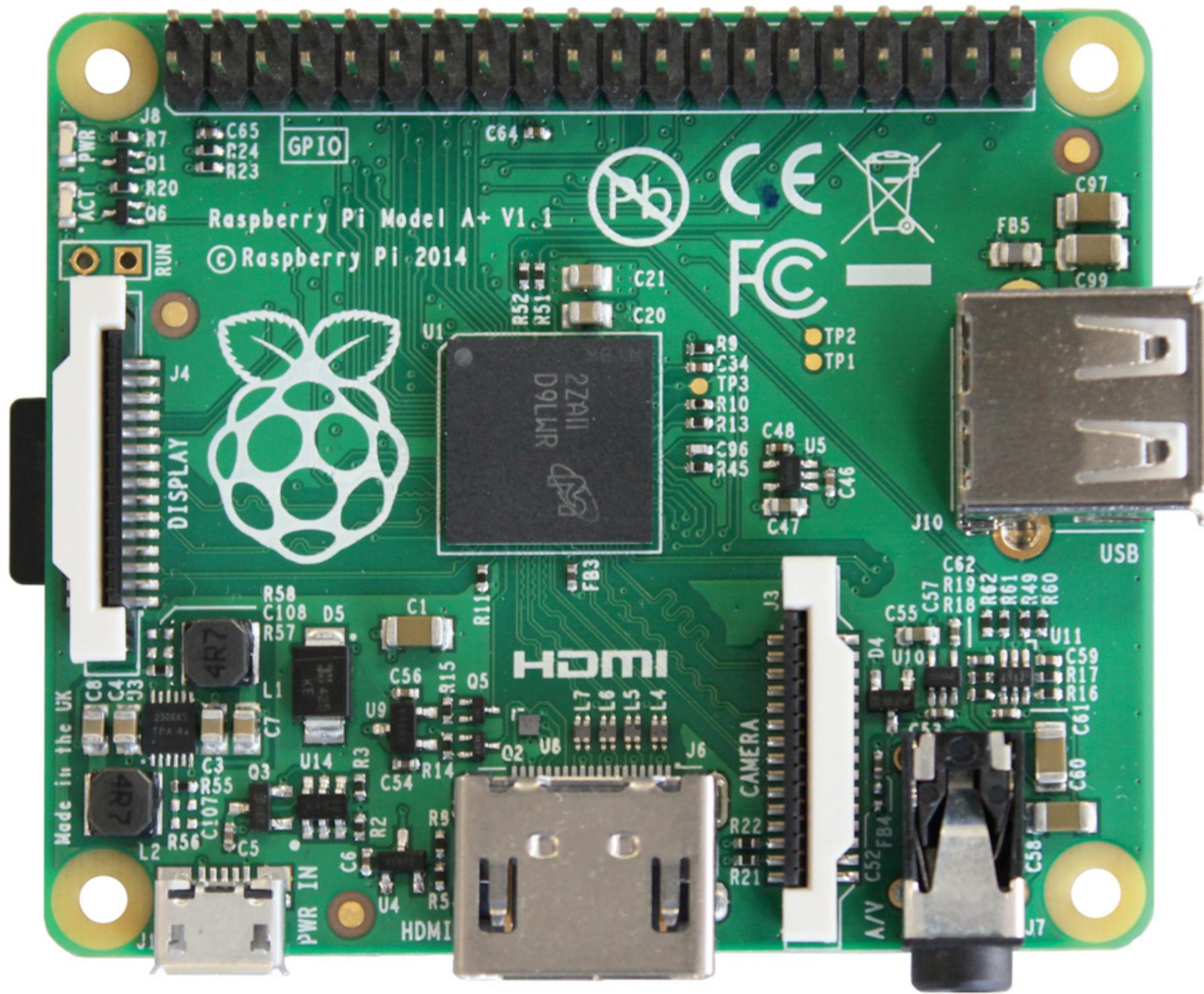


ARM Processor and Memory Architecture

Goal: Turn on an LED



ARUKCE MC1
V-OF3
1439 1-6



MICRO SD CARD



J9

C66

R12	C10	C17	C36	C69	C37	R25
C50	C9	F8	C49	C18	C12	C35
C51	C9	F8	C49	C14	C12	C30
C52	C9	F8	C49	C13	C12	C31
C45	C29					

PP21 C23 C28 L3
R31 C26 C42 C3
C19 C41 C43 C67
C44 C16 C10 C32
C25 C27 C37 X1
C10 C11 C38 C24
C15 PP15 PP18
PP14 PP19 PP16
PP17 PP5

J551 N
PP31 TDI
PP32 TDO
PP29 TMS
PP34 TCK
PP33 GND
PP31



PP40
PP39
PP38
PP37
J5

C24
R1m
PP8 PP4
F1 3602
PP7 PP1
PP2
PP3

PP22

PP35

PP23

PP27

PP26

PP25

PP24

PP40

PP39

PP38

PP37

PP30

PP31

PP32

PP33

PP34

PP35

PP36

PP37

PP38

PP39

PP40

PP41

PP42

PP43

PP44

PP45

PP46

PP47

PP48

PP49

PP50

PP51

PP52

PP53

PP54

PP55

PP56

PP57

PP58

PP59

PP60

PP61

PP62

PP63

PP64

PP65

PP66

PP67

PP68

PP69

PP70

PP71

PP72

PP73

PP74

PP75

PP76

PP77

PP78

PP79

PP80

PP81

PP82

PP83

PP84

PP85

PP86

PP87

PP88

PP89

PP90

PP91

PP92

PP93

PP94

PP95

PP96

PP97

PP98

PP99

PP100

PP101

PP102

PP103

PP104

PP105

PP106

PP107

PP108

PP109

PP110

PP111

PP112

PP113

PP114

PP115

PP116

PP117

PP118

PP119

PP120

PP121

PP122

PP123

PP124

PP125

PP126

PP127

PP128

PP129

PP130

PP131

PP132

PP133

PP134

PP135

PP136

PP137

PP138

PP139

PP140

PP141

PP142

PP143

PP144

PP145

PP146

PP147

PP148

PP149

PP150

PP151

PP152

PP153

PP154

PP155

PP156

PP157

PP158

PP159

PP160

PP161

PP162

PP163

PP164

PP165

PP166

PP167

PP168

PP169

PP170

PP171

PP172

PP173

PP174

PP175

PP176

PP177

PP178

PP179

PP180

PP181

PP182

PP183

PP184

PP185

PP186

PP187

PP188

PP189

PP190

PP191

PP192

PP193

PP194

PP195

PP196

PP197

PP198

PP199

PP200

PP201

PP202

PP203

PP204

PP205

PP206

PP207

PP208

PP209

PP210

PP211

PP212

PP213

PP214

PP215

PP216

PP217

PP218

PP219

PP220

PP221

PP222

PP223

PP224

PP225

PP226

PP227

PP228

PP229

PP230

PP231

PP232

PP233

PP234

PP235

PP236

PP237

PP238

PP239

PP240

PP241

PP242

PP243

PP244

PP245

PP246

PP247

PP248

PP249

PP250

PP251

PP252

PP253

PP254

PP255

PP256

PP257

PP258

PP259

PP260

PP261

PP262

PP263

PP264

PP265

PP266

PP267

PP268

PP269

PP270

PP271

PP272

PP273

PP274

PP275

PP276

PP277

PP278

PP279

PP280

PP281

PP282

PP283

PP284

PP285

PP286

PP287

PP288

PP289

PP290

PP291

PP292

PP293

PP294

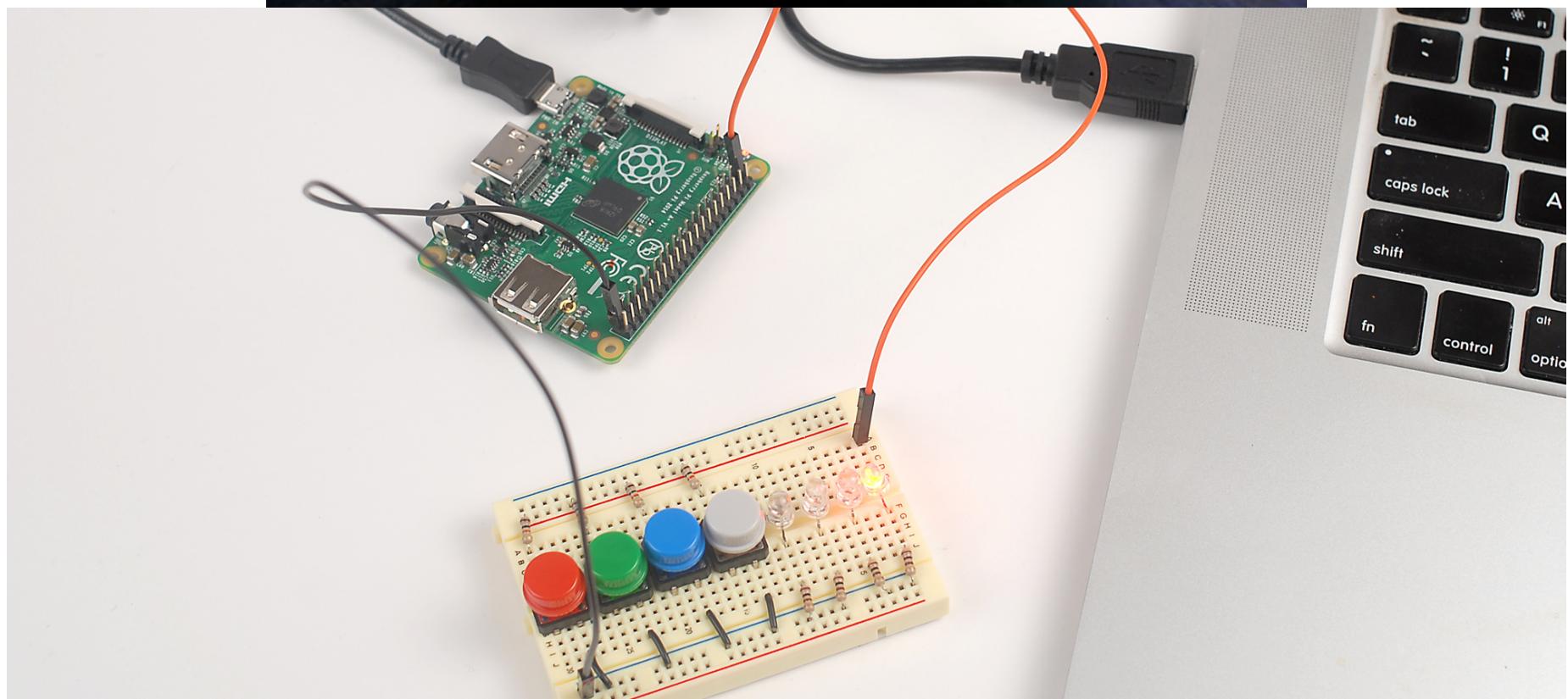
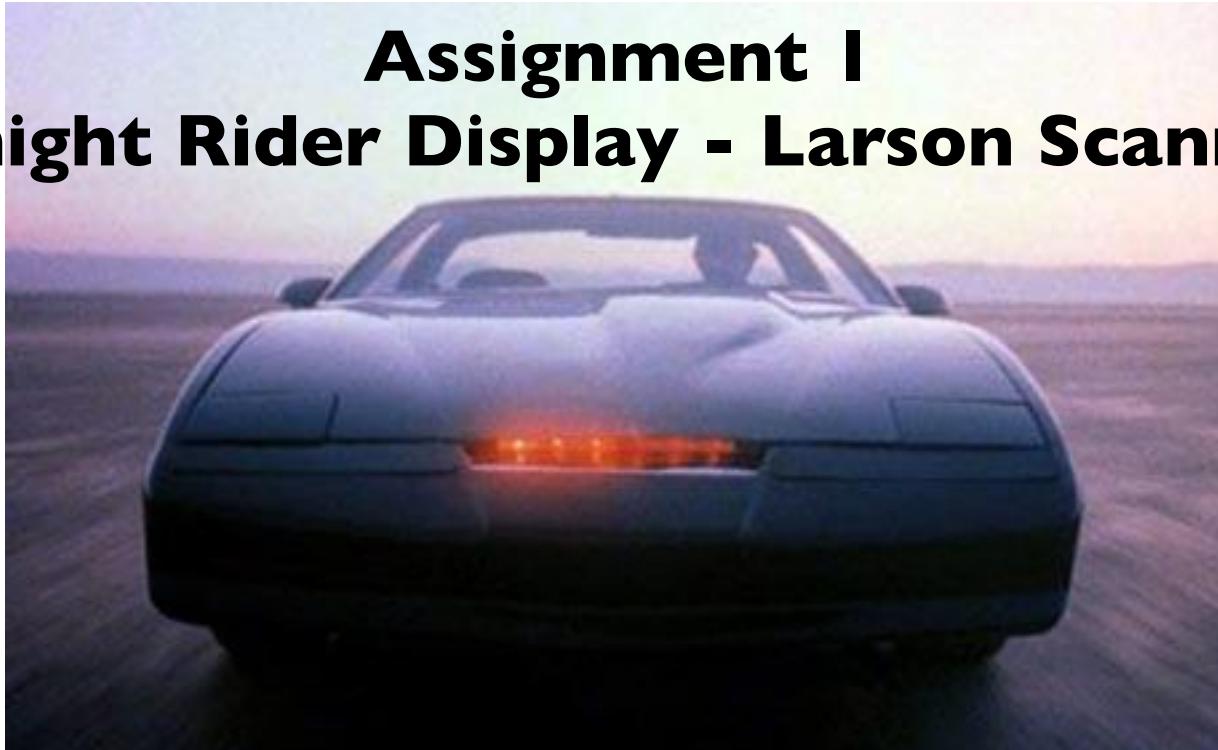
PP295

