CSE 331 Computer Organization

Single Cycle MIPS with Structural Verilog

Final Project – Report

Ahmet Yuşa Telli

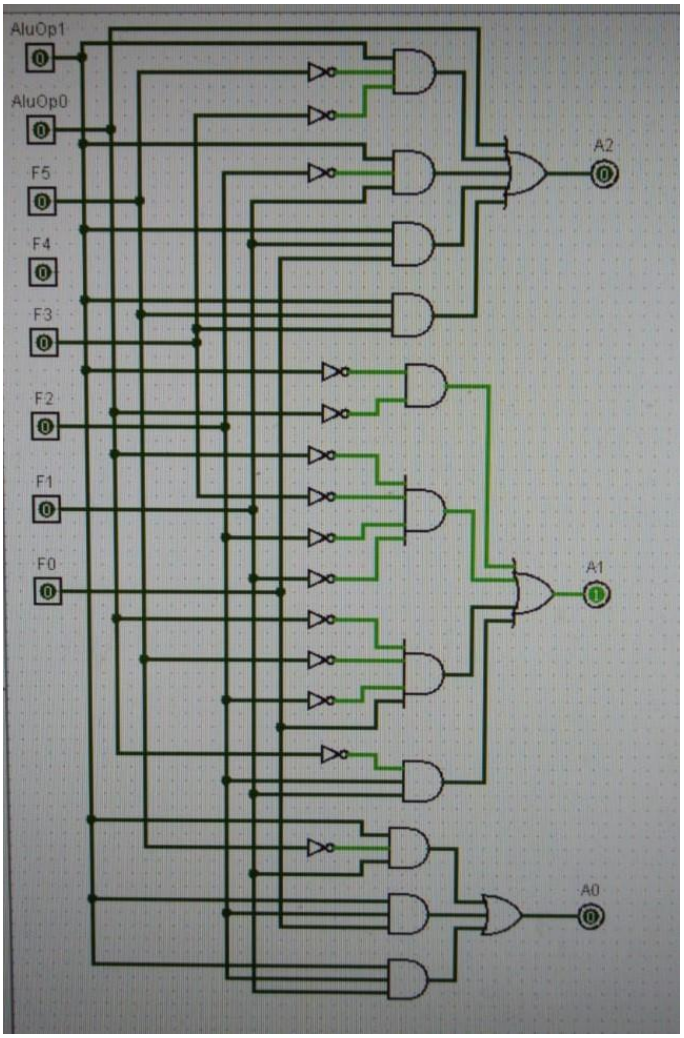
151044092

Bu final projesinde single cycle datapath tasarlamaya çalıştık. Önceki projenin devamı niteliğindeki bu projede ek olarak şunlar vardır:

SignExtend: instruction son 16 bitini alıp 32 bit olacak şekilde most sign bitini tek tek ekliyoruz.

