8-BIT MICROPROCESSOR

V 1.0

JANUARY 25, 2017

PORTLAND STATE UNIVERSITY
vinosake@pdx.edu

1. Introduction

I have designed an 8-bit microprocessor using system Verilog and simulated using Modelsim/Questasim. This document describes my processor design, testing codes used to verify it.

Challenges faced in the project:

- Testing:
  a) Performed testing on each module individually using system Verilog test benches to ensure all are performing well as per functional design.
  b) Once all the modules are put together, I had to ensure the system is performing operations as per timing requirements.
- To overcome the painstaking process of creating hexadecimal Instruction for CPU program, I wrote an Assembler for my processor which parses an input file and creates output files for system Verilog simulation.

2. CPU Architecture

As the processor is an 8-bits microprocessor, it has 8-bits internal data bus which transfers data between internal modules and connects to external RAM & ROM. All the registers in the CPU are designed using big-endian format, with the bits ordered from most significant bit to least significant bit.

2.1 Instruction format

Length of Instruction format is 16 bits which has opcodes, source type, source register, destination register & condition

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Opcode					**Source Type**		**NA**	**Source register**			**Destination register**			**Condition**	

2.2 Opcodes

Instruction	Opcode	Operands
NOP	0x00	-
ADD	0x01	Reg, Reg/Immediate
SUB	0x02	Reg, Reg/Immediate
MUL	0x03	Reg, Reg/Immediate
AND	0x04	Reg, Reg/Immediate
OR	0x05	Reg, Reg/Immediate
LSR	0x06	Reg
LSL	0x07	Reg
NOT	0x08	Reg
ASR	0x09	Reg
ASL	0x0A	Reg
NEG	0x0B	Reg
LOAD	0x10	Reg, Immediate/Memory
STORE	0x11	Reg, Immediate/Memory
MOVE	0x12	Memory, Memory
JUMP	0x0F	Memory
HALT	0x1F	-

2.3 Jump conditions

These conditions are used when Jump conditions are executed.

Condition	Bit Designation
Always	00
Carry	01
Zero	10
Negative	11

2.4 Instruction set

2.4.1 NOP

No operation. After reading this instruction, processor continuously to next instruction without any extra clock cycle.

2.4.2 ADD

Instruction has two operands, where first operand can be only from registers and another operand can be register or immediate. Performs Addition operation between operands and result is stored in the destination register of GP registers. If the sum is greater than 8-bits then carry bit is set high. If the sum is zero, then zero flag is set high and Neg flag is set high if MSB of sum is 1.

2.4.3 SUB

Instruction has two operands, where first operand can be only from registers and another operand can be register or immediate. Performs subtraction operation between operands and result is stored in the destination register of GP registers. Carry bit is set high when there is borrow. If the sub is zero, then zero flag is set high and Neg flag is set high if MSB of sub is 1.

2.4.4 MUL

Instruction has two operands, where first operand can be only from registers and other operand can be from register or immediate. Performs multiplication operation between operands. As the product 16 bits, low 8 bits are stored in destination register and high 8 bits are stored in adjacent 8-bit general purpose register. Good practice is to use 16-bit general purpose register pair.

2.4.5 AND

Instruction has two operands, where first operand can be only from registers and another operand can be register or immediate. Performs bitwise AND operation between operands and result is stored in destination register of GP registers. If the result is zero, the zero flag is set high and neg flag is set high if MSB of result is 1.

2.4.6 OR

Instruction has two operands, where first operand can be only from registers and another operand can be register or immediate. Performs bitwise OR operation between operands and result is stored in destination register of GP registers. If the result is zero, the zero flag is set high and neg flag is set high if MSB of result is 1.

2.4.7 LSR

Instruction has only one operand and can be from registers. Performs logical shift right operation where all eight bits are shifted right by introducing zero in MSB. Result is stored in the same register. If the result is zero, the zero flag is set high and neg flag is set high if MSB of result is 1. Carry flag is set high when 1 is shifted out.

2.4.8 LSL

Instruction has only one operand and can be from registers. Performs logical shift left operation where all eight bits are shifted left by introducing zero in LSB. Result is stored in the same register. If the result is zero, the zero flag is set high and neg flag is set high if MSB of result is 1. Carry flag is set high when 1 is shifted out.

2.4.9 ASR

Instruction has only one operand and can be from registers. Performs Arithmetic shift right operation where MSB remains same and lower 7 bits are shifted right. Result is stored in the same register. If the result is zero, the zero flag is set high and neg flag is set high if MSB of result is 1. Carry flag is set high when 1 is shifted out.

