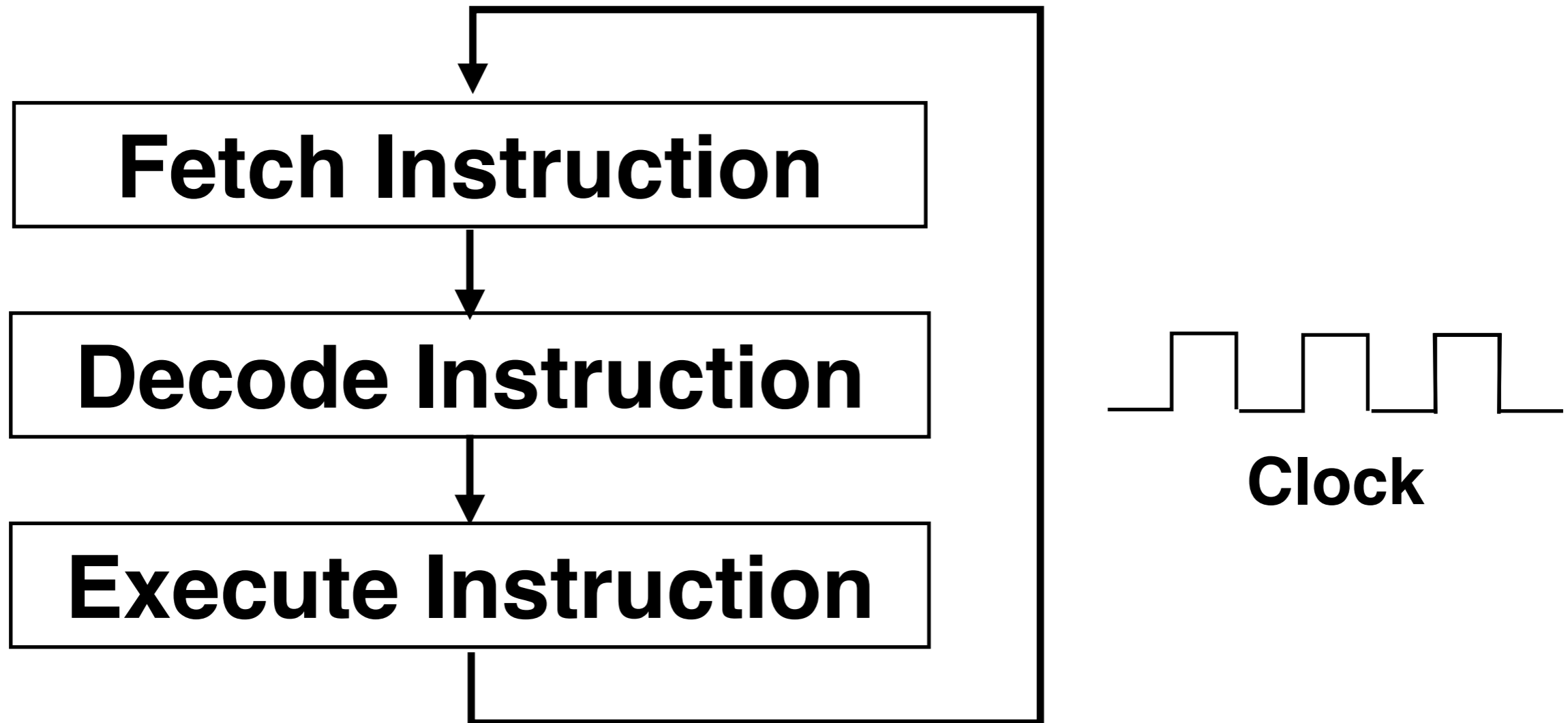


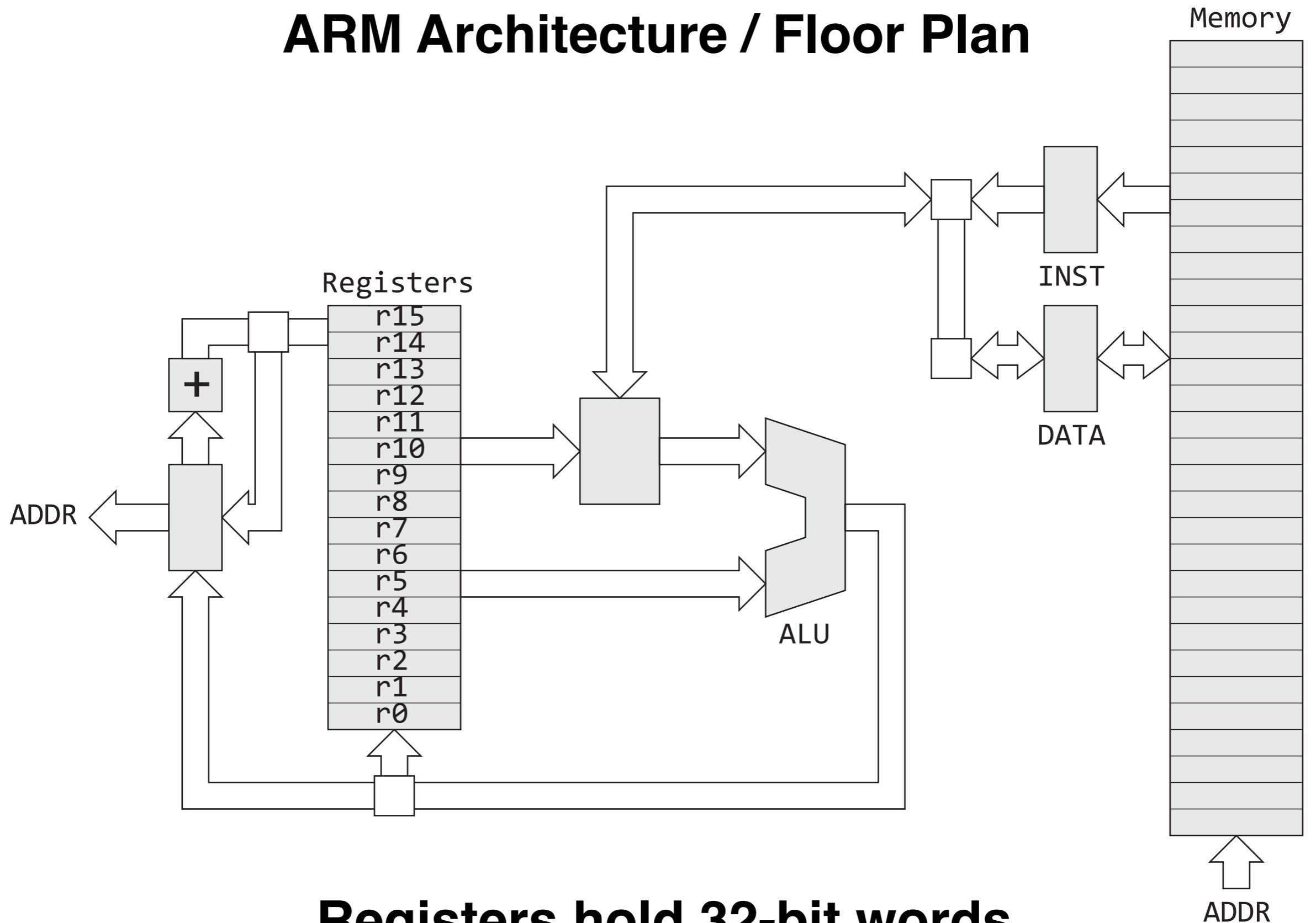
ARM Processor and Memory Architecture

Goal: Turn on an LED

Running a "Program"