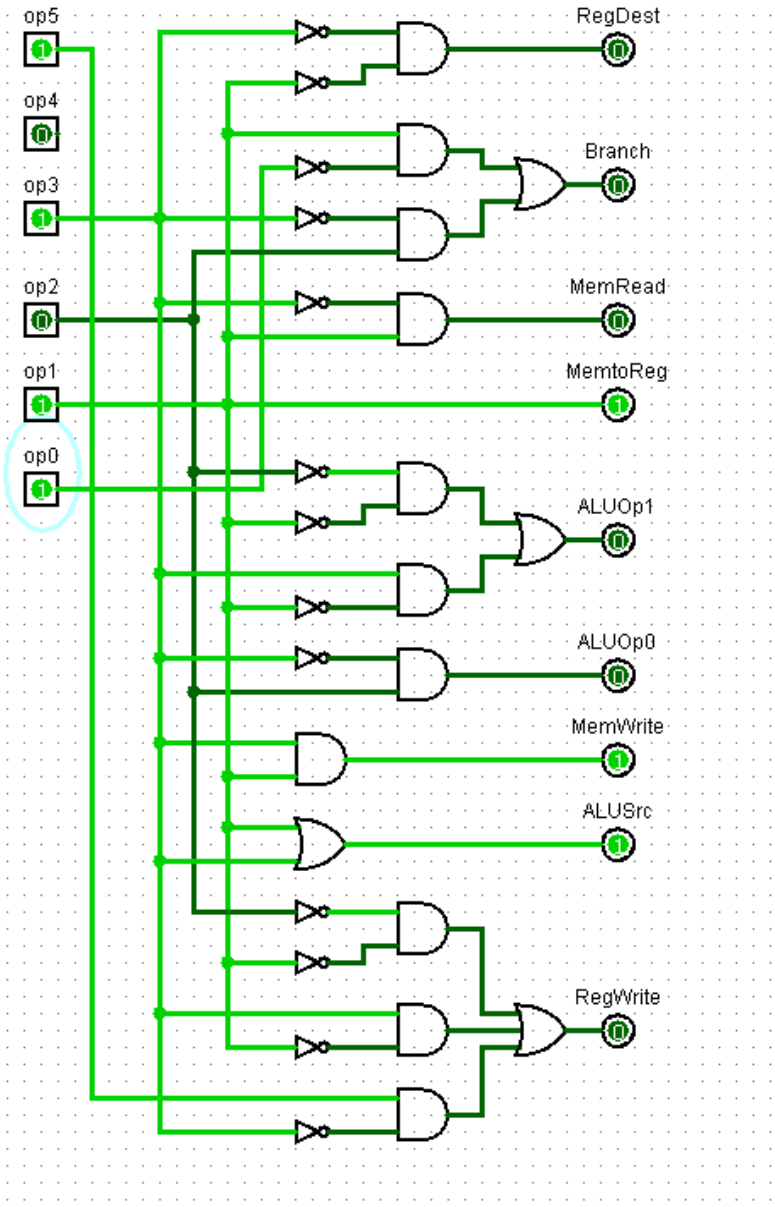
ZeroExtend: instruction son 16 bitini alıp 32 bit olacak şekilde tek tek 1'b0 ekliyoruz.

ALU ControlUnit: Önceki projeden farklı olarak burada Control unit'ten 2 bit ALUOp sinyali ve bu projede gerekli olan andi, ori, addiu sinyalleri geliyor. Bu sinyallere göre aşağıdaki devre tablosu ortaya çıkıyor. En son olarak andi, ori, addiu sinyallerine göre and, or veya add işlemi yapılacak 3 bitlik sinyali ALU ya gönderiliyor.



ProgramCounter: Burada instructions.mem dosyasından okuduğumuz değerleri tek satır olarak tutuyoruz ve sırayla ilerliyoruz. Gelen 32 bitlik değeri clk değerine göre dışarı veriyoruz.

Control Unit: Burada instruction'ın 31-26 bitleri input olarak alıyoruz. Bu gelen 6 bitlik opcode a göre yapılacak işlemi hesaplamak için aşağıdaki gibi devreyi tasarlıyoruz. Ek olarak andi, ori, addiu ve jump sinyallerinide üretiyoruz. ALU control için gerekli. Bu extra sinyalleri opcode'un her bitini ayrı ayrı and veya or işlemine tabii tutarak gerekli sinyalleri üretiyoruz.



Data Memory: dataMem.mem dosyasını içindeki 256 adet 32 bitlik sayıyı buraya alıyoruz. Burada memWrite sinyali ve memRead sinyallerine göre memory e yazıyoruz veya okuyoruz. Eğer memWrite 1'b1 gelirse gelen datayı datamemory'e yazıyoruz. memRead 1'b1 gelirse memory den datayı output olarak veriyoruz.

Datamem.mem: Bu dosyada 256 tane 32 bitlik data var. Bu datalar üzerinden işlem yapıyoruz.