PP296

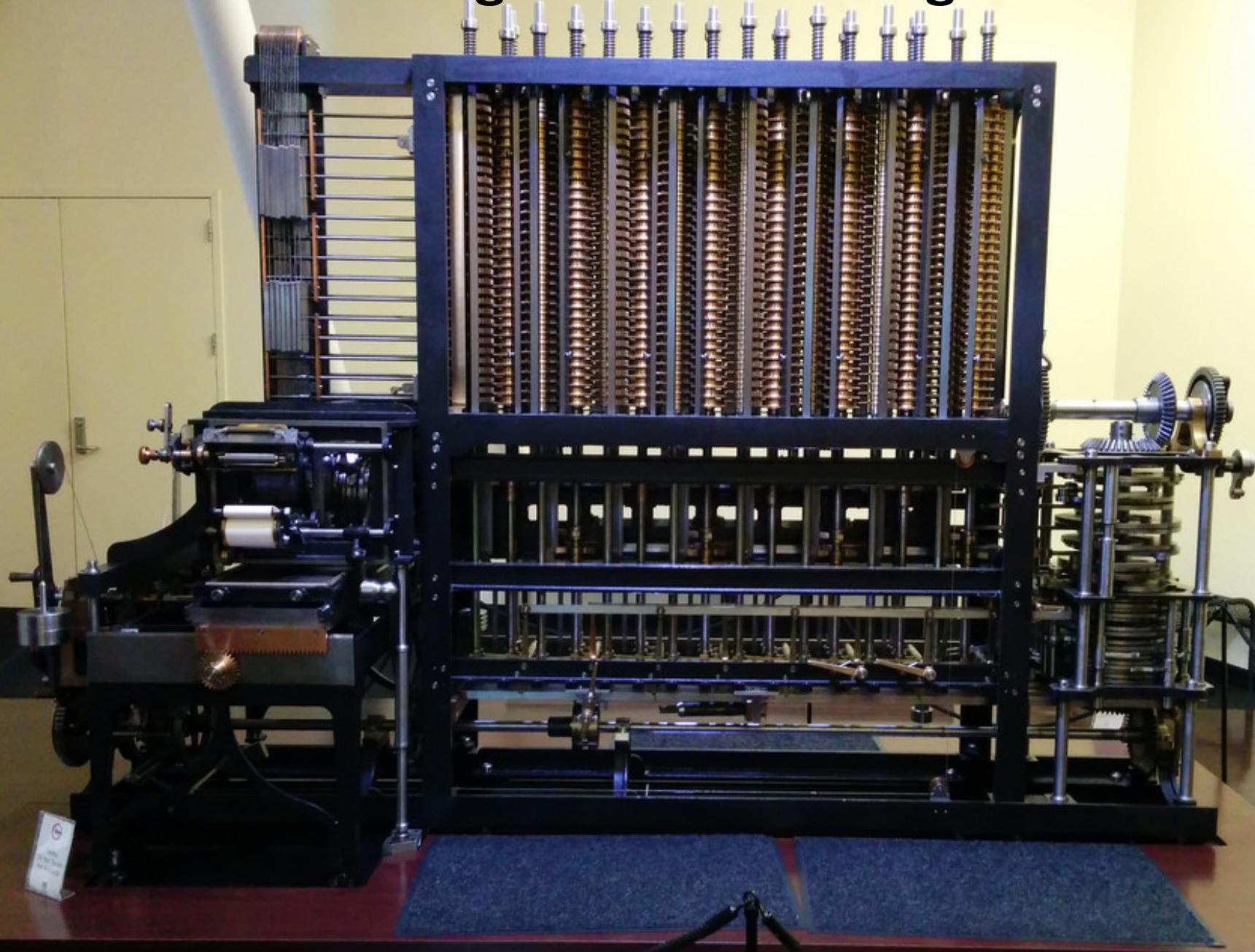
PP297

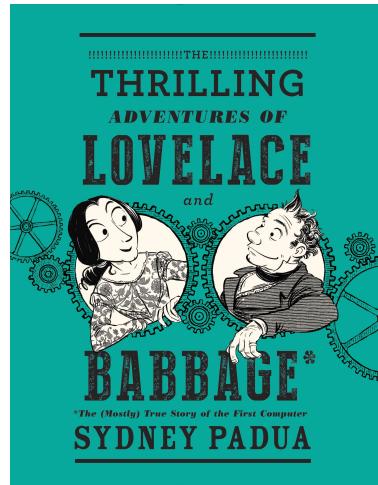
Assignment I

Knight Rider Display - Larson Scanner

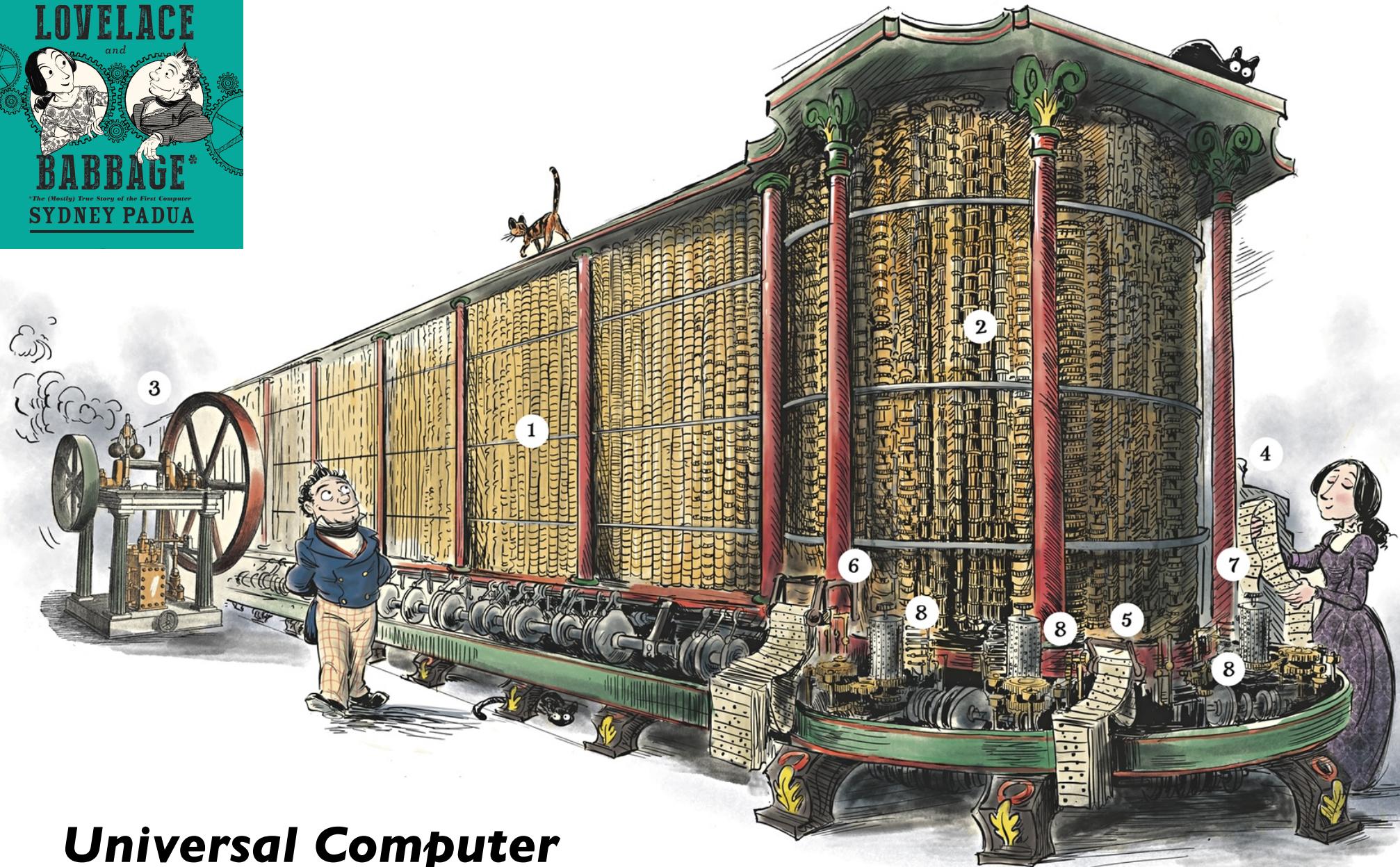


Babbage Difference Engine





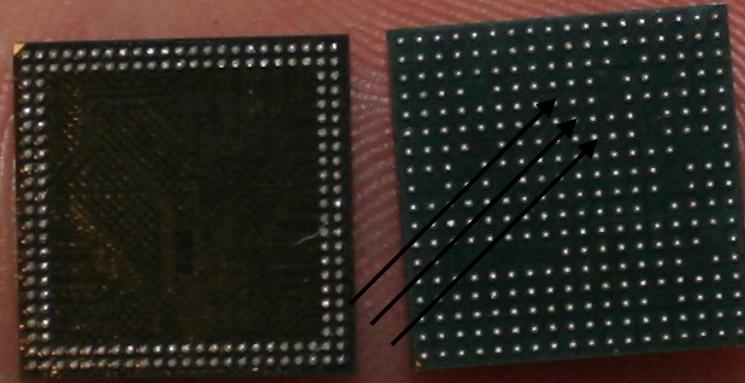
Analytical Engine



Universal Computer

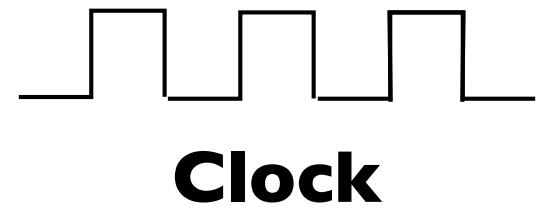
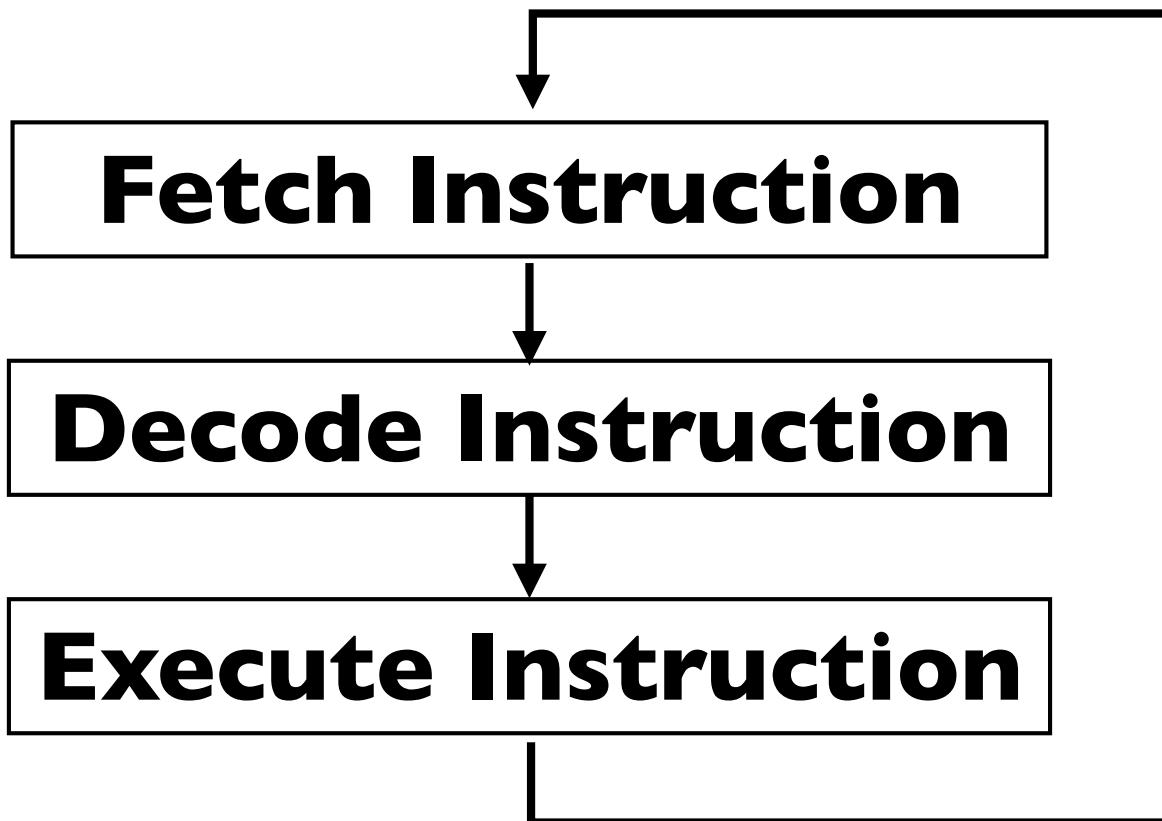
Package on Package

Broadcom 2865 ARM Processor



Samsung 4Gb (gigabit) SDRAM

Running a Program



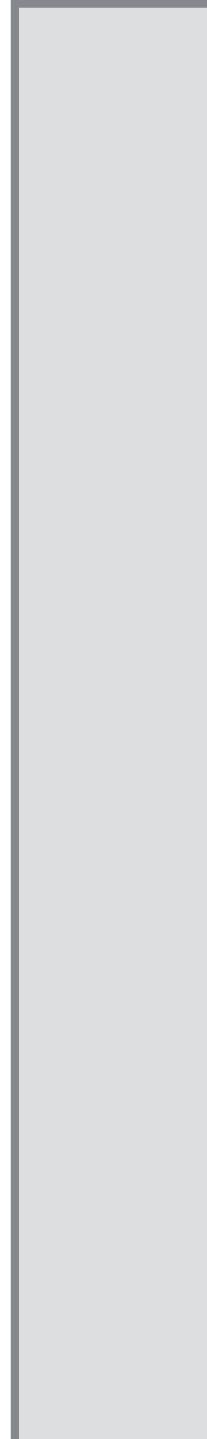
100000000_{16}

Memory used to store both instructions and data

Storage locations are accessed using 32-bit addresses

Maximum addressable memory is 4 GB (gigabyte)

Address refers to a byte (8-bits)