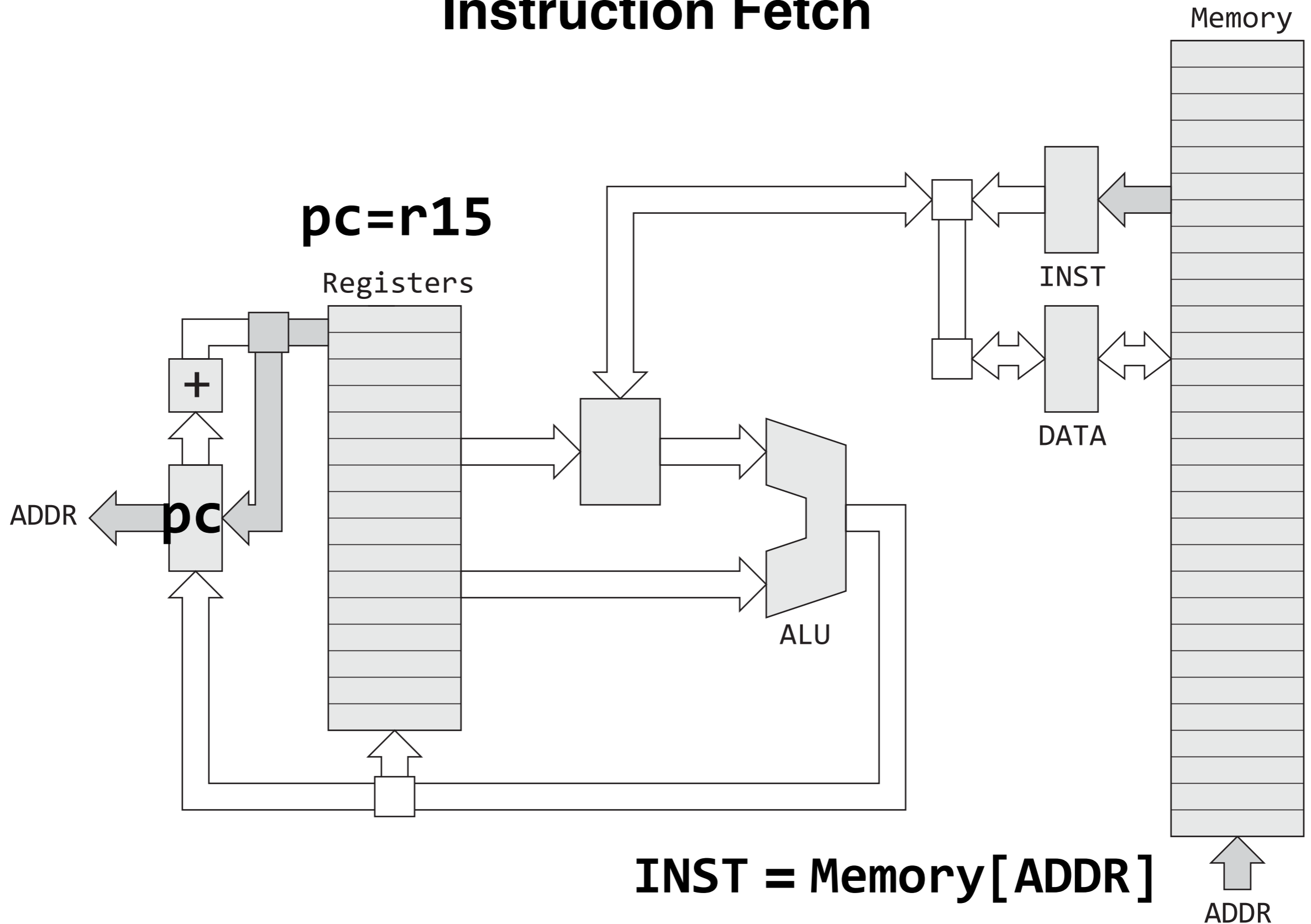
ARM Architecture / Floor Plan



Registers hold 32-bit words

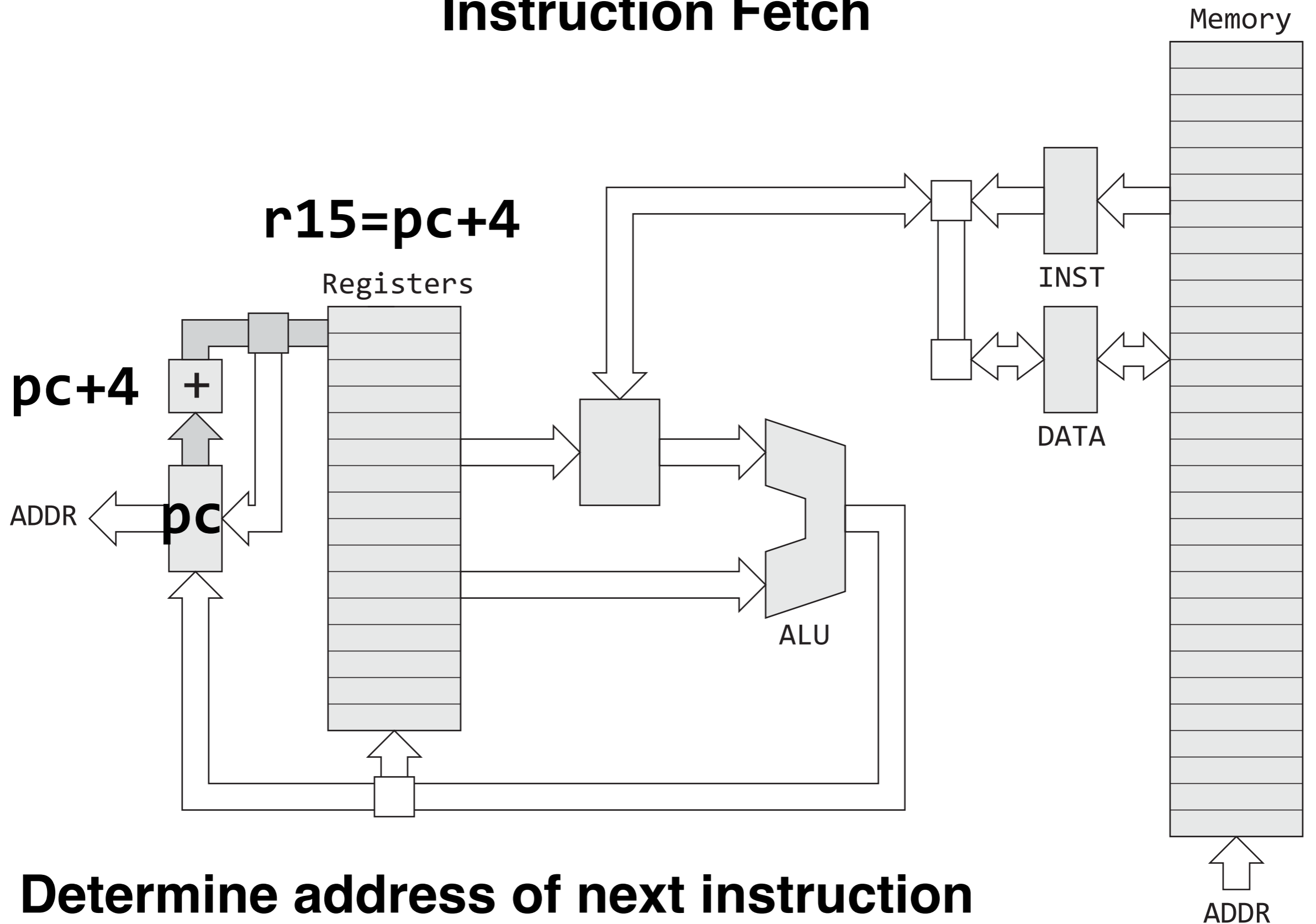
Arithmetic-Logic Unit (ALU) operates on 32-bit words

