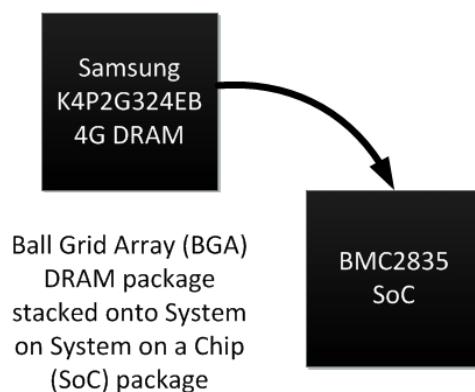


# **ARM**

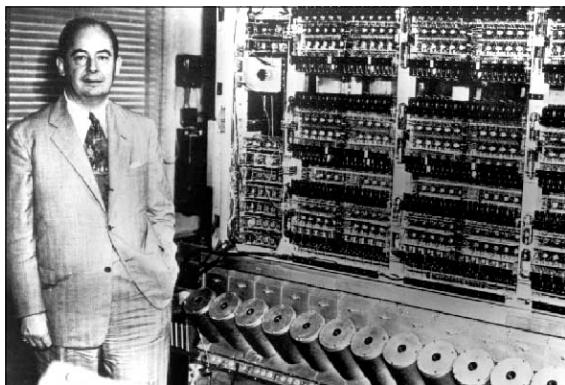
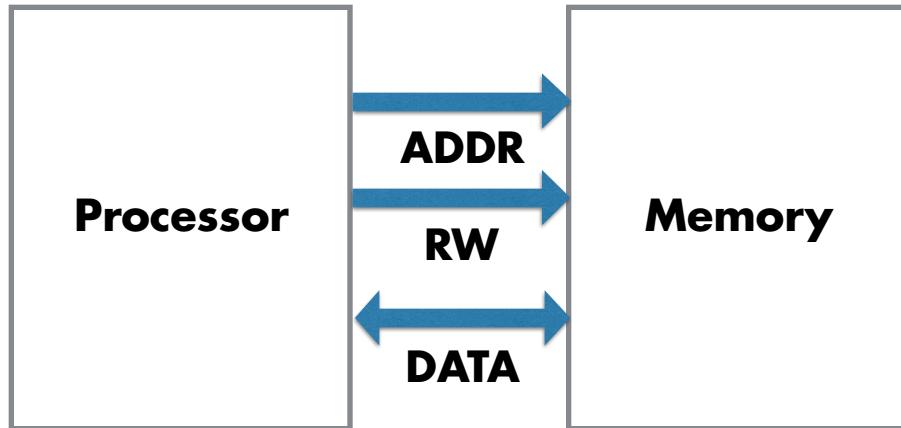
## **Architecture and Assembly**

