# ARCHITETURE

## Registers

### General Purpose Registers

|  |  |  |  |
| --- | --- | --- | --- |
| **32-bit** | **16-bit** | **8-bit** | **8-bit** |
| EAX - 0 | AX - 0 | AH - 0 | AL - 4 |
| EDX - 1 | DX - 1 | DH - 1 | DL - 5 |
| ECX - 2 | CX - 2 | CH - 2 | CL - 6 |
| EBX - 3 | BX - 3 | BH - 3 | BL - 7 |
| EFP - 4 | FP - 4 | | |
| ESP - 5 | SP - 5 | | |
| ESS - 6 | SS - 6 | | |
| ESD - 7 | SD - 7 | | |

The general purpose registers are dedicated to any software operation and interaction with system operations.

* EAX => Extended Acumulator Register, mainly used to hold result or status of operations, return values and accumulative operations;
* EDX => Extended Data Register, mainly used as data holder for input or output operations, arithmetic operations and comparisions;
* ECX => Extended Counter Register, mainly used as counter for loop operations, or extra index;
* EBX => Extended Base Register, mainly used as base address for accesses to memory, or extra index;
* EFP => Extended Frame Pointer, holds the pointer to current frame, wich can be used for enclosed scopes accesses like in function calling;
* ESP => Extended Stack Pointer, holds the pointer to bottom of stack, it may be 4 bytes aligned;
* ESS => Extended Stream Source, holds a pointer for a stream data, wich may be source of operation;
* ESD => Extended Stream Destination, holds a pointer for a stream data, wich may be source of operation;
* AX => EAX least significant word;
* DX => EDX least significant word;
* CX => ECX least significant word;
* BX => EBX least significant word;
* FP => EFP least significant word;
* SP => ESP least significant word;
* SS => ESS least significant word;
* SD => ESD least significant word;
* AL/AH => AX respectively least and highest significant byte;
* DL/DH => DX respectively least and highest significant byte;
* CL/CH => CX respectively least and highest significant byte;
* BL/BH => BX respectively least and highest significant byte;

### Hardware Register

|  |
| --- |
| **32-bit** |
| EPC |
| FDT |

Both registers are hidden from software access, they are system instruction operation execution controllers.

* EPC => Extended Program Counter; holds the pointer to actual instruction, can be manipulated with jumps, branchs, control operations and exchange instructions;
* FDT => Fetched Data, holds the last fetched data from the memory, cannot be accessed by the software. It holds instructions and immediates data.

### Control Register

|  |
| --- |
| **32-bit** |
| PDTR |
| IDTR |
| TDTR |
| STT |

Both control registers holds direct memory address (without pagining translation).

* PDTR => Pagining Descriptor Table Register; holds address for global vector for directories of pages;
* IDTR => Instruction Descriptor Table Register; holds address for global vector of interruption addresses.
* TDTR => Task Descriptor Table Register; holds address for global vector of tasks descriptors.
* STT => Seletor Task Table; holds a seletor for Task Descriptor Table, used in swap of task procedure.

## Flags

|  |
| --- |
| **Software Flags** |
| CF - Carry Flag |
| BF - Borrow Flag |
| VF - Overflow Flag |
| NF - Negative Flag |
| ZF - Zero Flag |
| OF - Odd Flag |
| DF - Direction Stream Flag |

|  |
| --- |
| **System Flags** |
| RN - Return from Nested call |
| EI - Enable External Interrupt |
| PL - Privilege Level |
| PG - Enable Pagining |
| IR - Instruction Repeating |

## Interruption

If a interrupt is requested, the PL flag is set to zero and the PG flag is disabled, allowing the access to real address.

