Örnek

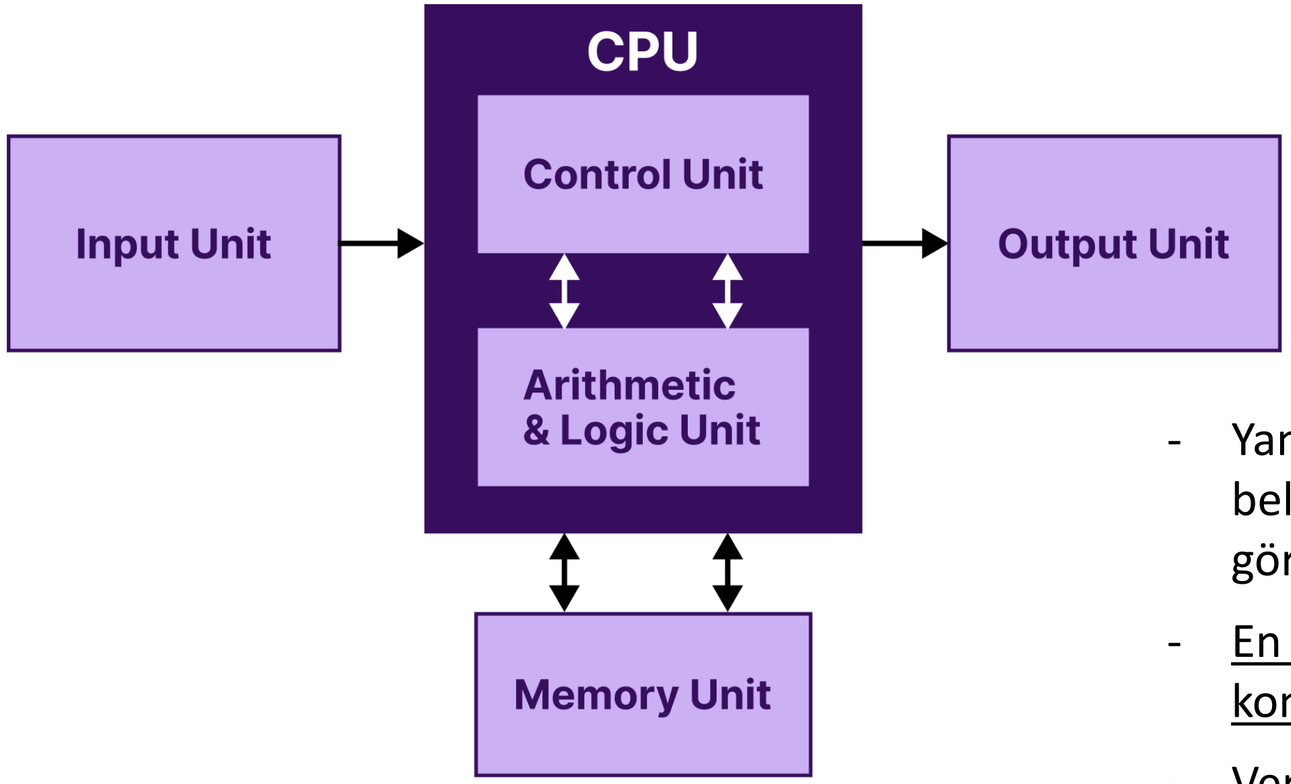
8-bit CPU Tasarımı

Doç. Dr. Serkan Dereli

ISERA

ISERA AKADEMİ

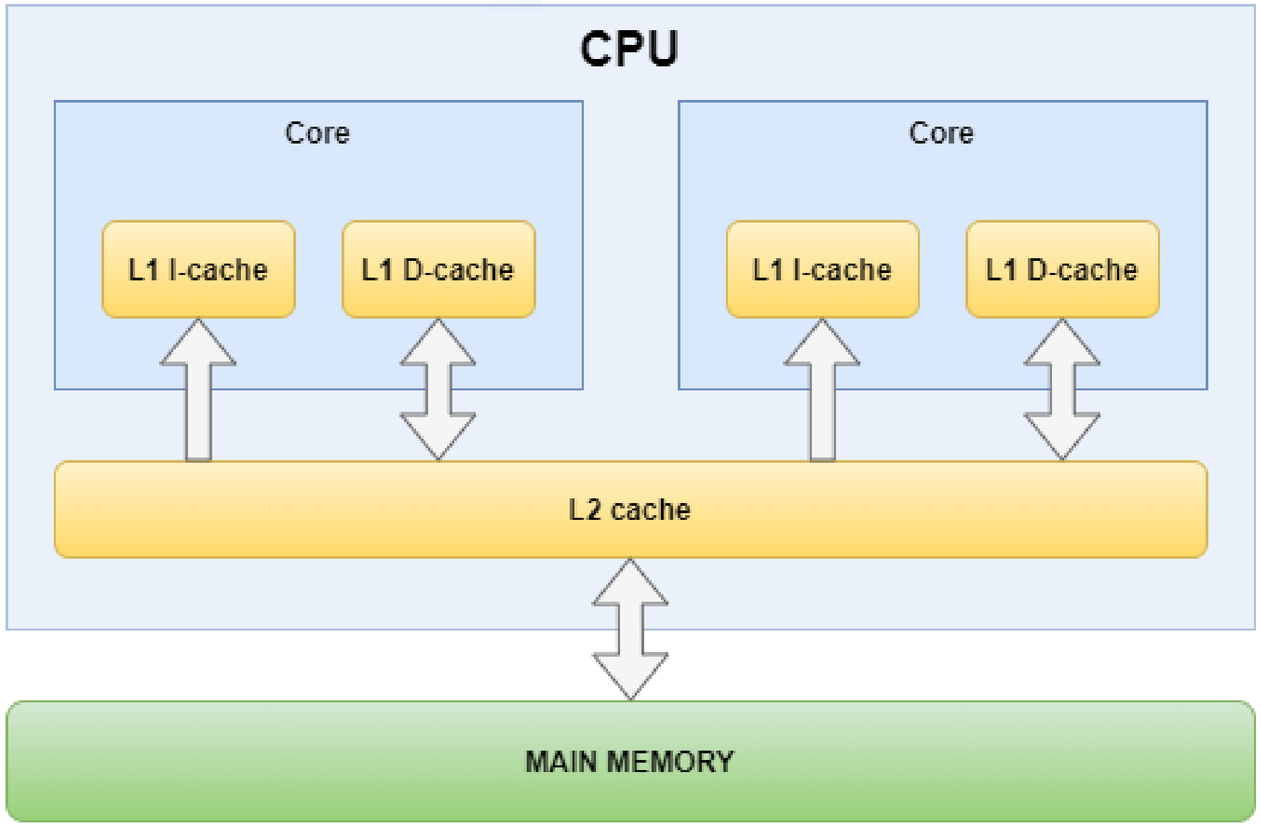
Genel CPU ve Bellek Konumu



- Yanda temel CPU mimarisi, komponentleri, bellek ve giriş/çıkış birimleriyle bağlantısı görülmektedir.
- En temel/basit genel CPU yapısı Veriyolu ve kontrol biriminden oluşur.
- Veriyolu ise kaydediciler ve ALU birimlerinden oluşmaktadır.

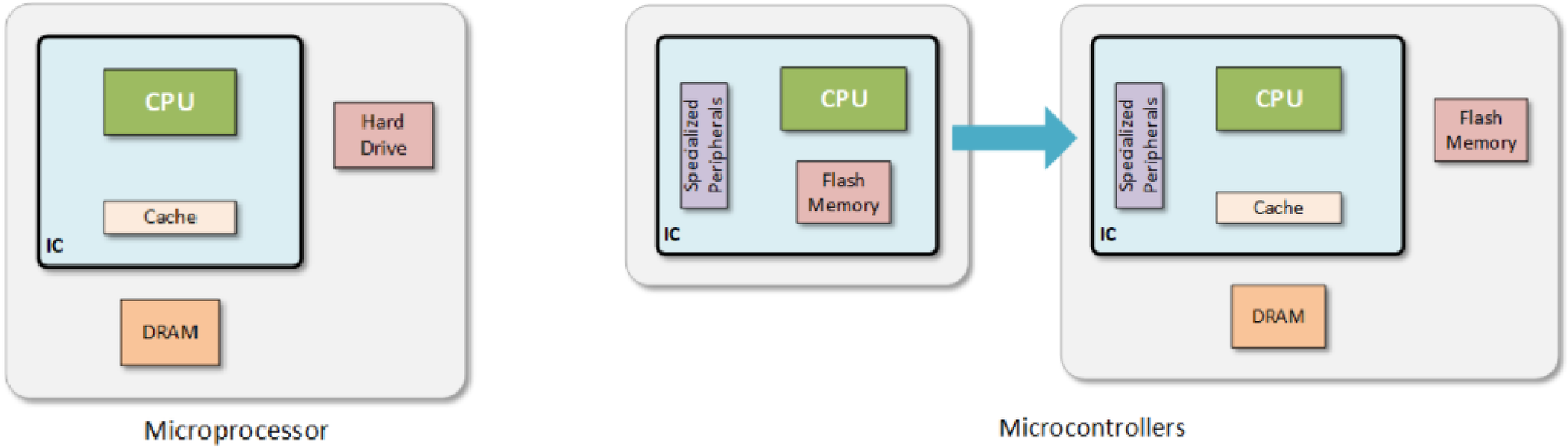
ISERA AKADEMİ

Genel CPU ve Bellek Konumu



- Modern CPU’lar ise kontrol birimi, veriyolu ve önbellek birimlerinden oluşmaktadır.
- Günümüzde CPU’lar çok çekirdeklidir.
- L1, çekirdekler içerisine yerleştirilirken L2 önbellek CPU yongası içerisinde çekirdek paylaşımlı olabilmektedir.
- Veriyolu hem komutları hem de verileri L1 önbellek biriminden alır.

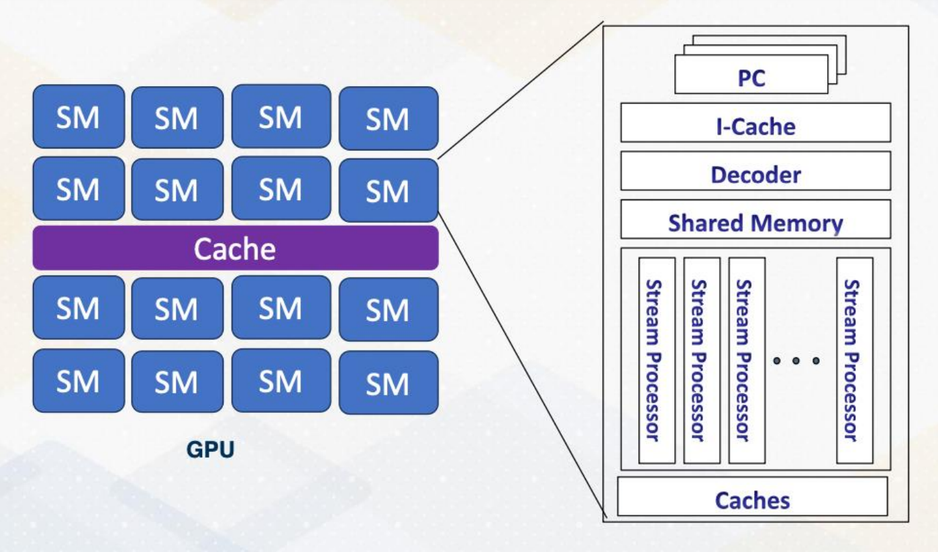
Genel CPU ve Bellek Konumu

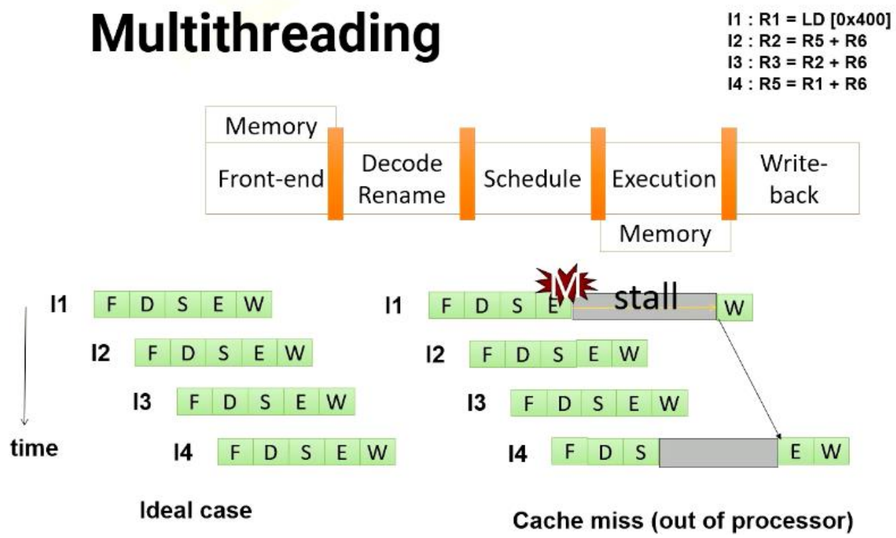


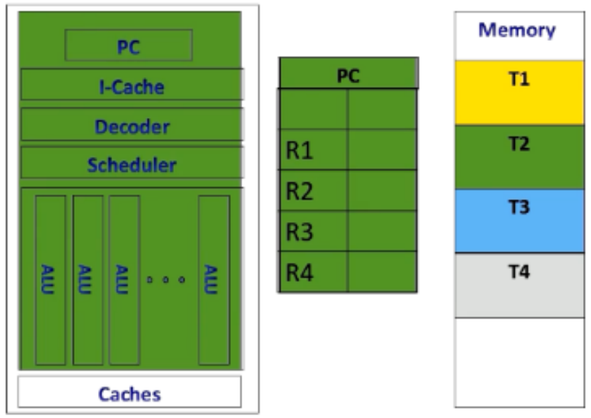
Günümüzde CPU çekirdekleri MPU gibi genel amaçlı mimarilerin yanı sıra ve özel amaçlı mimarilerde (MCU, GPU) işlem birimi veya çekirdek olarak kullanılır.

