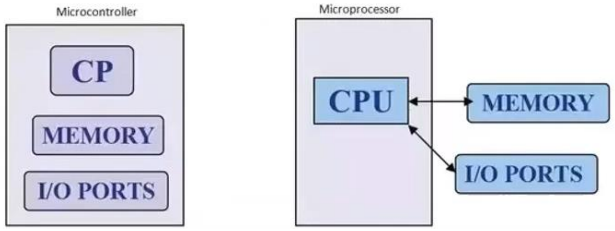
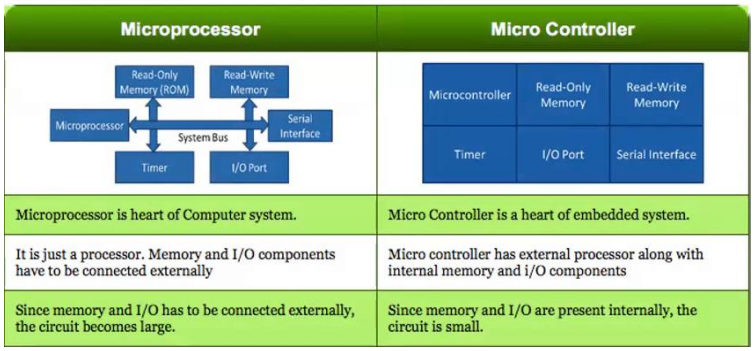
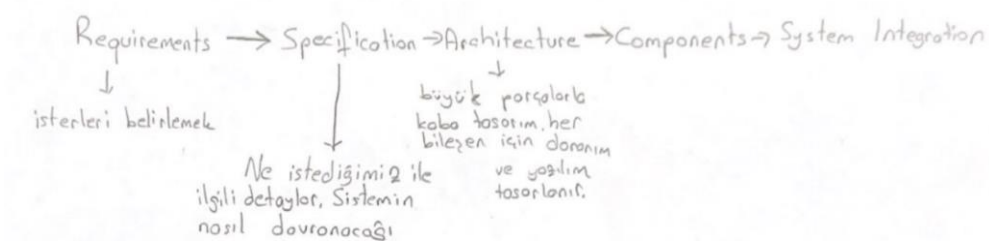
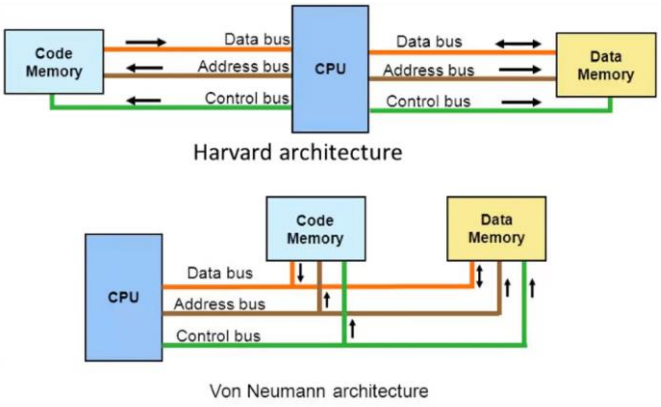
GİRİŞ

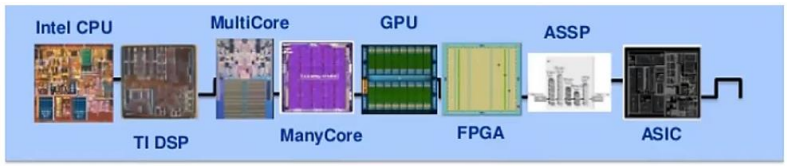
- Microcontroller vs microprocessor:
  - 

  - 

- Gömülü Tasarım Süreçleri:
  - 

    - Tecrübemiz varsa bottom-up design tercih edilebilir (tersten)
- Von Neumann vs Harvard:
  - 