### Interruption types

* Abort - A: Does not return to old procedure in IRET instruction, no any register or status flag is pushed to stack in his invoke. Occurs when a program try to do some dangerous action without privilege level;
* Trap - T: Returns normally to old procedure in IRET instruction, but, resume from the next instruction in program, EPC is pushed to stack, including flags;
* Fault - F: Returns normally to old procedure, but, tries to re-execute the last instruction wich invoked it, EPC is pushed to stack, including flags;

### Interuption vector

|  |  |
| --- | --- |
| 00.T | <Reserved by System> |
| 01.A | PL restrictions tresspassing |
| 02.A | Invalid Opcode |
| 03.T | <Reserved by System> |
| 04.F | Directory not available |
| 05.F | Page not available |
| 06.T | Page virtual function |
| 07.T | Division by zero |
| 08.T | Module by zero |
| 09.T | FP Division by zero |
| 10.T | FP Module by zero |
| 11.A | Denied memory data access |
| 12.A | Denied memory code access |
| 13.T | Overflow (asked by program) |
| 14.T | Not defined interruption |
| 15.T | Debugger interruption |
| 16-127.T | Application interruptions |
| 128-255.E | Hardware interruptions |

# MEMORY MANAGEMENT UNIT

## Pagining

Pagining envolves turning the Logical Address into the Physical Address.

|  |  |  |
| --- | --- | --- |
| 0 :: 11 | 12 :: 21 | 22 :: 31 |
| offset | page selector | table selector |

The page translating only occurs if the flag PG is active, first, the table wich is located the pages directory is obtained by the sum between the PDTR register and the “table selector” as index. The following structure defines the table descriptor:

|  |  |  |
| --- | --- | --- |
| 0 | 1 :: 11 | 12 :: 31 |
| enabled | padded | directory address |

If the flag “enabled” is zero, Interrupt Page Fault is throwed.

The directory may be located at a 4 KB bank.

The following structure defines the page descriptor:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 2 :: 3 | 4 | 5 | 6 | 7 :: 11 | 12 :: 31 |
| enabled | padded | privilege level | read | write | execute | padded | page address |

If the flag “enabled” is zero, Interrupt Directory is throwed.

The “privilege level” defines the maximum privilege level for program acess to the page, any temptative to access at a higher PL may throw a restriction PL Interruption.

The read, write and execute flags define the allowable mode for memory access at required page. Example, if the current access is for executing instruction, if flag “execute” is deactivated, the system may throw a Denied Memory Code Access Interruption.

If the flag “virtualized” is active, the

The offset in logical address is joined with the address of page as the less significants bits in physical address translating.

# SOFTWARE PROGRAMMING

# INSTRUCTION SET

## Instruction Formats

### DRO - Direct Register Operation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 :: 15 | 16 :: 18 | 19 :: 21 | 22 :: 23 | 24 :: 31 |
| iext | iregd | irsd | iopr | iopc |

* iopr = 0 (8-bit Registers)
* Operands: irsd:reg8, iregd:reg8;
* Affects: $data\_src:8, $data\_dest:8, $func\_set:8;
* iopr = 1 (16-bit Registers)
* Operands: irsd:reg16, iregd:reg16;
* Affects: $data\_src:16, $data\_dest:16, $func\_set:16;
* iopr = 2 (32-bit Registers)
* Operands: irsd:reg32, iregd:reg32;
* Affects: $data\_src:32, $data\_dest:32, $func\_set:32;
* iopr = 3 (Register and Immediate operands)
* irsd = 0 (16-bit Immediate and 8-bit Register)
* Operands: iregd:reg8;
* Affects: $data\_src:16, $data\_dest:8, $func\_set:8;
* irsd = 1 (16-bit Immediate and 16-bit Register)
* Operands: iregd:reg16;
* Affects: $data\_src:16, $data\_dest:16, $func\_set:16;
* irsd = 2 (16-bit Immediate and 32-bit Register)
* Operands: iregd:reg32;
* Affects: $data\_src:16, $data\_dest:32, $func\_set:32;
* irsd = 3 (32-bit Immediate and 8-bit Register)
* Operands: iregd:reg8;
* Affects: $data\_src:32, $data\_dest:8, $func\_set:8;
* irsd = 4 (32-bit Immediate and 16-bit Register)
* Operands: iregd:reg16;
* Affects: $data\_src:32, $data\_dest:16, $func\_set:16;
* irsd = 5 (32-bit Immediate and 32-bit Register)
* Operands: iregd:reg32;
* Affects: $data\_src:32, $data\_dest:32, $func\_set:32;
* $data\_extra:16;