ISERA AKADEMİ

Genel GPU Mimarisi







GPU'lar işleri paralel gerçekleştirmek için çoklu Stream Multiprocessor mimarisi kullanır.

Her bir SM pek çok işlemci çekirdeği barındırır.

İşler Schedule aşamasında Therad yapılarına bölüştürülür ve her thread farklı bir Stream Processor de işlenir.

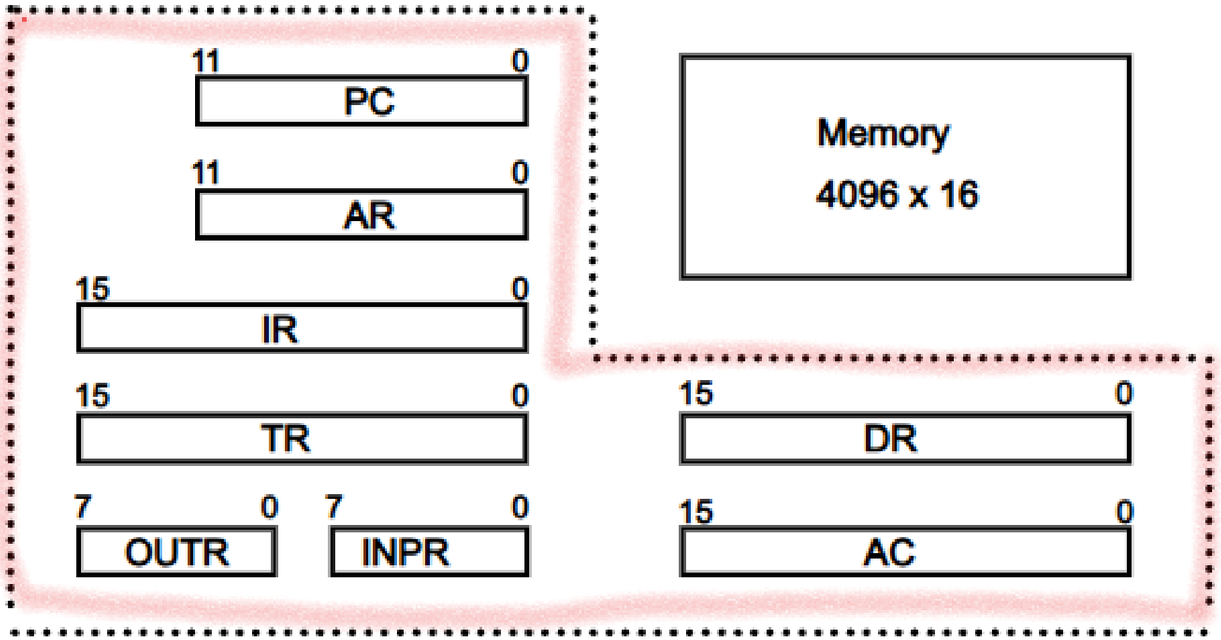
Bölüm-1

8-bitlik CPU tasarımı

ZindeRV8

ISERA

Mano Bilgisayar Komutları



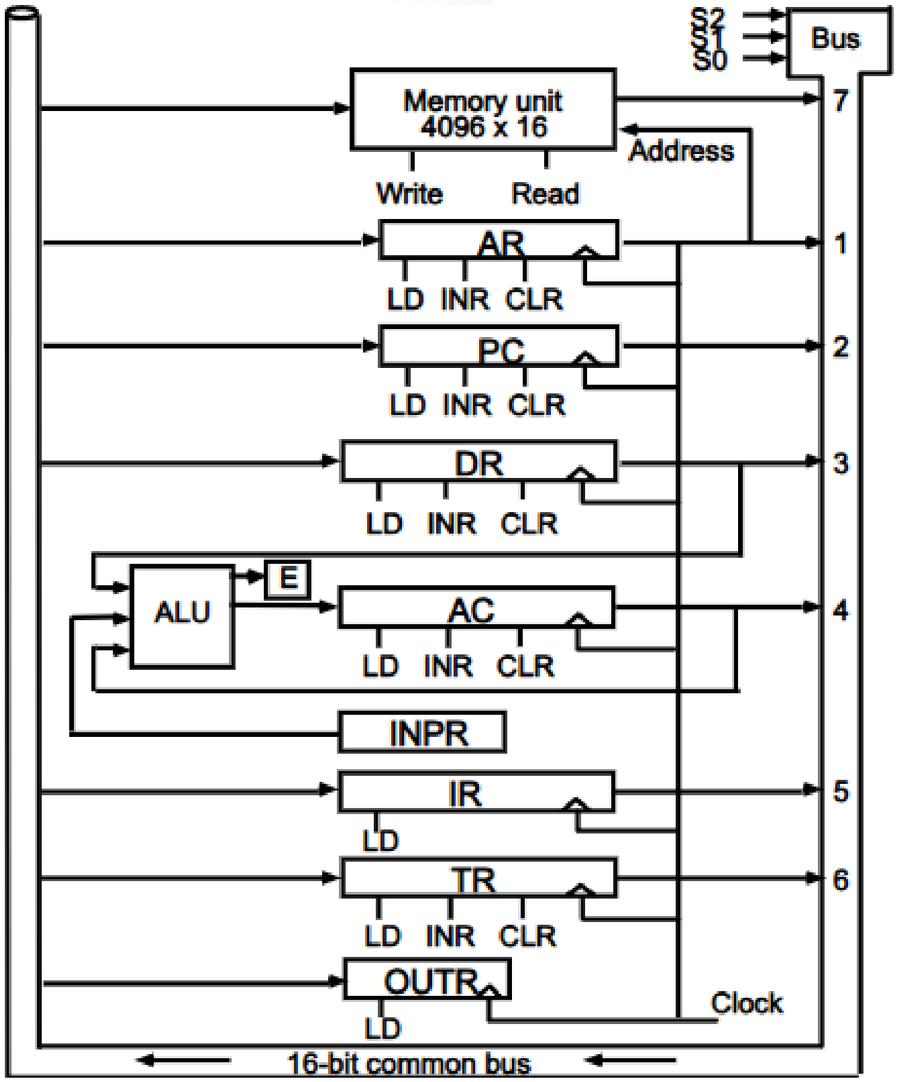
Çizelge 5.1 Temel bilgisayar için yazaçların listesi

Yazaç Sembolü	bit sayıları	Yazaç ismi	Fonksiyonu
DR	16	Veri yazacı	bellek verisini tutar
AR	12	Adres yazacı	bellek için adres tutar
AC	16	Birikeç	İşlemci yazacı
IR	16	Buyruk yazacı	buyruğun kodunu tutar
PC	12	Program sayıcı	buyruğun adresini tutar
TR	16	Geçici yazaç	Geçici veriyi tutar
INPR	8	Giriş yazacı	Giriş karakterini tutar
OUTR	8	Çıkış yazacı	Çıkış karakterini tutar

- Yukarıda ise Mano Bilgisayar da bulunan kaydediciler görülmektedir.
- Yukarıda ise kaydedicilerle ilgili detaylı bilgiler yer almaktadır.

ISERA AKADEMİ

Mano Sistem BUS Yapısı ve Komutlar

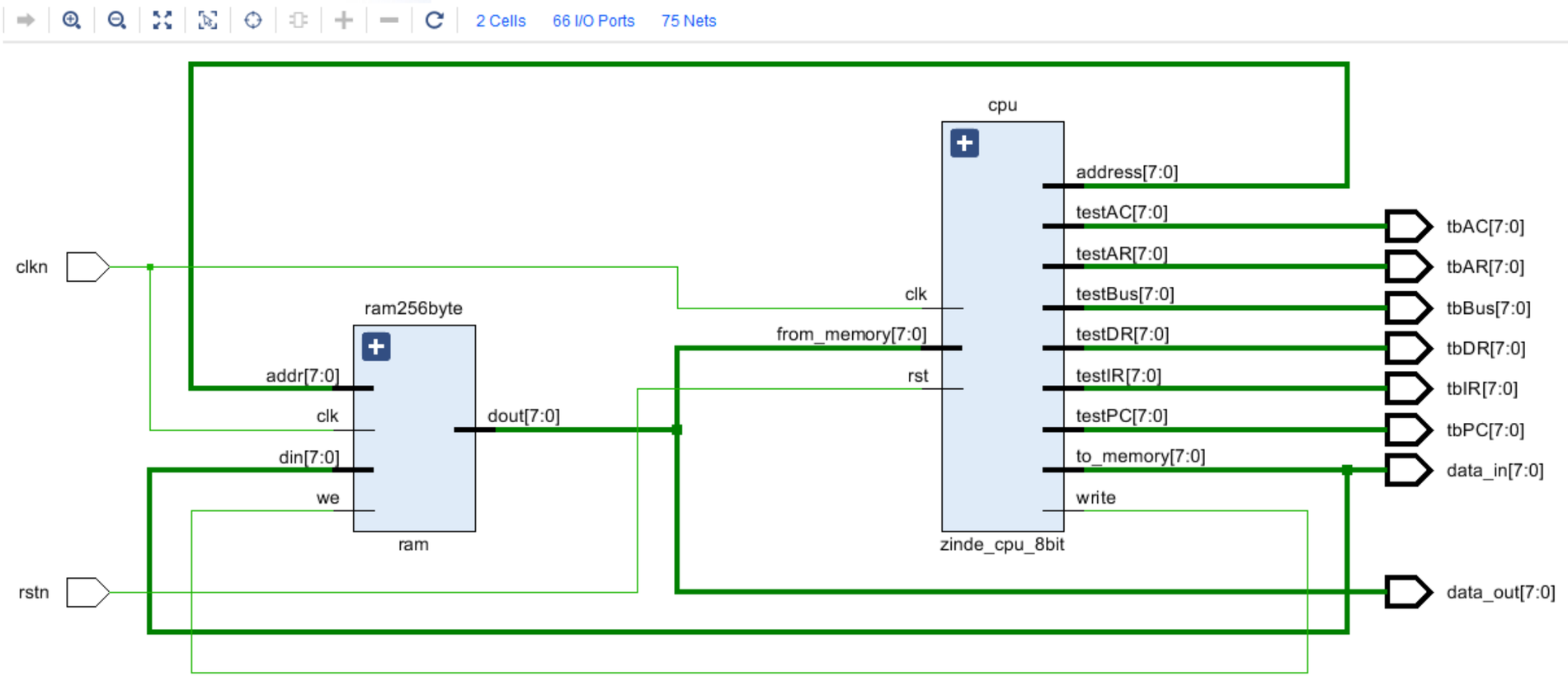


Çizelge 5.2. Temel bilgisayar buyrukları

Sembol	Onaltılı kodu $I = 0$	Onaltılı kodu $I = 1$	Tanımlama
AND	0XXX	8XXX	Bellekteki kelimeyi VE leyerek *AC* ye aktar
ADD	1XXX	9XXX	Bellek kelimesini toplayarak *AC* ye aktar
LDA	2XXX	AXXX	Bellek kelimesini *AC* ye yükle
STA	3XXX	BXXX	*AC* nin içeriğini belleğe depola
BUN	4XXX	CXXX	Şartsız dallan
BSA	5XXX	DXXX	Dallan ve geri dönüş adresini sakla
ISZ	6XXX	EXXX	Arttır ve eğer sıfır ise atla
CLA		7800	*AC* yi sil
CLE		7400	*E* yi sil
CMA		7200	*AC* nin tümleyenini al
CME		7100	*E* nin tümleyenini al
CIR		7080	*AC* ve E nin içeriğini sağa dairesel olarak kaydır
CIL		7040	*AC* ve E nin içeriğini sola dairesel olarak kaydır
INC		7020	*AC* nin değerini 1 arttır.
SPA		7010	*AC* pozitif ise bir sonraki buyruğu atla
SNA		7008	*AC* negatif ise bir sonraki buyruğu atla
SZA		7004	*AC* sıfır ise bir sonraki buyruğu atla
SZE		7002	*E* sıfır ise bir sonraki buyruğu atla
HLT		7001	Programı durdur
INP		F800	Giriş karakterini *AC* ye al
OUT		F400	*AC* den çıkış karakterini al
SKI		F200	giriş bayrağını atla
SKO		F100	çıkış bayrağını atla
ION		F080	Kesmeyi aktif yap
IOF		F040	Kesmeyi pasif yap

