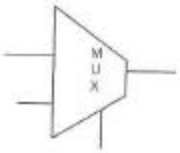
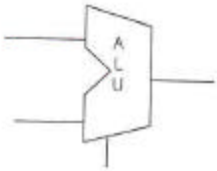
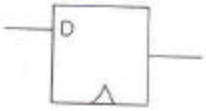
CH 5



Multiplexor

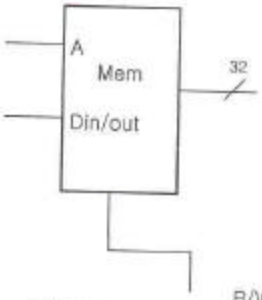


ALU

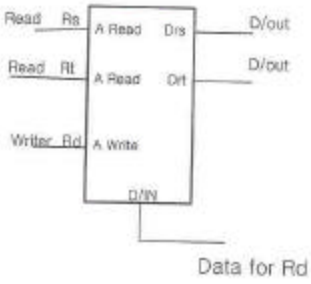


D-FF
(Register)

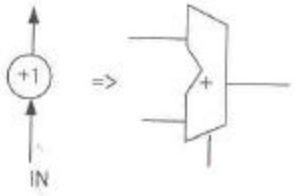
Program Counter
(Register)



Memory

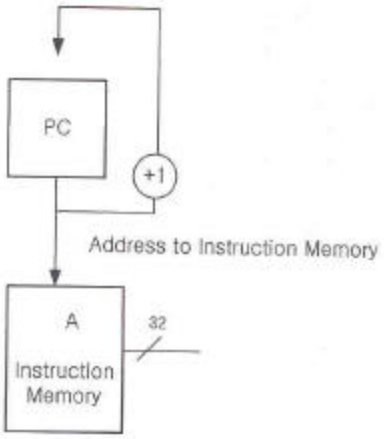


Register file



Incrementor

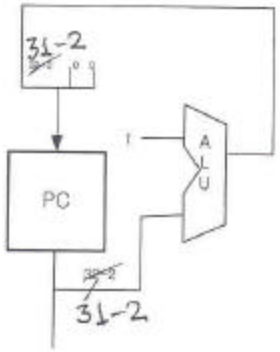
#Program Counter

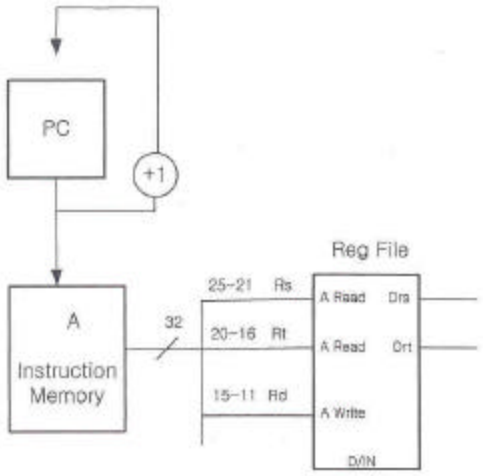


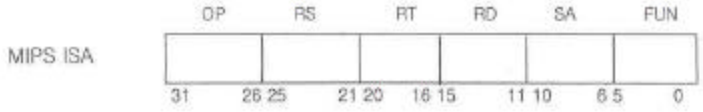
Why just increment 1?

"Because of Alignment"

"Every instruction is 4 bytes long".

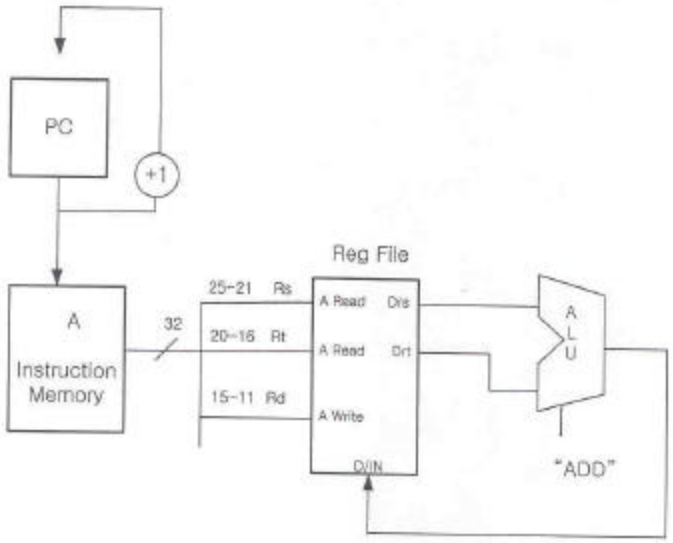




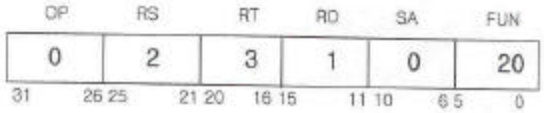


Instruction

How to Implement ADD Instruction to work?

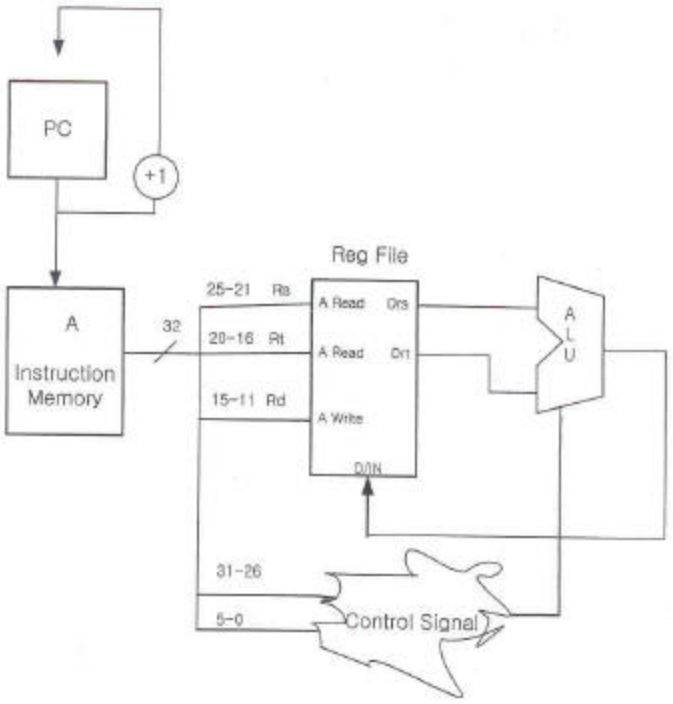


ADD R1, R2, R3

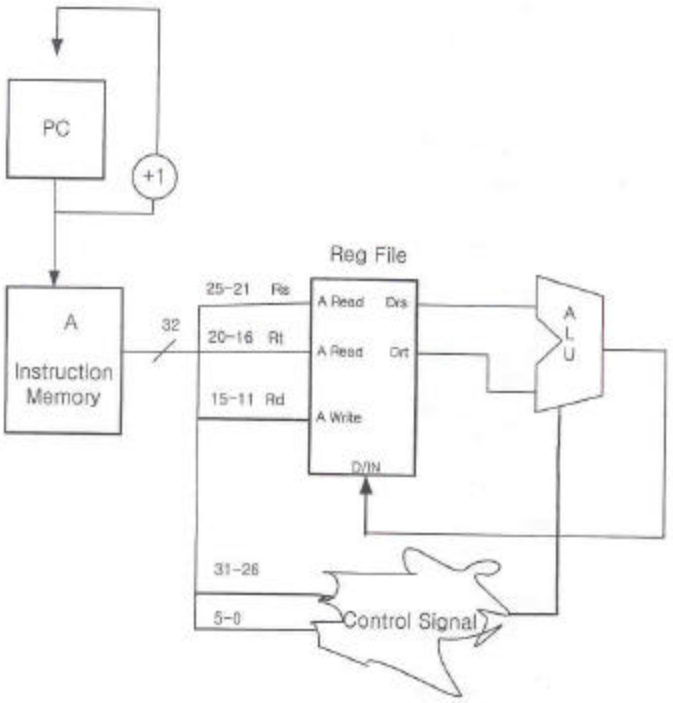


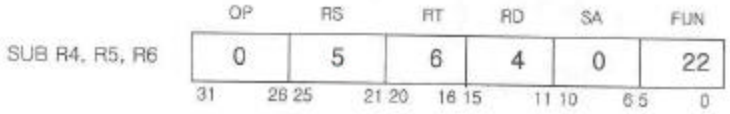
ADD Rd, Rs Rt

Where the "ADD" control signal for ALU comes?
"From the instruction field "5-0" [function field]
and OPCode(31-26)"



