



BLM4021 Gömülü Sistemler

Doç. Dr. Ali Can KARACA

ackaraca@yildiz.edu.tr

Yıldız Teknik Üniversitesi – Bilgisayar Mühendisliği

Sunum 6 – ARM Komut Setleri ve Raspberry Pi Kitleri

- ARM komut setleri (devam)
- Visual ARM üzerinde örnekler
- Raspberry Pi Versiyon ve Özellikleri
- Genel amaçlı giriş çıkış (GPIO) birimleri

Gerekli Kaynaklar:

- Derek Molloy, Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux, Wiley, 2016.
- M. Wolf, Computers as Components: Principles of Embedded Computing System Design, Elsevier, 2008.

Yardımcı Kaynaklar:

- P. Membrey, D. Hows, Learn Raspberry Pi 2 with Linux and Windows 10, Apress, 2015.
- Ali Saidi, The ARM architecture slide.
- Farid Farahmand, Chapter- 3, Embedded Systems with ARM Cortex-M, 2018.
- O. Urhan, Gömülü Sistem Lisansüstü Ders Notları, 2018.
- V. Weaver, ECE 471 – Embedded Systems Lecture 3, 2020.

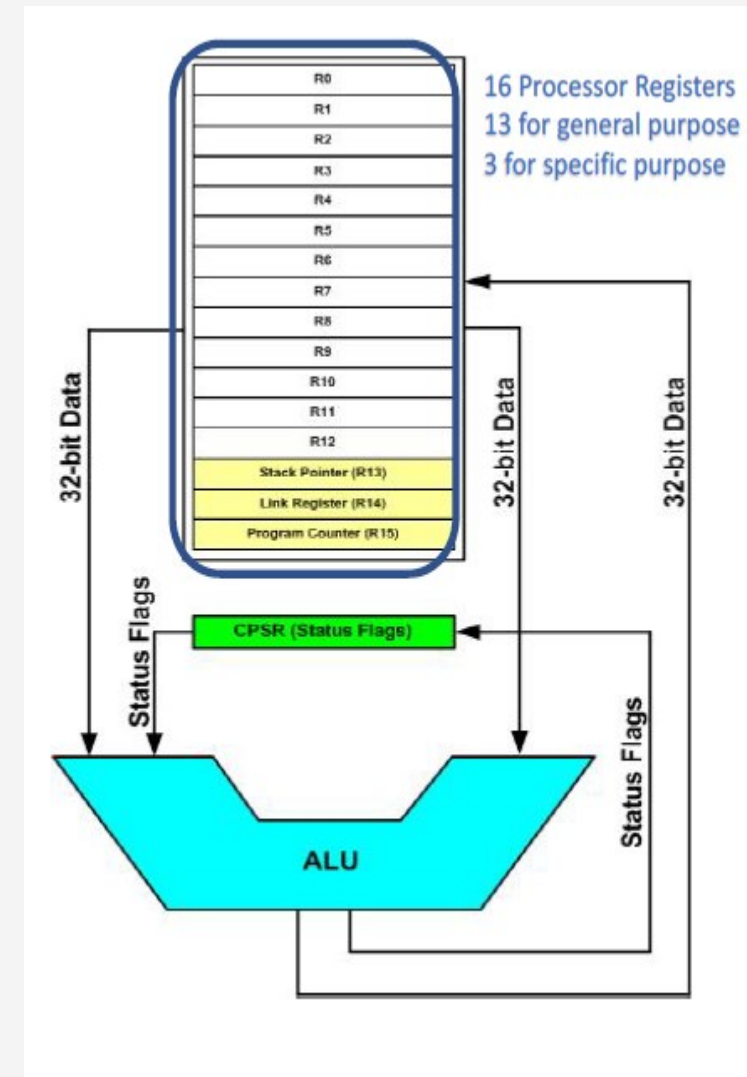
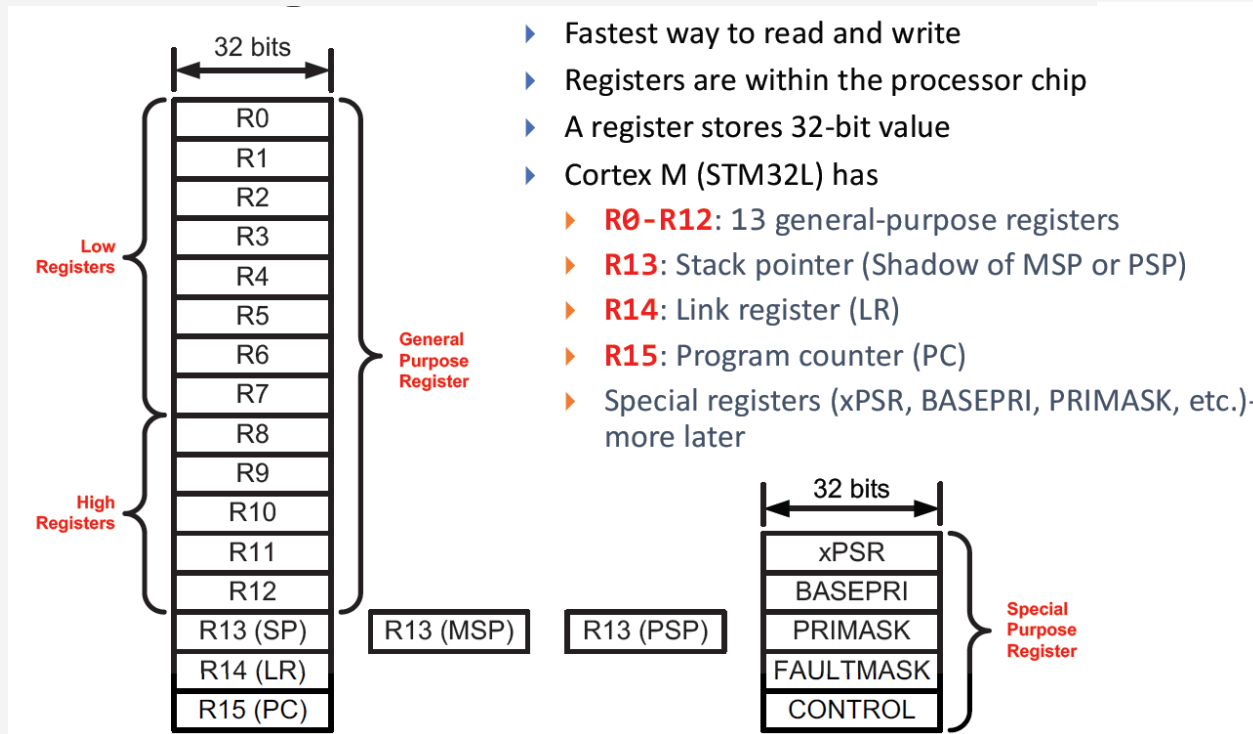
Haftalık Konular