2.4.10 ASL

Instruction has only one operand and can be from registers. Performs Arithmetic shift left operation where LSB remains same and higher 7 bits are shifted left. Result is stored in the same register. If the result is zero, the zero flag is set high and neg flag is set high if MSB of result is 1. Carry flag is set high when 1 is shifted out.

2.4.11 NOT

Instruction has only one operand and can be from registers. Performs complement operation where all bits are inverted. Result is stored in the same register. If the result is zero, the zero flag is set high and neg flag is set high if MSB of result is 1. Carry flag is set high when 1 is shifted out.

2.4.12 NEG

Instruction has only one operand and can be from registers. Performs 2’s complement operation. Result is stored in the same register. If the result is zero, the zero flag is set high and neg flag is set high if MSB of result is 1. Carry flag is set high when 1 is shifted out.

2.4.13 LOAD

Instruction loads a value into a general-purpose register. Value can be from register, immediate value or memory location.

2.4.14 STORE

Instruction stores a value in register into RAM. Address of RAM is provided.

2.4.15 MOVE

Instruction moves data from one memory location into other memory location. Source and destination memory locations are provided. Destination memory location can only be from RAM.

2.4.16 JUMP

Instruction relocates the instruction address in program counter based on jump conditions.

- JMP – Always jump
- JC – Jump if carry
- JZ – Jump if zero
- JS – Jump if Negative

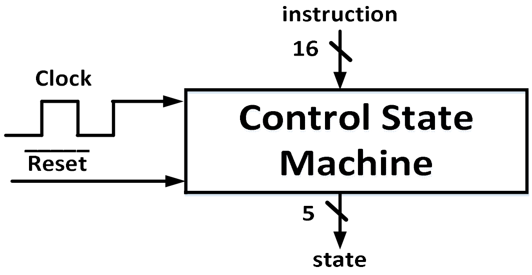
2.4.17 HALT

Instruction halts the processor and no more instructions are read. Processor stays in HALT state until it is reset.

