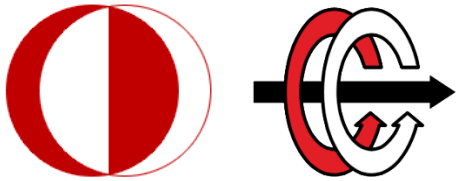
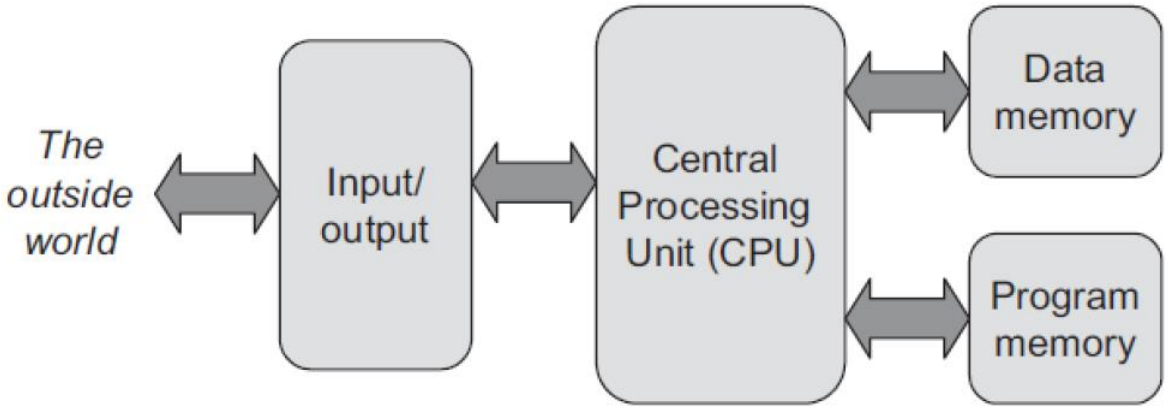


Microprocessor vs Microcontroller, Memory concepts, address decoding

Week 1

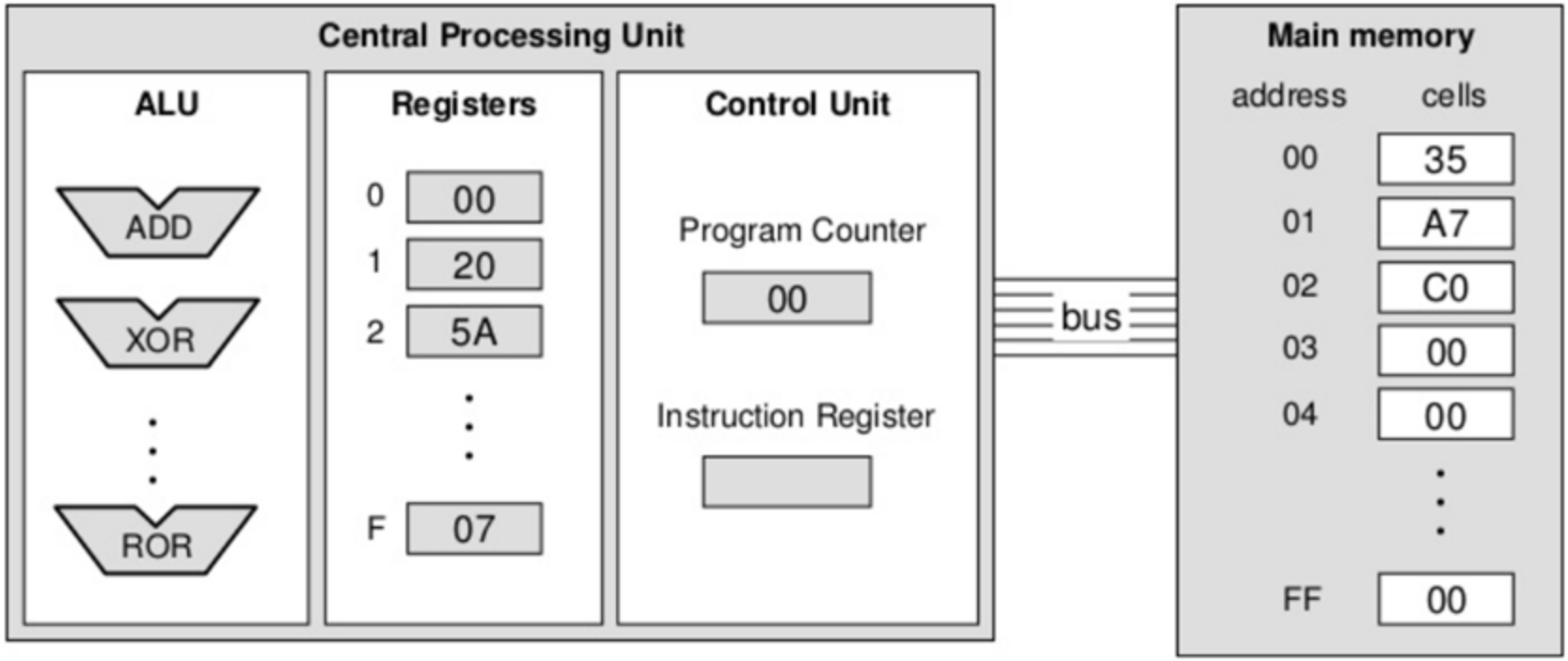
General Features

- CPU performs computations
- Program memory stores the instructions of the desired program
- Data memory stores application data values
- Input/output block for providing input data and reading output data
- Arrows represent data exchange



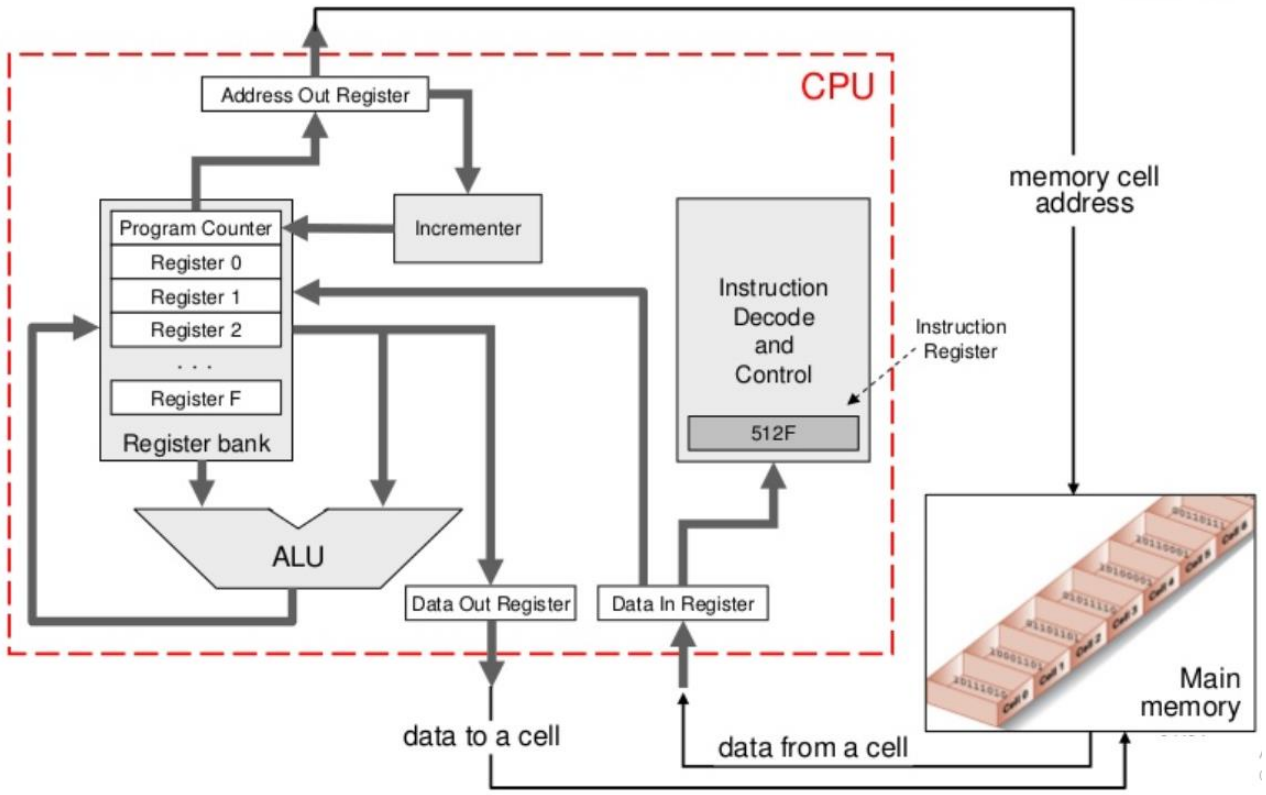
Microprocessor: Overview

- Central Processing Unit (CPU)
  - A single LSI chip, known as to perform all computation (without memory and I/O interface circuit)
  - ALU (arithmetic logic unit), registers, control unit, internal busses



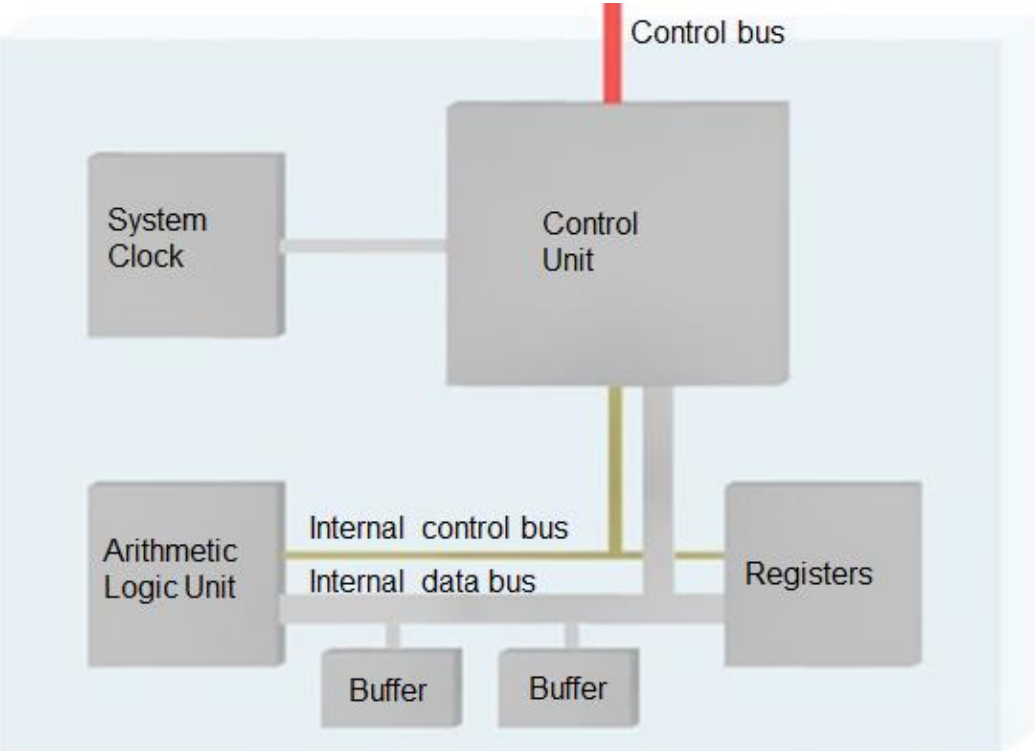
Microprocessor: Common Number Systems

- Stored Program Concept
  - Memory stores the program instructions



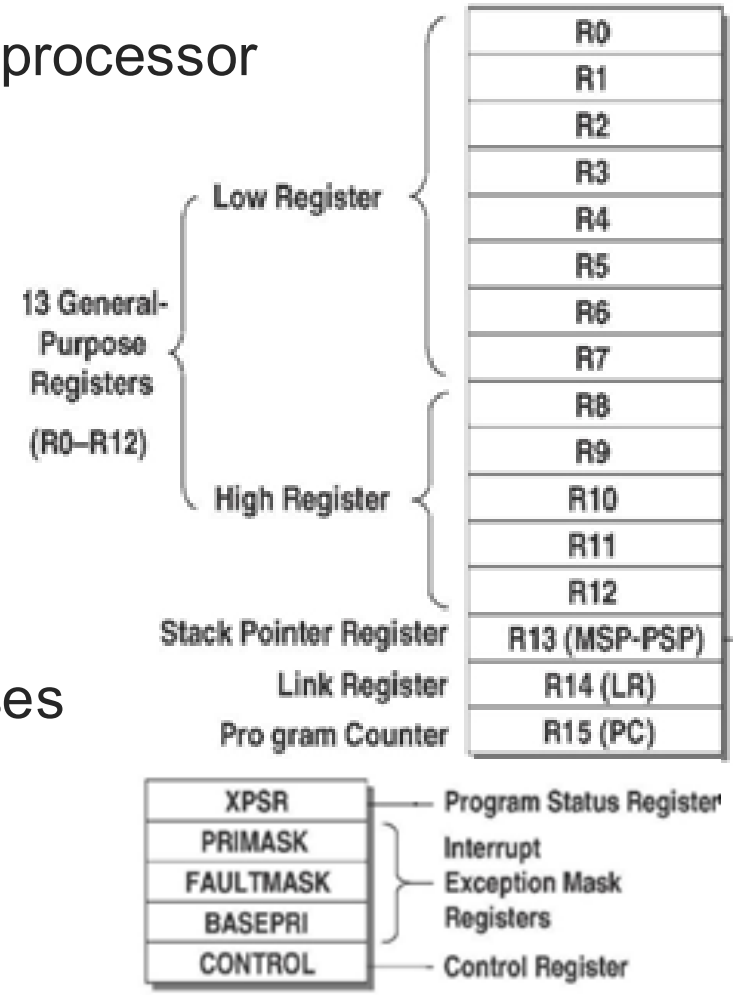
Microprocessor: Control Unit

- Main Task
  - Determine which operation should be executed
  - Configure the data path accordingly



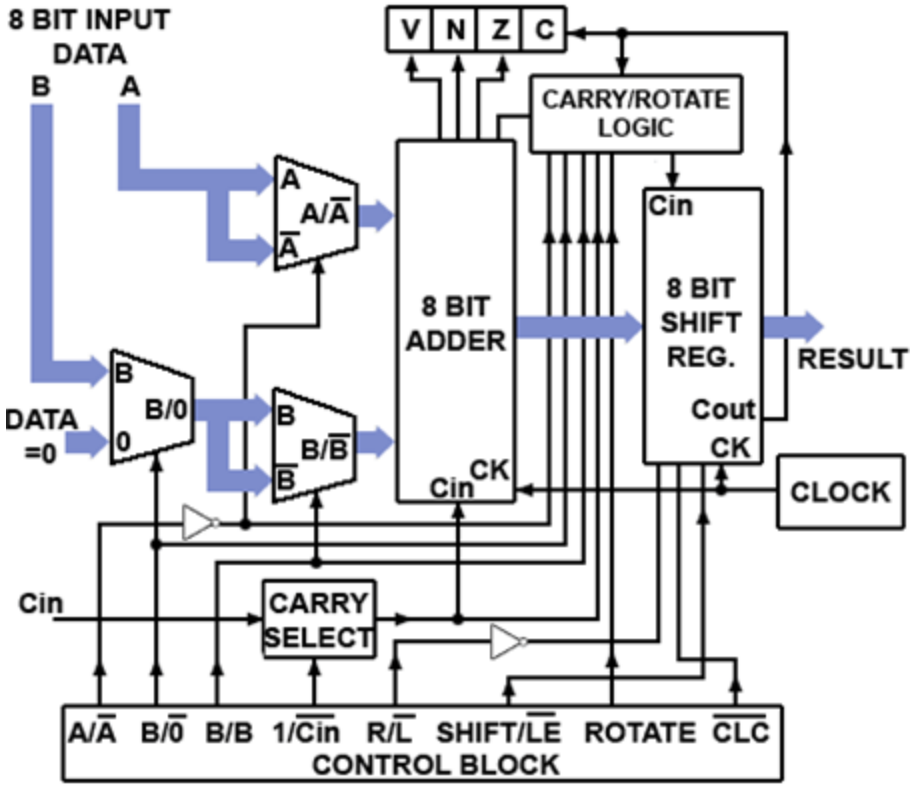
Microprocessor: Registers

- Most fundamental storage area in the processor
  - Closely located to the processor
  - Provide very fast access, operating at the same frequency as the processor clock
  - Limited quantity (typically less than 100)
- Most are of the general purpose type and can store any type of information
  - Data – e.g., timer value, constants
  - Address – e.g., ASCII table, stack
- Some are reserved for specific purposes
  - Program counter (r15 in ARM)
  - Program status register (XPSR in ARM)



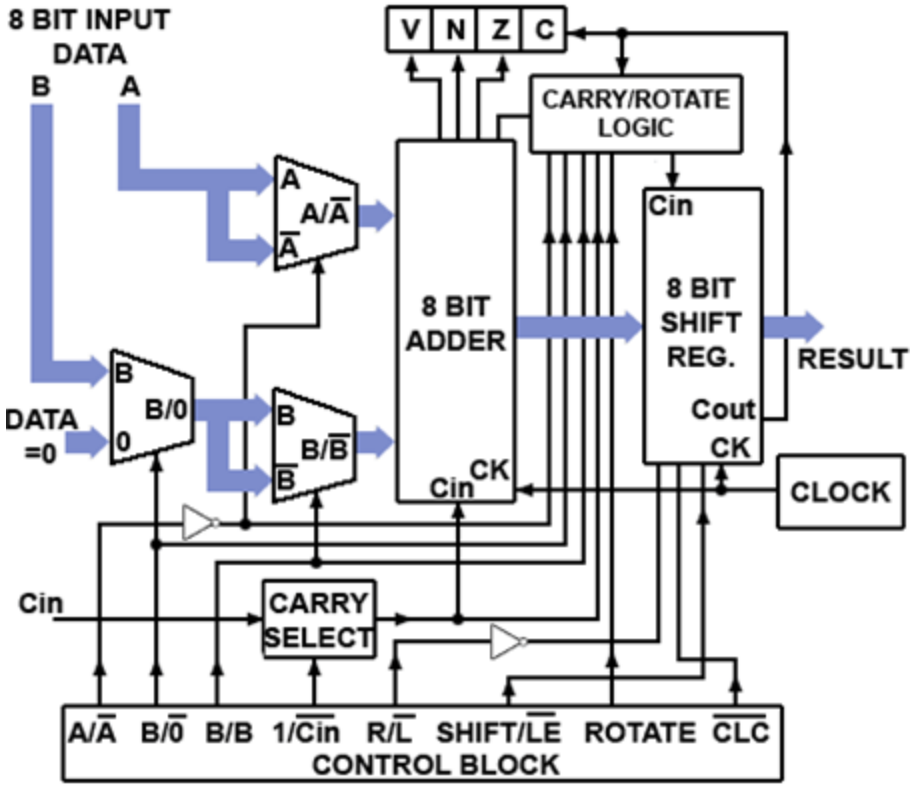
Microprocessor: Arithmetic Logic Unit (ALU)

- Main Tasks
  - Performs operations on inputs
  - Returns the result as its output
  - Source and destination are registers or memory
- Status Register
  - ALU stores information about the result in the status register
  - Z (Zero) bit, N (Negative) bit, O (Overflow) bit, C (Carry) bit



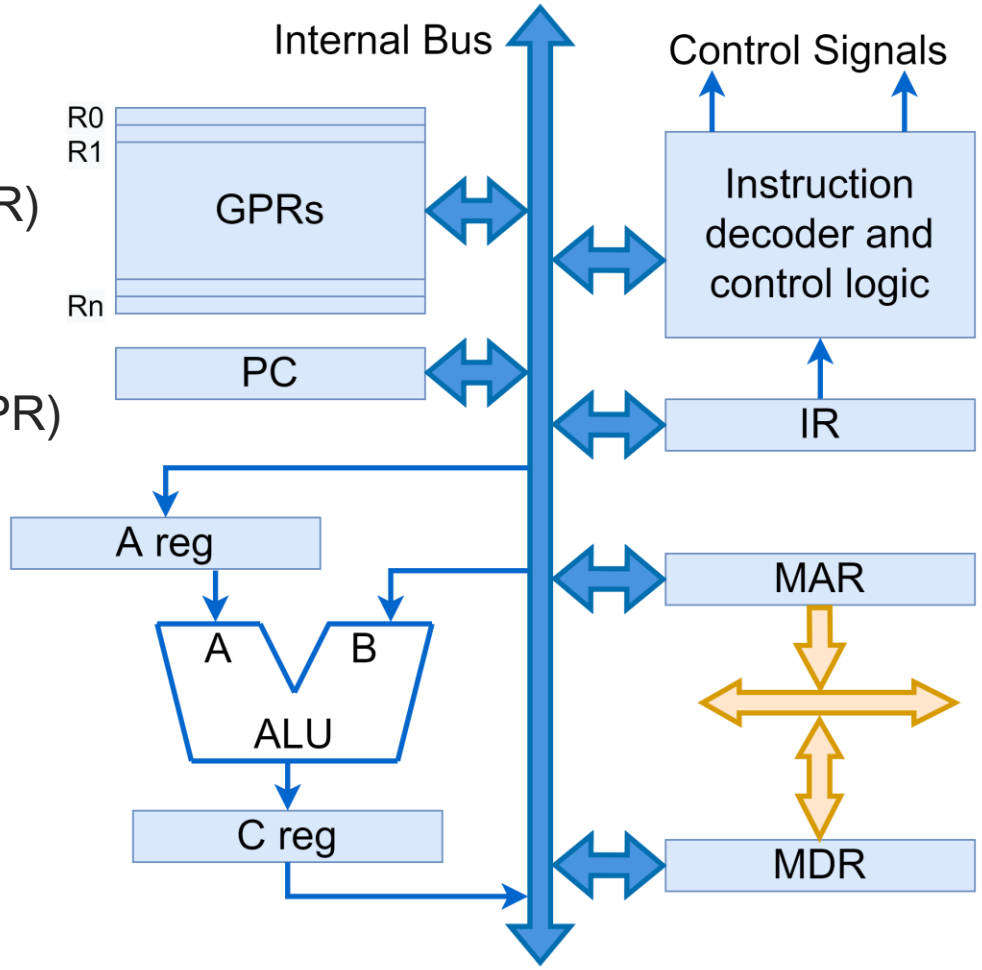
Microprocessor: Arithmetic Logic Unit (ALU)

- Example: Functions of Conventional Binary Adder
  - Assume that the ALU inputs are A and B
  - A + B: Normal adder output (if carry-in is zero)
  - A - B: Implemented as A + B' + 1 (B input is inverted and carry-in is forced to 1)
  - B: Implemented by forcing A to 0 and carry-in to 0
  - B + 1: Implemented by forcing A to 0 and carry-in to 1