### **Modest Goal: Blink an LED**

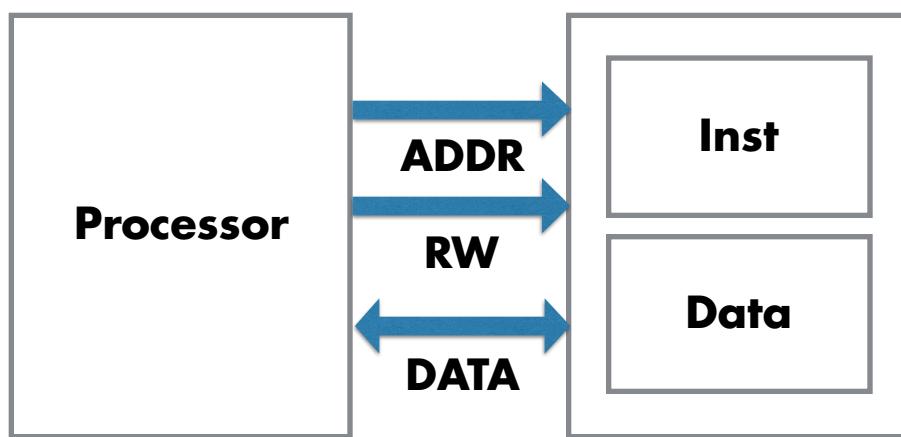
#### **Samsung 2 Gb SDRAM**



**ARMv6 Instruction set architecture (ISA)**  
**ARM1176JZF-S Microarchitecture**  
**Broadcom 2865 ARM Chip**



**(John) von Neumann  
Architecture**



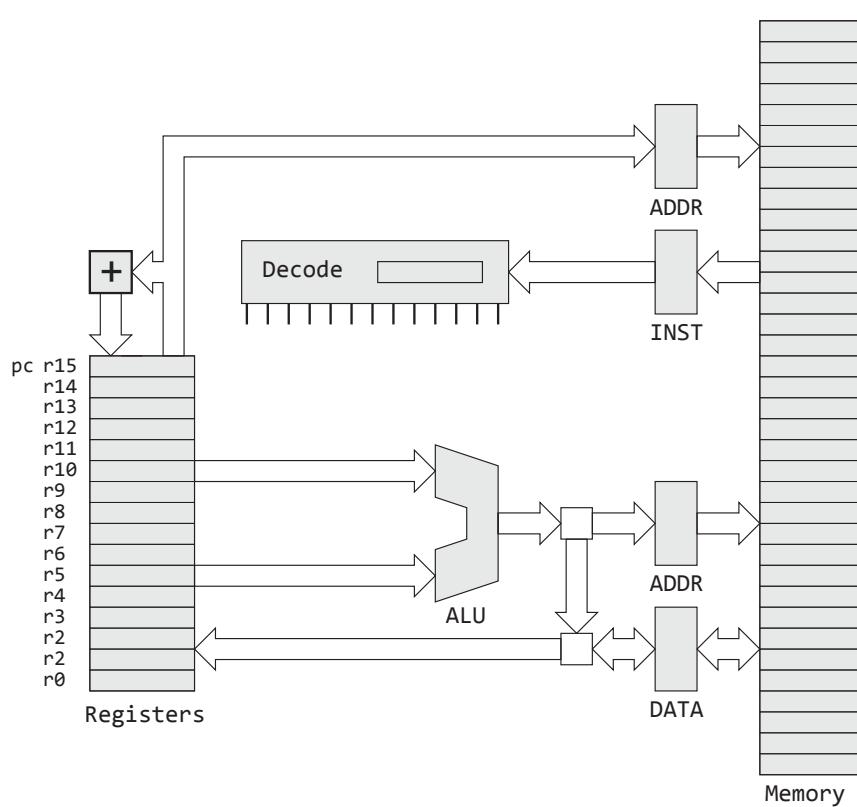
# Memory Map

$100000000_{16}$   
**4 GB**

**32-bit addresses**

**Address refers to a byte**

$01000000_{16}$   
**256 MB**



# ARM 32-bit Processor

**Processor designed around 32-bit “words”**

**Registers are 32-bits**

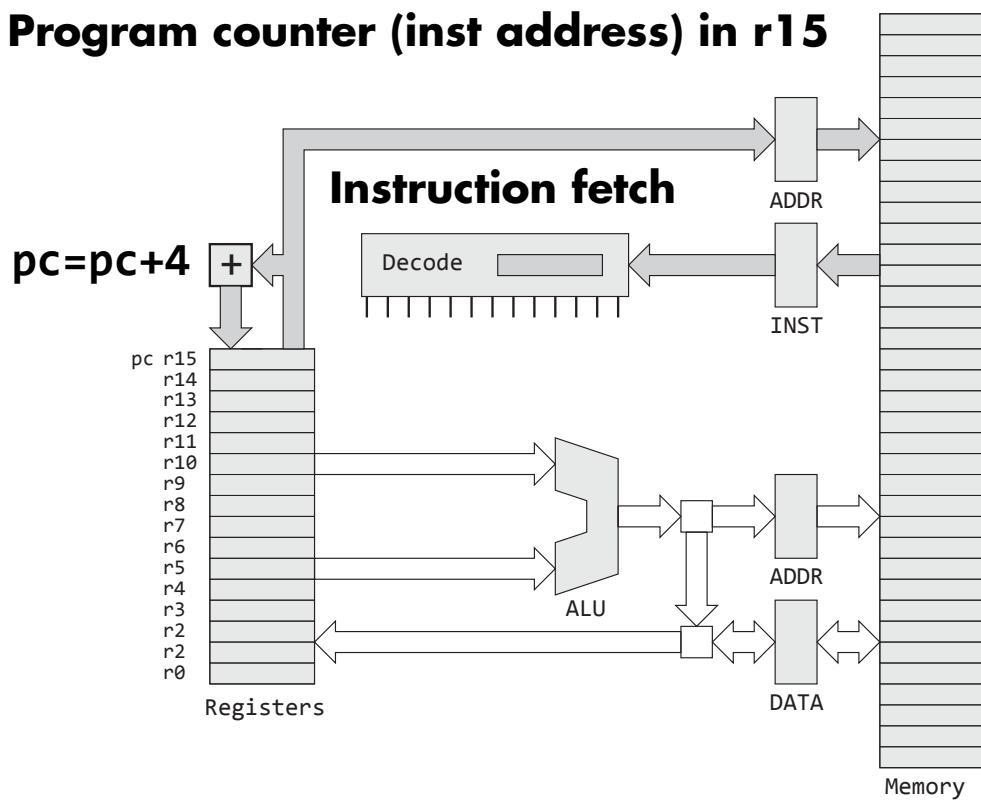
**Arithmetic-Logic Unit (ALU) works on 32-bits**