### URO - Unique Register Operation

|  |  |  |  |
| --- | --- | --- | --- |
| 0 :: 17 | 18 :: 20 | 21 :: 23 | 24 :: 31 |
| iext | iregd | iregs | iopc |

* Operands: iregs:reg32, iregd:reg32;
* Affects: $data\_src:32, $data\_dest:32;
* $data\_extra:16;

### IDO - Immediate Direct Operation

|  |  |
| --- | --- |
| 0 :: 23 | 24 :: 31 |
| iext | iopc |

* $data\_extra:24;

### IFO - Immediate Functional Operation

|  |  |  |
| --- | --- | --- |
| 0 :: 19 | 20 :: 23 | 24 :: 31 |
| iext | ifunc | iopc |

* $data\_extra:20;

### DFR - Different Registers types Operation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 :: 15 | 16 :: 18 | 19 :: 21 | 22 :: 23 | 24 :: 31 |
| iext | iregd | iregs | itm | iopc |

* itm = 0 (Signal-extended 8-bit Register to 32-bit Register)
* Operands: iregs:reg32, iregd:reg32;
* Affects: $data\_src:s8, $data\_dest:32;
* itm = 1 (Signal-extended 16-bit Register to 32-bit Register)
* Operands: iregs:reg32, iregd:reg32;
* Affects: $data\_src:s16, $data\_dest:32;
* itm = 2 (Signal-extended 8-bit Register to 16-bit Register)
* Operands: iregs:reg32, iregd:reg32;
* Affects: $data\_src:s8, $data\_dest:16;
* itm = 3 (Signal-extended 16-bit Immediate to 32-bit Register)
* Operands: iregd:reg32;
* Affects: $data\_dest:32;
* $data\_extra:16;

### FRO - Functional Register Operation

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 :: 12 | 13 :: 15 | 16 :: 18 | 19 :: 23 | 24 :: 31 |
| iext | iregs | iregd | ifunc | iopc |

* Operands: iregs:reg32, iregd:reg32;
* Affects: $data\_src:32, $data\_src:32;
* $data\_extra:13;

### FUR - Functional Unique Register

|  |  |  |  |
| --- | --- | --- | --- |
| 0 :: 15 | 16 :: 18 | 19 :: 23 | 24 :: 31 |
| iext | irego | ifunc | iopc |

* Operands: irego:reg32;
* Affects: $data\_opr:32;
* $data\_extra:16;

### MIO - Memory Indexed Operation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 0 :: 10 | 11 :: 12 | 13 :: 15 | 16 :: 18 | 19 :: 21 | 22 :: 23 | 24 :: 31 |
| iext | isc | iregi | irego | iregb | iadm | iopc |

* iadm = 0 (Memory Direct Mode)
* Operands: irego:reg32;
* Affects: $data\_opr:32, $data\_mem:32;
* iadm = 1 (Memory Indirect Mode)
* Operands: irego:reg32, iregb:reg32;
* Affects: $data\_opr:32, $data\_mem:32;
* iadm = 2 (Memory Indirect Indexed Direct Mode)
* Operands: irego:reg32, iregb:reg32;
* Affects: $data\_opr:32, $data\_mem:32;
* iadm = 3 (Memory Indirect Indexed Indirect Mode)
* Operands: irego:reg32, iregb:reg32, iregi:reg32;
* Affects: $data\_opr:32, $data\_mem:32;
* $data\_extra:11;

