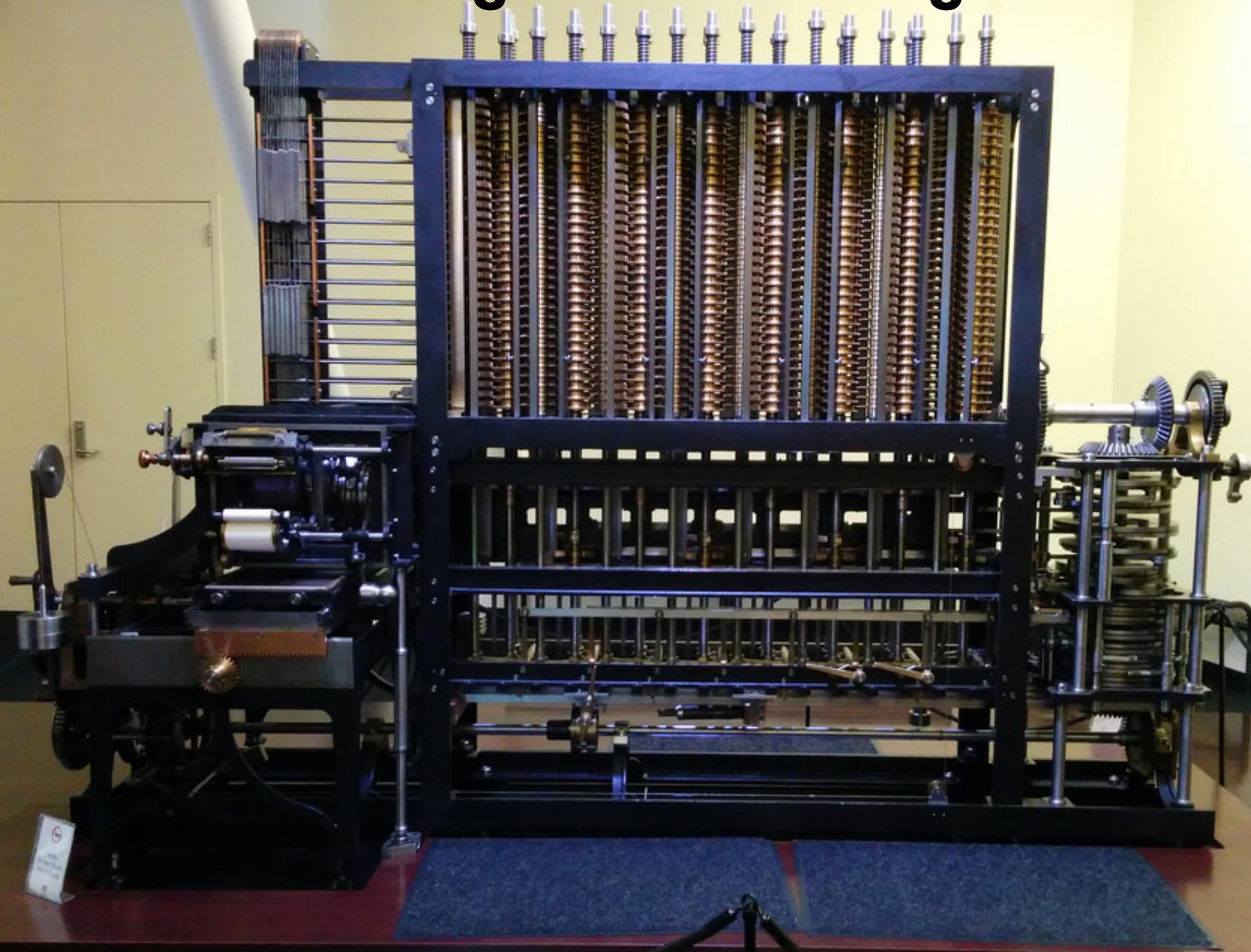
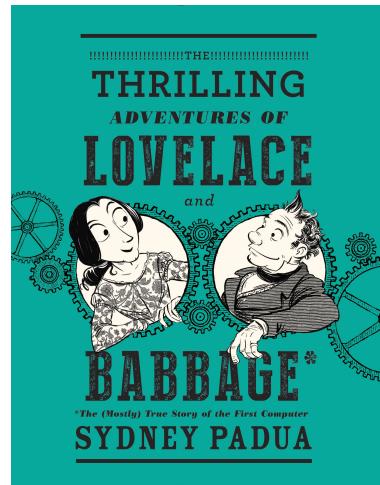