Instructions.mem: Bu dosyada 11 adet 32bit inrtructionlar var. Onları teker teker alıp gerekli işlemleri yapıyoruz.

```
opcode  rs    rd    rd    rt    funct
000000 01110 01101 00001 00000 100000 add
001001 00001 00001 00010 00000 100001 addiu
001100 00001 00010 11111 11111 111111 andi
001101 00010 00011 00000 00000 000001 ori
101011 00000 00011 00000 00000 000001 sw
101011 00000 00011 00000 00000 000001 sw
100011 00000 11111 00000 00000 000001 lw
000010 00000 00000 00000 00000 000000
000000 01110 01101 00001 00000 100000 add
000000 00001 00001 00010 00000 100001
```

Mips32_Single_cycle: Bir instrcution cycle ı burada gerçekleşiyor. Tüm olaylar bağlantılar burada. Öncelikle programCounter modülü ile pc iyi alıyoruz. Ordan gelen pc ye 1 eklemek için adder a gönderiyoruz. Daha sonra extender işlemini yapıp zero veya sign extend işlemini tamamlıyoruz. PC+1 ile extend outputlarını add işlemi uygulayıp jumpaddress buluyoruz. Lazım olup olmayacağı henüz belli değil. Jump olup olmayacağını belirlemek için ALU nun zero biti ile control unitin branch sinyalini and uygulayarak bir sonraki PC nin ne olacağını seçecek bit olarak kullanıyoruz. Jumpaddress ile PC+1 arasında seçim yapmak için bu biti mux a select biti olarak gönderiyoruz. Ve PC değiştirmiş oluyoruz.

ControlUnit modülünü çağırıp gerekli sinyalleri alıyoruz.

Mips registers modülü ile register dan okuma işlemini yapıyoruz. Gelen instruction a göre rs rt rd concent lerini alıyoruz.

ALU Control modülüyle aluop ve gerekli sinyalleri buraya gönderip, ALU ya gidecek olan 3 bit select bitlerini belirliyoruz.

Önceki projedeki shamt işlemini yapıp ALU nun inputlarını belirliyoruz. Tabi burda eklememiz gereken bir mux daha var. ALU nun ikinci input kısmına signextend edilmiş 32 bit mi girecek yada önceki mux un result ı mı girecek onu control unitteki ALUSrc sinyali belirliyor.

Önceki projedeki sltu işlemi için gerekli 32 bit sayıyı hesaplıyoruz.

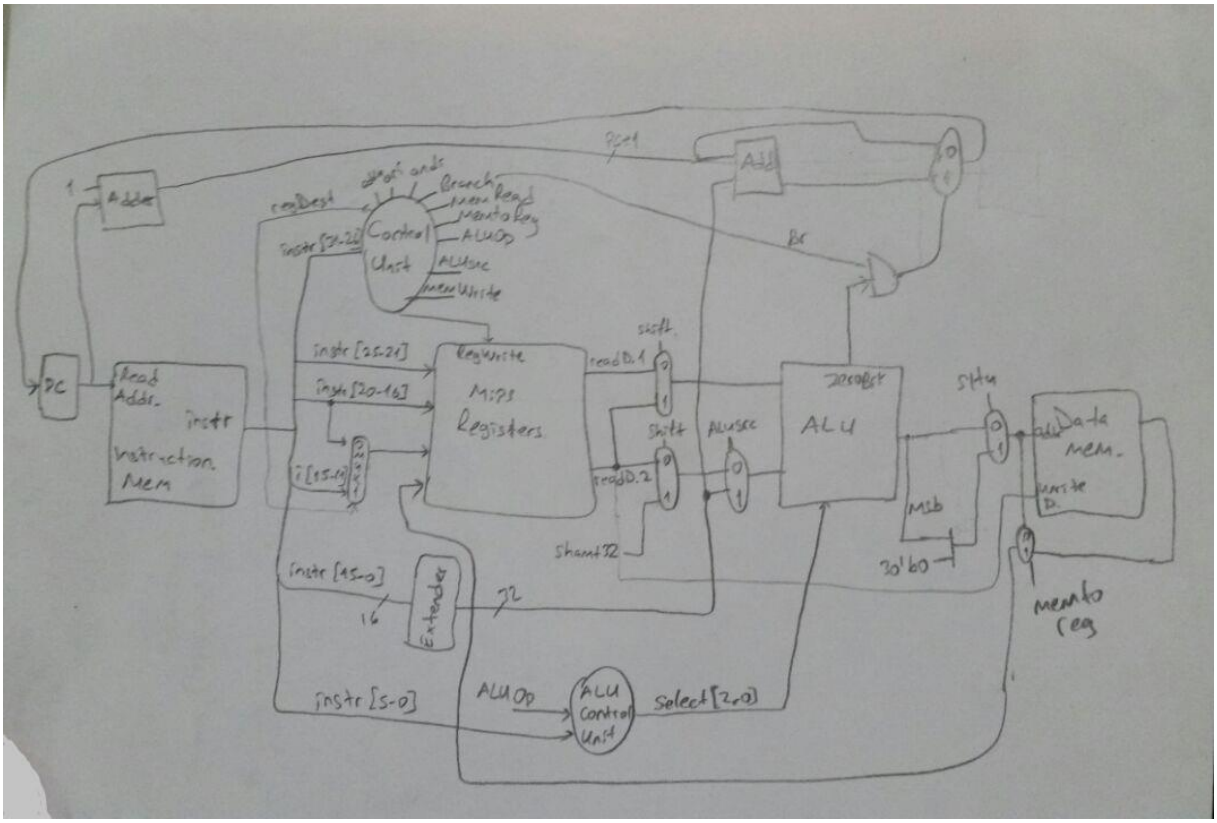
Data memory ile burada yazma veya okuma işlemi olucaksa ona göre data memory den output alıyoruz. MemtoReg sinyaline göre registera ne yazıcalağını belirliyoruz. Data memorynin output mu yoksa ALU result mı.

