reg, [XX] 16](#_Toc517635776)

[Example: 16](#_Toc517635777)

[ld reg, [reg + XX] 16](#_Toc517635778)

[Example: 16](#_Toc517635779)

[ld reg, [regx + regy + XX] 16](#_Toc517635780)

[Example: 16](#_Toc517635781)

[STORE GROUP (0x03) 17](#_Toc517635782)

[st [reg], reg 17](#_Toc517635783)

[Example: 17](#_Toc517635784)

[st [XX], reg 17](#_Toc517635785)

[Example: 17](#_Toc517635786)

[st [reg + XX], reg 17](#_Toc517635787)

[Example: 17](#_Toc517635788)

[st [regx + regy + XX], reg 17](#_Toc517635789)

[Example: 18](#_Toc517635790)

[JUMP GROUP (0x0004) 18](#_Toc517635791)

[jmp xx 18](#_Toc517635792)

[Example: 18](#_Toc517635793)

[jz xx 18](#_Toc517635794)

[Example: 18](#_Toc517635795)

[jnz xx 19](#_Toc517635796)

[Example: 19](#_Toc517635797)

[jc xx 19](#_Toc517635798)

[Example: 19](#_Toc517635799)

[jnc xx 19](#_Toc517635800)

[Example: 19](#_Toc517635801)

[jo xx 19](#_Toc517635802)

[Example: 20](#_Toc517635803)

[jno xx 20](#_Toc517635804)

[Example: 20](#_Toc517635805)

[jp xx 20](#_Toc517635806)

[Example: 20](#_Toc517635807)

[jnp xx 20](#_Toc517635808)

[Example: 20](#_Toc517635809)

[CALL/RET GROUP (0x0005) 21](#_Toc517635810)

[call xx 21](#_Toc517635811)

[Example: 21](#_Toc517635812)

[callz xx 21](#_Toc517635813)

[Example: 21](#_Toc517635814)

[callnz xx 21](#_Toc517635815)

[Example: 21](#_Toc517635816)

[callc xx 22](#_Toc517635817)

[Example: 22](#_Toc517635818)

[callnc xx 22](#_Toc517635819)

[Example: 22](#_Toc517635820)

[callo xx 22](#_Toc517635821)

[Example: 22](#_Toc517635822)

[callno xx 22](#_Toc517635823)

[Example: 23](#_Toc517635824)

[callp xx 23](#_Toc517635825)

[Example: 23](#_Toc517635826)

[callnp xx 23](#_Toc517635827)

[Example: 23](#_Toc517635828)

[ret 23](#_Toc517635829)

[Example: 23](#_Toc517635830)

[iret 24](#_Toc517635831)

[Example: 24](#_Toc517635832)

[ADD/SUB GROUP (0x0006) 24](#_Toc517635833)

[add regx, regy 24](#_Toc517635834)

[Example: 24](#_Toc517635835)

[add reg, xx 24](#_Toc517635836)

[Example: 24](#_Toc517635837)

[add regx, [regy] 25](#_Toc517635838)

[Example: 25](#_Toc517635839)

[add reg, [xx] 25](#_Toc517635840)

[Example: 25](#_Toc517635841)

[add regx, [regy + xx] 25](#_Toc517635842)

[Example: 25](#_Toc517635843)

[sub regx, regy 26](#_Toc517635844)

[Example: 26](#_Toc517635845)

[sub reg, xx 26](#_Toc517635846)

[Example: 26](#_Toc517635847)

[sub regx, [regy] 26](#_Toc517635848)

[Example: 26](#_Toc517635849)

[sub reg, [xx] 26](#_Toc517635850)

[Example: 27](#_Toc517635851)

[sub regx, [regy + xx] 27](#_Toc517635852)

[Example: 27](#_Toc517635853)

[AND/OR/XOR GROUP (0x0007) 27](#_Toc517635854)

[and regx, regy 27](#_Toc517635855)

[Example: 27](#_Toc517635856)

[and reg, xx 28](#_Toc517635857)

[Example: 28](#_Toc517635858)

[and regx, [regy] 28](#_Toc517635859)

[Example: 28](#_Toc517635860)

[and reg, [xx] 28](#_Toc517635861)

[Example: 28](#_Toc517635862)

[and regx, [regy + xx] 28](#_Toc517635863)

[Example: 29](#_Toc517635864)

[or regx, regy 29](#_Toc517635865)

[Example: 29](#_Toc517635866)

[or reg, xx 29](#_Toc517635867)

[Example: 29](#_Toc517635868)

[or regx, [regy] 29](#_Toc517635869)

