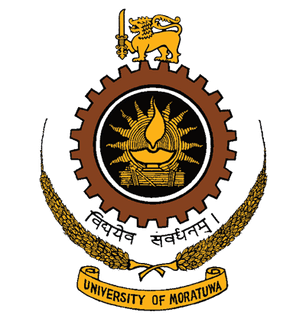
**Department of Electronic & Telecommunication Engineering**

**University of Moratuwa**



**EN3030 - Circuit and System Design**

**FPGA BASED PROCESSOR IMPLEMENTATION**

**(ISA Design)**

Group Members

De Silva K. D. M. K. 170106V

Nimashi K. H. 170407U

Nusha M. N. F. 170415R

Palihakkara A. T. 170418E

**This report is submitted in partial fulfillment of the requirements for the module EN3030 – Circcuit and System Design.**

**05 October 2020**

**INTRODUCTION**

The Instruction Set Architecture (ISA) described below is built for the task of implementing a processor that can multiply matrices with maximum efficiency. The ISA will be implemented on a multiple-cores processor for better speed and space management.

When carrying out the matrix multiplication the data will be input as a series of numbers which will be read row by row. For easy comprehension, the input will include three constants m, n, k which represent the sizes of the two matrices. The first matrix size is (m, n) and second matrix size is (n, k).

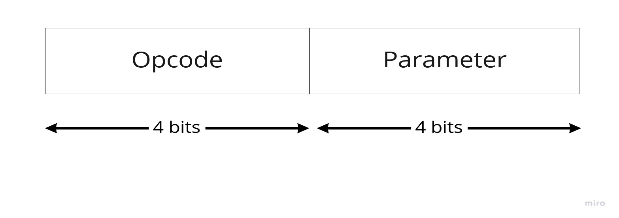
The final output will similarly be a series of numbers which can be read row-wise and the output matrix size will be (m, k).

|  |  |
| --- | --- |
| Max (m, n, k) | Maximum possible value |
| 2 | 181 |
| 3 | 147 |
| 4 | 128 |
| 5 | 114 |
| 6 | 104 |
| 7 | 96 |
| 8 | 90 |
| 9 | 85 |
| 10 | 80 |

**Instruction Set Architecture.**

The ISA includes 10 instructions, each of which are 8 bits long.

Instruction Format:



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OPCODE | BIN | | PARAMETERS | EXAMPLE | DESCRIPTION |
| NOOP | 0000 | 0000 | - | NOOP | No operation |
| END | 0001 | 0000 | - |  | Terminates program |
| LOAD | 0010 | 0000  0001 | 0 - ZERO  1 - AC |  | 0 - Loading zero value to AC  1 - Loading value from Memory to AC |
| STORE | 0011 | 0000 | - | M [ ] ← AC | Moving value in AC to memory |
| MOVE | 0100 | 0000  0001  0010  0011  0100  0101  0110  0111 | 0 - R0  1 - R1  2 - REG\_P  3 - REG\_Q  4 - REG\_M  5 - REG\_K  6 - REG\_I  7 - REG\_N | R1 ← AC | Moving value in AC to a register |
| ADD | 0101 | 0001 | - | AC ← AC + R1 | Add the value in R1 to AC and store in AC |
| MUL | 0110 | 0000 | - | AC ← AC\*R1 | Multiply the value in R1 with the value in AC and store in AC |
| DEC | 0111 | 0000  0001  0010 | 0 - REG\_P  1 - REG\_Q  2 - REG\_I | REG\_P ← REG\_P - 1 | Decrement register by 1 |
| JPN | 1000 | 0000  0001  0010 | 0 - LOOP\_M  1 - LOOP\_K  2 - LOOP\_N |  | If value of selected register is non-zero, then go to relevant memory address (loop) |
| COPY | 1001 | 0000  0001  0010 | 0 - REG\_M\_P  1 - REG\_K\_Q  2 - REG\_I\_N | REG\_M, REG\_P← M[ ] | Reads from memory and writes to two registers at once |

**General Purpose Registers**

There are six 8 - bit registers.

* REG\_P
* REG\_Q
* REG\_M
* REG\_K
* REG\_I
* REG\_N

And there are two 16 - bit registers.

* R0
* R1

**Assembly Code**

COPY REG\_M\_P  
COPY REG\_K\_Q  
COPY REG\_I\_N  
mloop:

kloop:

LOAD ZERO  
MOVE R0

nloop:

LOAD ZERO  
MOVE R1  
LOAD AC  
MUL R1  
ADD R0  
MOVE R0  
DEC REG\_I  
JPN LOOP\_N

STORE  
DEC REG\_Q  
JPN LOOP\_K

DEC REG\_P  
JPN LOOP\_M  
END