Instruction Fetch



Addresses and instructions are 32-bit words

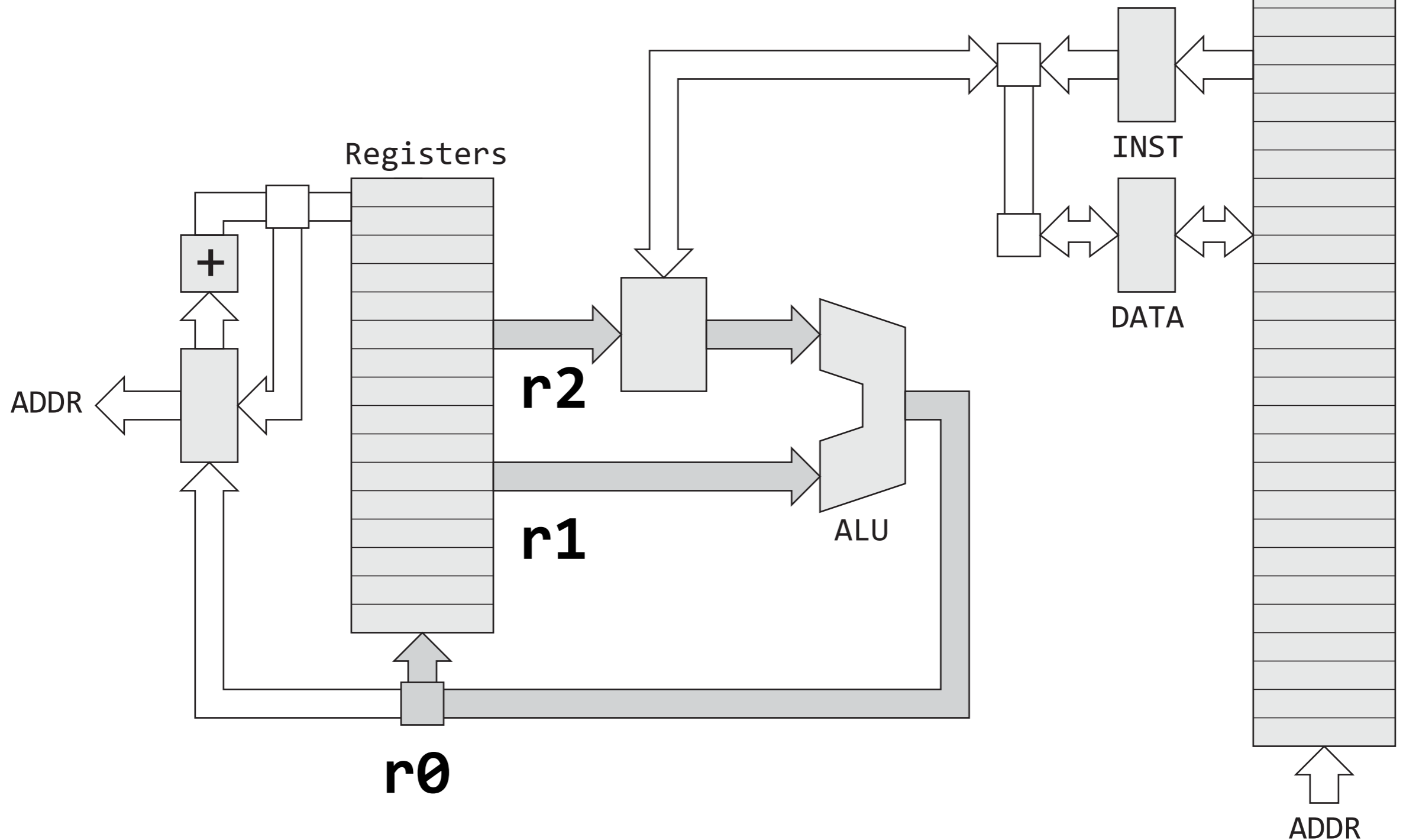
Instruction Fetch



Arithmetic-Logic Unit (ALU)

add r0, r1, r2

r0 = r1 + r2



ALU operates on 32-bit words

Add Instruction

Meaning (defined as math or C code)

$r0 = r1 + r2$

Assembly language (result is leftmost register)

add r0, r1, r2

Machine code (more on this later)

E0 81 00 02

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

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

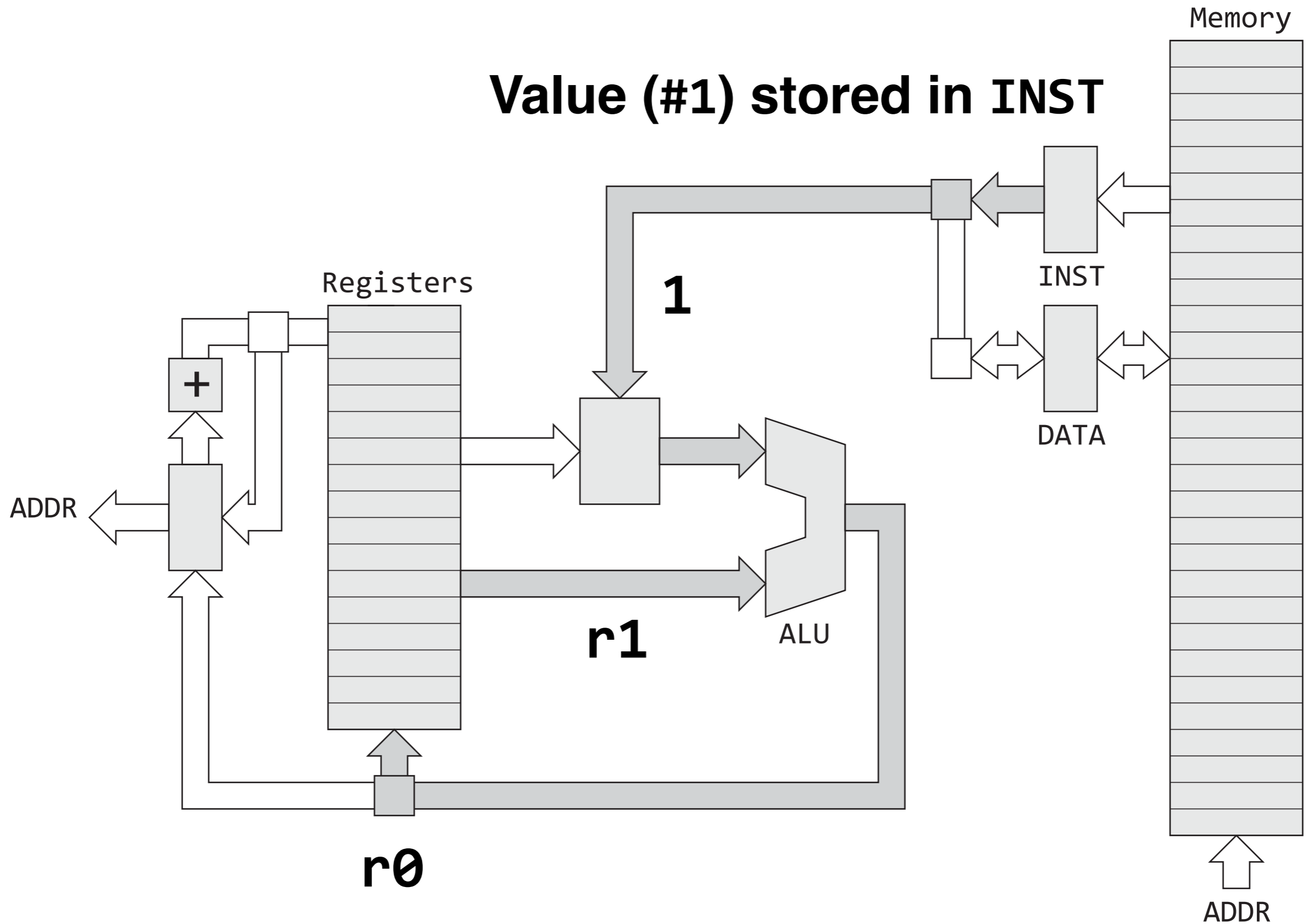
# Find size (in bytes)
% ls -l add.bin
-rw-r--r--+ 1 cgregg  staff  4 add.bin

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

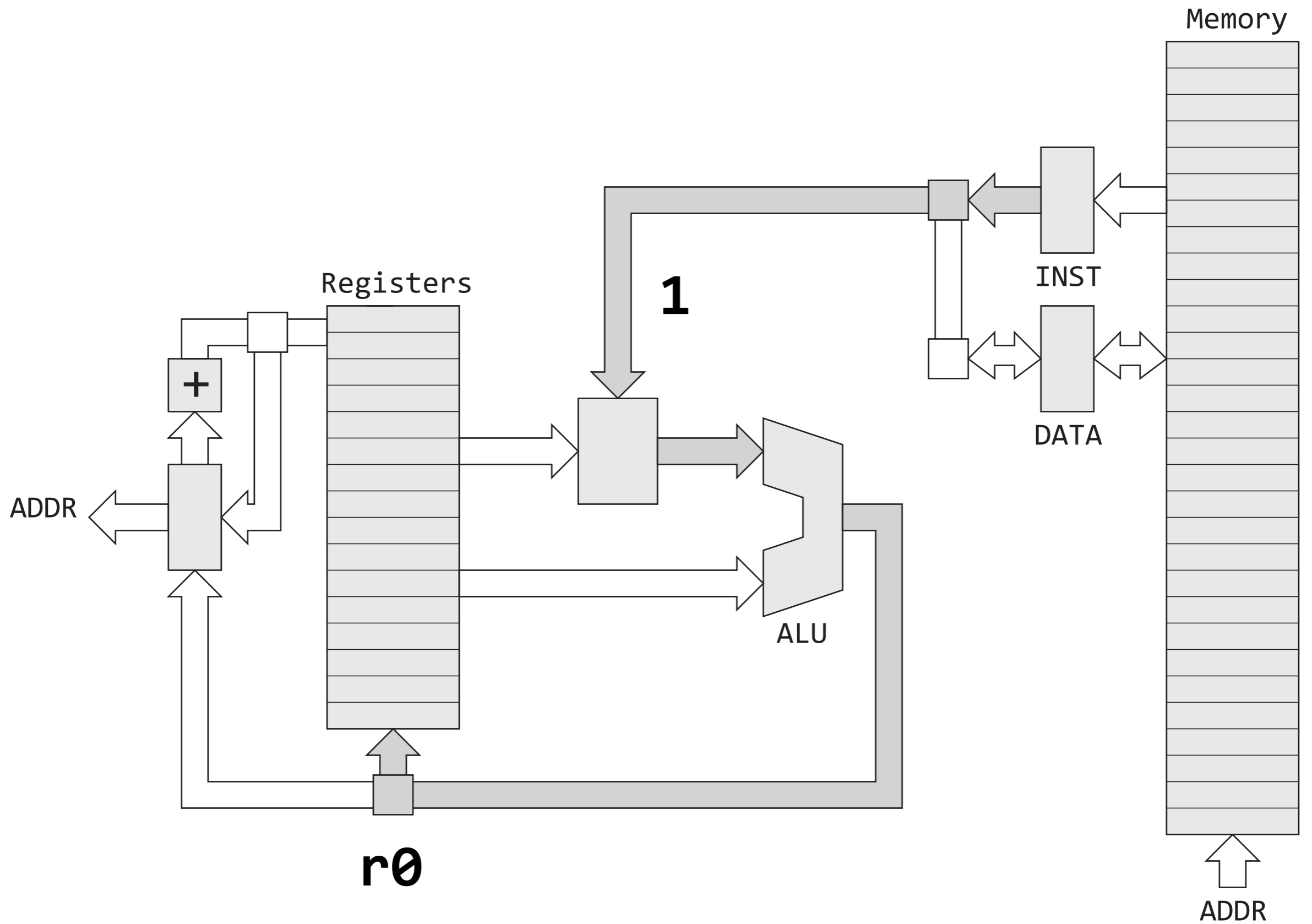
# Look at ARM Disassembly:
% objdump -d add.o

add.o:  file format elf32-littlearm
Disassembly of section .text:
00000000 <$a>:
        0: 02 00 81 e0  add    r0, r1, r2
```

Value (#1) stored in INST



`add r0, r1, #1`



`mov r0, #1`

VisUAL

untitled.S - [Unsaved] - VisUAL

New

Open

Save

Settings

Tools



▶ Emulation Running

Line 3
Issues 0

Execute

Reset

Step Backwards

Step Forwards

Reset to continue editing code