## Conditionals List

|  |  |
| --- | --- |
| EQ:00 | opr1 == opr2 |
| NE:01 | opr1 != opr2 |
| LT:02 | u32(opr1) < u32(opr2) |
| GT:03 | u32(opr1) > u32(opr2) |
| LE:04 | u32(opr1) <= u32(opr2) |
| GE:05 | u32(opr1) >= u32(opr2) |
| BL:06 | opr1 < opr2 |
| AB:07 | opr1 > opr2 |
| BE:08 | opr1 <= opr2 |
| AE:09 | opr1 >= opr2 |
| EZ:10 | opr1 == 0 |
| NZ:11 | opr1 != 0 |
| LTZ:12 | opr1 < 1 |
| GTZ:13 | opr1 > 0 |
| LEZ:14 | opr1 <= 0 |
| GEZ:15 | opr1 >= 0 |
| SZ:16 | ZF == 1 |
| CZ:17 | ZF == 0 |
| NEVN:18 | VF != NF |
| EQVN:19 | VF == NF |
| NEVNOSZ:20 | (VF != NF) || ZF == 1 |
| EQVNOSZ:21 | (VF == NF) || ZF == 1 |
| SB:22 | BF == 1 |
| CB:23 | BF == 0 |
| SBOSZ:24 | BF == 1 || ZF == 1 |
| CBOSZ:25 | BF == 0 || ZF == 1 |
| SV:26 | VF == 1 |
| CV:27 | VF == 0 |
| SC:28 | CF == 1 |
| CC:29 | CF == 0 |
| SO:30 | OF == 1 |
| CO:31 | OF == 0 |

iedi => instruction extra dword immediate

## Instruction Set

### ADC - Adds with Carry

* DRO
* 06
* **adc** r8:iregd, r8:irsd
* **adc** r16:iregd, r16:irsd
* **adc** r32:iregd, r32:irsd
* **adc** reg:iregd, imm16:iext
* **adc** reg:iregd, imm32:iedi **#**
* IMO
* 07
* **adc** r32:irego, [imm32:iedi] **#**
* **adc** r32:irego, [r32:iregb]
* **adc** r32:irego, [r32:iregb, imm32:iedi#isc] **#**
* **adc** r32:irego, [r32:iregb, r32:iregi#isc]

**Model:**

ADC dest, src

**Description:**

Adds src operand value with carry to destination register.

**Flags Affects:**

CF, VF, NF, ZF and OF

**Interrupts:**

- none -

### ADD - Adds

* DRO
* 08
* **add** r8:iregd, r8:irsd
* **add** r16:iregd, r16:irsd
* **add** r32:iregd, r32:irsd
* **add** reg:iregd, imm16:iext
* **add** reg:iregd, imm32:iedi **#**
* IMO
* 09
* **add** r32:irego, [imm32:iedi] **#**
* **add** r32:irego, [r32:iregb]
* **add** r32:irego, [r32:iregb, imm32:iedi#isc] **#**
* **add** r32:irego, [r32:iregb, r32:iregi#isc]

**Model:**

ADD dest, src

**Description:**

Adds src operand value to destination register.

**Flags Affects:**

CF, VF, NF, ZF and OF

**Interrupts:**

- none -

### CMP - Comparates

* DRO
* 12
* **cmp** r8:iregd, r8:irsd
* **cmp** r16:iregd, r16:irsd
* **cmp** r32:iregd, r32:irsd
* **cmp** reg:iregd, imm16:iext
* **cmp** reg:iregd, imm32:iedi **#**
* IMO
* 13
* **cmp** r32:irego, [imm32:iedi] **#**
* **cmp** r32:irego, [r32:iregb]
* **cmp** r32:irego, [r32:iregb, imm32:iedi#isc] **#**
* **cmp** r32:irego, [r32:iregb, r32:iregi#isc]

**Model:**

CMP dest, src

**Description:**

Do a comparision subtracting src operand value from destination register without saving the result, and setting the flags accordling.

**Flags Affects:**

BF, VF, NF, ZF and OF

**Interrupts:**

- none -

### JMP - Jumps

* IDO
  + 0E
    - **jmp** imm:iext<2
* FRO
  + 0F
    - 00: **jmp** imm32:iedi **#**
    - 01: **jmp** r32:iregd

**Model:**

JMP dest

**Description:**

Jumps to a absolute address in program.