ARM Processor and Memory Architecture

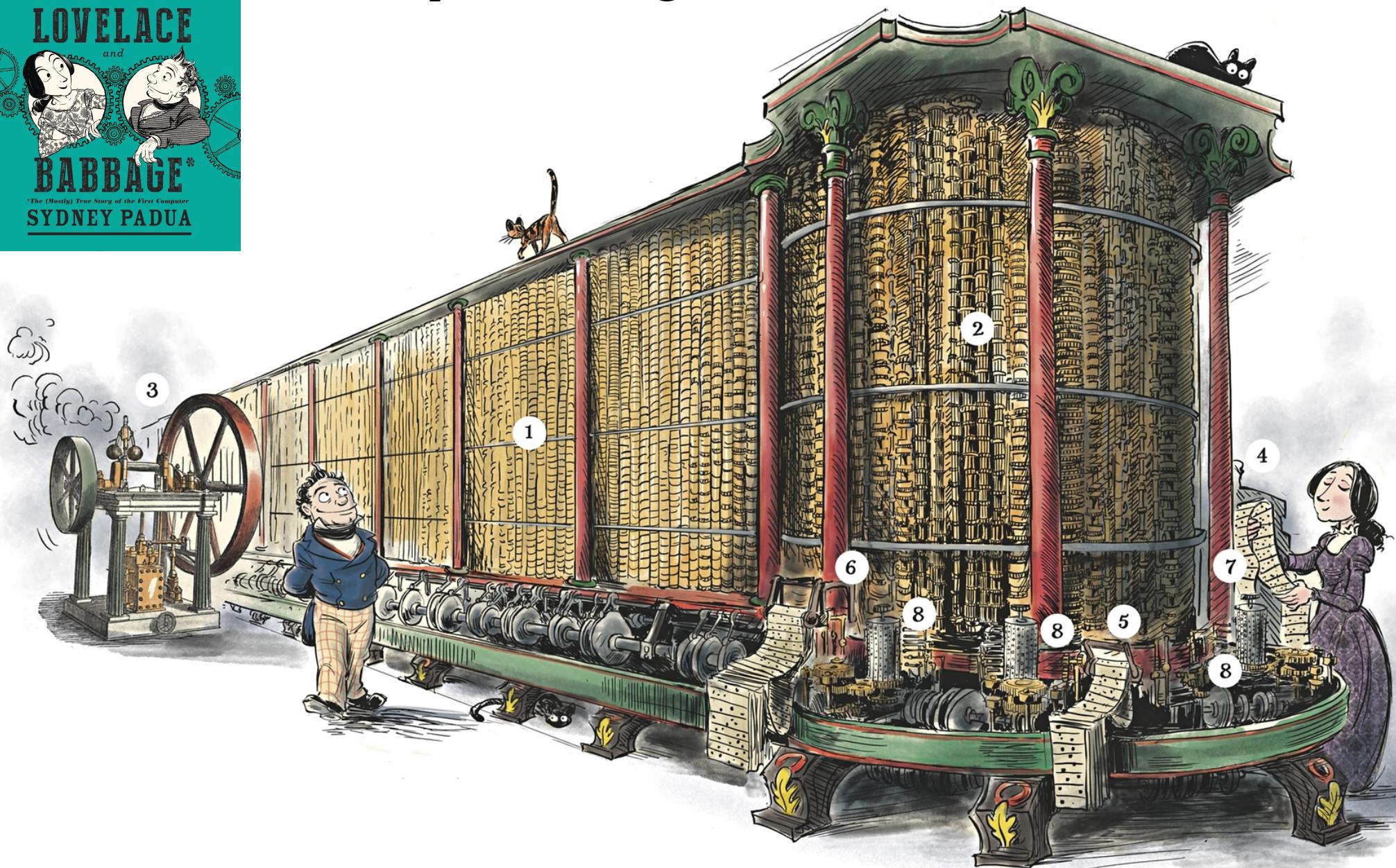
Goal: Turn on an LED

Babbage Difference Engine

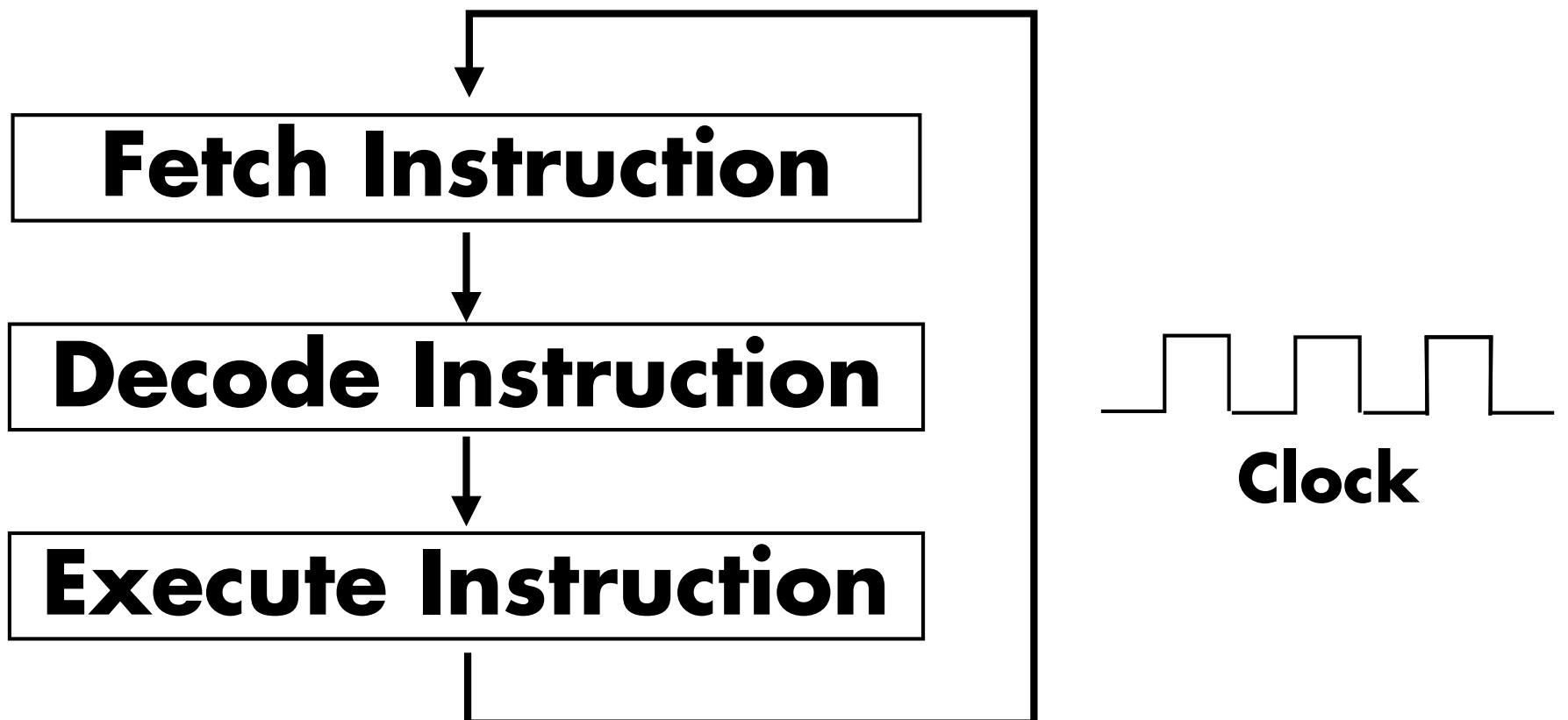


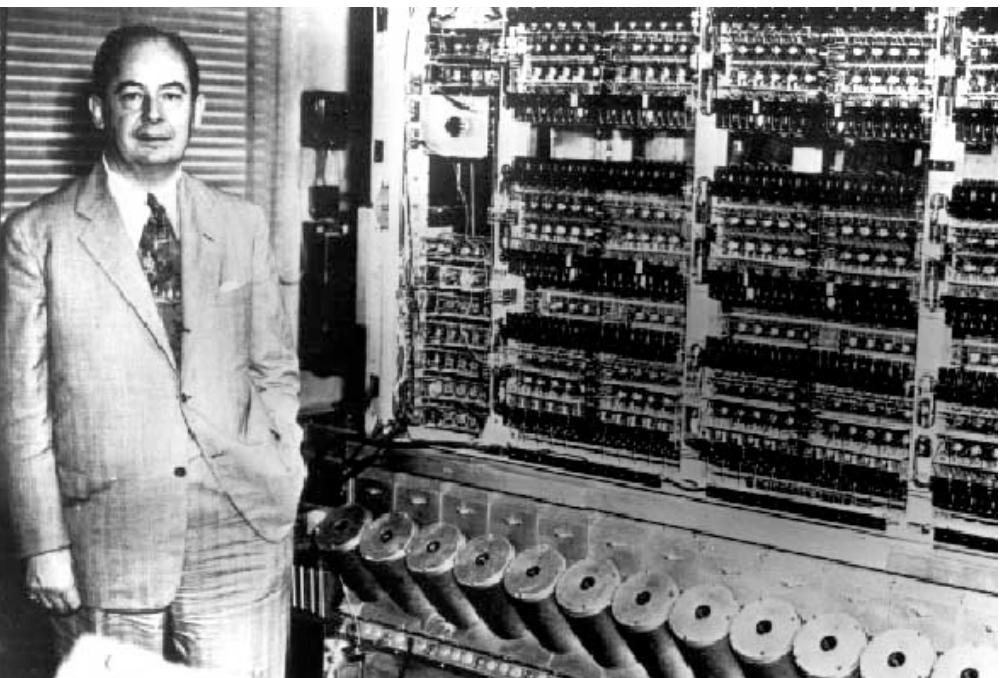


Analytical Engine



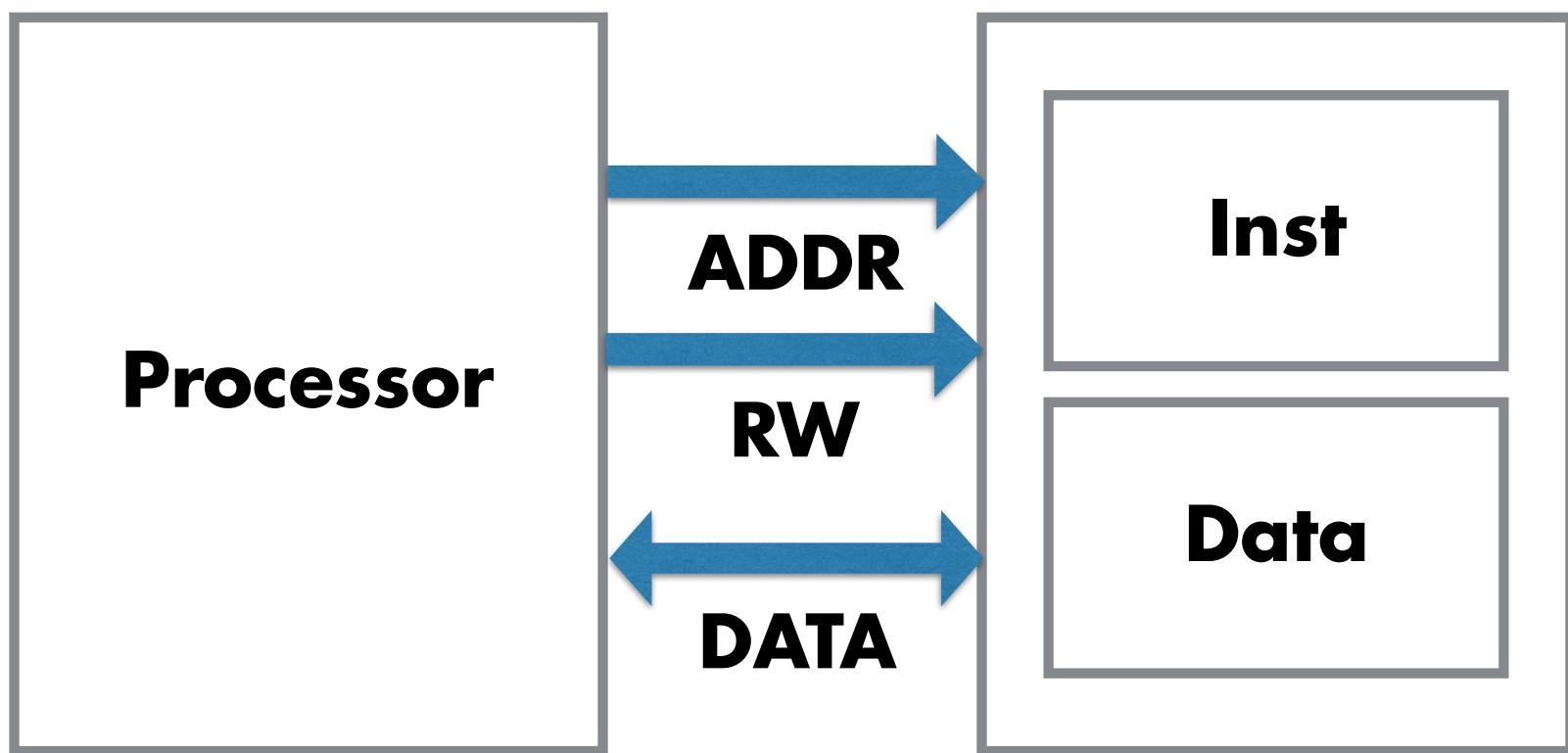
Running a "Program"