Microprocessor: Data Path and Microarchitecture

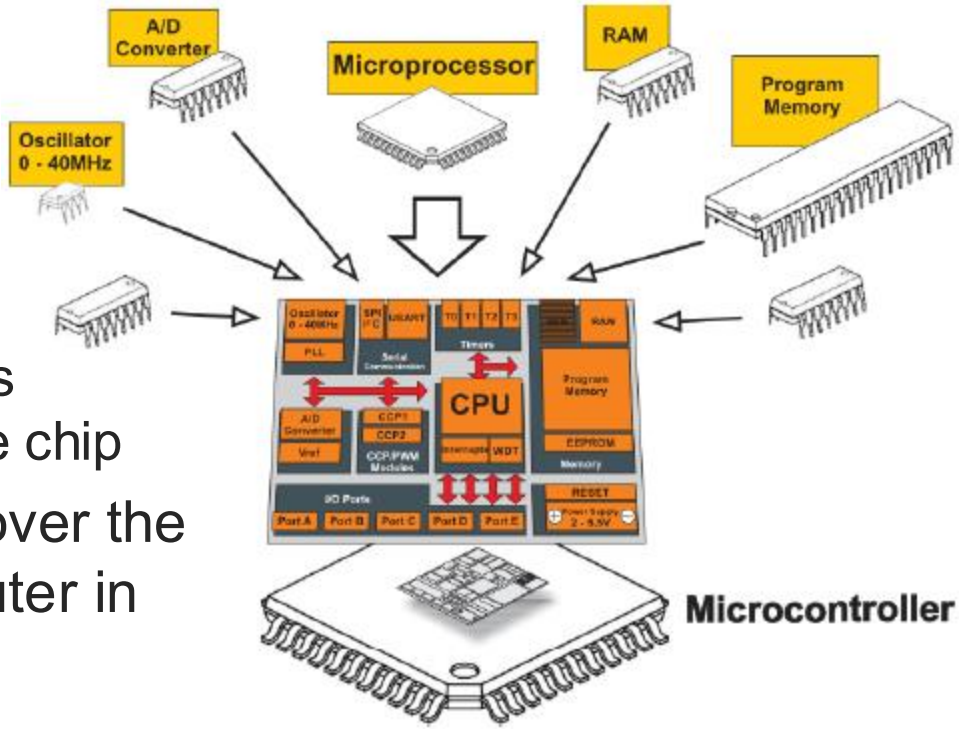
- Single Internal Bus
  - Memory data register (MDR)
  - Memory address register (MAR)
  - Program counter (PC)
  - Instruction register (IR)
  - General purpose registers (GPR)



Microcontroller: Explanation

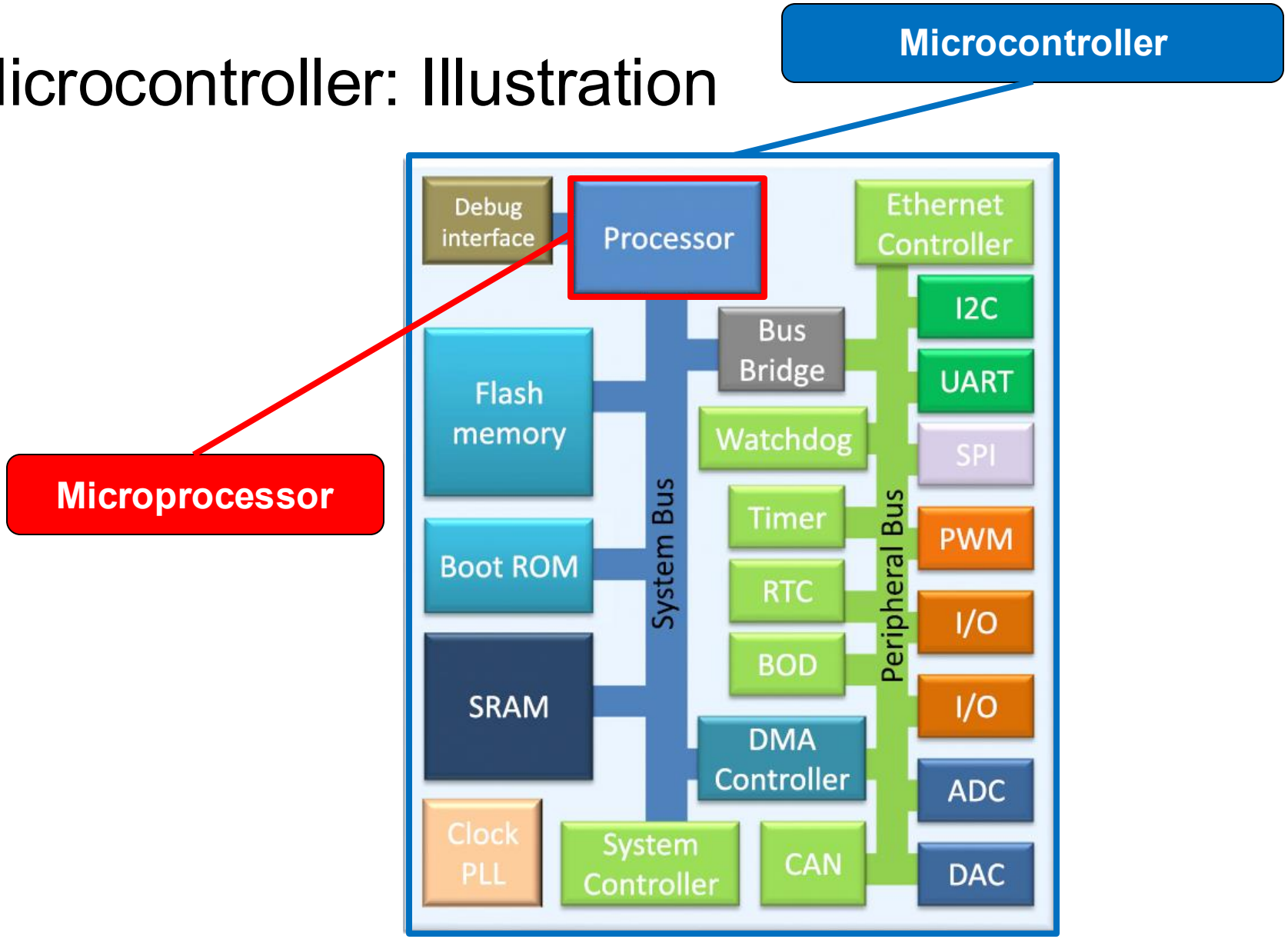
- Components of a Microcontroller (MCU)
  - Microprocessor
  - Memory
  - I/O, and other
  - Necessary functionalities for control activities mainly

  → MCU: all these components are integrated onto a single chip



- Today, the MCU has taken over the role of the embedded computer in embedded systems
- Important Notes
  - Microprocessor is used as a component of an MCU
  - Same microprocessor can be used in different MCUs

Microcontroller: Illustration

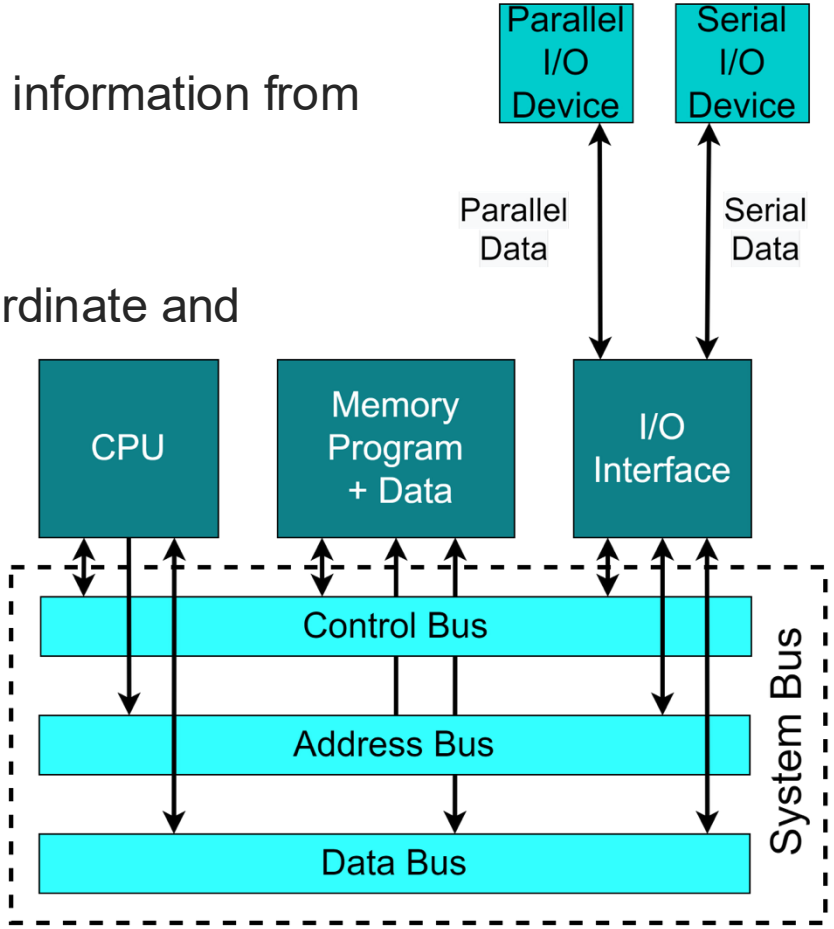


Microcontroller: Input/Output (I/O)

- Input Peripherals/Devices
  - Switches and Keypads
  - Digital and analog input ports
  - Provide binary information to the MPU
- Output Peripherals/Devices
  - LEDs and LCDs
  - Digital output ports
  - Receive binary information from the MPU
- Communication Peripherals/Devices
  - Serial Peripheral Interface (SPI)
  - Universal Asynchronous Receiver/Transmitter (UART)
  - Inter-Integrated Circuit (I^2C)
  - Etc.

Microcontroller: Bus

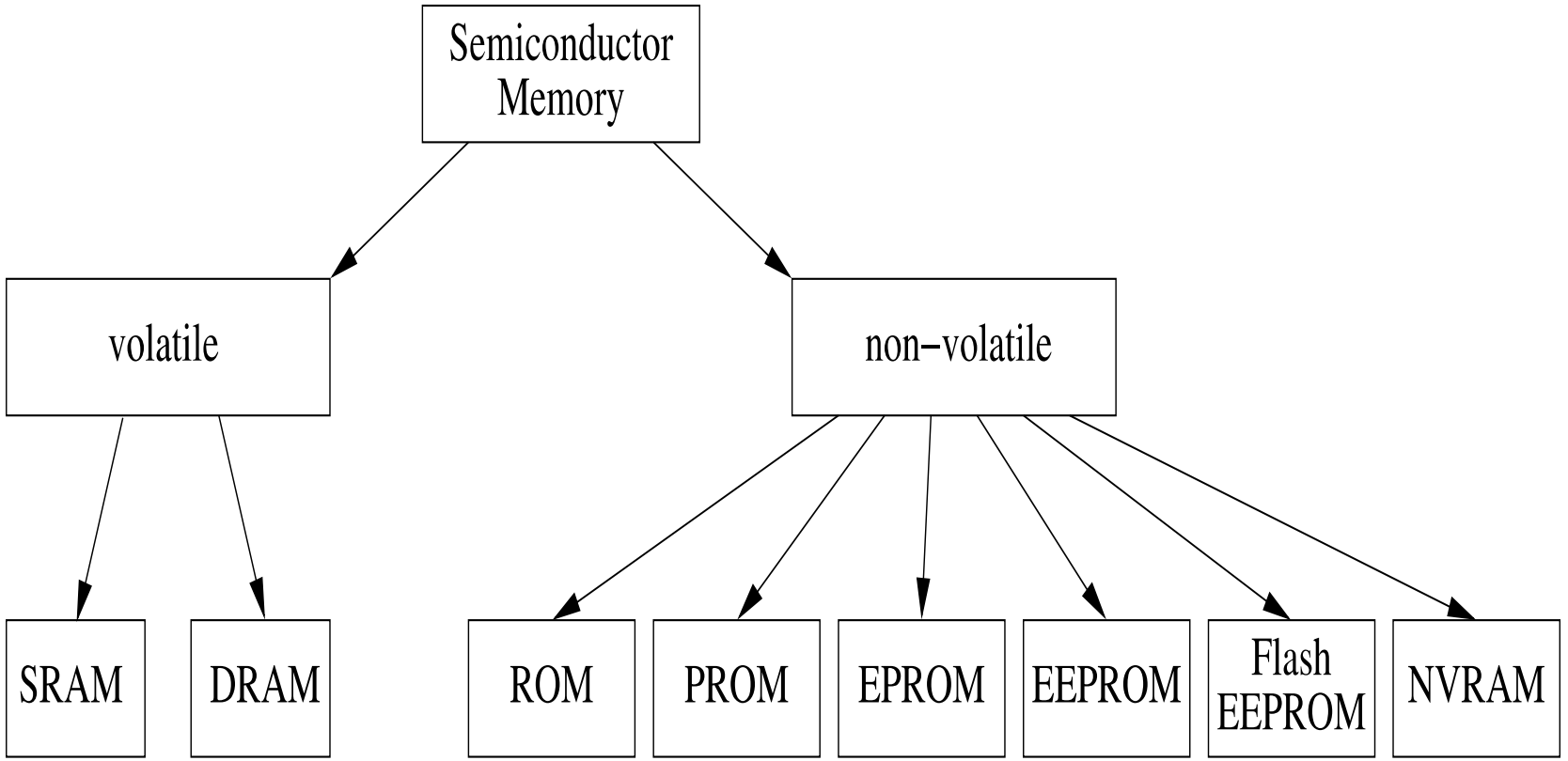
- Bus Explanation
  - A set of signal lines that carry digital information from source to destination
- Control Bus
  - Collection of control signals that coordinate and synchronize the whole system
- Address Bus
  - Carries the address of a unique memory location or I/O device
- Data Bus
  - Carries data to or from the CPU



Microcontroller: Memory

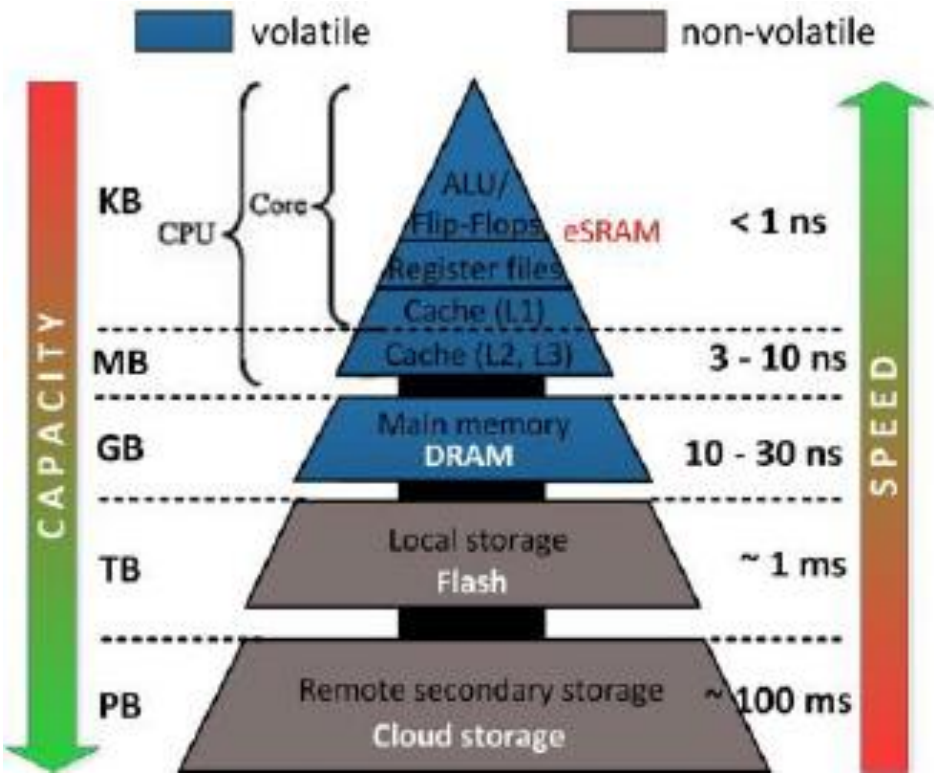
- Memory in a Computer System
  - Stores the data and instructions of the programs
  - Organized as a number of locations each of which can hold the same size data value (usually a byte)
- Memory Addressing
  - Each location has a unique address
  - A 16-bit address allows for addressing 65,536 (64K) memory locations
- A transfer of data from memory is a read
- A transfer of data to memory is a write

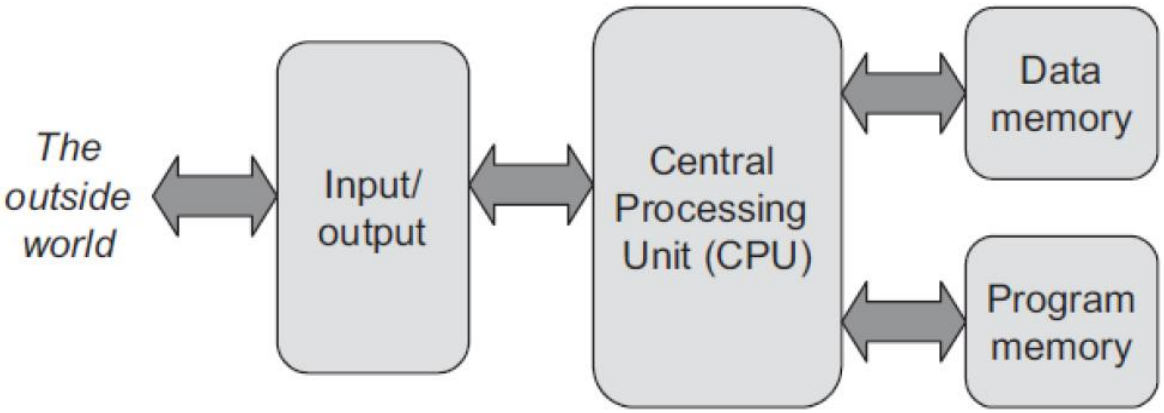
Microcontroller: Memory Types



Microcontroller: Volatile/Nonvolatile Memory

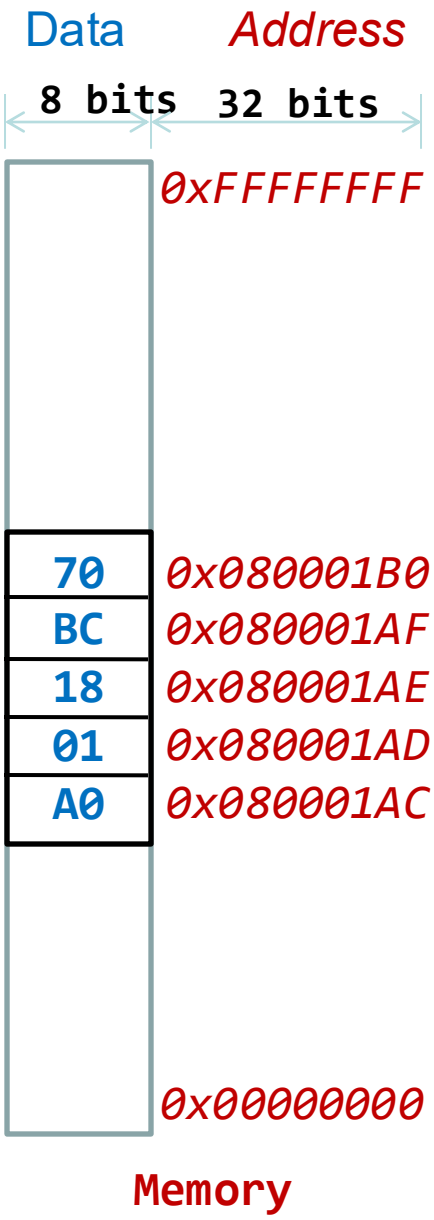
Volatile	Nonvolatile
Data fetch/store is fast and economical. It is the memory hardware that fetches/stores data at a high-speed.	It's not economical and slow in fetch/store as compared to volatile memory however stores higher volume of data.
It is also referred as temporary memory.	All such information that needs to be stored for an extended amount of time is stored in non-volatile memory.
Once the system is turned off the data within the volatile memory is deleted automatically.	Data or information is not lost within the memory even power is shut-down.
RAM (Random Access Memory) and Cache Memory are some common examples.	ROM (Read Only Memory) is the most common example.




