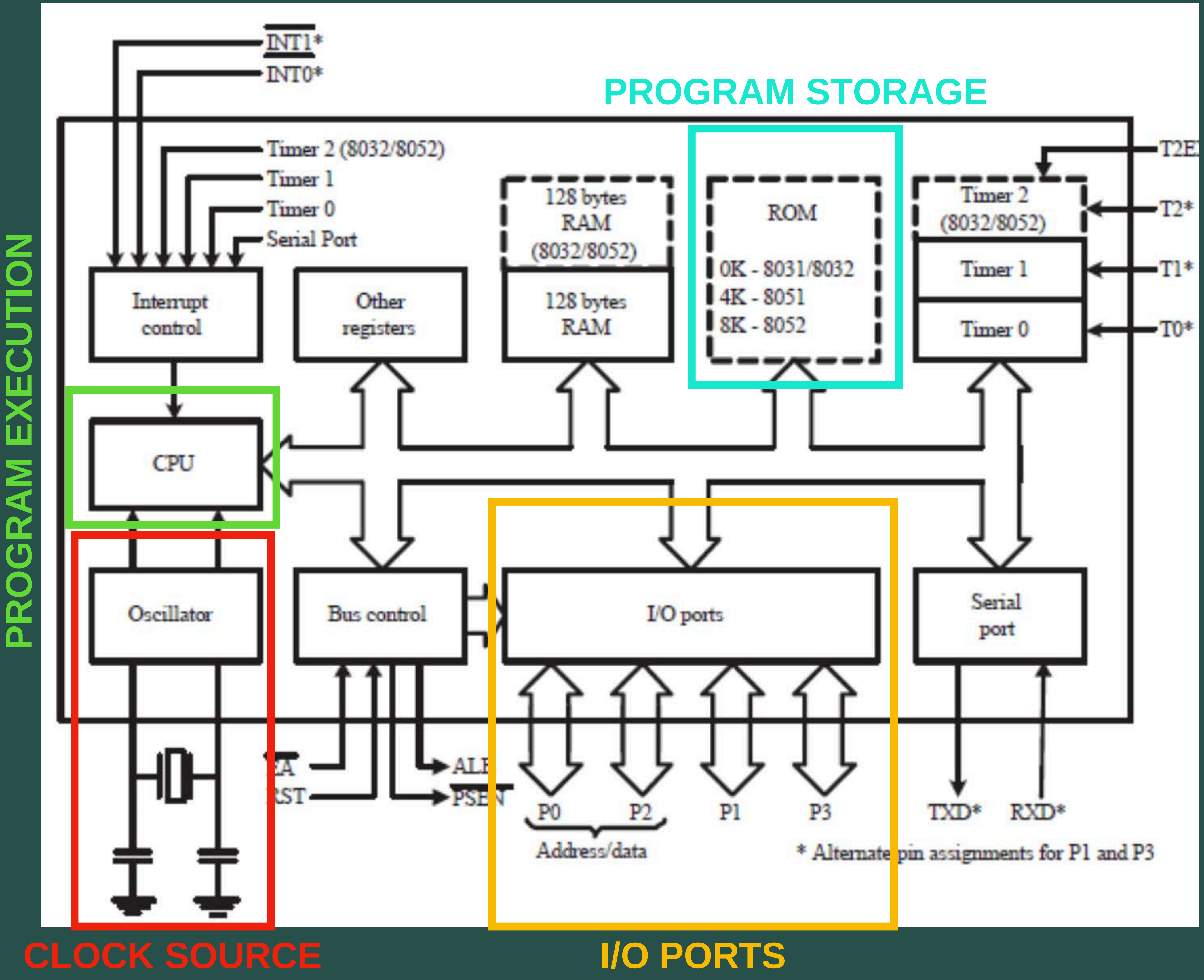
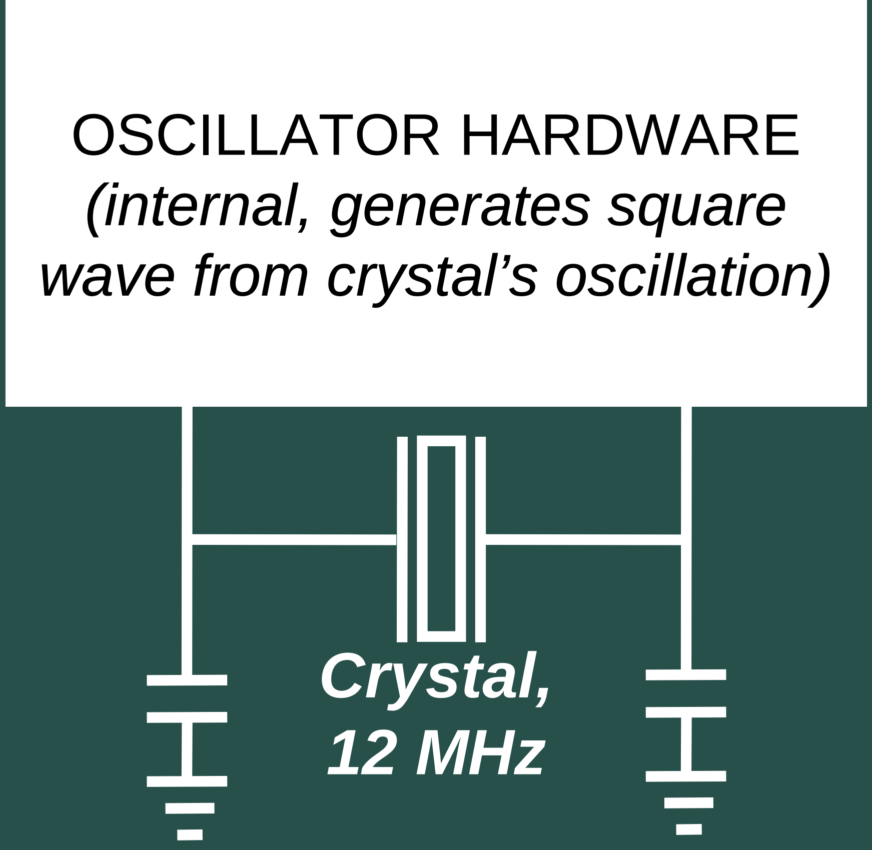
8051: BLOCK DIAGRAM



CLOCK SOURCE

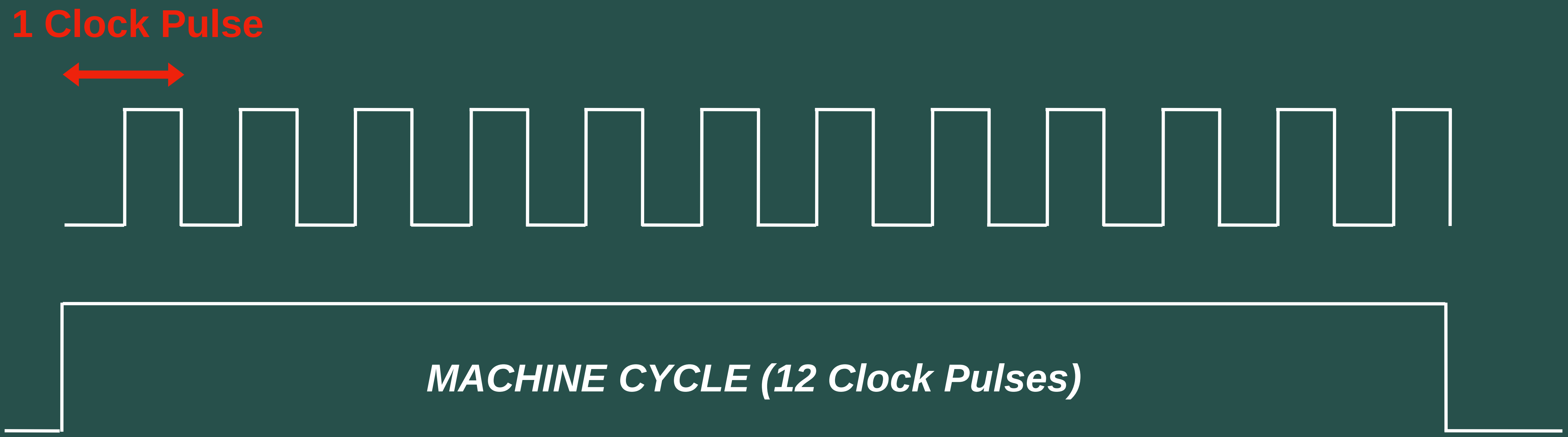
- 8051 programs are run sequentially, one instruction executing after the previous one.
  - A clock source is needed to increment a program counter (discussed later).
  - Other functions within the microcontroller (timers, communications, etc.) need to operate synchronously as well.
    - To make sure that all of these operate synchronously, the 8051 uses a master oscillator.
- The 8051's oscillator outputs a square wave. This square wave's period is determined by a relatively stable crystal (external).


