T.C.

İSTANBUL ÜNİVERSİTESİ – CERRAHPAŞA

MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

2021 – 2022 BAHAR YARIYILI

BİLGİSAYAR MİMARİSİ

DÖNEM PROJESİ

Emin Can ÖZGE - 1306190022

Mustafa Emre TAŞKIN - 1306190005

Nihat PAMUKÇU - 1306190035

Işıl VARDARLI - 1306180052

Siraç PETMEZÇİLER – 1306190011

İçindekiler

PLANLAMA VE TOPLANTI TABLOLARI

Tarih	Toplantı İçeriği
10.05.2022	Projede kullanılacak componentlar araştırılmak üzere paylaştırıldı. Ders PDF'leri üzerinden component tasarımları incelendi.
14.05.2022	Componentlar hakkında yapılan araştırmalar sunularak tasarım fikirleri oluşturularak componentların genel yapısı oluşturuldu. Oluşturulan tasarım planları grup üyelerine eşit olarak dağıtıldı.
19.05.2022	Dağıtılan component tasarımları bitti. Oluşturulan yapıların kontrolünün ve hata ayıklamanın sağlanması için grup üyelerine farklı componentlar dağıtıldı.
22.05.2022	Hatalı bulunan kısımlar üyeler tarafından ayrıştırılarak düzeltildi. Böylece datapath için oluşturulan componentlar nihai haline ulaştı. Componentlar birleştirilerek datapath oluşturma süreci başladı.
24.05.2022	Datapathin son hali oluşturuldu. Toplantı anında bazı hatalar çözüldü. Bunun sonucunda temel komutların çalıştığı görüldüğünden rapor hazırlanması ve test işlemleri başladı.
25.05.2022	Rapor ve test sonuçları kontrol edildi. Teslim edildi.

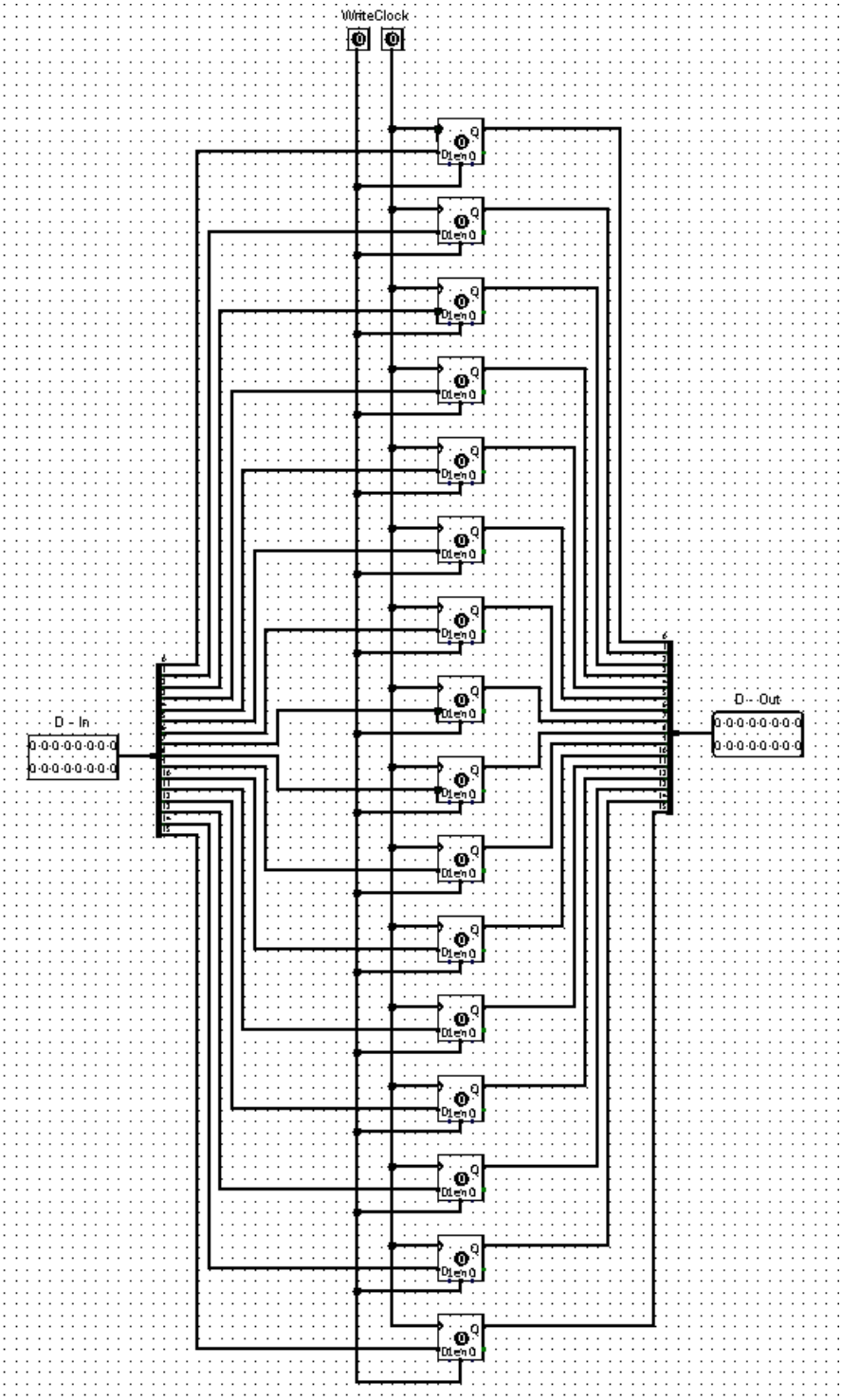
Üye / Görev	Emin Can ÖZGE	Mustafa Emre TAŞKIN	Nihat PAMUKÇU	Işıl VARDARLI	Siraç PETMEZÇİLER
Register				+	
Register Set				+	
Extender		+			
ALU	+				
ALUCtrl	+				
Program Counter Control		+			
IF Buffer			+		
EX Buffer					+
MEM Buffer					+
ID Buffer			+		
Forward				+	
Control Unit			+		
Datapath (Processor)	+	+			+

1- TASARIM AŞAMASI

1. Register (16-bit)

İlk olarak 16-bit pipelined bir işlemci oluşturmak için 16-bitlik register yapısı oluşturduk. Burada 16-bitlik girdi, yazdırılabilirlik için bir flag ve saat darbesiyle çalışabilmesi için bir Clock girişi kullanıldı. Registerın ana amacı olan kayıt işlemini gerekli D flip flop devreleriyle sağladık. Bu sayede girdiler registerın çıkışına kaydedilmiş oldu.

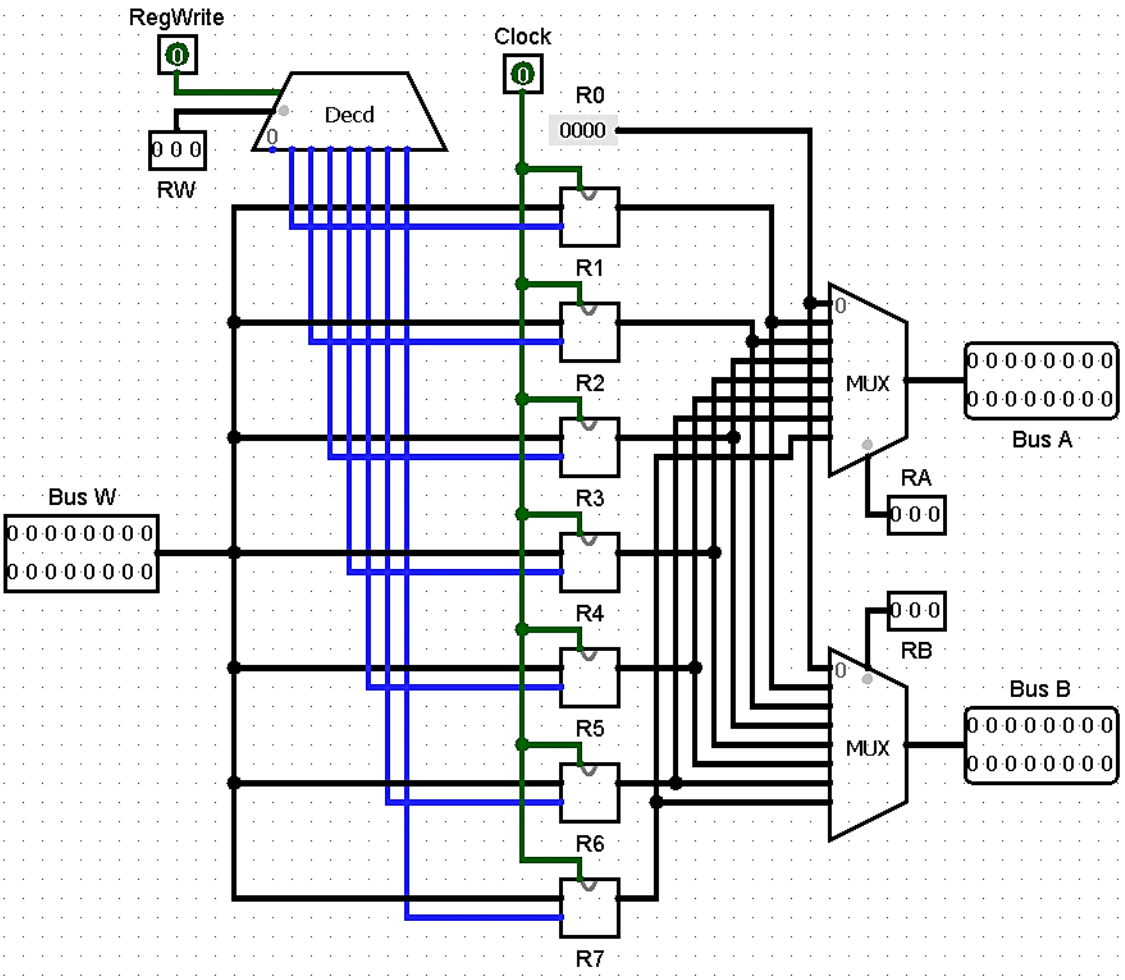
Registerın iç yapısı aşağıdaki gibidir.



2. Register Set

Ödev tanımında toplamda 8 adet register tutulacağı için registerlar küme halinde tek bir componentta toplanmıştır. Burada ilk register değiştirilemeyen ve daima sıfır değerini barındıran bir register olduğu için sabit bir sıfır değeri ve 7 adet 1.maddede bahsettiğimiz değişken registerlar kullanılmıştır. 8 adet registerı seçebilmek için 3 bitlik bir girdi seçildi. Bu seçme işlemini açabilmek için bir decoder ve çıktılar içinse mux kullanıldı.

