# TITLE

Contents

[TITLE 1](#_Toc170242819)

[OVERVIEW 3](#_Toc170242820)

[INSTRUCTIONS 4](#_Toc170242821)

[MOV destination source Moves the value from the source into the destination 4](#_Toc170242822)

[CMP source1 source2 Compares source1 and source2 and sets the flags 4](#_Toc170242823)

[INSTRUCTION LAYOUT 8](#_Toc170242824)

[ARGUMENT MODES 9](#_Toc170242825)

[INSTRUCTION SET 9](#_Toc170242826)

[INTERRUPTS 9](#_Toc170242827)

[SOFTWARE INTERRUPTS 9](#_Toc170242828)

[CACHES 10](#_Toc170242829)

[File Data Cache 10](#_Toc170242830)

[PORTS 10](#_Toc170242831)

[PS/2 PORTS 10](#_Toc170242832)

[Screen (more in [Screen](#screen)) 10](#_Toc170242833)

[Floppy Disk Ports 10](#_Toc170242834)

[Screen 10](#_Toc170242835)

[Writing to the screen 11](#_Toc170242836)

[Video Layout 11](#_Toc170242837)

[Flags 11](#_Toc170242838)

[Calling convention 11](#_Toc170242839)

[Caller 12](#_Toc170242840)

[Callee 12](#_Toc170242841)

[MEMORY LAYOUT 13](#_Toc170242842)

[VARIABLE RAM in 0xB90000 - 0xC6FFFF 13](#_Toc170242843)

[VRAM in 0xA3FFFF - 0xAFB7FF 14](#_Toc170242844)

[REGISTERS 14](#_Toc170242845)

## OVERVIEW

* Name: BGC-16
* Data bus: 16-bits.
* Address bus: 24-bits.
* Max clock speed 10 MHz

The BCG-16 CPU is a 16-bit processor designed to reflect the technology and performance characteristics of the early DOS era (versions 1.x, 2.x, and potentially 3.x). It aims to compete with the Intel 8088/8086 and possibly the 80286 in terms of raw performance, with a 24-bit address bus and a 16 bit data bus meaning that the CPU can address up to 16 megabytes.

## INTERRUPTS

Interrupts can be triggered by software or hardware specified by the interrupt location port.

### SOFTWARE INTERRUPTS

A software interrupt can be generated using the BRK Instruction and will make a long jump to 0x010000 in bank 0xF, before jumping the PC, MB and the F(flags register) will be push onto the stack and the PUSHR Instruction will also be called.

## CACHES

### File Data Cache

- size:       0xFFFF

- location:   0xFF0000 - 0xFFFFFF

## PORTS

The Ports can be communicated to and from using the OUT and In instructions

Ports are memory-mapped IO.

### PS/2 PORTS

The PS/2 Mouse is at `0x01`

The PS/2 Keyboard is at `0x02`

### Screen (more in [Screen](#screen))

The Screen is at `0x03-07`

### Floppy Disk Ports

The Floppy Disk Ports are at `0x08 to 0x0A`

## Screen

The screen is a 320×200 VGA screen with a 5 bpp and 1 bit for Alpha

### Writing to the screen

to write to the screen you can use the [INT 0x10 interrupt routine](./interrupts.md#int-0x10-services) or write to the [video memory](#memory-layout) at `0xA3FFFF - 0xB7FFFF`

### Video Layout

- RRRRGGGG\_BBBBAFFF

- R = red color

- G = green color

- B = blue color

- A = Alpah

- F = Flags

### Flags

NOTE: if all the bit are 0 then it will be a pixel in bitmap mode, the jumpper can also be set and the bitmap mode will allways be active.

- bit 0:   is Foreground Layer?

- bit 1:   is Middleground Layer?

- bit 2:   is Background Layer?

- bit 3:   is char?

- bit 4:   is WIP?

- bit 5:   is WIP?

- bit 6:   is WIP?

- bit 7:   is WIP?