OSCILLATOR HARDWARE *(internal, generates square wave from crystal's oscillation)*

Crystal, 12 MHz

- Many modern processors can execute one instruction per clock cycle. (or more!)
  - The 8051 requires 12 clock cycles per "machine cycle".
    - Instructions on the 8051 require 1 or 2 machine cycles
  - Given a 12 MHz clock, this means that we can execute 0.5-1 million instructions per second (MIPS).

MACHINE CYCLE



1 Clock Pulse

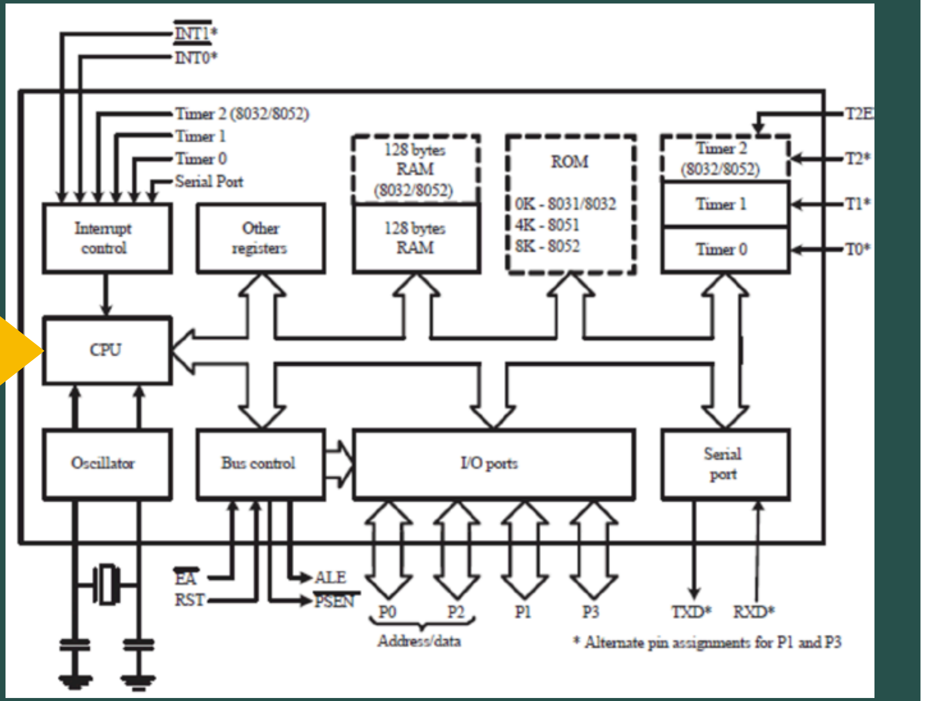
MACHINE CYCLE (12 Clock Pulses)

Given a 40 MHz Crystal, find the time required for one machine cycle.

40 MHz / 12 = 3.33 MHz
1/3.33 MHz = 0.30 µs

The oscillator generates clock pulses which are converted to machine cycles (1/12 clock pulses).
The CPU (etc.) is sequenced by these machine cycles.

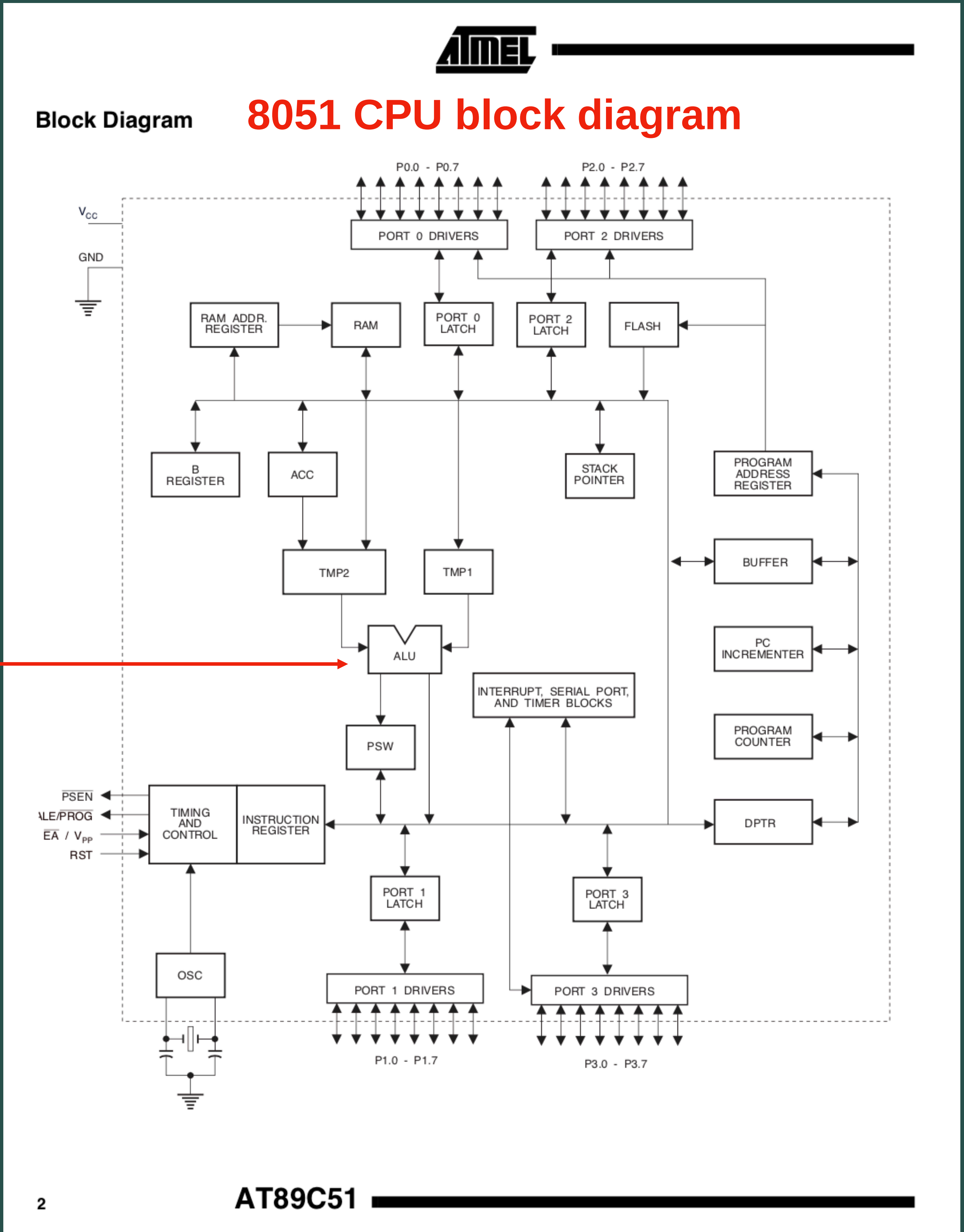
8051 CPU



ALU: Arithmetic Logic Unit

Processes the values of TMP1 and TMP2 registers.

