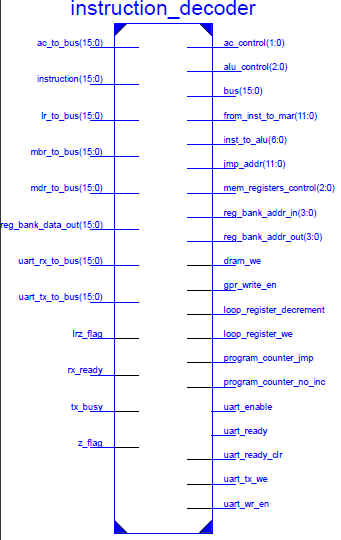
## Instruction Decoder



Instruction Decoder receives the binary instruction from the Instruction Register and issues all the control signals required for executing the instruction. Other than control signals it also issues immediate constants extracted from the instruction to other modules. It also includes the Main Bus multiplexer which drives the Main Bus. In total, Instruction Decoder issues 19 control signals and 6 immediate constant outputs. Other than the instruction itself, outputs of AC, LR, MBR, MDR, UARTTX, UARTRX and Register Bank (registers which will be driving the bus) and the 4 flags (Z flag, LRZ flag, TXBUSY flag, RXREADY flag) are given as inputs to this module. Instruction Decoder is a combinational module. A summary of the outputs is given below.

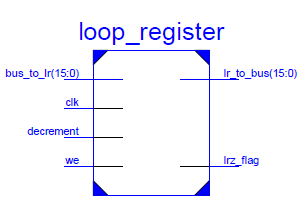
**Immediate constants:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Reg\_bank\_addr\_out [4 bits] | Reg\_bank\_addr\_in [4 bits] | Inst\_to\_alu [7 bits] | Jmp\_addr [12 bits] | From\_inst\_to\_mar  [12 bits] |
| Address of the general purpose register which is read into the main bus | Address of the general purpose register which is written from the main bus | Immediate constant from the instruction to the ALU | Jump address from the instruction to the Program Counter | Immediate LOAD/STORE address offset from the instruction |

**Control signals:**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| INSTRUCTION | ACCUMULATOR | | ALU | | | MEMORY REGISTERS | | | REGISTER BANK | PROGRAM COUNTER | | LOOP REGISTER | | UART | | | | | DATA RAM |
| AC WRITE ENABLE | AC INPUT SELECT (0:BUS, 1:ALU) | ALU OPERATION SELECTION (0:ADD, 1:SUB, 2:MUL, 3:DIV, 4:SHR, 5:SHL) | | | MBR WRITE ENABLE | MDR WRITE ENABLE | MDR INPUT SELECT (0:BUS, 1:MEMORY) | REGISTER BANK WRITE ENABLE | PROGRAM COUNTER JMP | PROGRAM COUNTER NO\_INCREMENT | LOOP REGISTER DECREMENT | LOOP REGISTER WRITE ENABLE | UART READY | UART READY\_CLR | UART WR\_EN | UART ENABLE | UARTTX WRITE ENABLE | DATA RAM WRITE ENABLE |
| NOP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| UARTSEND TXBUSY = 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| UARTSEND TXBUSY = 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| UARTREAD RXREADY = 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| UARTREAD RXREADY = 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| ADD | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SUB | 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MUL | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| DIV | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SHR | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SHL | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| LOAD | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| STORE | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| JUMP | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JMPZ Z=0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JMPZ Z=1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JMPDEC LRZ=0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| JMPDEC LRZ=1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| MOVE | WRITE ENABLE SIGNAL OF THE CORRESPONDING REGISTER WILL BE 1 DEPENDING ON THE DESTINATION. ALL OTHER SIGNALS WILL REMAIN 0 | | | | | | | | | | | | | | | | | | |

## Loop Register



Loop register is a normal 16-bit data register except it has its own subtractor and associated LRZ (Loop Register Zero) flag. This register is sensitive to the positive edge of the **clk.** The included subtractor will decrement the value in the register in each clock cycle until the **decrement** signal remains high. **bus\_to\_lr** is the input from the Main Bus and when **we** is high, the register value will be replaced by this value.

