BROOKSHEAR MAKINE DILI YORUMLAYICISI VE MIMARISI

BURSA TEKNİK ÜNİVERSİTESİ
BLM101 - BILGISAYAR MÜHENDISLIĞINE GIRIŞ

SÜMEYYE ŞİMŞEK
24360859058

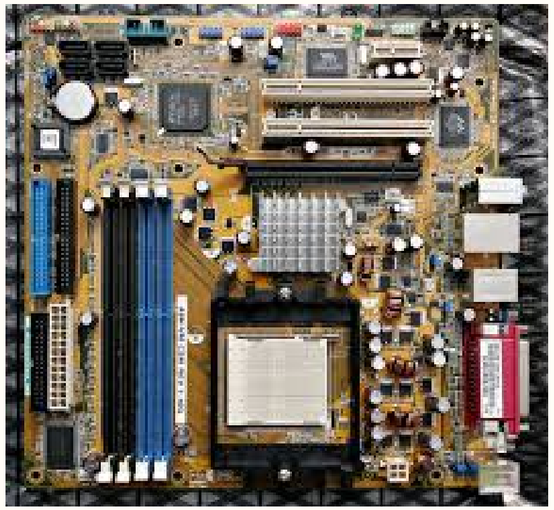
SUNUM İÇERİĞİ

- Bilgisayar Mimarisine Giriş ve Temel Bileşenler
- Donanım Analizi: Register ve RAM Bellek Yapısı
- Makine Dili ve Komut Yapısı (Op-code & Operand)
- Makine Döngüsü (Fetch-Decode-Execute)
- Brookshear Mimarisi ve Komut Seti Detayları
- Python Simülasyonu ve Algoritma Analizi

BILGISAYAR MIMARISI NEDIR?

Tanım:

- Bir bilgisayar sisteminin donanımsal yapısını ifade eder.
- Sistemi oluşturan bileşenlerin (CPU, RAM, I/O) birbirleriyle nasıl iletişim kurduğunu belirler.
- Verilerin sistem içinde nasıl işlendiğini tanımlayan kurallar bütünüdür.



Neden Önemlidir?

- Yazılım (Software) ve Donanım (Hardware) arasındaki köprüdür.
- Yazdığımız yüksek seviyeli kodların (Python, C vb.), donanım üzerinde nasıl elektriksel sinyallere dönüştüğünü anlamamızı sağlar.

TEMEL BILEŞENLER: CPU VE ALU

1 CPU (Merkezi İşlem Birimi)

- Bilgisayarın "beyni" olarak kabul edilir.
- Hafızadaki (RAM) program komutlarını sırasıyla alır ve yorumlar.
- Sistemdeki diğer tüm donanım birimlerini yönetir ve senkronize eder.

2 ALU (Aritmetik Mantık Birimi)

- CPU’nun işlem yapan "kas gücüdür".
- Tüm matematiksel ve mantıksal işlemler burada gerçekleşir.
  - Aritmetik: Toplama, çıkarma (Brookshear’da toplama çok önemli).
  - Mantıksal: VE (AND), VEYA (OR), XOR gibi karşılaştırma işlemleri.