**Addresses are 32-bits**

**Instructions are 32-bits**

***The fact that everything is 32-bits simplifies things quite a bit!***

**Program counter (inst address) in r15**



# Instructions

## Meaning (C)

$r0 = r1 + 1$

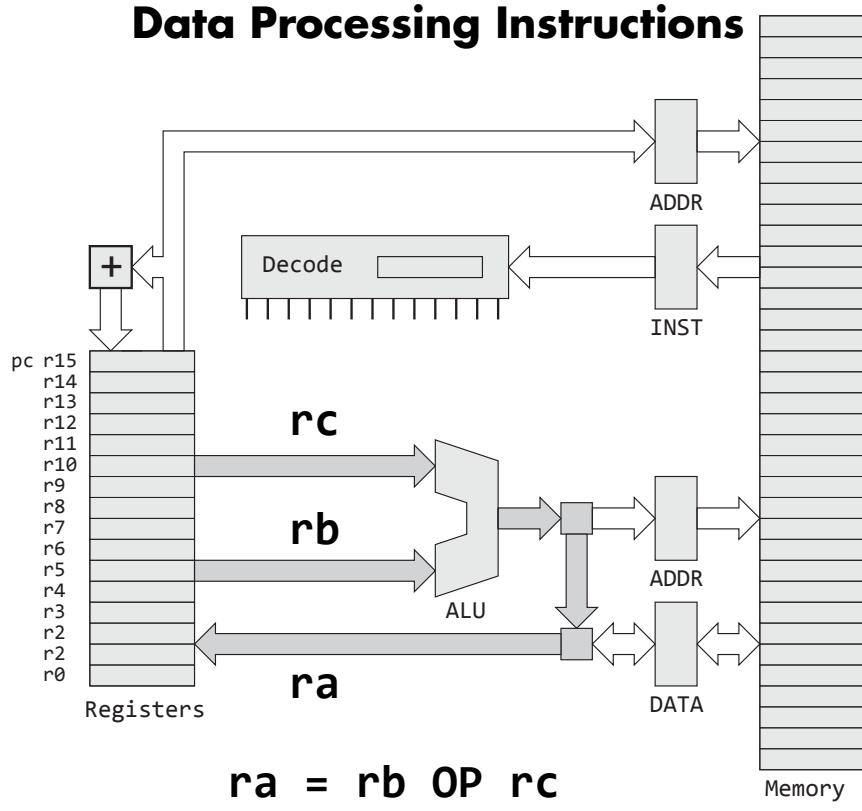
## Assembly language

add r0, r1, #1

## Machine language

E2 81 00 01

## Data Processing Instructions



# **Operations (OP)**

**Arithmetic: ADD, SUB, MUL, (no DIV), ...**

**Logical/Bit: AND, ORR, EOR, LSL, ...**

**Comparison: CMP, TST, ...**

**Move: MOV, ...**

***More detail in the next lecture ...***

```
// Single instruction program  
add r0, r1, #1 // #n is an immediate
```

