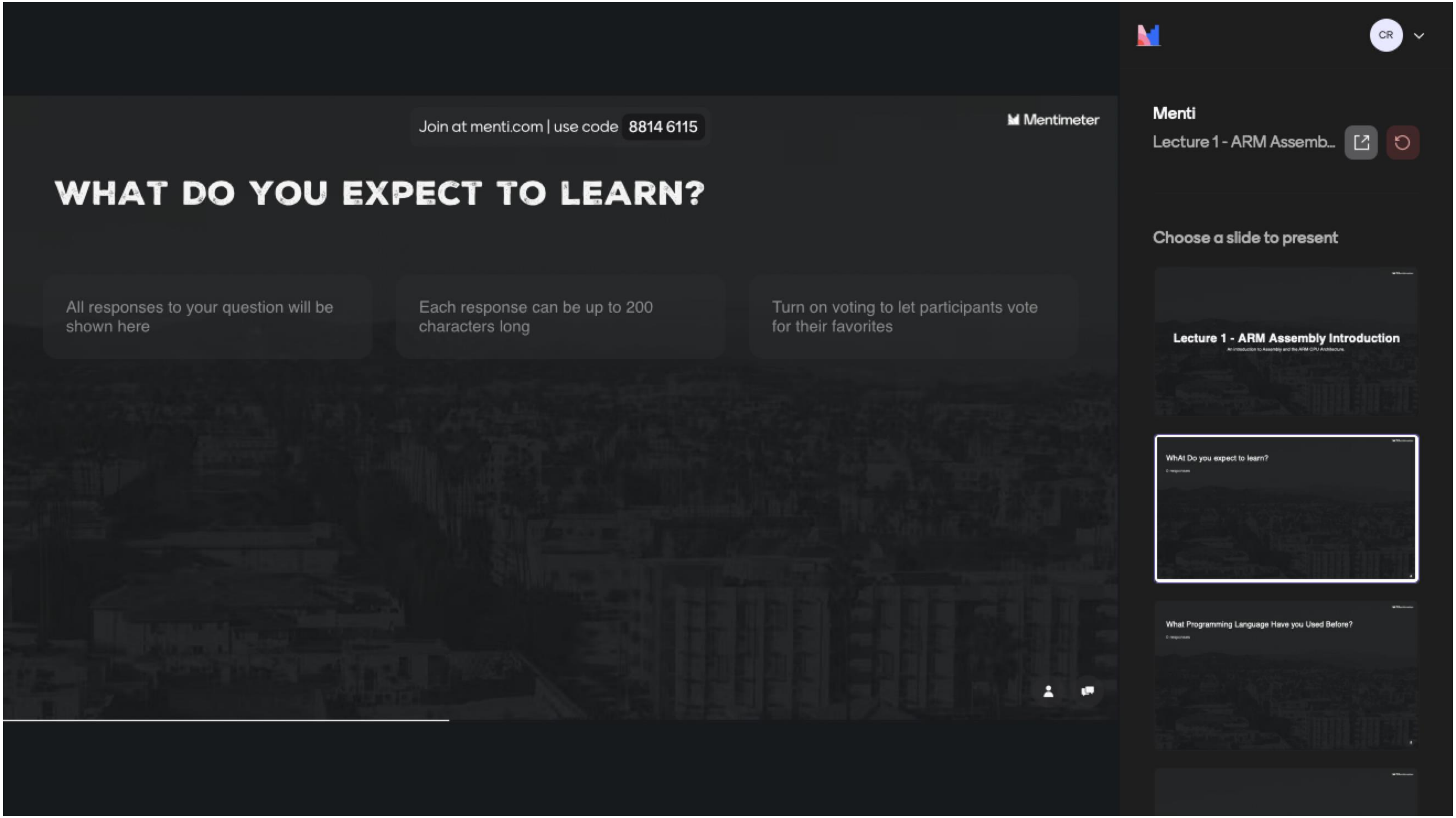
ARM Assembly Introduction

Haris Rotsos



CR

Join at menti.com | use code 8814 6115

Mentimeter

WHAT DO YOU EXPECT TO LEARN?

All responses to your question will be shown here

Each response can be up to 200 characters long

Turn on voting to let participants vote for their favorites

Menti

Lecture 1 - ARM Assemb...

Choose a slide to present

Lecture 1 - ARM Assembly Introduction

WhAt Do you expect to learn?

What Programming Language Have you Used Before?

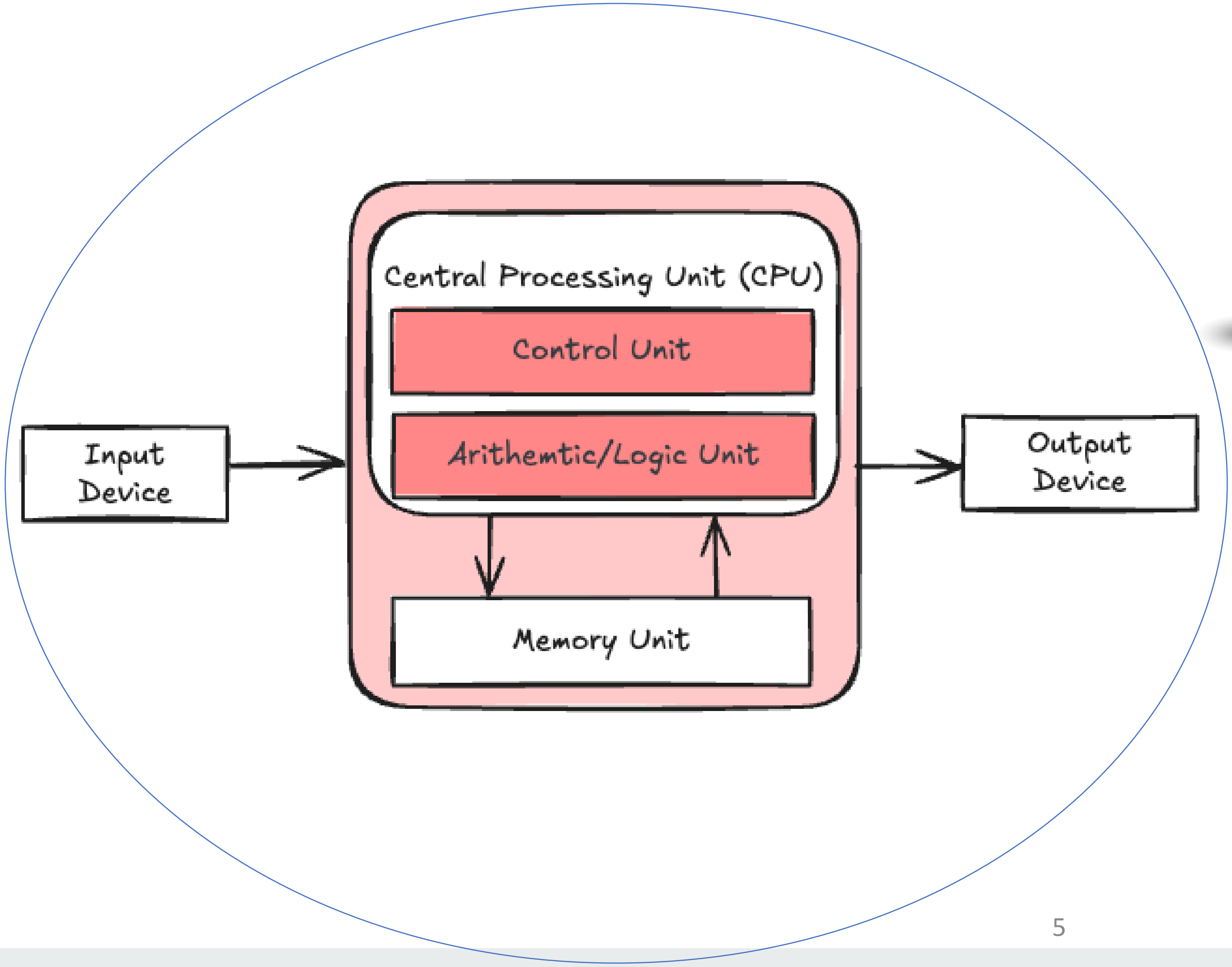
Outline

- What you learned so far:
  - 131: Computer Architecture Theory, Microbit Architecture
  - 111: C/C++ programming language
- How do we execute a program written in a high-level language (C/C++/Java) on a Computer Architecture?
- Introduction to Assembly
  - What is Assembly?
  - How do we execute assembly in the micro:bit?
  - How do all these relate to computer architecture?

Learning Goals

- Assembly Programming
  - New programming paradigm — no variables, just registers (few) and memory
- Understand how a computer works
  - Troubleshoot runtime errors
  - OS concepts
  - Embedded programming
- Assembly programming is unique.
- Ask questions (labs, FAST Hub), follow labs, revise material.

Reminder of the von Neumann architecture



What is an "instruction set architecture (ISA)"?

- The view of the processor that is seen by programs being executed
  - What is the set of available instructions?
  - What is the set of available registers?
  - How many operands do instructions need?
  - What are the sizes and types of the operands?
  - How are operands accessed (e.g., stack-based ISAs don't support random access to memory – see next slide)?
  - How many operands can be in registers (vs. in memory)?
  - How many clock ticks does it take to execute an instruction?
  - …

```
MOV r0, #0x11
LSL r1, r0, #1
LSL r2, r1, #1
stop:
B stop
```

In this part, we will learn how to program using an ARM v6 ISA.

