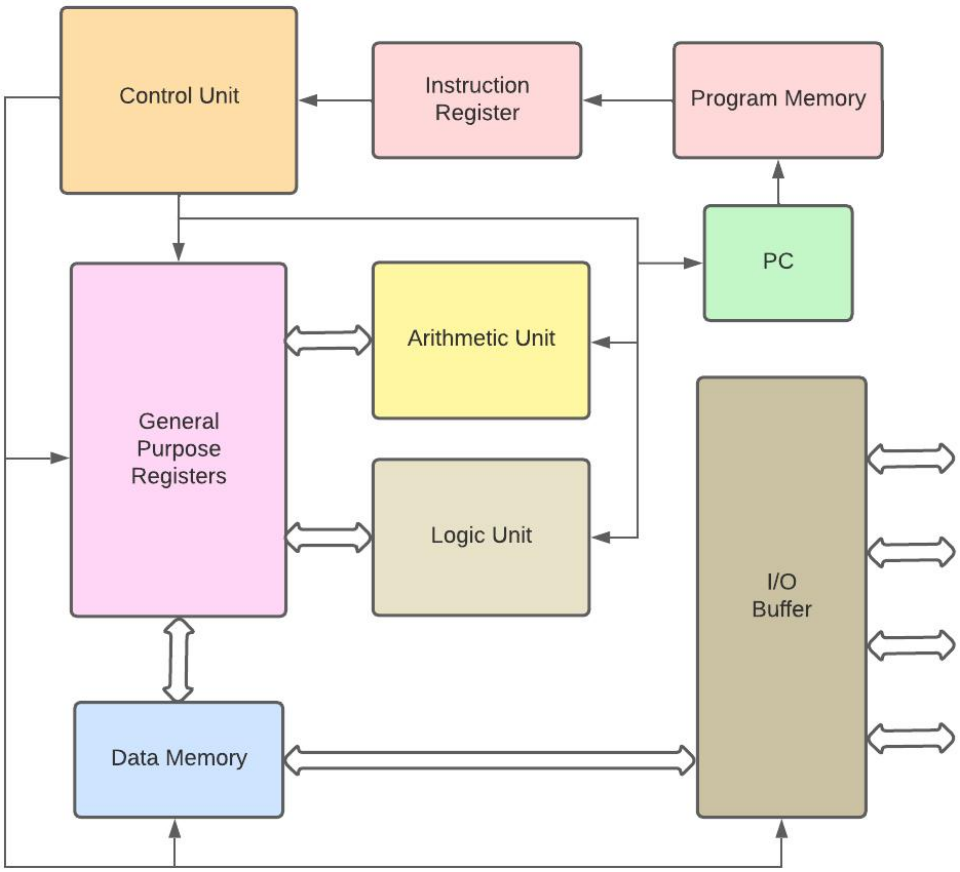
General Processor Architecture



1 Registers and Memory

1.1 General Purpose Registers (GPR[x]) :- There are 32 GPRs ,each of 16 bit in size. The registers are acronymed as Rx (where x ranges from 0 - 31 eg: R0,R11..R29).

1.2 Program Counter (PC) :- The program counter is a 32-bit register which stores the address of the next instruction to be executed.

1.3 Instruction Register (IR[x]) :- The Instruction Register is a 32-bit register which fetches the instruction from the instruction memory. The address of the instruction to be fetched is pointed by the PC.

1.4 Instruction Memory (inst_mem[x]) : The processor uses Hardvard Architecture which employs dedicated memory for data and program. A total of 32 instructions each of 32-bit size can be stored at a time in the memory.

1.5 Data Memory (data_mem[x]): Data memory acts as a interlink between the GPRs and the I/O buffer. Any data which needs to be stored in the GPRs via I/O pins or vice versa is required to be first stored in the data memory. There are a total of 32 (16-bit) memory locations available within the CPU.

1.6 Special General Purpose Register (SGPR) : This is a special purpose register which stores the MSB 16-bits result of the multiplication operation.

1.7 Operation Status Register (OSR[15:0]) : The OSR consists of flag and state variables which store values of some parameters after an operation is performed.

1.7.1 Zero Flag (OSR[0]) : Zero flag is raised if the destination register results in a zero value (0x0000).

1.7.2 Sign Flag (OSR[1]) : Indicates whether the sign bit (MSB) of a resultant value is set or not. Sign bit is set to 1 for a negative value and set to 0 otherwise.

