KGP -RISC

Our processor KGP Risc has the above Instruction Set Architecture (ISA). Assume that the processor has a 32 bit word, with all the registers and memory elements having 32 bit data. The address line of the memory is also 32 bits.

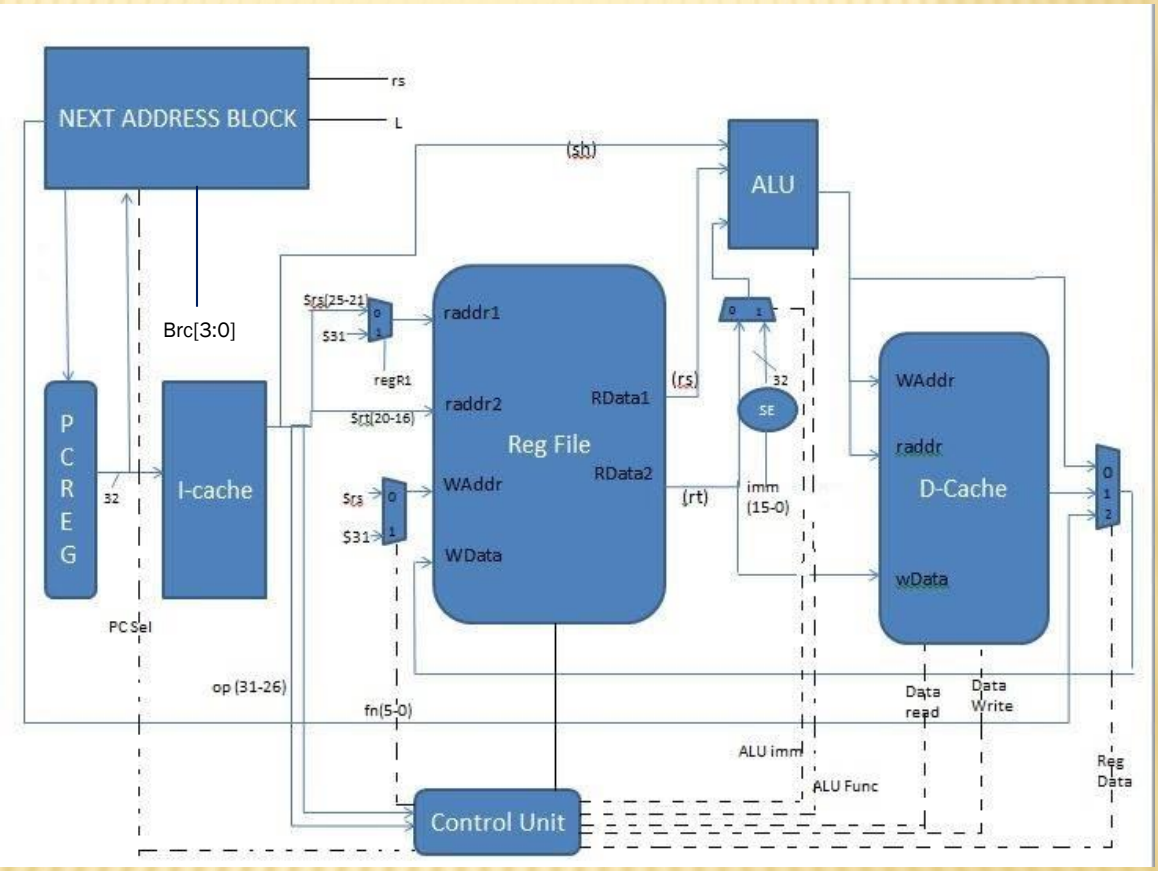
We are to develop first the op-code format for the above instruction set, identify the data path element(s), and design the data path along with the necessary control signals. Subsequently, we shall develop a single-cycle instruction execution unit for KGP-RISC.

11CS10024 : KRISHNENDU SAHA
11CS10045 : SRAJAN GARG

KGP-RISC ISA

1.1 PREFACE

The Processor KGP – RISC has a 32 bit word , with all the registers and memory having 32 bit data . The address line of the memory is 32 bits . We have encoded the instructions in such a format that we have possibility to add more instructions as well . The processor is designed for single cycle execution developed on Xilinx ISE 14.2.



