EEE447 Introduction to Microprocessors

Week 1

Course overview and introduction

Microprocessor, microcontroller and programming basics

Course Schedule (@d131)

S1: Tuesday 9:40–11:30 and Thursday 8:40-10:30

S2: Tuesday 11:40–13:30 and Thursday 10:40-12:30

Textbook

- Embedded Systems: Introduction to ARM Cortex-M Microcontrollers, Fifth Edition, Jonathan W Valvano
- The Definitive Guide to ARM Cortex M3 and Cortex M4 Processors, Third Edition, Joseph Yiu
- Embedded Systems with ARM Cortex-M Microcontrollers in Assembly Language and C (Second Edition), Oct 2015, Yifeng Zhu.
- Digital Design and Computer Architecture
  ARM Edition, Harris and Harris, 2015

Other references:

- Cortex M4 Technical Reference Manual
- Cortex M4 Devices Generic User Guide
- Tiva TM4C123G Microcontroller Data Sheet
- Tiva TM4C123G Development Board User's Guide
- ARM and Thumb-2 Instruction Set Quick Reference Card

Grading

MT Exam 1	20%
Final	25%
Project	15%
Quiz +HW+Att	15%
Laboratory Work	25%

Course objectives

- To develop an in-depth understanding of
  - the operation of microprocessors and microcontrollers,
  - machine language programming,
  - microprocessor interfacing techniques,
  - designing and interfacing of microcontroller-based embedded systems in both hardware and software,
  - basic programming of ARM Cortex chips in assembly language/C and learning the fundamentals of embedded system design.
- Ultimate goal: to be able to apply this knowledge to more advanced structures.

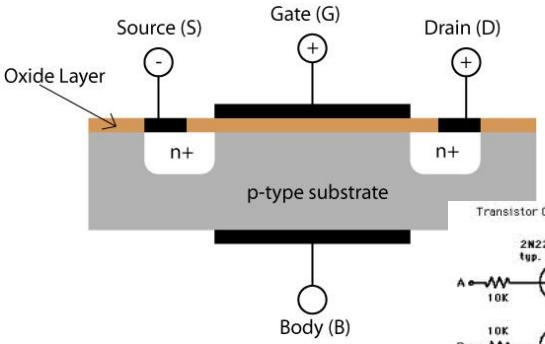
What we expect of you ...

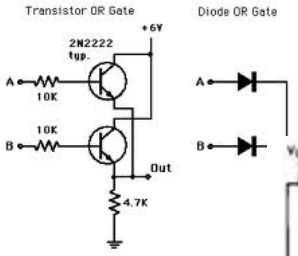
- A desire to learn about microprocessors and microcontrollers and their applications

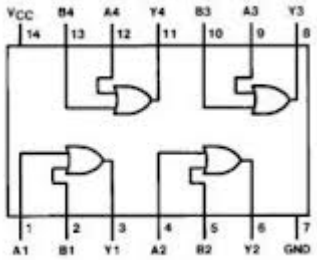
- To do some work:
  - Attend the classes and study regularly
  - Attend exams
  - Do the laboratory assignments
  - Submit homeworks, etc.

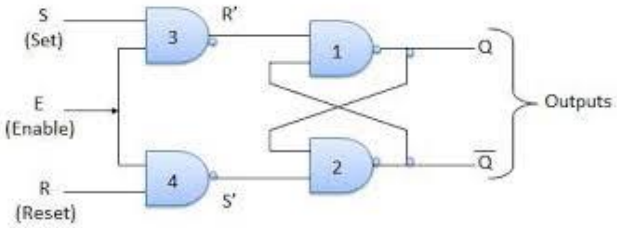
Outline

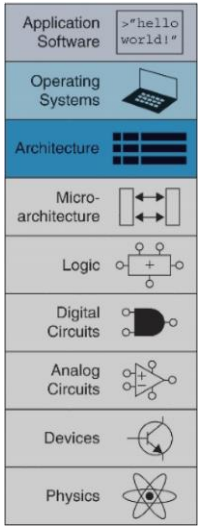
- Introduction
- ARM Architecture
- Assembly Programming
- Programming Examples
- Stacks, Subroutines, Interrupts
- Timer Subsystem
- Parallel I/O System
- Analog to Digital Conversion System
- Serial System

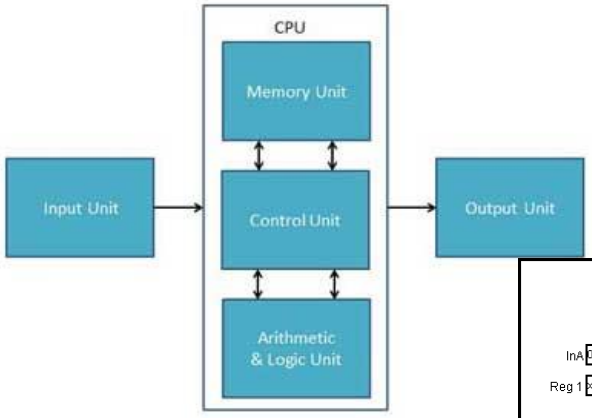


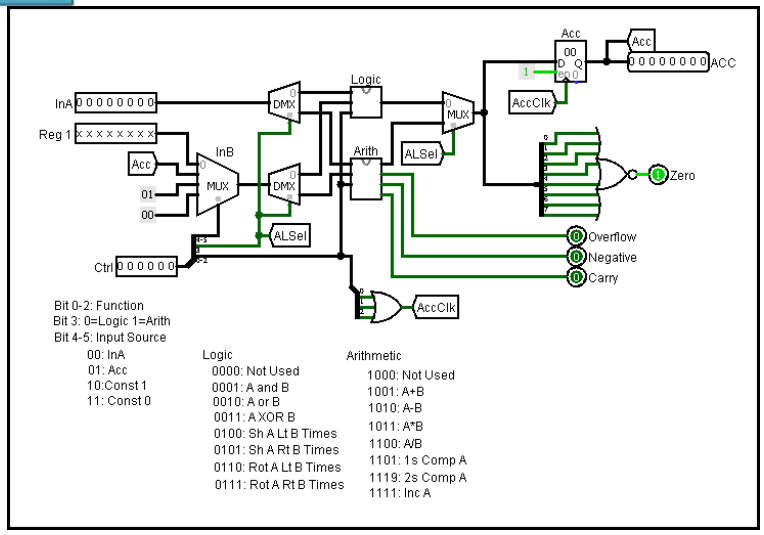


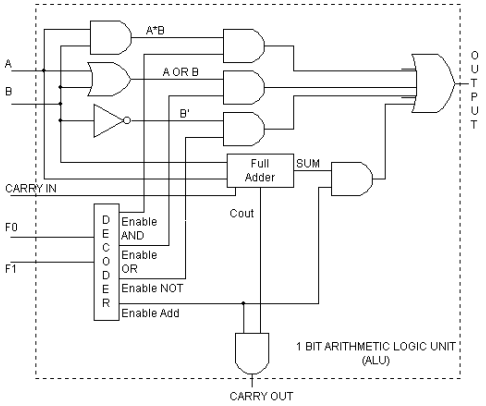












Enable AND = F0'*F1'

Enable OR = F0'* F1

Enable NOT = F0 * F1'

Enable Add = F0 * F1

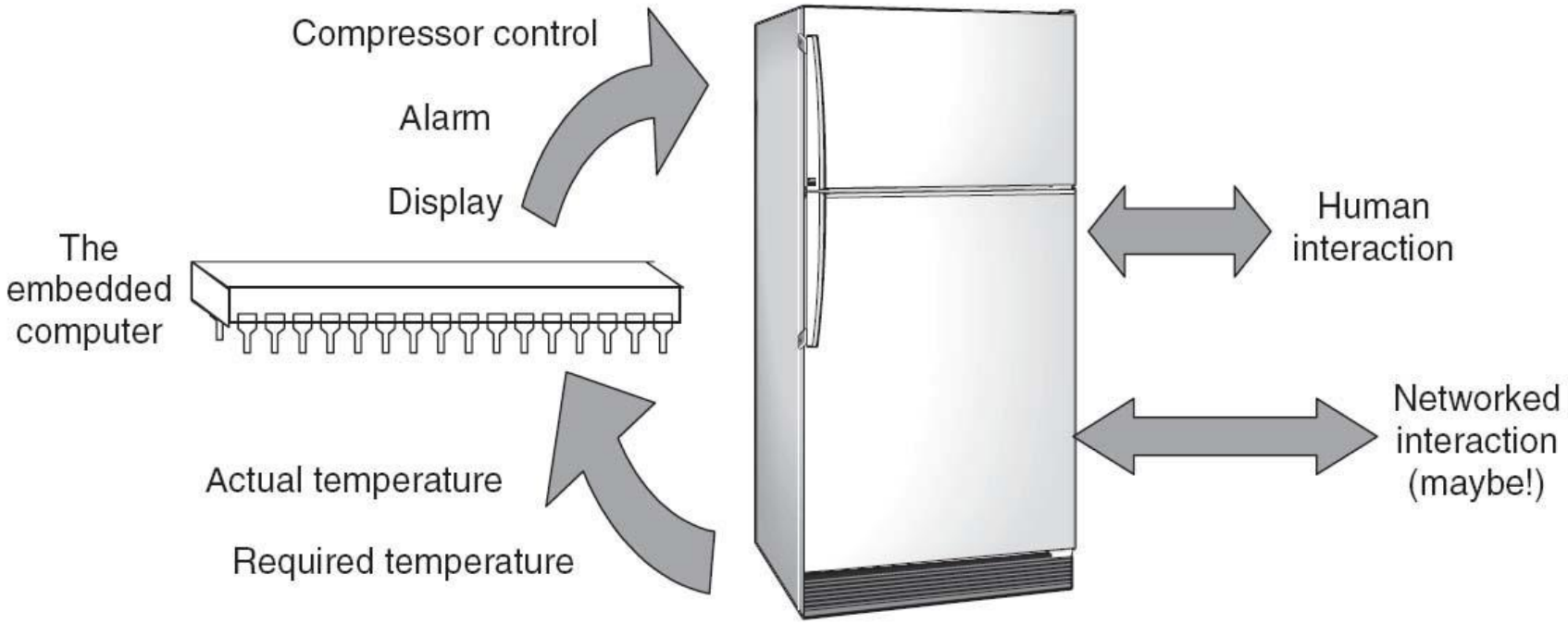
Embedded Systems

Embedded systems

- It is a special purpose system that is used to perform one of few dedicated functions. Simply any computer system embedded inside an electronic device but is not itself a general-purpose computer.

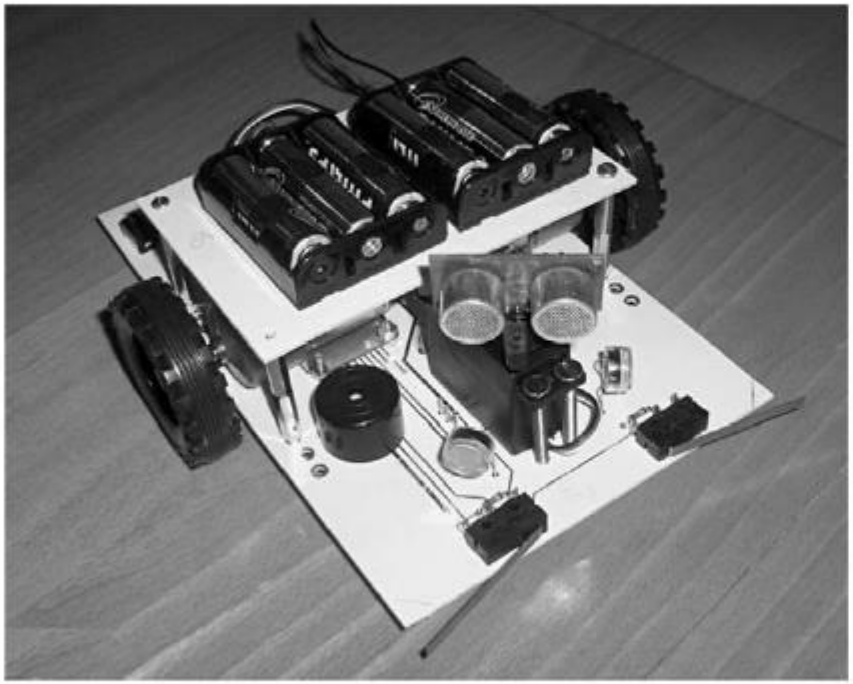
  – dedicated to specific function(s)
  – interact with environment
  – real-time requirements

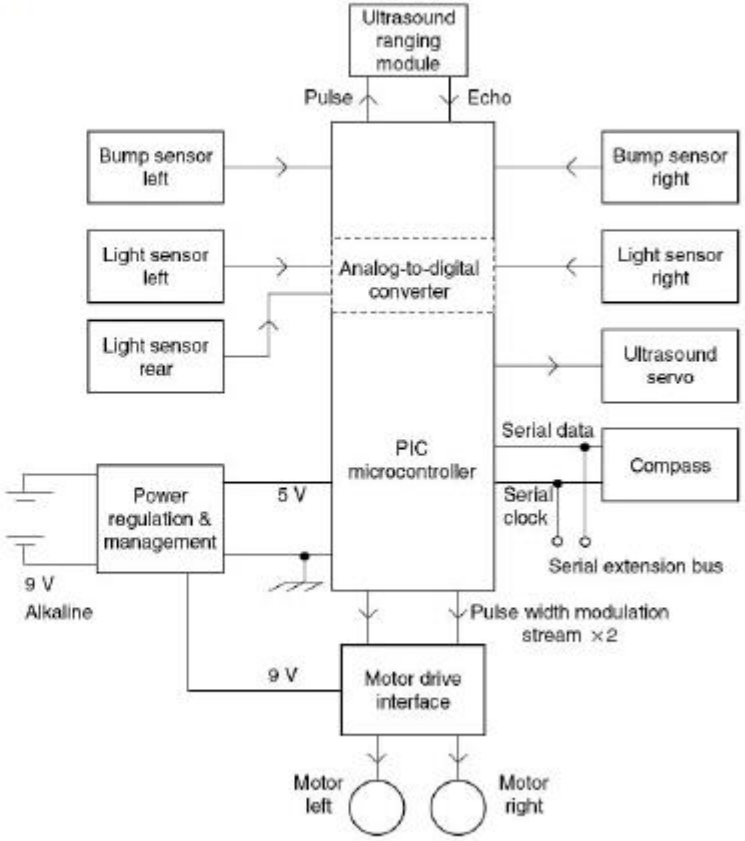
EXAMPLES



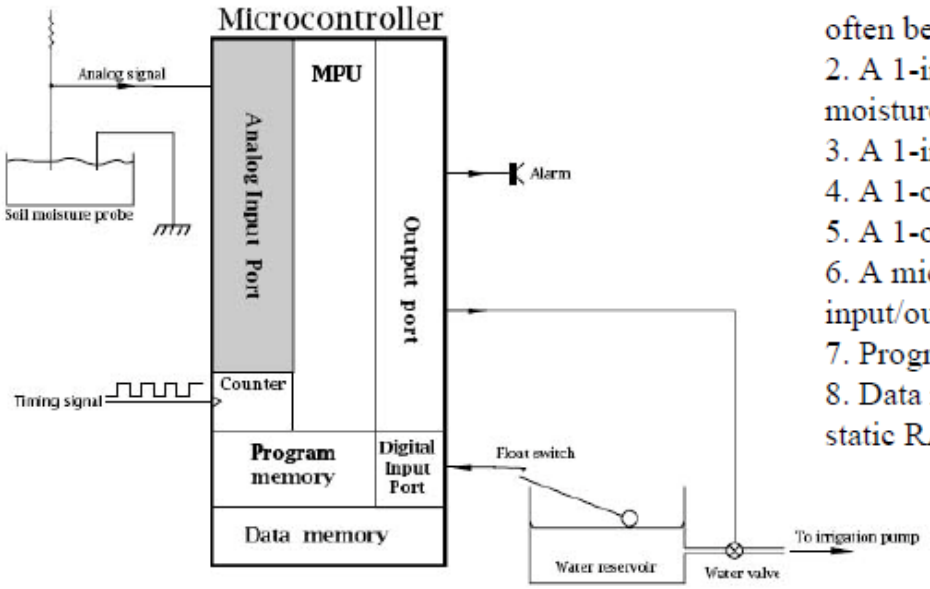
Embedded system example 1: the refrigerator

Autonomous Guided Vehicle (AVG)





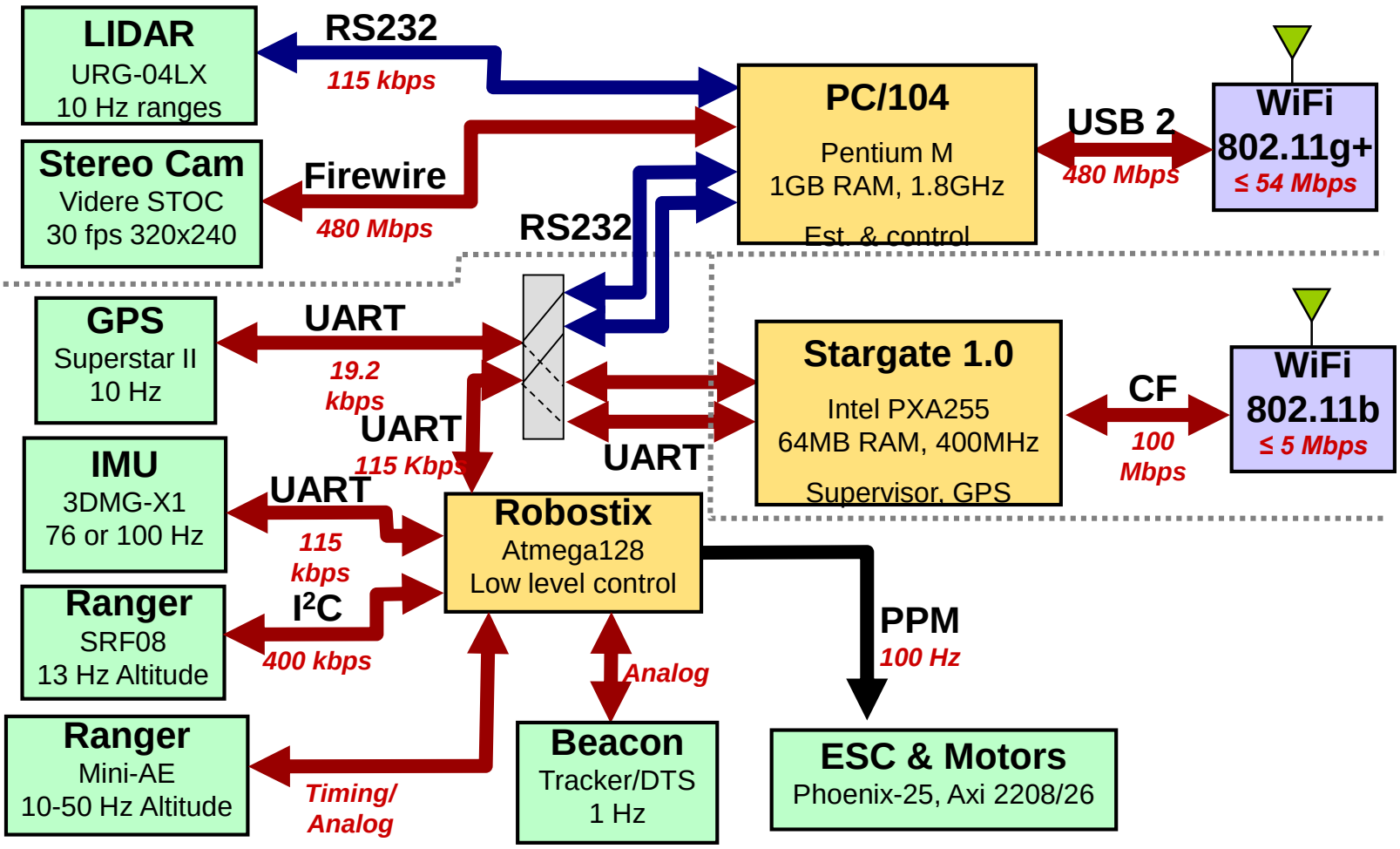
- A greenhouse controller is to monitor an analog signal from a soil moisture probe and if below a certain value turn on a water valve for 5 seconds and off for 5 seconds.



Resource Budget (List of subsystem):

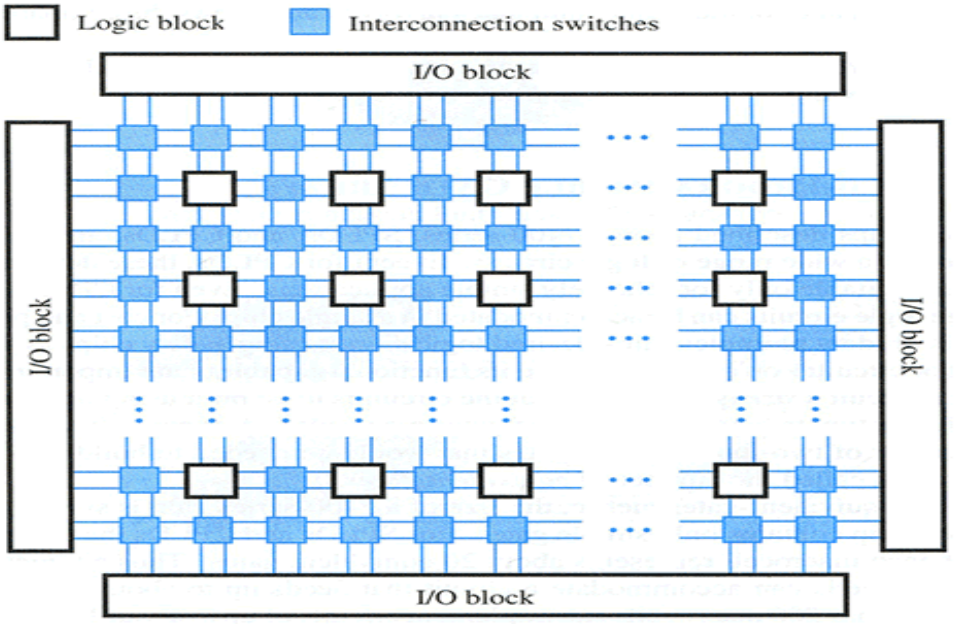
1. An input for an external oscillator, connected to a counter/timer to allow the MCU to calculate time. In practice the system clock can often be used by this internal timer to measure duration.
2. A 1-input analog input line to measure the analog signal from the moisture detector
3. A 1-input digital line to check the level of the reservoir water tank
4. A 1-output digital line to open and close the water valve
5. A 1-output digital line to activate the buzzer alarm
6. A microprocessor to do the calculations and to read/write to the input/output ports, respectively
7. Program memory, usually ROM of some kind
8. Data memory for temporary storage of program variables, usually static RAM

STARMAC quadrotor aircraft

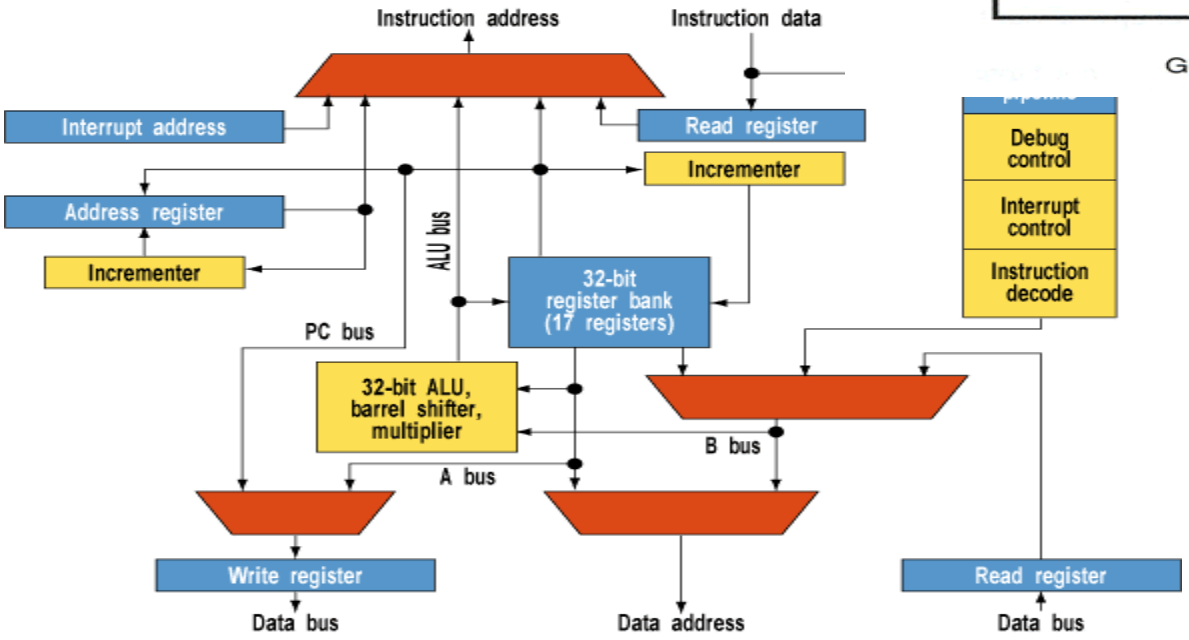


Since this is an introductory course to processors and embedded systems, will look at simpler systems here… but the learning will be applicable and scalable to more complex systems.

By using microcontrollers (or microprocessor based systems): Slower (technology dependent), only software is needed to be updated.



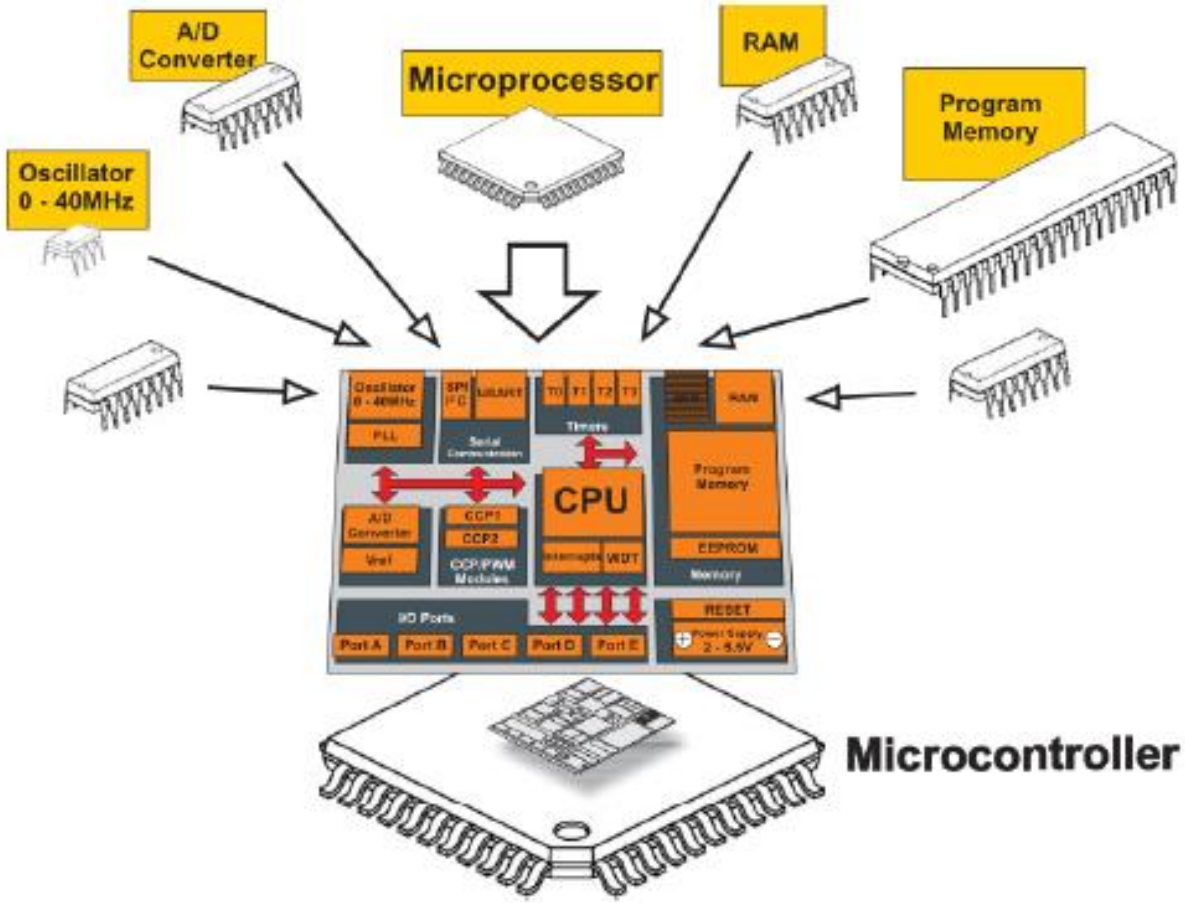
General structure of an FPGA



The Cortex M3's Thumbnail architecture looks like a conventional Arm processor. The differences are found in the Harvard architecture and the instruction decode that handles only Thumb and Thumb 2 instructions.

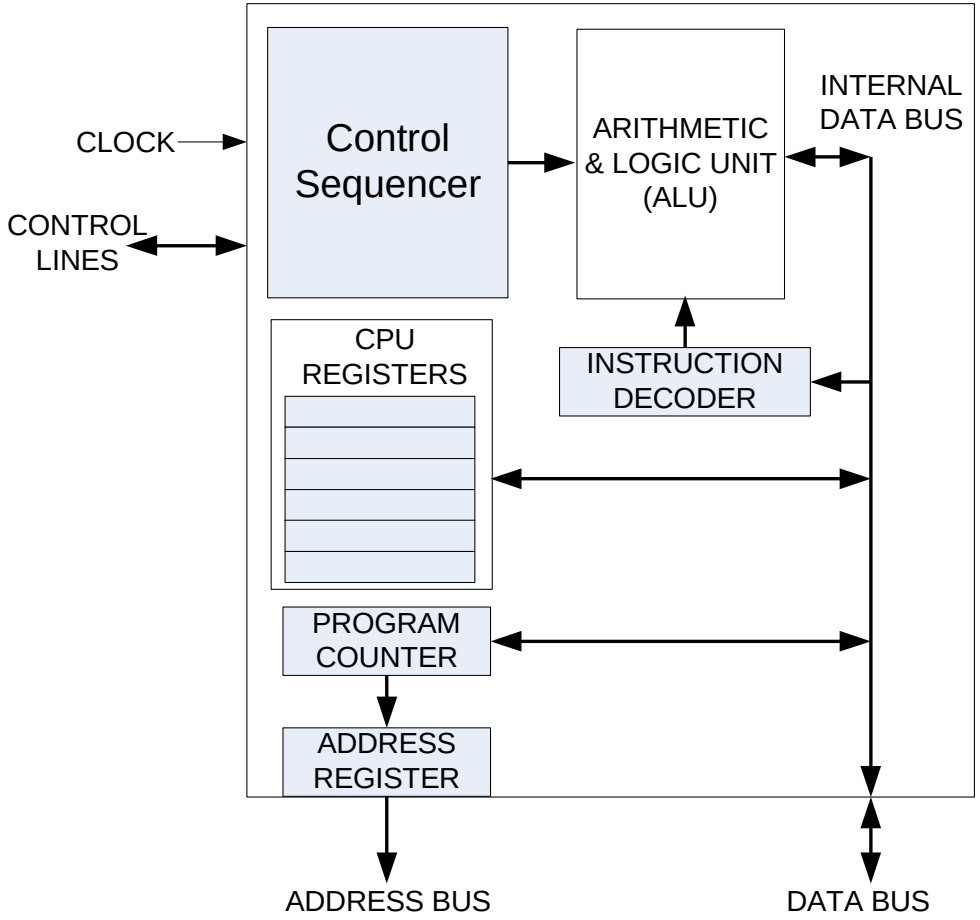
By using digital circuits (Field Programmable Gate Array - FPGA): Faster (only propagation delay), functions they perform can't be changed easily.

Microprocessor based systems

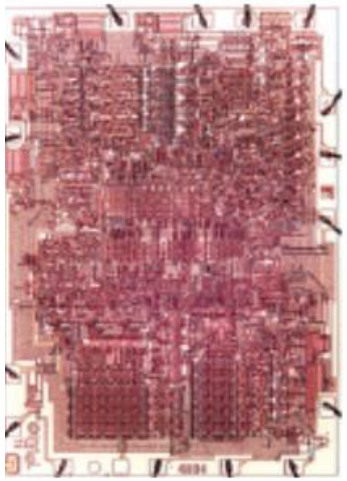

A/D Converter
Microprocessor
RAM
Program Memory
Oscillator 0 - 40MHz
Oscillator 0 - 40MHz
PLL
Timers
CPU
RAM
Program Memory
A/D Converter
CCP1
CCP2
CCP/PWM Modules
EEPROM
I/O Ports
Port A
Port B
Port C
Port D
Port E
RESET
2 - 5.5V
Microcontroller

Microprocessor

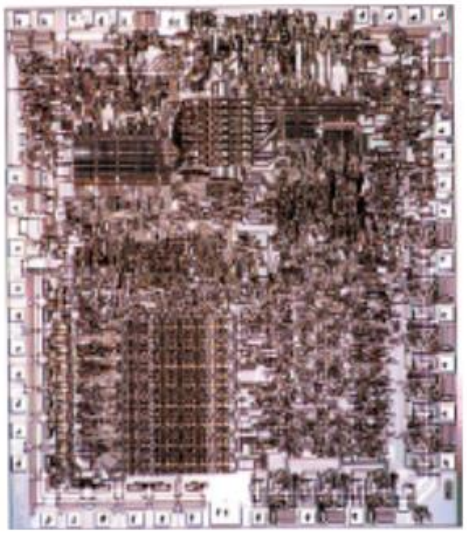
- A single LSI chip, known as Central Processing Unit (CPU), to perform all computation (without memory and I/O interface circuit)



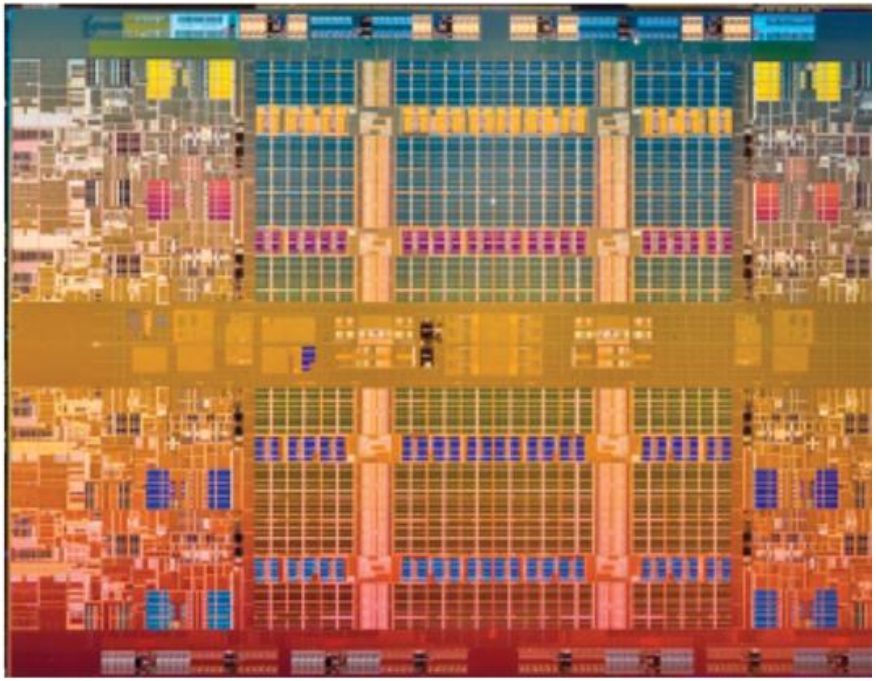
The first microprocessors appeared in Nov 1971 (Intel 4004)



Intel 4004, 1971
1 core, no cache
23K transistors

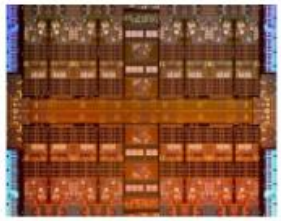


Intel 8008, 1978
1 core, no cache
29K transistors

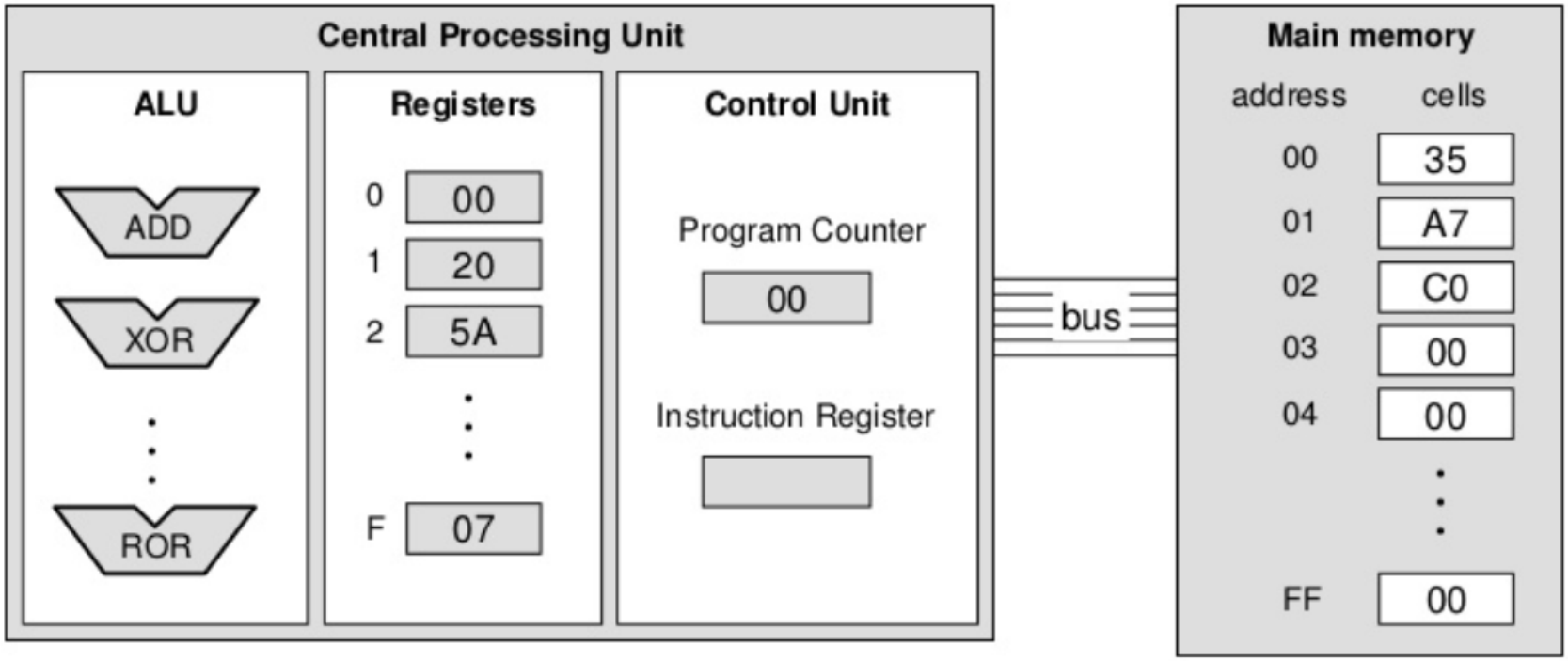


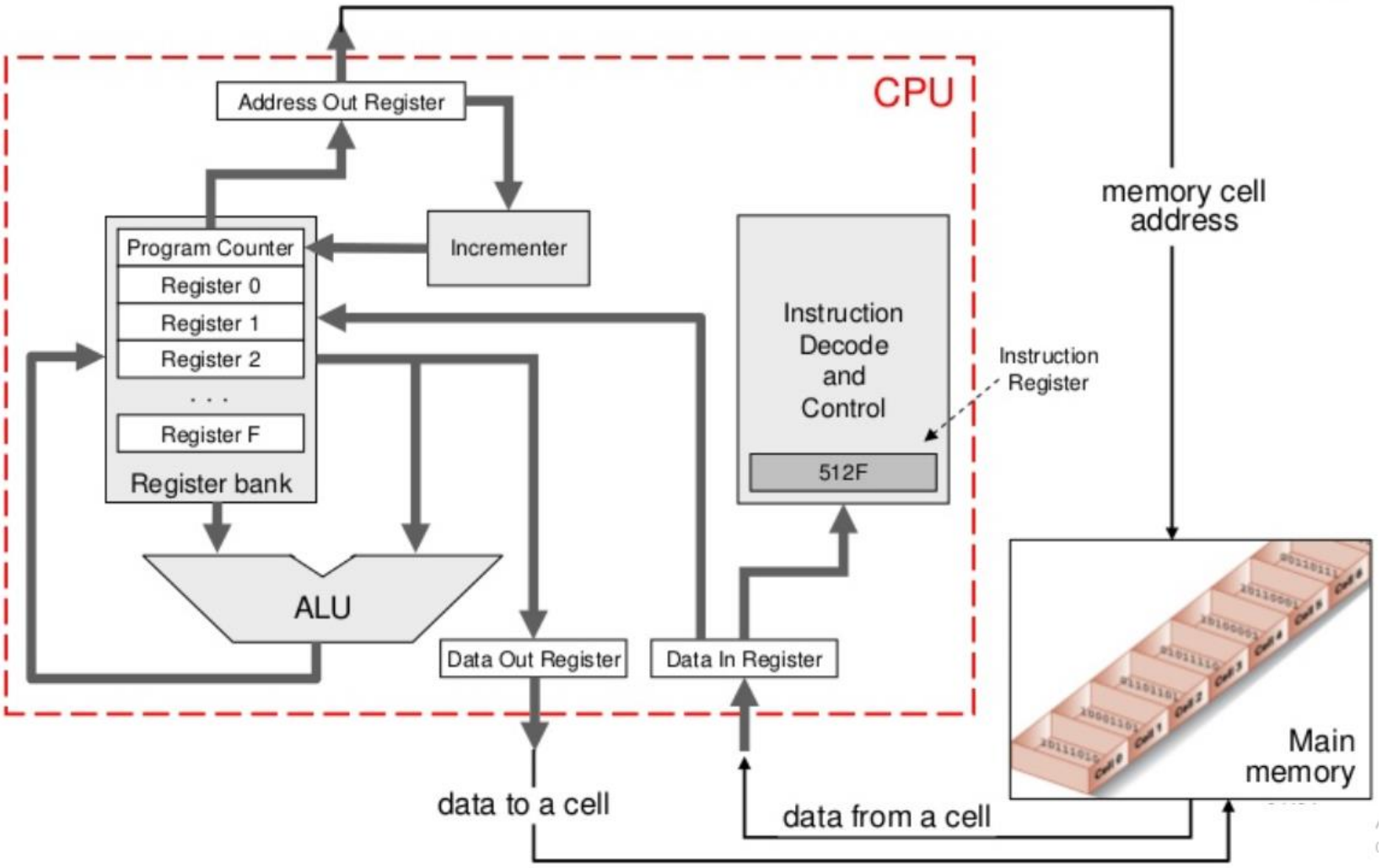
Intel Nehalem-EX, 2009
8 cores, 24MB cache
2.3B transistors

Figure credit: Shekhar Borkar, Andrew A. Chien, The Future of Microprocessors. Communications of the ACM, Vol. 54 No. 5, Pages 67-77 10.1145/1941487.1941507.



