KGP-RISC PROCESSOR

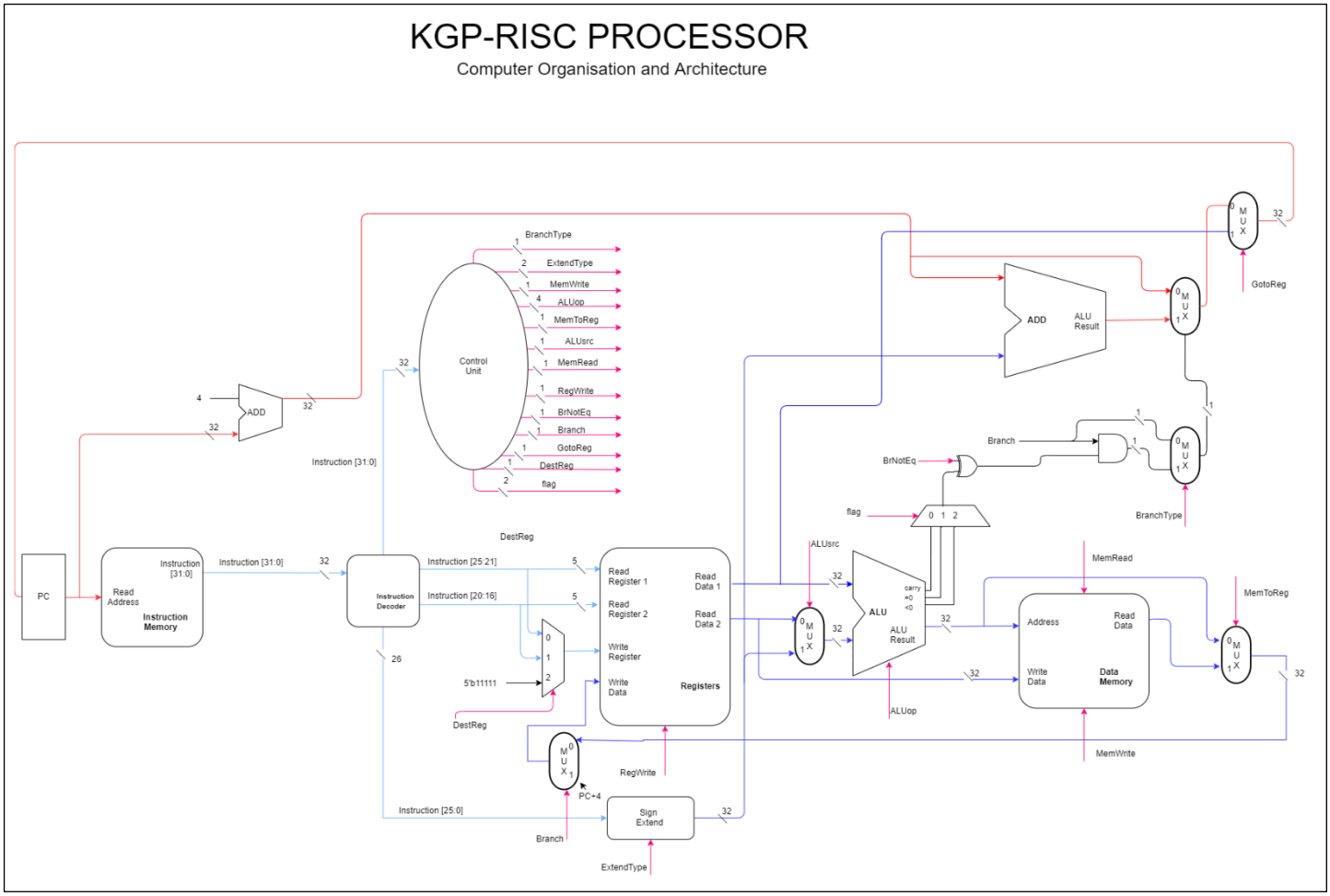
INSTRUCTION SET ARCHITECTURE

Computer Organisation and Architecture Laboratory
Autumn-2021

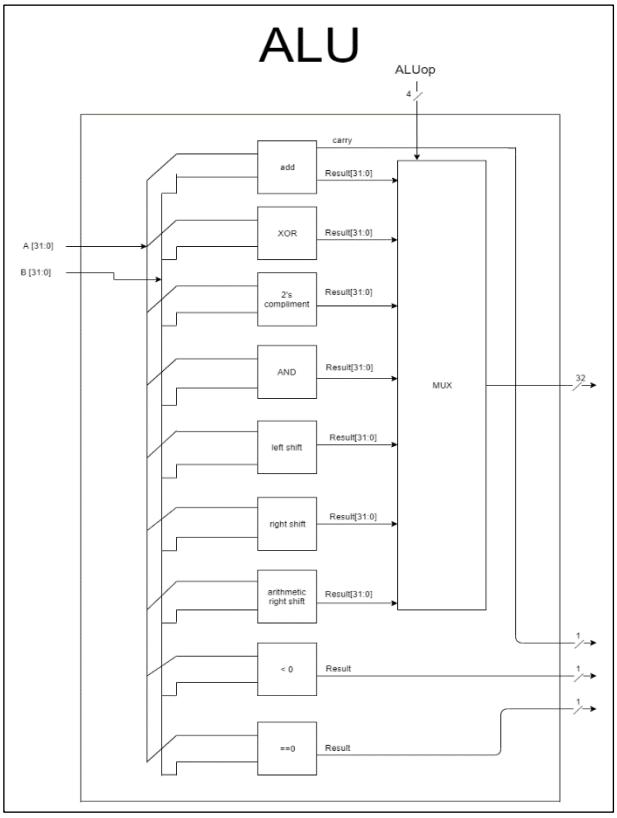
GROUP: 59
Ramtej Doma (19CS10028)
Vamshi Lavoori (19CS30026)

Contents:

1. Architecture Diagram



Please refer **Architecture.png** and **ALU.png** for better visibility



2. Instruction Set and Format

Type 1: Register-type instructions

Instruction	usage	Action	Opcode [31:26]	Register 1 [25:21]	Register 2 [20:16]	Shift Amount [15:10]	Function Code [9:0]
Class: Arithmetic							
Addition	**add rs,rt**	rs ← (rs)+(rt)	000000	rs	rt	N.A	0000000000
Compliment	**comp rs, rt**	rs←2'scomplement (rt)	000000	rs	rt	N.A	0000000001
Class: Logic							
AND	**and rs, rt**	rs ← (rs) & (rt)	000000	rs	rt	N.A	0000000010
XOR	**xor rs, rt**	rs ← (rs) ^ (rt)	000000	rs	rt	N.A	0000000011
Class: Shift							
Shift left logical	**shll rs, sh**	rs ← left-shifted by sh	000000	rs	N.A	sh	0000000100
Shift right logical	**shrl rs, sh**	rs ← right shifted by sh	000000	rs	N.A	sh	0000000101
Shift left logical variable	**shllv rs, rt**	rs ← (rs) left-shifted by (rt)	000000	rs	rt	N.A	0000000110
Shift right logical variable	**shrlv rs, rt**	rs ← (rs) right-shifted by (rt)	000000	rs	rt	N.A	000000111