[Example: 30](#_Toc517635870)

[or reg, [xx] 30](#_Toc517635871)

[Example: 30](#_Toc517635872)

[or regx, [regy + xx] 30](#_Toc517635873)

[Example: 30](#_Toc517635874)

[xor regx, regy 30](#_Toc517635875)

[Example: 31](#_Toc517635876)

[xor reg, xx 31](#_Toc517635877)

[Example: 31](#_Toc517635878)

[xor regx, [regy] 31](#_Toc517635879)

[Example: 31](#_Toc517635880)

[xor reg, [xx] 31](#_Toc517635881)

[Example: 32](#_Toc517635882)

[xor regx, [regy + xx] 32](#_Toc517635883)

[Example: 32](#_Toc517635884)

[CMP GROUP (0x0008) 32](#_Toc517635885)

[cmp regx, regy 32](#_Toc517635886)

[Example: 32](#_Toc517635887)

[cmp reg, xx 32](#_Toc517635888)

[Example: 33](#_Toc517635889)

[cmp regx, [regy] 33](#_Toc517635890)

[Example: 33](#_Toc517635891)

[cmp reg, [xx] 33](#_Toc517635892)

[Example: 33](#_Toc517635893)

[cmp regx, [regy + xx] 33](#_Toc517635894)

[Example: 34](#_Toc517635895)

[INC/DEC/NEG GROUP (0x0009) 34](#_Toc517635896)

[inc reg 34](#_Toc517635897)

[Example: 34](#_Toc517635898)

[inc [reg] 34](#_Toc517635899)

[Example: 34](#_Toc517635900)

[inc [xx] 34](#_Toc517635901)

[Example: 35](#_Toc517635902)

[inc [reg + xx] 35](#_Toc517635903)

[Example: 35](#_Toc517635904)

[dec reg 35](#_Toc517635905)

[Example: 35](#_Toc517635906)

[dec [reg] 35](#_Toc517635907)

[Example: 35](#_Toc517635908)

[dec [xx] 36](#_Toc517635909)

[Example: 36](#_Toc517635910)

[dec [reg + xx] 36](#_Toc517635911)

[Example: 36](#_Toc517635912)

[neg reg 36](#_Toc517635913)

[Example: 36](#_Toc517635914)

[neg [reg] 37](#_Toc517635915)

[Example: 37](#_Toc517635916)

[neg [xx] 37](#_Toc517635917)

[Example: 37](#_Toc517635918)

[neg [reg + xx] 37](#_Toc517635919)

[Example: 37](#_Toc517635920)

[MUL/DIV GROUP (0x000A) 38](#_Toc517635921)

[mul regx, regy 38](#_Toc517635922)

[Example: 38](#_Toc517635923)

[mul reg, XX 38](#_Toc517635924)

[Example: 38](#_Toc517635925)

[mul regx, [regy] 38](#_Toc517635926)

[Example: 38](#_Toc517635927)

[mul reg, [xx] 39](#_Toc517635928)

[Example: 39](#_Toc517635929)

[mul regx, [regy + xx] 39](#_Toc517635930)

[Example: 39](#_Toc517635931)

[div regx, regy 39](#_Toc517635932)

[Example: 39](#_Toc517635933)

[div reg, XX 39](#_Toc517635934)

[Example: 40](#_Toc517635935)

[div regx, [regy] 40](#_Toc517635936)

[Example: 40](#_Toc517635937)

[div reg, [xx] 40](#_Toc517635938)

[Example: 40](#_Toc517635939)

[div regx, [regy + xx] 40](#_Toc517635940)

[Example: 41](#_Toc517635941)

[PUSH/POP GROUP (0x0b) 41](#_Toc517635942)

[push reg 41](#_Toc517635943)

[Example: 41](#_Toc517635944)

[push XX 41](#_Toc517635945)

[Example: 41](#_Toc517635946)

[pop reg 41](#_Toc517635947)

[Example: 42](#_Toc517635948)

[SHIFT GROUP (0x000c) 42](#_Toc517635949)

[shl regx, regy 42](#_Toc517635950)

[Example: 42](#_Toc517635951)

[shl reg, xx 42](#_Toc517635952)

[Example: 42](#_Toc517635953)

[shl regx, [regy] 42](#_Toc517635954)

[Example: 43](#_Toc517635955)

[shl reg, [xx] 43](#_Toc517635956)

[Example: 43](#_Toc517635957)

[shl regx, [regy + xx] 43](#_Toc517635958)

[Example: 43](#_Toc517635959)

[shr regx, regy 43](#_Toc517635960)

[Example: 44](#_Toc517635961)

[shr reg, xx 44](#_Toc517635962)

[Example: 44](#_Toc517635963)

[shr regx, [regy] 44](#_Toc517635964)

[Example: 44](#_Toc517635965)

[shr reg, [xx] 44](#_Toc517635966)

[Example: 44](#_Toc517635967)

[shr regx, [regy + xx] 45](#_Toc517635968)

[Example: 45](#_Toc517635969)

16-bit CPU

# General information

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).