Seçilen result direk register writedata kısmına gidiyor.

Modelsim çıktısı:

```
VSIM 5> step -current
#           PC : 00000000000000000000000000000000
#                opcode/__rs_/__rt_/__rd_/shamt/_funct     immediate
# instruction : 000000 00110 01110 00001 00000 100000     0000100000100000
# result      : 00000000000000000000000000000000
#
#           PC : 00000000000000000000000000000001
#                opcode/__rs_/__rt_/__rd_/shamt/_funct     immediate
# instruction : 001001 00110 01110 00010 00000 100001     0001000000100001
# result      : 00000000000000000000000000000000
#
#           PC : 00000000000000000000000000000010
#                opcode/__rs_/__rt_/__rd_/shamt/_funct     immediate
# instruction : 001100 00110 01110 11111 11111 111111     1111111111111111
# result      : 00000000000000000000000000000000
#
#           PC : 00000000000000000000000000000011
#                opcode/__rs_/__rt_/__rd_/shamt/_funct     immediate
# instruction : 001101 00110 01110 00000 00000 000001     0000000000000001
# result      : 00000000000000000000000000000001
#
#           PC : 00000000000000000000000000000100
#                opcode/__rs_/__rt_/__rd_/shamt/_funct     immediate
# instruction : 101011 00110 01110 00000 00000 000001     0000000000000001
# result      : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#
#           PC : 00000000000000000000000000000101
#                opcode/__rs_/__rt_/__rd_/shamt/_funct     immediate
# instruction : 101011 00110 01110 00000 00000 000001     0000000000000001
# result      : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#
#           PC : 00000000000000000000000000000110
#                opcode/__rs_/__rt_/__rd_/shamt/_funct     immediate
# instruction : 100011 00110 01110 00000 00000 000001     0000000000000001
# result      : xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
#
#           PC : 00000000000000000000000000000110
#                opcode/__rs_/__rt_/__rd_/shamt/_funct     immediate
# instruction : 100011 00110 01110 00000 00000 000001     0000000000000001
# result      : 00000000000000000000000000000001
#
#           PC : 00000000000000000000000000000111
#                opcode/__rs_/__rt_/__rd_/shamt/_funct     immediate
# instruction : 001100 00001 00010 11111 11111 111111     1111111111111111
# result      : 00000000000000001110111111000100
#
#           PC : 00000000000000000000000000001000
#                opcode/__rs_/__rt_/__rd_/shamt/_funct     immediate
# instruction : 000010 00110 01110 00000 00000 000000     0000000000000000
# result      : 00000000000000000000000000000001
#
# Break in Module mips32_single_cycle_testbench at C:/Users/ASUS/Desktop/ay/
```

Genel olarak yapmaya çalıştığım datapath aşağıdaki gibi :



Ahmet Yuşa Telli

151044092

Rapor sonu