TEMEL BILEŞENLER: KONTROL BIRIMI VE REGISTER

3

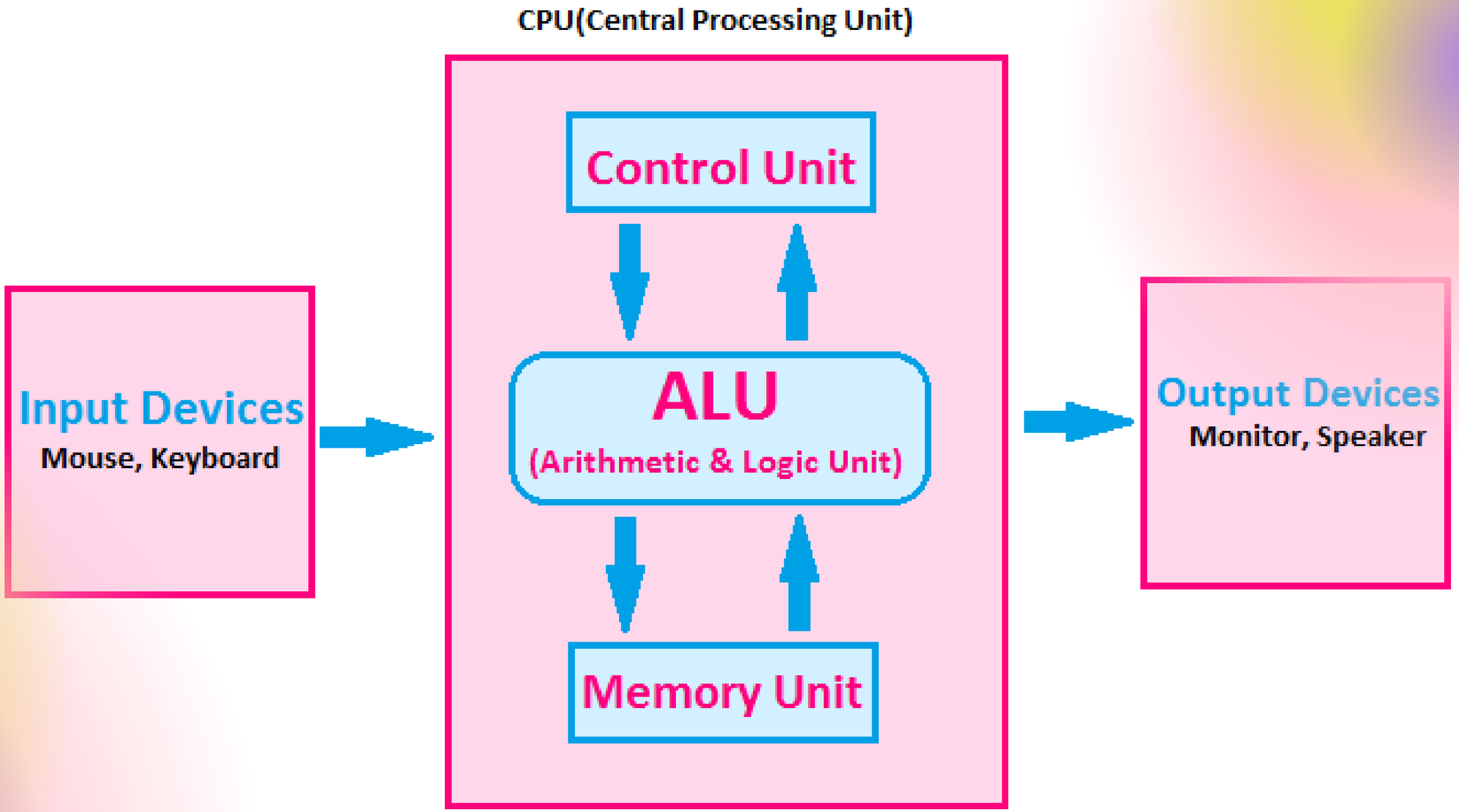
Kontrol Birimi (Control Unit)

- İşlemci içerisindeki "trafik polisi" gibidir.
- Hafızadan (Memory) komutları getirir ve şifresini çözer.
- ALU ve diğer birimlere ne yapmaları gerektiği sinyalini gönderir (Örn: "Şimdi toplama yap").

4

Register (Kaydedici)

- CPU'nun hemen içinde bulunan, en hızlı ve geçici hafıza birimleridir.
- Veriler işlenmeden önce RAM'den alınıp Register'lara getirilir.
- Projemizdeki Önemi: Brookshear mimarisinde 16 adet genel amaçlı Register bulunur.
- İşlemler (toplama, kopyalama vb.) doğrudan bu Register'lar üzerinde yapılır.



CPU Block Diagram & Architecture

MAKİNE DÖNGÜSÜ (MACHINE CYCLE)

1-Getir (Fetch):
- CPU, sıradaki komutun adresini Program Sayacı'ndan (Program Counter) öğrenir.
- Komutu RAM'den alır ve kendi içindeki Komut Register'ına (IR) getirir.

2-Çöz (Decode):
- Gelen 16-bitlik (örneğin 14A3) kodun ne anlama geldiği çözülür.
- Kontrol Birimi (CU) devreye girer: "Bu bir Yükleme (LOAD) işlemi mi yoksa Toplama (ADD) işlemi mi?" diye bakar.

3-Yürüt (Execute):
- İşlem gerçekleştirilir.
- Gerekirse ALU matematiksel işlemi yapar veya veri bir yerden bir yere kopyalanır.

MAKINE DILI (MACHINE LANGUAGE) NEDIR?

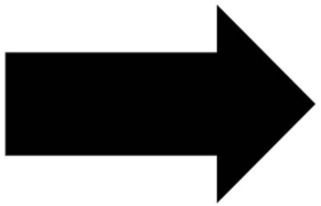
Tanım:

- Bilgisayar donanımının (CPU) herhangi bir çeviriye ihtiyaç duymadan doğrudan anlayıp yürütebildiği tek dildir.
- Elektriksel sinyalleri temsil eden 0 ve 1'lerden (Binary / İkili Sistem) oluşur.

Neden Zorlu?

- İnsanlar için bu dili okumak ve yazmak çok karmaşıktır.
- Milyonlarca 0 ve 1 arasında hata yapmak çok kolaydır.

