8051 FLAGS (INTRO)

- When the 8051's processor enters certain states, it raises 'flags' to indicate these states.
  - These flags are stored in the Program Status Word register (PSW), which uses six of the register's eight bits.
- We'll explore some examples of these flags.

PSW.7 CY (Carry flag, raised when the processor needs to carry in addition)	PSW.6 AC (Aux. carry, used during BCD math)	PSW.5 F0 (User-assignable flag)	PSW.4 RS1 (Register bank selector, don't worry about for now)	PSW.3 RS2 (Register bank selector, don't worry about for now)	PWS.2 OV (Overflow, raised when a signed number overflows into the sign bit)	PSW.1 - (User assignable)	PSW.0 P (Parity: 0 if acc. holds even number of 1's)

Program Status Word structure

PSW.7	PSW.6	PSW.5	PSW.4	PSW.3	PSW.2	PSW.1	PSW.0
CY	AC	F0	RS1	RS0	OV	--	P

Bit	Symbol	Flag name and description		
7	C (or CY)	Carry; Used in arithmetic, logic and Boolean operations		
6	AC	Auxiliary carry ; useful only for BCD arithmetic		
5	F0	Flag 0; general purpose user flag		
4	RS1	Register bank selection bit 1		
3	RS0	Register bank selection bit 0		
		RS1	RS0	
		0	0	Bank 0
		0	1	Bank 1
		1	0	Bank 2
		1	1	Bank 3
2	0V	Overflow; used in arithmetic operations		
1	--	Reserved; may be used as a general purpose flag		
0	P	Parity; set to 1 if A has odd number of ones, otherwise reset to 0		

A SIMPLE ASSEMBLY LANGUAGE PROGRAM

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

Let’s connect the LED to Port 1 and then toggle Port 1 between 0 and 1 every 1 ms.

The biggest challenge will probably be figuring out how to get a good precise timer to let the light stay on/off for 1 ms

```
START:
MOV A,#0FFH        ;;Move 0xFF(1) to accumulator
MOV P1,A           ;;Move accumulator value to P1

                   ;;TODO: delay for 1 ms!

MOV A,#00H         ;;Move 0x0(0) to accumulator
MOV P1,A           ;;Move accumulator value to P1

                   ;;TODO: delay for 1 ms again

SJMP START         ;;Jump back to ‘START’
```

Note: Practical 8051 assembly language programs need a few other things to get working (e.g., setting the start address, specifying when the program has ended, etc.)

SUBROUTINES

- In high-level languages, we often use functions to compartmentalise blocks of code that we might reuse.
  - This allows us to avoid copy+paste of code.
- Somewhat similar to this is the assembly language concept of subroutines
  - We can jump to particular blocks of code, execute them, and then jump back to our ‘main’ program.
    - Let’s try to do this with the 1 ms delay…

```
//pseudocode, high-level example of port writing
main(){
 Port1.write(HIGH);
 delay1Ms(); //call function routine
 Port1.write(LOW);
 delay1Ms();
}

function delay1Ms(){
 //code to make the CPU wait for 1 ms
}
```