Hafta	Teorik	Laboratuvar
1	Giriş ve Uygulamalar, Mikroişlemci, Mikrodenetleyici ve Gömülü sistem kavramlarının açıklanması	Grupların oluşturulması & Kitlerin Testi
2	Bir Tasarım Örneği, Mikroişlemci, Mikrodenetleyici, DSP, FPGA, ASIC kavramları	Kitlerin gruplara dağıtımı + Raspberry Pi Kurulumu
3	16, 32 ve 64 bitlik mikrodenetleyiciler, pipeline, PIC ve MSP430 özellikleri	Raspberry Pi ile Temel Konfigürasyon
4	ARM tabanlı mikrodenetleyiciler ve özellikleri	---- Resmi Tatil ---
5	ARM Komut setleri ve Assembly Kodları-1	Uygulama 1 – Raspberry Pi ile Buzzer Uygulaması
6	ARM Komut setleri ve Assembly Kodları-2, Raspberry Pi vers. ve GPIO'ları	Uygulama 2 – Raspberry Pi ile Ivme ve Gyro Uygulaması
7	Veri toplama; algılayıcı, örnekleme teoremi, ADC, DAC	Uygulama 3 – Raspberry Pi ile Motor Kontrol Uygulaması
8	Vize Sınavı	
9	Çoklu ortam algılayıcıları ve arayüzleri (SPIE, I2C...)	Proje Soru-Cevap Saati - 1
10	Zamanlayıcı ve kesmeler	Proje Soru-Cevap Saati -2
11	Görüntü, Ses ve Video	Proje kontrolü-1
12	Gerçek Zaman Sistemlerinde temel kavramlar	Proje Kontrolü-2
13	Gerçek zaman İşletim Sistemleri	Mazeret sebepli son proje kontrollerinin yapılması
14	Nesnelerin İnterneti	
15	Final Sınavı	

For more details -> Bologna page: <http://www.bologna.yildiz.edu.tr/index.php?r=course/view&id=9463&aid=3>

ARM Registers

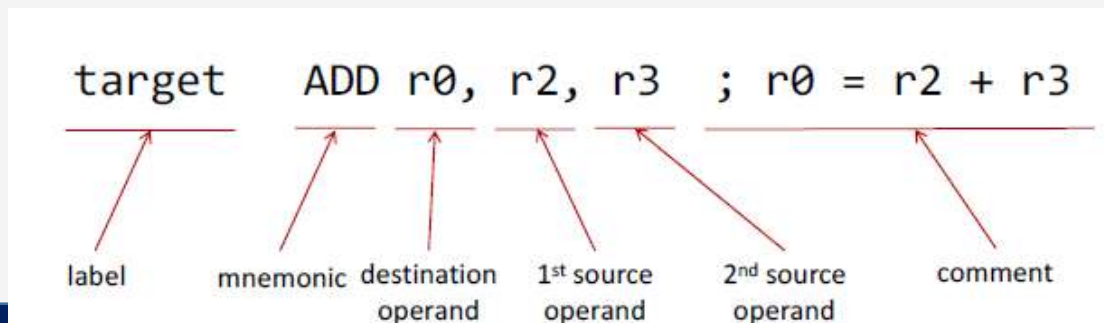


Credit by Farih Farahmand

Instruction Syntax in ARM

label	mnemonic operand1, operand2, operand3	; comments
-------	---------------------------------------	------------

- ▶ Label is a reference to the memory address of this instruction.
- ▶ Mnemonic represents the operation to be performed (ADD, SUB, etc.).
- ▶ The number of **operands** varies, depending on each specific instruction. Some instructions have no operands at all.
 - ▶ Typically, operand1 is the **destination** register, and operand2 and operand3 are source operands.
 - ▶ operand2 is usually a register.
 - ▶ operand3 may be a register, an immediate number, a register shifted to a constant amount of bits, or a register plus an offset (used for memory access).
- ▶ Everything after the semicolon “;” is a comment, which is an annotation explicitly declaring programmers’ intentions or assumptions.