Memory Map

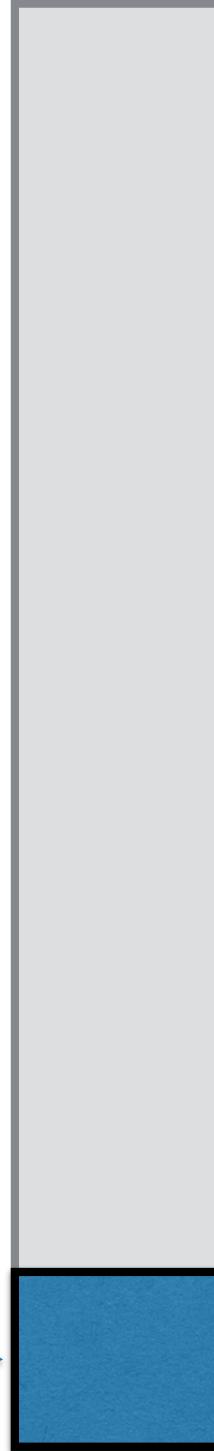
00000000_{16}

100000000_16

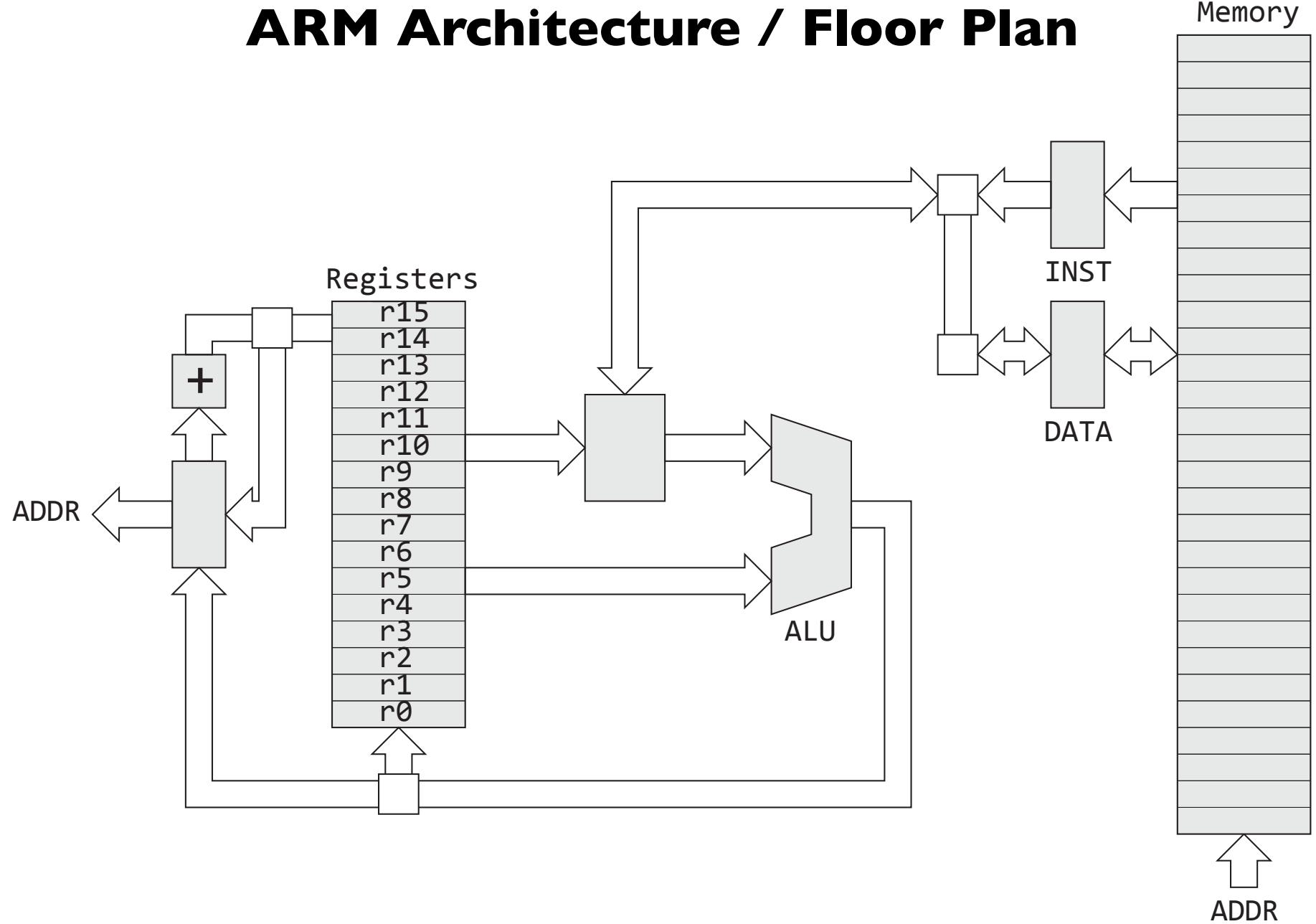
Memory Map

02000000_16

512 MB Actual Memory →

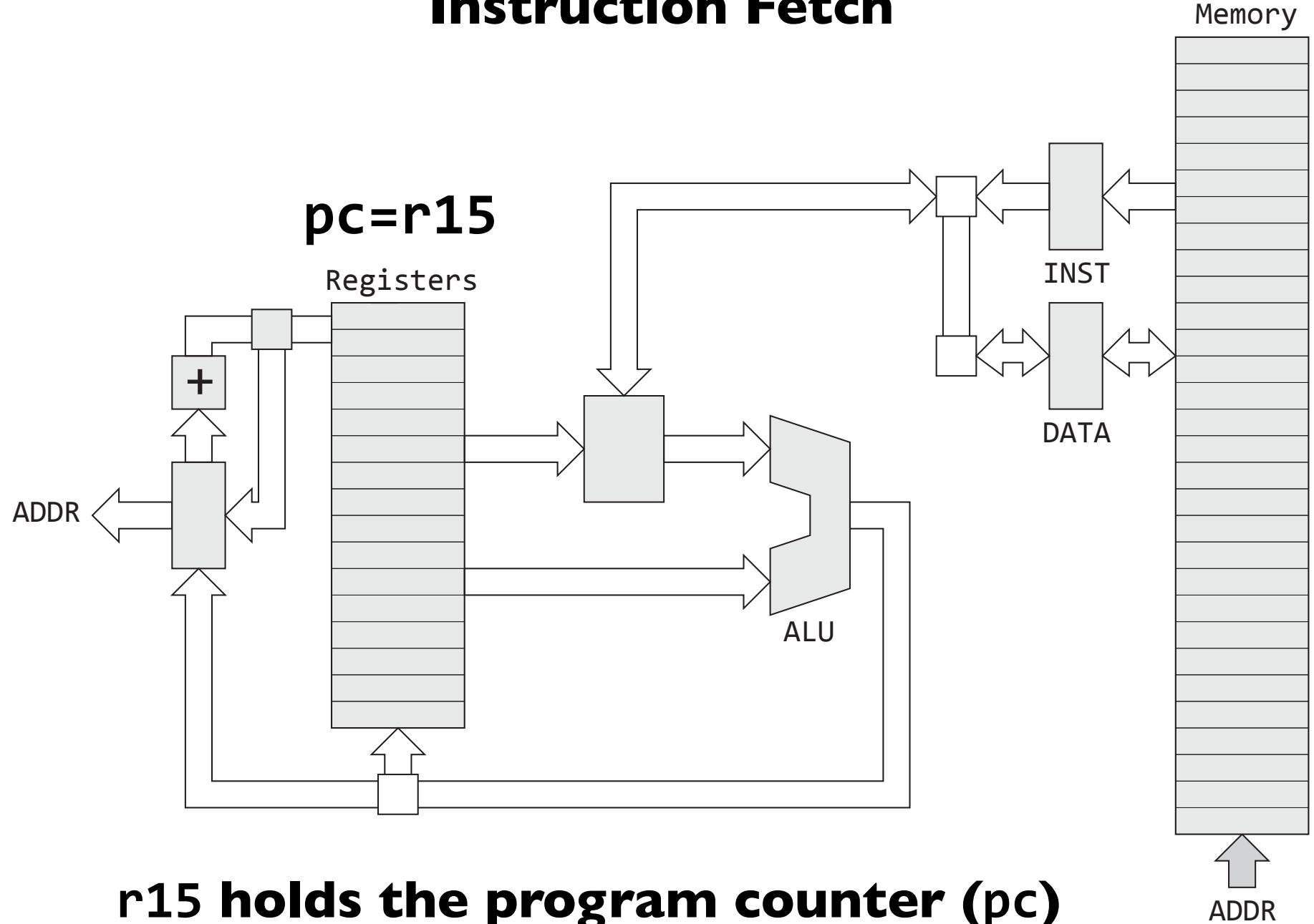