# Cross-Development

**Run tools on your personal computer**

**Tools prefixed with arm-none-eabi-**

**Tools generate binary for raspberry pi**

**Copy binary to the raspberry pi**

**Power-cycle the raspberry pi to run**

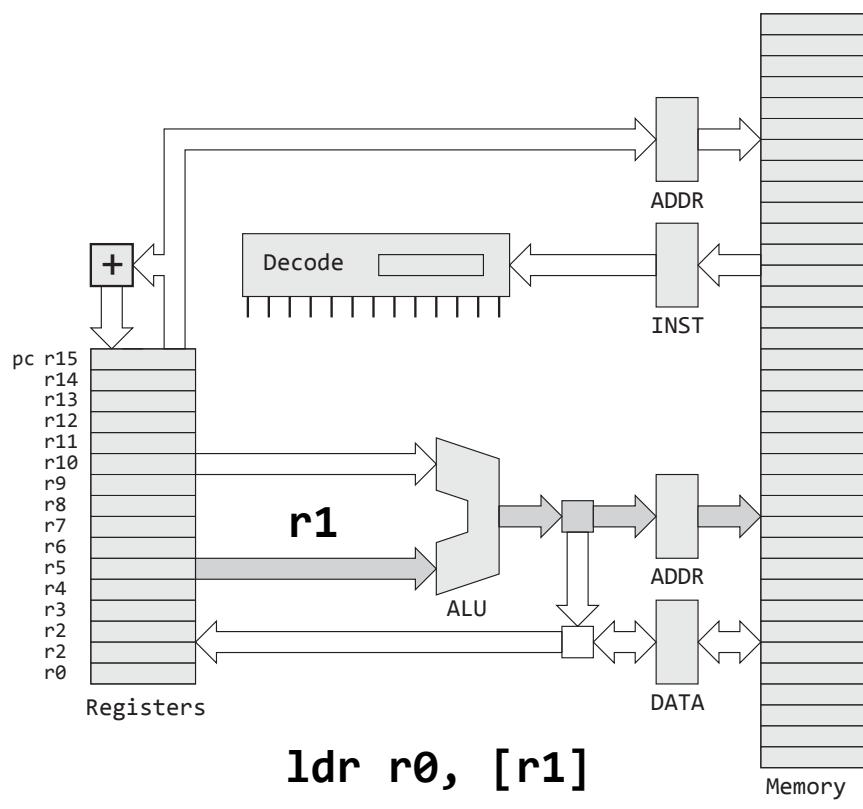
```
# Assemble and generate listing
% arm-none-eabi-as add.s -o add.o -a

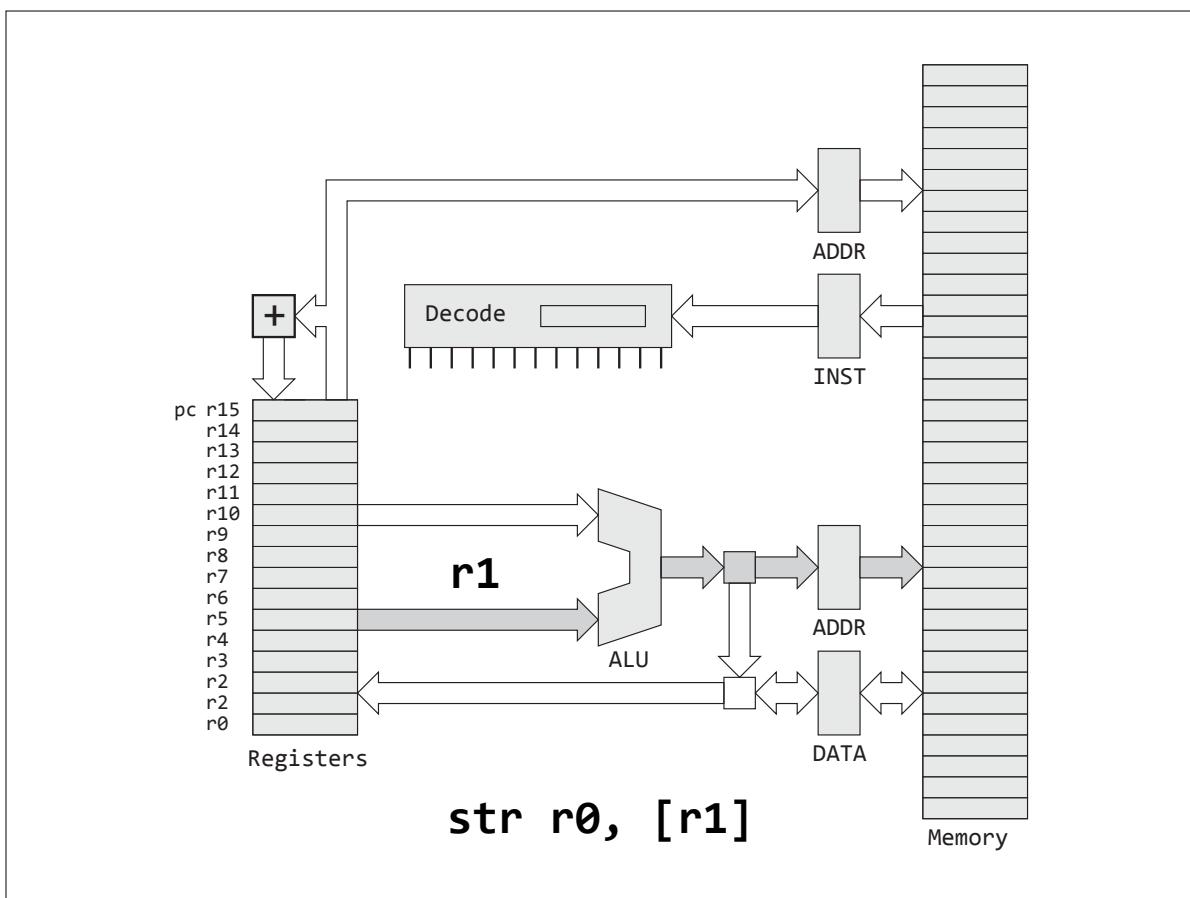
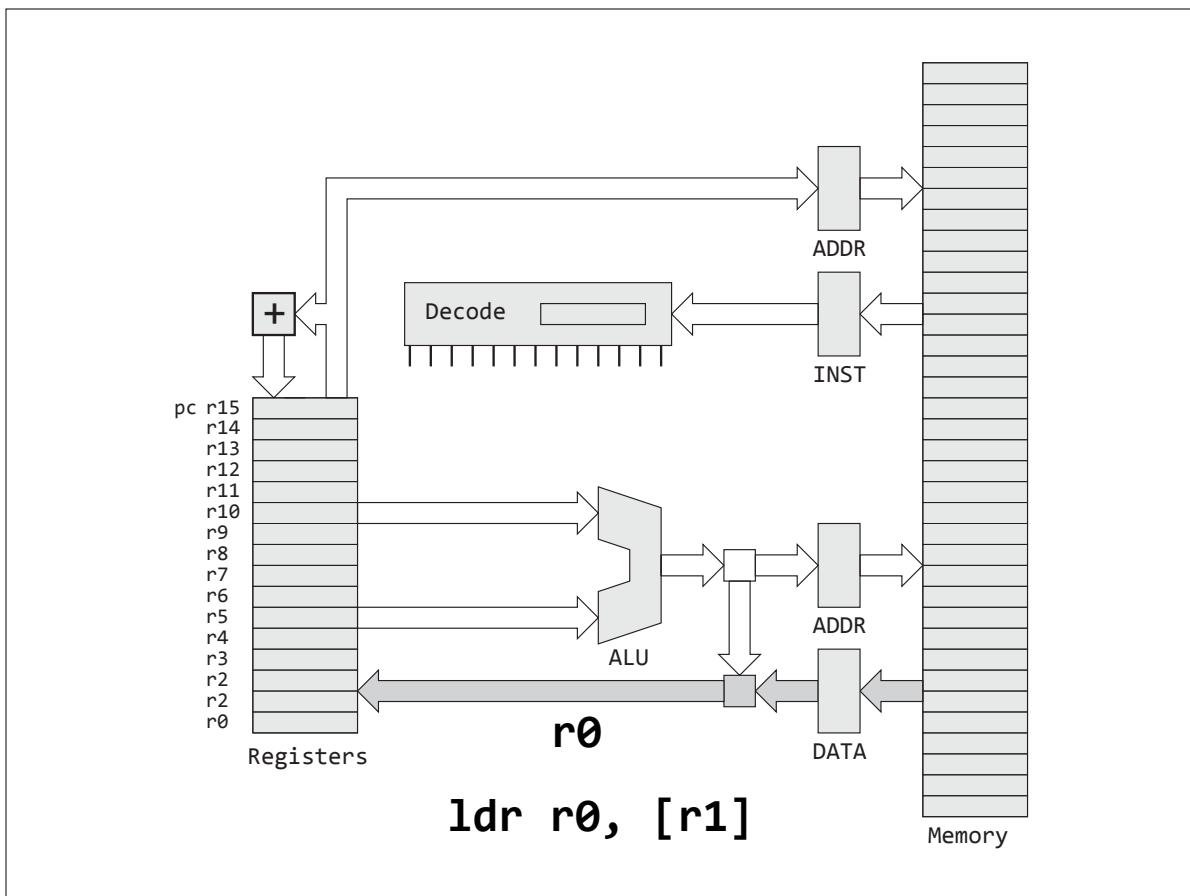
# Create binary
% arm-none-eabi-objdump add.o -O binary add.bin

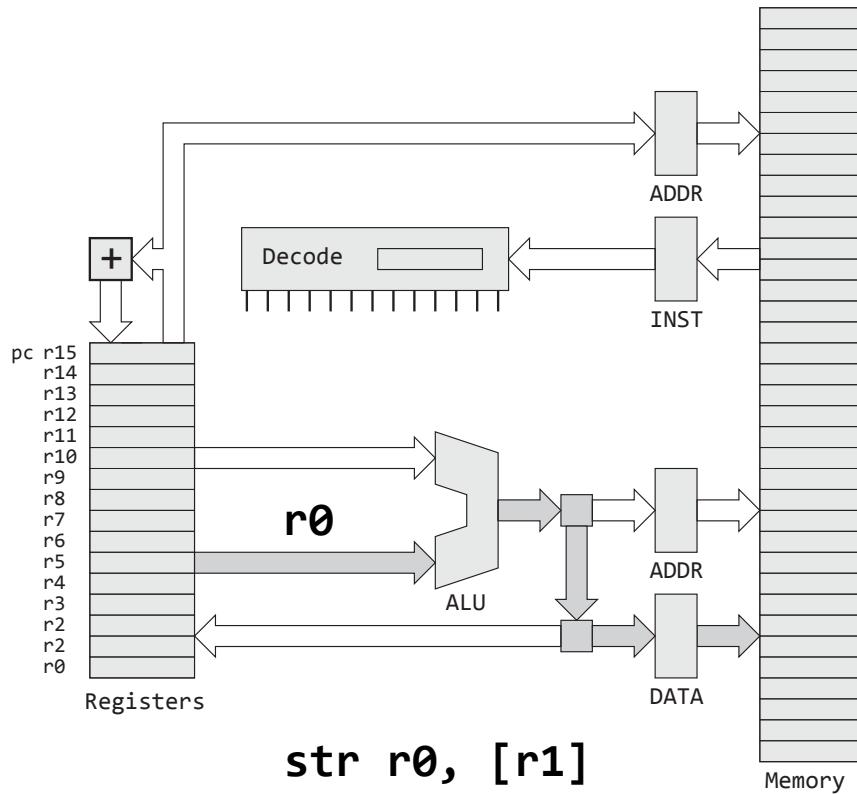
# Size
% ls -l add.bin
-rw-r--r-- 1 hanrahan  staff  4 add.bin

# Dump binary
% xxd -g 1 add.bin
0000000: 01 00 81 e2 // little-endian
```