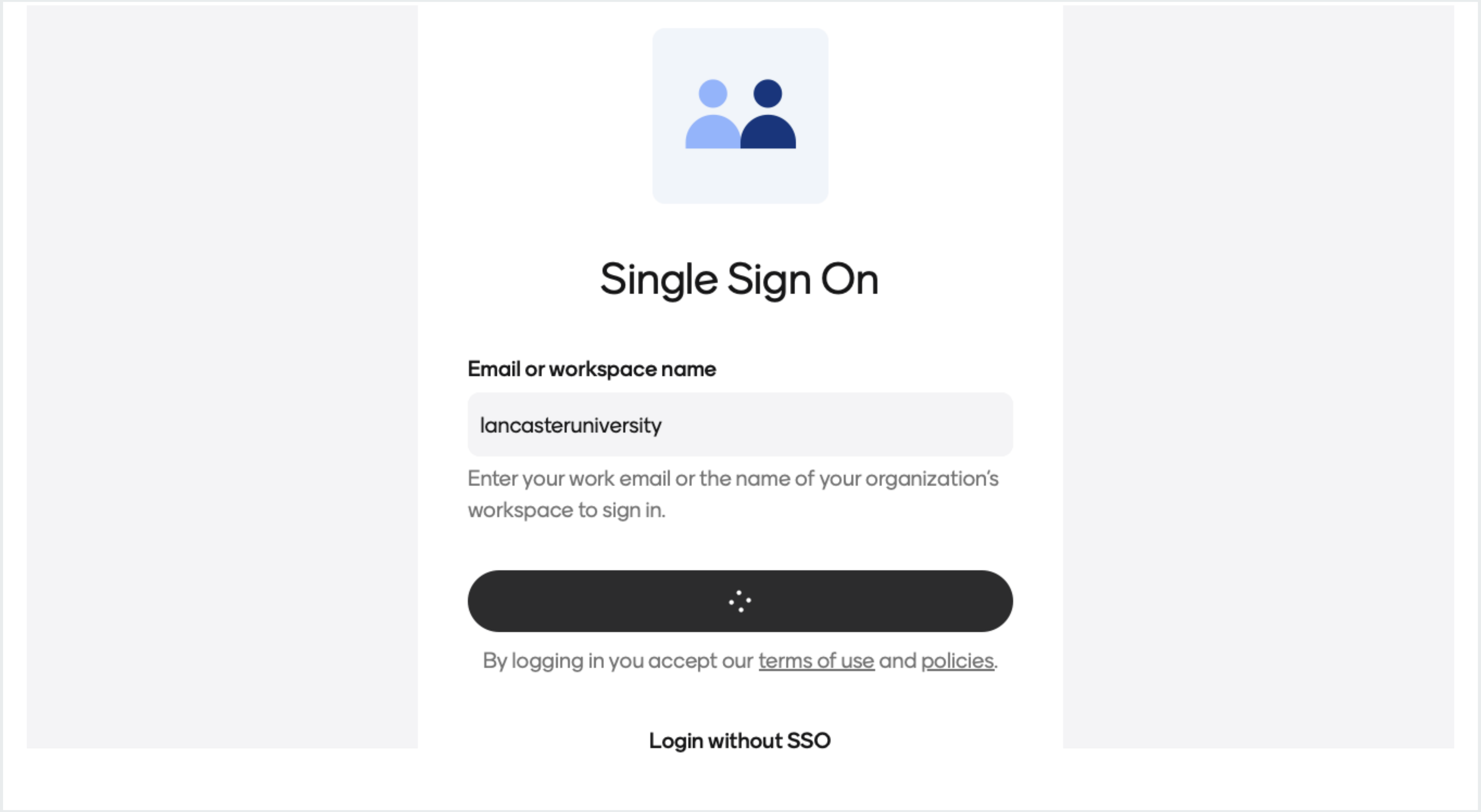
Architecture Paradigms

- The Acorn RISC Machine (ARM) architecture is a VonNeuman architecture.

- Complex Instruction Set Computer (CISC):
  - Rich ISA, a single instruction executes several low-level operations.
  - Parallel processing pipelines, improved processor speed at a lower clock rate.
- Reduced Instruction Set Computer (RISC):
  - Small, highly optimized instruction set.
  - Keep pipeline simple, raise complexity in software.
  - Power and heat efficiency.
- ARM is a RISC architecture.

ARM Instruction Set (1)

- ARM design goals
  - lowering of the compiler to the hardware level
  - not raising of hardware to the software level (as with CISC)
- ARM is a **32-bit architecture**. This defines:
  - the range of values in basic arithmetic — How big is an Integer
  - the number of addressable bytes — 2^{32} bytes, $[0, 2^{32}-1]$
  - the width of a standard register



Single Sign On

Email or workspace name

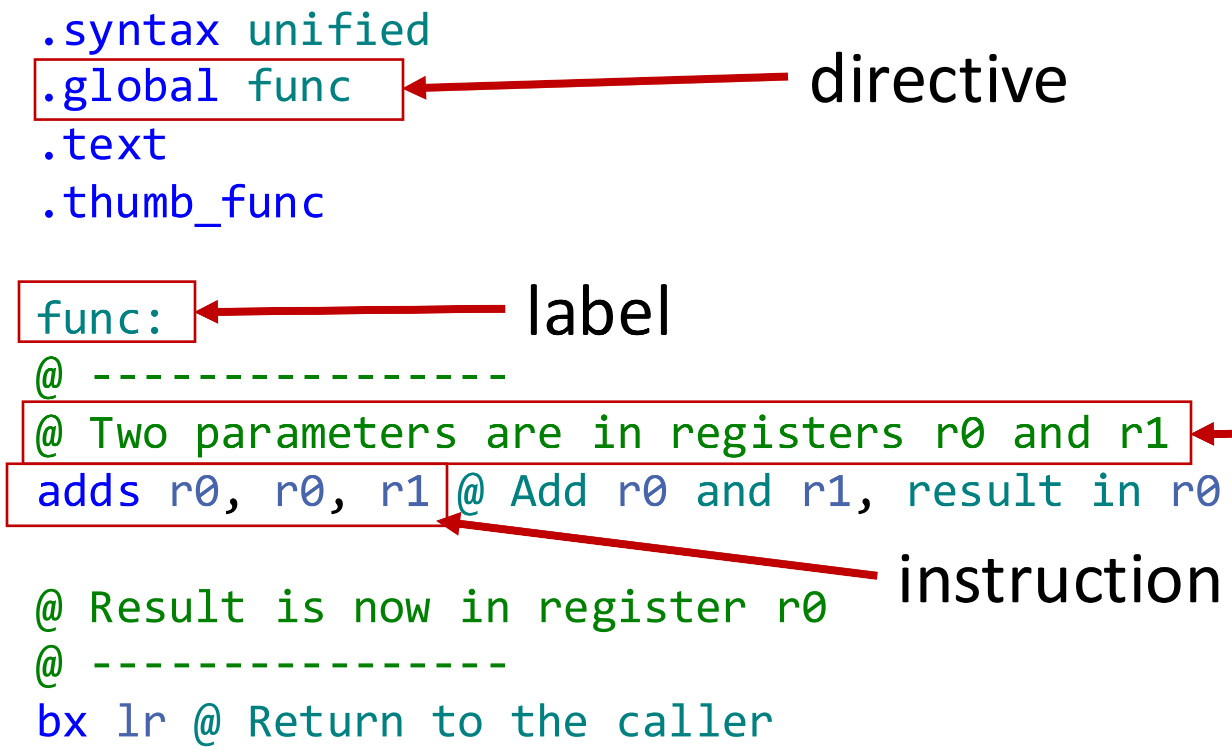
lancasteruniversity

Enter your work email or the name of your organization’s workspace to sign in.

By logging in you accept our terms of use and policies.

Login without SSO

Example Assembly code



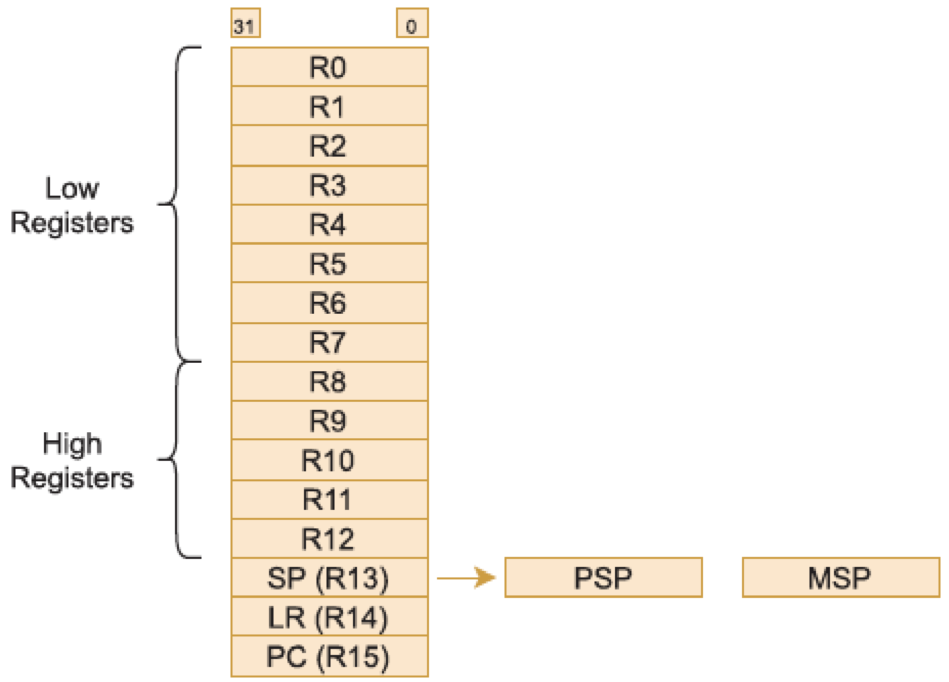
```
.syntax unified
.global func
.text
.thumb_func

func:
@ -----------------
@ Two parameters are in registers r0 and r1
adds r0, r0, r1 @ Add r0 and r1, result in r0

@ Result is now in register r0
@ -----------------
bx lr @ Return to the caller
```

```
int func(int a, int b) {
    return a + b;
}
```

Registers (1)

- An ALU cannot process information from main memory.
  - You need to load data to a register.
- The register is the most fundamental storage area on the chip.
  - Can carry many types: integer, float, pointer.
  - Must be 32-bit long, i.e. can hold a pointer to a string, but not a string.
- 16 registers in an ARM CPU.
- Some registers reserver for operational CPU state, e.g. current instruction address.