Shift right arithmetic	**shra rs, sh**	rs ←(rs) arithmetic right-shifted by sh	000000	rs	N.A	sh	0000001000
Shift right Arithmetic variable	**shrav rs, sh**	rs ←(rs) arithmetic right-shifted by (rt)	000000	rs	rt	N.A	0000001001

No of instructions of this type: 10
No of instruction can be added further: (max) 65,524, (min up to) 1014

Type2: Immediate-Type Instructions

Instruction	usage	Action	Opcode [31:26]	Register 1 [25:21]	Register 2 [20:16]	imm [15:0]
Class: Memory						
Load Word	**lw rt, imm(rs)**	rt ← [mem[(rs) + imm]	000001	rs	rt	Immediate value
Store Word	**sw rt, imm(rs)**	mem[(rs) + imm] ← (rt)	000010	rs	rt	Immediate value

Class: Arithmetic						
Addition immediate	**addi rs, imm**	rs ← (rs) + imm	000011	rs	N.A	Immediate value
Compliment immediate	**compi rs, imm**	rs ← 2'sCompliment(imm)	000100	rs	N.A	Immediate value

No of instructions of this type: 4
No of instruction can be added further: max up to 51
Immediate value range: [-32768, 32767]

Type3: Branch_1

Instruction	**usage**	**Action**	**Opcode [31:26]**	**Label Address [25:0]**
Class: Branch				
Unconditional Branch	**b L**	goto L	000101	L
Branch and Link	**bl L**	goto L; (31) ← (PC)+4	000110	L
Branch on Carry	**bcy L**	goto L if Carry = 1	000111	L
Branch on No Carry	**bncy L**	goto L if Carry = 0	001000	L