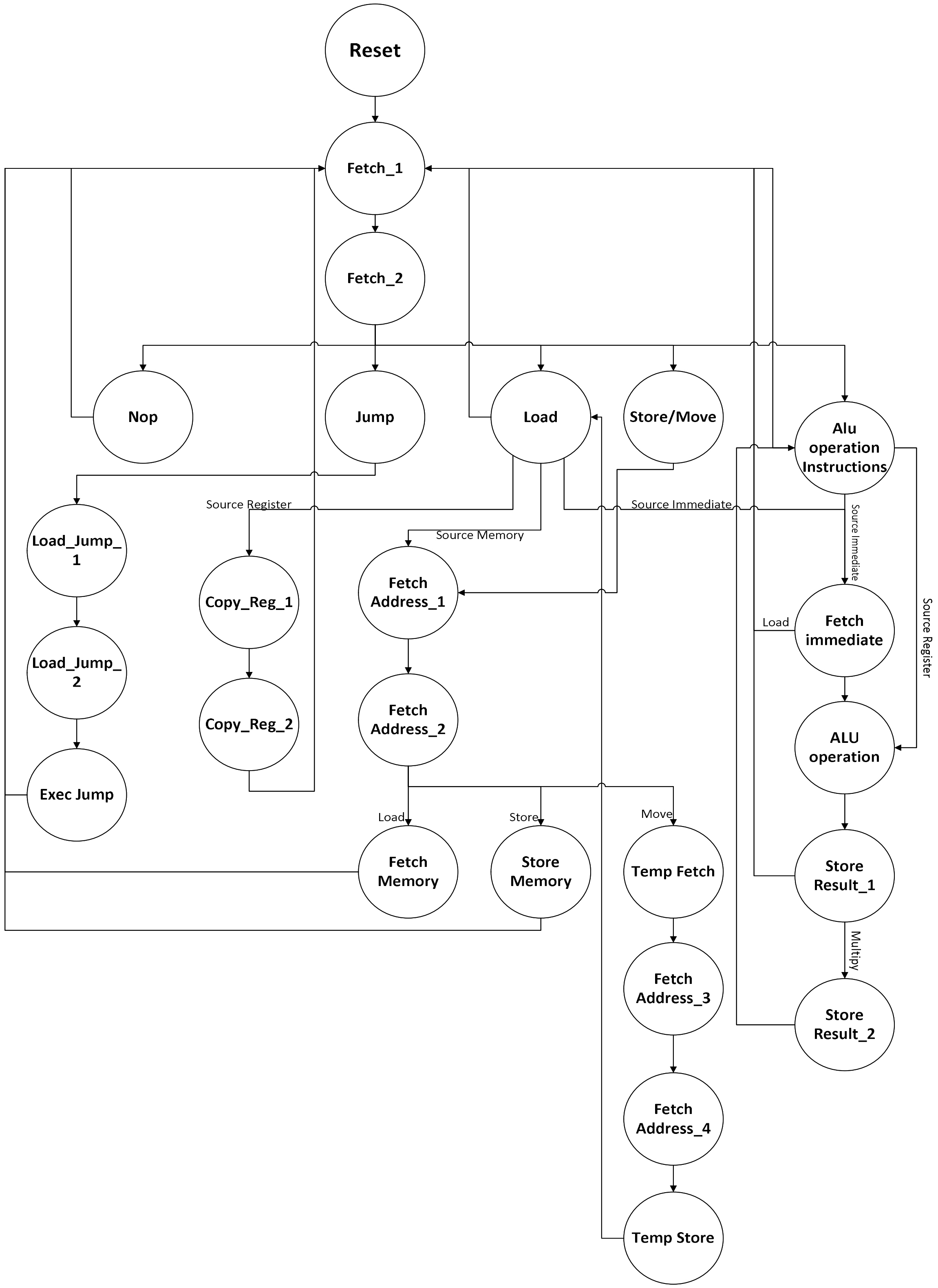
3. Modules Design

3.1 Control State Machine

Control state machine reads the instruction from the instruction register, decodes it and decides the next state based on the instruction.