The address bus is 16 bits wide, addressing 65536 memory locations (words). Data bus is also 16 bits wide, having each memory location 16 bits wide. This gives 65536 16-bits words, or 128KB.

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

It has two interrupts: IRQ0 and IRQ1. IRQ0 is connected to the KEY2 of the DE0-NANO, while IDQ1 is conencted to the UART. Whenever a byte comes to the UART, it generates an IRQ1. Interrupt causes CPU to push flags to the stack, then to push PC to the stack and then to jump to the location designated for the CPU:

* for the IRQ0, it is 0x0004, and
* for the IRQ1, it is 0x0008.

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 put the IRET instruction. It pops the return address, and then pops the flags register, and then goes back into the interrupted 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. 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), it is white. 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.

# 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. |

# Instruction set

|  |  |  |  |
| --- | --- | --- | --- |
| Group number | Group name | Group members | Group description |
| 0 | [NOP/HALT](#_NOP/HALT_GROUP_(0x00)) | nop – no operation,  halt – freezes the cpu |  |
| 1 | [MOV/IN/OUT](#_MOV_GROUP_(0x01)) | mov reg, xx  mov reg, reg  in reg, [xx]  out [xx], reg | Inter-register operations  destination: register  source: number following the opcode, or register  direction: reg <-- xx, or reg <-- reg. |
| 2 | [LOAD](#_LOAD_GROUP_(0x02)) | ld reg, [xx]  ld reg, [reg]  ld reg, [reg + xx] | Load from memory into the register  destination: register  source: memory address given by the number, or by the register, or by the register+number. |
| 3 | [STORE](#_STORE_GROUP_(0x03)) | st [xx], reg  st [reg], reg  st [reg + xx], reg | 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 | [JUMP](#_JUMP_GROUP_(0x0004)) | jmp xx  jc xx  jnc xx  jz xx  jnz xx  jo xx  jno xx  jp xx  jnp xx | Jump to the given location. |
| 5 | [CALL/RET](#_CALL/RET_GROUP_(0x0005)) | call xx  callc xx  callnc xx  callz xx  callnz xx  callo xx  callno xx  callp xx  callnp xx  ret  iret | Call subroutine / return from the subroutine. |
| 6 | [ADD/SUB](#_ADD/SUB_GROUP_(0x0006)) | add reg, reg  add reg, xx  add reg, [reg]  add reg, [xx]  add reg, [reg + xx]  sub reg, reg  sub reg, xx  sub reg, [reg]  sub reg, [xx]  sub reg, [reg + xx] | Addition / subraction. |
| 7 | [AND/OR/XOR](#_AND/OR/XOR_GROUP_(0x0007)) | and reg, reg  and reg, xx  and reg, [reg]  and reg, [xx]  and reg, [reg + xx]  or reg, reg  or reg, xx  or reg, [reg]  or reg, [xx]  or reg, [reg + xx]  xor reg, reg  xor reg, xx  xor reg, [reg]  xor reg, [xx]  xor reg, [reg + xx] | And / or / xor operations.  destination: register  source: memory address given by the number, or by the register, or by the register+number, or by the register+register+number. |
| 8 | [CMP](#_CMP_GROUP_(0x0008)) | cmp reg, reg  cmp reg, xx  cmp reg, [reg]  cmp reg, [xx]  cmp reg, [reg + xx] | Compare registers with registers/memory. |
| 9 | [INC/DEC/NOT](#_INC/DEC/NEG_GROUP_(0x0009)) | inc reg  inc [reg]  inc [xx]  inc [reg + xx]  dec reg  dec [reg]  dec [xx]  dec [reg + xx]  not reg  not [reg]  not [xx]  not [reg + xx] | Increment, decrement and negation. |
| 10 | [MUL/DIV](#_MUL/DIV_GROUP_(0x000A)) | mul reg, reg  mul reg, xx  mul reg, [reg]  mul reg, [xx]  mul reg, [reg + xx]  div reg, reg  div reg, xx  div reg, [reg]  div reg, [xx]  div reg, [reg + xx] | Multiplies/divides registers with registers/memory. |
| 11 | [PUSH/POP](#_PUSH/POP_GROUP_(0x0b)) | push reg  push xx  pop reg | Push/pop. |
| 12 | [SHIFT](#_SHIFT_GROUP_(0x000c)) | shl reg, reg  shl reg, xx  shl reg, [reg]  shl reg, [xx]  shl reg, [reg + xx]  shr reg, reg  shr reg, xx  shr reg, [reg]  shr reg, [xx]  shr reg, [reg + xx] | Shifts registers with registers/memory. |