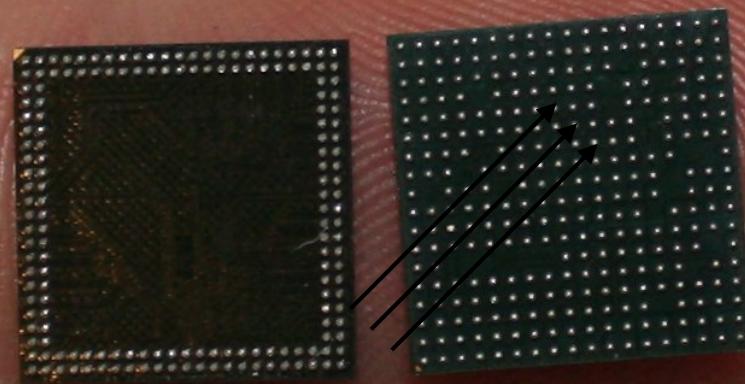
(John) von Neumann Architecture

Instructions and data stored in the same memory



Package on Package

Broadcom 2865 ARM Processor



Samsung 2Gb SDRAM

Memory used to store information

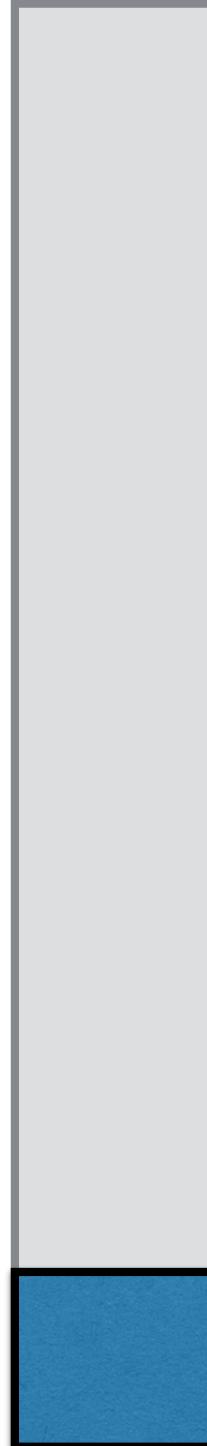
Stores both instructions and data

Storage locations are accessed using 32-bit addresses

Address refers to a byte (8-bits)

Maximum addressable memory 4 GB

Actual memory is 256 MB



10000000_{16}

Memory Map

01000000_{16}

256 MB

ARM 32-bit Architecture

Processor designed around 32-bit “words”

Registers are 32-bits

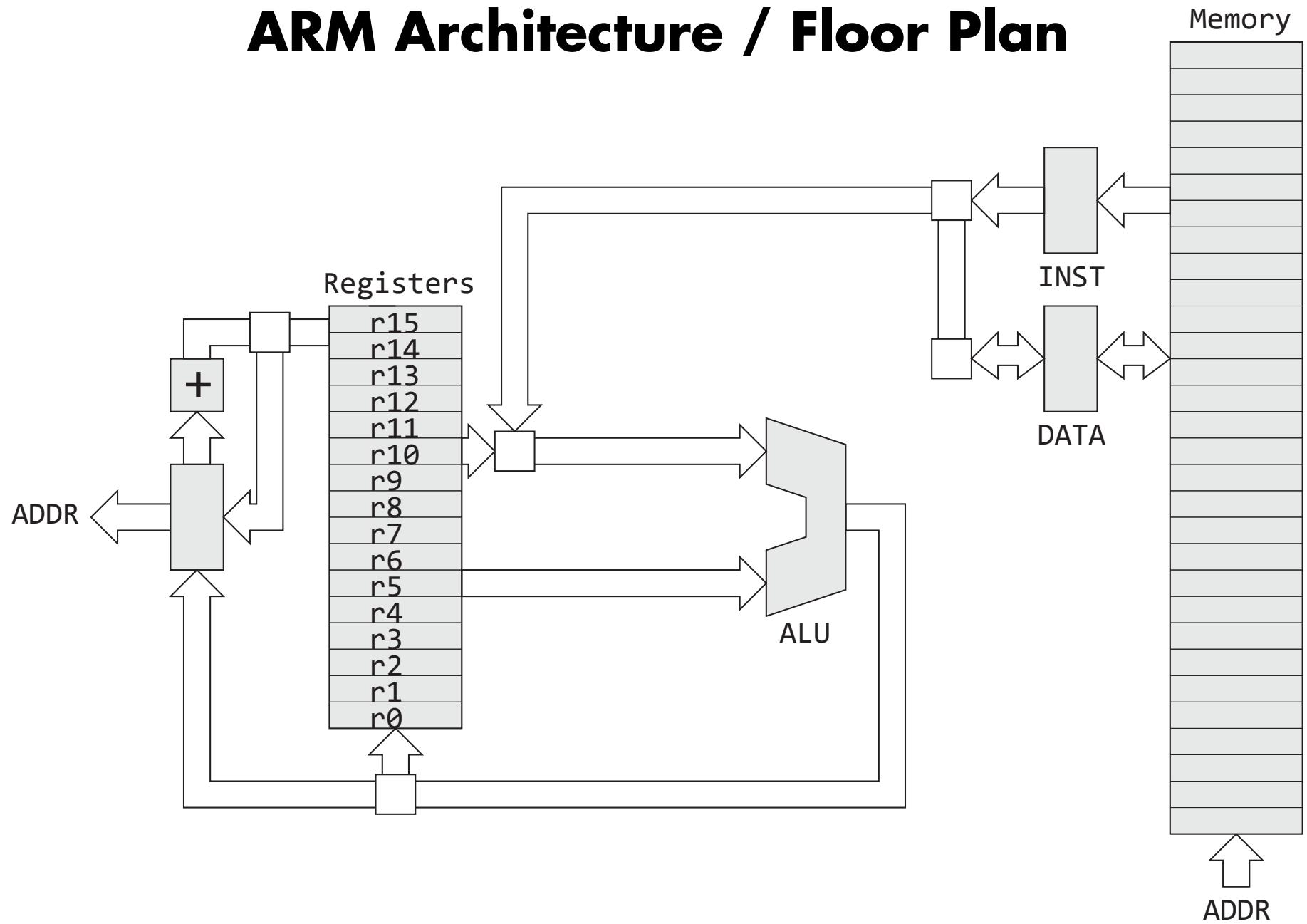
Arithmetic-Logic Unit (ALU) works on 32-bits