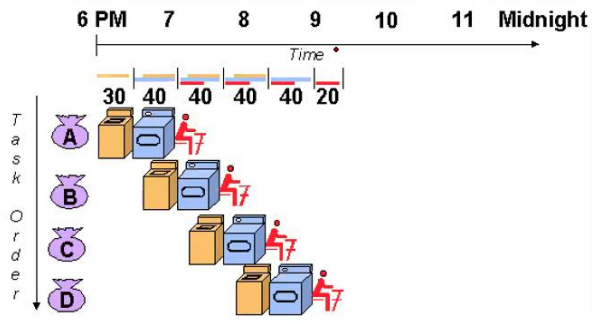
MİMARİLER

- RISC vs CISC:
  - CISC:
    - Geniş komut kümesi (100-300)
    - Karmaşık komutlar ve farklı adresleme kipleri
    - Doğrudan bellek üzerinde erişim yapan komutlar
    - Çalıştırma için birden fazla çevrim gerektirir
    - İşlemci yapıları karmaşık ve yüksek ücretlidir
    - Bellek kullanımı RISC'e göre daha verimlidir
  - RISC:
    - Dar komut kümesi (10-30)
    - Basit komutlar ve az sayıda adresleme kipleri
    - Belleğe sadece yazma ve okuma için erişilir
    - İşlemci yapıları basit ve düşük ücretlidir
    - Kullanıcının aynı işlem için daha fazla kod yazması gerekir
    - Az güç tüketirler
  - $$\frac{\text{time}}{\text{program}} = \frac{\text{time}}{\text{cycle}} \times \frac{\text{cycles}}{\text{instruction}} \times \frac{\text{instructions}}{\text{program}}$$
- FPGA (Field Programmable Gate Array):
  - Lojik devrelerle daha spesifik tasarım, geliştirme daha uzun sürer
  - Daha yüksek performans
  - Genelde RTOS'ta kullanılır
- ASIC (Application Specific Integrated Circuit):
  - SoC
  - İstenilen görevler için özel dizayn edilmiş çipler
  - Sadece istenilen amacı gerçekleştirdiği için düşük güç tüketir
  - Yüksek hacimli satışlar ile maliyet çıkarılabilir
  - Sıfırdan geliştirildiği için dizayn riski ve çevrimi fazla
  - Örn: A12 Bionic, Qualcomm Snapdragon, M1
Characteristic	Processor/ DSP	FPGA	ASIC
Programmability	High	Medium	Low
Development Cycle	HW+SW	HW+SW	HW
Area Efficiency	Medium	Low	High
Power Efficiency	Medium	Low	High
Efficiency	Low	Medium	High
- 

- CPU:
  - Market-agnostic
  - Accessible to many programmers (C++)
  - Flexible, portable

  FPGA:
  - Somewhat Restricted Market
  - Harder to Program (Verilog)
  - More efficient than SW
  - More expensive than ASIC

  ASIC
  - Market-specific
  - Fewer programmers
  - Rigid, less programmable
  - Hard to build (physical)

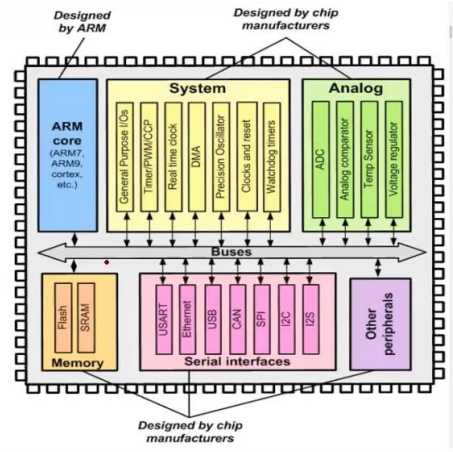
- Adres, Veri ve Kontrol Yolu:
  - Adres Yolu (Address Bus):
    - Tek yönlü, hafıza ve IO adreslerini içerebilir
    - 16-bit adres yolu olan bir işlemci 64KB alanı adresleyebilir.
      - 2^16 = 65.536
    - 32-bit adres yolu olan bir işlemci 4GB alanı adresleyebilir
      - 2^32 = 4.294.967.296
  - Veri Yolu (Data Bus):
    - İki yönlüdür. İkili veri ve komutları transfer eder
  - Kontrol Yolu (Control Bus):
    - Belleğe yazma mı okuma mı yapılacağını belirtir
- Komut İşletimi (4 aşamalı pipeline):
  - Fetch (Komutu al getir)
  - Decode (Komutu çöz)
  - Execute (Komutu çalıştır)
  - Write back (Registerlara gerekli değerleri yaz):
  - 