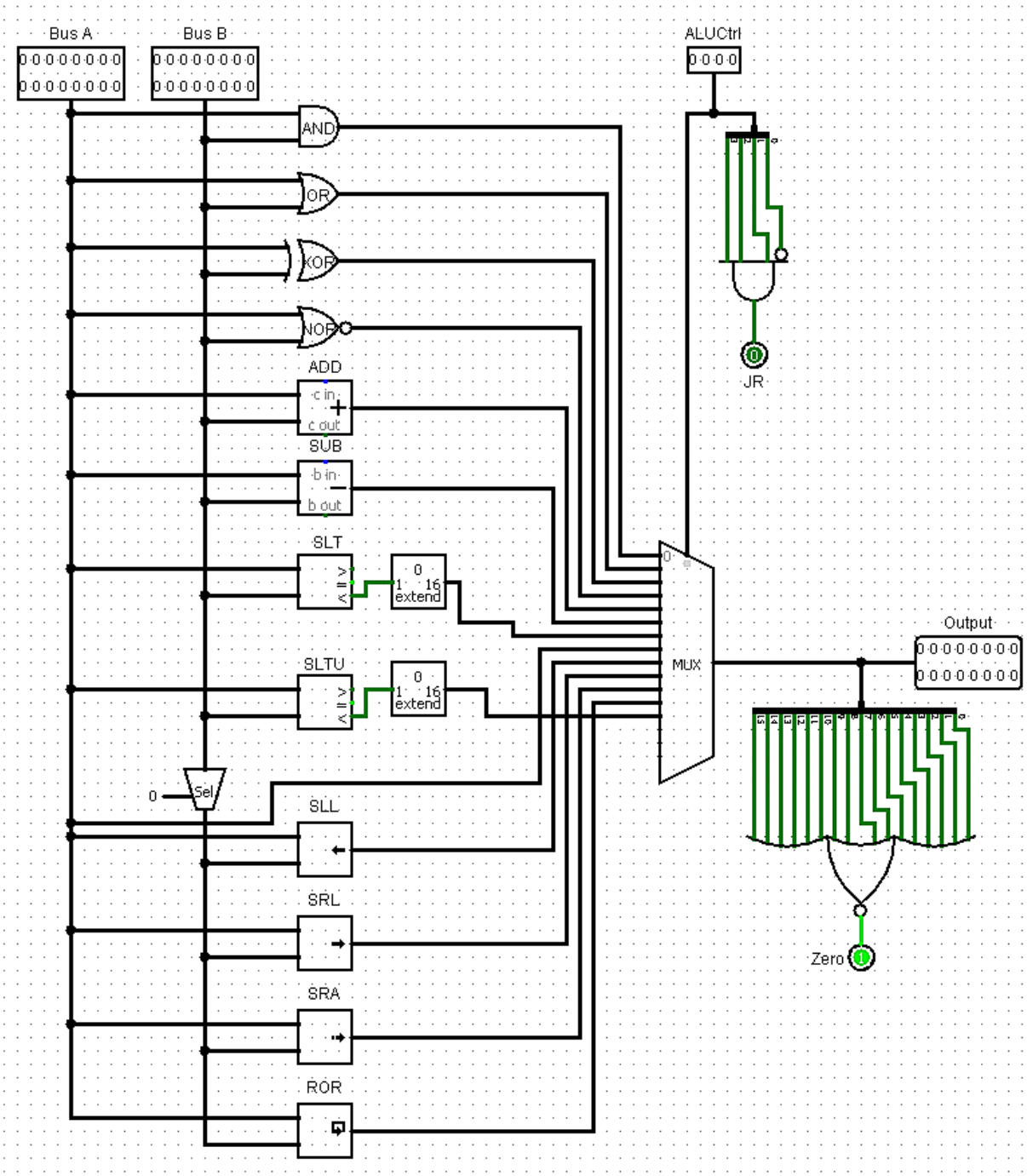
Register Setin iç yapısı aşağıdaki gibidir.



3. ALU

Tabloda verilen işlemlerin yapılabilmesi için bir ALU tasarlandı. Burada var olan lojik işlemler için (AND, OR, SHIFT vs.) Logisim içerisinde hazır olan componentler kullanıldı. SLT ve SLTU çıktılarında sign extend kullanıldı. Tüm işlemlerin sonunda 4-16 bir MUX kullanılarak uygun işlem yönlendirildi. MUX'un seçme girişine ise ALUCtrl'dan gelen çıktı bağlandı. MUX'un sonucu ise ALU output ve splitter ile NOR kullanılarak zero olarak çıkarıldı.

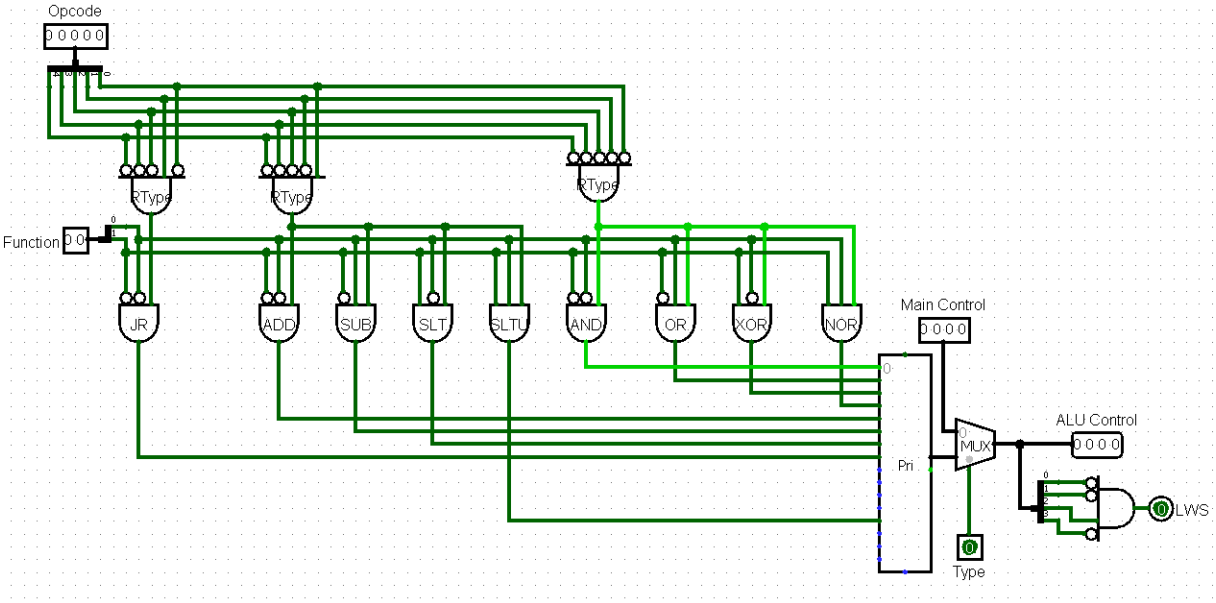
ALU'nun iç yapısı aşağıdaki gibidir.



4. ALUCtrl

ALUCtrl içerisinde proje tablosunda verilen biçimde 5-bit Opcode ve 2-bit Function code kullanılarak gerekli ALU işlemi ve tipinin çıktı olarak verilmesi amaçlanmıştır. Proje tanımında normalden farklı olarak R-Type bir instructiona sıfırdan farklı opcode değerleri verildiğinden, işlemlerin 3 farklı gruba ayrılarak (0-1-2) seçim aşamasından geçmesi gerekmiştir. Main Controldan gelen değerler öncelik seçiciden gelen değer ile MUX'tan geçerek ALU Control çıkışına verilmiştir. Bunun haricinde aynı çıktı işlemin load işlemi olup olmadığına da karar verebilmek adına splitterda ayrılarak AND kapısı ile çıktı verilmiştir.

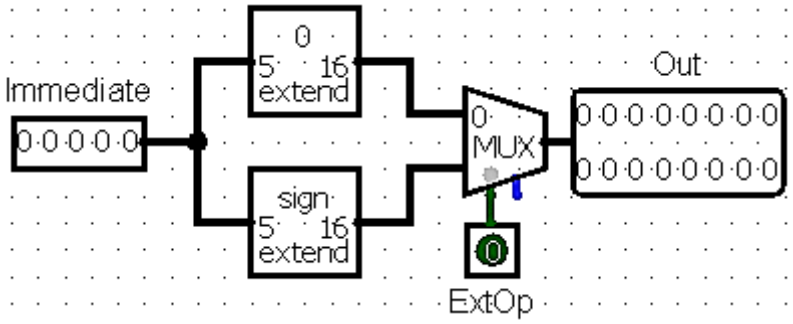
ALUCtrl'nin iç yapısı aşağıdaki gibidir.



5. Extender

Extender hem signed hem de unsigned gelen immediate değerlerin ExtOp flagden gelen değere göre seçilmesi ile birlikte 16-bite tamamlanması için oluşturulmuş bir yapıdır. Signed ve unsigned extendden gelen veriler MUX'lanmıştır.

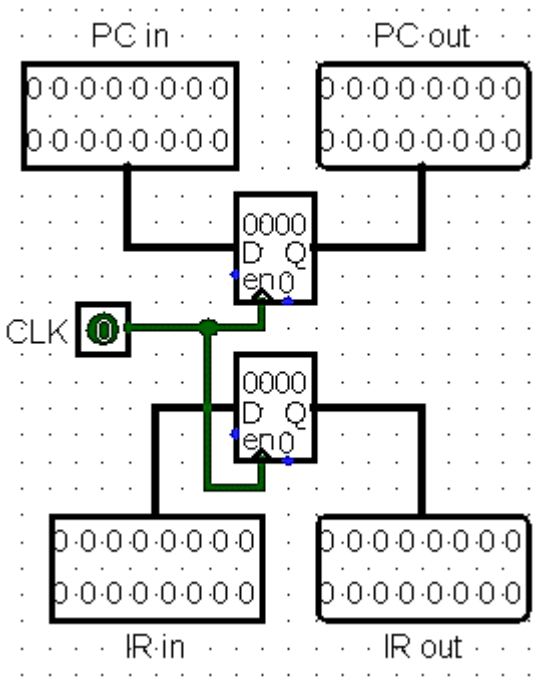
Extenderın iç yapısı aşağıdaki gibidir.



