- **PDS1**

The aim is to use 32 registers each of which stores 32-bit words. The numbering of registers is identical to that of MIPS.

- **PDS2**

We are using 65536 of 8 byte (32 bit) memory elements in instruction memory to hold the instructions. Similarly, we use 65536 of 8 byte memory elements in data memory to store words.

- **PDS3**

Instruction encoding is similar to that of MIPS standard like-

a. All instructions are encoded in binary.
b. All instructions are 32 bits long.
c. Formats of R-, I- and J- type are same to that of MIPS as given below.

1. R-type Instruction

opcode	rs	rt	rd	shamt	funct
6 bits	5 bits	5 bits	5 bits	5 bits	6 bits

2. I-type Instruction

opcode	rs	rt	Constant / Address
6 bits	5 bits	5 bits	16 bits

3. J-type Instruction

opcode	Coded address of label
6 bits	26 bits

- **PDS4**

  At every posedge of clock, in the InstructMem module, the instruction corresponding to the program counter will be given to the output wire “Instr”.

- **PDS5**

  The following are the formats and opcodes for the instructions in this architecture. The instruction decode part is done by the control unit, which changes the ALU control signal according to how the instruction is encoded.

Instruction	Format	Opcode(decimal representation)	Funct(decimal representation)
ADD	R	0	32
SUB(subtract)	R	0	34
ADDU	R	0	33
SUBU (subtract unsigned)	R	0	35
AND	R	0	36
OR	R	0	37
SLL	R	0	0
SRL	R	0	2
SLT	R	0	42
jr (jump register)	R	0	8
ADDI(add immediate)	I	8	N.A.
ADDIU	I	9	N.A.
lw (load word)	I	35	N.A.
sw (store word)	I	43	N.A.

j (jump)	J	2	N.A.
jal (jump and link)	J	3	N.A.

- **PDS6**

  The ALU takes two inputs and assigns the output to the output wire after performing the operation according to the ALU Control signal generated by the control unit. The following are the ALU Control signals.

Operation	ALUctrl
ADD	0
SUB	1
AND	2
OR	3
SLL	4
SRL	5
SLT	6
BEQ	7
BNE	8
BGT	9
BGTE	10
BLE	11
BLEQ	12

- **PDS7 and PDS8**

  This is the MIPS code for bubble sort

```
add $s0 , $a0, $0  #address of array in $s0
add $s1 , $a1 , $0 #size of array in $s0

addi $s2 , $s1 , -1  #i=n-1
```

```
Loop1:  beq $s2 , $0 , exit1   #exit1 when i=0
add $s3 , $0 , $0   #j=0
Loop2:  beq $s3 , $s2 , exit2   #exit2 when j=i
add $t0,$s0,$s3     #t0 = a+j
lw $s4 , 0($t0)     #s4 = a[j]
lw $s5 , 1($t0)     #s5 = a[j+1]
ble $s4 , $s5 , afterswap
sw $s4, 1($t0)
sw $s5 , 0($t0)
afterswap:
addi $s3 , $s3 , 1  #j=j+1
j Loop2
exit2:
addi $s2, $s2 , -1  #i = i-1
j Loop1     #
exit1:
beq $0 , $0 , 0
```

These instructions were encoded in the format of this architecture and loaded into the Instruction memory and this was simulated using the test bench.