PIC, MSP430, ARM

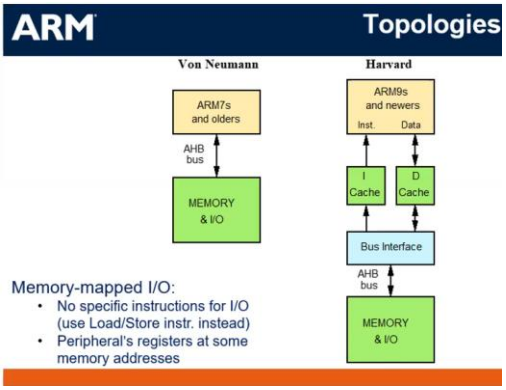
- PIC Mikrodenetleyici:
  - Genel Özellikler:
    - Düşük ücretli
    - Harvard mimarisinde
    - RISC mimarisinde -> 50'den az komut seti ve az sayıda adresleme modu
    - 8-bit veri yolu
  - Hız:
    - Instruction speed = 1/4 clock speed, yani
    - 4 clock cycle -> 1 instruction
    - 20MHz'lik clock -> 5MIPS
  - Avantajlar:
    - Bulunması en kolay mik. dnt.
    - En ucuz ve en basit
    - Harvard mimarisi kullanıldığından veri yolları farklı genişliklerdedir
    - Sabit kesme gecikmesi
  - Dezavantajları:
    - Tek akümülatör ve küçük komut seti
    - Program hafızasına doğrudan ulaşılamaz
    - Adresleme modları oldukça az çoğunlukla hafızaya doğrudan erişerek yapılır
  -

- MSP430 Mikrodenetleyicileri:
  - Genel Özellikler:
    - Saf 16-bit
    - Düşük ücretlidirler ve düşük güç tüketim modlarında çok düşük güç tüketirler
    - Von Neumann mimarisinde
    - RISC mimarisinde
    - 16 tane genel amaçlı register
    - 7 farklı adresleme modu
    - Hafızadan hafızaya veri transferi
    - Birden fazla clock (MCLK, ACLK)
  - Adresleme Modları:
    - **Table 4.3** MSP430 addressing modes

Addressing mode	Syntax	Comment
Immediate mode	#X	Data is X
Register Mode	Rn	Data is Register Rn Contents
Data in Memory:		
Indexed mode	X(Rn)	Address is X + Rn contents
Indirect mode	@Rn	Address is Register Rn contents
Indirect autoincrement	@Rn+	Address is Register Rn contents as before. Now, Rn is incremented after execution by 2 for word instructions and by 1 for byte ones.
Direct mode	X	Address is X
Absolute mode	&X	Address is X

