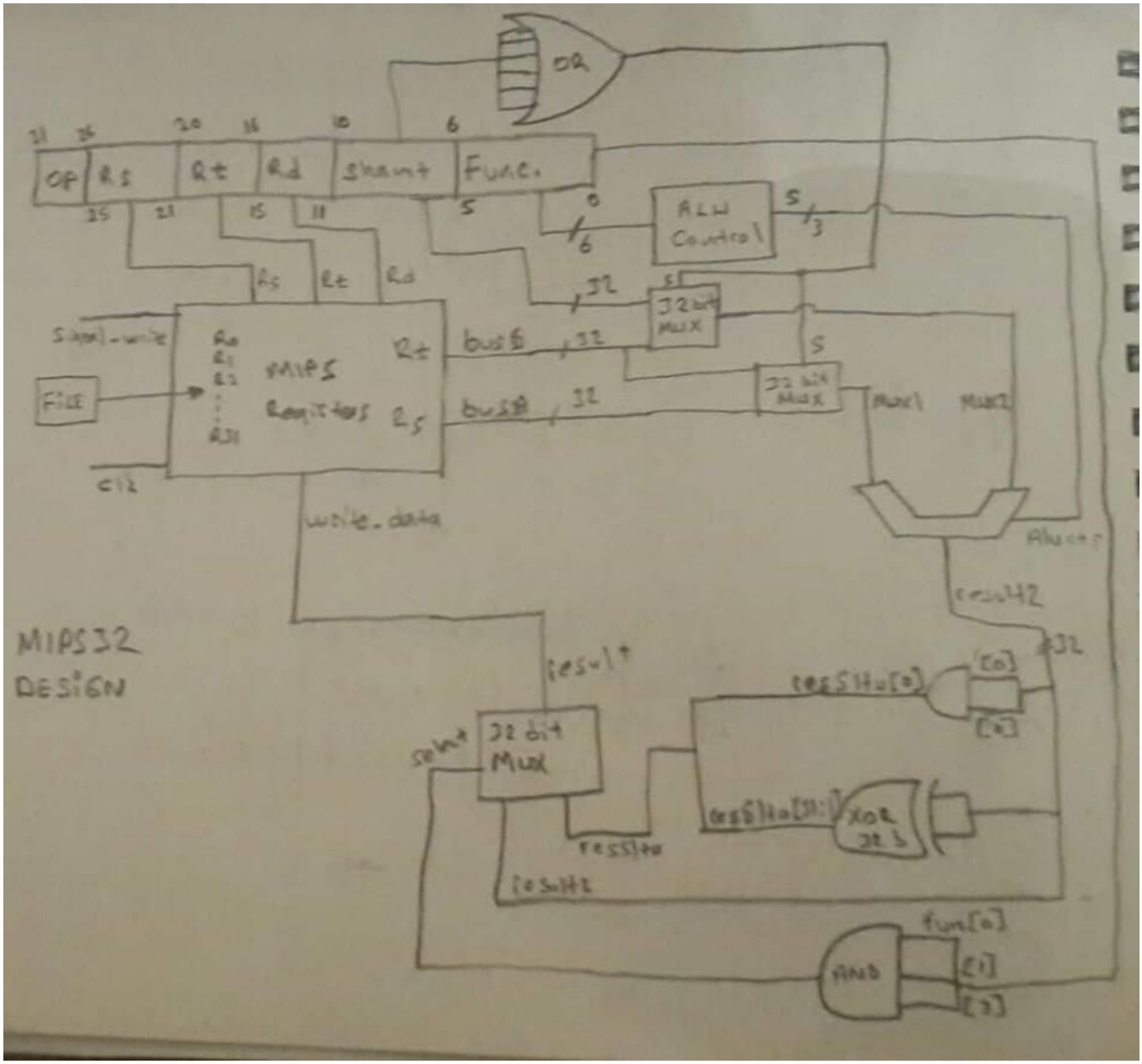
HOMEWORK 3 – 151044085 – EMRE KAVAK

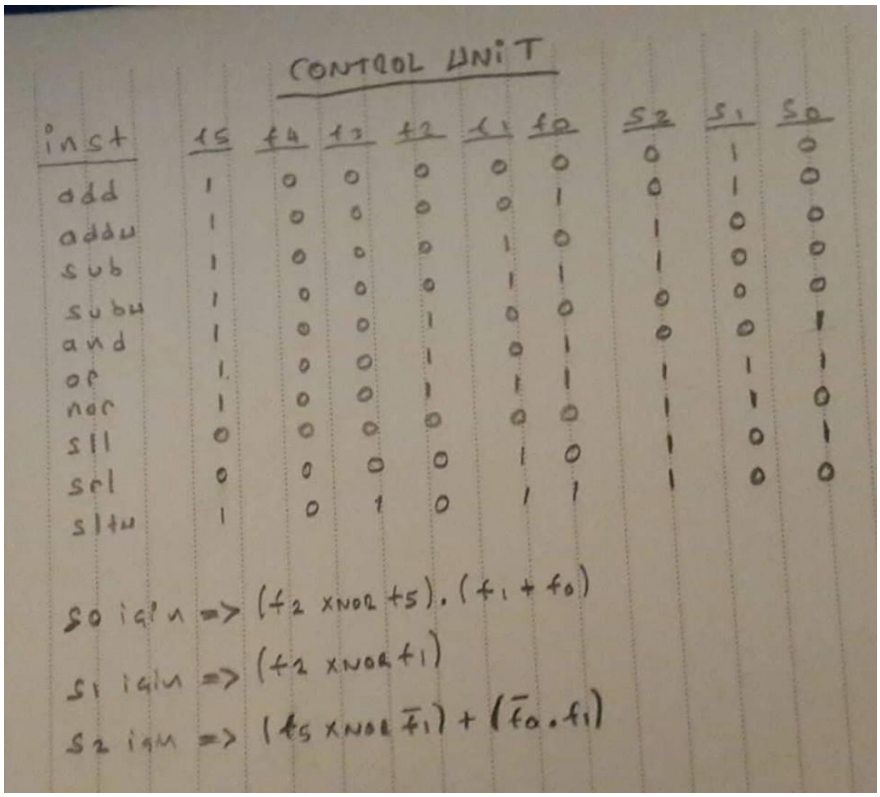
MODÜLLER

1) **mips32 :** Bu modülde parametre olarak alınan instruction sırasıyla **opcode, rs, rt, rd, shamt ve function code** bitlerine bölünerek değişkenlere atanıyor. Shift yaparken kullanılacak olan 5 bit **shamt** değişkeni, 32 bit olarak (**shamt32**) yeniden ataması yapılıyor. Function code or lanarak shift işleminin yapılıp yapılmayacağı belirleniyor. Bu or işleminin sonucu **selectShamt** değişkenine atanıyor. Bu değişkenler öncelikle **mips_registers modülüne** gönderilerek ,(registerların contenleri yükleniyor; **register.mem** dosyasından) modül içinde contentleri atanıyor.Bu aşamada register a yazma olmayacağı için sadece okuma yapması için **mips_register** modülünün **signal_reg_write** parametresine 1'b0, **clk** parametresine de 1'b1 gönderiliyor. Daha sonra control ünite **function code** gönderiliyor ve alınan sonuç **select** değişkenine atanıyor. Bu işlemden sonra, shifter durumunda **rt** yerine **shampt** ve **rs** yerine de **rt** gönderilmesini seçmek için **32bitlikmux** a bu değişkenler **selectShamt** select biti olmak üzere gönderiliyor. Muxların sonuçlarına göre **alu32** modulüne **rs** ve **rt** (veya **rt - shamt**) gönderilerek işlemlerin sonucu **result** değişkenine gönderiliyor. Daha sonra, Seçilen işlemin **sltu** olup olmadığını anlamak için function kodun 0-1-3 bitleri and işlemine giriyor ve oluşan **result** değişkeni **and** ve **xor** işlemlerine girerek **sltu** işleminin sonucunu oluşturuyor. Son olarak, yine **sltu** işlemimi yoksa diğer işlemler mi seçilmiş anlamak için **32bitmux** kullanılıyor ve seçilen sonuç, rd registerına yazmak için **mips_register** modülüne gönderiliyor.