1.7.3 Carry Flag (OSR[2]) : The carry flag is set if there arises a carry as a result of addition of two numbers.

1.7.4 Overflow Flag (OSR[3]) : If there is an overflow condition as a result of performing an addition or subtraction, the overflow flag is raised.

1.7.5 Stop Flag (OSR[8]) : The stop flag is raised when the processor 'halt' command is executed.

1.7.6 FSM State (OSR[12:10]) : The state/cycle of the machine can be determined by FSM state bits. '000 - IDLE' , '001 - FETCH' , '010 - DECODE/EXECUTE' , '011 - ASSIGN FLAGS' , '100 - STOP'

0	1	2	3	4-7	8	9	10-12	13-15
ZERO	SIGN	CARRY	OVFLOW	-	STOP	-	STATE	-

Operation Status Register

STATE	'10-12'
IDLE	000
FETCH	001
DECODE/EXECUTE	010
ASSIGN	011
STOP	100

State Table

2 Instruction Set Architecture (ISA)

Each Instruction code has a fixed size of 32-bits. The functionalities of each word may be vary with the mode of operation. The ISA supports Immediate addressing and register addressing modes. In case of immediate addressing, the value to be used is preceded by '#'. The GPRs can be addressed by specifying the index of the register to be used preceded by R. For instance , the 8th GPR can be addressed as R8 and the xth register can be addressed as Rx.

The Instruction set of the processor can be represented as below:

2.1 Register Addressing Mode : The mode bit is set to '0'.

31 : 27	26 : 22	21 : 17	16	15 : 11	10 : 0
Opcode	Dest. Reg	Src. Reg 1	Mode	Src. Reg 2	-Unused-
xxxxx	xxxxx	xxxxx	0	xxxxx	xxxxxxxxxxx

2.2 Immediate Addressing Mode : The mode bit is set to '1'. The LSB 16 bits are used to represent the immediate value.

31 : 27	26 : 22	21 : 17	16	15 : 0
Opcode	Dest. Reg	Src. Reg 1	Mode	Imm. value
xxxxx	xxxxx	xxxxx	1	xxxxxxxxxxxxxxxx

2.3 Instruction Set

There are a total of 26 instructions which can be performed by the CPU. They are as follows.

S No.	Instruction	Description	Mode	Syntax
1.	MOVSGPR	Move the contents of SGPR to Rx	Reg.	MOVSGPR Rx
2.	MOV	Move contents of Ry to Rx	Reg.	MOV Rx Ry
3.	ADD	Add contents of Ry and Rz then assign the result to Rx	Reg.	ADD Rx Ry Rz
4.	SUB	Subtract contents of Ry and Rz then assign the result to Rx	Reg.	SUB Rx Ry Rz
5.	MUL	Multiply contents of Ry and Rz then assign the LSB 16 bits to Rx	Reg.	MUL Rx Ry Rz
6.	OR	Bitwise OR Ry with Rz and assign the result to Rx	Reg.	OR Rx Ry Rz
7.	AND	Bitwise AND Ry with Rz and assign the result to Rx	Reg.	AND Rx Ry Rz
8.	XOR	Bitwise XOR Ry with Rz and assign the result to Rx	Reg.	XOR Rx Ry Rz