Register: An accessible amount of storage that may be quickly acted upon by the CPU

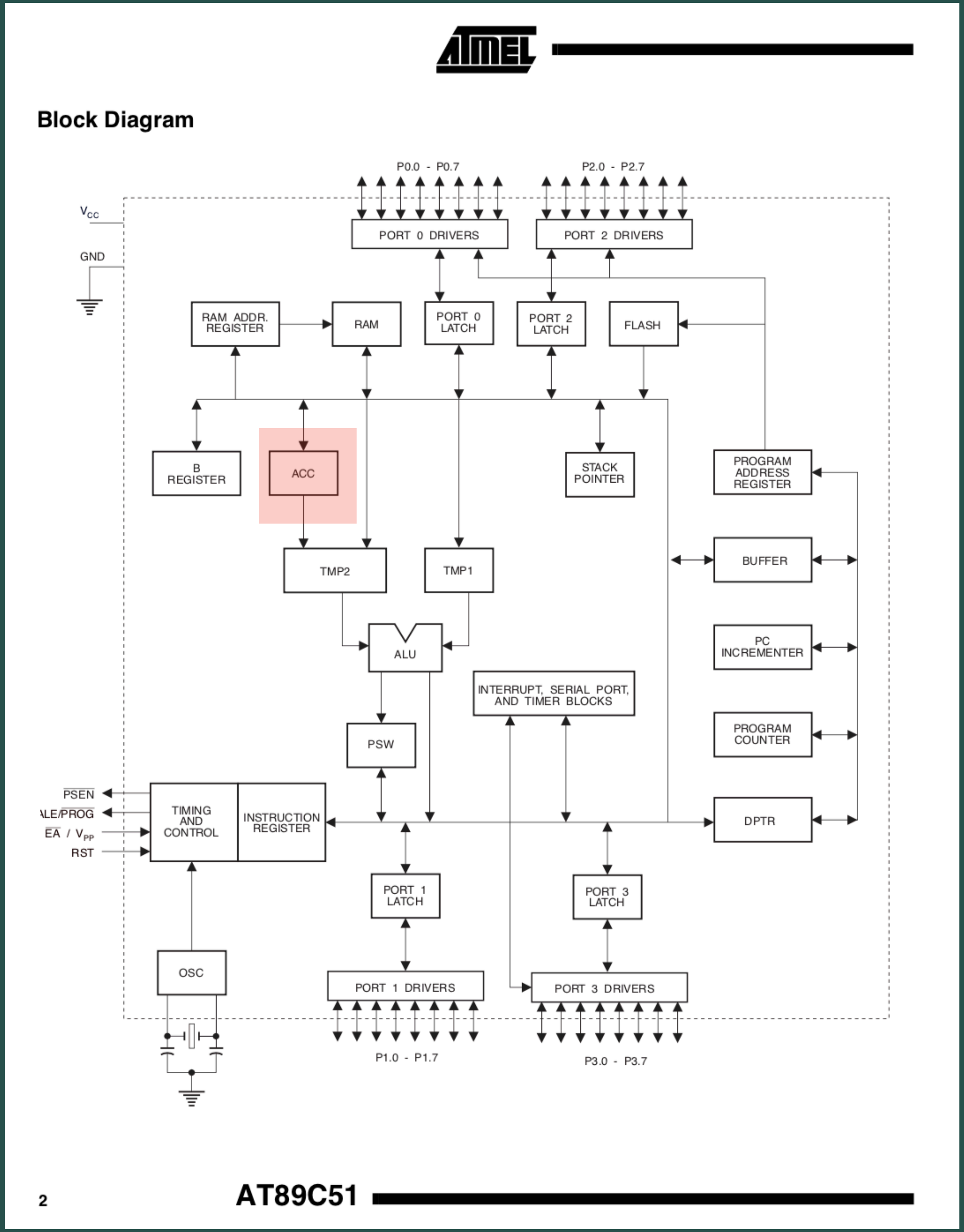


Block Diagram

8051 CPU block diagram

2 **AT89C51**

8051 ACCUMULATOR

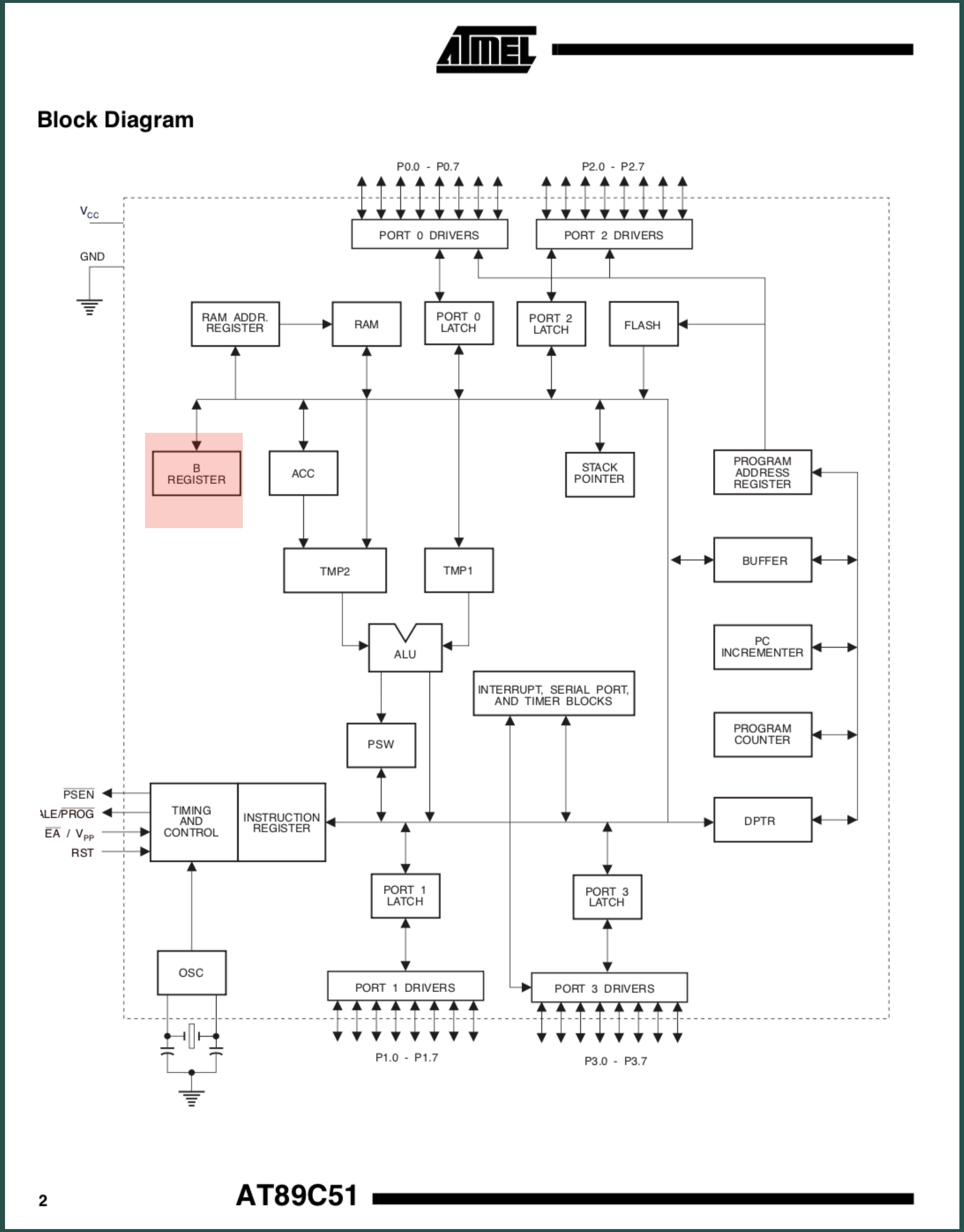


Block Diagram

2 AT89C51

- ACCUMULATOR (ACC)
  - Often referred to as Register A
  - Results of ALU operations are stored to this register.
  - 8-bit (1 byte)
  - Many, many instructions on the 8051 make use of the A register.
  - Values in the accumulator may be further processed (sent back to the ALU, etc.)
    - The accumulator value may also be stored to other memory locations.
      - Very common!