The outside world
Input/ output
Central Processing Unit (CPU)
Data memory
Program memory

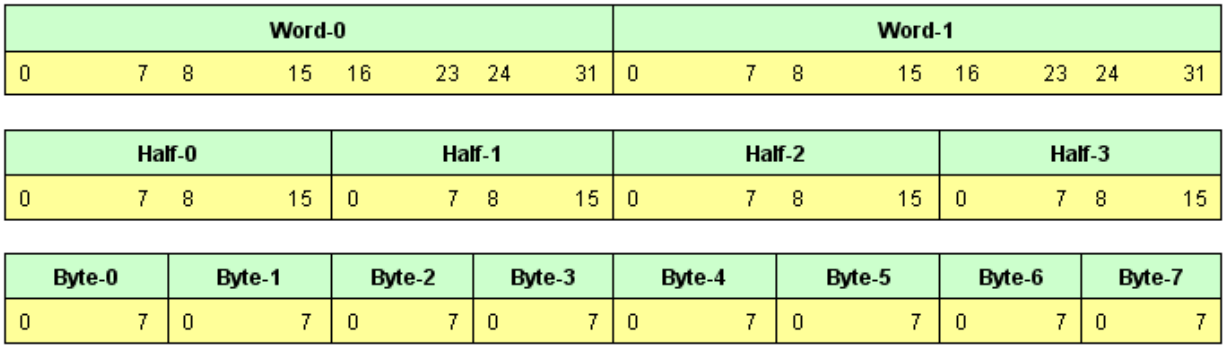
Memory: Organization

- Memory is arranged as a series of “locations”
  - Each location has a unique “address”
  - Each location holds a Byte (Byte-addressable)
  - Example: Memory location at address 0x080001B0 contains the byte value 0x70, i.e., 112
- The number of memory locations is limited
  - e.g. 4 GB of RAM
  - 1 Gigabyte (GB) = 2^{30} Bytes
  - 2^{32} locations $\rightarrow$ 4,294,967,296 locations!
- Values stored at each location can represent either program data or program instructions
  - e.g. the value 0x70 might be the code used to tell the processor to add two values together

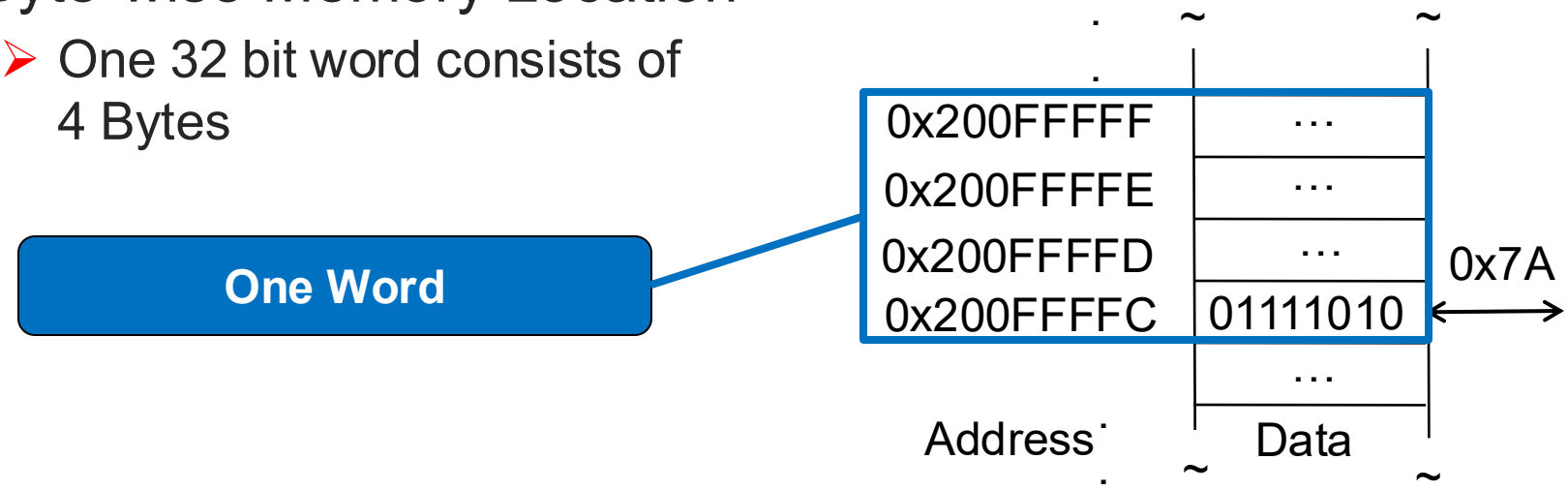


Memory: 32 bit Word

- Definition
  - Word: 32 bit
  - Half word: 16 bit
  - Byte: 8 bit
  - Nibble: 4 bit

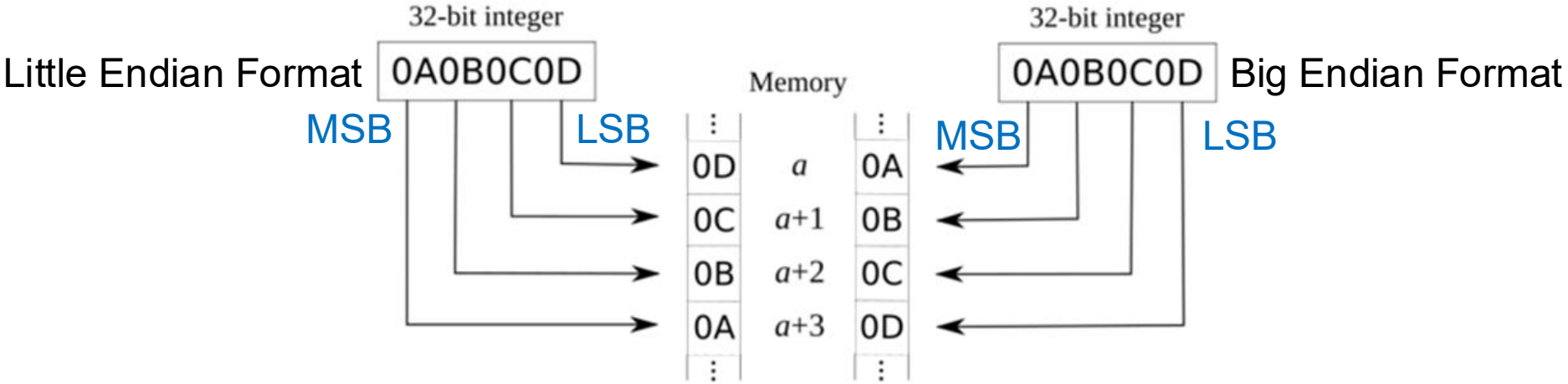


- Byte-wise Memory Location
  - One 32 bit word consists of 4 Bytes



Memory: Data Encoding

- Little Endian Format
  - Least significant byte (LSB) is stored in the lowest address of the memory
  - Most significant byte (MSB) is stored in the highest address of the memory
  - Note: ARM is Little Endian by default
- Big Endian Format
  - Least significant byte (LSB) is stored in the highest address of the memory
  - Most significant byte (MSB) is stored in the lowest address of the memory



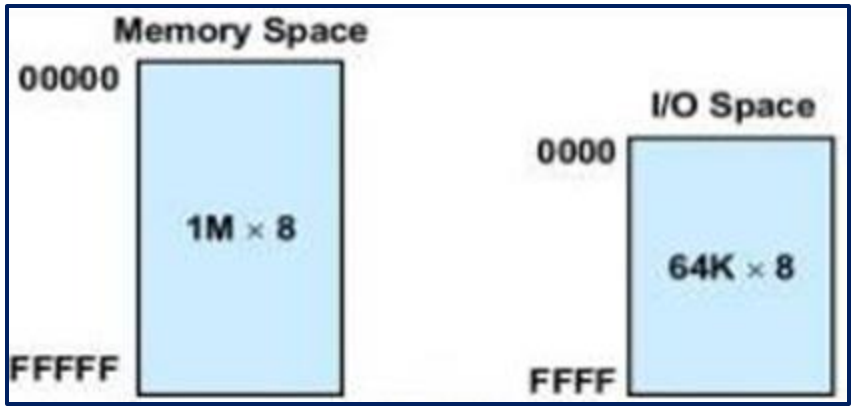
Memory: Address Space

- Address space of a processor depends on its address decoding mechanism
- Size of address space depends on the number of address bits
- There are different approaches for address space usage
  - Isolated I/O (E.g. Intel)
  - Memory mapped I/O (E.g. ARM, Motorola)

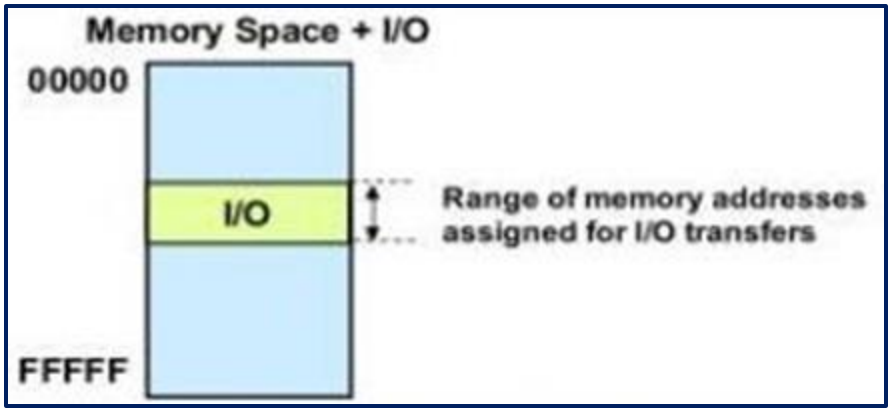
Memory: Address Space

- Isolated I/O (E.g. Intel):
  - One space is used by normal memory access
  - Another space is reserved for I/O peripheral registers (control, status, data)
  - Requires extra control signal or special means of accessing these alternate address spaces
- Memory mapped I/O (E.g. ARM, Motorola)
  - Utilize only one address space for both memory and I/O devices

Isolated I/O



Memory mapped I/O

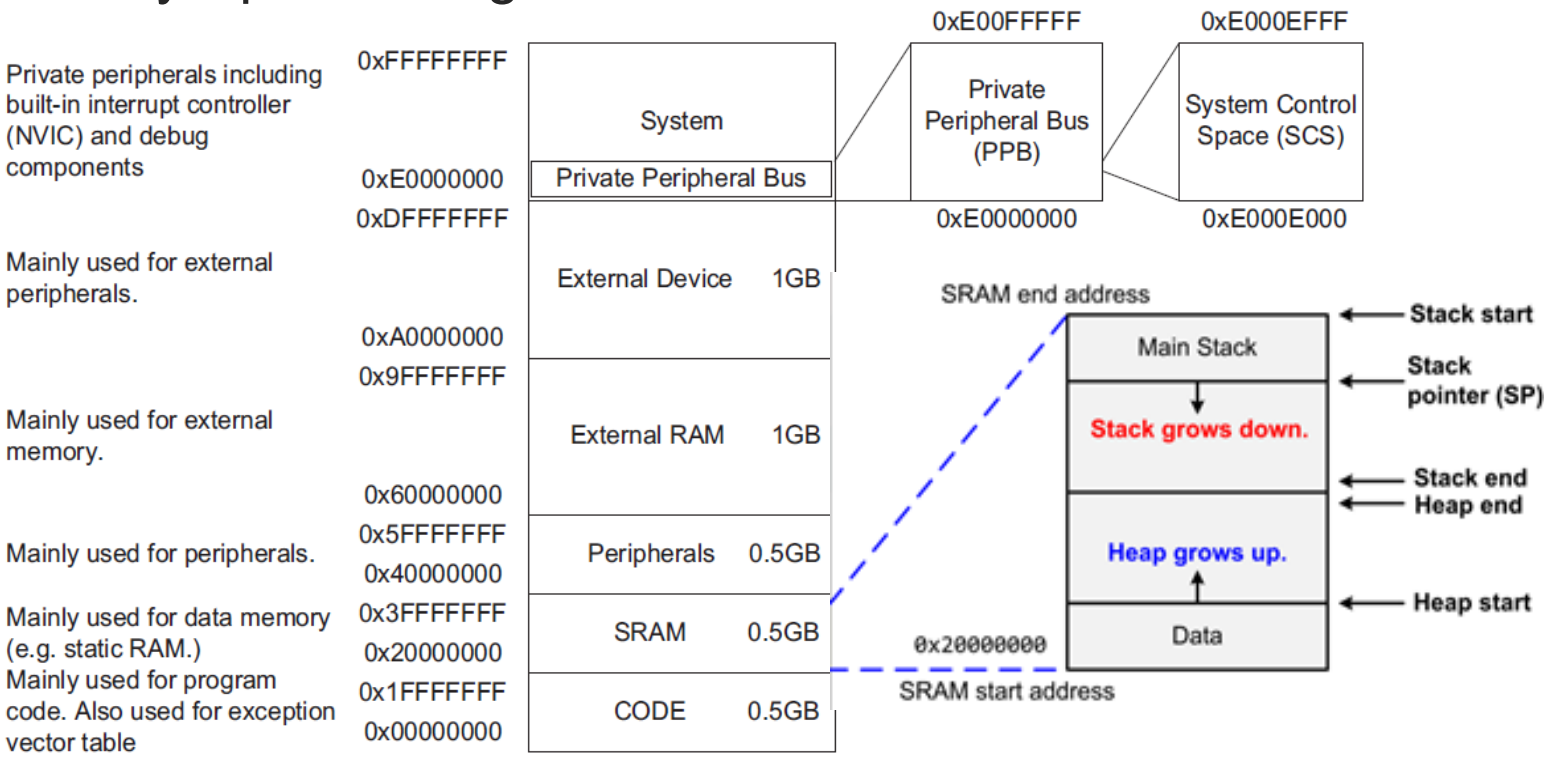


ARM Cortex-M4 Address Space

- Code memory (normally read-only memory)
  - Program instructions
  - Constant data
- Data memory (normally read/write memory – RAM)
  - Variable data/operands
- Stack (located in data memory)
  - Special Last-In/First-Out (LIFO) data structure
  - Save information temporarily and retrieve it later
  - Return addresses for subroutines and interrupt/exception handlers
  - Data to be passed to/from a subroutine/function
  - Stack Pointer register (R13/SP) points to last item placed on the stack
- Peripheral addresses
  - Used to access registers in “peripheral functions” (timers, ADCs, communication modules, etc.) outside the CPU

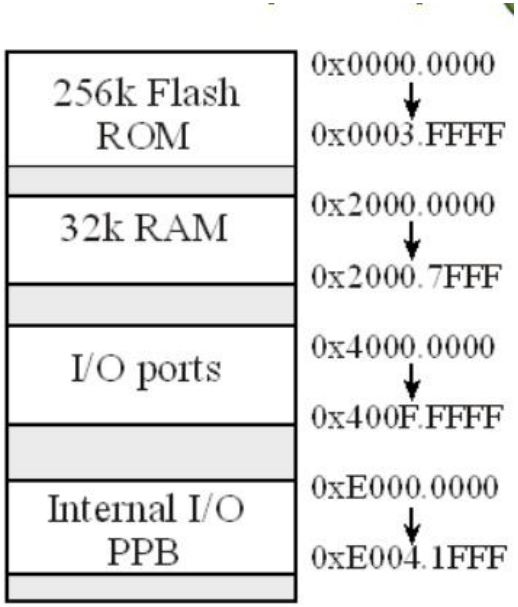
ARM Cortex-M4: Memory Map

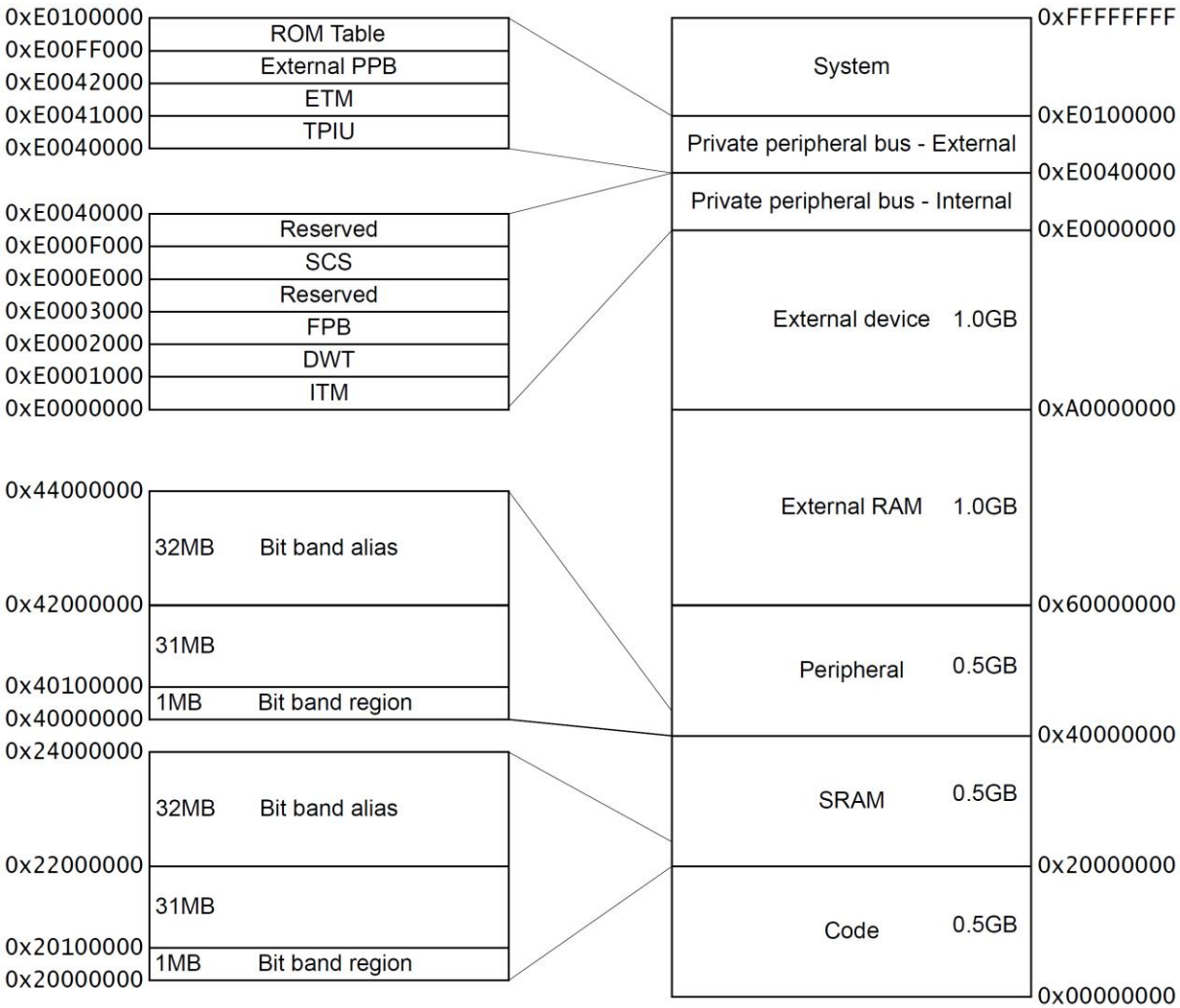
- Addressing
  - 32 bits → 4GB of memory space
- Memory Space Usage



Cortex-M4 Fixed Memory Map

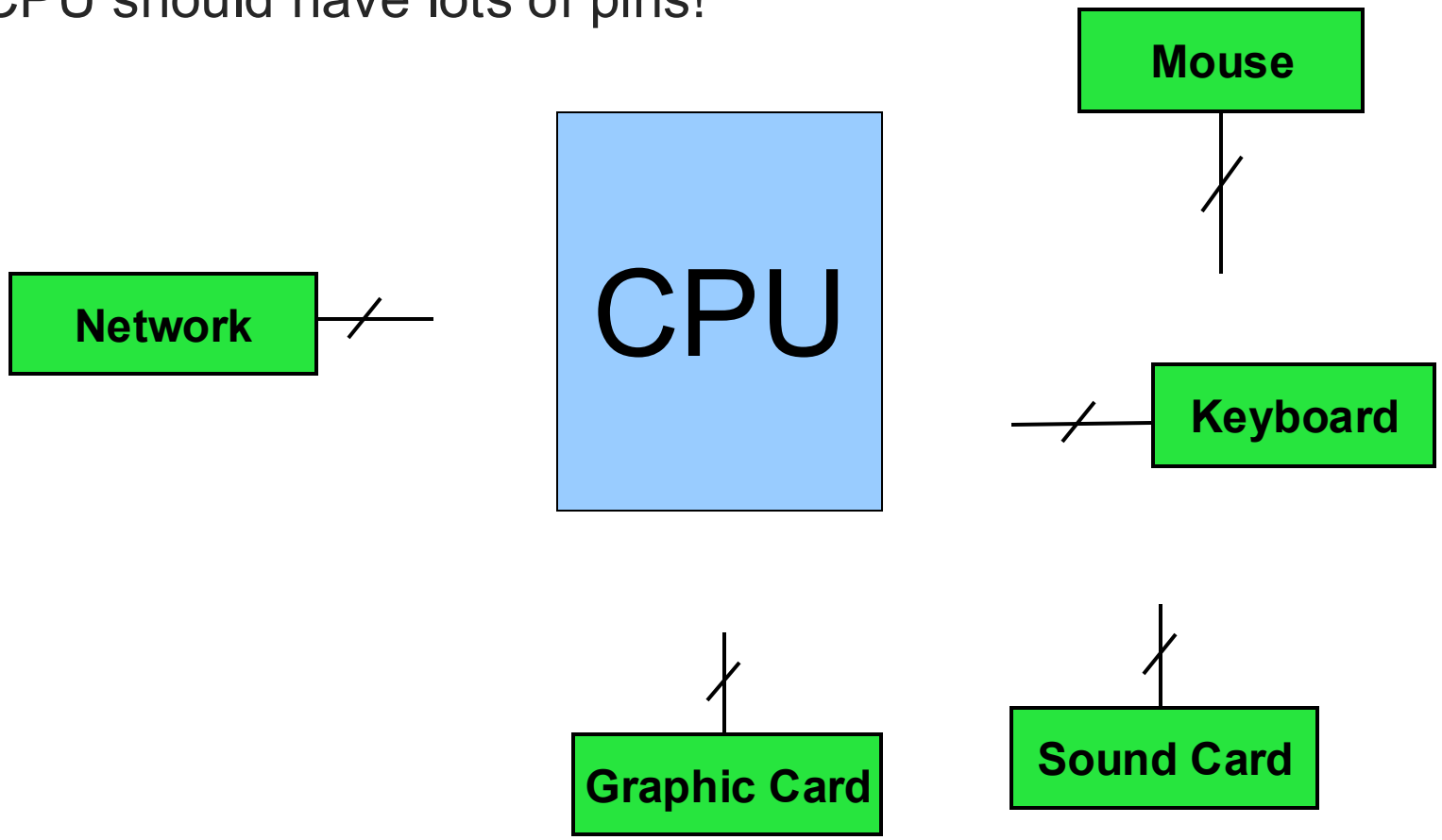
- Example: TM4C123G MCU
  - 256 KB Flash
  - 32 KB RAM
  - 2KB EEPROM



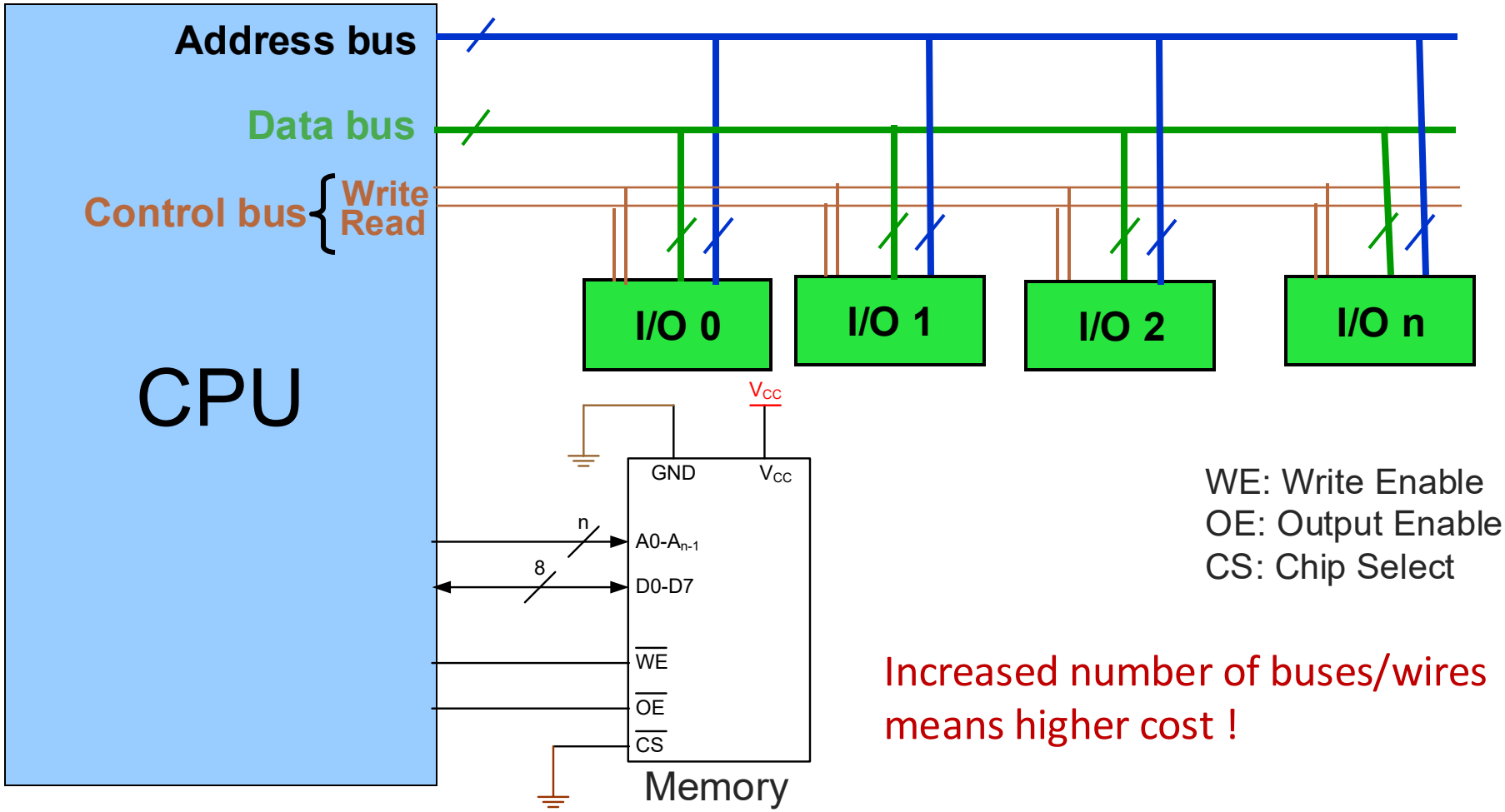


Connecting I/O devices to CPU

- CPU should have lots of pins!