`0001 0100 1010 0011`



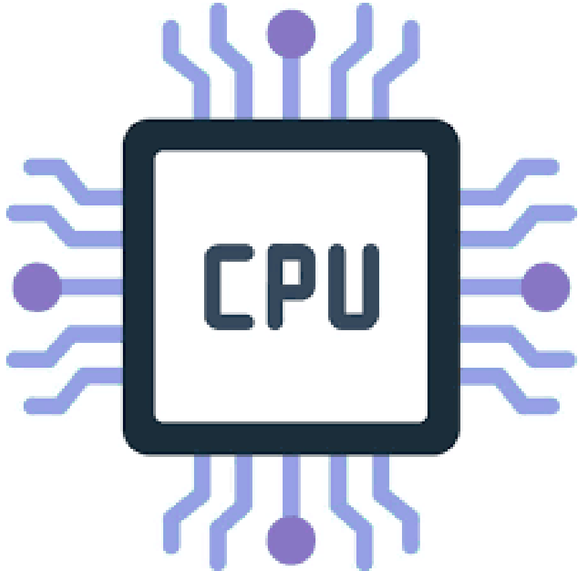
`14A3`

"Aynı verinin insan tarafından okunabilir hali"

KOMUT SETI (INSTRUCTION SET) NEDIR?

Tanım:

- Bir işlemcinin donanım olarak anlayabildiği ve yürütebildiği tüm komutların listesidir.
- Tıpkı bir dilin kelime hazinesi gibi düşünebilirsiniz. İşlemci sadece bu listedeki komutları bilir.



İşlevi:

- Yazılımcının donanıma ne yapacağını söylemesini sağlar.
- Farklı işlemcilerin (Intel, ARM vb.) farklı komut setleri vardır.

!!! Bizim kullandığımız sanal Brookshear işlemcisi 12 temel komuttan oluşan sade bir komut setine sahiptir. Bu komutlar LOAD, STORE, MOVE, ADD gibi temel işlemleri kapsar.

KOMUT YAPISI (INSTRUCTION FORMAT)

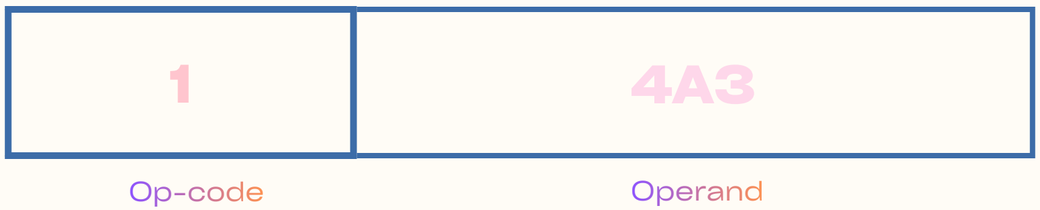
Genel Yapı:

- Brookshear makinesinde her komut 16 bit uzunluğundadır.
- Kolay okunabilmesi için bu 16 bit, 4 haneli Hexadecimal (Onaltılık) kod ile gösterilir.
- Örnek: 14A3

1

Op-code (Operasyon Kodu):
- Komutun ilk hanesidir.
- İşlemciye "Ne yapacağını" söyler.

Örnek: 1 -> "Yükle (LOAD)", 5 -> "Topla (ADD)"



2

Operand (İşlenen/Parametre):
- Komutun geriye kalan 3 hanesidir.
- İşlemciye "Hangi veriyi" veya "Hangi adresi" kullanacağını söyler.

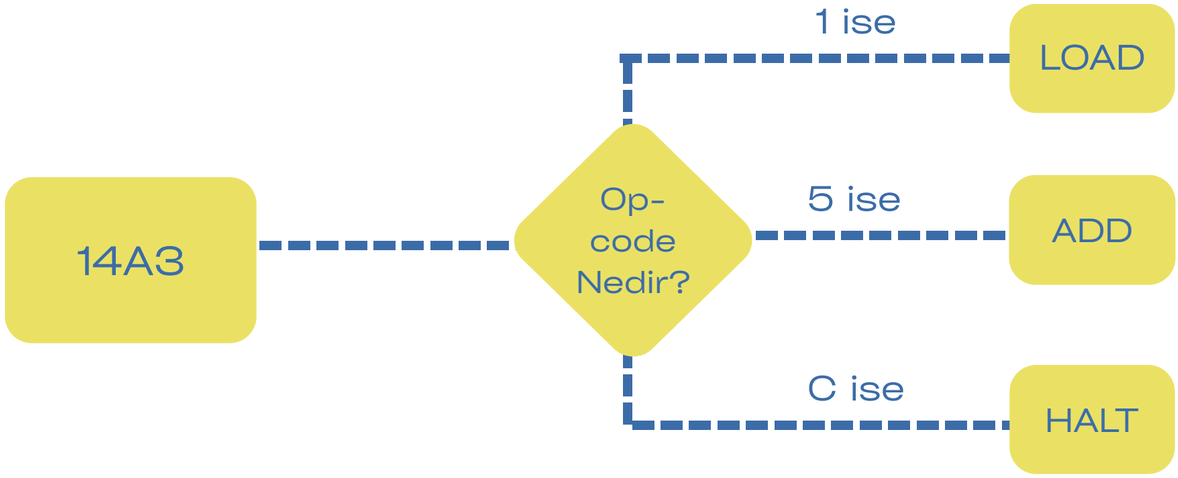
OP-CODE (İŞLEM KODU) NEDİR?