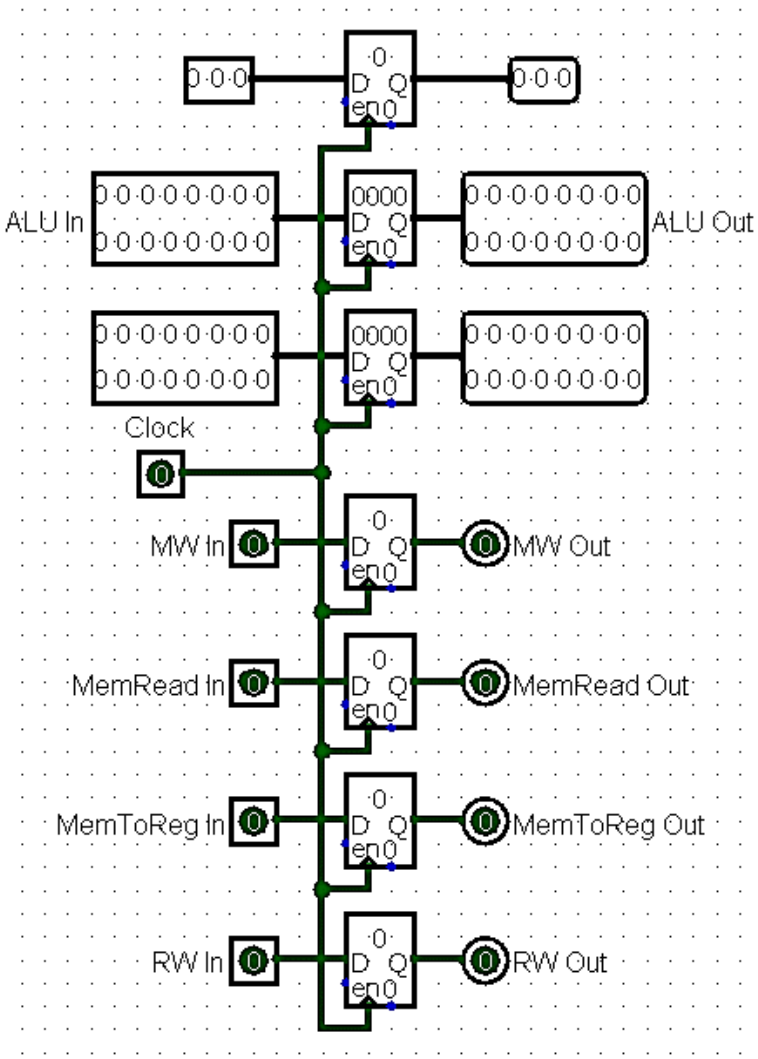
6. Bufferlar

Pipeline yapısında aşamalar için birer buffer oluşturduk. IF Buffer, EX Buffer, MEM Buffer, ID Buffer olmak üzere toplamda 4 adet buffer bulunuyor. Bu sayede aşamalar arasında geçerken birbirleri arasında karışmanın önüne geçilmiş oldu. Her clock cycleda bir aşama atlanacağı için yapıda kullandığımız D flip floplara clock bağlanarak bir enable girişi olarak kullanılmıştır.

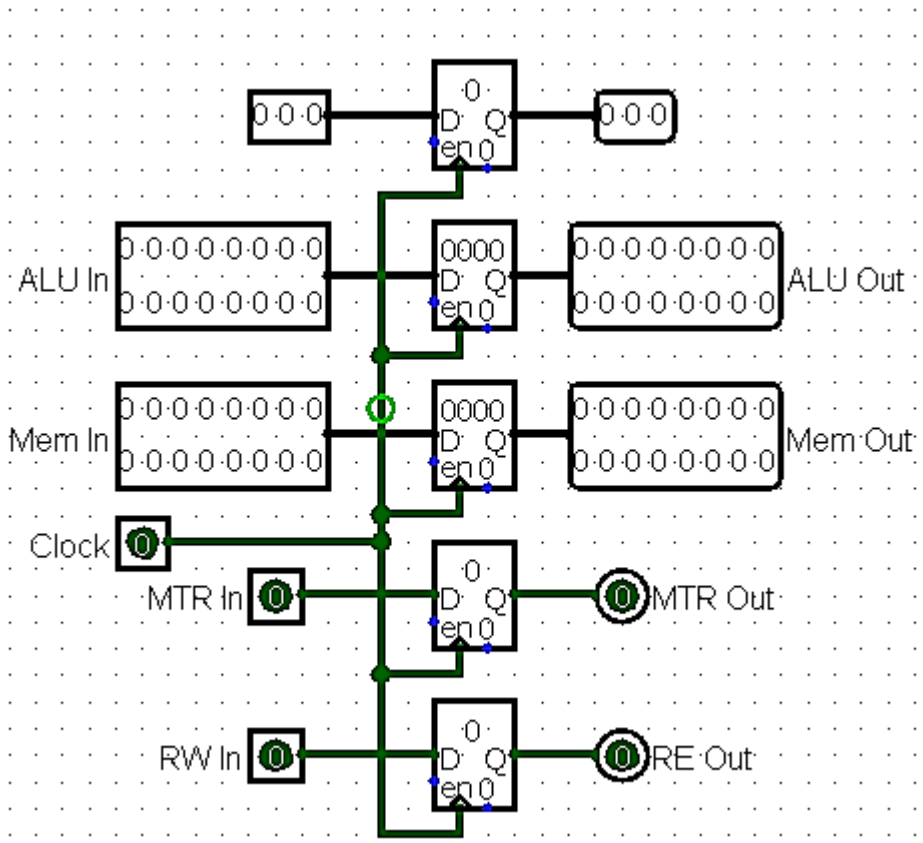
IF Buffer'ın iç yapısı aşağıdaki gibidir.



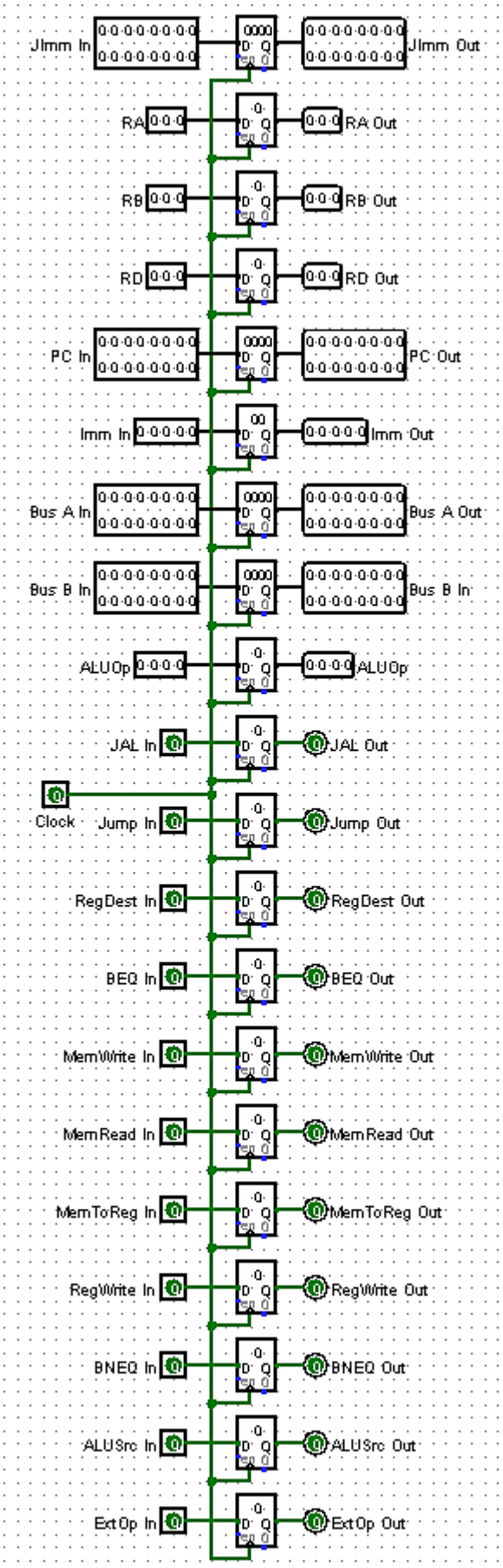
EX Buffer'ın iç yapısı aşağıdaki gibidir.



MEM Buffer’ın iç yapısı aşağıdaki gibidir.



ID Buffer’ın iç yapısı aşağıdaki gibidir.