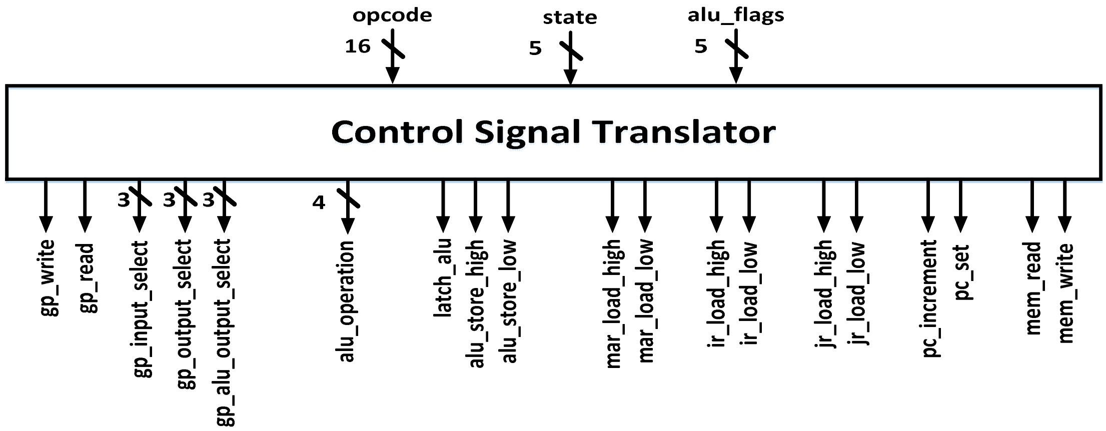


3.1.1 State Diagram

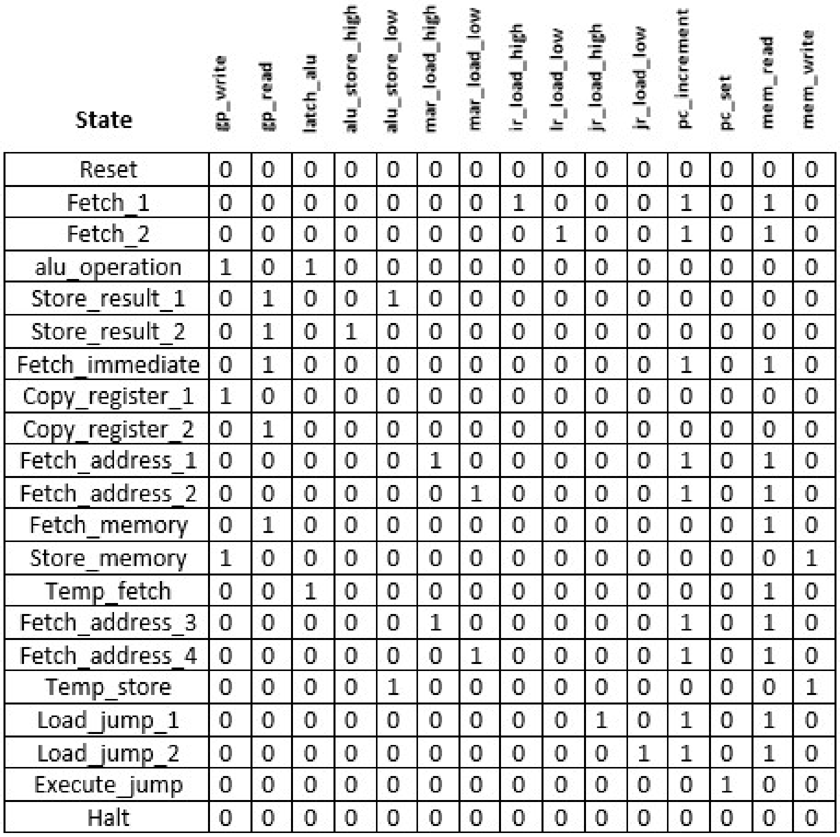


3.2 Control Signal Translator

Control signal translator produces control signals to other modules based on the opcode, state & alu flags.



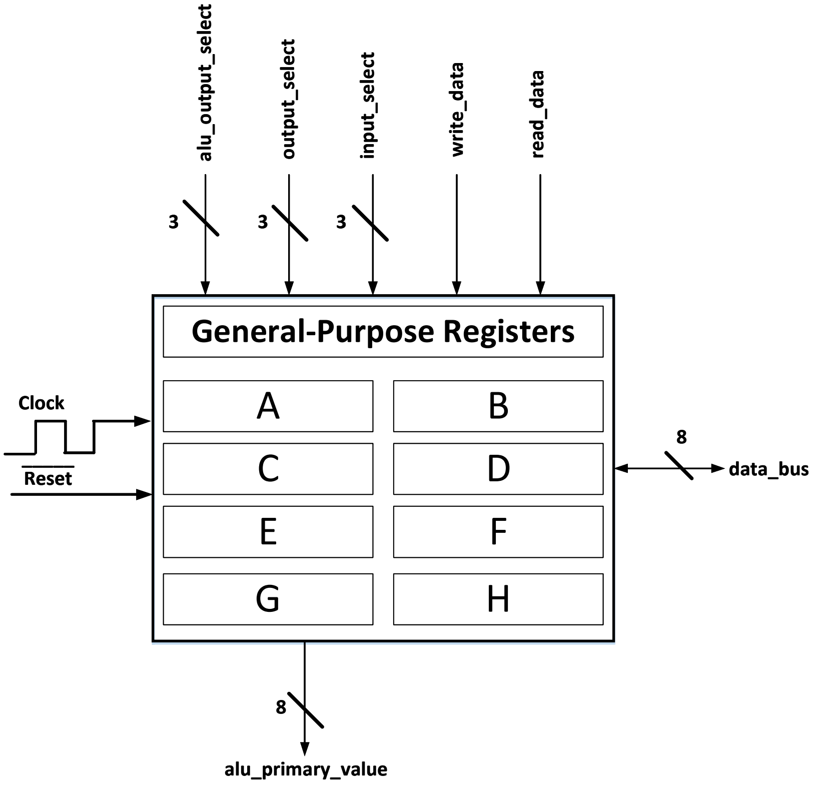
3.2.1 State to control signals matrix



State	gp_write	gp_read	latch_alu	alu_store_high	alu_store_low	mar_load_high	mar_load_low	ir_load_high	ir_load_low	jr_load_high	jr_load_low	pc_increment	pc_set	mem_read	mem_write
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Fetch_1	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0
Fetch_2	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0
alu_operation	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0
Store_result_1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0
Store_result_2	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0
Fetch_immediate	0	1	0	0	0	0	0	0	0	0	0	1	0	1	0
Copy_register_1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Copy_register_2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
Fetch_address_1	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0
Fetch_address_2	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0
Fetch_memory	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0
Store_memory	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1
Temp_fetch	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0
Fetch_address_3	0	0	0	0	0	1	0	0	0	0	0	1	0	1	0
Fetch_address_4	0	0	0	0	0	0	1	0	0	0	0	1	0	1	0
Temp_store	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1
Load_jump_1	0	0	0	0	0	0	0	0	0	1	0	1	0	1	0
Load_jump_2	0	0	0	0	0	0	0	0	0	0	1	1	0	1	0
Execute_jump	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
Halt	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