Oracle SPARC M7 (2015)
32 cores; > 10B transistors


Central Processing Unit
ALU
ADD
XOR
ROR
Registers
0
00
1
20
2
5A
F
07
Control Unit
Program Counter
00
Instruction Register
bus
Main memory
address
cells
00
35
01
A7
02
C0
03
00
04
00
FF
00

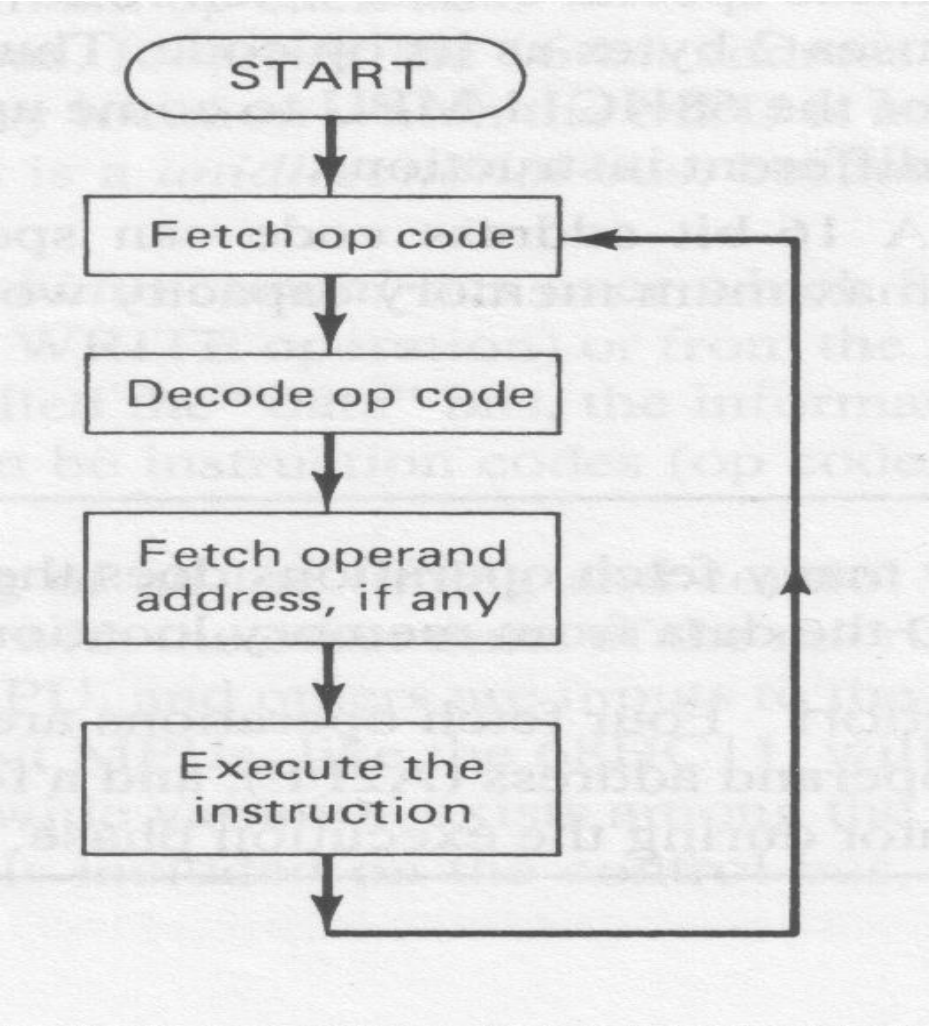

CPU
Address Out Register
memory cell address
Program Counter
Register 0
Register 1
Register 2
. . .
Register F
Register bank
Incrementer
Instruction Decode and Control
512F
Instruction Register
ALU
Data Out Register
Data In Register
Main memory
data to a cell
data from a cell

CPU Structure

- The register array:
  - consists of at least
    - one data register (ACC,Rx)
    - program counter (PC)
    - stack pointer (SP)

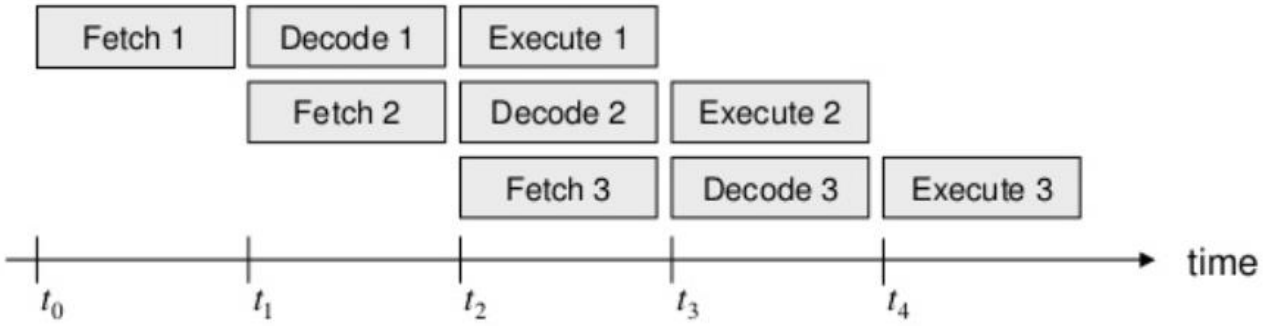
- The control unit
  - controls all the operations in a CPU and basically it puts the CPU in one of the *fetch* and *execution* phases

Operation of the CPU



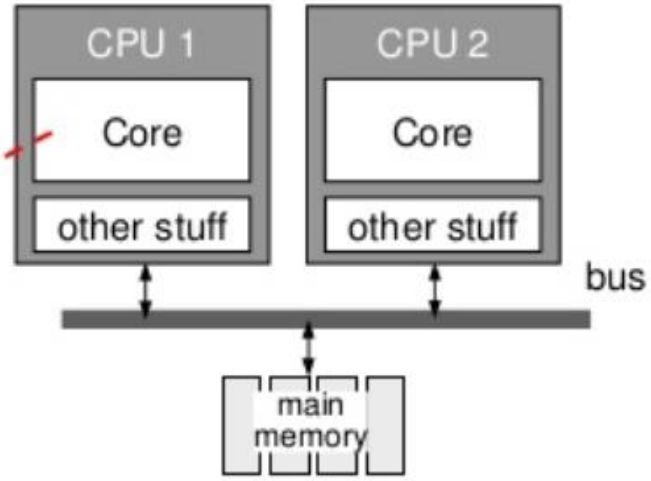
Improving CPU architecture

- Pipelining (overlap steps of CPU operation cycle)

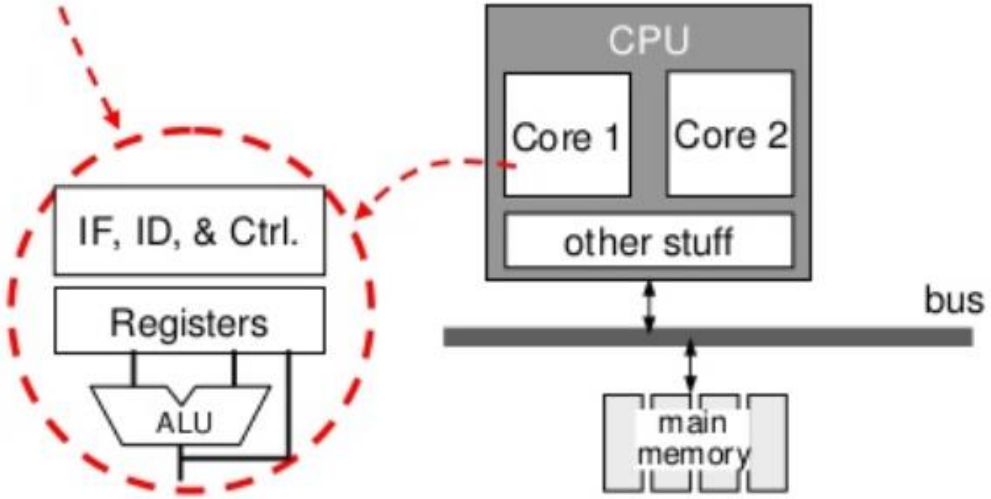


- Parallel processing: Multiple operations simultaneously within CPU or across CPU

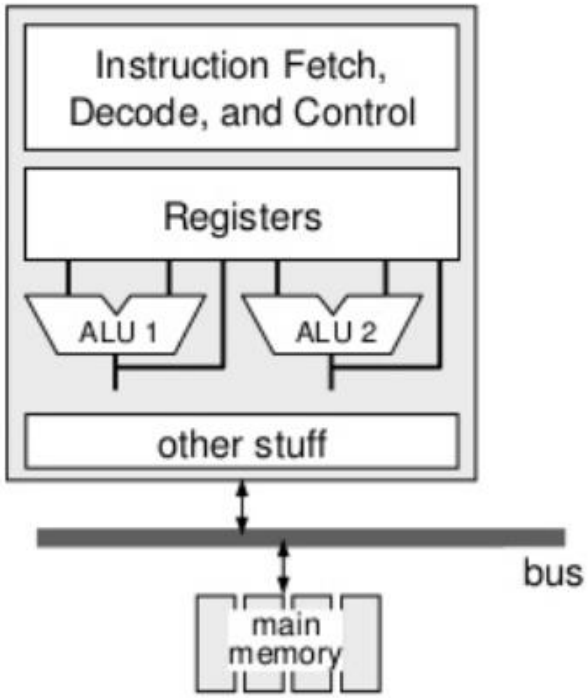
❑ Multi-CPU Architecture:



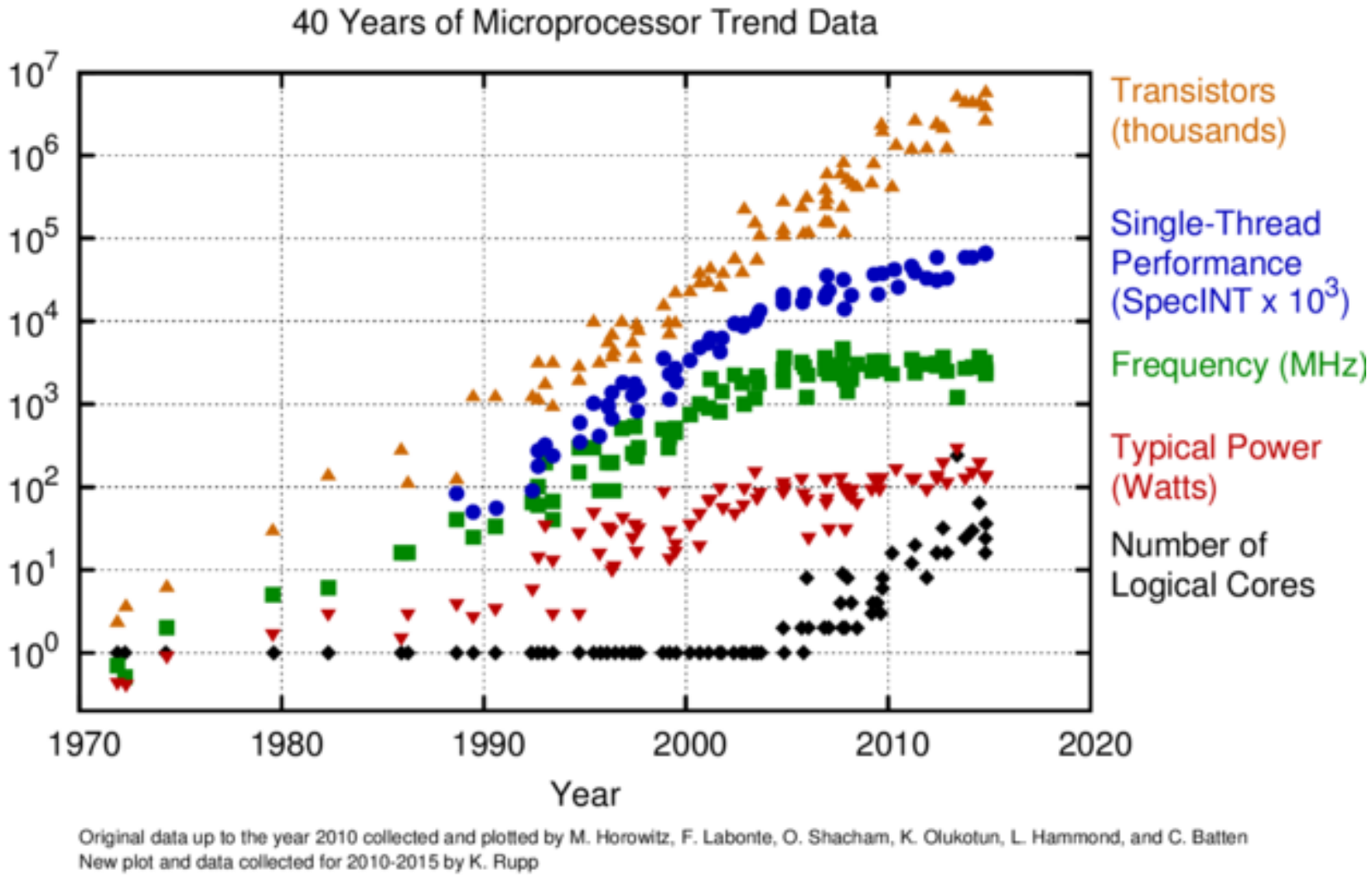
❑ Multi-Core Architecture:



❑ Multi-ALU Architecture:



Note: "other stuff" could be interface logic, cache, MMU, timer, … etc.



https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/

Some design issues

- Power consumption
  - multiple cores + customization can improve energy efficiency
- Accelerators for specialized tasks
  - graphics
  - media
  - image
- Faster transistor switching supports higher clock rates
- Microarchitecture advances
  - enabled by additional transistors
  - examples: pipelining, out of order execution, branch prediction

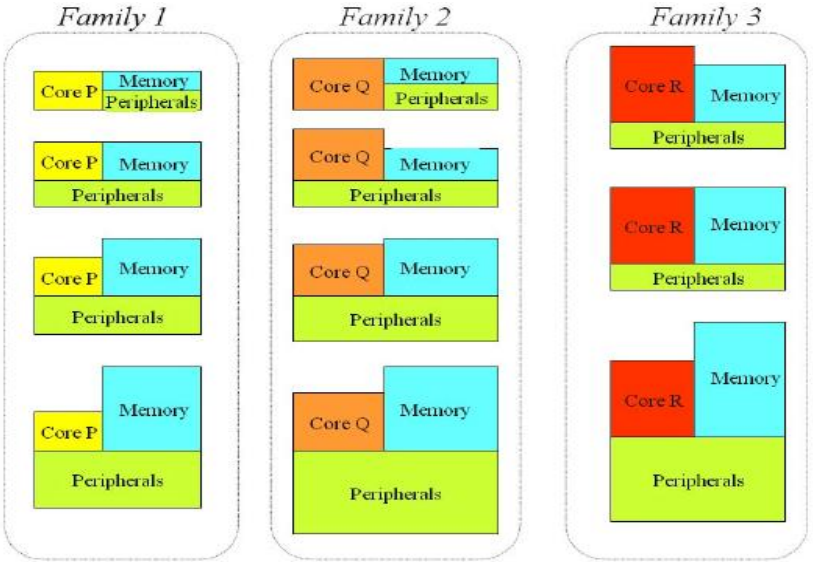
We are going to study ARM Cortex-M4 in this course

- Cortex-M4 Processor
  - Introduced in 2010
  - Designed with a large variety of highly efficient signal processing features
  - Features extended single-cycle multiply accumulate instructions, optimized SIMD arithmetic, saturating arithmetic and an optional Floating Point Unit.
- High Performance Efficiency
  - 1.25 DMIPS/MHz (DhrystoneMillion 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

Why ARM?

- 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)

- ARM has several designs which can be grouped in different architectures, families and cores.
  - The architectures are the specifications, i.e. the set of registers, instructions and operation modes that should be supported by implementations of the architecture.
  - A family is a specific detailed implementation of an architecture, i.e. the actual hardware details needed to create an ARM core.
  - Finally a core is a specific implementation of an architecture, i.e. the actual blue-print of the transistors and other discrete parts needed to create a ARM CPU.



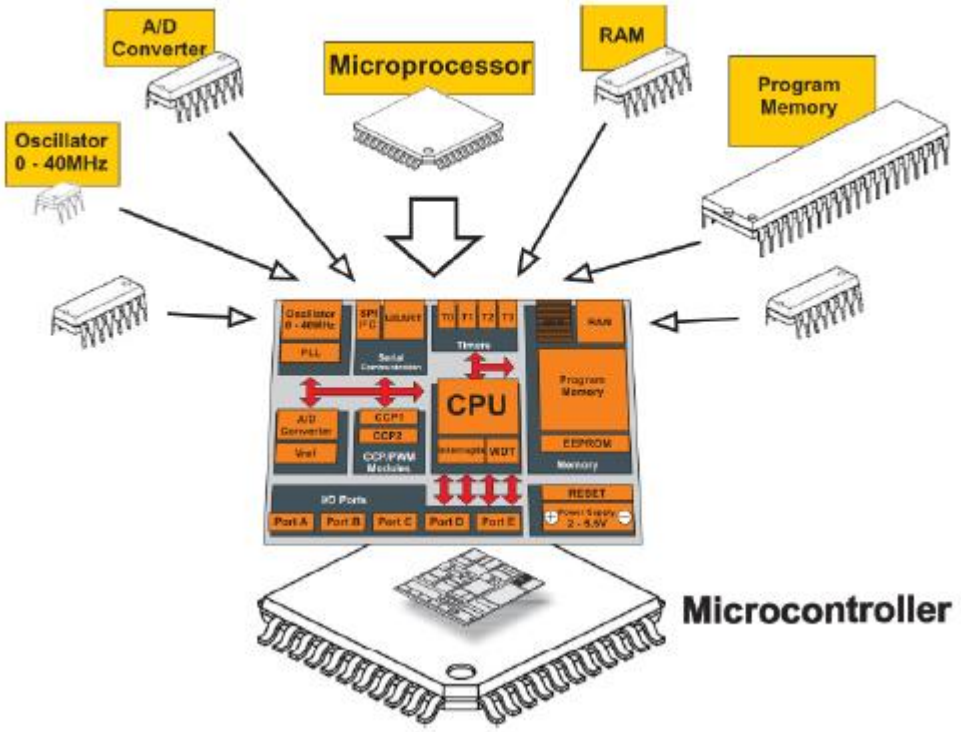
Family	Architecture	Cores
ARM7TDMI	ARMv4T	ARM7TDMI(S)
ARM9 ARM9E	ARMv5TE(J)	ARM926EJ-S, ARM966E-S
ARM11	ARMv6 (T2)	ARM1136(F), 1156T2(F)-S, 1176JZ(F), ARM11 MPCore™
Cortex-A	ARMv7-A	Cortex-A5, A7, A8, A9, A15
Cortex-R	ARMv7-R	Cortex-R4(F)
Cortex-M	ARMv7-M	Cortex-M3, M4
	ARMv6-M	Cortex-M1, M0
NEW !	ARMv8-A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

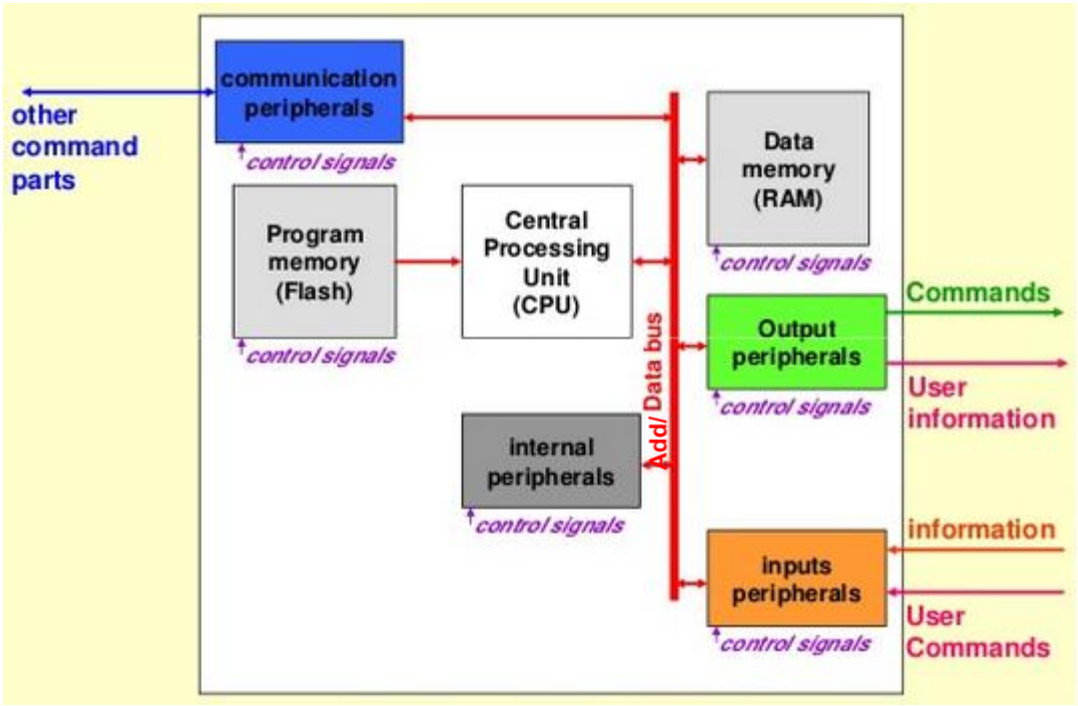
Features	ARM7TDMI-S	Cortex-M3
Architecture	ARMv4T (von Neumann)	ARMv7-M (Harvard)
ISA Support	Thumb / ARM	Thumb / Thumb-2
Pipeline	3-Stage	3-Stage + branch speculation
Interrupts	FIQ / IRQ	NMI + 1 to 240 Physical Interrupts
Interrupt Latency	24-42 Cycles	12 Cycles
Sleep Modes	None	Integrated
Memory Protection	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)*

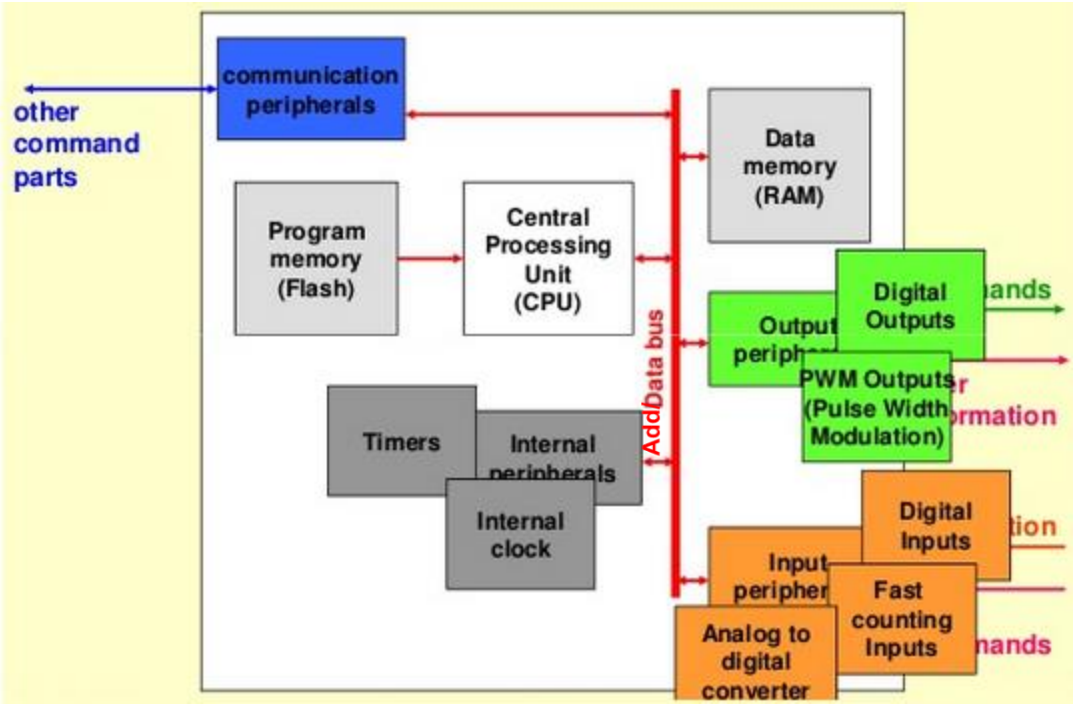
Microcontroller (MCU)

A microcontroller differs from a microprocessor in many ways

- A system where microprocessor along with memory, I/O, and other necessary functionalities are added for **control activities mainly –** all integrated onto a single chip
- Like a microprocessor, a microcontroller needs to be able to compute also, though not necessarily big numbers
- Now the microcontroller has taken over the role of the systems




other
command
parts
communication
peripherals
control signals
Program
memory
(Flash)
control signals
Central
Processing
Unit
(CPU)
Add/ Data bus
internal
peripherals
control signals
Data
memory
(RAM)
control signals
Output
peripherals
control signals
inputs
peripherals
control signals
Commands
User
information
information
User
Commands

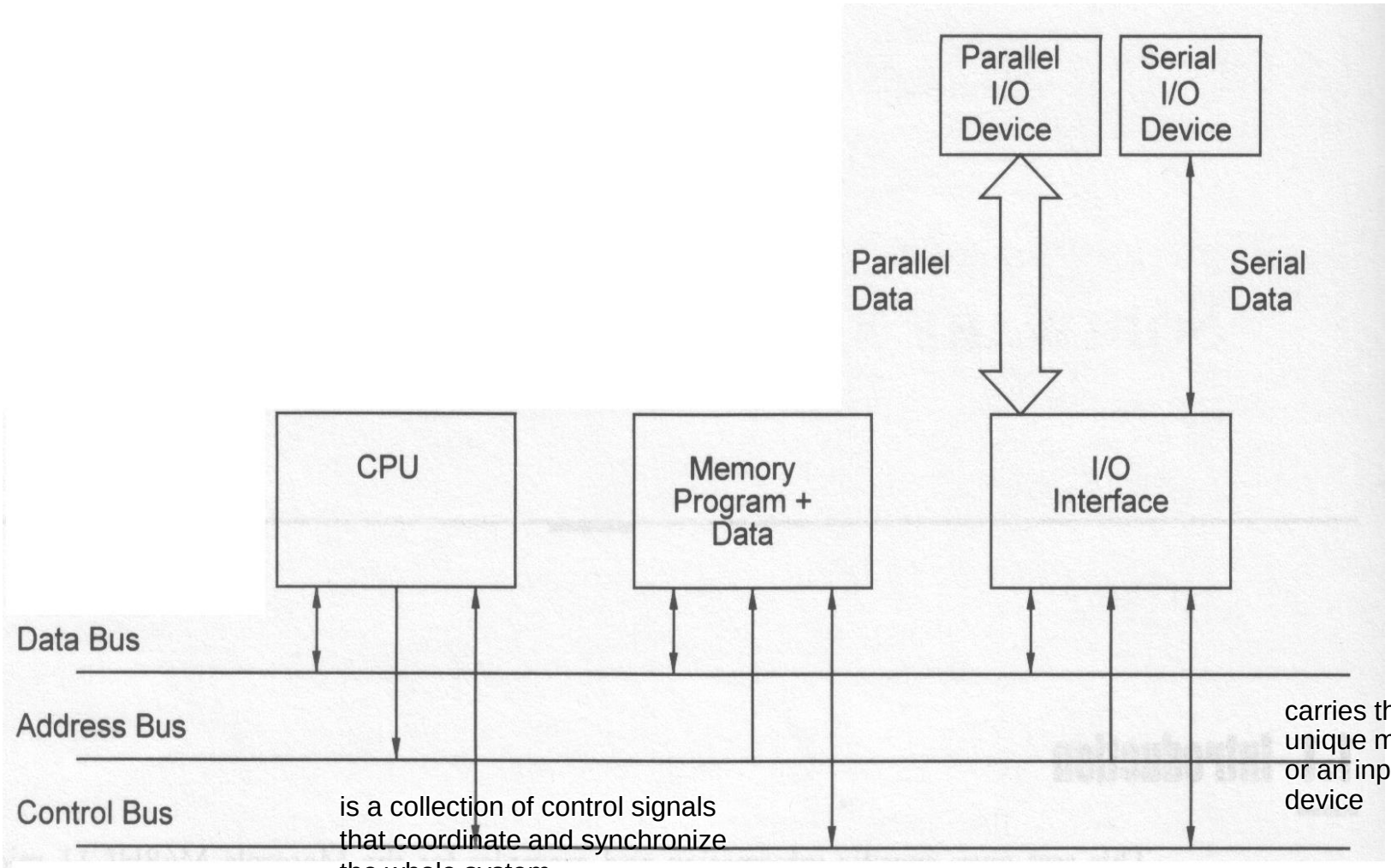

other
command
parts
communication
peripherals
Data
memory
(RAM)
Program
memory
(Flash)
Central
Processing
Unit
(CPU)
Digital
Outputs
PWM Outputs
(Pulse Width
Modulation)
Timers
Internal
peripherals
Internal
clock
Digital
Inputs
Fast
counting
Inputs
Analog to
digital
converter
Add
Data bus

Input/Output (I/O)

- Input Devices
  - Switches and Keypads
  - Provide binary information to the MPU
- Output devices
  - LEDs and LCDs
  - Receive binary information from the MPU

What's a "bus"?

A set of signal lines that carry digital information from source to destination



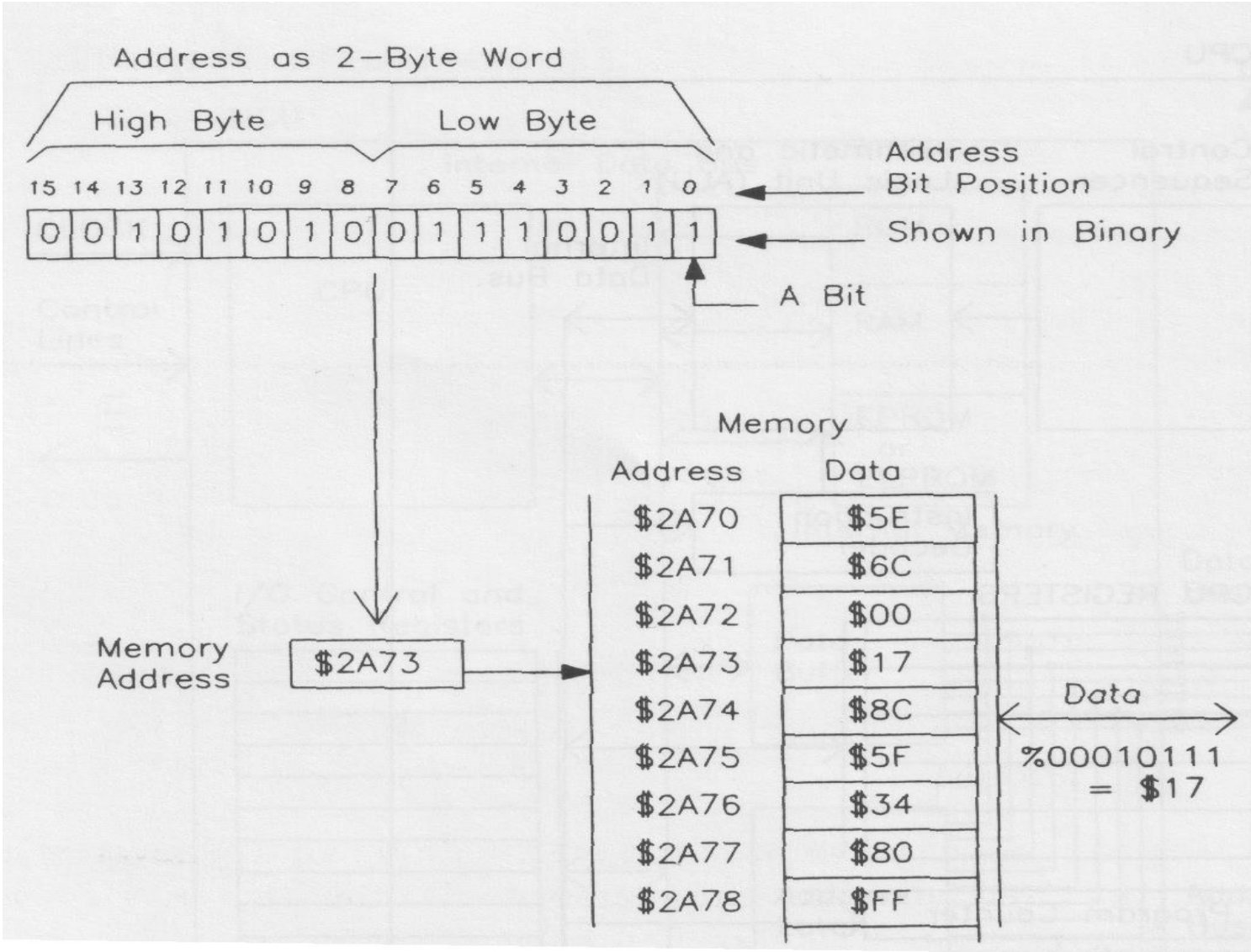
carries data to the CPU or from the CPU

carries the address of a unique memory location or an input/output (I/O) device

is a collection of control signals that coordinate and synchronize the whole system

Memory (1/3)

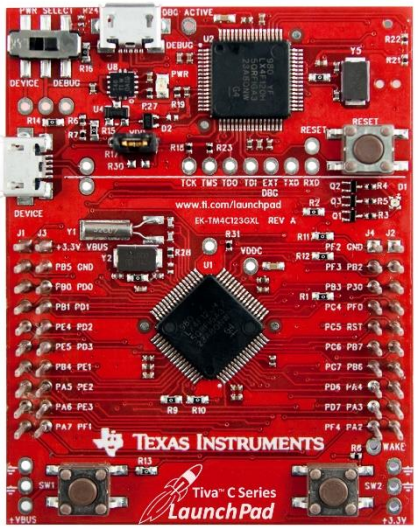
- Memory in a computer system
  - stores the data and instructions of the programs.
  - is organized as a number of locations each of which can hold the same size data value (usually a byte).
- Each location has a unique address.
  - a 16-bit address allows for addressing 65,536 (64K) memory locations.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

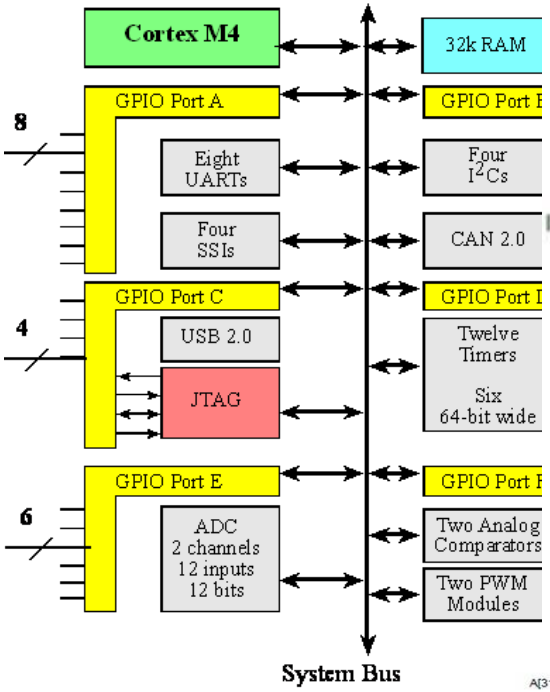

Address as 2-Byte Word
High Byte
Low Byte
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0 0 1 0 1 0 1 0 0 1 1 1 0 0 1 1
Address
Bit Position
Shown in Binary
A Bit
Memory
Address
Data
$2A70 $5E
$2A71 $6C
$2A72 $00
$2A73 $17
$2A74 $8C
$2A75 $5F
$2A76 $34
$2A77 $80
$2A78 $FF
Memory
Address
$2A73
Data
%00010111
= $17