OP	FUN	ALUOP
0	20	ADD

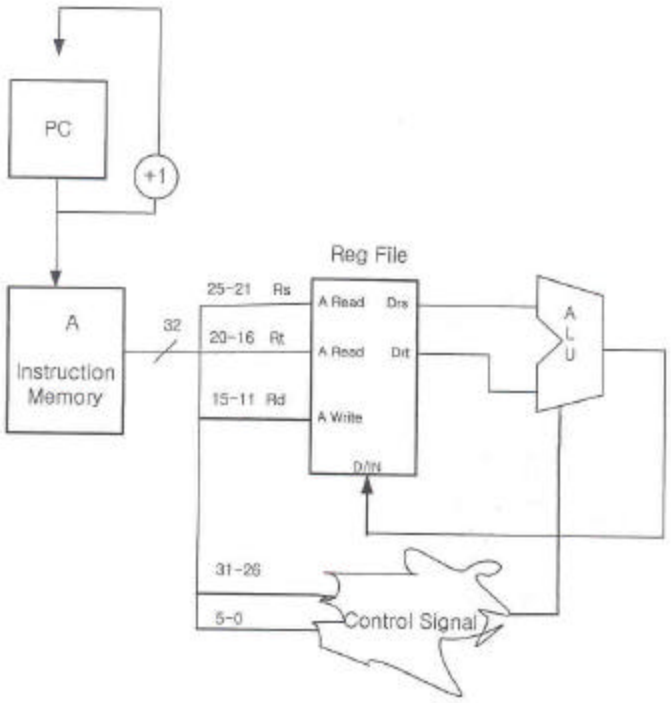




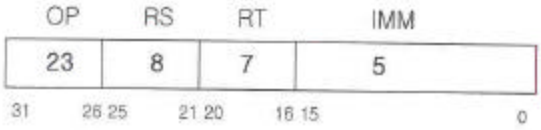
	OP	FUN	ALUOP
ADD R1, R2, Rs	0	20	ADD
SUB R4, R5, R6	0	22	SUB

Can the above H/W perform ADDi Instructions?"

What do we need?



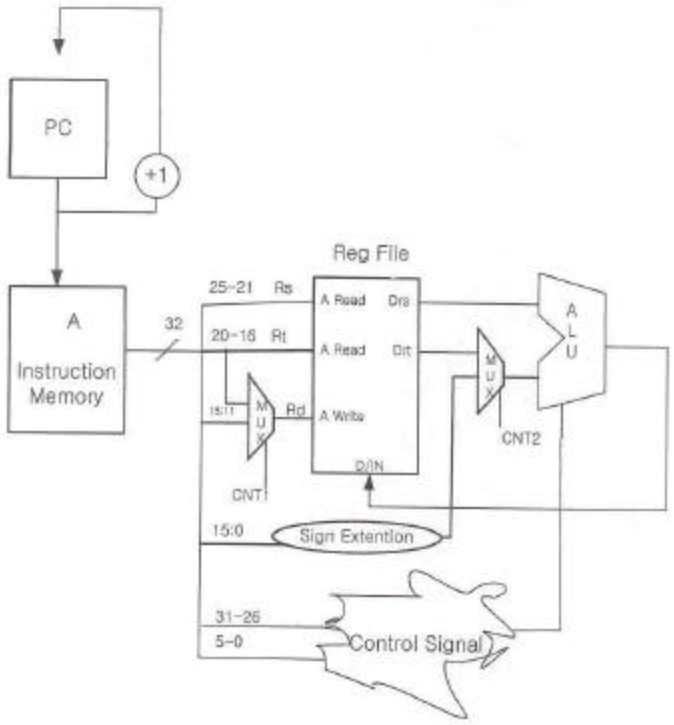
ADDi R7, R8, 5



I format

ADDi rt,rs, imm

We need to get the immediate data, and "RT" field becomes acting like destination field at "R" format.

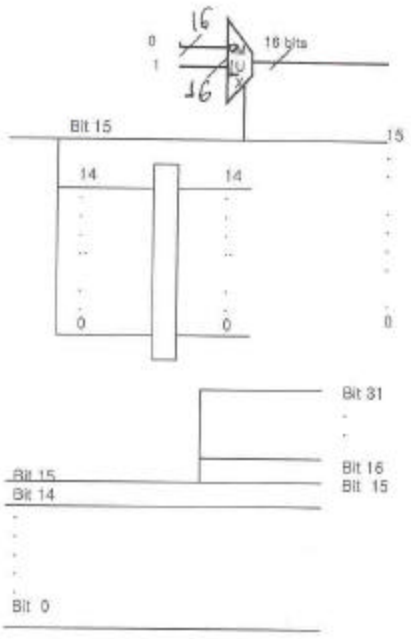


Where the "CNT" signal come?
"From the OPCode field (31-26)

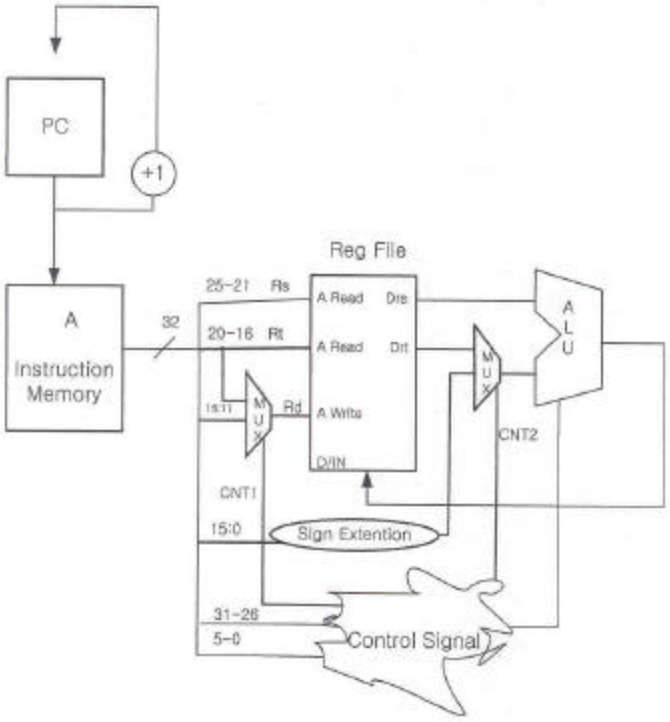
	OP	FUN	ALUOP
ADD R1, R2, Rs	0	20	ADD
SUB R4, R5, R6	0	22	SUB
ADDi R7, R8, 5	8	X	ADD

25-21 Rs

20-16 Rt



(Probably not work due to fan-out)



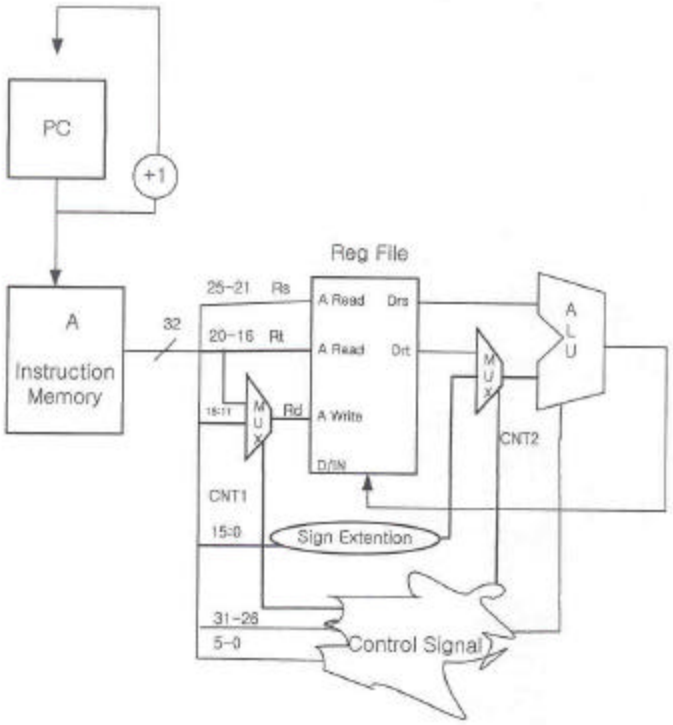
"Can the above H/W perform OR Instruction?"

"Do we need more H/W?"