SUBROUTINES

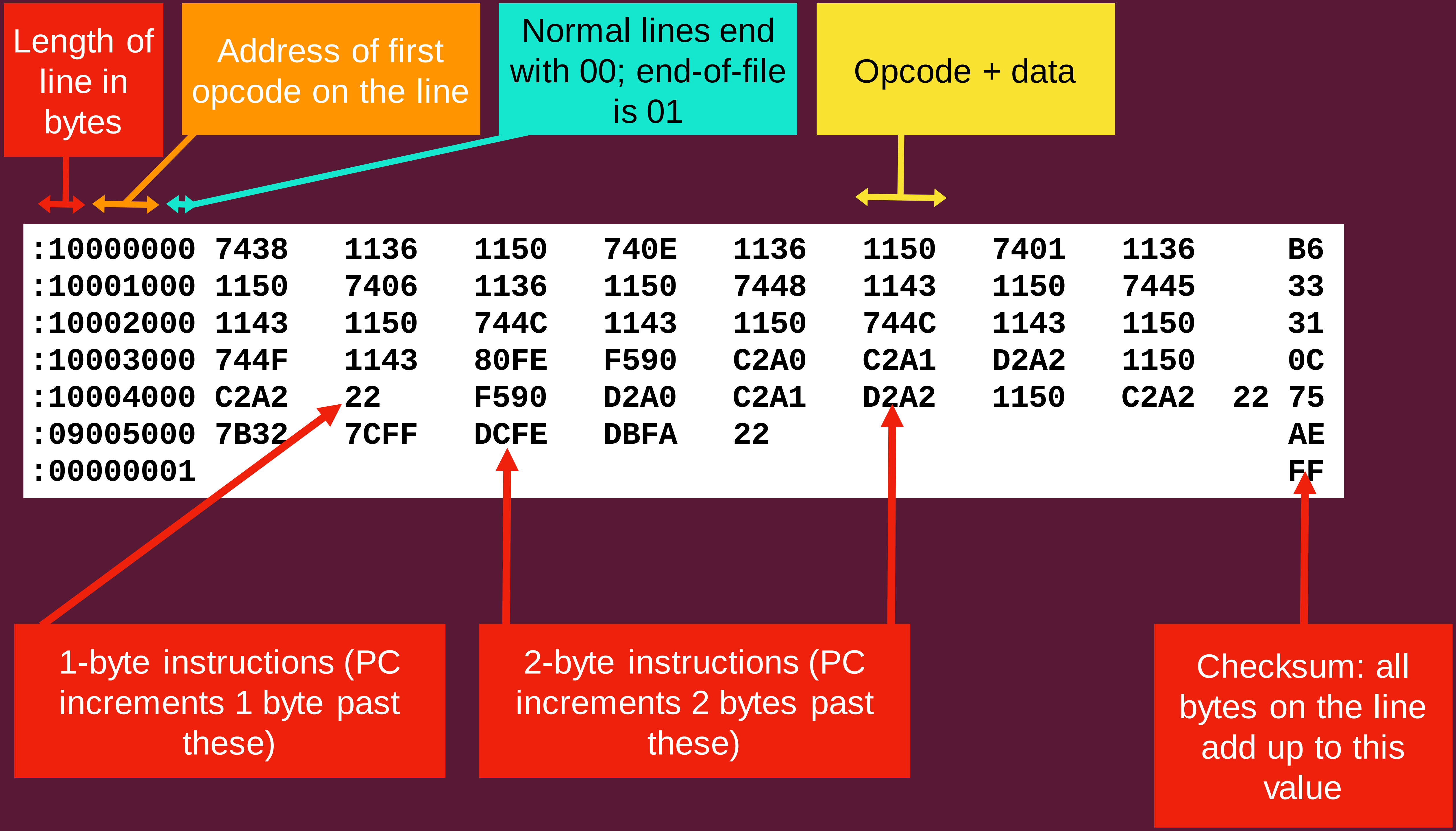
```
START:
MOV A,#0FFH         ;;Move 0xFF(1) to accumulator
MOV P1,A            ;;Move accumulator value to P1

ACALL DELAY         ;;Calls subroutine at ‘delay’

MOV A,#00H          ;;Move 0x0(0) to accumulator
MOV P1,A            ;;Move accumulator value to P1

ACALL DELAY         ;;Delay for another 1 ms

SJMP START          ;;Jump back to ‘START’

DELAY:              ;;1 ms delay Subroutine
MOV R6,#250D        ;;Place 0d250 into Register 6
MOV R7,#250D        ;;Place 0d250 into Register 7
DEL1: DJNZ R6,DEL1  ;;DJNZ: Decrement R6 & jump if not 0
DEL2: DJNZ R7,DEL2  ;;DJNZ is 2-cycles, 2uS to run. 2X500us=1ms
RET                 ;;Return to ACALL
```

