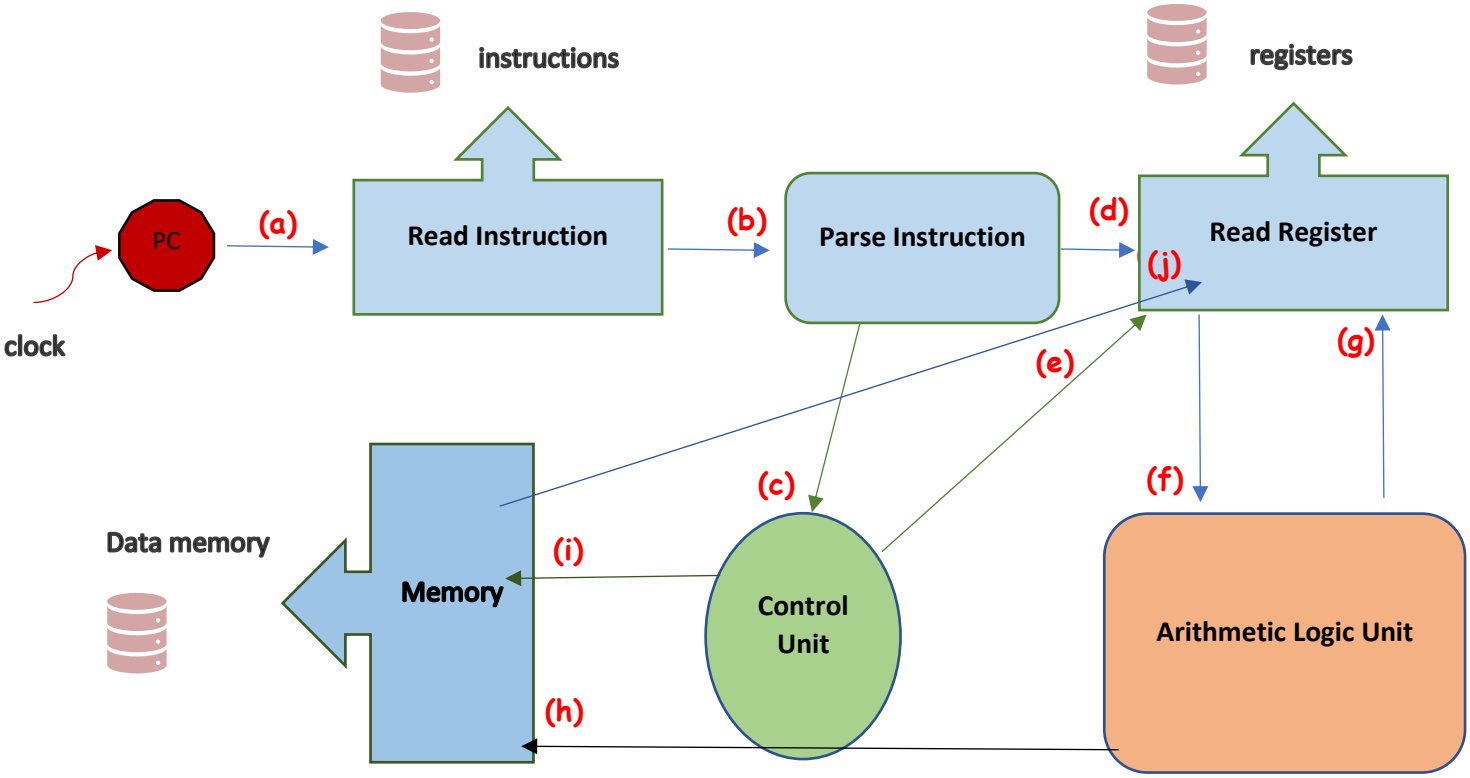
1. INTRODUCTION

1.1 Big Picture



1.2 Lifecycle

(a) Clock tarafından tetiklenen program counter, instruction okuyan read_instructions modülünden, sıradaki instruction'ı almamıza yardımcı olur. Instruction okuyan blok, program başladığında dosyadan programı yani instructionları okur ve daha sonra içerisinde tuttuğu arrayde değişiklikler yaparak kullanır.

(b) Instruction'ı tipine göre; opcode, rs, rt, rd, shift amount, function, address ve immeadiate olarak bölecek olan modül inst_parser'dır. Instruction okuyan modül sayesinde, instructionda oluşan değişikliğe göre tetiklenir ve parse işlemine başlar.

(c) opcode, ve function code değişince control_unit modülünün kontrol ettiği tüm sinyaller sıfırlanır. Ve yeni opcode ya da function değerine göre tekrar değer alırlar. Bunun sayesinda "registerdan okuma yapılacak mı?", "registera yazma yapılacak mı?", "memoryden okuma yapılacak mı?", "memorye yazma yapılacak mı?" gibi sorular cevap bulur.

(d) Eğer rs ve rt'de bir değişiklik olmuşsa register okumak için olan read_registers modülü tetiklenir. Ancak (e) olarak gösterilen ilişkiden okuma sinyali gelmezse okuma yapılmaz.