## Test Bench

Test Bench was used for the simulation and debugging process of the processor in ‘ISim’, a Verilog simulation software. This module includes an instance of the **processor\_top\_module** and an instance of the **UART** module placed externally. This allowed to carry out a complete test regarding the functionality of all the modules comprising the processor. Test Bench does not have any inputs or outputs.

## Design Summary

|  |  |  |  |
| --- | --- | --- | --- |
|  | processor\_top\_module Project Status (05/03/2019 - 15:50:36) | | |
| Project File: | processor\_top\_module.xise | Parser Errors: | No Errors |
| Module Name: | processor\_top\_module | Implementation State: | Placed and Routed |
| Target Device: | xc6slx45-2csg324 | Errors: | No Errors |
| Product Version: | ISE 14.7 | Warnings: | 32 Warnings (1 new) |
| Design Goal: | Balanced | Routing Results: | All Signals Completely Routed |
| Design Strategy: | Xilinx Default (unlocked) | Timing Constraints: | All Constraints Met |
| Environment: | System Settings | Final Timing Score: | 0 (Timing Report) |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Device Utilization Summary | | | | | [-] |
| Slice Logic Utilization | Used | Available | Utilization | Note(s | ) |
| Number of Slice Registers | 202 | 54,576 | 1% |  |  |
| Number used as Flip Flops | 202 |  |  |  |  |
| Number used as Latches | 0 |  |  |  |  |
| Number used as Latch-thrus | 0 |  |  |  |  |
| Number used as AND/OR logics | 0 |  |  |  |  |
| Number of Slice LUTs | 843 | 27,288 | 3% |  |  |
| Number used as logic | 827 | 27,288 | 3% |  |  |
| Number using O6 output only | 651 |  |  |  |  |
| Number using O5 output only | 24 |  |  |  |  |
| Number using O5 and O6 | 152 |  |  |  |  |
| Number used as ROM | 0 |  |  |  |  |
| Number used as Memory | 12 | 6,408 | 1% |  |  |
| Number used as Dual Port RAM | 12 |  |  |  |  |
| Number using O6 output only | 0 |  |  |  |  |
| Number using O5 output only | 0 |  |  |  |  |
| Number using O5 and O6 | 12 |  |  |  |  |
| Number used as Single Port RAM | 0 |  |  |  |  |
| Number used as Shift Register | 0 |  |  |  |  |
| Number used exclusively as route-thrus | 4 |  |  |  |  |
| Number with same-slice register load | 0 |  |  |  |  |
| Number with same-slice carry load | 4 |  |  |  |  |
| Number with other load | 0 |  |  |  |  |
| Number of occupied Slices | 295 | 6,822 | 4% |  |  |
| Number of MUXCYs used | 336 | 13,644 | 2% |  |  |
| Number of LUT Flip Flop pairs used | 881 |  |  |  |  |
| Number with an unused Flip Flop | 693 | 881 | 78% |  |  |
| Number with an unused LUT | 38 | 881 | 4% |  |  |
| Number of fully used LUT-FF pairs | 150 | 881 | 17% |  |  |
| Number of unique control sets | 19 |  |  |  |  |
| Number of slice register sites lost to control set restrictions | 46 | 54,576 | 1% |  |  |
| Number of bonded IOBs | 20 | 218 | 9% |  |  |
| Number of LOCed IOBs | 11 | 20 | 55% |  |  |
| Number of RAMB16BWERs | 68 | 116 | 58% |  |  |
| Number of RAMB8BWERs | 0 | 232 | 0% |  | |
| Number of BUFIO2/BUFIO2\_2CLKs | 0 | 32 | 0% |  | |
| Number of BUFIO2FB/BUFIO2FB\_2CLKs | 0 | 32 | 0% |  | |
| Number of BUFG/BUFGMUXs | 2 | 16 | 12% |  | |
| Number used as BUFGs | 2 |  |  |  | |
| Number used as BUFGMUX | 0 |  |  |  | |
| Number of DCM/DCM\_CLKGENs | 0 | 8 | 0% |  | |
| Number of ILOGIC2/ISERDES2s | 0 | 376 | 0% |  | |
| Number of IODELAY2/IODRP2/IODRP2\_MCBs | 0 | 376 | 0% |  | |
| Number of OLOGIC2/OSERDES2s | 0 | 376 | 0% |  | |
| Number of BSCANs | 0 | 4 | 0% |  | |
| Number of BUFHs | 0 | 256 | 0% |  | |
| Number of BUFPLLs | 0 | 8 | 0% |  | |
| Number of BUFPLL\_MCBs | 0 | 4 | 0% |  | |
| Number of DSP48A1s | 1 | 58 | 1% |  | |
| Number of ICAPs | 0 | 1 | 0% |  | |
| Number of MCBs | 0 | 2 | 0% |  | |
| Number of PCILOGICSEs | 0 | 2 | 0% |  | |
| Number of PLL\_ADVs | 0 | 4 | 0% |  | |
| Number of PMVs | 0 | 1 | 0% |  | |
| Number of STARTUPs | 0 | 1 | 0% |  | |
| Number of SUSPEND\_SYNCs | 0 | 1 | 0% |  | |
| Average Fanout of Non-Clock Nets | 5.35 |  |  |  | |