Tanım:

- İngilizce "Operation Code" kelimesinin kısaltmasıdır.
- Komutun türünü belirleyen kimlik numarasıdır.

İşlevi:

- İşlemci bir komutu okuduğunda ilk olarak Op-code kısmına bakar.
- Op-code ne ise, donanım ona göre davranır.
- Eğer Op-code 1 ise -> Hafızadan veri getirilir.
- Eğer Op-code C ise -> Program durdurulur (HALT).



OPERAND (İŞLENEN) NEDİR?

Tanım:

- Komutun işlem yapacağı veriyi veya verinin bulunduğu adresi belirten kısımdır.
- Op-code'dan sonra gelen son 3 Hex hanesidir.
- Örnek: 14A3 komutundaki 4A3 kısmı operanddır.

İşlevi:

****Op-code'un türüne göre Operand'ın anlamı değişebilir:

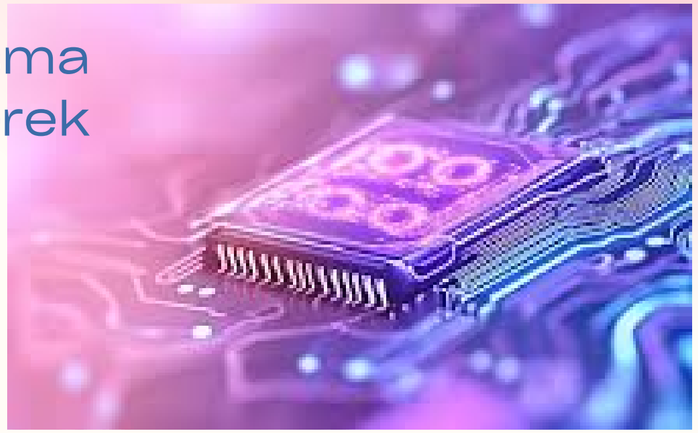
1. Adres Olabilir: Verinin RAM'deki yerini gösterir (Örn: "4A3 numaralı kutudaki bilgiyi getir").
2. Saf Veri Olabilir: Doğrudan sayının kendisidir (Örn: "Register'a 5 sayısını yükle").
3. Register Numarası Olabilir: Hangi Register'ların kullanılacağını söyler (Örn: "Register 1 ile Register 2'yi topla").

1<u>4A3</u> → Parametreler (Adres, Veri, Register)

BROOKSHEAR MAKINESI (SANAL MIMARI)

NEDIR?

- J. Glenn Brookshear'ın "Computer Science: An Overview" kitabında tanımlanan, eğitim amaçlı tasarlanmış sanal bir bilgisayardır.
- Fiziksel bir çip değildir; bilgisayarın çalışma mantığını (Fetch-Decode-Execute) basitleştirerek öğretmek için kullanılır.



TEKNIK ÖZELLIKLER (SPECS):

- Ana Bellek (RAM): 256 hücre (Her biri 8 bit).
- Komut Uzunluğu: 16 bit (4 Hex hanesi).
- Mimari Tipi: Von Neumann Mimarisi (Veri ve komutlar aynı hafızada tutulur).

REGISTER MIMARISININ TEKNIK DETAYLARI

MIMARI YAPILANMA

- Brookshear işlemcisi, verileri tutmak için 16 adet bağımsız Register kullanır.
- Her bir Register, veriyi 8-bit (1 Byte) uzunluğunda saklar.

ADRESLEME VE ERIŞIM

- Registerlara erişmek için 4-bitlik adresleme kullanılır.
- Bu yüzden isimleri 0, 1, ... 9, A, B, C, D, E, F şeklindedir (16 ihtimal = 2^4).
- Örneğin: 14A3 komutundaki "4", doğrudan 4. Register'ı işaret eder.

0	1	2	3
4	5	6	7
8	9	A	B
C	D	E	F

CPU İÇİNDEKİ HIZLI ALAN

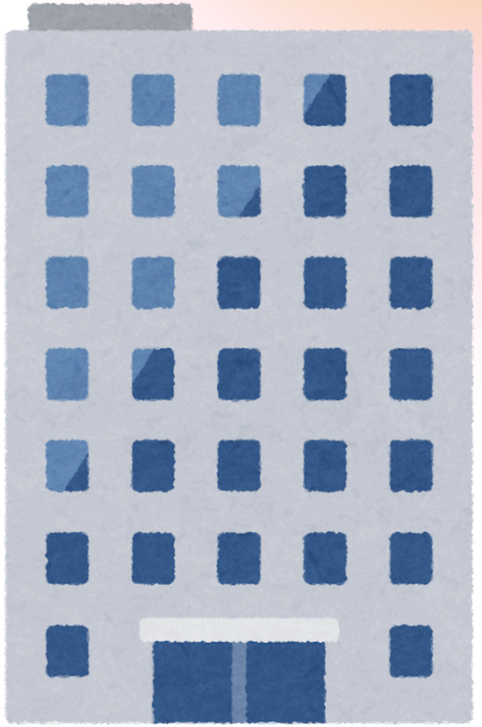
RAM BELLEK HARITASI VE ADRESLEME

BELLEK ORGANIZASYONU

- Sistemde toplam 256 adet hafıza hücresi bulunur
  (2^8 = 256).
- Bu sayede her hücreye 2 haneli Hex kod (00 - FF) ile ulaşılabilir.