OR R9, R10, R11

OR rd, rs, rt

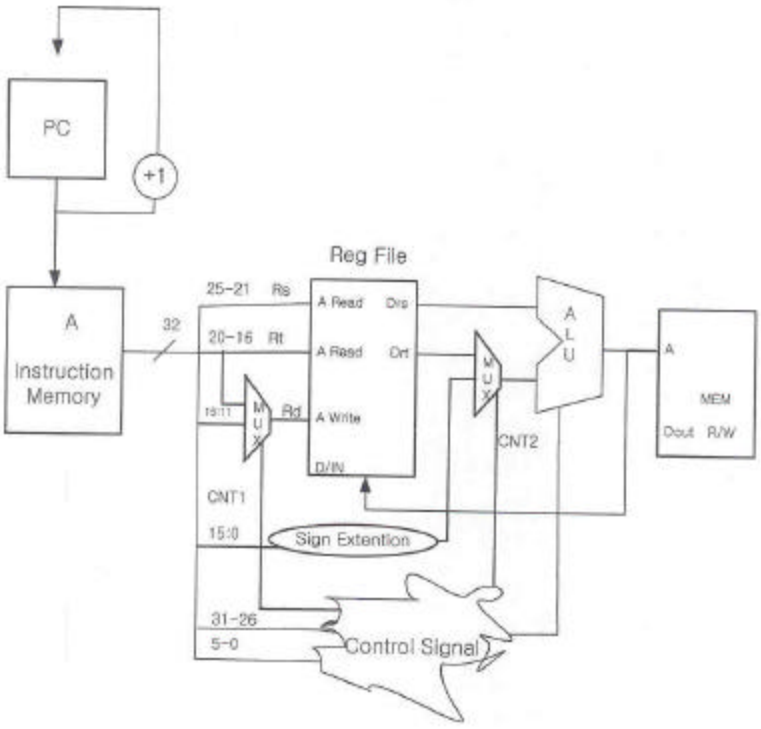
OP	RS	RT	RD	SA	FUN
0	10	11	9	0	25
31 26	25 21	20 16	15 11	10 6	5 0



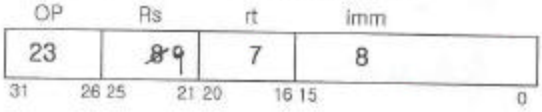
	OP	FUN	ALUOP
ADD R1, R2, Rs	0	20	ADD
SUB R4, R5, R6	0	22	SUB
ADDI R7, R8, 5	8	X	ADD
OR R9, R10, R11	0	25	OR

It also can perform "ORi" without having more H/W."

So far we don't have any instructions to access memory and we don't have data memory unit yet.

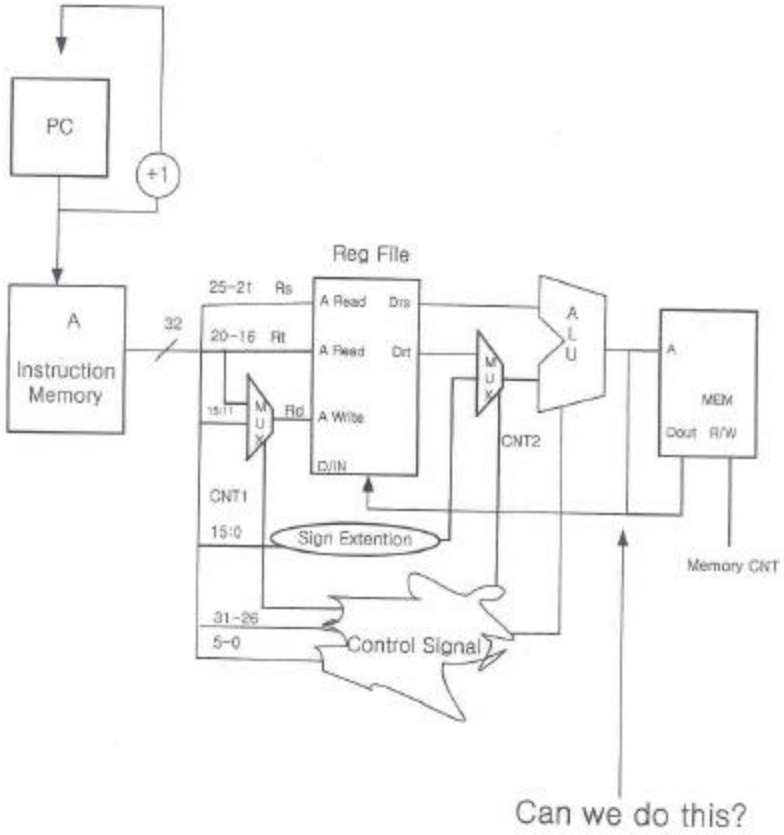


LW R7, 8(R9) ; LW rt, address



How to implement this? And what will be ALU operation?

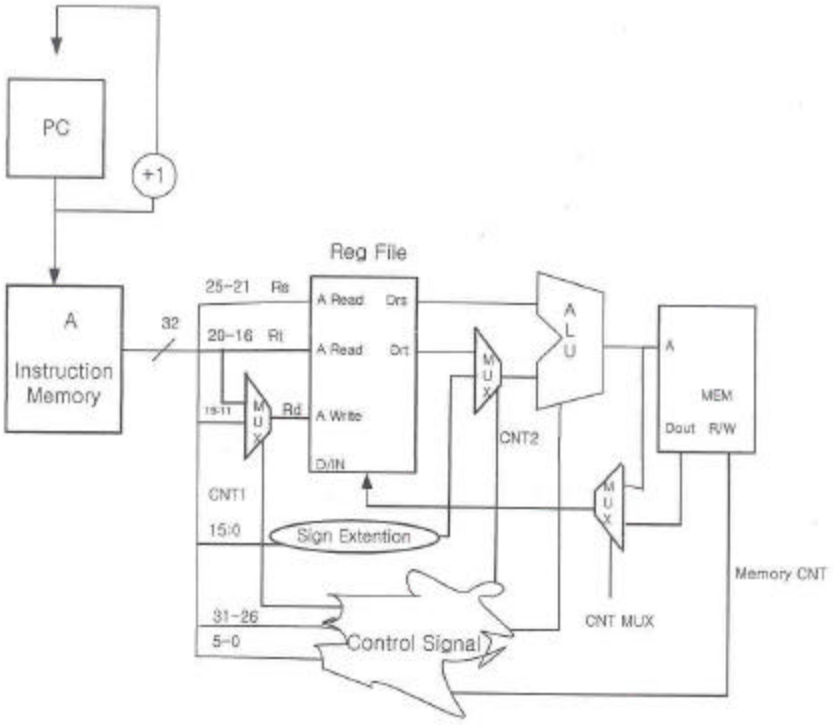
One way to do this is using the ALU to compute memory address and put the output of ALU (computed memory address: (R9+8)) to the address of memory and put the data from the memory register file (R7).



Memory output should go back to register file and ALU output should go back to register file.

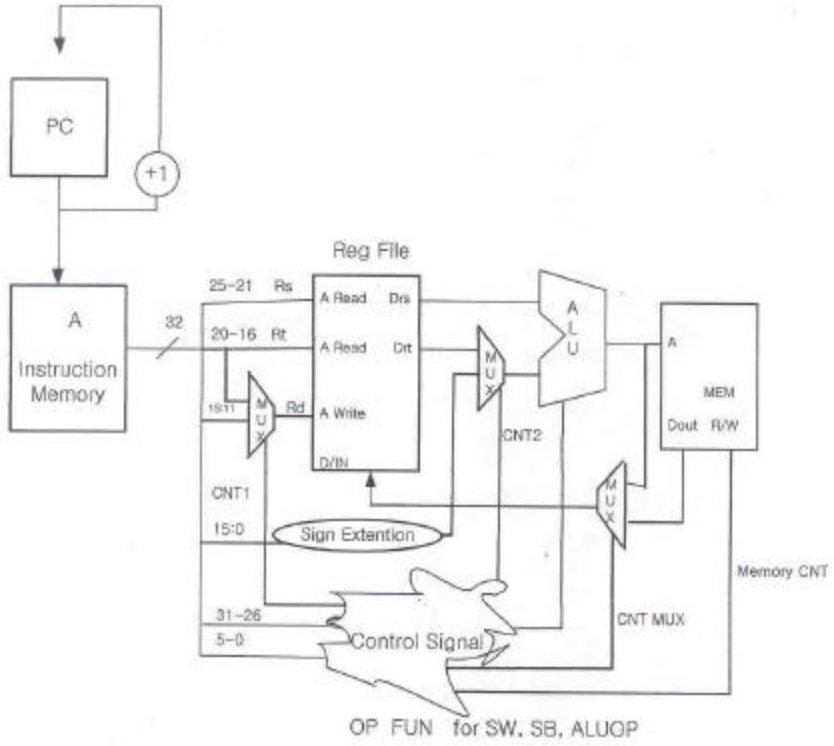
But this time ALU output is memory address and Dout is data. This is a collision.

How can we solve this? => Multiplexor



Where the CNTMUX come?

=> If "LW" instruction, then the mux should select input from memory.
=> So, from Control Signal.



ADD R1, R2, R3	0	20	ADD
SUB R4, R5, R6	0	22	SUB
ADDi R7, R8, 5	8	X	ADD
OR R9, R10, R11	0	25	OR
LW R7, 8(R9)	23	X	ADD

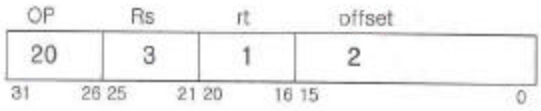
Can the above H/W perform "LB"?
What will be problems?
How can you modify above H/W to perform "LB"?

LB is loading only one byte and LW is loading 4bytes. So we have to think about either modifying memory unit (Memory Module) to get one byte data at a time (LW, we assume the memory unit giving us 4 bytes data).