# NOP/HALT GROUP (0x00)

## nop

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

Example:

nop

binary: 0000 0000 0000 0000

hex: 00 00

## halt

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

Example:

halt

binary: 1111 1111 1111 0000

hex: ff f0

# MOV GROUP (0x01)

## mov regx, regy

regx <-- regy

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

### Example:

mov r2, r1

binary: 0001 0010 0000 0001

hex: 12 01

## mov reg, XX

reg <-- XX

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

### Example:

mov r1, 0x0f

binary: 0000 0001 0001 0001, 0000 0000 0000 1111

hex: 01 11, 00 0f

## in reg, [XX]

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | xbbb  0-7: r0-r7 | 0010  2 =>in reg, [xx] | 0001 |

### Example:

in r1, [0x0f]

binary: 0000 0001 0010 0001, 0000 0000 0000 1111

hex: 01 21, 00 0f

## out [XX], reg

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | xbbb  0-7: r0-r7 | 0011  3 =>out [xx], reg | 0001 |

### Example:

out r1, 0x0f

binary: 0000 0001 0011 0001, 0000 0000 0000 1111

hex: 01 21, 00 0f

# LOAD GROUP (0x02)

## ld reg, [reg]

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xbbb  0-7: r0-r7 | 0000  00NN  NN==00=>register | 0010 |

### Example:

ld r2, [r1]

binary: 0001 0010 0000 0010

hex: 12 02

## ld reg, [XX]

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxx | xbbb  0-7: r0-r7 | 0001  00NN  NN==01 =>number | 0010 |

### Example:

ld r1, [0x0f]

binary: 0000 0001 0001 0010, 0000 0000 0000 1111

hex: 01 12, 00 0f

## ld reg, [reg + XX]

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xbbb  0-7: r0-r7 | 0010  00NN  NN==10 =>reg+number | 0010 |

### Example:

ld r4, [r3 + 0x0f]

binary: 0011 0100 0010 0010, 0000 0000 0000 1111

hex: 34 22, 00 0f

## ld reg, [regx + regy + XX]

Adds regx+regy+numberXX, reads from that location and stores the content into the reg.

Note: regy can be r0-r3 only. regx and reg can be any regular register, excluding sp.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| from | to | reg | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xbbb  0-7: r0-r7 | bb  0-3: r0-r3 | 11  NN  NN==11 =>reg+reg+number | 0010 |

### Example:

ld r5, [r3 + r0 + 0x0f]

binary: 0011 0101 0011 0010, 0000 0000 0000 1111

hex: 35 32, 00 0f

# STORE GROUP (0x03)

## st [reg], reg

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | 0000  00NN  NN==00=>register | 0011 |

### Example:

st [r1], r2

binary: 0010 0001 0000 0011

hex: 21 03

## st [XX], reg

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | xxxx | 0001  00NN  NN==01 =>number | 0011 |

### Example:

st [0x0f], r1

binary: 0001 0000 0001 0011, 0000 0000 0000 1111

hex: 10 13, 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 | 0010  00NN  NN==10 =>reg+number | 0011 |

### Example:

st [r3 + 0x0f], r4

binary: 0100 0011 0010 0011, 0000 0000 0000 1111

hex: 43 23, 00 0f

## st [regx + regy + XX], reg

Adds regx+regy+numberXX and stores reg into that location.

Note: regy can be r0-r3 only. regx and reg can be any regular register.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| from | to | reg | what | group |
| bbbb  0-7: r0-r7  8: sp  9: h | bbbb  0-7: r0-r7  8: sp  9: h | bb  0-3: r0-r3 | 11  NN  NN==11 =>reg+reg+number | 0011 |

### Example:

st [r3 + r0 + 0x0f], r1

binary: 0001 0011 0011 0011, 0000 0000 0000 1111

hex: 13 33, 00 0f

# JUMP GROUP (0x0004)

jo xx

jno xx

jp xx

jnp xx

## jmp xx

Jumps to the given location.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0000  jump without condition | 0100 |