  - if this bit is set, the char data will contain the ASCII char to be displayed and colored using the COLOR info

## Calling convention

The calling convention for the BCG is as follows:

### Caller

1. Push Argments

   1. Push all the arguments from left to rigth onto the stack

2. Call the function using [CALL](./instrs.md#special-instructions)

3. Retrieve Return Value

   1. The return value will be in either the R1 or HL register:

      1. Use the [HL](#registers) register if it's a pointer/address.

      2. Use the [R1](#registers) register if it's an immediate value.

### Callee

This is the sequence within the called function:

1. Save the Base Pointer

   1. Push the old [BP](#registers) onto the stack

2. Set the Base Pointer

   1. Move the [BP](#registers) register to the [SP](#registers) register

3. Save All Registers

   1. Push all registers using the [PUSHR](./instrs.md#special-instructions) instruction.

   2. To access arguments, pop them off the stack or reference them using an offset.

4. Return Value

   1. Move the return value to the appropriate register:

      1. HL register if it's a pointer.

      2. R1 register if it's an immediate value.

5. Restore Registers

   1. Pop all registers using the [POPR](./instrs.md#special-instructions) instruction.

6. Restore Base Pointer

   1. Pop the BP register.

7. Return from Function

    1. Return from the function with an offset depending on the size of the stack frame using the [RET](./instrs.md#general-instructions) instruction.

## MEMORY LAYOUT

note: `all cells is in bytes`

- BANK 0x0:

|Address Range        |Name                       |Size (bytes) |Description

|---------------------|---------------------------|-------------|-

|`0x000000 - 0x9FFFFF`|GENERAL PURPOSE BANKED RAM | ~10,5 mb    |unused

|`0xA00000 - 0xA000FF`|PORTS                      | 256 bytes   |unused

|`0xA00100 - 0xA2FFFF`|STACK                      | 196 kb      |unused

|`0xA3FFFF - 0xAFB7FF`|VIDEO RAM                  | 768 kb      |unused

|`0xAFB800 - 0xB7FFFF`|RESERVED                   | 542,7 kb    |unused

|`0xB80000 - 0xB8FFFF`|BIOS RESERVED RAM          | 64 kb       |unused

|`0xB90000 - 0xC6FFFF`|VARIABLE RAM               | ~13 mb      |

|`0xC70000 - 0xC77FFF`|INTERRUPT MEMORY RESERVED  | 32 kb       |unused

|`0xC78000 - 0xFDFFFF`|RESERVED                   | 3,5 mb      |unused

|`0xFE0000 - 0xFEFFFF`|INTERRUPT TABEL            | 64 kb       |unused

|`0xFF0000 - 0xFFFFFF`|FILE DATA CACHE            | 64 kb       |unused

- BANK 0xF:

|Address Range|Name |Description

|-------------|-----|-

|`0x000000 - 0x9FFFFF`|PROGRAM ROM                | yes this will be a part of the `GENERAL PURPOSE BANKED RAM` and only in the bank

