KGP RISC DOCUMENTATION

Group - 18

Kumar Abhishek - 17CS10022

Gurjot Singh Suri - 17CS10058

Register Usage Conventions

Register	Function	Register Number	Register Code
$zero	Stores the constant 0	0	00000
$st	Stores various flags	1	00001
$v0-$v1	Return values	2-3	00010 - 00011
$a0-$a3	Function parameters	4-7	00100 - 00111
$t0-$t10	Temporaries	8-18	01000 - 10010
hi	Stores most significant word of product	19	10011
lo	Stores least significant word of product	20	10100
$s0-$s8	Saved variables	21-29	10101 - 11101
$sp	Stack pointer	30	11110
$ra	Return address	31	11111

Instruction Format And Encoding

Opcodes

Arithmetic	add	00	0000
	multu		0001
	mult		0010
	comp		0011
	addi		0100
	compi		0101
Logic	and	00	0110
	xor		0111
Shift	shll	00	1000
	shrl		1001
	shllv		1010
	shrlv		1011
	shra		1100
	shrav		1101
Memory	lw	01	0000
	sw	10	0000
Branch	b	11	0000
	br		0001
	bz		0010
	bnz		0011

	bcy		0100
	bncy		0101
	bs		0110
	bns		0111
	bv		1000
	bnv		1001
	Call		1010
	Ret		1011

INSTRUCTION FORMATS

Format 1

Functions
add,multu,mult,comp,and,xor,shll,shrl,shllv,shrlv,shra,shrav,lw,sw

opcode	**rs**	**rt/sh**	**Don't care/offset**
6 bits	5 bits	5 bits	16 bits

Format 2

Functions
addi, compi

opcode	**rs**	**immediate**
6 bits	5 bits	21 bits

Format 3

Functions
br

opcode	**rs**	**Don’t care**
6 bits	5 bits	21 bits

Format 4

\

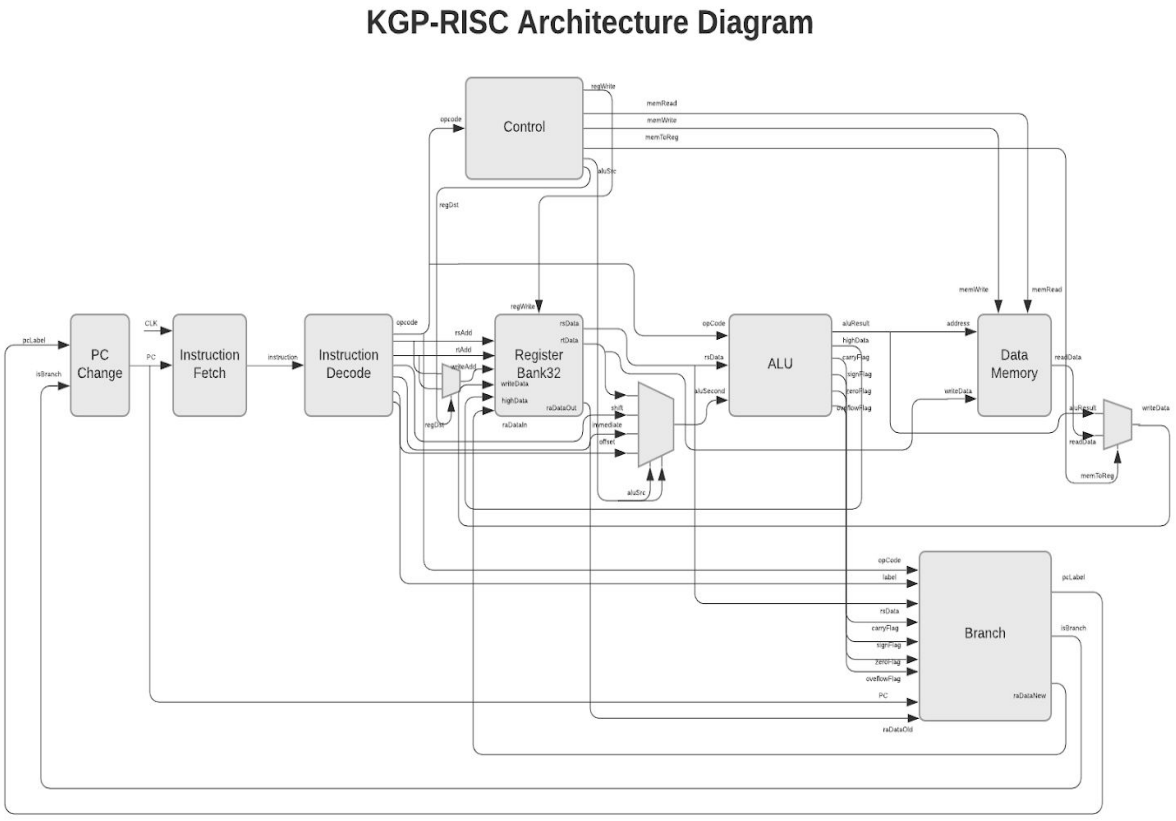
Functions
b,bz,bnz,bcy,bncy,bs,bns,bv,bnv,Call