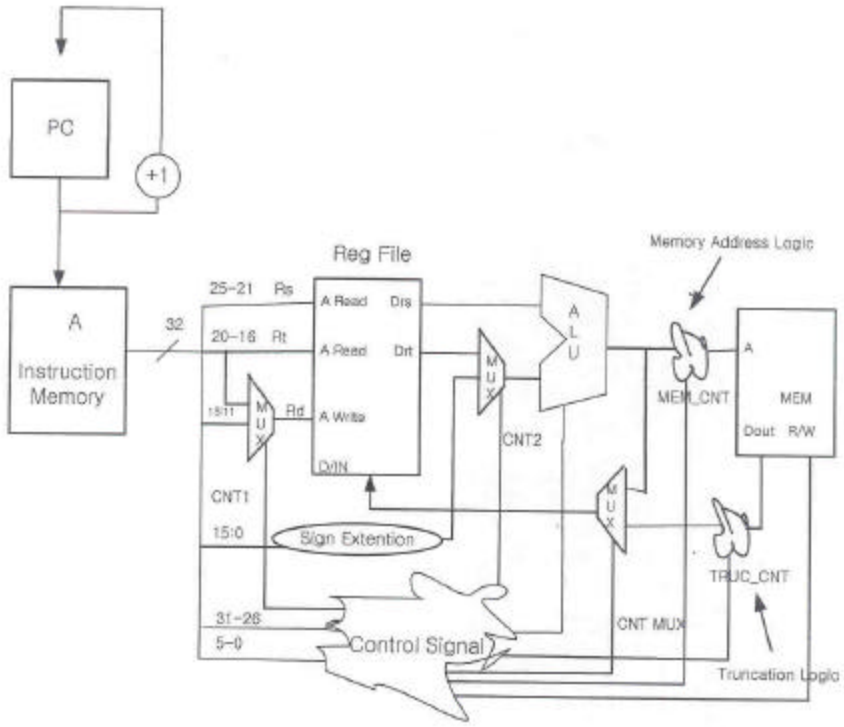
So Let's assume the memory unit giving us 4 bytes data, then we have to truncate extra 3 bytes data, and we have to think about memory address.

If we assume we could access arbitrary memory address, then we need logic to do that.

LB R1 2(R3) LB rt, address



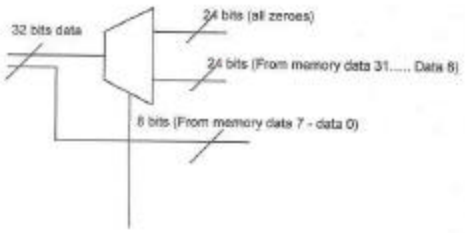
We need logic to truncate extra 3 bytes data and memory address logic to access arbitrary address (Alignment problem).



So when we perform "LBU", truncation logic will give us byte data with ~~20~~ 24 zeroes padded to upper part.
So we could make TRUC_CNT and MEM_CNT from Control Signal.
It(LBU) can be done with above H/W.
But think about "SB".
Can "SB" be done with above approach?

64-a

TRUNCATE LOGIC



MUX-select (from Control Signal)

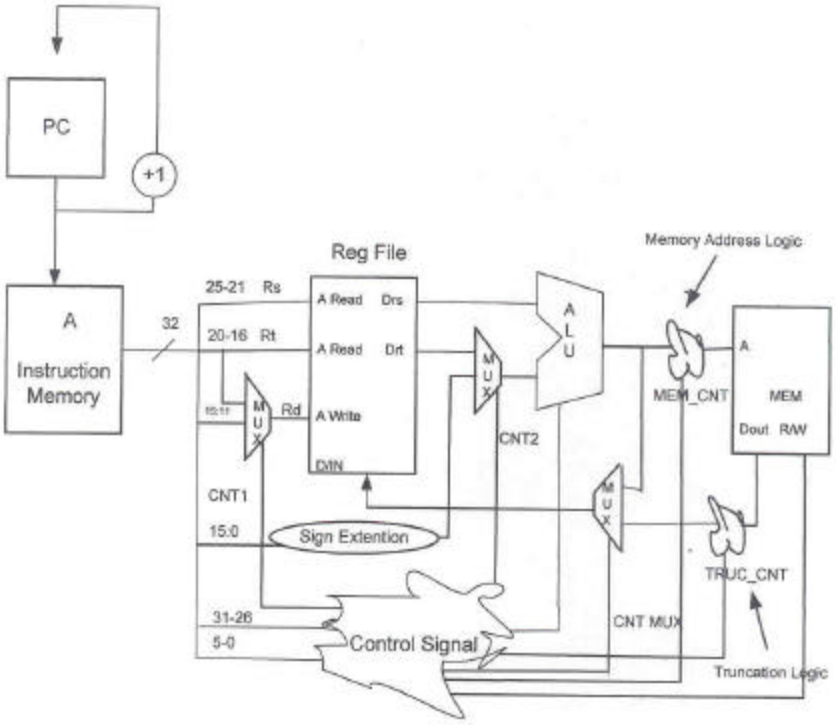
if LB=> select 24 bits zeroes
if LW=> select 24 bits from memory

But because of alignment problem and accessing arbitrary address the logic will be complicated.

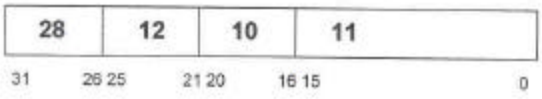
Address =>					<= data
0	0	0	0	1	
0	0	0	1	2	
0	0	0	2	3	
0	0	0	3	4	

The above logic is working when accessing " 0000" but what about accessing " 0001"?

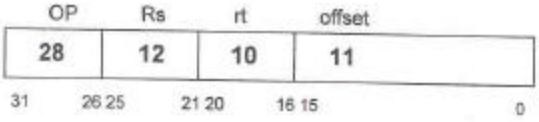
The problem of SB is we are saving only one byte out of 4 bytes. So we have to think about how to save data to the memory.
For SW, if we assume can save 4 bytes by giving an address, "SW will be easily implemented.

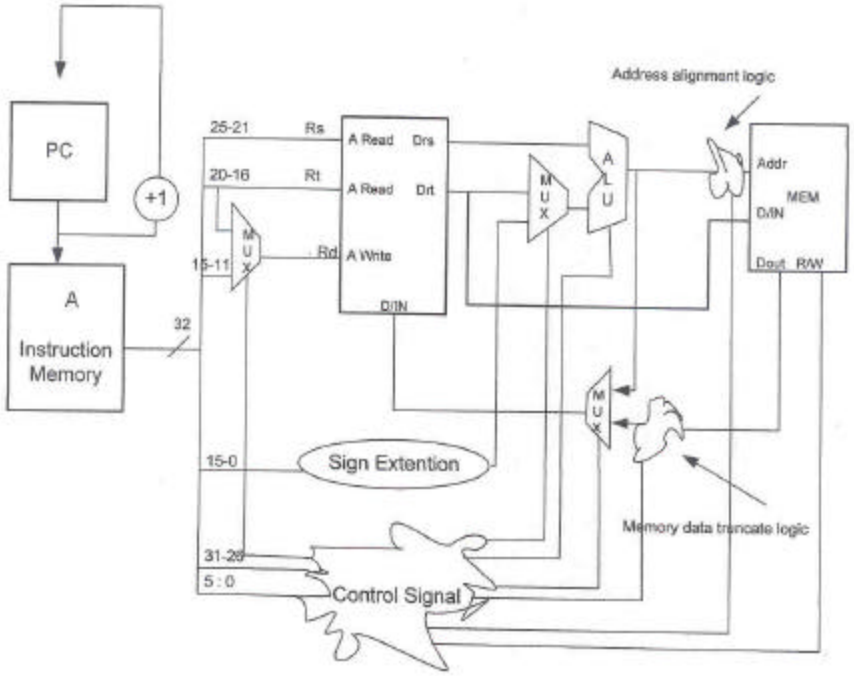


SW R10, 11(R12)



SW rt, address





Can "SB" be done with above "HW"?
Can we remove the new datapath?
Can we just by pass through ALU?

When we cover Verilog, we implement ALU unit.

It can perform “ADD”, “SUBTRACTION”, “SLT” ,” AND”, “OR”, “ByPassA” and “ByPassB”.

So, we could bypass Drt value through ALU, we can remove the new datapath.

BUT, The Answer is NO.

No., ALU is performing Address calculation (RS + offset).

So, we can not bypass this time.

Then we need the new datapath.

Again, even we can perform SW, doing "SB" is not OK.

For "LB", we could use "LW" and then truncate some bits (24 bits). But for "SB" case, we can not do that.