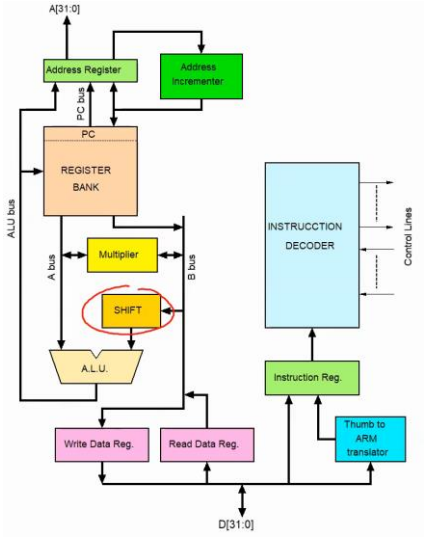
- ARM:
  - 

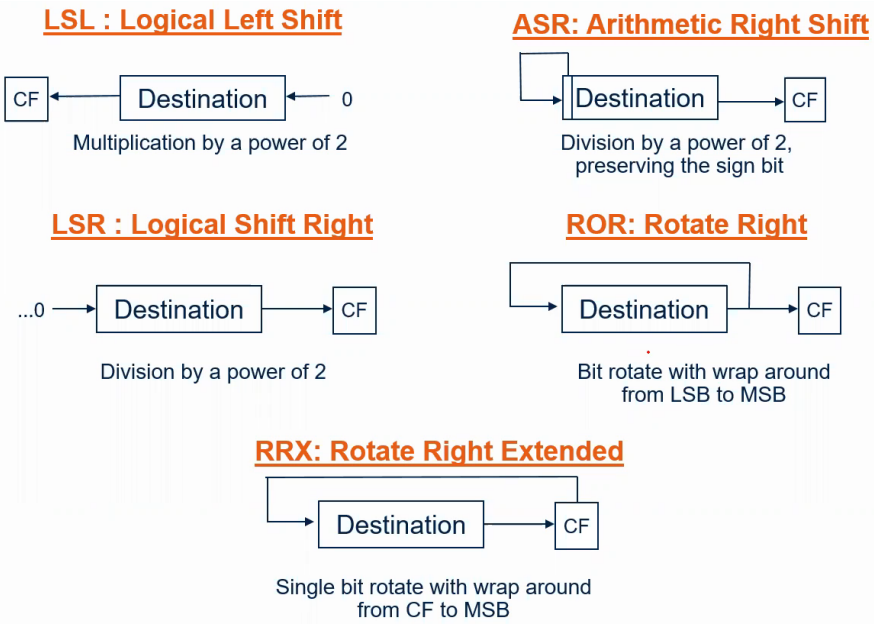
  - ARM Tabanlı İşlemciler:
    - İyi MIPS/Watt oranı
    - Taşınabilir ve işlem gücü gerektiren cihazlar için ideal
    - 32-bit veya 64-bit
    - Özel donanımları ile hızlı matematik ve çarpım işlem kapasitesi.
    - Tek yongada USB, Ethernet, CAN, I2C vb kompleks donanımlar bulunur
    - PIC ve MSP430'a göre daha fazla register
    - SIMD (Single Instruction Multiple Data)

- AHB:
  - Advanced High Performance Bus
  - ARM9 ve sonrasında Harvard mimarisi ile Komut ve Veri Arabellekleri kullanılmıştır.
  - 

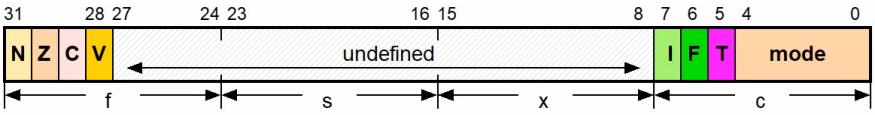
- ARM Versiyonları Karşılaştırma:
  - ARM family attribute comparison.

year	1995	1997	1999	2003
	ARM7	ARM9	ARM10	ARM11
Pipeline depth	three-stage	five-stage	six-stage	eight-stage
Typical MHz	80	150	260	335
mW/MHz[a]	0.06 mW/MHz	0.19 mW/MHz (+ cache)	0.5 mW/MHz (+ cache)	0.4 mW/MHz (+ cache)
MIPS[b]/MHz	0.97	1.1	1.3	1.2
Architecture	Von Neumann	Harvard	Harvard	Harvard
Multiplier	8 × 32	8 × 32	16 × 32	16 × 32

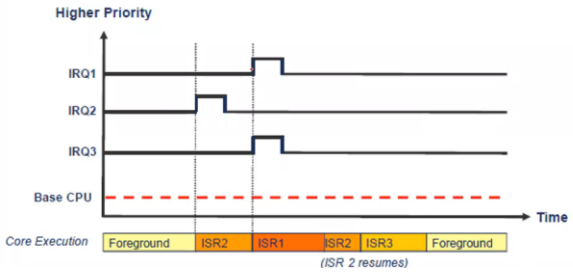
[a] Watts/MHz on the same 0.13 micron process.
[b] MIPS are Dhrystone VAX MIPS.
- ARM7TDMI:
  - Genel Özellikler:
    - 16 ADET 32-bit register
    - 3 aşamalı pipeline
    - Von Neumann
    - Thumb modu ile 16-bit komutlar işleyebilir
    - 8x32 donanımsal çarpıcı (MAC)
    - Barrell shifter ile bit kaydırma
  - Block Diagram:
    - 