Instructions: Branch Operations



- Branch instruction

```
B label
```

```
...
```

```
label:  ...
```

- Conditional branches

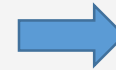
```
MOV R0, #0
```

```
loop:  ...
```

```
ADD R0, R0, #1
```

```
CMP R0, #10
```

```
BNE loop
```



Compare	Signed	Unsigned
==	EQ	EQ
≠	NE	NE
>	GT	HI
≥	GE	HS
<	LT	LO
≤	LE	LS



Compare	Signed	Unsigned
==	BEQ	BEQ
!=	BNE	BNE
>	BGT	BHI
≥	BGE	BHS
<	BLT	BLO
≤	BLE	BLS

Credit by Yung-Yu Chuang

Instructions: Branch with Link Operations



1. Example of using 'B' instruction:

```
CMP    r0,#0        ; check if r0 == 0
BNE    r2inc        ; if r0 !=0 branch to 'r2inc'
ADD    r1,r1,#1      ; r1 += 1
B      next         ; unconditional branch to 'next'
r2inc  ADD    r2,r2,#1 ; r2 += 1
next   ----         ; continue
```

2. Example of using 'BL' instruction

```
BL      funct1       ; save return addr. & subroutine
CMP     r0,#5        ; next instruction
----
func1  ADD    r0,r0,#1 ; subroutine
----
MOV     pc,lr        ; return to program
```

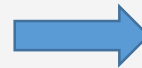
```
MOV r1, #10
MOV r2, #15
BL subLoop
CMP r1, #10
MOVEQ r1, #100
```

```
subLoop MOV pc, lr
```


Example for a subroutine



```
1 int main(int argc, char **argv)
2 {
3
4     int y;
5     y = diffofsums(2, 3, 4, 5);
6 }
7
8 int diffofsums(int f, int g, int h, int i)
9 {
10
11     int result;
12     result = (f + g) - (h + i);
13     return result;
14 }
```



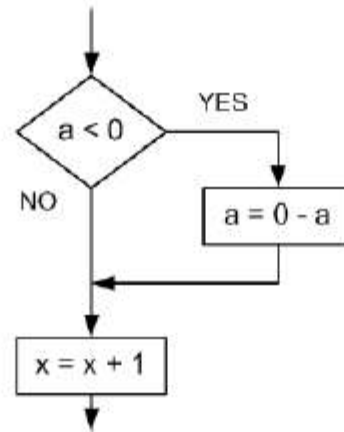
```
1 MAIN
2     MOV     R0, #0X2
3     MOV     R1, #0X3
4     MOV     R2, #0X4
5     MOV     R3, #0X5
6     BL      DIFFOFSUMS
7     MOV     R4, R0
8     END
9 DIFFOFSUMS
10    ADD     R8, R0, R1
11    ADD     R9, R2, R3
12    SUB     R4, R8, R9
13    MOV     R0, R4
14    MOV     PC, LR
```

IF – THEN statement



If-then Statement

```
C Program
if (a < 0) {
    a = 0 - a;
}
x = x + 1;
```



```
;CONDITIONAL BRANCH
MOV R1, #-5
CMP R1, #0
BGE MENDIF ;BRANCH IF R1 > 0 (SIGNED)
RSB R1, R1, #0
MENDIF ADD R2, R2, #1
```

Implementation 1:

```
        ; r1 = a, r2 = x
        CMP r1, #0          ; Compare a with 0
        BGE endif          ; Go to endif if a ≥ 0
then     RSB r1, r1, #0      ; a = - a
endif    ADD r2, r2, #1      ; x = x + 1
```

Credit by Farid Farahmand

Compound Boolean Expression



```
x > 20 && x < 25
x == 20 || x == 25
!(x == 20 || x == 25)
```

```
60 ;IF-THEN WITH COMPOUND LOGICAL OR
61 ; IF (R0 <= 20 || R0 >= 25) --> R1=1
62 MOV R0, #-2
63 CMP R0, #20
64 BLE S_THEN
65 CMP R0, #25
66 BLT S_ENDIF
67 S_THEN MOV R1, #1
68 S_ENDIF
```

Veya:

C Program	Assembly Program
// x is a signed integer if(x <= 20 x >= 25){ a = 1 }	; r0 = x CMP r0, #20 ; compare x and 20 BLE then ; go to then if x ≤ 20 CMP r0, #25 ; compare x and 25 BLT endif ; go to endif if x < 25 then MOV r1, #1 ; a = 1 endif

```
70 ;SAME AS ABOVE BUT SIMPLER
71 ;IF-THEN WITH COMPOUND LOGICAL OR
72 ; IF (R0 <= 20 || R0 >= 25) --> R1=1
73 MOV R1, #0
74 MOV R0, #19
75 CMP R0, #20
76 MOVLE R1, #1
77 CMP R0, #25
78 MOVGE R1, #1
```

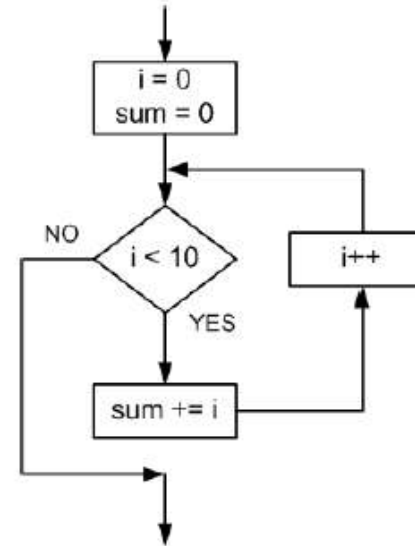
For Loop



For Loop

C Program

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



One Implementation is

```
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
```