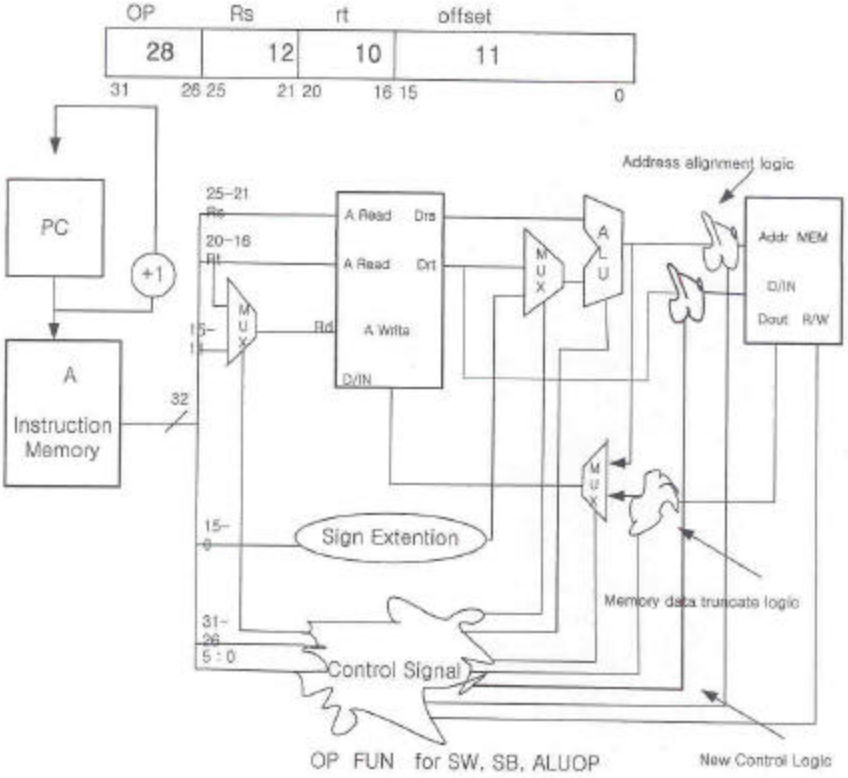
One way of doing that is "LW" and modifying byte information and using "SW".

But this brings extra "LW".

- So, we may assume that we could store byte and for SW instruction, we assume we could store 4 bytes.

Then we need control circuit between Drt and Din.

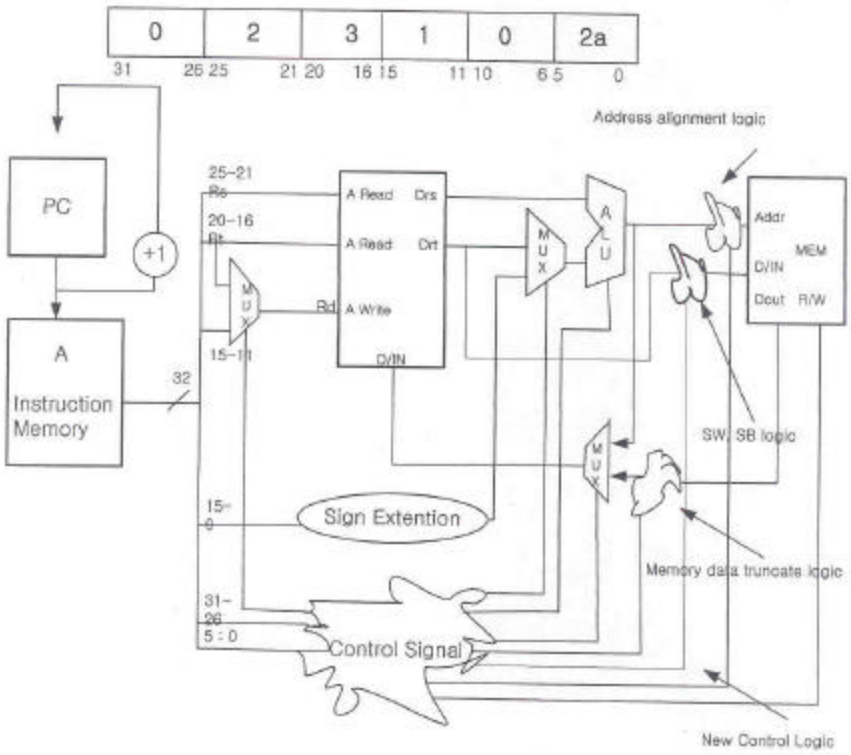
SB rt, address
SB R10, 11(R12)



OP FUN for SW, SB, ALUOP

ADD R1, R2, R3	0	20	ADD
SUB R4, R5, R6	0	22	SUB
ADDI R7, R8, 5	8	X	ADD
OR R9, R10, R11	0	25	OR
LW R7, 8(R9)	23	X	ADD
LB R1, 2(R3)	20	X	ADD
SW R10, 11(R12)	2b	X	ADD
SB R10, 11(R12)	28	X	ADD

Slt R1, R2, R3
Slt rd, rs, rt

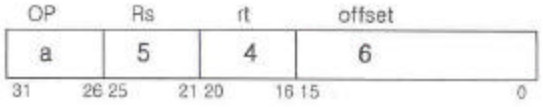


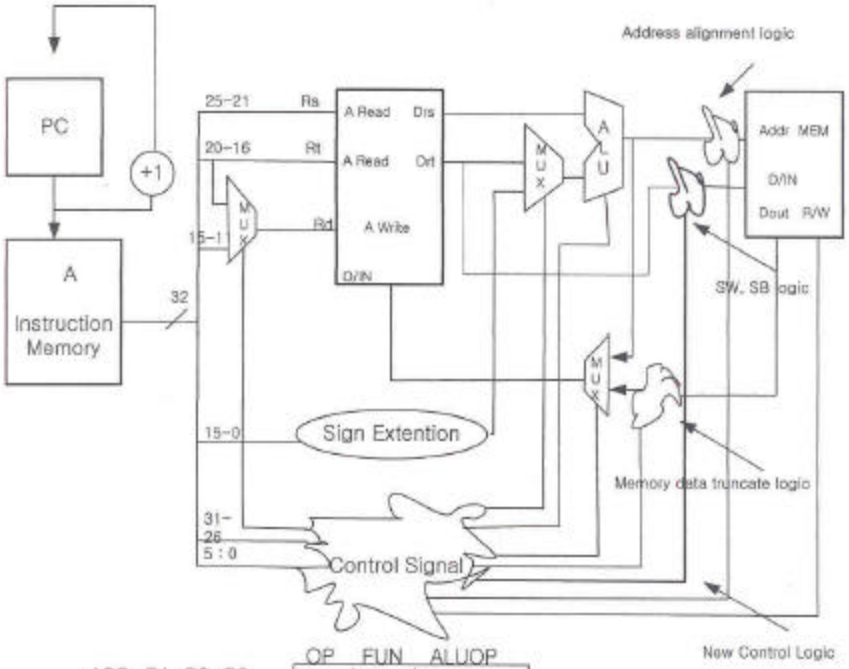
Do we need extra datapath to perforrm Slt?

Do we need extra HW to perform Slt?

Can the above H/W perform "SLTI"?

Slti R4, R5, 6
Slti ~~rt~~ rs, imm
rt



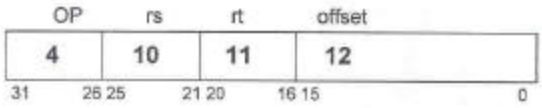


	OP	FUN	ALUOP
ADD R1, R2, R3	0	20	ADD
SUB R4, R5, R6	0	22	SUB
ADDi R7, R8, 5	8	X	ADD
OR R9, R10, R11	0	25	OR
LW R7, 8(R9)	23	X	ADD
LB R1, 2(R3)	20	X	ADD
SW R10, 11(R12)	2b	X	ADD
SB R10, 11(R12)	28	X	ADD
SLT R1, R2, R3	0	2a	~~SUB~~ SLT

ox1000: BEQ R10, R11,12

What will be the target address?
How to compute the target address?