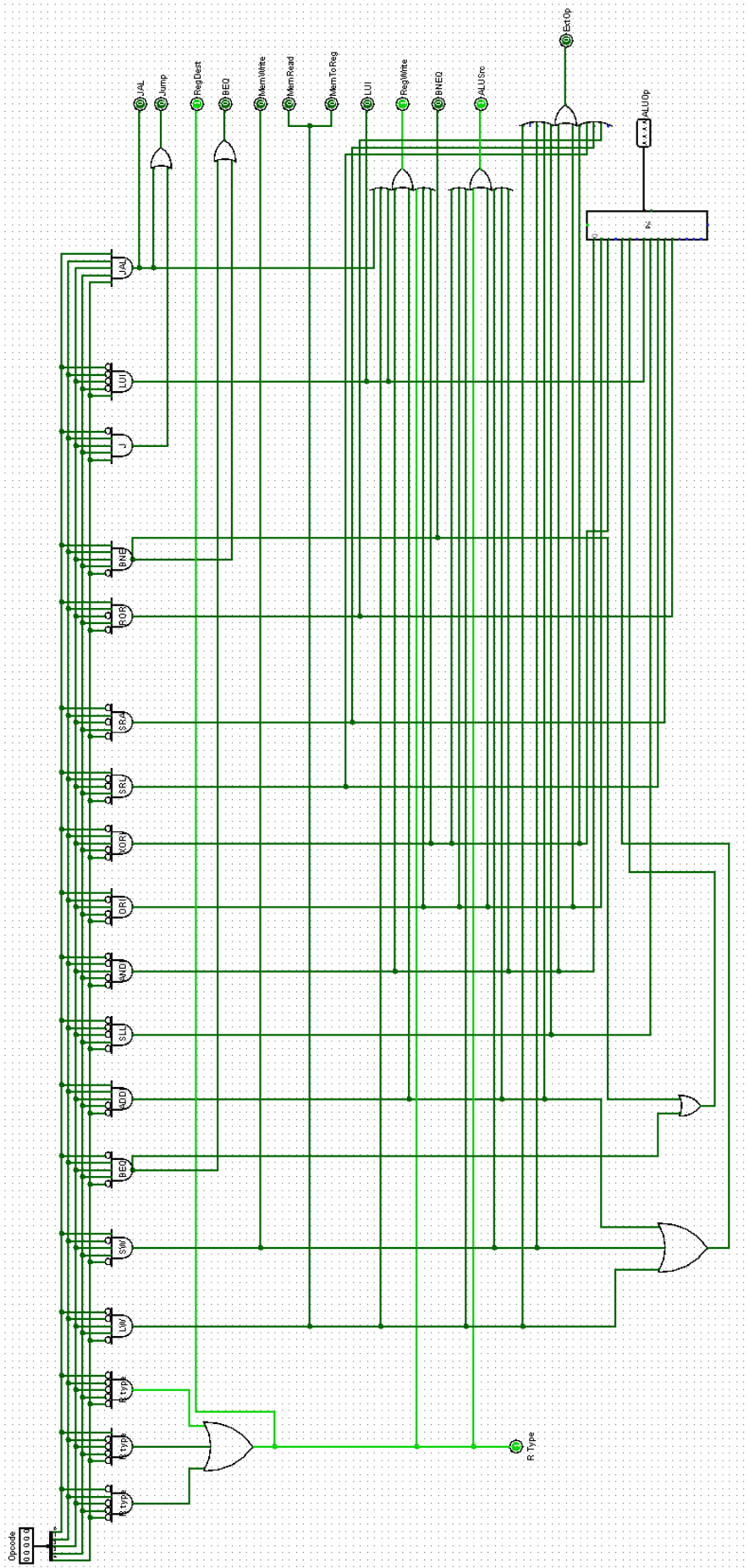

JImm In
JImm Out
RA
RA Out
RB
RB Out
RD
RD Out
PC In
PC Out
Imm In
Imm Out
Bus A In
Bus A Out
Bus B In
Bus B In
ALUOp
ALUOp
JAL In
JAL Out
Clock
Jump In
Jump Out
RegDest In
RegDest Out
BEQ In
BEQ Out
MemWrite In
MemWrite Out
MemRead In
MemRead Out
MemToReg In
MemToReg Out
RegWrite In
RegWrite Out
BNEQ In
BNEQ Out
ALUSrc In
ALUSrc Out
ExtOp In
ExtOp Out

7. Control Unit

Control Unit Opcode'un çözümlendiği ve instruction seçiminin yapıldığı componenttir. Proje tanımında verilen tabloda opcodelar 5-bit olarak verildiğinden 5-bitlik opcode girişi alarak gerekli flagler seçilir. Bu işlem tablodaki opcode değerlerine göre AND ve NOT kapıları kullanılarak elde edilmiştir. Çıktılardan biri de ALUOp olduğundan ve 3-bit yetmeyeceğinden ($>2^3$) 4-bitlik bir ALUOp çıkışı kullanılmıştır.

Inst	Opcode	JAL	Jump	Reg Dest	BEQ	Mem Write	Mem Read	Mem To Reg	Reg Write	BNEQ	ALU Src	ExtOp	ALU Op
AND	00000	0	0	1	0	0	0	0	1	0	1	0	0000
OR	00000	0	0	1	0	0	0	0	1	0	1	0	0001
XOR	00000	0	0	1	0	0	0	0	1	0	1	0	0010
NOR	00000	0	0	1	0	0	0	0	1	0	1	0	0011
ADD	00001	0	0	1	0	0	0	0	1	0	1	0	0100
SUB	00001	0	0	1	0	0	0	0	1	0	1	0	0101
SLT	00001	0	0	1	0	0	0	0	1	0	1	0	0110
SLTU	00001	0	0	1	0	0	0	0	1	0	1	0	0110
ANDI	00100	0	0	0	0	0	0	0	1	0	1	1	0000
ORI	00101	0	0	0	0	0	0	0	1	0	1	1	0001
XORI	00110	0	0	0	0	0	0	0	1	0	1	1	0010
ADDI	00111	0	0	0	0	0	0	0	1	0	1	1	0100
SLL	01000	0	0	0	0	0	0	0	0	0	0	1	1000
SRL	01001	0	0	0	0	0	0	0	0	0	0	1	1001
SRA	01010	0	0	0	0	0	0	0	0	0	0	1	1010
ROR	01011	0	0	0	0	0	0	0	0	0	0	1	1011
LW	01100	0	0	0	0	0	1	1	1	0	1	1	0100
SW	01101	0	0	0	0	1	0	0	0	0	1	1	0100
BEQ	01110	0	0	0	1	0	0	0	0	0	0	0	0101
BNE	01111	0	0	0	0	0	0	0	0	1	0	0	0101
LUI	10000	0	0	0	0	0	0	0	1	0	0	0	XXXX
J	11110	0	1	0	0	0	0	0	0	0	0	0	XXXX
JAL	11111	1	1	0	0	0	0	0	0	0	0	0	XXXX
JR	00010	0	0	1	0	0	0	0	1	0	1	0	0111

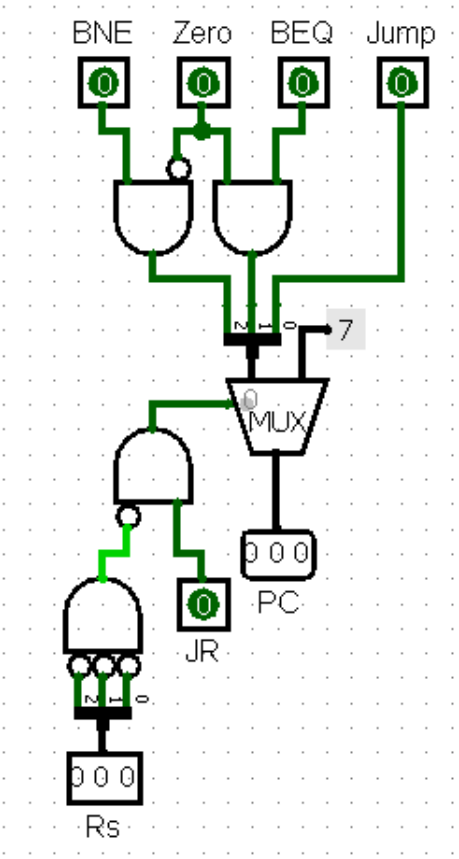
Control Unit'in iç yapısı aşağıdaki gibidir.


JAL
Jump
RegDest
BEQ
MemWrite
MemRead
MemToReg
LUI
RegWrite
BNEQ
ALUSrc
ExtOp
ALUOp
Opcode
R Type

8. Program Counter Control

Program Counter Control içerisinde Jump, BEQ, BNE instructionları birer input olarak alınarak MUX'a gönderilmiştir. Burada Rs girişini JR komutunu ayırt edebilmek için 2 kez AND kapısından JR ile geçirilerek MUX'a bağlanmıştır. Program Counter'ın amacı sıradaki instructionı belirlemek olduğundan, MUX'un sonucu Program Counter çıktısı olarak verilmiştir.

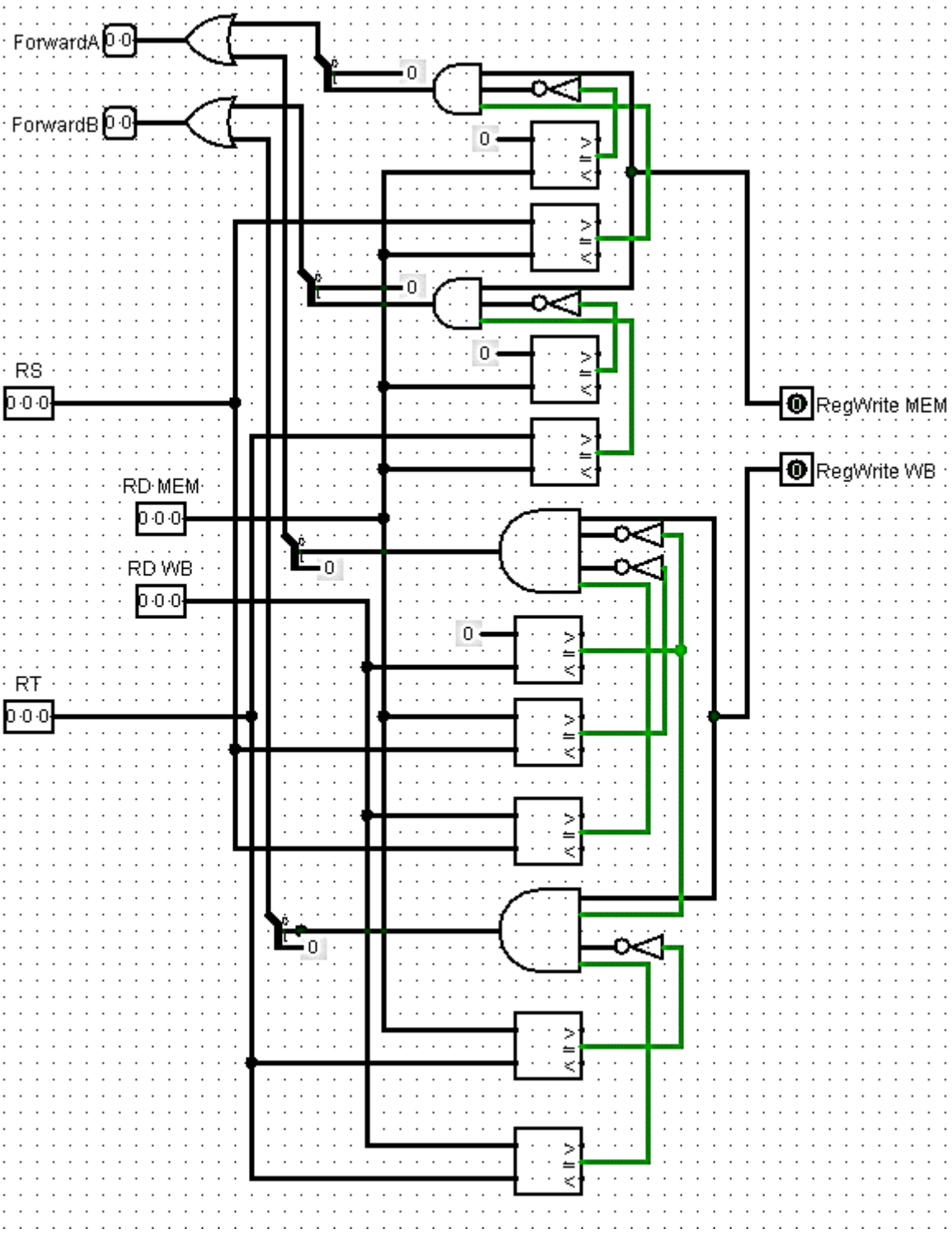
Program Counter Control'ün iç yapısı aşağıdaki gibidir.