# Loads and Stores







## Conceptual Exercises

- 1. Suppose you have 0x8000 stored in r0, how would you jump and start executing instructions at that location?**
- 2. All instructions are 32-bits. Can you add a 32-bit immediate constant to a register?**
- 3. What instruction do you think takes longer to execute, ldr or add?**

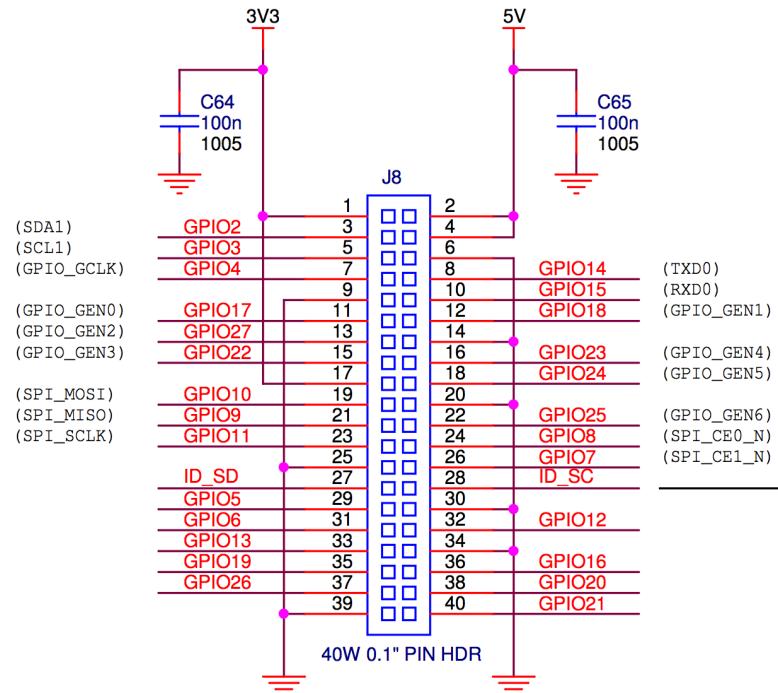
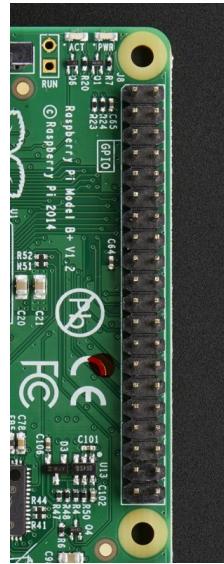
# **Blinking an LED**

## **Powering**

**The USB port on my Macbook Pro provides 500 mA @ 5V**

**How much power does the Pi need?**

# General-Purpose Input/Output (GPIO)

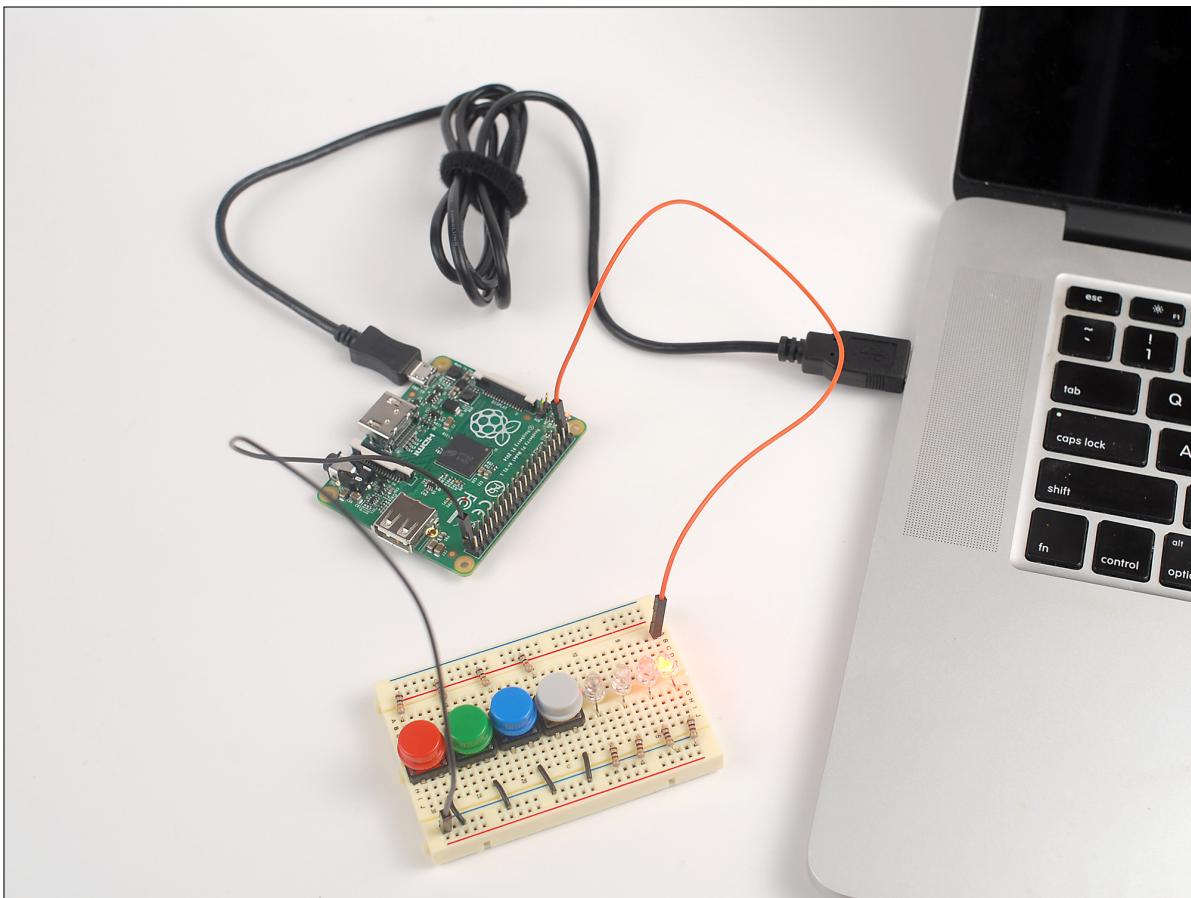
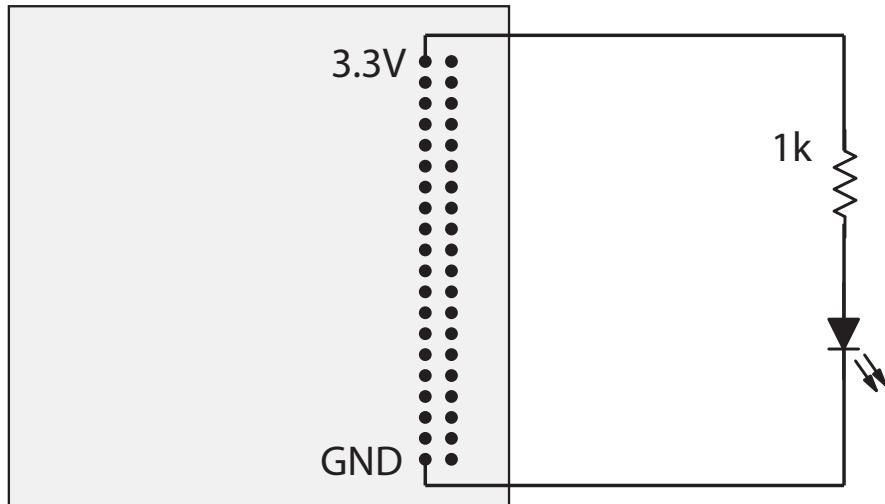


