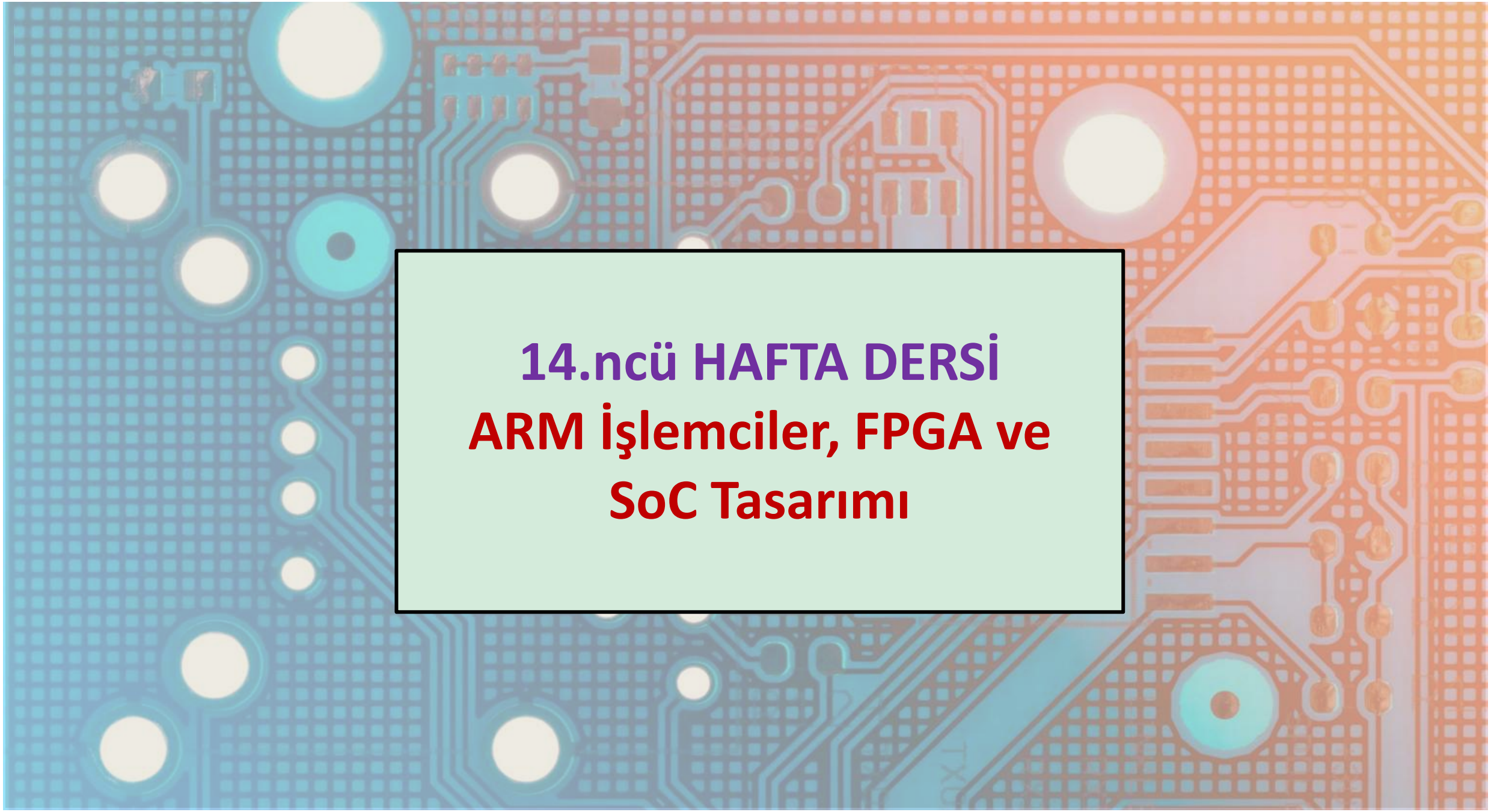
BİLGİSAYAR MİMARİSİ VE ORGANİZASYONU

Ders Kodu : BM303
L+T+P : 0-3-1
AKTS : 4

Dersi Veren :
Prof. Dr. Ahmet Akbaş

Mühendislik Fakültesi
Bilgisayar Müh. Bölümü



BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ

BM303

BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU

Prof. Dr. A. Akbaş

Ders İçeriği :

- Gömülü Sistemler
- SoC
- FPGA
- ARM işlemciler
- Komut seti mimarisi
- Giriş/çıkış yeteneği
- Gömülü uygulamalar için destek

14.ncü HAFTA DERSİ

1- Bilgisayar Sistemlerine Giriş (Fonksiyonel Birimler, Sistem Mimarisi, Performans, Teknolojik Arka Plan);

2- Programcı Bakış Açısıyla bir Sistem Turu;

3- Bilgisayar Sistemlerinde Bilginin (information) Temsili ve Manüplasyonu;

4- Komut Seti Mimarisi, Programların Makine Düzeyinde Temsili-I (RISC ve CISC mimariler);

5- Komut Seti Mimarisi, Programların Makine Düzeyinde Temsili-II (IA-32 ve x86-64 mimarileri);

6- İşlemci ve Giriş/Çıkış Birimleri Arasında Veri Transferi (Program Kontrollü G/Ç, Kesmeli G/Ç);

7- Yazılım (Program Hazırlama ve Çalıştırma Süreçleri, C ve Assembly Dilleri Arasındaki Etkileşim, İşletim Sistemleri);

8- İşlemci (CPU) Birimi (Fonksiyonel Birimler, Komut Getirme vr Yürütme Adımları, Kontrol Sinyalleri, HDL);

9- Program Performansının Optimizasyonu (Komut Düzeyinde Paralelleme, Eş-zamanlılık, Süperskaler İşlemciler);

10- Giriş/Çıkış Organizasyonu (Yol Yapısı, Yol Hakimliği, Arayüz Devreleri, Ara-bağlantı Standartları);

11- Bellek Sistemi (Temel Bellek Devreleri, Ana Bellek Organizasyonu, Bellek Teknolojileri);

12- Aritmetik (Tam Sayı ve Kayan Noktalı Sayılar için Aritmetik İşlem Devreleri);

13- Paralel Çalışma ve Performans (Çoklu İş Parçaları, Vektörel Süreçler, Paylaşımlı Bellekli Çoklu İşlemciler);

14- ARM İşlemciler, FPGA ve SoC Tasarımı.

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ
BM303
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU
Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

GÖMÜLÜ SİSTEMLER

Gömülü sistemler, genel amaçlı hesaplama yerine belirli bir fiziksel süreçle ilgili verilerin işlendiği, haberleşme ve/veya kontrol gibi özel bir takım görevlerin gerçekleştirildiği bilgisayar sistemleridir. Temel ilkeleri genel amaçlı bilgisayar sistemleri ile aynidir. Genellikle sistem açıldığında otomatik olarak yürütülmeye başlanan tek bir uygulama programını yürütür ve bir işletim sistemi desteğine ihtiyaç duymazlar. Ancak, tasarımcıların komut seti, programlama, giriş/çıkış yöntemleri, bellek yapıları ve sisteme dahil olabilecek çeşitli cihazlarla ilgili arayüz şemaları hakkında kapsamlı bir bilgiye sahip olması gereklidir.