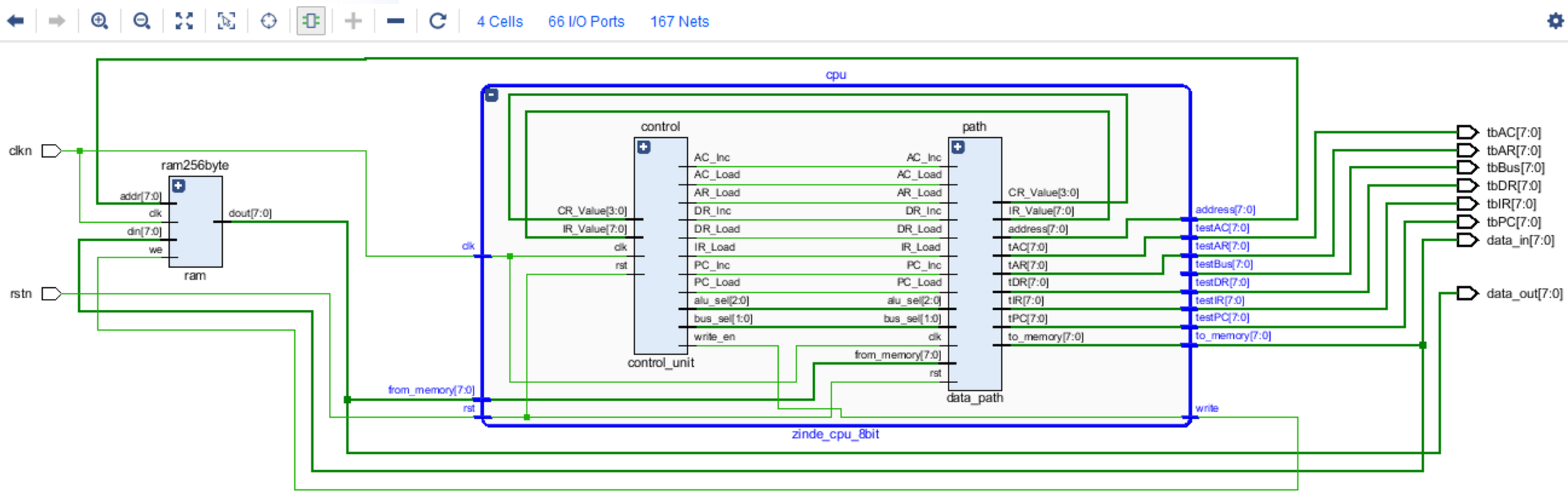
ISERA AKADEMİ

ZindeRV8 CPU, (temel mimari)



Bu proje kapsamında geliştirilen 8-bitlik CPU mimarisi yukarıda görülmektedir.

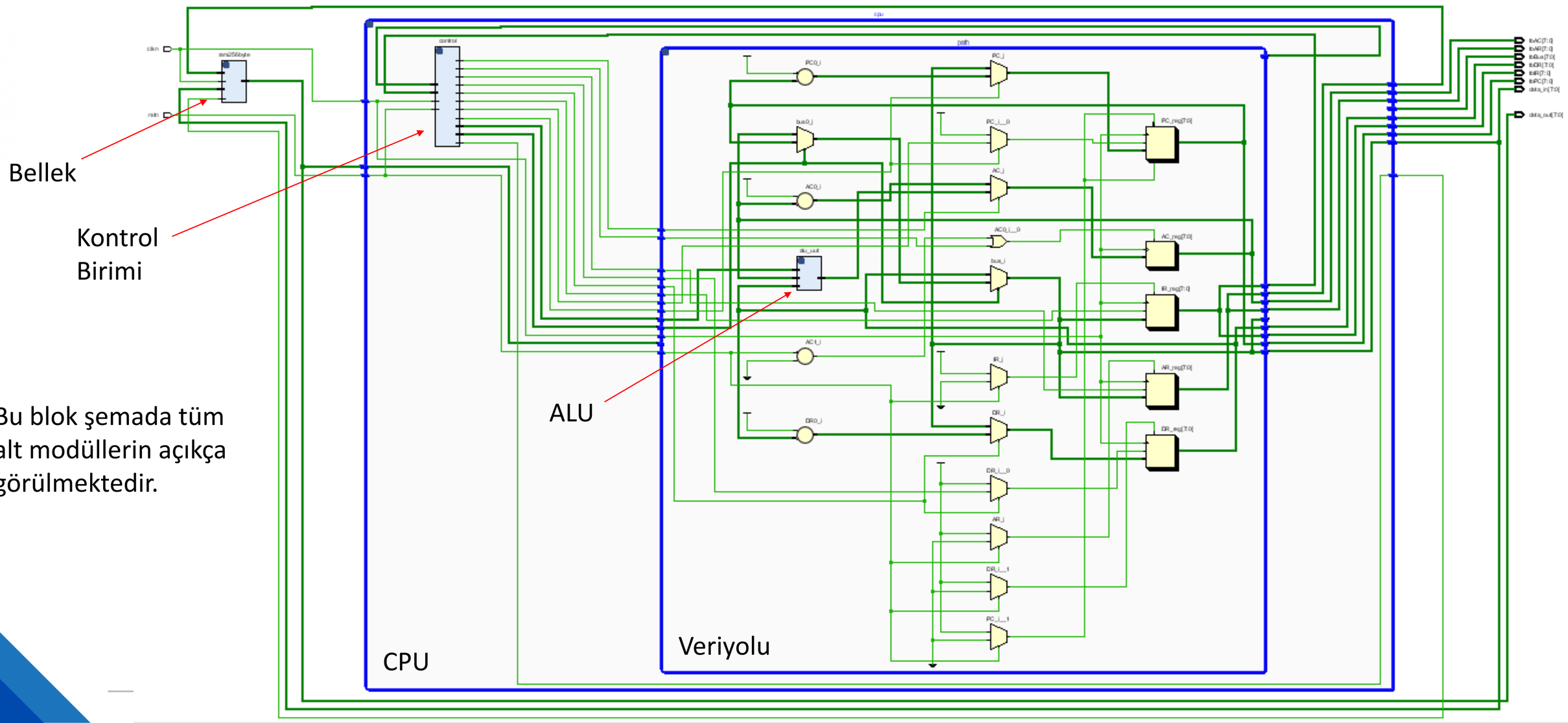
ZindeRV8 CPU, (temel mimari)



Yukarıda CPU iç yapısı görülmektedir. Bu yapıda Veriyolu ve kontrol birimi açıkça görülmektedir.

ISERA AKADEMİ

ZindeRV8 CPU, (temel mimari)



Bu blok şemada tüm alt modüllerin açıkça görülmektedir.

ISERA AKADEMİ

Performans Görevi-3 Kapsamı

Aşağıdaki özelliklere sahip bilgisayar mimarisi tasarlanacaktır:

- Kendine özgü ismi olmalı (özümüze uygun olmalı),
- Kendine özgü registerlar, ALU, bellek elemanı…… olmalı,
- Komut yapısı belirlenmeli,
- Komut kümesi olmalı (giriş ve çıkış komutları muhakkak olmalı),
- Her bir komutun mikro işlem adımları yazılmalı,
- Kontrol devresi hardwired (lojik mantık) olarak tasarlanmalı
- Kesmeyi desteklemeli,
- En az 4 adet adresleme modunu desteklemeli
- Assembly dil kuralları olmalı

NOT: Yukarıdaki aşamaları Performans-2 de gerçekleştirmiştiniz, Performans-3 te oluşturduğunuz blok şema kapsamında HDL ile CPU tasarımınızı gerçekleştirmeniz beklenmektedir.

- Not-1: Bu proje kapsamında da çalışmalarınızı Vivado, ISE vb. uygulamalarla VHDL/Verilog dilinde yazabilirsiniz. Yazdığınız kodun muhakkak simülasyonunu gerçekleştirmelisiniz. Ancak son proje kapsamında sunulacaklar, sizlere dağıtılacak olan deney setleri üzerinde çalışacağından şimdiden gerekli önlemleri almalısınız.(dağıtılacak deney setleri ISE ortamını kullanıyor.)

- Not-2: Dönem sonundaki proje sunumlarında dönem başında ilan edilen şekilde PS/2 klavye aracılığıyla(metin yazımı olabilir, kod girilmesi ……) tasarladığınız bilgisayar mimarisi kullanılarak 7 segmentli display üzerinde çıkış elde edilecektir. Detaylar bir sonraki proje kapsamında ilan edilecektir.

ISERA AKADEMİ

RISC Mimari Özellikleri

1. Basit ve Sabit Uzunlukta Talimatlar

a) Tüm talimatlar genellikle aynı uzunluktadır (örneğin RISC-V'de 32 bit).
b) Talimatlar genellikle tek bir işlem yapar.
c) Bu yapı, donanımı sadeleştirir ve hızlı instruction decode sağlar.

3. Çok Sayıda Register

a) RISC mimarilerinde genellikle çok sayıda genel amaçlı register bulunur (örneğin RISC-V'de 32 tane).
b) Bu, değişken saklama ve ara hesaplamaları bellek yerine register'larda tutarak daha hızlı işlem yapılmasını sağlar.

5. Basit Adresleme Modları

a) RISC mimarileri sadece birkaç temel adresleme modu sunar (örneğin base + offset).
b) Karmaşık adresleme modları yoktur (örneğin CISC'teki "base + index * scale + displacement" gibi yapılar RISC'te bulunmaz).

2. Register-Register Mimarisi (Load/Store Mimarisi)

a) Bellek erişimi sadece load/store komutlarıyla yapılır.
b) Diğer tüm işlemler (toplama, mantıksal işlemler, vb.) yalnızca register'lar arasında gerçekleştirilir.

4. Sınırlı ve Basitleştirilmiş Talimat Kümesi

a) Komut sayısı azdır ve her biri donanım tarafından tek saat döngüsünde (ideal olarak) çalışacak şekilde tasarlanır.
b) Karmaşık işlemler (örneğin string işlemleri veya bellekten doğrudan toplama gibi) donanıma değil, yazılıma bırakılır.

6. Donanım Basitliği ve Boru Hattı (Pipelining) Desteği

a) Basit komutlar sayesinde RISC mimarileri boru hattı (pipeline) uygulamasına uygundur.
b) Bu, komutların ardışık aşamalarda paralel işlenmesini kolaylaştırır.

8-bitlik Bilgisayar Organizasyonu (Zinde CPU)

AKIŞ (fetch, decode, execute)

1. PC'ye ilk komut adresi yüklenecek (PC<-M)
2. PC içeriği AR'ye aktarılacak (AR<-PC)
    1. PC içeriği 1 artırılacak
3. AR içeriği doğrudan belleğe aktarılacak
4. M[AR] içeriği IR'ye aktarılacak

5. Komutlar 8-bit olacak, bellekte 8-bit veri saklanabilecek, ilk adresteki 8-bit komut, sonraki adresteki 8-bit veri/adres olacak. M[AR] içeriği DR'ye aktarılır.
6. DR içeriği ALU ile AC'ye aktarılır.