8051 B REGISTER

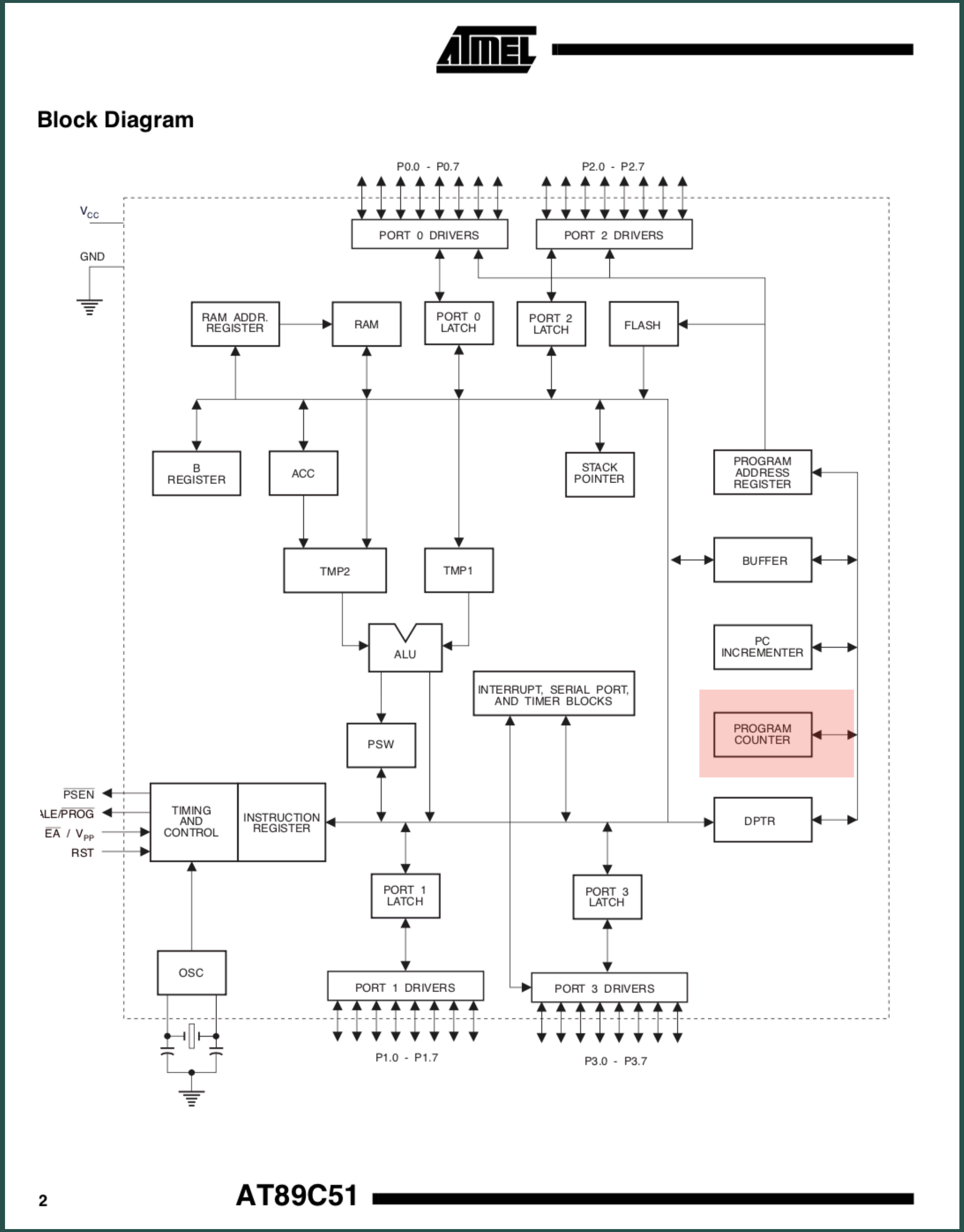


Block Diagram

2 AT89C51

- B REGISTER
  - Like the A register (ACC), B Register is 8-bit.
  - Used by only two instructions (MUL AB and DIV AB)
    - Multiply the values of A and B registers
    - Divide the values of A and B registers
  - The B register may also be used as a more general purpose storage register.

8051 PROGRAM COUNTER

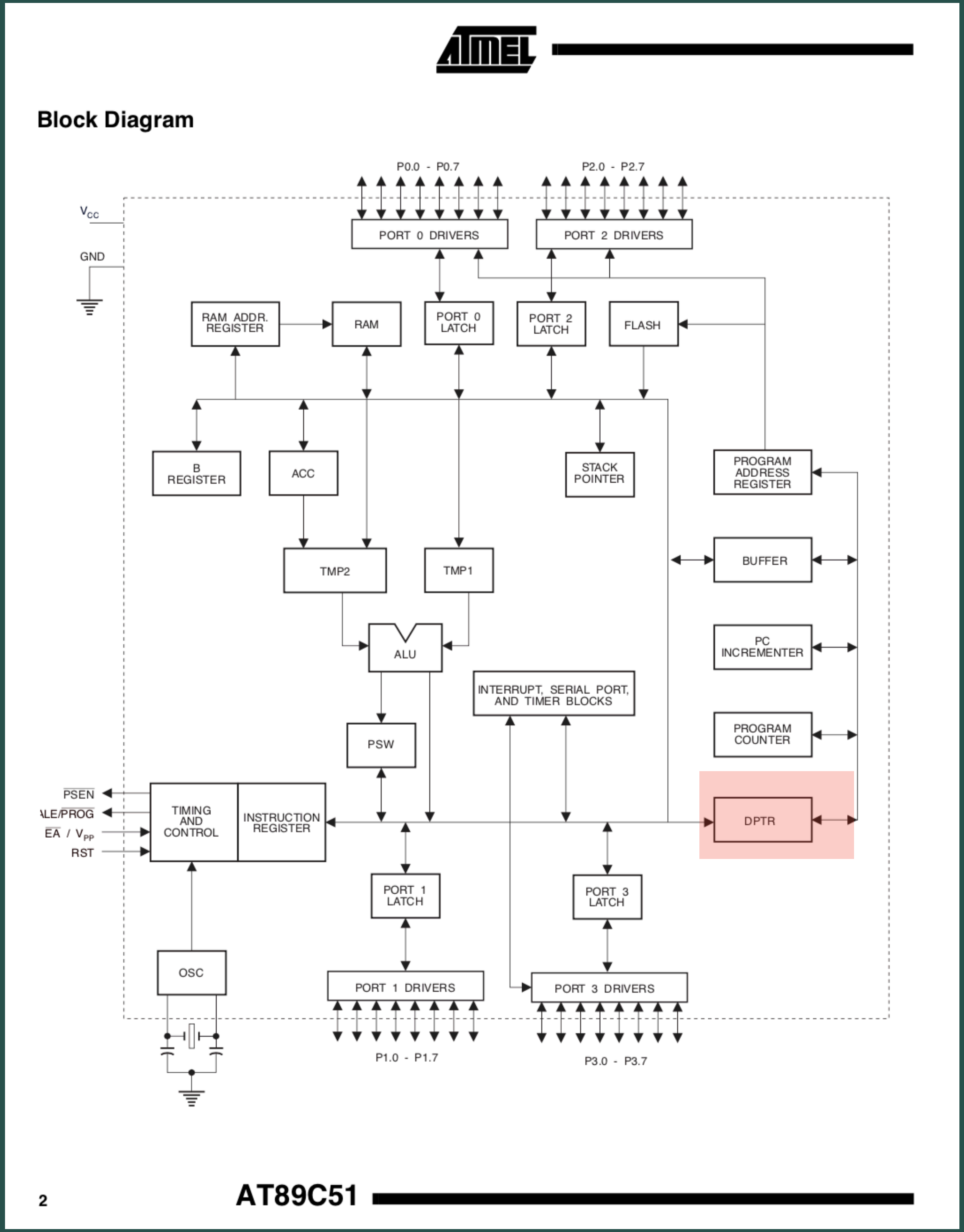


Block Diagram

2 AT89C51

- PROGRAM COUNTER (PC)
  - Unlike previously discussed registers, this is 2-byte (16 bit)
    - 0000 0000 0000 0000
  - The program counter holds the address of the next instruction to be executed.
    - When the 8051 is booted up, by default the program counter is set to 0x 00 00
    - If an instruction makes use of 1 byte, the PC is incremented by 1.
      - Some instructions require 2 (or 3) bytes, which result in the PC incrementing by 2 or 3 to move past these to the next instruction.