9. Forwarding Unit

Normalde data hazard oluşmaması için bir cycle bekleme(stall) yapılması gibi yöntemler bulunur. Bunlardan biri de Forwarding Unit kullanarak önceki çıktıların işlenerek ALU'ya gidecek inputları belirlemektir. Biz de aynı şekilde ID Bufferdan gelen iki çıktı olan Rs ve Rt'yi, bellek aşamasından çıktı olarak ise Rd'yi kullandık. Böylece bir cycle stalldan ve data hazarddan kurtulmuş olduk.

Forwarding Unit'in iç yapısı aşağıdaki gibidir.


ForwardA
ForwardB
RS
RD MEM
RD WB
RT
RegWrite MEM
RegWrite WB

10. Datapath (Processor)

Tüm bileşenlerin birleştirildiği ve işlemcinin oluşturulduğu yapıdır. Bu kısımda yapı klasik pipelined datapath şemasıyla aynı yapıda oluşturulmuştur.

Datapath 5 aşamadan oluşur ve temel açıklamaları aşağıdaki gibidir.

1. Fetch: Bu aşamada Program Counterdan adres Instruction Memorye geçer ve PC 1 artırılır. Bu adres IF Buffera aktarılarak sıradaki aşamaya hazır hale getirilir.

2. Decode: Fetch aşamasında oluşturulan verinin opcode kısmı Control Unit'e gönderilir. Kalan kısım parçalanarak Register Set'e gönderilir. Bu veriler ID Buffer'a gönderilerek sıradaki aşamaya hazır hale getirilir.

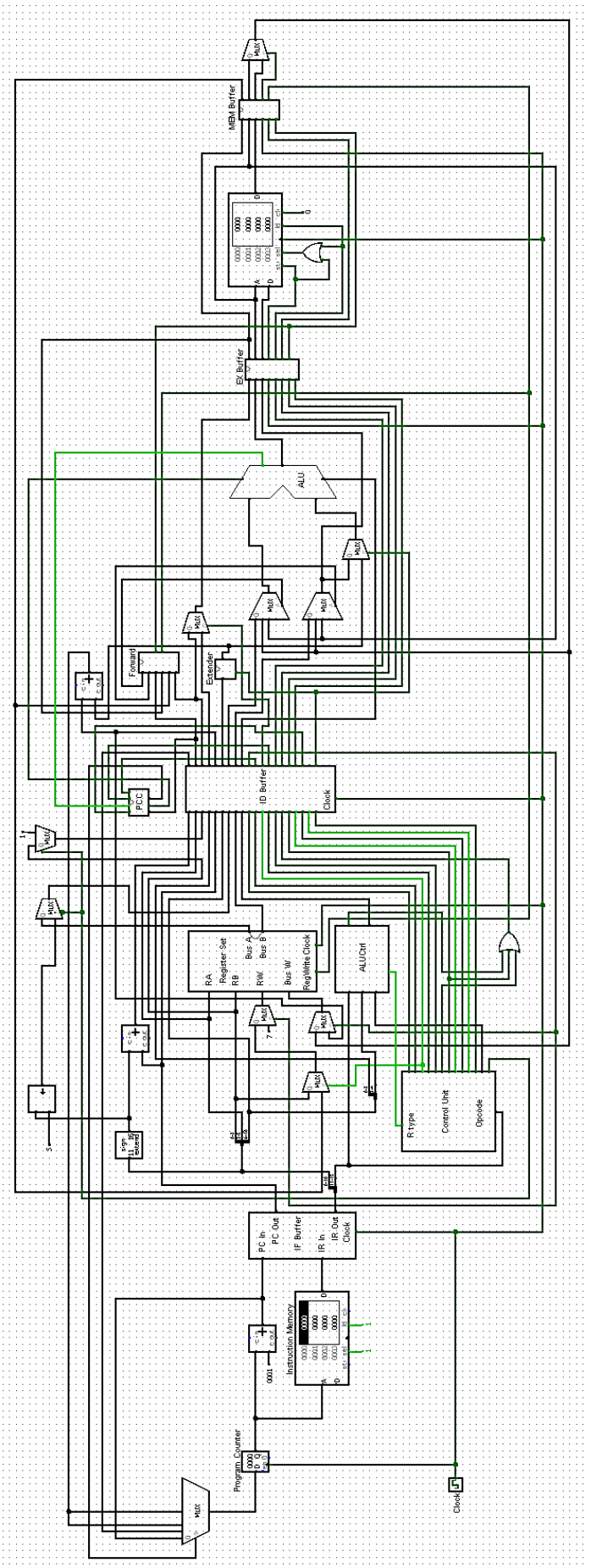
3. Execute: Bu aşamada Register Setteki outputlar ve geçmiş cyclelardaki ALU outputları Forwarding Unitte karşılaştırılarak ALU'dan Program Counter Control'a gönderilir. Kalan bilgiler ise EX Buffer ile sıradaki aşamaya gönderilir.

4. Memory: Bu aşamada Load ya da Store komutu geldiyse memorye erişim sağlanır. Branch komutuysa adres Program Counter'a atanır ve MEM Buffer ile sıradaki aşamaya geçiş yapılır.

5. Write Back: Çıktılar gerekli registerlara yazılır.

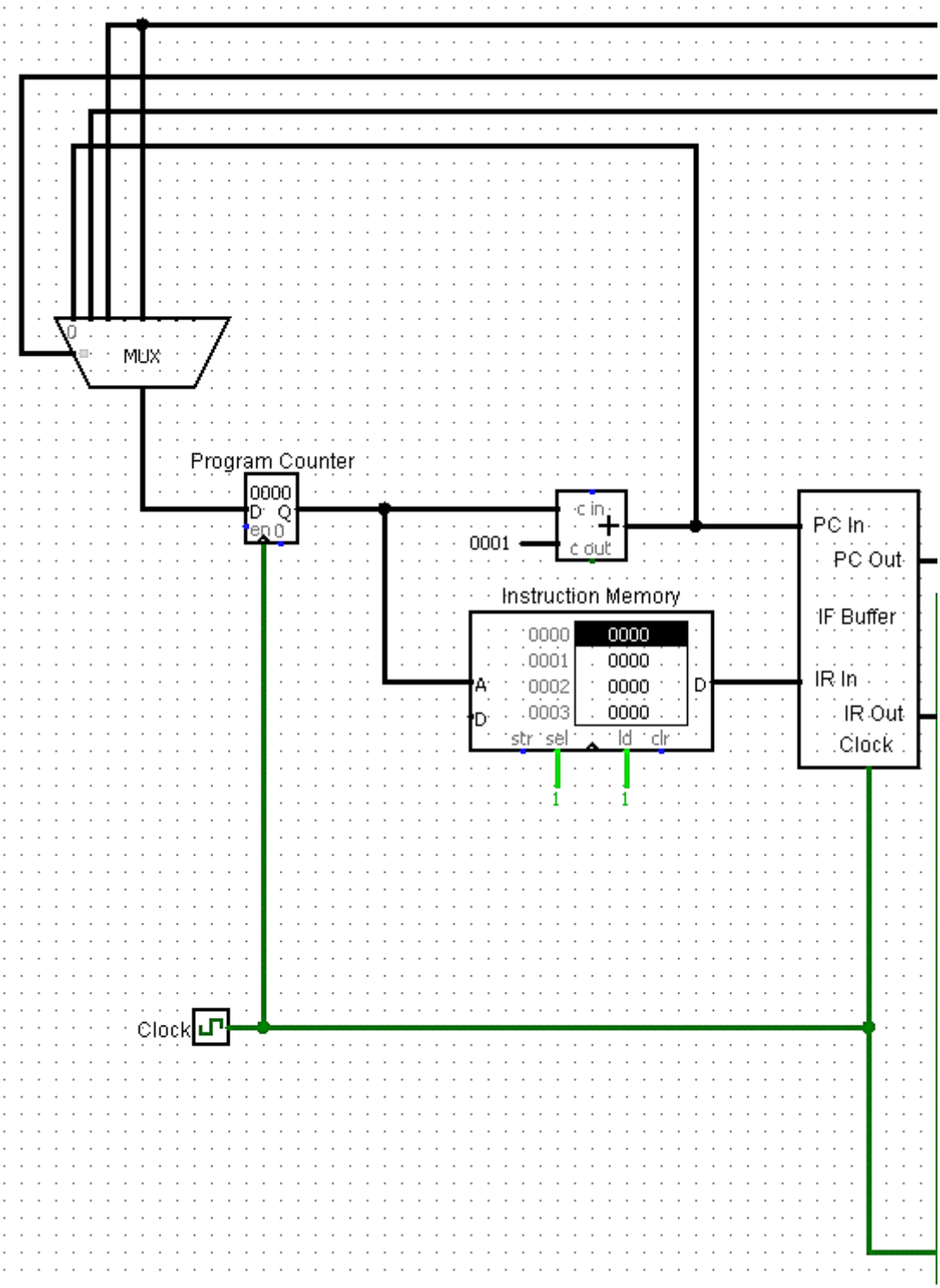
Bu aşamalara göre bufferlar arasında geçiş sağlanmıştır.

Datapathin yapısı aşağıdaki gibidir.