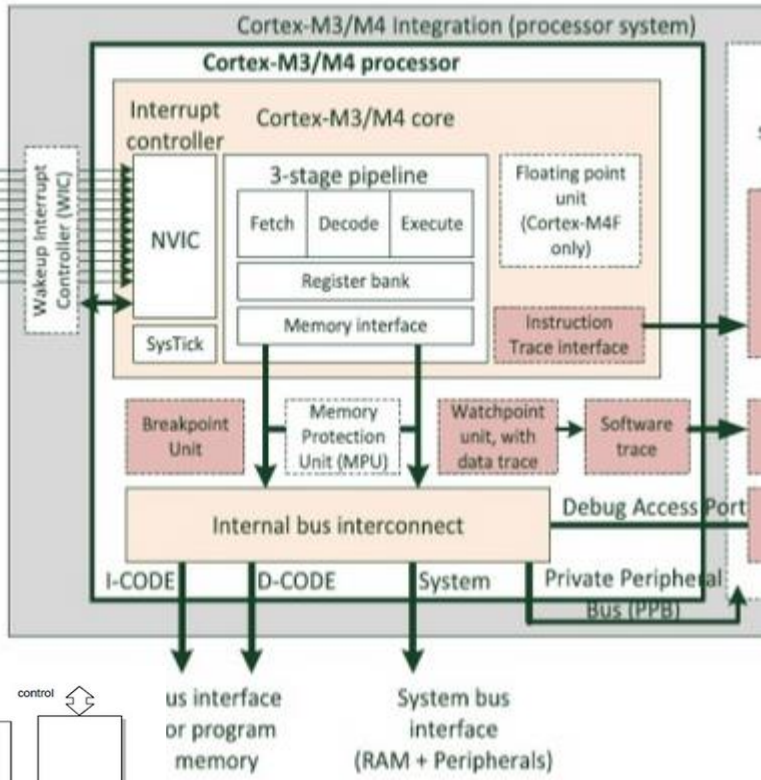
Texas instruments Tiva C Series TM4C123G board including ARM Cortex 4 microprocessor

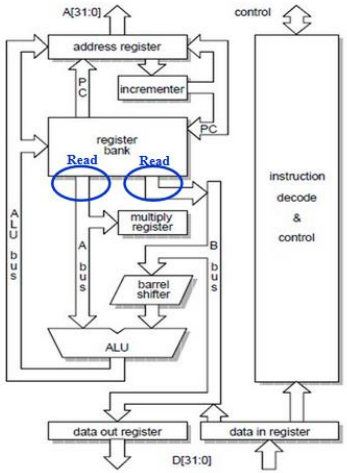
Feature	Description
Performance	
Core	ARM Cortex-M4F processor core
Performance	80-MHz operation; 100 DMIPS performance
Flash	256 KB single-cycle Flash memory
System SRAM	32 KB single-cycle SRAM
EEPROM	2KB of EEPROM
Internal ROM	Internal ROM loaded with TivaWare™ for C Series software
Security	
Communication 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

Texas instruments Tiva C Series TM4C123G board including ARM Cortex 4 microprocessor



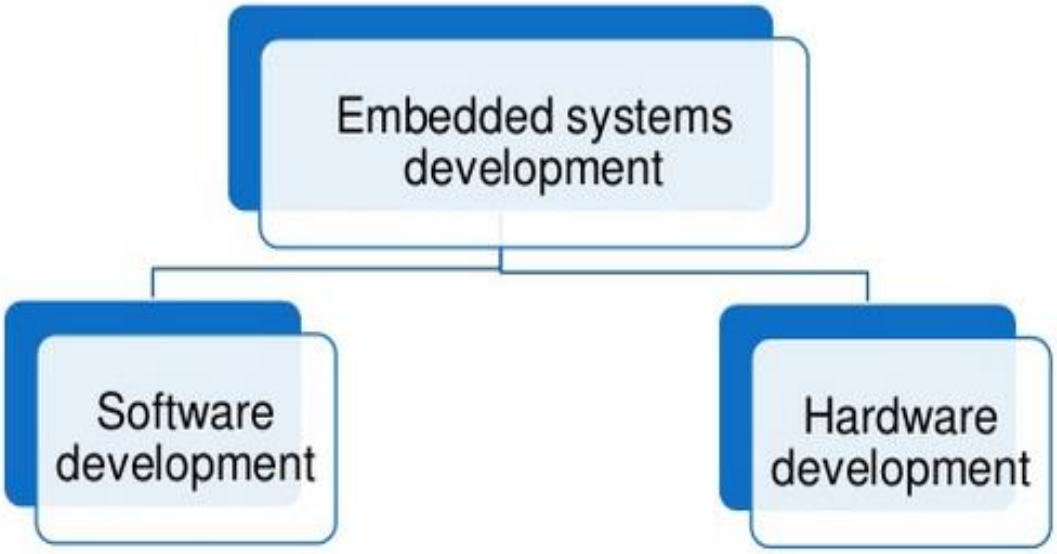




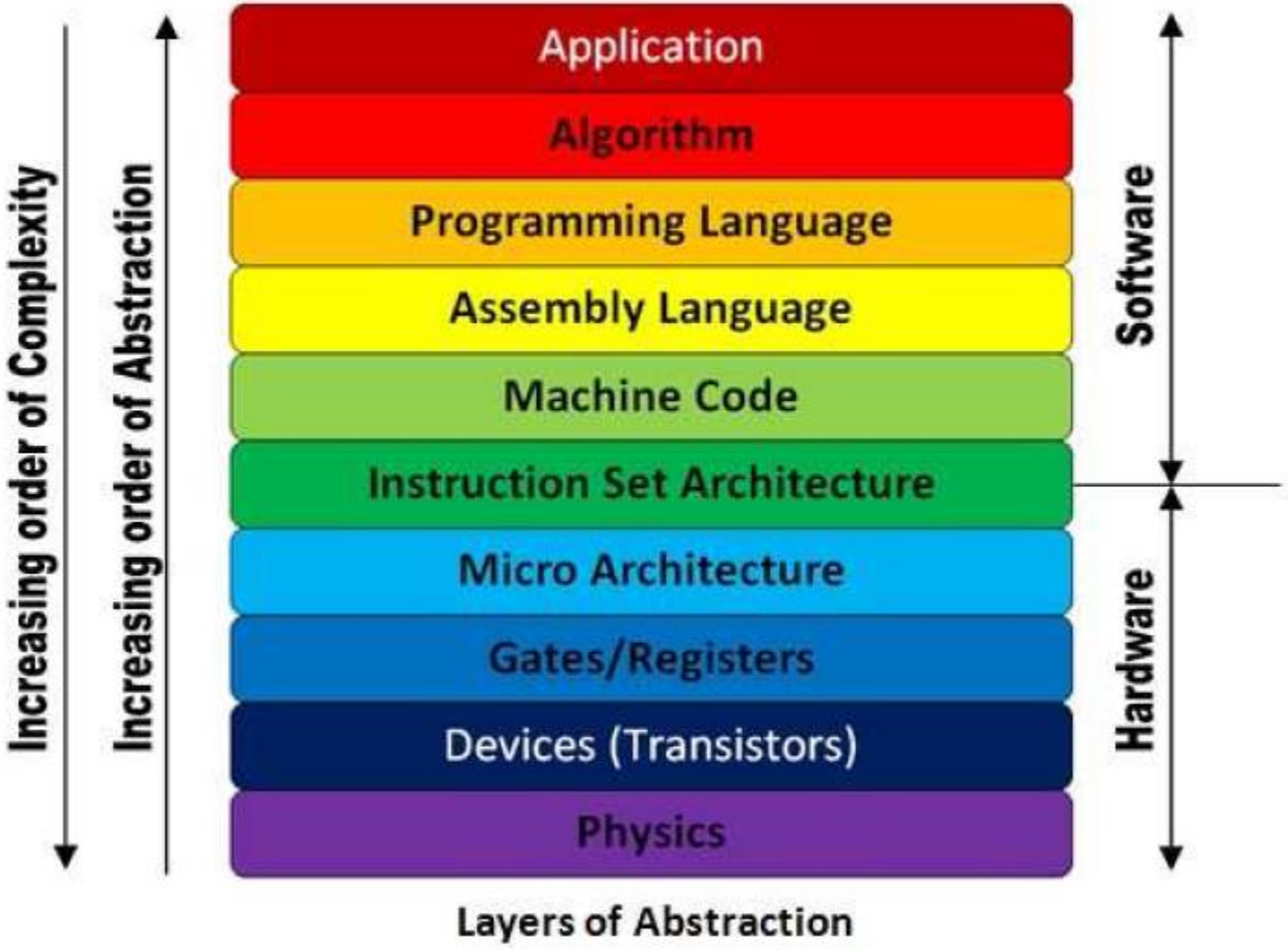


16MHz
internal clock

Developing an embedded application



Hardware/Software



Which microcontroller is the best Microchip, Motorola, Atmel or Intel?

- *The question can't be answered definitively, it is like asking which drink is best pepsi, sprite or ice-tea?*

- The best MCU depends on the specific task where it will be used.

- Selection criteria may
  - cost,
  - speed,
  - memory,
  - Number of available peripherals
  - size
  - power consumption, there are too many factors.

Processor efficiency

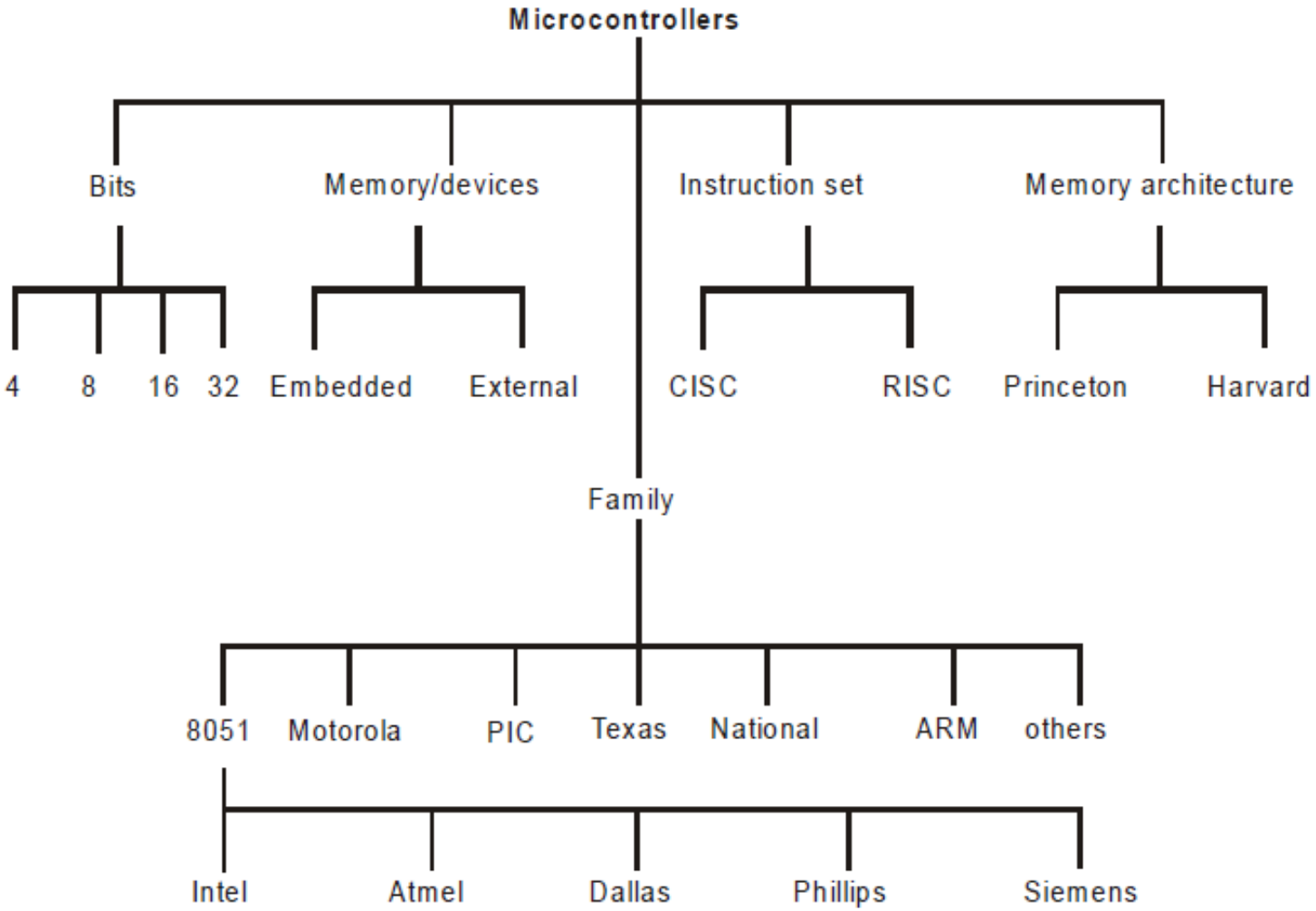
- MIPS (Million Instructions per Second)
  - Many reported IPS values have represented "peak" execution rates on artificial instruction sequences
  - synthetic benchmarks such as Dhrystone are now generally used
- CPI (cycles per instruction)

$$CPI = \frac{\Sigma_i (IC_i)(CC_i)}{IC}$$

Number of instructions of type i

Cycles for instruction type i

- MIPS/watt
  - The energy required to run a particular benchmark


Microcontrollers
Bits
Memory/devices
Instruction set
Memory architecture
4
8
16
32
Embedded
External
CISC
RISC
Princeton
Harvard
Family
8051
Motorola
PIC
Texas
National
ARM
others
Intel
Atmel
Dallas
Phillips
Siemens

Software

- Machine Instruction
  - A sequence of binary digits that can be executed by the processor – Hard to understand, program, and debug for human being.
  - E.g. in the ARM Cortex M4, instruction 0010 0010 0110 0100 (in binary, 0x2264 in hexadecimal) puts the number 100 in register #2.
- Assembly Language
  - Defined by machine instructions - An assembly instruction is a mnemonic representation of a machine instruction.
  - E.g. ADD represents a register add, ADD R2 R1 R0 means R1 and R0 should be added and the result should be saved to R2.
  - Assembly programs must be translated to binary before it can be executed: Assembler does this translation: ADD R2 R1 R0 -> 0001100 010 001 000 = 0x1888
  - In this course we'll be mostly dealing with Assembly Language.
- High-level Language
  - Syntax of a high-level language is similar to English.
  - A translator is required to translate the program written in a high-level language – this is done by a compiler.
  - High-level languages allow the user to work on the program logic at a more conceptual level (e.g. you don't have to worry about what's in register R2).
  - In this course we will see some C language examples, which are helpful if you understand C, but is not necessary to this course.

Levels of Program Code

C Program

```
int main(void){
  int i;
  int total = 0;
  for (i = 0; i < 10; i++) {
    total += i;
  }
  while(1); // Dead loop
}
```

Compile →

Assembly Program

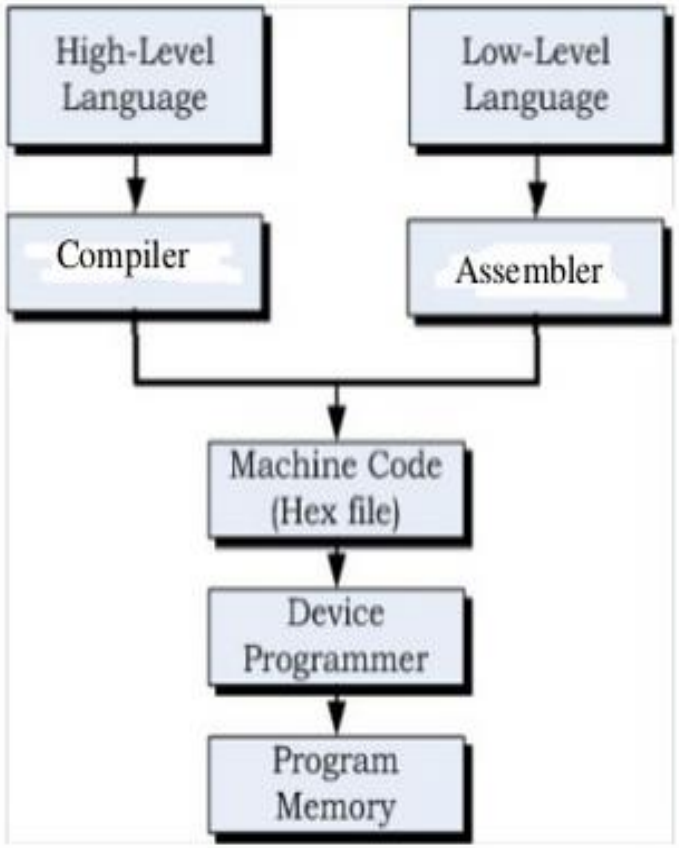
```
        MOVS r1, #0
        MOVS r0, #0
        B    check
loop    ADD  r1, r1, r0
        ADDS r0, r0, #1
check   CMP  r0, #10
        BLT  loop
self    B    self
```

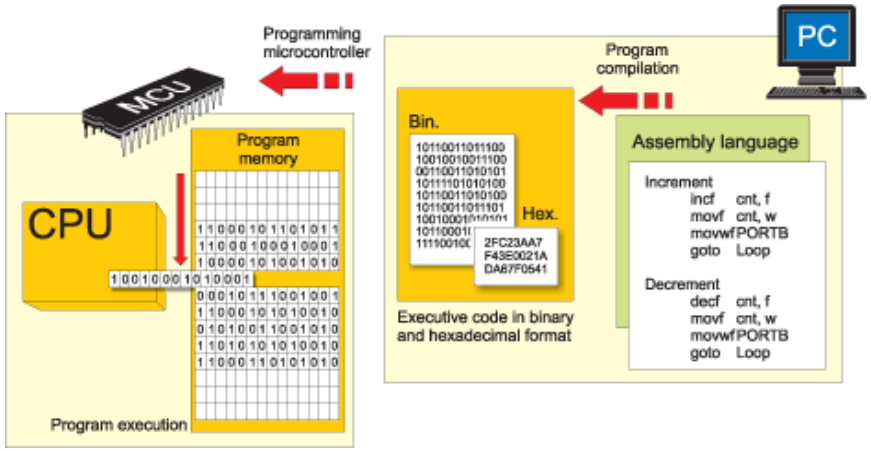
Assemble →

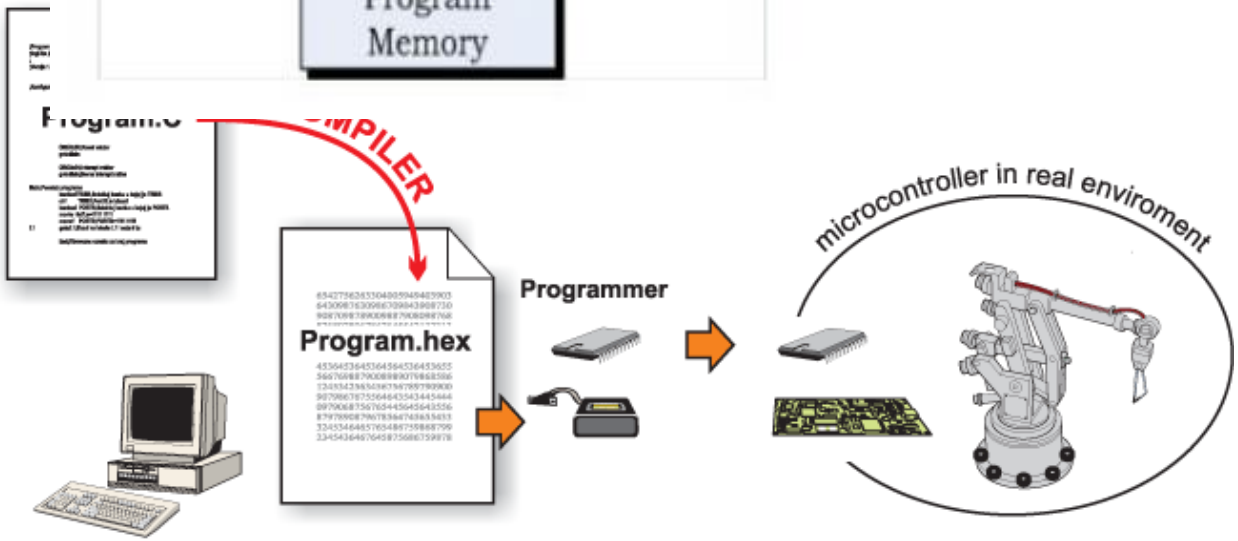
Machine Program

```
0010000100000000
0010000000000000
1110000000000001
0100010000000001
0001110001000000
0010100000001010
1101110011111011
1011111100000000
1110011111111110
```

- **High-level language**
  - Level of abstraction closer to problem domain
  - Provides for productivity and portability
- **Assembly language**
  - Textual representation of instructions
- **Hardware representation**
  - Binary digits (bits)
  - Encoded instructions and data


High-Level Language
Low-Level Language
Compiler
Assembler
Machine Code (Hex file)
Device Programmer
Program Memory


Programming microcontroller
MCU
Program memory
CPU
Program execution
Program compilation
PC
Bin.
Hex.
2FC23AA7
F43E0021A
DA87F0541
Executive code in binary and hexadecimal format
Assembly language
Increment
incf cnt, f
movf cnt, w
movwf PORTB
goto Loop
Decrement
decf cnt, f
movf cnt, w
movwf PORTB
goto Loop


Program.hex
Programmer
microcontroller in real enviroment

Abstraction vs reality

Abstraction is good, but don’t forget reality!

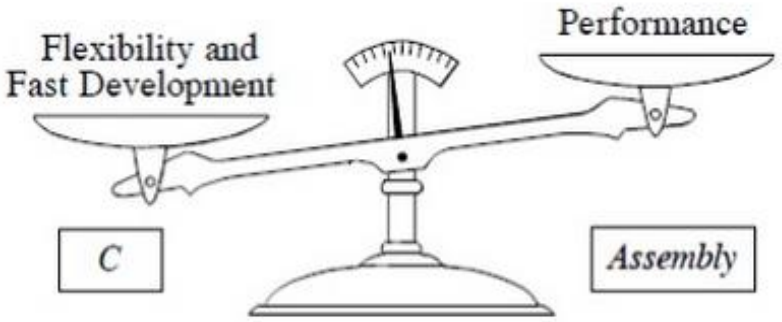
These abstractions have limits!

- Especially in the presence of bugs
- Need to understand underlying implementations
- Need to have a working understanding of architecture

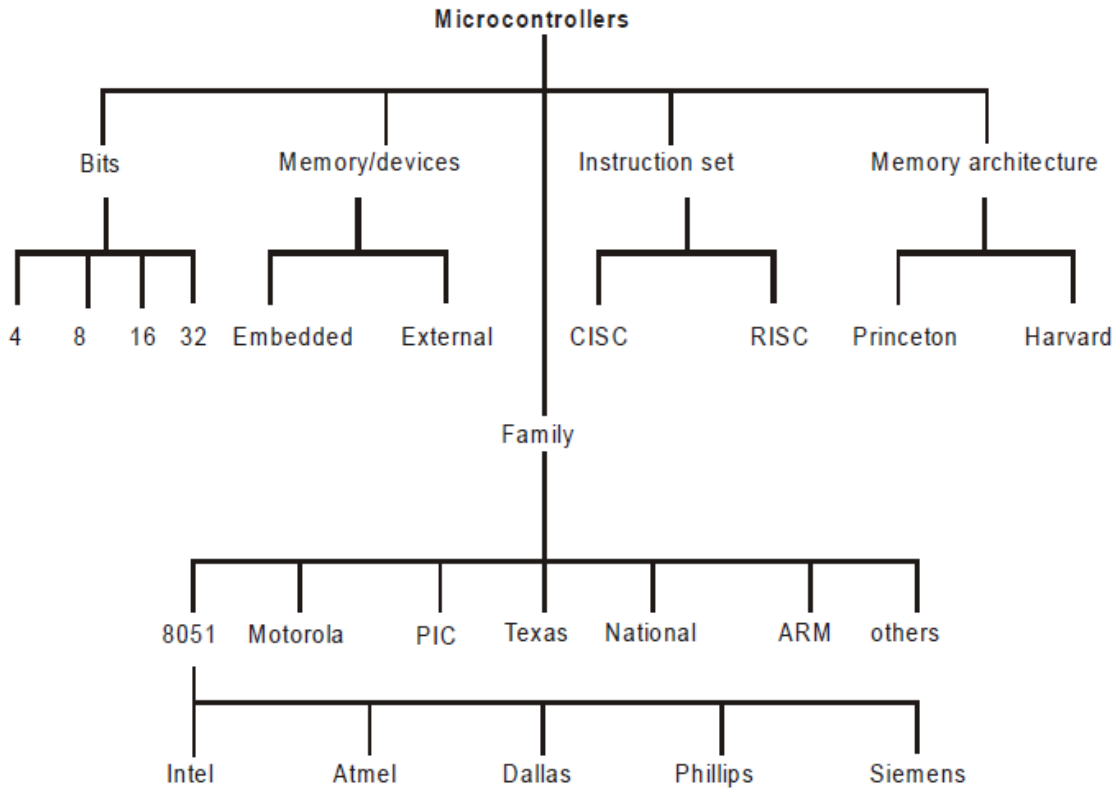
```c
void main() {
   printf("40000 * 40000 = %d\n", 40000 * 40000);
   printf("50000 * 50000 = %d\n", 50000 * 50000);
   printf("1e20 + (-1e20 + 3.14) = %f\n", 1e20 + (-1e20
       + 3.14));
   printf("(1e20 + -1e20) + 3.14 = %f\n", (1e20 + -1e20)
        + 3.14);
}
```

```
> gcc tester.c
> a.out
40000 * 40000 = 1600000000
50000 * 50000 = -1794967296
1e20 + (-1e20 + 3.14) = 0.000000
(1e20 + -1e20) + 3.14 = 3.140000
```

- Need to know architecture
- Understand assembly -> key to understand what really happens on the machine.
  - Behavior of programs in presence of bugs; high-level language model breaks down.
  - Tuning program performance and understanding sources of program inefficiency.
  - Implementing system software
    - Compiler has machine code as target
    - Operating systems must manage proc



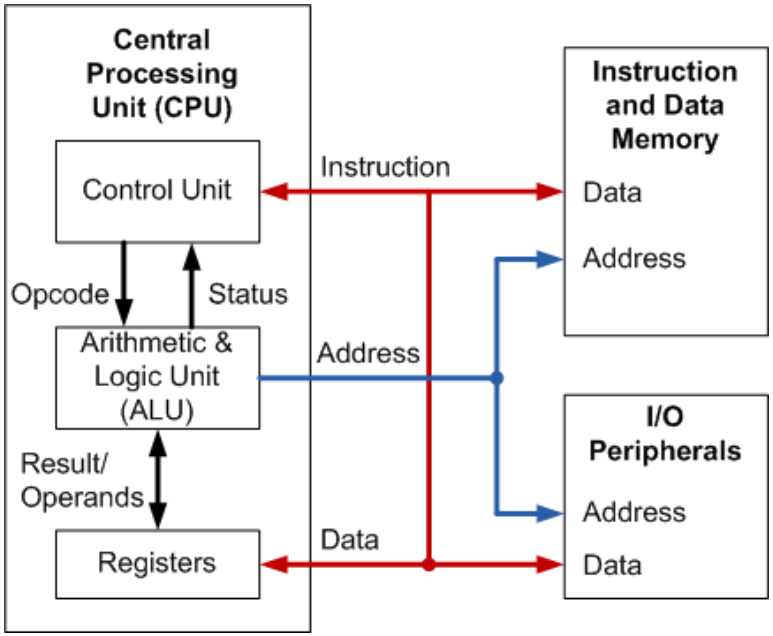
Back to Microcontollers



Computer Architectures

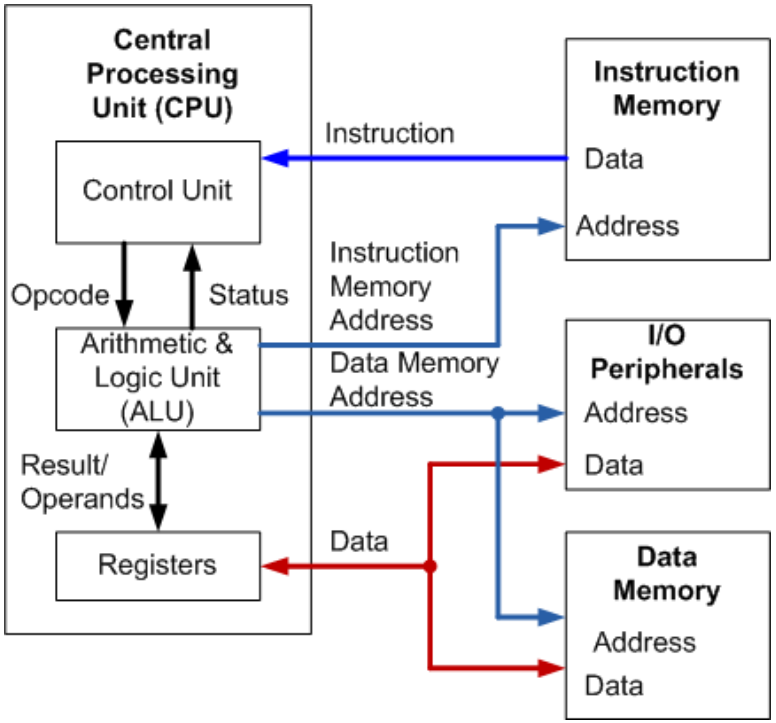
Von-Neumann/Princeton

Instructions and data are stored in the same memory.

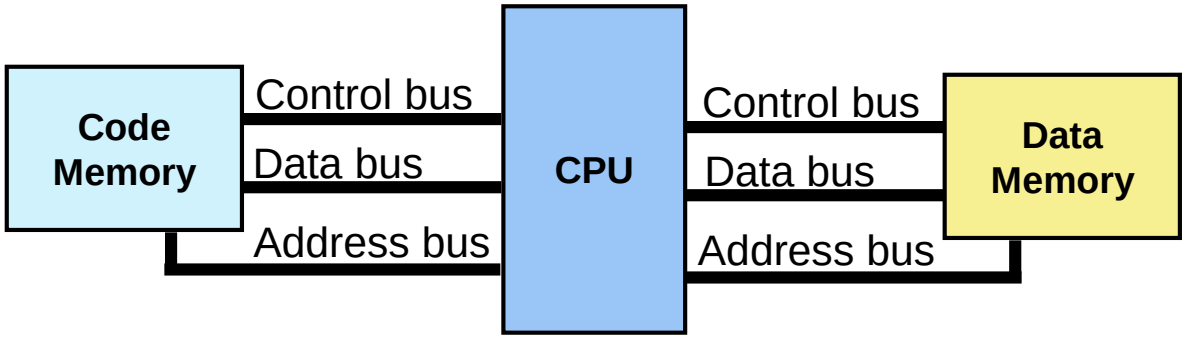


Harvard

Data and instructions are stored into separate memories.

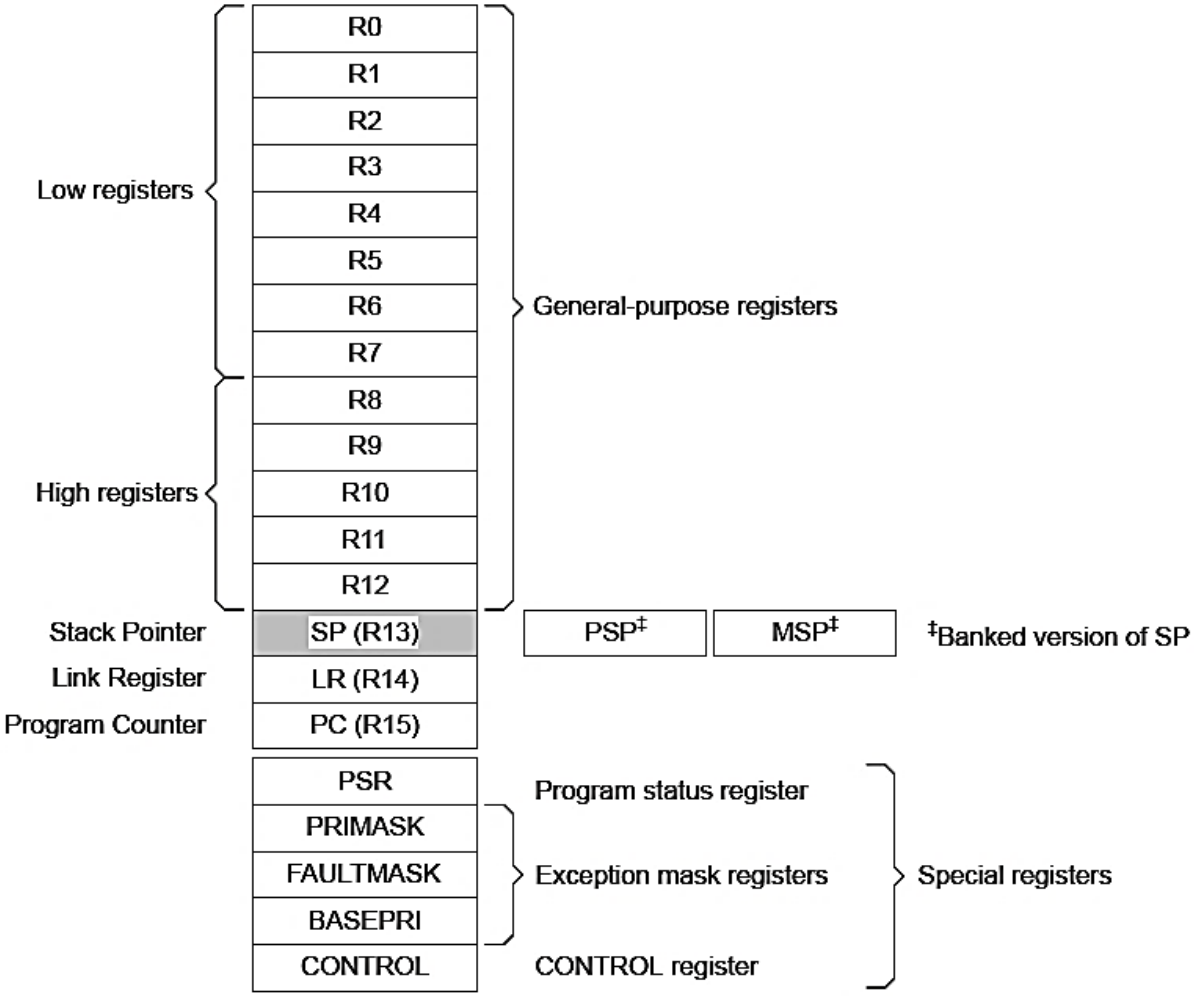


- Cortex M4 - Harvard architecture
  - Separated data bus and instruction bus
  - Advantage: opcodes and operands can go in and out of the CPU together.
  - Disadvantage: leads to more cost in general purpose computers



Registers

- The most fundamental storage area in the processor
  - is closely located to the processor
  - provides very fast access, operating at the same frequency as the processor clock
  - but is of 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 (CPSR in ARM)


R0
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
SP (R13)
LR (R14)
PC (R15)
PSR
PRIMASK
FAULTMASK
BASEPRI
CONTROL
Low registers
High registers
General-purpose registers
Stack Pointer
Link Register
Program Counter
PSP‡
MSP‡
‡Banked version of SP
Program status register
Exception mask registers
CONTROL register
Special registers

Instruction Set architecture

- An interface to allow easy communication between the programmer and the hardware. ISA prepares microprocessor to respond to all the user commands like execution of data, copying data, deleting it, editing it and several such and diverse operations.
- Two major items in ISA are:
  - Instruction Set: It is a group of instructions that can be given to the computer. These instructions direct the computer in terms of data manipulation. A typical instruction consists of two parts: Opcode and Operand.
    - Opcode or operational code is the instruction applied. It can be loading data, storing data etc.
    - Opcode encoding depends on the number of bits used
      - Example: For ARM, all instructions are of 32-bit length, but only 8 bits (bit 20 to 28) are used to encode the instruction. Hence a total of 28 = 256 different instructions are possible. (THUMB2-some 16 bit)
    - Operand is the memory register or data upon which instruction is applied.
    - ADD R3,R2,R1; R2+R1 -> R3
  - Addressing Modes: Addressing modes are the manner how the data is accessed. Depending upon the type of instruction applied, addressing modes are of various types such as direct mode where straight data is accessed or indirect mode where the location of the data is accessed.
    - ADD R3,R2,R1; R2+R1 -> R3
    - ADD R3,R3,#1; R3+ 1 -> R3

Instruction code format

- Opcode encoding depends on the number of bits used
  - Example: For ARM, all instructions are of 32-bit length, but only 8 bits (bit 20 to 28) are used to encode the instruction. Hence a total of 28 = 256 different instructions are possible. (THUMB2- some 16 bit)
- A typical instruction is encoded with a specific bit pattern that consists of the following:
  - an opcode field specifying the operation to be performed
  - an operand(s) identification (address) field that depends on the modes of addressing;
    - this provides the address of the register/memory location (s) that store the operand(s), or the operand itself.