Connecting I/Os and Memory to CPU (Isolated I/0)

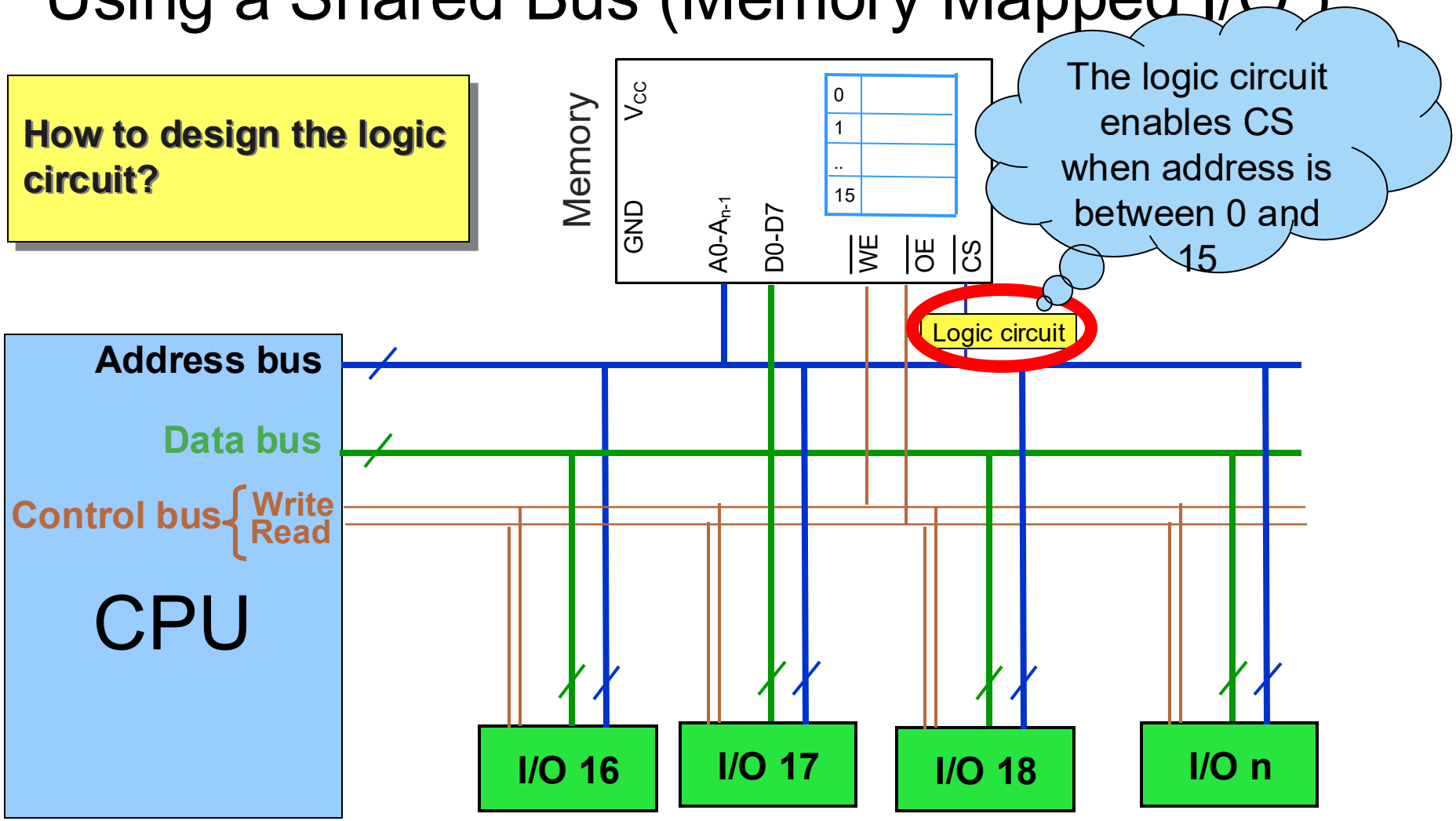


WE: Write Enable
OE: Output Enable
CS: Chip Select

Increased number of buses/wires means higher cost !

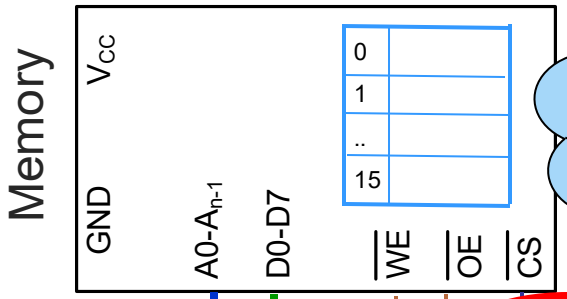
Connecting I/Os and Memory to CPU Using a Shared Bus (Memory Mapped I/O)

How to design the logic circuit?



Connecting I/Os and Memory to CPU Using a Shared Bus (Memory Mapped I/O)

How to design the logic circuit?



The logic circuit enables CS when address is between 0 and 15

Add

Control

CI

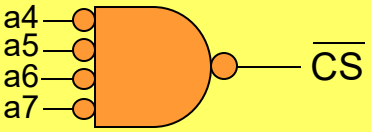
Solution

1. Write the address range in binary
2. Separate the fixed part of address
3. Using a NAND, design a logic circuit whose output activates when the fixed address is given to it

a7 a6 a5 a4 a3 a2 a1 a0

From address 0 ➔ 0 0 0 0 0 0 0 0

To address15 ➔ 0 0 0 0 1 1 1 1



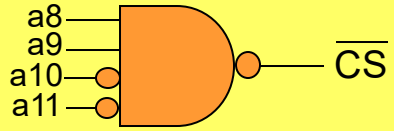
Address Decoding: Example 1

- Design an address decoder for address of 300H to 3FFH (for a 12-bit address bus)

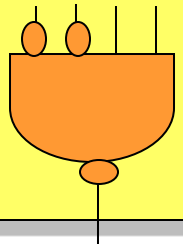
Solution

1. Write the address range in binary
2. Separate the fixed part of address
3. Design the logic circuit

	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
From address 0x300 ➔	0	0	1	1	0	0	0	0	0	0	0	0
To address 0x3FF ➔	0	0	1	1	1	1	1	1	1	1	1	1

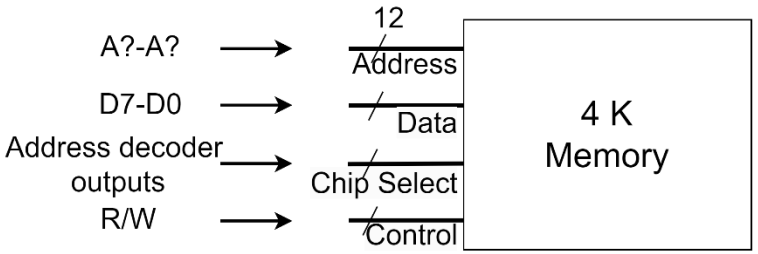


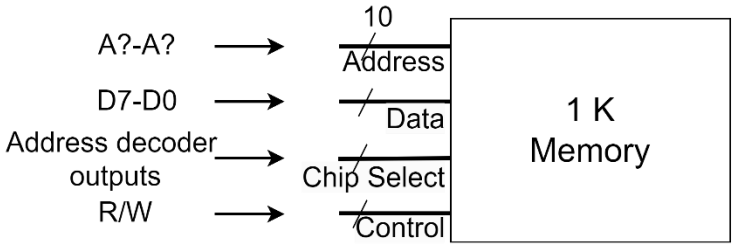
An easy way of designing

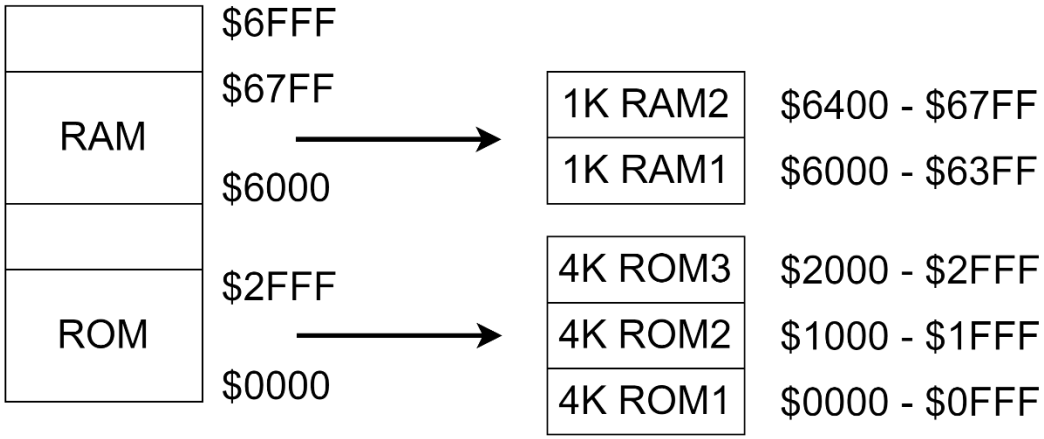


Address Decoding: Example 2

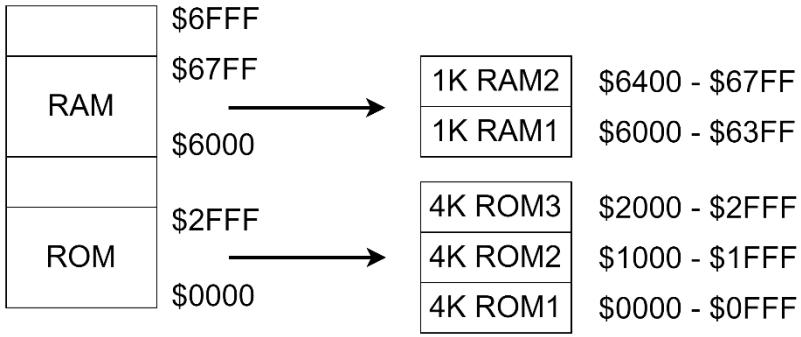
- Use three 4K ROM chips
  - Need 12 bits for addressing
  - A12-A15 are left for ROM selection
- Use two 1K RAM chips
  - Need 10 bits for internal addressing
  - A10-A15 are left for RAM selection
- Memory Map







Address Decoding: Example 2



1 of 3 ROM selection

Internal address decoding for ROMs

A15	A14	A13	A12	A11	A10	A9	.....	A1	A0		
0	0	0	0	x	x	x		x	x	ROM1	0000-0FFF 4K
0	0	0	1	x	x	x		x	x	ROM2	1000-1FFF 4K
0	0	1	0	x	x	x		x	x	ROM3	2000-2FFF 4K
0	1	1	0	0	0	x		x	x	RAM1	6000-63FF 1K
0	1	1	0	0	1	x		x	x	RAM2	6004-67FF 1K

1 of 2 RAM selection

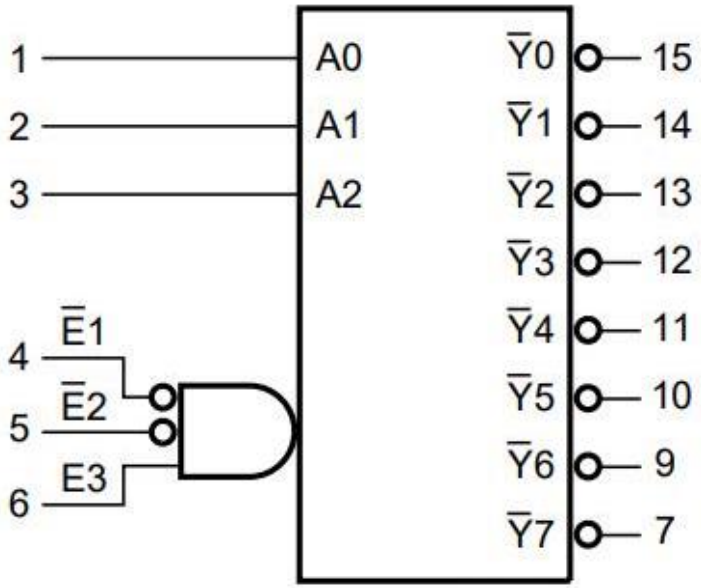
Internal address decoding for RAMs

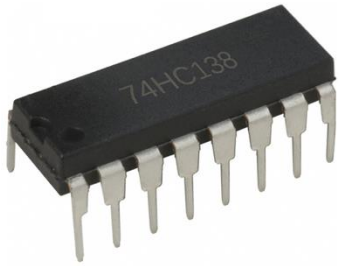
RAM/ROM selection

May be used for active enable input if a PROM decoder is used

74HC138 Decoder

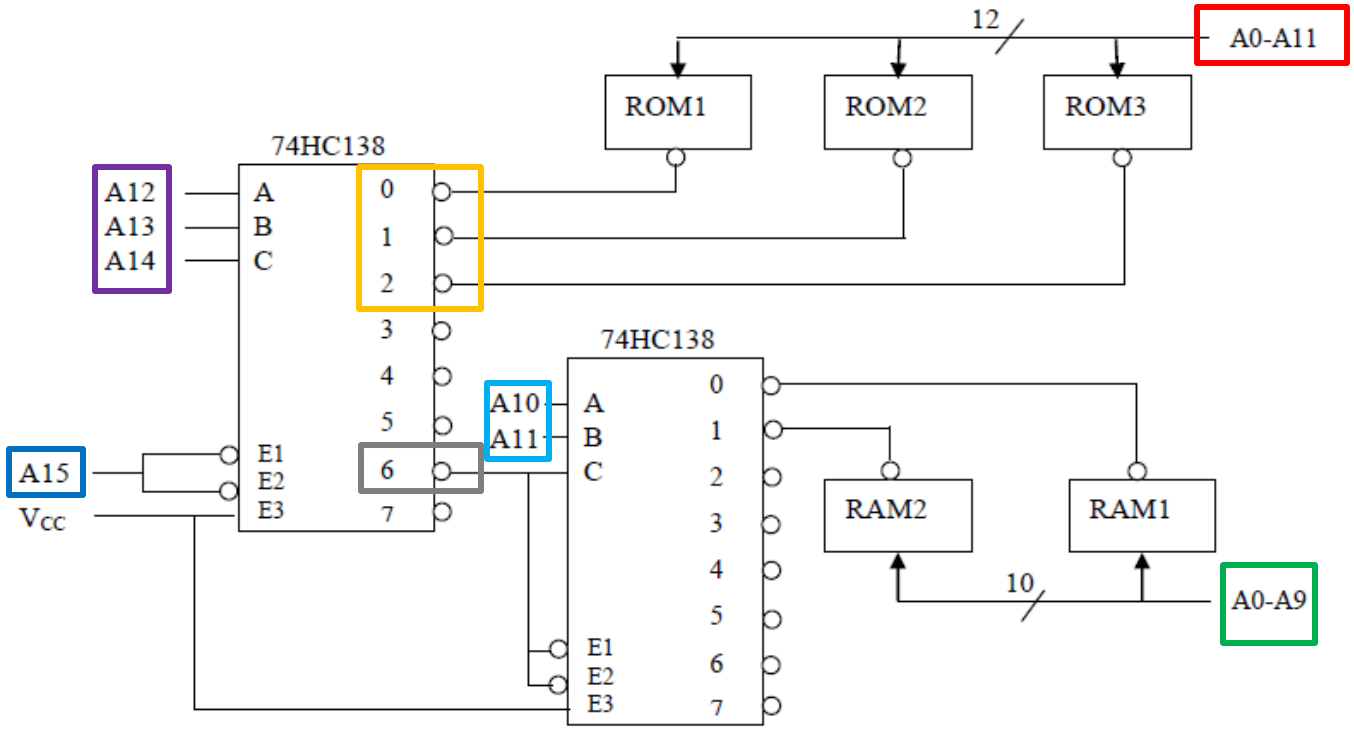
- Description
  - 3 x 8 decoder
  - 3 enable inputs: E1 and E2 active low, E3 active high
  - Address inputs A0, A1, A2
  - Active low outputs Y0 to Y7



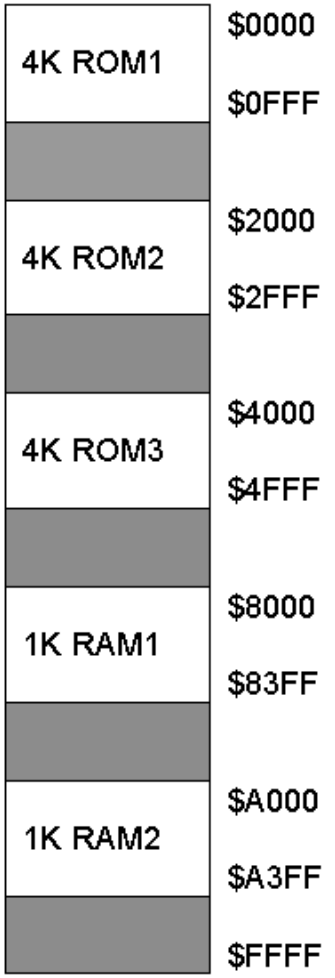


Address Decoding: Example 2 with 74HC138

A15	A14	A13	A12	A11	A10	A9	A1	A0		
0	0	0	0	x	x	x	x	x	ROM1	0000-0FFF 4K
0	0	0	1	x	x	x	x	x	ROM2	1000-1FFF 4K
0	0	1	0	x	x	x	x	x	ROM3	2000-2FFF 4K
0	1	1	0	0	0	x	x	x	RAM1	6000-63FF 1K
0	1	1	0	0	1	x	x	x	RAM2	6004-67FF 1K

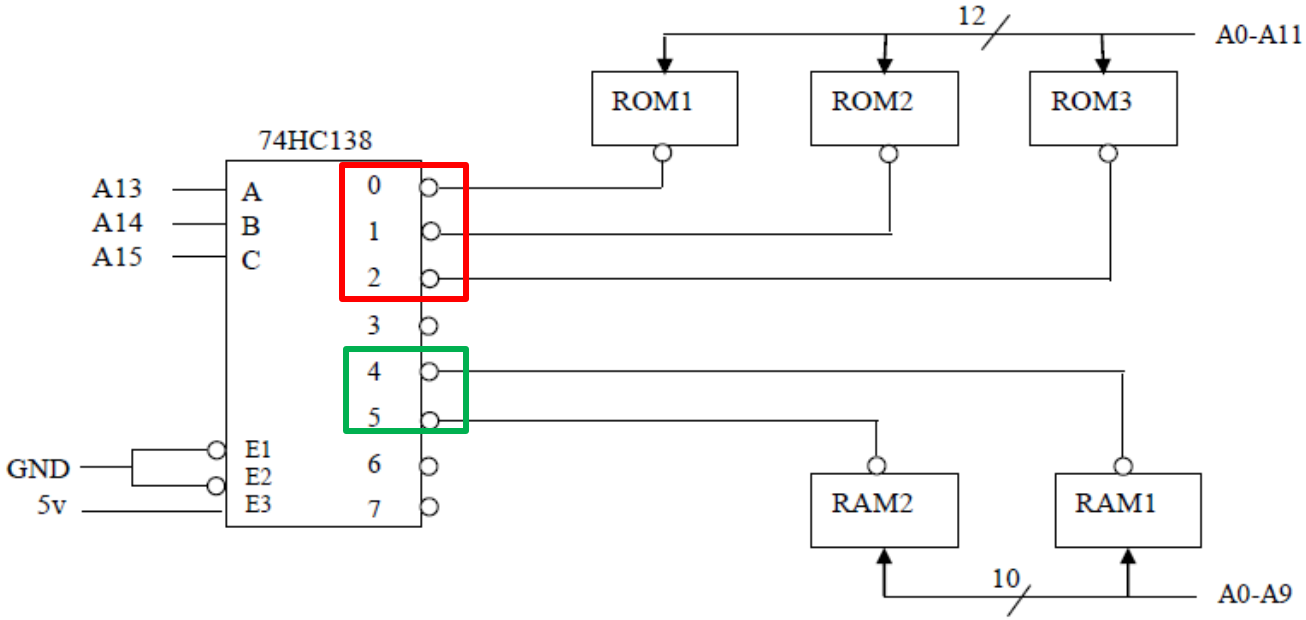


Memory Overlaying



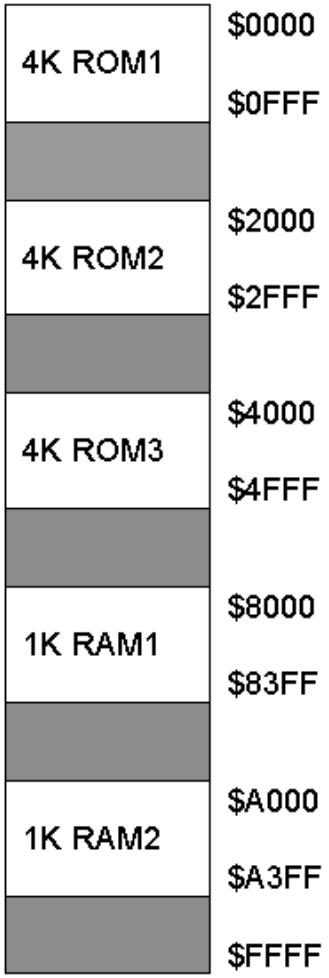
A15	A14	A13	A12	A11	A10	A9	.....	A1	A0	
0	0	0	0	x	x	x		x	x	ROM1
0	0	1	0	x	x	x		x	x	ROM2
0	1	0	0	x	x	x		x	x	ROM3
1	0	0	0	0	0	x		x	x	RAM1
1	0	1	0	0	0	x		x	x	RAM2

A12 is always 0 and hence not used in decoding circuit



Question: How much memory is virtually addressed per chip?