Addresses are 32-bits

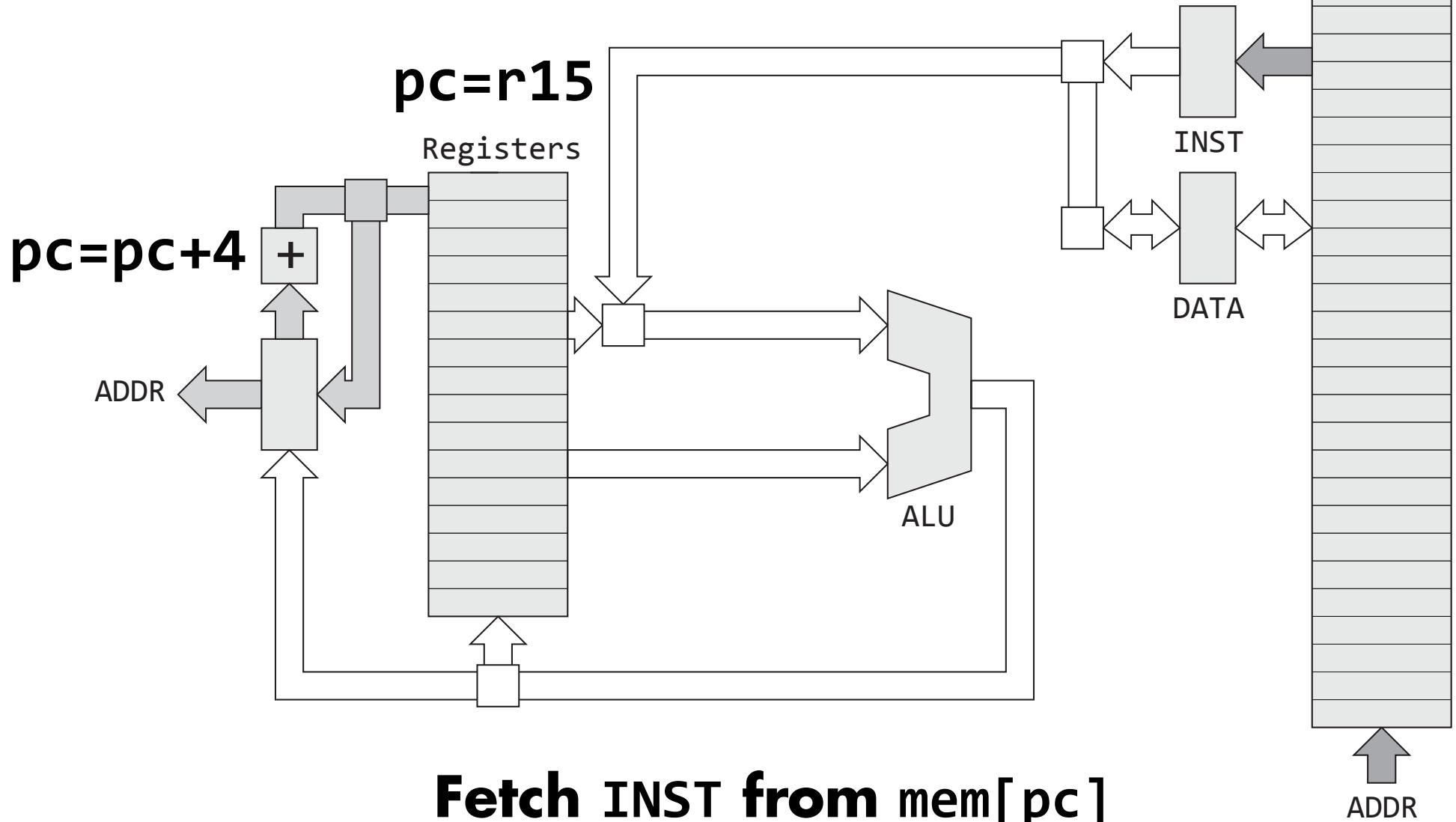
Instructions are 32-bits

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

ARM Architecture / Floor Plan



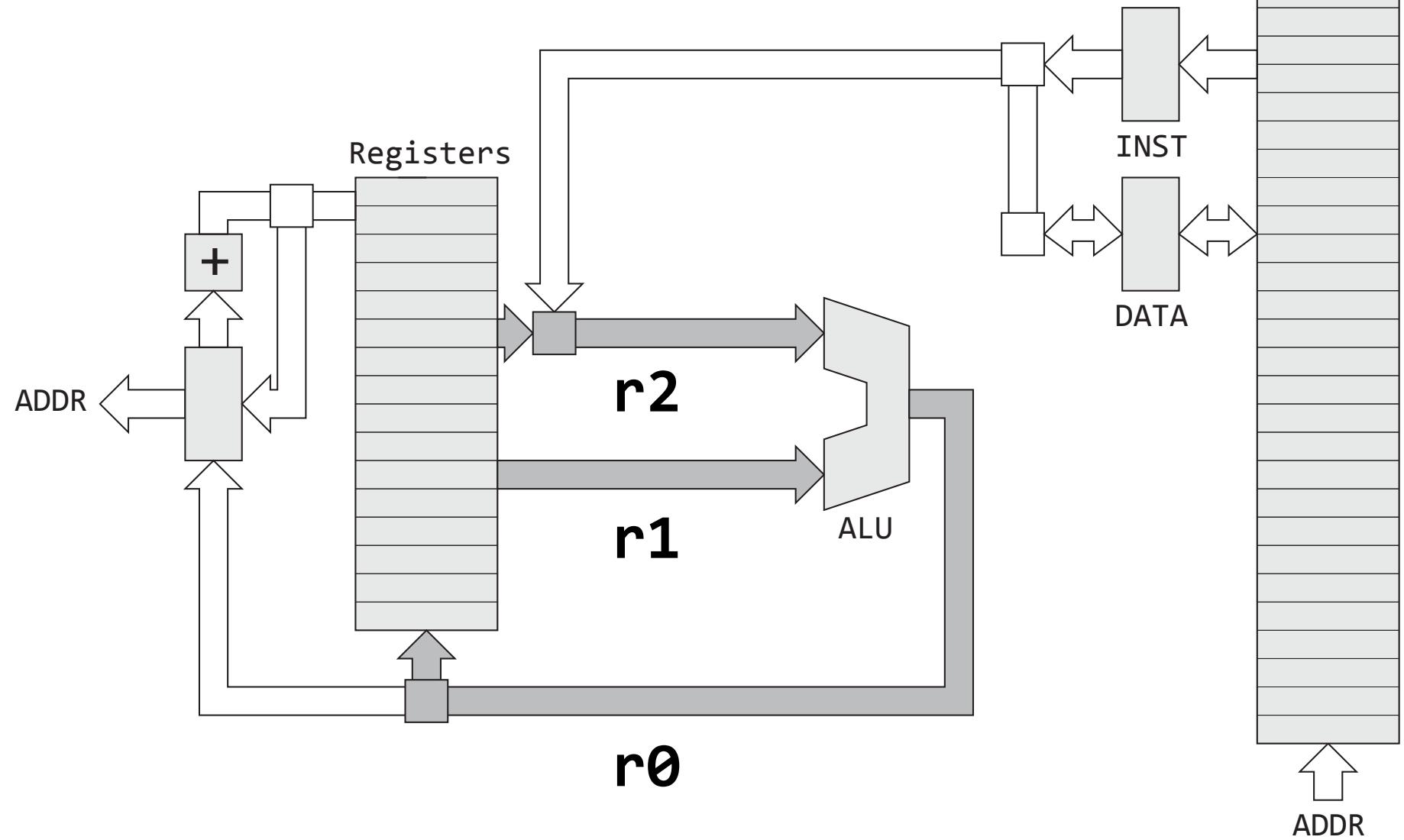
pc : program counter



Decode

$$r_0 = r_1 + r_2$$

add r0, r1, r2



Execute INST

Instructions

Meaning (C)

$$r0 = r1 + r2$$

Assembly language

add r0, r1, r2

Machine code

E0 81 00 02

// Single instruction program

add r0, r1, r2

```
# Assemble (.s) into 'object' file (.o)
% arm-none-eabi-as add.s -o add.o

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

# Size in bytes?
% ls -l add.bin
-rw-r--r--+ 1 hanrahan  staff  4 add.bin

# Dump binary in hex
% hexdump add.bin
0000000: 02 00 81 e0
```

Instructions

Meaning (C)