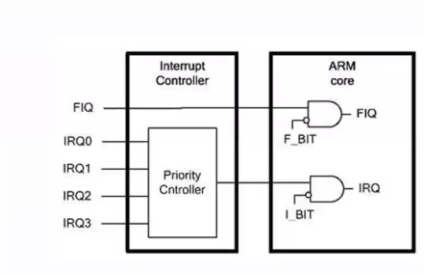
- Barrell Shifter:
  - 

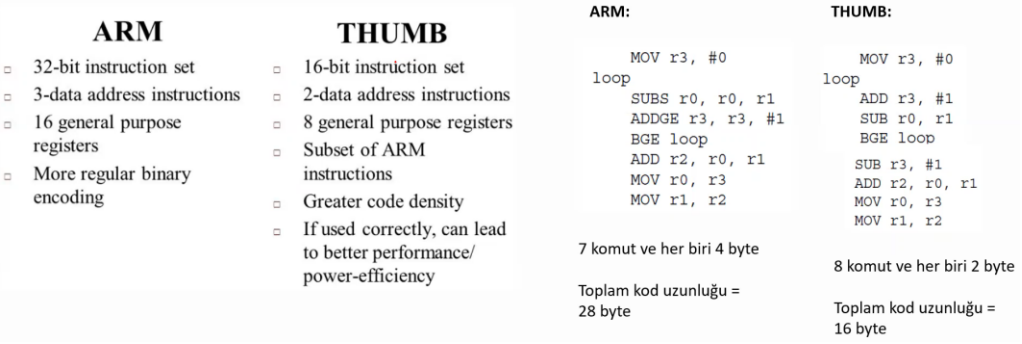
- Optional Shift Operation Example:
  - ADD R1, R2, R3 : R1 <- R2 + R3
  - ADD R1, R2, R3, LSL #2 : R1 <- R2 + (R3 * 4)

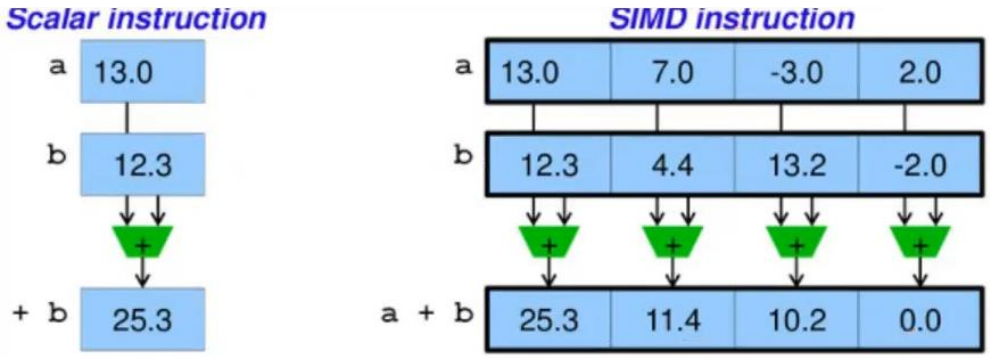
- ARM İşlemci Modları:
  - User – Normal çalışma modu
  - FIQ – Yüksek hızlı veri transferi için hızlı interrupt (Yüksek öncelik)
  - IRQ – Genel amaçlı interrupt (Düşük öncelik)
  - Çalışma modu değişimlerinde R13 ve R14 moda özel görev yapar.
    - R0-R12 – Genel amaçlı registerlar
    - R13 – Stack pointer
    - R14 – Link register (Fonksiyon tamamlandığında dönülecek adres)
    - R15 – Program counter
    - FIQ sırasında R8-R12 arası sıfırlanabilir
  - Program Durum Kaydedicileri (CPSR – Current Program Status Register):
    - 

