Inst	RegDst	AluSrc	MemtoReg	RegWr	MemRd	MemWr	Branch	Aluop	Not	Function Code
R-type	1	0	0	1	0	0		111		
lw	0	1	1	1	1	0		101		
sw	X	1	X	0	0	1				
beq	X	0	X	0	0	0	1			
Add(R)	1	0	0	1	0	0		Ctr: 101		000010
Sub(R)	1	0	0	1	0	0		Ctr: 110		000011
Addi	1	1	0	1	0	0				
Subi	1	1	0	1	0	0				
And(R)	1	0	0	1	0	0		Ctr: 000		000100
Or(R)	1	0	0	1	0	0		Ctr: 001		000101
Andi	1	1	0	1	0	0				
Ori	1	1	0	1	0	0				
Lb	0	1	1	1	1	0?		101(add)		
Sb	X	1	X	0	0	1				
Slt(R)	1	0	0	1	0	0		Ctr: 100		000111
Slti	1	1	0	1	0	0				
Bne	x	0	x	0	0	0	1		*1	
Jump	x	x	x	x	0	0	x	x	?	
Jal				1		0				
Jr (R)										
move	0			1						

Dipnot:

1 = bne için ayrı bir sinyal gerekli.

Not:

Lb'de lb $1, 100($2) şeklindedir. Rt'de register adresi taşır. O regster'dan değer alır. Aldığı değeri alu'ya sokmadan data memory'nin adress kısmına taşır. O adresteki word'e bakılır. İmm16 kısmında ise offset taşınır. Bu yanlış değil mi? Memory gibi bir şey için 5 biti mi adres'e ayıracak? 16 biti ayırması daha mantıklı gibi. Mantıksız değil 5 bitlik kısım register'ın adresini temsil ediyor. Register'dan ise 32 bitlik şey alınıyor.

Şimdi anlaşıldı. Yerler ters olduğu için kafam karışıyordu

lw $1, 100($2)

Burada $1'i rs sanıyordum ancak o rt.

$1 = rt

$2 = rs

100 = imm16

Rs ile imm16'ı topluyoruz.

Load byte için datapath'te eksik var.

Memory_block farklı olacak o zaman. Her satır 8 bit olmalı. Evet çünkü bir sonraki word'ü okumak için bile offset'e 4 yazıyorduk.

Jump komutlarında nasıl bir çıktı vermeliyiz?

Jump için ayrı bir mux ekleyeceğiz

-jal'de ise sadece pc+4 değerini register'da bir yere koymamız gerekiyor. 32. Register'a koyarız. Bunun için de register_block'ta write_register önüne ve write_data önüne 1 mux daha ekleyeceğiz ki bu da jump'a bağlı olacak. Ayrıca register'a write komutu verilecek sadece

-jr için ise önce register'da 32. Register okunacak. Bunun için read register1 önüne bir mux eklenebilir. Okunan değer read data'da çıktığında o değeri pc'a göndermeliyiz.

- jump ve jal arasında sıkıntı çıkmayacaktır ancak jr sıkıntı çıkaracaktır. jr'yi mips'te hallet diyor. Şimdilik jr'yi düşünmeyelim. Jr zaten R type imiş enterasan.

Instruction binary:

R type: instr $rd, $rs, $rt

I type: instr $rt, $rs, imm

Instruction	Binary
add $1,$2,$3	000000 00010 00011 00001 00000 000010
sub $5,$1,$4	000000 00001 00100 00101 00000 000011
addi $6,$5,100	000010 00101 00110 0000000001100100
subi $8,$7,100	000011 00111 01000 0000000001100100

and $11,$9,$10	000000 01001 01010 01011 00000 000100
or $14,$12,$13	000000 01100 01101 01110 00000 000101
andi $15,$6,100	000100 00110 01111 0000000001100100
lw $16,0($0)	001000 00000 10000 0000000000000000
sw $8,0($4) //bunun rs'i $8 galiba. YOK!	~~010000 01000 00100 0000000000000000~~ 010000 00100 01000 0000000000000000
lb $17,0($4)	001001 00100 10001 0000000000000000
sb $1,0($5) Bunda da $1 rt o zaman	~~010001 00001 01001 0000000000000000~~ 010001 00101 00001 0000000000000000
slt $18,$1,$2	000000 00001 00010 10010 00000 000111
slti $19,$1,100	000111 00001 10011 0000000001100100
slti $20,$1,0	000111 00001 10100 0000000000000000
slti $21,$1, -10	000111 00001 10101 1111111111110110
Sw ile farklı offset	
slti $22,$8, -10	000111 01000 10110 1111111111110110
lw $23,4($0)	001000 00000 10111 0000000000000100
sw $24,13($4)	010000 00100 11000 0000000000001101
lb $25,3($4)	001001 00100 11001 0000000000000011
sb $26,5($5)	010001 00101 11010 0000000000000101
move $27, $8	100000 01000 11011 0000000000000000
Branch ve jumplar	
beq $1,$2,5 (bunda rs ve rt farketmez galiba)	100011 00001 00010 0000000000000101
beq $30,$31, -5	100011 11110 11111 1111111111111011
bne $1,$2, -5	100111 00001 00010 1111111111111011
bne $30,$31,-5	100111 11110 11111 1111111111111011
j 5	111000 00000000000000000000000101
jal 5	111001 00000000000000000000000101
jr $1	000000 00001 00000 00000 00000 001000

-veri hafızada nasıl tutulmalı? Lsb en üstte mi olmalı? Ben öyle yapıyorum.

- beq bambaşka bir sistem. Yaptığı şey şu, karşılaştırdığı 2 şey eşit ise offsetindeki kadar ileri ya da geri gider. E neden 2 kere sola kaydırıyoruz?