Memory Overlaying



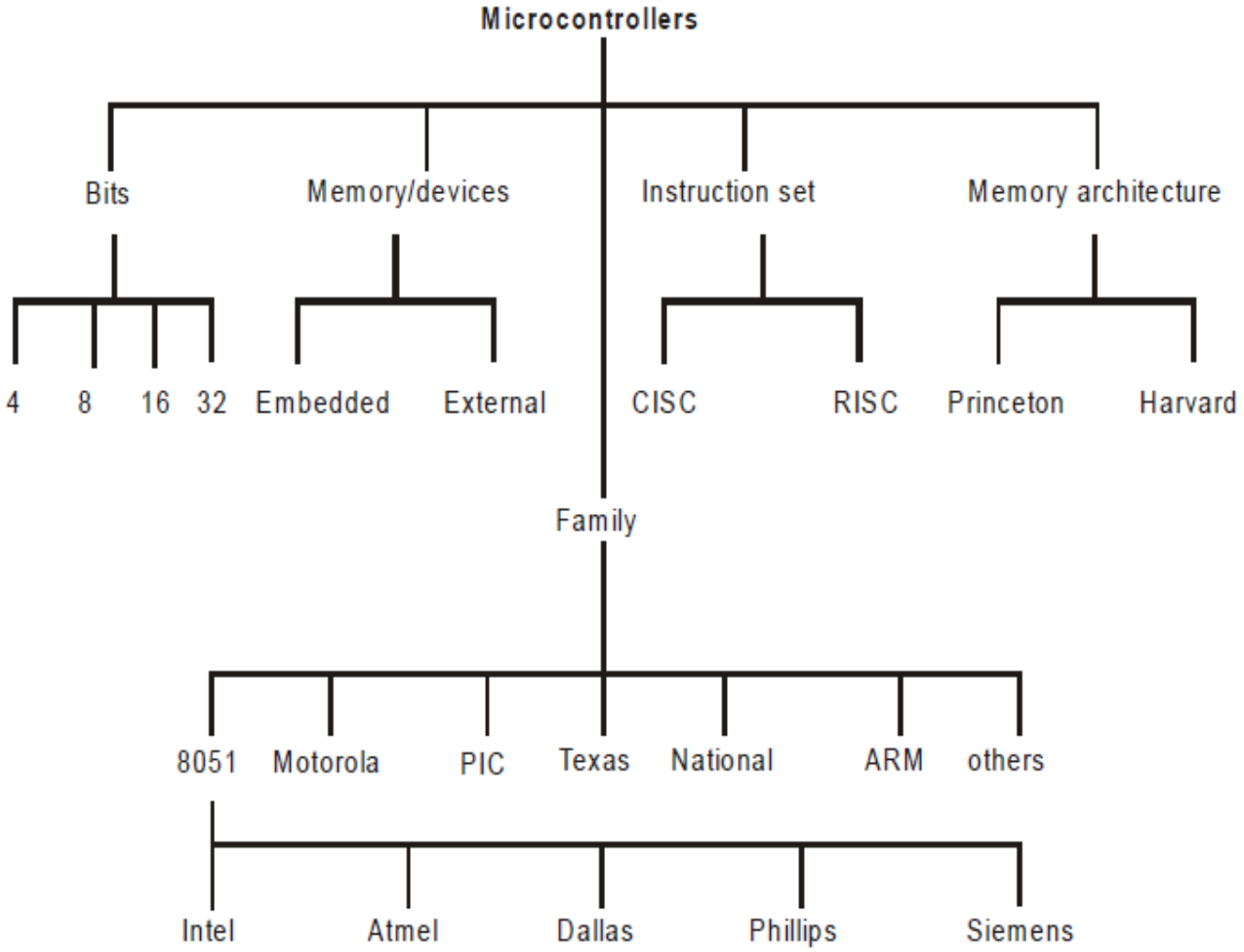
A15	A14	A13	A12	A11	A10	A9	.....	A1	A0		Virtual Memory
0	0	0	0	x	x	x		x	x	ROM1	8K
0	0	1	0	x	x	x		x	x	ROM2	8K
0	1	0	0	x	x	x		x	x	ROM3	8K
1	0	0	0	0	0	x		x	x	RAM1	8K
1	0	1	0	0	0	x		x	x	RAM2	8K

Overlayed Memory

ROM 1 (4K physical, 8K virtual): $0000 same as $1000

RAM 1 (1K physical, 8K virtual): $8000, $8400, $8800, $8C00, $9000, $9400, $9800, $9C00 are same in terms of the memory chip

Microcontrollers: Classification



Microcontrollers: Usage and Selection

- Up to now
  - Cover all hardware components of microcontrollers
  - Learn about address space and address decoding
- Observation
  - There are many different microcontroller types
    → Question: Which one to use?
  - We want to actually use microcontrollers
    → Question: How does it really work?

Instruction Set Architecture (ISA): Background

- ISA Description
  - ISA defines the interface between a user and a microprocessor

    → "Contract" between programmer/compiler + HW
  - ISA is an abstract interface between the hardware and the lowest level software of a machine

    → Low-level details of microprocessor are "invisible" to user
- ISA
  - Defines operations that the processor can execute
  - Defines data transfer mechanisms + how to access data
  - Defines control mechanisms (branch, jump, etc)
  - Enables microprocessors of varying cost/performance to run same software
  - Usually defines a "family" of microprocessors

    → Examples: Intel x86 (IA32), Sun Sparc, DEC Alpha, IBM/360, IBM PowerPC, M68K, DEC VAX

ISA: Illustration

- Motorola 6800 / Intel 8085 (1970s)
  - 1-address architecture: ADDA <addr>
  - (A) = (A) + (addr)
- Intel x86 (1980s)
  - 2-address architecture: ADD EAX, EBX
  - (A) = (A) + (B)
- MIPS (1990s)
  - 3-address architecture: ADD $2, $3, $4
  - ($2) = ($3) + ($4)
- ARM (1990s)
  - 3-address architecture: ADD R2, R3, R4
  - (R2) = (R3) + (R4)

ISA: What leads to a good/bad ISA?

- Ease of Implementation (Job of Architect/Designer)
  - Does the ISA lead itself to efficient implementations?
- Ease of Programming (Job of Programmer/Compiler)
  - Can the compiler use the ISA effectively?
- Future Compatibility
  - ISAs may last 30+yrs
  - Special features, address range, etc. need to be thought out

ISA: RISC versus CISC

- CISC (Complex Instruction Set Computer)
  - Put as many instruction as you can into the CPU
  - Have multi-word instructions
  - Allow operands directly from memory
- RISC (Reduced Instruction Set Computer)
  - Reduce the number of instructions, and use your facilities in a more proper way
  - Have one-word instructions
  - Require arithmetic operands to be in registers

ISA: RISC Instruction Sets

- Consider high-level language statement

  C = A + B

- A, B, and C correspond to memory locations
- RTL specification with these symbolic names

  C ← [A] + [B]

- Steps
  - Fetch contents of locations A and B,
  - Compute sum, and
  - Transfer result to location C
- Sequence of simple RISC instructions for the task

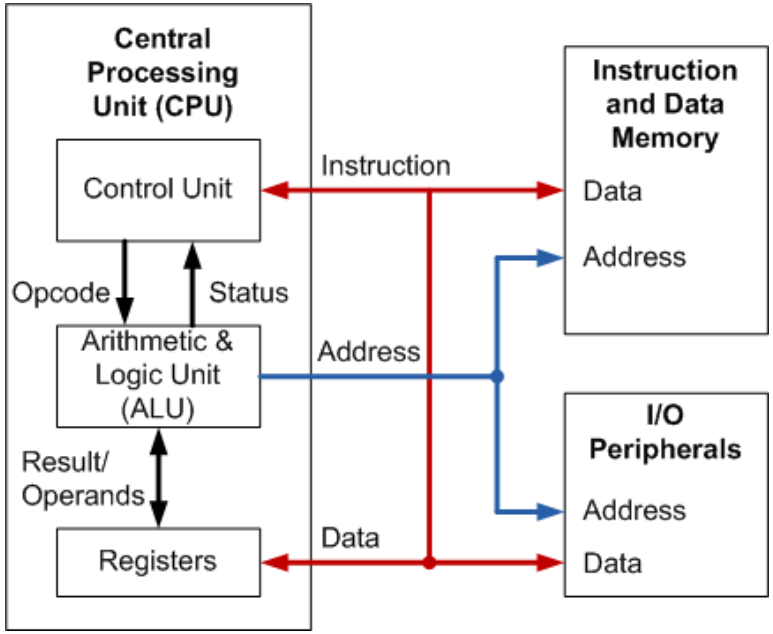
```
Load    R2, A
Load    R3, B
Add     R2, R2, R3
Store   R2, C
```

In CISC it can be just Add A, B

Computer Architectures: Comparison

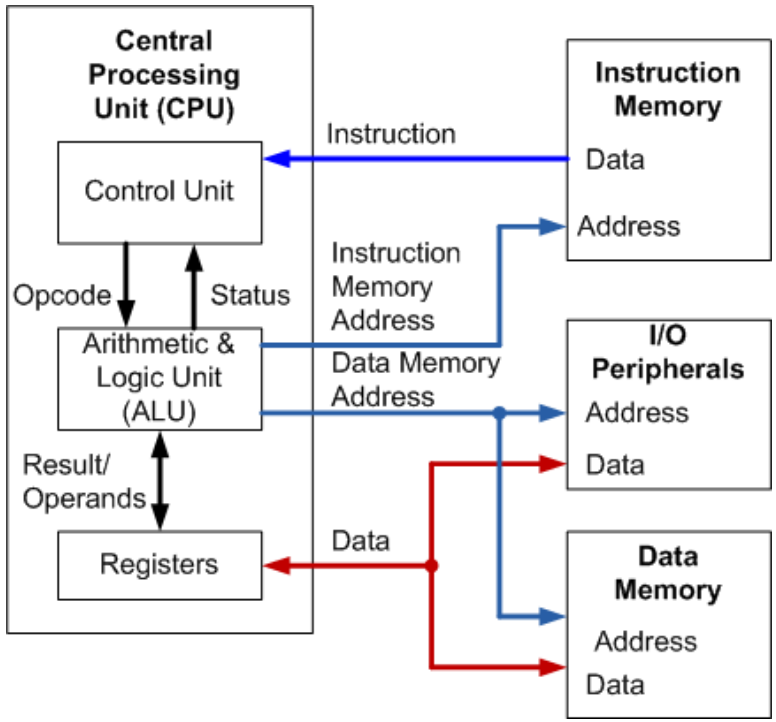
Von-Neumann/Princeton

Instructions and data are stored in the same memory.



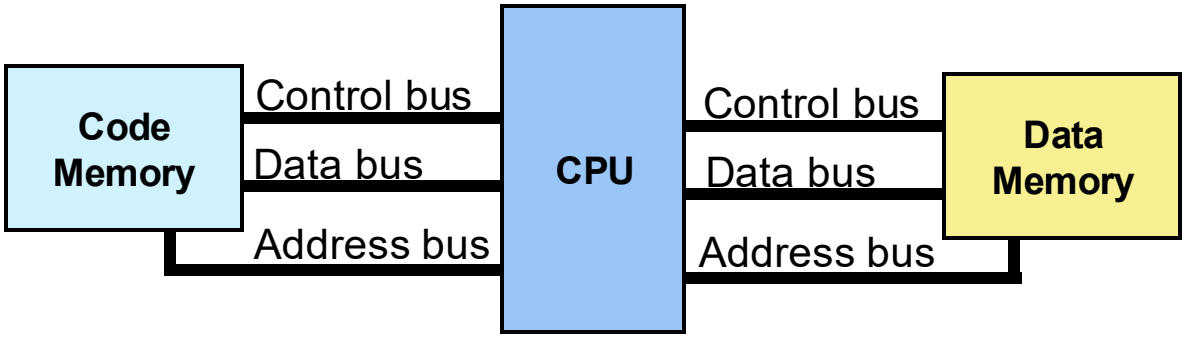
Harvard

Data and instructions are stored into separate memories.



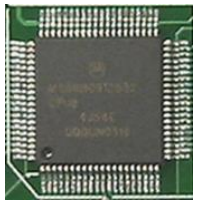
Computer Architecture: Cortex M4

- Cortex M4 - Harvard architecture
  - Separated data bus and instruction bus
  - Advantage: opcodes and operands can go in and out of the CPU together.
  - Disadvantage: leads to more cost in general purpose computers



Most Common Microcontrollers

- 8-bit microcontrollers
  - Motorola
  - AVR
  - PIC
  - HCS12
  - 8051
- 32-bit microcontrollers
  - ARM
  - PIC32

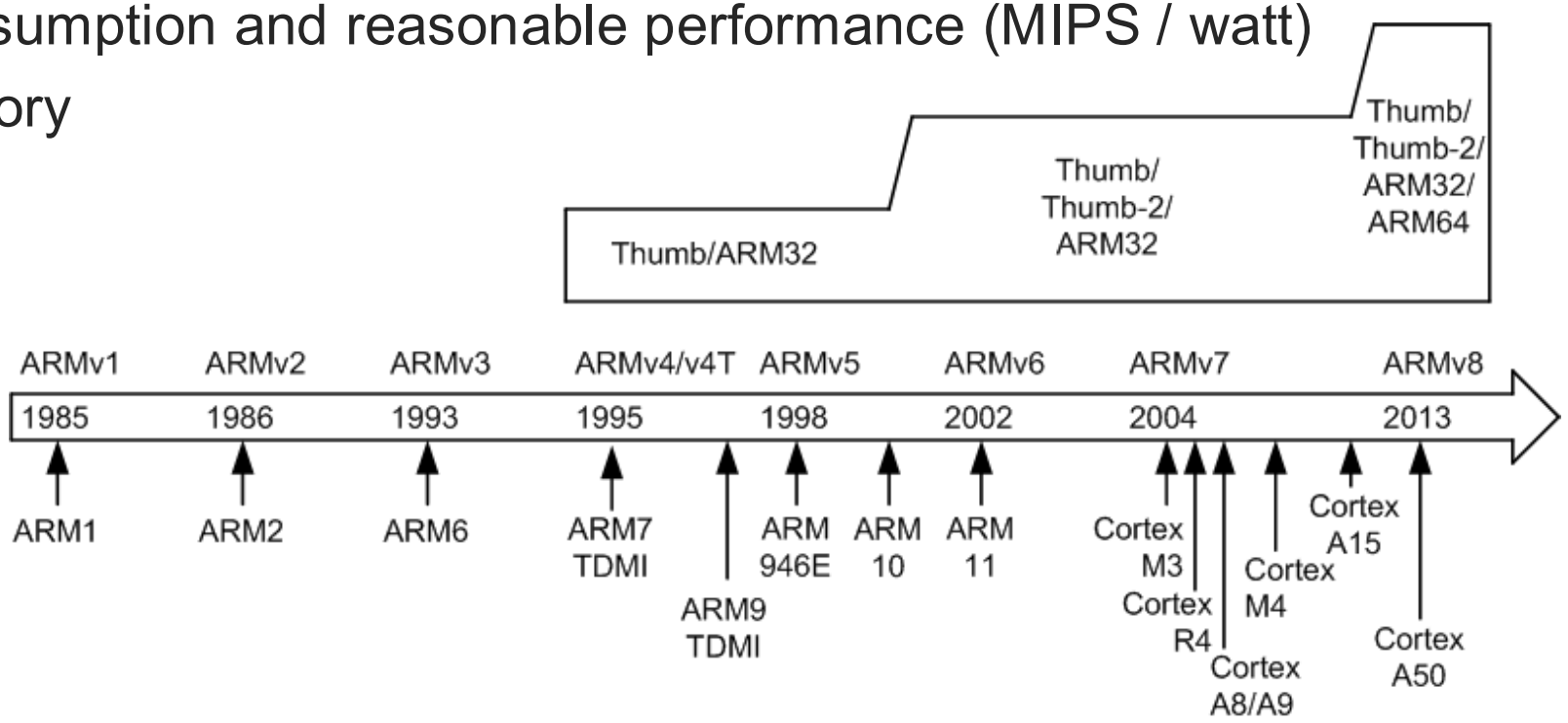


Microcontrollers: How to Choose

- Peripherals and interface features
  - Number of I/O ports
  - Serial communication modules
  - Peripherals like timer, ADC, PWM, etc
- Memory size requirements of the application
- Processing speed requirements
- Low power requirements
- Chip package
- Operation conditions (voltage, temperature, electromagnetic interference)
- Cost and availability
- Software development tool support and development kits
- Future upgradability
- Firmware packages and firmware security
- Availability of application notes, design examples, and support

Microcontrollers: ARM Technology

- ARM is one of the most licensed and thus widespread processor cores in the world
- Used especially in portable devices due to low power consumption and reasonable performance (MIPS / watt)
- History



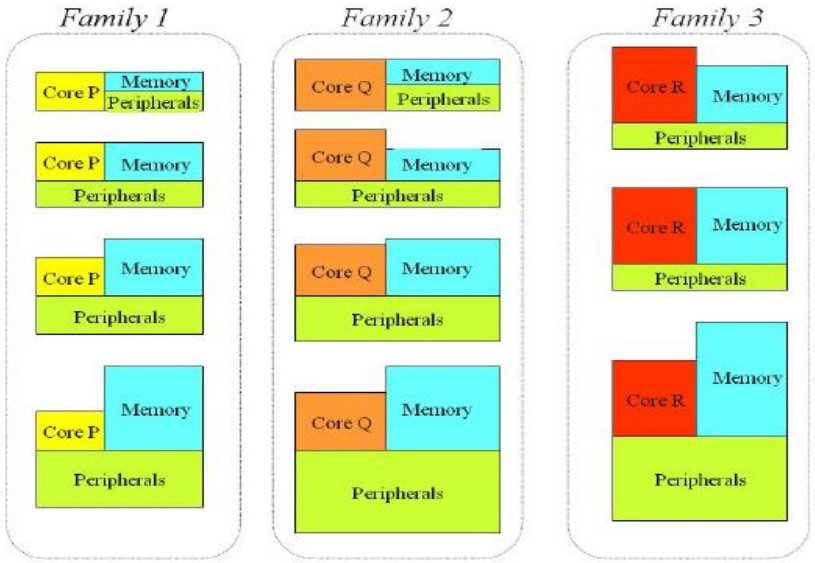
ARM: History

Year	Billion units	Relative size
2015	15	
2014	12	
2013	10	
2012	8.7	
2011	7.9	
2010	6.1	
2009	3.9	
2008	4.0	
2007	2.9	
2006	2.4	
2005	1.662	
2004	1.272	
2003	0.782	
2002	0.456	
2001	0.420	
2000	0.367	
1999	0.175	
1998	0.051	
1997	0.009	
Total	**78.094**	

- Acorn Computers
  - British computer company based in Cambridge.
  - Set-up in 1978 by Chris Curry and Hermann Hauser.
  - Most famous computer was the BBC Micro (1980)
  - First RISC machine – 1985
  - First RICS based home computer – Archimedes - 1987
- ARM (Advanced RISC Machine)
  - Founded in November 1990 by Acorn Comp, Apple, VLSI Tech.
  - Licencing – 1992
  - ARM holdings float on to Nasdaq and London Stock Exch. – 1998
  - 1 billion ARM based chips have been shipped – 2002
  - Energy efficient Cortex chips have been announced – 2004
  - 10 billion chips shipped (**95%** of smartphones, **35%** of digital televisions and set-top boxes and **10%** of mobile computers) – 2009
  - 64-bit capable ARMv8 architecture is introduced – 2011
  - ARM named as fifth most innovative company in the world – 2014
  - ARM reported profit of $448.4 million – 2015
  - SoftBank acquired ARM for $31 billion - 2016

ARM: Technical Terms for Different Designs

- Architecture
  - Specifications, i.e., set of registers, instructions and operation modes that should be supported by implementations of the architecture
- Family
  - Specific detailed implementation of an architecture, i.e. the actual hardware details needed to create an ARM core



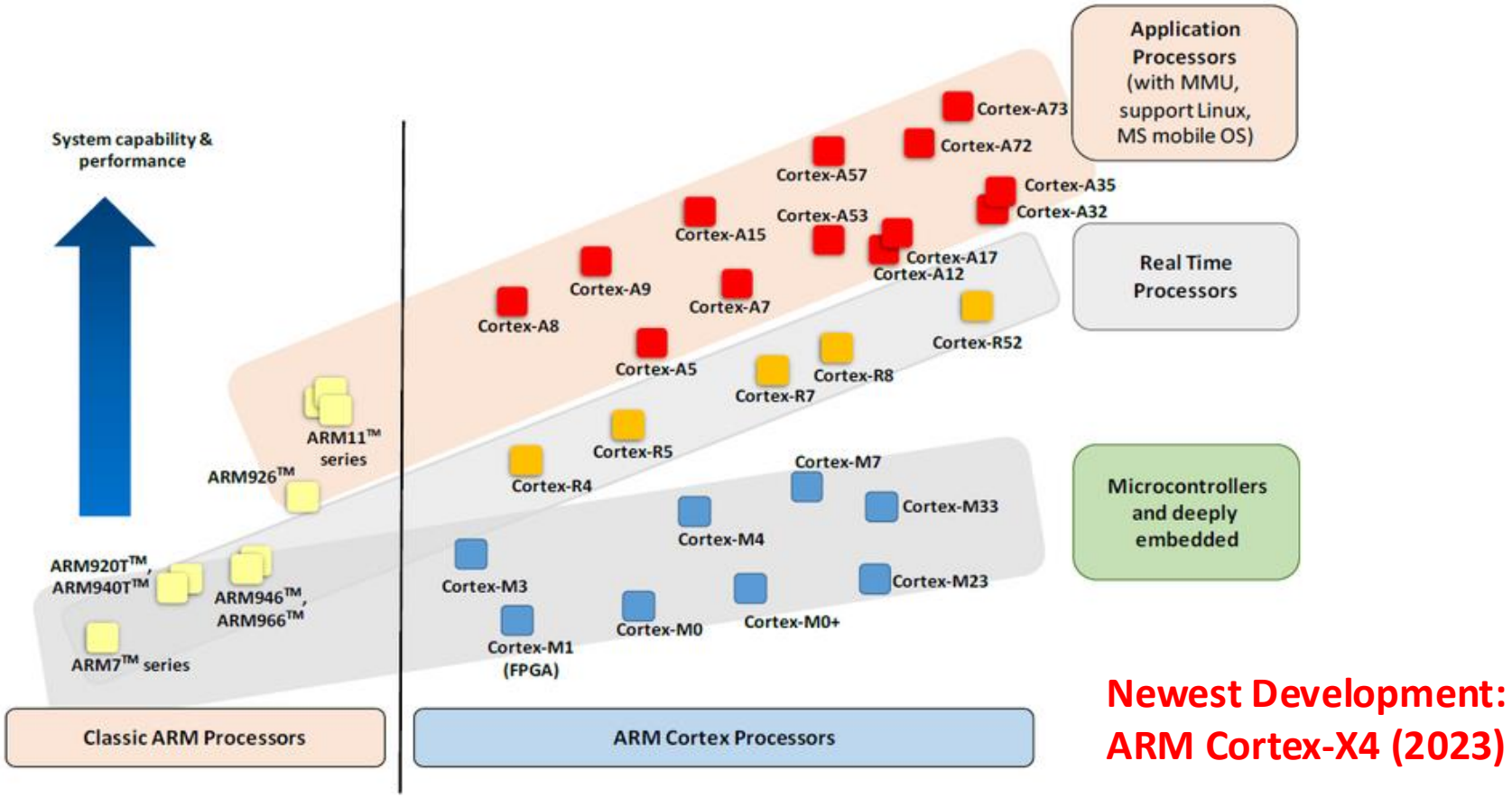
- Core
  - Specific implementation of an architecture, i.e. the actual blue-print of the transistors and other discrete parts needed to create a ARM CPU