1.2 INSTRUCTION FORMAT

We are using three types of instructions :

1.2.1 R- Type : Uses values from rs and rt registers for calculation

6 bit opcode	5 bit rs	5 bit rt	unused	5 bit shamt	6 bit fn

Here the first 6 bits denote the opcode followed by 5 bits for rs, five bits for rt and 5 bits for shift amount. The next five bits are not used. They can be brought in use if the ISA is changed later. The last 6 bits are for function class. Since we are using 32 general purpose registers and so 5 bits are required to represent them we assign 5 bits each to rs and rt. Also since the width for register and data in data cache is also 32 bits the shift amount can also be represented by 5bits. For the ALU type R operations the op-code is zero and the further decision of the instruction is made by the 6 bits in the function class.

1.2.2 I-Type : Uses Immediate value from the instruction for calculation.

6 bit opcode	5 bit rs	5 bit rt	16 bit immediate value

Here the first 6 bits are opcode that decide the operation to be performed. The next 5 bits are of rs, which is one of the operands and the destination as well. This is followed by 5 bits of rt which is the destination register for Load and Store type operations. The last 16 bits are reserved for immediate value. This value is sign extended to 32 bits before performing any operation.

1.2.3 B-Type : These instructions check for branching conditions.

6 bit opcode	26 bit destination address

The first 6 bits of the B-Type instructions decide the type of branch and the later 26 bits store the destination address for jump. We should ideally copy the 6 MSB's of the PC to this value as done in the pseudo-direct addressing but since anyway we are using 13 bits for address, to make it compatible with our current system we append its 6 MSB's with 0 to make it 32 bits.

1.3 ENCODING

The following table shows the opcodes and function class for the instructions supported by our processor.

Type	Instruction	Symbolic Instruction	Opcode decimal	Opcode	Function decimal	Fn code
R	Add	add	0	000000	2	000010
R	Comp	comp	0	000000	3	000011
I	Add Immediate	addi	1	000001		XXXXXX
I	Complement Immediate	compi	2	000010		XXXXXX
R	AND	And	0	000000	4	000100
R	XOR	Xor	0	000000	5	000101
R	Shift Left Logical	shll	0	000000	0	000000
R	Shift Right Logical	shrl	0	000000	8	001000
R	Shift Left Logical Variable	shllv	0	000000	1	000001
R	Shift Right Logical Variable	shrlv	0	000000	9	001001
R	Shift Right Arithmetic	shra	0	000000	24	011000
R	Shift Right Arithmetic Variable	shrav	0	000000	25	011001
I	Load Word	lw	9	001001		XXXXXX
I	Store Word	sw	18	010010		XXXXXX
B	Unconditional branch	b	26	011010		XXXXXX
R	Branch Register	br	0	000000	7	000111
B	Branch on zero	bz	6	000110		XXXXXX
B	Branch on not zero	bnz	14	001110		XXXXXX

Type	Instruction	Symbolic Instruction	Opcode decimal	Opcode	Function decimal	Fn code
B	Branch on Carry	bcy	22	010110		XXXXXX
B	Branch on No Carry	bncy	30	011110		XXXXXX
B	Branch on Sign	bs	38	100110		XXXXXX
B	Branch on Not Sign	bns	46	101110		XXXXXX
B	Branch on Overflow	bv	54	110110		XXXXXX
B	Branch on No Overflow	bnv	62	111110		XXXXXX
B	Call	Call	51	110011		XXXXXX
R	Return	Ret	0	000000	6	000110

1.4 FETCH OPERATION :

The communication between CPU and instruction memory , I-cache is done using PC . PC (Programme counter) contains the address of next instruction to be fetched . We do not need to store the instruction in a register as that Instruction is required for one -cycle. .The RTL operations and corresponding signals are as follows –

