|  |
| --- |
| 0xVM |
| 0xVM datasheet |

|  |
| --- |
|  |

[Glossary 2](#_Toc77531285)

[Registers and Operands 2](#_Toc77531286)

[Status Register 3](#_Toc77531287)

[Instruction set 4](#_Toc77531288)

[HALT – Halt 4](#_Toc77531289)

[NOP – No Operation 5](#_Toc77531290)

[MOVR – Move to register 6](#_Toc77531291)

[MOVM – Move to memory 7](#_Toc77531292)

[MOVRR – Move register to register 8](#_Toc77531293)

[MOVRM – Move register to memory 9](#_Toc77531294)

[MOVMR – Move memory to register 10](#_Toc77531295)

[POP – Pop 11](#_Toc77531296)

[PUSH – Push 12](#_Toc77531297)

[PUSHR – Push register 13](#_Toc77531298)

[ADD – Add 14](#_Toc77531299)

[ADDR – Add register 15](#_Toc77531300)

Glossary

Registers and Operands

**PC** – Program counter

Stores the next instruction to execute.

**ACC** – Accumulator

Stores the result of the last arithmetic operation.

**SP** – Stack pointer

**FP** – Frame pointer

**SF** – Stack frame

**Rd** – Destination (and source) register

**Rr** – Source register

**K** – Constant data

**k** – Constant address

**mX** – Maximum memory address

**rX** – Highest register

**s** – Bit in status register

**sX** – Highest bit in status register

Status Register

**SR** – Status register

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |

**Z –** Zero flag

1 if last operation resulted in zero.

**O** – Overflow flag

1 if last operation resulted in an overflow.

Instruction set

HALT – Halt

**Description:**

Halts the program.

**Operation:** None

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| HALT | None | None |

**Opcode:** 1111 1111

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | **-** | **-** |

NOP – No Operation

**Description:**

Does nothing.

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| NOP | None | PC 🡨 PC + 1 |

**Opcode:** 0000 0000

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | **-** | **-** |

MOVR – Move to register

**Description:**

Moves a value into a register.

**Operation:** Rd 🡨 K

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| MOVR K, Rd |  | PC 🡨 PC + 1 |

**Opcode:** 0001 0000

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

MOVM – Move to memory

**Description:**

Moves a value into memory.

**Operation:** k 🡨 K

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| MOVM K, k |  | PC 🡨 PC + 1 |

**Opcode:** 0001 0001

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

MOVRR – Move register to register

**Description:**

Moves a register into another register.

**Operation:** Rd 🡨 Rr

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| MOVM Rr, Rd |  | PC 🡨 PC + 1 |

**Opcode:** 0001 0010

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

MOVRM – Move register to memory

**Description:**

Moves a register into memory.

**Operation:** k 🡨 Rr

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| MOVM Rr, k |  | PC 🡨 PC + 1 |

**Opcode:** 0001 0011

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

MOVMR – Move memory to register

**Description:**

Moves a value from memory into a register.

**Operation:** Rd 🡨 k

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| MOVM k, Rd |  | PC 🡨 PC + 1 |

**Opcode:** 0001 0100

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

POP – Pop

**Description:**

Pops a value from the stack into a register and increments SP.

**Operation:** Rd 🡨 k, SP 🡨 SP + 4

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| POP Rd |  | PC 🡨 PC + 1 |

**Opcode:** 0000 0101

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

PUSH – Push

**Description:**

Pushes a value onto the stack and decrements SP.

**Operation:** k 🡨 K, SP 🡨 SP - 4

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| PUSH K |  | PC 🡨 PC + 1 |

**Opcode:** 0001 0101

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

PUSHR – Push register

**Description:**

Pushes a register onto the stack and decrements SP.

**Operation:** k 🡨 Rr, SP 🡨 SP - 4

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| POP Rr |  | PC 🡨 PC + 1 |

**Opcode:** 0001 0110

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

ADD – Add

**Description:**

Adds a value and a register and stores it in ACC.

**Operation:** ACC 🡨 Rr + K

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| ADD K, Rr |  | PC 🡨 PC + 1 |

**Opcode:** 0010 0000

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | x | x |

**Z** – 1 if the operation results in 0

**O** – 1 if the operation overflows

ADDR – Add register

**Description:**

Adds two registers together and stores the result in ACC.

**Operation:** ACC 🡨 Rr1 + Rr2

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| ADD Rr1, Rr2 |  | PC 🡨 PC + 1 |

**Opcode:** 0010 0001

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | x | x |

**Z** – 1 if the operation results in 0

**O** – 1 if the operation overflows

BRBS – Branch if bit is set

**Description:**

Conditional branch. Checks if a bit in the SR is set and branches to the absolute address if true.

**Operation:** If SR(s) = 1 then PC 🡨 k else PC 🡨 PC + 1

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| BRBS s, k |  | PC 🡨 k  PC 🡨 PC + 1 |

**Opcode:** 0011 0000

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

BRBC – Branch if bit is clear

**Description:**

Conditional branch. Checks if a bit in the SR is clear and branches to the absolute address if true.

**Operation:** If SR(s) = 0 then PC 🡨 k else PC 🡨 PC + 1

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| BRBC s, k |  | PC 🡨 k  PC 🡨 PC + 1 |

**Opcode:** 0011 0001

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

BREQ – Branch if equal

**Description:**

Conditional branch. Checks if the ACC is equal to a value and branches to the absolute if true.

**Operation:** If ACC = K then PC 🡨 k else PC 🡨 PC + 1

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| BRBC K, k |  | PC 🡨 k  PC 🡨 PC + 1 |

**Opcode:** 0011 0001

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

BRNQ – Branch if not equal

**Description:**

Conditional branch. Checks if the ACC isn’t equal to a value and branches to the absolute if true.

**Operation:** If ACC ≠ K then PC 🡨 k else PC 🡨 PC + 1

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| BRBC K, k |  | PC 🡨 k  PC 🡨 PC + 1 |

**Opcode:** 0011 0001

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

CALL – Call subroutine

**Description:**

Calls a subroutine at a specific memory address and creates a stack frame.

**Operation:** SF, PC 🡨 k

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| CALL k |  | PC 🡨 k |

**Opcode:** 0000 0010

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

CALLR – Call subroutine from register

**Description:**

Calls a subroutine at a specific memory address from a register and creates a stack frame.

**Operation:** SF, PC 🡨 k

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| CALLR k |  | PC 🡨 k |

**Opcode:** 0000 0011

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |

RET – Return from subroutine

**Description:**

Recovers the stack frame and returns from the subroutine.

**Operation:** Rd 🡨 SF, PC 🡨 SF

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Operands** | **Program counter** |
| RET | None | PC 🡨 SF |

**Opcode:** 0000 0100

**Status Register:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  | **O** | **Z** |
|  |  |  |  |  |  | - | - |