opcode	**label**
6 bits	26 bits

Format 5

Functions
Ret

opcode	**Don’t care**
6 bits	26 bits



Control Signals

RegWrite: is set to 1 for writing to writeAddress, 2 for writing to registers high & low, else no write.
MemRead: is set to 1 for reading from memory.
MemWrite: is set to 1 for writing to memory.
MemToReg: is set to 1 for writing in register from memory.
AluSrc: is set to 0 when second ALU operand is rtData, 1 for shift, 2 for immediate, 3 for offset.
RegDst: is set to 1 when write register is rt and 0 when write register is rs.

Flags

CarryFlag: is set to 1 when carry is 1, else 0.
SignFlag: is set to 1 when sign is negative, else 0.
OverflowFlag: is set to 1 when there is overflow, else 0.
ZFlag: is set to 1 when result is 0, else 0.

Modules and their purposes

Instruction Fetch:
Used to fetch the instruction from BRAM using the program counter.

Instruction Decode:
Decodes the instruction into opcode, rsAdd, rtAdd, shift, immediate, offset, label.

PC Change:
Uses current PC, isBranch and pcLabel to generate the new PC.

Register Bank:
Register file, used to write and read from registers. Allows two registers to be read at a time.

Control:
Takes opcode and generates regWrite, memRead, memWrite, memToReg, aluSrc, regDst.

Branch:
Uses the ALU flags, opcode, label, PC, raDataOld to assign pcLabel, isBranch and raDataNew.

ALU:
Based on opcode and inputs, computes the aluResult, highData and flags.

Data Memory:
Based on address and writeData along with control flags reads and writes data from memory.

Running the Linear Search Algorithm

```
        0. add $t0, $zero # &Mem[0]
        1. add $t1, $zero # i=0
        2. addi $s0, 8 # n=8
        3. addi $s1, 4 # num to be searched
        4. comp $s2, $s0 # -n=-8
        5. comp $s3, $s1 # -num=-4
Loop:   6. xor $t2, $t2 # t2=0
        7. add $t2, $t1 # t2=i
        8. add $t2, $s2 # t2=i-n
        9. bz Exit # if t2=0, goto Exit
        10. lw $t3, 0($t0) # t3=Mem[i]
        11. add $t3, $s3 # t3=Mem[i]-num
        12. bz Exit
        13. addi $t1, 1 # i++
        14. addi $t0, 4 # &arr[i+1]
        15. b Loop
Exit:   16. Call Func
Func:   17. Ret
```