Power

5 in a column are connected

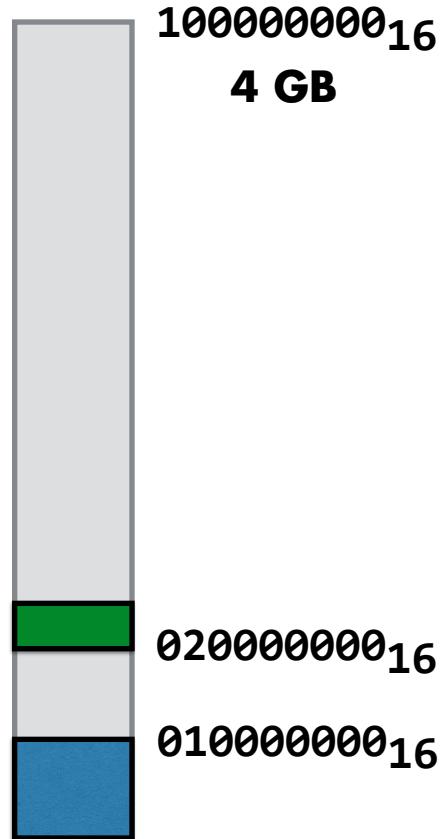
not connected across center divide

Ground



# Controlling GPIO Pins

## Memory Map



Address	Field Name	Description	Size	Read/ Write
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0000	GPFSEL0	GPIO Function Select 0	32	R/W
0x 7E20 0004	GPFSEL1	GPIO Function Select 1	32	R/W
0x 7E20 0008	GPFSEL2	GPIO Function Select 2	32	R/W
0x 7E20 000C	GPFSEL3	GPIO Function Select 3	32	R/W
0x 7E20 0010	GPFSEL4	GPIO Function Select 4	32	R/W
0x 7E20 0014	GPFSEL5	GPIO Function Select 5	32	R/W
0x 7E20 0018	-	Reserved	-	-

## Notes

1. 0x 7E00 0000 -> 0x 2000 0000
2. 54 GPIO pins
3. 3-bits per GPIO pin
4. => 6 GPIO function select registers

Ref: **BCM2835-ARM-Peripherals.pdf**

## GPIO Function Select Register



Bit Pattern	Pin Function
000	The pin is an input
001	The pin is an output
010	The pin does alternate function 0
011	The pin does alternate function 1
100	The pin does alternate function 2
101	The pin does alternate function 3
110	The pin does alternate function 4
111	The pin does alternate function 5

```

// Turn on an LED via GPIO 20

// FSEL2 controls pins 20-29

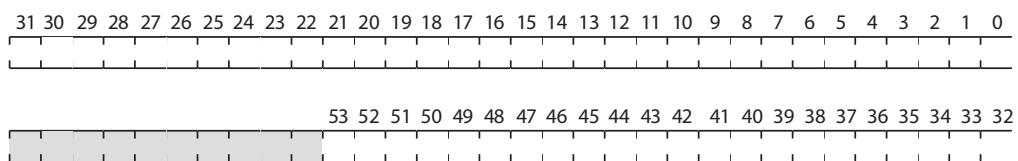
// load r0 with GPIO FSEL2 address
ldr r0, =0x20200008

// GPIO 20 function select is bits 0-2
// load r1 with 1 (OUTPUT)
mov r1, #1

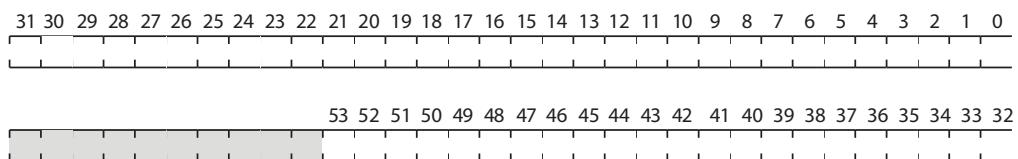
// store r1 in FSEL2 register
str r1, [r0]

```

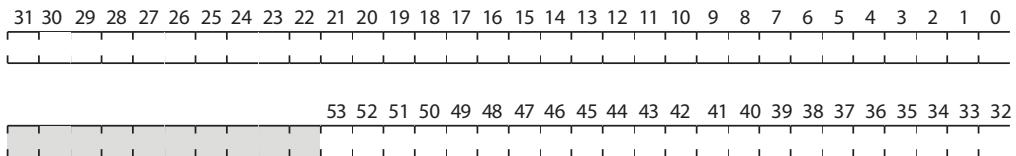
### **20 20 00 1C : GPIO SET0 Register**