For Loop-2



```
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

Using Stack Pointer & LDR/STR

Stack Pointer -> Register 13

```
1  int main(int argc, char **argv)
2  {
3
4      int y;
5      y = diffofsums(2, 3, 4, 5);
6  }
7
8  int diffofsums(int f, int g, int h, int i)
9  {
10
11     int result;
12     result = (f + g) - (h + i);
13     return result;
14 }
```



- First, store all used registers to stack,
- Second, use it for the equations
- Last, 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
```



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
```

Before entering
subroutine:

Registers	Memory	Symbols
R0	2	
R1	3	
R2	5	
R3	2	
R4	0	
R5	0	
R6	0	
R7	0	
R8	0	
R9	0	
R10	0	
R11	0	
R12	0	
R13	-16777216	
R14	0	
R15	16	

Register values
in subroutine: line 23

Registers	Memory	Symbols
R0	-2	
R1	3	
R2	5	
R3	2	
R4	-2	
R5	0	
R6	0	
R7	0	
R8	5	
R9	7	
R10	0	
R11	0	
R12	0	
R13	-16777228	
R14	20	
R15	56	

After
subroutine:

Registers	Memory	Symbols
R0	-2	
R1	3	
R2	5	
R3	2	
R4	0	
R5	0	
R6	0	
R7	0	
R8	0	
R9	0	
R10	0	
R11	0	
R12	0	
R13	-16777216	
R14	20	
R15	20	

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



Registers	Memory	Symbols
Enable Byte View		
Enable Reverse Direction		
Symbol	Address	Value
SP →	0xFEFFFFFF4	0
	0xFEFFFFFF8	0
(DIFFOFSUMS)	0xFEFFFFFFC	0
Uninitialized memory is zeroed		



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

That's an important example about how to use STR and LDR instructions?

What about LDM ? STM?

Instructions: Multiple Load/Store Registers



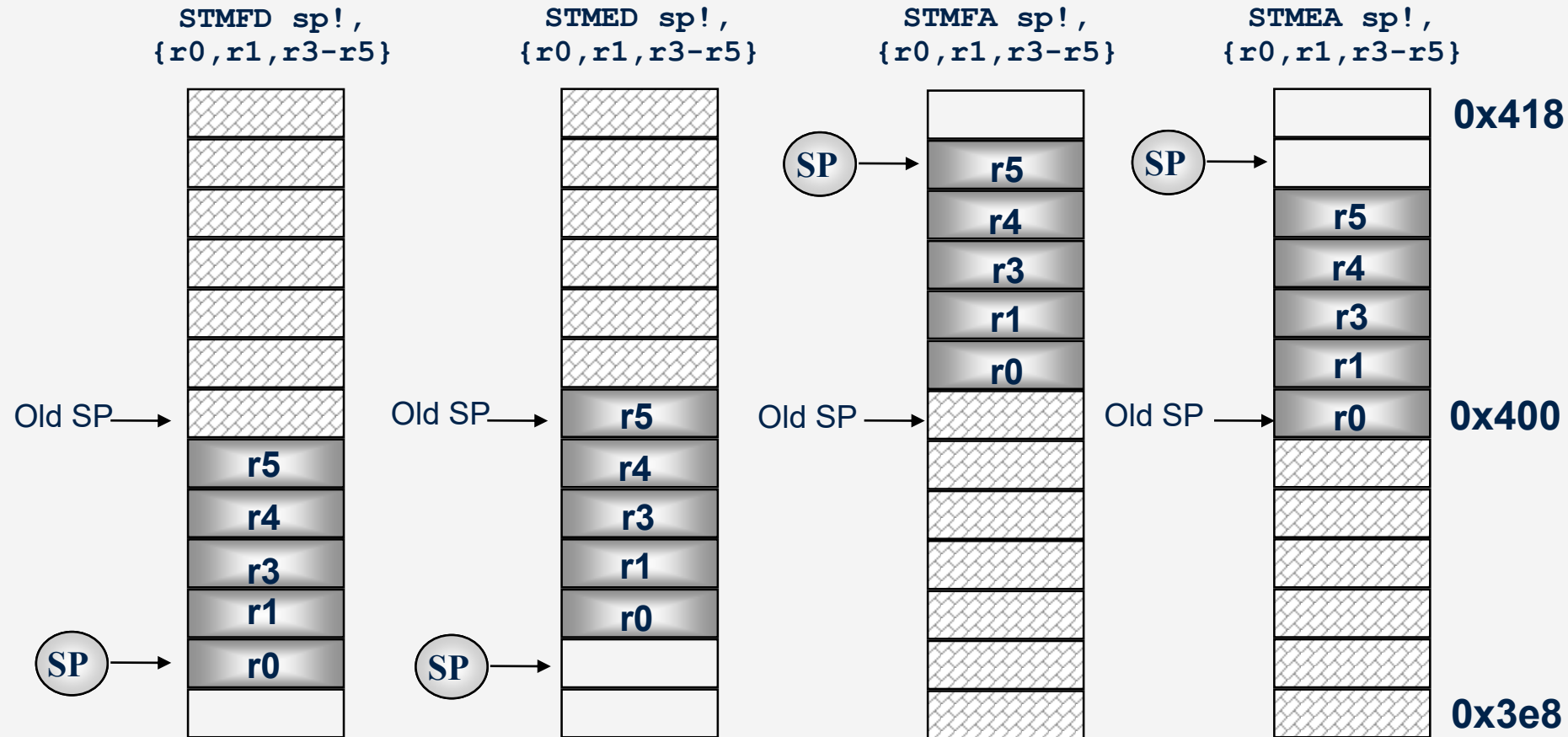
LDM load multiple registers
STM store multiple registers

suffix	meaning
IA	increase after
IB	increase before
DA	decrease after
DB	decrease before

Syntax: `<LDM|STM>{<cond>}<addressing mode> Rn{!},<registers>{^}`

Addressing mode	Description	Start address	End address	$Rn!$
IA	increment after	Rn	$Rn + 4 * N - 4$	$Rn + 4 * N$
IB	increment before	$Rn + 4$	$Rn + 4 * N$	$Rn + 4 * N$
DA	decrement after	$Rn - 4 * N + 4$	Rn	$Rn - 4 * N$
DB	decrement before	$Rn - 4 * N$	$Rn - 4$	$Rn - 4 * N$

Stack Examples



- STMIA, STMIB, STMDA, STMDB are the same instructions as STMEA, STMFA, STMED, STMFD, respectively

Using Stack Pointer & LDM/STM



Stack Pointer -> Register 13