No of instructions used of this type: 4
No of instruction can be added further: max up to 51
label address offset range: [-33554432, 33554431]

Type 4: Branch_2

Instruction	**usage**	**Action**	**Opcode [31:26]**	**Register 1 [25:21]**	**Label Address [20:0]**
Class: Branch					
Branch register	**br rs**	goto (rs)	001001	rs	N.A
Branch on less than zero	**bltz rs, L**	if(rs) < 0 then goto L	001010	rs	L
Branch on flag zero	**bz rs, L**	if(rs) = 0 then goto L	001011	rs	L
Branch on flag not zero	**bnz rs, L**	if(rs) != 0 then goto L	001100	rs	L

No of instructions used of this type: 4
No of instruction can be added further: max up to 51
label address offset range: [-1048576, 1048575]

Upon further modelling, no of instructions that can be further added increase at the cost of max value of L (label) address jump value

3. Control Signals and Data path

Instruction	DestReg	ExtendType	MemWrite	ALUOp	MemToReg	ALUsrc	MemRead	Branch	RegWrite	Flag	GoToReg	BranchType	BrNotEq
add	0	x	0	0000	0	0	0	0	1	xx	0	x	x
comp	0	x	0	0001	0	0	0	0	1	xx	0	x	x
XOR	0	x	0	0010	0	0	0	0	1	xx	0	x	x
AND	0	x	0	0011	0	0	0	0	1	xx	0	x	x
shllv	0	x	0	0100	0	0	0	0	1	xx	0	x	x
shrlv	0	x	0	0101	0	0	0	0	1	xx	0	x	x
shrav	0	x	0	0110	0	0	0	0	1	xx	0	x	x

Data Path:

fetch instruction
read register’ s data1, data2
control unit outputs the respective ALUop, control signals
input register data1, data2 to ALU
ALU performs desired operation
write ALUresult in desired register

Instruction	DestReg	ExtendType	MemWrite	ALUOp	MemToReg	ALUsrc	MemRead	Branch	RegWrite	Flag	GoToReg	BranchType	BrNotEq
shll	00	xx	0	0111	0	1	0	0	1	xx	0	x	x
shrl	00	xx	0	1000	0	1	0	0	1	xx	0	x	x
shra	00	xx	0	1001	0	1	0	0	1	x	0	x	x

Data Path:

fetch instruction
read register data1
control unit outputs the respective ALUop and control signals
input shift amount to ALU
ALU performs desired shift operation
write ALUresult in desired register

Instruction	DestReg	ExtendType	MemWrite	ALUOp	MemToReg	ALUsrc	MemRead	Branch	RegWrite	Flag	GoToReg	BranchType	BrNotEq
addi	00	00	0	0000	0	1	0	0	1	xx	0	x	x
compi	00	00	0	0001	0	1	0	0	1	xx	0	x	x

Data Path:

fetch instruction
read register data 1
control unit output the respective ALUop and control signal
input imm value to Sign Extend module
input 32bit extended value to ALUsrc mux (control signal = 1)
input register1 data and sign extended imm value (32bit) to ALU
ALU performs desired operation
write ALUresult in desired register

Instruction	DestReg	ExtendType	MemWrite	ALUOp	MemToReg	ALUsrc	MemRead	Branch	RegWrite	Flag	GoToReg	BranchType	BrNotEq
sw	xx	00	1	0000	0	1	0	0	0	xx	0	x	x
lw	01	00	0	0000	1	1	1	0	1	xx	0	x	x

Data Path:

sw	lw
read instruction read register data 1 control unit outputs the respective ALUop and control signal input imm value to Sign Extend module input 32bit extended value to ALUsrc mux (control signal =1) Input register1 data and sign extended imm value (32bit) to ALU ALU performs addition pass ALUresult to Data Memory module as address input register data 2 to Data Memory block as write data write input data at input address (Data Memory block)	read instruction read register data 1 control unit output the respective ALUop and control signal input imm value to Sign Extend module input 32bit extended value to ALUsrc mux (control signal =1) input register1 data and sign extended imm value (32bit) to ALU ALU performs addition ALUresult is passed to Data Memory module as address Data Memory output (data at memory location [ALUresult]) passed to MemToReg mux (control signal set to 1) (set DestReg control signal to 01) Write result in desired register