    - I = 1: Disables IRQ
    - F = 1: Disables FIQ
    - T = 1: Thumb mode on
    - SPSR: Saved program status register
  - Kesme Geldiğinde
    - Yeni çalışma modunun R14'üne geri dönüş adresi saklanır
    - CPSR değeri yeni çalışma modunun SPSR'sine kopyalanır
    - CPSR yeni çalışma moduna göre güncellenir
    - R15 (PC) ilgili kesme vektör adresini yükleyerek kesme başlatılır
    - İşlem bitince CPSR ve R14 geri yüklenir

- Exception Handling (Interrupts):
  - 

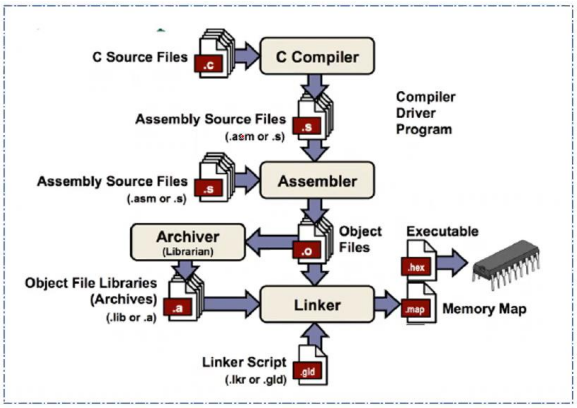
  - 

- Thumb Mode:
  - 

- DMA (Direct Memory Access):
  - Veri bir yerden alınıp başka bir yere transfer ediliyorsa ve aritmetik bir işlem yapılmıyorsa CPU bu veriyi alıp başka bir yere yazmak için gereksiz meşgul oluyor
  - Tamamen veri transferinden sorumlu bir birim olan DMA bu işi üstlenerek CPU'yu rahatlatıyor
- SIMD:
  - 

  - Bkz.: Vectorization

ARM ASSEMBLY

- Assemblers and Compilers:
  - 

- Instructions:
Operation Mnemonic	Meaning
ADC	Add with Carry
ADD	Add
AND	Logical AND
BAL	Unconditional Branch
B⟨cc⟩	Branch on Condition
BIC	Bit Clear
BLAL	Unconditional Branch and Link
BL⟨cc⟩	Conditional Branch and Link
CMP	Compare
EOR	Exclusive OR
LDM	Load Multiple
LDR	Load Register (Word)
LDRB	Load Register (Byte)
MLA	Multiply Accumulate
MOV	Move
MRS	Load SPSR or CPSR
MSR	Store to SPSR or CPSR
MUL	Multiply
  - **ADD** operand1 + operand2 ; Add
    ADC operand1 + operand2 + carry ; Add with carry
    SUB operand1 - operand2 ; Subtract
    SBC operand1 - operand2 + carry -1 ; Subtract with carry
    RSB operand2 - operand1 ; Reverse subtract
    RSC operand2 - operand1 + carry - 1 ; Reverse subtract with carry
- Postfixing:
  - Assemby

    ```
    CMP   r3,#0
    BEQ   skip
    ADD   r0,r1,r2
skip
    ```

    ARM Assembly

    ```
    CMP   r3,#0
    ADDNE r0,r1,r2
    ```
Suffix	Description	Flags tested
EQ	Equal	Z=1
NE	Not equal	Z=0
CS/HS	Unsigned higher or same	C=1
CC/LO	Unsigned lower	C=0
MI	Minus	N=1
PL	Positive or Zero	N=0
VS	Overflow	V=1
VC	No overflow	V=0
HI	Unsigned higher	C=1 & Z=0
LS	Unsigned lower or same	C=0 or Z=1
GE	Greater or equal	N=V
LT	Less than	N!=V
GT	Greater than	Z=0 & N=V
LE	Less than or equal	Z=1 or N=!V
AL	Always	

    kırmızı işaretlileri bil yeter

- General Syntax:
  - {OPCODE}: İşlem sonucunda status flagler etkilenmez (MOV, ADD)
  - {OPCODE} + {CONDITION} İşlem belli bir koşulda yapılır (MOVEQ, ADDGT)
  - {OPCODE} + S: İşlem sonucuna göre status flagler güncellenir (MOVS, ADDS)
  - {OPCODE} + {CONDITION} + S: (MOVEQS, ADDLES)