```
1  int main(int argc, char **argv)
2  {
3
4      int y;
5      y = diffofsums(2, 3, 4, 5);
6  }
7
8  int diffofsums(int f, int g, int h, int i)
9  {
10
11     int result;
12     result = (f + g) - (h + i);
13     return result;
14 }
```



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
```

What is Raspberry Pi?



- The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries.
- Over 5 million Raspberry Pis have been sold before February 2015, making it the best-selling British computer. By November 2016 they had sold 11 million units.
- Till now, Raspberry has lots of model versions. (Zero, 1-4)

The Timeline

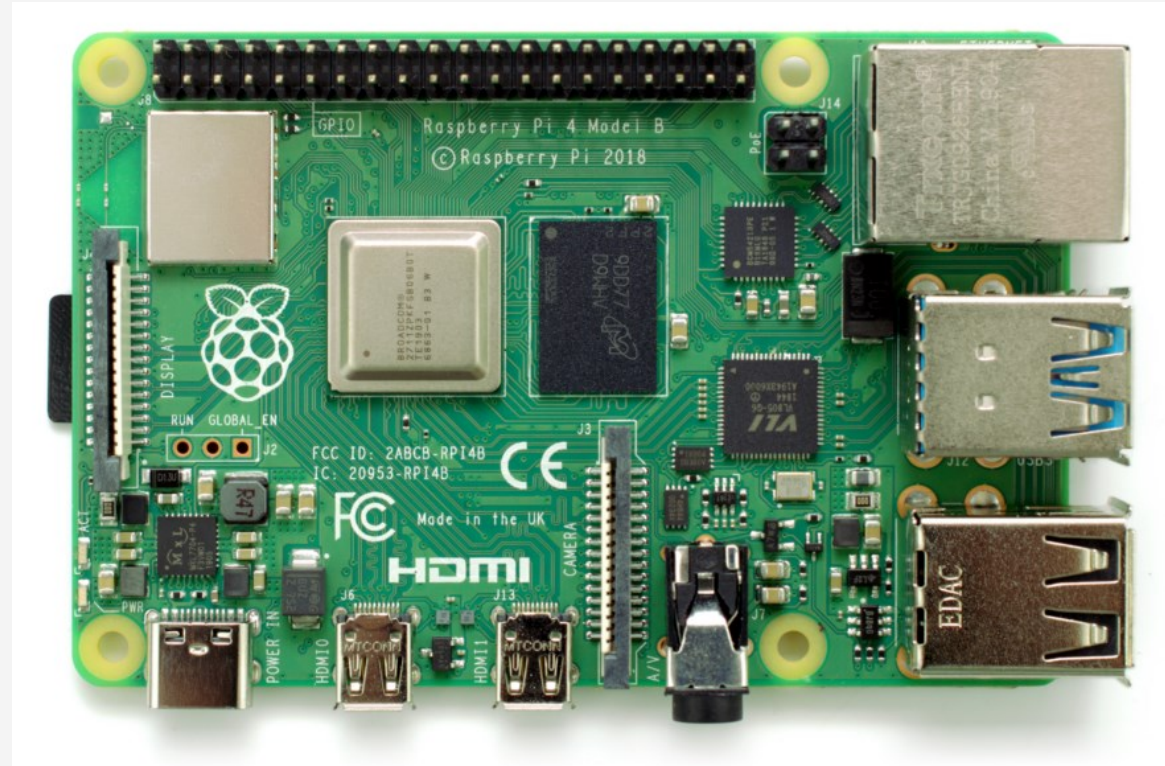
- The first generation ([Raspberry Pi 1 Model B](#)) was released in February 2012. It was followed by a simpler and inexpensive model Model A.
- In 2014, the foundation released a board with an improved design in [Raspberry Pi 1 Model B+](#). These boards are approximately credit-card sized and represent the standard mainline form-factor.
- Improved A+ and B+ models were released a year later. A "compute module" was released in April 2014 for embedded applications, and a [Raspberry Pi Zero](#) with smaller size and reduced input/output (I/O) and general-purpose input/output (GPIO) capabilities was released in November 2015 for US\$5.