**Flags Affect:**

- none -

### JMPR – Jumps relative

* IDO
  + 10
    - **jmpr** imm:iext<2
* FRO
  + 0F
    - 02: **jmpr** imm32:iedi **#**
    - 03: **jmpr** r32:iregd

**Model:**

JMPR dest

**Description:**

Jumps by a relative offset in program.

**Flags Affect:**

- none -

### JR – Jumps relative conditionally

* FUR
  + 11
    - 00: **jeq** imm:iext<2, r32:irego
    - 01: **jne** imm:iext<2, r32:irego
    - 02: **jlt** imm:iext<2, r32:irego
    - 03: **jgt** imm:iext<2, r32:irego
    - 04: **jle** imm:iext<2, r32:irego
    - 05: **jge** imm:iext<2, r32:irego
    - 06: **jbl** imm:iext<2, r32:irego
    - 07: **jab** imm:iext<2, r32:irego
    - 08: **jbe** imm:iext<2, r32:irego
    - 09: **jae** imm:iext<2, r32:irego
    - 10: **jez** imm:iext<2, r32:irego
    - 11: **jnz** imm:iext<2, r32:irego
    - 12: **jltz** imm:iext<2, r32:irego
    - 13: **jgtz** imm:iext<2, r32:irego
    - 14: **jlez** imm:iext<2, r32:irego
    - 15: **jgez** imm:iext<2, r32:irego
    - 16: **jeq** imm:iext<2
    - 17: **jne** imm:iext<2
    - 18: **jlt** imm:iext<2
    - 19: **jgt** imm:iext<2
    - 20: **jle** imm:iext<2
    - 21: **jge** imm:iext<2
    - 22: **jbl** imm:iext<2
    - 23: **jab** imm:iext<2
    - 24: **jbe** imm:iext<2
    - 25: **jae** imm:iext<2
    - 26: **jsv** imm:iext<2
    - 27: **jcv** imm:iext<2
    - 28: **jsc** imm:iext<2
    - 29: **jcc** imm:iext<2
    - 30: **jso** imm:iext<2
    - 31: **jco** imm:iext<2

**Model:**

JMPR dest, opr

**Description:**

Jumps by a relative offset in program with condition, comparates the opr with EDX, or analyzes the Software Flags.

**Flags Affect:**

- none -

### LDM - Load Value from Memory

* DRO
* 04
* **ldmd** r8:iregd, r8:irsd (, imm16:iext)
* **ldmd** r16:iregd, r16:irsd (, imm16:iext)
* **ldmd** r32:iregd, r32:irsd (, imm16:iext)
* **ldmd** reg:iregd, imm16:iext
* **ldmd** reg:iregd, imm32:iedi **#**
* FUR
* 05
* 00: **ldmw** r16:irego, EBX (, imm16:iext)
* 01: **ldmw** r16:irego, EBX, ECX
* 02: **ldmw** r16:irego, ESP (, imm16:iext)
* 03: **ldmw** r16:irego, EFP (, imm16:iext)
* 04: **ldmw** r16:irego, imm32:iedi **#**
* 05: **ldmw** r16:irego, EBX, imm32:iedi **#**
* 06: **ldmw** r16:irego, ESP, imm32:iedi **#**
* 07: **ldmw** r16:irego, EFP, imm32:iedi **#**
* 08: **ldmb** r8:irego, EBX (, imm16:iext)
* 09: **ldmb** r8:irego, EBX, ECX
* 10: **ldmb** r8:irego, ESP (, imm16:iext)
* 11: **ldmb** r8:irego, EFP (, imm16:iext)
* 12: **ldmb** r8:irego, imm32:iedi **#**
* 13: **ldmb** r8:irego, EBX, imm32:iedi **#**
* 14: **ldmb** r8:irego, ESP, imm32:iedi **#**
* 15: **ldmb** r8:irego, EFP, imm32:iedi **#**
* 16: **ldmd** r32:irego, EBX, imm32:iedi **#**
* 17: **ldmd** r32:irego, EBX, ECX

**Model:**

LDM dest, src\_adr, src\_index

**Description:**

Loads a value from memory address (with optional index), to a register operand.