- Barrell Shifter Instructions

```
MOV R2, R0, LSL #2      ; Shift R0 left by 2, write to R2, (R2=R0x4)
ADD R9, R5, R5, LSL #3  ; R9 = R5 + R5 x 8 or R9 = R5 x 9
RSB R9, R5, R5, LSL #3  ; R9 = R5 x 8 - R5 or R9 = R5 x 7
SUB R10, R9, R8, LSR #4 ; R10 = R9 - R8 / 16
MOV R12, R4, ROR R3     ; R12 = R4 rotated right by value of R3
```

- Logical Operations:
  - AND r8, r7, r2
    - R8 = R7 & R2
  - ORR r11, r11, #1
    - R11 |= 1
  - BIC r11, r11, #1
    - R11 &= ~1
  - EOR r11, r11, #1
    - R11 ^= 1

- Comparison Operations:

```
– CMP  operand1 - operand2   ; Compare
– CMN  operand1 + operand2   ; Compare negative
– TST  operand1 AND operand2 ; Test
– TEQ  operand1 EOR operand2 ; Test equivalence
```

- Single Register Data Transfer:

```
STR r0, [r1]    ; Store contents of r0 to location pointed to
                ; by contents of r1.
LDR r2, [r1]    ; Load r2 with contents of memory location
                ; pointed to by contents of r1.
```

  - STRB, LDRB: Bitwise
  - <LDR | STR> + {CONDITION}? + {SIZE}?: (LDREQB)
  - Pre-Indexed Addressing:
    - STR R0, [R1, #12]: R0'ın içeriğini, (R1'in gösterdiği adres + 12) adresine kopyala
    - STR R0, [R1, #12]!: Same operation with auto-increment R1 by 12
  - Post-Indexed Addressing:
    - STR R0, [R1], #12: R0'ın içeriğini, R1'in gösterdiği adrese kopyala ve sonra R1'in değerini 12 arttır.

- Addressing Modes Summary:

1. Immediate:-

```
MOV  R0 , #25H
ADD R0,R1,#25H
```

2. Register:-

```
MOV  R0 , R1
ADD R0,R1,R2
```

3. Direct:-

```
LDR  R0, Var
STR  R0, Var
```

4. Indirect:-

```
LDR  R0.[R1]
STR  R0,[R1]
```

5.Register Relative:-

Normal:-
LRD R0,[R1,#04H]

PreIndex:-
LRD R0,[R1,#04H]!

PostIndex:-
LRD R0,[R1],#04H

6.Base Indexed:-

Normal:-
LRD R0,[R1,R2]

PreIndex:-
LRD R0,[R1,R2]!

PostIndex:-
LRD R0,[R1],R2

7. Base with scaled Index:-

Normal:-
LRD R0,[R1,R2,LSL #4]

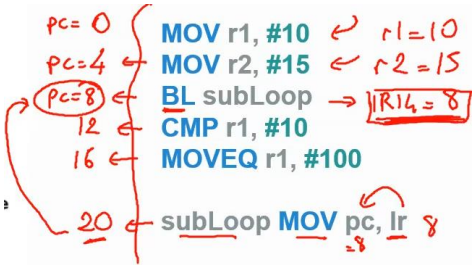
PreIndex:-
LRD R0,[R1,R2,LSL #4]!