Gömülü sistemlerin bellek gereksinimleri az olduğundan sistem tasarımı için genellikle yeterli çip üstü belleğe sahip **mikrodenetleyiciler** kullanılır. Yüksek hesaplama kapasitesi ile beraber maliyet ve güç tüketiminin nispeten düşük olması gereken gömülü uygulamalar için **ARM mimarisi** caziptir. **ARM işlemci çekirdekleri** tüm sistemin bir çip üzerinde gerçekleştirme amacına uygundur. Bu kapsamda **ARM6**, **ARM7**, **ARM9**, **ARM10** ve **ARM Cortex** dahil olmak üzere bir dizi **ARM işlemci çekirdeği** geliştirilmiştir. Bununla beraber, **ARM mikrodenetleyiciler** pazarda ayrıca ayrı çipler olarak da bulunmaktadır.

Temel **ARM mimarisi**, **32 bit**lik bir organizasyon ve tüm komutların 3**2 bit** uzunluğunda olduğu bir **komut seti** kullanır. Bunun **Thumb** olarak bilinen ve **16 bit**lik komutlarla **16** bitlik veri transferleri kullanan başka bir sürümü, **ARM komutları**nın bir alt kümesini kullanır. Ayrıca **ARM** mimarisinden daha az kaydediciye sahiptir.

Thumb'ın bir avantajı, büyük ölçüde **16** bitlik komutlar içeren programları depolamak için daha küçük bir belleğe ihtiyaç duyulmasıdır. Bununla beraber, **Thumb** komutları yürütme sürecinde, normal **32** bitlik **ARM** komutlarına genişletilerek kullanılır.

Bu dersin ilk haftalarında tanıştığımız **sembolik makine dili** (assembly) programların bir kısmı **ARM** işlemcilerin komut seti kullanılarak yazılmıştır. Tüm **ARM** işlemcileri temel bir makine komut setini paylaşır.

Burada kullanılan **ISA sürümü**, **ARM** tarafından **sürüm 4** olarak adlandırılır.

İleri sürümlerde, bu derste verilen örneklerde görülmeyen uzantılar eklenmiştir. Bu uzantılardan bazıları bu derste kısaca özetlenecektir.

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ
BM303
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU
Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

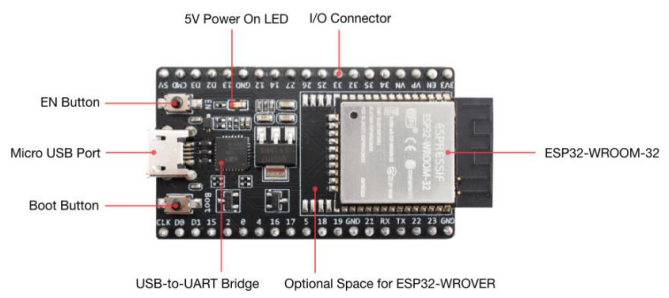
ARM İŞLEMCİLER ve SoC

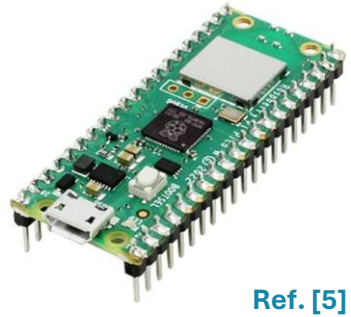
Basit gömülü sistem uygulamaları pazarda mevcut çeşitli mikrodenetleyici yongalarıyla gerçekleştirilebilir. Daha karmaşık fonksiyonlar için, güçlü işlemci veya mikrodenetleyicilerin kullanılması gerekir. Bu amaçla, güçlü işlemci çekirdeklerinin belirli görevleri gerçekleştiren hazır çevresel modüller ile entegre edildiği sistemler geliştirilir. Bu kapsamda, **ARM** komut setinde yer almayan kayan noktalı sayılar üzerinde aritmetik işlemler, dijital sinyaller veya video verilerinin işlenmesi gibi uygulamaları gerçekleştirmek için **yardımcı işlemciler** (coprocessor) kullanılır.

Yardımcı işlemci birimini **ARM** işlemcisiyle bütünleştiren **tek bir çip**, temel bir **ARM işlemcisini tanımlayan yazılımla birleştirilerek sentezlenebilir**. Yardımcı işlemcileri kullanan programlar **ARM** komut setine eklenen uzantılarla kolayca gerçekleştirilebilir. Bu şekilde tek bir yonga veya baskılı devre üzerinde entegre edilen bilgisayar sistemleri **Tek Yongalı Sistem** (**SoC**, **System on Chip**) şeklinde anılır. **ARM** işlemcisinin tasarımının temel amaçlarından biri de, onu **SoC** sistem ortamında kullanıma uygun hale getirmektir.

Günümüzde çok sayıda firma, tasarımcı firmalarla lisans anlaşmaları yaparak belirli bir uygulama için gerekli işlemci çekirdeği ve diğer modüllerin **fikri mülkiyet** (intellectual property, **IP**) haklarını satın almakta ve bu bileşenleri kendi tasarladıkları **SoC** sistemine entegre etmektedirler. Bu yaklaşımla geliştirilmiş iki ayrı **SoC** sistem örneğinin görüntüleri (**ESP32 DEVKiTC** ve **Raspery Pico W**) aşağıda verilmiştir.

SoC sistemlerinde entegre edilen yardımcı işlemciler, pazarda mevcut işlemci, bellek, giriş/çıkış arayüzleri, **A/D**, **D/A** veya **DSP** gibi ayrık devre modülleri olabilir. Pazara genellikle aşağıdaki resimlerdeki gibi iki sıralı pinlerle donatılmış bir entegre devre şeklinde sunulan bu sistemlerin tasarımı, uygulamaya özel diğer bileşenleri de içerecek şekilde **bilgisayar destekli tasarım** (**CAD**) araçları kullanılarak tamamlanmaktadır.





Ref. [5]

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ
BM303
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU
Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

FPGA ile SoC TASARIMI

CAD uygulamaları ile özel bir çipin tasarlanmasında maliyeti önemli bir faktördür. Bu tür çiplerin üretimi pahalıdır ve daha iyi performans ve daha düşük güç tüketimi sunsalar bile, ancak çok sayıda üretilerek maliyetlerinin düşürülmesi halinde kullanılmaları uygundur. Bu durumda alternatif olarak Alanda Programlanabilir Kapı Dizisi (Field Programmable Gate Array, **FPGA**) ürünleri kullanılabilir.

FPGA üreticileri, gömülü sistemlerin tasarımını nispeten kolaylaştıran güçlü **CAD** araçları sağlar. Tasarımcı, önceden tanımlanmış çeşitli modülleri dahil ederek ve uygulama gereksinimlerine uyacak şekilde parametreler atayarak bir sistem oluşturur. Bu tür modüllere örnek olarak **İşlemci çekirdekleri**, **Bellek modülleri** ve **arayüzleri**, **Paralel G/Ç arayüzleri**, **Seri G/Ç arayüzleri**, **Zamanlayıcı/sayaç devreleri** verilebilir.