9.	XNOR	Bitwise XNOR Ry with Rz and assign the result to Rx	Reg.	XNOR Rx Ry Rz
10.	NAND	Bitwise NAND Ry with Rz and assign the result to Rx	Reg.	NAND Rx Ry Rz
11.	NOR	Bitwise NOR Rywith Rz and assign the result to Rx	Reg.	NOR Rx Ry Rz
12.	NOT	Bitwise NOT Ry and assign the result to Rx	Reg.	NOT Rx Ry
13.	STRREG	Store the value of Rx to the xx location in data memory.	Reg.	STRREG Rx xx
14.	STRIN	Store the value of Din bus to the xx location in data memory.	Reg.	STRIN xx
15.	SNDOUT	Send the value of data memory at xx location to Dout bus	Reg.	SNDOUT xx
16.	SNDREG	Send the value of data memory at xx location to Rx	Reg.	SNDREG Rx xx
17.	JMP	Jump to instruction at xx location in inst. memory	Reg.	JMP xx
18.	JC	Jump to xx if carry = 1	Reg.	JC xx
19.	JNC	Jump to xx if carry = 0	Reg.	JNC xx
20.	JSGN	Jump to xx if sign = 1	Reg.	JSGN xx
21.	JNSGN	Jump to xx if sign = 0	Reg.	JNSGN xx
22.	JZ	Jump to xx if zero = 1	Reg.	JZ xx
23.	JNZ	Jump to xx if zero = 0	Reg.	JNZ xx
24.	JVF	Jump to xx if overflow = 1	Reg.	JVF xx
25.	JNVF	Jump to xx if overflow = 0	Reg.	JNVF xx
26.	HLT	Stop execution	-	HLT
------	-------------------	Immediate Addressing Modes	---------	----------
2.	MOV	Move immediate value xx to Rx	Imm	MOV Rx xx
3.	ADD	Add immediate value xx to Ry assign the result to Rx	Imm	ADD Rx Ry xx
4.	SUB	Subtract immediate value xx from Ry and assign the result to Rx	Imm	SUB Rx Ry xx
5.	MUL	Multiply immediate value xx with Ry and assign the LSB 16 bits to Rx	Imm	MUL Rx Ry xx
6.	OR	Bitwise OR immediate value xx with Ry and assign the result to Rx	Imm	OR Rx Ry xx
7.	AND	Bitwise AND immediate value xx with Ry and assign the result to Rx	Imm	AND Rx Ry xx
8.	XOR	Bitwise XOR immediate value xx with Ry and assign the result to Rx	Imm	XOR Rx Ry xx
9.	XNOR	Bitwise XNOR immediate value xx with Ry and assign the result to Rx	Imm	XNOR Rx Ry xx
10.	NAND	Bitwise NAND immediate value xx with Ry and assign the result to Rx	Imm	NAND Rx Ry xx
11.	NOR	Bitwise NOR immediate value xx with Ry and assign the result to Rx	Imm	NOR Rx Ry xx
12.	NOT	Bitwise NOT immediate value xx and assign the result to Rx	Imm	NOT Rx xx

2.3 Opcode Table

Operation	Opcode
MOVSGPR	‘00000’
MOV	‘00001’
ADD	‘00010’
SUB	‘00011’
MUL	‘00100’
OR	‘00101’
AND	‘00110’
XOR	‘00111’
XNOR	‘01000’
NAND	‘01001’
NOR	‘01010’
NOT	‘01011’
STRREG	‘01100’
STRIN	‘01101’
SNDOUT	‘01110’
SNDREG	‘01111’
JMP	‘10000’
JC	‘10001’
JNC	‘10010’
JSGN	‘10011’
JNSGN	‘10100’
JZ	‘10101’
JNZ	‘10110’
JVF	‘10111’
JNVF	‘11000’
HLT	‘11001’

- Eg: [Instruction]: **ADD R1 R5 R14 (**Register Addressing Mode)

31 : 27	26 : 22	21 : 17	16	15 : 11	10 : 0
Opcode	Dest. Reg	Src. Reg 1	Mode	Src. Reg 2	-Unused-
‘00010’	00001	00101	0	01110	xxxxxxxxxxx

- Eg: [Instruction]: **XOR R4 R21 #16215 (**Immediate Addressing Mode)

31 : 27	26 : 22	21 : 17	16	15 : 0
Opcode	Dest. Reg	Src. Reg 1	Mode	Imm. value
‘00111’	00100	10101	1	0011111101010111

3. Assembly to Machine code compiler

- The *decode_compile.py* is a python program which converts the assembly language to machine code.
- All instructions need to be in uppercase.
- After all instructions are fed into the the program type “GENERATE” command followed by the file name to generate the program file (GENERATE *file_name*).
- Rename the file as ‘inst_mem.mem’ or the $readmemb() argument can be changed to the file name.
- Specify the full address of the file inside the $readmemb() function.

```
INST0 : MOV R1 #4267
00001000010000010001000010101011
INST1 : MOV R2 #298
00001000100000010000000100101010
INST2 : ADD R0 R1 R2
00010000000000100001000000000000
INST3 : HLT
11001000000000000000000000000000
INST4 : GENERATE inst_mem.mem
File Generated Successfully .....!
>>>
```

4. Input and output ports

```
module top(
    input clk,sys_rst,
    input [15:0] din,
    output reg [15:0] dout
);
```

- The clk port requires a clock signal for the synchronous state machine.
- The sys_rst port resets the processor and sets the PC to zero.
- Din bus is used to send data to the data memory.
- Dout is used to send data out of the processor module.