$r0 = r1 + \#1$

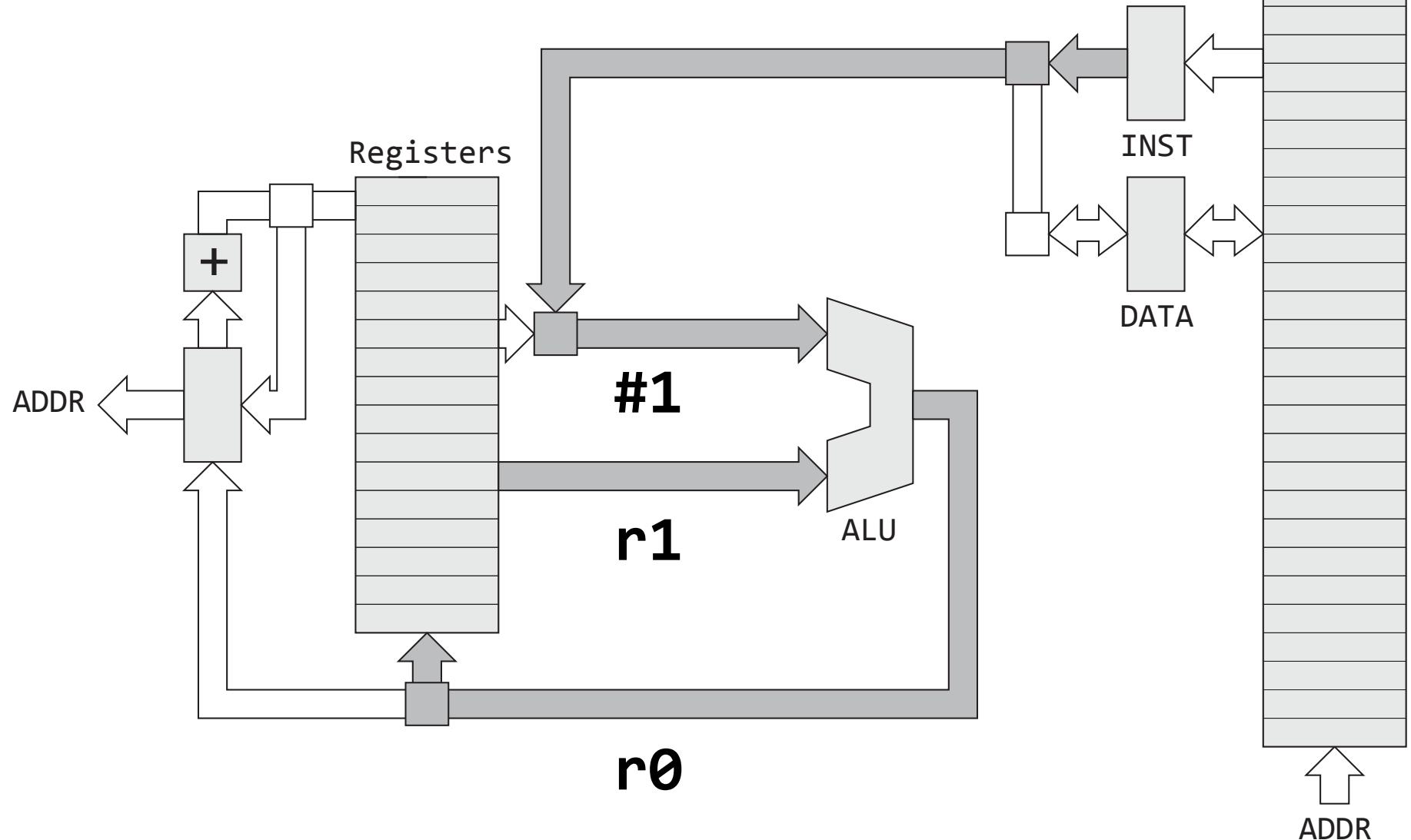
Assembly language

add r0, r1, #1

Machine code

E2 81 00 01

$$r0 = r1 + 1$$



add r0, r1, #1

// Single instruction program

mov r0, #1

```
# Assemble (.s) into 'object' file (.o)
% arm-none-eabi-as mov.s -o mov.o

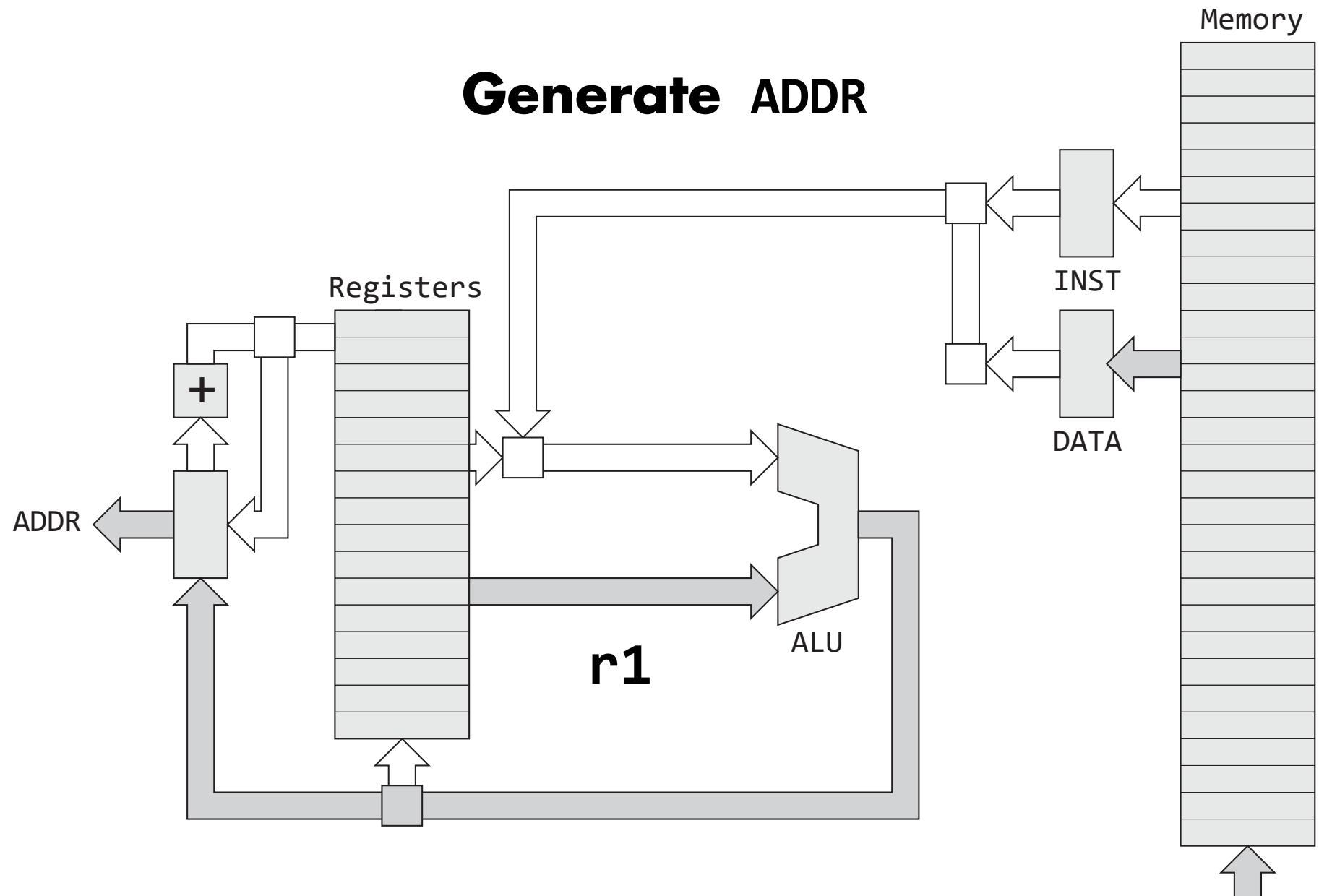
# Create binary (.bin)
% arm-none-eabi-objdump add.o -O binary mov.bin

# Size in bytes?
% ls -l mov.bin
-rw-r--r--+ 1 hanrahan  staff  4 mov.bin

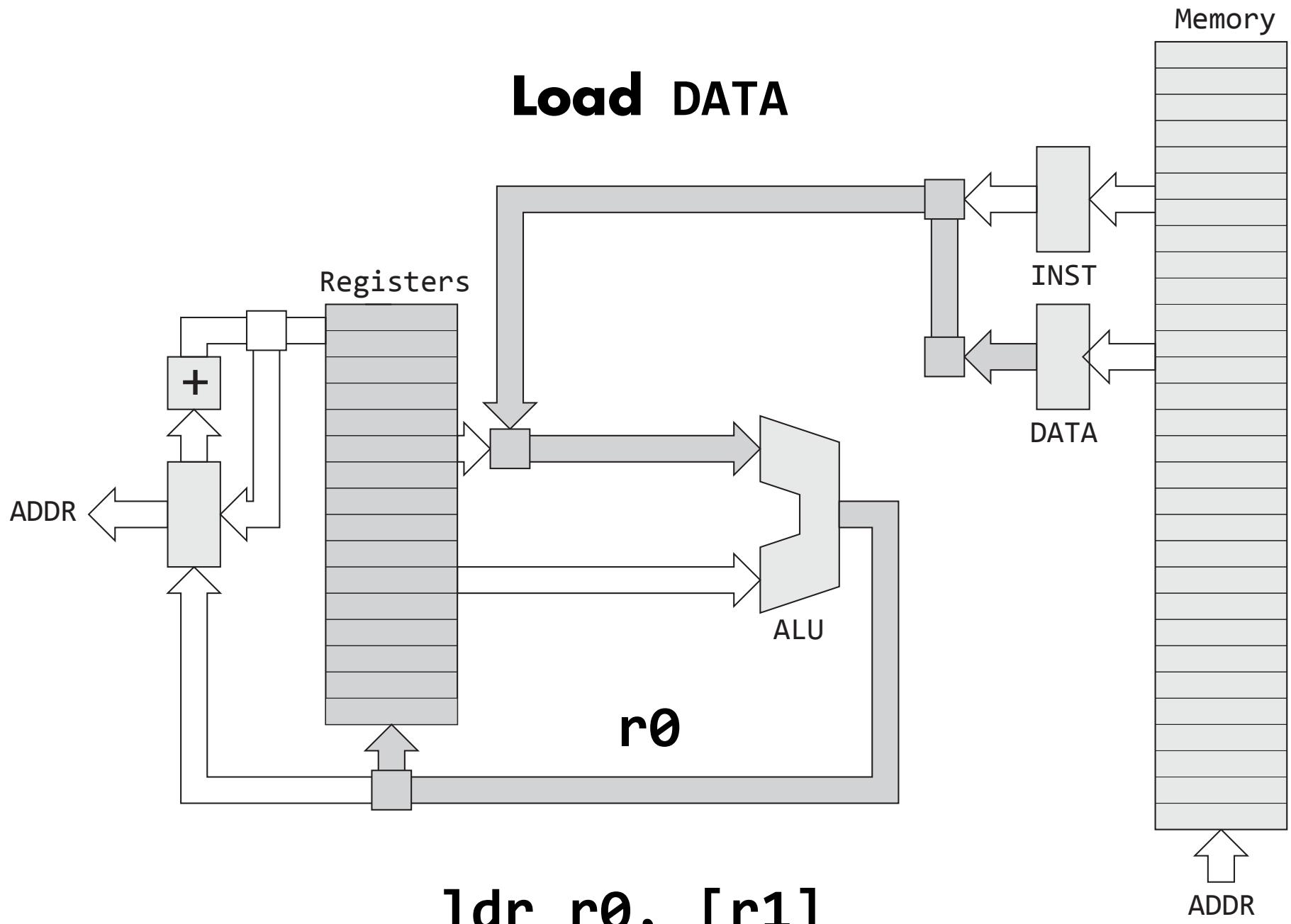
# Dump binary in hex
% xxd -g 1 mov.bin
0000000: 01 00 a0 e3
```