### Example:

j 0001

binary: 0000 0000 0000 0100, 0000 0000 0000 0001

hex: 00 04, 00 01

## jz xx

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

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0001  jump if zero | 0100 |

### Example:

jz 0001

binary: 0000 0000 0001 0100, 0000 0000 0000 0001

hex: 00 14, 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 | 0100 |

### Example:

jnz 0001

binary: 0000 0000 0010 0100, 0000 0000 0000 0001

hex: 00 24, 00 01

## jc xx

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

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0011  jump if Carry | 0100 |

### Example:

jc 0001

binary: 0000 0000 0011 0100, 0000 0000 0000 0001

hex: 00 34, 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 | 0100 |

### Example:

jnc 0001

binary: 0000 0000 0100 0100, 0000 0000 0000 0001

hex: 00 44, 00 01

## jo xx

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

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0101  jump if overflow | 0100 |

### Example:

jo 0001

binary: 0000 0000 0101 0100, 0000 0000 0000 0001

hex: 00 54, 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 | 0100 |

### Example:

jno 0001

binary: 0000 0000 0110 0100, 0000 0000 0000 0001

hex: 00 64, 00 01

## jp xx

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

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0111  jump if positive | 0100 |

### Example:

jp 0001

binary: 0000 0000 0111 0100, 0000 0000 0000 0001

hex: 00 74, 00 01

## jnp xx

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

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 1000  jump if not positive | 0100 |

### Example:

jnp 0001

binary: 0000 0000 1000 0100, 0000 0000 0000 0001

hex: 00 84, 00 01

# CALL/RET GROUP (0x0005)

## call xx

Calls the given subroutine.

|  |  |  |
| --- | --- | --- |
| unused | call/ret kind | group |
| xxxx xxxx | 0000  jump without condition | 0101 |

### Example:

call 0001

binary: 0000 0000 0000 0101, 0000 0000 0000 0001

hex: 00 05, 00 01

## callz xx

Calls the given location, if Zero flag is 1.

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

### Example:

callz 0001

binary: 0000 0000 0001 0101, 0000 0000 0000 0001

hex: 00 15, 00 01

## callnz xx

Calls the given location, if Zero flag is 0.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0010  jump if not zero | 0101 |

### Example:

callnz 0001

binary: 0000 0000 0010 0101, 0000 0000 0000 0001

hex: 00 25, 00 01

## callc xx

Calls the given location, if Carry flag is 1.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0011  jump if carry | 0101 |

### Example:

callc 0001

binary: 0000 0000 0011 0101, 0000 0000 0000 0001

hex: 00 35, 00 01

## callnc xx

Calls the given location, if Carry flag is 0.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0100  jump if not carry | 0101 |

### Example:

callnc 0001

binary: 0000 0000 0100 0101, 0000 0000 0000 0001

hex: 00 45, 00 01

## callo xx

Calls the given location, if Overflow flag is 1.

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

### Example:

callo 0001

binary: 0000 0000 0101 0101, 0000 0000 0000 0001

hex: 00 55, 00 01

## callno xx

Calls the given location, if Overflow flag is 0.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 0110  jump if not overflow | 0101 |

### Example:

callno 0001

binary: 0000 0000 0110 0101, 0000 0000 0000 0001

hex: 00 65, 00 01

## callp xx

Calls the given location, if Positive flag is 1.

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

### Example:

callp 0001

binary: 0000 0000 0111 0101, 0000 0000 0000 0001

hex: 00 75, 00 01

## callnp xx

Calls the given location, if Positive flag is 0.

|  |  |  |
| --- | --- | --- |
| unused | jump kind | group |
| xxxx xxxx | 1000  jump if not positive | 0101 |

### Example:

callnp 0001

binary: 0000 0000 1000 0101, 0000 0000 0000 0001

hex: 00 85, 00 01

## ret

Returns from the subroutine.

|  |  |  |
| --- | --- | --- |
| unused | call/ret kind | group |
| xxxx xxxx | 1001  return | 0101 |

### Example:

ret

binary: 0000 0000 1001 0101

hex: 00 95

## iret

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

|  |  |  |
| --- | --- | --- |
| unused | call/ret kind | group |
| xxxx xxxx | 1010  return | 0101 |

### Example:

ret

binary: 0000 0000 1010 0101

hex: 00 A5

# ADD/SUB GROUP (0x0006)

## add regx, regy

regx = regx + regy

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

### Example:

add r0, r1

binary: 0001 0000 0000 0110

hex: 10 06

## add reg, xx

regx = regx + xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp | 0001  1=>regx = regx + xx | 0110 |

### Example:

add r0, 2

binary: 0000 0000 0001 0110, 0000 0000 0000 0010

hex: 00 16, 00 02

## add regx, [regy]

regx = regx + memory\_content(regy)