**Flags Affect:**

- none -

**Interrupts:**

- none -

### MOV - Move Value

* DRO
* 00
* **mov** r8:iregd, r8:irsd
* **mov** r16:iregd, r16:irsd
* **mov** r32:iregd, r32:irsd
* **mov** reg:iregd, imm16:iext
* **mov** reg:iregd, imm32:iedi **#**
* MIO
* 01
* **mov** r32:irego, [imm32:iedi] **#**
* **mov** r32:irego, [r32:iregb]
* **mov** r32:irego, [r32:iregb, imm32:iedi#isc] **#**
* **mov** r32:irego, [r32:iregb, r32:iregi#isc]

**Model:**

MOV dest, src

**Description:**

Moves a value from a operand to a register.

**Flags Affect:**

- none -

**Interrupts:**

- none -

### SBB - Subtracts with Borrow

* DRO
* 0A
* **sbb** r8:iregd, r8:irsd
* **sbb** r16:iregd, r16:irsd
* **sbb** r32:iregd, r32:irsd
* **sbb** reg:iregd, imm16:iext
* **sbb** reg:iregd, imm32:iedi **#**
* IMO
* 0B
* **sbb** r32:irego, [imm32:iedi] **#**
* **sbb** r32:irego, [r32:iregb]
* **sbb** r32:irego, [r32:iregb, imm32:iedi#isc] **#**
* **sbb** r32:irego, [r32:iregb, r32:iregi#isc]

**Model:**

SBB dest, src

**Description:**

Subtracts src operand value with borrow from destination register.

**Flags Affects:**

BF, VF, NF, ZF and OF

**Interrupts:**

- none -

### STR - Store Value to Memory

* DRO
* 02
* **strd** r8:irsd, r8:iregd (, imm16:iext)
* **strd** r16:irsd, r16:iregd (, imm16:iext)
* **strd** r32:irsd, r32:iregd (, imm16:iext)
* **strd** imm16:iext, reg:iregd
* **strd** imm32:iedi, reg:iregd **#**
* FUR
* 03
* 00: **strd** r32:irego, imm16:iext, EBX
* 01: **strd** r32:irego, imm16:iext, ECX
* 02: **strd** r32:irego, imm32:iedi (, imm16:iext) **#**
* 03: **strd** r32:irego, imm32:iedi, EBX **#**
* 04: **strd** r32:irego, imm32:iedi, ECX **#**
* 05: **strd** r32:irego, EAX, ECX
* 06: **strd** r32:irego, EAX, imm32:iedi **#**
* 07: **strd** r32:irego, EDX, imm32:iedi **#**
* 08: **strd** EBX, r32:irego, imm32:iedi **#**
* 09: **strd** imm32:iedi, r32:irego **#**
* 10: **strw** r32:irego, imm16:iext
* 11: **strw** r32:irego, AX (, imm16:iext)
* 12: **strw** r32:irego, BX (, imm16:iext)
* 13: **strw** r32:irego, CX (, imm16:iext)
* 14: **strw** r32:irego, DX (, imm16:iext)
* 15: **strw** r32:irego, imm16:iext, EBX
* 16: **strw** r32:irego, imm16:iext, ECX
* 17: **strw** imm32:iedi, imm16:iext **#**
* 18: **strw** imm32:iedi, r32/16:irego **#**
* 19: **strw** EBX, r32/16:irego (, imm16:iext)
* 20: **strw** EBX, r32/16:irego, ECX
* 21: **strb** r32:irego, imm8:iext
* 22: **strb** r32:irego, AL (, imm16:iext)
* 23: **strb** r32:irego, AH (, imm16:iext)
* 24: **strb** r32:irego, DL (, imm16:iext)
* 25: **strb** r32:irego, DH (, imm16:iext)
* 26: **strb** r32:irego, imm8:iext, EBX
* 27: **strb** r32:irego, imm8:iext, ECX
* 28: **strb** imm32:iedi, imm8:iext **#**
* 29: **strb** imm32:iedi, AL **#**
* 30: **strb** EBX, AL (, imm16:iext)
* 31: **strb** EBX, AL, ECX

**Model:**

STR dest\_adr, src, dest\_index

**Description:**

Stores a value from operand to memory specified by the address and a optionally index.