Instruction Set architecture
RISC vs. CISC

- CISC (Complex Instruction Set Computer)
  - Put as many instruction as you can into the CPU
  - Have multi-word instructions and 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 and require arithmetic operands to be in registers

CISC

Features of the Complex Instruction Set Computing (CISC):

- many instructions
- complex instructions
  - each instruction can execute several low level operations
- complex addressing modes
  - smaller number of registers needed

A semantically rich instruction set is accommodated by allowing instructions of variable length

Adv/Disadv of CISC

ADV

- As each instruction can execute several low level operations,
  - the code size is reduced to save on memory requirements
  - less main memory access is required and hence processing time is reduced (faster)
- Backward code compatibility is maintained
  - can add new (and more powerful) instructions while retaining the ‘old’ instruction set for code compatibility (i.e. legacy programs can still run)
- Easy to program
  - complex instructions that fit well with high-level language expressions

DISADV

- A highly encoded instruction set needs to be decoded by hardwired microcode electronic circuitry
  - more complex hardware design
  - slower instruction decoding/execution
- Variable length instructions
  - different execution time among instructions
  - affects pipelined operations

RISC

- RISC instructions each occupy a single word
- A load/store architecture is used, meaning:
  - only Load and Store instructions are used to access memory operands
  - operands for arithmetic/logic instructions must be in registers, or one of them may be given explicitly in instruction word
  - Instructions/data are stored in the memory
  - Because RISC requires register operands, data transfers are required before arithmetic
- Large number of registers

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:

  $A \leftarrow [A] + [B]$

- Steps: fetch contents of locations A and B, compute sum, and transfer result to location C
- Sequence of simple RISC instructions for task:

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

- In CISC it can be just Add A,B

Adv/Disadv of RISC

ADV

- Simpler instructions
  - one clock per instruction gives faster execution than on a CISC processor with the same clock speed
- Simpler addressing mode
  - faster decoding
- Fixed length instructions
  - faster decoding and better pipeline performance
- Simpler hardware
  - less silicon area
  - less power consumption

M4 adds FPU and DSP-like instructions

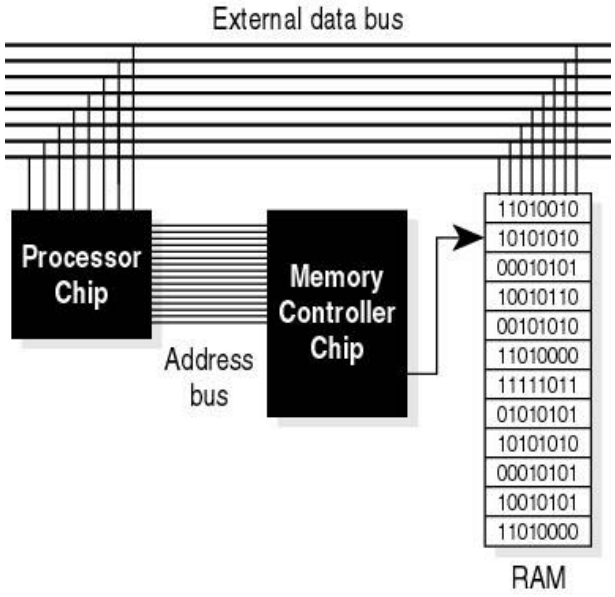
DISADV

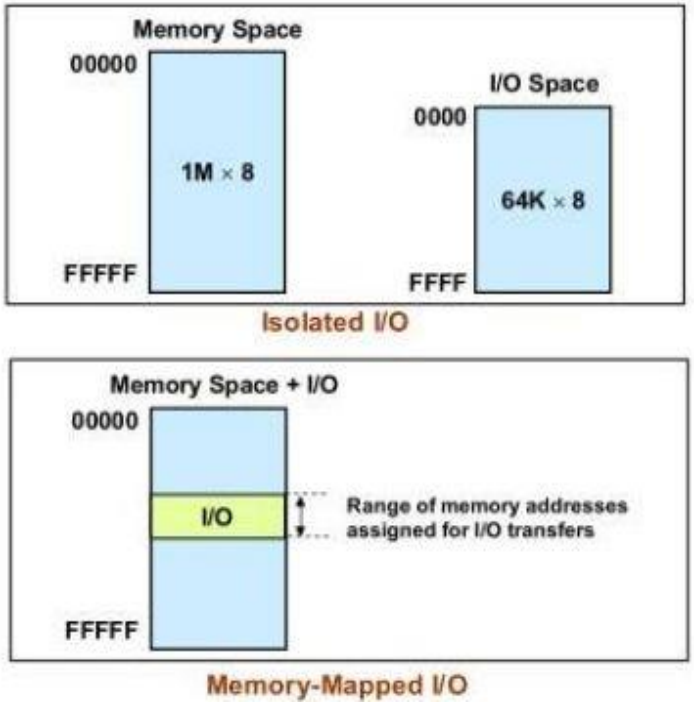
- Fewer instructions than CISC
  - as compared to CISC, RISC needs more instructions to execute one task
  - code density is less
  - needs more memory
- No complex instructions
  - no hardware support for division or floating-point arithmetic operations
  - needs a more complex compiler and longer compiling time

To reduce memory requirements and cost ARM provides the 16-bit Thumb instruction set and 16/32-bit Thumb2 instruction set as an option for its RISC processor cores

Address Space/data organization

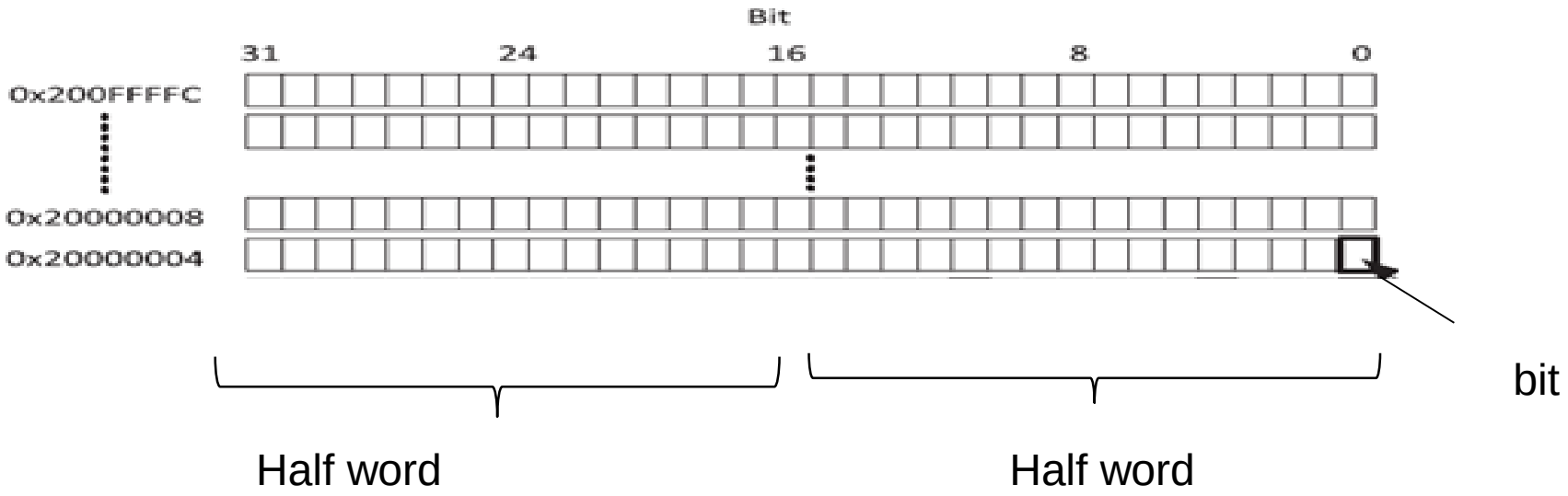
- A memory location can be used to store data and instructions. A memory location has two components: an address and its contents.
- Data transfers between the CPU and the memory are done over the common buses: address bus and data bus.
- The address space of a processor depends on its address decoding mechanism: Its size will depend on the number of address bits used.
  - Depending on the processor design, there may be two types of address space (Isolated I/O -Intel)
    - one is used by normal memory access
    - another one is reserved for I/O peripheral registers (control, status, and data)
    - need extra control signal or special means of accessing the alternate address space
  - Some processor families utilize only one address space for both memory and I/O devices (Memory mapped I/O - ARM, Motorola)
- Data Organization in Memory: Memory contains storage locations that can store data of a certain fixed size. Each location is provided with a unique address. Depending on the data path size of the processor, the memory content (8-bit) is accessible in the size of an 8-bit byte, a 16-bit half word, a 32-bit word, and even a 64-bit double word. Order of the four bytes of data depends on the Endianness adopted.

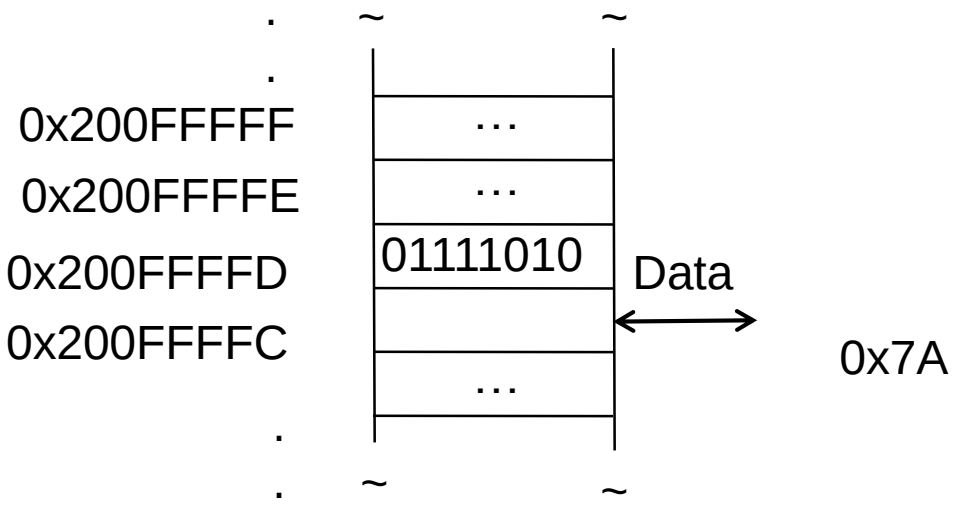




Cortex-M4 processor has 32 bits for addressing and this means that it is supporting 4GB of memory space. The memory space is used by the program code, data, peripherals, and some of the debug support components inside the processors.

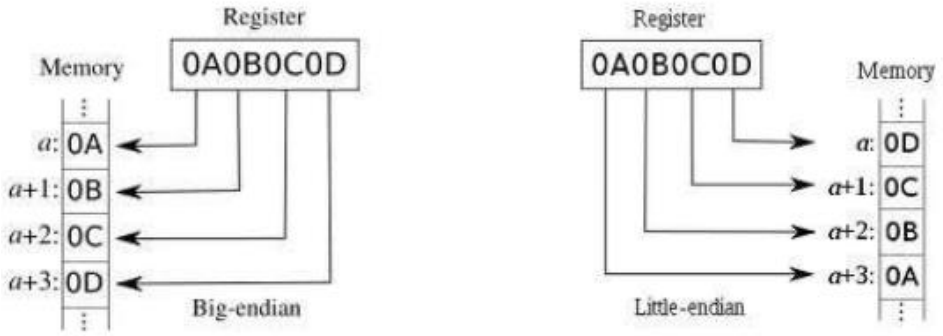
Memory, I/O Access



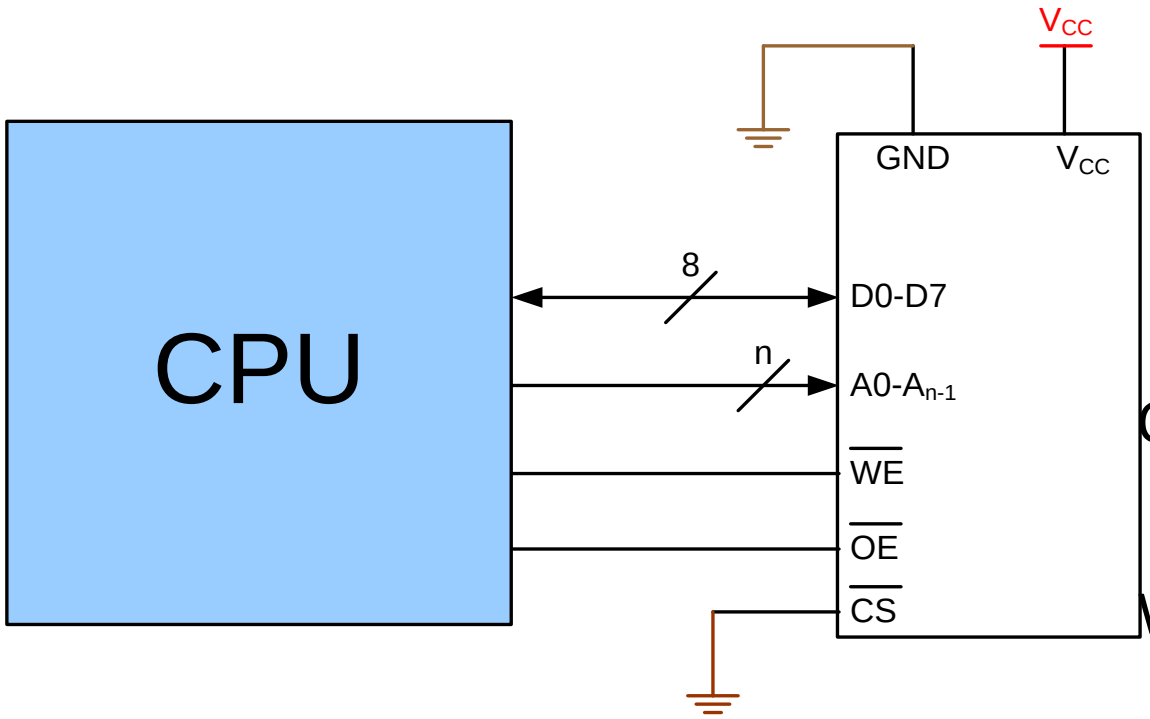


Endianness

- In the Little Endian format, the least significant byte (LSB) is stored in the lowest address of the memory, with the most significant byte (MSB) stored in the highest address location of the memory. (ARM is Little Endian by default)
- In the Big Endian format, the least significant byte (LSB) is stored in the highest address of the memory, with the most significant byte (MSB) stored in the lowest address location of the memory.



Interfacing Memory to CPU



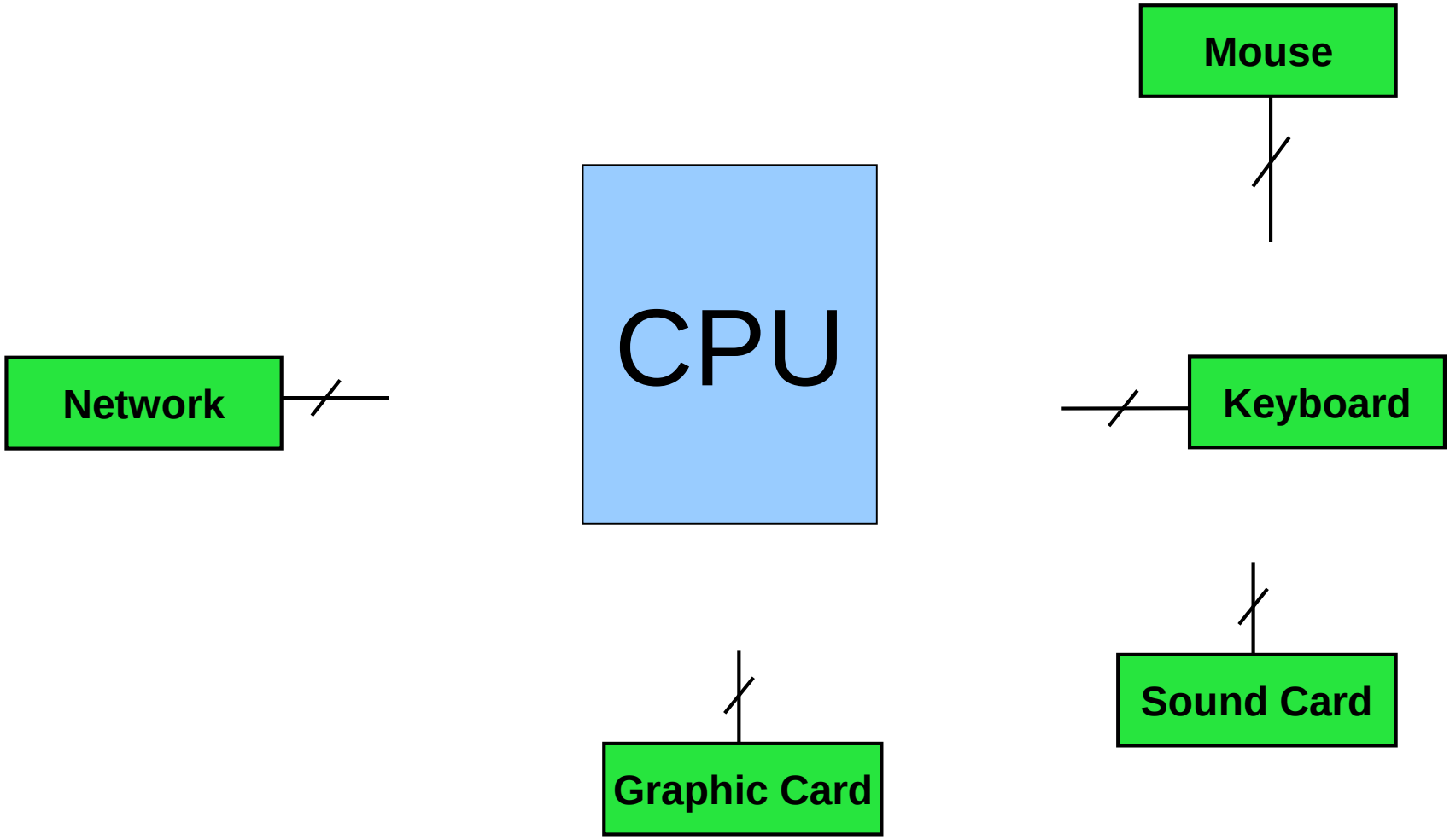
CS': Active low Chip Select

WE': Active low Write Enable

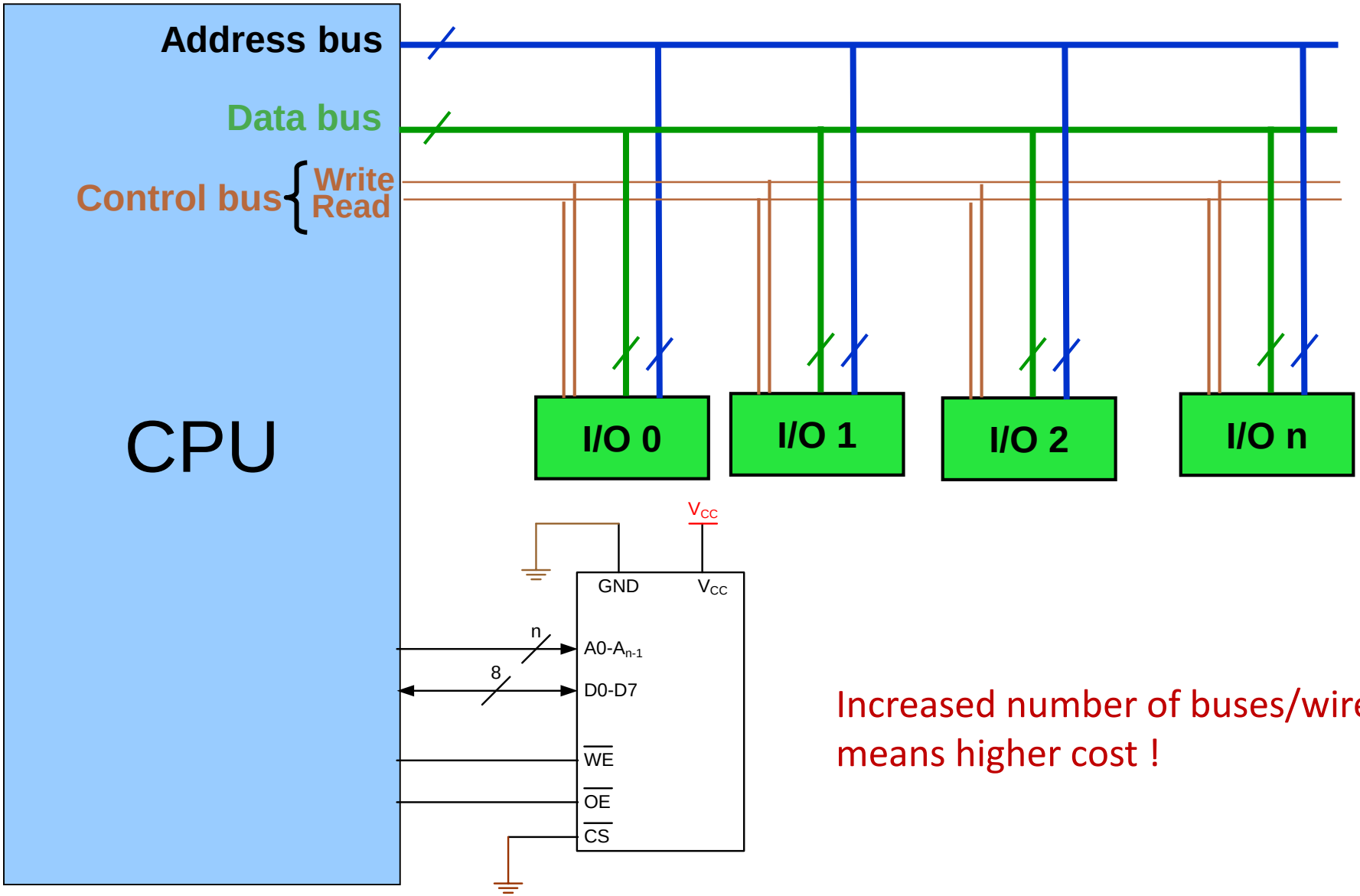
OE': Active low Output Enable

Connecting I/O devices to CPU

- CPU should have lots of pins!

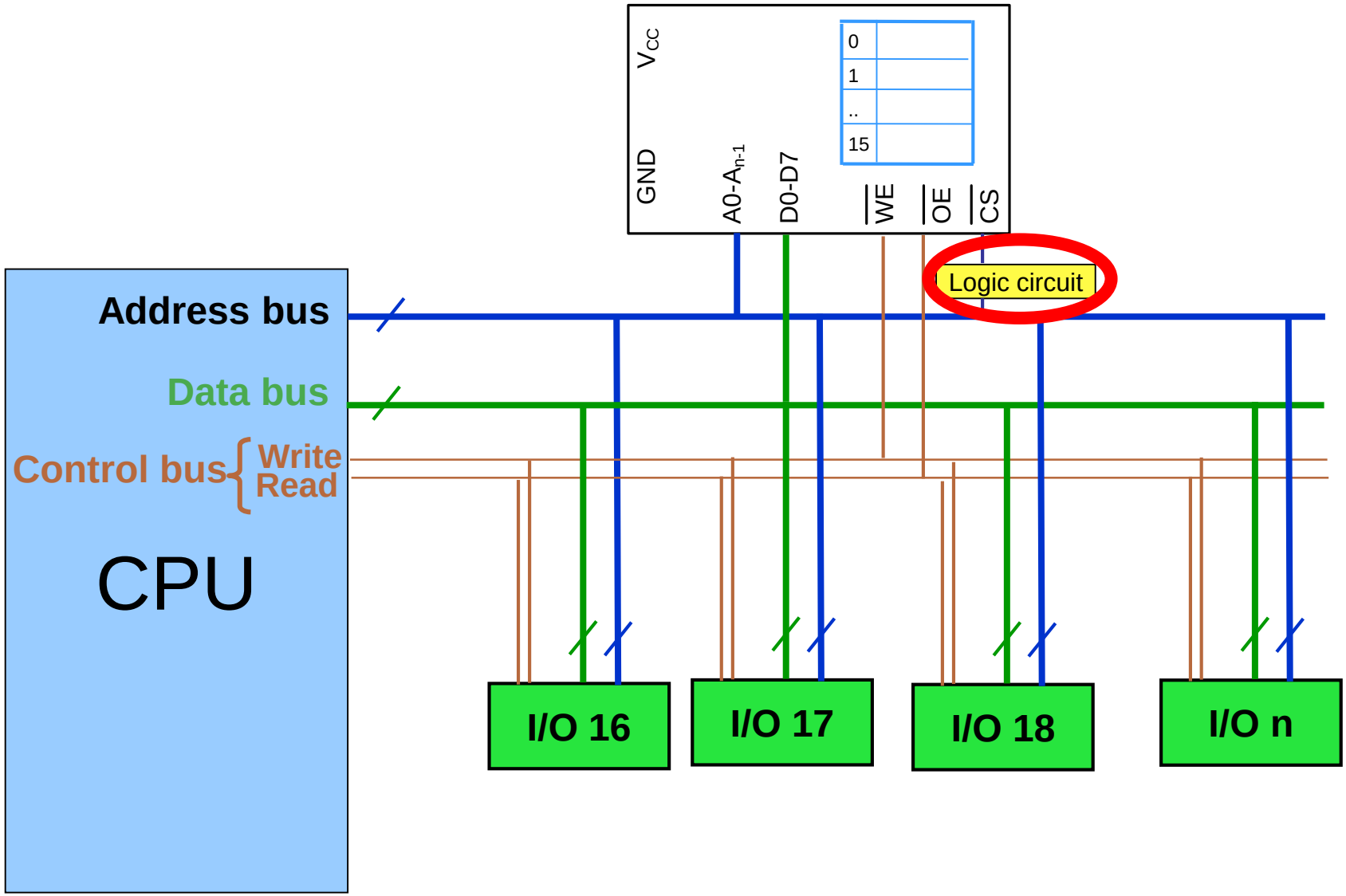


Connecting I/Os and Memory to CPU (Isolated I/0)



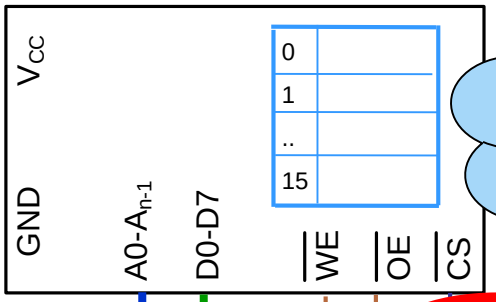
Increased number of buses/wires means higher cost !

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



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

How to design the logic circuit?



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

Logic circuit

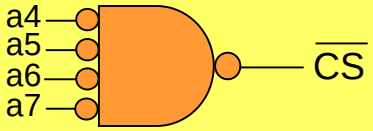
Address bus 8

Co

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



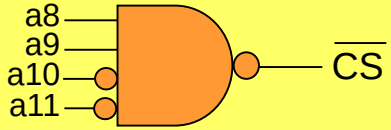
Address Decoding

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

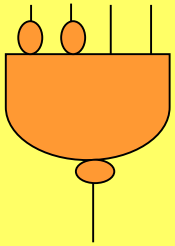
Solution

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

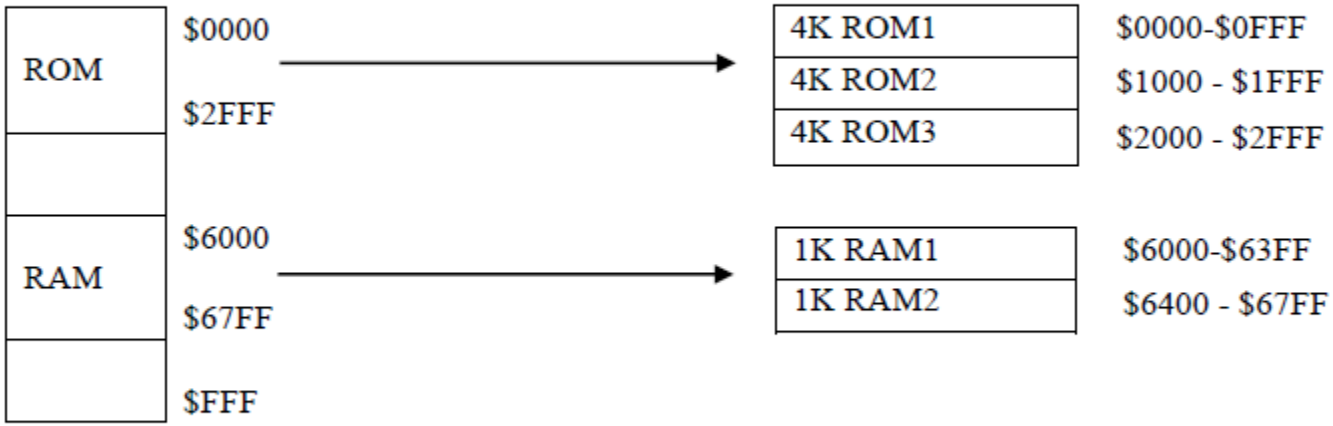
	a11	a10	a9	a8	a7	a6	a5	a4	a3	a2	a1	a0
From address 300H ➔	0	0	1	1	0	0	0	0	0	0	0	0
To address 3FFH ➔	0	0	1	1	1	1	1	1	1	1	1	1



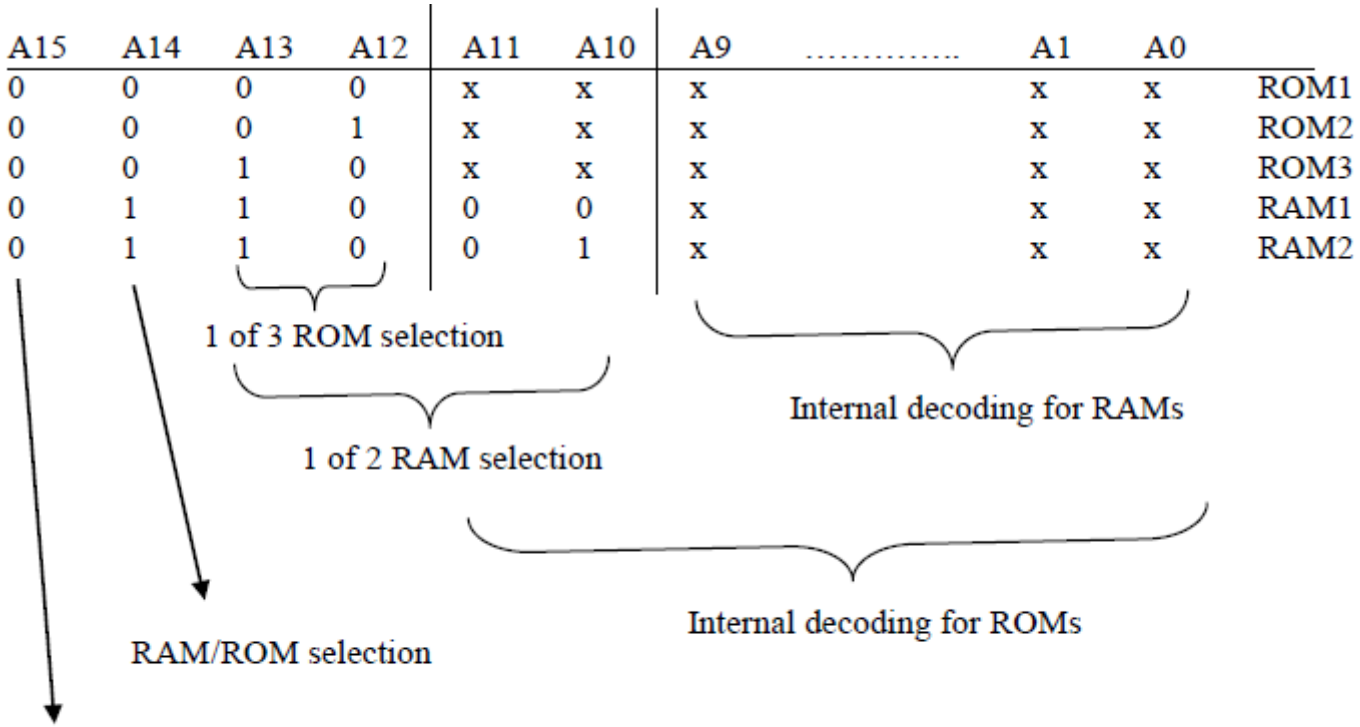
An easy way of designing



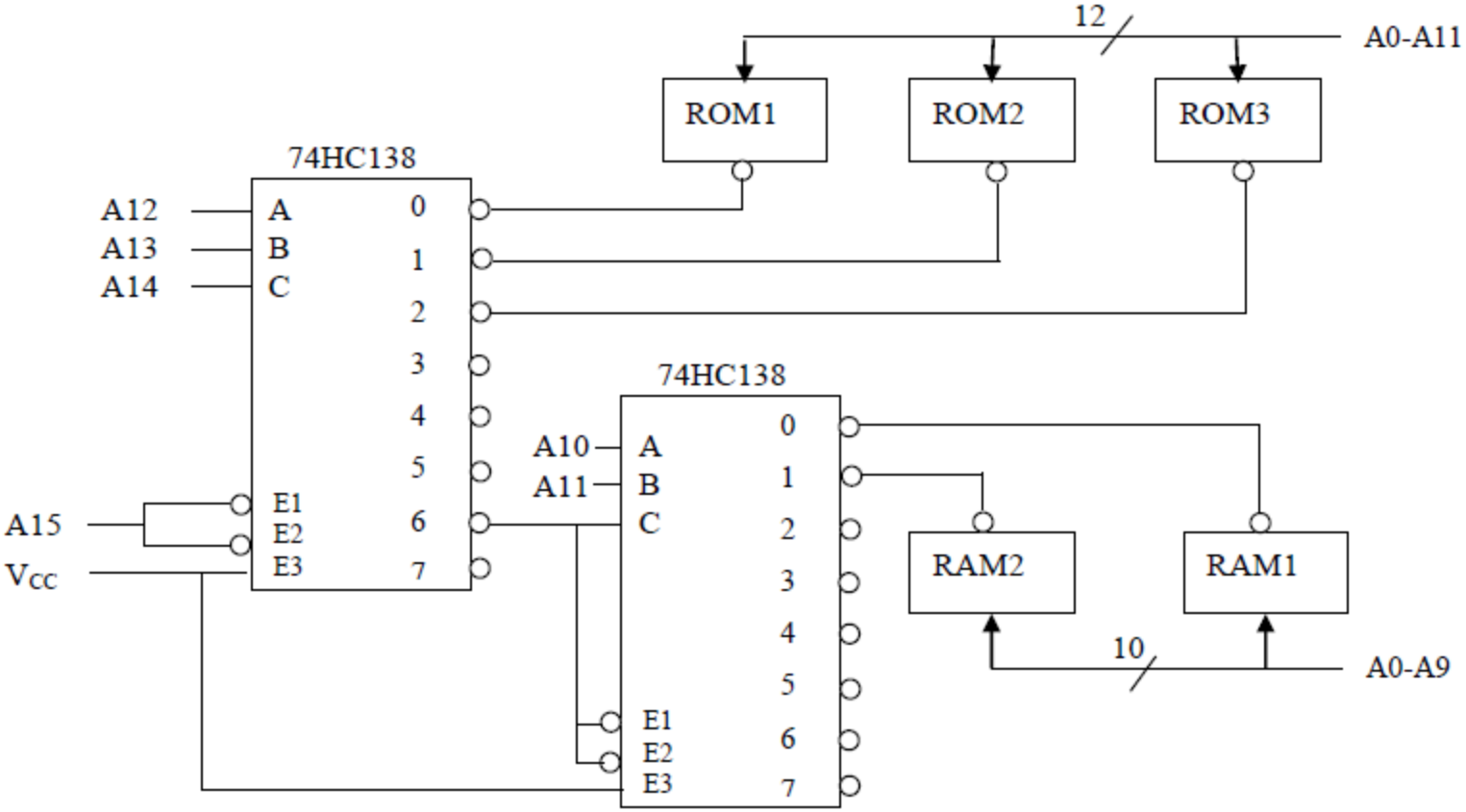
Example: The following memory map is given and it is required to be implemented using three 4K ROM and two 1K RAM chips available.



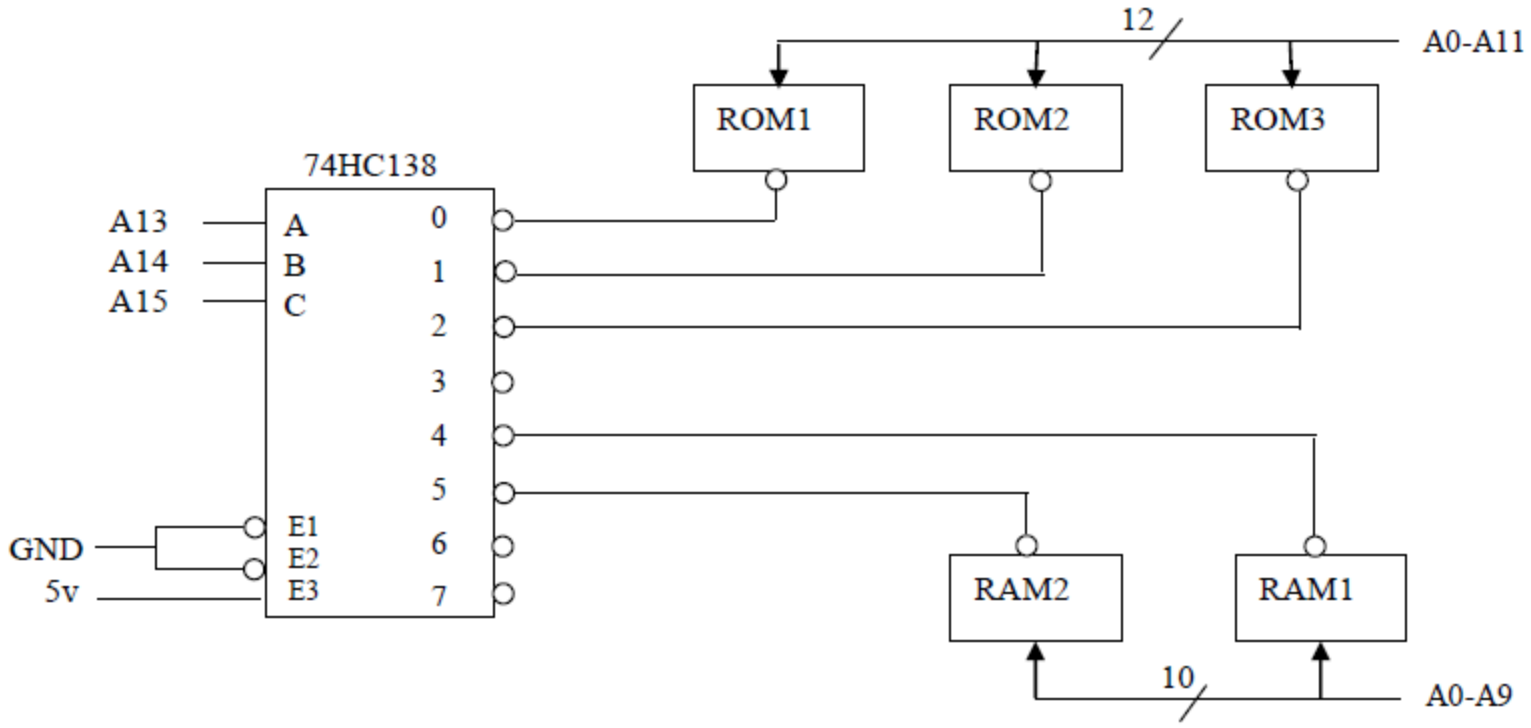
For RAM internal addressing we need 10 bits (A0-A9). A10-A15 are left for RAM selection.
For ROM internal addressing we need 12 bits (A0-A11). A12-A15 are left for ROM selection.


A15 A14 A13 A12 A11 A10 A9 ..... A1 A0
0 0 0 0 x x x x x ROM1
0 0 0 1 x x x x x ROM2
0 0 1 0 x x x x x ROM3
0 1 1 0 0 0 x x x RAM1
0 1 1 0 0 1 x x x RAM2
1 of 3 ROM selection
1 of 2 RAM selection
Internal decoding for RAMs
Internal decoding for ROMs
RAM/ROM selection
May be used for active enable input if a PROM decoder is used.