Gömülü sistemde ihtiyaç duyulan tüm işlevler bu modüllerle gerçekleştirilebilir. Yine de yeterli olmazsa, ek özel devreler tasarlanarak sisteme dahil edilir.

SoC tasarımına genellikle, bir **işlemci çekirdeği** ve diğer parametreli modüllerden oluşan bir **alt sistem** tanımlama ile başlanır. Alt sistemi uygulayan bir modül oluşturmak için bir **CAD** aracı kullanılır.

Bu modülün tanımı bir **donanım tanımlama dili**nde (Hardware Design Language, **HDL**) yapılır. Daha sonra, oluşturulan uygulamaya özgü ek devrelerle birlikte genel tasarım özelleştirilir. Genel tasarımı **FPGA** cihazını yapılandırmak için kullanılabilecek bir biçimde sentezlemek ve uygulamak için farklı bir **CAD** aracı kullanılır.

Sonuç olarak, gömülü sistem tasarımcısı kaçınılmaz olarak en basit ve en uygun maliyetli yaklaşımı arar. Tüm bir sistemi uygulamak için yeterli kaynaklara sahip bir mikrodenetleyici çipi en iyi seçim olabilir. Sistemi gerçekleştirmek için ek çiplere ihtiyaç duyulursa durum farklıdır. Bu durumda sistemi uygulamak için daha az çipe ihtiyaç duyma olasılıkları yüksek olduğundan **FPGA** çözümleri cazip hale gelir.

Bir **FPGA** cihazı, tasarımcının her türlü dijital devreyi tasarlamasına olanak tanır. Modern **FPGA** cihazlarında çok büyük ve karmaşık devreler uygulanabilir.

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ
BM303
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU
Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

ARM İŞLEMCİLER

Advanced RISC Machines (**ARM**) **Limited**, bir **RISC** tarzı işlemci ailesi tasarlamıştır. **ARM**, bu tasarımları çip üretimi için diğer şirketlere, sistem geliştirme ve simülasyon için gerekli yazılım araçlarıyla birlikte lisanslamaktadır. **ARM** işlemcilerinin ana kullanım alanları, mobil telefonlar, iletişim modemleri ve otomotiv motor yönetim sistemleri gibi düşük güç ve düşük maliyetli gömülü sistem uygulamalarıdır.

ARM İşlemcilerin Özellikleri: **ARM** işlemcilerin kelime uzunluğu **32** bittir, bellek **32 bit adresler** kullanılarak **bayt adreslenebilir** ve işlemci **kaydedicileri** **32 bit** uzunluğundadır. Bellek ile işlemci kaydedicileri arasında veri taşımada **3** farklı **operant uzunluğu** kullanılır: **bayt** (**8** bit), **yarım kelime** (**16** bit) ve **kelime** (**32** bit). Kelime ve yarım kelime adresleri hizalanmalıdır, yani sırasıyla **4** ve **2**'nin katları olmalıdır. Hem küçükle sonlanan hem de büyükle sonlanan bellek adresleme şemalarını destekler. Seçim, harici bir giriş kontrol hattı tarafından belirlenir.

ARM ISA çoğu bakımdan **RISC** tarzı bir mimariyi yansıtır, ancak bazı **CISC** tarzı özelliklere de sahiptir.

RISC tarzı Özellikleri:

- Tüm komutların **32 bit** sabit uzunlukludur.
- Yalnızca **Load** ve **Store** komutları belleğe erişir.
- Tüm aritmetik ve mantık işlem komutları işlemci kaydedicilerindeki operantlar üzerinde çalışır.

CISC tarzı Özellikleri:

- Otomatik artırma, Otomatik azaltma ve PC'ye göre adresleme modları sağlar.
- Koşul kodları (**N**, **Z**, **V** ve **C**) dallanma ve komutların koşullu yürütülmesi için kullanılır.
- Birden fazla kayıt, ardışık bellek sözcüklerinden oluşan bir bloktan yüklenebilir veya tek bir komut kullanılarak bir blokta saklanabilir.

ARM İşlemcilerin Sıradışı Özellikleri: ARM mimarisi, modern işlemcilerde genellikle bulunmayan bir dizi özelliğe sahiptir. Bunlar, **Komutların Koşullu Yürütülmesi** ve **Kaydırma** veya **Bölme Komutlarının olmayışı**dır.

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ
BM303
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU
Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

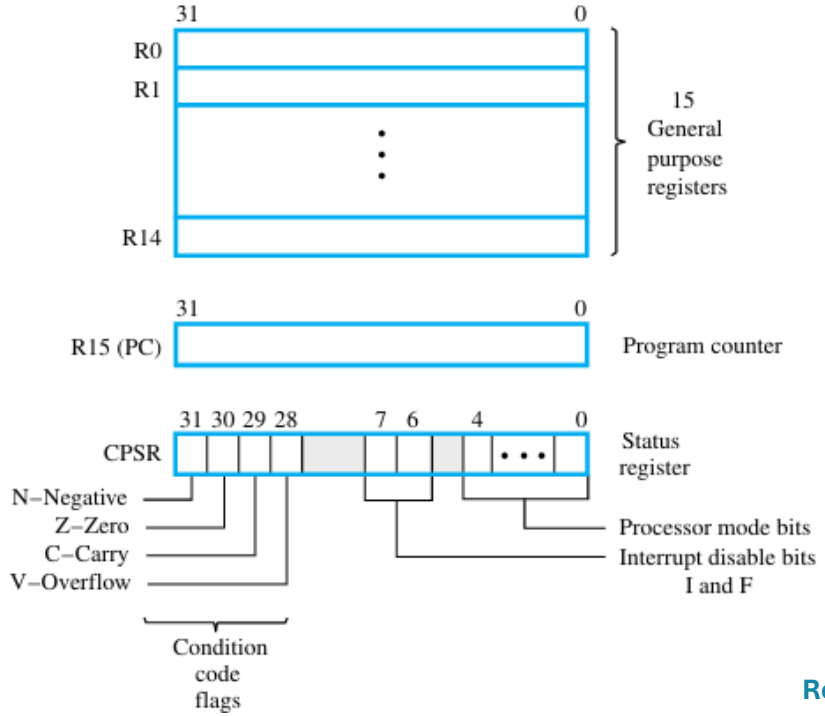
ARM İŞLEMCİ KAYDEDİCİLERİ

Komutların Koşullu Yürütülmesi: **ARM** işlemcilerinin sıra dışı bir özelliği, tüm komutların koşullu olarak yürütülmesidir. Bir komut, yalnızca koşul kodu bayraklarının geçerli değerleri komutun **4** bitlik bir alanında belirtilen koşulu karşılıyorsa yürütülür. Aksi takdirde, işlemci bir sonraki komuta geçer. Olası koşullardan biri, komutun her zaman yürütüleceğini belirtir. Koşullu yürütmenin yararlılığı, ileride bir örnekle gösterilecektir. Şimdilik, koşul alanının "her zaman yürütülen" kodu belirttiğini varsayacağız.

Kaydırma veya Bölme Komutları Yok: Kaydırma komutları açıkça sağlanmaz. Ancak, aritmetik, mantık ve taşıma komutlarındaki bir anlık değer veya kaydedici operantlarından biri, bir işlemde kullanılmadan önce öngörülen miktarda kaydırılabilir. Bu özellik, kaydırma komutlarını örtük olarak uygulamak için de kullanılabilir.