```
1  mov    r0, #1
2  mov    r1, #2
3  add    r2, r0, r1
4
```

R0	0x1	Dec	Bin	Hex
R1	0x2	Dec	Bin	Hex
R2	0x3	Dec	Bin	Hex
R3	0x0	Dec	Bin	Hex
R4	0x0	Dec	Bin	Hex
R5	0x0	Dec	Bin	Hex
R6	0x0	Dec	Bin	Hex
R7	0x0	Dec	Bin	Hex
R8	0x0	Dec	Bin	Hex
R9	0x0	Dec	Bin	Hex
R10	0x0	Dec	Bin	Hex
R11	0x0	Dec	Bin	Hex
R12	0x0	Dec	Bin	Hex
R13	0xFF000000	Dec	Bin	Hex
LR	0x0	Dec	Bin	Hex
PC	0x10	Dec	Bin	Hex

🕒 Clock Cycles

Current Instruction: 1 Total: 3

CSPR Status Bits (NZCV)

0 0 0 0

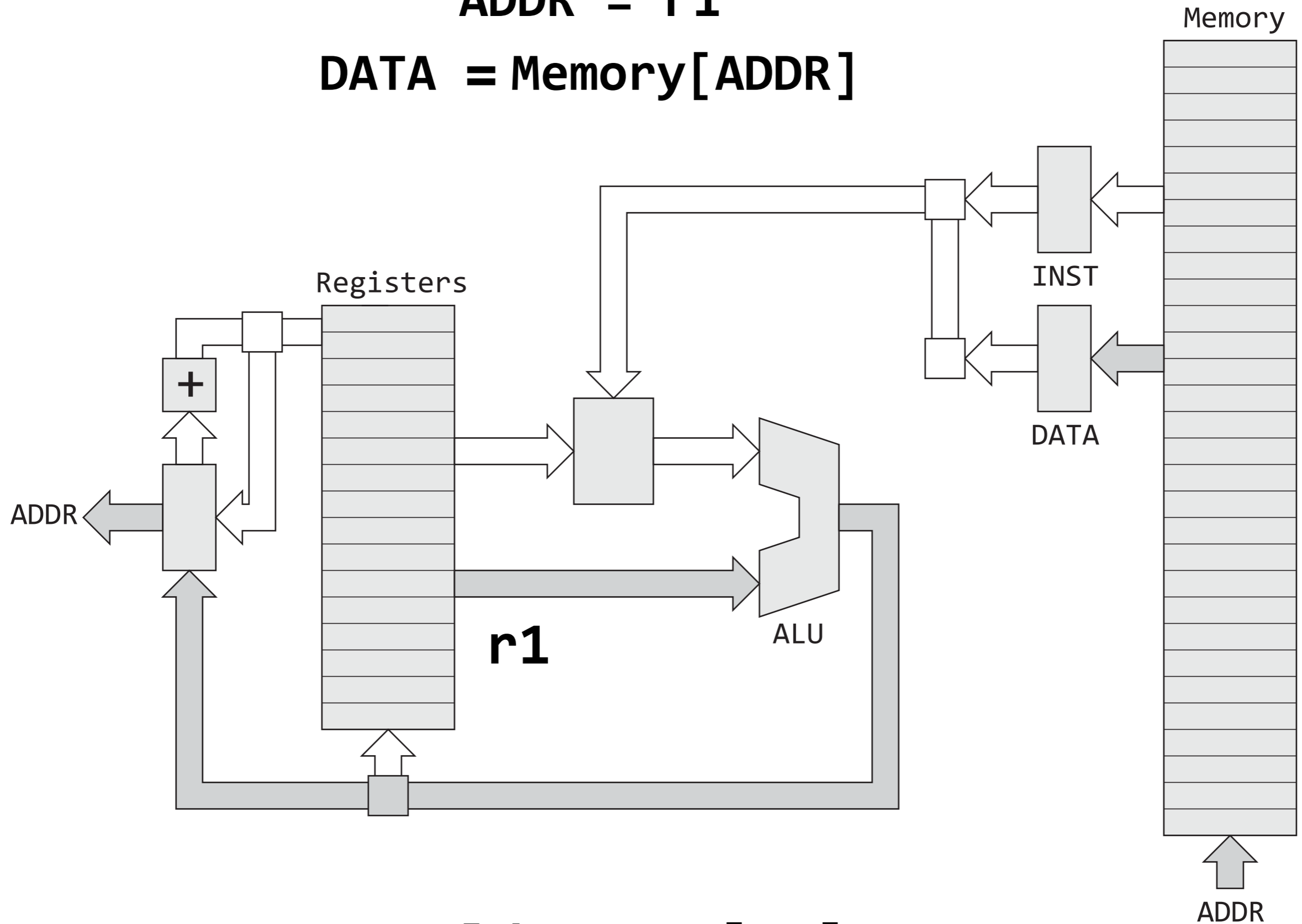
Conceptual Questions

- 1. Suppose your program starts at $0x8000$, what assembly language program will jump to and start executing instructions at that location.**