VERI KAPASITESI

- Toplam 256 adet bağımsız hafıza hücresinden oluşur.
- Her hücrenin 00'dan FF'ye kadar değişen benzersiz bir adresi vardır.
- Her bir hücre 8-bit (1 Byte) büyüklüğünde veri saklar.
- Her bir hücrenin veri yolu genişliği 8-bittir.
- İşlemci ile RAM arasındaki veri alışverişi bu 8-bitlik veri yolu (Data Bus) üzerinden gerçekleşir.



Adres: 00

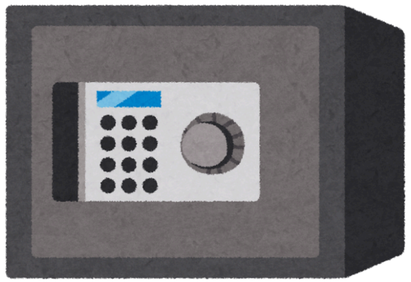
... (Toplam 256 Hücre) ...

Adres: FF

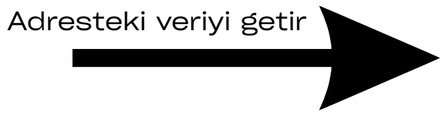
VERI YÜKLEME (LOAD) KOMUTLARI

Op-code [1]: Hafızadan Yükle (LOAD from Memory)

- Format: 1 R X Y
- Görevi: XY adresindeki bellek hücresinin içeriğini alır, R numaralı Register'a kopyalar.
- Örnek: 14A3 -> "A3 adresine git, oradaki veriyi al ve 4 numaralı Register'a koy."



Adres:XY



HADEF REGİSTER

Op-code [2]: Sabit Değer Yükle (LOAD Immediate)

- Format: 2 R X Y
- Görevi: XY değerinin kendisini doğrudan R numaralı Register'a yükler.
- Örnek: 24A3 -> "4 numaralı Register'ın içine doğrudan A3 sayısını yaz."

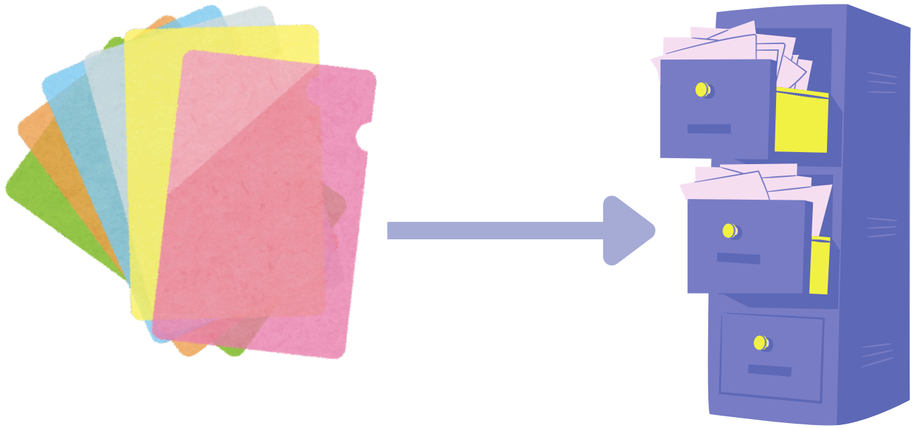
Veri

HADEF REGİSTER

VERİ SAKLAMA VE KOPYALAMA (STORE & MOVE) KOMUTLARI

Op-code [3]: Hafızaya Kaydet (STORE)

- Format: 3 R X Y
- Görevi: R numaralı Register'ın içindeki veriyi alır, Hafızadaki XY adresine yazar.
- Önemli: Bu işlem, bilgisayarda "Kaydet (Save)" tuşuna basmak gibidir. Geçici hafızadaki (Register) veriyi, kalıcı saklama alanına (RAM) gönderir.
- Örnek: 35B1 -> "5 numaralı Register'daki veriyi al, B1 adresine sakla."



Register'dan al -> Hafıza Adresine (Dolaba) koy

Op-code [4]: Kopyala (MOVE)

- Format: 4 0 R S
- Görevi: R numaralı Register'ın içeriğini kopyalar ve S numaralı Register'a yapıştırır.
- Önemli: Kaynak (R) değişmez, sadece kopyası oluşturulur.
- Örnek: 40A5 -> "A numaralı Register'ın içindekini kopyala, 5 numaralı Register'a yapıştır."

Bir Register'dan diğerine kopyala. (Veri çoğalır, silinmez.)