ARM Architecture / Floor Plan



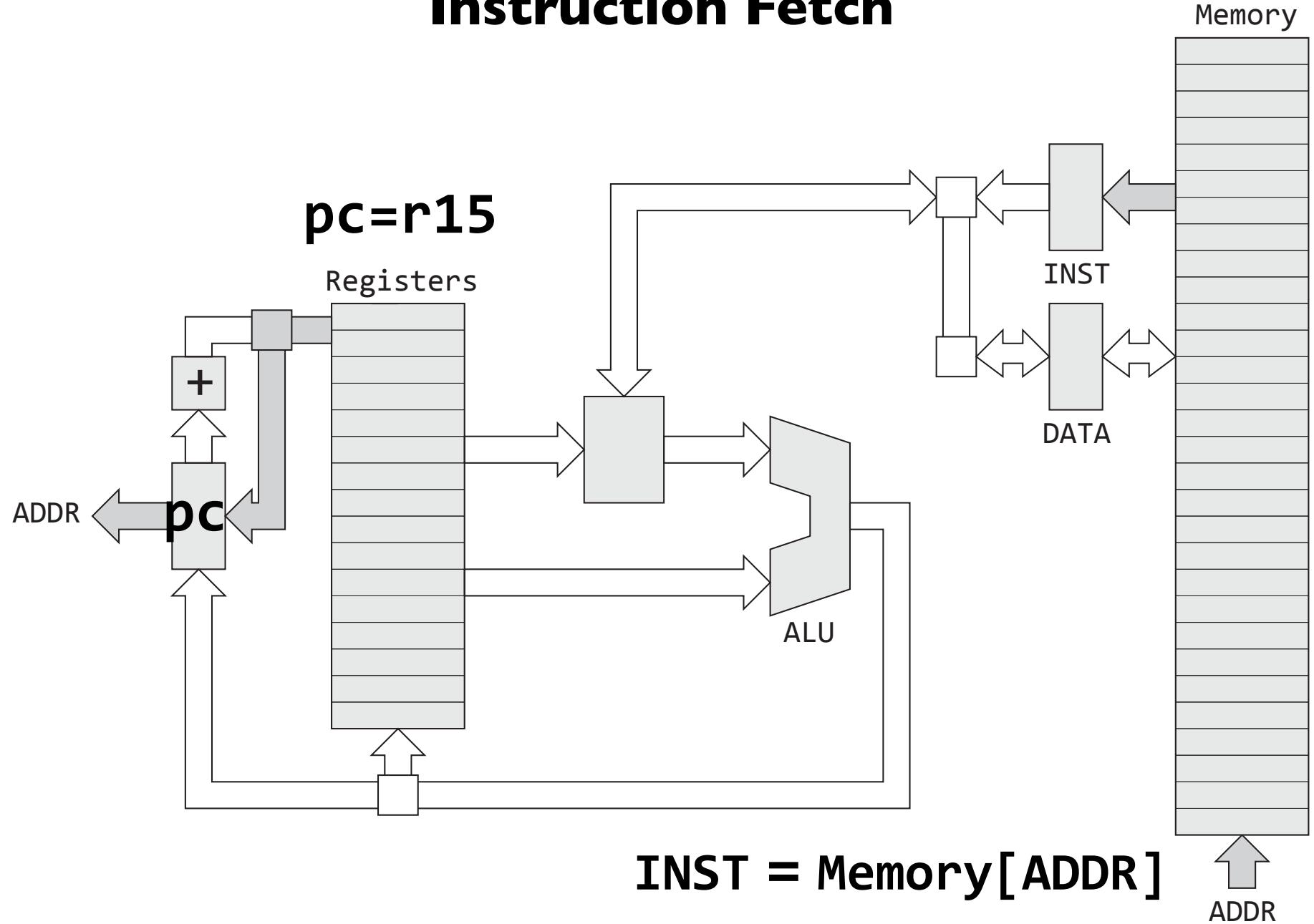
Key Fact: Everything is organized into 32-bit words

Instruction Fetch



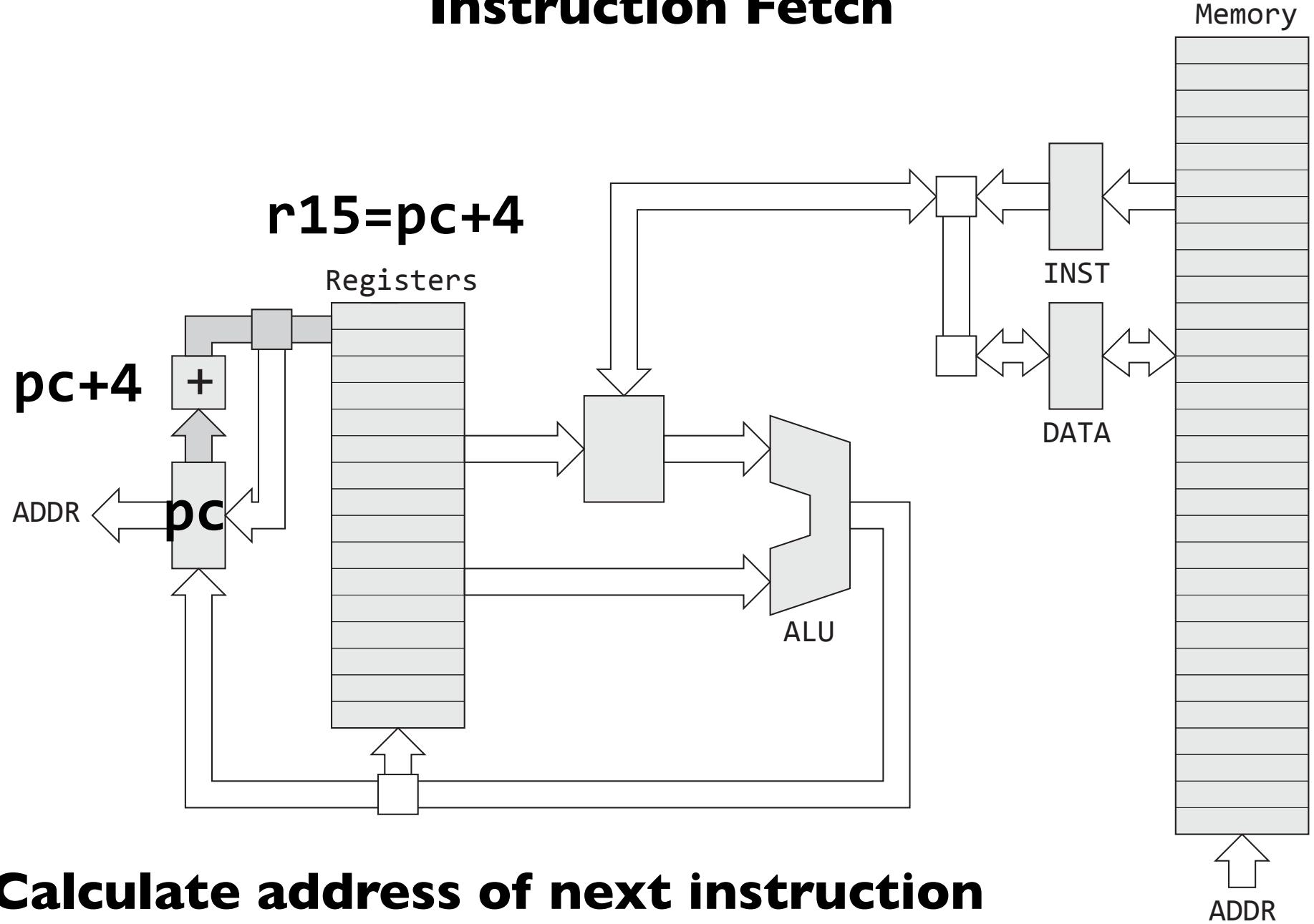
r15 holds the program counter (pc)