Using 74HC138 decoders

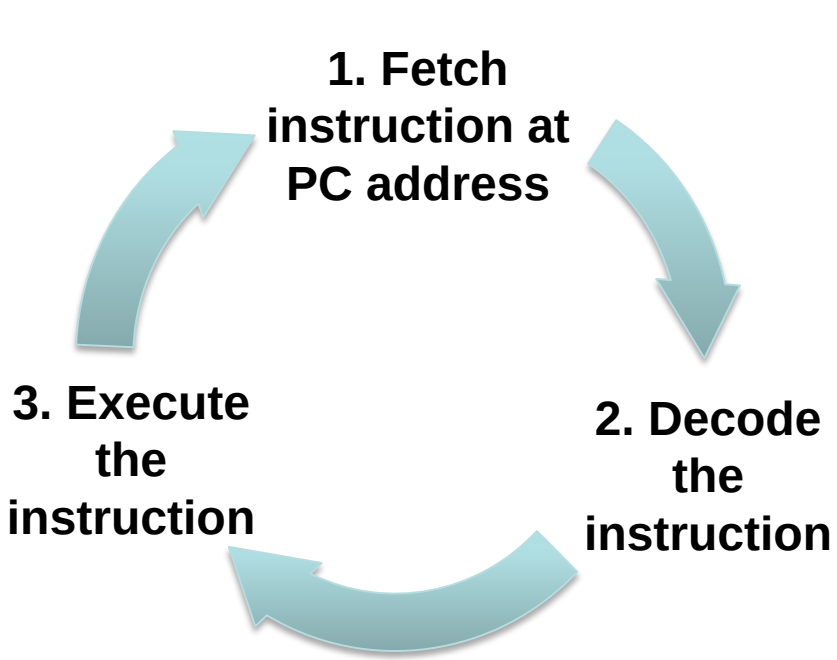


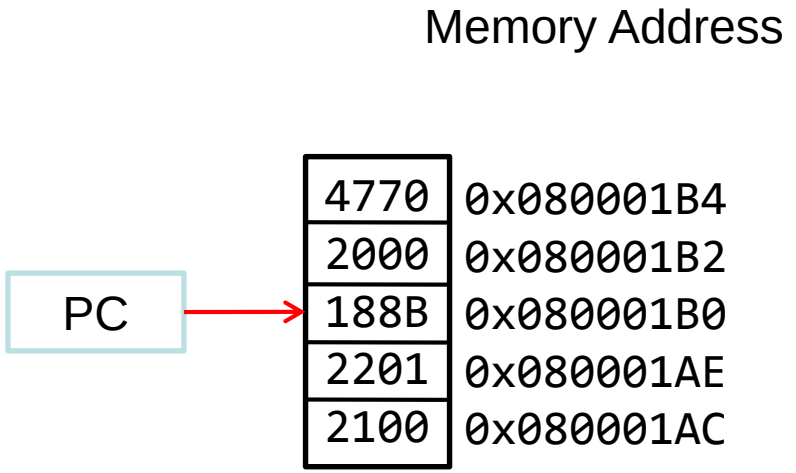
Memory overlaying



Program Execution

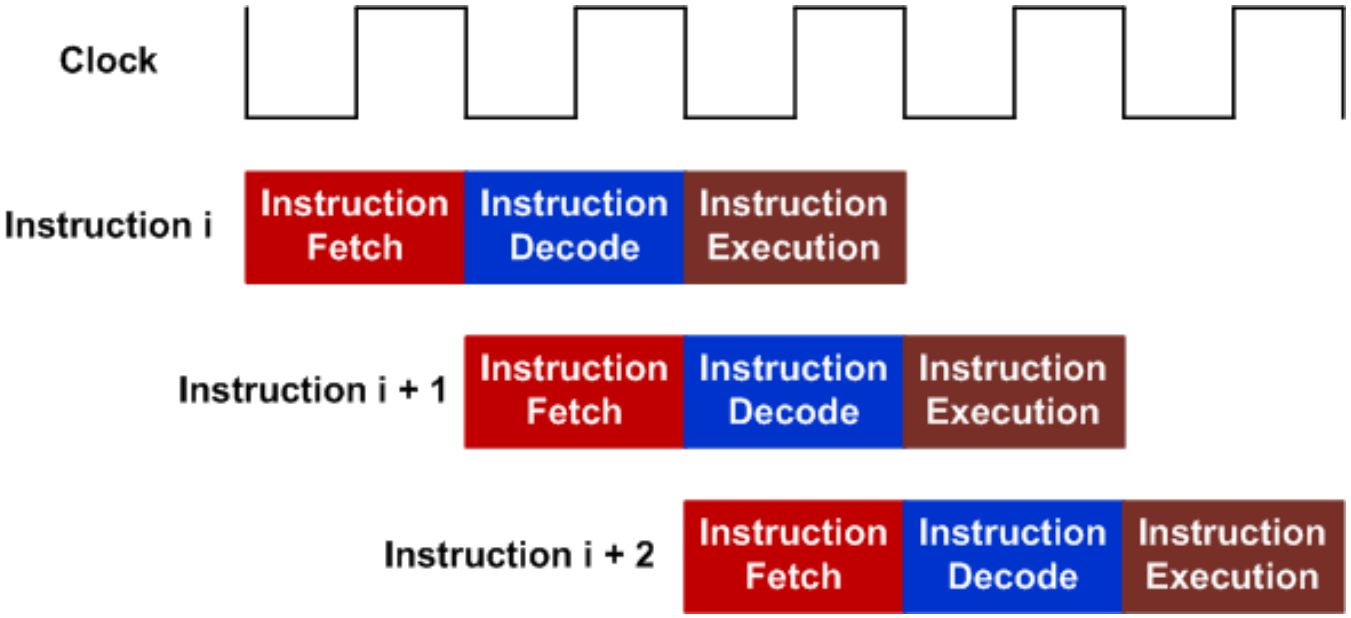
- **Program Counter (PC)** is a register that holds the memory address of the next instruction to be fetched from the memory.



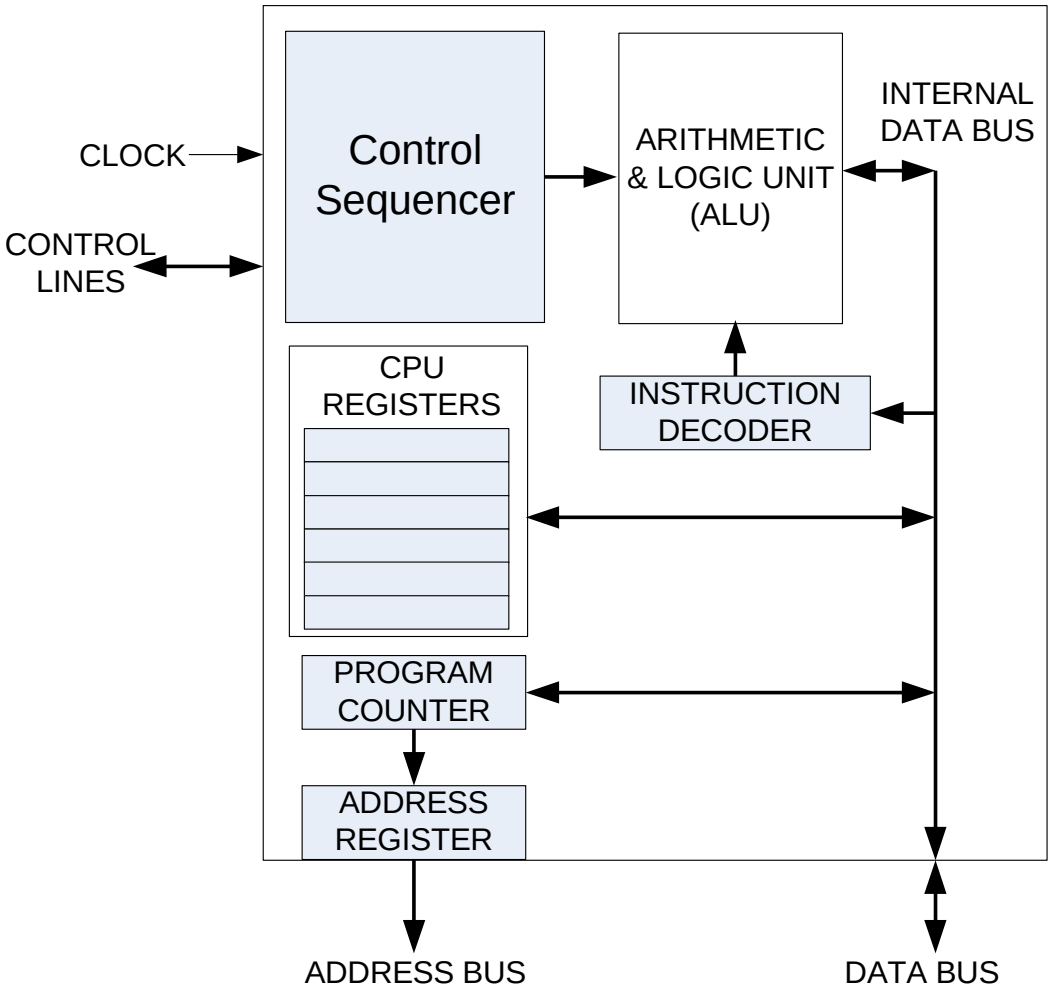


Three-state pipeline: Fetch, Decode, Execution

- **Pipelining** allows hardware resources to be fully utilized.



Instruction Execution



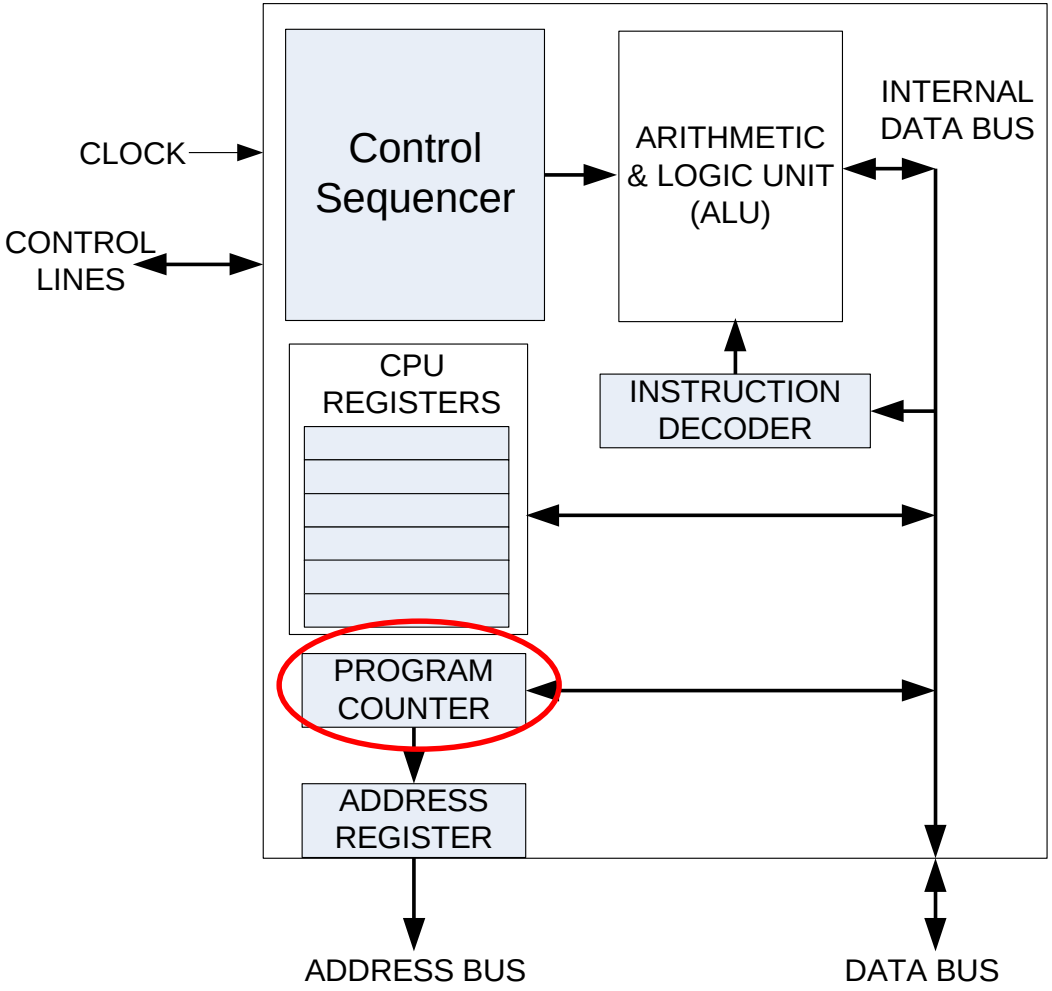
The main functions of the CPU are:

- Data transfer
- Arithmetic and logic operations
- Decision making (instruction flow control)

<u>Control Sequencer:</u>
Controls the operations in the CPU

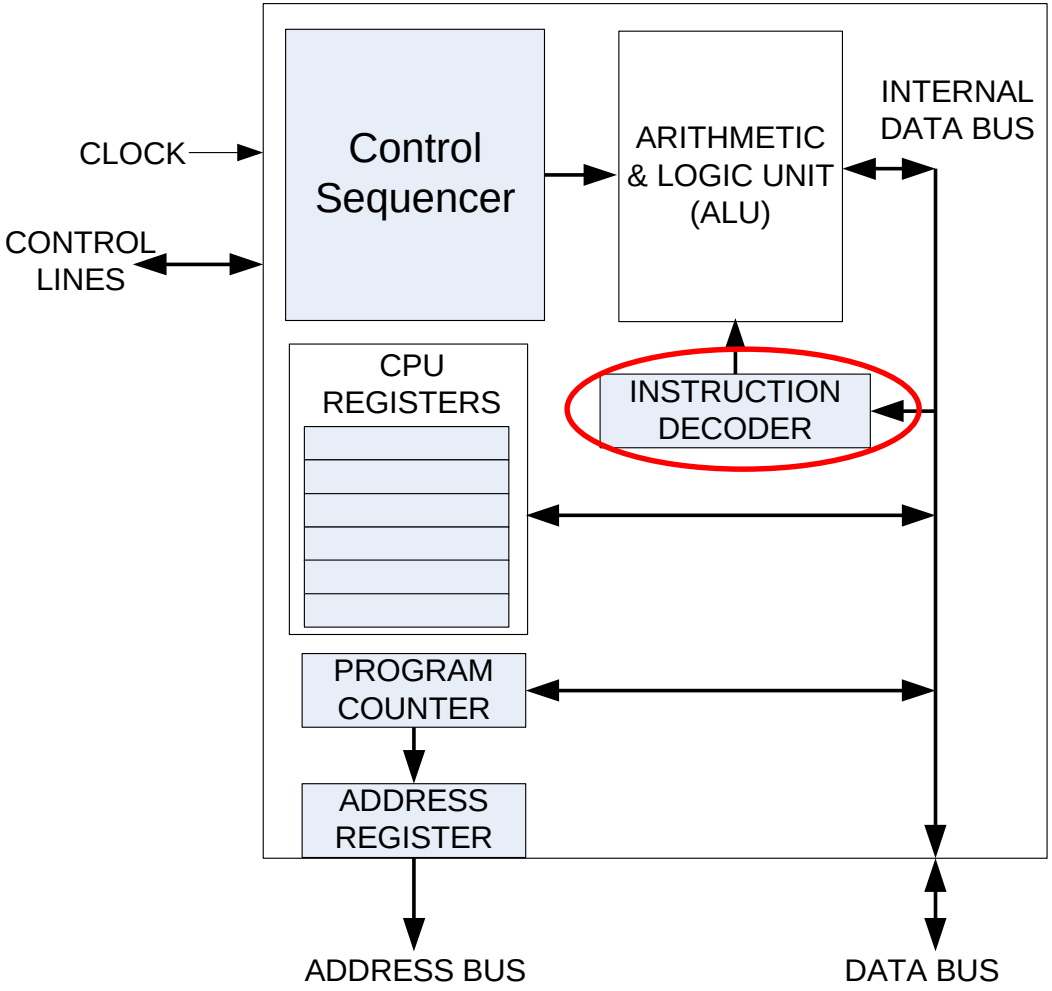
<u>Register Array:</u>
Temporary storage (faster than memory)

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).

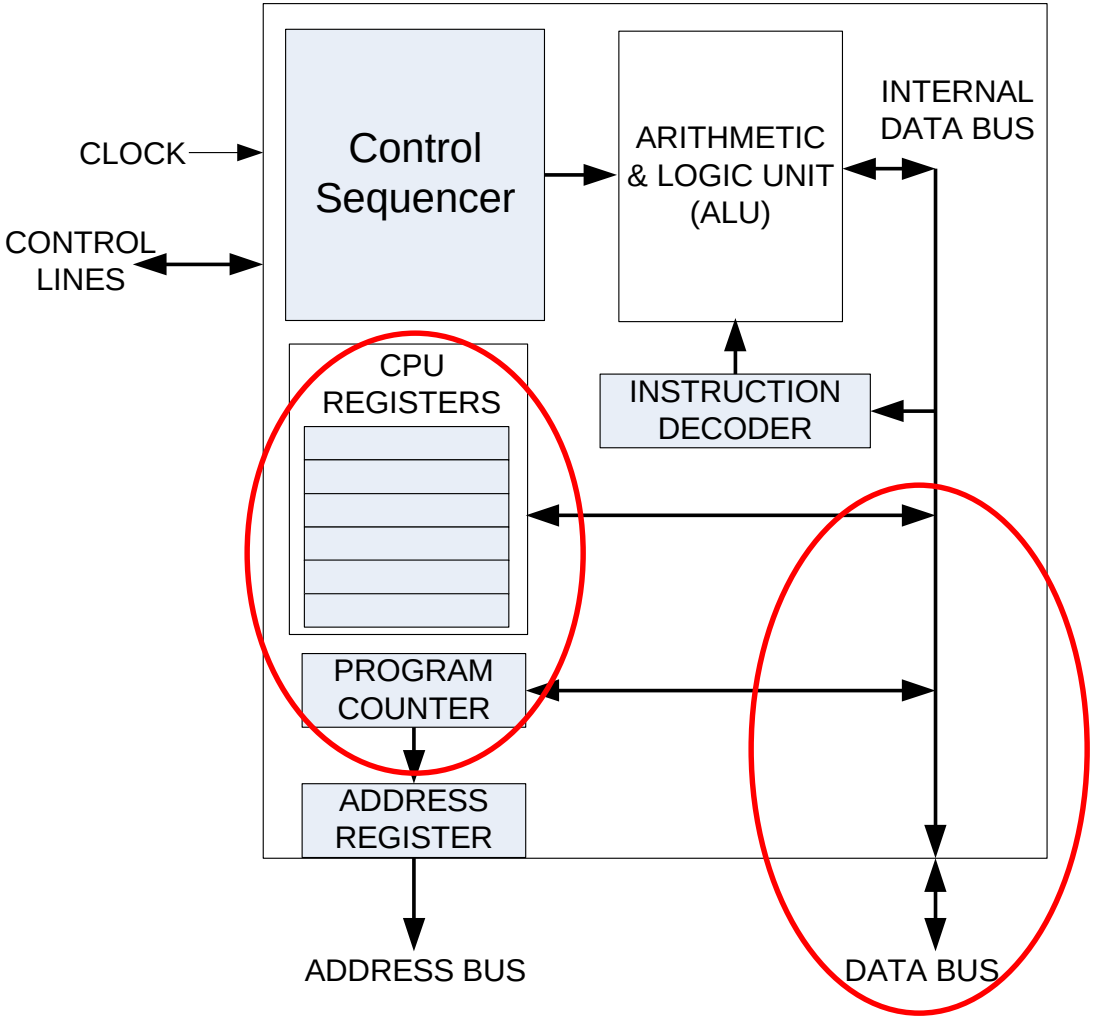
Fetch/(Decode)/Execute Cycle



1. CPU keeps track of the location of the next instruction or data byte (to fetch) through the Program Counter (PC).

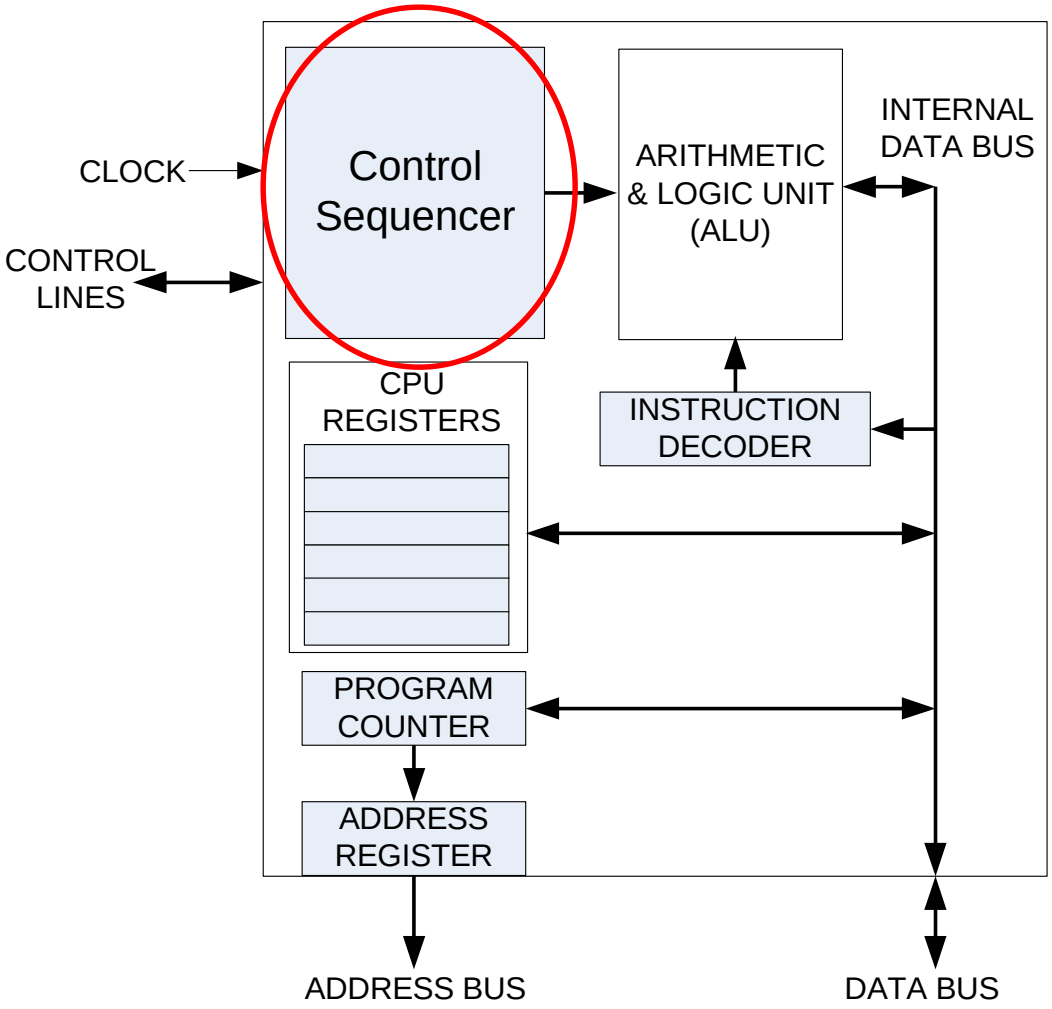
2. Instruction Decoder decodes the received data and configures the Arithmetic Logic Unit (ALU) to process it. Arithmetic or logic operations execute in the ALU.

Fetch/(Decode)/Execute Cycle

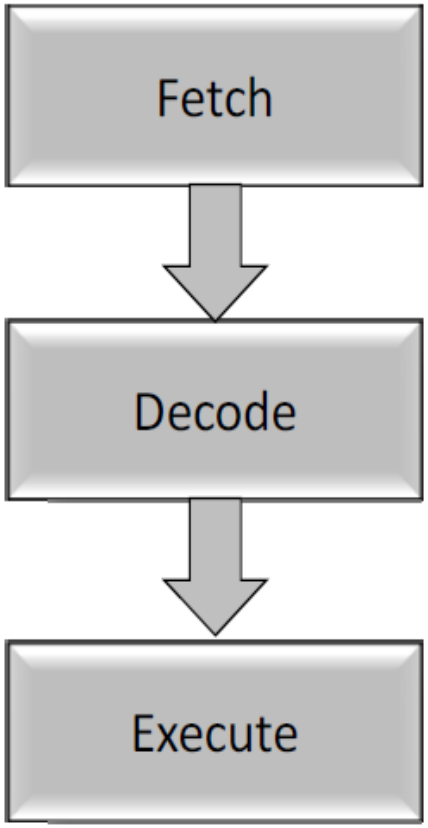


1. CPU keeps track of the location of the next instruction or data byte (to fetch) through the Program Counter (PC).
2. Instruction Decoder decodes the received data and configures the Arithmetic Logic Unit (ALU) to process it. Arithmetic or logic operations execute in the ALU.
3. After the execution, the relevant bits are set, and data gets stored to a register or memory location if applicable.

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.

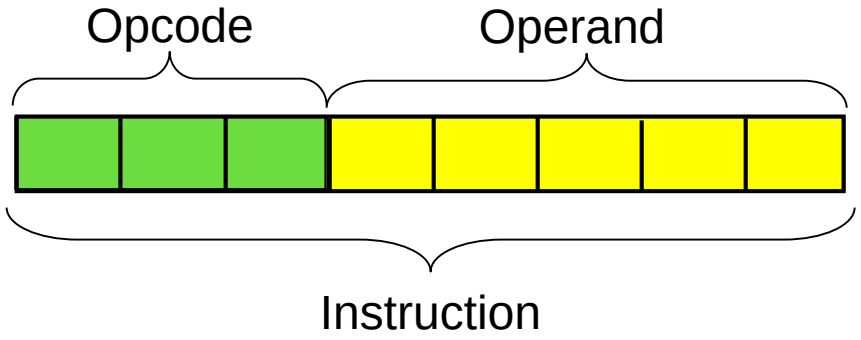


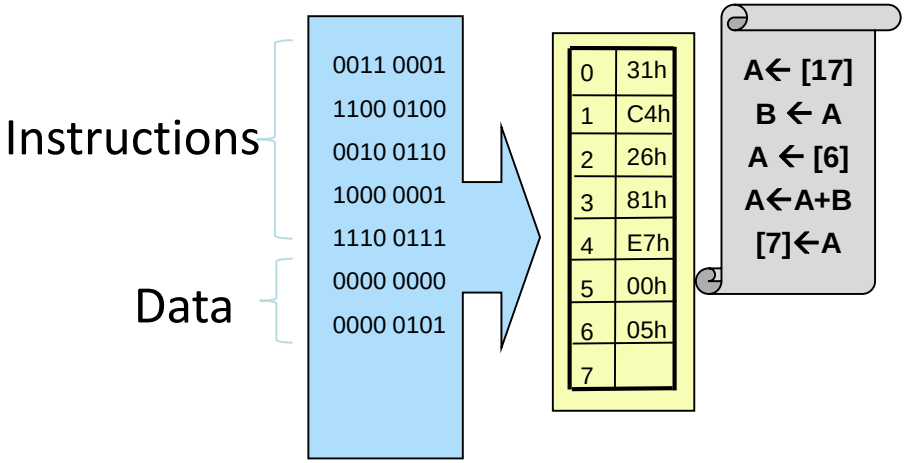
16-bit Instruction fetched
from memory

Decompress thumb instruction, Decode
ARM instruction
Select registers

Read register(s) from Register Bank,
Shift and ALU operation,
Write register(s) back to Register Bank

Instruction Decode

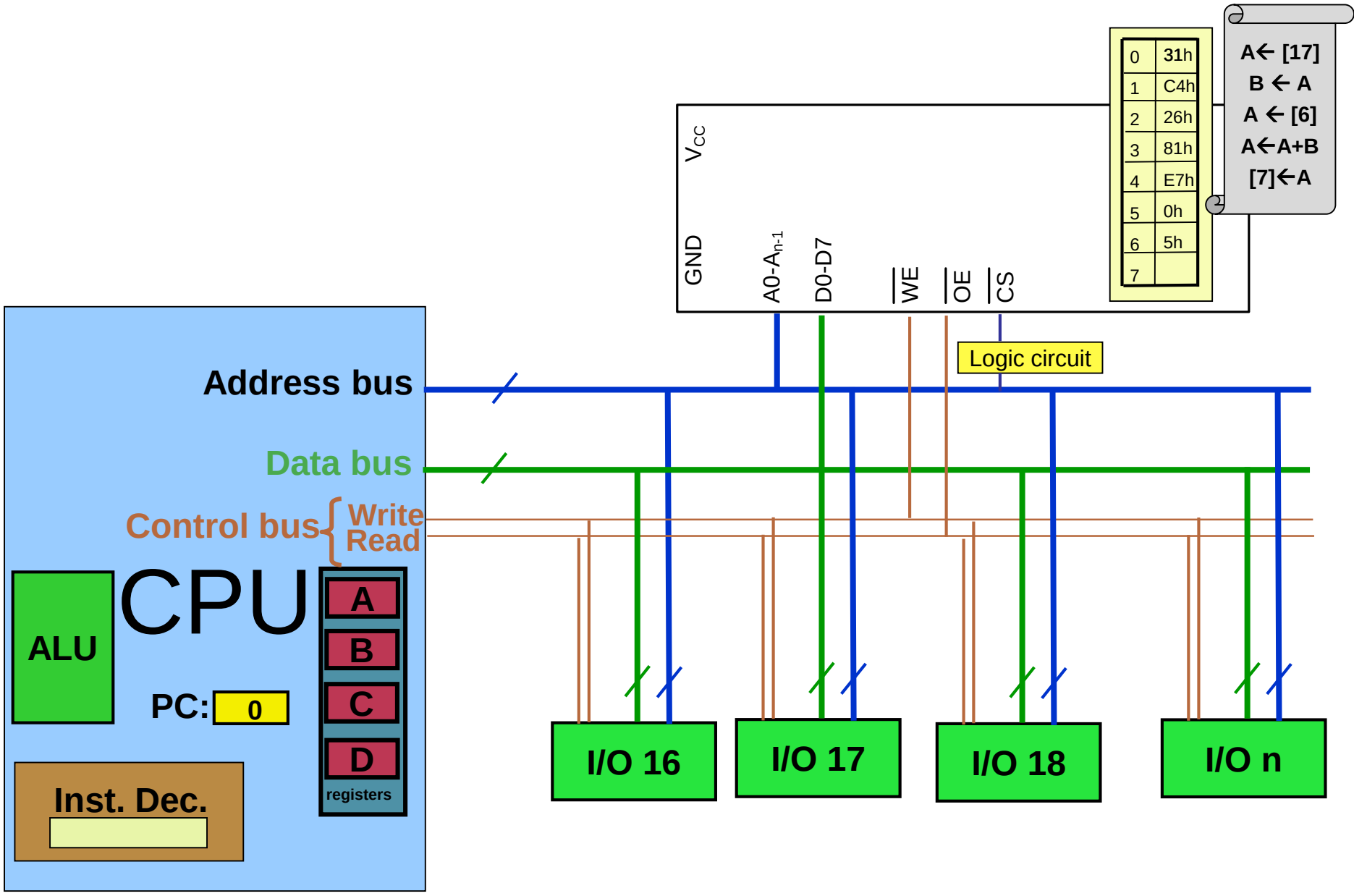




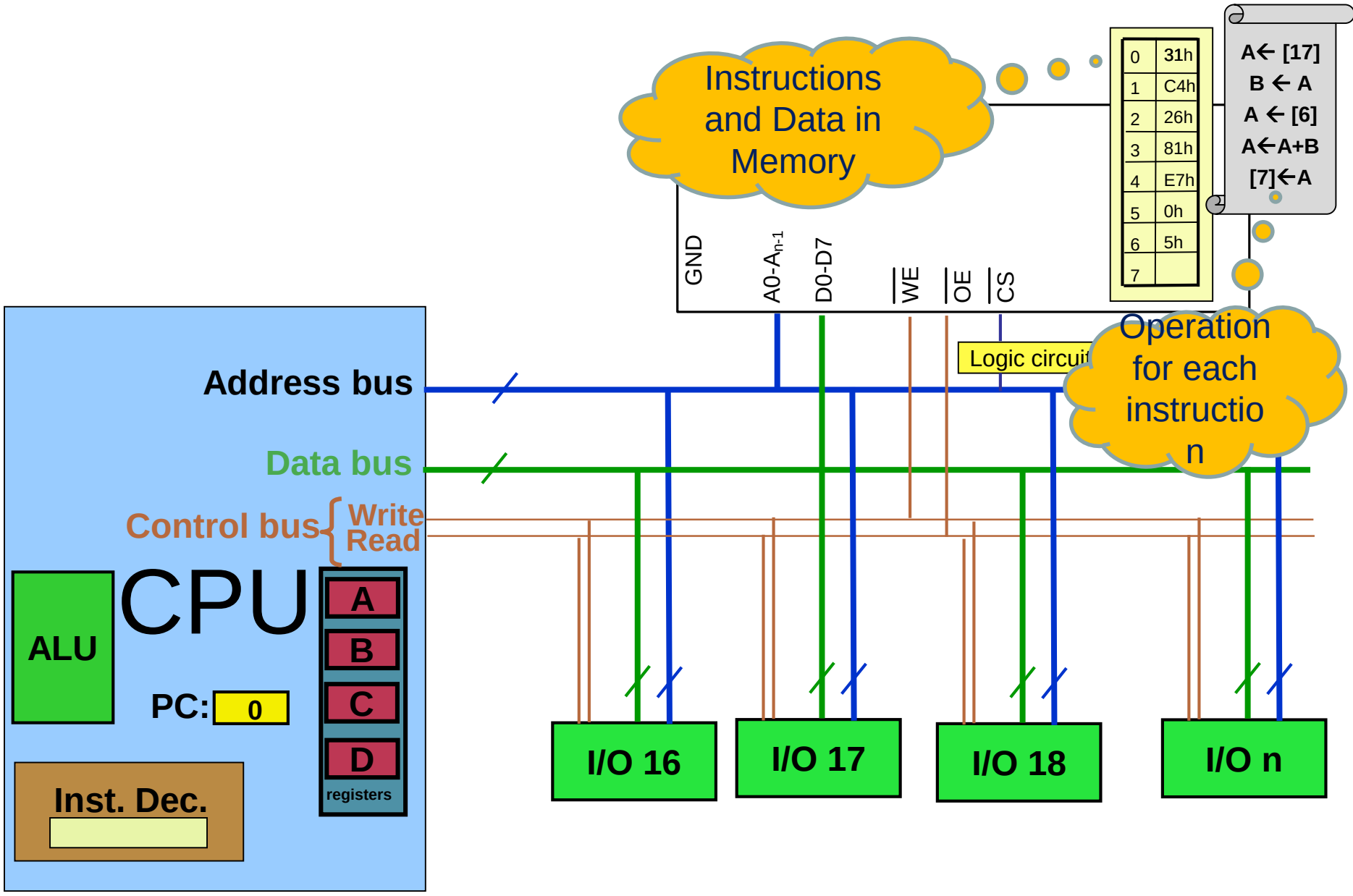
Operation Code	Meaning
000	A ← x
001	A ← [x]
010	A ← A – register (x)
011	A ← A + x
100	A ← A + register (x)
101	A ← A – x
110	Register (x_H) ← Register (x_L)
111	[x] ← A

Note: These are hypothetical instructions used for this example…

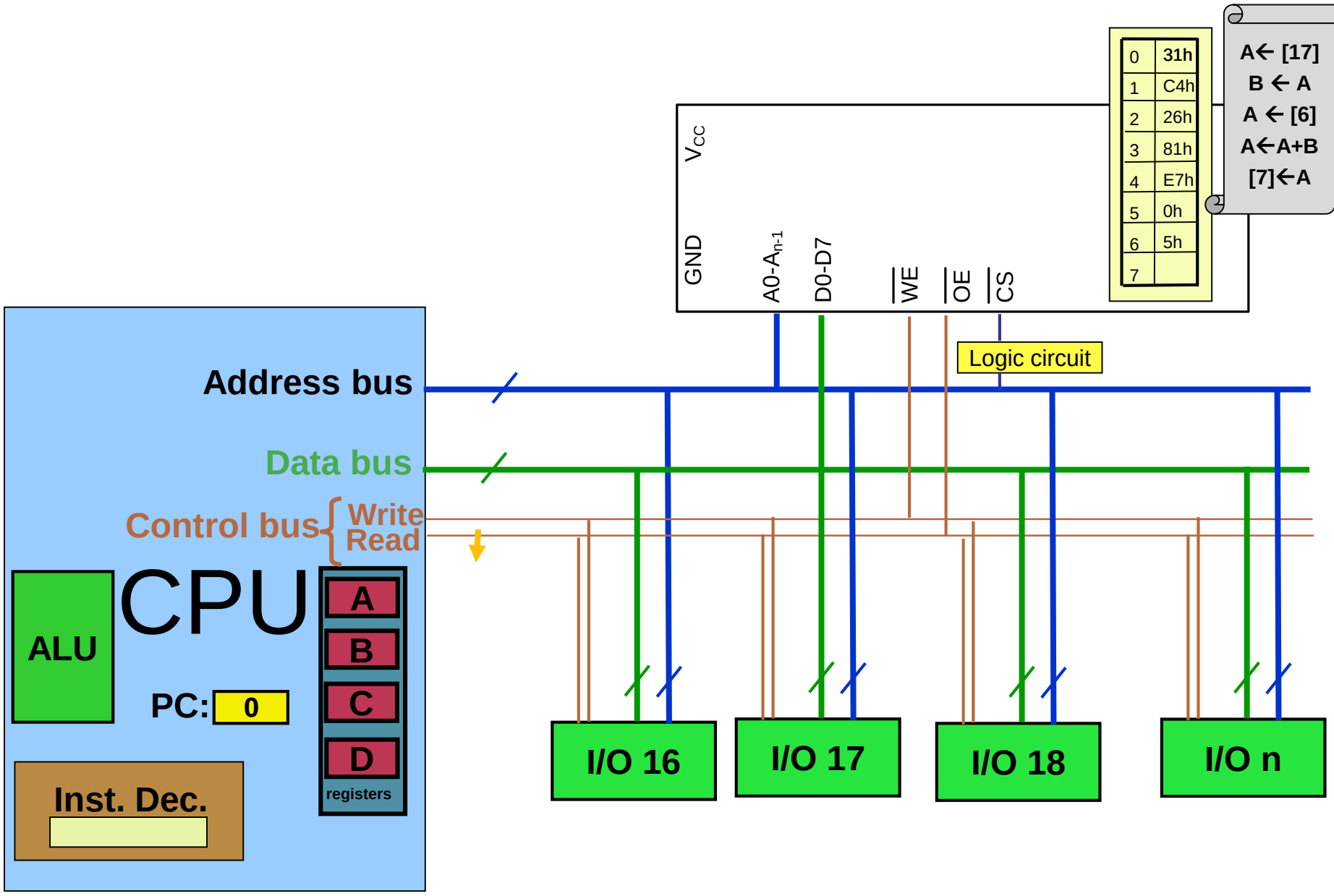
Instruction Fetch➔(Decode)➔Execute in CPU