RTL : IR <-Icache[PC], PC <-Next_PC

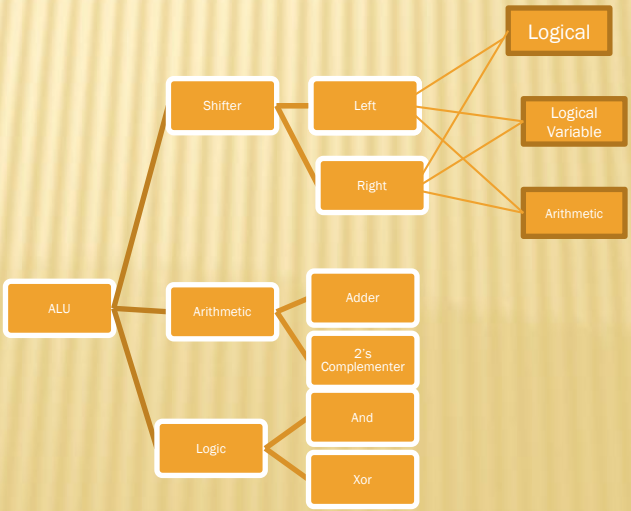
We have not particularly used control signal for fetch operation as on every posedge clock cycle we fetch an instruction from I-cache and send the instruction for processing.

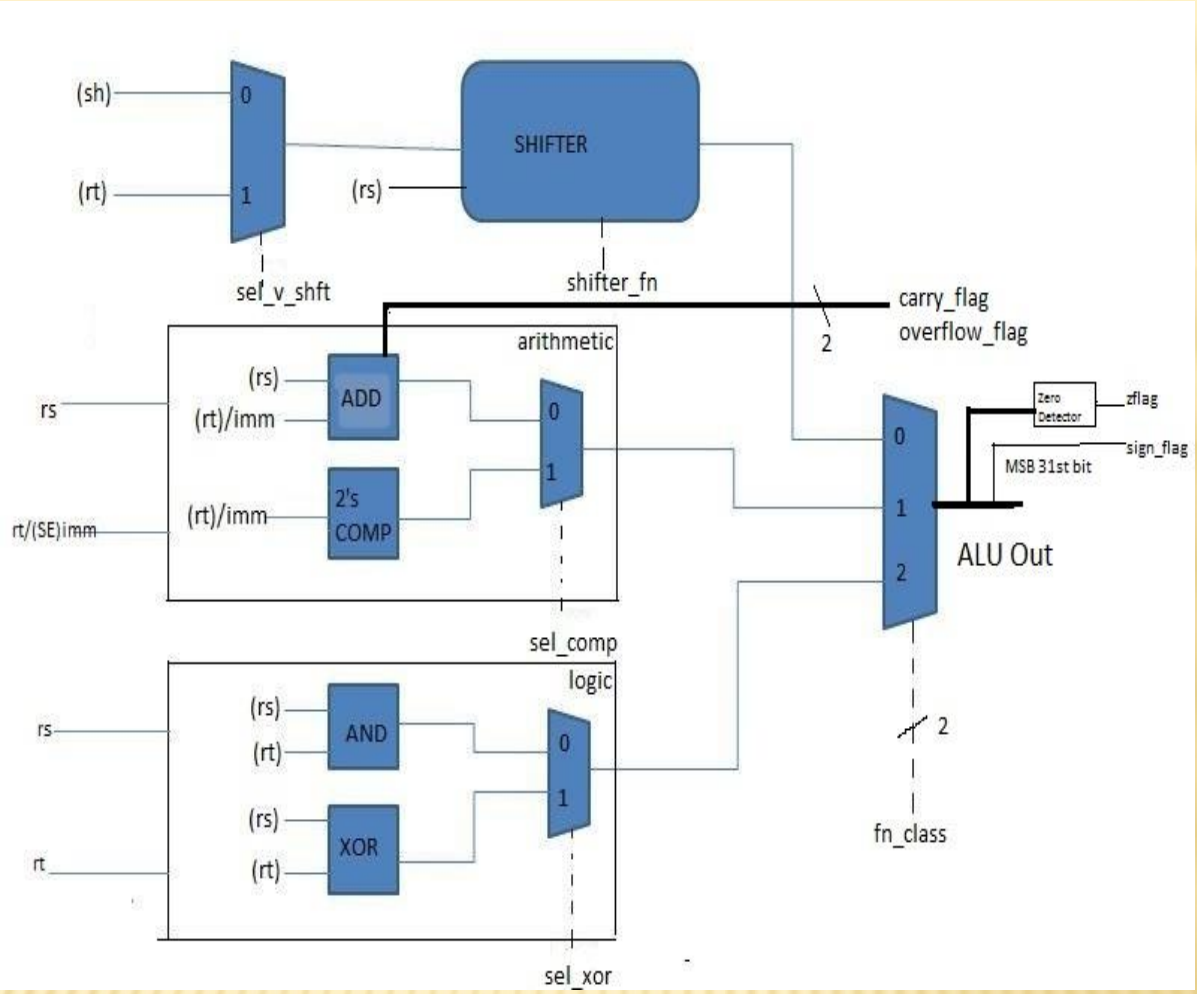
1.5. ALU:

Input: 32 Bit operand1(comes from read port1(rs)) , 32 Bit operand2(comes from read port2(rt/SE-Imm)) , shamt(shift amount for shift instructions) , 5control_signals from control unit

Output: 32 bit alu output, 4 flags

ALU has 3 step hierarchy





1.5.1 Arithmetic Block – A control signal called sel_comp goes to this module . If sel_comp is 0,operand1 +operand2 is calculated in the ripple carry adder . If sel_comp is 1,y is complemented and passed to output.
There are few flags associated with this module.
1.5.1.1 Carry flag - It might happen that there can be a carry over produced in the 32 bit addition module . This is termed as carry flag.
1.5.1.2 Zero flag - It is 1 when the output of ALU is 0.It lets us know that when the output is 0.
1.5.1.3 Overflow flag – This overflow is obtained by Xor -ing the last two carries produced in the adder . This tells us whether there is an overflow in the addition or not.
1.5.1.4 Sign flag – There should be some flag that tells us the sign of an output , whether the output is positive or negative . This sign flag does that job which is simply the MSB of the output.
The above mentioned flags are passed to the next PC module further for the branch checking conditions.

1.5.2 Shifter –

A 2 bit control signal called shift_fn[2:0] (Actually this is Fn[4], Fn[3], Fn[0],)is passed in this module .Now , according to this signal value , different shifting operations are performed . If the signal shift_fn[2] is 1 then it is a arithmetic shift. The shift is a right shift if signal shift_fn[1] is 1 otherwise it is 0 . And shift_fn[0] says the shift operation is by fixed shamt if it is 1 or by variable amt(operand2[4:0]) if 0.

1.5.3 Logic – A signal called sel_xor is passed in this module . Now , if sel_xor is 0, **(**operand1 **and** operand2 **)** is computed and passed to the output . If sel_xor is 1,**(**operand1 **xor** operand2**)** is computed and passed to the output.
Again , this module also sends the sign flag and zero flag.

Instruction	RTL operations
Add	Reg[rs]<- Reg[rs]+ Reg[rt]
Comp	Reg[rs]<- 2's complement Reg[rt]
Add immediate	Reg[rs]<- Reg[rs]+ SE-imm
Comp immediate	Reg[rs]<- 2's complement Reg[rt]
AND	Reg[rs]<- Reg[rs]& Reg[rt]
XOR	Reg[rs]<- Reg[rs]^ Reg[rt]
Shift left logical	Reg[rs]<-(logical left shift by shamt) Reg[rt]
Shift right logical	Reg[rs]<-(logical right shift by shamt) Reg[rs]
Shift left logical variable	Reg[rs]<-(logical left shift by operand2[4:0]) Reg[rs]
Shift right logical variable	Reg[rs]<-(logical right shift by operand2[4:0]) Reg[rs]
Shift right arithmetic	Reg[rs]<-(logical arithmetic right shift by shamt) Reg[rs]
Shift right arithmetic variable	Reg[rs]<-(logical arithmetic right shift by operand2[4:0]) Reg[rs]

Instruction	Alu imm	Block_sel	Function _ sel	Shift_operation
Add	0	01	0	xx
Comp	0	01	1	xx
Addi	1	01	0	xx
Compi	1	01	1	xx
And	0	10	0	xx
Xor	0	10	1	xx
Shll	0	00	0	00
Shrl	0	00	0	01
Shllv	0	00	1	00
Shrlv	0	00	1	01
Shra	0	00	0	11
Shrav	0	00	1	11
lw	1	xx	xx	xx
sw	1	xx	xx	xx

All the other Instructions do not need Arithmetic signals. So all these control signals are don't care for those .