Instruction Fetch



Addresses and instructions are 32-bit words

Instruction Fetch



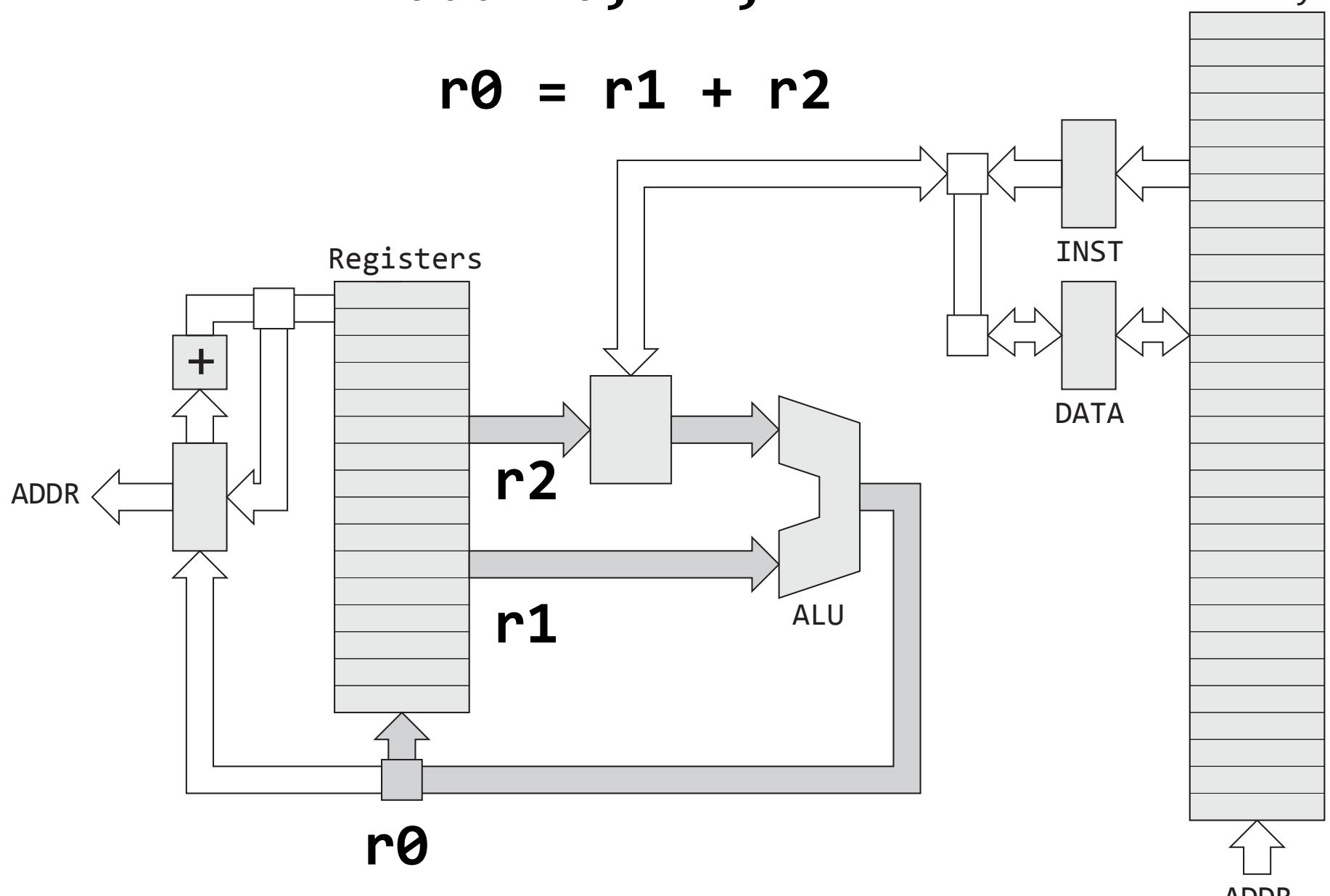
Calculate address of next instruction

Why pc+4?

Arithmetic-Logic Unit (ALU)

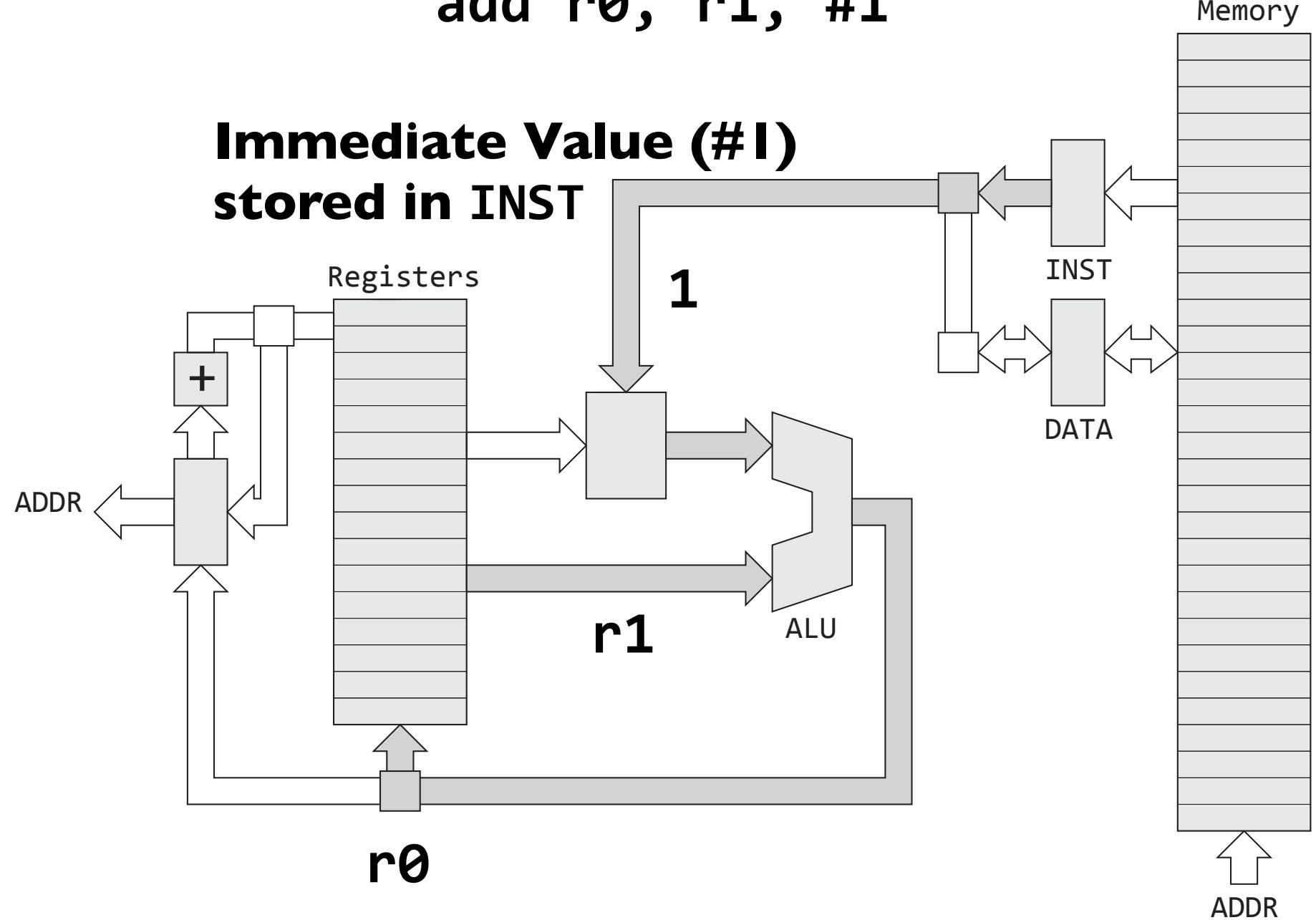
add r0, r1, r2

$$r0 = r1 + r2$$



**ALU only operates on registers
Registers are also 32-bit words**

add r0, r1, #1



Add Instruction

Meaning (defined as math or C code)

$$r_0 = r_1 + r_2$$

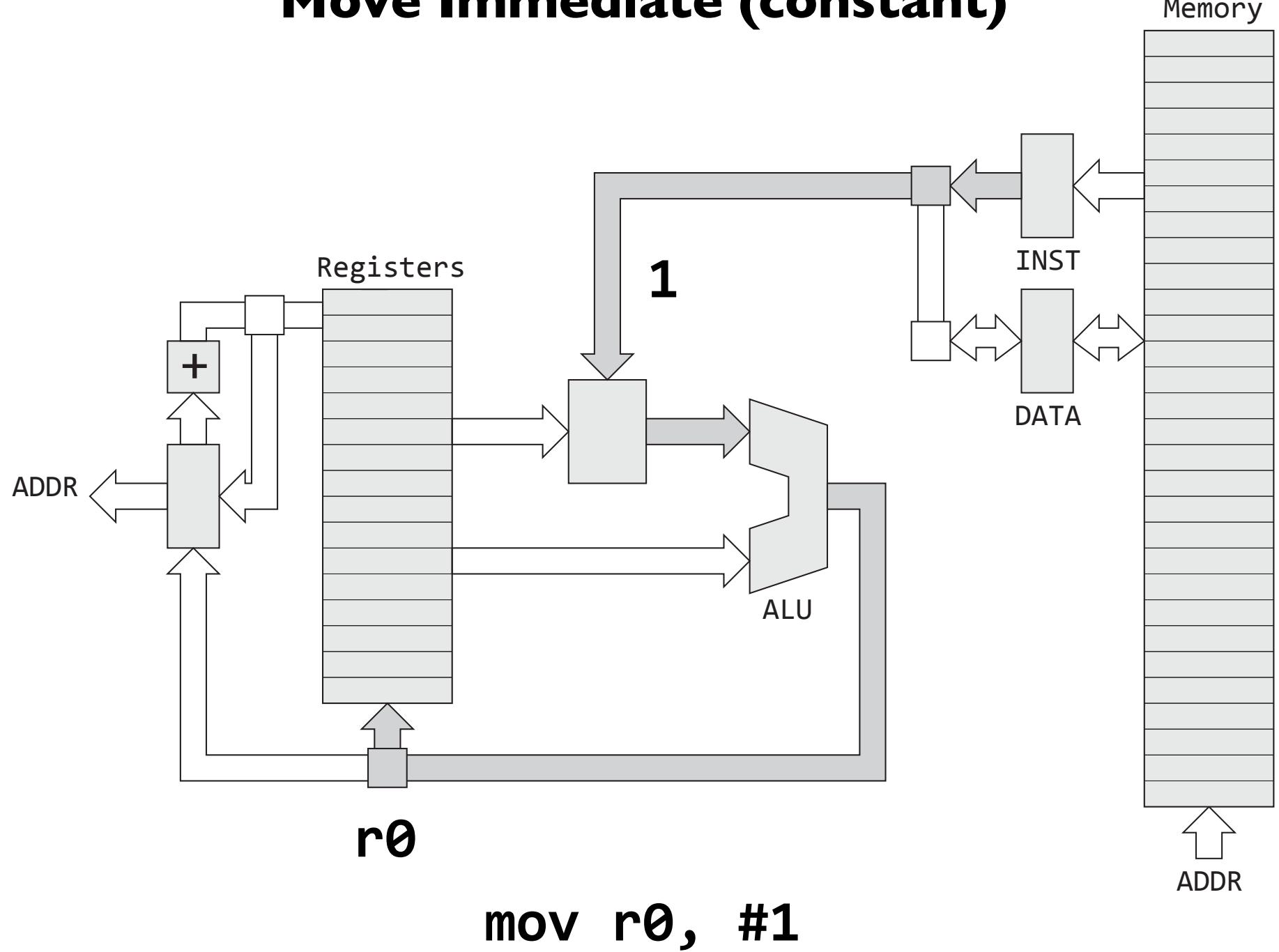
Assembly language (result is leftmost register)

add r0, r1, r2

Machine code (more on this later)