TOPLAMA (ADD) KOMUTLARI

Op-code [5]: Tamsayı Toplama (Two's Complement)

- Format: 5 R S T
- Görevi: S ve T numaralı Register'lardaki değerleri tamsayı olarak toplar, sonucu R numaralı Register'a yazar.
- Kullanım: Tam sayılarla (1, 5, -10 gibi) işlem yaparken kullanılır.
- Örnek: 5312 -> "Register 1 ile Register 2'yi topla, sonucu Register 3'e yaz."

Op-code [6]: Ondalıklı Toplama (Floating Point)

- Format: 6 R S T
- Görevi: S ve T numaralı Register'lardaki değerleri kayan noktalı (ondalıklı) sayı olarak toplar, sonucu R numaralı Register'a yazar.
- Kullanım: Kesirli sayılarla (3.14, 0.5 gibi) işlem yaparken kullanılır.

"İki kaynaktan al, topla, hedefe yaz."

MANTIKSAL İŞLEMLER (LOGIC)

**Bu işlemler sayısal değerlerle değil, sayıları oluşturan bitler (0 ve 1) üzerinde yapılır.

Op-code [7]: OR (VEYA)

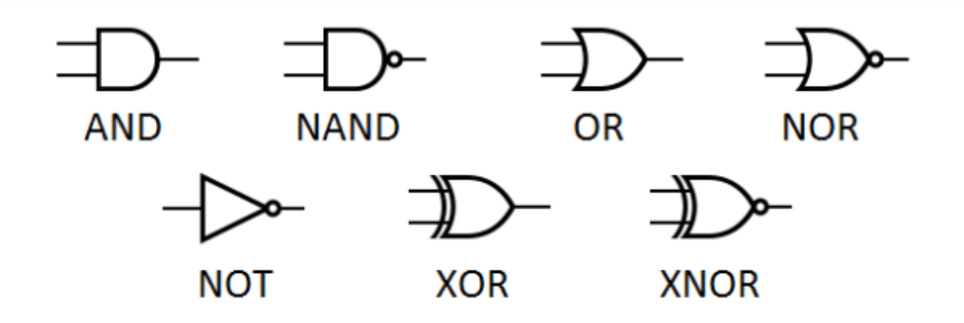
- Format: 7 R S T
- İşlevi: İki register'daki bitleri karşılaştırır. En az bir tanesi 1 ise sonuç 1 olur. (Toplama mantığına benzer).

Op-code [8]: AND (VE)

- Format: 8 R S T
- İşlevi: İki register'daki bitleri karşılaştırır. Sadece ikisi de 1 ise sonuç 1 olur. (Çarpma mantığına benzer).

Op-code [9]: XOR (Özel VEYA)

- Format: 9 R S T
- İşlevi: Bitler birbirinden farklıysa 1, aynıysa 0 sonucunu verir.



KAYDIRMA VE KOŞULLU DALLANMA (ROTATE & JUMP)

Op-code [A]: ROTATE (Kaydırma)

- Format: A R 0 X
- İşlevi: R register'ındaki bitleri X adım kadar sağa kaydırır.
- Örnek: A302 -> "3 numaralı Register'ın içindeki bitleri 2 adım sağa döndür."

Op-code [B]: Koşullu Dallanma (JUMP)