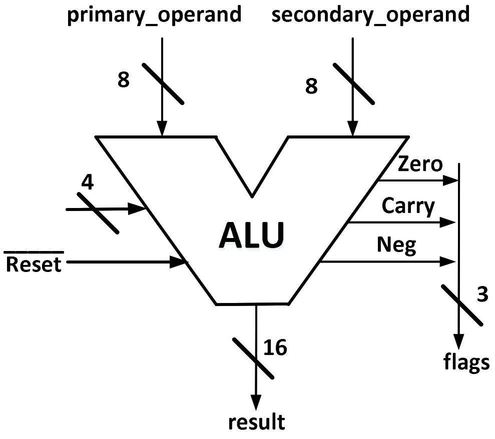
3.3 General Purpose Registers

General purpose register block consists of 8 eight-bit registers which are used for data manipulation. They are grouped into pairs to form 4 sixteen-bit registers to store multiplication result.



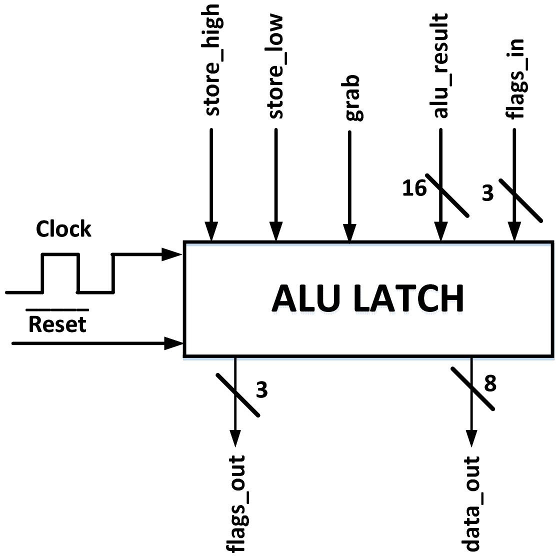
3.4 ALU

Arithmetic logical unit receives 2 operands where first operand is from General Purpose Unit and other operand is from Data bus. It produces flags based on the result for an operation. Output of ALU is a 16-bit data.



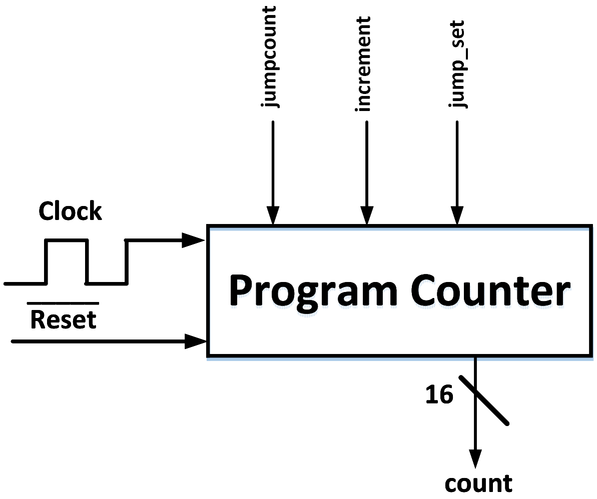
3.5 ALU Latch

The ALU latch grabs the result of the ALU operation, holds it and puts on the data bus when store signals are asserted. It also grabs alu flags.



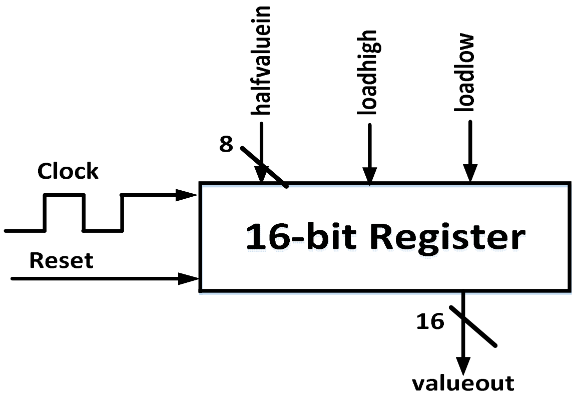
3.6 Program Counter

The Program counter holds the next instruction which is to be executed. Count is incremented when increment signal is set at every positive edge of clock. It also loads Jump address when performing Jump operations. When jump, set is asserted count is loaded with jumpcount.



