ORGANİZASYON FİNAL PROJE

FURKAN AKTAŞ

141044029

Uyarı : Zip isminde ‘ş’ (aktaş) olduğundan klasöre çıkart denildiğinde quartus projeyi oluşturamıyor, .qar dosyası sadece ingilizce karakter içeren bir klasör içinde olmalı !!!

Tanıtım:

Proje single cycle olarak tasarlanmıştır. Ana dosya mips_core ’dur. mips_core ’da önce instruction ’dan elde edilen opcode control_signals fonksiyonuna gider ve buradan kontrol sinyalleri üretilir. control_signals single cycle data path ‘deki control unit ‘in aynısıdır.

Sinyaller üretildikten sonra, instruction jal ise 11111, değilse regDest sinyaline göre rd yada rt hangi register adresine yazılacağı belirlenir.

Daha sonra belirlenen registerlardan mips_registers fonksiyonu ile veriler okunur. Bu verilerden rs’ten okunan eğer gerekliyse her zaman kullanılıyor ama rt değeri yerine signExtend, zeroExtend vs. gelebilir , sonraki adımda bu belirlenir.

Daha sonra, control_signals ‘ten üretilen ALUop, ALU_Control_Sig ‘e verilir ve buradan yapılacak işleme göre ALU ‘ya gönderilmek için ALU_Control_Sig_out üretilir. Daha sonra bu sinyale göre ALU ‘da işlemler yapılır ve ALU_result sonucu üretilir.

Sonra sll ve srl işlemleri ‘de mips_core içinde yapılır ve bunların sonucu shifted adlı wire ‘da tutulur.

ALU_result sonucu mips_data_mem fonksiyonuna yollanır. Eğer memory ‘ye yazma yapılacaksa bu sonuç yazdırılır, memory ‘den okuma yapılacaksa bu sonuca göre memory ‘den okunur ve okunan değer döndürülür. Okunan değer lw, lbu ve lhu ya göre mips_core içinde düzenlenir ve data_mem_result adlı wire ‘a verilir.

Daha sonra register ‘a yazılacak değer belirlenir; eğer sll veya srl ise sihfted, jal ise PC+2, memory ‘den okuma yoksa(memtoReg =0) ALU ‘dan çıkan ALU_result, varsa (memtoReg =1) data_mem_result register ‘a yazmak üzere seçilir.

Daha sonra önce branch adress değeri hesaplanır ve eğer işlem branch ise branch_adress+1 değilse 1 b_or_pc wire ‘ına atanır. Sonra jump adress değeri hesaplanır.

Daha sonra PC ‘ye eklenecek olan değer belirlenir; eğer jr rs – PC, jal ve j ise jump_adress-PC, değilse b_or_pc seçilir ve add_part ‘a atanır.

Daha sonra eğer işlem jr ise, jr R type olduğundan reg write sinyali 1 olarak gelir bu sinyal 0 , eğer jr değilse instruction ‘a göre gelen reg write sinyali signal_reg_write ‘a atanır.Sonra bu sinyale göre registers yazma için tekrar çalıştırılır.

En son, PC = PC + add_part yapılarak bir sonraki instruction alınır.

Methodlar:

control_signals:

Data path ’deki control unit ‘in aynısıdır. İnput olarak opcode alır. opcode ‘a göre regDest, memtoReg, regWr, memRd, memWr, branch, ALUsrc, ALUop output ‘larını döndürür.

ALU_Control_Sig:

Data path ‘deki ALU control unit ile aynı mantıkla çalışır. ALU ‘nun yapacağı işlem için sinyal üretir. İnput olarak ALUop, func alır. func R type lar içindir. ALU_Control_Sig_out değerini output olarak döndürür.