Birçok farklı Çarpma komutu vardır ve bunların çoğu sinyal işleme uygulamalarında kullanılmak üzere tasarlanmıştır. Ancak donanım Bölme komutu yoktur. Bölme yazılımda uygulanmalıdır.

ARM İşlemcilerin Kaydedici Yapısı: Kullanıcı uygulama programları için **R0** ile **R15** arasında etiketlenen **16** adet **32** bit işlemci kaydedicisi vardır. Bunların on beşi genel amaçlı kaydedici (**R0** - **R14**) ve **R15** de **Program Sayacı**dır (**PC**). Genel amaçlı kaydediciler bellek adreslerini veya veri operantlarını tutabilir. **R13** ve **R14**, işlemci yığınının ve alt rutinlerin yönetimiyle ilgili özel kullanıma sahiptir. .



Ref. [1]

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ
BM303
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU
Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

ARM İŞLEMCİ KAYDEDİCİLERİ

Kaydedici Yapısı: Bankalanmış kaydediciler adı verilen bir dizi ek genel amaçlı kaydedici vardır. Bunlar, **R0** ila **R14** kaydedicilerinin bazılarının kopyalarıdır. İşlemci Kullanıcı modundan diğer çalışma modlarına geçtiğinde farklı bankalanmış kaydediciler kullanılır. Bankalanmış kaydedicilerin kullanımı, mod anahtarlarında Kullanıcı modu kayıt içeriklerinin bir kısmını kaydetme ve geri yükleme ihtiyacını ortadan kaldırır. Durum kaydedicisinin kaydedilmiş kopyaları, Kullanıcı modu haricindeki modlarda da mevcuttur.

Bir istisna oluştuğunda, Kullanıcı modundan uygun istisna moduna geçişte aşağıdaki eylemler gerçekleştirilir:

1. Program Sayacı'nın (**R15**) içerikleri istisna modunun bankalanmış Bağlantı kaydedicisine (**R14_mode**) yüklenir.
2. Durum kaydedicinin (**CPSR**) içerikleri bankalanmış Kaydedilmiş Durum kaydedicisine (**SPSR_mode**) yüklenir.
3. **CPSR**'nin mod bitleri uygun istisna modunu temsil edecek şekilde değiştirilir ve kesme devre dışı bırakma bitleri **I** ve **F** ayarlanır.
4. Program Sayacı (**R15**), istisna için ayrılmış vektör adresiyle yüklenir ve bu adresteki komut, istisna hizmeti rutinini başlatmak için alınır ve yürütülür.

General-purpose registers and program counter

Ref. [1]

User/System	FIQ	IRQ	Supervisor	Abort	Undefined
R0	R0	R0	R0	R0	R0
R1	R1	R1	R1	R1	R1
R2	R2	R2	R2	R2	R2
R3	R3	R3	R3	R3	R3
R4	R4	R4	R4	R4	R4
R5	R5	R5	R5	R5	R5
R6	R6	R6	R6	R6	R6
R7	R7	R7	R7	R7	R7
R8	R8_fiq	R8	R8	R8	R8
R9	R9_fiq	R9	R9	R9	R9
R10	R10_fiq	R10	R10	R10	R10
R11	R11_fiq	R11	R11	R11	R11
R12	R12_fiq	R12	R12	R12	R12
R13	R13_fiq	R13_irq	R13_svc	R13_abt	R13_und
R14	R14_fiq	R14_irq	R14_svc	R14_abt	R14_und
R15	R15	R15	R15	R15	R15

Processor status register

CPSR	CPSR	CPSR	CPSR	CPSR	CPSR
	SPSR_fiq	SPSR_irq	SPSR_svc	SPSR_abt	SPSR_und

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ
BM303
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU
Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

ARM İŞLEMCİ ADRESLEME MODLARI

Adresleme Modları: Anında, Kaydedici, Mutlak, Dolaylı ve İndeks adresleme modlarının hepsi bir biçimde mevcuttur. Genellikle **RISC** işlemcilerde bulunan bu temel modlara ek olarak, Göreceli mod ve Otomatik Arttırma ve Otomatik Azaltma modlarının varyantları da sağlanır. **ARM** mimarisinde, bu modların çoğu farklı indeksli adresleme modlarından türetilmiştir.

Name	Assembler syntax	Addressing function
With immediate offset:		
Pre-indexed	[R*n*, #offset]	EA = [R*n*] + offset
Pre-indexed with writeback	[R*n*, #offset]!	EA = [R*n*] + offset; R*n* ← [R*n*] + offset
Post-indexed	[R*n*], #offset	EA = [R*n*]; R*n* ← [R*n*] + offset
With offset magnitude in R*m*:		
Pre-indexed	[R*n*, ± R*m*, shift]	EA = [R*n*] ± [R*m*] shifted
Pre-indexed with writeback	[R*n*, ± R*m*, shift]!	EA = [R*n*] ± [R*m*] shifted; R*n* ← [R*n*] ± [R*m*] shifted
Post-indexed	[R*n*], ± R*m*, shift	EA = [R*n*]; R*n* ← [R*n*] ± [R*m*] shifted
Relative (Pre-indexed with immediate offset)	Location	EA = Location = [PC] + offset

EA = effective address
offset = a signed number contained in the instruction
shift = direction #integer
where direction is LSL for left shift or LSR for right shift; and
integer is a 5-bit unsigned number specifying the shift amount
±R*m* = the offset magnitude in register R*m* can be added to or subtracted from the contents of base register R*n*

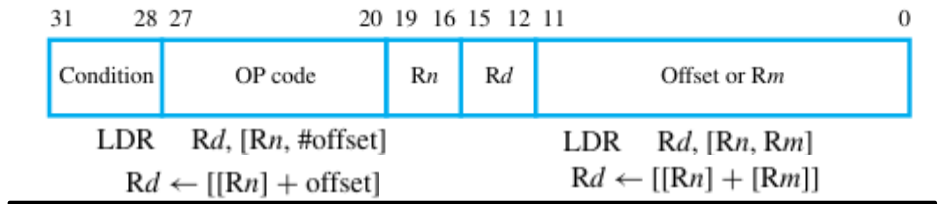
LDR R*d*, [R*n*]
R*d* ← [[R*n*]]

LDR R0, [R1, −R2]!
R0 ← [[R1] − [R2]]