8051 DATA POINTER

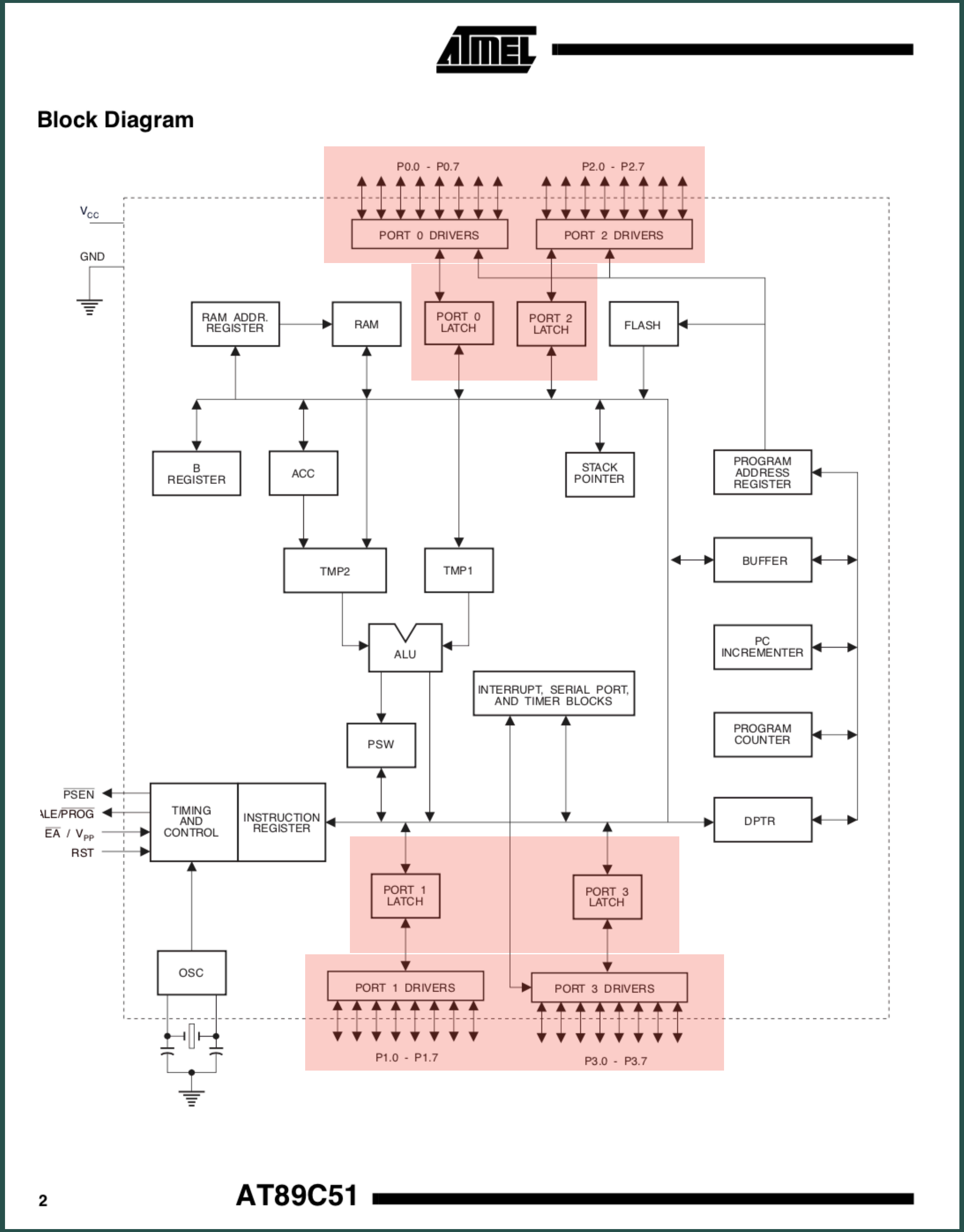


Block Diagram

2 AT89C51

- DATA POINTER (DPTR)
  - User-accessible 2-byte register
- Typically used to access external memory.
  - 2^16 bits of memory may be addressed.
  - The DPTR may also be used to store 2-byte values conveniently.