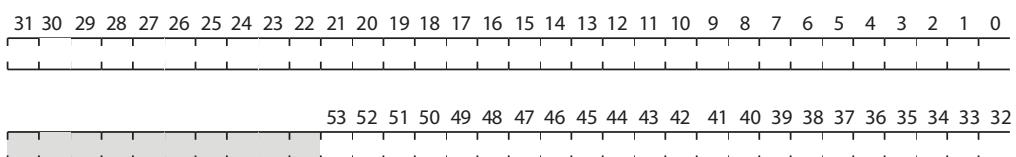
### **20 20 00 20 : GPIO SET1 Register**



## **20 20 00 28 : GPIO CLR0 Register**



## **20 20 00 2C : GPIO CLR1 Register**



...

```
// load r0 with GPIO SET0 register addr  
ldr r0, =0x2020001C  
  
// set bit 20 in r1  
mov r1, #0x100000 // 0x100000 = 1 << 20  
  
// store bit in GPIO SET0 register  
str r1, [r0]
```

```
...  
  
// load r0 with GPIO SET0 register addr  
ldr r0, =0x2020001C  
  
// set bit 20 in r1  
mov r1, #(1<<20)  
  
// store bit in GPIO SET0 register  
str r1, [r0]
```

```
...  
  
// load r0 with GPIO SET0 register addr  
ldr r0, =0x2020001C  
  
// set bit 20 in r1  
mov r1, #(1<<20)  
  
// store bit in GPIO SET0 register  
str r1, [r0]  
  
// loop forever using a branch  
b .
```

# **Booting on Power On**

- 1. Run hardware boot sequence**
  - 1. Initializes the processor and memory**
  - 2. Reads files from SDHC card**
- 2. GPU runs bootcode.bin**
- 3. GPU runs start.elf**
- 4. GPU loads kernel.img at 0x8000**
- 5. ARM processor jumps to 0x8000**

```
# What to do on your laptop

# Insert SD card - Volume mounts
% ls /Volumes/
BARE  Macintosh HD MobileBackups

# Copy to SD card
% cp on.bin /Volumes/BARE/kernel.img

# Eject and remove SD card
```

```
#  
# Insert SD card into SDHC slot on pi  
#  
# Apply power using usb console cable  
# PWR LED - 3.3V present  
# ACT LED - SD card access  
#  
# Raspberry pi boots  
#  
# LED connected to GPIO20 turns on!!  
#
```



## Blinking an LED

```
// Configure GPIO 20 for OUTPUT

loop:

    // Turn on LED

    // delay

    // Turn off LED

    // delay

    b loop
```

```
// Program to turn on an LED

// load r0 with address of GPIO FSEL2
ldr r0, =0x20200008
// load r1 with bit pattern for OUTPUT
mov r1, #1
// store r1 in memory address in r0
str r1, [r0]

// Bit 20 for GPIO 20
mov r1, #(1<<20)
```

```
...

// r0 points to GPIO SET2 register
ldr r0, =0x2020001C
str r1, [r0]

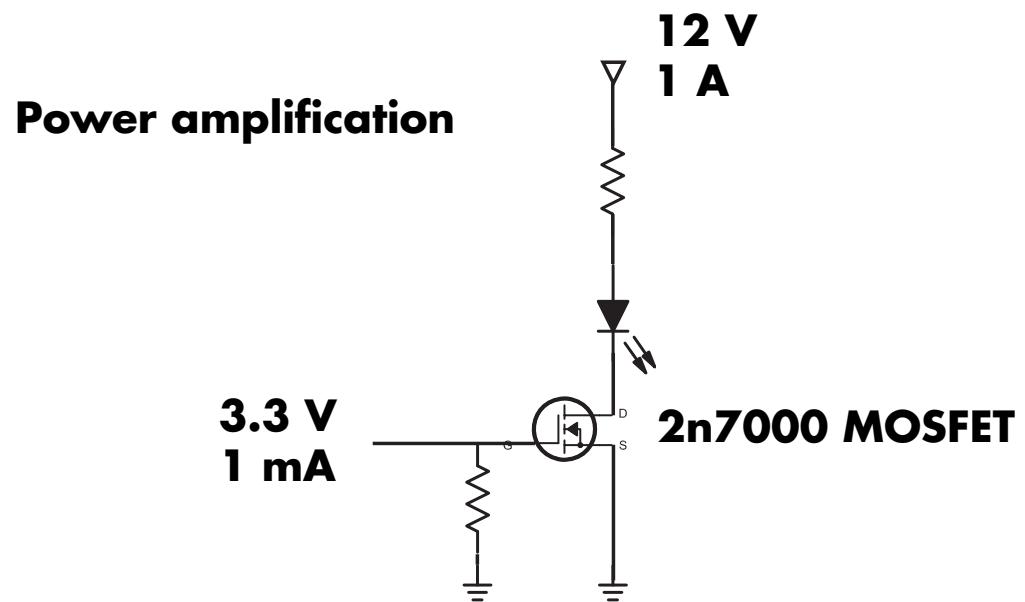
// delay
mov r2, #0x3F0000
wait1:
sub r2, #1
cmp r2, #0
bne wait1
```

```
...

// r0 points to GPIO CLR2 register
ldr r0, =0x20200028
str r1, [r0]

// delay
mov r2, #0x3F0000
wait2:
sub r2, #1
cmp r2, #0
bne wait2
```