### VARIABLE RAM in 0xB90000 - 0xC6FFFF

|Address Range        |Name                       |Size (bytes) |Description

|---------------------|---------------------------|-------------|-

|`0xB90000 - 0xC5FFFF`|VARIABLE RAM               | ~13 mb      |unused

|`0xC60000 - 0xC6FFFF`|BSS                        | 64 kb       |unused

### VRAM in 0xA3FFFF - 0xAFB7FF

|Address Range        |Name                       |Size (bytes) |Description

|---------------------|---------------------------|-------------|-

|`0xA3FFFF - 0xA5F3FF`|Foreground Layer Buffer 1  | 128 kb      |unused

|`0xA5F400 - 0xA7E7FF`|Middleground Layer Buffer 1| 128 kb      |unused

|`0xA7E800 - 0xA9DBFF`|Background Layer Buffer 1  | 128 kb      |unused

|`0xA9DC00 - 0xABCFFF`|Foreground Layer Buffer 2  | 128 kb      |unused

|`0xABD000 - 0xADC3FF`|Middleground Layer Buffer 2| 128 kb      |unused

|`0xADC400 - 0xAFB7FF`|Background Layer Buffer 2  | 128 kb      |unused

## REGISTERS

- AX: (AH + AL)   16  bit general purpose register

- BX: (BH + BL)   16  bit general purpose register

- CX: (CH + CL)   16  bit general purpose register

- DX: (CH + DL)   16  bit general purpose register

- HL: (H + L)     32  bit general purpose address register

- DS:             16  bit segment register

  the Data segment register

- FDS:            16  bit segment register

  this register will initalize to be at `0xFF00`

- S:              16  bit segment register

- PC: (PCH + PCL) 32  bit progarm counter

- PMB:            4   bit program memory bank

  - this register can switch the bank the program counter is pointing to, it's initalize to be `0xF`

- FA:             32  bit float register

- FB:             32  bit float register

- SP:             16  bit Stack register

- BP:             16  bit Stack register

- IL:             8   bit register

  the IL(interrupt location register) it's a register that can be read from to get the interrupt location as in where the interrupt came from for example from the keyboard or somewhere else.

- R1:             16  bit temp register

- R2:             16  bit temp register

- R3:             16  bit temp register

- R4:             16  bit temp register

- F1:             32  bit float register

- F2:             32  bit float register

- F3:             32  bit float register

- F4:             32  bit float register

- MB:             4   bit memory bank register

- F:              16: bit (F)lags register

  - 0x0001 zero

  - 0x0002 equals

  - 0x0004 signed

  - 0x0008 carry

  - 0x0010 overflow

  - 0x0020 less

  - 0x0040

  - 0x0080

  - 0x0100 error

  - 0x0200 interrupt enable

  - 0x0400

  - 0x0800

  - 0x1000

  - 0x2000

  - 0x4000 USING 32 BIT ADDRESSING

  - 0x8000 HALT

## INSTRUCTION LAYOUT

the instructions are built up using this layout `XXXXXXXX\_AAAAAAAA\_BBBBBBBB\_MMMMEUUU`

- X = Op code

- A = argument1

- B = argument2

- M = Memory Bank

- E = Enable Memory Bank

- U = unused (for now)

## ARGUMENT MODES

### immediate byte

### immediate word

### immediate dword

### address

### register

### register address

### Stack address

### immediate tri

### long address

### float immediate

### 32-bit segment address

### 32-bit segment address immediate

### None

## INSTRUCTIONS

### MOV destination source Moves the value from the source into the destination

|  |  |
| --- | --- |
| MOV reg, imm | Moves the immediate into the register |
| MOV reg, addr | Moves the contents of the memory address into the register |
| MOV reg1, reg2 | Moves the contents of register2 into register1 |
| MOV addr, imm | Moves the immediate into memory at the memory address |
| MOV addr, reg | Moves the contents of the register into the memory address |
| MOV addr1, addr2 | Moves the contents of the memory address2 into the memory address1 |

### CMP source1 source2 Compares source1 and source2 and sets the flags

|  |  |
| --- | --- |
| CMP reg, imm | Compares the register and the immediate |
| CMP reg, addr | Compares the contents of the memory address and the register |
| CMP reg1, reg2 | Compares the contents of register1 and register2 |
| CMP addr, imm | Compares the memory at the memory address and the immediate |
| CMP addr, reg | Compares the contents of the memory address and the register |
| CMP addr1, addr2 | Compares the contents of the memory address1 and memory address2 |

### PUSH source Pushes the source onto the stack and increments the SP

|  |  |
| --- | --- |
| PUSH reg | Pushes the contents of the register onto the stack |
| PUSH imm | Pushes the immediate onto the stack |
| PUSH addr | Pushes the contents of the memory address onto the stack |

### POP register Decrements the SP and pops the current byte into the register

|  |  |
| --- | --- |
| POP reg | Popes a byte off the stack and puts it in the register |

### POP WORD register Decrements the SP and pops the current word into the address

|  |  |
| --- | --- |
| POP word reg | Popes a word off the stack and puts it in the register |