Registers (2)



- Registers are represented with a 4-bit number.
- **R0–R7**: lower registers (bit-4 = 0)
- **R8-R12**: higher registers (bit-4=1)
  - specific instructions store data to them automatically.

You can use these in your program to process data.

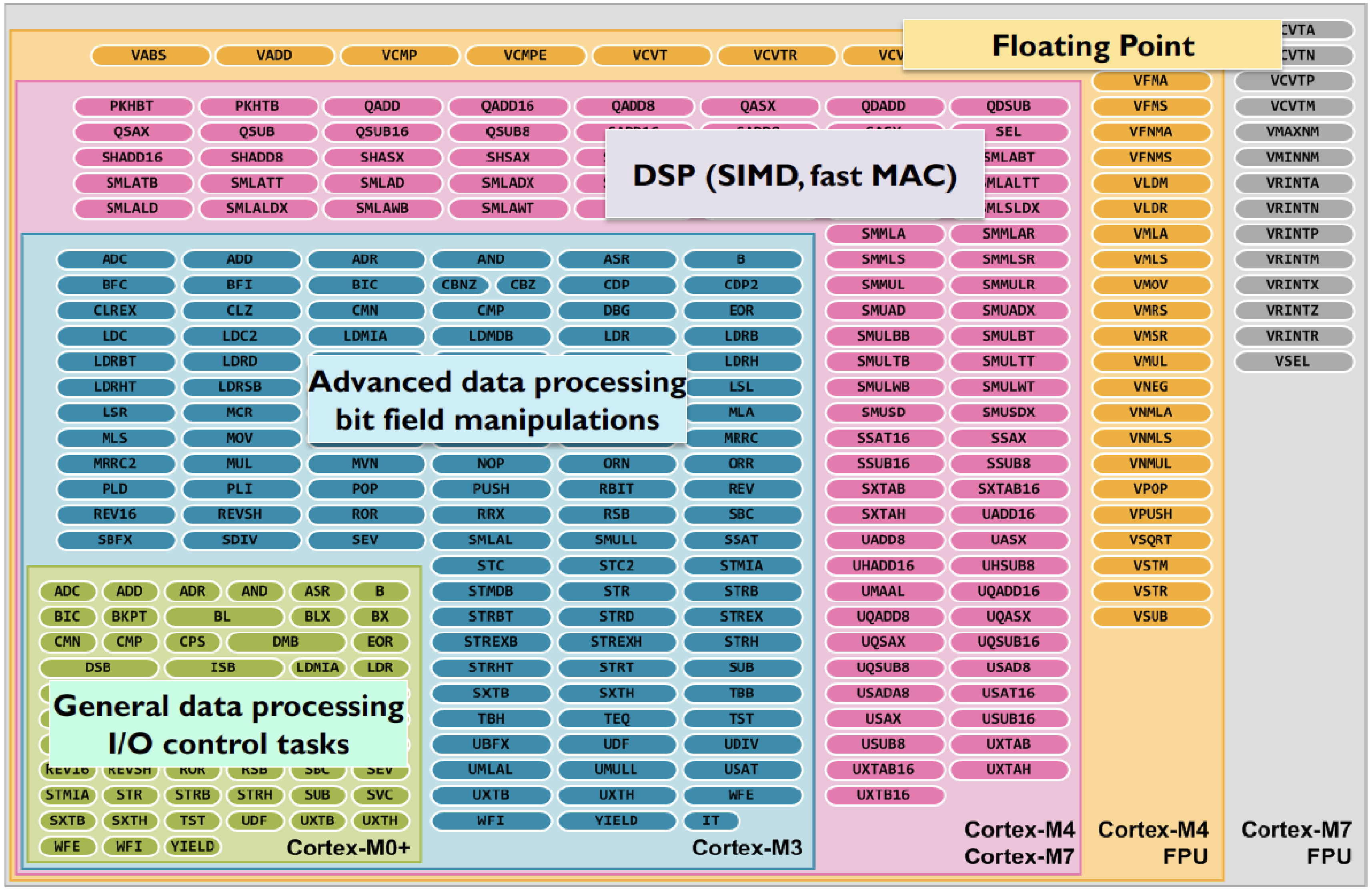
- **R13** is the Stack Pointer (SP)
- **R14** is the Link Register (LR)
- **R15** is the program counter (PC)
- **Current Program Status Register (CPSR)**: CPU Status register on co-processor.

Used by your CPU to keep track of your program execution.

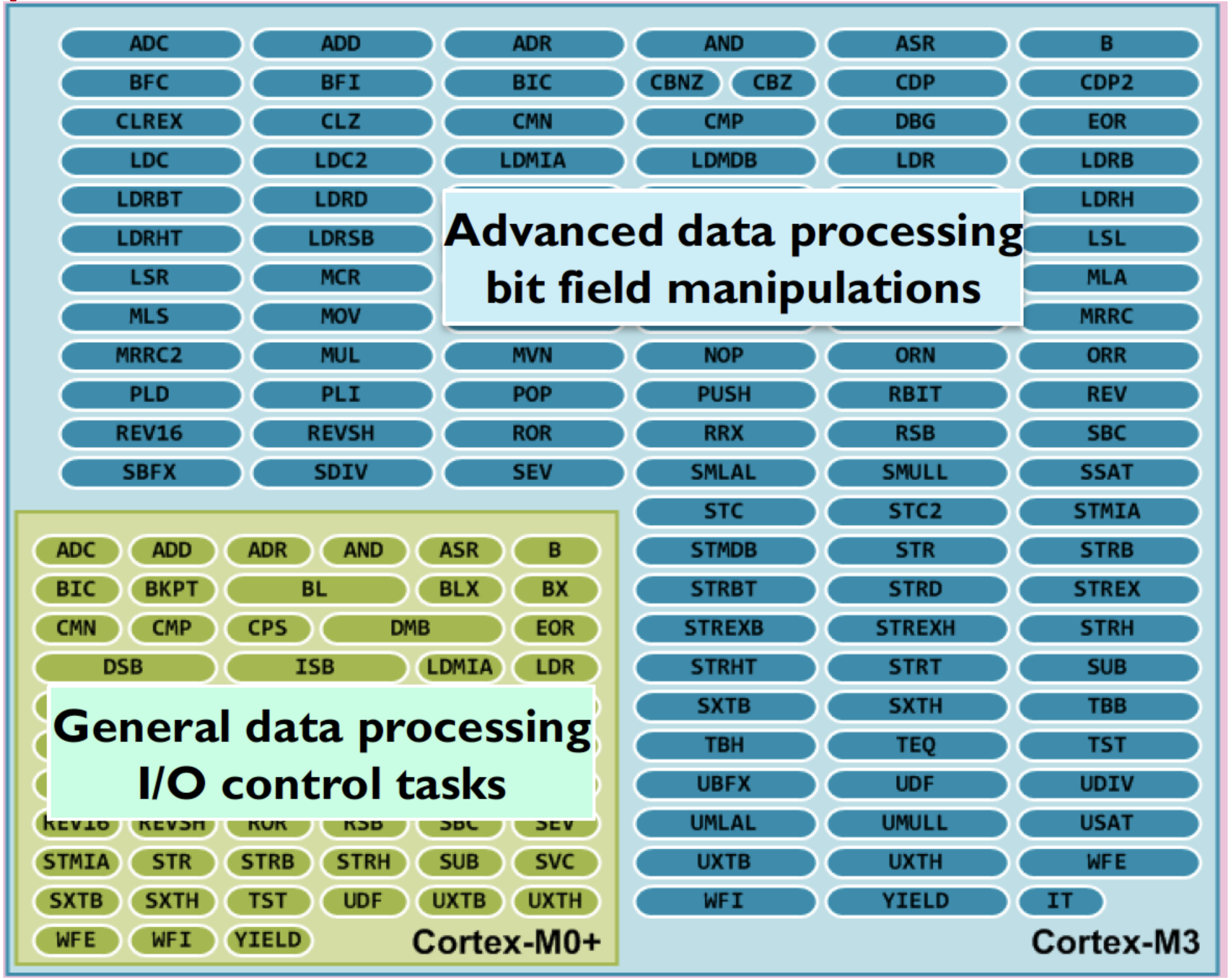
The Instruction

- An instruction is the most basic unit of computer processing
  - Instructions are words in the language of a computer
  - Instruction Set Architecture (ISA) is the vocabulary
- The language of the computer can be written as
  - Machine language: Computer-readable representation (that is, 0’s and1’s)
  - Assembly language: Human-readable representation
- We will study ARM (in detail)
- Principles are similar in all ISAs (x86, SPARC, RISC-V, …)