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

### Example:

add r0, [r1]

binary: 0001 0000 0010 0110

hex: 10 26

## add reg, [xx]

regx = regx + memory\_content(xx)

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

### Example:

add r0, [2]

binary: 0000 0000 0011 0110, 0000 0000 0000 0010

hex: 00 36, 00 02

## add regx, [regy + xx]

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

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

### Example:

add r0, [r1 + 5]

binary: 0001 0000 0100 0110, 0000 0000 0000 0101

hex: 10 46, 00 02

## sub regx, regy

regx = regx - regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 0110  6=>regx = regx - regy | 0110 |

### Example:

sub r0, r1

binary: 0100 0000 0110 0110

hex: 40 66

## sub reg, xx

regx = regx + xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp | 0111  7=>regx = regx + xx | 0110 |

### Example:

sub r0, 2

binary: 0000 0000 0111 0110, 0000 0000 0000 0010

hex: 00 76, 00 02

## sub regx, [regy]

regx = regx - memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 1000  8=>regx = regx - memory\_content(regy) | 0110 |

### Example:

sub r0, [r1]

binary: 0001 0000 1000 0110

hex: 10 86

## sub reg, [xx]

regx = regx - memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp | 1001  9=>regx = regx - memory\_content(xx) | 0110 |

### Example:

sub r0, [2]

binary: 0000 0000 1001 0110, 0000 0000 0000 0010

hex: 00 96, 00 02

## sub regx, [regy + xx]

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

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 0100  10=>regx = regx - memory\_content(regy + xx) | 0110 |

### Example:

sub r0, [r1 + 5]

binary: 0001 0000 1010 0110, 0000 0000 0000 0101

hex: 10 A6, 00 05

# AND/OR/XOR GROUP (0x0007)

## and regx, regy

regx = regx AND regy

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

### Example:

and r0, r1

binary: 0100 0000 0000 0111

hex: 40 07

## and reg, xx

regx = regx AND xx

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

### Example:

and r0, 2

binary: 0000 0000 0001 0111, 0000 0000 0000 0010

hex: 00 17, 00 02

## and regx, [regy]

regx = regx AND memory\_content(regy)

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

### Example:

and r0, [r1]

binary: 0001 0000 0010 0111

hex: 10 27

## and reg, [xx]

regx = regx AND memory\_content(xx)

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

### Example:

and r0, [2]

binary: 0000 0000 0011 0111, 0000 0000 0000 0010

hex: 00 37, 00 02

## and regx, [regy + xx]

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

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

### Example:

and r0, [r1 + 5]

binary: 0001 0000 0100 0111, 0000 0000 0000 0101

hex: 10 47, 00 02

## or regx, regy

regx = regx OR regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 0101  5=>regx = regx OR regy | 0111 |

### Example:

or r0, r1

binary: 0100 0000 0101 0111

hex: 40 57

## or reg, xx

regx = regx OR xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp | 0110  6=>regx = regx OR xx | 0111 |

### Example:

or r0, 2

binary: 0000 0000 0110 0111, 0000 0000 0000 0010

hex: 00 67, 00 02

## or regx, [regy]

regx = regx OR memory\_content(regy)

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

### Example:

or r0, [r1]

binary: 0001 0000 0111 0111

hex: 10 77

## or reg, [xx]

regx = regx OR memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp | 1000  8=>regx = regx OR memory\_content(xx) | 0111 |

### Example:

or r0, [2]

binary: 0000 0000 1000 0111, 0000 0000 0000 0010

hex: 00 87, 00 02

## or regx, [regy + xx]

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

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 1001  9=>regx = regx OR memory\_content(regy + xx) | 0111 |

### Example:

or r0, [r1 + 5]

binary: 0001 0000 1001 0111, 0000 0000 0000 0101

hex: 10 97, 00 05

## xor regx, regy

regx = regx XOR regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 1010  10=>regx = regx XOR regy | 0111 |

### Example:

xor r0, r1

binary: 0100 0000 1010 0111

hex: 40 A7

## xor reg, xx

regx = regx XOR xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp | 1011  11=>regx = regx XOR xx | 0111 |

### Example:

xor r0, 2

binary: 0000 0000 1011 0111, 0000 0000 0000 0010

hex: 00 B7, 00 02

## xor regx, [regy]

regx = regx XOR memory\_content(regy)

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

### Example:

xor r0, [r1]

binary: 0001 0000 1100 0111

hex: 10 C7

## xor reg, [xx]

regx = regx XOR memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp | 1101  13=>regx = regx XOR memory\_content(xx) | 0111 |