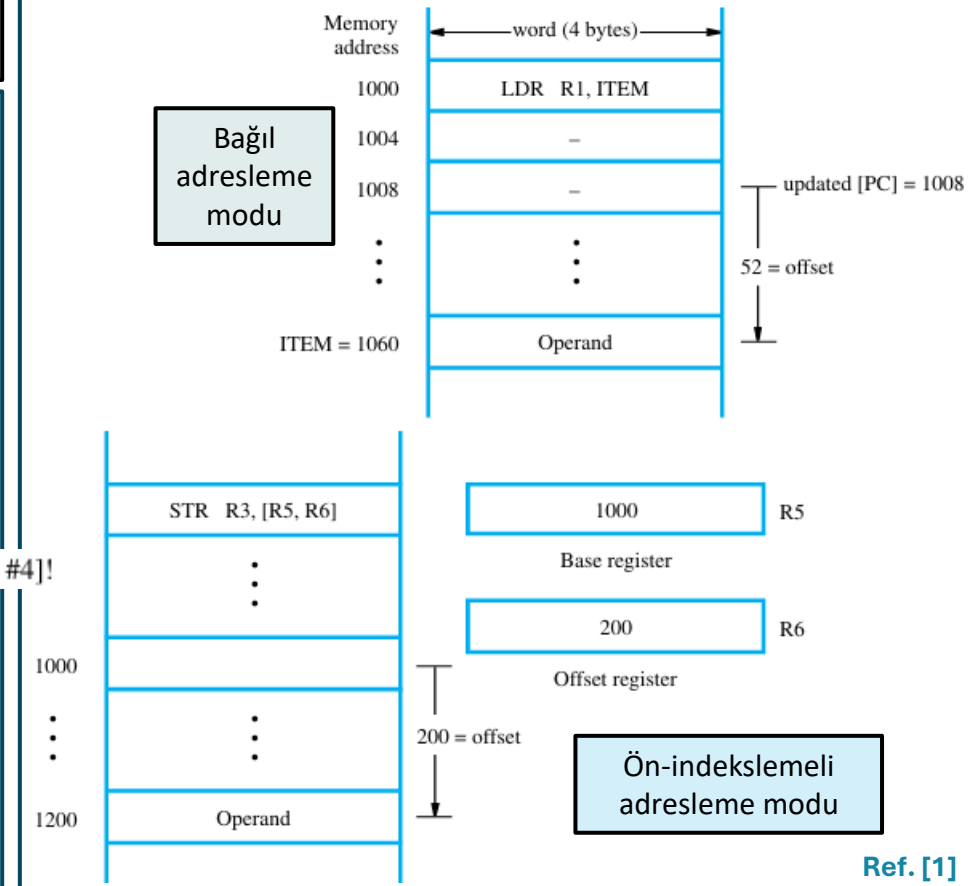
LDR R1, ITEM

LDR R0, [R1, −R2, LSL #4]!
R0 ← [[R1] −16 × [R2]]

Ref. [1]



Load ve Store komutlarının formatı



Ref. [1]

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ

BM303

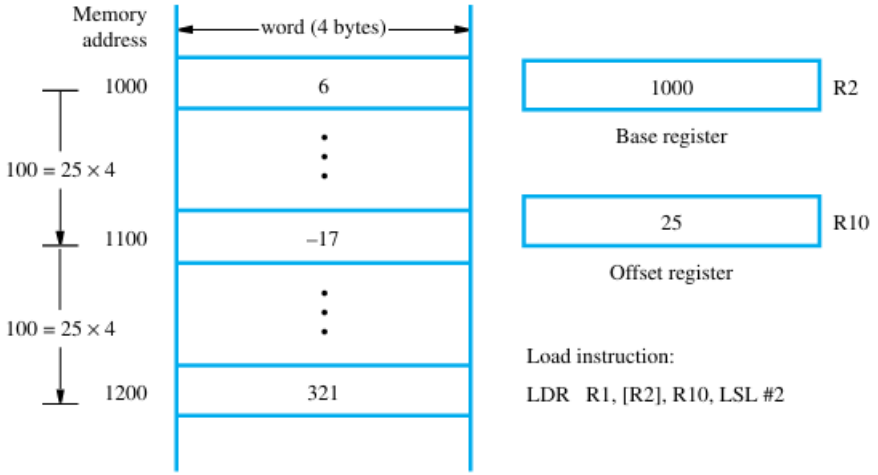
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU

Prof. Dr. A. Akbaş

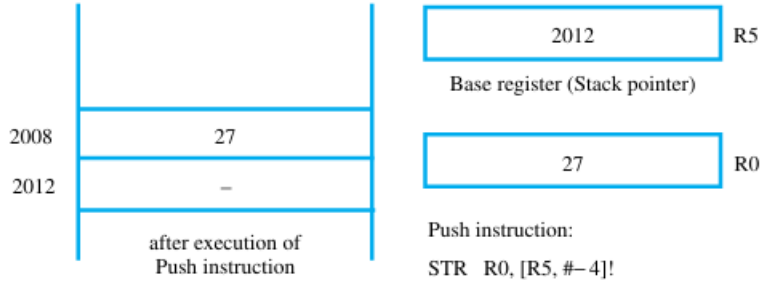
ENG. TERMINOLOGY & ACRONYMS

ARM İŞLEMCİ ADRESLEME MODLARI

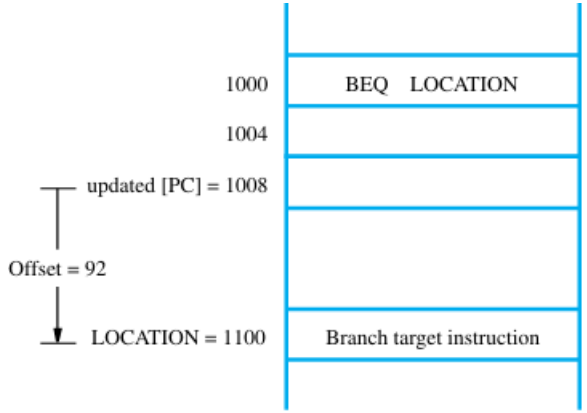
Ref. [1]



Indeksleme sonrası adresleme



Geri yazmalı önüindekslemeli adresleme



Bir dallanma komutu için hedef adresin belirlenmesi

ARM komutları için koşul kodları

Condition field $b_{31} \ldots b_{28}$	Condition suffix	Name	Condition code test
0 0 0 0	EQ	Equal (zero)	Z = 1
0 0 0 1	NE	Not equal (nonzero)	Z = 0
0 0 1 0	CS/HS	Carry set/Unsigned higher or same	C = 1
0 0 1 1	CC/LO	Carry clear/Unsigned lower	C = 0
0 1 0 0	MI	Minus (negative)	N = 1
0 1 0 1	PL	Plus (positive or zero)	N = 0
0 1 1 0	VS	Overflow	V = 1
0 1 1 1	VC	No overflow	V = 0
1 0 0 0	HI	Unsigned higher	$\overline{C} \vee Z = 0$
1 0 0 1	LS	Unsigned lower or same	$\overline{C} \vee Z = 1$
1 0 1 0	GE	Signed greater than or equal	$N \oplus V = 0$
1 0 1 1	LT	Signed less than	$N \oplus V = 1$
1 1 0 0	GT	Signed greater than	$Z \vee (N \oplus V) = 0$
1 1 0 1	LE	Signed less than or equal	$Z \vee (N \oplus V) = 1$
1 1 1 0	AL	Always	
1 1 1 1		not used	

Ref. [1]

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ
BM303
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU
Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

ARM İŞLEMCİ KOMUTLARI

ARM Komutları: **ARM** mimarisindeki her komut **32** bitlik bir kelimeye kodlanır. Belleğe erişim yalnızca **Load** ve **Store** komutları tarafından sağlanır. Tüm aritmetik ve mantık işlem komutları işlemci kaydedicilerinde çalışır.

