**ISA Design:**

The ISA encoding supports a stack machine with a hybrid encoding strategy. It includes address modes for Immediate, Indirect, and Direct addressing.

**Hybrid Encoding Strategy**

* Arithmetic operations are 4 bytes long
* MOV is 2 bytes for register to register, 4 bytes for immediate to register
* LOAD/STORE is 4 bytes for an immediate memory address and indirect address
* NOP, HALT, CMP and Jumps are 2 bytes

**Address Modes**

* Immediate mode – instructions that use an immediate
* Indirect – LOAD/STORE using registers

**Memory Layout**

* All memory is stored in sequences of 16 bits with no defined data types
* Memory is stored in little endian
* ROM is located at the lowest part of memory
  + Only the first 16-bits are addressable as the program counter has only 16 bits compared to the 20-bit addressable memory.
* RAM is located after the ROM
* The stack starts at the highest addressable memory.

**Instruction Structure:**

Arithmetic Immediate (4 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Destination Register​** | | | | **1 Reg OP​** | | | | **16 – Bit Immediate​** | | | | | | | | | | | | | | | |
| X​ | X​ | X​ | X​ | X​ | X​ | X​ | 1​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ |

(ADD, ADDC, SUBB, AND, OR, NOR, SHL, SHR, SHAR, ROR, RORC, ROLC, XOR)

Arithmetic Register (4 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Destination Register​** | | | | **1 Reg OP​** | | | | **Reserved​** | | | | | | | | | | | | **2 Reg OP​** | | | |
| X​ | X​ | X​ | X​ | X​ | X​ | X​ | 0​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | X​ | X​ | X​ | X​ |

(ADD, ADDC, SUBB, AND, OR, NOR, SHL, SHR, SHAR, ROR, RORC, ROLC,XOR, SUB)

NEG Immediate (4 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Destination Register​** | | | | **Reserved** | | | | **16 – Bit Immediate​** | | | | | | | | | | | | | | | |
| X​ | X​ | X​ | X​ | X​ | X​ | X​ | 1​ | X​ | X​ | X​ | X​ | 0​ | 0​ | 0​ | 0​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ |

NEG Register (4 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Destination Register​** | | | | | **1 Reg OP​** | | | | | **Reserved​** | | | | | | | | | | | | | | | | |
| X​ | X​ | X​ | X​ | X​ | X​ | X​ | 0​ | X​ | X​ | X​ | X​ | X​ | | X​ | X​ | X​ | 0​ | | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0 | 0​ | 0​ | 0​ |

LOAD/STOR Immediate (4 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Destination Register​** | | | | **20 – Bit Immediate Address Being Loaded From or Stored Into​** | | | | | | | | | | | | | | | |
| X​ | X​ | X​ | X​ | X​ | X​ | X​ | 1​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ |

LOAD/STOR Register (4 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Destination Register​** | | | | **1 Reg OP​** | | | | **Reserved​** | | | | | | | | | | | | **Reserved**  **(was reg2OP)​** | | | |
| X​ | X​ | X​ | X​ | X​ | X​ | X​ | 0​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0 | 0​ | 0 | 0​ |

MOV Immediate (4 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Destination Register​** | | | | **Reserved​** | | | | **16 – Bit Immediate ​** | | | | | | | | | | | | | | | |
| 0​ | 0​ | 1​ | 0​ | 0​ | 0​ | 0​ | 1​ | X​ | X​ | X​ | X​ | 0​ | 0​ | 0​ | 0​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ |

MOV Register (2 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code** | | | | | | | **Immediate Identifier** | **Destination Register** | | | | **REG OP1** | | | |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | X | X | X | X | x | x | x | x |

PUSH Immediate (4 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Reserved​** | | | | | | | | **16 – Bit Immediate​** | | | | | | | | | | | | | | | |
| 0​ | 0​ | 1​ | 0​ | 0​ | 0​ | 1​ | 1​ | 0 | 0 | 0 | 0​ | 0​ | 0​ | 0​ | 0​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ |

PUSH Register (2 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code** | | | | | | | **Immediate Identifier** | **REG OP1** | | | | **Reserved** | | | |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | X | X | X | X | 0 | 0 | 0 | 0 |

POP (2 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code** | | | | | | | **Immediate Identifier** | **Destination Register** | | | | **Reserved** | | | |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | X | X | X | X | 0 | 0 | 0 | 0 |

LDA (4 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Reserved​** | | | | **20–Bit Immediate​** | | | | | | | | | | | | | | | | | | | |
| 0​ | 0​ | 1​ | 1​ | 1​ | 1​ | 1​ | 1​ | 0​ | 0​ | 0​ | 0​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ | X​ |

CMP (2 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code** | | | | | | | **Immediate Identifier** | **1st Reg OP** | | | | **2nd Reg OP** | | | |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | X | X | X | X | X | X | X | X |

Jump Instructions (2 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Reserved​** | | | | | | | |
| X​ | X​ | X​ | X​ | X​ | X​ | X​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ |

(JZ, JNZ, JG JGE, JL, JLE, JA​, JAE, JB, JBE)

HALT (2 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Reserved​** | | | | | | | |
| 1​ | 1​ | 1​ | 1​ | 1​ | 1​ | 1​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ |

NOP (2 bytes):

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **OP Code ​** | | | | | | | **Immediate Identifier​** | **Reserved​** | | | | | | | |
| 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ |

**Registers:**

|  |  |
| --- | --- |
| **16-Bit Registers​** | **OP Code** |
| rA​ | 0000 |
| rB​ | 0001 |
| rC​ | 0010 |
| rD​ | 0011 |
| rE​ (LDR loads into rE only) | 0100 |
| rF​ | 0101 |
| rG​ | 0110 |
| rH​ | 0111 |
| rI​ | 1000 |
| rJ​ | 1001 |
| rK​ -cmp reg | 1010 |
| rL | 1011 |
| rM | 1100 |
| PC | 1101 |
| SP | 1110 |
| FLAG | 1111 |

**Register Design:**

* rG/rH – Designated for return values from functions. Registers are used as needed depending on size of return.
* rE – Indirect Memory Address Register
  + rE stores 20 bit immediate (Now is a 32 bit register
  + LDA loads its 20-bit immediate into the rE registers.
  + All Jump commands jump based on value in rE
* Stack Pointer - represented by register pair SP
  + SP holds the 20 bit memory address of the stack pointer
  + The stack starts at the highest memory address and grows downwards
  + Decremented on Pushes, architecture handles overflow
  + Incremented on Pops, architecture handles overflow
  + The programmer is expected to handle SP overflow and underflow when manually manipulating the stack
* PC- Program Counter
  + 32 bits
  + Operates on 20-bit address space, so it only looks at first 20 bits.
* FLAG
  + Flags are affected only by arithmetic and logical instructions
  + Carry flag – Indicates arithmetic carry or borrow
  + Less Than Flag -Indicates if an arithmetic or logical operation results in zero
  + Equality Flag - Indicates that the two operands in a cmp operation are equal
  + Overflow Flag - Indicates that there has been an overflow in an arithmetic operation
  + Sign Flag - Indicates the sign of the last resulting arithmetic or logical operation

 FLAG Register:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Reserved​** | | | | | | | | | | | **Sign​** | **Overflow​** | **Equality​** | **Zero​** | **Carry​** |
| 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | 0​ | X​ | X​ | X​ | X​ | X​ |

**Instruction Op Codes:**

|  |  |
| --- | --- |
| Operation​ | OP Code​ |
| NOP​ | 0000000​ |
| ADD​ | 0000001​ |
| ADDC​ | 0000010​ |
| SUBB​ | 0000011​ |
| AND​ | 0000100​ |
| OR​ | 0000101​ |
| NOR​ | 0000110​ |
| SHL​ | 0000111​ |
| SHR​ | 0001000​ |
| SHAR​ | 0001001​ |
| ROR​ | 0001010​ |
| ROL​ | 0001011​ |
| RORC | 0001100​​ |
| ROLC | 0001101​​ |
| LOAD | 0001110​​ |
| SOTR | 0001111​​ |
| MOV | 0010000​​ |
| PUSH | 0010001 |
| POP | 0010010 |
| CMP | 0010011 |
| JZ | 0010100​​​ |
| JNZ | 0010101 |
| JG | 0010110​​​ |
| JGE | 0010111​​​ |
| JL | 0011000 |
| JLE | 0011001​ |
| JA | 0011010 |
| JAE | 0011011 |
| JB | 0011100​ |
| JBE | 0011101 |
| LDA | 0011110 |
| NEG | 0011111 |
| XOR | 0100000 |
| SUB | 0100001 |
| HALT | 1111111 |

**Special Considerations:**

Since the ISA is designed for a single 16-bit processor, it cannot address an entire mebibyte of memory using a single register. To map an entire mebibyte, that would require at least a 20-bit register. To allow our ISA to address its entire mebibyte of memory using only 16-bit registers, some special features were implemented.

* Immediates cannot have a ‘-‘ or ‘+’ sign out front. Negative values must be put in as their positive 2’s complement equivalents. All immediates must be in decimal.
* The program counter uses two 16-bit registers
  + ROM starts at address 0 and increments
  + The RAM/Heap begins where ROM ends and goes up in memory addresses as it grows
  + The stack begins at the highest memory address and goes down in memory addresses as it grows
* The stack pointer lies in two registers, SP1 and SP2
  + SP2 is the first 16 least significant bits of the stack address
  + SP1 is the 4 most significant bits of the stack address – The most significant 12 bits of SP1 are set to 1 no matter what operations are done. This ensures manipulating the stack pointer works with ADDC
  + The programmer is expected to handle SP1/SP2 overflow and underflow when manually manipulating the stack. A bit bucket ensures proper underflow will occur
* rE/rF are used for indirect memory addressing
  + rE stores the 4 most significant bits of a memory address
  + rF stores the 16 least significant bits of a memory address
  + LDA loads its 20-bit immediate into the rE/rF registers.
  + These registers can be used for general purpose, but LDA will always trash them
  + Jump commands will jump to address pointed to by rE/
* LOAD/STOR Registers take three operands
  + The first operand/destination register is the destination/source of the value
    - For LOAD, this is where the 16-bit value from memory will be placed
    - For STOR, this is the 16-bit value which will be placed in memory
  + The second operand/1st register operand is the 16 least significant bits of the memory address to load/store from or to
  + The third operand/2nd register operand is the 4 most significant bits of the memory address to load/store from or to
  + Any general-purpose r registers can be used for the 1st and 2nd register operands for LOAD/STOR
  + The most significant 12 bits of the 1st register operand will be ignored by the hardware’s logic on a LOAD or STOR instruction, only the 4 least significant bits will be used