ARM Instruction


Floating Point
VABS VADD VCMP VCMPE VCVT VCVTR VCV
VCVTA VCVTN VCVTP VCVTM VMAXNM VMINNM VRINTA VRINTN VRINTP VRINTM VRINTX VRINTZ VRINTR VSEL
VFMA VFMS VFNMA VFNMS VLDM VLDR VMLA VMLS VMOV VMRS VMSR VMUL VNEG VNMLA VNMLS VNMUL VPOP VPUSH VSQRT VSTM VSTR VSUB
DSP (SIMD, fast MAC)
PKHBT PKHTB QADD QADD16 QADD8 QASX QDADD QDSUB
QSAX QSUB QSUB16 QSUB8 SEL
SHADD16 SHADD8 SHASX SHSAX SMLABT
SMLATB SMLATT SMLAD SMLADX SMLALTT
SMLALD SMLALDX SMLAWB SMLAWT SMLSLDX
SMMLA SMMLAR SMMLS SMMLSR SMMUL SMMULR SMUAD SMUADX SMULBB SMULBT SMULTB SMULTT SMULWB SMULWT SMUSD SMUSDX SSAT16 SSAX SSUB16 SSUB8 SXTAB SXTAB16 SXTAH UADD16 UADD8 UASX UHADD16 UHSUB8 UMAAL UQADD16 UQADD8 UQASX UQSAX UQSUB16 UQSUB8 USAD8 USADA8 USAT16 USAX USUB16 USUB8 UXTAB UXTAB16 UXTAH UXTB16
Advanced data processing
bit field manipulations
ADC ADD ADR AND ASR B
BFC BFI BIC CBNZ CBZ CDP CDP2
CLREX CLZ CMN CMP DBG EOR
LDC LDC2 LDMIA LDMDB LDR LDRB
LDRBT LDRD LDRH
LDRHT LDRSB LSL
LSR MCR MLA
MLS MOV MRRC
MRRC2 MUL MVN NOP ORN ORR
PLD PLI POP PUSH RBIT REV
REV16 REVSH ROR RRX RSB SBC
SBFX SDIV SEV SMLAL SMULL SSAT
STC STC2 STMIA
STMDB STR STRB
STRBT STRD STREX
STREXB STREXH STRH
STRHT STRT SUB
SXTB SXTH TBB
TBH TEQ TST
UBFX UDF UDIV
UMLAL UMULL USAT
UXTB UXTH WFE
WFI YIELD IT
General data processing
I/O control tasks
ADC ADD ADR AND ASR B
BIC BKPT BL BLX BX
CMN CMP CPS DMB EOR
DSB ISB LDMIA LDR
REV16 REVSH ROR RSB SBC SEV
STMIA STR STRB STRH SUB SVC
SXTB SXTH TST UDF UXTB UXTH
WFE WFI YIELD
Cortex-M0+
Cortex-M3
Cortex-M4
Cortex-M7
Cortex-M4
FPU
Cortex-M7
FPU

M0/M0+/M3 Instructions



ADC	ADD	ADR	AND	ASR	B
BFC	BFI	BIC	CBNZ CBZ	CDP	CDP2
CLREX	CLZ	CMN	CMP	DBG	EOR
LDC	LDC2	LDMIA	LDMDB	LDR	LDRB
LDRBT	LDRD				LDRH
LDRHT	LDRSB				LSL
LSR	MCR				MLA
MLS	MOV				MRRC
MRRC2	MUL	MVN	NOP	ORN	ORR
PLD	PLI	POP	PUSH	RBIT	REV
REV16	REVSH	ROR	RRX	RSB	SBC
SBFX	SDIV	SEV	SMLAL	SMULL	SSAT
			STC	STC2	STMIA
			STMDB	STR	STRB
			STRBT	STRD	STREX
			STREXB	STREXH	STRH
			STRHT	STRT	SUB
			SXTB	SXTH	TBB
			TBH	TEQ	TST
			UBFX	UDF	UDIV
			UMLAL	UMULL	USAT
			UXTB	UXTH	WFE
			WFI	YIELD	IT

Advanced data processing
bit field manipulations

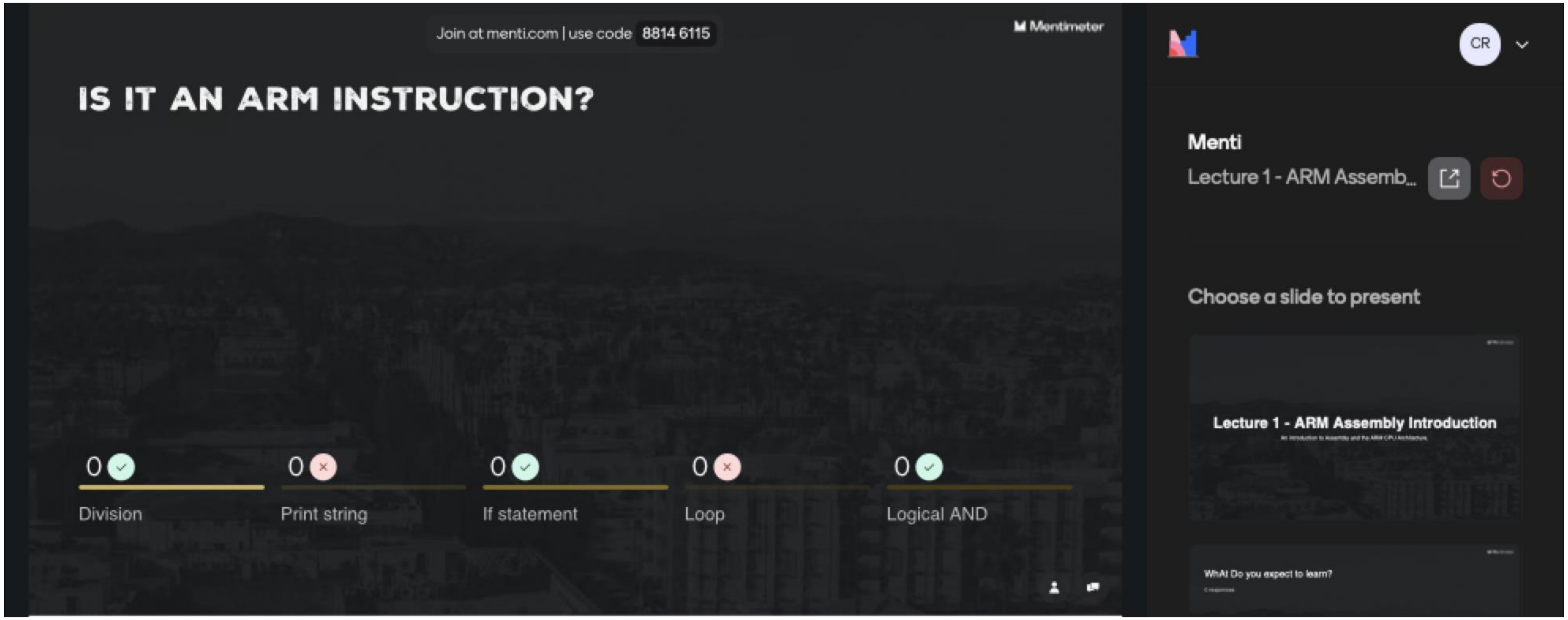
ADC	ADD	ADR	AND	ASR	B
BIC	BKPT	BL		BLX	BX
CMN	CMP	CPS	DMB		EOR
DSB		ISB		LDMIA	LDR
REV16	REVSH	ROR	RSB	SBC	SEV
STMIA	STR	STRB	STRH	SUB	SVC
SXTB	SXTH	TST	UDF	UXTB	UXTH
WFE	WFI	YIELD			

General data processing
I/O control tasks

Cortex-M0+

Cortex-M3

Instruction examples



Join at menti.com | use code 8814 6115

IS IT AN ARM INSTRUCTION?

0 ✓	0 ✗	0 ✓	0 ✗	0 ✓
Division	Print string	If statement	Loop	Logical AND

Menti

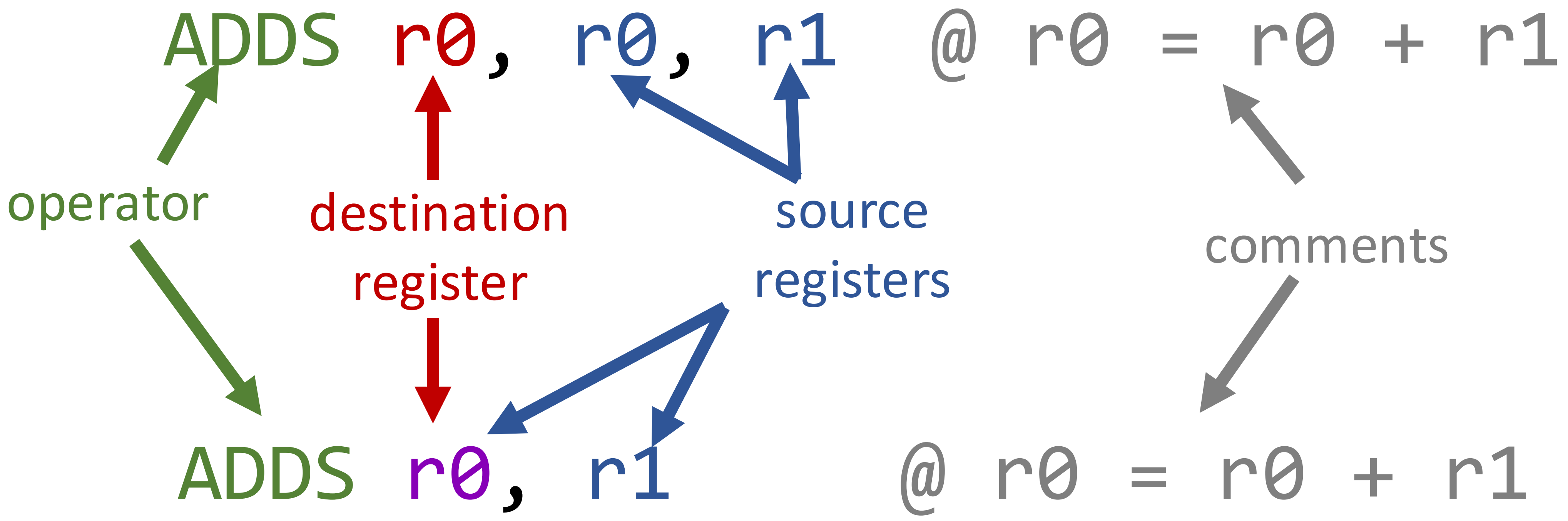
Lecture 1 - ARM Assemb...