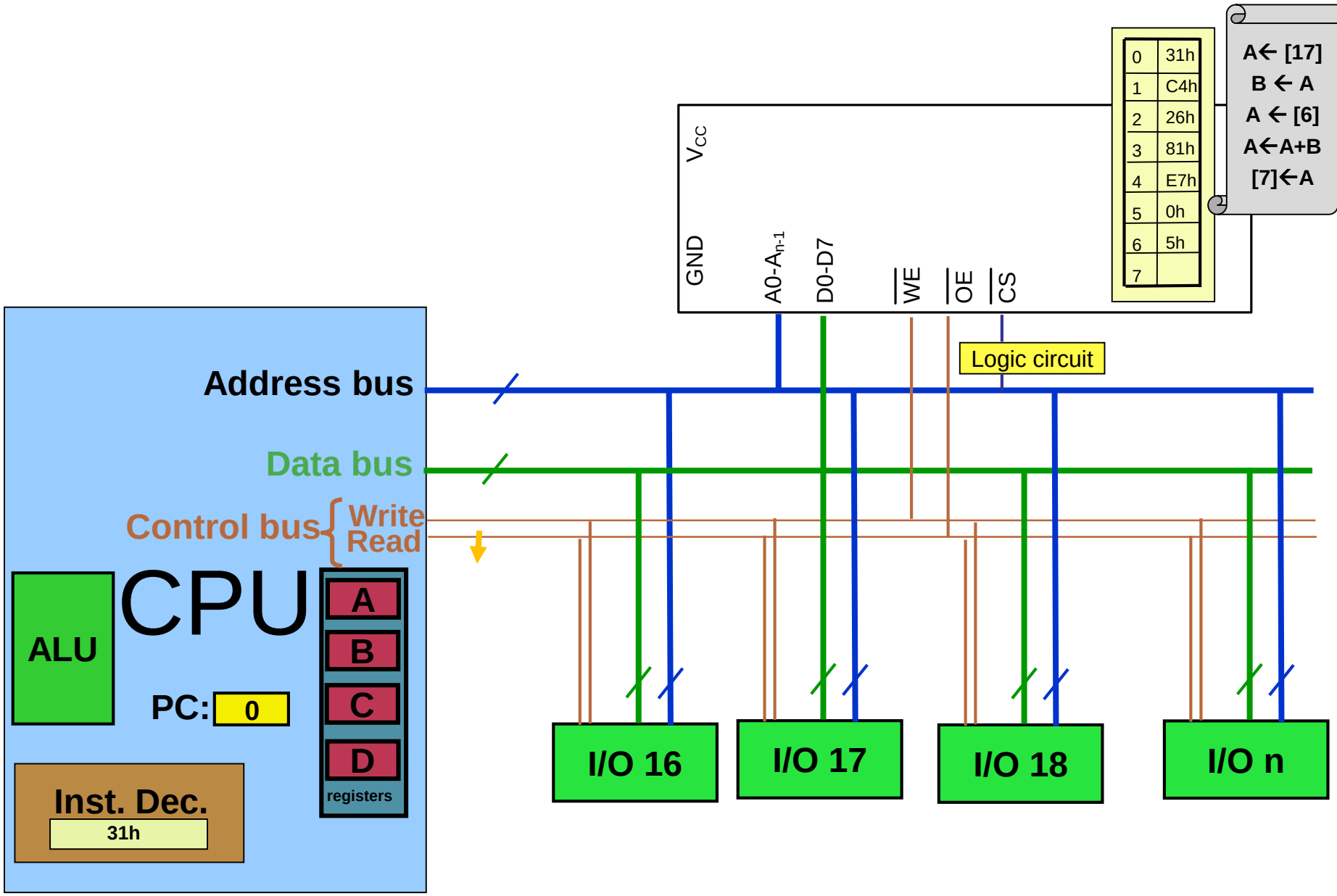
Instruction Fetch➔(Decode)➔Execute in CPU



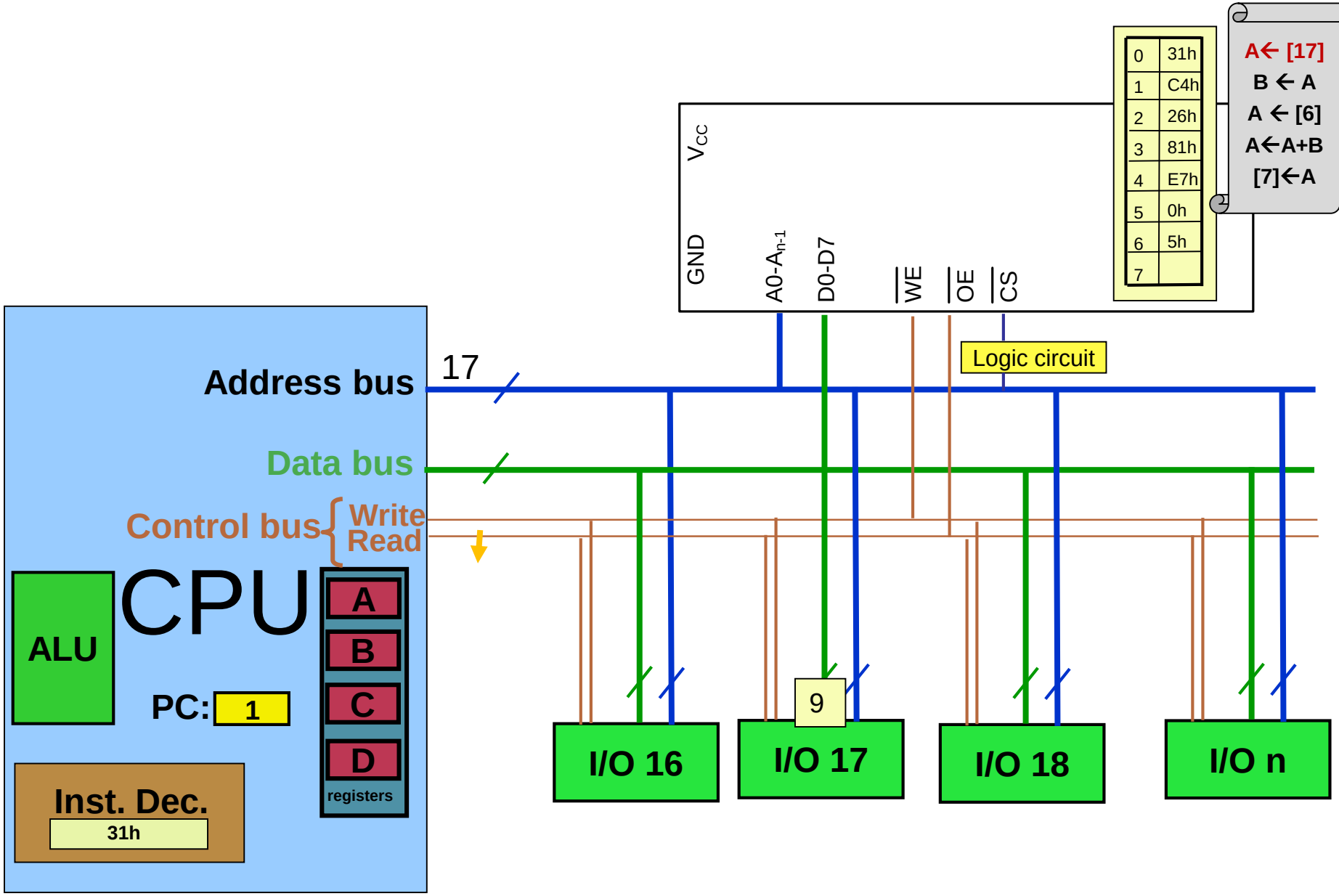
Instruction Fetch➔(Decode)➔Execute in CPU



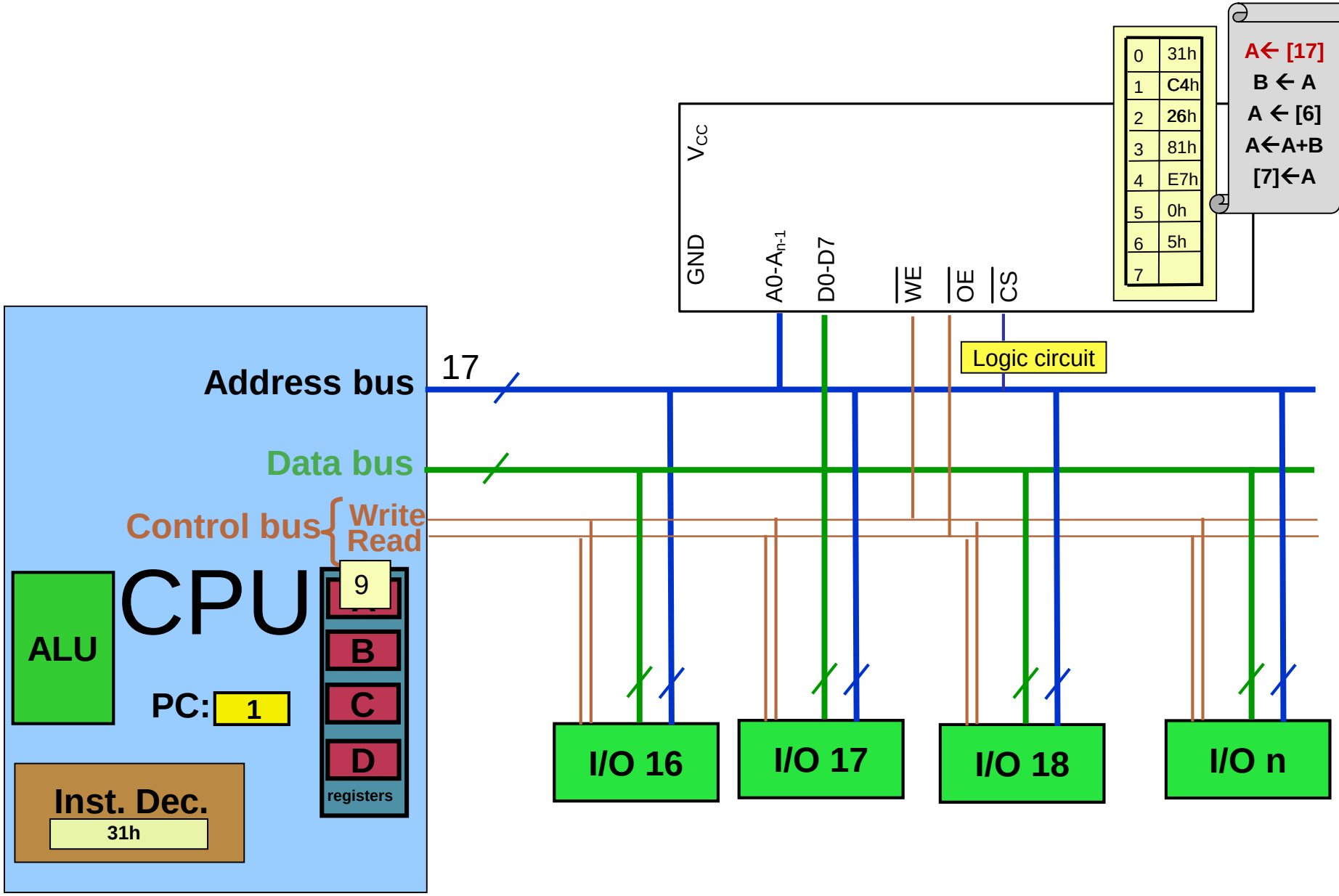
Instruction Fetch➔(Decode)➔Execute in CPU



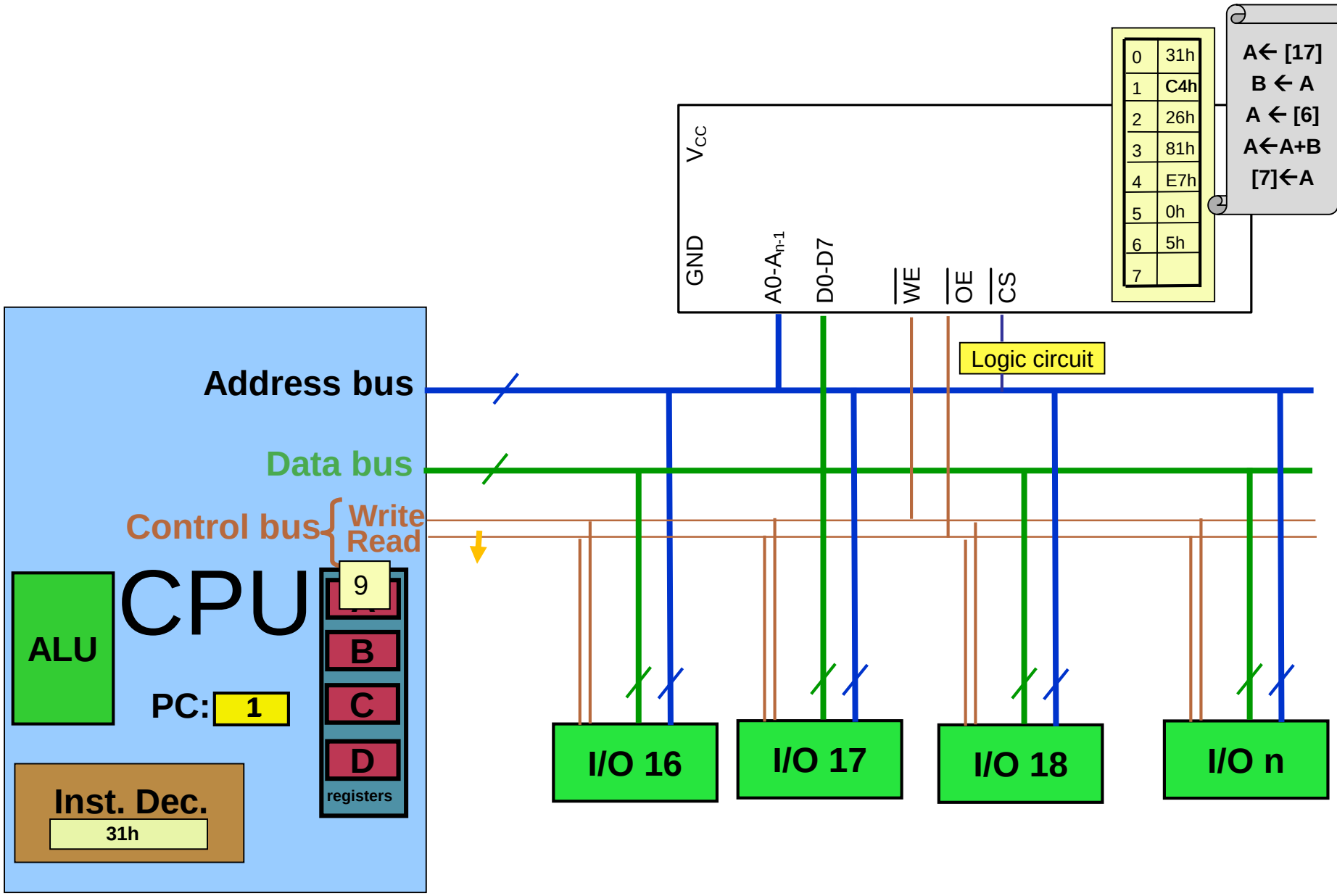
Instruction Fetch➔(Decode)➔Execute in CPU



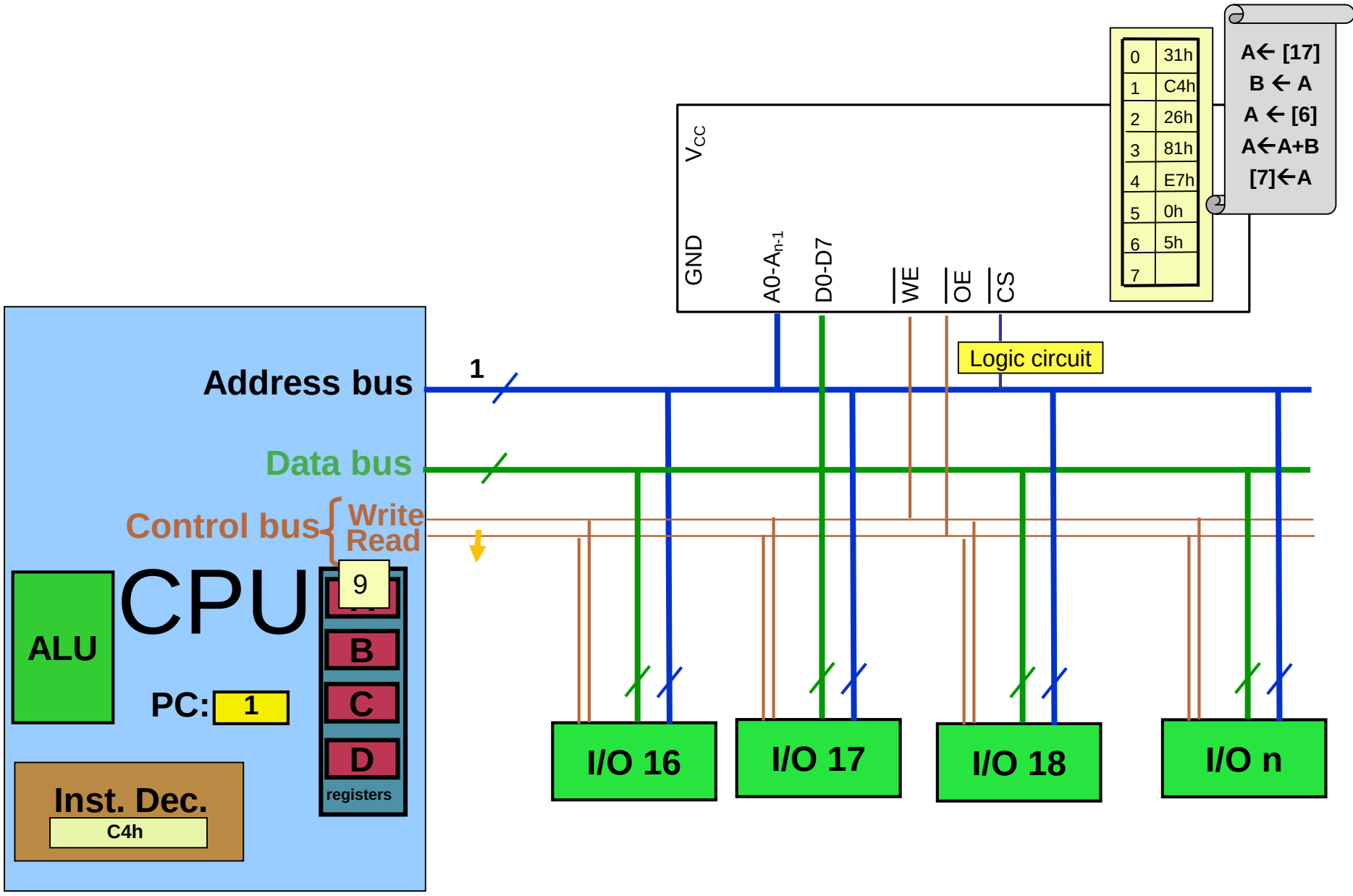
Instruction Fetch➔(Decode)➔Execute in CPU



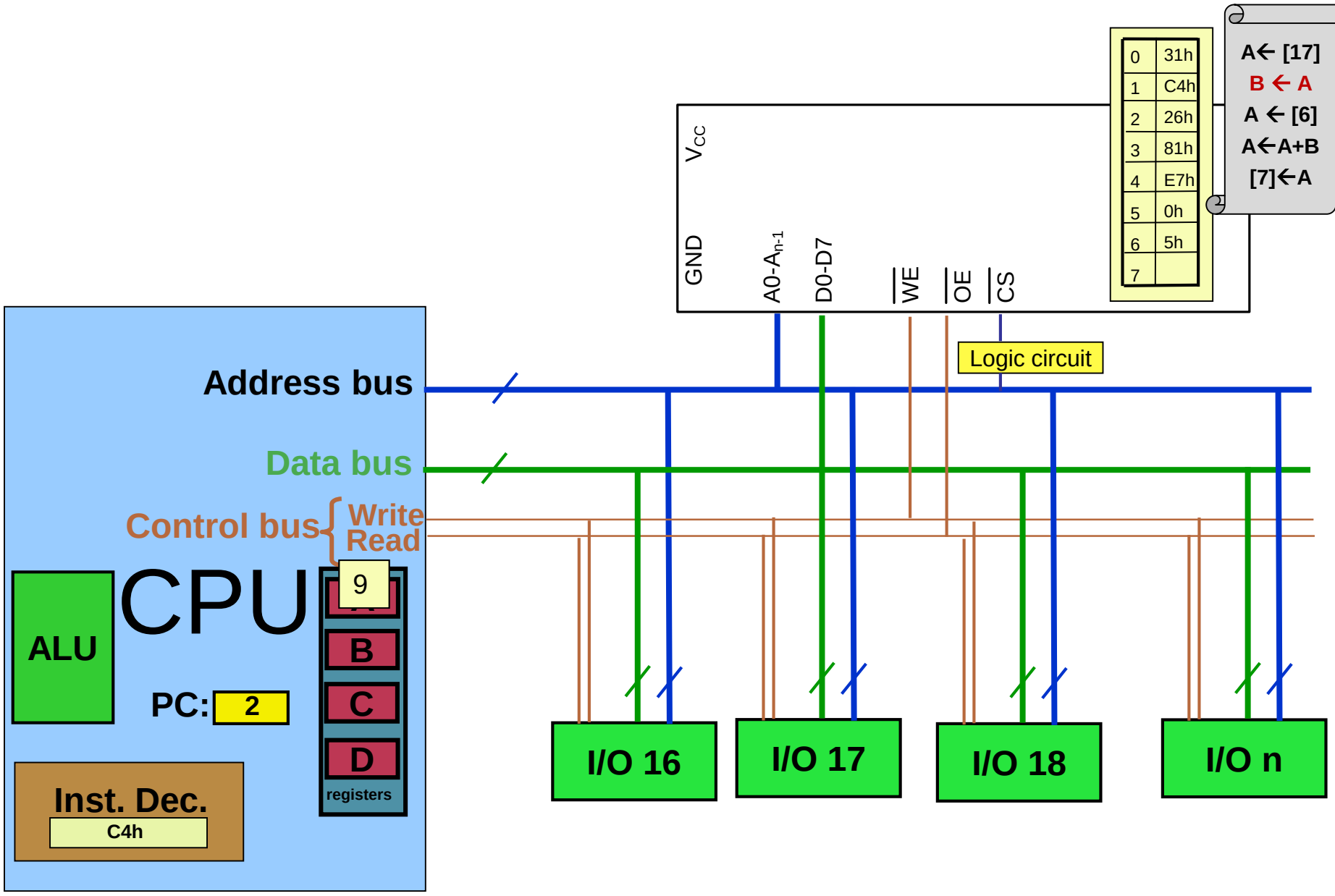
Instruction Fetch➔(Decode)➔Execute in CPU



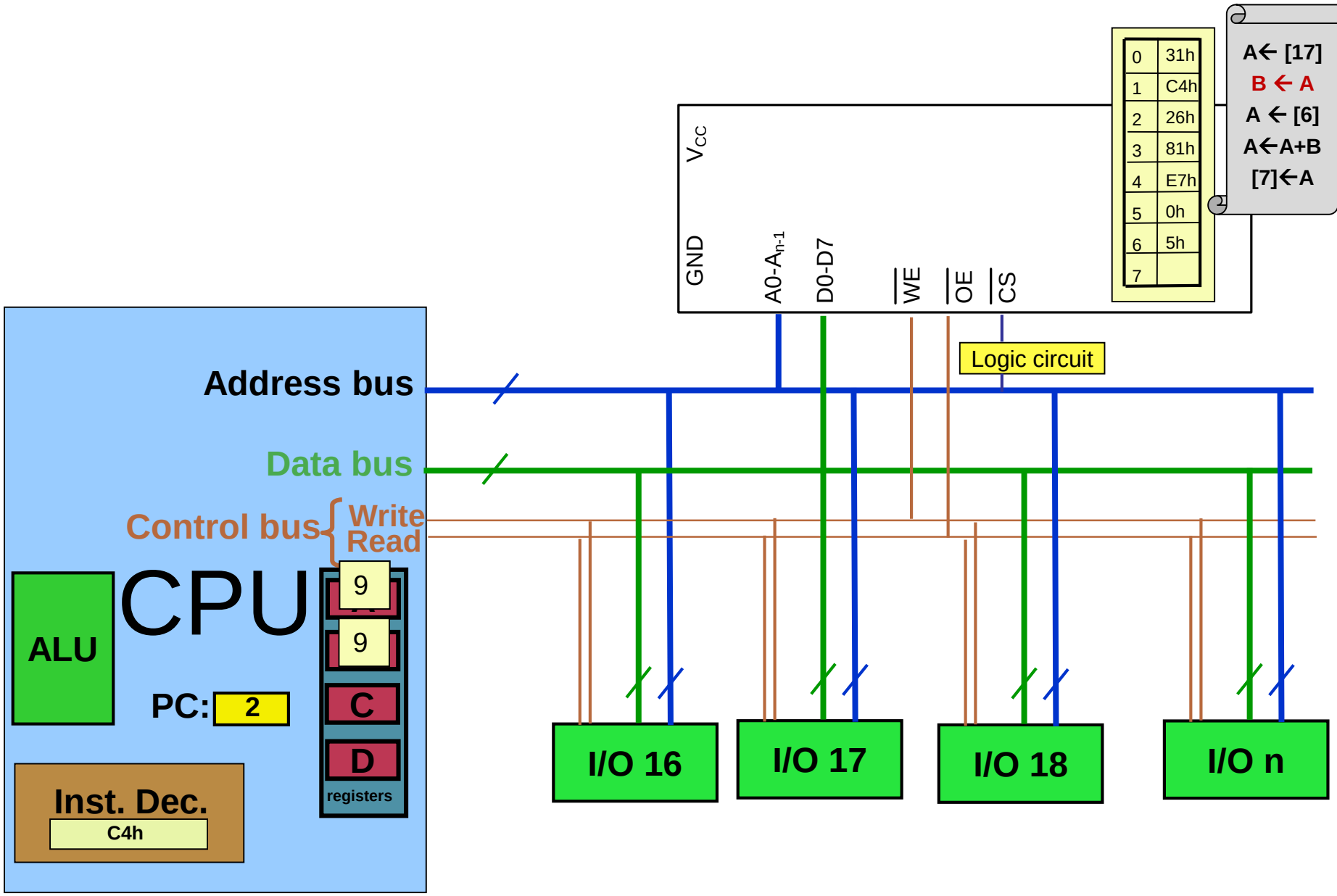
Instruction Fetch➔(Decode)➔Execute in CPU



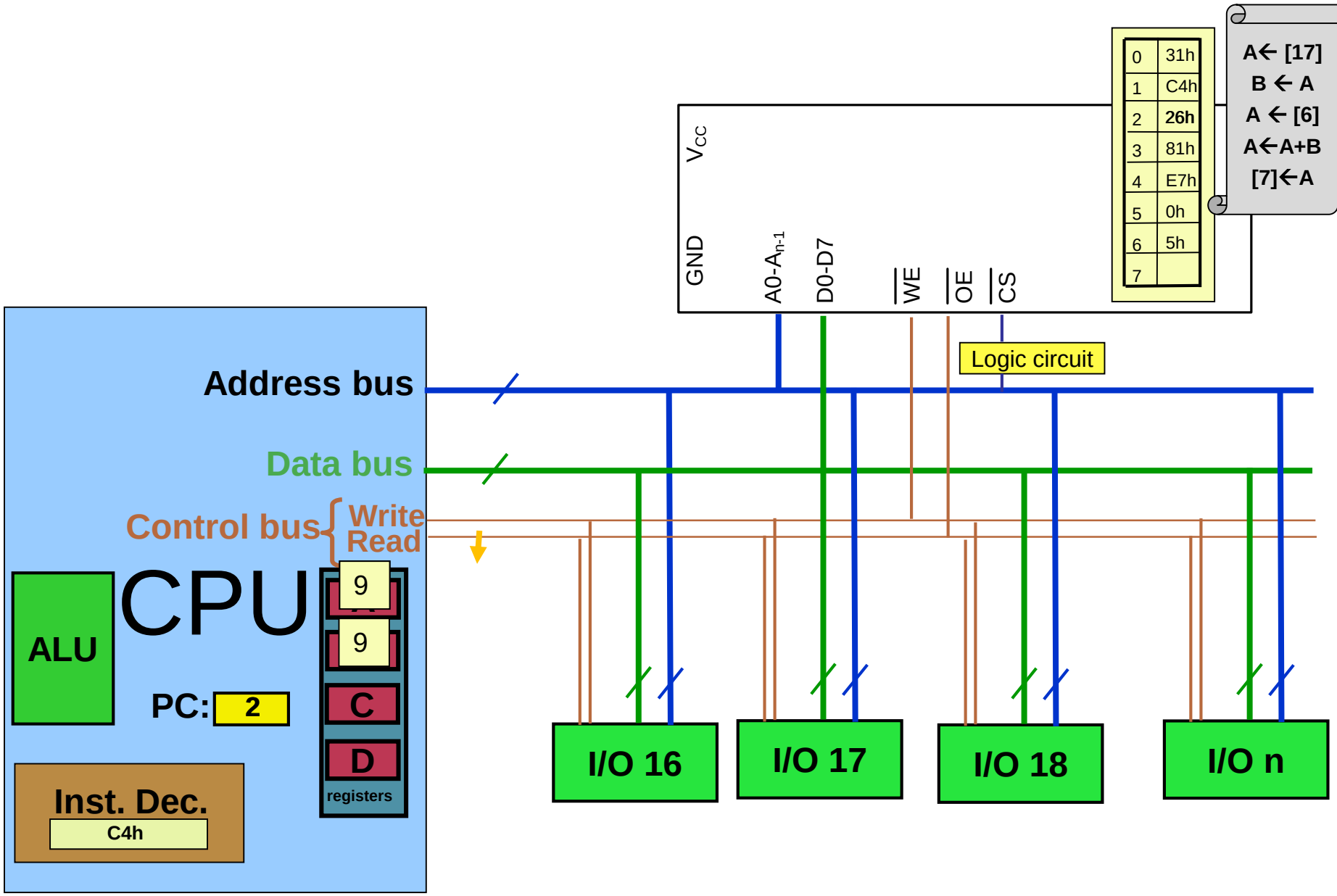
Instruction Fetch➔(Decode)➔Execute in CPU



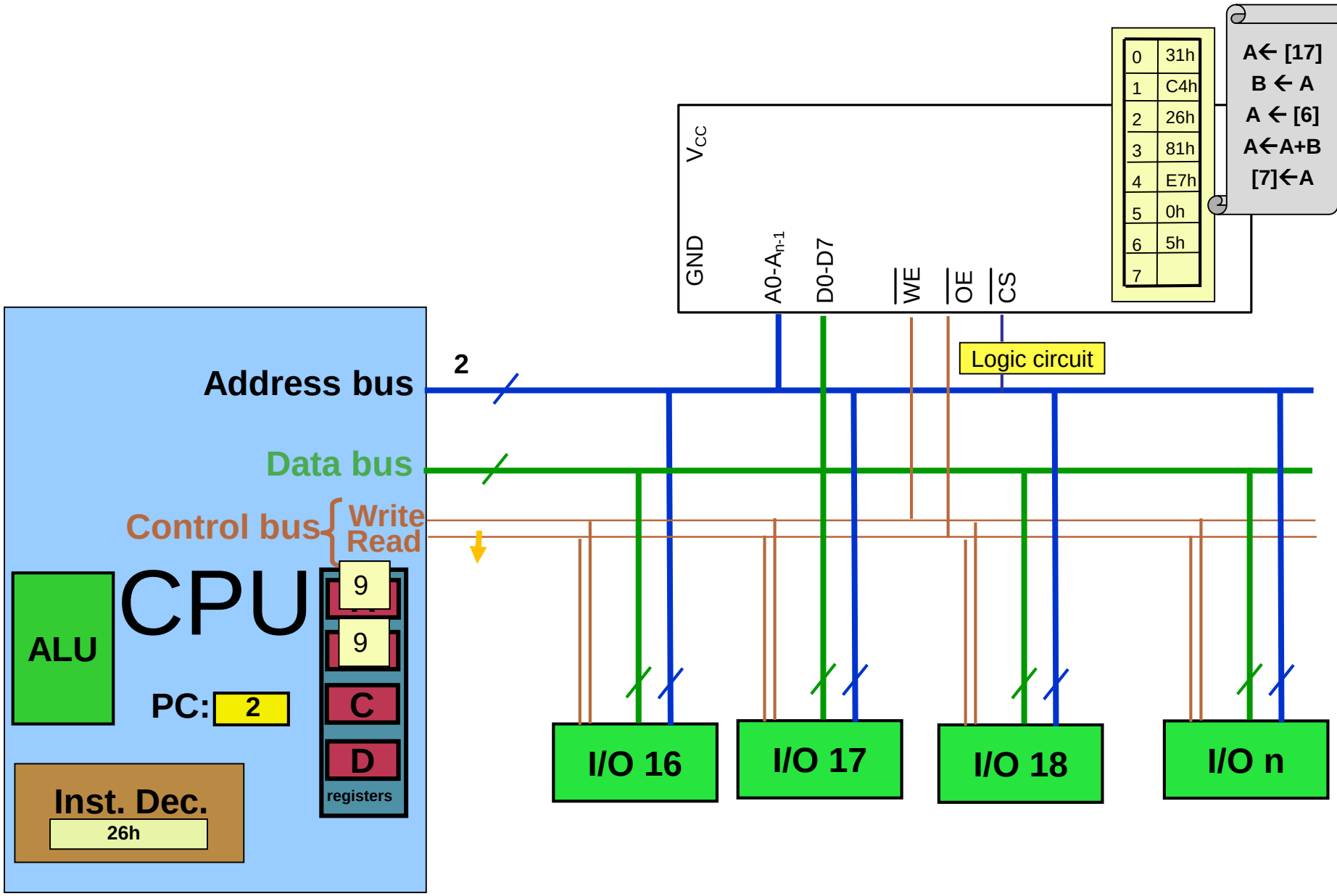
Instruction Fetch➔(Decode)➔Execute in CPU



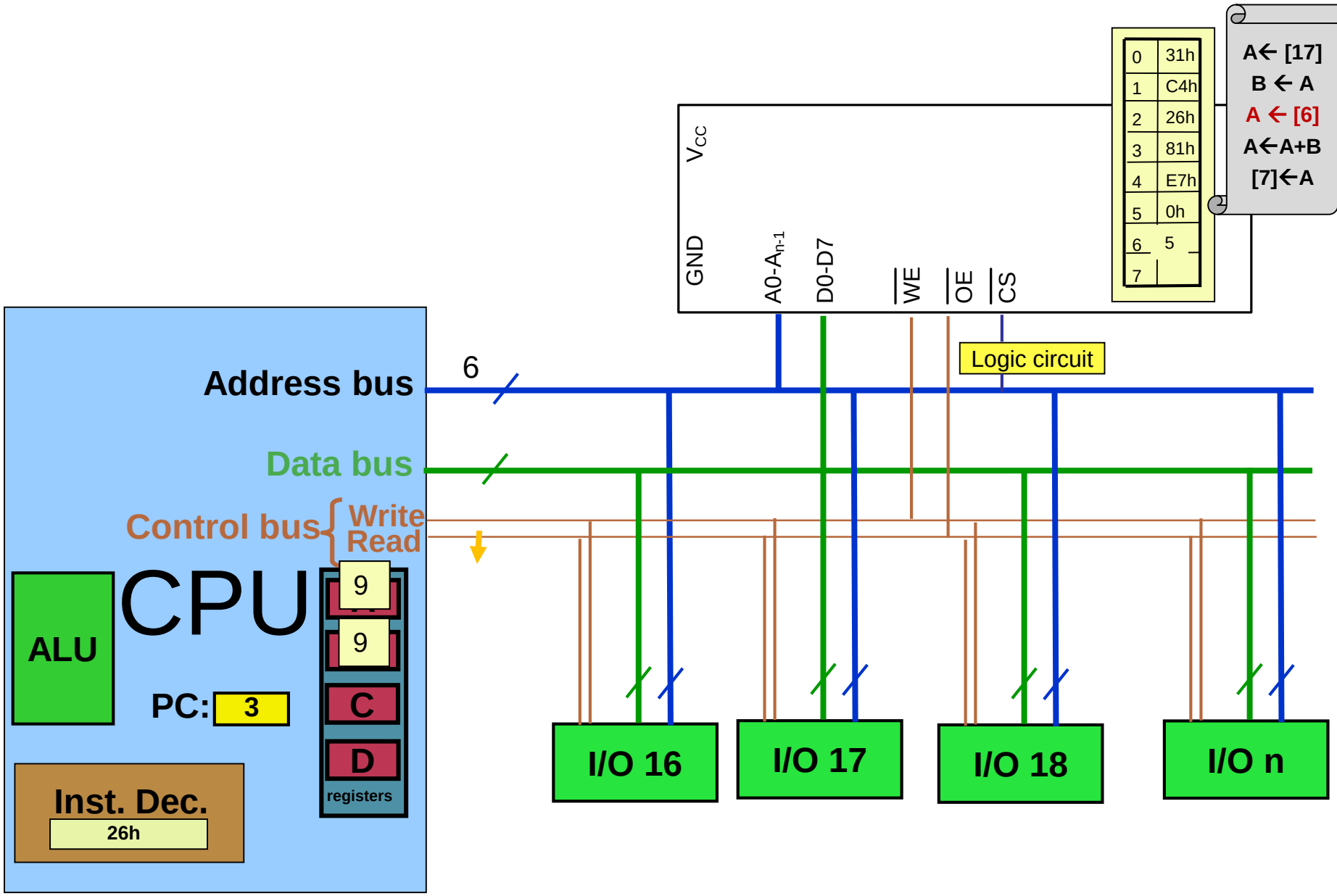
Instruction Fetch➔(Decode)➔Execute in CPU



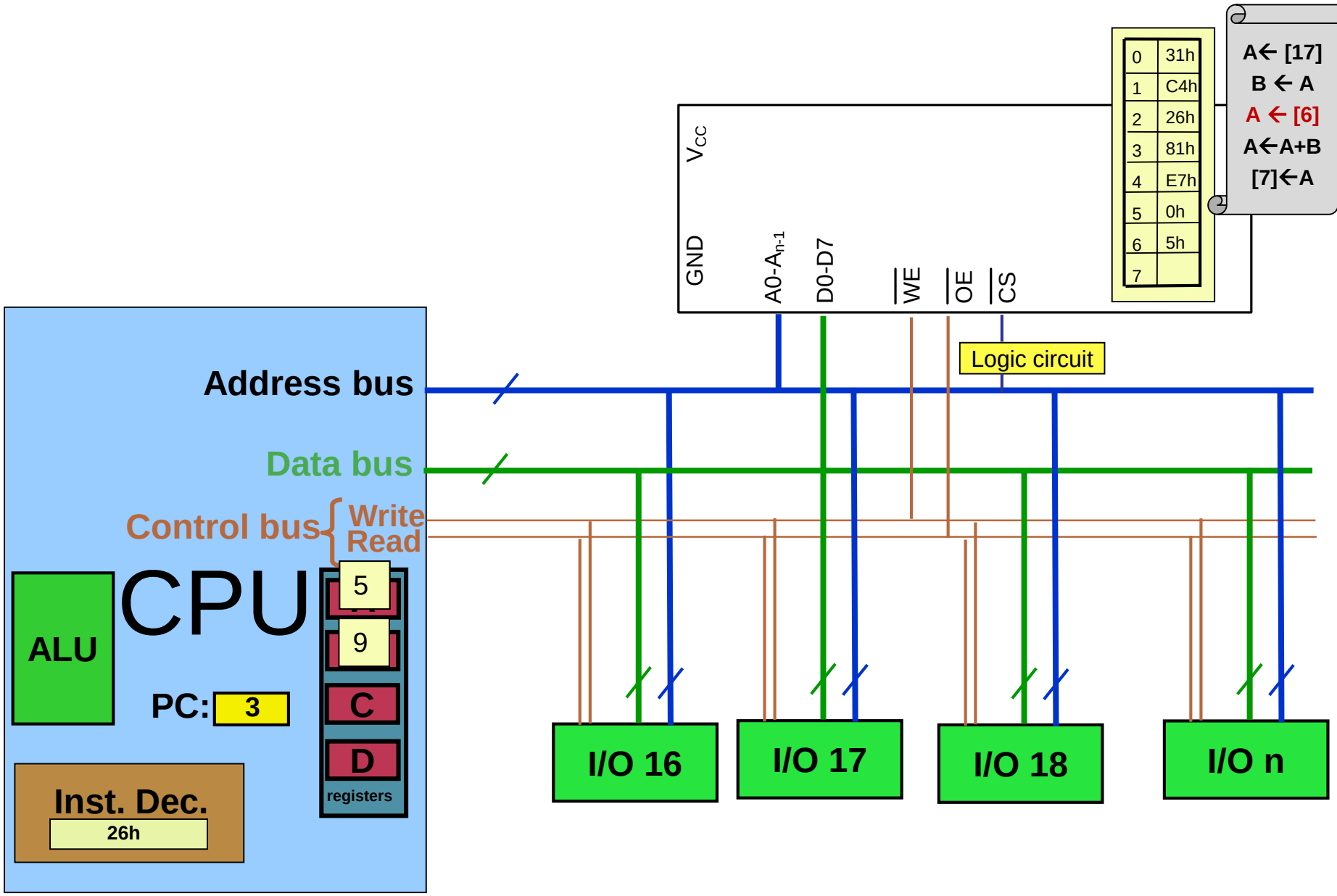
Instruction Fetch➔(Decode)➔Execute in CPU