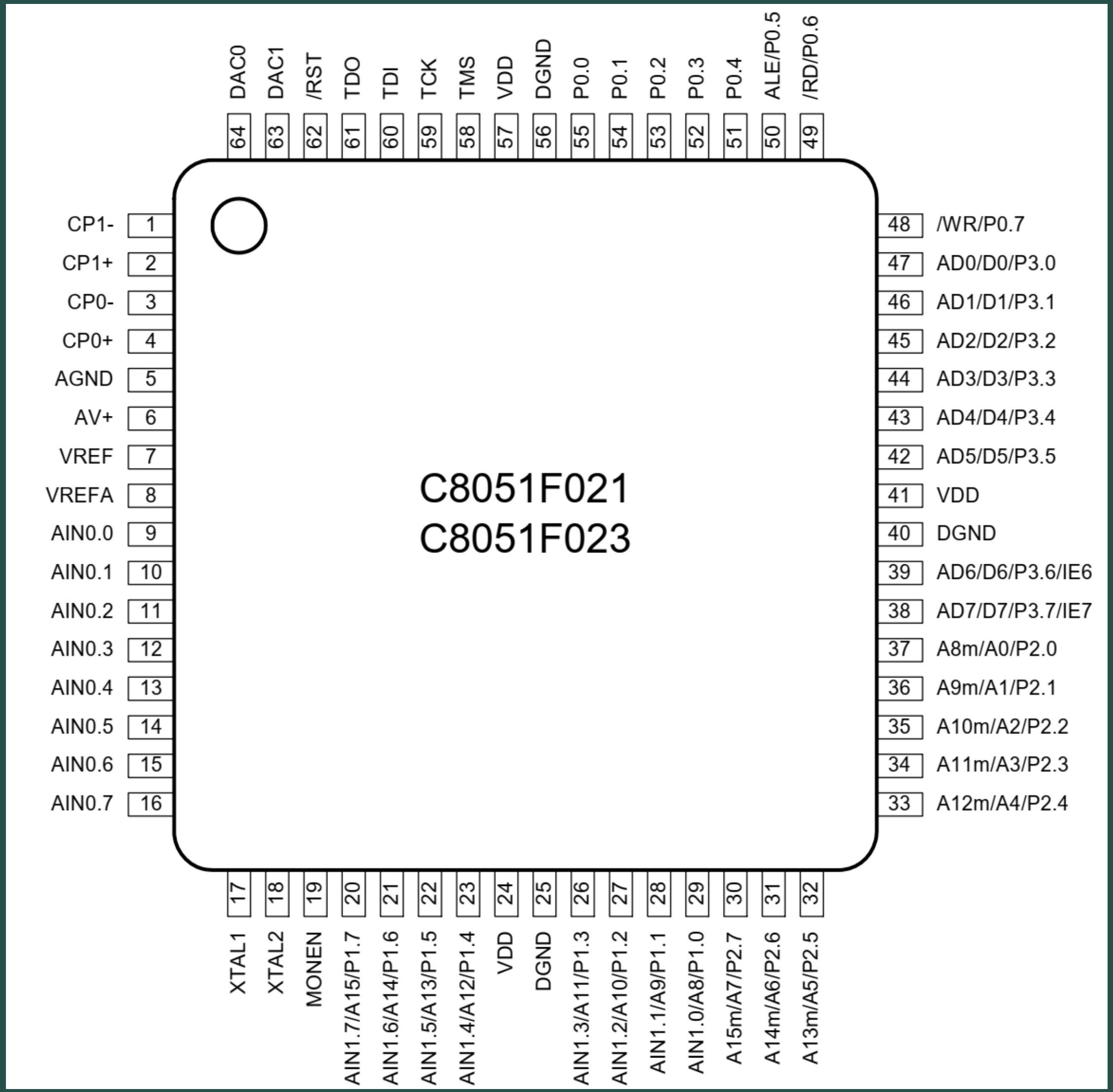
8051 INPUT/OUTPUT PORTS

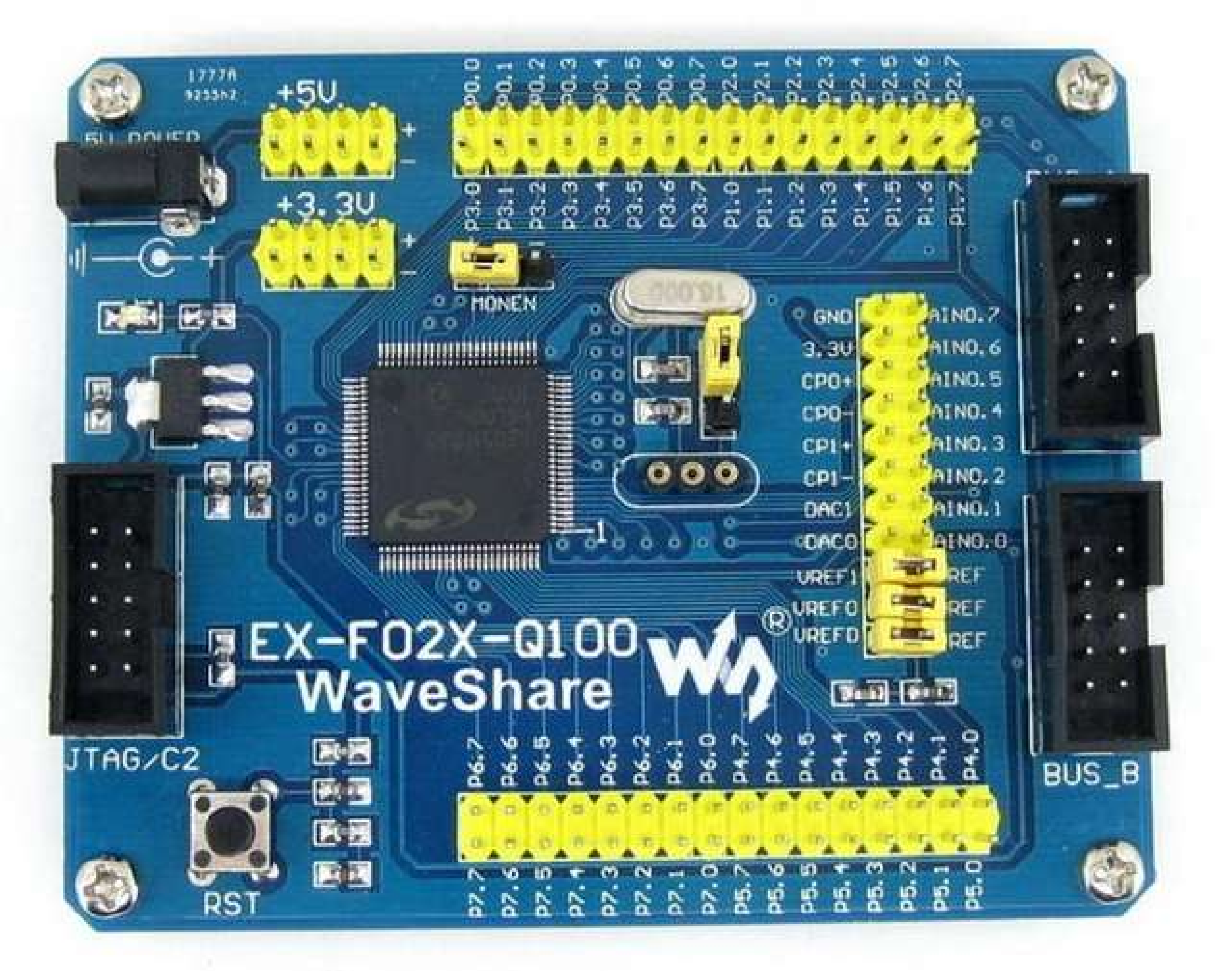


Block Diagram

2 AT89C51

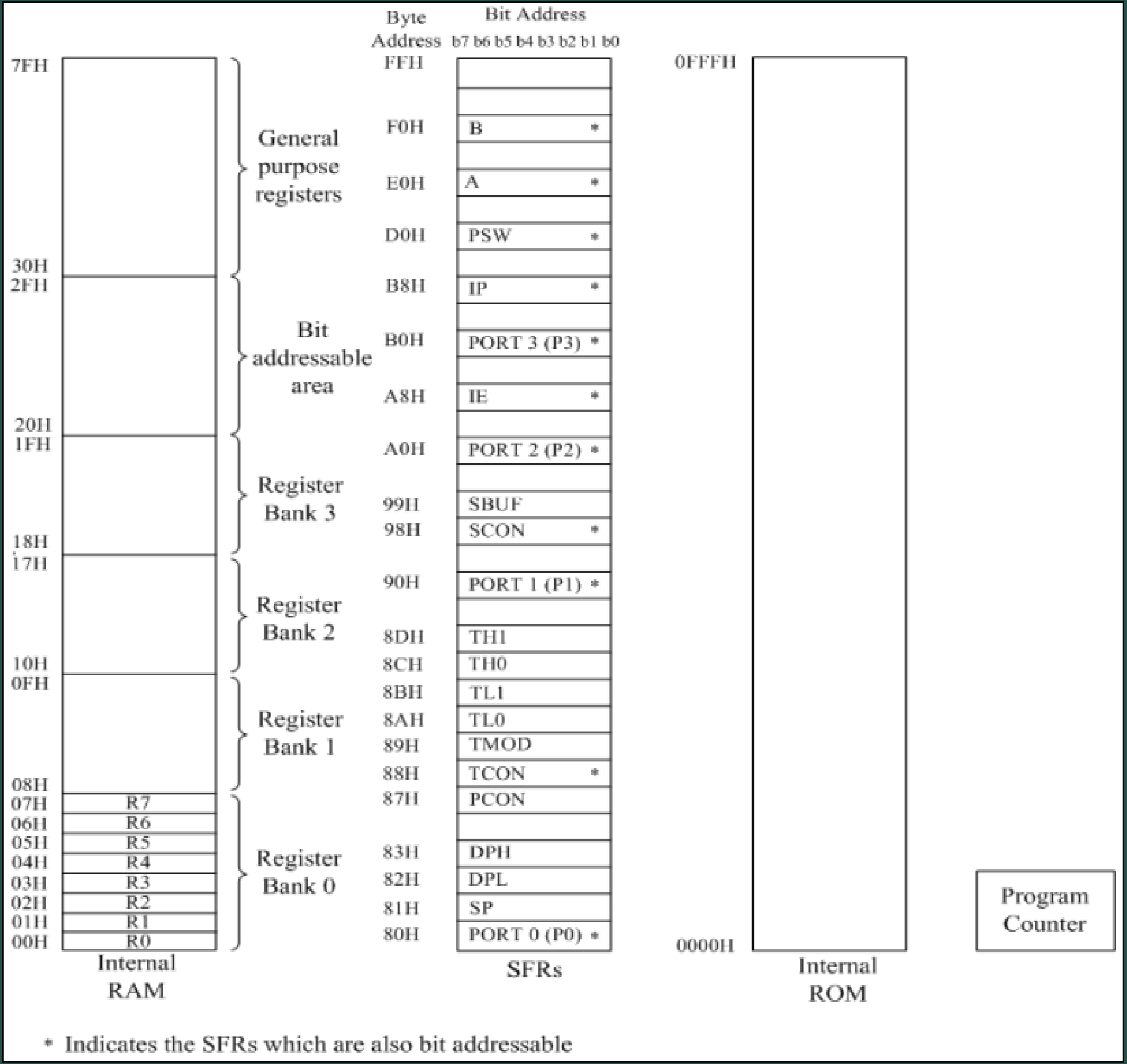
- Microcontrollers are all about interfacing with the world around them.
  - They typically feature an extensive number of I/O ports.
  - The 8051 features 4 8-bit I/O ports.
    - P0, P1, P2, P3
    - These pins default to input; you must specify if you wish them to be set up as outputs.
      - This is done by simply writing to the port.