Choose a slide to present

Lecture 1 - ARM Assembly Introduction

What Do you expect to learn?

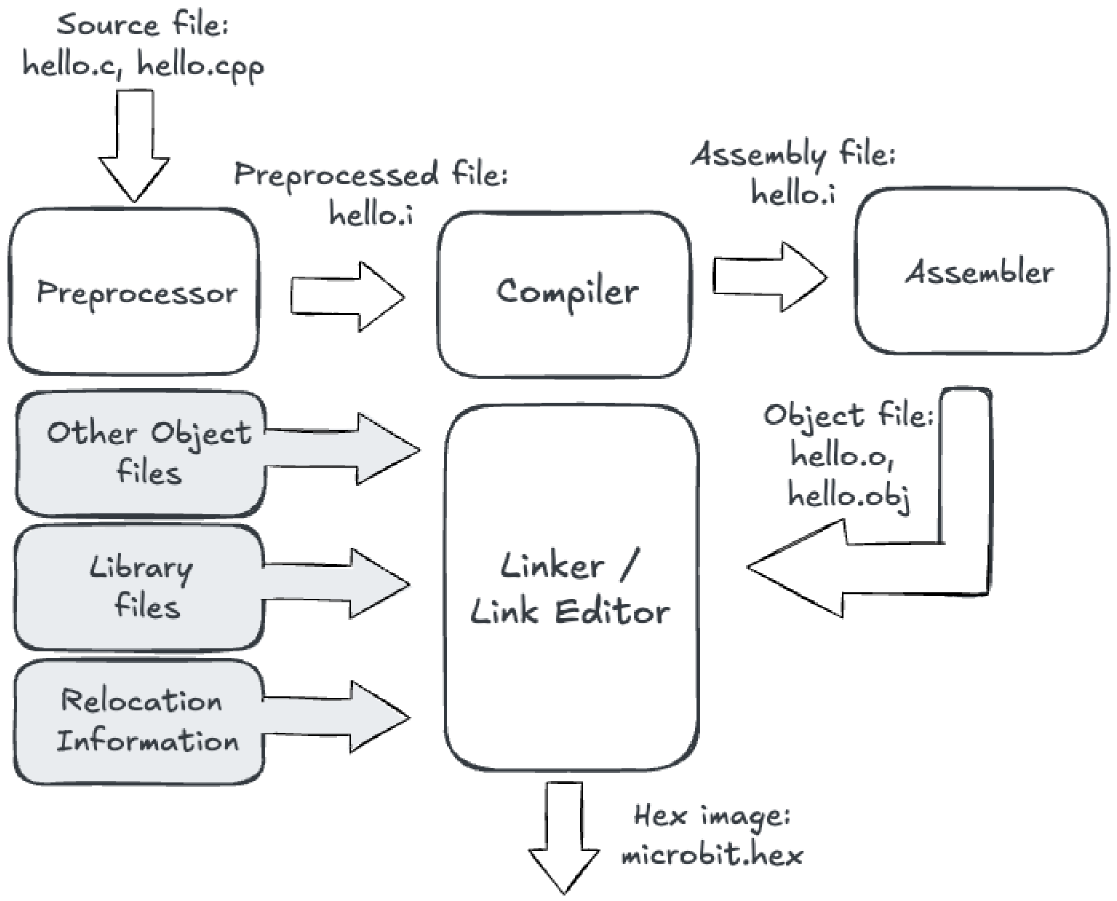
Instruction Format



```
ADDS r0, r0, r1  @ r0 = r0 + r1
ADDS r0, r1      @ r0 = r0 + r1
```

Assembler

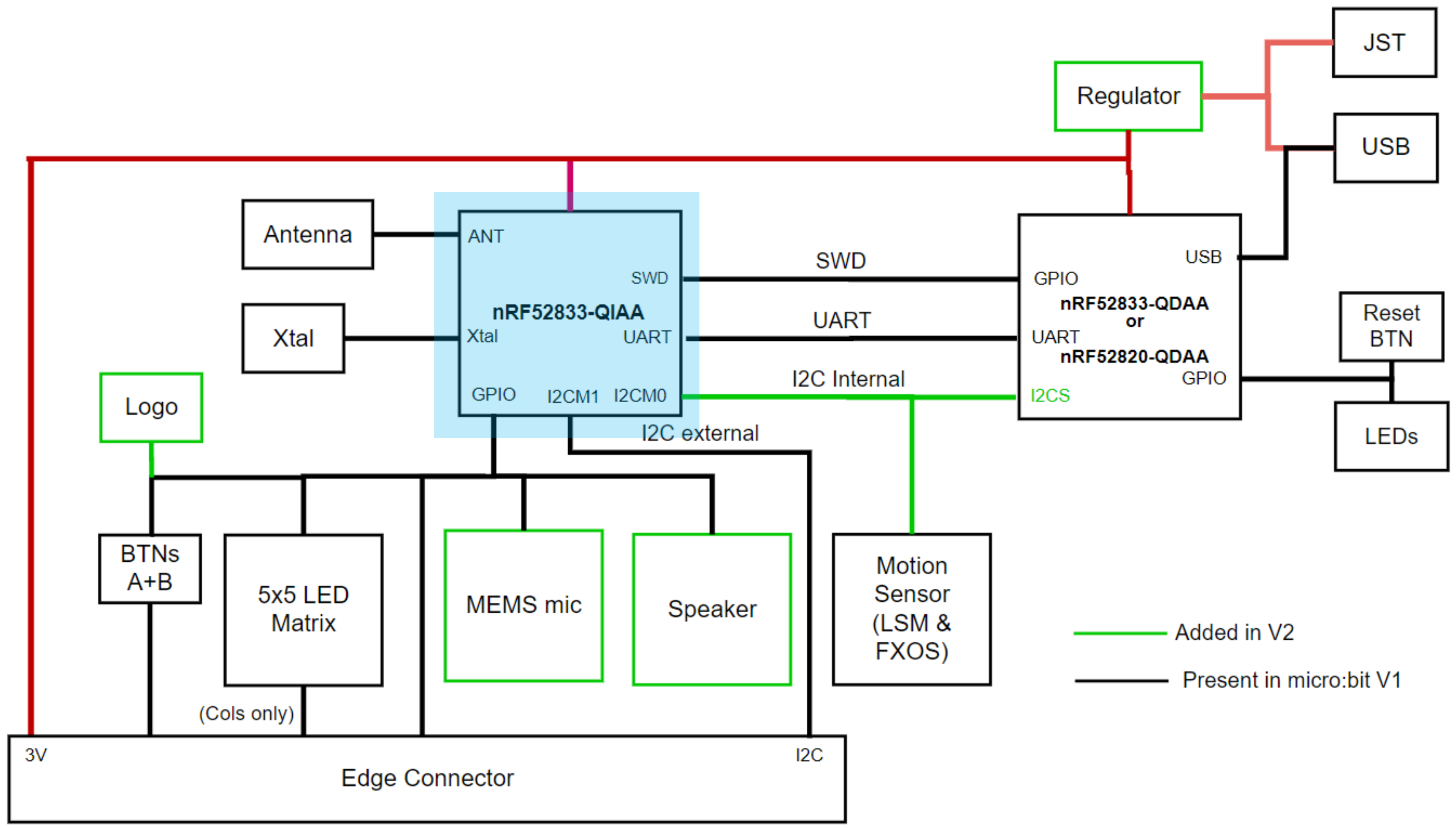
- You will write assembly in a human-readable format.
  - The CPU will need a binary format for execution.
- The **Assembler** converts human-readable assembly into binary.
  - Assemblers are platform-specific.
- Unlike the compiler, the assembler does not do any smart decision-making.
  - One-to-one match between instruction and 16-bit binary format.
- **Pseudo-instructions** are assembler instructions converted into multiple statements by the assembler.
  - Simplify code writing.

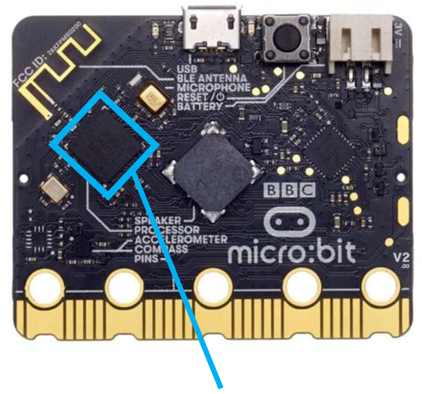


Directives

- Directives are assembler directions or setting changes.
  - Directives are not instructions; *not included in machine code*.
  - Begin with a "."
  - Must exist on separate lines from other assembler directives or instructions.
- Directives can define symbols, Data & regions, control flows, generate reports and define assembly parameters.
- https://developer.arm.com/documentation/dui0802/b/Directives-Reference/Alphabetical-list-of-directives

Running assembly code





ARM Cortex-M4 32-bit processor

Use DAPLink to load your code into the ARM Cortex flash memory and start execution.