E0 81 00 02

Move Immediate (constant)



VisUAL

untitled.S - [Unsaved] - VisUAL

New Open Save Settings Tools ▾  Emulation Running Line Issues 3 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
```

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

(L) Clock Cycles Current Instruction: 1 Total: 3

CSPR Status Bits (NZCV) 0 0 0 0

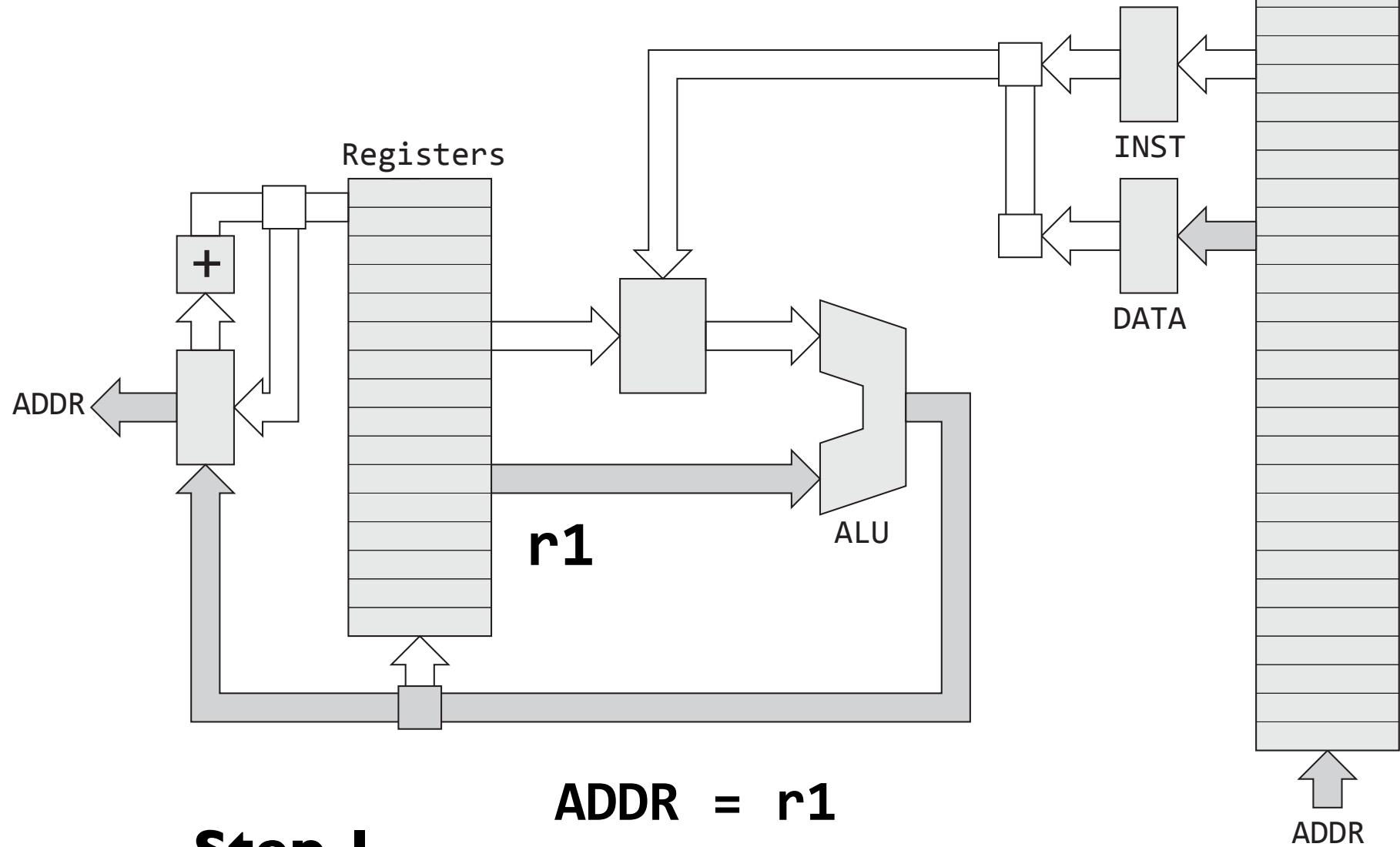
Conceptual Questions

1. Suppose your program starts at 0x8000, what assembly language instruction could you execute to jump to and start executing instructions at that location.
2. If all instructions are 32-bits, can you move any 32-bit constant value into a register using a single mov instruction?
3. What is the difference between a memory location and a register?

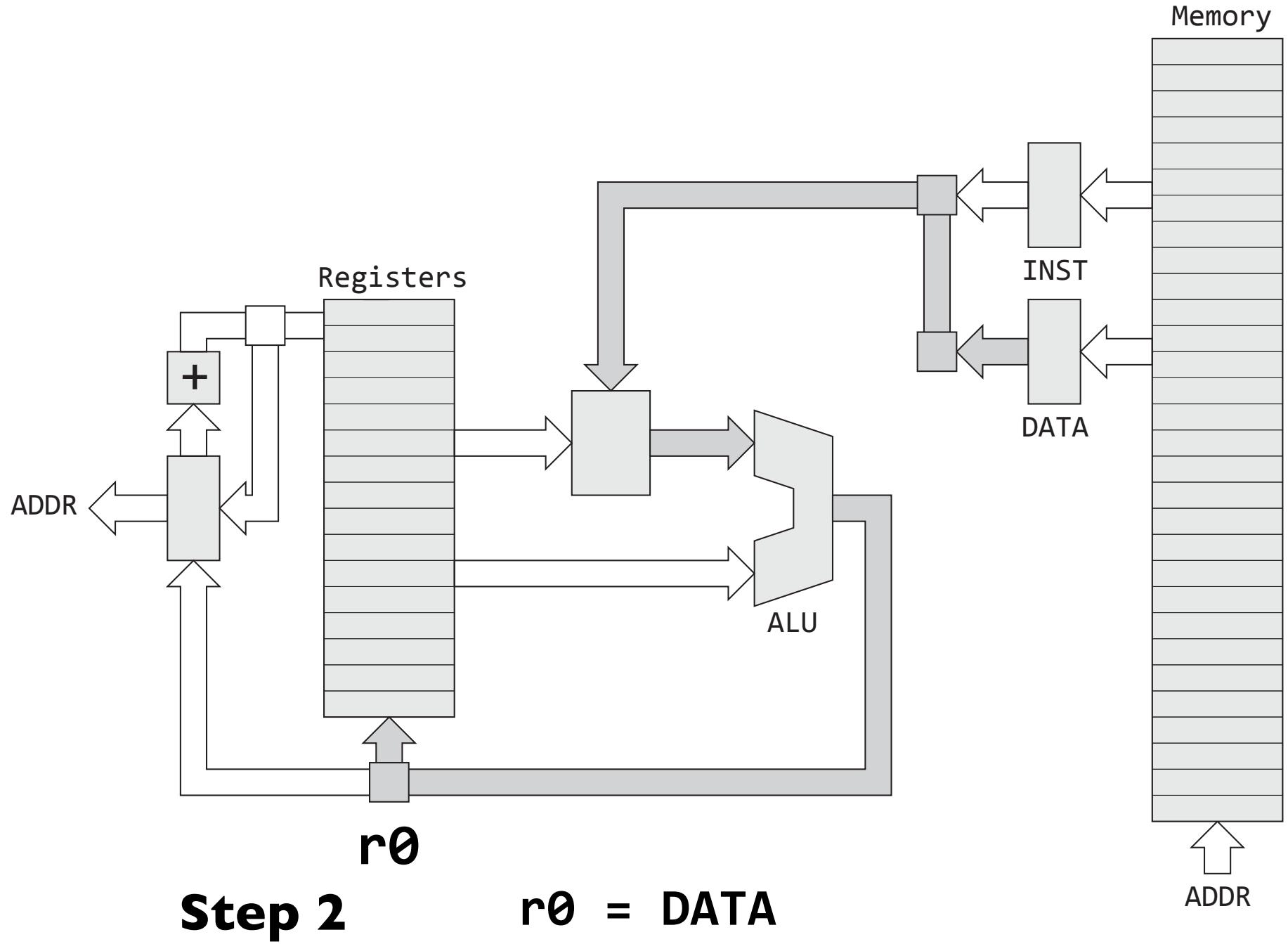
Load and Store Instructions

Load from Memory to Register (LDR)

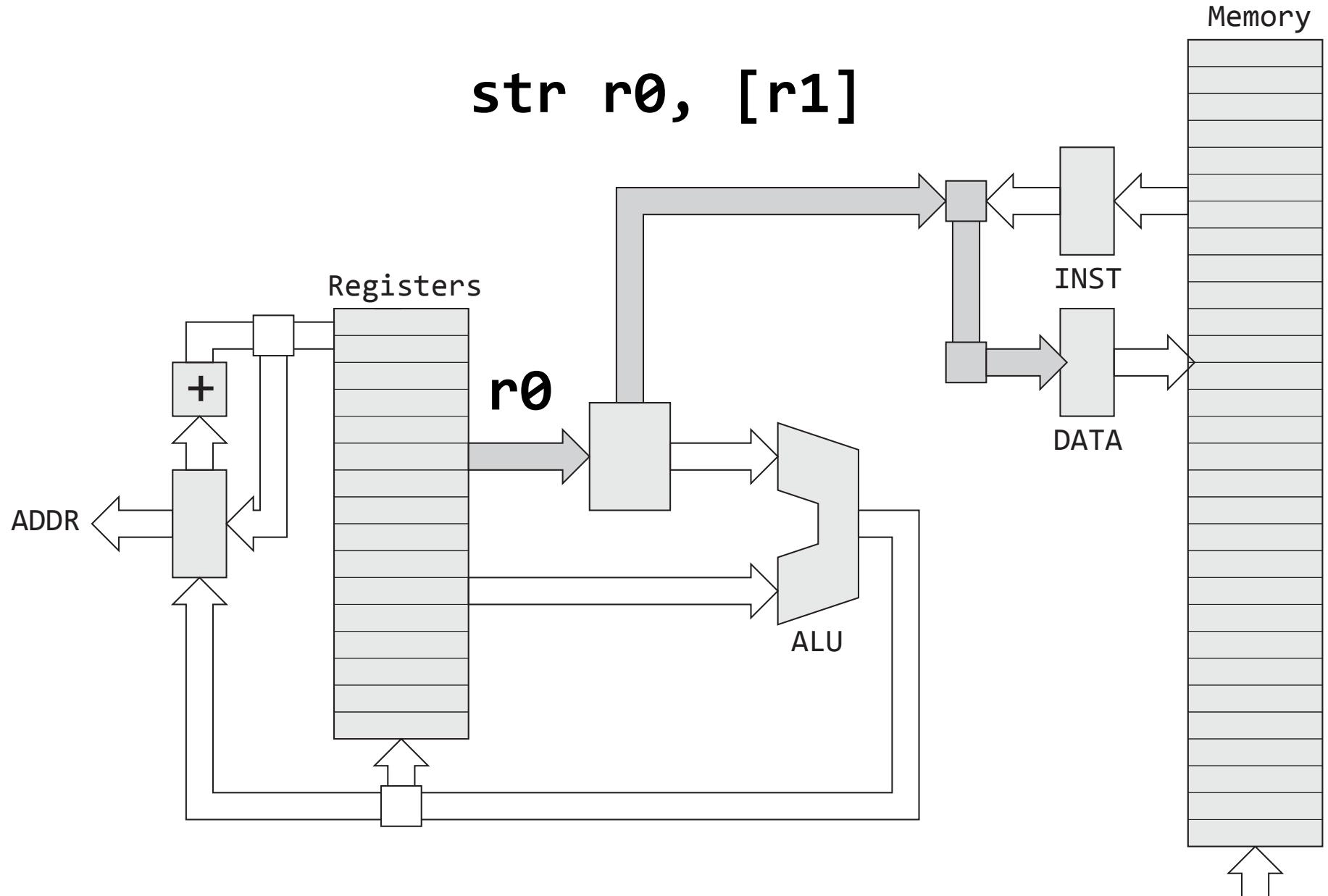
ldr r0, [r1]



Load from Memory to Register (LDR)



Store Register in Memory (STR)

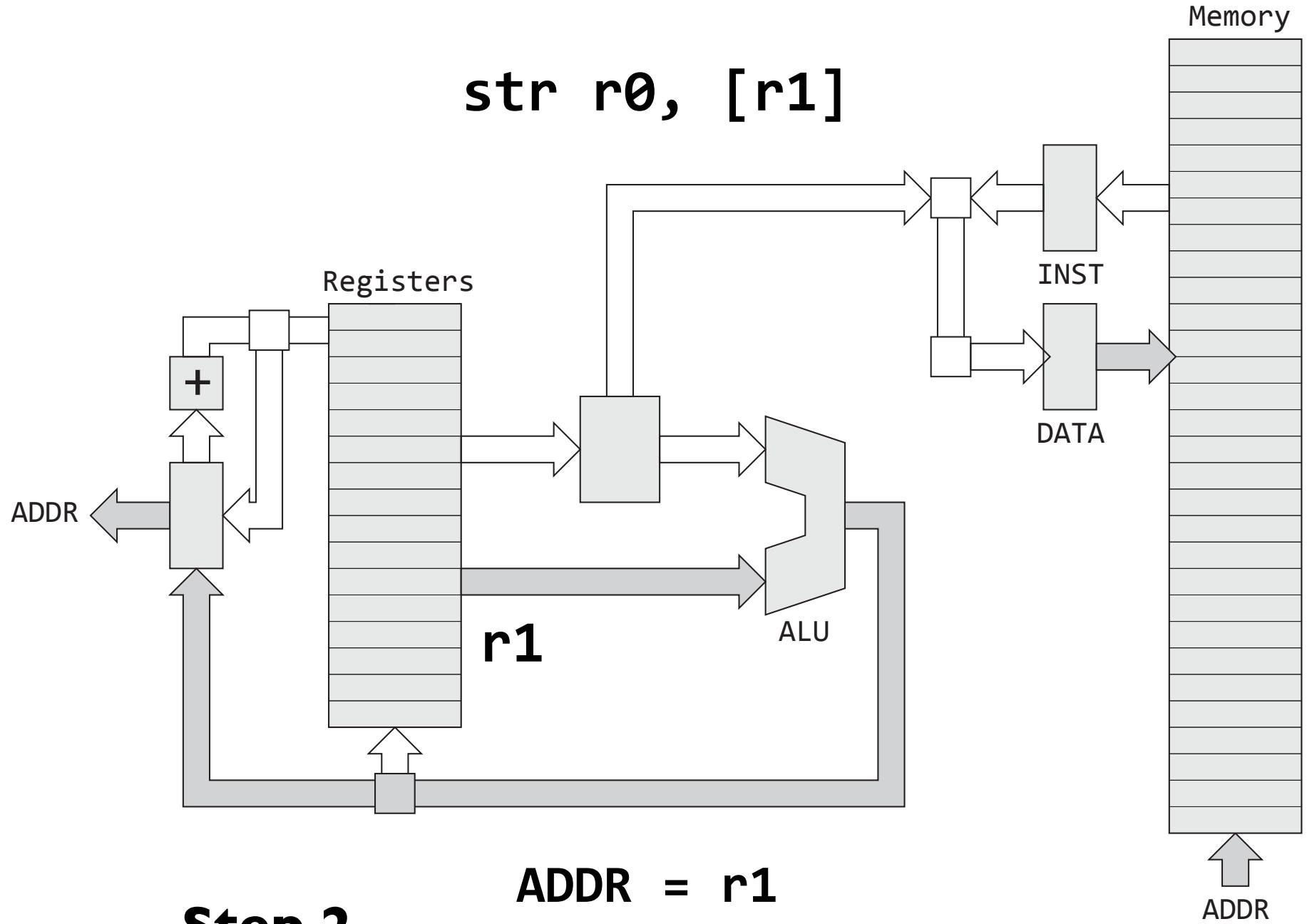


Step I

DATA = r0

ADDR

Store Register in Memory (STR)



New

Open

Save

Settings

Tools ▾



Emulation Running

Line Issues
4 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]
```