C8051F021
C8051F023
1 CP1-
2 CP1+
3 CP0-
4 CP0+
5 AGND
6 AV+
7 VREF
8 VREFA
9 AIN0.0
10 AIN0.1
11 AIN0.2
12 AIN0.3
13 AIN0.4
14 AIN0.5
15 AIN0.6
16 AIN0.7
17 XTAL1
18 XTAL2
19 MONEN
20 AIN1.7/A15/P1.7
21 AIN1.6/A14/P1.6
22 AIN1.5/A13/P1.5
23 AIN1.4/A12/P1.4
24 VDD
25 DGND
26 AIN1.3/A11/P1.3
27 AIN1.2/A10/P1.2
28 AIN1.1/A9/P1.1
29 AIN1.0/A8/P1.0
30 A15m/A7/P2.7
31 A14m/A6/P2.6
32 A13m/A5/P2.5
33 A12m/A4/P2.4
34 A11m/A3/P2.3
35 A10m/A2/P2.2
36 A9m/A1/P2.1
37 A8m/A0/P2.0
38 AD7/D7/P3.7/IE7
39 AD6/D6/P3.6/IE6
40 DGND
41 VDD
42 AD5/D5/P3.5
43 AD4/D4/P3.4
44 AD3/D3/P3.3
45 AD2/D2/P3.2
46 AD1/D1/P3.1
47 AD0/D0/P3.0
48 /WR/P0.7
49 /RD/P0.6
50 ALE/P0.5
51 P0.4
52 P0.3
53 P0.2
54 P0.1
55 P0.0
56 DGND
57 VDD
58 TMS
59 TCK
60 TDI
61 TDO
62 /RST
63 DAC1
64 DAC0


EX-F02X-Q100
WaveShare
+5V
+3.3V
5V POWER
MONEN
16.000
P0.0 P0.1 P0.2 P0.3 P0.4 P0.5 P0.6 P0.7 P2.0 P2.1 P2.2 P2.3 P2.4 P2.5 P2.6 P2.7
P3.0 P3.1 P3.2 P3.3 P3.4 P3.5 P3.6 P3.7 P1.0 P1.1 P1.2 P1.3 P1.4 P1.5 P1.6 P1.7
GND
3.3V
CP0+
CP0-
CP1+
CP1-
DAC1
DAC0
VREF1
VREF0
VREFD
AIN0.7
AIN0.6
AIN0.5
AIN0.4
AIN0.3
AIN0.2
AIN0.1
AIN0.0
REF
REF
REF
JTAG/C2
RST
BUS_B
P6.7 P6.6 P6.5 P6.4 P6.3 P6.2 P6.1 P6.0 P4.7 P4.6 P4.5 P4.4 P4.3 P4.2 P4.1 P4.0
P7.7 P7.6 P7.5 P7.4 P7.3 P7.2 P7.1 P7.0 P5.7 P5.6 P5.5 P5.4 P5.3 P5.2 P5.1 P5.0

The AT89C51
micro controller core

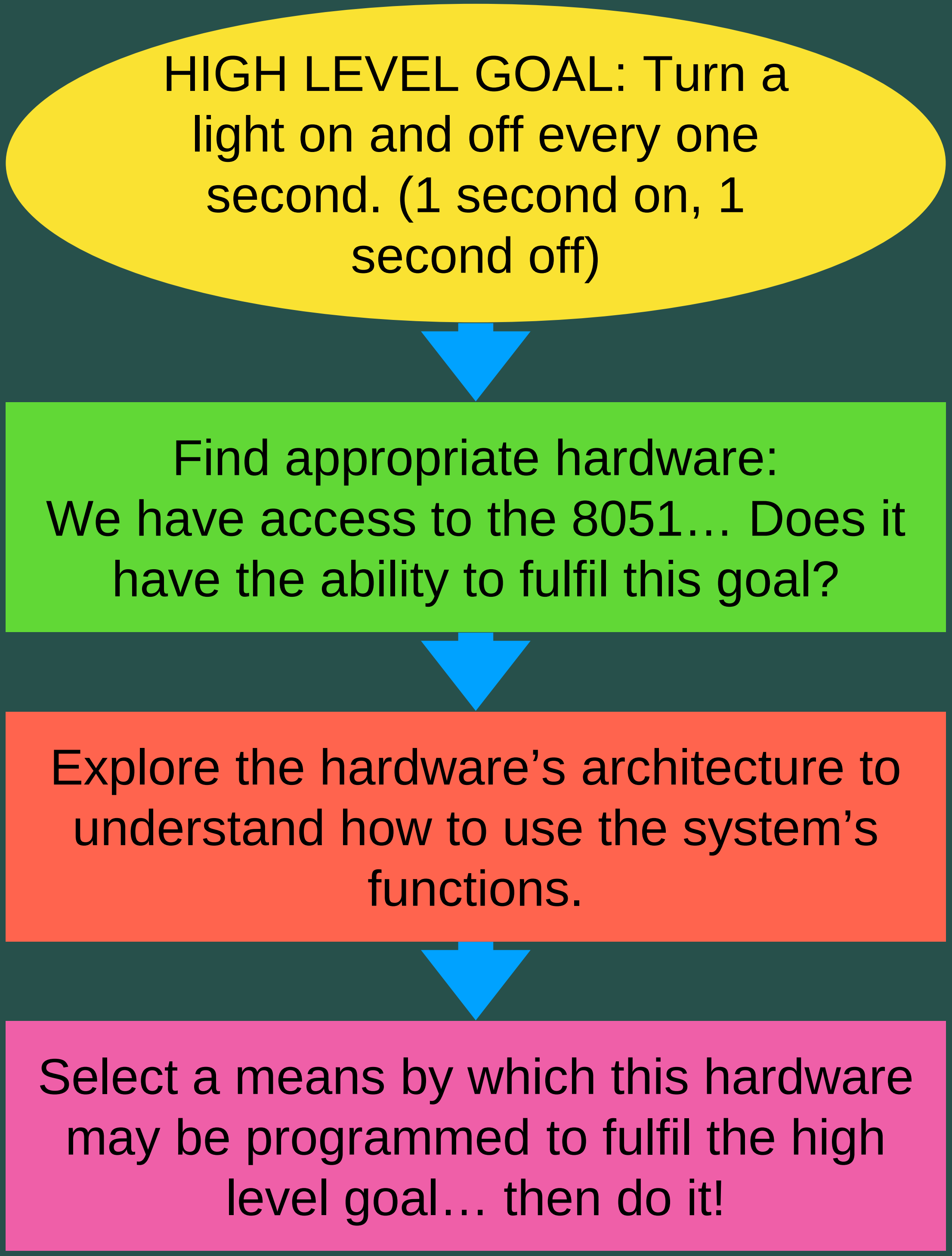
The programming model shows all the registers that are available to the software developer. Notice that the registers are all assigned addresses apart from the program counter.