A sample program in memory

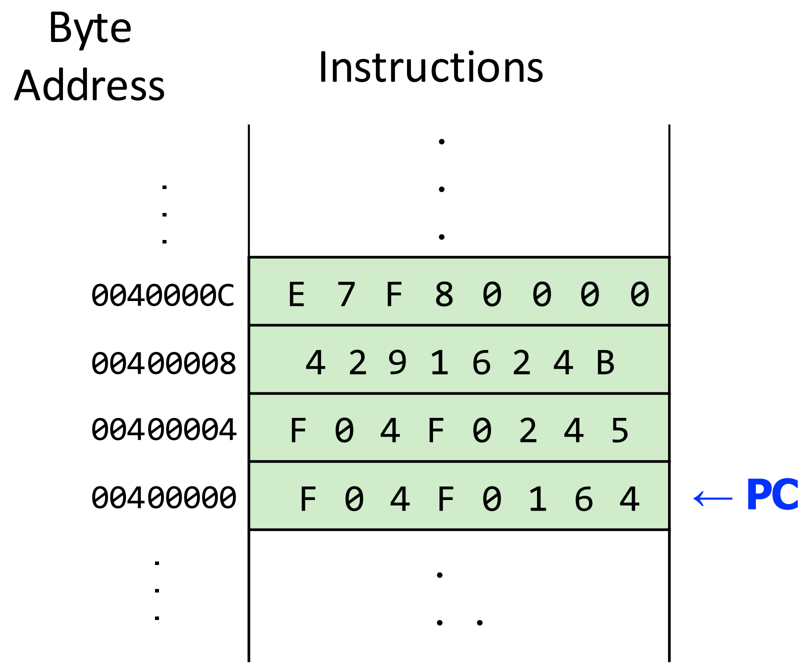
- A sample ARM program
  - 4 instructions stored in consecutive words in memory
    - No need to understand the program now. We will get back to it

ARM assembly code

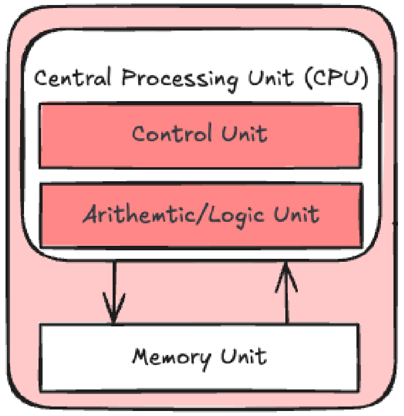
```
MOV     R1,   #100
MOV     R2,   #69
CMP     R1,   R2
STR     R3,   [R1, #0x24]
NOP
```

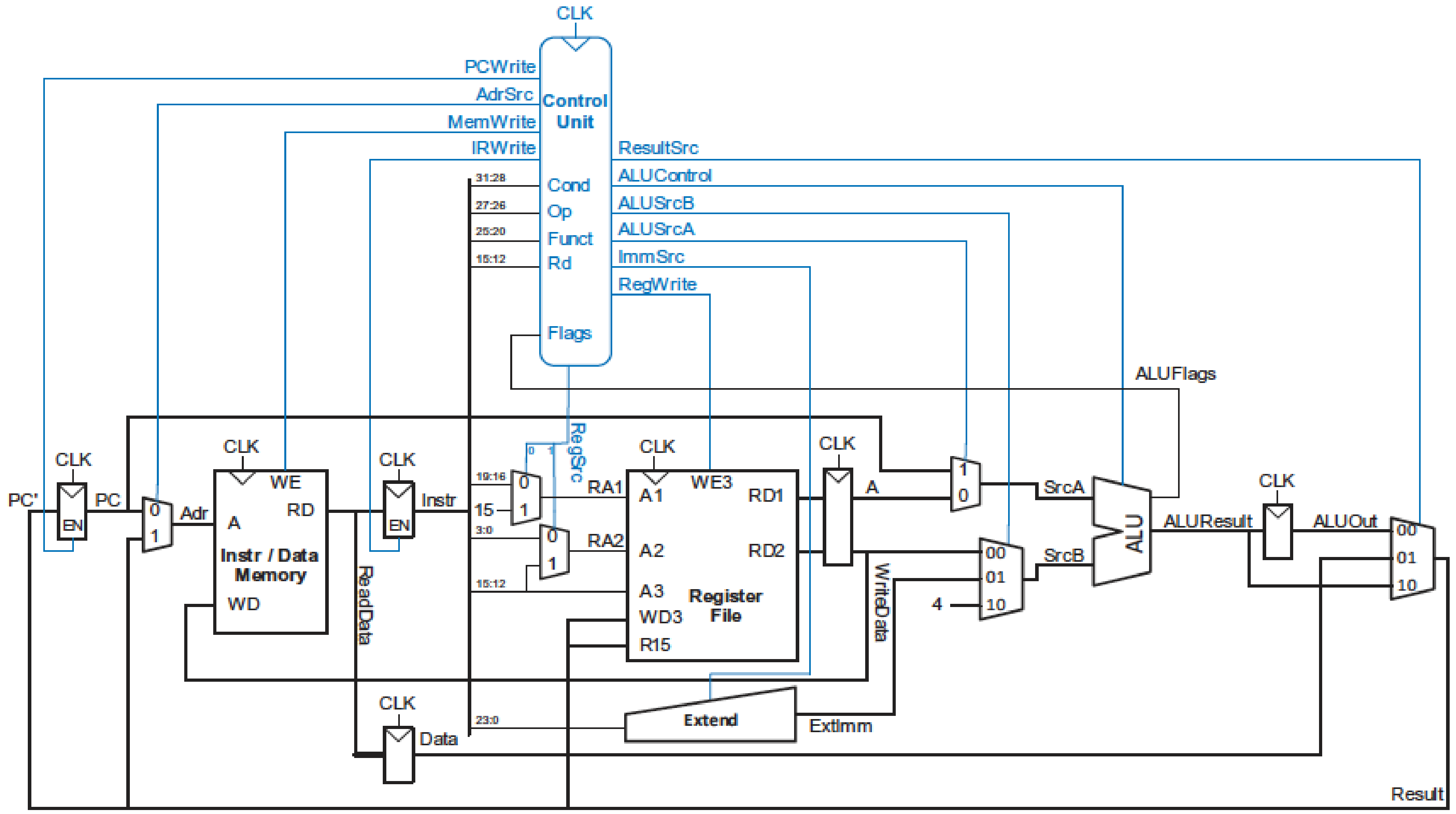
Machine code (binary code)

```
0xF04F0164
0xF04F0245
0x4291
0x624B
0xE7F8
0x0000
```

