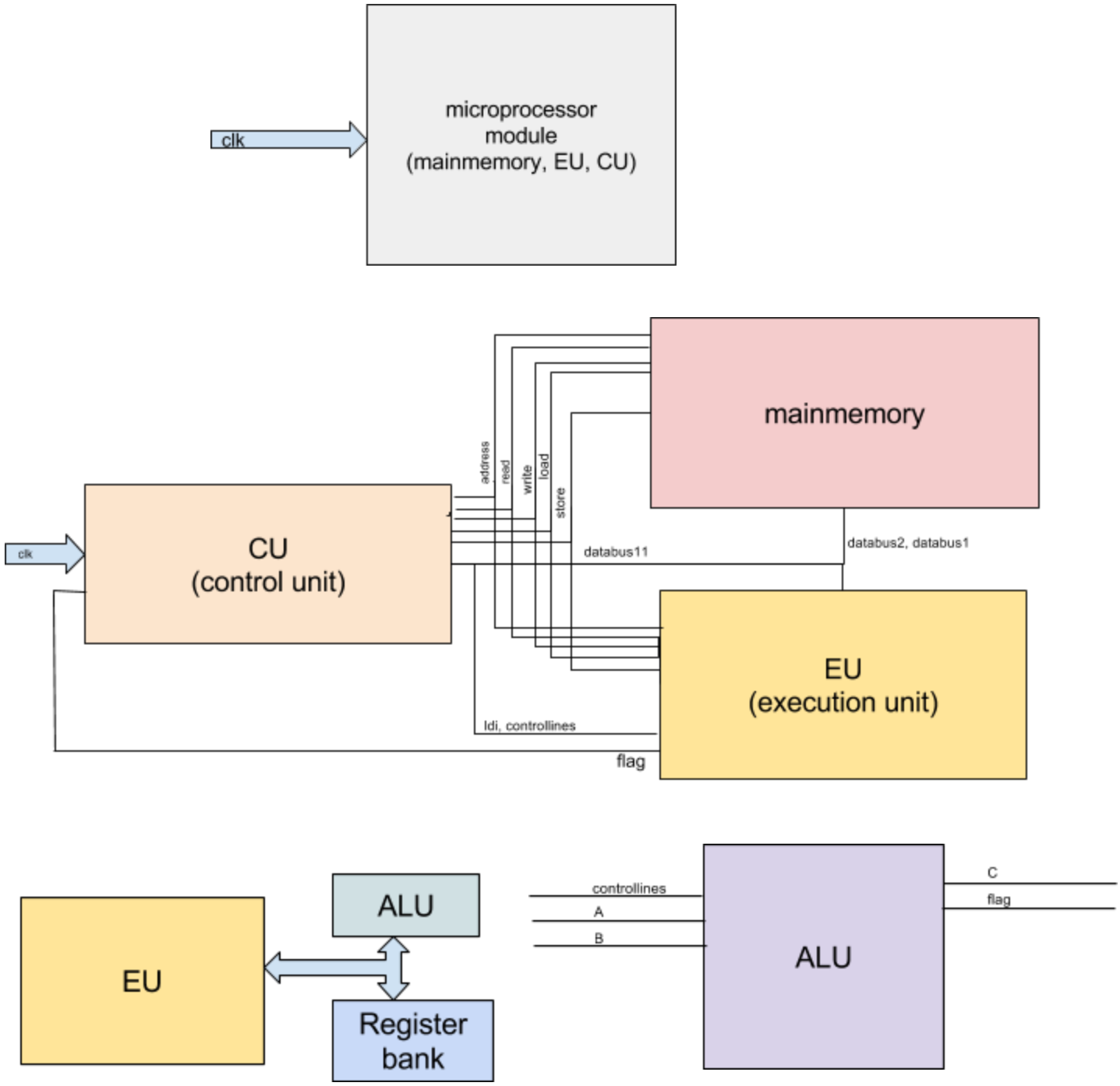
# 8 bit Microprocessor

Milind Kumar V
EE16B025

Specifications-

- Byte sized registers
- 16 byte register memory
- 7 ALU operations- add, subtract, bitwise AND, bitwise OR, left shift, right shift, compare
- Registers R0, R1, R2 mapped for ALU operations- not to be used for storage
- 32 byte main memory
- 19 bit instructions
- 32 word program memory
- Source code always obtained from “machinecode.txt”
- LOAD, STORE, MOV, LOAD DIRECT features available
- 8 bit flag register
- Jump variants available - unconditional, if carry flag set, if borrow flag set, if zero flag set, if greater than flag set
- Hard coded Assembler available- “Assembler.py” - takes assembly language input program file as command line argument
- Clock time period- 2 time units

Microprocessor module wise split-



Instruction set (assembly)-

- Registers are of the form Rx i.e R0, R1, R2,..., R15
- Main memory locations are accessed using Mx i.e M0, M1, …, M31