(e) Register ile alakalı olan sinyaller, register write ve register read sinyalleri gönderilir.

(f) Eğer yapılacak işlem aritmetik bir işlemse, işlem için gerekli olan, rs register contenti, rt register conteti, shift amount gibi bilgiler ALU'ya gönderilir.

(g) İşlem sonucu oluşan result, registera geri gönderilir ve eğer (e) ilişkisinden gelen sinyal yazma sinyaliyle rd registerına bu değer yazılır.

(h) ALU'nun hesapladığı sonuç memory'den o adresi okumak ya da memory'deki o adrese yazmak için gönderilir.

(i) Control unit memory read ya da memory write yapılması ile ilgili sinyalleri ayarlar.

(j) Memoryden okuma yapılmışsa ve okunan değer de register'a yazılacaksa (lw) son aşama olarak bu işlem gerçekleştirilir. Memoryden alınan read_data yazılmak üzere register'a geri gönderilir.

1.3 Missing parts, some add informations

- Sc ve jal instructionları implement edilmedi.
- Modüller, sinyaller ve yapılan işlemler olabildiğince datapath örnek alınarak dizayn edildi.
- read_register ve read_Data_mem modüllerinin opcode alma sebebi, lbu ve lhu instructionlarının lw'den farkının fark edilmesi ve sb sh instructionlarının sw instruction'ından farkının fark edilmesi içindir.
- Jump komutunda address kısmından okunan değer 5, bu projede bu 5. Satır yerine, 5. index olarak kullanıldı. Instructionları tutan arrayde 5. İndex satır sayısı olarak 6. Satıra denk gelmektedir. Aynı işlem register ve memory için de bu şekilde uygulandı.
- Data.mem dosyasına bir kere yazma işlemi yapılırsa, dolu olmayan memory'ler için xxxx basabilmekte ve bu da tekrar çalışırken compile hatasına sebep olmaktadır. Xxx değerleri silindiğinde sorun ortadan kalkar.

2. METHOD

Proje için 7 adet modül kullandım.

1. Mips_core: Bütün datapath gibi düşünülebilir. Ya da bir projenin main fonksiyonu gibi. Çalışacak tüm modüller sırası ile çağırıldı. Onun haricinde burada program counter handle işlemleri yapıldı. Clock'un posedge hareketi ile tetiklenerek PC'nin değişmesi sağlandı. Jump, jr ve branch instructionları PC'yi değiştirdiği için onlara özel koşul yazıldı, onun haricinde program counter sadece 1 arttırılarak devam edildi.
2. read_instructions: Instruction okumaya yardımcı modül, program counter değiştikçe okuma yapar. Program ilk başladığında initial değerleri dosyadan çeker.
3. inst_parser: opcode yardımı ile instruction un hangi tip (R,J,I) olduğunu anlayarak ona göre pars eden bir modüldür.
4. control_unit: Datapath'teki olarak düşünülebilir. Registera yazılacak mı yazılmayacak mı, memoryde ne işlemi yapılacak, destination register hangisi olacak gibi problemleri ayarladığı sinyaller ile halletmesi sağlandı.
5. ALU32bit: 32 bit sayılarla Arithmetic işlemler yapan modül. Tüm instructionların yapması gerekenler burada halledildi.
6. read_data_memory: Datapath'teki data memory'e karşılık gelmektedir. ALU sonucunu address olarak, rt contentini ise write data olarak alır. Gelen sinyaller ile belirlenen işlemi yaptıktan sonra read data olarak output verir. Program ilk başladığında initial değerleri dosyadan çeker.
7. read_registers: register contentlerinin okunması ya da registera yazılma işleminin yapılması için kullanıldı. Program ilk başladığında initial değerleri dosyadan çeker.

3. RESULTS

Şu an burada gösterilmek üzere:

- 1 adet R type -> add
- 1 adet I type -> slti
- 1 adet brach -> beq
- 1 adet memory access -> lw

Yapan instructionlar test edilecektir.

Test case:

PC = 0. 000000/01000/01001/11111/00010/100000 $31 = $8 + $9

$8 = 00000000000000000000000000001111

$9 = 00000000000000000000000011110000

$31 = 00000000000000000000000011111111 olmalı.

PC = 1. 001010/11111/00101/0000010011111111 $5 = ($31 < immediate) ? 1: 0

Evet küçüktür, 5. Registera 1 yazılmalı.

PC = 2. 000100/00001/00101/0000000000000011 if($1 == $5), PC = PC + 1 + 3

$5 ve $1 registerlarında da 1 değeri vardır. PC = 2 + 1 + 3 olarak 6 değerini almalı.

