KGP RISC Instruction Format and Encoding

Group 32

Anirban Haldar (20CS10008) Saikat Moi (20CS10050)

Instruction Formats:

Instruction formats are of the following 3 formats:
1. R (register) - Format
2. I (Immediate) - Format
3. J (Jump) - Format

R-Format:

Field Size	6	5	5	5	6	5	Instructions
R-format	op	rs	rt	shamt	func	dc	Add, cmp, and, xor, all-shift instructions

At present, all R-format instructions are kept under the same op-code:

Op-code - **000000**

Instruction	Function Code
Add	1
Comp	2
AND	3
XOR	4
shll	1100
shrl	1110
shra	1111
shrav	1011
shrlv	1010
shllv	1000
dff	

I-Format:

Field-size	6	5	5	16	Instructions
I-format	op	rs	rt/ dc	address/ immediate	addi, compi, lw, sw

The table given below gives the op-code for all the above instructions:

Instruction	**op-code**
addi	001000
compi	001010
lw	010000
sw	010010

J-Format:

Field Size	6	26	Instructions
J-Format	op	Target address	b, br, bl, bcy, bncy

The table given below gives the op-code for all the above instructions:

Instruction	**op-code**	**fmt**
b	100000	op | rs | xxxxxxxx |
b	101000	op | label |
bny	101001	op | label |
bncy	101010	op | label |
bl	101011	op | label |

bltz	110000	op | rs | xx | label |
bz	110001	op | rs | xx | label |
bnz	110010	op | rs | xx | label |

Register Convention:

The architectural design is to be used in a similar fashion as per the MIPS convention for regular use as mentioned:

Symbolic Name	**Number**	**Use**
zero	0	Constant 0
at	1	Reserved for assembler
v0 - v1	2-3	Result registers
a0 - a3	4-7	Argument registers
t0 - t9	8 - 15, 24 - 25	Temporary registers
s0 - s7	16 - 23	Saved registers
k0 - k1	26 - 27	Kernel registers
gp	28	Global Data Pointer
sp	29	Stack Pointer
fp	30	Frame Pointer
ra	31	Return Address

ALU Design:

ALU Flags:

1. Zero
2. Sign (0 if positive, 1 for negative)
3. Carry

ALU-operation	funcode[3]	funcode[2]	funcode[1]	funcode[0]
forward	0	0	0	0
add	0	compl/ not-compl	0	1
and	0	0	1	0
xor	0	0	1	1
shift	1	shamt/ reg	right/ left	log/ arithm

Truth table for ALU control signals:

operation	opcode	funcode	alucode[3]	alucode[2]	alucode[1]	alucode[0]
add	000000	000001	0	0	0	1
comp	000000	000101	0	1	0	1
addi	001000	-	0	0	0	1
compi	001001	-	0	1	0	1
and	000000	000010	0	0	1	0
xor	000000	000011	0	0	1	1
shll	000000	001100	1	1	0	0
shrl	000000	001110	1	1	1	0

shllv	000000	001000	1	0	0	0
shrlv	000000	001010	1	0	1	0
shra	000000	001111	1	1	1	1
shrav	000000	001011	1	0	1	1
lw	010000	-	0	0	0	1
sw	011000	-	0	0	0	1
b	101000	-	0	0	0	0
br	100000	-	0	0	0	0
bltz	110000	-	0	0	0	0
bz	110001	-	0	0	0	0
bnz	110010	-	0	0	0	0
bl	101011	-	0	0	0	0
bcy	101001	-	0	0	0	0
bncy	101010	-	0	0	0	0

ISA Datapath and Control Signals:

There are 3 main ways in which instruction memory can be referenced to in the instruction set:

1. Direct PC addressing
2. PC-relative addressing
3. (Pseudo) Direct Jump Addressing

Direct Addressing:

This type of addressing takes place for ‘br’ instruction where the argument register has the exact address to jump to.

PC_Relative Addressing:

This type of addressing takes in any 16-bit Label instruction like bz, bnz, blitz where the absolute address is calculated using the following formula:

Address = (PC + 4) + SignExtended (Label)

Pseudo Direct Addressing:

This type of addressing takes in any 26-bit label instruction like b, bl, bcy, bncy where the absolute address is calculated using the following formula:

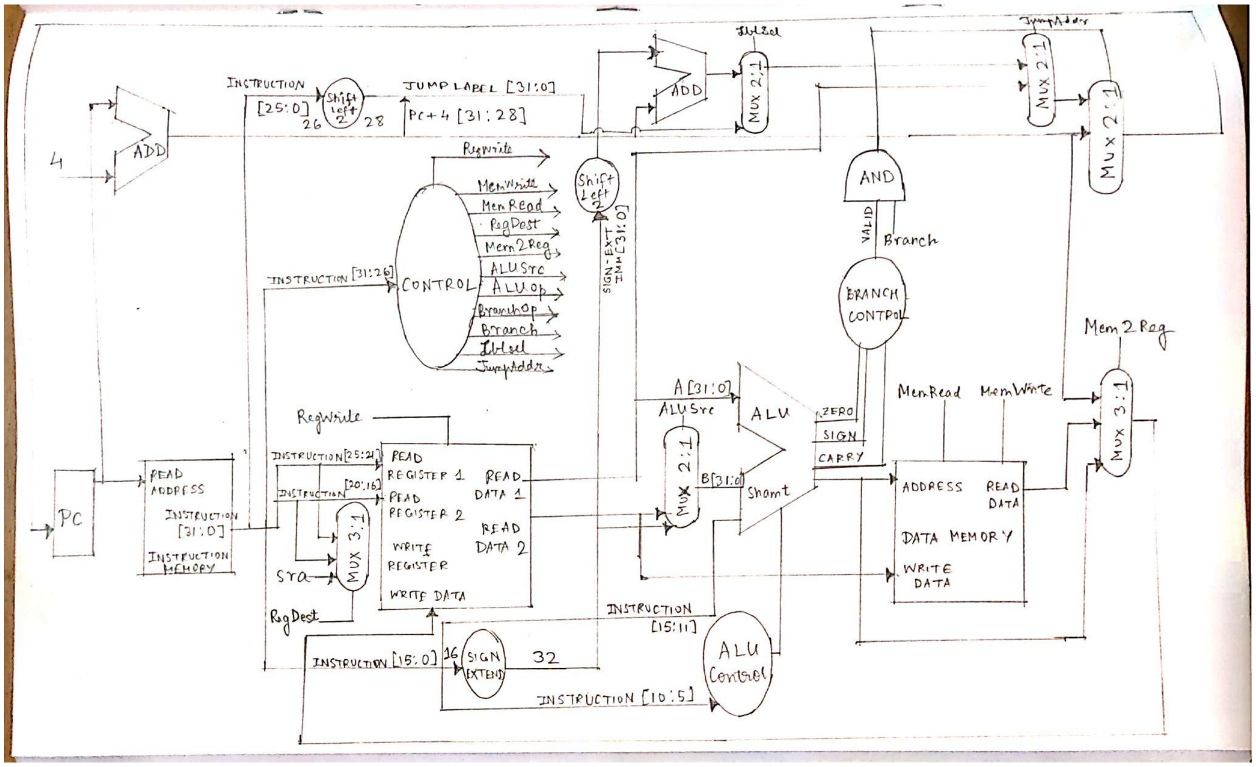
Address = {(PC + 4)[31:28], {Label, 2b’00}}

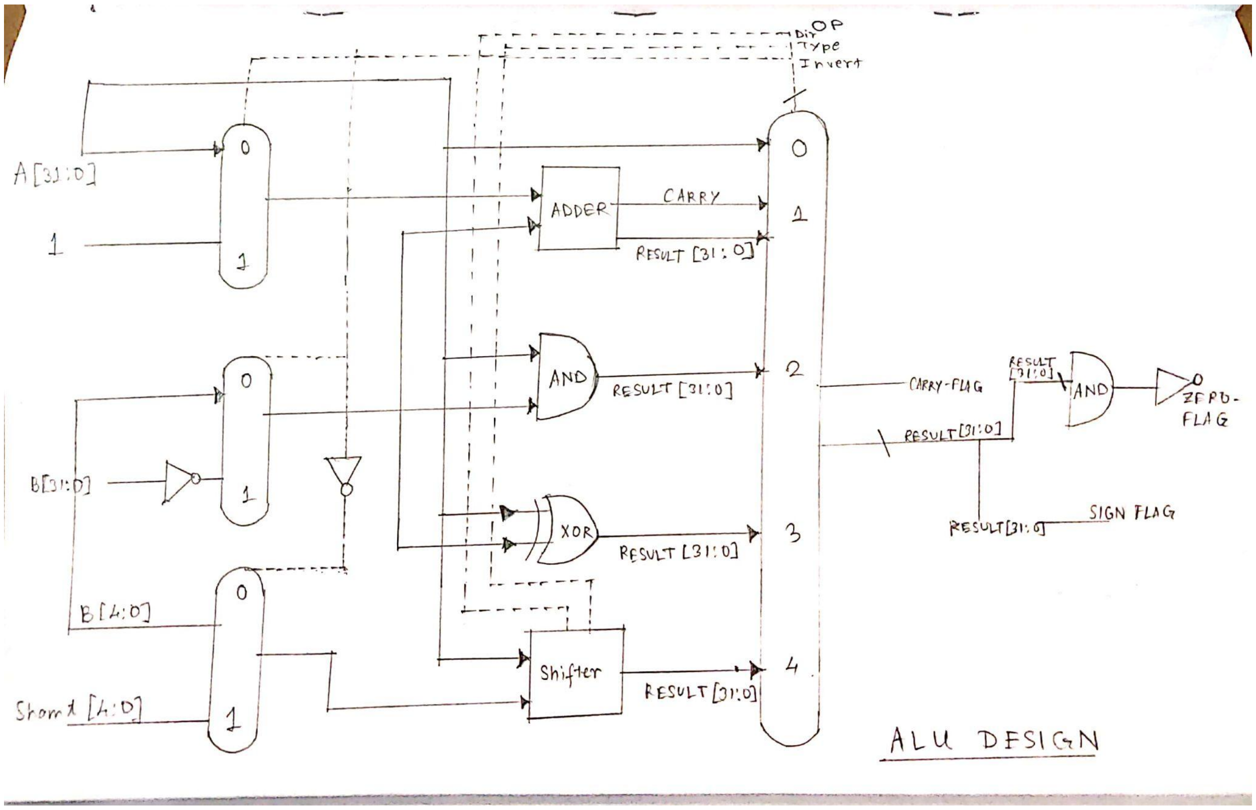
Control Signals:

1. Rewrite: to decide whether to write into the register file or not
2. RegDst[1:0]: for the destination register for the write-register (can be $ra, rs, rt)
3. ALUSrc: for the Source for the 2nd input to the ALU (can be rt, sgn-extend(imm))
4. MemRead: to decide whether to read from Data Memory or not
5. MemWrite: to decide whether to write into the Data Memory or not
6. Mem2Reg[1:0]: for write-data for the register files (can be one of: PC+4, mem[], result_ALU)
7. LblSel: for select type of addressing for PC-Relative and PseudoDirect
8. JumpAdd: to decide whether the jump address comes from a source reg (rs) or from a label

Truth Table for control signals

Op	Opcode	RegDst	RegWrite	ALUSrc	MemRead	MemWrite	Mem2Reg	LblSel	JumpSel
add	000000	00	1	0	0	0	00	x	x
comp	000000	00	1	0	0	0	00	x	x
addi	001000	00	1	1	0	0	00	x	x
compi	001001	00	1	1	0	0	00	x	x
and	000000	00	1	0	0	0	00	x	x
xor	000000	00	1	0	0	0	00	x	x
shll	000000	00	1	0	0	0	00	x	x
shrl	000000	00	1	0	0	0	00	x	x
shllv	000000	00	1	0	0	0	00	x	x
shrlv	000000	00	1	0	0	0	00	x	x
shra	000000	00	1	0	0	0	00	x	x
shrav	000000	00	1	0	0	0	00	x	x
lw	010000	01	1	1	1	0	01	x	x
sw	011000	x	0	1	0	1	x	x	x
b	101000	x	0	x	0	0	x	0	0
br	100000	x	0	x	0	0	x	x	1
bltz	110000	x	0	x	0	0	x	1	0
bz	110001	x	0	x	0	0	x	1	0
bnz	110010	x	0	x	0	0	x	1	0
bl	101011	10	1	x	0	0	10	0	0
bcy	101001	x	0	x	0	0	x	0	0
bncy	101010	x	0	x	0	0	x	0	0


ADD
4
INSTRUCTION [25:0]
Shift Left 2
26
28
JUMP LABEL [31:0]
PC+4 [31:28]
ADD
lblsel
Mux 2:1
JumpAdder
Mux 2:1
Mux 2:1
RegWrite
MemWrite
MemRead
RegDest
Mem2Reg
ALUSrc
ALU op
BranchOp
Branch
lblsel
JumpAdder
CONTROL
INSTRUCTION [31:26]
Shift Left 2
SIGN-EXT IMM [31:0]
AND
VALID
Branch
BRANCH CONTROL
Mem2Reg
PC
READ ADDRESS
INSTRUCTION [31:0]
INSTRUCTION MEMORY
RegWrite
INSTRUCTION [25:21]
INSTRUCTION [20:16]
READ REGISTER 1
READ REGISTER 2
WRITE REGISTER
WRITE DATA
READ DATA 1
READ DATA 2
MUX 3:1
Sra
RegDest
A [31:0]
ALUSrc
MUX 2:1
B [31:0]
ALU
Shamt
ZERO
SIGN
CARRY
MemRead
MemWrite
ADDRESS
READ DATA
DATA MEMORY
WRITE DATA
MUX 3:1
INSTRUCTION [15:11]
INSTRUCTION [15:0]
16
SIGN EXTEND
32
ALU Control
INSTRUCTION [10:5]


OP
Dir
Type
Invert
A [31:0]
1
0
1
B [31:0]
0
1
B [4:0]
Shamt [4:0]
0
1
ADDER
CARRY
RESULT [31:0]
AND
RESULT [31:0]
XOR
RESULT [31:0]
Shifter
RESULT [31:0]
0
1
2
3
4
CARRY-FLAG
RESULT [31:0]
RESULT [31:0]
AND
ZERO-FLAG
RESULT [31:0]
SIGN FLAG
ALU DESIGN