BEQ rs, rt, label



What kind of action do we need?

a) compute the target address
b) check the branch condition (rs=rt)

Action a) needs Addition
Action b) needs Addition [Subtraction]

Target Address

PC +4 + offset *4

[12] 1100

0x1004 [12*4] 11 0000

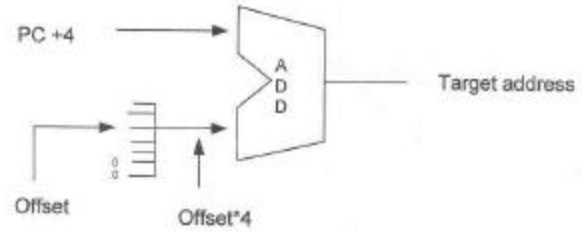
ox3 0

1004

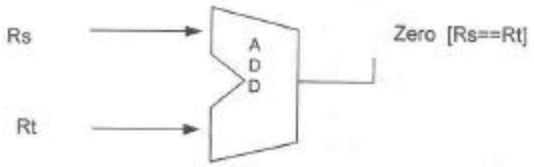
\+ 30

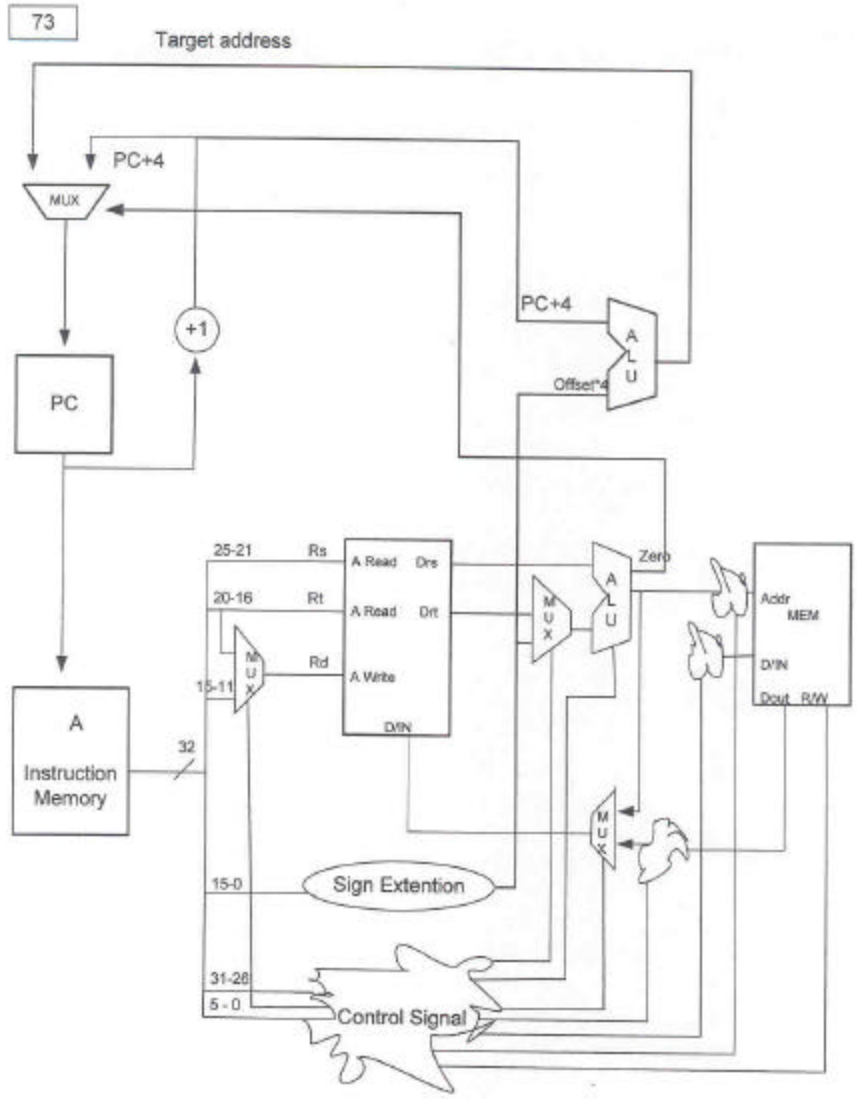
ox1034 [Target address]

a)



b)





Target Address Computation logic: Added
Branch condition logic : Added
Multiplexor for PC+4 and Target address : Added

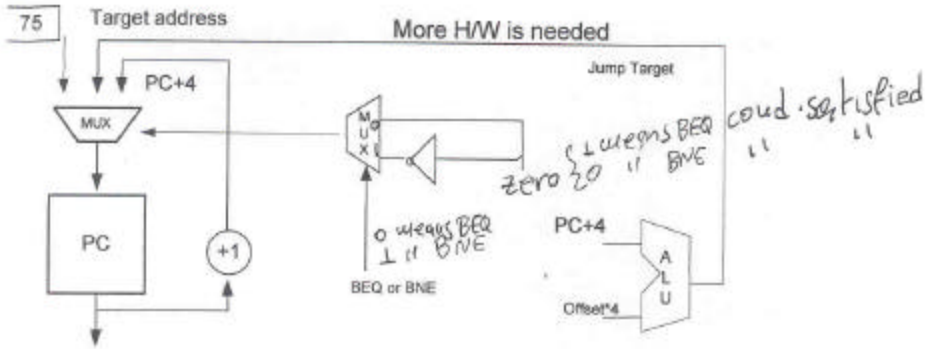
What's wrong with this?(What if an instruction performs subtraction and the result is zero; what will be the next address?

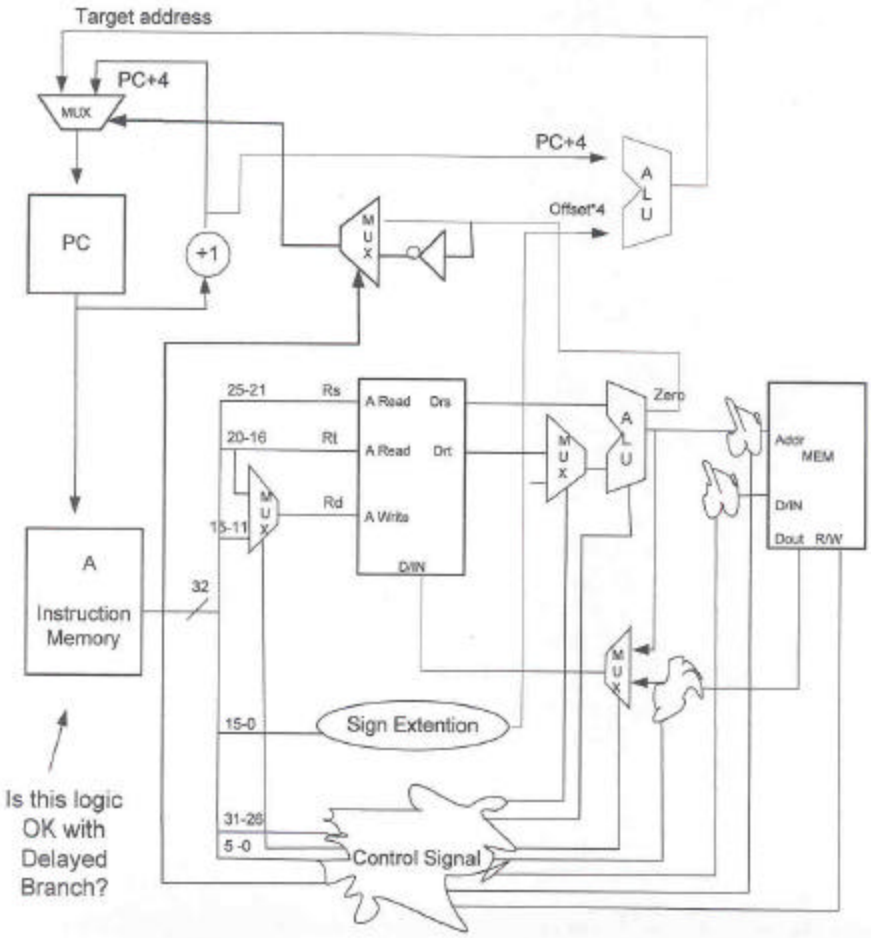
	OP	FUN	ALUOP
ADD R1, R2, R3	0	20	ADD
SUB R4, R5, R6	0	22	SUB
ADDi R7, R8, 5	8	X	ADD
OR R9, R10, R11	0	25	OR
LW R7, 8(R9)	23	X	ADD
LB R1, 2(R3)	20	X	ADD
SW R10, 11(R12)	2b	X	ADD
SB R10, 11(R12)	28	X	ADD
SLT R1, R2, R3	0	2a	~~SUB~~ SLT
SLTi R4, R5, 6	a	X	~~SUB~~ SLT
BEQ R10, R11, 12	4	X	SUB

Can the previous page H/W perform "BNE"?

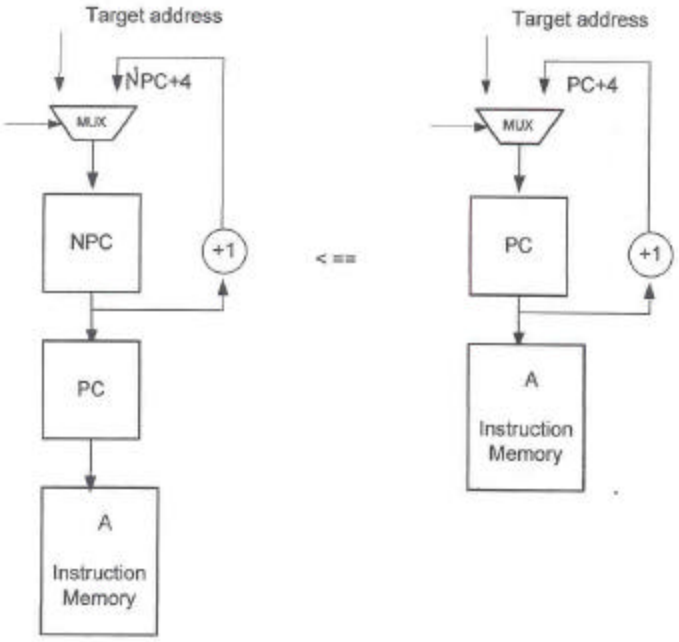
a) computing target address

b) computing branch condition

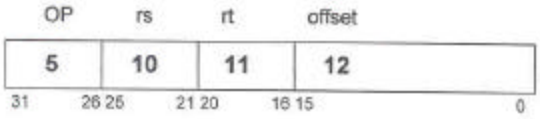

Target address
More H/W is needed
Jump Target
PC+4
MUX
PC
+1
BEQ or BNE
PC+4
Offset*4
ALU
zero
could satisfied


Target address
PC+4
MUX
PC
+1
PC+4
Offset*4
ALU
25-21
Rs
20-16
Rt
Rd
15-11
A Read
A Read
A Write
Drs
Drt
D/IN
Zero
Ador
MEM
D/IN
Dout
R/W
A
Instruction
Memory
32
15-0
Sign Extention
31-26
5-0
Control Signal
Is this logic
OK with
Delayed
Branch?