### POP DWORD register Decrements the SP and pops the current dword into the address

|  |  |
| --- | --- |
| POP dword reg | Popes a dword off the stack and puts it in the register |

### CALL address pushes the PC register and jumps to the specified address

|  |  |
| --- | --- |
| CALL addr | Jumps to a subroutine |

- `0x07`: RET       operand1                `pops the PC register and subtracts the SP by operand1`

- `0x08`: SEZ       register                `Sets a register to zero`

- `0x09`: TEST      register                `Tests the destination and sets the flag`

# Conditional jumps

- `0x10`: JMP       address                 `Jumps to the specified address`

- `0x11`: JZ        address                 `Jumps to the specified address if the zero flag is set`

- `0x12`: JNZ       address                 `Jumps to the specified address if the zero flag is cleared`

- `0x13`: JS        address                 `Jumps to the specified address if the signed flag is set`

- `0x14`: JNS       address                 `Jumps to the specified address if the signed flag is cleared`

- `0x15`: JE        address                 `Jumps to the specified address if the equal flag is set`

- `0x16`: JNE       address                 `Jumps to the specified address if the equal flag is cleared`

- `0x17`: JL        address                 `Jumps to the specified address if the less flag is set`

- `0x18`: JG        address                 `Jumps to the specified address if the less flag is cleared`

- `0x19`: JLE       address                 `Jumps to the specified address if the equal flag or the less flag is set`

- `0x1A`: JGE       address                 `Jumps to the specified address if the equal flag is set or the less flag is cleared`

- `0x1B`: JNV       address                 `Jumps to the specified address if the overflow flag is cleared`

# IO instructions

- `0x2A`: SPIIN     port source             `Transmiting data using SPI to the slave`

- `0x2B`: SPIOUT    port destination        `Receiveing data using SPI from the slave`

- `0x2C`: IN        port source

- `0x2D`: OUT       port destination

# Arithmetic and logic operations

- `0x2E`: SEF       flag                    `Sets a flag specified by the flag operand`

- `0x2F`: CLF       flag                    `Clears a flag specified by the flag operand`

- `0x30`: ADD       destination source      `Adds the values of the source and the destination and stores the value in destination.`

- `0x31`: SUB       destination source      `Subtracts the values of the source and the destination and stores the value in destination.`

- `0x32`: MUL       destination source      `Multiplies the values of the source and the destination and stores the value in destination.`

- `0x33`: DIV       destination source      `Divides the values of the source and the destination and stores the value in destination.`

- `0x34`: AND       destination source      `Performs a bitwise AND operation between the destination and source and stores the value in destination.`

- `0x35`: OR        destination source      `Performs a bitwise OR operation between the destination and source and stores the value in destination.`

- `0x36`: NOR       destination source      `Performs a bitwise NOR operation between the destination and source and stores the value in destination.`

- `0x37`: XOR       destination source      `Performs a bitwise XOR operation between the destination and source and stores the value in destination.`

- `0x38`: NOT       destination             `Performs a bitwise NOT operation on the destination and stores the value in destination.`

- `0x39`: SHL       destination operand1    `Shifts all the bits left in the destination by specified operand1. The {SHIFT\_FLAG} is the overflowing bit, and the next bit is zero.`

- `0x3A`: SHR       destination operand1    `Shifts all the bits rigth in the destination by specified by operand1. The {SHIFT\_FLAG} is the overflowing bit, and the next bit is zero.`

- `0x3B`: ROL       destination operand1    `Rotates all the bits left in the destination by specified by operand1. The {SHIFT\_FLAG} is used as the next bit, and the overflowing bit.`

- `0x3C`: ROR       destination operand1    `Rotates all the bits right in the destination by specified by operand1. The {SHIFT\_FLAG} is used as the next bit, and the overflowing bit.`

- `0x3D`: INC       destination             `Increments the value at the destination by 1.`

- `0x3E`: DEC       destination             `Decrements the value at the destination by 1.`

- `0x3F`: NEG       destination             `Sets/clears the signed bit of the destination.`