| INSTRUCTION SYNTAX | NUMBER OF CLOCK CYCLES | FLAGS | EXAMPLE | MACHINE CODE FOR EXAMPLE |
|---|---|---|---|---|
| LOAD Rx My | 3 | | LOAD R4 M2 | 0001_00100_00010_XXXXX |
| STORE RX My | 3 | | STORE R3 M5 | 0010_00011_00101_XXXXX |
| MOV Rx Ry | 3 | | MOV R3 R4 | 0011_00011_00100_XXXXX |
| LDI Rx constant (<255) | 3 | | LDI R6 35 | 0100_00110_00100011_XX |
| JUMP condition tag | 1 | | JUMP U LOOP (loop is a tag for some line- see example) | - |
| operation destination source_register1 source_register2 - all single space separated | | | | |
| ADD Rx Ry Rz | 8 | Carry flag set if carry generated | ADD R3 R5 R6 | 1000_00011_00100_00110 |
| SUB Rx Ry Rz | 8 | Borrow flag set if borrow generated<br><br>Zero flag set if result is zero | SUB R3 R4 R6 | 1001_00011_00100_00110 |
| AND Rx Ry Rz | 8 | Zero flag set if result is zero | AND R3 R4 R6 | 1010_00011_00100_00110 |
| OR Rx Ry Rz | 8 | Zero flag set if result is zero | OR R3 R4 R6 | 1011_00011_00100_00110 |
| SL Rx Ry Rz | 8 | Shift flag contains outgoing bit | SL R3 R4 R6 | 1100_00011_00100_00110 |
| SR Rx Ry Rz | 8 | Shift flag | SR R3 R4 R6 | 1101_XXXXX_00011_ |

| | | | | |
|---|---|---|---|---|
| | | contains outgoing bit | | 00111 |
| CMP Rx Ry | 7 | Greater than flag set if Rx> Ry | CMP R3 R7 | 1110_XXXXX_00011_00111 |

JUMP conditions-

| CONDITION | EFFECT |
|---|---|
| U | Jump to location of tag unconditionally |
| C | Jump to location of tag if carry flag is set |
| B | Jump to location of tag if borrow flag is set |
| Z | Jump to location of tag if zero flag is set |
| G | Jump to location of tag if greater than flag is set |

| FLAG REGISTER | | | | | | | |
|---|---|---|---|---|---|---|---|
| 8 | 7 | 6 | 5 | 4 | 3 | 2 | 0 |
| Not assigned | Not assigned | Not assigned | Greater-than | Zero | Shift out | Borrow | Carry |

| Instruction structure (19 bits) | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Opcode | | | | Operand1 | | | | | Operand2 | | | | | Operand3 | | | | | |
| 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Features and a brief overview-

- Top module microprocessor divided into 3 sub modules- EU (executionunit), CU (controlunit), mainmemory (mainmemory1).
- write, read, load, store and address lines are outputs of CU and are common to mainmemory and EU. ldi and controllines are exclusive inputs to the EU. The common data bus is connected to data buses of both the main memory and CU and assigned on the basis of ldi.
- The main memory module contains 32 bytes of RAM and is written to or read from depending on the combination of read, write, load and store.
- The control unit has 32 words of program memory each of size 19 bits. It possesses a program counter to iterate through instructions. Instructions are written into program memory through file IO. The machine code is copied from a file called “machinecode.txt” written to by the “Assembler.py” assembler python file that takes the assembly language code file as a command line argument.
- Program execution terminates when the opcode 0000 is encountered. The opcode is decoded in the CU by means of an instruction decoder that is implemented through a case statement. Each instruction translates into a state machine that always begins at a rest state State0 ( during which all the piano control signals are set to 0 and program counter incremented) and ends at State0- the end being inclusive.
- The presence of a rest state that each instruction returns to implies that MOV, LOAD, STORE and LDI take 3 cycles- one to read, one to write, one to rest.
- All ALU operations take 8 cycles, although CMP takes only 7 effectively due to redundancy of a third move. ALU utilises mapped registers R0, R1 and R2. Consequently these registers should not be used for data storage. All ALU operations happen via three moves- the first two to the mapped input registers followed by the ALU operation followed by the third move to the destination register.
- The EU contains a 16 byte register bank, an internal bus and the ALU. The EU also alters the flag register which is an input to the CU module. All ALU operations are achieved through behavioural code.

The Assembler-

- The Assembler supports only the current microprocessor structure as it is hard coded.
- Assembly language requires that instructions be separated only by newline characters and that the mnemonics and operands be separated by a single space within an instruction. Tab spaces, multiple white spaces, special characters, comments are not allowed in this version of the Assembler.
- The machine code is always written to the file “machinecode.txt”. The assembly language program file is a command line argument to the “Assembler.py”.

Illustrative examples-

Program to find sum of first 10 natural numbers-

Assembly-

```
LDI R5 10
LDI R4 1
LDI R3 0
FIRST: ADD R3 R3 R5
SUB R5 R5 R4
JUMP Z LOOP
JUMP U FIRST
LOOP: STORE R3 M0
```

Result= 55 stored in M0 (main memory 0)

Machine code-

```
0100_00101_00001010_XX
0100_00100_00000001_XX
0100_00011_00000000_XX
1000_00011_00011_00101
1001_00101_00101_00100
0101_00011_00111_XXXXX
0101_00000_00011_XXXXX
0010_00011_00000_XXXXX
0000_00000_00000_00000
```

To compute 4 factorial-

Assembly-

```
LDI R11 0
LDI R8 1
LDI R7 5
LDI R6 1
LDI R5 2
LOOP2: LDI R3 0
MOV R4 R5
LOOP1: ADD R3 R6 R3
SUB R4 R4 R8
CMP R4 R11
JUMP G LOOP1
MOV R6 R3
ADD R5 R5 R8
CMP R7 R5
JUMP G LOOP2
STORE R6 M1                    result 24 stored in mainmemory location 1
```

Machine code-

```
0100_01011_00000000_XX
0100_01000_00000001_XX
0100_00111_00000101_XX
0100_00110_00000001_XX
0100_00101_00000010_XX
0100_00011_00000000_XX
0011_00100_00101_XXXXX
1000_00011_00110_00011
1001_00100_00100_01000
1110_XXXXX_00100_01011
0101_00100_00111_XXXXX
0011_00110_00011_XXXXX
1000_00101_00101_01000
1110_XXXXX_00111_00101
0101_00100_00101_XXXXX
0010_00110_00001_XXXXX
0000_00000_00000_00000
```