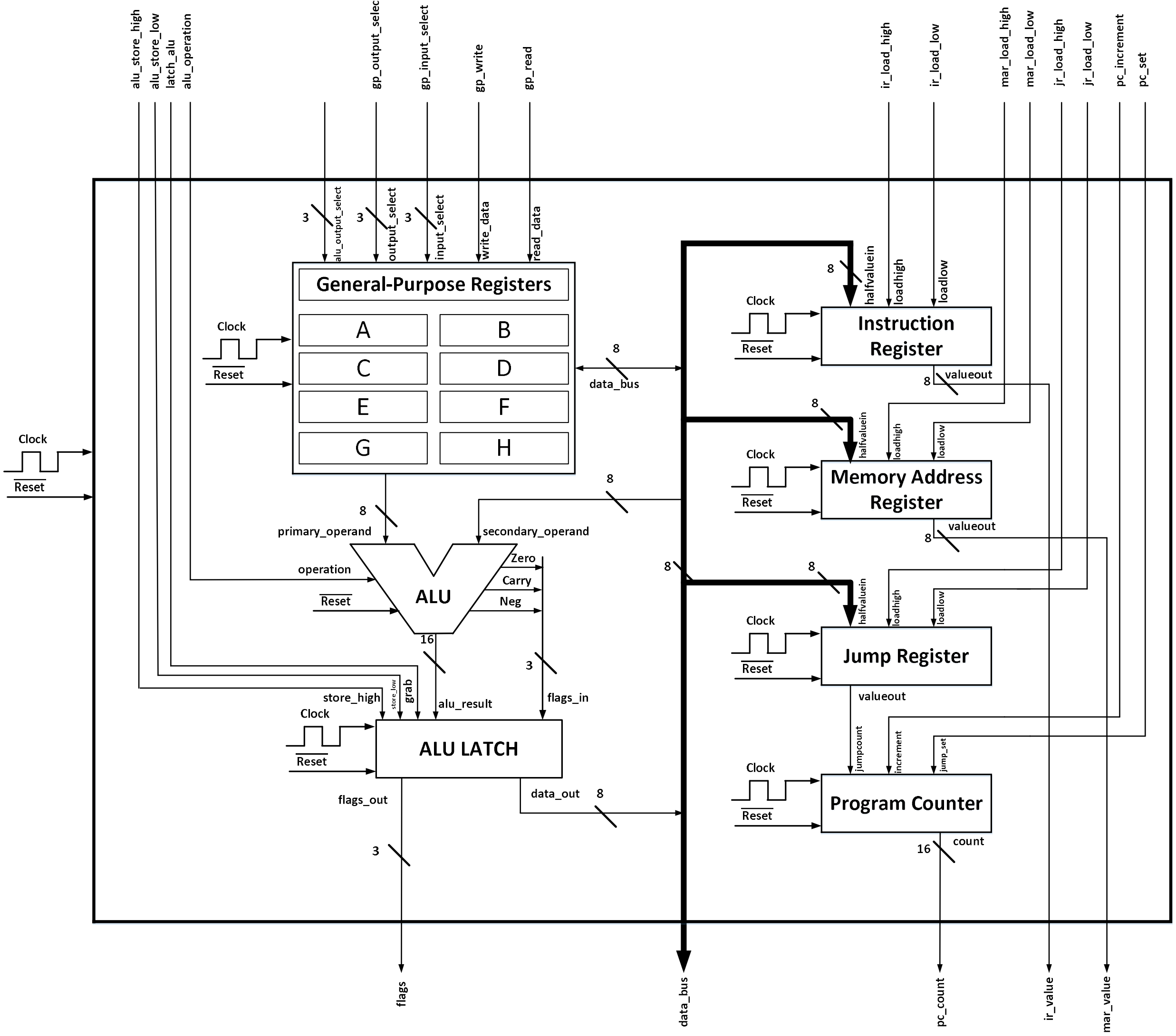
3.7 16-bit Register

The 16-bit register is instantiated 3 times to create Memory Address Register, Instruction Register & Jump Register. 8-bits are loaded into 16-bit register from the databus based on the load signals. Output is a 16-bit data.