- 2. All instructions are 32-bits. Can you mov any 32-bit constant value to a register using the mov instruction?**
- 3. What are some advantages of always using 32-bits for instructions, addresses, and data?**

Load and Store Instructions

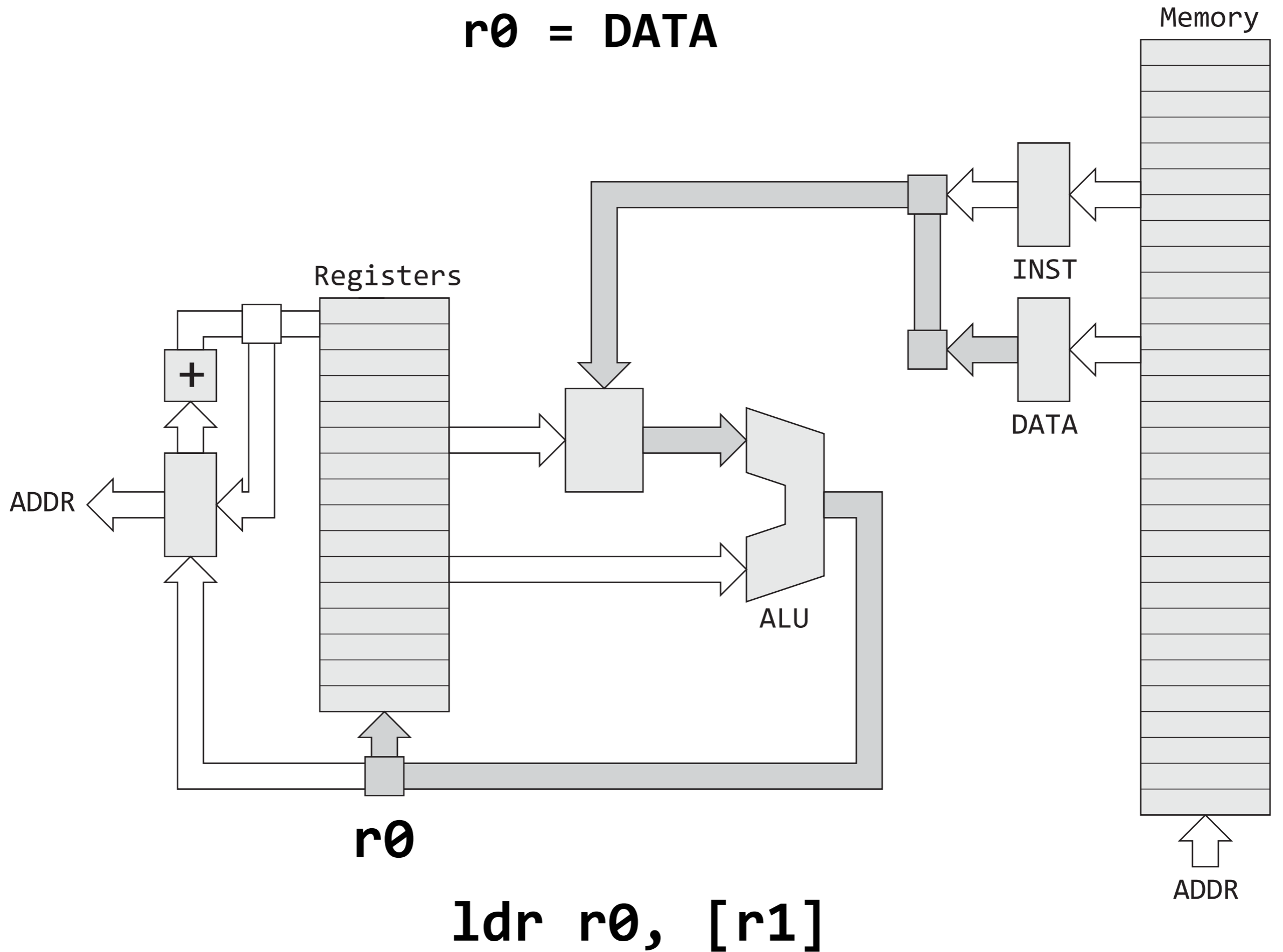
ADDR = r1

DATA = Memory[ADDR]

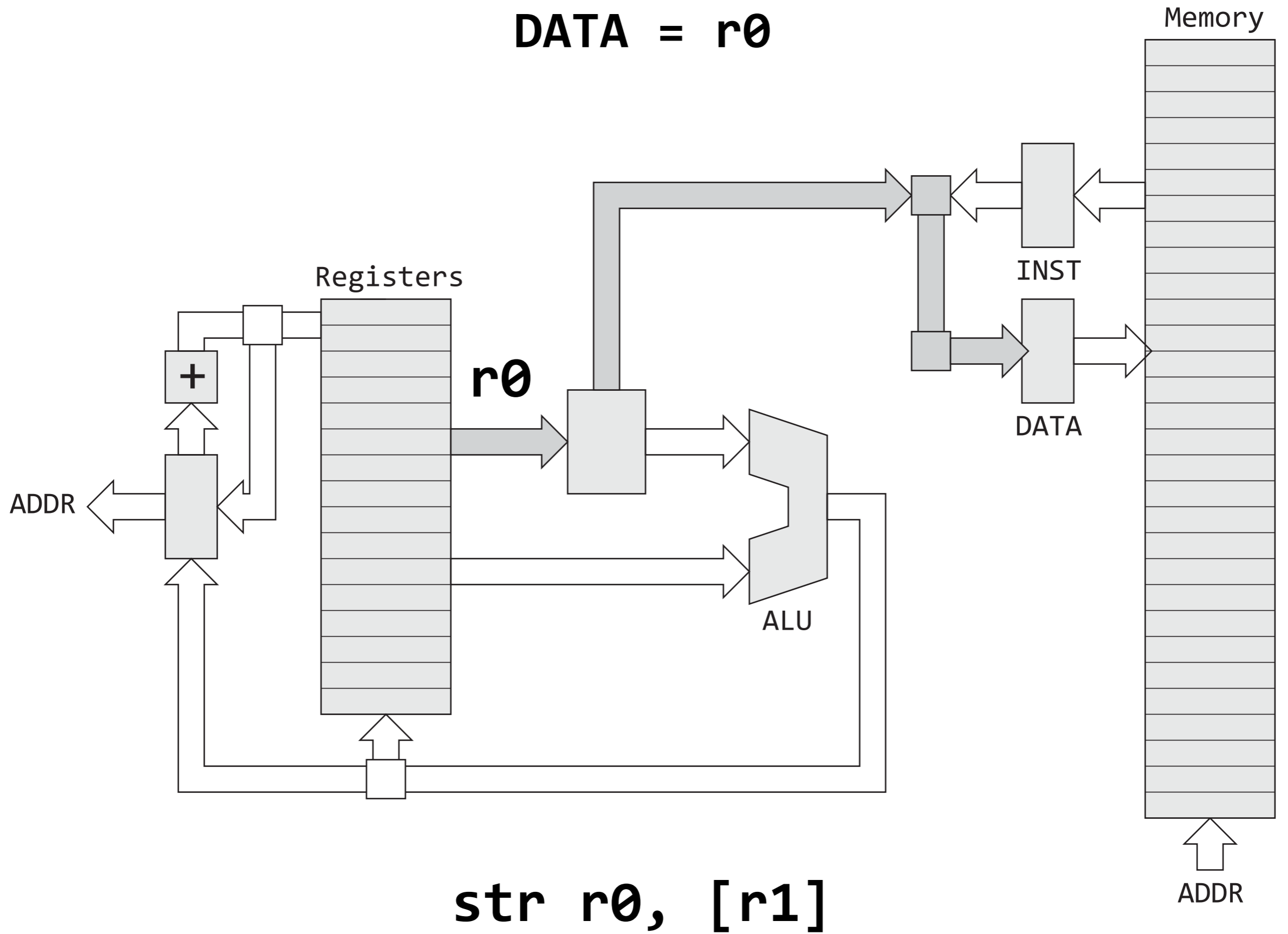


ldr r0, [r1]

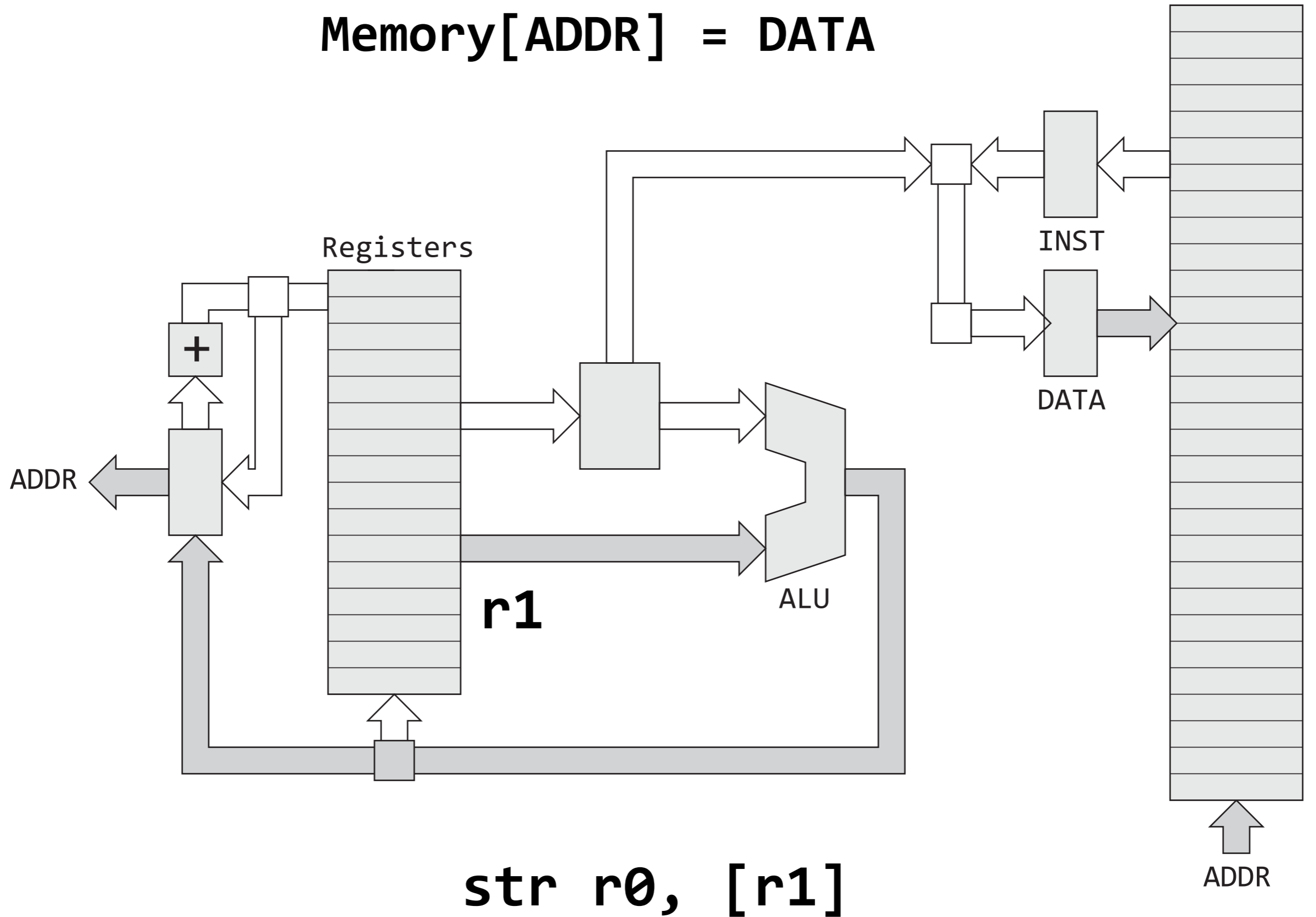
r0 = DATA



DATA = r0



ADDR = r1
Memory[ADDR] = DATA



New Open Save Settings Tools Emulation Running Line 4 Issues 0 Execute Reset Step Backwards Step Forwards

Reset to continue editing code