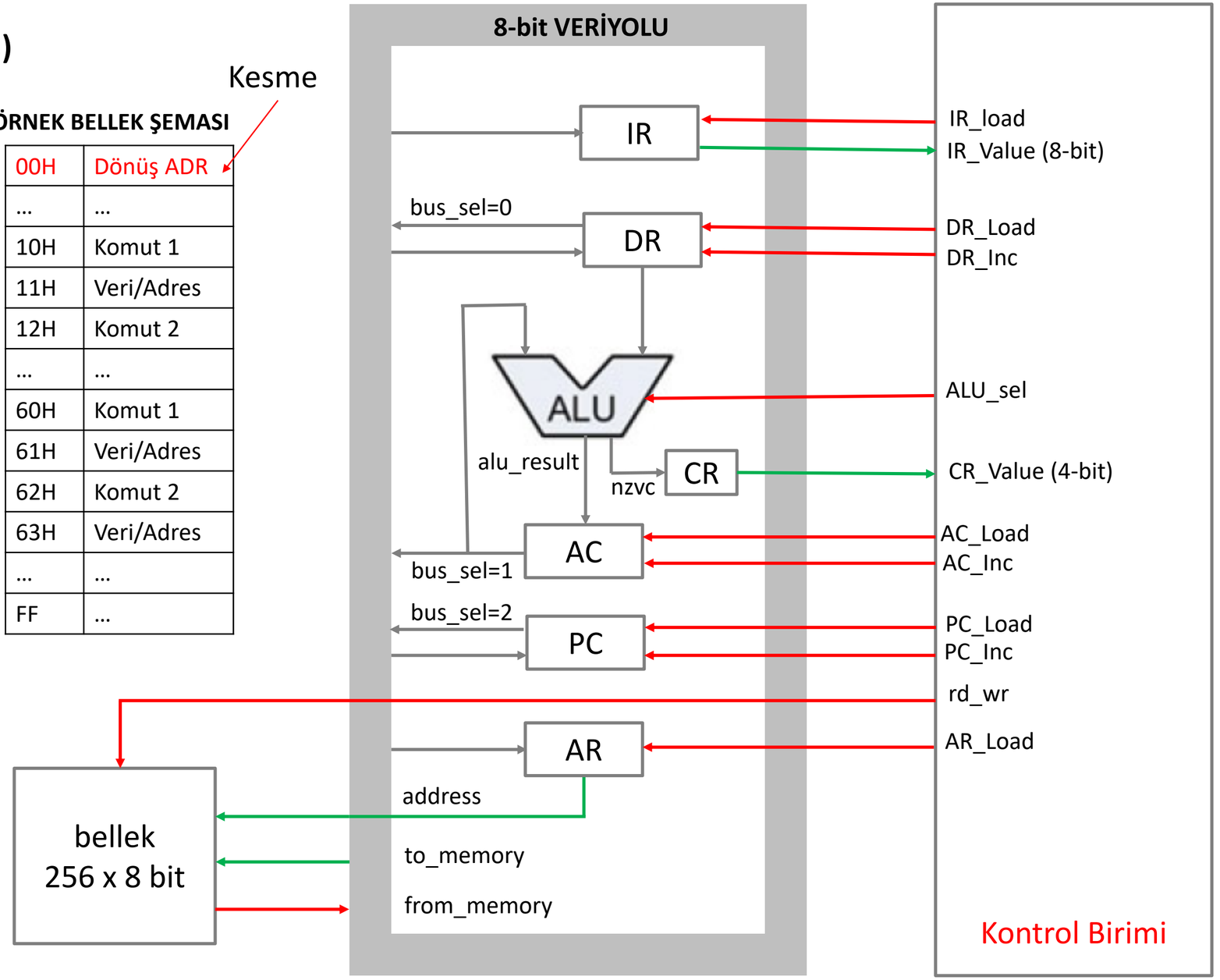
ÖRNEK KOMUT ÇALIŞTIRMA

1- LDA #5 → burada 5 bir veri ve DR ← 5
2- ADD $80 → burada 80 bir adres ve DR ← M[80]
3- STA $85 → burada 80 bir adres ve M[80] ← AC

Adres: 8-bit
Komut: 8-bit
IR: 8-bit
DR: 8-bit
AR: 8-bit
PC: 8-bit

ÖRNEK BELLEK ŞEMASI

00H	Dönüş ADR
…	…
10H	Komut 1
11H	Veri/Adres
12H	Komut 2
…	…
60H	Komut 1
61H	Veri/Adres
62H	Komut 2
63H	Veri/Adres
…	…
FF	…



ISERA AKADEMİ

Komut Yapısı

Komut Türleri

İşlemcimiz 8-bit olup ismi **ZindeRV8 CPU** olarak anılacaktır.

Buradaki 8-bit ifadesi kaydedicilerin ve dolayısıyla veriyolunun 8-bit olmasından ileri gelmektedir.

ZindeRV8 CPU, 8-bitlik komutlardan oluşmaktadır ve bu komutların yapısı yanda görüldüğü gibi olacaktır.

Komutlar 4 farklı biçimde veriye ulaşabilecektir. Bu da adresleme türü olarak geçmekte olup yine yandaki tabloda görülmektedir.

F (2-bit)	Adresleme Türü (2-bit)	Opcode (4-bit)
00	(Kaydedici) R – 00	XXXX
00	(Immediate) I – 01	XXXX
00	(Bellek) S – 10	XXXX
00	(Dallanma) B – 11	XXXX

ISERA AKADEMİ

Komutlar, Bellek Tipi Adresleme

Komut Türleri
R Tip: Kaydedici,
I tip: Immediate (sabit değerler),
S tip: Belleğe kayıt (Store)
B tip: Dallanma (Branch)

Komut uzunluğu: 8-bit,

R→ 00 – 00[3:0]
I → 00 – 01[3:0]
S → 00 – 10[3:0] 1 – 10[4:0]
B → 00 – 11[3:0]

S (store) Tipi Komutlar			
AND	0000	Ve	AC ile adresteki değeri AND'le.
OR	0001	Veya	AC ile adresteki değeri OR'la.
XOR	0010	Özel veya	AC ile adresteki değeri EXOR'la.
ADD	0011	Toplama	AC ile adresteki değeri topla
SUB	0100	Çıkarma	AC'den adresteki değeri çıkar
LDA	0101	Yükle	adresteki değeri AC'ye yükle
STA	0110	Kaydet	AC'deki değeri adrese kaydet.

Örnekler:
LDA $20, komutu 20 nolu adresteki veriyi AC'ye yükler
Örnek komut kodu: 00 10 0101 0001 0100

ADD $30, komutu 30 nolu adresteki veriyi AC ile toplayıp sonucu yine AC'ye aktarır.
Örnek komut kodu: 00 10 0011 0001 1110

Komutlar, Immediate Tipi Adresleme

Komut Türleri
R Tip: Kaydedici,
I tip: Immediate (sabit değerler),
S tip: Belleğe kayıt (Store)
B tip: Dallanma (Branch)

Komut uzunluğu: 8-bit,

R→ 00 – 00[3:0]
I → 00 – 01[3:0]
S → 00 – 10[3:0] 1 – 10[4:0]
B → 00 – 11[3:0]

I Tipi Komutlar			
ANDI	0000	Ve	AC ile sabit değeri AND'le.
ORI	0001	Veya	AC ile sabit değeri OR'la.
XORI	0010	Özel veya	AC ile sabit değeri EXOR'la.
ADDI	0011	Toplama	AC ile sabit değeri topla
SUBI	0100	Çıkarma	AC'den sabit değeri çıkar
LDAI	0101	Yükle	sabit değeri AC'ye yükle

LDA #5, komutu doğrudan 5 verisini AC'ye yükler.
Örnek komut kodu: 00 01 0101 0000 0101
AC sıfırlanacağı zaman doğrudan bu komut ile işlem gerçekleştirilebilecektir. Dolayısıyla AC için ilave bir sıfırlama bağlantısına gerek olmayacaktır.

Komutlar, Kaydedici Tipi Adresleme

Komut Türleri
R Tip: Kaydedici,
I tip: Immediate (sabit değerler),
S tip: Belleğe kayıt (Store)
B tip: Dallanma (Branch)

Komut uzunluğu: 8-bit,

R→ 00 – 00[3:0]
I → 00 – 01[3:0]
S → 00 – 10[3:0] 1 – 10[4:0]
B → 00 – 11[3:0]

R Tipi Komutlar		
RRA	0000	AC'yi sağa döndür
RLA	0001	AC'yi sola döndür
SRA	0010	AC'yi sağa kaydır
SLA	0011	AC'yi sola kaydır.
CMA	0100	AC'nin tümleyenini al
INC	0101	AC'yi 1 artır
DEC	0110	AC'yi 1 azalt
SPA	0111	Skip if AC is Positive
SNA	1000	Skip if Ac is Negative
SZA	1001	Skip if AC is Zero
BRK	1111	Programı durdur

Komutlar, Dallanma Tipi Adresleme

Komut Türleri
R Tip: Kaydedici,
I tip: Immediate (sabit değerler),
S tip: Belleğe kayıt (Store)
B tip: Dallanma (Branch)

Komut uzunluğu: 8-bit,

R→ 00 – 00[3:0]
I → 00 – 01[3:0]
S → 00 – 10[3:0] 1 – 10[4:0]
B → 00 – 11[3:0]

B (branch) Tipi Komutlar		
JMP	0000	Belirlenen bir satıra veya adrese atla
BEQ	0001	AC ile ilgili değer eşitse sonraki komutu atla
BNE	0010	AC ile ilgili değer eşit değilse sonraki komutu atla

Bölüm-2

Kontrol Birimi

ISERA

Kontrol Birimi

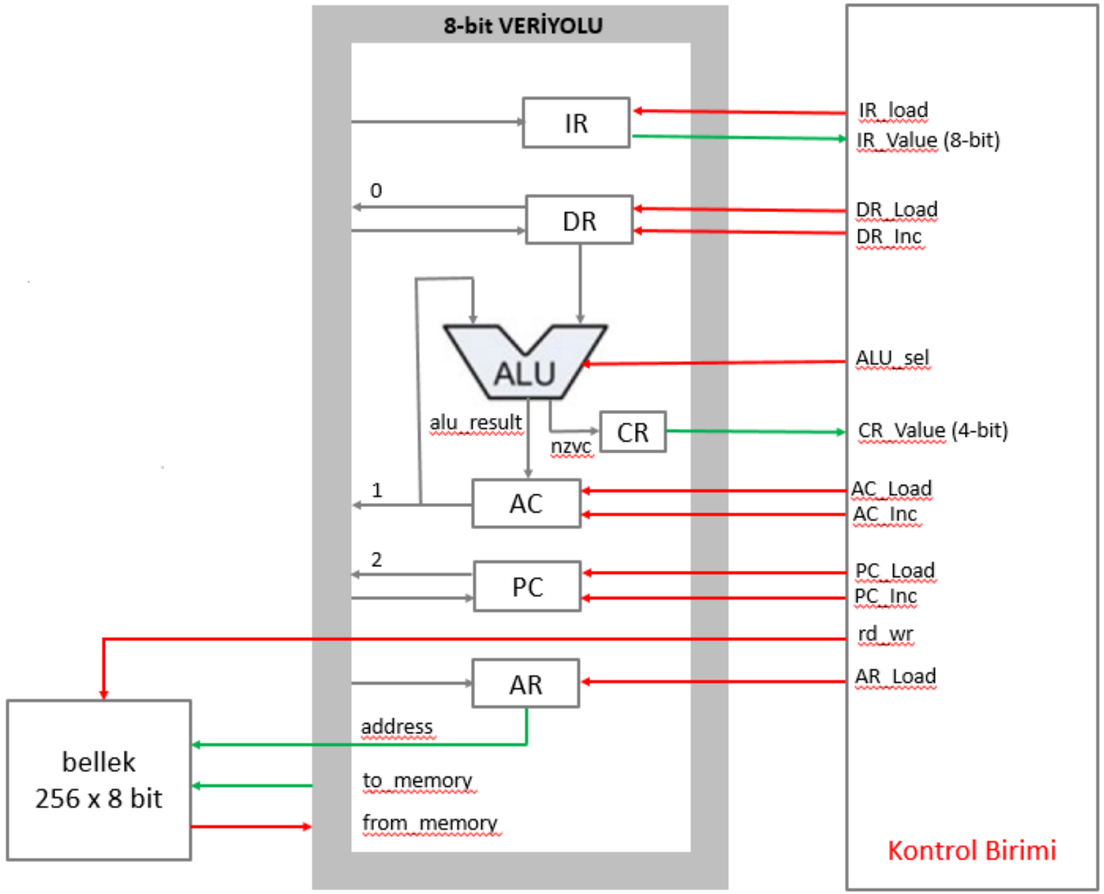
Fetch aşaması

1. Programın ilk komutu adresten getirilecek
   1. Adım
   T0: Bus <- PC, Bus_Sel=11
   T1: AR <- Bus, AR_Load=1

2. Adres bilgisi RAM belleğe iletilir
   2. Adım
   T1: address <- AR

3. Bellekten okunan değer Veriyolu üzerinden IR ye aktarılır
   3. Adım
   T2: Bus <- from_memory
   T3: IR <- Bus, IR_Load=1