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

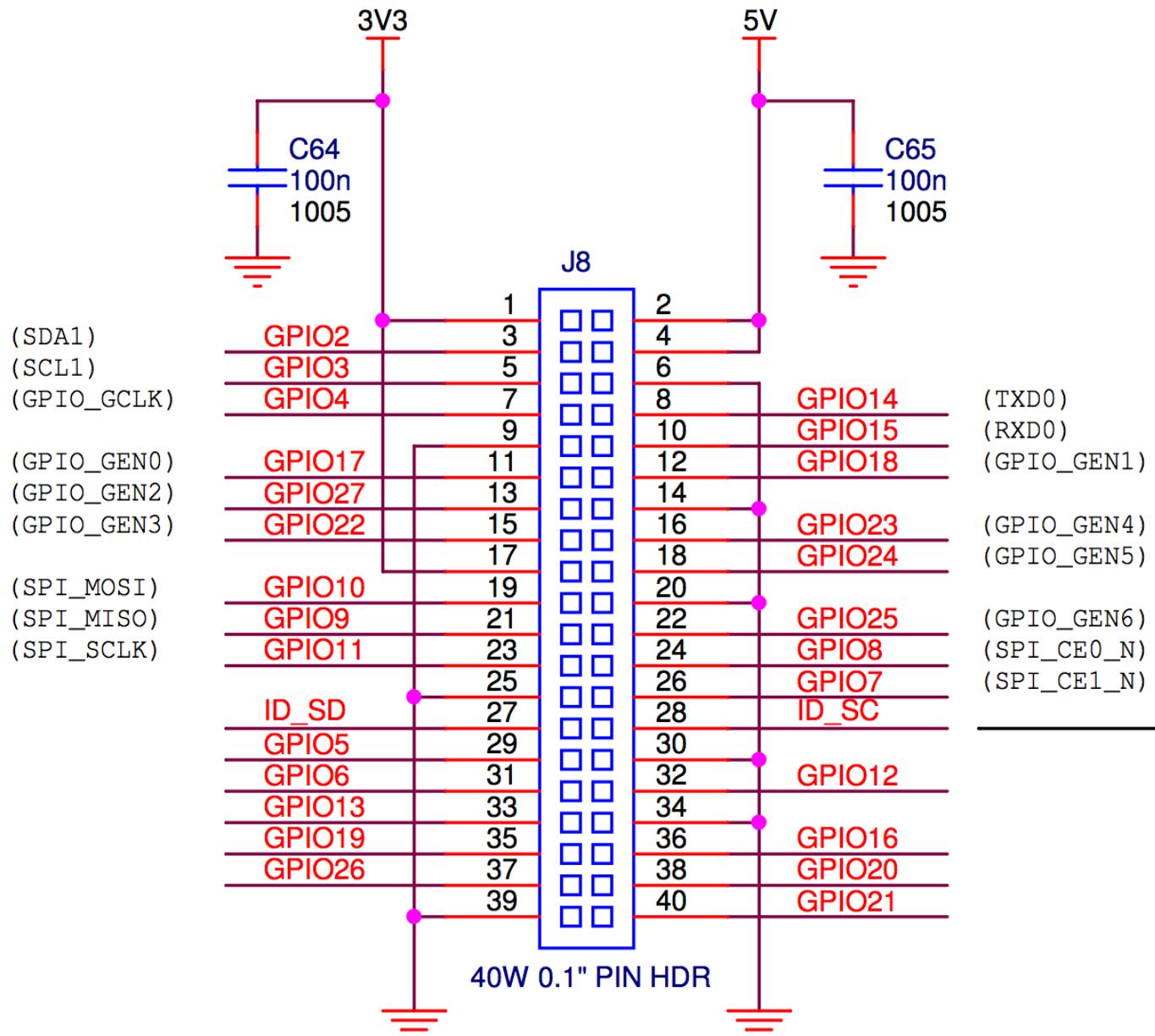
Current Instruction: 2 Total: 6

CSPR Status Bits (NZCV)