**Flags Affects:**

- none -

**Interrupts:**

- none -

### SUB - Subtracts

* DRO
* 0C
* **sub** r8:iregd, r8:irsd
* **sub** r16:iregd, r16:irsd
* **sub** r32:iregd, r32:irsd
* **sub** reg:iregd, imm16:iext
* **sub** reg:iregd, imm32:iedi **#**
* IMO
* 0D
* **sub** r32:irego, [imm32:iedi] **#**
* **sub** r32:irego, [r32:iregb]
* **sub** r32:irego, [r32:iregb, imm32:iedi#isc] **#**
* **sub** r32:irego, [r32:iregb, r32:iregi#isc]

**Model:**

SUB dest, src

**Description:**

Subtracts src operand value from destination register.

**Flags Affects:**

BF, VF, NF, ZF and OF

**Interrupts:**

- none -

## Instruction List

**Data stream**

* **mov** => Moves a value from operand to a register.
* **str**[b, w, d] => Stores a value from operand to the memory.
* **ldm**[b, w, d] => Loads a value from memory to a register.
* **psh** => Pushes a data value to stack (4-bytes aligned).
* **pop** => Pops a data value from stack (4-bytes aligned).
* **psha** => Pushes all general registers to stack.
* **popa** => Pops all general registers from stack.
* **xcg** => Exchange a register value by other.
* **sts** => Store stream data at ess pointed to esd pointed, increments or decrements both registers.
* **stsdc** => Store stream data at ess pointed to esd pointed, increments or decrements both registers, and decrements the counter register.

**Register data processing**

* **cv**[bd, bw, wd] => Converts a operand value with signal-extention to a higher bit-size operand.
* **fcv**[fd, fw, fb, df, wf, bf] => Converts value in fp format to integer with sign-extention or vice-versa.
* **inc** => Increments the value in the register operand.
* **dec** => Decrements the value in the register operand.
* **finc** => Increments the vaue in the register operand in fp format.
* **fdec** => Decrements the value in the register operand in fp format.

**Data arithmetic (ALU)**

* **adc** => Adds a operand value to another with carry flag.
* **add** => Adds a operand value to another.
* **sbb** => Subtracts a operand value from another with borrow flag.
* **sub** => Subtracts a operand value from another.
* **mul** => Multiplies a operand value by other.
* **div** => Divides a operand value by other.
* **mod** => Operates division and retrieves remainder from operand by other.
* **and** => Operates and bitwise between a operand value and another.
* **or** => Operates or bitwise between a operand value and another.
* **xor** => Operates xor bitwise between a operand value and another.
* **not** => Operates not bitwise in a operand.
* **shl** => Operates shift left bitwise between a operand value and another.
* **shr** => Operates shift right bitwise between a operand value and another.
* **sal** => Operates shift left arithmetical between a operand value and another.
* **sar** => Operates shift right arithmetical between a operand value and another.
* **rotl** => Rotates left a operand value by another.
* **rotr** => Rotates right a operand value by another.
* **cmp** => Operates subtract between two operands and only set the flags acorddling.
* **test** => Operates and bitwise between two operands and only set the flags acorddling.
* **bts** => Operates a bit test in one operand value and set the flags acorddling.