The Timeline

- The [Raspberry Pi 2](#) which added more RAM was released in February 2015.
- [Raspberry Pi 3 Model B](#) released in February 2016, is bundled with on-board WiFi, Bluetooth and USB boot capabilities.
- As of January 2017, [Raspberry Pi 3 Model B](#) is the newest mainline Raspberry Pi.
- Raspberry Pi boards are priced between US\$5–35.
- As of 28 February 2017, the [Raspberry Pi Zero W](#) was launched, which is identical to the Raspberry Pi Zero, but has the Wi-Fi and Bluetooth functionality of the Raspberry Pi 3 for US\$10.

The Timeline

- Now, there are also Raspberry Pi 4 model.
- Raspberry Pi 4 was released in June 2019. 1.5 MHz, 64-bit quad core ARM Cortex-A72
- 4K resolution and USB Type-C.
- Price: ~1000 TL

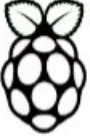






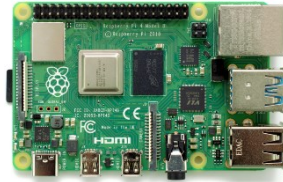


- The Foundation provides [Raspbian](#), a Debian-based Linux distribution for download, as well as third party Ubuntu, Windows 10 IOT Core, RISC OS, and specialised media center distributions.
- It promotes Python and Scratch as the main programming language, with support for many other languages.
- The default firmware is closed source, while an unofficial open source is available.

- The [Broadcom BCM2835](#) SoC used in the first generation Raspberry Pi is somewhat equivalent to the chip used in first modern generation smartphones (its CPU is an older ARMv6 architecture), which includes a 700 MHz ARM1176JZF-S processor, VideoCore IV graphics processing unit (GPU), and RAM.
- It has a level 1 (L1) cache of 16 KB and a level 2 (L2) cache of 128 KB. The level 2 cache is used primarily by the GPU. The SoC is stacked underneath the RAM chip, so only its edge is visible.
- The Raspberry Pi 2 uses a [Broadcom BCM2836](#) SoC with a 900 MHz 32-bit quad-core ARM Cortex-A7 processor, with 256 KB shared L2 cache.
- The Raspberry Pi 3 uses a [Broadcom BCM2837](#) SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, with 512 KB shared L2 cache

- The Raspberry Pi 3, with a quad-core Cortex-A53 processor, is described as **10 times** the performance of a Raspberry Pi 1. This was suggested to be highly dependent upon task threading and instruction set use. Benchmarks showed the Raspberry Pi 3 to be approximately **80%** faster than the Raspberry Pi 2 in parallelized tasks.
- Raspberry Pi 2 includes a quad-core Cortex-A7 CPU running at 900 MHz and 1 GB RAM. It is described as **4–6 times** more powerful than its predecessor. The GPU is identical to the original. In parallelized benchmarks, the Raspberry Pi 2 could be up to **14 times** faster than a Raspberry Pi 1 Model B+.

All Versions

							
Raspberry Pi	Modelo A	Modelo A+	Modelo B	Modelo B+	RPi V2 modelo B	RPi 3 modelo B	Rpi 4
SoC	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2835	Broadcom BCM2836	Broadcom BCM2837	BroadCom BCM2711
CPU	700MHz ARM1176JFZ-S	700MHz ARM1176JFZ-S	700MHz ARM1176JFZ-S	700MHz ARM1176JFZ-S	900MHz Quad-core ARM Cortex-A7	1.2Ghz Quad Cortex A53	1.5 GHz Quad Cortex
GPU	VideoCore IV	VideoCore IV	VideoCore IV	VideoCore IV	250Mhz VideoCore IV	400Mhz VideoCore IV	2-8 GB
RAM	256Mb	512Mb	512Mb	512Mb	1Gb	1Gb	2 HDMI Port
USB	1	1	2	4	4	4	MicroSD
Video	RCA, HDMI	Jack, HDMI	RCA, HDMI	Jack, HDMI	Jack, HDMI	Jack, HDMI	Bluetooth&wireless
Audio	Jack, HDMI	Jack, HDMI	Jack, HDMI	Jack, HDMI	Jack, HDMI	Jack, HDMI	40 pin
Boot	Memoria SD	Memoria microSD	Memoria SD	Memoria microSD	Memoria microSD	Memoria microSD	H265 support
Wireless	No tiene	No tiene	No tiene	No tiene	No tiene	802.11n / Bluetooth 4.1	
Red Ethernet	No tiene	No tiene	Ethernet 10/100	Ethernet 10/100	Ethernet 10/100	Ethernet 10/100	
Alimentación	5V / 2Amp	5V / 2Amp	5V / 2Amp	5V / 2Amp	5V / 2Amp	5V / 2,5Amp	
GPIO	26 pines GPIO	40 pines GPIO	26 pines GPIO	40 pines GPIO	40 pines GPIO	40 pines GPIO	
Tamaño	85,6 x 53,98 mm	65 x 56 mm	85,6 x 53,98 mm	85 x 56 x 17 mm	85 x 56 x 17 mm	85 x 56 x 17 mm	

- The Raspberry Pi Foundation recommends the use of [Raspbian](#), a Debian-based Linux operating system.
- Other third party operating systems available via the official website include Ubuntu MATE, Snappy Ubuntu Core, Windows 10 IoT Core, RISC OS and specialised distributions for the Kodi media center and classroom management.