### Example:

xor r0, [2]

binary: 0000 0000 1101 0111, 0000 0000 0000 0010

hex: 00 D7, 00 02

## xor regx, [regy + xx]

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

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 1110  14=>regx = regx XOR memory\_content(regy + xx) | 0111 |

### Example:

xor r0, [r1 + 5]

binary: 0001 0000 1110 0111, 0000 0000 0000 0101

hex: 10 E7, 00 05

# CMP GROUP (0x0008)

## 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 | bbbb  0-7: r0-r7  8: sp | 0000  0=>cmp regx, regy | 1000 |

### Example:

cmp r0, r1

binary: 0001 0000 0000 1000

hex: 10 08

## 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 | 0001  1=>cmp reg, xx | 1000 |

### Example:

cmp r0, 2

binary: 0000 0000 0001 1000, 0000 0000 0000 0010

hex: 00 18, 00 02

## cmp regx, [regy]

Compare regx and memory\_content(regy).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 0010  2=> compare regx and memory\_content(regy) | 1000 |

### Example:

cmp r0, [r1]

binary: 0001 0000 0010 1000

hex: 10 28

## cmp reg, [xx]

Compare regx and memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp | 0011  3=> compare regx and memory\_content(xx) | 1000 |

### Example:

cmp r0, [2]

binary: 0000 0000 0011 1000, 0000 0000 0000 0010

hex: 00 38, 00 02

## cmp regx, [regy + xx]

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

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 0100  4=> Compare regx and memory\_content(regy + xx) | 1000 |

### Example:

cmp r0, [r1 + 5]

binary: 0001 0000 0100 1000, 0000 0000 0000 0101

hex: 10 48, 00 05

# INC/DEC/NEG GROUP (0x0009)

## inc reg

Increments the given register.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp | 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 | 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 | 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

## dec reg

Decrements the given register.

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp | 0100  4=>dec reg | 1001 |

### Example:

dec r0

binary: 0000 0000 0100 1001

hex: 00 49

## dec [reg]

Decrements the memory\_content(reg).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp | 0101  5=> dec memory\_content(reg) | 1001 |

### Example:

dec [r1]

binary: 0000 0001 0101 1001

hex: 01 59

## dec [xx]

Decrements the memory\_content(xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | xxxx | 0110  6=> dec memory\_content(xx) | 1001 |

### Example:

dec [2]

binary: 0000 0000 0110 1001, 0000 0000 0000 0010

hex: 00 69, 00 02

## dec [reg + xx]

Decrements the memory\_content(reg + xx).

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp | 0111  7=> dec memory\_content(reg + xx) | 1001 |

### Example:

dec [r1 + 5]

binary: 0000 0001 0111 1001, 0000 0000 0000 0101

hex: 01 79, 00 05

## neg reg

Negates the given register.

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

### Example:

neg r0

binary: 0000 0000 1000 1001

hex: 00 89

## neg [reg]

Negates the memory\_content(reg).

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

### Example:

neg [r1]

binary: 0000 0001 1001 1001

hex: 01 99

## neg [xx]

Negates the memory\_content(xx).

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

### Example:

neg [2]

binary: 0000 0000 1010 1001, 0000 0000 0000 0010

hex: 00 A9, 00 02

## neg [reg + xx]

Negates the memory\_content(reg + xx).

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

### Example:

dec [r1 + 5]

binary: 0000 0001 1011 1001, 0000 0000 0000 0101

hex: 01 B9, 00 05

# MUL/DIV GROUP (0x000A)

## mul regx, regy

regx = regx \* regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | xbbb  0-7: r0-r7 | 0000  0=>regx = lower\_16\_bits(regx \* regy)  h = upper\_16\_bits(regx \* regy) | 1010 |

### Example:

mul r0, r1

binary: 0001 0000 0000 1010

hex: 10 0A

## 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 1010, 0000 0000 0000 0101

hex: 01 1a, 00 05

## mul regx, [regy]

regx = regx \* memory\_content(regy)

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

### Example:

mul r0, [r1]

binary: 0001 0000 0010 1010

hex: 10 2A

## mul reg, [xx]

regx = regx \* memory\_content(xx)

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

### Example:

mul r0, [2]

binary: 0000 0000 0011 1010, 0000 0000 0000 0010

hex: 00 3A, 00 02

## mul regx, [regy + xx]

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

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

### Example:

mul r0, [r1 + 5]

binary: 0001 0000 0100 1010, 0000 0000 0000 0101

hex: 10 4A, 00 05

## div regx, regy

regx = regx / regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | xbbb  0-7: r0-r7 | 0110  6=>regx = regx / regy  h = regx % regy | 1010 |