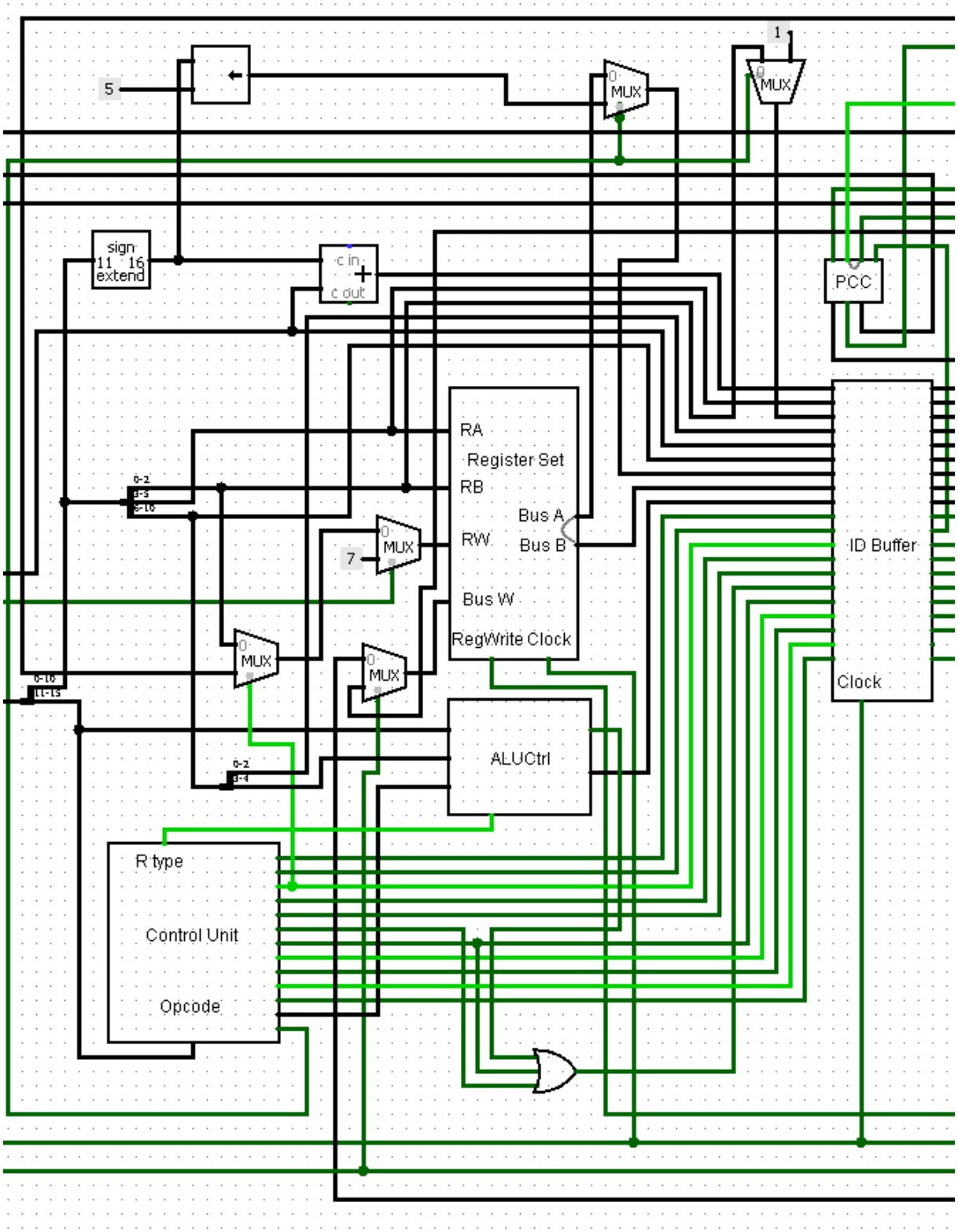

MEM Buffer
EX Buffer
ALU
Forward
Extender
ID Buffer
Clock
PCC
Register Set
RA
RB
Bus A
Bus B
RWr
Bus W
RegWrite Clock
ALUCtrl
R type
Control Unit
Opcode
IF Buffer
PC In
PC Out
IR In
IR Out
Clock
Instruction Memory
Program Counter
Clock

Sırasıyla aşamalar aşağıda gösterilmiştir.

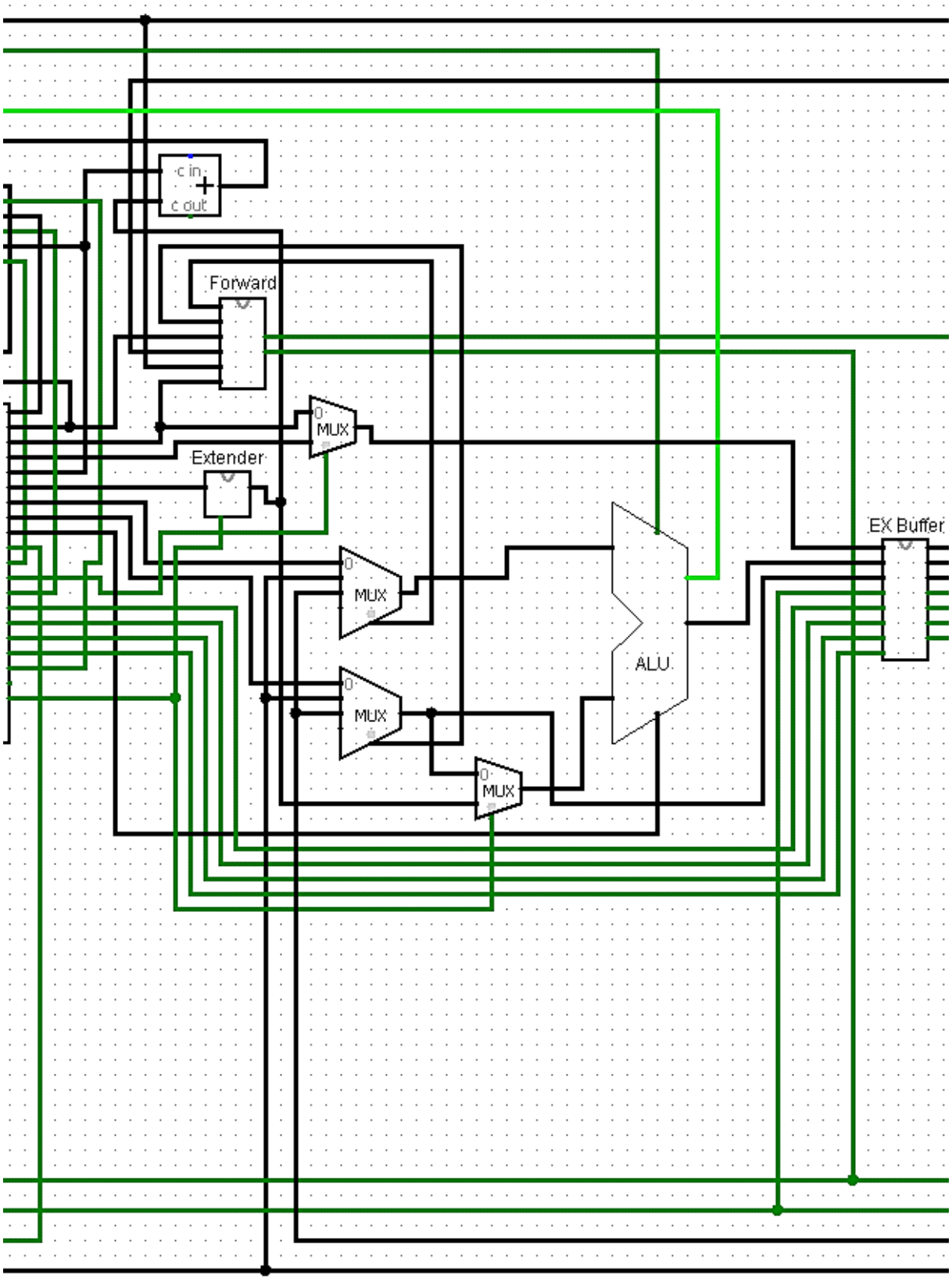
1. **Fetch Stage**



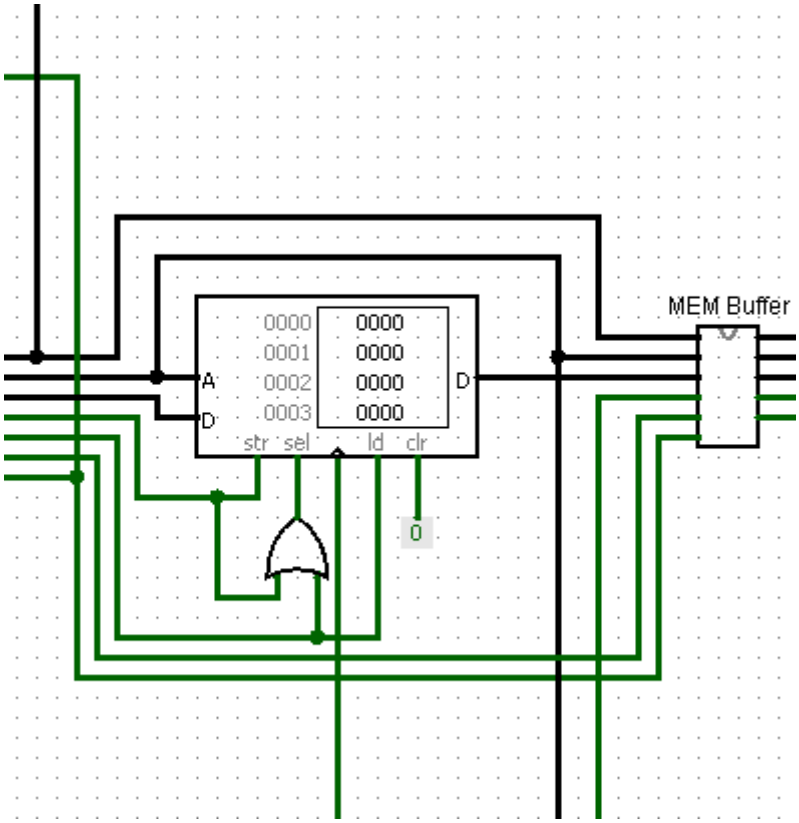
2. Decode Stage



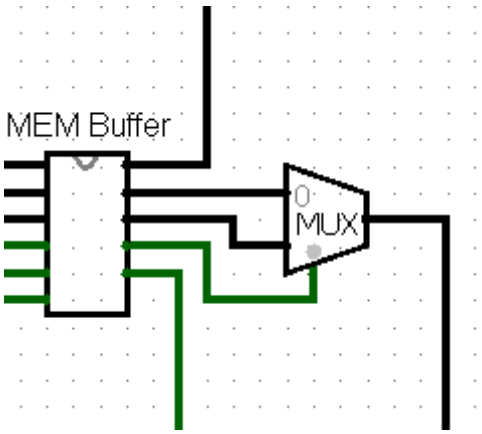
3. Execute Stage



4. Memory Stage



5. Write Back Stage



2- SİMÜLASYON VE TEST AŞAMASI

Tüm componentlar ve datapath Logisim üzerinde çizildiğinden simülasyon da bu ortamda yapılmıştır.

Logisim Avantajları

1. Java ile yazıldığından platform bağımsız bir çalışma ortamı sundu.
2. Farklı cihazlarda sadece component tasarlayarak import/export işlemi yapabilmek bize grup olarak ayrı cihazlarda çalışma fırsatı verdi.
3. Çizim işlemleri kolay olduğundan projeyi olduğundan karmaşık hale getirmedi.
4. İçinde barındırdığı temel componentler sayesinde daha kolay bir şekilde implementasyon yapabildik.
5. Barındırdığı toolbox yapısı kolay ve erişilebilir olduğundan daha basit bir kullanıcı deneyimi sundu.
6. Renklendirmeleri sayesinde tasarım aşamasında kablo karmaşasından kurtulduk.