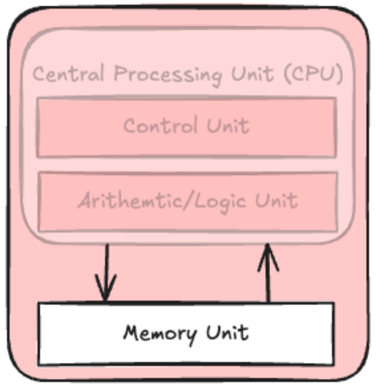


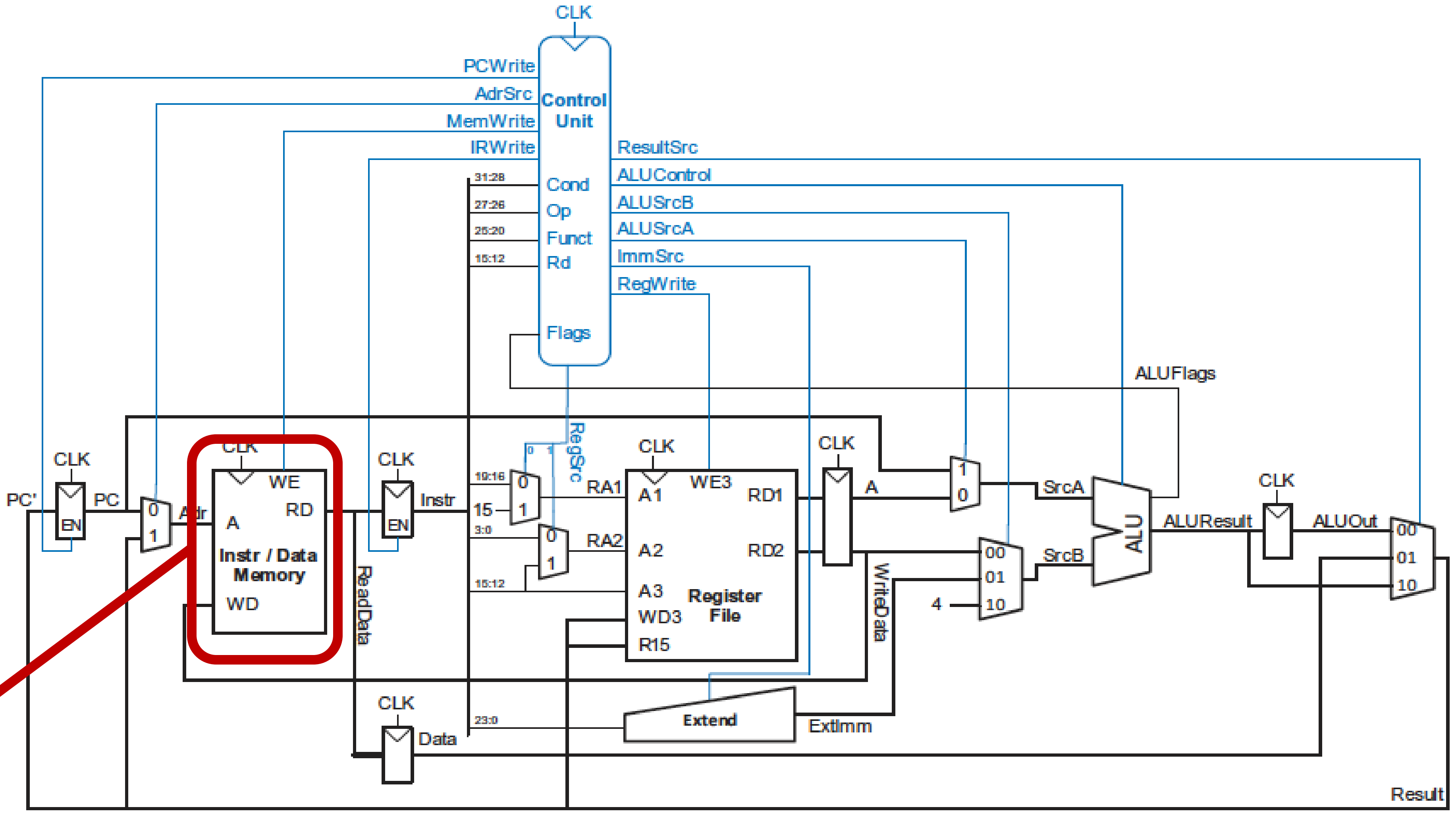
ARMv4 (Multi-Cycle) 32-bit





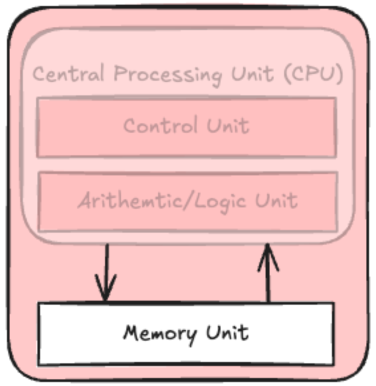
ARMv4 (Multi-Cycle) 32-bit - Memory

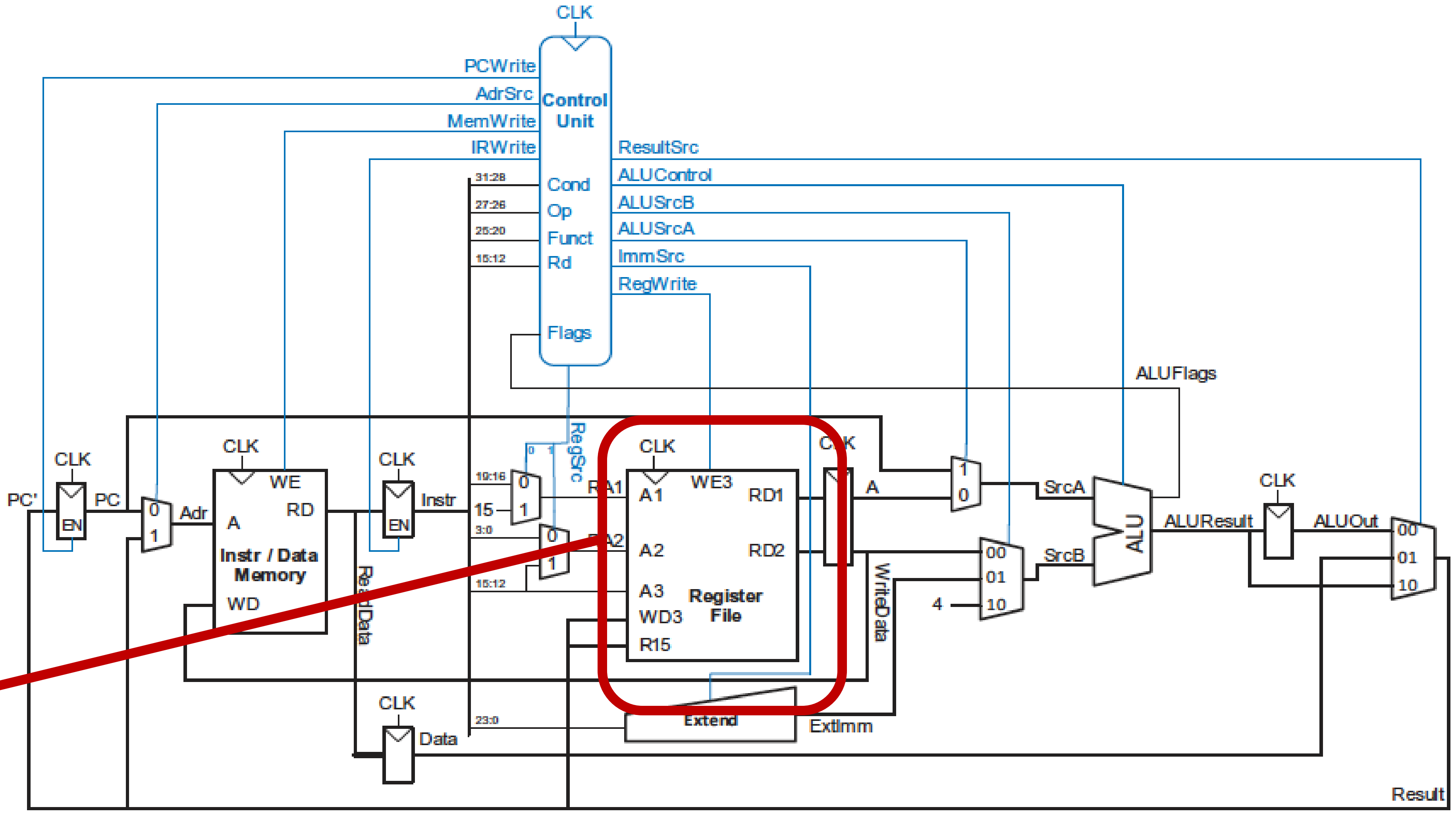




Reads data from memory (instructions and data)

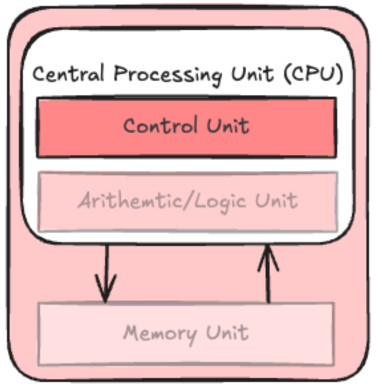
ARMv4 (Multi-Cycle) 32-bit – Registers

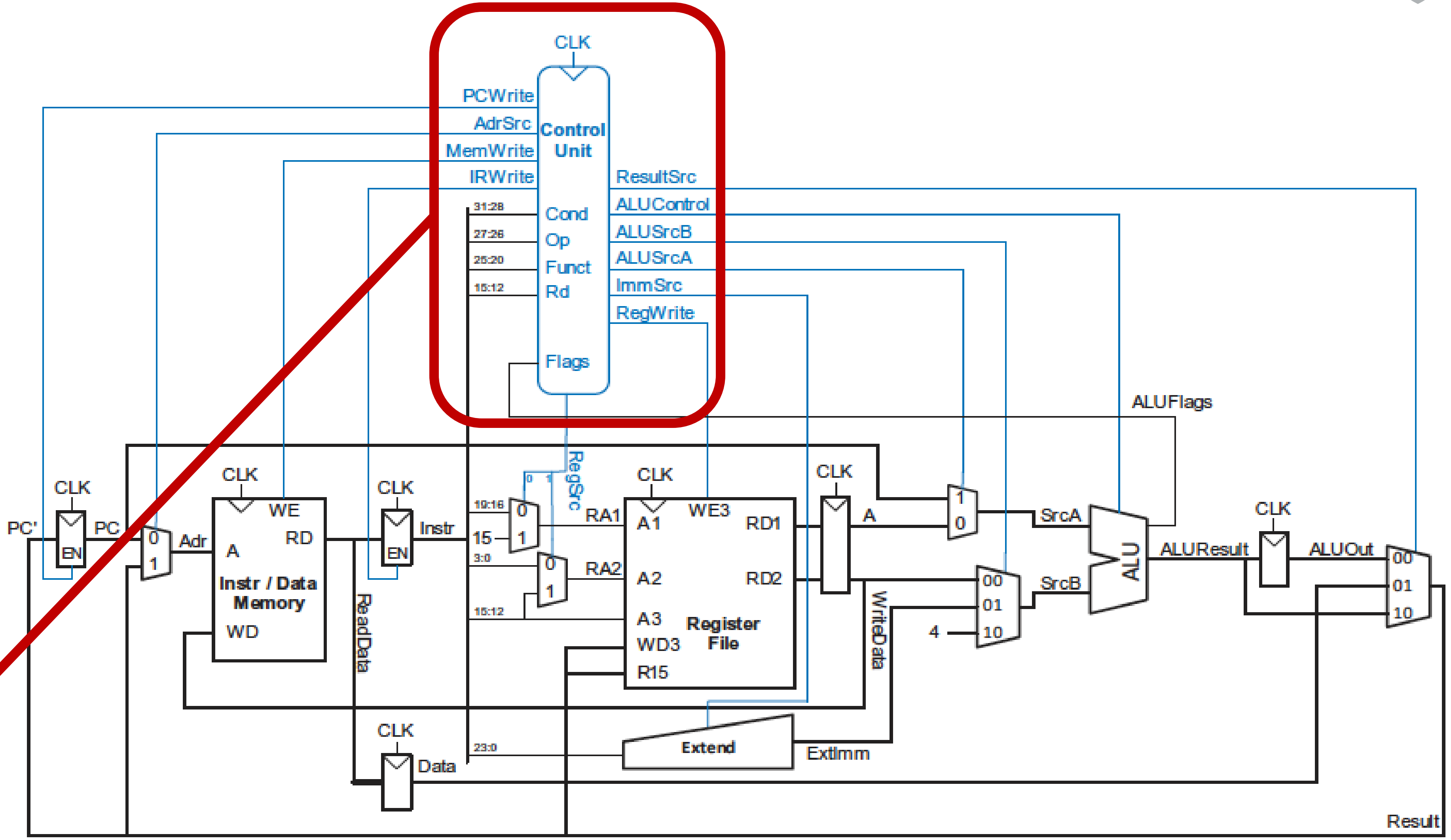




Manages the registers.