ARM: Examples

Family	Architecture	Cores
ARM7TDMI	ARMv4T	ARM7TDMI(S)
ARM9 ARM9E	ARMv5TE(J)	ARM926EJ-S, ARM966E-S
ARM11	ARMv6 (T2)	ARM1136(F), 1156T2(F)-S, 1176JZ(F), ARM11 MPCore™
Cortex-A	ARMv7-A	Cortex-A5, A7, A8, A9, A15
Cortex-R	ARMv7-R	Cortex-R4(F)
Cortex-M	ARMv7-M	Cortex-M3, M4
	ARMv6-M	Cortex-M1, M0
NEW !	ARMv8-A	64 Bit

Features	ARM7TDMI-S	Cortex-M3
Architecture	ARMv4T (von Neumann)	ARMv7-M (Harvard)
ISA Support	Thumb / ARM	Thumb / Thumb-2
Pipeline	3-Stage	3-Stage + branch speculation
Interrupts	FIQ / IRQ	NMI + 1 to 240 Physical Interrupts
Interrupt Latency	24-42 Cycles	12 Cycles
Sleep Modes	None	Integrated
Memory Protection	None	8 region Memory Protection Unit
Dhrystone	0.95 DMIPS/MHz (ARM mode)	1.25 DMIPS/MHz
Power Consumption	0.28mW/MHz	0.19mW/MHz
Area	0.62mm2 (Core Only)	0.86mm2 (Core & Peripherals)*

ARM: Processors Family Evolution



Newest Development: ARM Cortex-X4 (2023)

ARM: Usage

- ARM 7 – Introduced in 1994, used for simple 32-bit devices.
- ARM9 – Most popular ARM family (> 5 billion sold), used for smartphones, HDD controllers, etc.
- ARM11 – Extreme low power, many of today's smartphones
- Cortex M-series: Cost-sensitive solutions for microcontroller applications, system clock < 200MHz
  - Cortex M0 – Ultra low-power, ultra low gate count
  - Cortex M1 – First ARM processor designed specifically for implementation in FPGAs
  - Cortex M3 – High performance and energy efficiency. Microcontroller applications
  - Cortex M4 – Embedded processor for DSP
  - Cortex M7 – 2x Performance of M4
- Cortex R-series: Exceptional performance for real-time applications, system clock < 600MHz
  - Cortex R4 - First embedded real-time processor based on the ARMv7-R architecture for high-volume deeply-embedded System-on-Chip applications such as hard disk, wireless baseband processors, electronic control units for automotive systems
  - Cortex R5 – Extends the feature set of Cortex-R4, increased efficiency and reliability
  - Cortex R7 – High-performance dual core
- Cortex A-series: High performance processors for open operating systems, system clock > 1GHz
  - Cortex A5 – Power and cost sensitive applications, smartphones
  - Cortex A8 – Suitable for high-end phones, printers, DTVs
  - Cortex A9 – 1 to 4 cores. High performance-low power devices
  - Cortex A15 – Ultra low-power. Suitable for mobile computing, wireless infrastructure

ARM: Diverse Embedded Processing Needs

Cortex - A

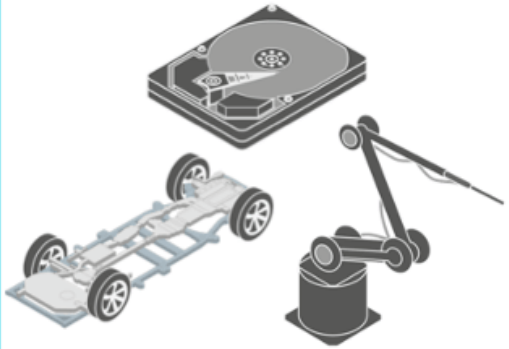
Highest performance

Optimised for
rich operating systems

Cortex - R

Fast response

Optimised for
high performance,
hard real-time applications

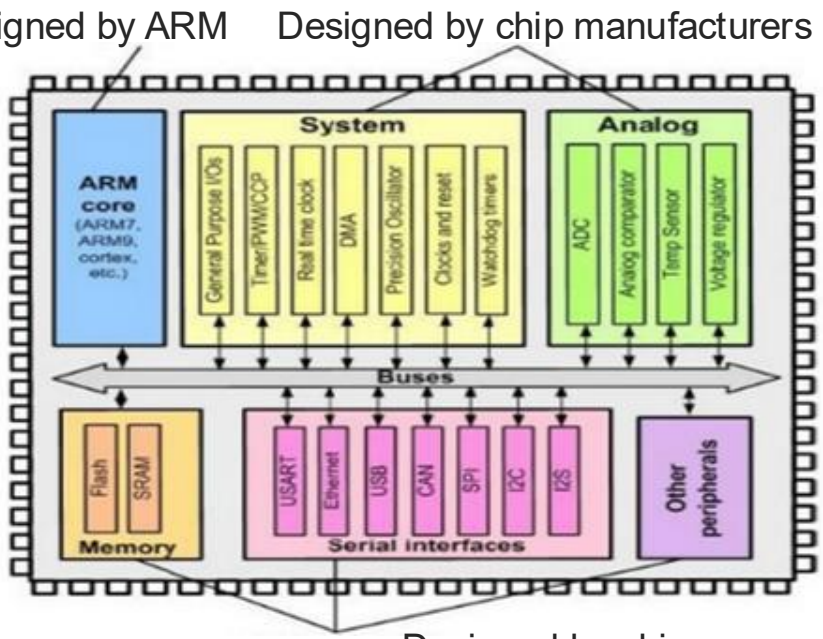


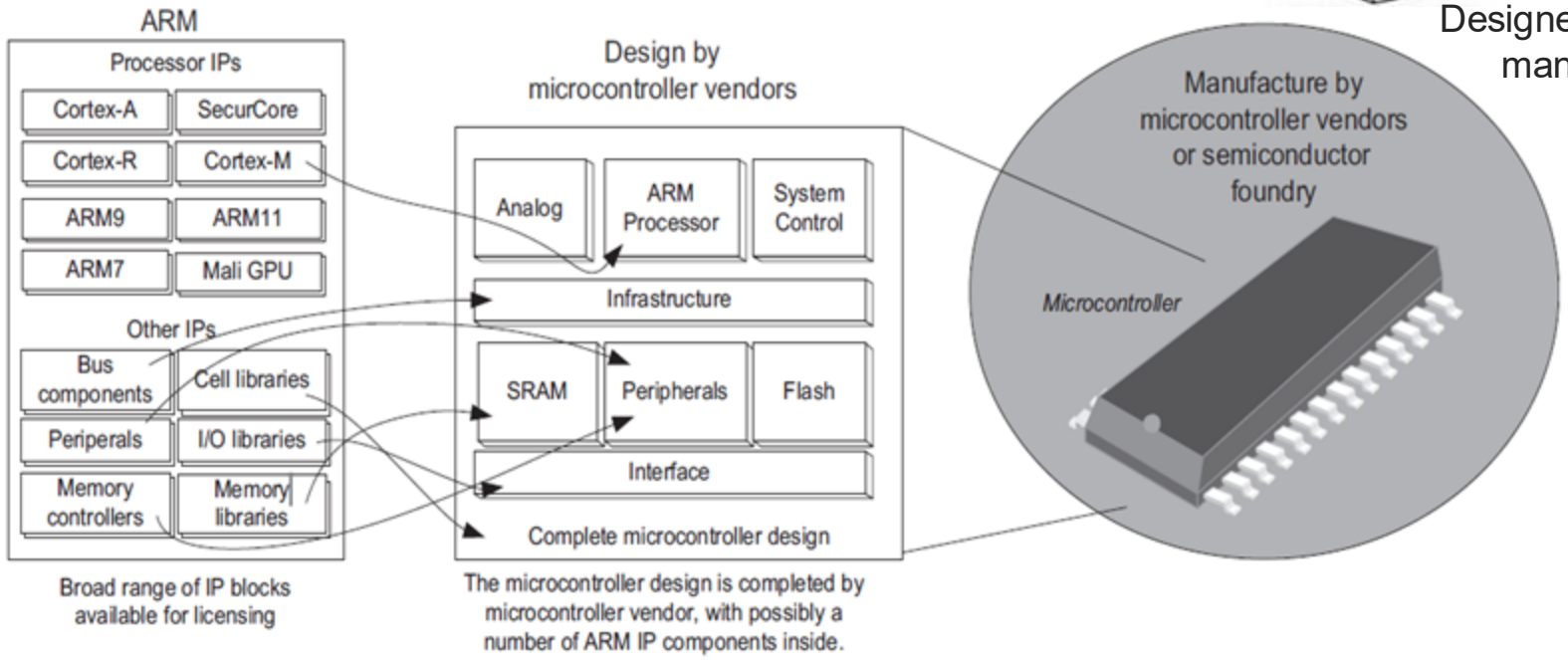
Cortex - M

Smallest/lowest power

Optimised for
discrete processing and
microcontrollers

Microcontrollers with ARM Core





ARM: Sample Products

iPhone 5

http://www.ifixit.com

The A6 processor is the first Apple System-on-Chip (SoC) to use a custom design, based off the **ARMv7** instruction set

iPhone 6

http://www.ifixit.com

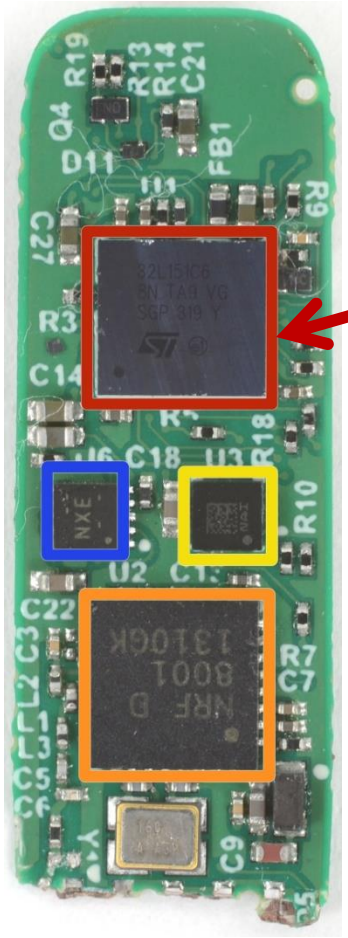
The A8 processor is the first 64-bit ARM based SoC. It supports **ARM A64, A32, and T32** instruction set.

Apple Watch

Apple S1 Processor

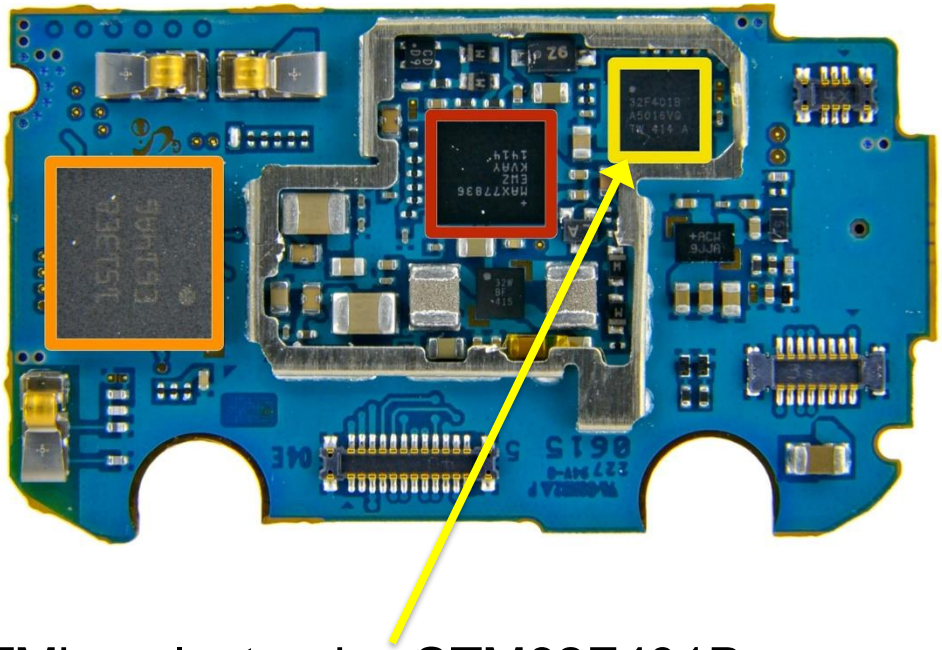
- **32-bit ARMv7-A** compatible
- # of Cores: **1**
- CMOS Technology: 28 nm
- L1 cache 32 KB data
- L2 cache 256 KB
- GPU PowerVR SGX543

Fitbit Flex Teardown



STMicroelectronics 32L151C6 Ultra Low Power ARM Cortex M3 Microcontroller

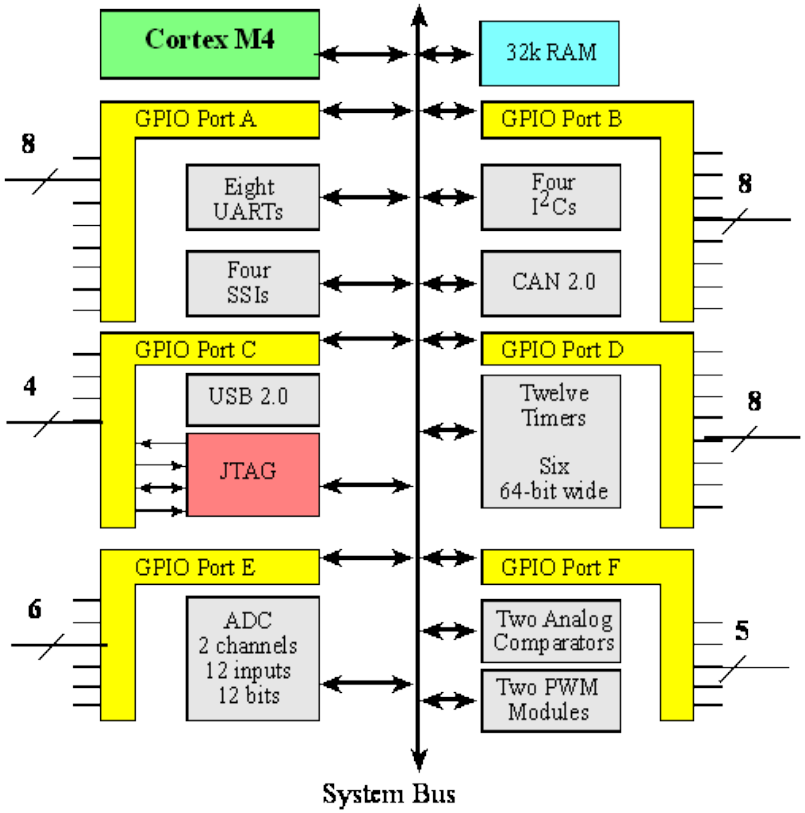
Samsung Galaxy Gear

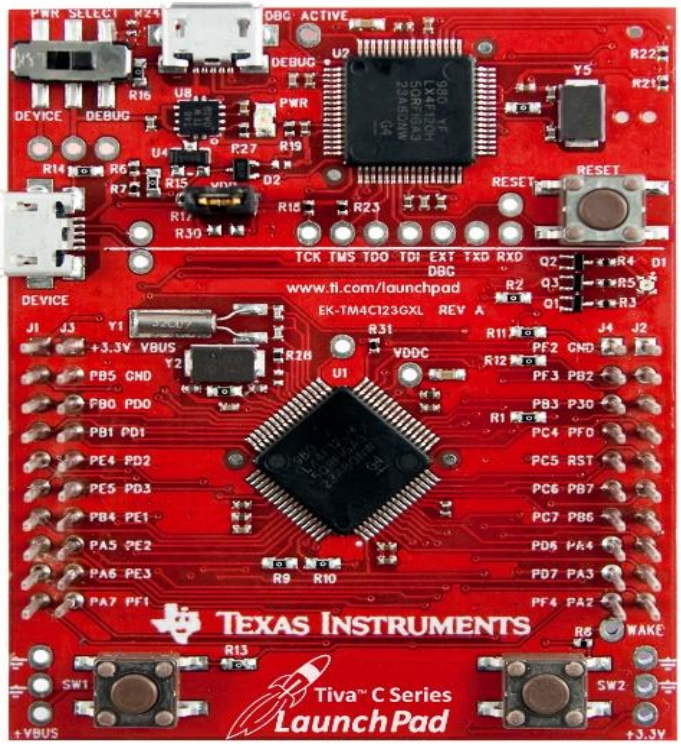


STMicroelectronics STM32F401B **ARM-Cortex M4** MCU with 128KB Flash

EE447: Our Board

- Texas Instruments Tiva C Series TM4C123G Board including ARM Cortex-M4





16MHz internal clock

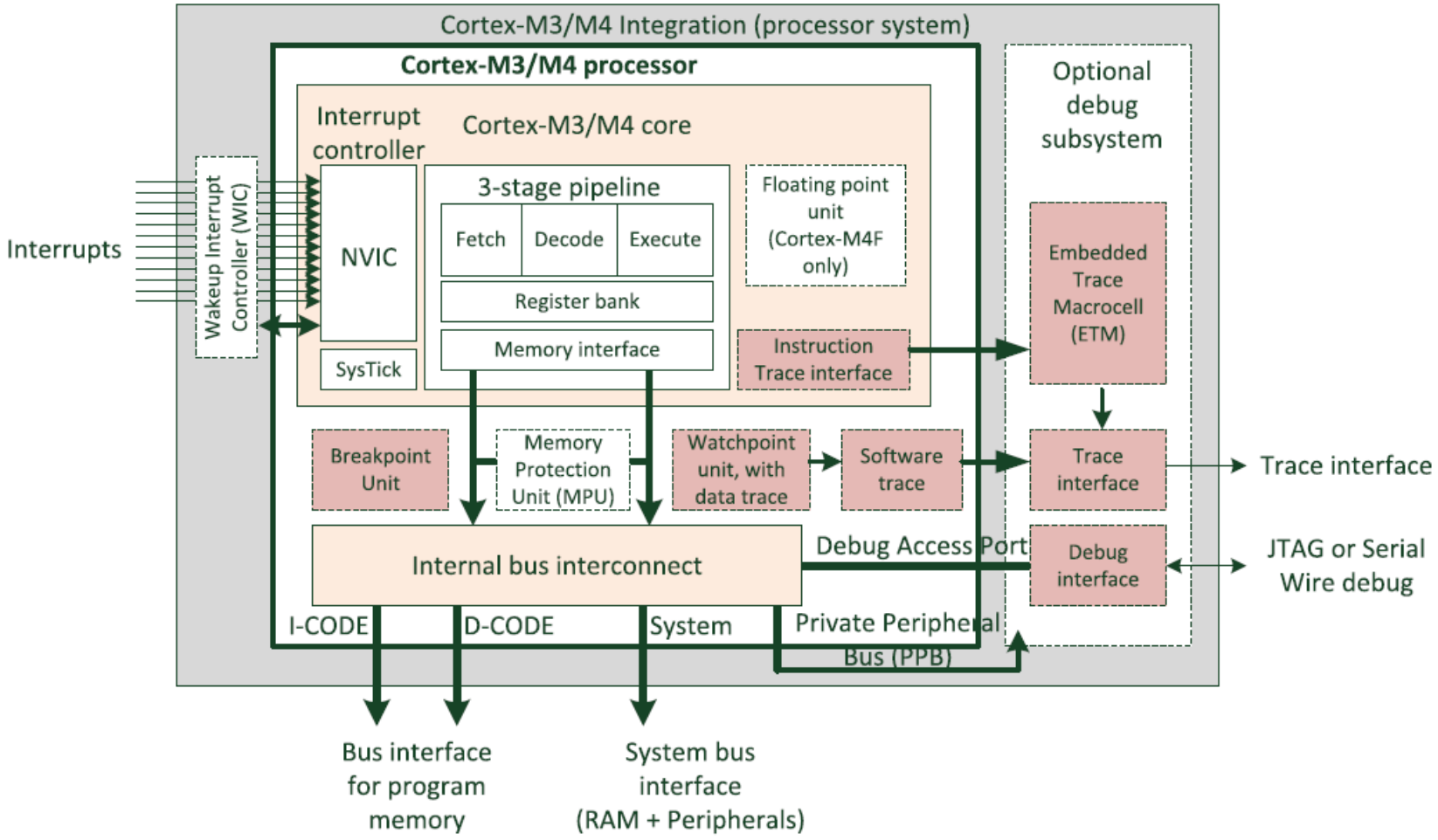
EE447: Our Board – Features

Feature	Description
Performance	
Core	ARM Cortex-M4F processor core
Performance	80-MHz operation; 100 DMIPS performance
Flash	256 KB single-cycle Flash memory
System SRAM	32 KB single-cycle SRAM
EEPROM	2KB of EEPROM
Internal ROM	Internal ROM loaded with TivaWare™ for C Series software
Security	
Communication Interfaces	
Universal Asynchronous Receivers/Transmitter (UART)	Eight UARTs
Synchronous Serial Interface (SSI)	Four SSI modules
Inter-Integrated Circuit (I^2C)	Four I^2C modules with four transmission speeds including high-speed mode
Controller Area Network (CAN)	Two CAN 2.0 A/B controllers
Universal Serial Bus (USB)	USB 2.0 OTG/Host/Device
System Integration	
Micro Direct Memory Access (µDMA)	ARM® PrimeCell® 32-channel configurable µDMA controller
General-Purpose Timer (GPTM)	Six 16/32-bit GPTM blocks and six 32/64-bit Wide GPTM blocks
Watchdog Timer (WDT)	Two watchdog timers
Hibernation Module (HIB)	Low-power battery-backed Hibernation module
General-Purpose Input/Output (GPIO)	Six physical GPIO blocks
Advanced Motion Control	
Pulse Width Modulator (PWM)	Two PWM modules, each with four PWM generator blocks and a control block, for a total of 16 PWM outputs.
Quadrature Encoder Interface (QEI)	Two QEI modules
Analog Support	
Analog-to-Digital Converter (ADC)	Two 12-bit ADC modules, each with a maximum sample rate of one million samples/second
Analog Comparator Controller	Two independent integrated analog comparators
Digital Comparator	16 digital comparators
JTAG and Serial Wire Debug (SWD)	One JTAG module with integrated ARM SWD
Package Information	
Package	64-pin LQFP
Operating Range (Ambient)	Industrial (-40°C to 85°C) temperature range Extended (-40°C to 105°C) temperature range