```

1  ldr    r0, =0x100
2  mov    r1, #0xff
3  str    r1, [r0]
4  ldr    r2, [r0]
5

```

Pointer Memory

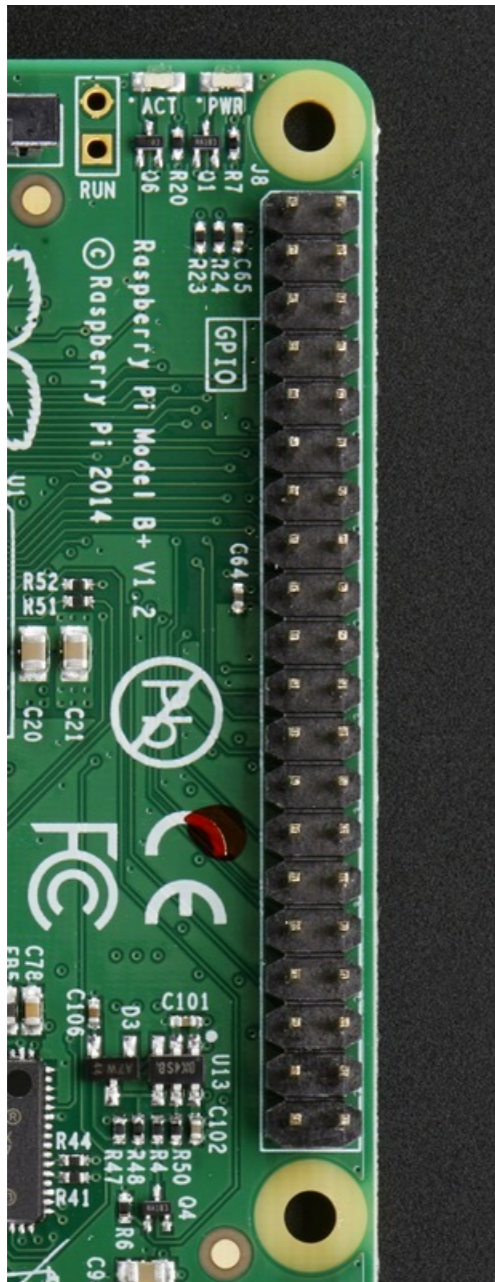
R0	0x100	Dec	Bin	Hex
R1	0xFF	Dec	Bin	Hex
R2	0xFF	Dec	Bin	Hex
R3	0x0	Dec	Bin	Hex
R4	0x0	Dec	Bin	Hex
R5	0x0	Dec	Bin	Hex
R6	0x0	Dec	Bin	Hex
R7	0x0	Dec	Bin	Hex
R8	0x0	Dec	Bin	Hex
R9	0x0	Dec	Bin	Hex
R10	0x0	Dec	Bin	Hex
R11	0x0	Dec	Bin	Hex
R12	0x0	Dec	Bin	Hex
R13	0xFF000000	Dec	Bin	Hex
LR	0x0	Dec	Bin	Hex
PC	0x14	Dec	Bin	Hex

Clock Cycles Current Instruction: 2 Total: 6

CSPR Status Bits (NZCV) 0 0 0 0

Turning on an LED

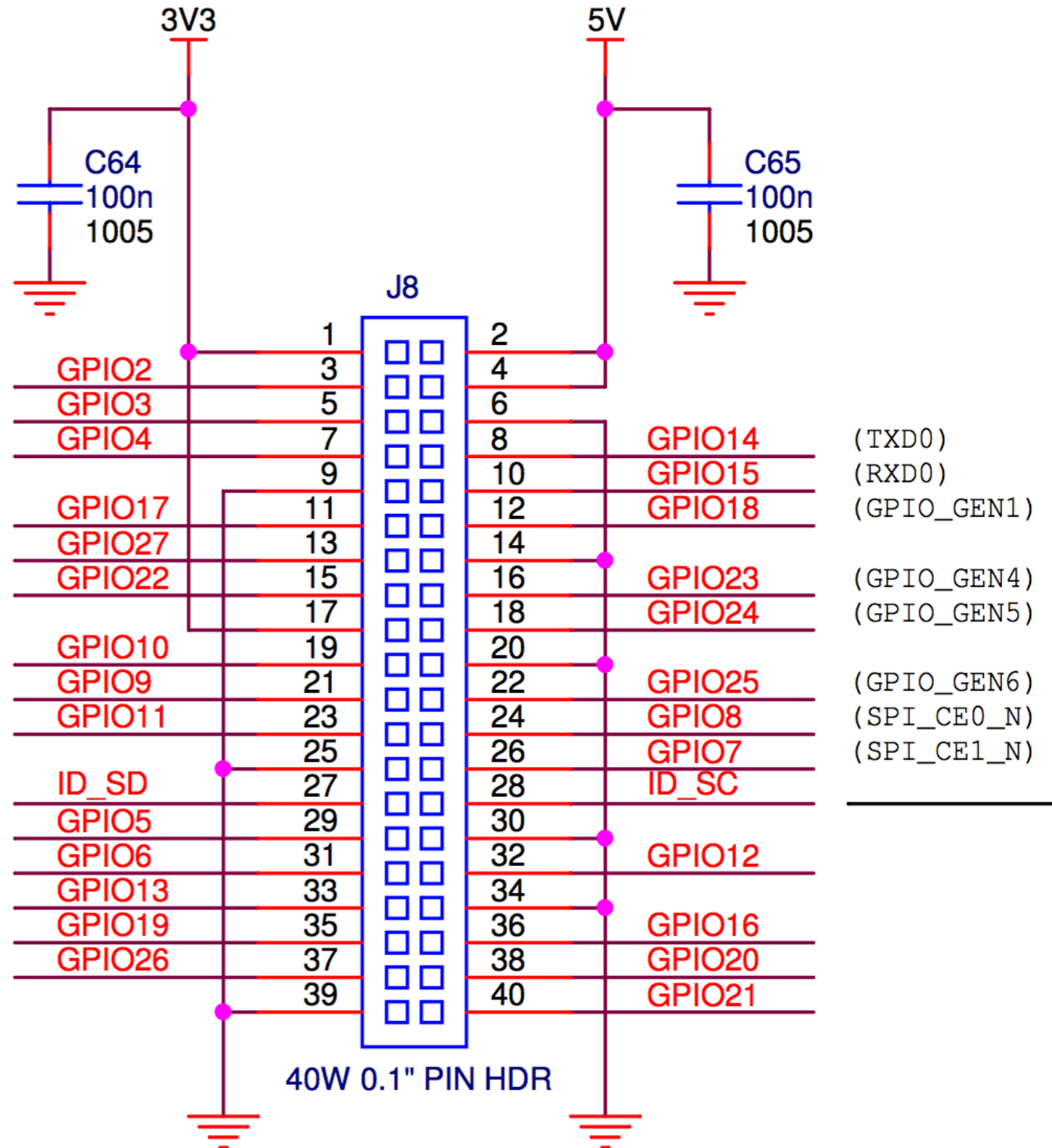
General-Purpose Input/Output (GPIO) Pins



(SDA1)
 (SCL1)
 (GPIO_GCLK)

 (GPIO_GEN0)
 (GPIO_GEN2)
 (GPIO_GEN3)

 (SPI_MOSI)
 (SPI_MISO)
 (SPI_SCLK)



54 GPIO Pins

**Computers have Peripherals
that Interface to the World**

GPIO Pins are Peripherals

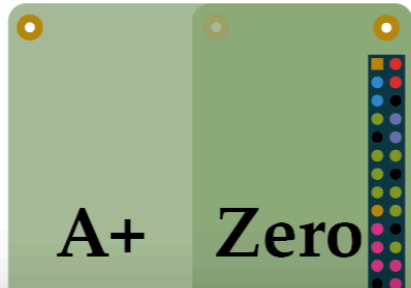


3v3 Power	1	2	5v Power
GPIO 2 (I2C1 SDA)	3	4	5v Power
GPIO 3 (I2C1 SCL)	5	6	Ground
GPIO 4 (GPCLK0)	7	8	GPIO 14 (UART TX)
Ground	9	10	GPIO 15 (UART RX)
GPIO 17	11	12	GPIO 18 (PCM CLK)
GPIO 27	13	14	Ground
GPIO 22	15	16	GPIO 23
3v3 Power	17	18	GPIO 24
GPIO 10 (SPI0 MOSI)	19	20	Ground
GPIO 9 (SPI0 MISO)	21	22	GPIO 25
GPIO 11 (SPI0 SCLK)	23	24	GPIO 8 (SPI0 CE0)
Ground	25	26	GPIO 7 (SPI0 CE1)
GPIO 0 (EEPROM SDA)	27	28	GPIO 1 (EEPROM SCL)
GPIO 5	29	30	Ground
GPIO 6	31	32	GPIO 12 (PWM0)
GPIO 13 (PWM1)	33	34	Ground
GPIO 19 (PCM FS)	35	36	GPIO 16
GPIO 26	37	38	GPIO 20 (PCM DIN)
Ground	39	40	GPIO 21 (PCM DOUT)

Legend

Orientate your Pi with the GPIO on the right and the HDMI port(s) on the left.