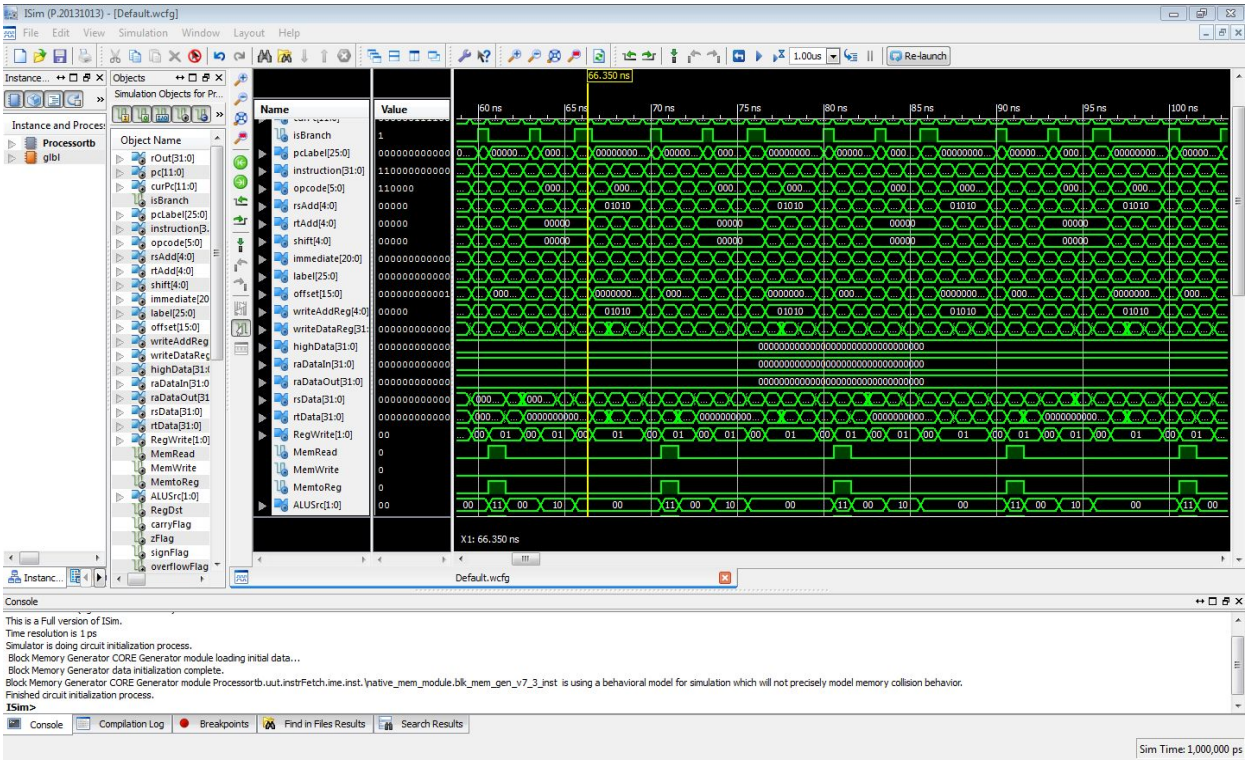
```
00000001000000000000000000000000,
00000001001000000000000000000000,
00010010101000000000000000001000,
00010010110000000000000000000100,
00001110111101010000000000000000,
00001111000101100000000000000000,
00011101010010100000000000000000,
00000001010010010000000000000000,
00000001010101110000000000000000,
11001000000000000000000001000000,
01000001000010110000000000000000,
00000001011110000000000000000000,
11001000000000000000000001000000,
00010001001000000000000000000001,
00010001000000000000000000000100,
11000000000000000000000000011000,
11101000000000000000000001000100,
11101100000000000000000000000000;
```

The index of the element 4 is to be searched, if the element is found register $t1 which is $9 stores the zero based index of element 4 in the memory array. If the element is not found $t1 stores 8 which is the length of the array.