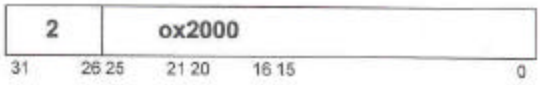
If branch condition is true, the next instruction to be executed is the branch target, instead of PC+4.
So, we need to change the H/W



BNE R10, R11, 12
BNE Rs, Rt, label



JUMP
J target j ox2000



How to compute the target address?

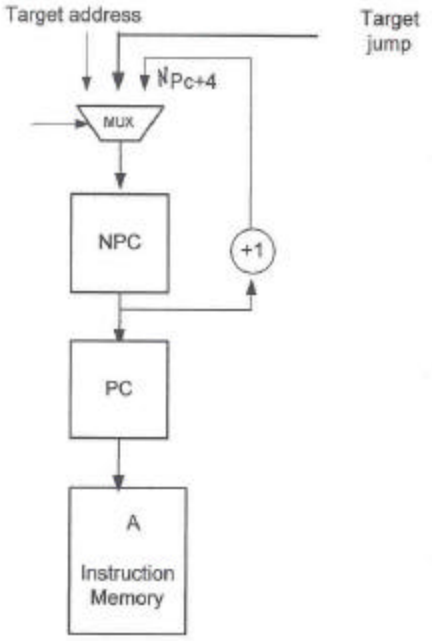
a) Address computation logic is needed.

Target address computation

{ pc [31:28], target, 00}

10000000 + 2000*4

=10008000 target address



Target address computation

{ pc [31:28], target, 00}

10000000 + 2000*4

=10008000 target address

0x10000000 jr 0x2000

0001 0000 0000 0000 0000 0000 0000 0000 (pc)

take 0001 (pc[31:28) 4bits

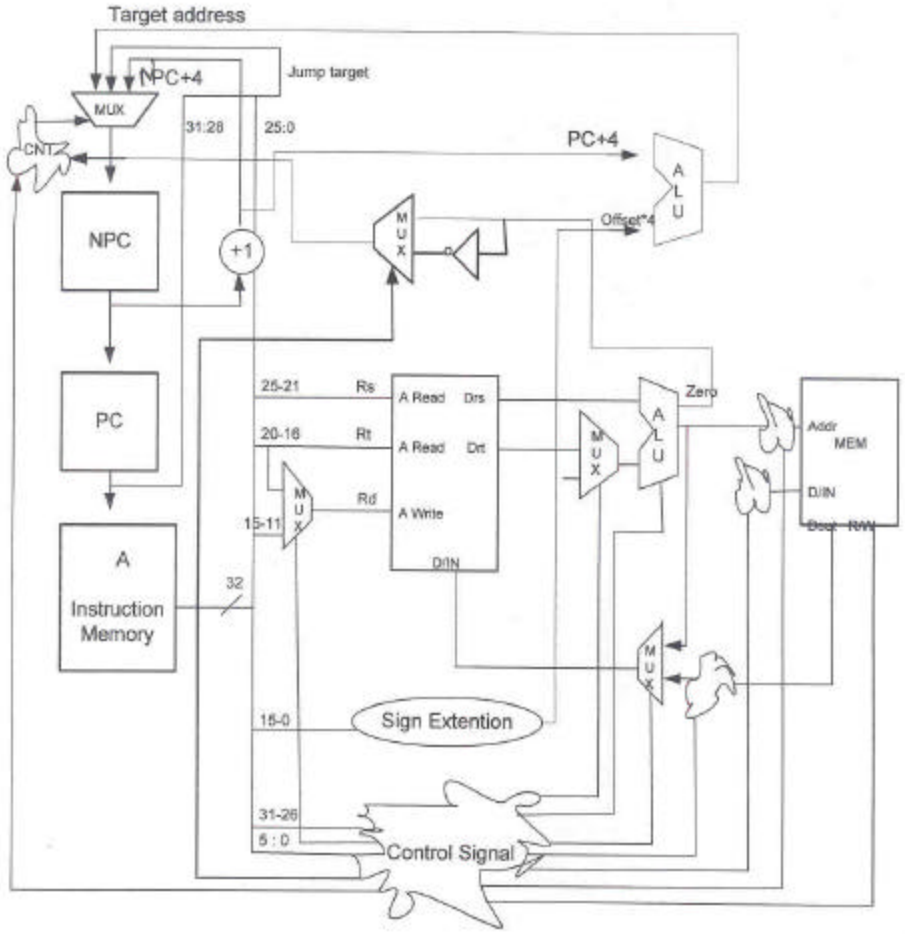
take 00 0000 0000 0010 0000 0000 0000 (target) 26bits.

0001 00 0000 0000 0010 0000 0000 0000 00(padding 2 zeros)

0001 00 0000 0000 0010 0000 0000 0000 00

0001 0000 0000 0000 1000 0000 0000 0000 (rearrange)

0X 1 0 0 0 8 0 0 0 (target address)

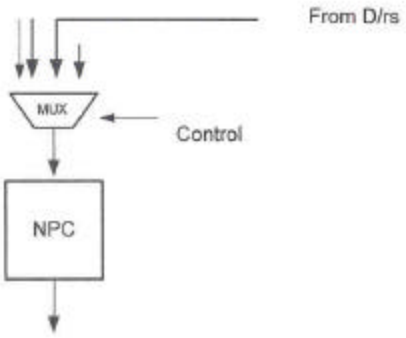


JUMP

"Can the above H/W perform jr R10?"

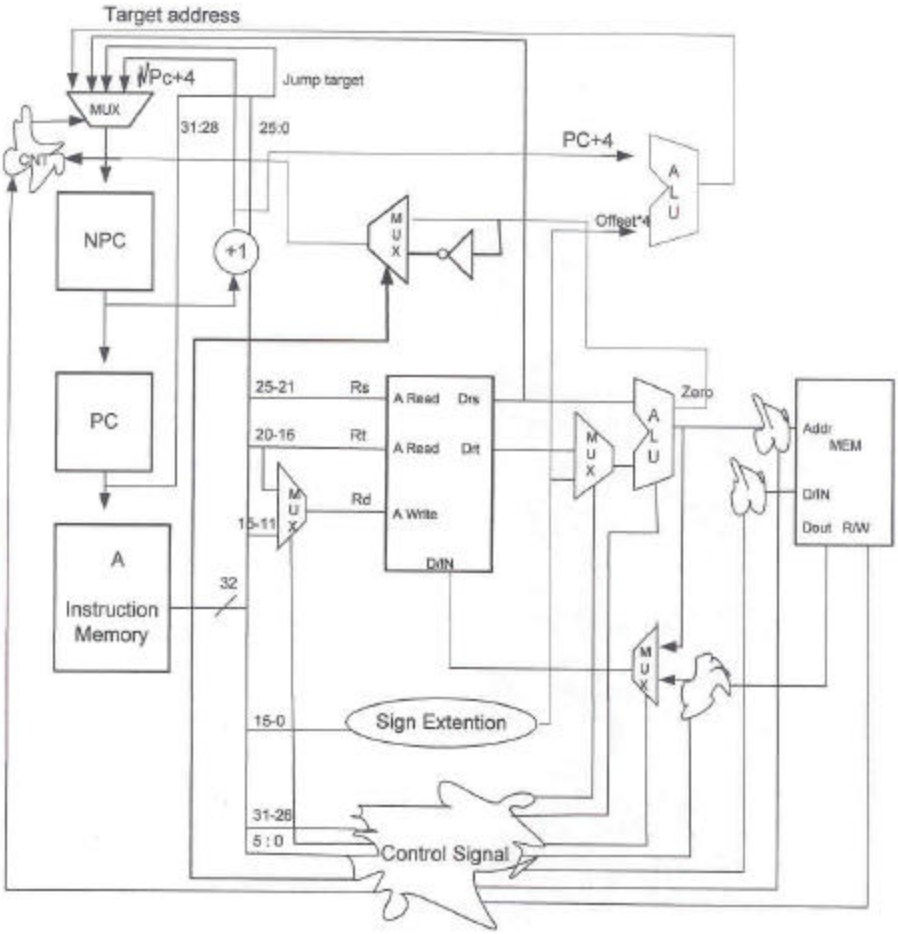
Jr R10;
jr Rs

Target address is in R10.
So we need a datapath from D/rs to NPC.