**Timing Summary:**

Speed Grade: -2

Minimum period: 119.206ns (Maximum Frequency: 8.389MHz)

Minimum input arrival time before clock: 5.158ns

Maximum output required time after clock: 4.240ns

Maximum combinational path delay: No path found

**Technical Aspects:**

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Value** | **Units** |
| Input clock speed | 100 | MHz |
| Processor clock speed | 6.25 | MHz |
| Average clock cycles per instruction | 1 | Clock cycles |
| Instruction memory size | 16 x 4096 | Bits |
| Data memory size | 8 x 131072 | Bits |
| Register width | 16 | Bits |
| I/O method | UART | - |
| UART speed | 9600 | Bauds |
| UART flow control | None | - |
| UART parity | None | - |
| UART stop bits | 1 | - |
| UART no of data bits | 8 | Bits |
| USB-UART bridge model | XR21V1410 | - |

Additional software required for communication:

* XR21V1410 USB-UART bridge driver – can be downloaded from [www.exar.com](http://www.exar.com).
* A serial communication terminal (Eg: Putty)

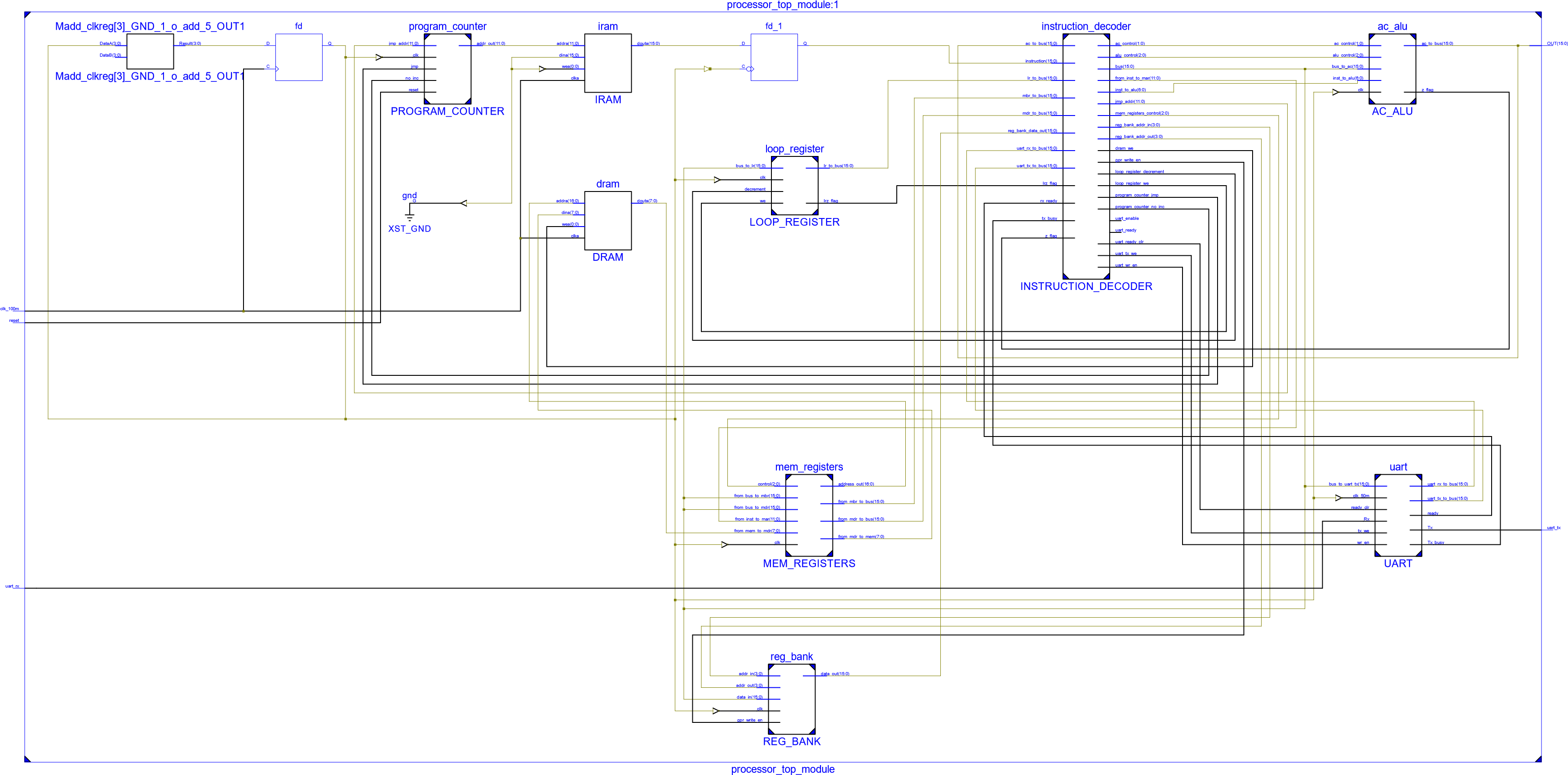
**Typical instruction cycle:**

Nth Instruction is loaded into Instruction Register.

Clock signal (6.25MHz)

Nth instruction is executed. Program Counter increments to N+1.

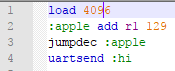
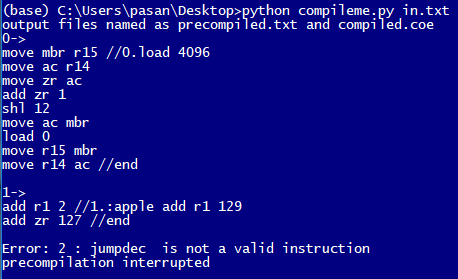
Previous instruction (N-1) is executed. Program Counter increments to N.

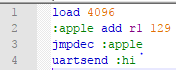