3.8 Data Path

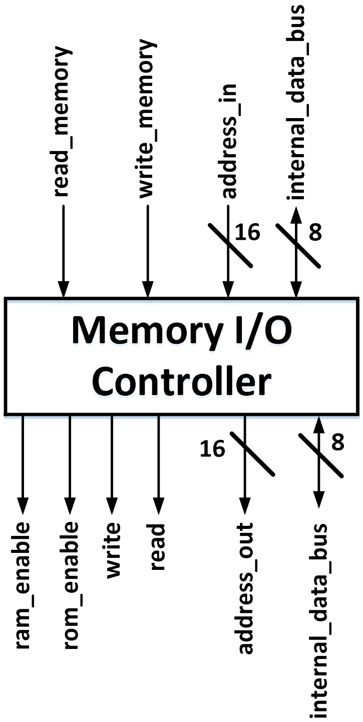
The Datapath module combines the general-purpose registers, ALU, ALU latch, Instruction register, Memory address register, Jump register, program counter modules and 8-bit data bus. Databus is a bidirectional and transfer data between modules internally in data path and externally with memory devices.



3.9 Memory I/O Controller

Memory I/O Controller performs memory mapping to locate ROM and RAM in address space. It also provides control signals to Enable and Disable Memory devices.

As the Address bus is 16-bit maximum address space would be 2^{16} i.e 0 – 65,535. I have assigned half of the address space to ROM (0000 – 1FFF) and other half of address space to RAM (2000 – 3FFF). The ram & rom enable signals are active low.



	Input Address	
{read_memory, write_memory}	ROM (0000 – 1FFF)	RAM (2000 - 3FFF)

00	rom_enable = 1
	ram_enable = 1
		rom_enable = 1
		ram_enable = 1

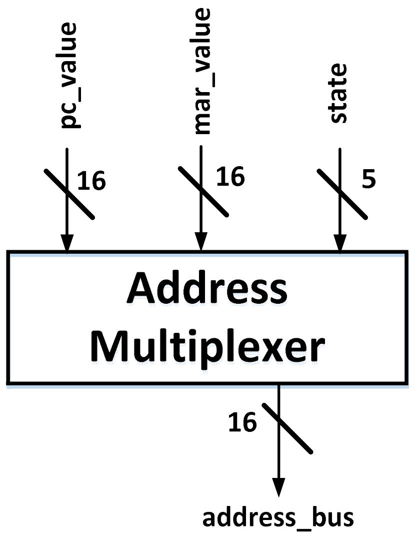
01	Invalid	rom_enable = 1
		ram_enable = 0

10	rom_enable = 0
	ram_enable = 1
		rom_enable = 1
		ram_enable = 0

11	rom_enable = 1
	ram_enable = 1
		rom_enable = 1
		ram_enable = 1

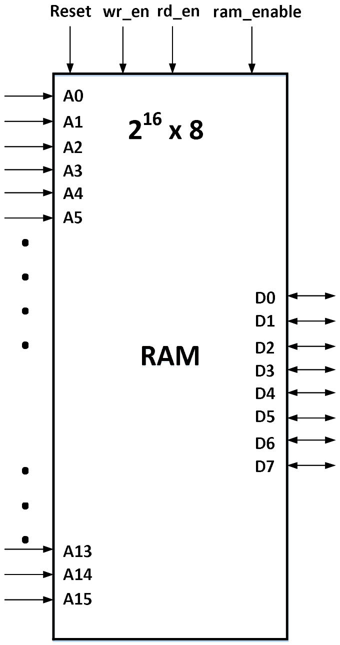
3.10 Address Multiplexer

Address Multiplexer switches address between MAR & PC based on state. When the processor is performing LOAD or STORE operation, address multiplexer uses address from MAR otherwise it uses address from PC.



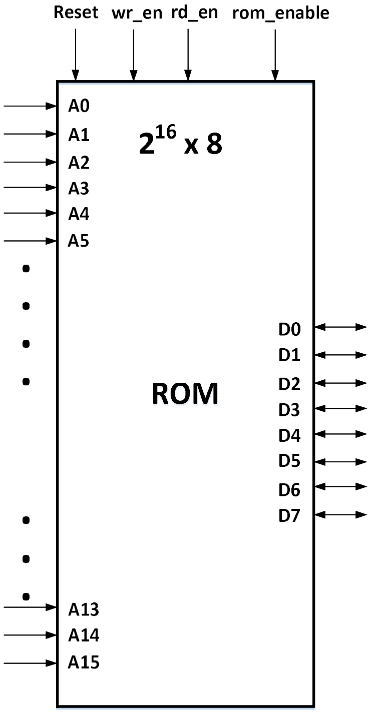
3.11 RAM

As there are 16 bits of address we get 2^{16} locations to store data. I have dedicated 2^8 locations of RAM i.e (2000 – 3FFF) for the processor to read and write data. RAM module gets activated for read or write operations when ram enable is low.