---

Instruction	DestReg	ExtendType	MemWrite	ALUOp	MemToReg	ALUsrc	MemRead	Branch	RegWrite	Flag	GotoReg	Branchtype	BrNotEq
b L	xx	10	0	xxxx	0	0	0	1	0	xx	0	0	0
bl L	10	10	0	xxxx	0	0	0	1	0	xx	0	0	0
bcy L	xx	10	0	xxxx	0	0	0	1	0	00	0	1	0
bncy L	xx	10	0	xxxx	0	0	0	1	0	00	0	1	1
br rs	xx	xx	0	xxxx	0	0	0	1	0	xx	1	1	0
bltz rs, L	xx	01	0	1010	0	0	0	1	0	10	0	1	0
bz rs, L	xx	01	0	1011	0	0	0	1	0	01	0	1	0
bnz rs, L	xx	01	0	1011	0	0	0	1	0	01	0	1	1

Data Path:
fetch instruction
control unit output respective control signals
input L (address) to sign extend module
input sign extended label offset to ALU ADD
ALU performs addition
input ALUresult to branch condition mux
perform Branch logic and decide next value of pc b/w label address and pc+4
next PC → output of branch condition mux

Data Path: Goto Register Type
fetch instruction
control unit output respective control signals
input L (label address) to Sign Extend Module
signextended address stored in register 1 is passed to GoToReg decider MUX
update PC with value of register 1

4. Modules:

PC: Increments the next program counter to either the jump value (L) or adds 4 to the current program counter. Works synchronously with clock.

Instruction Memory: Loads the instructions at the current program counter from the BROM instantiated for instructions only. 32-bit addressing is used.

Instruction Decoder: Segments the instruction into interpretable register addresses, sign extend values, immediate values, operation code and function code.

Registers: Stores 32 registers each of 32 bits. Outputs the values of registers read from instructions and writes in values of input registers on write enable signal(on posedge with certain delay) .

Register Numbering and conventions:

Register Number	Alternative Name
0	zero
1	$at
2-3	$v0 - $v1
4-7	$a0 - $a3
8-15	$t0 - $t7
16-23	$s0 - $s7
24-25	$t8 - $t9
26-27	$k0 - $k1
28	$gp
29	$sp
30	$s8/$fp
31	$ra

Control Unit: Depending on the operation it activates various modules/ hardware with different functionalities. Flags and controls discussed later.

Sign Extend: Depending on the operation, it sign-extends the segment of input data (i.e., 16-bit/ 21-bit/ 26-bit to 32-bit)

> **Sign Extend Type:** 00
> Immediate value (16 bits) is passed to Sign Extend module, desired extension is performed and output is given to ALUsrc mux as input to choose appropriate input data for ALU
> The opcode 00 is dedicated for this type of sign extension
>
> **Sign Extend Type:** 10
> Branch jump distance (26 bits) is passed to Sign Extend Block module, desired extension is performed and output is given to PC ALU as input to choose next appropriate instruction address
> The opcode 10 is dedicated for this type of sign extension
>
> **Sign Extend Type:** 01
> Branch Jump offset (21 bits) is passed to Sign Extend Block module, desired extension is performed and output is given to PC ALU as input to choose next appropriate instruction address
> The opcode 01 is dedicated for this type of sign extension

Input Decider MUX: Decides between sign extended and register values depending on the instruction type.

Data Memory: BRAM module instantiated for data memory. 32-bit addressing is used.

Branch Logic: Depending on the type of the branch instructions, returns the jump value(direct address of the next instruction).

ALU: Main arithmetic and logical unit of the processor, performs operations like addition, 2’s compliment, logical shift , arithmetic shift and controls carry, sign and zero flag signals