- `0x40`: AVG       destination operand1    `Calculates the average of the values in the destination and operand1 and puts the value in the destination.`

- `0x41`: EXP       destination operand1    `Raises the value at the destination to the power of the value specified by operand1.`

- `0x42`: SQRT      destination             `Calculates the square root of the destination.`

- `0x43`: RNG       destination             `Generates a random number and puts the value into the destination`

- `0x44`: SEB       source operand1         `Sets a bit in the source specified by the operand1`

- `0x45`: CLB       source operand1         `Clears a bit in the source specified by the operand1`

- `0x46`: TOB       source operand1         `Toggles a bit in the source specified by the operand1`

- `0x47`: MOD       destination source      `WIP`

# Float arithmetic operations

- `0x50`: FADD      destination source      `Adds the values of the source and the destination and stores the value in destination.`

- `0x51`: FSUB      destination source      `Subtracts the values of the source and the destination and stores the value in destination.`

- `0x52`: FMUL      destination source      `Multiplies the values of the source and the destination and stores the value in destination.`

- `0x53`: FDIV      destination source      `Divides the values of the source and the destination and stores the value in destination.`

- `0x54`: FAND      destination source      `Performs a bitwise AND operation between the destination and source and stores the value in destination.`

- `0x55`: FOR       destination source      `Performs a bitwise OR operation between the destination and source and stores the value in destination.`

- `0x56`: FNOR      destination source      `Performs a bitwise NOR operation between the destination and source and stores the value in destination.`

- `0x57`: FXOR      destination source      `Performs a bitwise XOR operation between the destination and source and stores the value in destination.`

- `0x58`: FNOT      destination             `Performs a bitwise NOT operation on the destination and stores the value in destination.`

# Memory Operations instructions

- `0x90`: MOVW      destination source      `moves a word from the specified source to the destination`

- `0x91`: MOVD      destination source      `moves an dword from the specified source to the destination`

- `0x92`: MOVT      destination source      `moves a 24 bit from the specified source to the destination`

- `0x93`: MOVS      destination address     `moves a null terminated string from the specified address to the destination`

- `0x94`: MOVF      destination immediate   `moves a float from the specified immediate to the float register(destination)`

- `0x95`: CMPSTR    address1 address2       `Compares to null terminated strings specified by the addresses and outputs the result in the {equal} flag`

# Time and Date instructions

- `0x9D`: DATE      destination             `Gets the date and puts it in the destination (more in the DATE AND TIME)`

- `0x9E`: DELAY     operand1                `Sets a delay specified by the operand1 in milliseconds`

- `0x9F`: TIME      destination             `Gets the time and puts it in the destination (more in the DATE AND TIME)`

# Converts instructions (WIP)

- `0xA0`: CTA       destination source      `Converts the source into acsii string and puts the value into the destination with an null char`

- `0xA1`: CTH       destination source      `Converts the source into 4 char HEX string and puts the value into the destination`

# Special instructions

- `0xF6`: SSF                               `this instruction will setup a stack fream in one instruction`

  - NOTE: this is what the SSF instruction does

  - `push    BP`

  - `mov     BP,     SP`

  - `pushr`

  - `0xF7`: SMBR      BANK                    `pushes the MB register and then sets the MB register to the specified BANK specified by the BANK operander`

- `0xF8`: RTI                               `returns from an intercept routine`

- `0xF9`: NOP                               `No operation`

- `0xFA`: RISERR    error\_source            `Raises the error flag and sets the A register with the value from error\_source`

- `0xFB`: PUSHR                             `Pushes (AX BX CX DX H L) on to the stack`

- `0xFC`: POPR                              `Pops (AX BX CX DX H L) off the stack`

- `0xFD`: INT       INTERRUPT\_ROUTINE       `Generates an interrupt routine (more in the INTERRUPTS)`

- `0xFE`: BRK       INDEX                   `Generates a software interrupt and the INDEX will be moved into the X register (more in the INTERRUPTS)`

- `0xFF`: HALT                              `Stops the CPU`