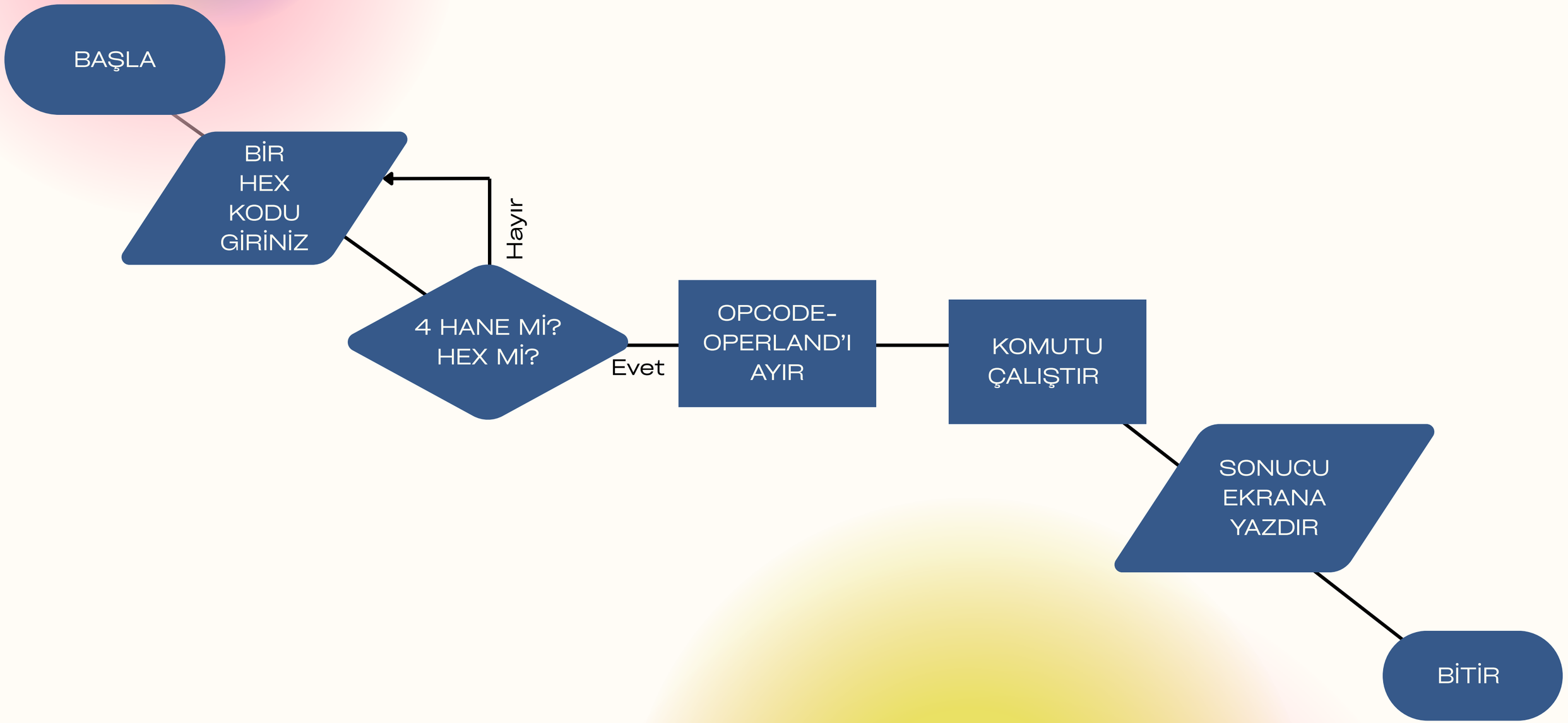
- Format: B R X Y
- Görevi: Programın akışını değiştirir.
- Şartı: Eğer Register R içindeki değer, Register 0 (her zaman) içindeki değere EŞİTSE; program XY adresine atlar (Jump).
- Değilse: Sıradaki komuttan normal şekilde devam eder.
- Kullanım: Döngüler (Loop) ve if-else yapıları kurmak için kullanılır.

DURDURMA (HALT)

Op-code [C]: Durdur (HALT)

- Format: C 0 0 0
- Görevi: Programın çalışmasını tamamen sonlandırır.
- Önemi: Bu komutu koymazsak işlemci sonsuza kadar çalışmaya veya boş hafıza hücrelerini okumaya devam eder.

UYGULAMA: PYTHON SIMÜLASYONU
**SİMÜLASYONUN ÇALIŞMA ALGORITMASI



VERI GIRIŞI VE HATA DENETIM ALGORITMASI

```
print("--- Brookshear Makine Dili Yorumlayıcısı ---")
print("Çıkış yapmak için 'exit', 'cikis' veya 'çıkış' yazabilirsiniz.\n")

while True:
    # Kullanıcıdan veriyi alıyorum ve büyük harfe çeviriyorum.
    # .strip() ile baştaki/sondaki boşlukları siliyorum, .upper() ile de küçük harf girilse bile büyütüyorum.
    giris = input("Onaltılık (Hex) kodu giriniz (Örn: 14A3): ").strip().upper()

    # Çıkış kontrolü yapıyorum.
    # Kullanıcı 'exit', 'cikis' veya Türkçe karakterle 'çıkış' yazarsa döngüyü kırıp çıkıyorum.
    cikis_kelimeleri = ["EXIT", "CIKIS", "ÇIKIŞ", "ÇIKIS", "EXİT"]
    if giris in cikis_kelimeleri:
        print("Programdan çıkılıyor...")
        break

    # Hata Kontrollerini yapıyorum.

    # Uzunluk kontrolü: Kodun tam olarak 4 karakter olup olmadığına bakıyorum.
    if len(giris) != 4:
        print("HATA: Lütfen tam olarak 4 haneli bir kod giriniz!")
        continue

    # Karakter kontrolü: Sadece onaltılık (Hex) karakterler mi var diye kontrol ediyorum.
    gecerli_karakterler = string.hexdigits # "0123456789abcdefABCDEF" listesini getiriyorum.

    # Girilen her bir harfi tek tek geziyorum, eğer listede olmayan bir harf varsa hata veriyorum.
    if not all(char in gecerli_karakterler for char in giris):
        print("HATA: Sadece 0-9 arası rakamlar ve A-F arası harfler kullanabilirsiniz!")
        continue
```

- Sonsuz Döngü: Program while True döngüsü ile sürekli yeni komut bekler duruma getirilmiştir.
- Veri Normalizasyonu: Kullanıcı küçük harf girse bile .upper() fonksiyonu ile otomatik olarak büyük harfe çevrilir (Örn: 14a3 -> 14A3).
- Hata Denetimi: Girilen verinin tam 4 hane olup olmadığı ve geçerli Hex karakterlerden (0-9, A-F) oluşup oluşmadığı kontrol edilir. Hatalı girişlerde sistem uyarı verir.