ARM Cortex-M4: Background

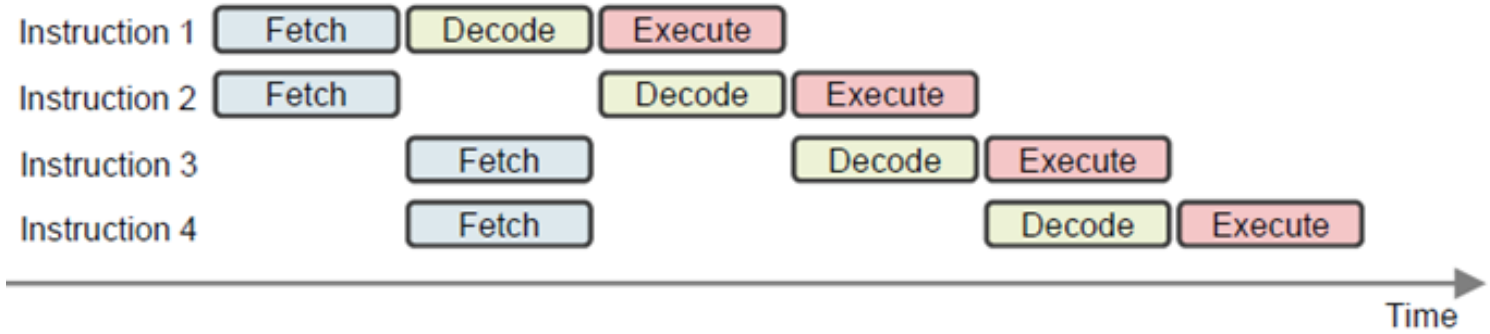
- Introduced in 2010
- Designed with many highly efficient signal processing features
- Features extended single-cycle multiply accumulate instructions, optimized SIMD (single instruction multiple data) arithmetic, saturating arithmetic and an optional Floating Point Unit
- High Performance Efficiency
  - 1.25 DMIPS/MHz (Dhrystone Million Instructions Per Second/MHz) at the order of µWatts/MHz
- Low Power Consumption
  - Longer battery life – especially critical in mobile products
- Enhanced Determinism
  - The critical tasks and interrupt routines can be served quickly in a known number of cycles

ARM Cortex-M4: Overview



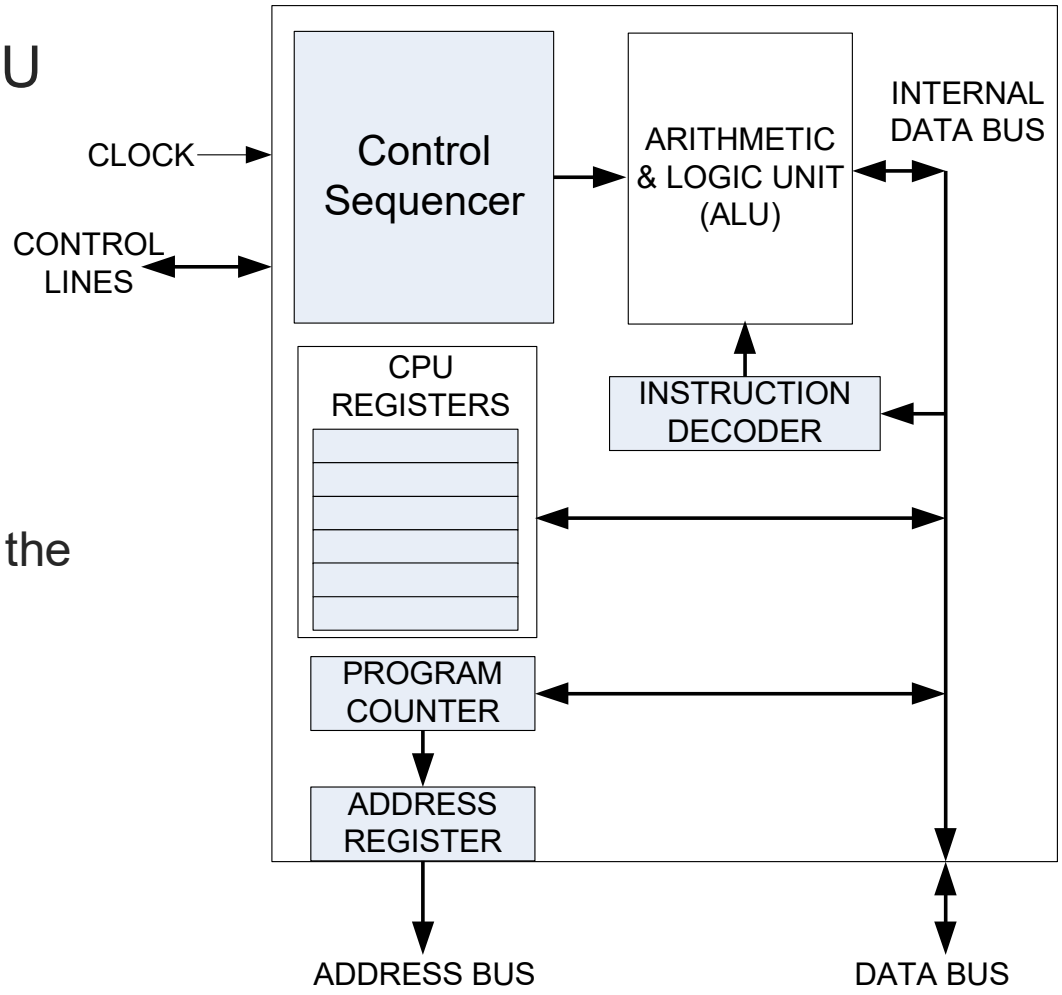
ARM Cortex-M4: Pipeline

- 3-Stage + Branch Speculation Pipeline
  - Three-stage pipeline: **fetch**, **decode**, and **execute**
  - Because of pipeline architecture: instruction fetch and a data access are performed at the same time
  - Since the buses are separated, accesses do not interfere with each other
  - Some instructions may take multiple cycles to execute, in which case the pipeline will be stalled
  - The pipeline will be flushed if a branch instruction is executed
  - Up to two instructions can be fetched in one transfer (16-bit instructions)



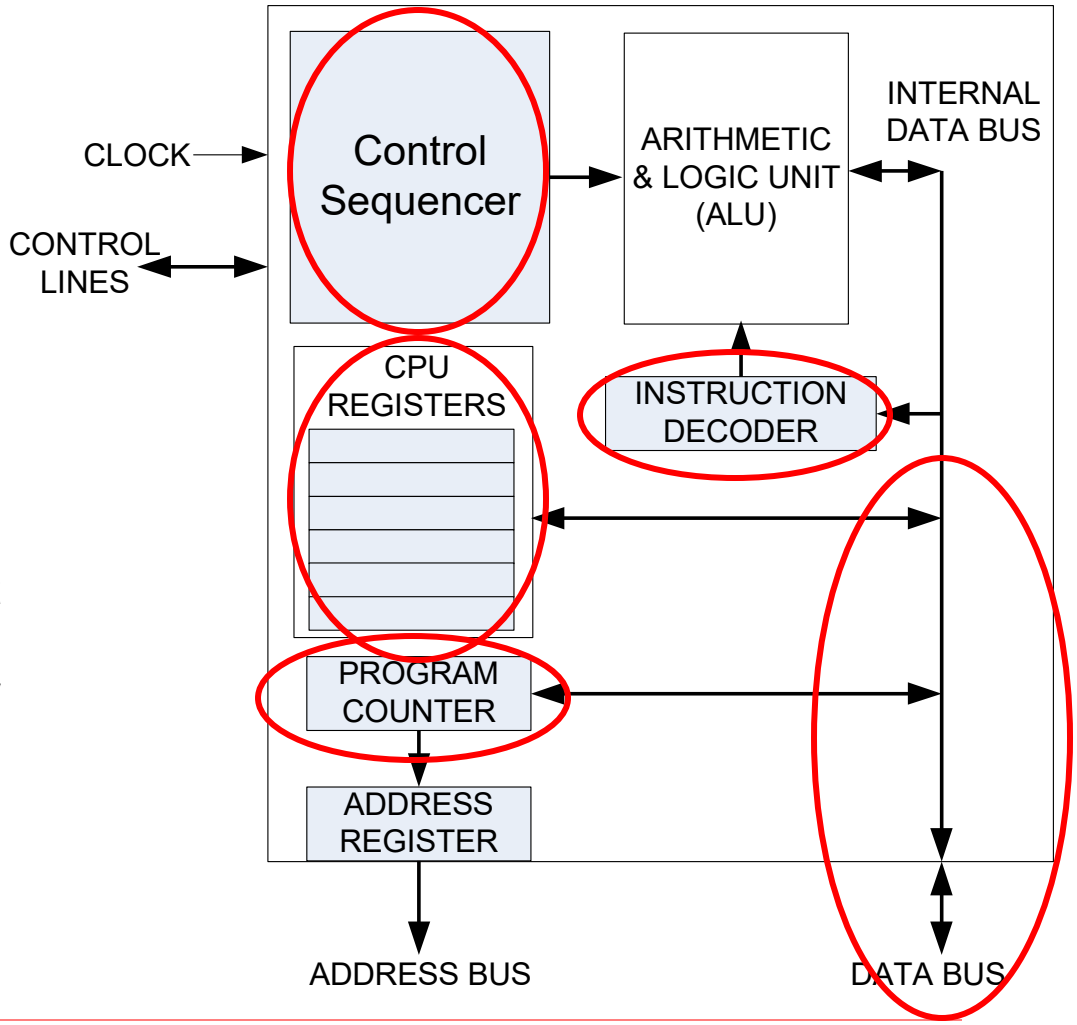
ARM Cortex-M4: Instruction Execution

- Main functions of the CPU
  - Data transfer
  - Arithmetic and logic operations
  - Decision making (instruction flow control)
- Control Sequencer
  - Controls the operations in the CPU
- Register Array
  - Temporary storage (faster than memory)

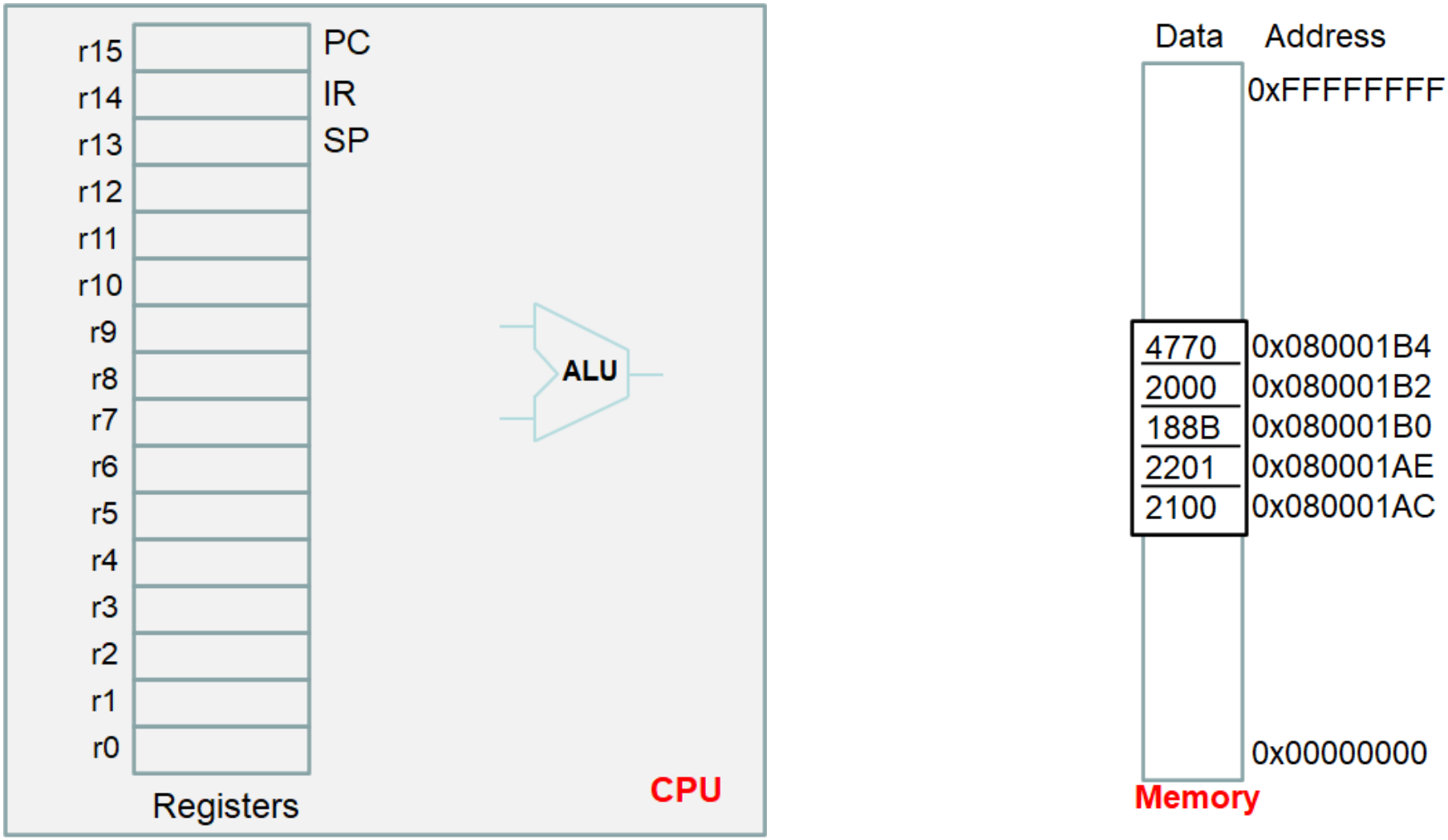


ARM Cortex-M4: Fetch/Decode/Execute Cycle

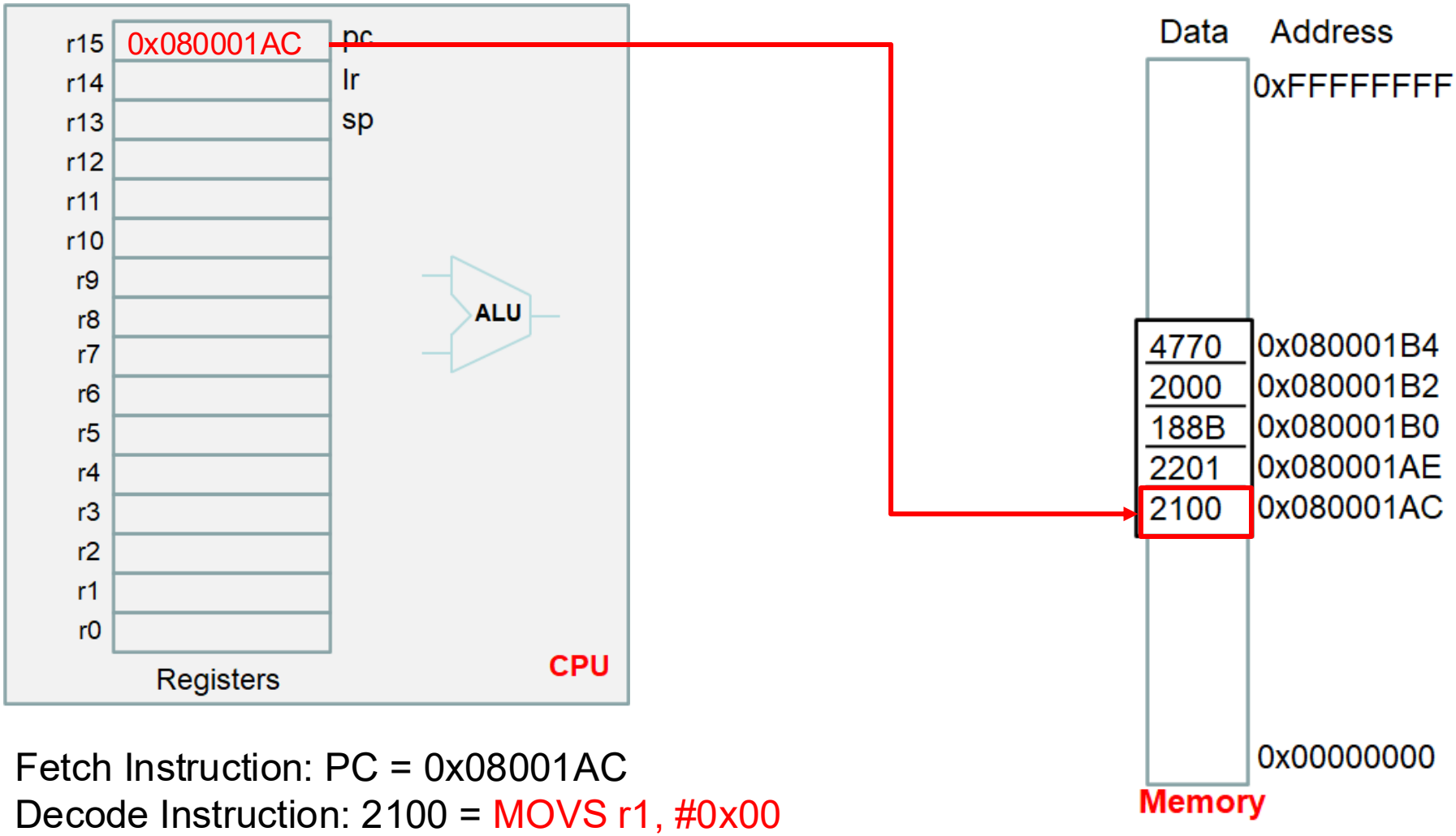
1. CPU keeps track of the location of the next instruction or data byte (to fetch) through the Program Counter (PC)
2. Instruction Decoder decodes the received data and configures the Arithmetic Logic Unit (ALU) to process it. Arithmetic or logic operations execute in the ALU
3. After the execution, the relevant bits are set, and data gets stored to a register or memory location if applicable
4. The Control Sequencer ensures the Fetch/Decode/Execute cycle repeats until the last instruction is detected