**Data Arithmetic (FPU)**

* **fadd** => Adds a operant to another in fp format.
* **faddi** => Adds a integer operand to another in fp format.
* **fsub** => Subtracts a operand from another in fp format.
* **fsubi** => Subtracts a integer operand from another in fp format.
* **fmul** => Multiplies a operand to another in fp format.
* **fmuli** => Multiplies a integer operand to another in fp format.
* **fdiv** => Divides a operand by another in fp format.
* **fdivi** => Divides a integer operand by another in fp format.
* **fmod** => Operates a division in a operand in fp format by a operand in integer format and retrieves the remainder.
* **fsqrt** => Operates square root in a operand in fp format.
* **fcbrt** => Operates cubic root in a operand in fp format.
* **fsin** => Operates sine in a operand in fp format.
* **fcos** => Operates cosine in a operand in fp format.
* **ftan** => Operates tangent in a operand in fp format.
* **fasin** => Operates arcsine in a operand in fp format.
* **facos** => Operates arcosine in a operand in fp format.
* **fatan** => Operates arctangent in a operand in fp format.
* **flog2** => Calculates logarithm of base 2 in fp format.
* **flog10** => Calculates logarithm of base 10 in fp format.
* **flogn** => Calculates natural logarithm of operand in fp format.
* **fpow** => Calculates power of operand by another in fp format.
* **flr** => Rounds floor value of operand in fp format.
* **fhr** => Rounds ceil value of operand in fp format.
* **frnd** => Rounds value of operand in fp format.
* **fcmp** => Subtracts value of operand by another and only set flags accordling, in fp format.
* **ftest** => Operates and bitwise of operand by another and only set flags accodling, in fp format.
* **fbts** => Test bits of a operand value ans set flags accodling, in fp format.
* **fxam** => Examinates value of operand in fp format, and retrieve his status to a register.

**Program Stream**

* **jmp** => Jumps to a absolute address in program.
* **jmpr** => Jumps by a relative offset in program.
* **j**[#cond] => Jumps by a relative offset in program conditionally.
* **bra** => Pushs next instrution address to stack, and jumps to a absolute address.
* **br** => Pushs next instruction address to stack and jumps by a relative offset in program conditionally.
* **ret** => Pops address from stack and jump to it in program (may be used with "bra" or "br").
* **enter** => Enters a procedure in a absolute address linked way, pushes and updates the efp register to current stack and pushes software flags to stack.
* **leave** => Returns from a procedure, restores latest efp value from stack and restores software flags from stack.
* **rep**[#cond] => Repeats the next instruction until the condition is true.
* **xcgpc** => Exchange a register value by epc register.

**System Control**

* **ver**[r, w] => Verify a absolute address for reading and writing, return result to register a.
* **set**[cf, bf, vf, nf, zf, of, df, ei] => Enables any of software flags.
* **clr**[cf, bf, vf, nf, zf, of, df, ei] => Cleans any of software flags.
* **tog**[cf, bf, vf, nf, zf, of, df, ei] => Toggles any of software flags.
* **pshstw** => Pushes software flags to stack;
* **popstw** => Pops software flags from stack;
* **pshsts** => Pushes control and system flags to stack;
* **popsts** => Pops control and system flags from stack;
* **pshst** => Pushes all status flags to stack;
* **popst** => Pops all status flags from stack;
* **jlpl** => Enters in a low privilege level and jumps to a absolute address in program. Trying to access higher privilege level can cause a exception.
* **jepg** => Enables pagining and jumps to a absolute address in program.
* **jdpg** => Disables pagining and jumps to a absolute address in program.
* **entp** => Enters in a task procedure, specified by a value used as seletor for task vector access.
* **ldpdtr** => Loads a value to pdtr register.
* **ldidtr** => Loads a value to idtr register.
* **ldtdtr** => Loads a value to tdtr register.
* **swtp** => Swaps of task procedure, specified by a value used as seletor for task vector acess. It saves the current task descriptor context, with all pushed registers from stack and up-valued esp and efp registers (because is expected it be used in a interruption procedure).
* **int** => Call for system interruption;
* **inp** => Input value from a port.
* **out** => Output value to a port.
* **hlt** => Stops/Halts instruction execution.