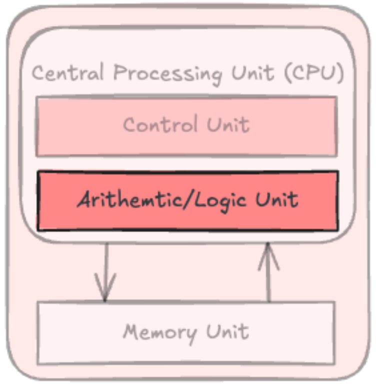
ARMv4 (Multi-Cycle) 32-bit – Control Unit

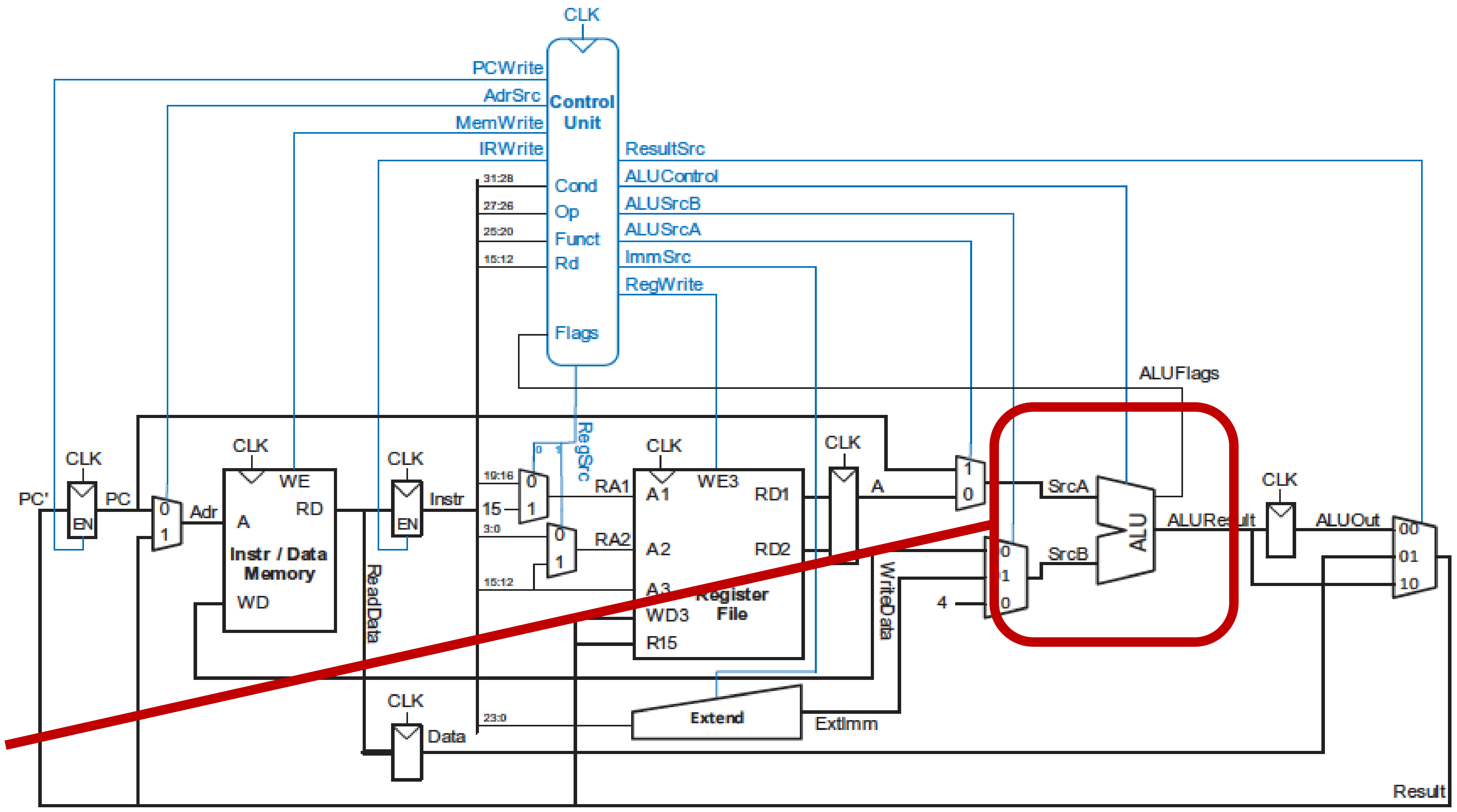




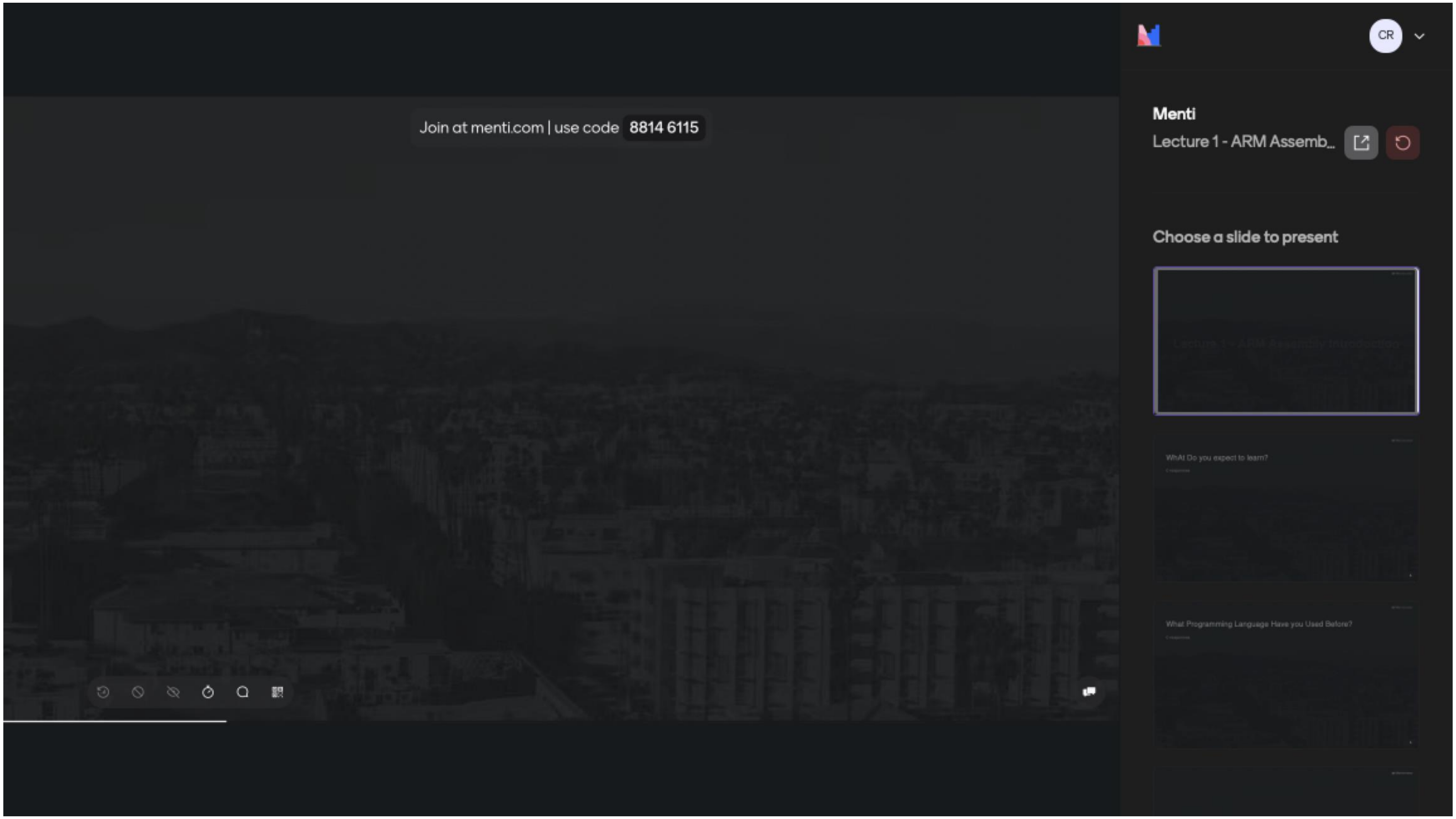
Decides which instruction to execute next.

ARMv4 (Multi-Cycle) 32-bit – ALU





Executes instruction



Join at menti.com | use code 8814 6115

Menti

Lecture 1 - ARM Assemb...

Choose a slide to present

Lecture 1 - ARM Assembly Introduction

WhAt Do you expect to learn?

What Programming Language Have you Used Before?

Summary

Introduction to Assembly

ARM CPU details

Next

- Arithmetic Operators