ARM Cortex-M4: Instruction Execution



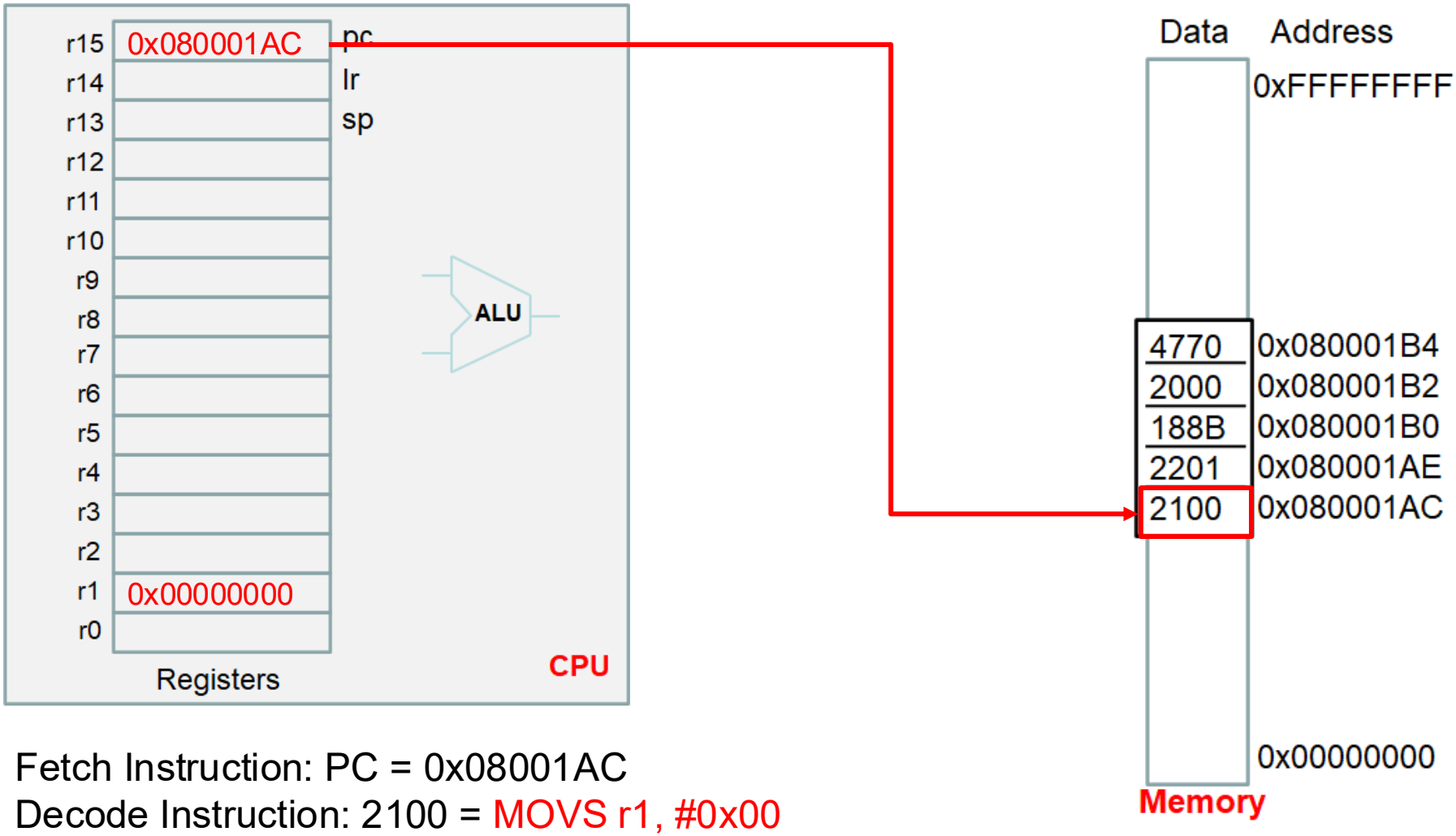
ARM Cortex-M4: Instruction Execution



Fetch Instruction: PC = 0x08001AC

Decode Instruction: 2100 = MOVS r1, #0x00

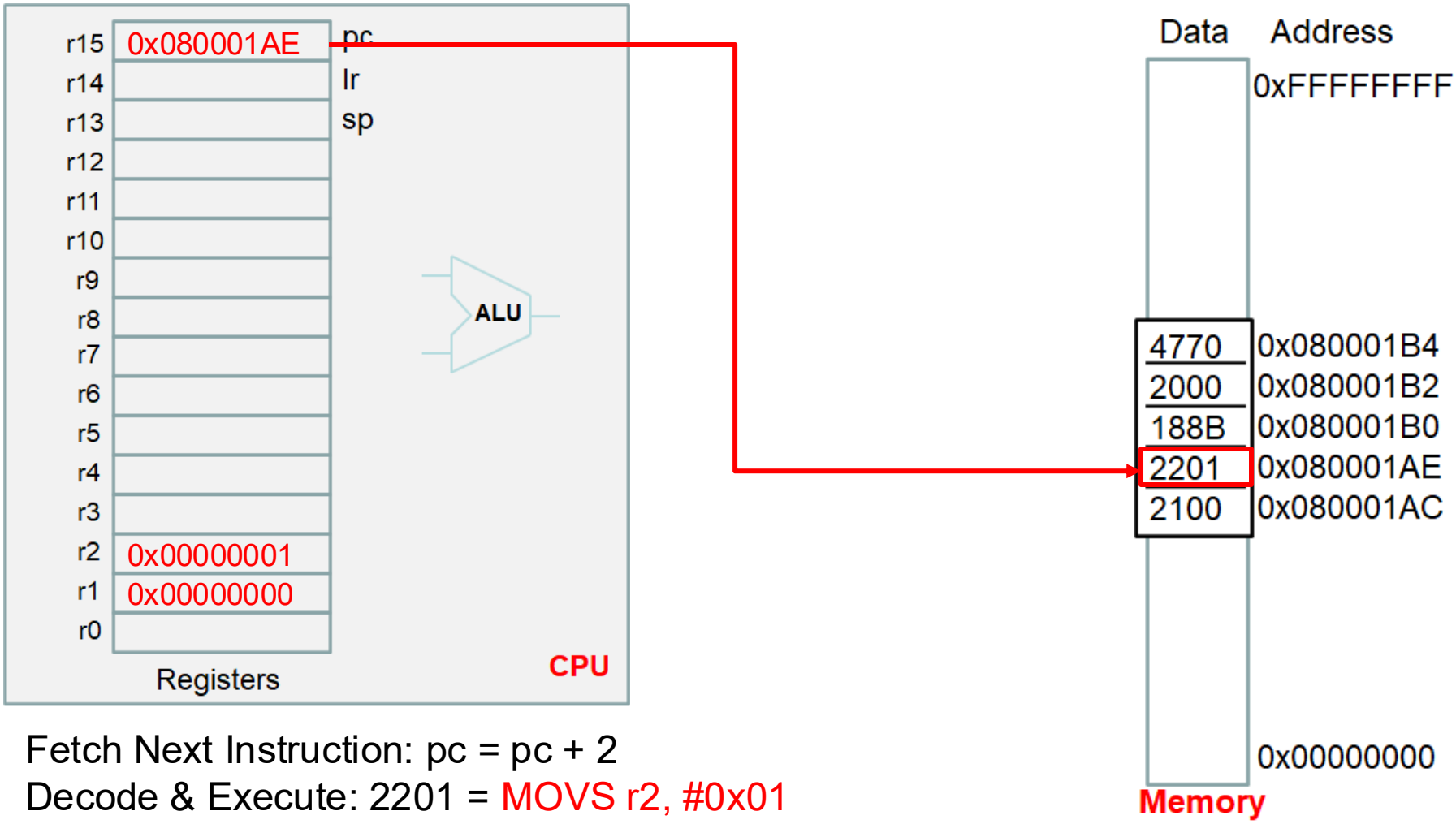
ARM Cortex-M4: Instruction Execution



Fetch Instruction: PC = 0x08001AC

Decode Instruction: 2100 = MOVS r1, #0x00

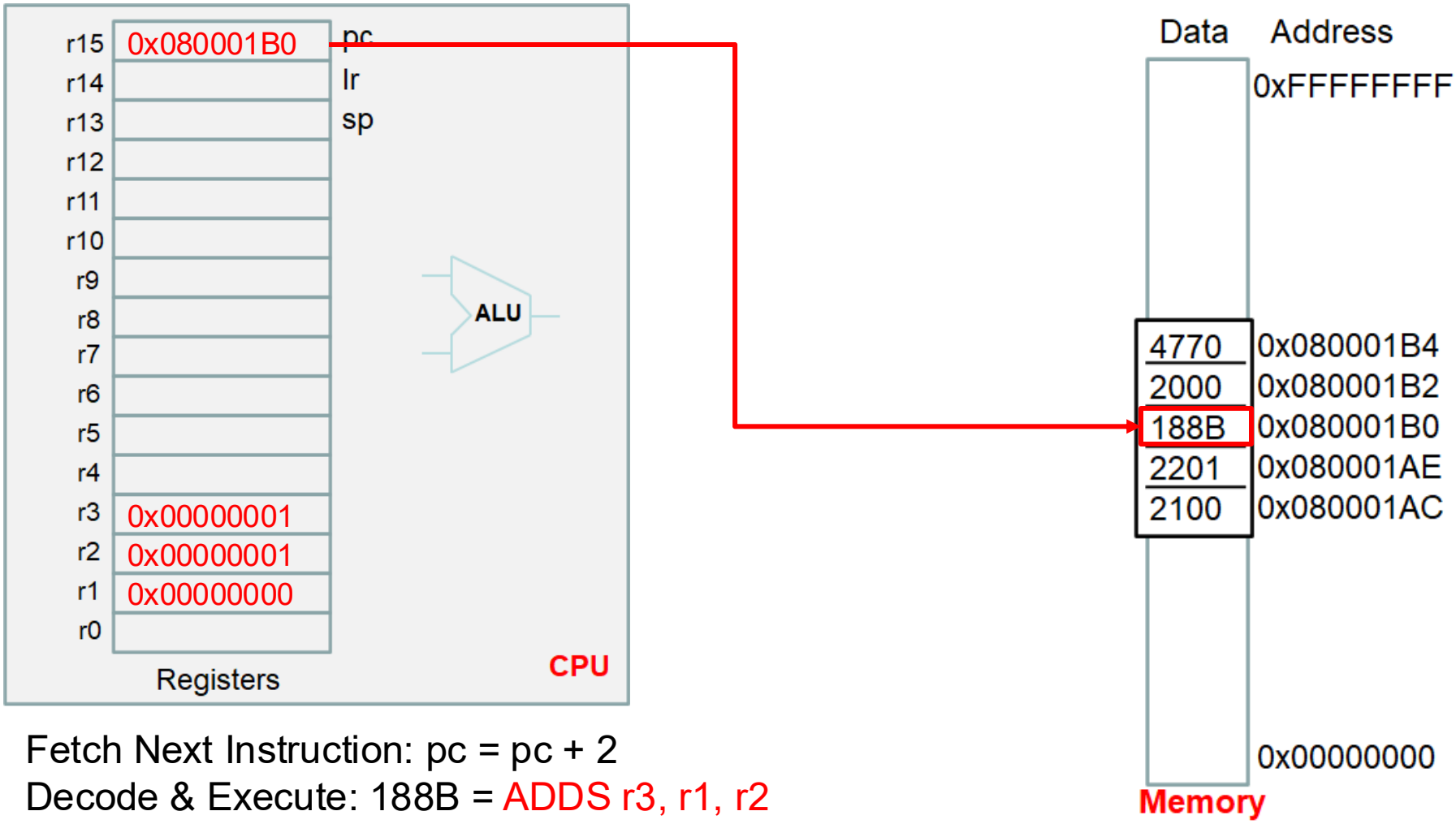
ARM Cortex-M4: Instruction Execution



Fetch Next Instruction: pc = pc + 2

Decode & Execute: 2201 = MOVS r2, #0x01

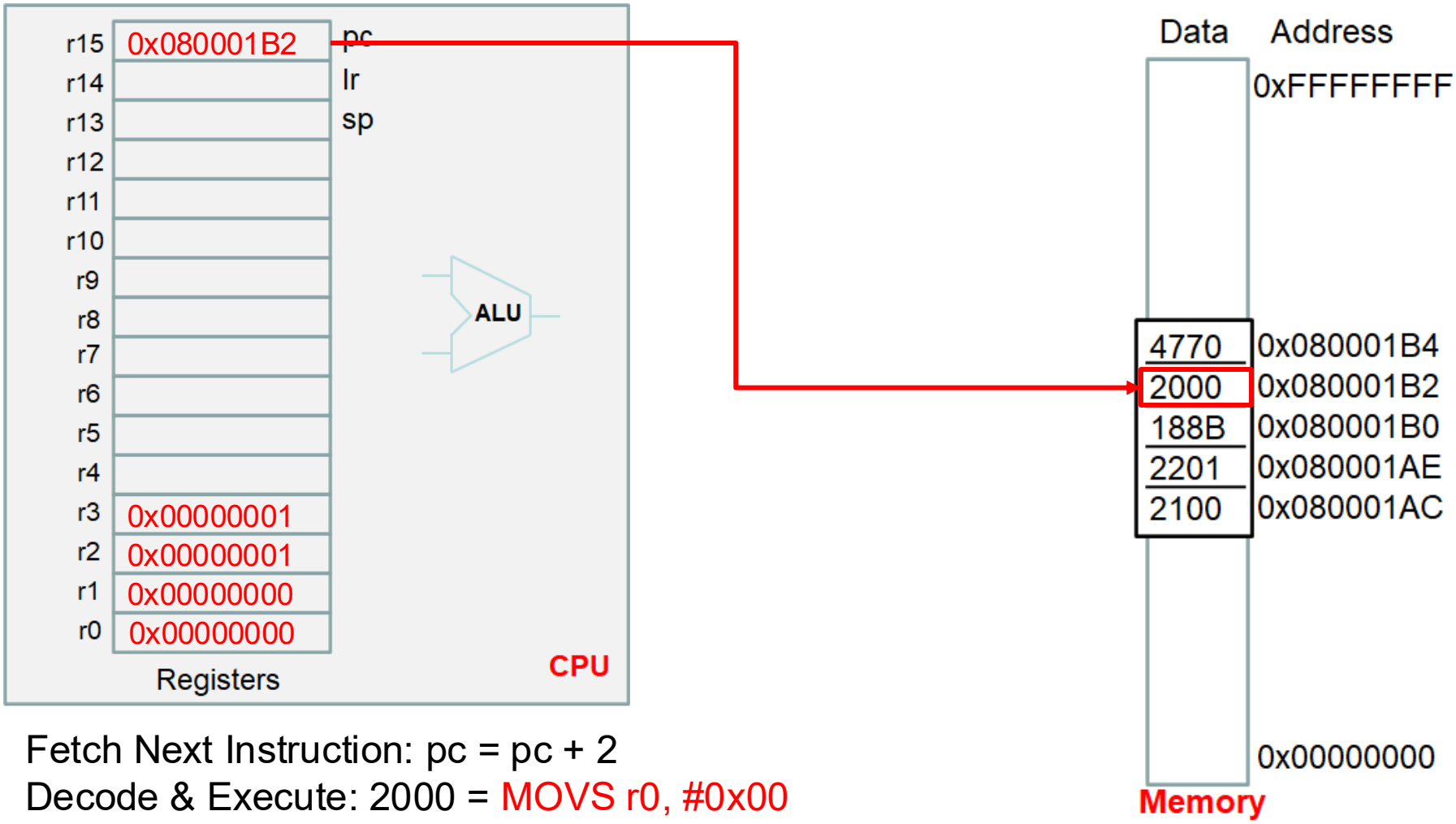
ARM Cortex-M4: Instruction Execution



Fetch Next Instruction: pc = pc + 2

Decode & Execute: 188B = ADDS r3, r1, r2

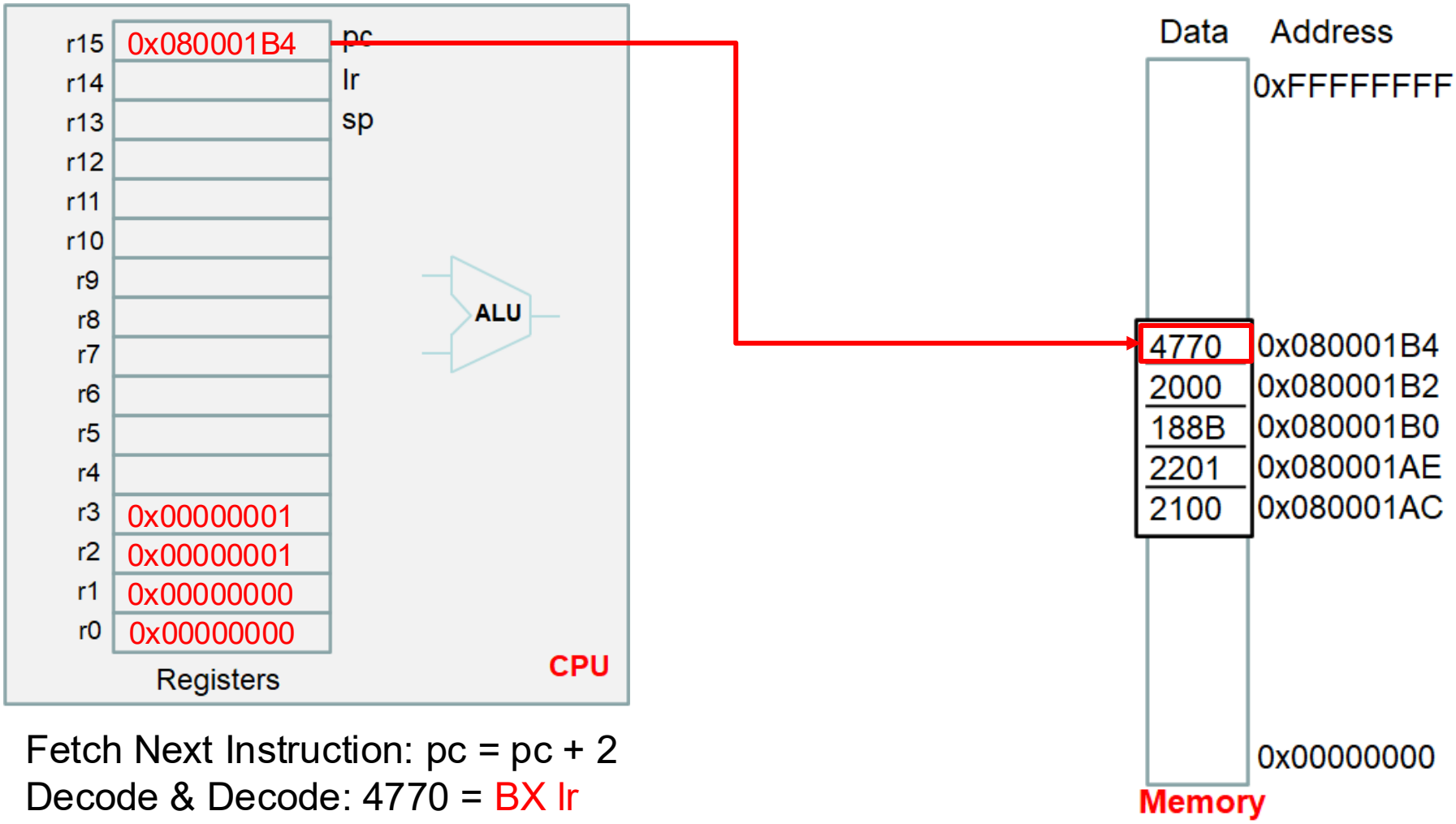
ARM Cortex-M4: Instruction Execution



Fetch Next Instruction: pc = pc + 2

Decode & Execute: 2000 = MOVS r0, #0x00

ARM Cortex-M4: Instruction Execution



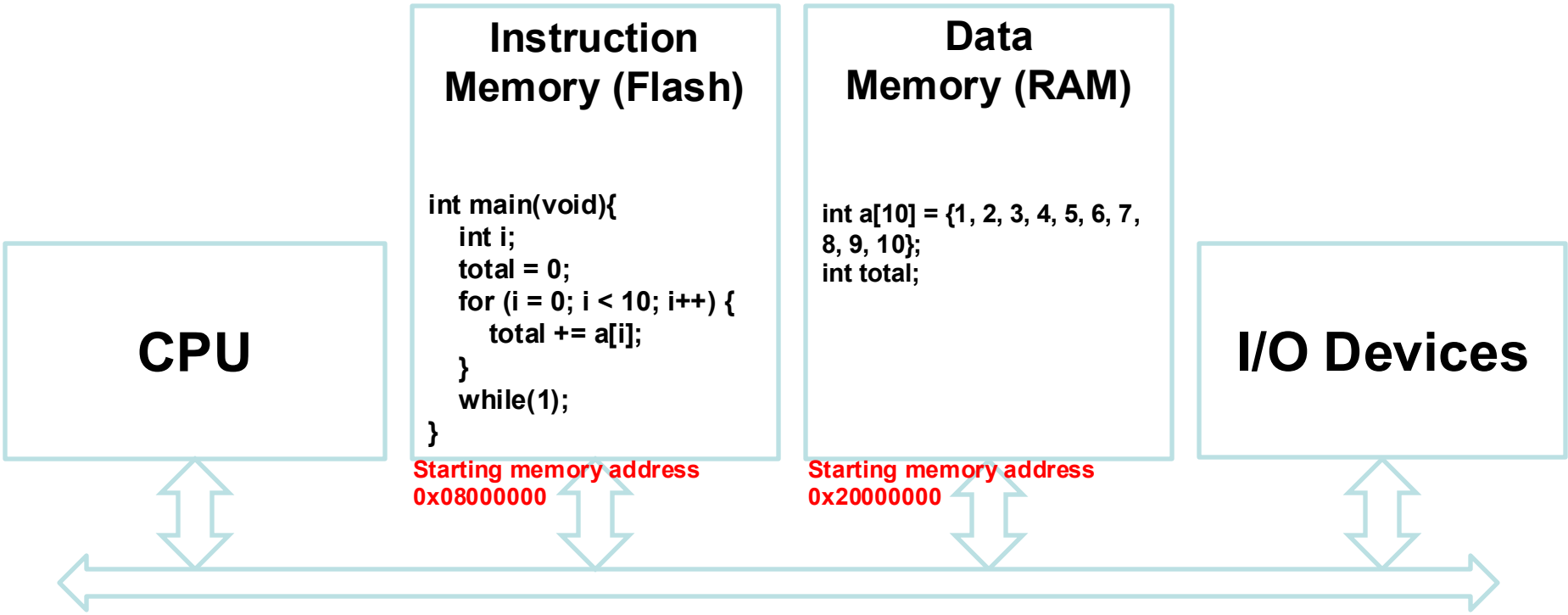
Fetch Next Instruction: pc = pc + 2
Decode & Decode: 4770 = BX lr

Example: Calculate the Sum of an Array

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int total;

int main(void){
        int i;
        total = 0;
        for (i = 0; i < 10; i++) {
                total += a[i];
        }
        while(1);
}
```

Example: Calculate the Sum of an Array



Example: Calculate the Sum of an Array

Instruction Memory (Flash)

```
int main(void){
   int i;
   total = 0;
   for (i = 0; i < 10; i++) {
      total += a[i];
   }
   while(1);
}
```

Starting memory address
0x08000000

```
0010 0001 0000 0000
0100 1010 0000 1000
0110 0000 0001 0001
0010 0000 0000 0000
1110 0000 0000 1000
0100 1001 0000 0111
1111 1000 0101 0001
0001 0000 0010 0000
0100 1010 0000 0100
0110 1000 0001 0010
0100 0100 0001 0001
0100 1010 0000 0011
0110 0000 0001 0001
0001 1100 0100 0000
0010 1000 0000 1010
1101 1011 1111 0100
1011 1111 0000 0000
1110 0111 1111 1110
```

```
        MOVS  r1, #0x00
        LDR   r2, = total_addr
        STR   r1, [r2, #0x00]
        MOVS  r0, #0x00
        B     Check
Loop: LDR   r1, = a_addr
        LDR   r1, [r1, r0, LSL #2]
        LDR   r2, = total_addr
        LDR   r2, [r2, #0x00]
        ADD   r1, r1, r2
        LDR   r2, = total_addr
        STR   r1, [r2,#0x00]
        ADDS  r0, r0, #1
Check: CMP  r0, #0x0A
       BLT  Loop
       NOP
Self:  B    Self
```

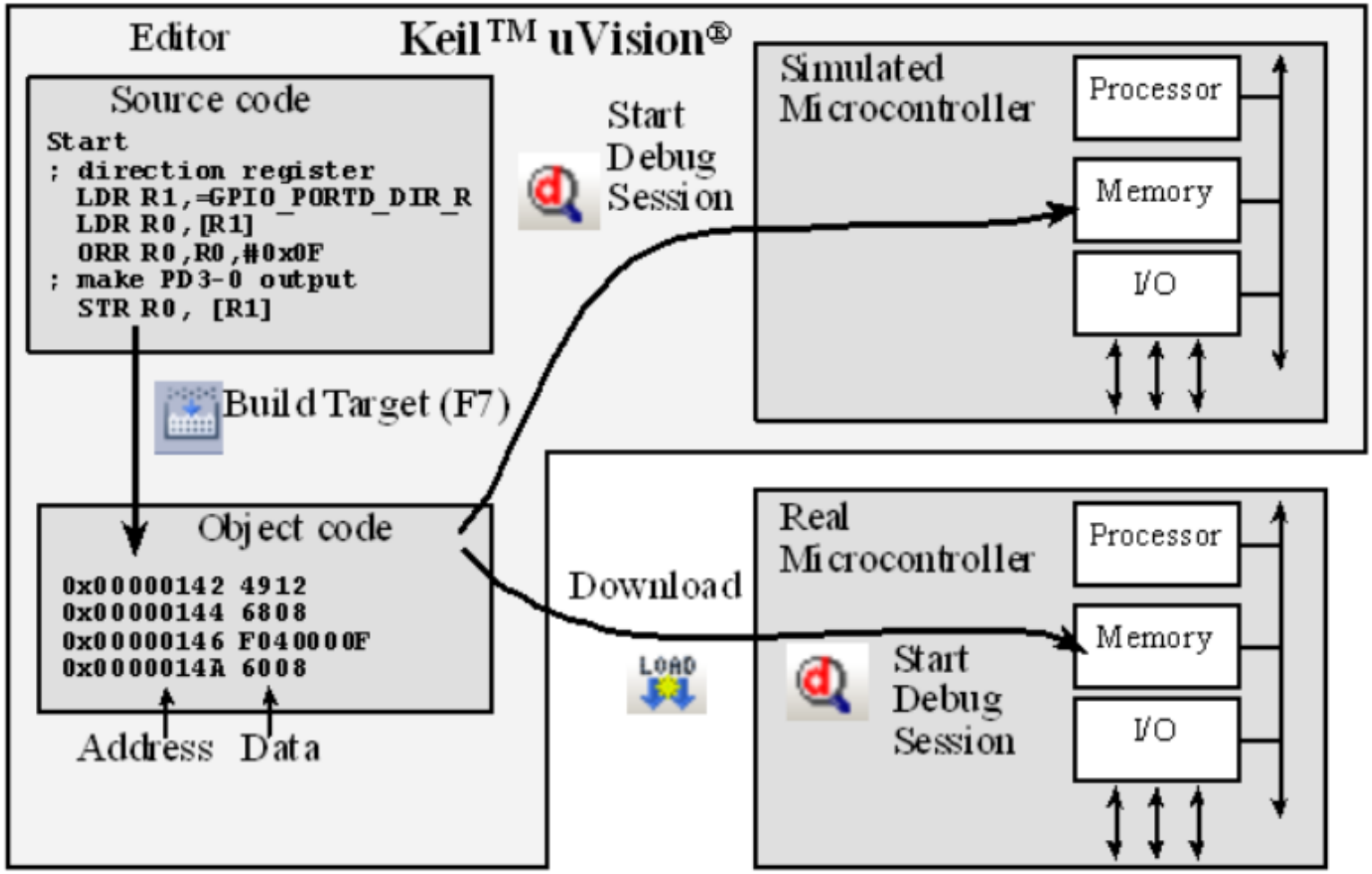
Example: Calculate the Sum of an Array

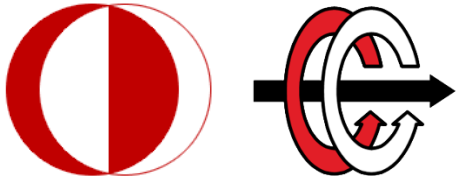
Data Memory (RAM)

```
int a[10] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int total;
```

Assume the starting memory address of the data memory is 0x20000000

Keil Integrated Development Environment (IDE)





Memory concepts, address decoding

Week 1