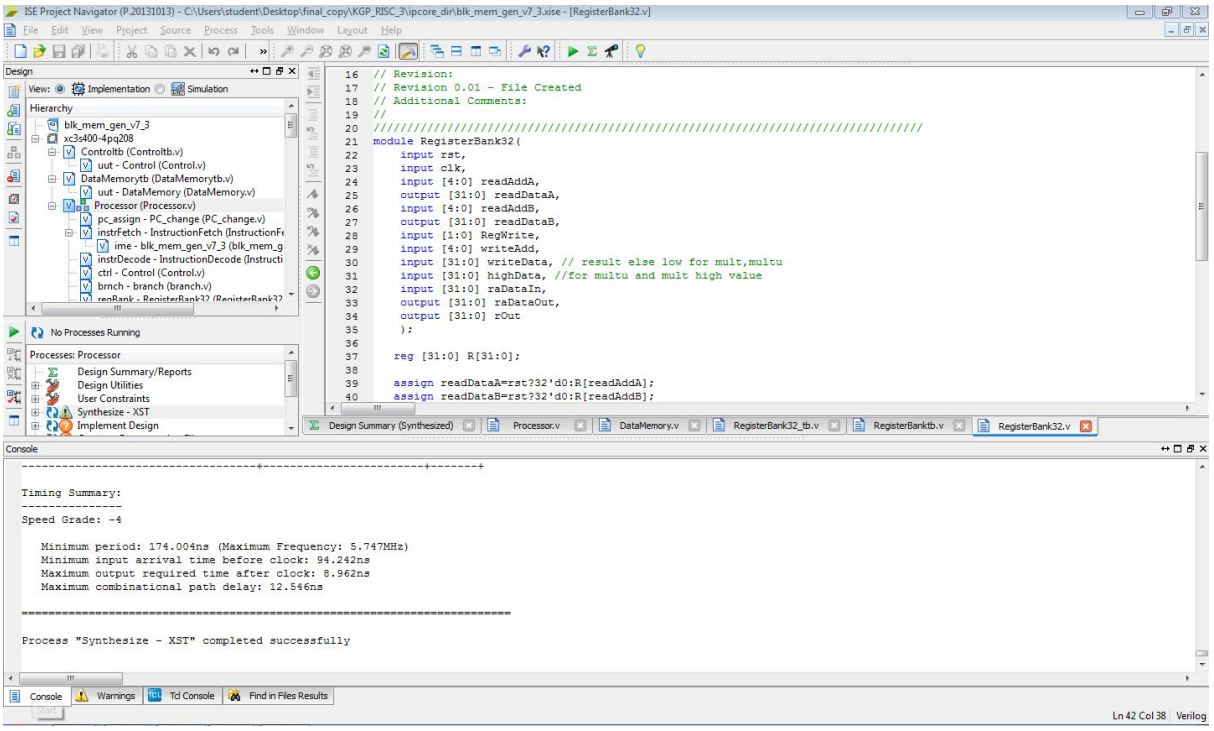
The corresponding Machine Language Instruction sequence is generated according to the encoding of each instruction in our language specification with appropriate values assigned to the opcode, rs, rt, shift, offset, label and immediate.

After the completion of linear search and storage of $t1 we come to the last instruction which is a ret instruction and it loops on itself so that any number of clock cycles can be given with no effect.

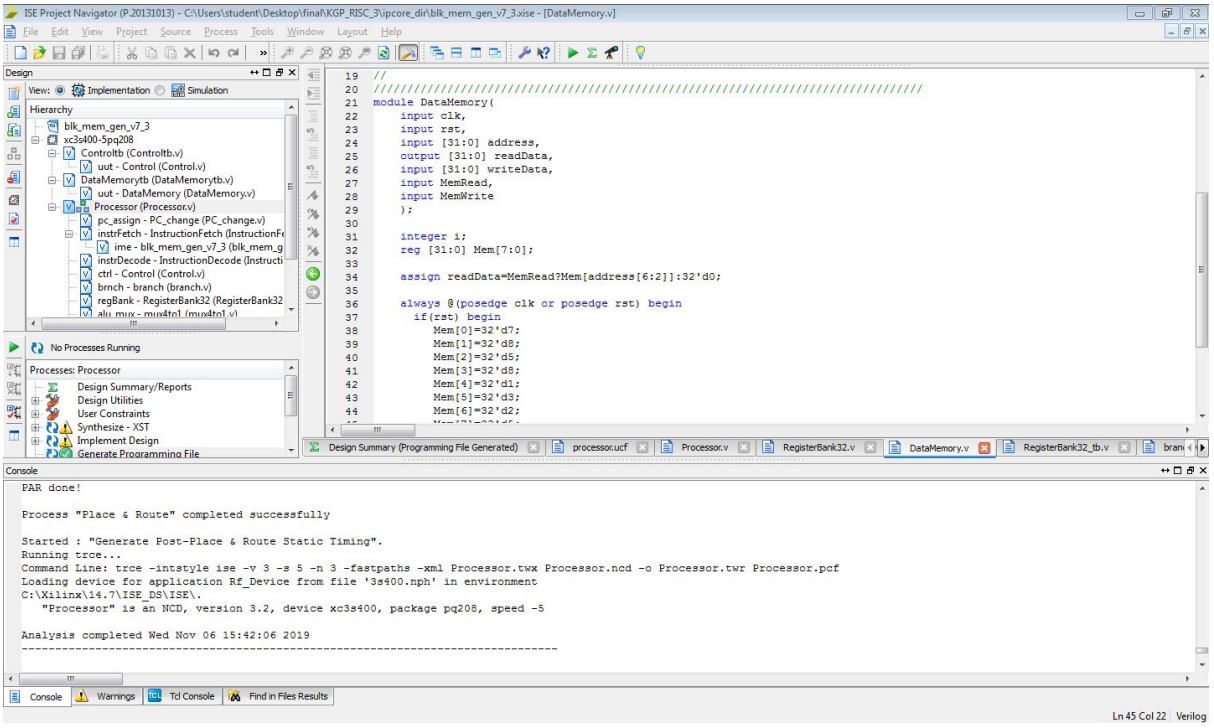
Simulation Screenshot:

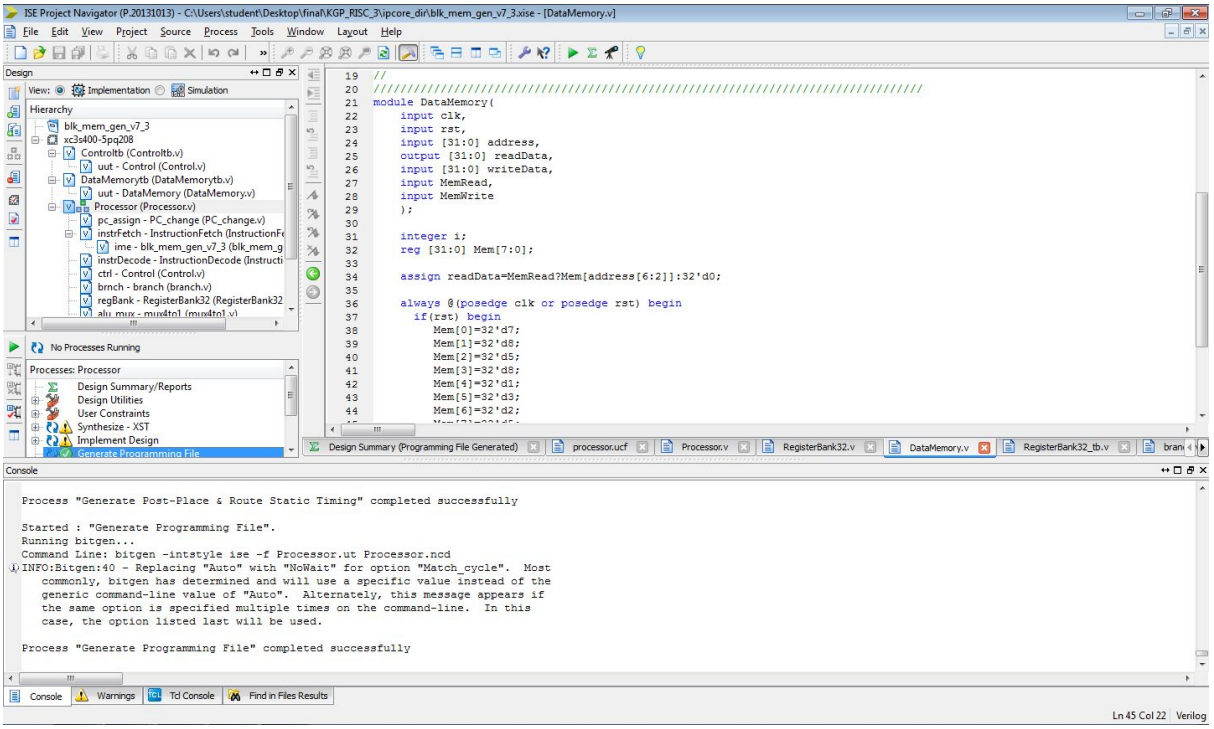


Behavioural Synthesis Screenshot:



Post Route Screenshot:





Thus, the **KGP-RISC processor** has been **successfully simulated**.