1.6.Register File:

The Register File is the fastest form of storage used for storing all data corresponding to on-going calculation steps . In our case it has 32 to registers of 32 bit size . The basic two operations we are able to do in a register is to write data from one address at a time and read data from two addresses simultaneously.

1.6.1 Register Reading

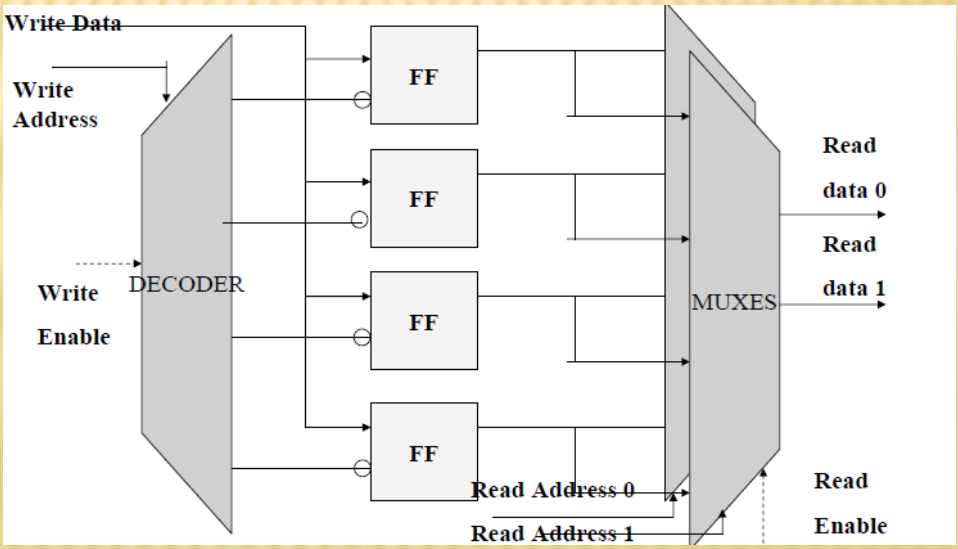
The register file consists of 32 32 bit registers . Now , to read a register , we need the address of the register . As there are 32 registers .,we can address the registers using 5 bits . There are two 5 bit input addresses that are passed in via rs and rt . There is a control signal Jreg . When 1 it gives reg $31 value to readport1 The data stored at those addresses i. e (rs) and (rt) are obtained at the 2 Read value ports available in the Register module . These are done in the positive edge of clock.

1.6.2 Register Writing

This module requires 3 signals from the controller . They are Regwrite , Regdst and Regindata . what we need to do is wait for the positive edge of the clock cycle . To write the data , we need to make Regwrite = 1. Now , that the time of writing is decided , we have to check so as to which data should be written . This comes from a multiplexer outside which tells us that the data to be written comes from ALU , D cache or IncrPC . Now , this data is stored at an address which comes from another multiplexer outside. This address can be rs,rt or $31.In this way the data is written at specified address . Also , there is a signal called Reg which is added as a signal to a multiplexer with rs and r31 as inputs. This is basically used for branching purposes in future.