Loads and Stores

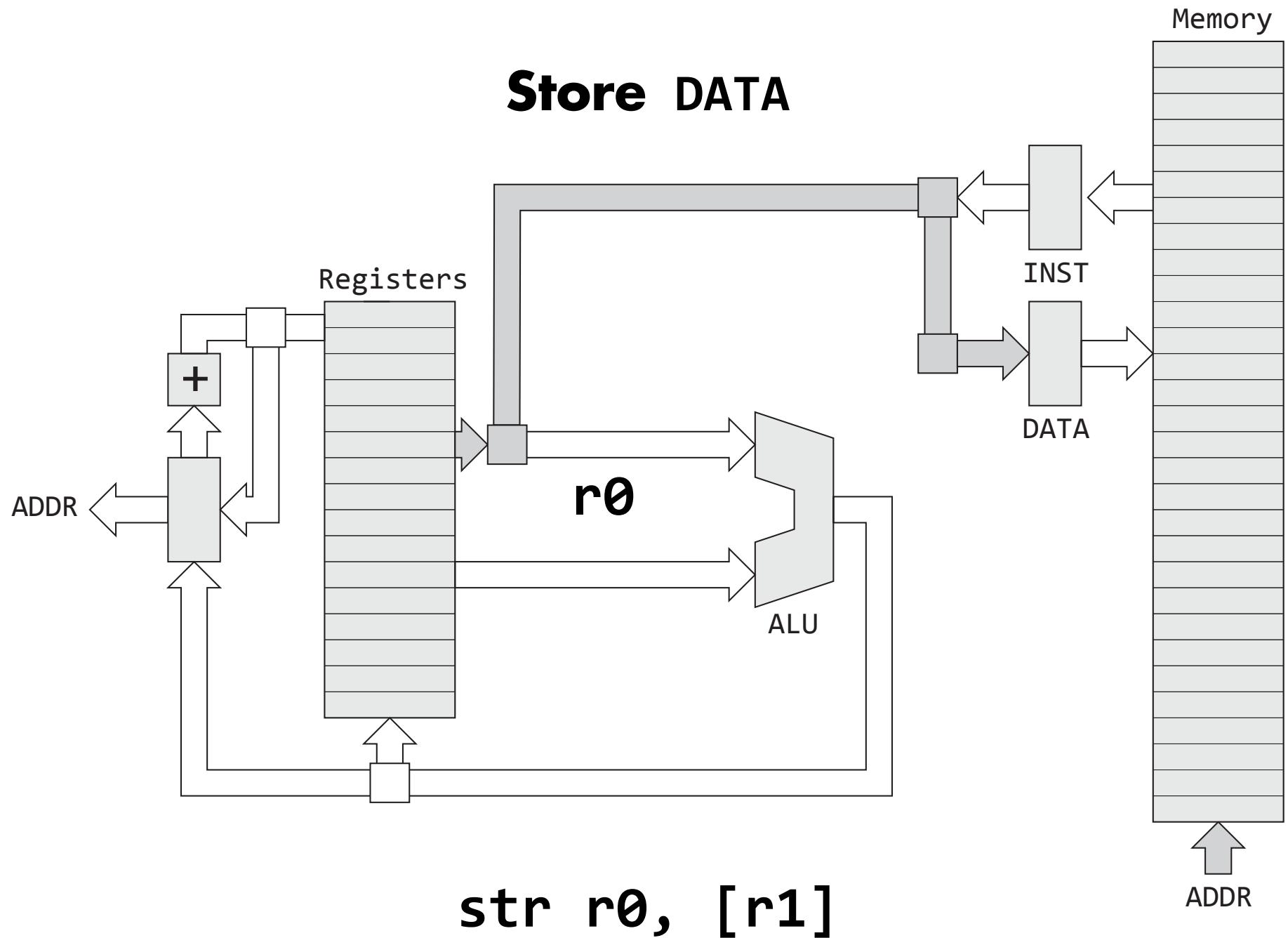
$$r0 = \text{mem}[r1]$$



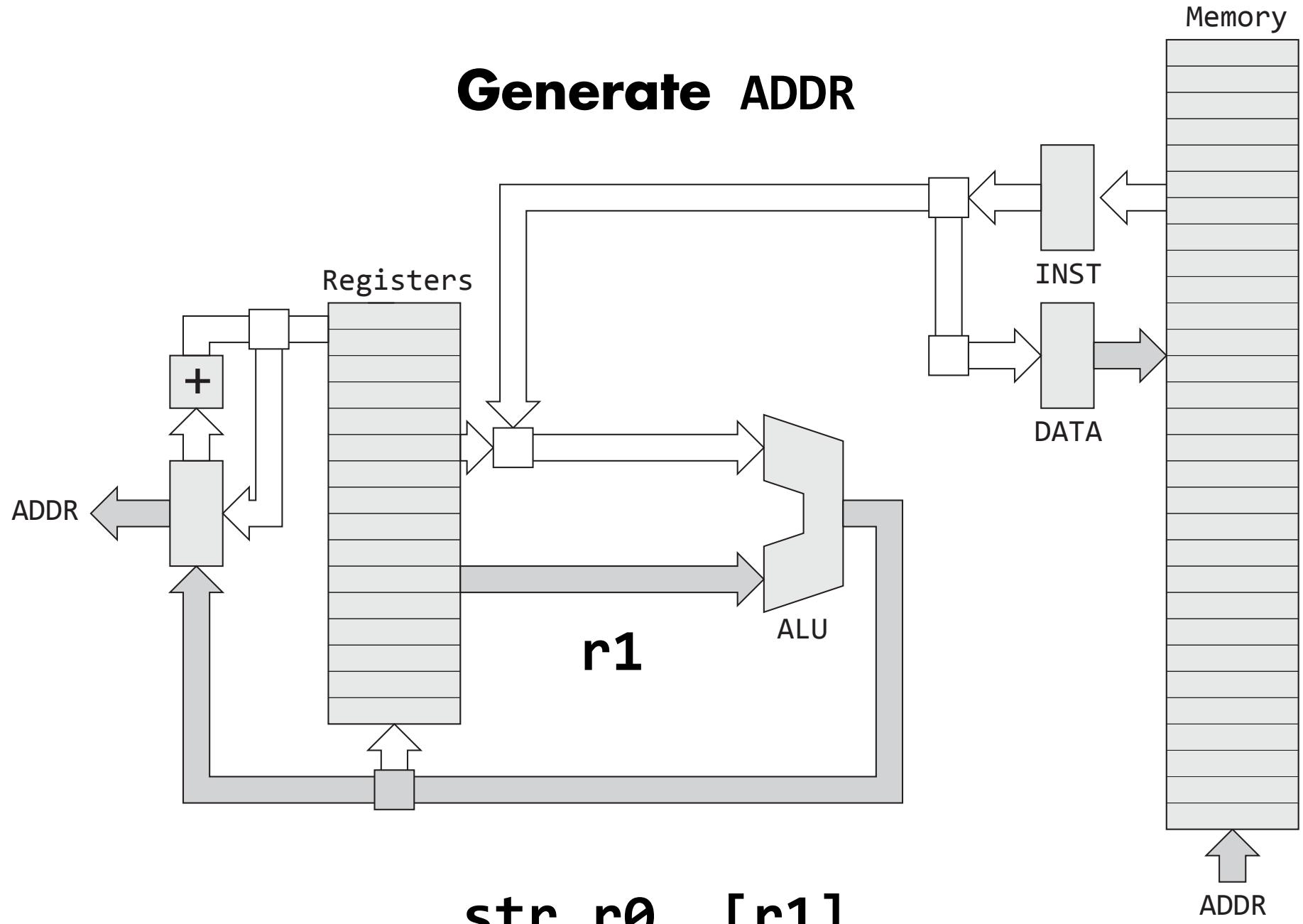
`ldr r0, [r1]`

$r0 = \text{mem}[r1]$ 

$\text{mem[r1]} = \text{r0}$



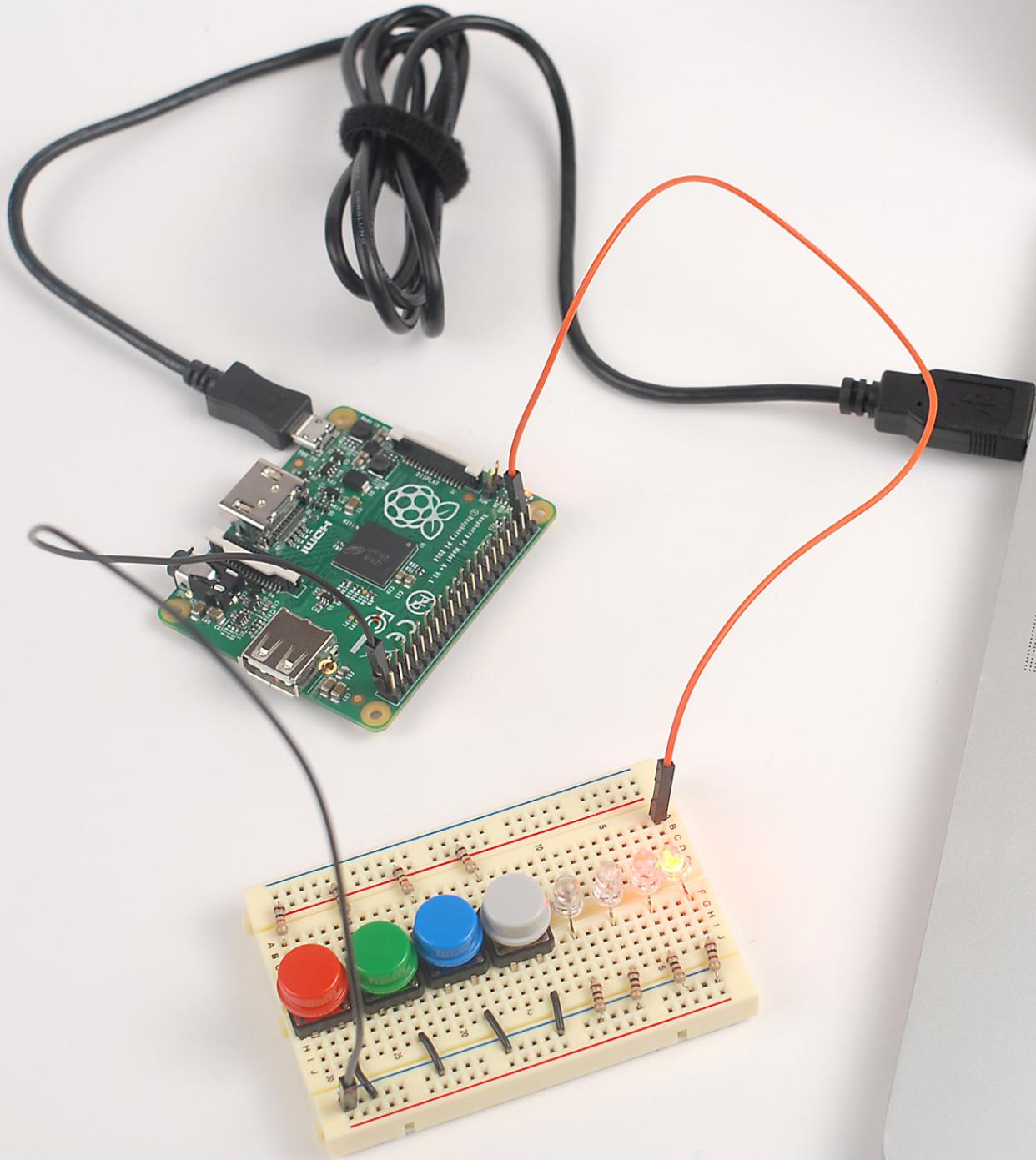
$\text{mem}[r_1] = r_0$



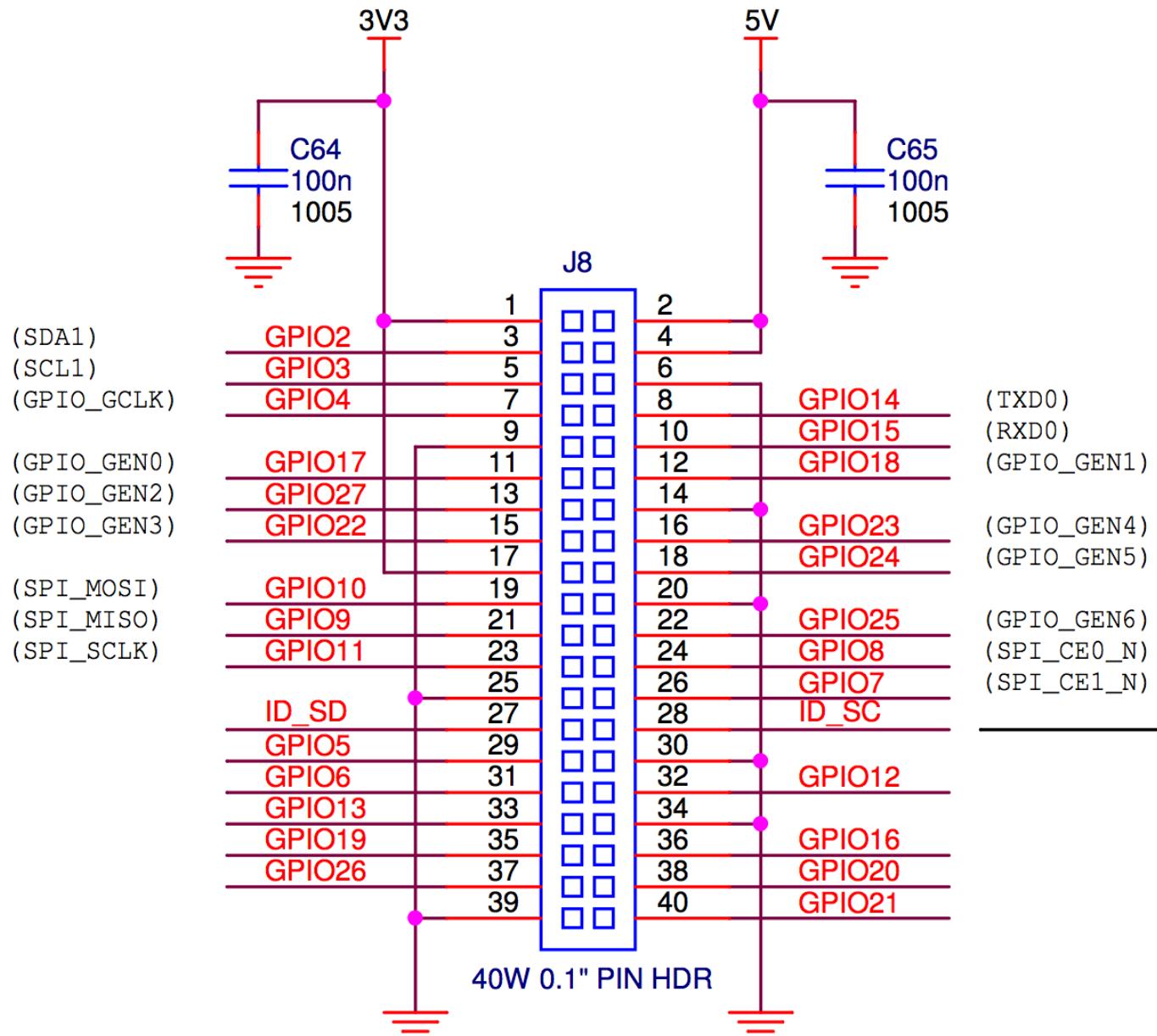
Conceptual Exercises

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

Turning on an LED

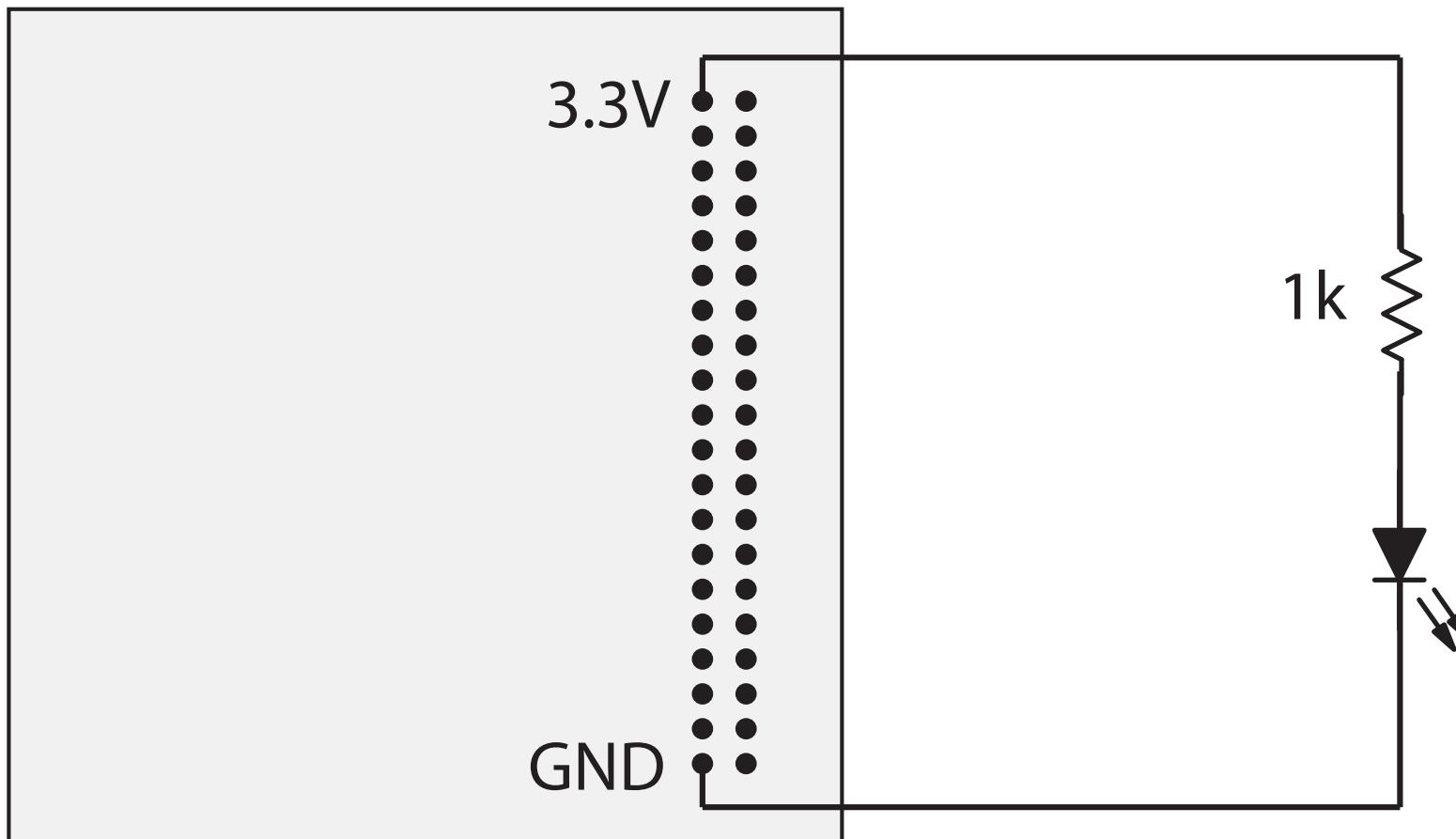


General-Purpose Input/Output (GPIO) Pins

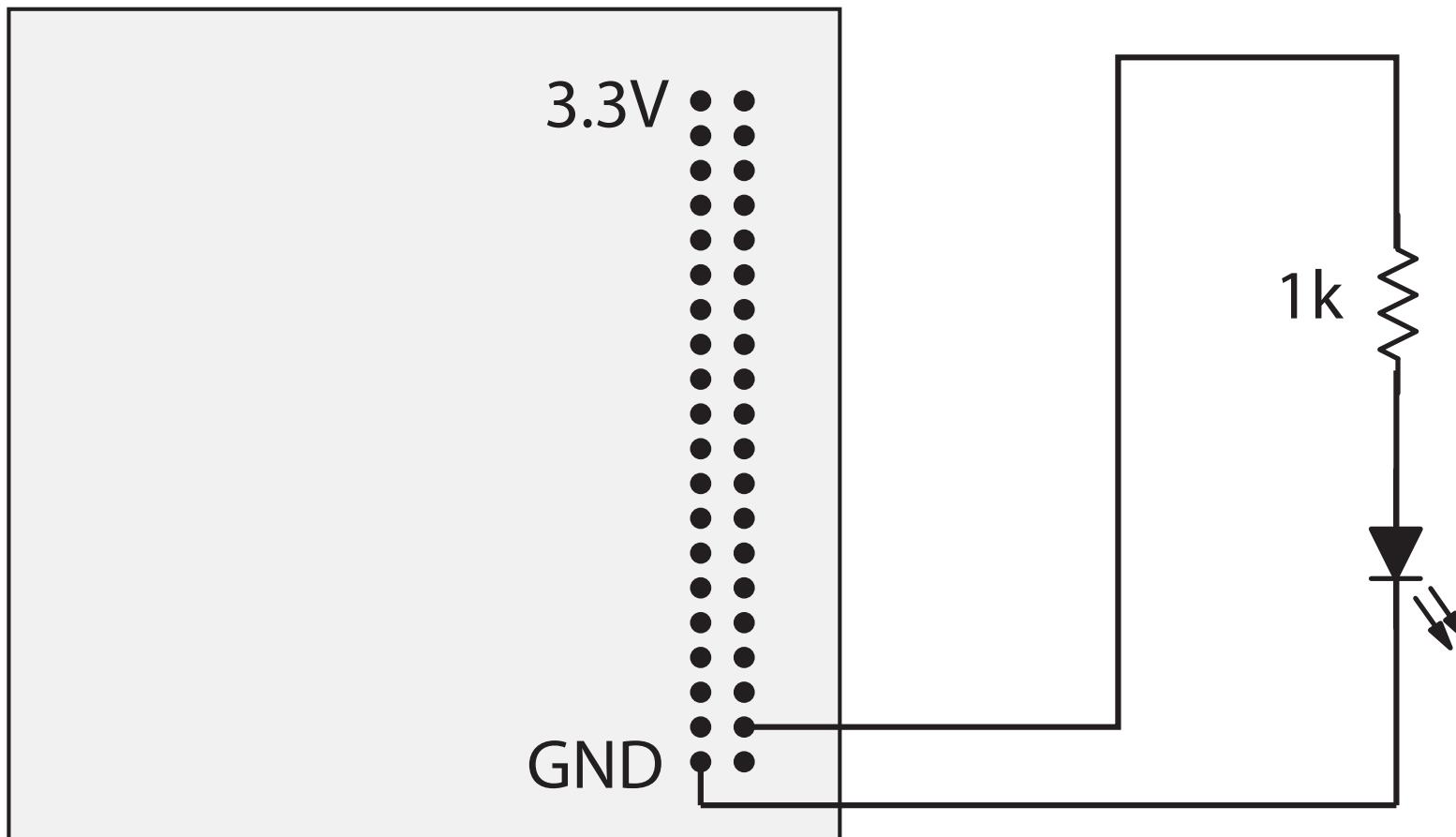


54 GPIO Pins

Powering an LED



Connect to GPIO 20

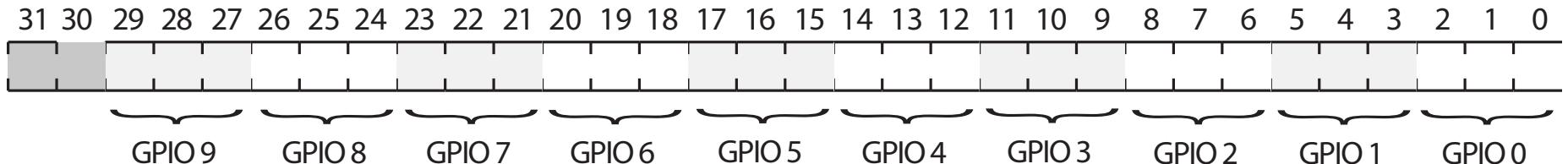


Logic 1 -> 3.3V (VCC)
Logic 0 -> 0.0V (GND)

Turn on an LED

1. Configure GPIO 20 for OUTPUT

GPIO Function Select Register



3 bits per GPIO pin

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

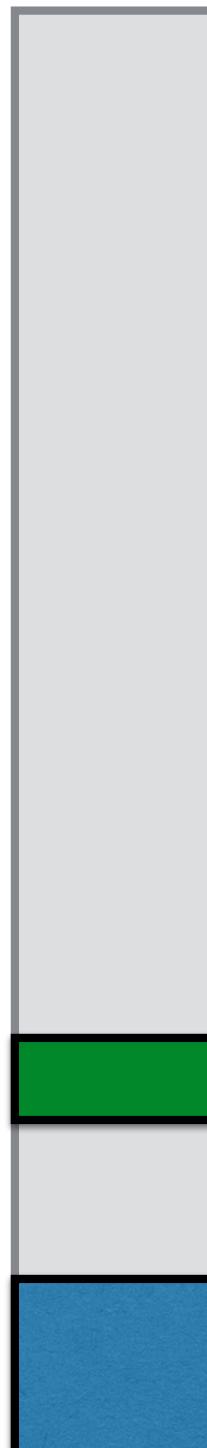
Max of 10 pins per 32-bit register

How to Control Peripherals?

**Peripheral Registers
are
"Mapped" to Memory Addresses**