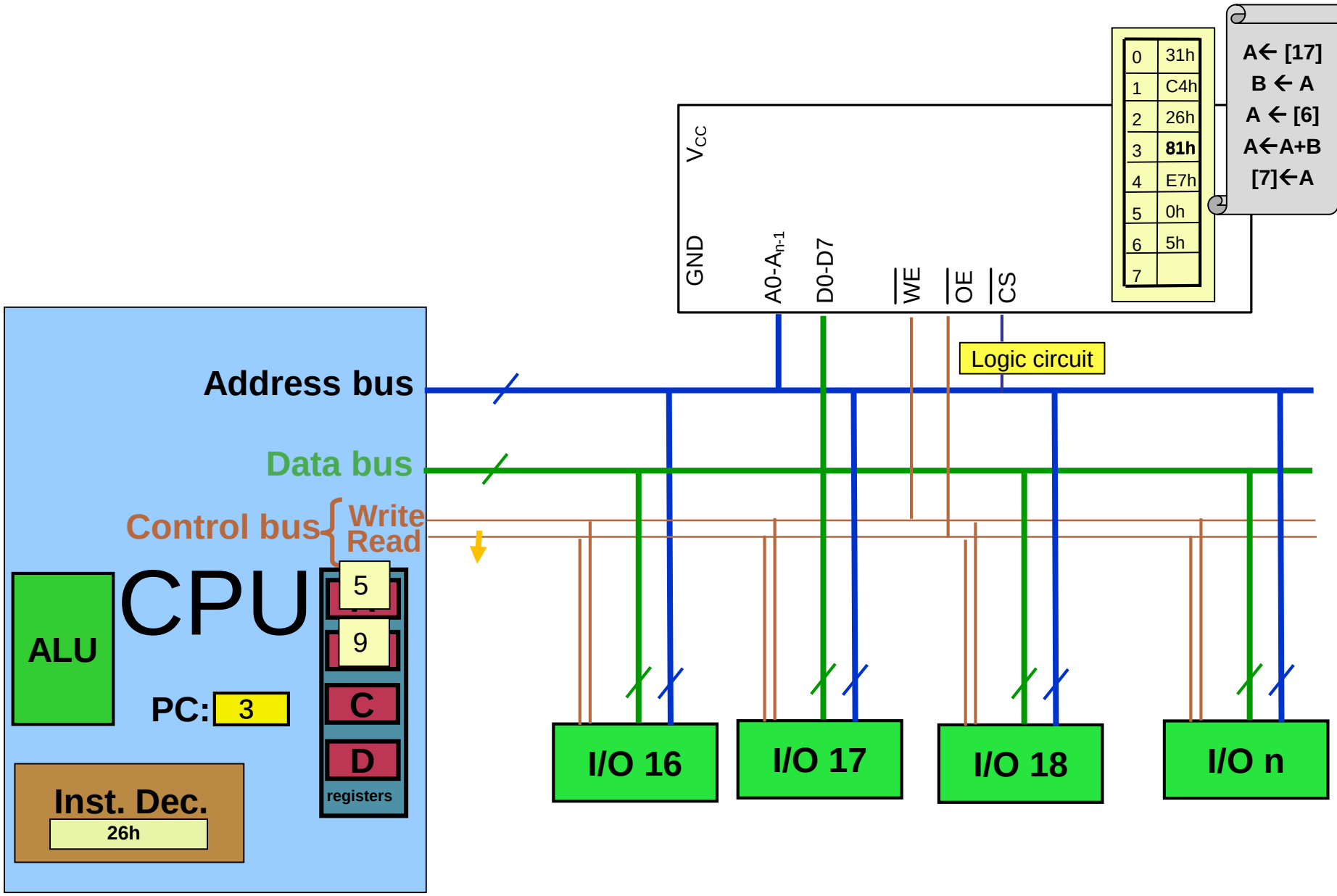
Instruction Fetch➔(Decode)➔Execute in CPU



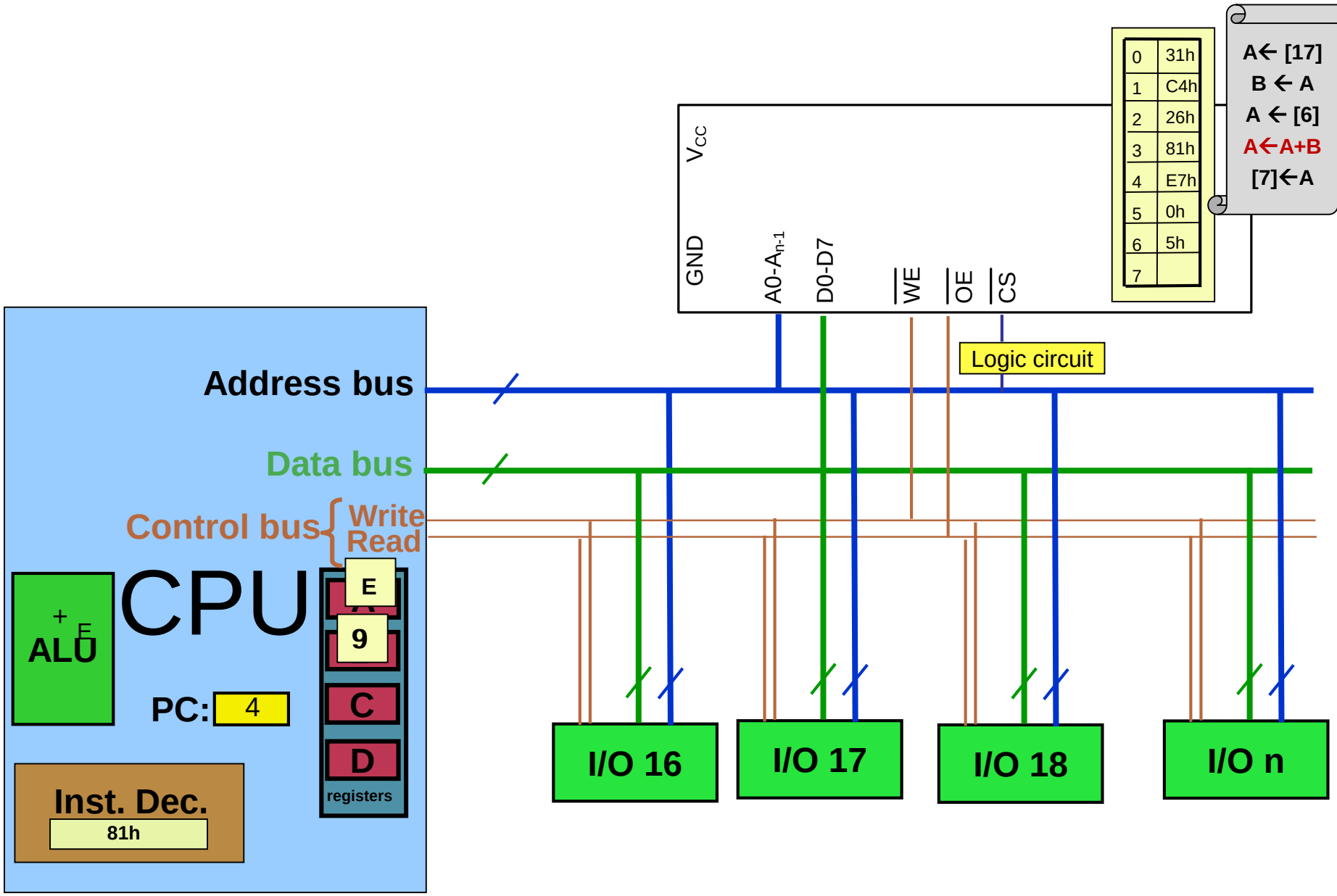
Instruction Fetch➔(Decode)➔Execute in CPU



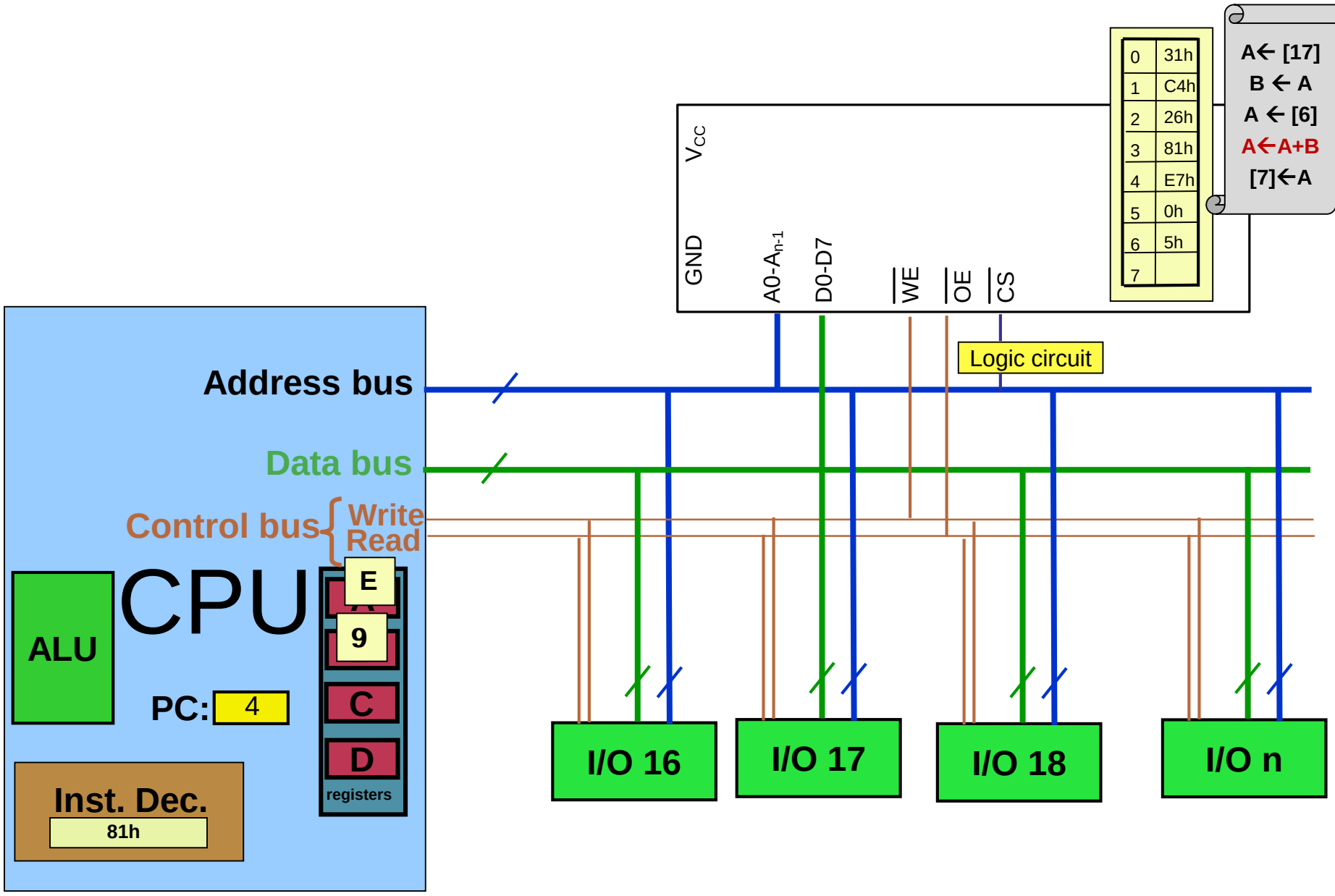
Instruction Fetch➔(Decode)➔Execute in CPU



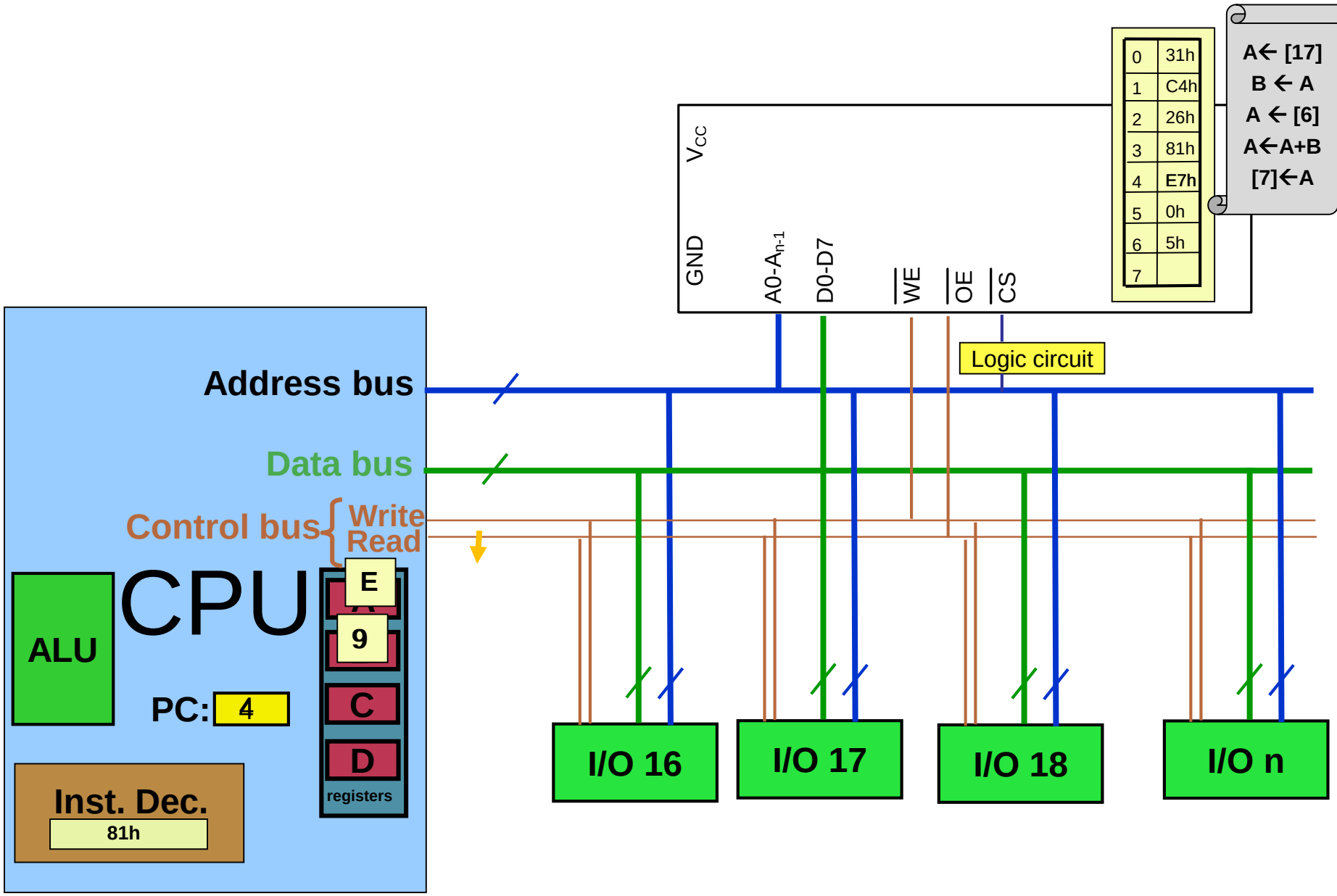
Instruction Fetch➔(Decode)➔Execute in CPU



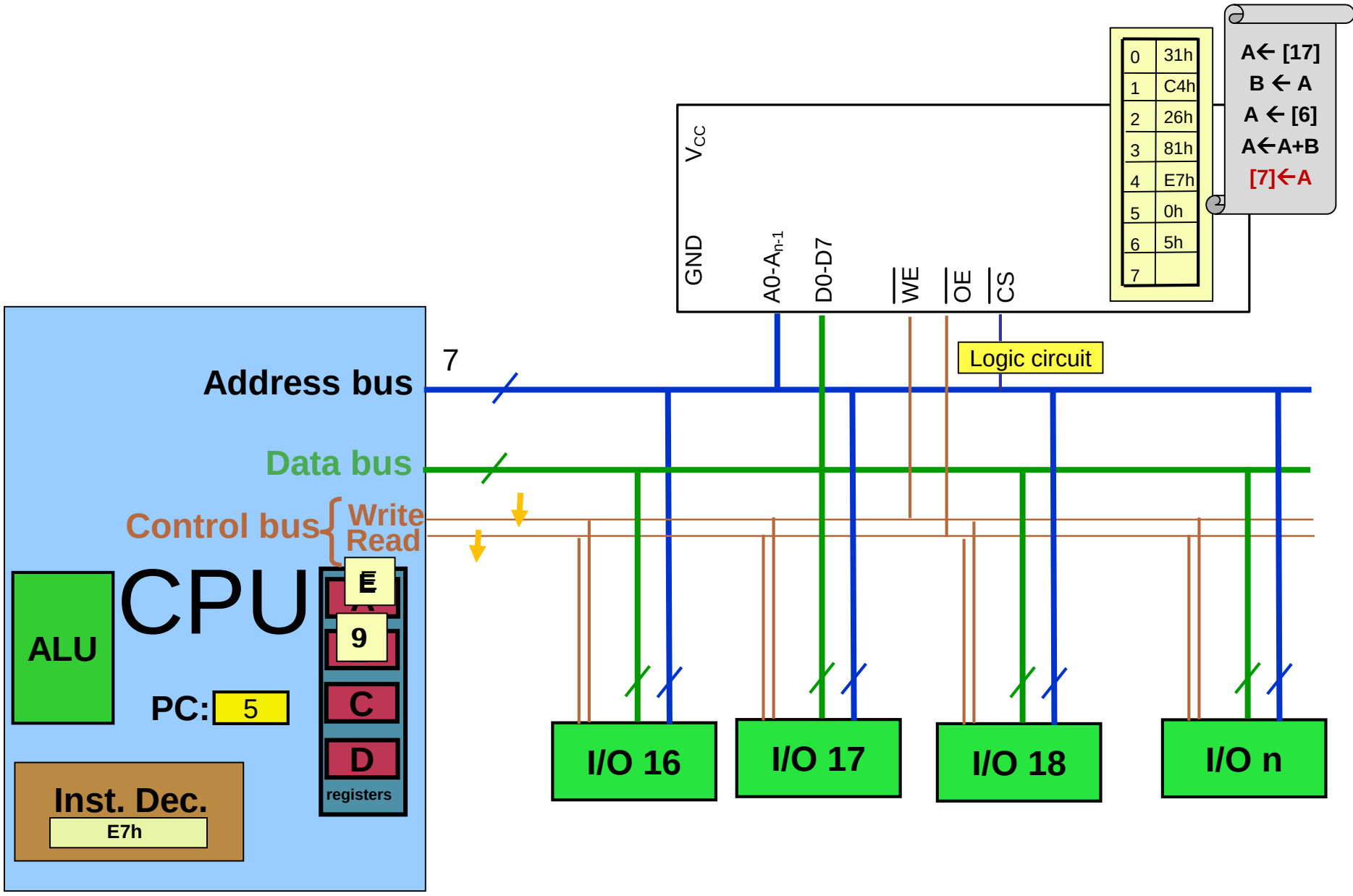
Instruction Fetch➔(Decode)➔Execute in CPU



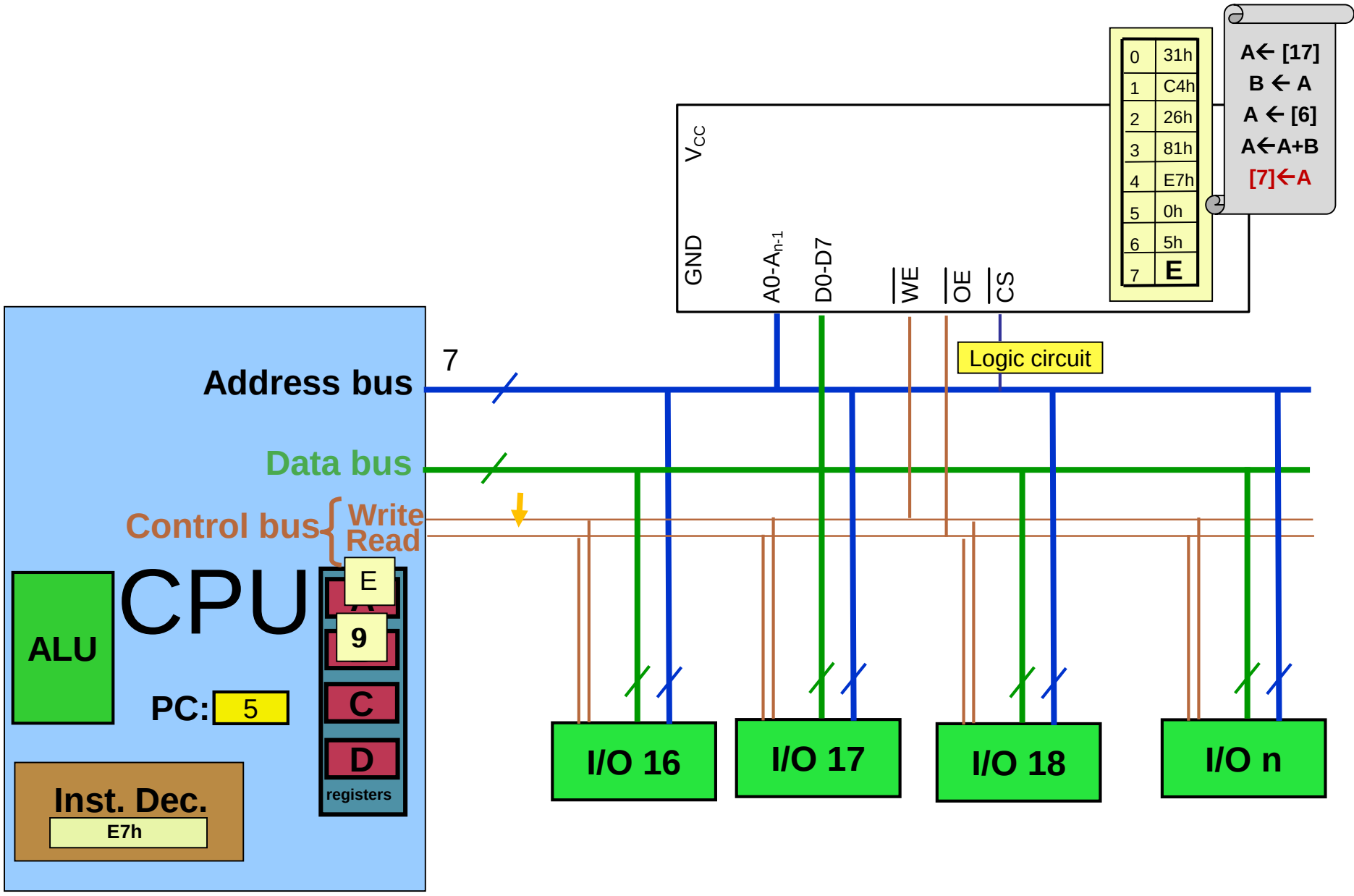
Instruction Fetch➔(Decode)➔Execute in CPU



Instruction Fetch➔(Decode)➔Execute in CPU



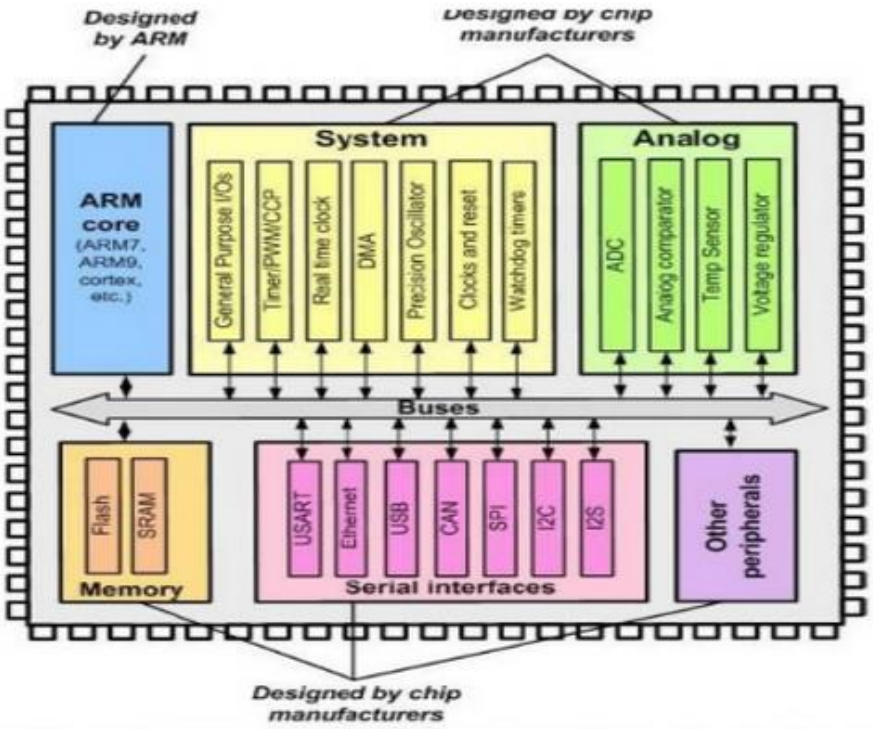
Instruction Fetch➔(Decode)➔Execute in CPU

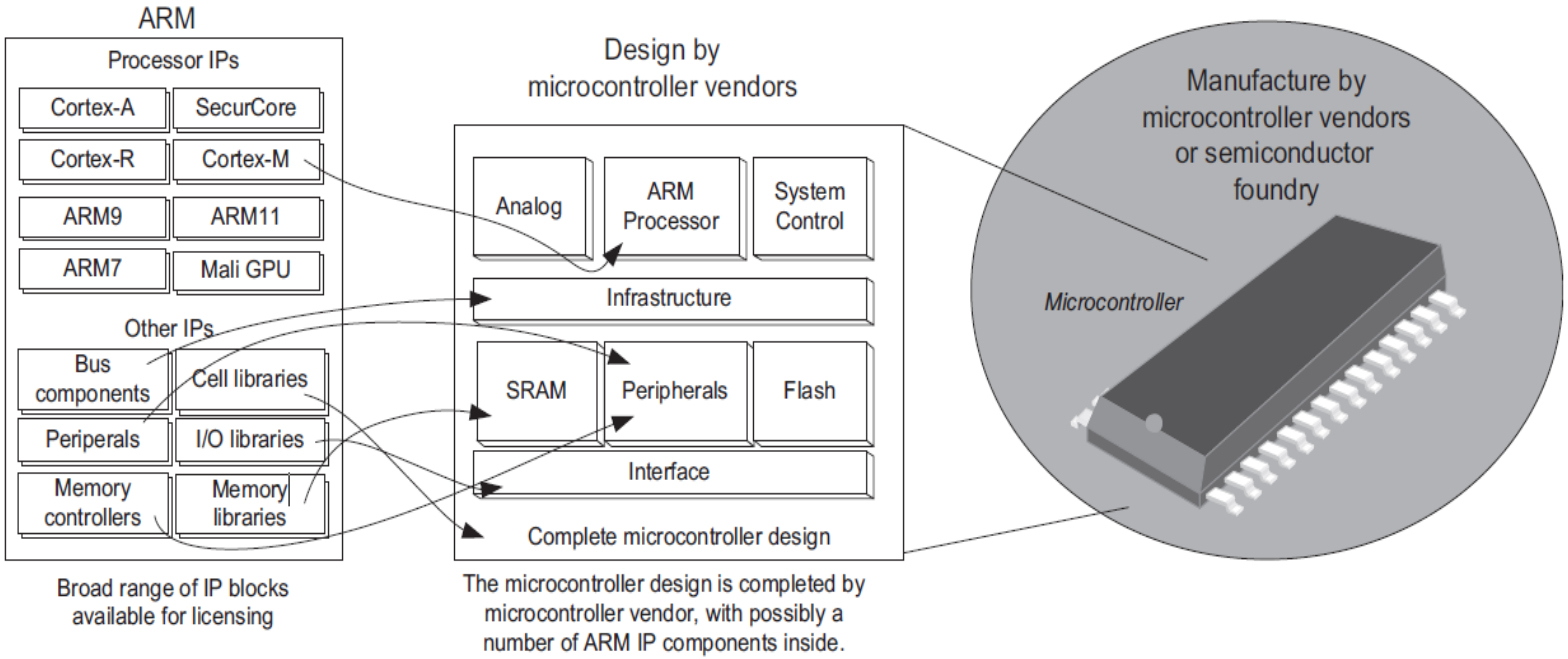


ARM features

Why ARM?

- 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)


Designed by ARM
Designed by chip manufacturers
ARM core (ARM7, ARM9, cortex, etc.)
System
General Purpose I/Os
Timer/PWM/CCP
Real time clock
DMA
Precision Oscillator
Clocks and reset
Watchdog timers
Analog
ADC
Analog comparator
Temp Sensor
Voltage regulator
Buses
Flash
SRAM
Memory
USART
Ethernet
USB
CAN
SPI
I2C
I2S
Serial interfaces
Other peripherals
Designed by chip manufacturers


ARM
Processor IPs
Cortex-A
SecurCore
Cortex-R
Cortex-M
ARM9
ARM11
ARM7
Mali GPU
Other IPs
Bus components
Cell libraries
Periperals
I/O libraries
Memory controllers
Memory libraries
Broad range of IP blocks available for licensing
Design by microcontroller vendors
Analog
ARM Processor
System Control
Infrastructure
SRAM
Peripherals
Flash
Interface
Complete microcontroller design
The microcontroller design is completed by microcontroller vendor, with possibly a number of ARM IP components inside.
Manufacture by microcontroller vendors or semiconductor foundry
Microcontroller

ARM History

- 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
  - 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

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**	

Sample ARM powered 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

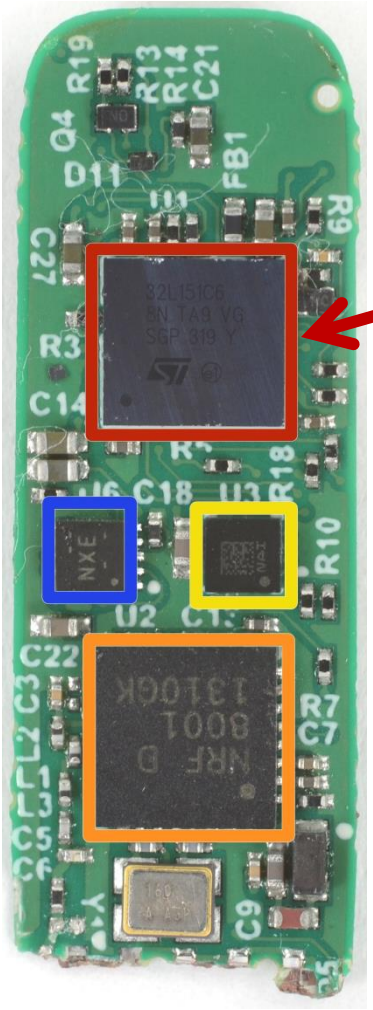
The A8 processor is the first 64-bit ARM based SoC. It supports **ARM A64, A32, and T32** instruction set.

http://www.ifixit.com

Apple Watch

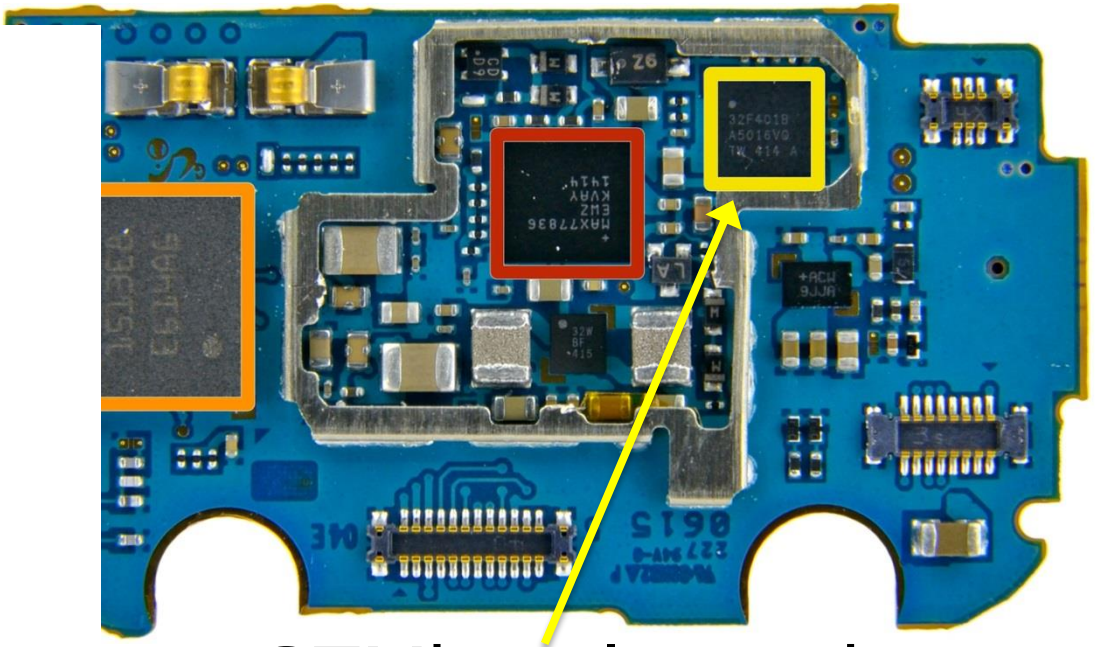
- Apple S1 Processor
  - **32-bit ARMv7-A** compatible
  - # of Cores: **1**
  - CMOS Technology: 28 nm
  - L1 cache 32 KB data
  - L2 cache 256 KB
  - GPU PowerVR SGX543

Fitbit Flex Teardown



STMicroelectronics 32L151C6 Ultra Low Power ARM Cortex M3 Microcontroller

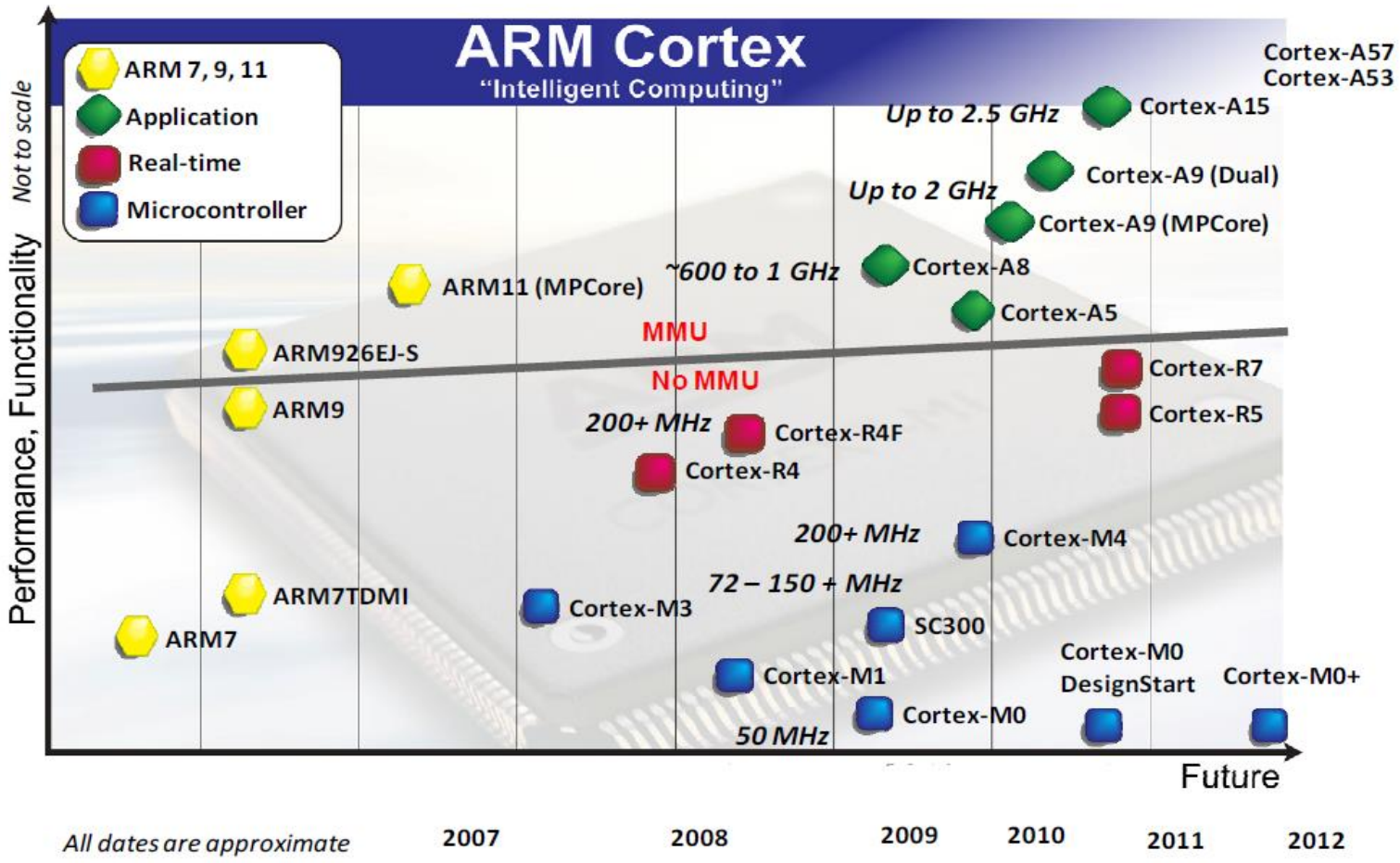
Samsung Galaxy Gear



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

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 Cortex
"Intelligent Computing"
ARM 7, 9, 11
Application
Real-time
Microcontroller
Not to scale
Performance, Functionality
Cortex-A57
Cortex-A53
Up to 2.5 GHz
Cortex-A15
Cortex-A9 (Dual)
Up to 2 GHz
Cortex-A9 (MPCore)
~600 to 1 GHz
Cortex-A8
Cortex-A5
ARM11 (MPCore)
MMU
ARM926EJ-S
No MMU
Cortex-R7
Cortex-R5
ARM9
200+ MHz
Cortex-R4F
Cortex-R4
200+ MHz
Cortex-M4
72 – 150 + MHz
ARM7TDMI
Cortex-M3
SC300
ARM7
Cortex-M0
DesignStart
Cortex-M0+
Cortex-M1
Cortex-M0
50 MHz
Future
All dates are approximate
2007
2008
2009
2010
2011
2012

Where to use?