0 0 0 0

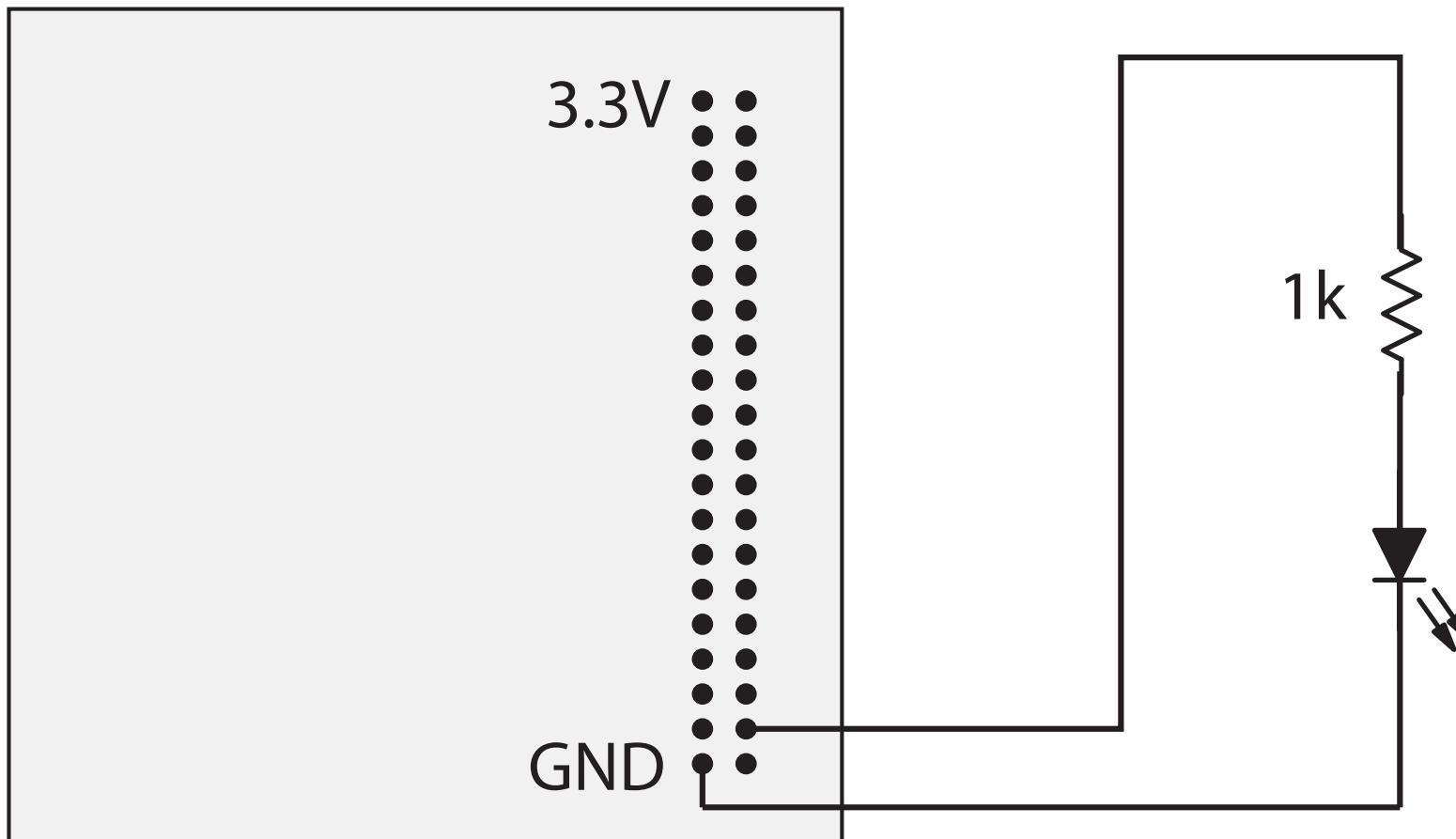
Turning on an LED

General-Purpose Input/Output (GPIO) Pins

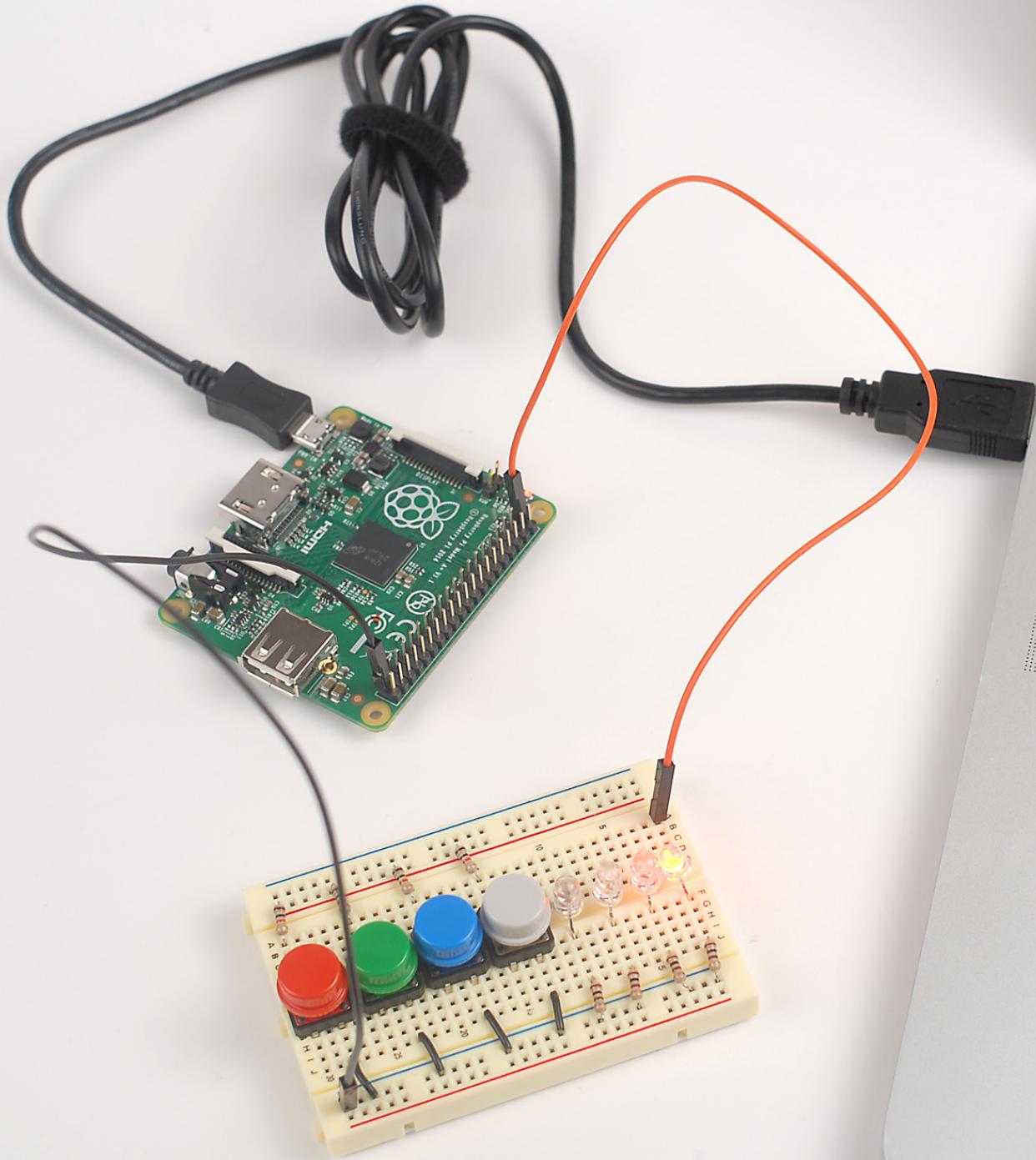


54 GPIO Pins

Connect LED to GPIO 20



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



GPIO Pins are *Peripherals*

**Peripherals are Controlled
by Special Memory Locations**

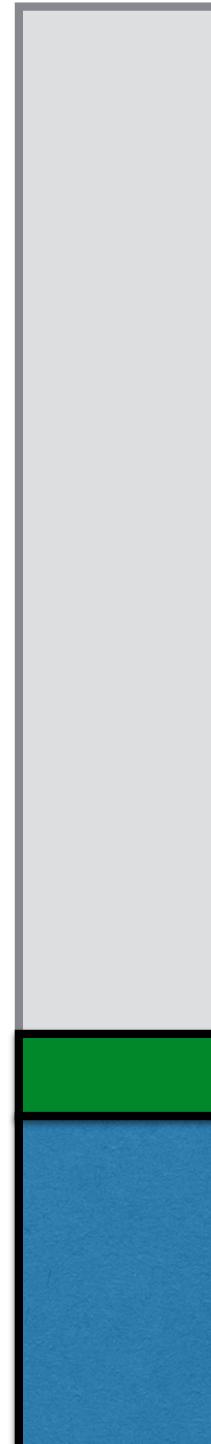
"Peripheral Registers"

Memory Map

**Peripheral registers
are mapped
into address space**

**Memory-Mapped IO
(MMIO)**

**MMIO space is above
physical memory**



100000000_{16}
4 GB

020000000_{16}

512 MB

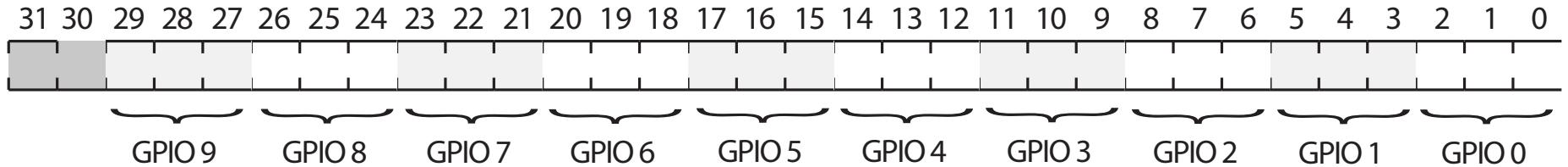
General-Purpose IO Function

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 times 3 bits = 30 bits

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 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, #0x20          // r0 = #0x00000020  
lsl r1, r0, #24        // r1 = #0x20000000  
lsl r2, r0, #16        // r2 = #0x00200000  
orr r1, r1, r2         // r1 = #0x20200000  
orr r0, r1, #0x08      // r0 = #0x20200008  
  
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

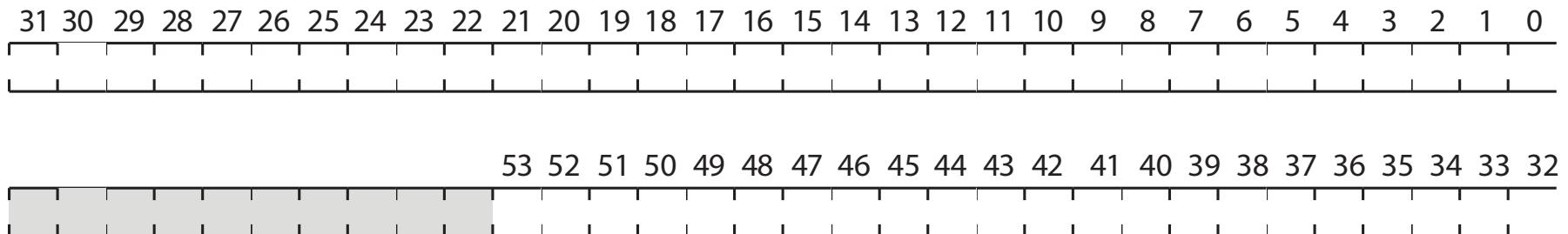
Bit(s)	Field Name	Description	Type	Reset
31-22	-	Reserved	R	0
21-0	SETn (n=32..53)	0 = No effect 1 = Set GPIO pin <i>n</i> .	R/W	0

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

Assignment 0

- Join forum piazza.com/stanford/winter2020/cs107e
- Read and understand our guides on basic topics (electricity, numbers, unix). Practice unix command line
- Checkout the course directory from gitub
- Checkout your assignment directory and answer the questionnaire, check-in to submit assignment
- Setup your development environment before lab

Key Concepts

- Bits are bits; fundamental bitwise operations
- Memory addresses refer to bytes (8-bits), words are 4 bytes
- Memory stores both instructions and data
- Computers repeatedly fetch, decode, and execute instructions
- Different types of ARM instructions: ALU, Loads and Stores, Branches
- General purpose IO (GPIO), peripheral registers, and MMIO

Further Reading

If you want to learn more about high-level computer organization and instructions, Chapter 2 of Computer Organization and Design: The Hardware/Software Interface (Patterson and Hennessy) is an excellent place to start.

Or take EE180 in Spring!