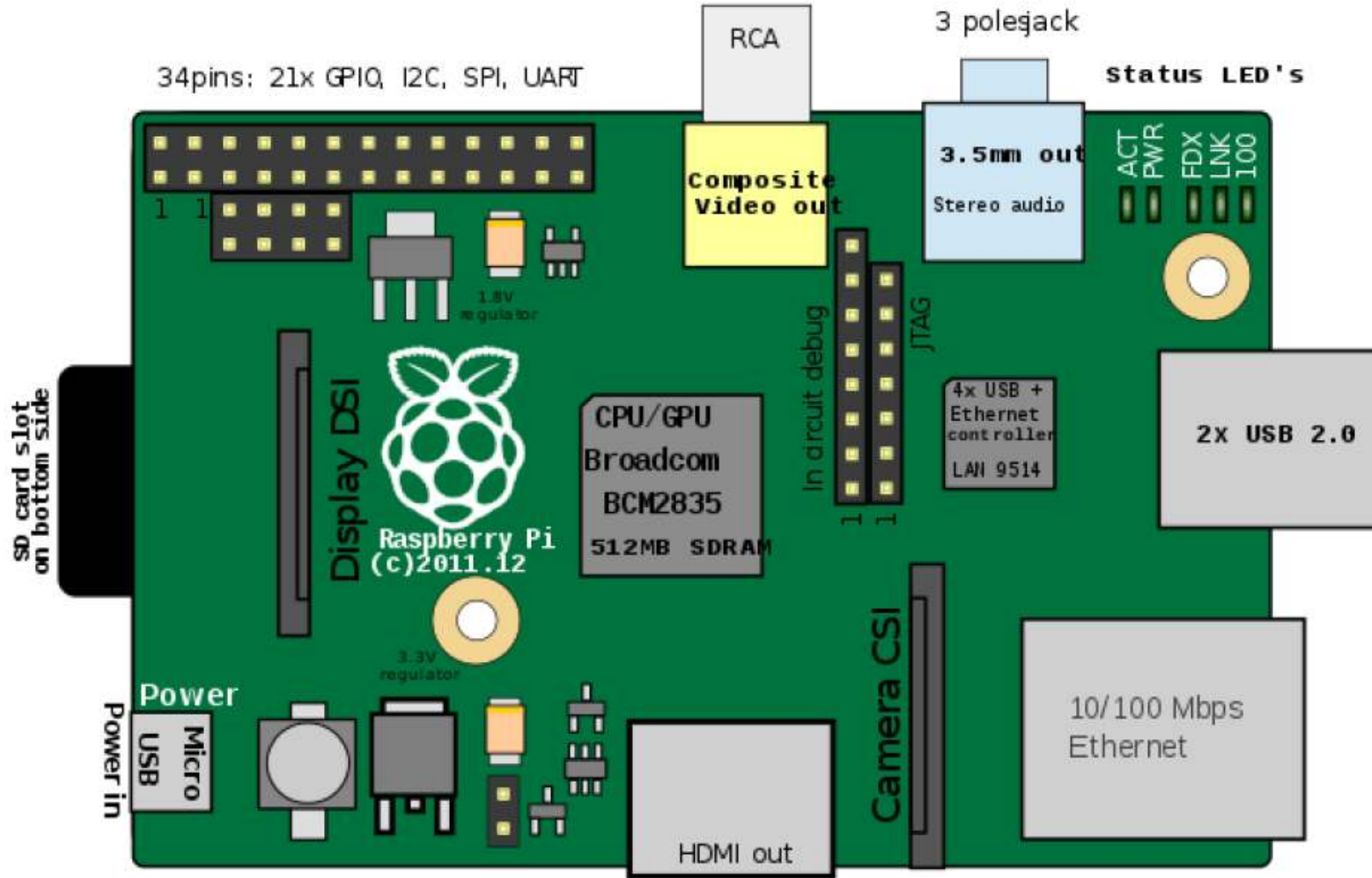
The MagPi



- The MagPi is a free fanzine for users of the Raspberry Pi computer.
- It was created by the community as an unofficial volunteer produced Raspberry Pi publication and in 2015 was handed over to the Raspberry Pi Foundation to be run in-house as the official Raspberry Pi magazine.
- It was launched in May 2012 and contains news, projects and tutorials.