**Memory-Mapped Input-Output
MMIO**

Memory Map



100000000_{16}
4 GB

Peripheral Registers

020000000_{16}
 010000000_{16}

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. 3-bits per GPIO pin, 54 GPIO pin
=> 6 GPIO function select registers

```
// Turn on an LED via GPIO 20  
  
// FSEL2=0x20200008 controls pins 20-29  
  
// load r0 with GPIO FSEL2 address  
ldr r0, =0x20200008  
  
// GPIO 20 function select is bits 0-2  
// The value of 1 indicates OUTPUT  
// load r1 with 1  
mov r1, #1  
  
// store value in r1 to address in r0  
str r1, [r0]
```

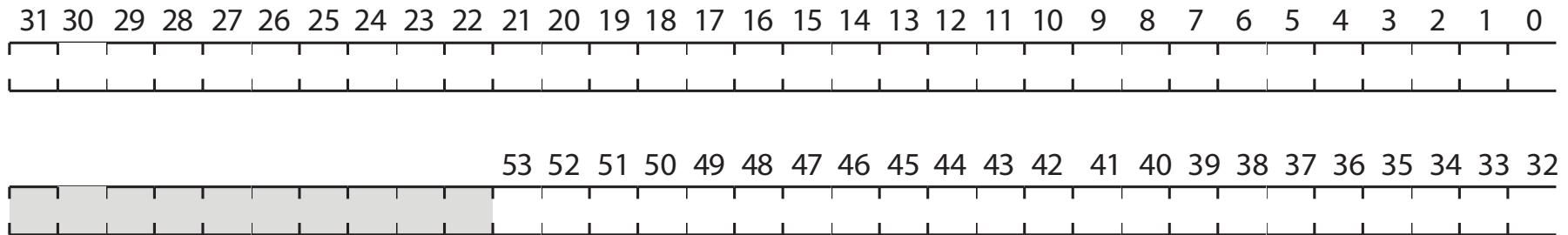
Turn on an LED

- 1. Configure GPIO 20 for OUTPUT**
- 2. Set GPIO 20 to 1 (HIGH, 3.3V)**

GPIO Function SET Register

20 20 00 1C : GPIO SET0 Register

20 20 00 20 : GPIO SET1 Register



Notes

- 1. 1 bit per GPIO pin**
- 2. 54 pins requires 2 registers**

...

```
// 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, #0x100000 // 0x100000 = 1 << 20
```

```
// store bit in GPIO SET0 register  
str r1, [r0]
```

```
// loop forever  
hang: b hang
```

Turn on an LED

- 1. Configure GPIO 20 for OUTPUT**
- 2. Set GPIO 20 to 1 (HIGH, 3.3V)**
- 3. Install program as kernel.img**

```
# What to do on your laptop
```

```
# Assemble language to machine code  
% arm-none-eabi-as on.s -o on.o
```

```
# Create binary from object file  
% arm-none-eabi-objcopy on.o -O binary  
on.bin
```

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

```
# What to do on your laptop
```

```
# Insert SD card - Volume mounts
```

```
% ls /Volumes/
```

```
BARE Macintosh HD
```

```
# 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.  
# Power LED (Red) should be on.  
#  
# Raspberry pi boots. ACT LED (Green)  
# flashes, and then is turned off  
#  
# LED connected to GPIO20 turns on!!  
#
```



Definitive References

BCM2865 peripherals document + errata

Raspberry Pi schematic

ARMv6 architecture reference manual

see Resources on cs107e.github.io