PostIndex:-
LRD R0,[R1],R2,LSL #4

- Örnek:
  - C:
    ```
    z = (a << 2) | (b & 15);
    ```
  - Assembler:
    ```
    ADR r4,a ; get address for a
    LDR r0,[r4] ; get value of a
    MOV r0,r0,LSL 2 ; perform shift
    ADR r4,b ; get address for b
    LDR r1,[r4] ; get value of b
    AND r1,r1,#15 ; perform AND
    ORR r1,r0,r1 ; perform OR
    ADR r4,z ; get address for z
    STR r1,[r4] ; store value for z
    ```
- Define Constant:
  - DCD: Defined Word value storage area

    ```
    aVal DCD 8
    bVal DCD 0

        ADR r4, aVal
        LDR r0, [r4]
        MOV r0, r0, lsl #2
    ```
- Branch With Link:
  - Dallanır ve şu anki PC değerini LR yani R14'e yedekler

- For Loop:
  - C Program

    ```
    int i;
    int sum = 0;
    for(i = 0; i < 10; i++){
      sum += i;
    }
    ```

    =>

    ```
            MOV r0, #0  ; i
            MOV r1, #0  ; sum

    loop    CMP r0, #10
            BGE endloop
            ADD r1, r1, r0
            ADD r0, r0, #1
            B   loop
    endloop
    ```
  - ```
    int main(int argc, char **argv)
    {

        int v[] = {1, 5, 4, 7, 9, 1, 6, 5};

        for (int i = 0; i < 8; ++i)
            v[i] += 10;

        return 0;
    }
    ```

    =>

    ```
    ARRAY DCD 1, 5, 4, 7, 9, 1, 6, 5
    LDR r0, =ARRAY
    MOV r1, #0x0

    FOR
        CMP r1, #8 ; r1 indisi tutuyor
        BGE BITIR ; 8'e eşit veya büyükse bitir
        LSL r2, r1, #2 ; r1'i 2 kaydır
        LDR r3, [r0,r2] ; r0'ın adresinin r2 sonrasını yükle
        ADD r3, r3, #10 ; r3 toplam değeri
        STR r3, [r0,r2]
        ADD r1,r1,#1 ; r1 program sayacını bir arttır
        B FOR

    BITIR
    ```
- Stack:
  - STR/LDR – Store/Load Registers
  - STM/LDM – Store/Load Multiple Registers
  - IA or EA – Increase SP After
  - IB or FA – Increase SP Before
  - DA or ED – Decrease SP After
  - DB or FD – Decrease SP Before
    ```

- Example:
  - ```c
    int main(int argc, char **argv)
    {

        int y;
        y = diffofsums(2, 3, 4, 5);
    }

    int diffofsums(int f, int g, int h, int i)
    {

        int result;
        result = (f + g) - (h + i);
        return result;
    }
    ```
  - First, store all registers to stack
  - Then, use it for the equations
  - Finally, restore all used registers to original
  - ```
    MAIN
        MOV r0, #2
        MOV r1,#3
        MOV r2, #5
        MOV r3, #2
        BL DIFFOFSUMS
    END

    DIFFOFSUMS
        SUB SP, SP, #0xc
        STR r9, [sp,#0x8]
        STR r8, [sp,#0x4]
        STR r4, [sp]

        ADD r8, r0,r1
        ADD r9, r2,r3
        SUB r4,r8,r9
        MOV r0,r4

        LDR r4, [sp]
        LDR r8,[sp,#0x4]
        LDR r9, [sp,#0x8]
        ADD sp,sp,#0xc

        MOV pc, lr
    ```
    or
    ```
    MAIN
        MOV r0, #2
        MOV r1,#3
        MOV r2, #5
        MOV r3, #2
        BL DIFFOFSUMS
        END

    DIFFOFSUMS
        STMFD sp!,{r4,r8,r9}

        ADD r8, r0,r1
        ADD r9, r2,r3
        SUB r4,r8,r9
        MOV r0,r4
        LDMFD sp!,{r4,r8,r9}

        MOV pc, lr
    ```
- Raspberry Pi:
  - Normal seri ve zero seri var
  - Zero seriler: daha az güç çeken ve kablosuz ağlarla çalışabilen tipler
  - Raspberry Pi 4 ile;
    - 1.5MHz, 64-bit quad core ARM Cortex-A72
    - 4K çözünürlük ve USB-C