OP kodu mnemotekleri LDR ve STR ile **bayt** ve **yarım sözcük** değerleri de bellek ve kaydediciler arasında aktarılabilir. Operant bir **bayt**sa, kaydedicinin düşük sıralı bayt konumunda bulunur. **Yarım sözcük**se, kaydedicinin düşük sıralı yarısında bulunur. **Load** komutları için, bayt ve yarım sözcük değerleri, **LDRB** ve **LDRH** komut mnemotekleri kullanılarak **32** bitlik kaydedici uzunluğuna sıfır uzatılır veya **LDRSB** ve **LDRSH** kullanılarak işaret uzatılır. Bayt ve yarım sözcük Store komutları, **STRB** ve **STRH**'dır.

Birden Fazla Operantı Yükleme ve Depolama: Birden fazla operantı yüklemek ve depolamak için iki komut vardır. Bunlara **Blok Aktarım komutları** denir. Genel amaçlı kaydeicilerin herhangi bir alt kümesi yüklenebilir veya depolanabilir. Yalnızca sözcük operantlarına izin verilir. Kullanılan **OP** kodları **LDM** (Çoklu Yükleme) ve **STM**'dir (Çoklu Depolama). Bellek operantları ardışık sözcük konumlarında olmalıdır.

Ön ve son indekslemenin tüm formları, geri yazma ile ve geri yazma olmadan kullanılabilir. Bu komutlarda ofset büyüklüğü her zaman **4**'tür, bu nedenle komutta açıkça belirtilmesi gerekmez. Kaydedici listesi, komutun makine dili gösteriminde artan sırada görünmelidir, ancak bitişik olmaları gerekmez. Kodlanmış makine komutları **b15–0** bitlerinde belirtilir, kaydedici **Ri** listedeyse bit **bi = 1** olur.

Örneğin, kayıt **R10**'un temel kaydedici olduğunu ve başlangıçta **1000** değerini içerdiğini varsayalım.

LDMIA R10!,{R0,R1, R6, R7}

komutları sözcükleri **1000**, **1004**, **1008** ve **1012** konumlarından **R0**, **R1**, **R6** ve **R7** kaydedicilerine aktarır ve son aktarımdan sonra **R10**'da **1016** adres değerini bırakır, çünkü geri yazma ünlem işaretiyle gösterilir. **OP** kodundaki **IA** eki, "**Sonra Artırma**"yı belirtir ve bu da son indekslemeye karşılık gelir.

Aritmetik İşlem Komutları

```
SUB   R0, R6, R5
R0 ← [R6] − [R5]
```

```
OP    Rd, Rn, Rm
```

```
ADD   R0, R2, R4        ADD   R0, R3, #17       ADD   R0, R3, #−17
R0 ← [R2] + [R4]        R0 ← [R3] + 17          SUB   R0, R3, #17
```

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ
BM303
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU
Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

ARM İŞLEMCİ KOMUTLARI

İkinci Kaynak Operantının Kaydırma veya Döndürülmesi: İkinci kaynak operant bir kaydedicinin içeriği olarak belirtildiğinde, işlemde kullanılmadan önce kaydırılabilir veya döndürülebilir. Mantıksal sola kaydırma (**LSL**), mantıksal sağa kaydırma (**LSR**), aritmetik sağa kaydırma (**ASR**) ve sağa döndürme (**ROR**) kullanılabilir. Taşıma biti **C**, bu işlemlerde yer almaz. Kaydırma veya döndürme, ikinci kaynak operant için kaydedici adından sonra belirtilir.

```
ADD     R0, R1, R5, LSL #4
```

R5 kaydedicisinde bulunan ikinci kaynak operantı, **4** bit konumu sola kaydırılır (**[R5] × 16**'ya eşdeğerdir), ardından **R1** kaydedicinin içeriğine eklenir. Toplam, **R0** kaydedicisine yerleştirilir. Kaydırma veya döndürme miktarı, dördüncü kaydın içeriği olarak da belirtilebilir.

Çok Sözcüklü Operantlar: Carry bayrağı, **C**, çok sözcüklü sayıları içeren toplama ve çıkarma işlemlerini kolaylaştırmak için kullanılabilir. Bu amaçla ayrı komutlar mevcuttur. Bunların derleme dili anımsatıcıları **ADC** (Carry ile ekle) ve **SBC**'dir (Carry ile çıkar). İki **64** bitlik operantın toplanacağını varsayalım. İlk operantın **R3**, **R2**; ikinci operantın **R5**, **R4** kaydedici çiftinde bulunuyorsa:

```
ADDS    R6, R2, R4
```

```
ADC     R7, R3, R5
```

Çarpma Komutları

```
MUL     R0, R1, R2          MLA     R0, R1, R2, R3
R0 ← [R1] × [R2]            R0 ← ([R1] × [R2]) + [R3]
```

Taçıma Komutları

```
MOV     Rd, Rm              MOV     R0, #−5
MOV     Rd, #value          MVN     R0, #4
```

Kaydırma ve Döndürme Komutları

```
MOV     Ri, Rj, LSL #4      LShiftL     Ri, Rj, #4
```

Mantıksal İşlem ve Test Komutları: **AND**, **OR**, **XOR** ve **Bit-Clear** mantık işlemleri sırasıyla **AND**, **ORR**, **EOR** ve **BIC OP** kodlarına sahip komutlarla uygulanır. Aritmetik işlem komutları ile aynı formattadırlar.

AND Rd, Rn, Rm

Komutu **Rn** ve **Rm** kaydeicilerindeki operantların bit bazında mantıksal **AND** işlemini gerçekleştirir ve sonucu **Rd** kaydedicisine yerleştirir. Örneğin, **R0 02FA62CA** onaltılık desenini ve **R1 0000FFFF** desenini içeriyorsa, **AND R0, R0, R1** komutu **000062CA** desenini **R0**'a yerleştirir. **Bit Clear** komutu, **BIC**, **AND** komutuyla yakından ilişkilidir. **Rm** operantındaki her biti, **Rn**'deki bitlerle **AND** işlemine tabi tutmadan önce tamamlar. Yukarıdaki örnekteki gibi aynı **R0** ve **R1** bit desenleri kullanılarak, **BIC R0, R0, R1** komutu **02FA0000** desenini **R0**'a yerleştirir.