ARM 7 – Introduced in 1994, used for simple 32-bit devices.

ARM9 – Was most popular ARM family (over 5 billion sold), used for smarthphones, HDD controllers, set top box, 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, oltra 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, incereased efficiency and reliability.
- Cortex R7 – High-performance dual core

Cortex A-series: High performance processors for open Operating sys, 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 Architecture: For Diverse Embedded Processing Needs

Cortex - A

Highest performance

Optimised for
rich operating systems

Cortex - R

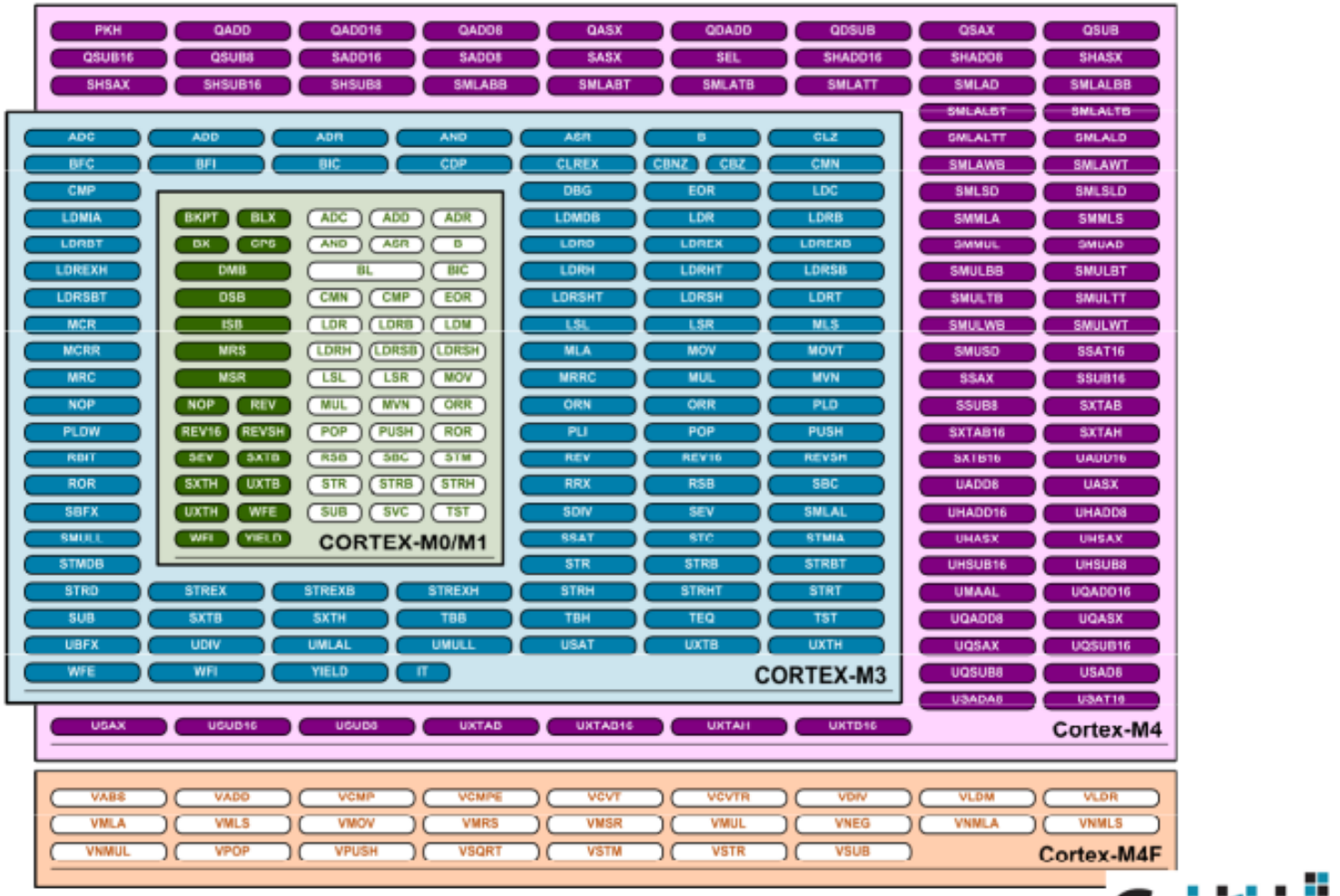
Fast response

Optimised for
high performance,
hard real-time applications

Cortex - M

Smallest/lowest power

Optimised for
discrete processing and
microcontrollers



Cortex™

Low-Power Leadership from ARM

ARM Cortex-M4

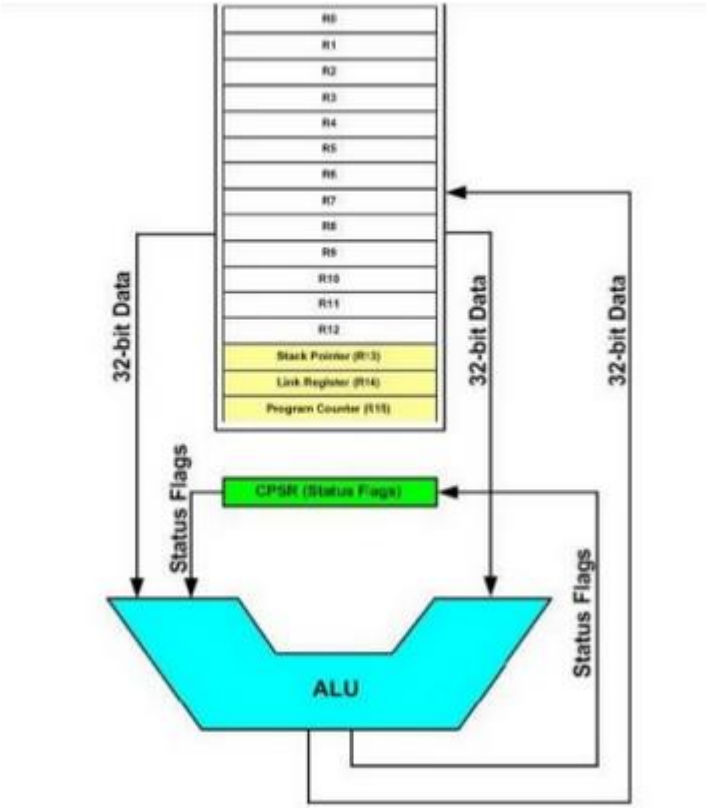
- Cortex-M4 Processor
  - Introduced in 2010
  - Designed with a large variety of highly efficient signal processing features
  - Features extended single-cycle multiply accumulate instructions, optimized SIMD arithmetic, saturating arithmetic and an optional Floating Point Unit.
- High Performance Efficiency
  - 1.25 DMIPS/MHz (DhrystoneMillion 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

Cortex-M4 Processor Features

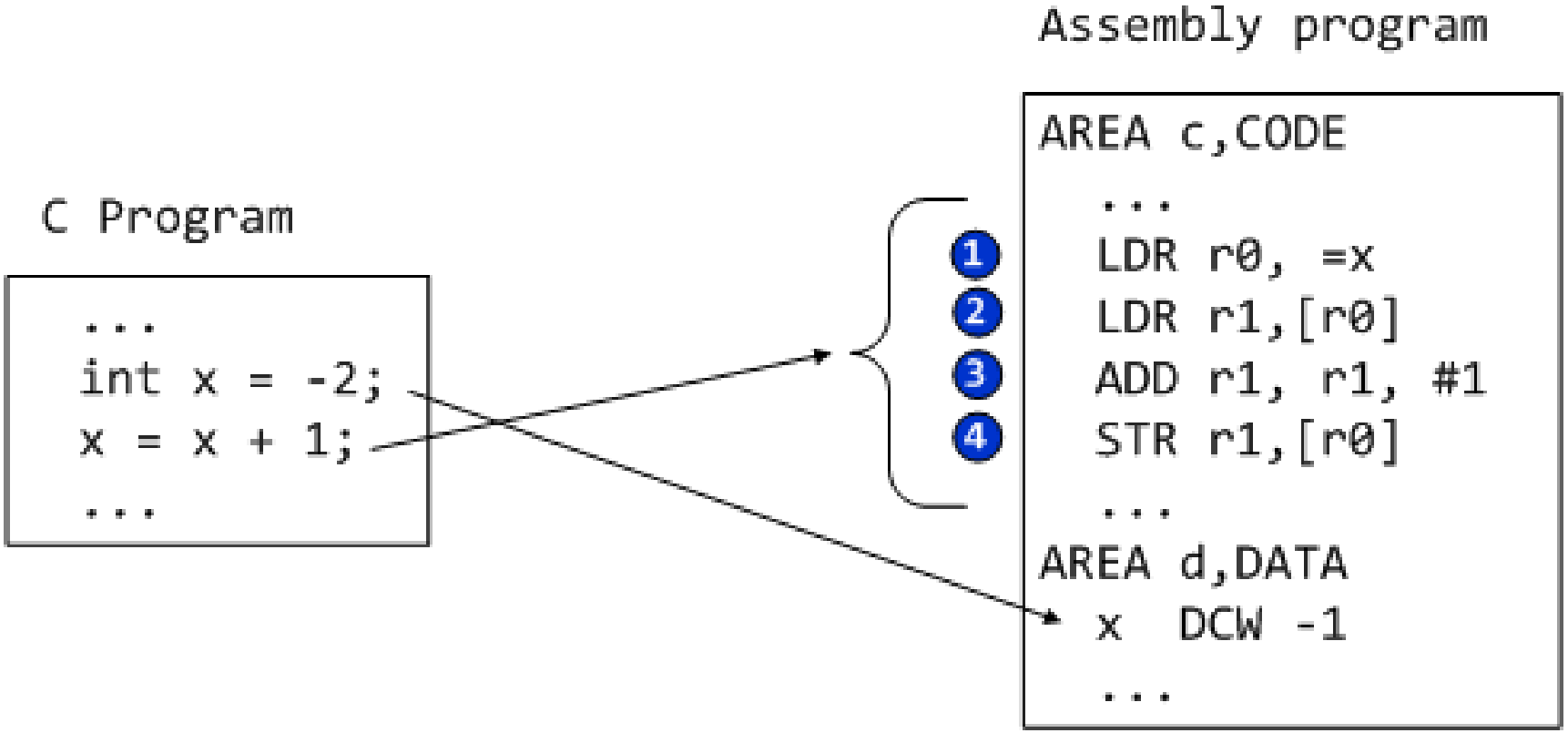
- 32-bit Reduced Instruction Set Computing (RISC) processor
  - fixed instruction size.
    - It makes the task of instruction decoder easier.
    - In Cortex M4 the instructions are 2 or 4 bytes.
  - reduced number of *simple* instructions
    - Pros: Reduces the number of used transistors
    - Cons: Can make the assembly programming more difficult , Can lead to using more memory
  - Limited number of addressing modes
    - Advantage: Hardwiring from instruction register to datapath
    - Disadvantage: Can make the assembly programming more difficult

Cortex-M4 Processor Features

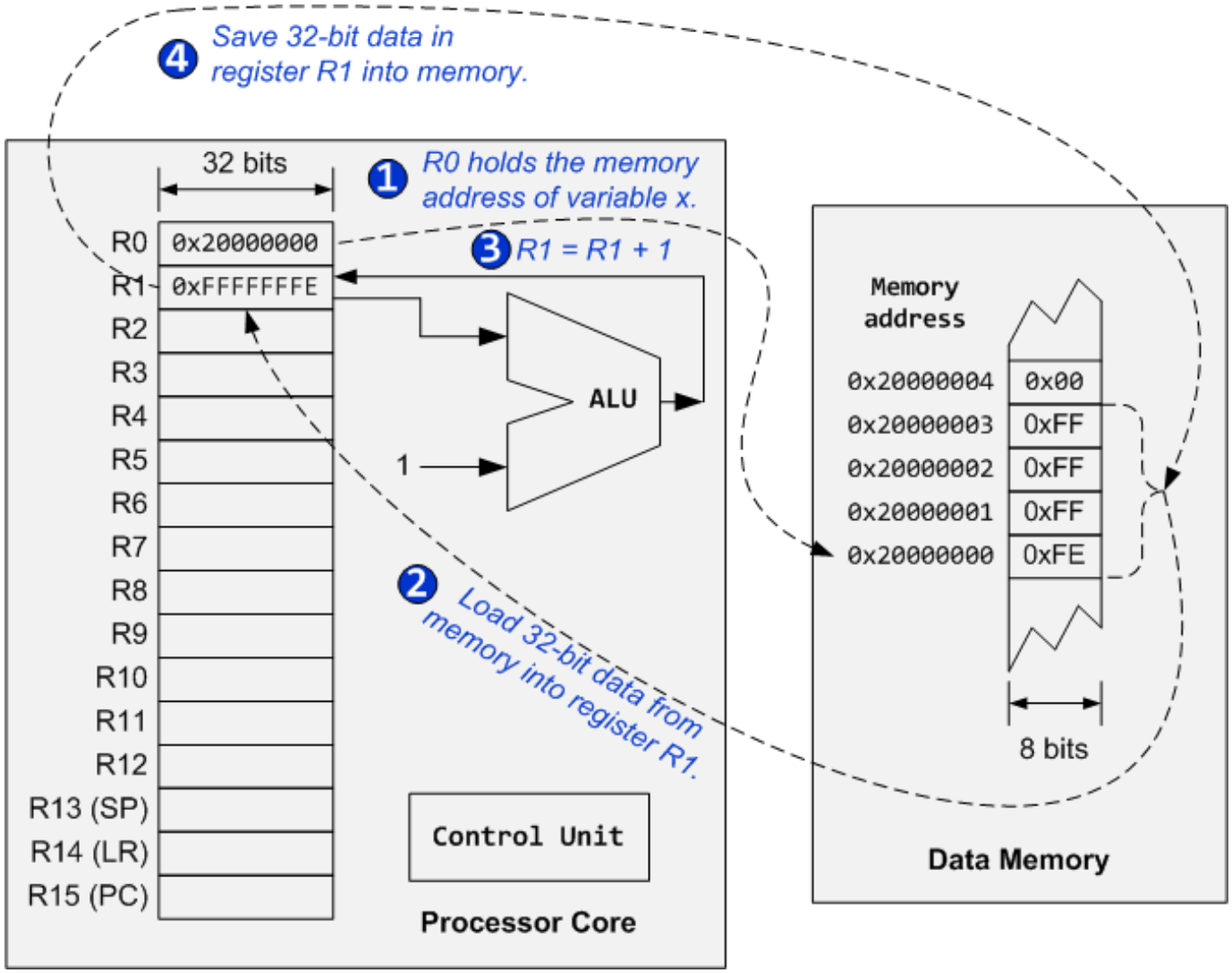
– Load/Store architecture
  - ALU operations are simple and are always between registers
  - Use simple dedicated load/store instructions for register-memory transfers (which should be less frequent than ALU operations)



Load-Modify-Store

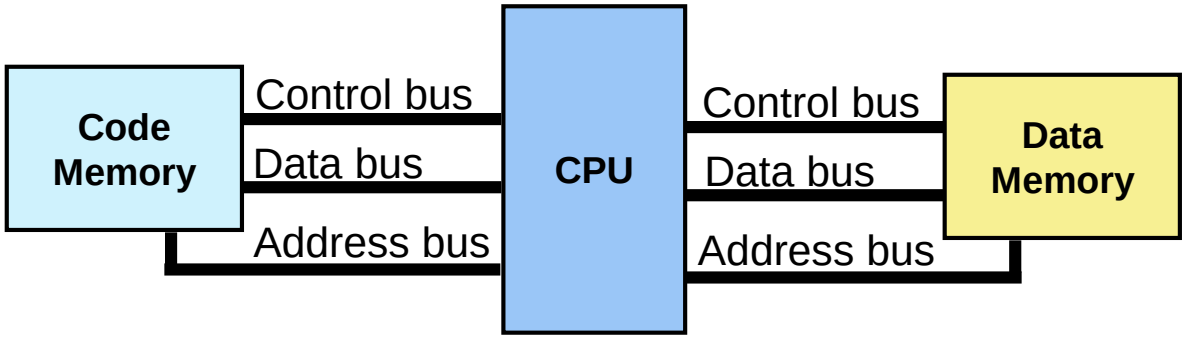


Load-Modify-Store

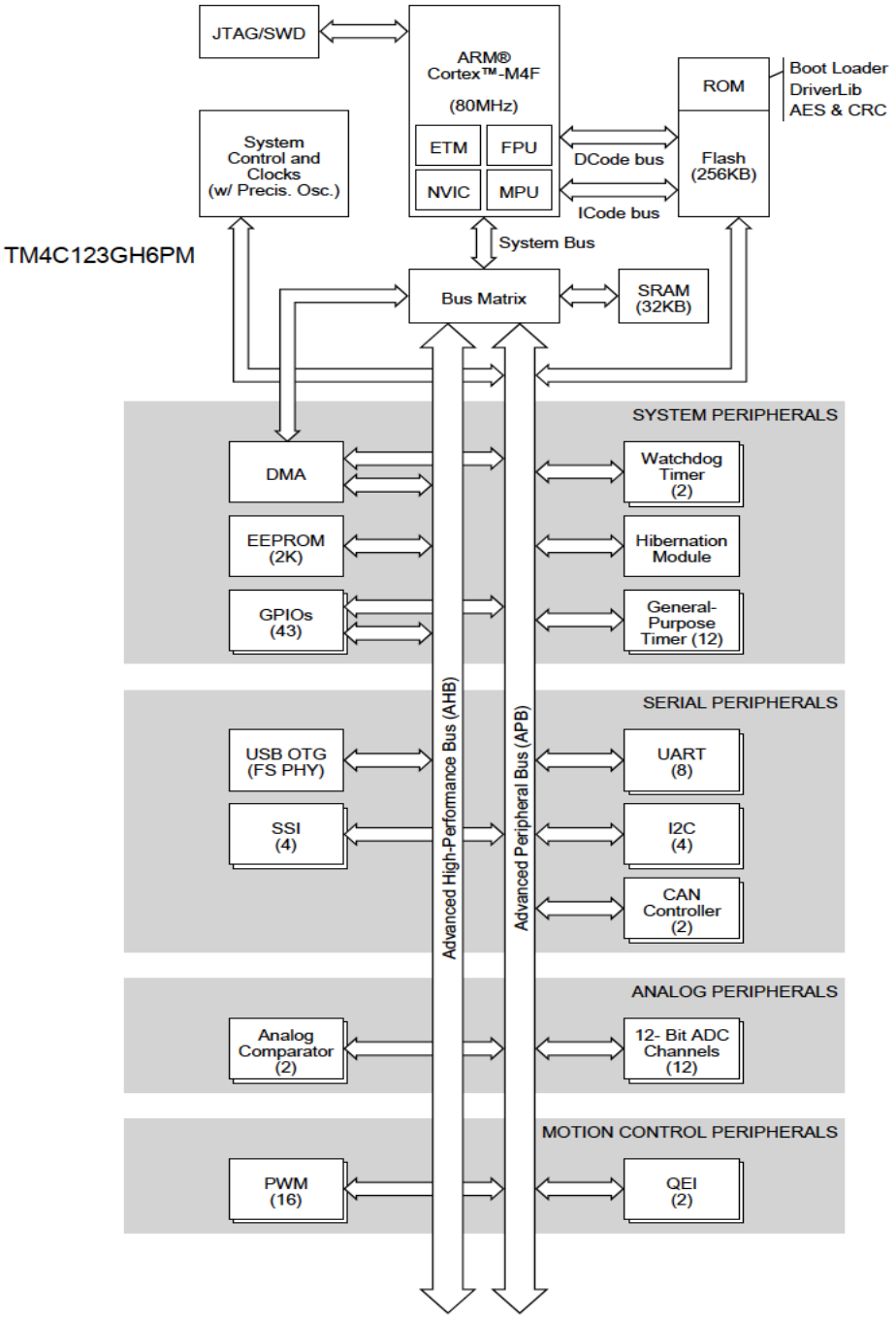


Cortex-M4 Processor Features

- 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



Bus structure



- The bus interfaces on the Cortex-M processors are 32-bit
- based on the Advanced Microcontroller Bus Architecture (AMBA) standard. AMBA contains a collection of several bus protocol specifications.
- The main bus interface protocol used by the Cortex-M3 and M4 processors is the AHB Lite (Advanced High-performance Bus), which is used in program memory and system bus interfaces.
- The Bus Matrix partitions memory access via the AHB and PPB (Private Peripheral Bus) buses.

Four external *Advanced High-performance Bus* (AHB)-Lite

- ICode memory interface:
  - Instruction fetches from Code memory space, 0x00000000 to 0x1FFFFFFF, are performed over this 32-bit AHB-Lite bus.
  - The Debugger cannot access this interface.
  - All fetches are word-wide.
  - The number of instructions fetched per word depends on the code running and the alignment of the code in memory.

- DCode memory interface:
  - Data and debug accesses to Code memory space, 0x00000000 to 0x1FFFFFFF, are performed over this 32-bit AHB-Lite bus.
  - Core data accesses have a higher priority than debug accesses on this bus.
  - DCode has a higher priority than ICode.

- System interface:
  - Instruction fetches, and data and debug accesses, to address ranges

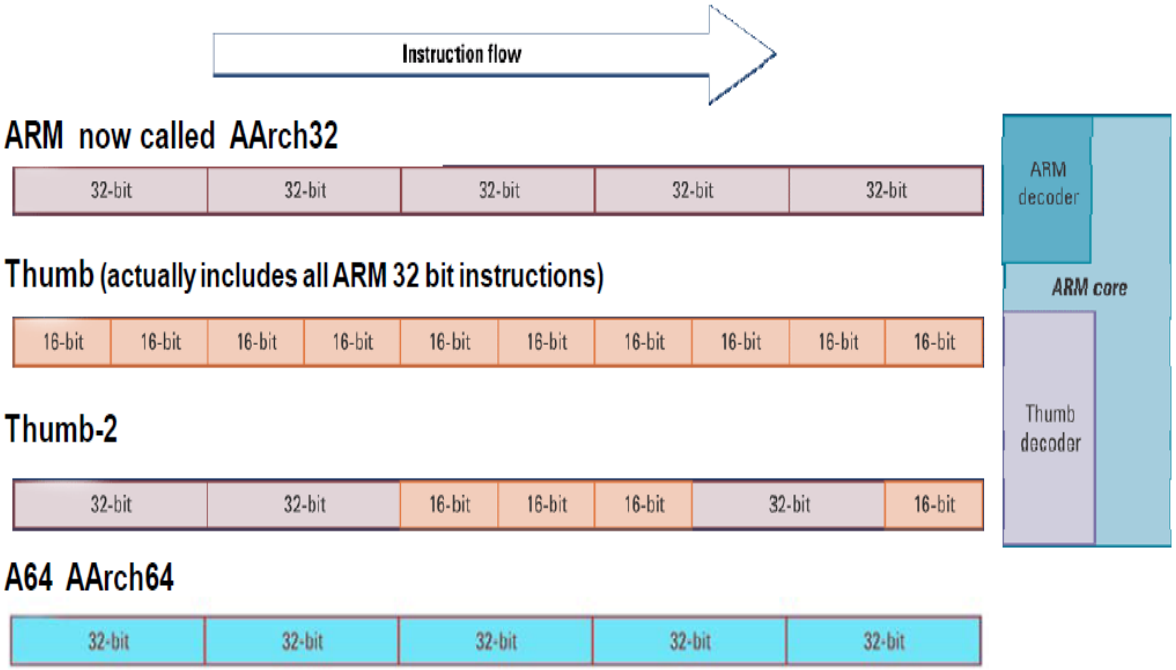
    0x20000000 to 0xDFFFFFFF and

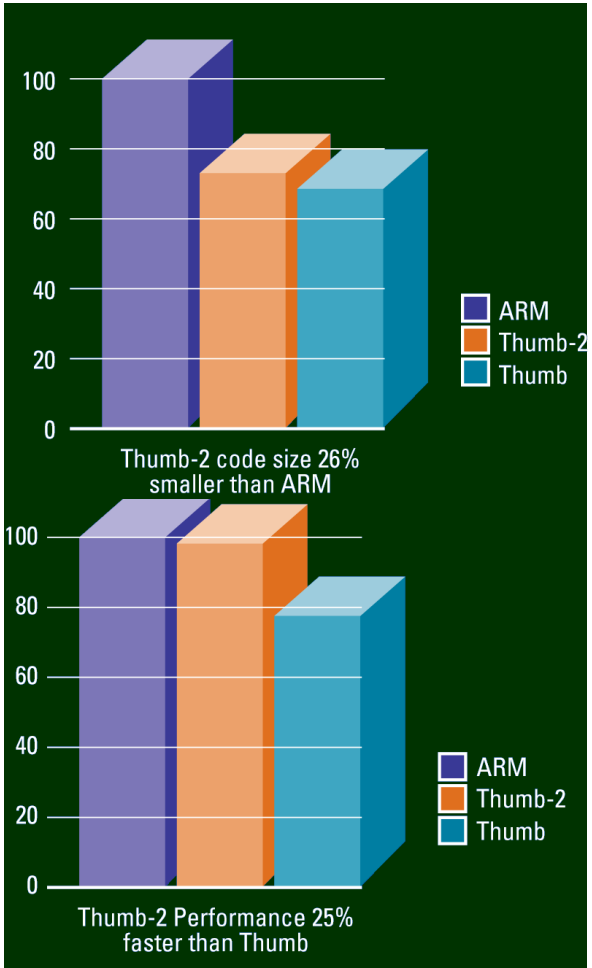
    0xE0100000 to 0xFFFFFFFF are performed over this 32-bit AHB-Lite bus.
  - For simultaneous accesses to this bus, the arbitration order in decreasing priority is:
    - data accesses
    - instruction and vector fetches
    - debug.

- Private Peripheral Bus (PPB):
  - Data and debug accesses to external PPB space, 0xE0040000 to 0xE00FFFFF, are performed over this 32-bit bus.
  - The *Trace Port Interface Unit* (TPIU) and vendor specific peripherals are on this bus.
  - Core data accesses have higher priority than debug accesses.
  - Only the address bits necessary to decode the External PPB space are supported on this interface.

Cortex-M4 Processor Features

- Instruction set
  - Include the entire Thumb®-1 (16-bit) and Thumb®-2 (16/ 32-bit) instruction sets



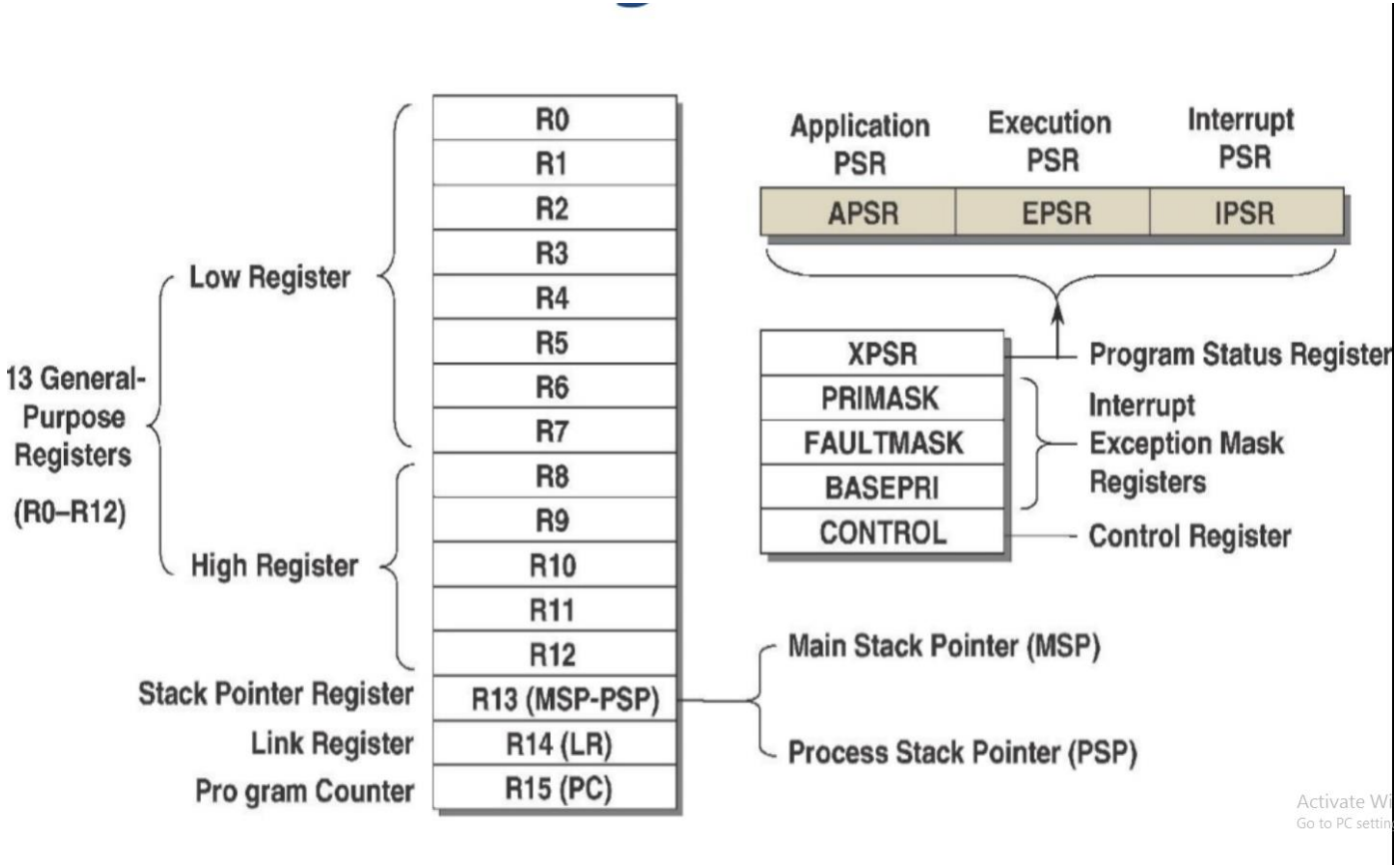


Cortex-M4 Processor Features

- Performance efficiency
  - 1.25 –1.95 DMIPS/MHz (DhrystoneMillion Instructions Per Second / MHz)
- Supported Interrupts
  - Non-maskableInterrupt (NMI) + 1 to 240 physical interrupts
  - 8 to 256 interrupt priority levels
- Enhanced Instructions
  - Hardware Divide (2-12 Cycles)
  - Single-Cycle 16, 32-bit MAC, Single-cycle dual 16-bit MAC
  - 8, 16-bit SIMD arithmetic
- Debug
  - Optional JTAG & Serial-Wire Debug (SWD) Ports
  - Up to 8 Breakpoints and 4 Watchpoints

Cortex-M4 Processor Features

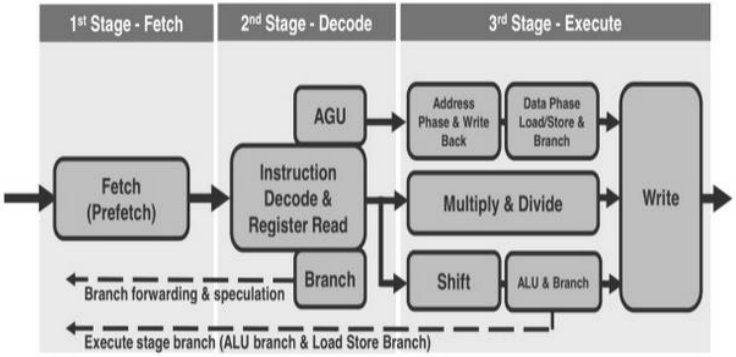
- High user accessible register count
  - RISC processors have at least 32 registers.
  - Decreases the need for stack and memory usages.

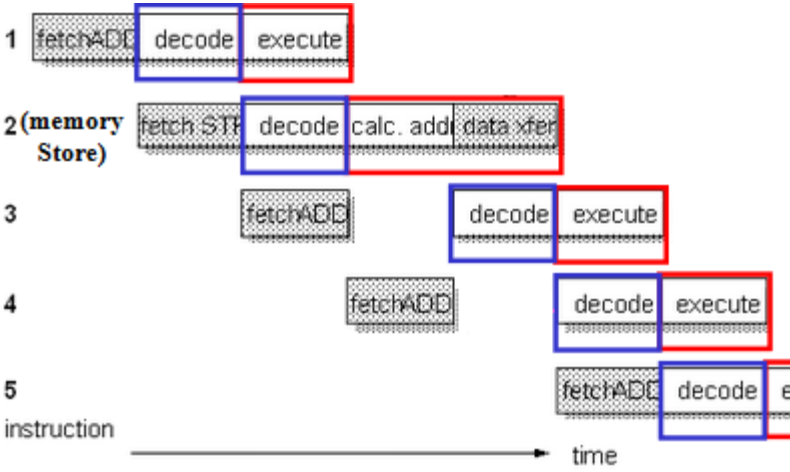


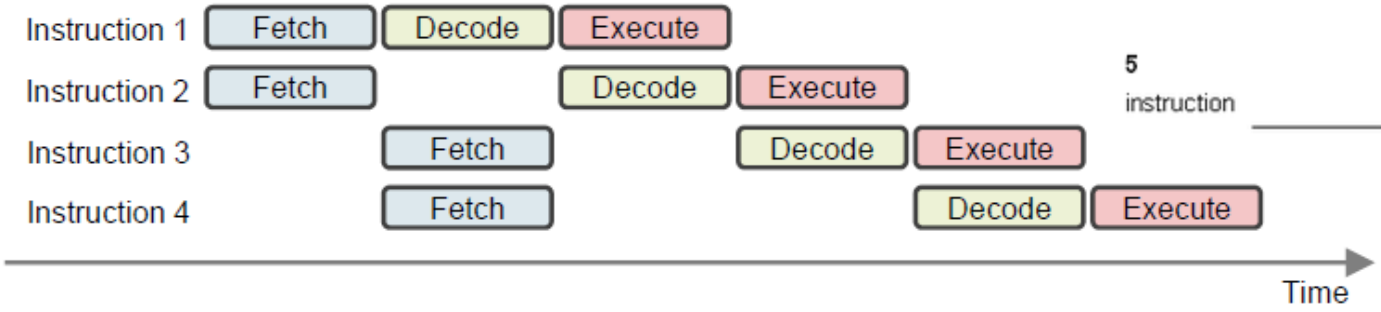
Cortex-M4 Processor Features

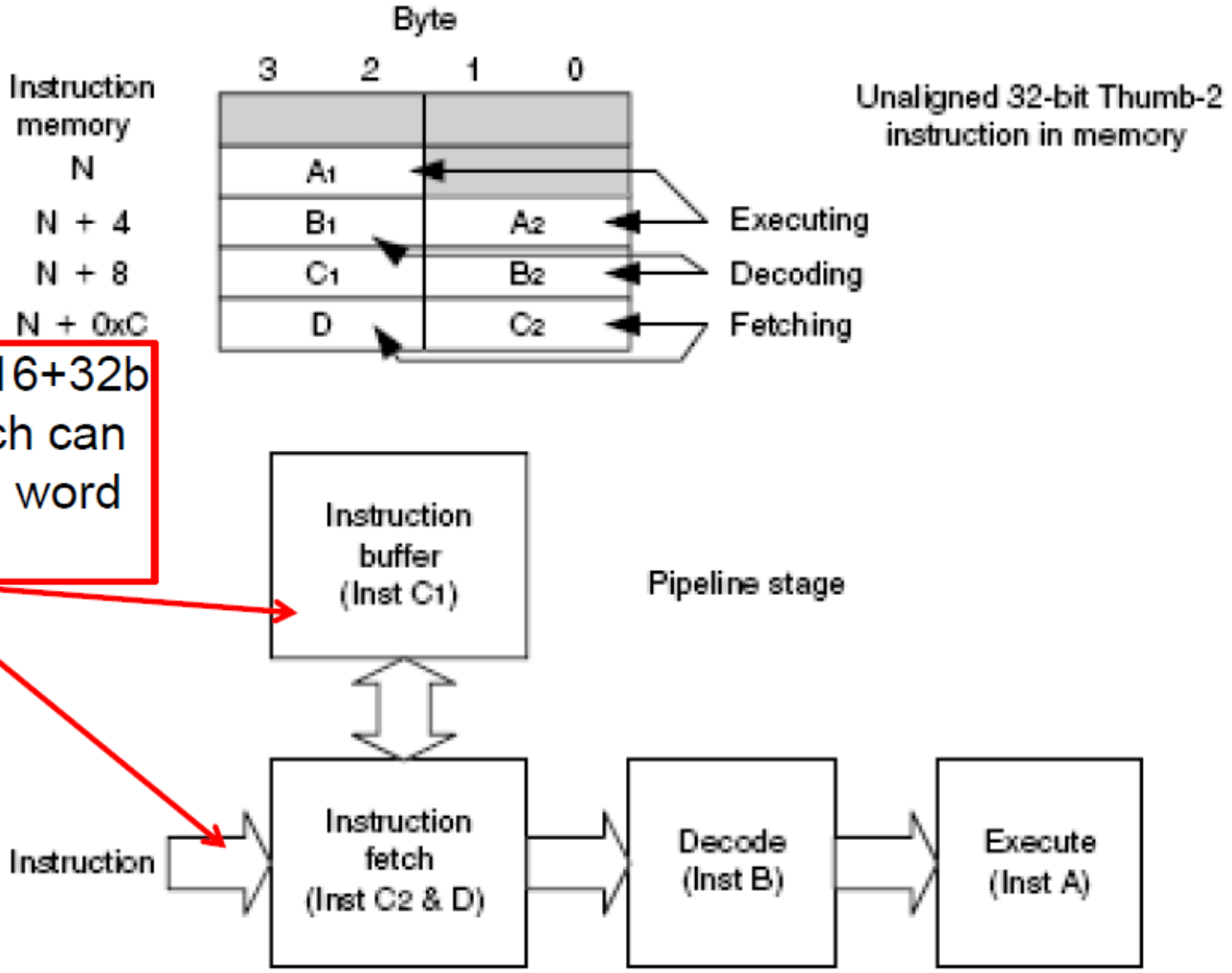
3-stage + branch speculation pipeline

- Three-stage pipeline: fetch, decode, and execute
- Because of its pipeline architecture, an instruction fetch and a data access are performed at the same time.
- Since the buses are separated, the accesses are not inter
- 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)








Byte
3
2
1
0
Instruction memory
N
N + 4
N + 8
N + 0xC
A1
B1
A2
C1
B2
D
C2
Executing
Decoding
Fetching
Unaligned 32-bit Thumb-2 instruction in memory
Handles mix of 16+32b instructions which can be misaligned in word address
Instruction buffer (Inst C1)
Pipeline stage
Instruction
Instruction fetch (Inst C2 & D)
Decode (Inst B)
Execute (Inst A)

Operating Modes

- Thread Mode for User tasks
- Handler Mode for OS tasks and exceptions