ALU:

İşlemleri yapan birimdir. İnput olarak control, inp1, inp2 alır. control ALU_Control_Sig ‘den gelir, inp1 rs , inp2 rt signExtend vs. dir. Out yapılan işlemin sonucudur. Zero branch için üretilen sinyaldir.

mips_registers:

Register birimidir. İlk seferinde dosyadan registers a veriler okunur. İnput olarak write_data(yazılacak değer), read_reg_1(rs index’i), read_reg_2(rt index’i), write_reg(rd,rt,11111 index’i), signal_reg_write(yazmak için sinyal), clk alır. Output olarak read_data_1, read_data_2 döndürür. signal_reg_write 1 ise register ‘da değer değiştirilir ve dosyaya yazılır.

mips_data_mem:

Memory birimidir. İlk seferinde dosyadan data_mem ‘e veriler okunur. İnput olarak mem_address(yazılacak yada okunacak index), write_data(yazılacak değer), sig_mem_read(okuma sinyali), sig_mem_write(yazma sinyali), opcode(opcode sh sb için write_data ‘yı düzenlemek için kullanılır) alır. Output olarak read_data(memory ‘den okunan veri) döndürür. Her yazma işleminde data_mem değeri değişir ve dosyaya yazma yapılır.

mips_instr_mem:

İnput olarak program_counter alır ve buna göre instruction ‘ı output olarak döndürür.

mips_core:

Diğer tüm fonksiyonları kullanarak işlemleri yapar. Tanıtım kısmında geniş anlatıldı.

mips_testbench:

Sadece clock değerleri verilecek. Dosyalara yazma işlemleri mips_registers ve mips_data_mem içinde yapılır.

Sonuçlar:

instructions

```
1  00000000000000010000001110100000 // add  $0 $0 $1
2  00100000010000100000000000000111 // addi $1 $1 7
3  10001100000000010000000000001111 // lw   $2 15($0)
4  10101100000010010000000000000011 // sw   $9 3($0)
5  00101000000000110000000000001111 // slti $3 $0 15
```

register

```
1  // memory data file (do not edit t
2  // instance=/mips_testbench/test/r
3  // format=bin addressradix=h datar
4  00000000000000000000000000000001
5  00000000000000000000000000001000
6  00000000000000000000000000001111
7  00000000000000000000000000000001
8  00000000000000000000000000000100
```

memory

```
1  // memory data file (do not edit the following l
2  // instance=/mips_testbench/test/mem0/data_mem
3  // format=bin addressradix=h dataradix=b version
4  00000000000000000000000000000000
5  00000000000000000000000000000001
6  00000000000000000000000000000010
7  00000000000000000000000001111000
8  00000000000000000000000000000100
```

---

instructions

```
00000000011000010000001110001000 // jr $3
00000000000000000000000000000000 // atlanacak
00000000000000000000000000000000 // atlanacak
00000000000000001000000000100111 // nor  $0 $0 $1
00111100000000001000000000001111 // lui  $1 15
00110000010000100000000000001111 // andi $2 $2 15
```

register

```
1  // memory data file (do not edit the fol
2  // instance=/mips_testbench/test/registe
3  // format=bin addressradix=h dataradix=b
4  11111111111111111111111111111110
5  00000000000011110000000000000000
6  00000000000000000000000000000010
7  00000000000000000000000000000011
8  00000000000000000000000000000100
```

---

instructions

```
00010000000000000000000000000001 // beq $0 $0 1 PC = 0 +1 +1
00000000000000000000000000000000 // atlanacak
00001000000000000000000000001100 // j 12  PC+1[31:28] + 2'b0 + adress  = 00000000000000000000000000001100 = 12
00000000000000000000000000000000 // buradan
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000
00000000000000000000000000000000 // bu satır dahil atlanacak
00000000000000010000100011000000 // sll  $1 $1 3
```

register

```
// memory data file (do not edit the followi
// instance=/mips_testbench/test/registers2/
// format=bin addressradix=h dataradix=b ver
00000000000000000000000000000000
00000000000000000000000000001000
00000000000000000000000000000010
00000000000000000000000000000011
00000000000000000000000000000100
00000000000000000000000000000101
```

pc

```
# Loading work.mips_data_mem
VSIM 70> step -current
# pc :           0
# pc :           2
# pc :          12
# pc :          13
# pc :           x

VSIM 71>
```