Logisim Dezavantajları

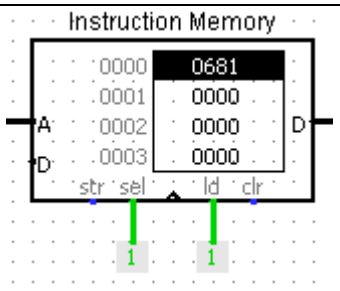
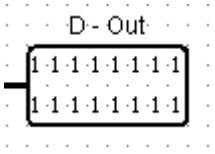
1. Nadir de olsa JVM'den dolayı Runtime Error aldık. Bu hatalar sebebiyle çalışmalarımızın bazı kısımlarını kaybettik ve bize zaman kaybettirdi.
2. Bazı componentler koyulduktan sonra kısmen çalışmaz durumda kaldı. Bundan dolayı uygulamayı birden çok yeniden başlatmak zorunda kaldık.

Bahsettiğimiz Logisim uygulamasıyla ile yaptığımız simülasyonda sırayla bazı temel kodlar denenmiş ve çıktıları alınmıştır.

2.1. R-Type Komutlar

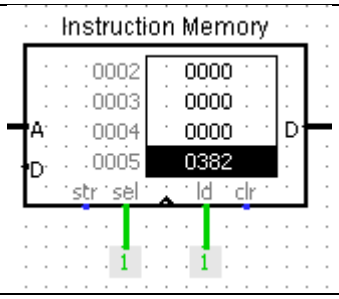
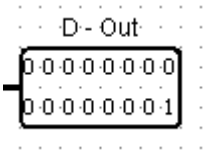
NOR:

0681 komutunu uyguladığımızda Reg(Rd) = ~(Reg(Rs) | Reg(Rt)) biçiminde halihazırda boş olan 1.register ve zero registerı norlanarak 2. registera sonuç olarak $(1111111111111111)_2$ yazmasını amaçlıyoruz.

GİRDİ	ÇIKTI
	

SLTU:

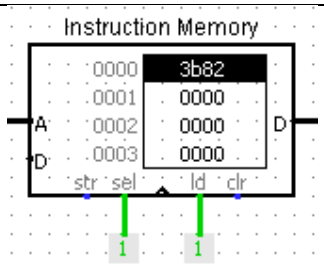
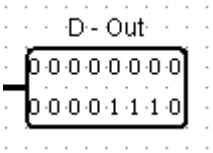
0F82 komutunu uyguladığımızda Reg(Rd) = Reg(Rs) unsigned< Reg(Rt) biçiminde zero registerı ile önceki komutta $(1111111111111111)_2$ ile doldurduğumuz 2. Register karşılaştırılmıştır.(r0<?r2) 6. registera sonuç olarak $(0000000000000001)_2$ yazmasını amaçlıyoruz.

GİRDİ	ÇIKTI
	

2.2. I-Type Komutlar

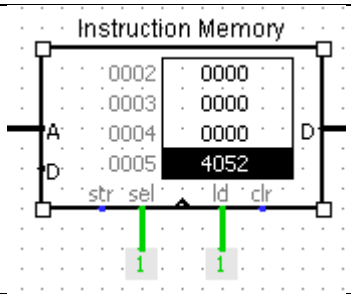
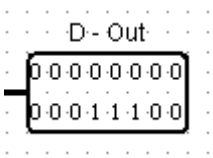
ADDI:

3b82 komutunu uyguladığımızda Reg(Rt) = Reg(Rs) + Immediate[5] biçiminde addi komutunu çalıştırarak 2. Registera $(01110)_2$ eklemesini amaçlıyoruz. Aşağıdaki tabloda görüldüğü üzere komut başarılı bir şekilde çıktı verdi.

GİRDİ	ÇIKTI
	

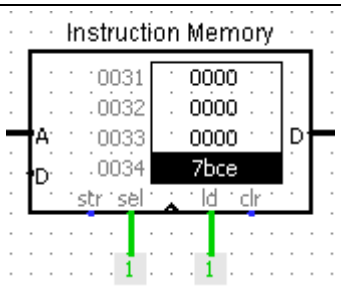
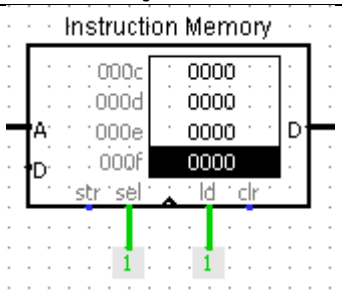
SLL:

4052 komutunu uyguladığımızda Reg(Rt) = Reg(Rs) << Immediate[4] biçiminde sll komutunu çalıştırarak 2. Registerda daha önce kaydettiğimiz $(01110)_2$ değerini bir bit lojik sola kaydırarak aynı registera kaydetmeyi amaçlıyoruz. Aşağıdaki tabloda görüldüğü üzere komut başarılı bir şekilde çıktı verdi.

GİRDİ	ÇIKTI
	

BNE:

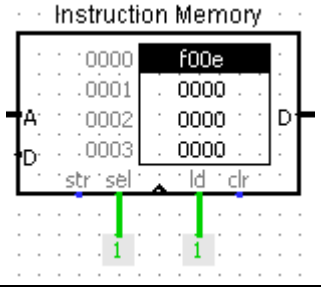
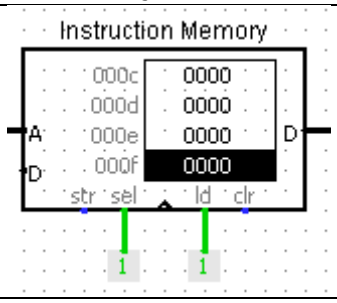
7bce komutunu uyguladığımızda Branch if (Reg(Rs) != Reg(Rt)) biçiminde bne komutunu çalıştırarak 1. Register $(00100)_2$ ile 6.register $(0111111111111000)_2$ içerisinde bulunan verilerin eşitliği kontrol edilerek eşitlik olmadığının bulunup 15.adrese $(01111)_2$ gitmesini amaçlıyoruz. Aşağıdaki tabloda görüldüğü üzere komut başarılı bir şekilde çalıştı.

GİRDİ	ÇIKTI
	

2.3. J-Type Komut

J:

f00e komutunu uyguladığımızda PC = PC + Immediate[11] biçiminde j komutunu çalıştırarak immediate data olarak verdiğimiz 11 bitlik $(00000001110)_2$ adres ve Program Counter'ın şu anki değeri olan 1'in toplanıp olan 15.adrese $(000f)_{16}$ gitmesini amaçlıyoruz. Aşağıdaki tabloda görüldüğü üzere komut başarılı bir şekilde çalıştı.

GİRDİ	ÇIKTI
	

Bu testler haricinde proje dosyasında verilen test dokümanının doldurulmuş hali aşağıdadır.

Initializing Registers (Testing I-Type ALU):

Instruction	Hexadecimal	Expected Result
lui 0x78f	0x878F	r1 = 0xf1e0 (shifted 5 bits left)
ori r2, r0, 4	0x2902	r2 = 4 = 0x0004
addi r3, r0, -2	0x3F83	r3 = -2 = 0xfffe (sign extension)
xori r4, r0, -2	0x3784	r4 = 0x001e (zero extension)

Testing R-Type ALU Instructions (NO RAW hazards – NO Forwarding):

Instruction	Hexadecimal	Expected Result
add r5, r1, r1	0x0949	r5 = 0xe3c0 (carry is ignored)
sub r6, r1, r2	0x0B8A	r6 = 0xf1dc
slt r7, r1, r2	0x0DCA	r7 = 1 (true) r1 < 0
sltu r5, r1, r2	0x0F4A	r5 = 0 (false)
and r6, r3, r4	0x019C	r6 = 0x001e
or r7, r1, r2	0x03CA	r7 = 0xf1e4
xor r5, r1, r3	0x054B	r5 = 0x0e1e
nor r6, r1, r2	0x078A	r6 = 0x0e1b
sll r7, r4, r2	Imm HATA!	r7 = 0x01e0
srl r5, r1, r2	Imm HATA!	r5 = 0x0f1e
sra r6, r1, r2	Imm HATA!	r6 = 0xff1e
ror r7, r3, r2	Imm HATA!	r7 = 0xefff

(Imm HATA!) olan kısımlar projede Immediate olarak verildiği için formata uymamaktadır.

Testing RAW hazards and Forwarding:

Instruction	Hexadecimal	Expected Result
add r5, r1, r1	0x0149	r5 = 0xe3c0
sub r6, r5, r4	0x0BAC	r6 = 0xe3a2 (depends on add)
and r7, r5, r6	0x01EE	r7 = 0xe380 (depends on add/sub)
ori r5, r5, 0xf	0x2BED	r5 = 0xe3cf (depends on add)

Testing SW and LW:

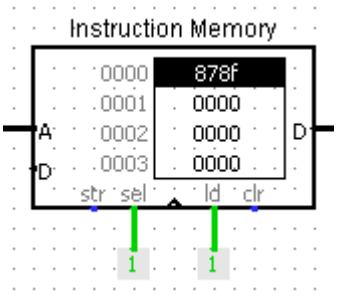
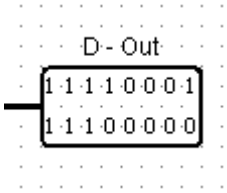
Instruction	Hexadecimal	Expected Result
sw r1, 0(r0)	0x6801	MEM[0] = 0xf1e0
sw r4, 1(r0)	0x6844	MEM[1] = 0x001e
lw r5, 0(r0)	0x6005	r5 = MEM[0] = 0xf1e0

Testing Load delay, stalling pipelining, and forwarding after LW:

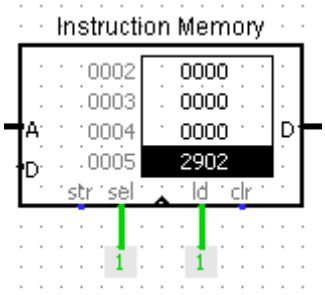
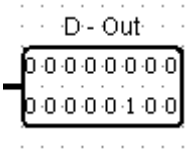
Instruction	Hexadecimal	Expected Result
lw r6, 1(r0)	0x6046	r6 = MEM[1] = 0x001e
andi r7, r6, 0xb	0x22F7	r7 = 0x000a (stall 1 cycle & forward)
sw r7, 2(r0)	0x6887	MEM[2] = 0x000a (forwarded from andi)
lw r5, 0(r0)	0x6005	r5 = MEM[0] = 0xf1e0
sw r5, 3(r0)	0x68C5	MEM[3] = 0xf1e0 (forwarded from lw)

Initializing Registers (Testing I-Type ALU)