* Indicates the SFRs which are also bit addressable

CONCEPTUALISING A PROGRAM

- As engineers, we're often given tasks and are expected to practically realise them.
  - "Make a robot that sweeps the floor!"
- As embedded systems engineers, we need to be able to:
  - Take a very high-level goal…
  - … and rationalise it with the tools that we have available.
- If we are told: "Turn a light on and off every one second!"…
  - … we must figure out how to do that with the available tools.



HIGH LEVEL GOAL: Turn a light on and off every one second. (1 second on, 1 second off)

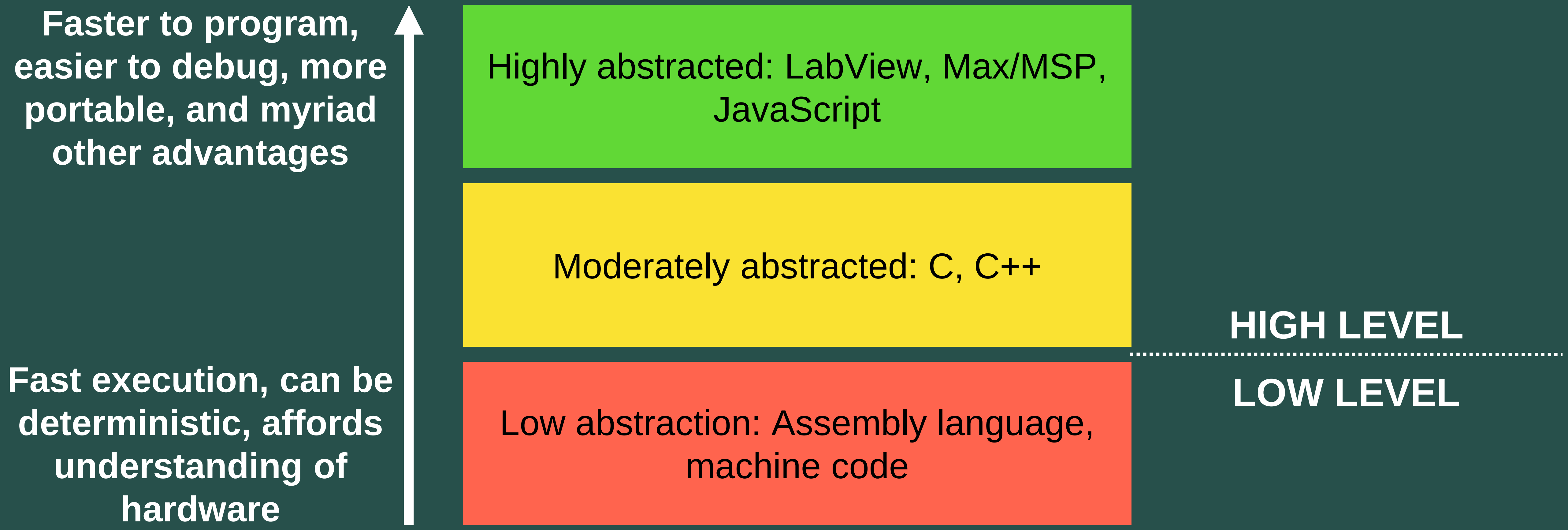
Find appropriate hardware:
We have access to the 8051… Does it have the ability to fulfil this goal?

Explore the hardware's architecture to understand how to use the system's functions.

Select a means by which this hardware may be programmed to fulfil the high level goal… then do it!

PROGRAMMING LANGUAGE HIERARCHIES

- Computer programming languages may be thought of as *high-level* or *low-level*.
  - High-level: high levels of abstraction between language and hardware.
    - C, Python, Javascript, LabView, Lua, C#, Java, Ada, Smalltalk, Swift, Forth…
      - There are varying levels of abstraction: C has less than, say, Javascript
  - Low-level: Programming commands are very closely related to the hardware’s actual operation.
    - Assembly languages are considered low-level: often one line per instruction cycle. Machine code is the lowest of all.



Faster to program, easier to debug, more portable, and myriad other advantages

Fast execution, can be deterministic, affords understanding of hardware

Highly abstracted: LabView, Max/MSP, JavaScript

Moderately abstracted: C, C++

Low abstraction: Assembly language, machine code

HIGH LEVEL

LOW LEVEL

ASSEMBLY LANGUAGES

- Every different computer architecture has unique commands that allow it to execute programs.
  - At the very lowest level, these are electrical signals that correspond to LOGIC HIGH and LOGIC LOW (0 or 1). Groups of these form instructions used by the computer.
  - Machine code might look like this: 10010111 10011101 11101110 10011111
    - This is very hard for most mortals to read, understand, debug, and expand
    - These instructions and memory locations may be represented by *mnemonics*: easier-to-read and easier-to-remember codes that represent the machine code.

MNEMONIC	MACHINE CODE (BINARY)
NOP	0000 0000
ADD A, R3	0010 1011

BINARY, HEXADECIMAL, DECIMAL

- We’re used to counting in base-10 (decimal, 0d): this is because we have 10 fingers!
  - Computers that use HIGH/LOW logic utilise a base-2 scheme: binary (0b)
  - It can get very clunky and verbose to represent digital systems using binary: you end up with lots and lots of 0’s.
  - To make things more compact, we often use hexadecimal (0x or #nH).
    - This is a counting system that goes from 0d0 to 0d16 (0x0 to 0xF)
      - 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F
      - 1 hexadecimal number takes the place of 4 binary numbers.
        - These 4 binary numbers form half of a byte, often called a ‘nibble’
        - 0b0000 0000 is the same as 0x0 0x0
- Hexadecimal is commonly used in low-level software development: machine codes and addresses are easily represented this way.

MNEMONIC	MACHINE CODE (0b)	MACHINE CODE (0x)
DIV AB	1000 0100	0x84
MOV A,#data	0111 0100	0x74

CISC VS. RISC

- Instruction set: the collected instructions (in machine code) that are executable by a CPU.
- CISC: Complex instruction set.
  - Many detailed operations that allow for single instructions to have fine-grained control over the computer.
  - Common in many early computers…
    - Good for low-level programming, as fewer commands are needed.
    - Pentium, 8051, etc. are CISC
- RISC: Reduced instruction set
  - As high-level languages became prevalent in the 1980’s and 1990’s, there was less need for very specific processor instructions.
    - Compilers could make use of a restricted (and fast!) instruction set to efficiently realise high-level programs.
    - ARM processors are the most prevalent RISC processors today.
    - Assembly language programs for CISC systems are often relatively human readable. RISC assembly language programs are significantly less so.

8051 INSTRUCTIONS

- 8051 instructions are one byte, and are often organised in a table with the high nibble on one axis and the low nibble on another: https://www.win.tue.nl/~aeb/comp/8051/set8051.html
  - In the Lab Exercise, you will find a PDF called at_c51ism.pdf (note that the axes in this document are arranged opposite to those in the above URL)
    - This contains a detailed list of the 8051's instructions. You are strongly encouraged to explore this document.
- Further resources:
  - KEIL 8051 instruction set reference: http://www.keil.com/support/man/docs/is51/is51_instructions.htm
  - Instructions organised by function: https://www.engineersgarage.com/tutorials/8051-instruction-set

8051 SELCECTED INSTRUCTIONS

- You aren’t expected to memorise the 8051’s instruction set.
  - Should the test contain instruction set-related questions, you will be provided with reference materials that you may consult during the test.

INSTRUCTION MNEMONIC	SYNTAX	Example OpCode	Number of bytes	Notes
NOP	NOP	0x00	1	Causes no operation that cycle
MUL	MUL AB	0xA4	1	Multiply accumulator by B
MOV	MOV operand1, operand2	0x74	1-3	Copies the val. of operand1 to operand2
INC	INC register	0x04	1-2	Increments the value of the register by 1