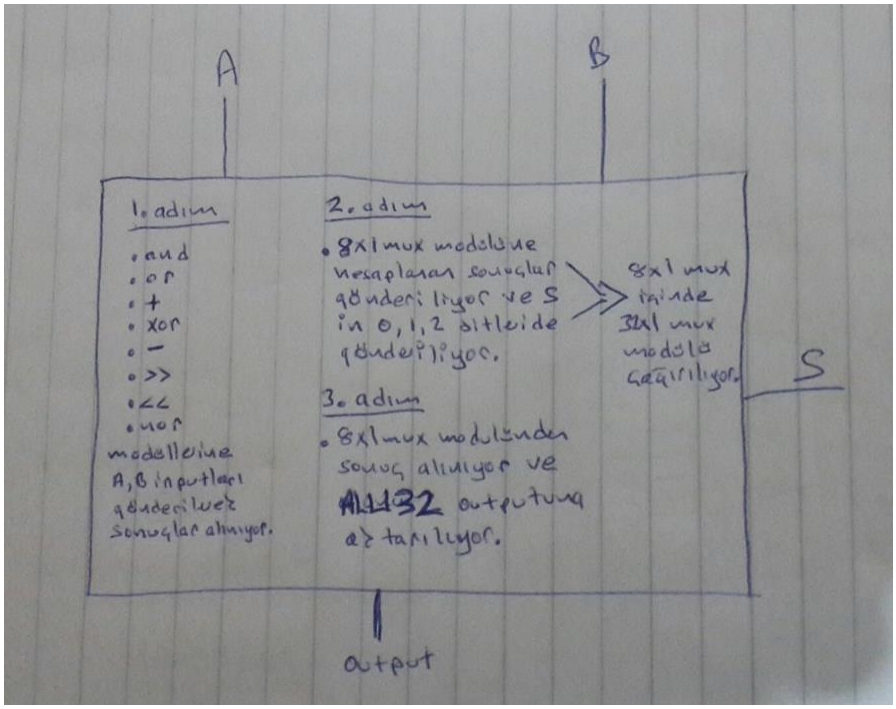


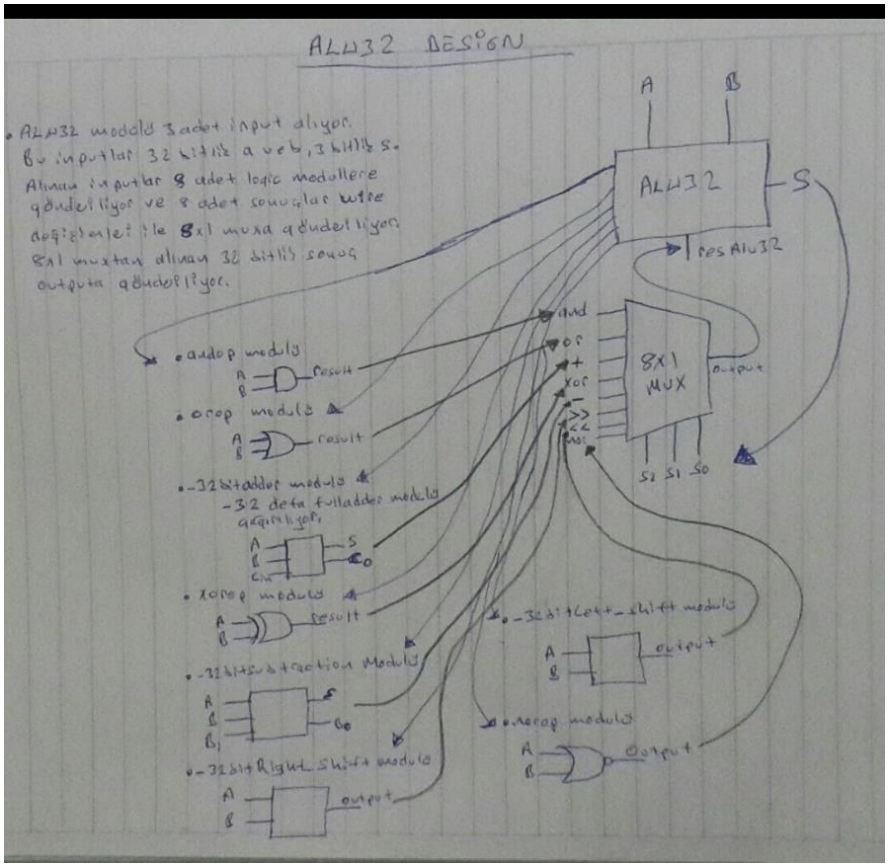
2) **mips_registers :** Bu modül içinde, register.mem dosyası okunuyor, 32 bitlik registers contentleri okunan bu dosyanın içeriği ile dolduruluyor. Parametre olarak alınan R type- instruction field leri bu modül içinde ataması yapılıyor. (örneğin aluya gönderilecek olan rs-rt parametrelerinin contenti burada atanıyor.) Daha sonra signal_reg_write sinyali kontrol edilerek; eğer 1 ise rd registerın'a yazma işlemi yapılıyor, eğer değilse yazma işlemi yapılmıyor. Ayrıca register0 a yazılma işlemi yapılmaması kontrolüde burada sağlanıyor.

3) **control_unit :** Bu modül içinde, parametre olarak alınan function code a göre, sadeleştirme işlemi sonucu oluşan logic gateler ile alunun select biti ne olacağı üretiliyor. 3 bit olan alunun hangi işlemin yapılcağını seçen değişkeni bu modülde belirlenmiş oluyor.