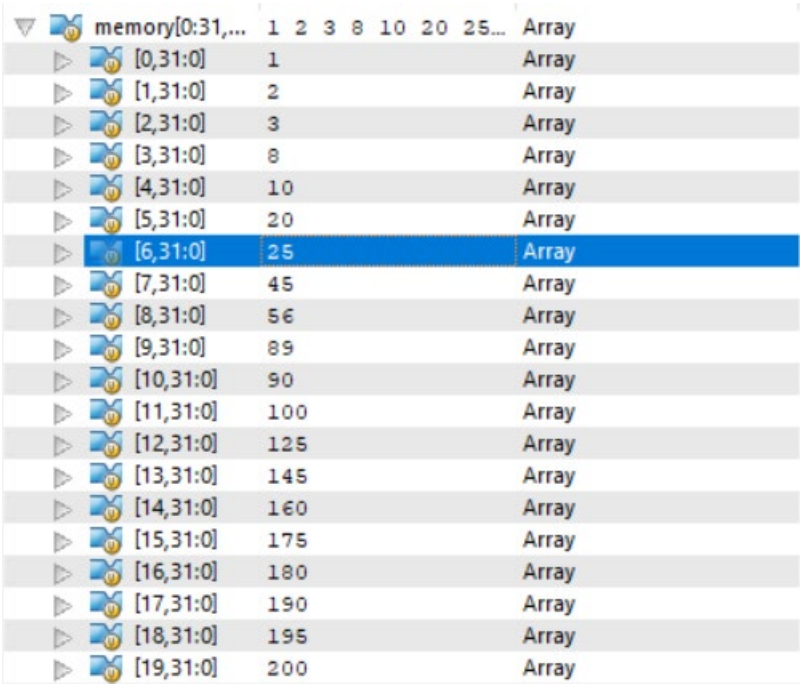
operation	ALU opcode
Addition	0000
Compliment	0001
And	0010
Xor	0011
Shift left logical	0111
Shift right logical	1000
Shift left logical value	0100
Shift right logical value	0101
Shift right arithmetic	1001
Shift right arithmetic value	0110
Output sign flag	1010
Output zero flag	1011

4. Controls/Flags:

1. branchType → decide b/w conditional and unconditional branching
2. memWrite → select line to data memory
3. memToReg → select b/w data memory output and ALU result to write in register
4. memRead → data memory read enable control
5. regwrite → write on register enable flag
6. brNotEq → control flag for branch not equal instructions
7. branch → branch instruction control flag
8. goToReg → jump to address stored in register control
9. flag → flag for conditional branching type
10. extendType → sign extend type control for three type of immediate and label address values
11. destReg → to decide destination register to write
12. ALUop → ALU operation type
13. ALUsrc → to decide second input for ALU

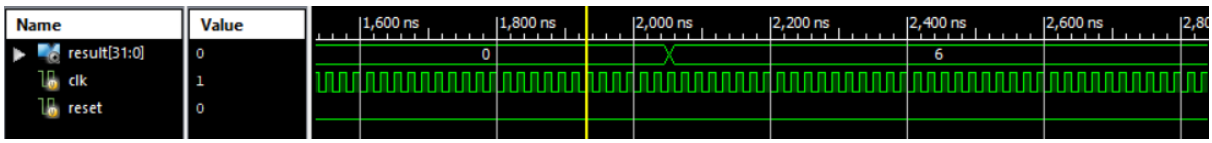
5. Testing:

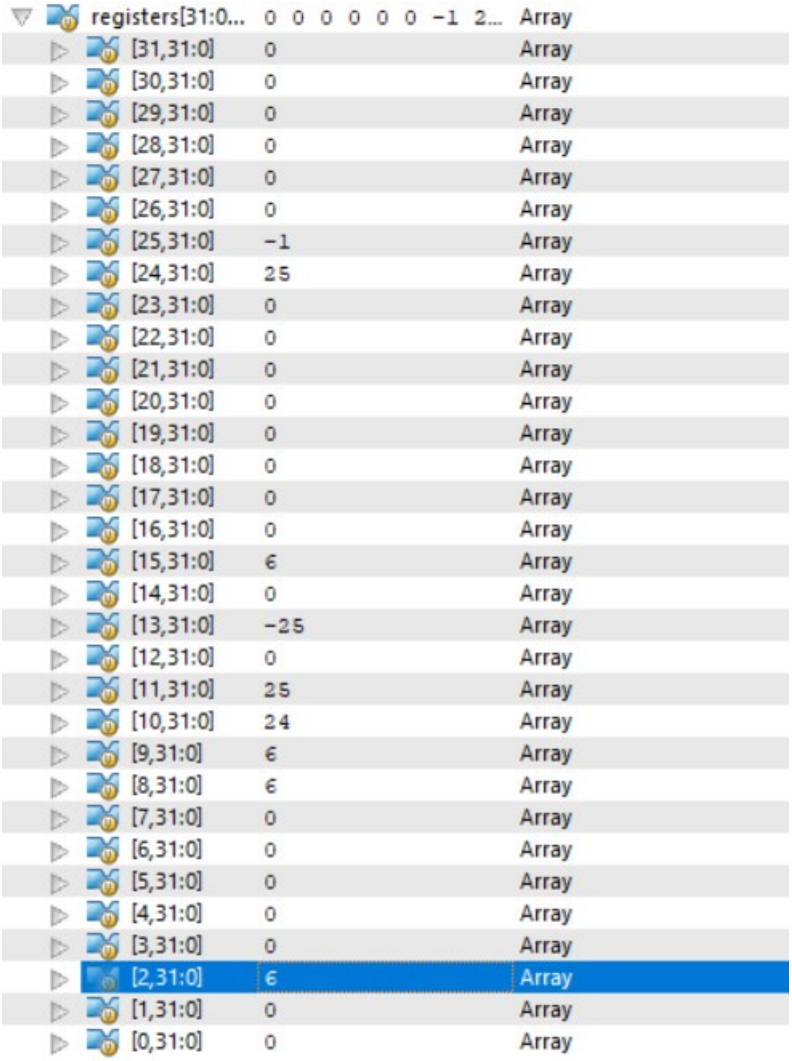
Binary Search: 20 elements (sorted) are loaded into the data memory using Bnary_Search_Data.coe file