4. IR değeri çözülmek üzere kontrol birimine aktarılır
   T3: IR_Value <-IR



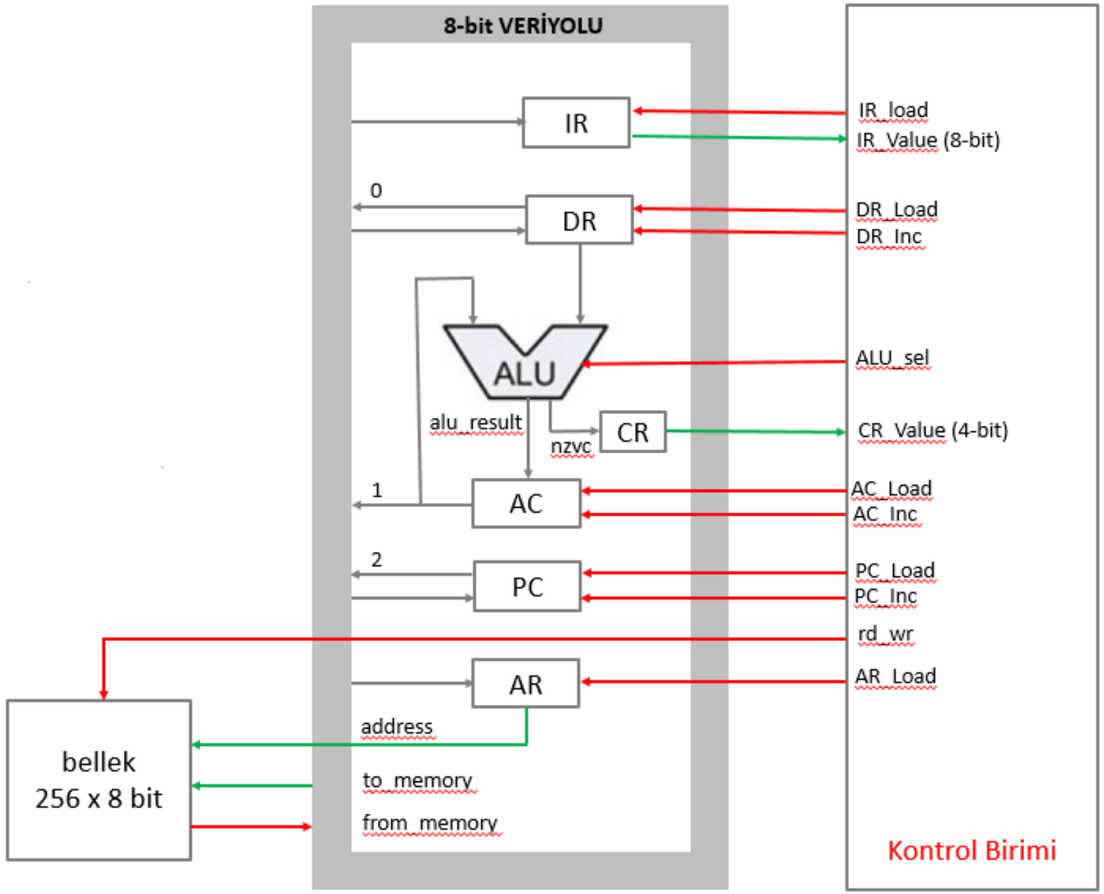
Kontrol Birimi-1

Fetch aşaması (2 clk), FETCH_0: AR <- PC
FETCH_0: Bus<-PC, Bus_Sel=11
FETCH_0: AR<-Bus, AR_Load=1
FETCH_0: Address<-AR

FETCH_1: PC<-PC+1

FETCH_2: IR <- M[AR]
FETCH_2: Bus <- from_memory
FETCH_2: IR <- Bus, IR_Load=1

DECODE Aşaması
DECODE_3: IR_Value <-IR



ISERA AKADEMİ

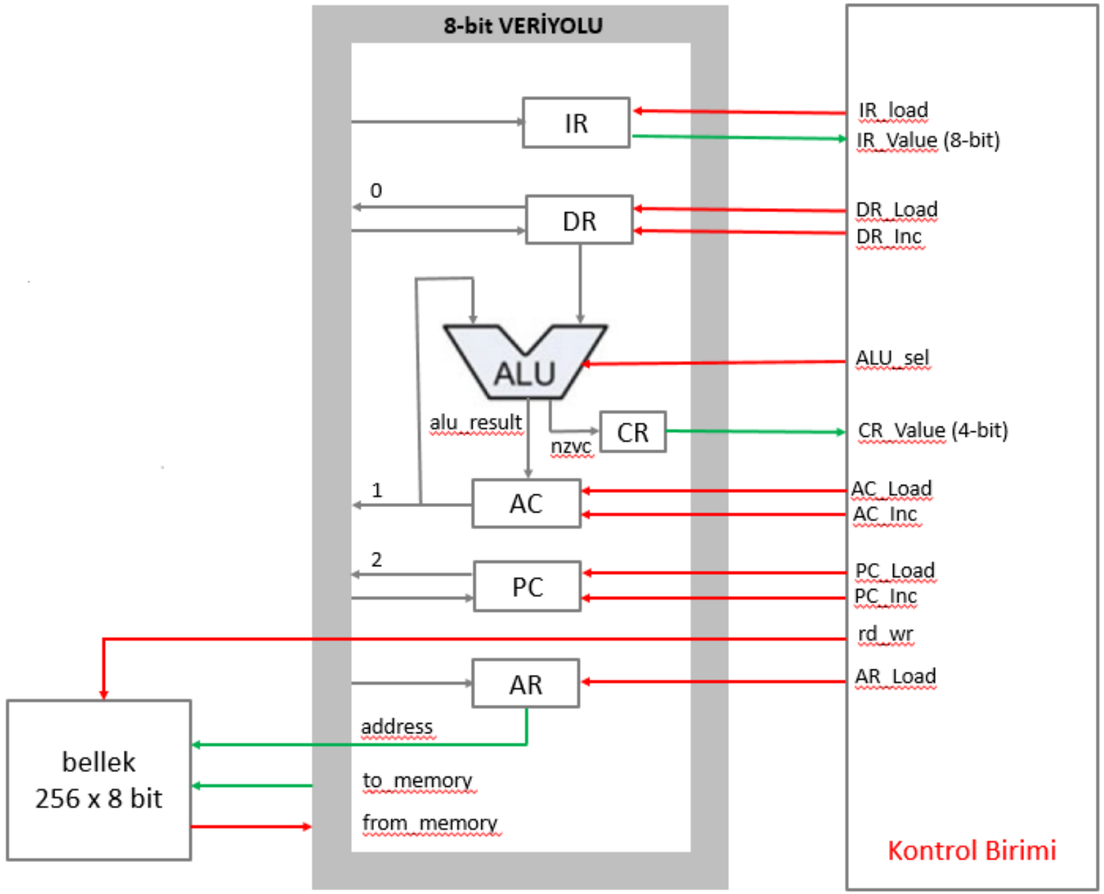
Kontrol Birimi-2, FETCH Aşaması

S_FETCH_0
- Bus <- PC, Bus_Sel=11
- AR <- Bus, AR_Load=1
- Address <- AR

S_FETCH_1
- PC<-PC+1
  (Adres bilgisi program belleğine iletildi, bellekten okunan veri sonraki saat darbesinde elde edilebilir.)

S_FETCH_2
- Bus <- from_memory
- IR <- Bus, IR_Load=1

S_DECODE_3
- IR_Value <-IR
  (Okunan OPCODE'un değerine bakılır ve komutun kimliği tespit edilir.)



Bölüm-3

Komutların Çalışma Biçimi

ISERA

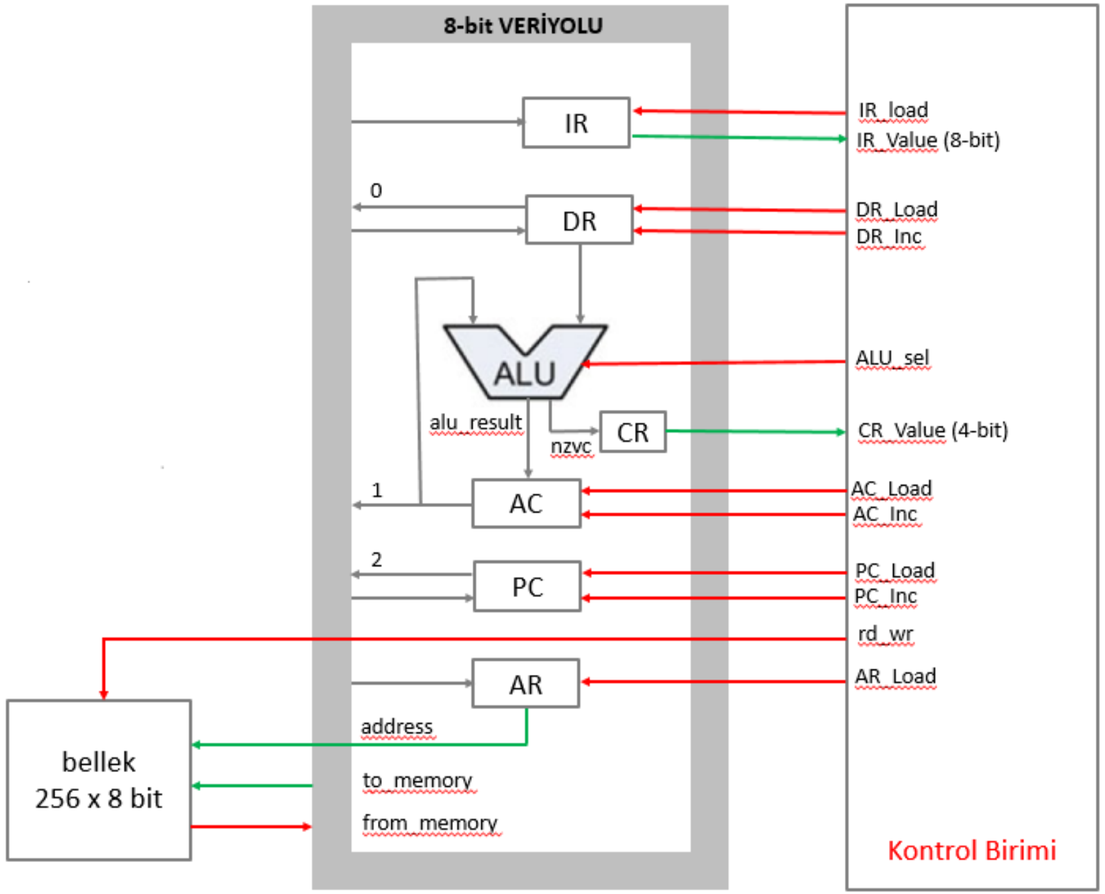
ISERA AKADEMİ

Kullanılacak Komutlar

Zinde CPU da şu komutlar örnek olarak kullanılacaktır:

1. I → LDA IMM A
2. S → LDA ADR A
3. S → STA ADR A
4. S → ADD ADR A
5. I → ADD IMM A
6. R → INC A

NOT: Bu aşamada gösterilen komutlar için verilen adımlar EXECUTE aşamasını göstermektedir.



Kontrol Birimi (LDA_IMM_A)

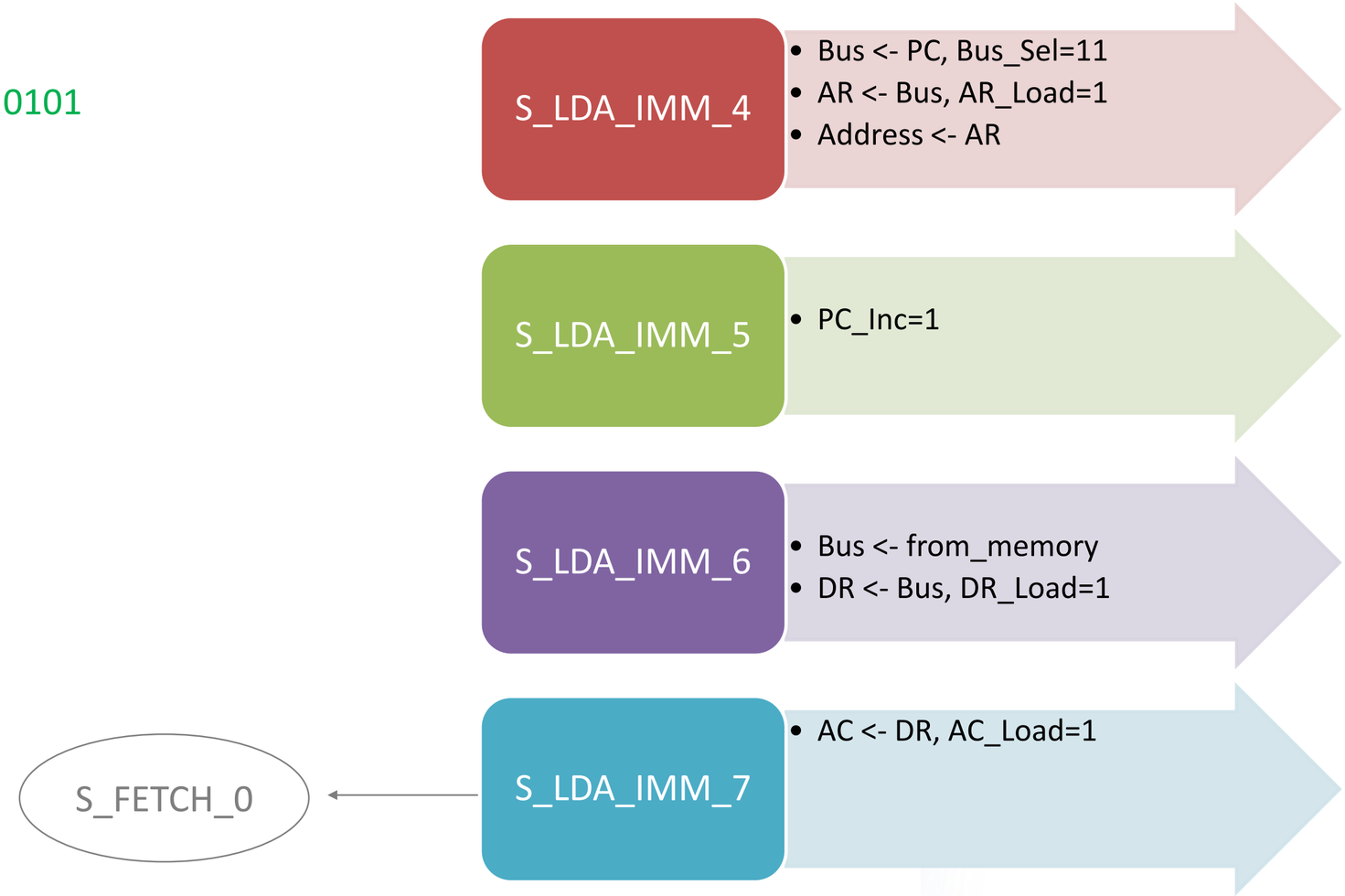
LDA #5 işlemini yapalım: (A<-05H)
Örnek komut kodu: 00 01 **0101** 0000 0101