The "Control" comes from

bit 31-26 and bit 5-0



"Jr Rs"

If we want to perform jal, what do we need more?

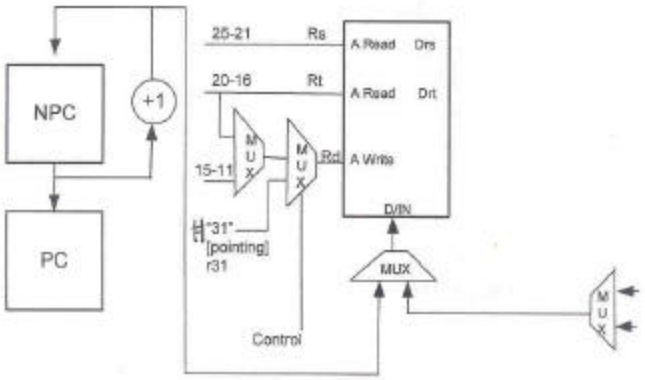
	OP	FUN	ALUOP
ADD R1, R2, R3	0	20	ADD
SUB R4, R5, R6	0	22	SUB
ADDi R7, R8, 5	8	X	ADD
OR R9, R10, R11	0	25	OR
LW R7, 8(R9)	23	X	ADD
LB R1, 2(R3)	20	X	ADD
SW R10, 11(R12)	2b	X	ADD
SB R10, 11(R12)	28	X	ADD
SLT R1, R2, R3	0	2a	SLT
SLTi R4, R5, 6	a	X	SLT
BEQ R10, R11, 12	4	X	SUB
BNE	5	X	SUB
J 2000	2	X	X
JR R10	0	8	X

For Jal instruction, everything is the same as j instruction, except, we have to save ~~NPC~~ to "R31".

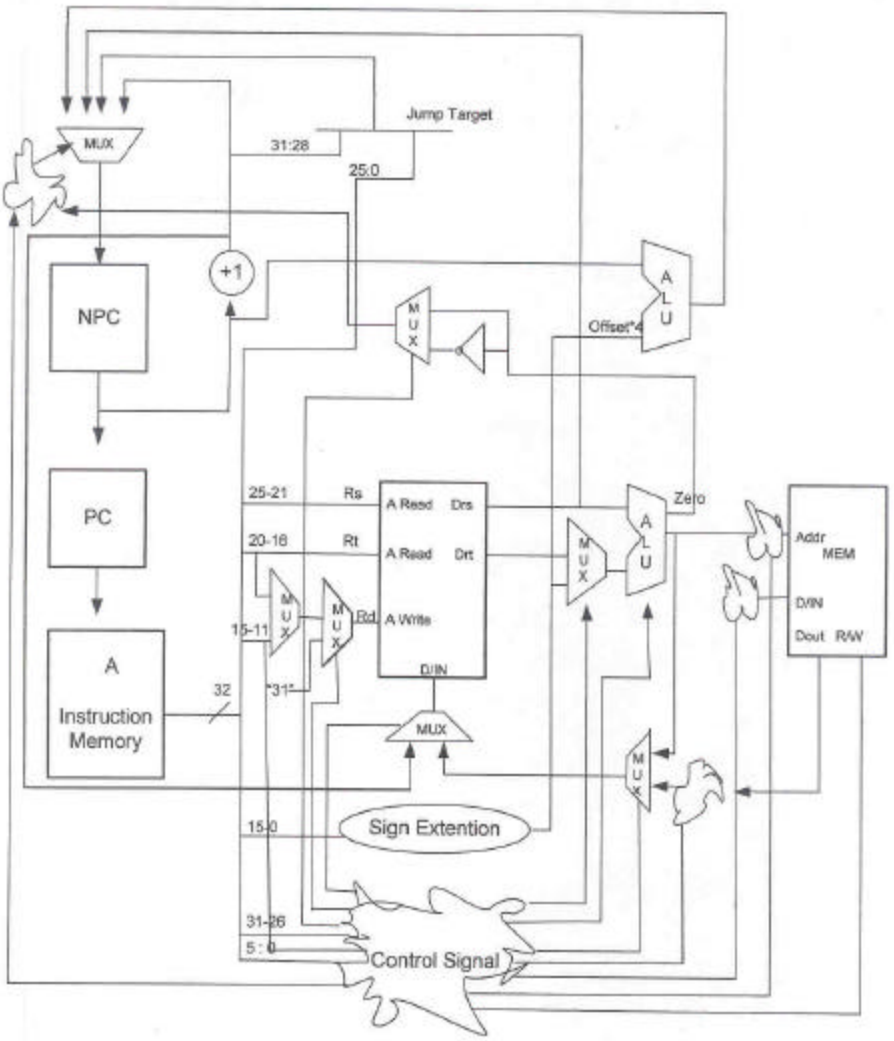
→ PC+8 or NPC+4 to R31

Jal ox2000

So we need data path to D/In from NPC and datapath to Awrite to point ra (r31)

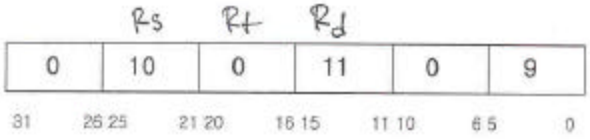


This way, we could save PC+8 to ra.
The control and CNT signals can be coming from Control logic.



"jal ox2000"
"Added HW for saving R31"

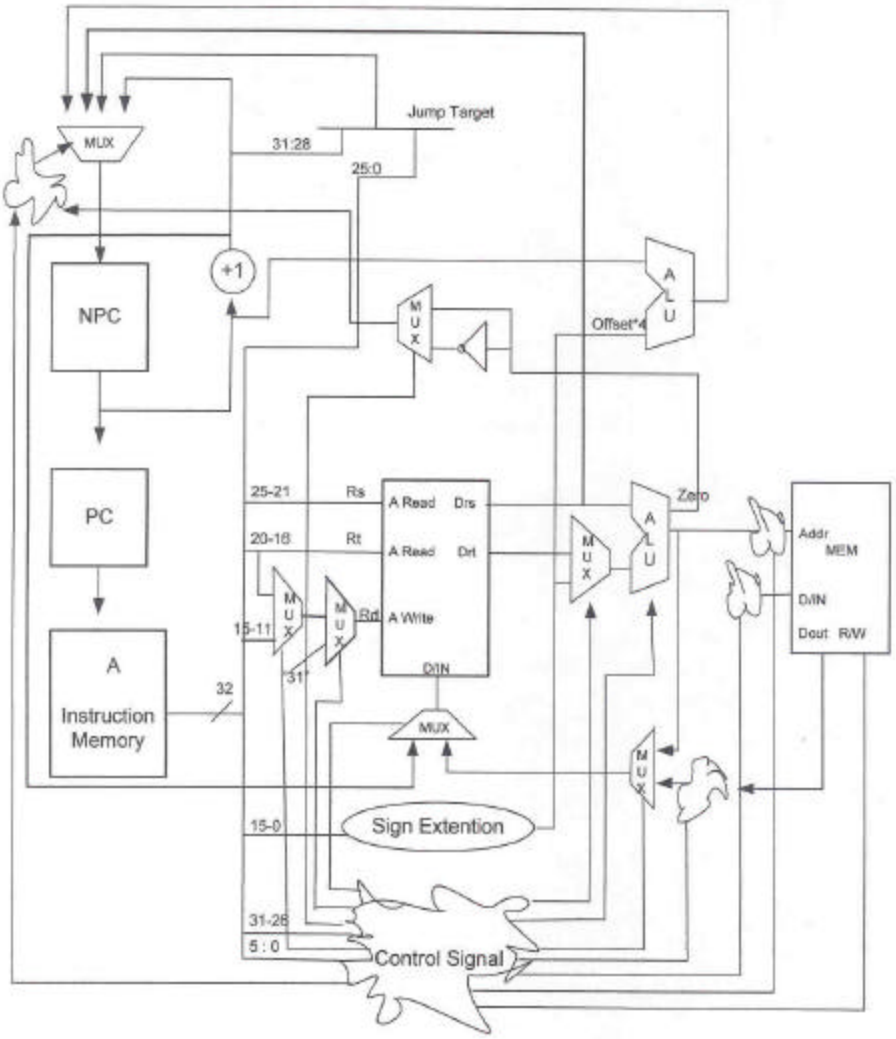
"Jalr Rs, Rd" Instruction.
Everything is same to jr rs instruction, except we have to save the return address to "rd".



Jalr R10, R11

Do we need new datapath? NO

NO



"jalr R10,R11
We don't need new datapath