PC = 6. 100011/01001/01000/0000000000001111 $8 = M[$9 + 18]

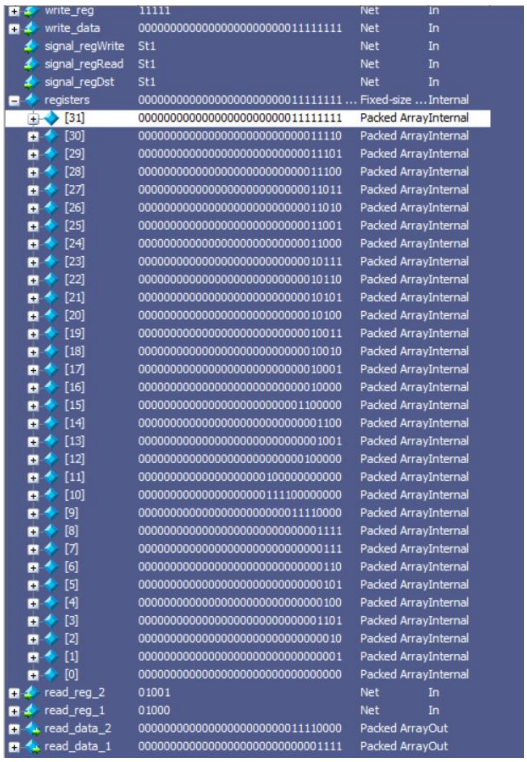
$9 = 00000000000000000000000011110000

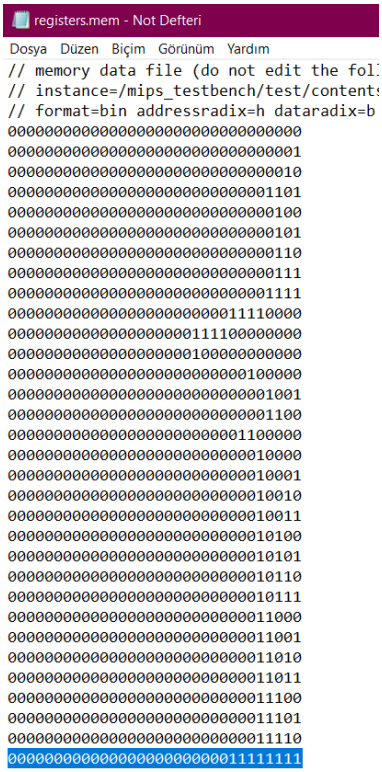
18 = 00000000000000000000000000001111

$9 = 00000000000000000000000011111111

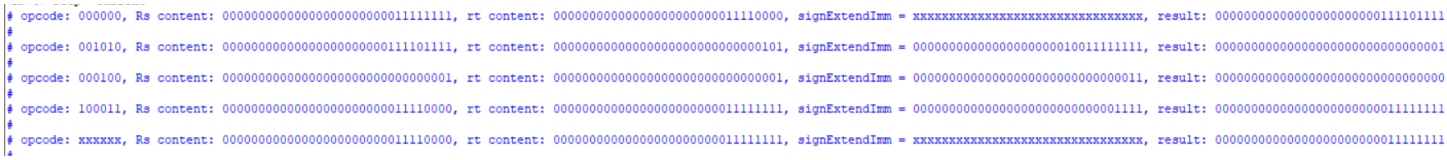
Ve şimdi.. sonuçlar diğer sayfadadır..

İlk instruction sonucu oluşan modelsim göstergeleri ve geri yazılan dosya çıktısı :





Süre sıkıntısından dolayı, devamını adım adım gösteremeyeceğim anca her adımda olan işleri şu şekilde özetleyebilirim:



En başta, R type olduğu için signExtended yoktur. En sonda da branch yapmış ve branch ile gönderilen en son instruction olduğu için daha fazla instruction okuyamamıştır.

Register değerleri diğer sayfadadır…

Ve program sonucundaki register değerleri:

registers.mem - Not Defteri

Dosya Düzen Biçim Görünüm Yardım

```
// memory data file (do not edit the following line - required for mem load use)
// instance=/mips_testbench/test/contents/registers
// format=bin addressradix=h dataradix=b version=1.0 wordsperline=1 noaddress
00000000000000000000000000000000
00000000000000000000000000000001 // 5. register ile aynı değere sahip
00000000000000000000000000000010
00000000000000000000000000001101
00000000000000000000000000000100
00000000000000000000000000000001 // değişmesi gereken 5. register
00000000000000000000000000000110
00000000000000000000000000000111
00000000000000000000000011111111 // son instruction ile değişen 8. register
00000000000000000000000011110000
00000000000000000000111100000000
00000000000000000001000000000000
00000000000000000000000000100000
00000000000000000000000000001001
00000000000000000000000000001100
00000000000000000000000001100000
00000000000000000000000000010000
00000000000000000000000000010001
00000000000000000000000000010010
00000000000000000000000000010011
00000000000000000000000000010100
00000000000000000000000000010101
00000000000000000000000000010110
00000000000000000000000000010111
00000000000000000000000000011000
00000000000000000000000000011001
00000000000000000000000000011010
00000000000000000000000000011011
00000000000000000000000000011100
00000000000000000000000000011101
00000000000000000000000000011110
00000000000000000000000111101111
```