Raspberry Pi Model B



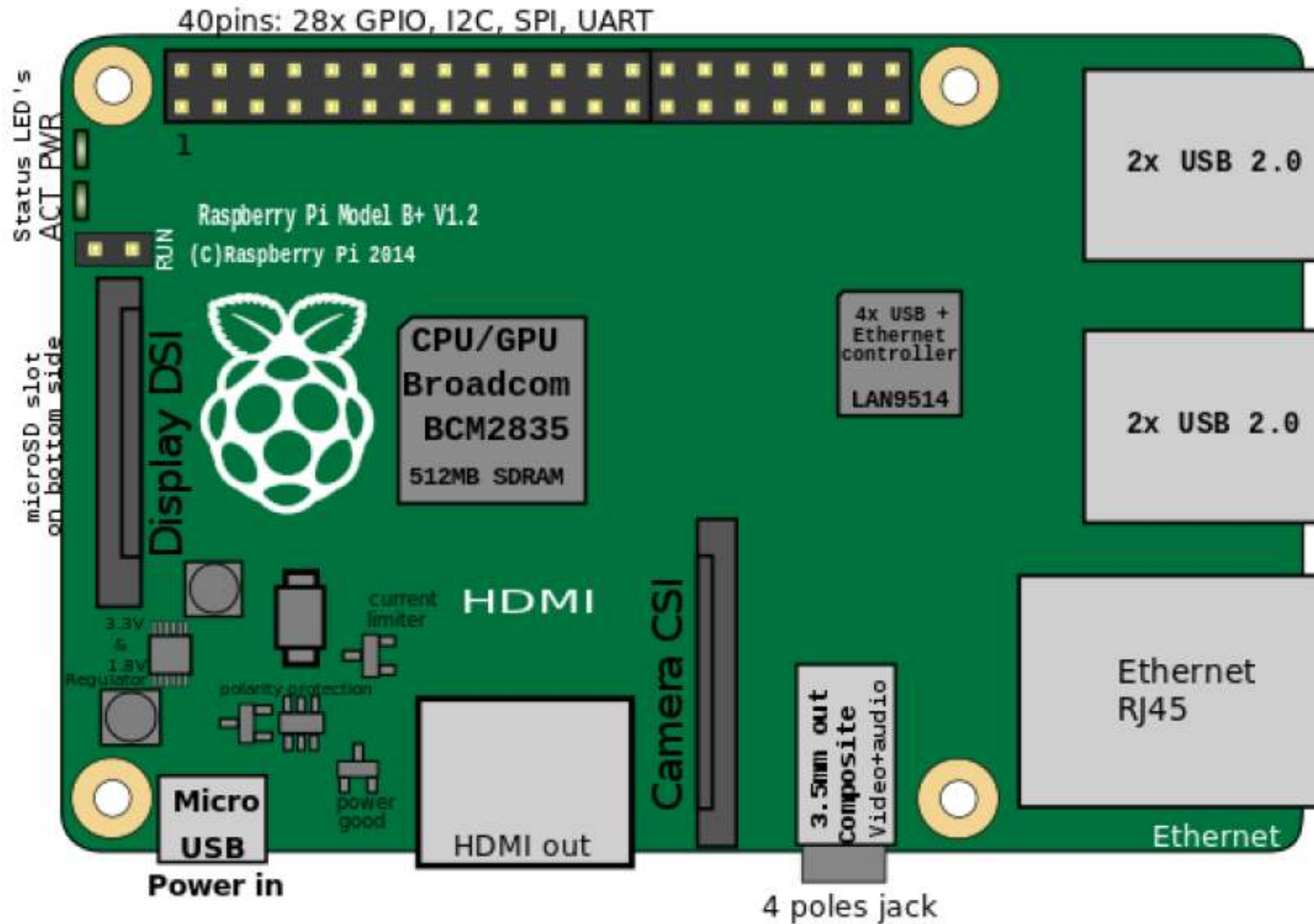
PIN #	NAME		NAME	PIN #
	3.3 VDC Power	1	5.0 VDC Power	2
8	SDA0 (I2C)	3	DNC	4
9	SCL0 (I2C)	5	0V (Ground)	6
7	GPIO 7	7	TxD (UART)	15
	DNC	9	RxD (UART)	16
0	GPIO 0	11	GPIO1	1
2	GPIO2	13	DNC	
3	GPIO3	15	GPIO4	4
	DNC	17	GPIO5	5
12	MOSI	19	DNC	
13	MISO	21	GPIO6	6
14	SCLK	23	CE0	10
	DNC	25	CE1	11

Attention! The GPIO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

<http://www.pi4j.com>

<https://pi4j.com/1.2/pins/model-b-rev2.html>

Raspberry Pi 2 Model B



GPIO#	NAME		NAME	GPIO#
	3.3 VDC Power	1	2	5.0 VDC Power
8	GPIO 8 SDA1 (I2C)	3	4	5.0 VDC Power
9	GPIO 9 SCL1 (I2C)	5	6	Ground
7	GPIO 7 GPCLK0	7	8	GPIO 15 TxD (UART)
	Ground	9	10	GPIO 16 RxD (UART)
0	GPIO 0	11	12	GPIO 1 PCM_CLK/PWM0
2	GPIO 2	13	14	Ground
3	GPIO 3	15	16	GPIO 4
	3.3 VDC Power	17	18	GPIO 5
12	GPIO 12 MOSI (SPI)	19	20	Ground
13	GPIO 13 MISO (SPI)	21	22	GPIO 6
14	GPIO 14 SCLK (SPI)	23	24	GPIO 10 CE0 (SPI)
	Ground	25	26	GPIO 11 CE1 (SPI)
30	SDA0 (I2C ID EEPROM)	27	28	SCL0 (I2C ID EEPROM)
21	GPIO 21 GPCLK1	29	30	Ground
22	GPIO 22 GPCLK2	31	32	GPIO 26 PWM0
23	GPIO 23 PWM1	33	34	Ground
24	GPIO 24 PCM_FS/PWM1	35	36	GPIO 27
25	GPIO 25	37	38	GPIO 28 PCM_DIN
	Ground	39	40	GPIO 29 PCM_DOUT

Attention! The GPIO pin numbering used in this diagram is intended for use with WiringPi / Pi4J. This pin numbering is not the raw Broadcom GPIO pin numbers.

<http://www.pi4j.com>