FETCH_0: AR <- PC
FETCH_0: Bus<-PC, Bus_Sel=11
FETCH_0: AR<-Bus, AR_Load=1
FETCH_0: Address<-AR

FETCH_1: PC<-PC+1

FETCH_2: IR <- M[AR]
FETCH_2: Bus <- from_memory
FETCH_2: IR <- Bus, IR_Load=1

DECODE Aşaması
DECODE_3: IR_Value <-IR



Kontrol Birimi (LDA_ADR_A)

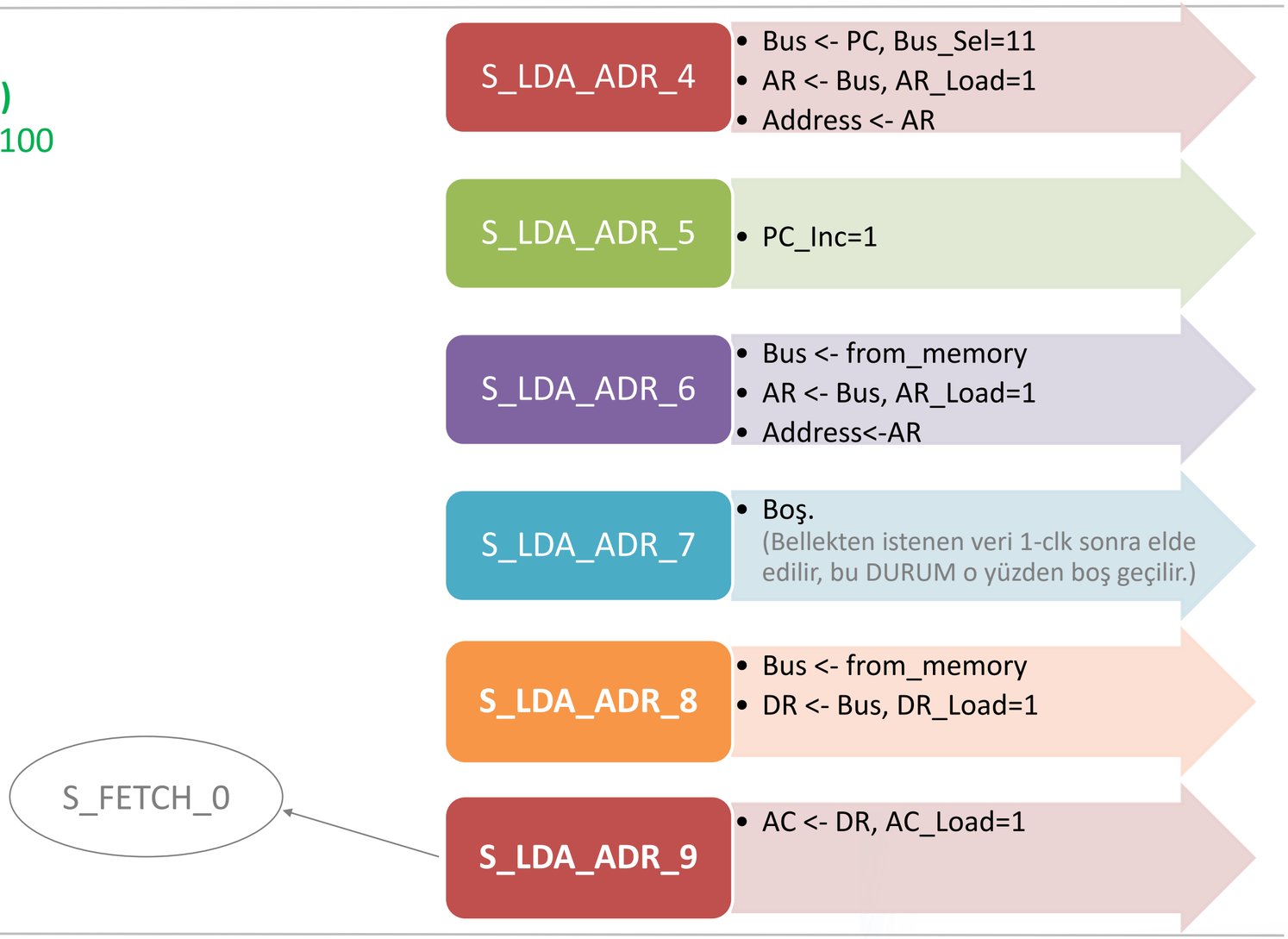
LDA $20 işlemini yapalım: (A <- M[20])
Örnek komut kodu: 00 10 **0101** 0001 0100

FETCH_0: AR <- PC
FETCH_0: Bus<-PC, Bus_Sel=11
FETCH_0: AR<-Bus, AR_Load=1
FETCH_0: Address<-AR

FETCH_1: PC<-PC+1

FETCH_2: IR <- M[AR]
FETCH_2: Bus <- from_memory
FETCH_2: IR <- Bus, IR_Load=1

DECODE_3:
DECODE_3: IR_Value <-IR



Kontrol Birimi (STA_ADR_A)

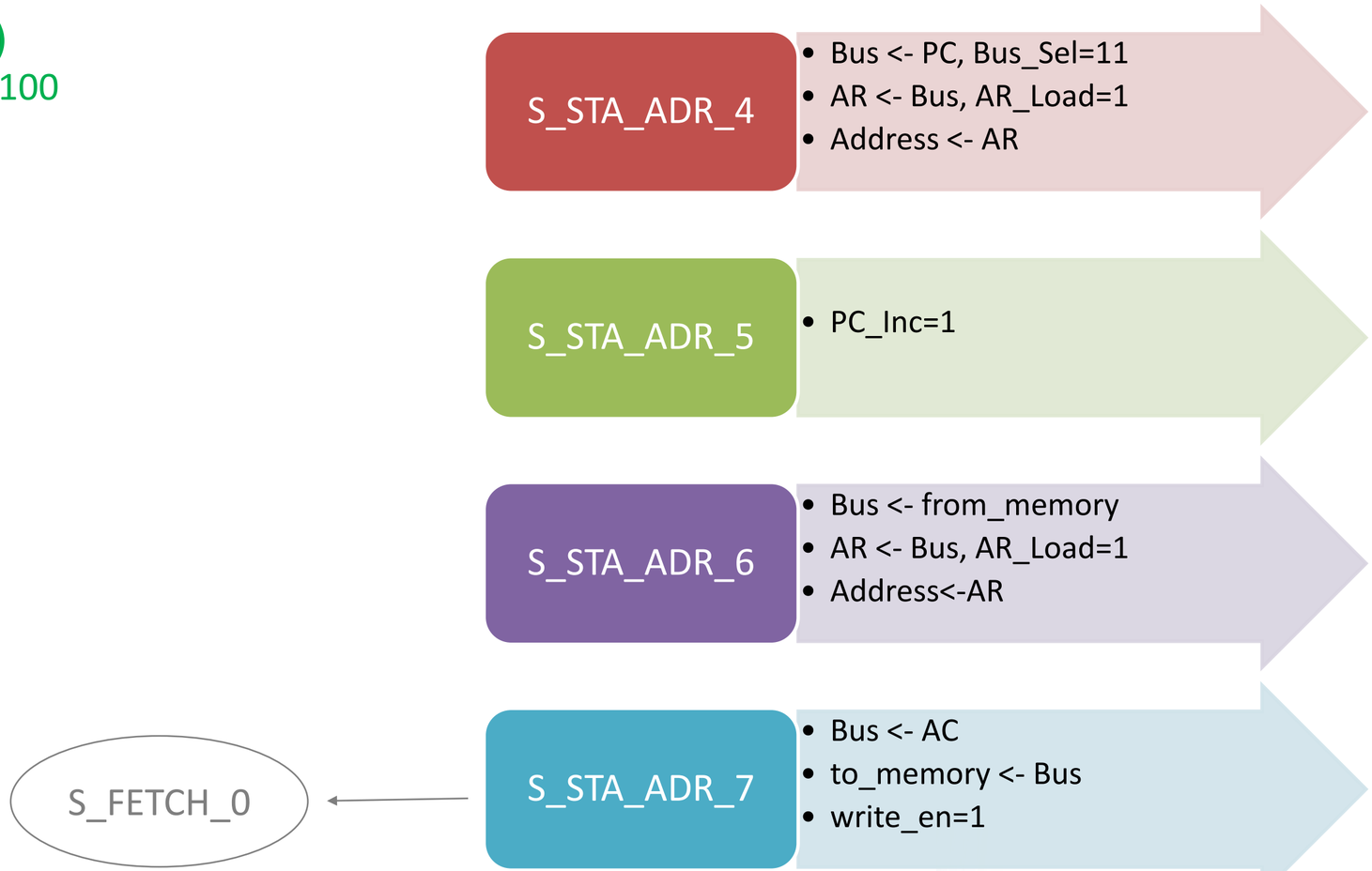
STA $20 işlemini yapalım: (M[20] <- A)
Örnek komut kodu: 00 10 **0110** 0001 0100

FETCH_0: AR <- PC
FETCH_0: Bus<-PC, Bus_Sel=11
FETCH_0: AR<-Bus, AR_Load=1
FETCH_0: Address<-AR

FETCH_1: PC<-PC+1

FETCH_2: IR <- M[AR]
FETCH_2: Bus <- from_memory
FETCH_2: IR <- Bus, IR_Load=1

DECODE Aşaması
DECODE_3: IR_Value <-IR



Kontrol Birimi (ADD_ADR_A)

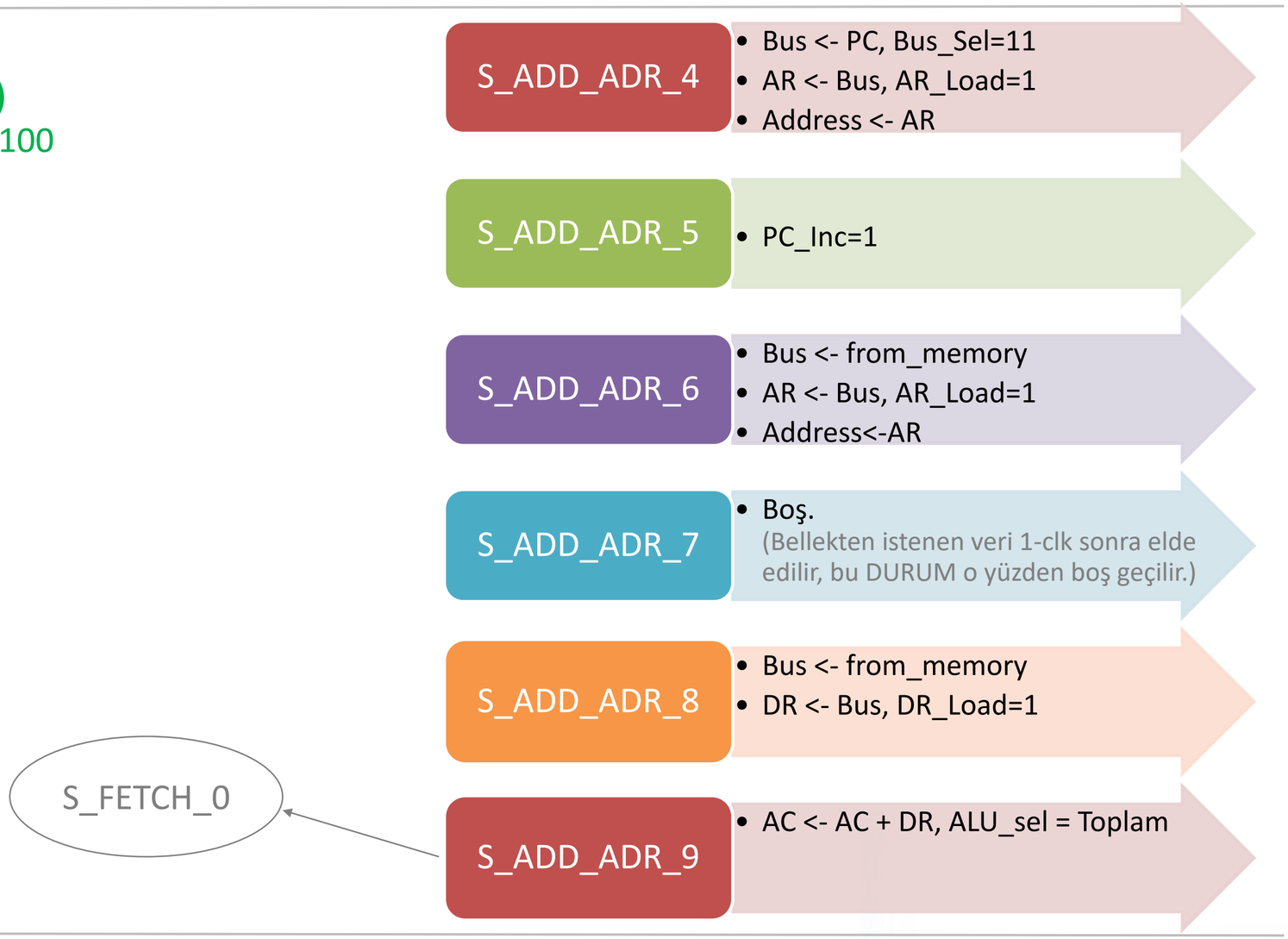
STA $20 işlemini yapalım: (M[20] <- A)
Örnek komut kodu: 00 10 **0110** 0001 0100