KOMUT ÇÖZÜMLEME (DECODING) VE AYRIŞTIRMA

```
# Kodu anlamlı parçalara ayırıyorum.
# Brookshear mimarisinde her hanenin yerinin bir anlamı var, bunları değişkenlere atıyorum.

islem_kodu = giris[0]   # İlk hane: Ne yapacağımı söyleyen kod (Op-code).
yazmac     = giris[1]   # İkinci hane: İşlemin yapılacağı hedef yazmaç (Register).

# Bazı komutlar son iki haneyi bir adres veya sayı değeri olarak kullanır, onları birleştiriyorum.
adres_veya_sayi = giris[2] + giris[3]

# Bazı komutlar (Toplama gibi) son iki haneyi iki ayrı kaynak yazmaç olarak kullanır.
kaynak_yazmac_1 = giris[2]
kaynak_yazmac_2 = giris[3]
```

- İşlemciye giren 4 haneli ham veri, anlamlı parçalara bölünür (Parsing).
- Op-code (İşlem Kodu): Komutun ilk hanesi giris[0], yapılacak işlemi belirler.
- Register (Hedef): İkinci hane giris[1], sonucun nereye yazılacağını belirler.
- Operand (Veri): Son iki hane giris[2:], kullanılacak adresi veya sayıyı temsil eder.
- Bu yapı, CPU içindeki Instruction Register (IR) biriminin yazılımsal simülasyonudur.

YÜRÜTME BIRIMI VE KARAR YAPILARI

```python
# İşlem koduna göre ne yapacağıma karar veriyorum (Match-Case Yapısı).
match islem_kodu:
    case '1': # LOAD: Bellekten veriyi alıp yazmaca yüklüyorum.
        print(f"SONUÇ: {adres_veya_sayi} adresindeki veriyi alıyorum, {yazmac} numaralı Register'a kopyalıyorum.")

    case '2': # LOAD: Verilen sayıyı doğrudan yazmaca yüklüyorum.
        print(f"SONUÇ: {adres_veya_sayi} sayısını (değerini), doğrudan {yazmac} numaralı Register'a yüklüyorum.")

    case '3': # STORE: Yazmaçtaki veriyi belleğe kaydediyorum.
        print(f"SONUÇ: {yazmac} numaralı Register'daki veriyi alıp, {adres_veya_sayi} bellek adresine yazıyorum.")

    case '4': # MOVE: Bir yazmaçtaki veriyi diğerine kopyalıyorum.
        print(f"SONUÇ: {kaynak_yazmac_1} numaralı kaydedicideki veriyi alıp, {kaynak_yazmac_2} numaralı Register'a kopyalıyorum.")
```

- Ayrıştırılan Op-code değerine göre ilgili fonksiyon çağrılır.
- Match-Case Yapısı: Python 3.10 ile gelen modern match-case yapısı kullanılarak kodun okunabilirliği ve performansı artırılmıştır.
- Her bir case bloğu, donanımdaki bir elektronik devre yolunu temsil eder.
- Tanımsız bir kod girildiğinde case _: bloğu devreye girerek hata mesajı verir.

SISTEM TESTI VE SIMÜLASYON SONUÇLARI

```
--- Brookshear Makine Dili Yorumlayıcısı ---
Çıkış yapmak için 'exit', 'cikis' veya 'çıkış' yazabilirsiniz.

Onaltılık (Hex) kodu giriniz (Örn: 14A3): 15c4
SONUÇ: C4 adresindeki veriyi alıyorum, 5 numaralı Register'a kopyalıyorum.
------------------------------
Onaltılık (Hex) kodu giriniz (Örn: 14A3): 12b8
SONUÇ: B8 adresindeki veriyi alıyorum, 2 numaralı Register'a kopyalıyorum.
------------------------------
Onaltılık (Hex) kodu giriniz (Örn: 14A3): 328c
SONUÇ: 2 numaralı Register'daki veriyi alıp, 8C bellek adresine yazıyorum.
------------------------------
Onaltılık (Hex) kodu giriniz (Örn: 14A3): çıkış
Programdan çıkılıyor...
```

- Geliştirilen simülasyonun test aşamasıdır.
- Doğru Girdiler: Komutların başarıyla işlendiği ve Türkçe açıklamalarla kullanıcıya sunulduğu görülmektedir.
- Hatalı Girdiler: Geçersiz karakter veya eksik hane girildiğinde sistemin çökmediği, kullanıcıyı uyardığı test edilmiştir.

KAYNAKÇA

- Computer Science: An Overview, J. Glenn Brookshear (13. Basım)
- Python 3.10 Documentation (Match-Case Structures)
- Von Neumann Architecture Principles (IEEE Computer Society)

TEŞEKKÜRLER

Sümeyye Şimşek
2436085905