Key = 25 is searched in the array of twenty elements and the result obtained is stored in $v0(also referred as $2) register (output as result in simulation)

e



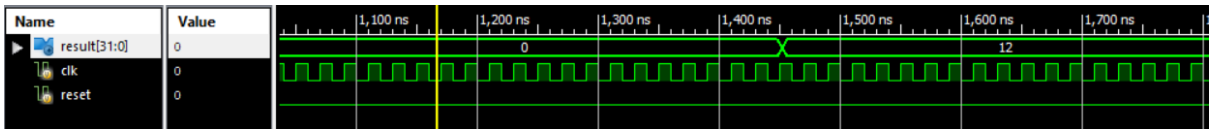


GCD:

Greatest common divisor of two numbers a=36 and b= 156 is computed and the result is stored in $v0 register

Values of a and b are loaded into the data memory using GCD_Data.coe file.

Values of a, b are loaded in registers $t0, $t1 from data memory for computation



GCD of a(36) and (156) is stored in register $2 ($v0)

registers[31:0...	0 0 0 0 0 0 0 0 ...	Array
[31,31:0]	0	Array
[30,31:0]	0	Array
[29,31:0]	0	Array
[28,31:0]	0	Array
[27,31:0]	0	Array
[26,31:0]	0	Array
[25,31:0]	0	Array
[24,31:0]	0	Array
[23,31:0]	0	Array
[22,31:0]	0	Array
[21,31:0]	0	Array
[20,31:0]	0	Array
[19,31:0]	0	Array
[18,31:0]	0	Array
[17,31:0]	0	Array
[16,31:0]	0	Array
[15,31:0]	0	Array
[14,31:0]	0	Array
[13,31:0]	0	Array
[12,31:0]	-12	Array
[11,31:0]	-12	Array
[10,31:0]	0	Array
[9,31:0]	0	Array
[8,31:0]	12	Array
[7,31:0]	0	Array
[6,31:0]	0	Array
[5,31:0]	0	Array
[4,31:0]	0	Array
[3,31:0]	0	Array
[2,31:0]	12	Array
[1,31:0]	0	Array
[0,31:0]	0	Array

Assumptions and Other Points to be Noted:

- Block RAMs have a peculiar issue that they have significant delay in fetching data from the RAM. Hence, we have introduced two slower clocks (w.r.t primary clock). These slower clocks are used to fetch and write data on Block RAM. The primary clock is used for other modules.
- 32-bit addressing is used in Block RAMs. Hence, pc is updated as pc+4 for every clock cycle.
- For better comprehensibility of simulation and working of the design, the top-level module (Processor.v) outputs the $v0 register which stores the result of input program(GCD of two numbers, index of key in binary search)

6. How to RUN:

1) Import all the Verilog files into a new project
2) Load Instructions and Data Memory
   a) Binary Search
   initialize the Instruction Memory with BinarySearch_Instruction.coe file
   initialize the Data Memory with BinarySearch_Data.coe file
   b) GCD
   initialize the Instruction Memory with GCD_Instruction.coe file
   initialize the Data Memory with GCD_Data.coe file

3) Make the simulation run time sufficiently large(default = 1000ns) for the result to be computed, under the simulation properties make it 3000ns
   Run the Processor_tb.v (Simulate Behavioural Model)
   Incase there is an error, please load the .coe files properly and check whether the instruction and data are loaded properly

Note: No higher-level module is implemented for Instruction Memory and Data Memory. They are instantiated directly in Processor.v module. Also 32-addressing is used for both.

The Input Programs(instructions) are written in a way, so that they will self-loop after the computation of final result and the result(once computed) remains same till the end of simulation.