### Example:

div r0, r1

binary: 0001 0000 0110 1010

hex: 10 6A

## div reg, XX

reg = reg / XX

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp  9: h | 0111  7 =>reg = reg / number  h = reg % number | 1010 |

### Example:

div r1, 5

binary: 0000 0001 0001 1010, 0000 0000 0000 0101

hex: 01 7A, 00 05

## div regx, [regy]

regx = regx / memory\_content(regy)

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

### Example:

div r0, [r1]

binary: 0001 0000 1000 1010

hex: 10 8A

## div reg, [xx]

regx = regx / memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp | 1001  9=>regx = regx / memory\_content(xx) | 1010 |

### Example:

div r0, [2]

binary: 0000 0000 1001 1010, 0000 0000 0000 0010

hex: 00 9A, 00 02

## div regx, [regy + xx]

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

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

### Example:

div r0, [r1 + 5]

binary: 0001 0000 1010 1010, 0000 0000 0000 0101

hex: 10 AA, 00 05

# PUSH/POP GROUP (0x0b)

## push reg

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8-sp | 0000  000N  N==0=>push register | 1011 |

### Example:

push r0

binary: 0000 0000 0000 1011

hex: 00 0B

## push XX

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | xxxx | 0001  000N  N==1 =>push number | 1011 |

### Example:

push 0x0f

binary: 0000 0000 0001 1011, 0000 0000 0000 1111

hex: 00 1B, 00 0f

## pop reg

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8-sp | 0010  001N  N==0=>pop register | 1011 |

### Example:

pop r0

binary: 0000 0000 0010 1011

hex: 00 2B

# SHIFT GROUP (0x000c)

## shl regx, regy

regx = regx << regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 0000  0=>regx = regx << regy | 1100 |

### Example:

shl r0, r1

binary: 0100 0000 0000 1100

hex: 40 0C

## shl reg, xx

regx = regx << xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp | 0001  1=>regx = regx << xx | 1100 |

### Example:

shl r0, 2

binary: 0000 0000 0001 1100, 0000 0000 0000 0010

hex: 00 1C, 00 02

## shl regx, [regy]

regx = regx << memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 0010  2=>regx = regx << memory\_content(regy) | 1100 |

### Example:

shl r0, [r1]

binary: 0001 0000 0010 1100

hex: 10 2C

## shl reg, [xx]

regx = regx << memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp | 0011  3=>regx = regx << memory\_content(xx) | 1100 |

### Example:

shl r0, [2]

binary: 0000 0000 0011 1100, 0000 0000 0000 0010

hex: 00 3C, 00 02

## shl regx, [regy + xx]

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

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 0100  4=>regx = regx << memory\_content(regy + xx) | 1100 |

### Example:

shl r0, [r1 + 5]

binary: 0001 0000 0100 1100, 0000 0000 0000 0101

hex: 10 4C, 00 05

## shr regx, regy

regx = regx >> regy

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 0110  6=>regx = regx >> regy | 1100 |

### Example:

shr r0, r1

binary: 0100 0000 0110 1100

hex: 40 6C

## shr reg, xx

regx = regx + xx

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxx | bbbb  0-7: r0-r7  8: sp | 0111  7=>regx = regx >> xx | 1100 |

### Example:

shr r0, 2

binary: 0000 0000 0111 1100, 0000 0000 0000 0010

hex: 00 7C, 00 02

## shr regx, [regy]

regx = regx >> memory\_content(regy)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 1000  8=>regx = regx >> memory\_content(regy) | 1100 |

### Example:

shr r0, [r1]

binary: 0001 0000 0010 1100

hex: 10 8C

## shr reg, [xx]

regx = regx << memory\_content(xx)

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| xxxxx | bbbb  0-7: r0-r7  8: sp | 1001  9=>regx = regx >> memory\_content(xx) | 1100 |

### Example:

shr r0, [2]

binary: 0000 0000 1001 1100, 0000 0000 0000 0010

hex: 00 9C, 00 02

## shr regx, [regy + xx]

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

|  |  |  |  |
| --- | --- | --- | --- |
| from | to | what | group |
| bbbb  0-7: r0-r7  8: sp | bbbb  0-7: r0-r7  8: sp | 1010  4=>regx = regx >> memory\_content(regy + xx) | 1100 |

### Example:

shr r0, [r1 + 5]

binary: 0001 0000 1010 1100, 0000 0000 0000 0101

hex: 10 AC, 00 05