**Schematic:**

## Supplementary Tools

**Compiler:**

An assembly-to-machine code compiler (compileme.py) was written using Python (version 3) in order to convert human readable instructions into machine code. This compiler includes features such as label resolution, immediate addressing resolution for LOAD and STORE instructions, string resolution for UARTSEND and immediate constant resolution for ALU instructions. A file including instructions written in assembly is given as the input. Output is two files, one in assembly itself but with resolved labels, addresses, strings and constants, and the other includes instructions in machine code. An automatic commenting and error detection mechanism is also included. A compilation example is given below.

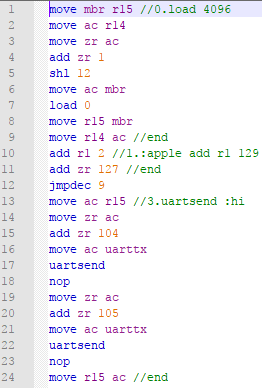
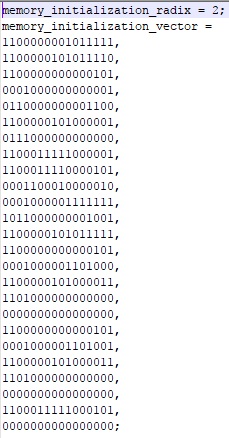




Compiler reporting about the incorrect instruction

Assembly code with corrected instruction

Assembly code with incorrect instruction



Output file 2: in machine code

Output file 1: in assembly (with resolved labels, addresses, strings and constants)

**Simulator:**

A simulator (tryme.py) was written in Python (version 3) for the purpose of testing and verifying the algorithms. This was based on the compiler mentioned above. The processor architecture was implemented using a dictionary resembling the register stack. Memory was also implemented using a dictionary. Values of these dictionaries were written into an output file by the simulator after the execution of each instruction. This allowed to investigate the values of each register and memory location through each step of the algorithm. Following image shows a part of an output file generated.