FETCH_0: AR <- PC
FETCH_0: Bus<-PC, Bus_Sel=11
FETCH_0: AR<-Bus, AR_Load=1
FETCH_0: Address<-AR

FETCH_1: PC<-PC+1

FETCH_2: IR <- M[AR]
FETCH_2: Bus <- from_memory
FETCH_2: IR <- Bus, IR_Load=1

DECODE Aşaması
DECODE_3: IR_Value <-IR



Kontrol Birimi (ADD_IMM_A)

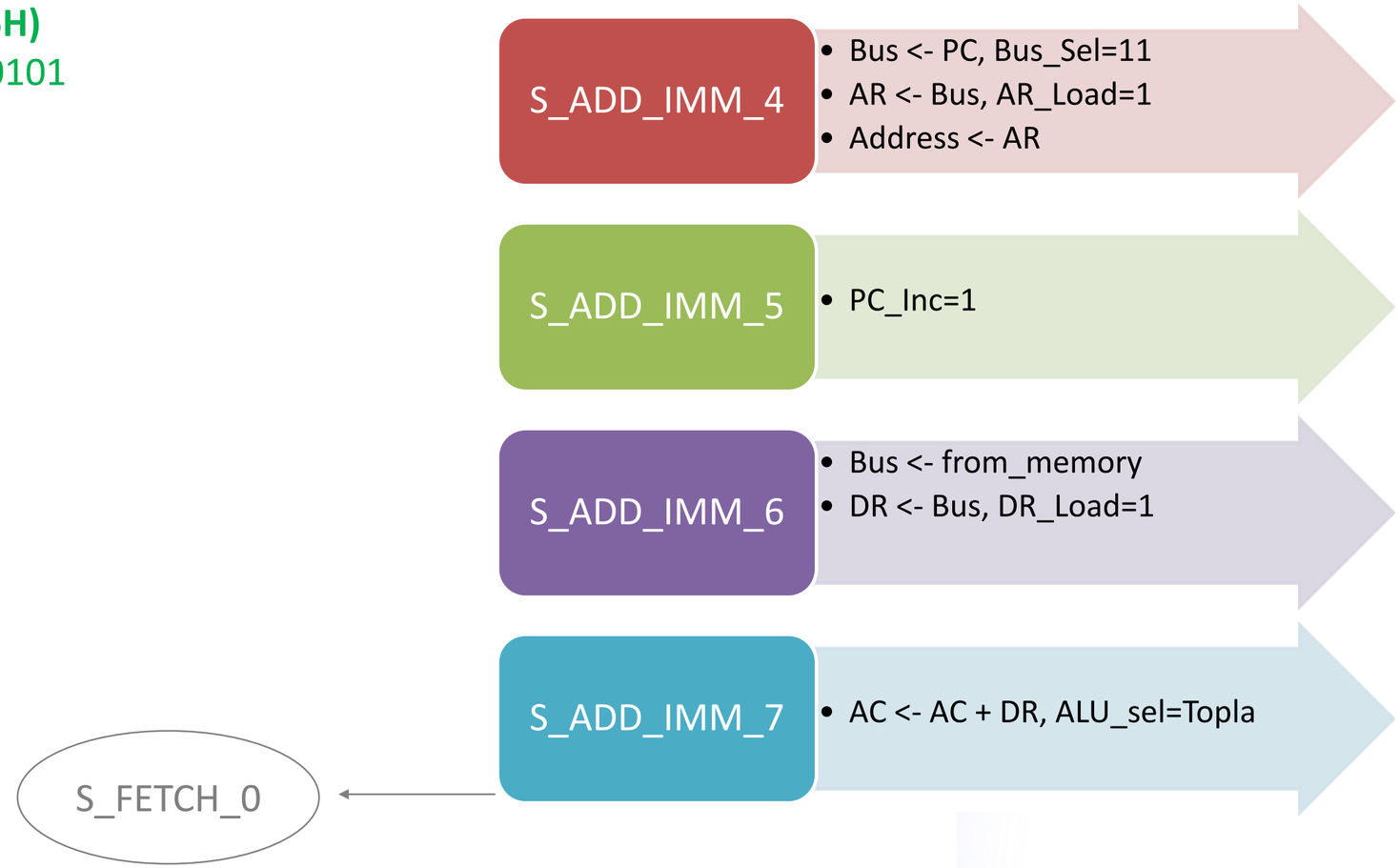
ADD #5 işlemini yapalım: (AC<-AC+05H)
Örnek komut kodu: 00 01 **0101** 0000 0101

FETCH_0: AR <- PC
FETCH_0: Bus<-PC, Bus_Sel=11
FETCH_0: AR<-Bus, AR_Load=1
FETCH_0: Address<-AR

FETCH_1: PC<-PC+1

FETCH_2: IR <- M[AR]
FETCH_2: Bus <- from_memory
FETCH_2: IR <- Bus, IR_Load=1

DECODE Aşaması
DECODE_3: IR_Value <-IR



Kontrol Birimi (INC_A)

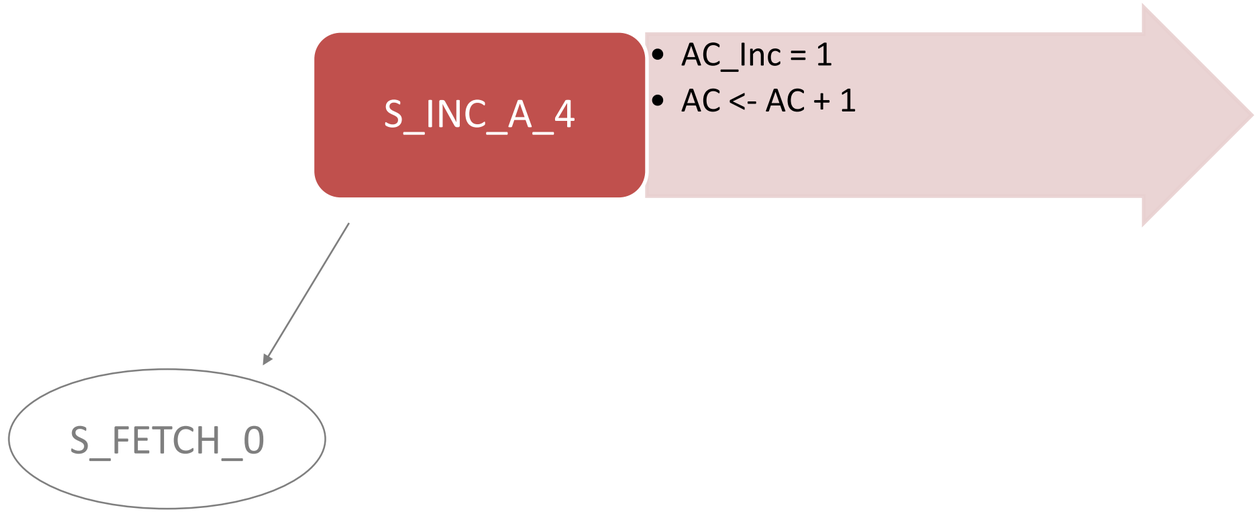
STA $20 işlemini yapalım: (M[20] <- A)
Örnek komut kodu: 00 10 **0110** 0001 0100

FETCH_0: AR <- PC
FETCH_0: Bus<-PC, Bus_Sel=11
FETCH_0: AR<-Bus, AR_Load=1
FETCH_0: Address<-AR

FETCH_1: PC<-PC+1

FETCH_2: IR <- M[AR]
FETCH_2: Bus <- from_memory
FETCH_2: IR <- Bus, IR_Load=1

DECODE Aşaması
DECODE_3: IR_Value <-IR

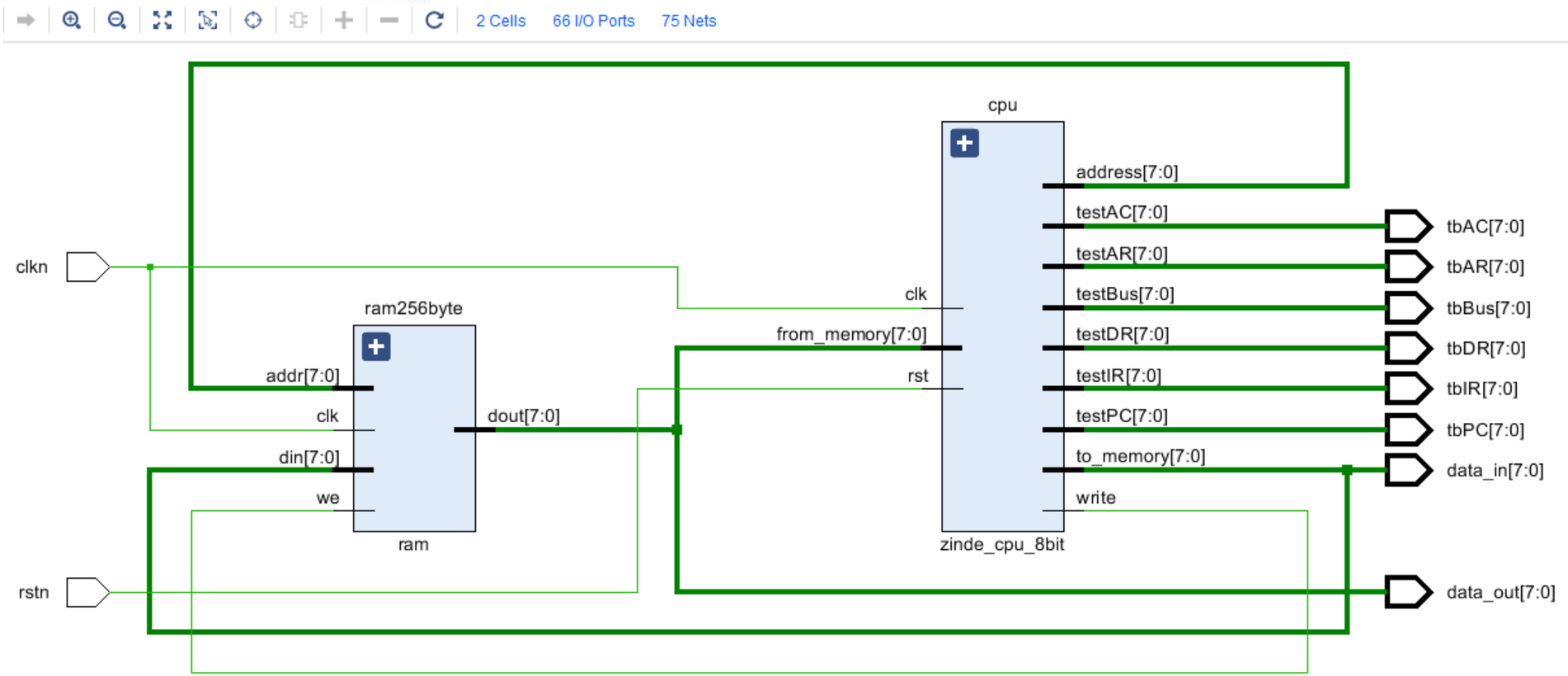


Bölüm-4

Test Senaryosu

ISERA

ZindeRV8 CPU, (temel mimari)



Bu proje kapsamında geliştirilen 8-bitlik CPU mimari tasarımı yukarıda görülmektedir.