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ

BM303

BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU

Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

ARM İŞLEMCİLİ PROGRAM ÖRNEKLERİ

İki 4-bitlik decimal sayıyı bir bayta paketleyen program

```
LDR     R0, =LOC        Load address LOC into R0.
LDRB    R1, [R0]        Load ASCII characters
LDRB    R2, [R0, #1]      into R1 and R2.
AND     R2, R2, #&F     Clear high-order 28 bits of R2.
ORR     R2, R2, R1, LSL #4  Shift contents of R1 left,
                              perform logical OR with
                              contents of R2, and place
                              result into R2.
STRB    R2, PACKED      Store packed BCD digits
                              into PACKED.
```

Ref. [1]

Sayıları toplayan bir program

```
        LDR     R1, N       Load count into R1.
        LDR     R2, =NUM1   Load address NUM1 into R2.
        MOV     R0, #0      Clear accumulator R0.
LOOP    LDR     R3, [R2], #4  Load next number into R3.
        ADD     R0, R0, R3  Add number into R0.
        SUBS    R1, R1, #1  Decrement loop counter R1.
        BGT     LOOP        Branch back if not done.
        STR     R0, SUM     Store sum.
```

Ref. [1]

Test skorlarını toplayan bir program ve onun ARM versiyonu

```
        LDR     R2, =N      Load address N into R2.
        MOV     R3, #0
        MOV     R4, #0
        MOV     R5, #0
        LDR     R6, N       Load the value n.
LOOP    LDR     R7, [R2, #8]!  Add current student mark
        ADD     R3, R3, R7      for Test 1 to partial sum.
        LDR     R7, [R2, #4]!  Add current student mark
        ADD     R4, R4, R7      for Test 2 to partial sum.
        LDR     R7, [R2, #4]!  Add current student mark
        ADD     R5, R5, R7      for Test 3 to partial sum.
        SUBS    R6, R6, #1  Decrement the counter.
        BGT     LOOP        Loop back if not finished.
        STR     R3, SUM1    Store the total for Test 1.
        STR     R4, SUM2    Store the total for Test 2.
        STR     R5, SUM3    Store the total for Test 3.
```

```
        Move            R2, #LIST   Get the address LIST.
        Clear           R3
        Clear           R4
        Clear           R5
        Load            R6, N       Load the value n.
LOOP:   Load            R7, 4(R2)   Add the mark for next student's
        Add             R3, R3, R7    Test 1 to the partial sum.
        Load            R7, 8(R2)   Add the mark for that student's
        Add             R4, R4, R7    Test 2 to the partial sum.
        Load            R7, 12(R2)  Add the mark for that student's
        Add             R5, R5, R7    Test 3 to the partial sum.
        Add             R2, R2, #16 Increment the pointer.
        Subtract        R6, R6, #1  Decrement the counter.
        Branch_if_[R6]>0  LOOP      Branch back if not finished.
        Store           R3, SUM1    Store the total for Test 1.
        Store           R4, SUM2    Store the total for Test 2.
        Store           R5, SUM3    Store the total for Test 3.
```

Ref. [1]

ARM İŞLEMCİLİ PROGRAM ÖRNEKLERİ

```
        LDR     R1, N           Load count into R1.
        LDR     R2, =NUM1       Load address NUM1 into R2.
        MOV     R0, #0          Clear accumulator R0.
LOOP    LDR     R3, [R2], #4    Load next number into R3.
        ADD     R0, R0, R3      Add number into R0.
        SUBS    R1, R1, #1      Decrement loop counter R1.
        BGT     LOOP            Branch back if not done.
        STR     R0, SUM         Store sum.
```

Sayıları toplayan programın alt program çağrılı hali

Calling program

```
        LDR     R1, N
        LDR     R2, =NUM1
        BL      LISTADD
        STR     R0, SUM
        ⋮
```

Subroutine

```
LISTADD STMFD   R13!, {R3, R14}   Save R3 and return address in R14 on
                                    stack, using R13 as the stack pointer.
        MOV     R0, #0
LOOP    LDR     R3, [R2], #4
        ADD     R0, R0, R3
        SUBS    R1, R1, #1
        BGT     LOOP
        LDMFD   R13!, {R3, R15}   Restore R3 and load return address
                                    into PC (R15).
```

Ref. [1]

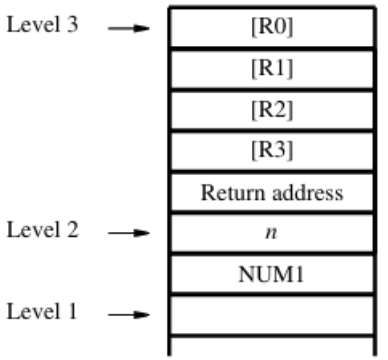
(Assume top of stack is at level 1 below.)

Calling program

```
        LDR     R0, =NUM1       Push NUM1
        STR     R0, [R13, #–4]!   on stack.
        LDR     R0, N           Push n
        STR     R0, [R13, #–4]!   on stack.
        BL      LISTADD
        LDR     R0, [R13, #4]   Move the sum into
        STR     R0, SUM           memory location SUM.
        ADD     R13, R13, #8    Remove parameters from stack.
        ⋮
```

Subroutine

```
LISTADD STMFD   R13!, {R0–R3, R14}   Save registers.
        LDR     R1, [R13, #20]       Load parameters
        LDR     R2, [R13, #24]         from stack.
        MOV     R0, #0
LOOP    LDR     R3, [R2], #4
        ADD     R0, R0, R3
        SUBS    R1, R1, #1
        BGT     LOOP
        STR     R0, [R13, #24]       Place sum on stack.
        LDMFD   R13!, {R0–R3, R15}   Restore registers and return.
```



Ayni programın parametreleri yığına aktarılan bir alt rutinle gerçeklenen şekli

Ref. [1]

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ

BM303

BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU

Prof. Dr. A. Akbaş

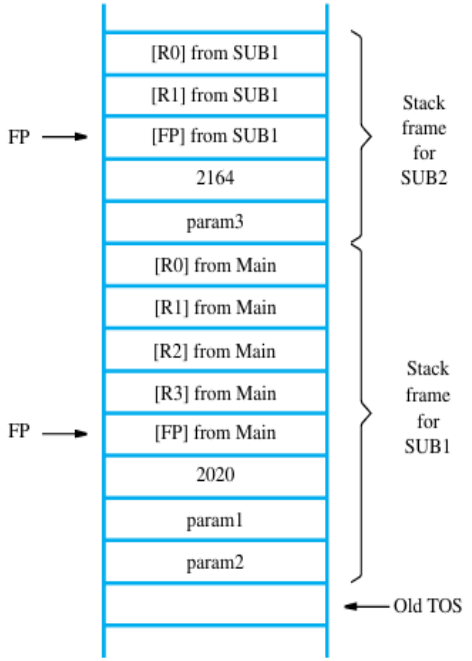
ENG. TERMINOLOGY & ACRONYMS

ARM İŞLEMCİLİ PROGRAM ÖRNEKLERİ

İç-içe alt rutinler

Memory location		Instructions		Comments
Main program				
		⋮		
2000		LDR	R10, PARAM2	Place parameters on stack.
2004		STR	R10, [SP, #–4]!	
2008		LDR	R10, PARAM1	
2012		STR	R10, [SP, #–4]!	
2016		BL	SUB1	
2020		LDR	R10, [SP]	Store SUB1 result.
2024		STR	R10, RESULT	
2028		ADD	SP, SP, #8	Remove parameters from stack.
2032		next instruction		
		⋮		
First subroutine				
2100	SUB1	STMFD	SP!, {R0–R3, FP, LR}	Save registers.
2104		ADD	FP, SP, #16	Load frame pointer.
2108		LDR	R0, [FP, #8]	Load parameters.
2112		LDR	R1, [FP, #12]	
		⋮		
		LDR	R2, PARAM3	Place parameter on stack.
		STR	R2, [SP, #–4]!	
2160		BL	SUB2	
2164		LDR	R2, [SP], #4	Pop SUB2 result into R2.
		⋮		
		STR	R3, [FP, #8]	Place result on stack.
		LDMFD	SP!, {R0–R3, FP, PC}	Restore registers and return.
Second subroutine				
3000	SUB2	STMFD	SP!, {R0, R1, FP, LR}	Save registers.
		ADD	FP, SP, #8	Load frame pointer.
		LDR	R0, [FP, #8]	Load parameter.
		⋮		
		STR	R1, [FP, #8]	Place result on stack.
		LDMFD	SP!, {R0, R1, FP, PC}	Restore registers and return.