4) **alu32 :** Bir önceki proje ile hemen hemen aynı modül olup, xor ve sra işlemleri kaldırılmış, yerine srl ve sltu işlemleri eklenmiştir. Bu modülü hatırlayacak olursak, 32 bitlik 2 adet **a** ve **b**, 3 bitlik **s(select)** inputları alınmaktadır. Bu inputlar alındıktan sonra modül içinde yapmamız istenen **8 adet(and,or,+ , -,>>,<<,nor)** işlemin modülleri çağırılmaktadır. Çağırılan bu modüllerin sonuçları **wire** değişkenlerinde tutularak , **_8x1mux** modülüne gönderilmektedir. _8x1 modülü 8 adet 32 bitlik , 1 adet 3 bitlik input almaktadır. Alınan bu inputlar **_32bitmux** modülüne, sırasıyla I0,I1…I7, şeklinde gönderiliyor ve s bitinin sırasıyla s[0], s[1] ve s[2] bitleri gönderilerek hangi inputun outputa gönderileceği belirleniyor. _8x1mux modülünde en son çağırılan _32bitmux modülünün outputu alınarak, _8x1mux modülünün outputuna gönderiliyor. Oluşan bu output, son olarakda alu32 icinde ki resultAlu32 outputuna atanarak istenilen işlemin sonucu elde edilmiş oluyor.