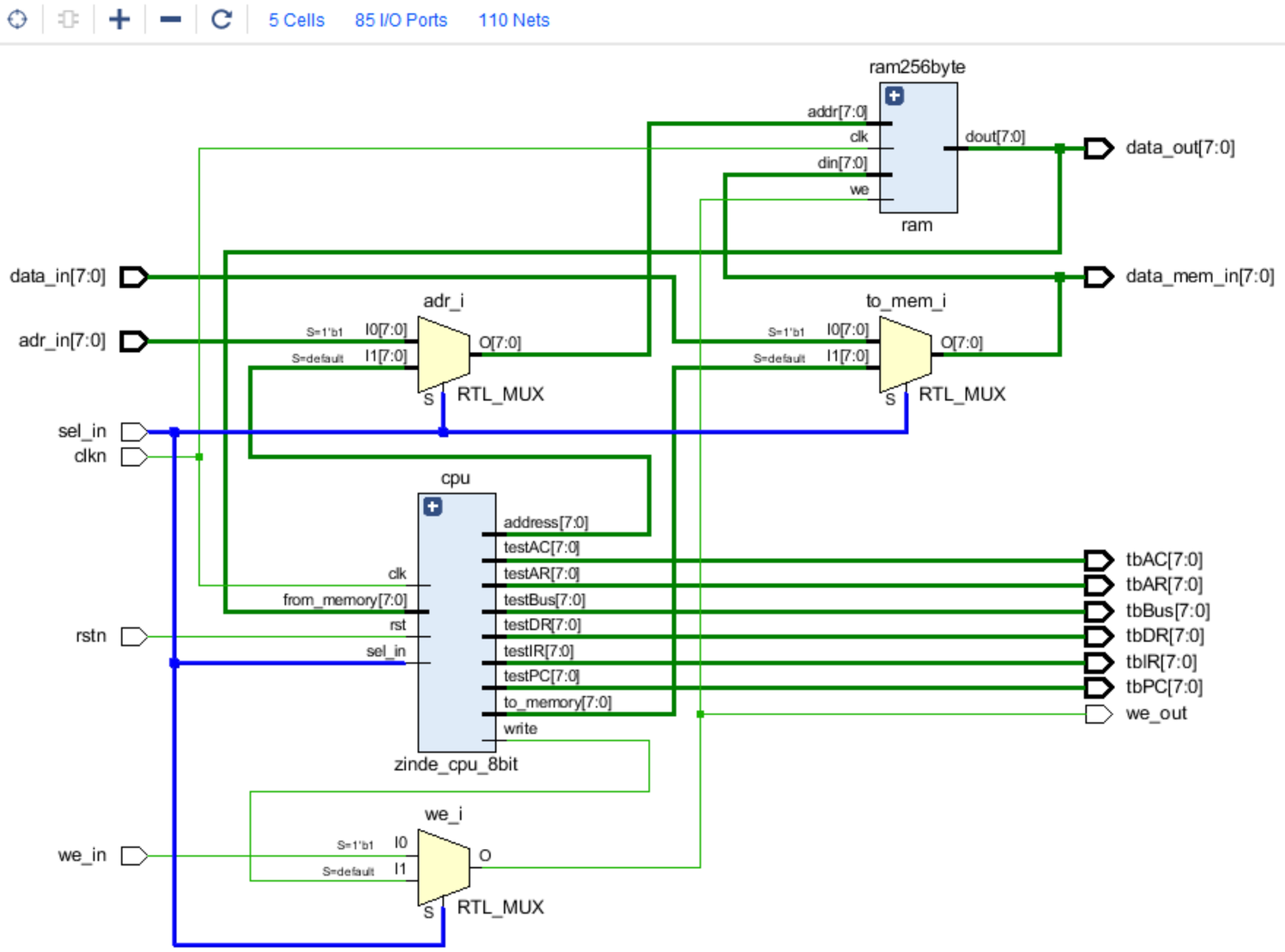
ZindeRV8 CPU, (testbench mimari)

İşlemcinin fonksiyonel testi için öncelikle dışardan RAM'a program verileri aktarılmalıdır.

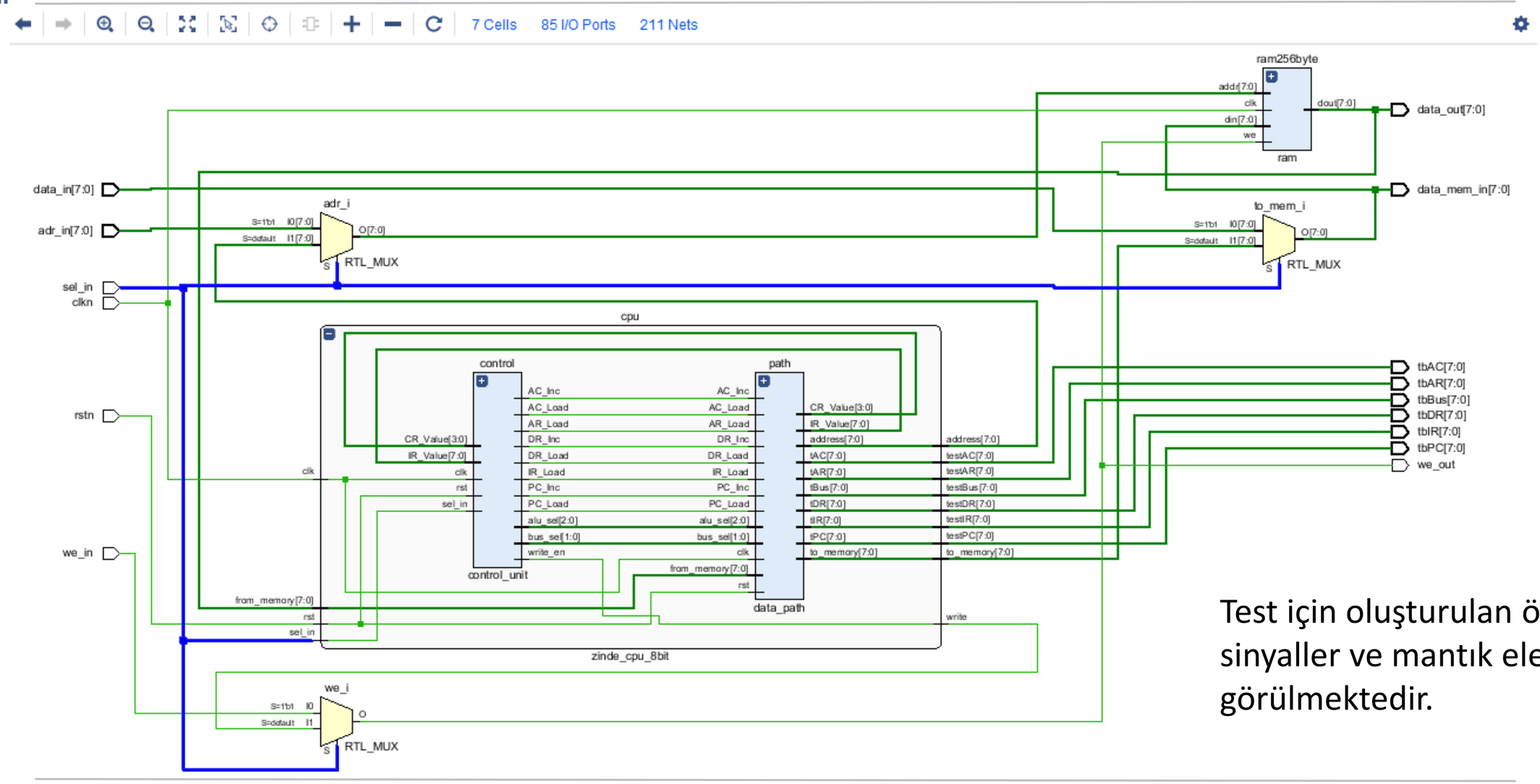
Bunun için yanda görüldüğü gibi seçim mantık elemanları eklendi.

Bu elemanlar vasıtasıyla öncelikle program yükleniyor sonrasında işlemci çalışmaya başlıyor.

Belleğe program girileceği zaman seçim ucu 1 olarak değiştirilir.



ZindeRV8 CPU, (testbench mimari)



Test için oluşturulan özel sinyaller ve mantık elemanları görülmektedir.

Test Senaryosu

Zinde CPU da şu komutlar örnek olarak kullanılacaktır:

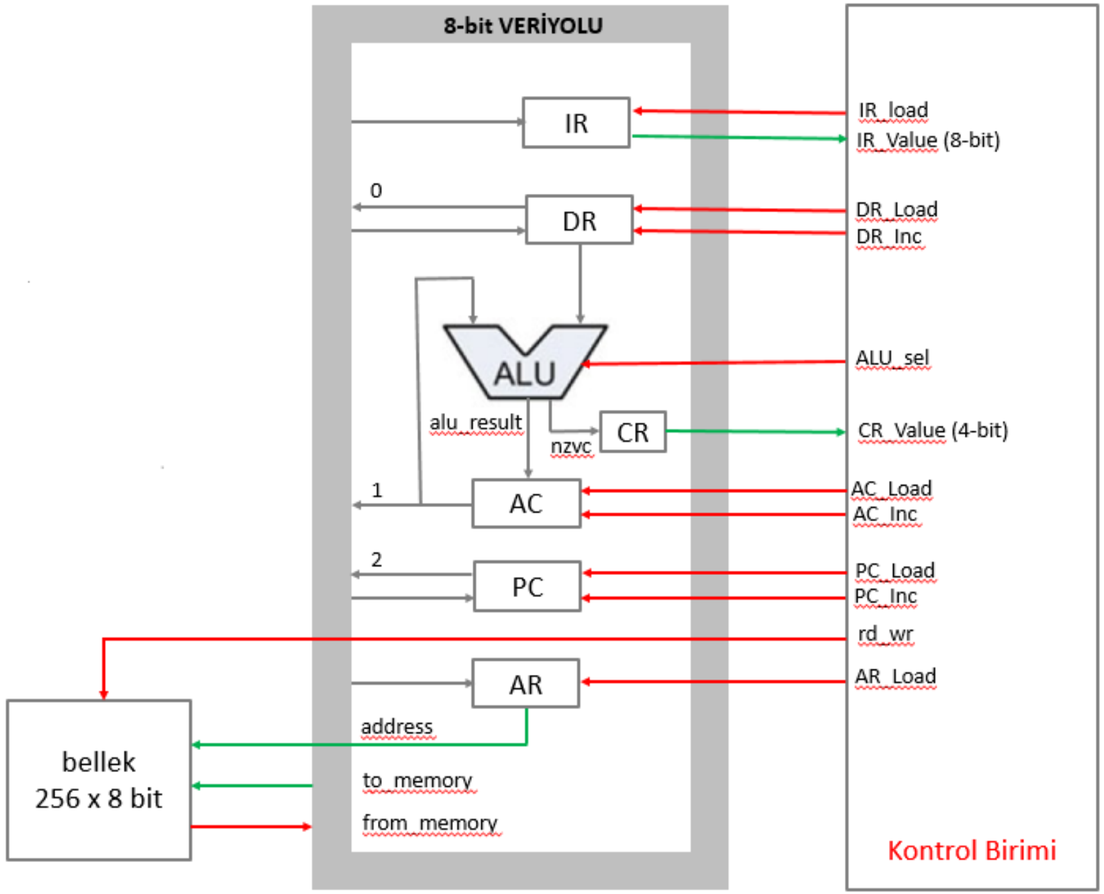
1. LDA #5h, AC ← 5
2. ADD $50h, AC ← AC + [50]
3. STA $60h, [60] ← AC

4. LDA_IMM → 32h
5. ADD_ADR → 46h
6. STA_ADR → 44h

- Bu test senaryosunda öncelikle AC'ye 5 değeri alınacaktır.
- Sonrasında AC'deki 5 değeri ile [50] nolu adresteki değer toplanıp sonuç yine AC'ye yazılacaktır.
- Son aşamada ise AC deki değer [60] nolu adrese aktarılacaktır.

ÖRNEK BELLEK ŞEMASI

…	…
…	…
10H	32 (opcode)
11H	05
12H	46 (opcode)
13H	50
14H	44 (opcode)
15H	60
16H	0F (BRK)
…	…
50H	09
…	…
60H	0e



Testbench Sonuçları ve Analizi

Fonsiyonel doğrulama aşamasında özellikle **RAM** ve **Buffer** gibi çoklu veri barındıran yapılardaki veri trafiği takip edilemeyebilir.

Son aşamada **TCL konsol** kullanılarak buradaki verilere erişim sağlanır ve sonuçlar doğrulanır.

TCL console sorgulamaları

belleğin 60 nolu satırında 14(0xe) değeri olmalıdır, bunun için aşağıdaki satırı TCL konsolda çalıştırabiliriz.

```
get_value /tb_ZindeRV8/tb_cpu/ram256byte/bellek(17)
```

Tüm RAM içeriği bir dosyaya yazılabilir:

```
set outfile [open "D:/ram_dump.txt" "w"]
for {set i 0} {$i < 256} {incr i} {
    set val [get_value /tb_ZindeRV8/tb_cpu/ram256byte/bellek($i)]
    puts $outfile "bellek[$i] = $val"
}
close $outfile
```

Zinde CPU da şu komutlar örnek olarak kullanılacaktır:

1. LDA #5h, AC ← 5
2. ADD $50h, AC ← AC + [50]
3. STA $60h, [60] ← AC
4. BRK,

5. LDA_IMM → 32h
6. ADD_ADR → 46h
7. STA_ADR → 44h

Yukarıda 1, 2, 3 ve 4 nolu satırlarda verilen kod ile öncelikle AC kaydedicisine sabit 5 sayısı aktarılıyor, sonra 5 verisi ile [50] nolu adreste bulunan değer toplanıyor ve son aşamada ise elde edilen sonuç [60] nolu adrese aktarılıyor.

TEŞEKKÜRLER

ISERA

AKADEMİ