Ref. [1]



İç-içe rutinler için yığın çerçeveleri

	Memory address label	Operation	Addressing or data information
Assembler directives		AREA	CODE
		ENTRY	
Statements that generate machine instructions		LDR	R1, N
		LDR	R2, POINTER
		MOV	R0, #0
	LOOP	LDR	R3, [R2], #4
		ADD	R0, R0, R3
		SUBS	R1, R1, #1
		BGT	LOOP
		STR	R0, SUM
Assembler directives		AREA	DATA
	SUM	DCD	0
	N	DCD	5
	POINTER	DCD	NUM1
	NUM1	DCD	3, –17, 27, –12, 322
		END	

Sembolik Makine Dili (Assembly) Programı

Ref. [1]

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ

BM303

BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU

Prof. Dr. A. Akbaş

ENG. TERMINOLOGY & ACRONYMS

ARM İŞLEMCİLİ PROGRAM ÖRNEKLERİ

İki vektörü skaler çarpan program

```
        LDR   R1, =AVEC     R1 points to vector A.
        LDR   R2, =BVEC     R2 points to vector B.
        LDR   R3, N         R3 is the loop counter.
        MOV   R0, #0        R0 accumulates the dot product.
LOOP    LDR   R4, [R1], #4  Load A component.
        LDR   R5, [R2], #4  Load B component.
        MLA   R0, R4, R5, R0 Multiply components and
                              accumulate into R0.
        SUBS  R3, R3, #1    Decrement the counter.
        BGT   LOOP          Branch back if not done.
        STR   R0, DOTPROD   Store dot product.
```

Karakter Dizisi (string) Arama Programı

```
        LDR   R2, =T        Load address T into R2.
        LDR   R3, =P        Load address P into R3.
        LDR   R4, N         Get the value n.
        LDR   R5, M         Get the value m.
        SUB   R4, R4, R5    Compute n – m.
        ADD   R4, R2, R4    R4 is the address of T(n – m).
        ADD   R5, R3, R5    R5 is the address of P(m).
LOOP1   MOV   R6, R2        Use R6 to scan through string T.
        MOV   R7, R3        Use R7 to scan through string P.
LOOP2   LDRB  R8, [R6], #1  Compare a pair of
        LDRB  R9, [R7], #1   characters in
        CMP   R8, R9         strings T and P.
        BNE   NOMATCH
        CMP   R5, R7        Check if at P(m).
        BGT   LOOP2         Loop again if not done.
        STR   R2, RESULT    Store the address of T(i).
        B     DONE
NOMATCH ADD   R2, R2, #1    Point to next character in T.
        CMP   R4, R2        Check if at T(n – m).
        BGE   LOOP1         Loop again if not done.
        MOV   R8, #–1       No match was found.
        STR   R8, RESULT
DONE    next instruction
```

Ref. [1]

Bir karakter satırını okuyan ve görüntüleyen program

```
READ    LDRB  R3, [R1, #4]  Load KBD_STATUS byte and
        TST   R3, #2          wait for character.
        BEQ   READ
        LDRB  R3, [R1]      Read the character and
        STRB  R3, [R0], #1    store it in memory.
ECHO    LDRB  R4, [R2, #4]  Load DISP_STATUS byte and
        TST   R4, #4          wait for display
        BEQ   ECHO            to be ready.
        STRB  R3, [R2]      Send
        TEQ   R3, #CR       If no
        BNE   READ          rea
```

Klavyeden girilen bir satırı kesme ile okuyan ve yoklama yöntemiyle çıkışa aktaran program

Interrupt-service routine

```
IRQLOC  STMFD R13!, {R2, R3}  Save R2 and R3 on the stack.
        LDR   R2, PNTR        Load address pointer.
        LDRB  R3, KBD_DATA    Read character from keyboard.
        STRB  R3, [R2], #1    Write character into memory
                                and increment pointer.
        STR   R2, PNTR        Update pointer in memory.
ECHO    LDRB  R2, DISP_STATUS Wait for display to be ready.
        TST   R2, #4
        BEQ   ECHO
        STRB  R3, DISP_DATA   Send character to display.
        CMP   R3, #CR         Check if character is Carriage Return
        BNE   RTRN              and return if not CR.
        MOV   R2, #1          If CR, indicate end of line.
        STR   R2, EOL
        MOV   R2, #0          Disable interrupts in
        STRB  R2, KBD_CONT      keyboard interface.
RTRN    LDMFD R13!, {R2, R3}  Restore registers
        SUBS  R15, R14, #4      and return from interrupt.
```

Main program

```
        LDR   R2, =LINE       Initialize buffer pointer.
        STR   R2, PNTR
        MOV   R2, #0          Clear end-of-line indicator.
        STR   R2, EOL
        MOV   R2, #2          Enable interrupts in keyboard interface.
        STRB  R2, KBD_CONT
        MOV   R2, #&50        Enable IRQ interrupts
        MSR   CPSR, R2          and switch to User mode.
        next instruction
```

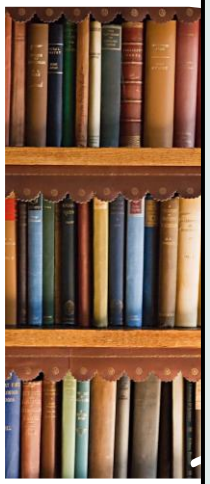
Ref. [1]

BİLECİK ŞEYH EDEBALİ ÜNİVERSİTESİ

2025-26 GYY DERSİ
BM303
BİLGİSAYAR MİMARİSİ ve ORGANİZASYONU
Prof. Dr. A. Akbaş

Kaynaklar

REFERANSLAR



1- Hamacher, C., Vranesic, Z., Zaky, S. and Manjikian, N. (2012). *Computer Organization and Embedded Systems.* McGraw Hill, New York. ISBN 978–0–07–338065–0

2- Bryant, R.E. and O'Hallaron, D.R. (2016). *Computer Systems, A Programmer's Perspective.* Pearson Education Limited, Malaysia. ISBN 10: 1-292-10176-8

3- Mono, M.M. (2001). *Bilgisayar Sistem Mimarisi*. Literatür Yayınları, İstanbul. ISBN 975-843-31-5

4- Stallings, W. (2005). *Computer Organisation and Architecture, Designing for Performance.* Pearson, Prentice Hall, NJ. ISBN 0-13-607373-5

5- Abd-El-Barr, M. and El-Rewini, H. (2005). *Fundamentals of Computer Organization and Architecture.* Wiley-Interscience, Hoboken-New Jersy. ISBN 0-471-46741-3

6- Reynolds, C. and Tymann, P. (2008). *Principles of Computer Science.* McGraw-Hill Schaum's Outline Series, New York. 0-07-151037-0

7- https://www.wikipedia.org v.b internet siteleri

14.ncü HAFTA
DERSİNİN SONU