Instructions	Reg_dst	Reg_write	ReginData	RegJ
Add	00	1	00	0
Comp	00	1	00	0
Add Immediate	00	1	00	0
Complement Immediate	00	1	00	0
AND	00	1	00	0
XOR	00	1	00	0
Shift Left Logical	00	1	00	0
Shift Right Logical	00	1	00	0
Shift Left Logical Variable	00	1	00	0
Shift Right Logical Variable	00	1	00	0
Shift Right Arithmetic	00	1	00	0
Shift Right Arithmetic Variable	00	1	00	0

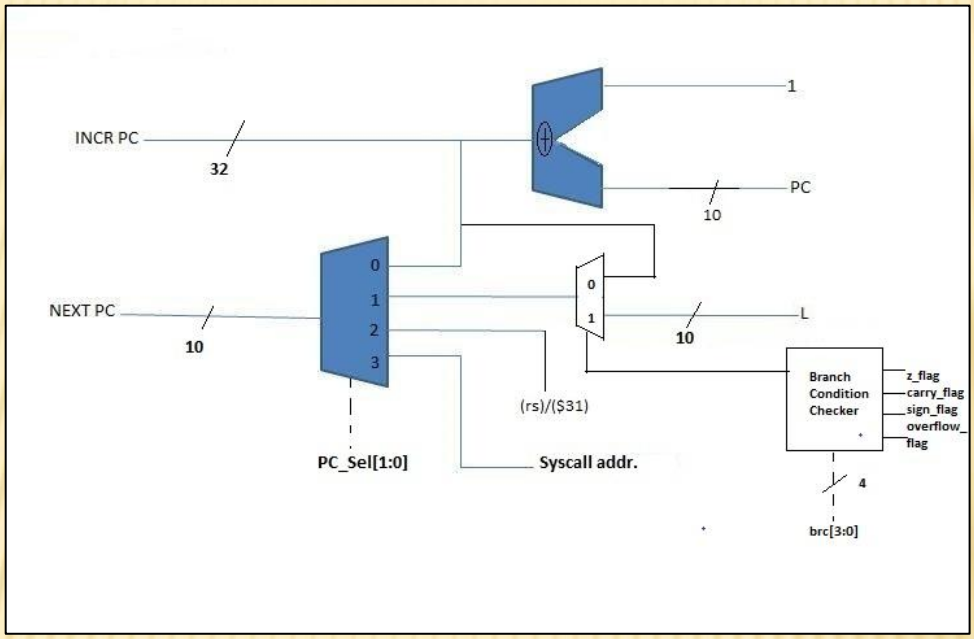
Instructions	Reg_dst	Reg_write	ReginData	Jreg
Load Word	01	1	01	0
Store Word	XX	0	XX	0
Unconditional branch	XX	0	XX	0
Branch Register	XX	0	XX	0
Branch on zero	XX	0	XX	0
Branch on not zero	XX	0	XX	0
Branch on Carry	XX	0	XX	0
Branch on No Carry	XX	0	XX	0
Branch on Sign	XX	0	XX	0
Branch on Not Sign	XX	0	XX	0
Branch on Overflow	XX	0	XX	0
Branch on No Overflow	XX	0	XX	0
Call	10	1	10	0
Return	XX	0	XX	1



1.7 Next Address Block:

Input: PC, jta(L)(comes from branch instructions),jump target from register (it comes from $rs or $31),syscall address,PC_sel [1:0] signals

Output : NextPc(Next instruction address to be accessed from ICache),IncrPc(PC+4 value used in call instruction to save the value of return address).



Instruction	PC_Sel[1:0]	Brc[]
Add	00	XXXX
Comp	00	XXXX
Add Immediate	00	XXXX
Complement Immediate	00	XXXX
AND	00	XXXX
XOR	00	XXXX
Shift Left Logical	00	XXXX
Shift Right Logical	00	XXXX
Shift Left Logical Variable	00	XXXX
Shift Right Logical Variable	00	XXXX
Shift Right Arithmetic	00	XXXX

Instruction	PC_Sel[1:0]	Brc[]
Load Word	00	XXXX
Store Word	00	XXXX
Unconditional branch	01	0000
Branch Register	10	XXXX
Branch on zero	01	0001
Branch on not zero	01	0010
Branch on Carry	01	0011
Branch on No Carry	01	0100
Branch on Sign	01	0101
Branch on Not Sign	01	0110
Branch on Overflow	01	0111
Branch on No Overflow	01	1000
Call	01	1001
Return	10	XXXX