# **Fun with LEDs**



**Transistor acts as a Switch**



**Details Omitted**

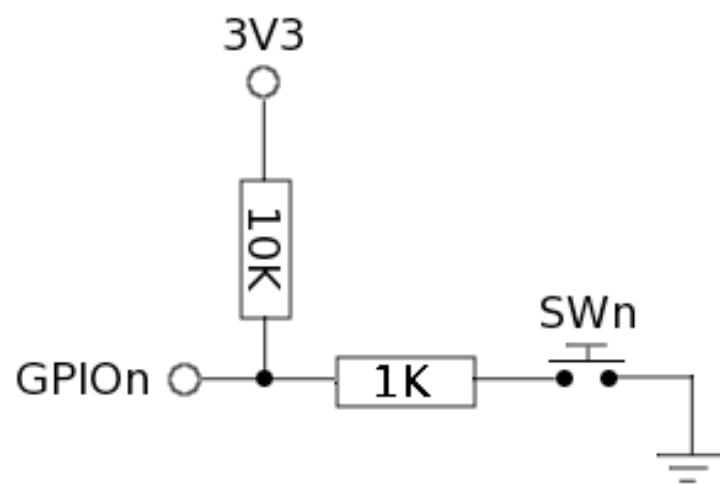
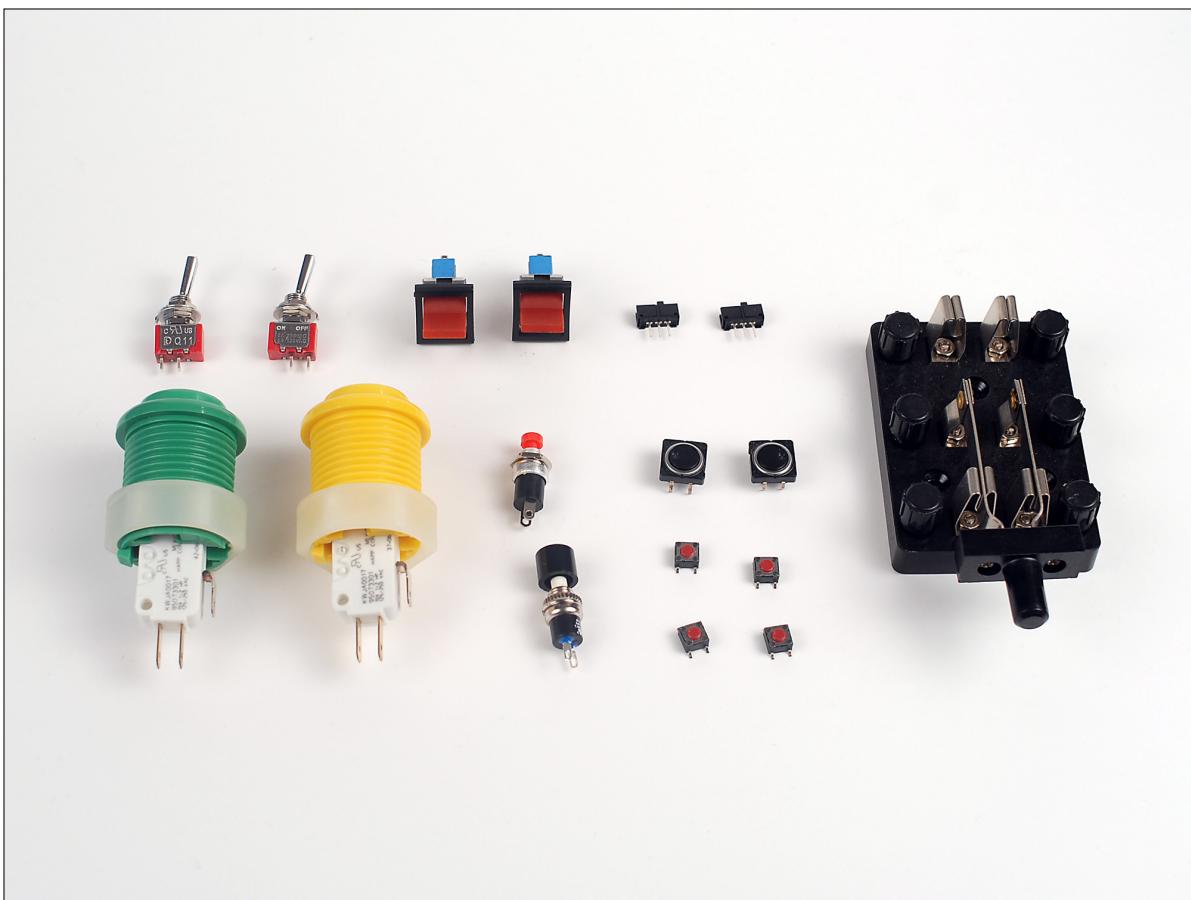
# **Definitive References**

**ARMv6 architecture reference manual**

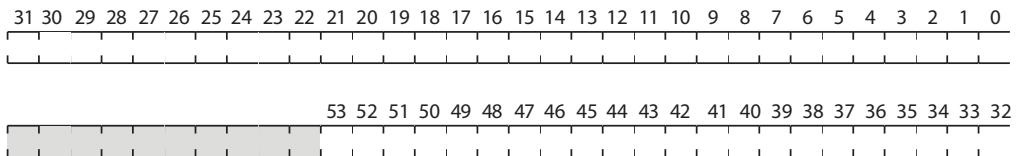
**BCM2865 peripherals document + errata**

**Raspberry pi schematic**

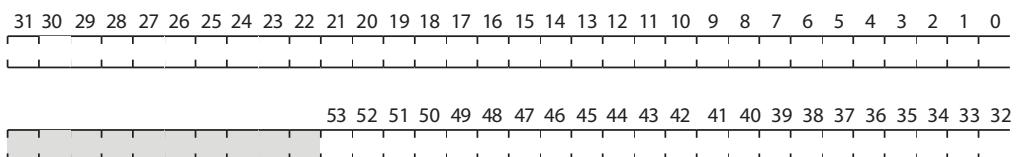
**Button → LED**



## **20 20 00 34 : GPIO LEV0 Register**



## **20 20 00 38 : GPIO LEV1 Register**



```
// Configure GPIO 10 for INPUT  
// Configure GPIO 20 for OUTPUT
```

```
loop:
```

```
    // Read push button  
  
    // if( button )  
    //     Turn on LED;  
    // else  
    //     Turn off LED;
```

```
b loop
```

```
// GPIO pins default to INPUT

// configure GPIO 20 for output

ldr r0, =0x20200008 // FSEL2
mov r1, #1
str r1, [r0]

// set bit 10
mov r2, #(1<<10)

// set bit 20
mov r3, #(1<<20)
```

```
loop:
// read GPIO 10
ldr r0, =0x20200034
ldr r1, [r0]
tst r1, r2
bne lo

// set GPIO 20 high
hi:
ldr r0, =0x2020001C
str r3, [r0]
b loop
// set GPIO 20 low
lo:
ldr r0, =0x20200028
str r3, [r0]
b loop
```