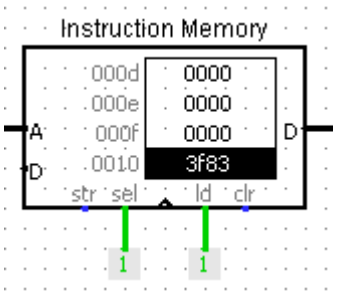
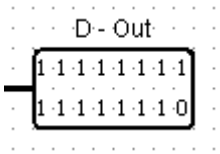
1. lui 0x78f (878F)

GİRDİ	ÇIKTI (r1)
	

2. ori r2, r0, 4 (2902)

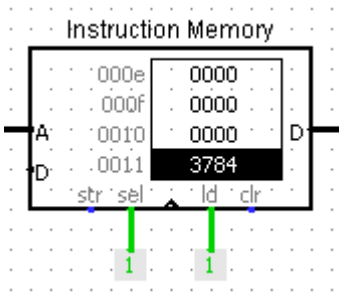
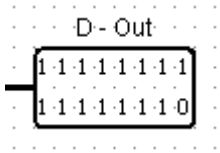
GİRDİ	ÇIKTI (r2)
	

3. addi r3, r0, -2 (3F83)

GİRDİ	ÇIKTI (r3)
	

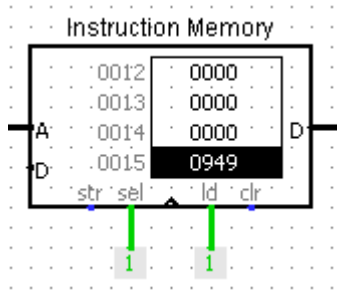
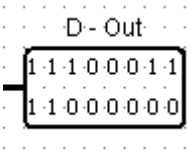
4. xori r4, r0, -2 (3784)

   Verilen test tablosunda beklenilen sonuç zero extension kullanılarak hesaplandığı için $(0000000000011110)_2$ sonucu istenmiş. Biz projede sign extension kullandığımız için sonuç $(1111111111111110)_2$ olarak çıkmıştır. Bundan dolayı takip eden işlemler farklı çıktı vermiştir.

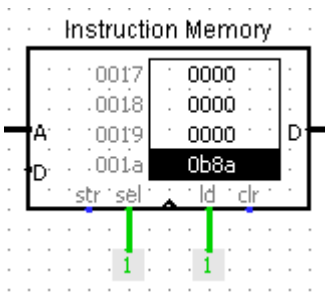
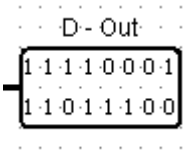
GİRDİ	ÇIKTI (r4)
	

Testing R-Type ALU Instructions (NO RAW hazards – NO Forwarding)

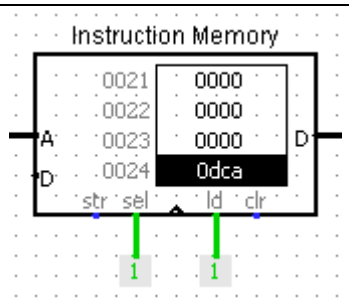
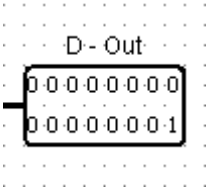
1. add r5, r1, r1 (0949)

GİRDİ	ÇIKTI (r5)
	

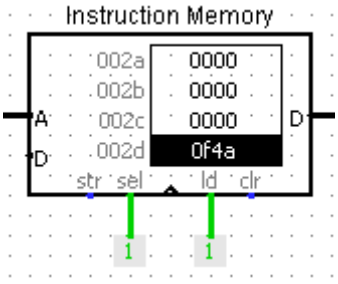
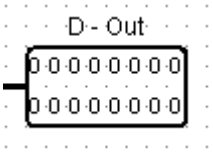
2. sub r6, r1, r2 (0B8A)

GİRDİ	ÇIKTI (r6)
	

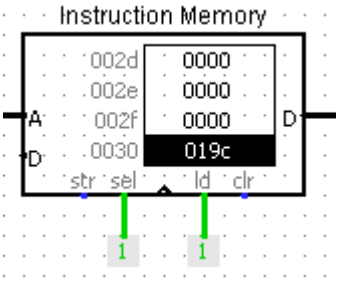
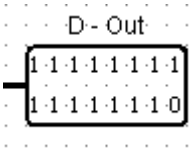
3. slt r7, r1, r2 (0DCA)

GİRDİ	ÇIKTI (r7)
	

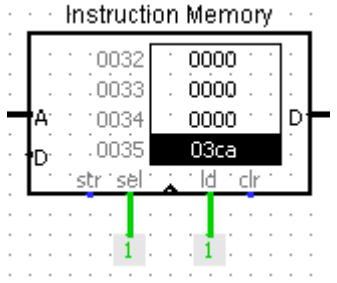
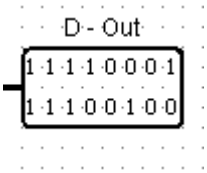
4. sltu r5, r1, r2 (0F4A)

GİRDİ	ÇIKTI (r5)
	

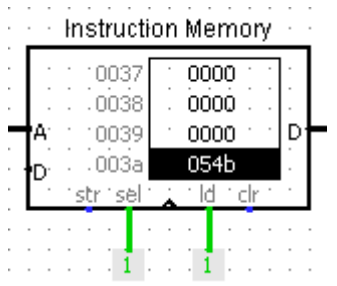
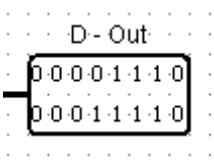
5. and r6, r3, r4 (019C)

GİRDİ	ÇIKTI (r6)
	

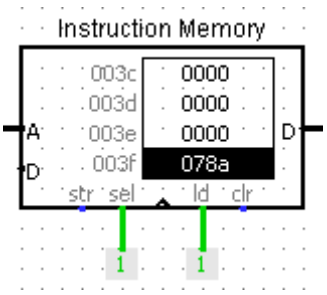
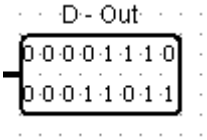
6. or r7, r1, r2 (03CA)

GİRDİ	ÇIKTI (r7)
	

7. xor r5, r1, r3 (054B)

GİRDİ	ÇIKTI (r5)
	

8. nor r6, r1, r2 (078A)

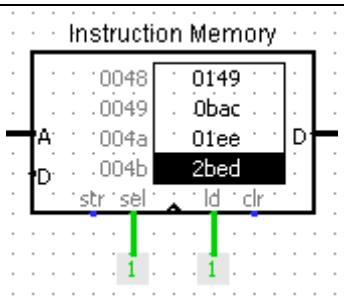
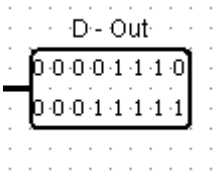
GİRDİ	ÇIKTI (r6)
	

- sll r7, r4, r2
- srl r5, r1, r2
- sra r6, r1, r2
- ror r7, r3, r2

  işlemlerinin tanımı hatalı olduğu için işleme sokulamamıştır.

Testing RAW hazards and Forwarding

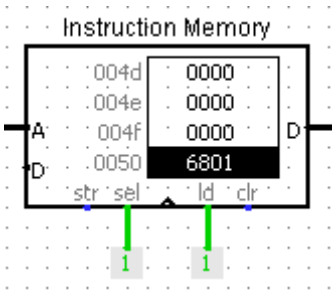
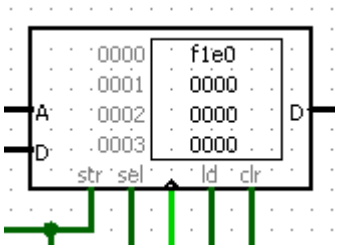
1. add r5, r1, r1 (0149)
2. sub r6, r5, r4 (0BAC)
3. and r7, r5, r6 (01EE)
4. ori r5, r5, 0xf (2BED)

GİRDİ	ÇIKTI (r5)
	

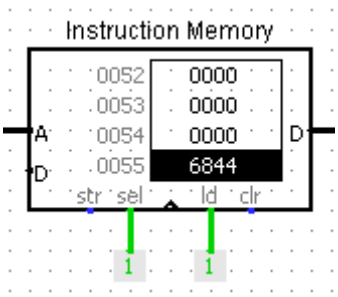
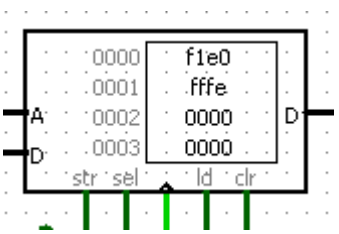
Forwarding ve data hazard konusunda bazı aksaklıklar bulunduğundan sonuç hatalı çıkmıştır.

Testing SW and LW

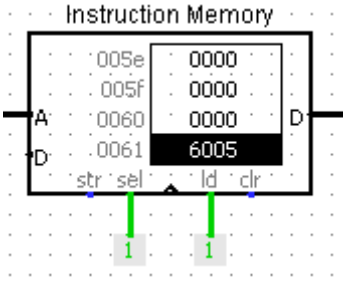
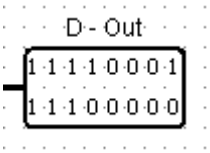
1. sw r1, 0(r0) (6801)

GİRDİ	ÇIKTI (Memory)
	

2. sw r4, 1(r0) (6844)

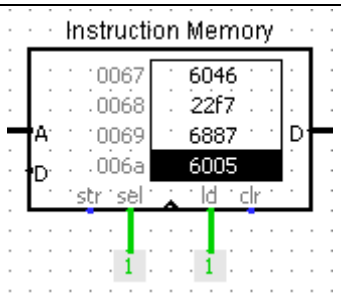
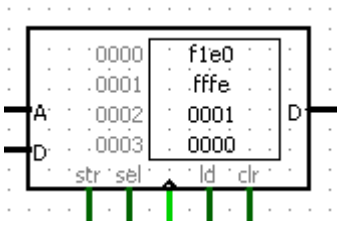
GİRDİ	ÇIKTI (Memory)
	

3. lw r5, 0(r0) (6005)

GİRDİ	ÇIKTI (r5)
	

Testing Load delay, stalling pipelining, and forwarding after LW:

1. lw r6, 1(r0) (6046)
2. andi r7, r6, 0xb (22F7)
3. sw r7, 2(r0) (6887)
4. lw r5, 0(r0) (6005)
5. sw r5, 3(r0) (68C5)

GİRDİ	ÇIKTI (r5)
	

Forwarding kısımlarında hata bulunuyor. Bundan dolayı çıktıda hata oldu.