- GPIO (General Purpose IO)
- SPI (Serial Peripheral Interface)



Navigation menu: 5v Power, SDIO, JTAG, 3v3 Power, UART, DPI, PCM, 1-WIRE, WiringPi, GPCLK, Ground, I2C, PWM, SPI

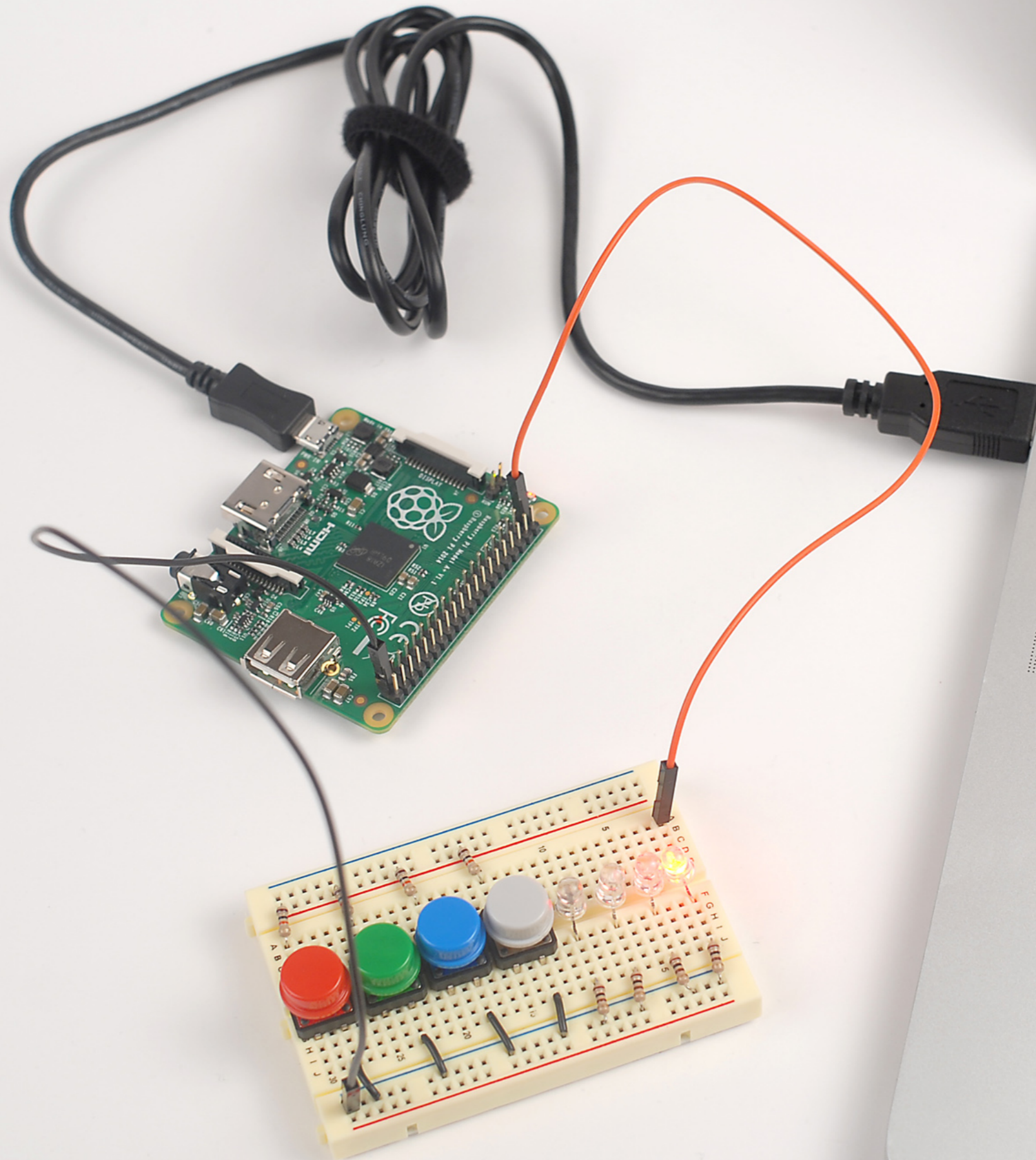
Browse pinouts for HATs, pHATs and add-ons »

GPIO 20 (PCM Data-In)

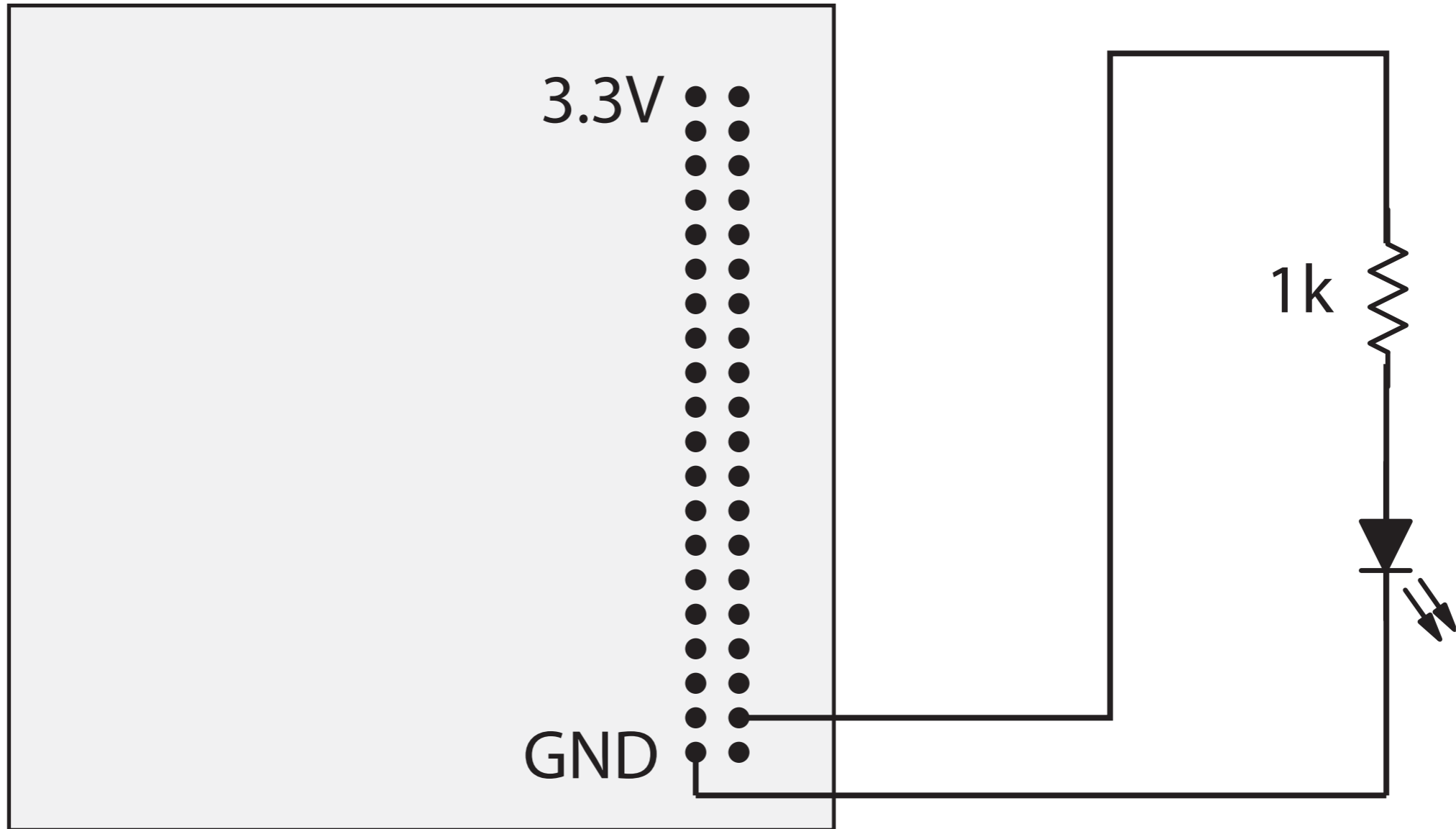
Alt0	Alt1	Alt2	Alt3	Alt4	Alt5
PCM DIN	SPI0 MISO	DPI D16	I2CSL MISO	SPI1 MOSI	GPCLK0

- Physical/Board pin 38
- GPIO/BCM pin 20
- Wiring Pi pin 28

GPIO 20 is used by PCM to as a data input from an audio device such as an I2C microphone.



Connect LED to GPIO 20



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

GPIO Pins are called Peripherals

**Peripherals are Controlled
by Special Registers**

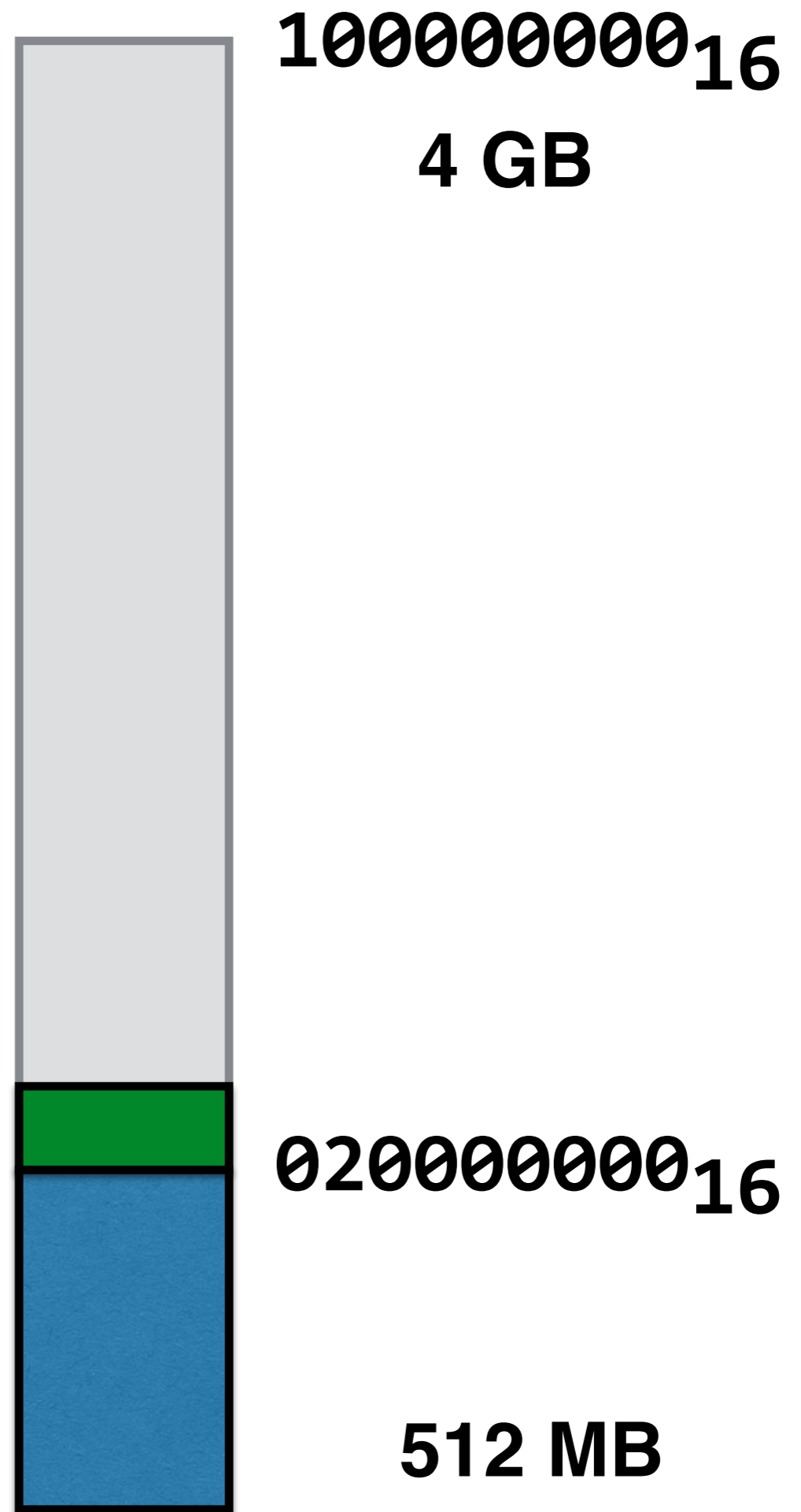
"Peripheral Registers"

Memory Map

Peripheral registers
are mapped
into address space

Memory-Mapped IO
(MMIO)

MMIO space is above
physical memory



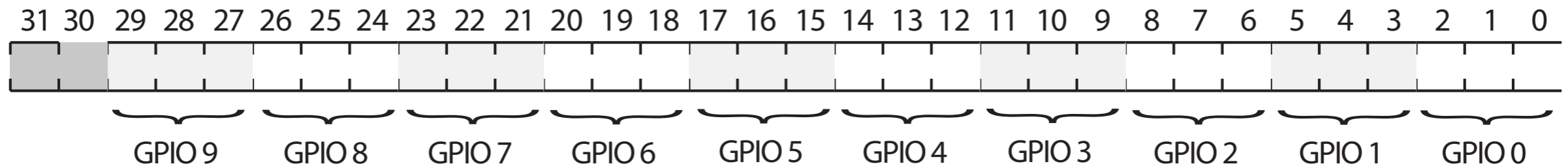
General-Purpose IO Function

**GPIO Pins can be configured to be
INPUT, OUTPUT, or ALT0-5**

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

3 bits required to select function

GPIO Function Select Register



Function is INPUT, OUTPUT, or ALT0-5

8 functions requires 3 bits to specify

10 pins per 32-bit register (2 wasted bits)

54 GPIOs pins requires 6 registers

GPIO Function Select Registers Addresses

Address	Field Name	Description	Size	Read/Write
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	-	-

Watch out for ...

Manual says: 0x7E200000

Replace 7E with 20: 0x20200000