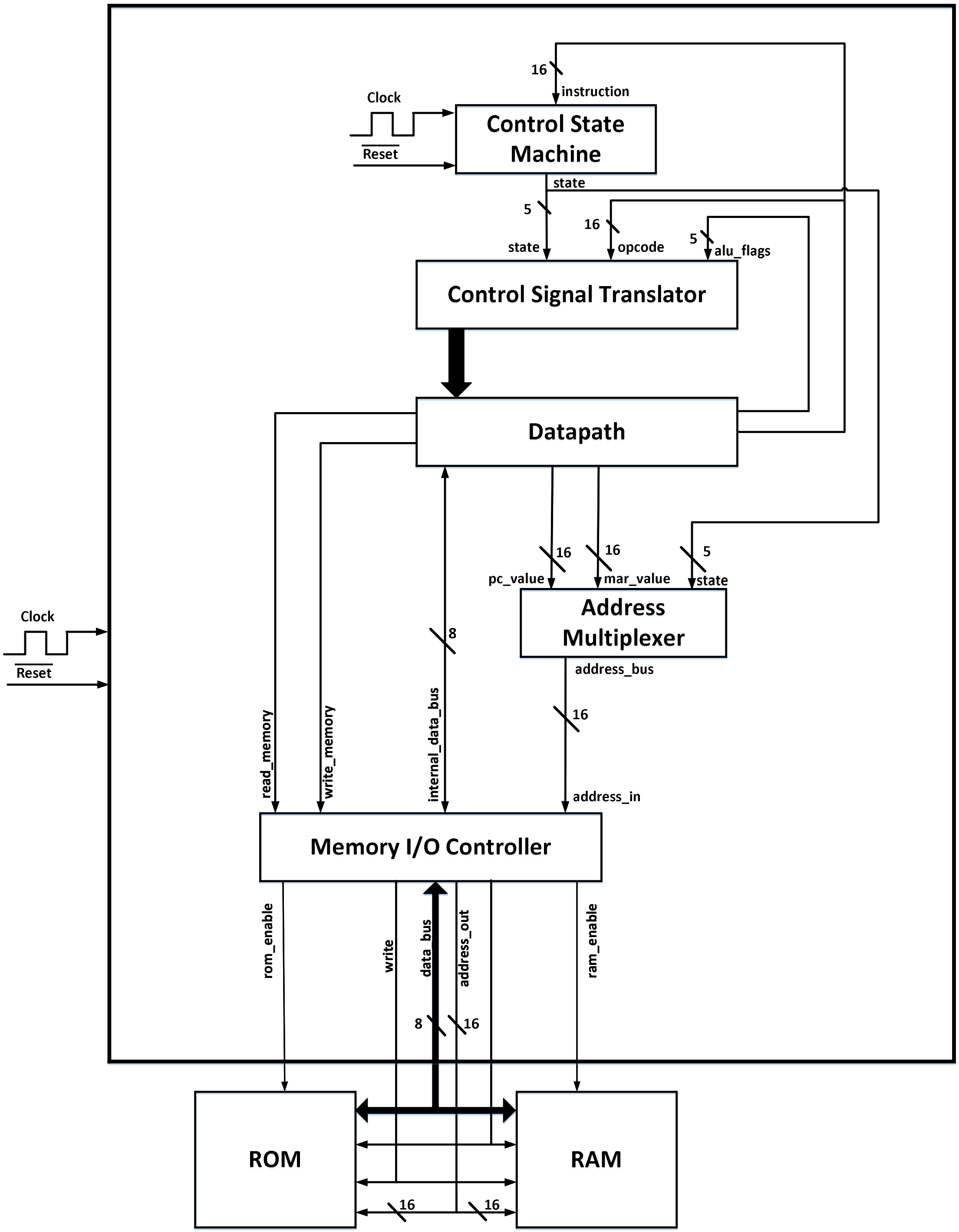


3.12 ROM

As there are 16 bits of address we get 2^{16} locations to store data. I have dedicated 2^{8} locations of ROM i.e (0000 – 1FFF) for the processor to read instructions. ROM module gets activated for read operations when rom enable is low.



3.13 CPU


16
instruction
Clock
Control State Machine
Reset
state
5
16
5
state
opcode
alu_flags
Control Signal Translator
Datapath
16
16
5
pc_value
mar_value
state
Clock
8
Address Multiplexer
Reset
address_bus
16
read_memory
write_memory
internal_data_bus
address_in
Memory I/O Controller
rom_enable
write
data_bus
address_out
ram_enable
8
16
ROM
RAM
16
16