A
B
1. adım
• and
• or
• +
• xor
• −
• >>
• <<
• nor
modellerine
A,B inputları
gönderilerek
sonuçlar alınıyor.
2. adım
• 8x1 mux modülüne hesaplanan sonuçlar gönderiliyor ve S için 0,1,2 bitleride gönderiliyor.
8x1 mux içinde 32x1 mux modülü çağırılıyor.
3. adım
• 8x1 mux modülünden sonuç alınıyor ve ALU32 outputuna atanıyor.
S
output


ALU32 DESIGN
• ALU32 modülü 3 adet input alıyor. Bu inputlar 32 bitlik a ve b, 3 bitlik s. Alınan inputlar 8 adet logic modüllere gönderiliyor ve 8 adet sonuçlar wire değişkenleri ile 8x1 muxa gönderiliyor. 8x1 muxtan alınan 32 bitlik sonuç outputa gönderiliyor.
A
B
ALU32
S
resALU32
and
or
+
xor
−
>>
<<
nor
8x1 MUX
output
S2 S1 S0
• and op modülü
result
• or op modülü
result
• 32 bit adder modülü
-32 defa fulladder modülü çağırılıyor.
S
Co
Cin
• xor op modülü
result
• 32 bit Subtraction Modülü
Bo
B
• 32 bit Right Shift modülü
output
• 32 bit Left Shift modülü
output
• nor op modülü
Output

Register.mem dosyası ilk hali :

00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000000110
00000000000000000000000000000111
00000000000000000000000000001111
00000000000000000000000011110000
00000000000000000000111100000000
00000000000000000001000000000000
00000000000000000000000000100000
00000000000000000000000000001001
00000000000000000000000000001100
00000000000000000000000001100000
00000000000000000000000000010000
00000000000000000000000000010001
00000000000000000000000000010010
00000000000000000000000000010011
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000000011100
00000000000000000000000000011101
00000000000000000000000000011110
00000000000000000000000000011111

İnstruction işlemlerinin sonuçunda rd contentleri :

opcode = 000000 | rs = 00100 | rt = 00101 | rd = 00110 | shampt = 00000 | func = 100000
 # New rd =00000000000000000000000000001001
#
opcode = 000000 | rs = 00110 | rt = 00111 | rd = 01000 | shampt = 00000 | func = 100001
 # New rd =00000000000000000000000000010000
#
opcode = 000000 | rs = 00111 | rt = 01000 | rd = 01001 | shampt = 00000 | func = 100010
 # New rd =11111111111111111111111111110111
#
opcode = 000000 | rs = 01001 | rt = 01000 | rd = 01010 | shampt = 00000 | func = 100011
 # New rd =11111111111111111111111111100111
#
opcode = 000000 | rs = 01001 | rt = 01000 | rd = 01011 | shampt = 00000 | func = 100100
 # New rd =00000000000000000000000000010000
#
opcode = 000000 | rs = 01001 | rt = 01000 | rd = 01100 | shampt = 00000 | func = 100101
 # New rd =11111111111111111111111111110111
#
opcode = 000000 | rs = 01001 | rt = 01000 | rd = 01101 | shampt = 00000 | func = 100111
 # New rd =00000000000000000000000000001000

#
opcode = 000000 | rs = 00000 | rt = 01001 | rd = 01110 | shampt = 00001 | func = 000000
 # New rd =11111111111111111111111111101110
#
opcode = 000000 | rs = 00000 | rt = 01001 | rd = 01111 | shampt = 00011 | func = 000010
 # New rd =00011111111111111111111111111110
#
opcode = 000000 | rs = 01110 | rt = 01111 | rd = 10000 | shampt = 00000 | func = 101011
 # New rd =00000000000000000000000000000001

Register.mem dosyası son hali : (değişen contentler koyu renk ile belirtilmiştir)

00000000000000000000000000000000
00000000000000000000000000000001
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
00000000000000000000000000001001
00000000000000000000000000000111
00000000000000000000000000010000
11111111111111111111111111110111
11111111111111111111111111100111
00000000000000000000000000010000
11111111111111111111111111110111
00000000000000000000000000001000
11111111111111111111111111101110
00011111111111111111111111111110
00000000000000000000000000000001
00000000000000000000000000010001
00000000000000000000000000010010
00000000000000000000000000010011
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000000011100
00000000000000000000000000011101
00000000000000000000000000011110
00000000000000000000000000011111