```
// Set GPIO20 to be an output
```

```
// FSEL2 = 0x20200008
```

```
mov r0, #0x20000000
```

```
orr r0, #0x00200000
```

```
orr r0, #0x00000008
```

```
mov r1, #1 // 1 indicates OUTPUT
```

```
str r1, [r0] // store 1 to 0x20200008
```

Note this also makes GPIO 21-29 into inputs



GPIO Pin Output Set Registers (GPSETn)

SYNOPSIS The output set registers are used to set a GPIO pin. The SET{n} field defines the respective GPIO pin to set, writing a “0” to the field has no effect. If the GPIO pin is being used as an input (by default) then the value in the SET{n} field is ignored. However, if the pin is subsequently defined as an output then the bit will be set according to the last set/clear operation. Separating the set and clear functions removes the need for read-modify-write operations

Bit(s)	Field Name	Description	Type	Reset
31-0	SETn (n=0..31)	0 = No effect 1 = Set GPIO pin <i>n</i>	R/W	0

Table 6-8 – GPIO Output Set Register 0

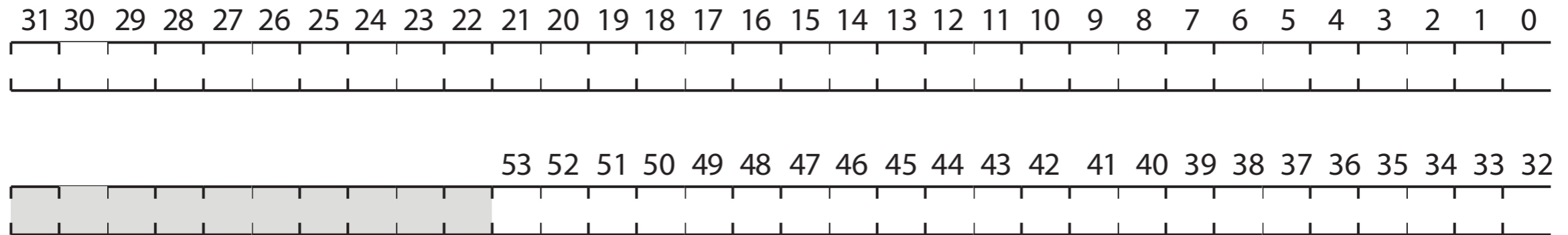
Bit(s)	Field Name	Description	Type	Reset
31-22	-	Reserved	R	0
21-0	SETn (n=32..53)	0 = No effect 1 = Set GPIO pin <i>n</i> .	R/W	0

Table 6-9 – GPIO Output Set Register 1

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**

...

```
// SET0 = 0x2020001c  
mov r0, #0x20000000  
orr r0, #0x00200000  
orr r0, #0x0000001c  
mov r1, #1  
lsl r1, #20 // bit 20 = 1<<20  
str r1, [r0] // store 1<<20 to 0x2020001c
```

...

```
// SET0 = 0x2020001c
mov r0, #0x20000000
orr r0, #0x00200000
orr r0, #0x0000001c
mov r1, #1
lsl r1, #20 // bit 20 = 1<<20
str r1, [r0] // store 1<<20 to 0x2020001c

// loop forever
loop:
b loop
```

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**

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!!  
#
```



Concepts

Memory stores both instructions and data

Bits, bytes, and words; bitwise operations

Different types of ARM instructions

- **ALU**
- **Loads and Stores**
- **Branches**

GPIOs, peripheral registers, and MMIO