Challenge: Change this 1 ms delay to a 1 second delay. Hint: call the delay 4 times in a row (4ms), then repeat this 4x call 250 times. Also, think about how you might realise this with clock frequencies other than 12 MHz

UNDERSTANDING HEX FILES

- Once written and carefully checked over, the assembly language program is assembled.
  - We’ll use the KEIL IDE to do this.
  - The result is a Hex file (.hex), with opcodes and accompanying data represented as hex numbers.
    - This hex file is in the Intel Hex format.
      - More good info about this here: https://www.edsim51.com/intelHex.html
- If you are going to do a lot of Hex file editing, a dedicated hex editor is recommended: https://mh-nexus.de/en/hxd/

UNDERSTANDING HEX FILES



Length of line in bytes

Address of first opcode on the line

Normal lines end with 00; end-of-file is 01

Opcode + data

```
:10000000 7438   1136   1150   740E   1136   1150   7401   1136      B6
:10001000 1150   7406   1136   1150   7448   1143   1150   7445      33
:10002000 1143   1150   744C   1143   1150   744C   1143   1150      31
:10003000 744F   1143   80FE   F590   C2A0   C2A1   D2A2   1150      0C
:10004000 C2A2   22     F590   D2A0   C2A1   D2A2   1150   C2A2   22 75
:09005000 7B32   7CFF   DCFE   DBFA   22                             AE
:00000001                                                         FF
```

1-byte instructions (PC increments 1 byte past these)

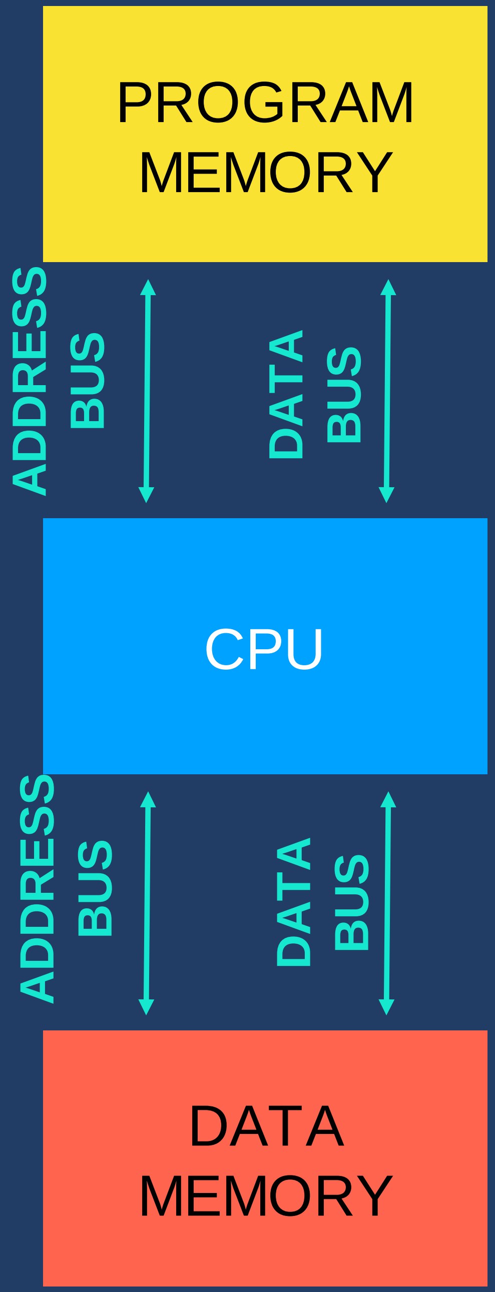
2-byte instructions (PC increments 2 bytes past these)

Checksum: all bytes on the line add up to this value

LAB 1 NOTES

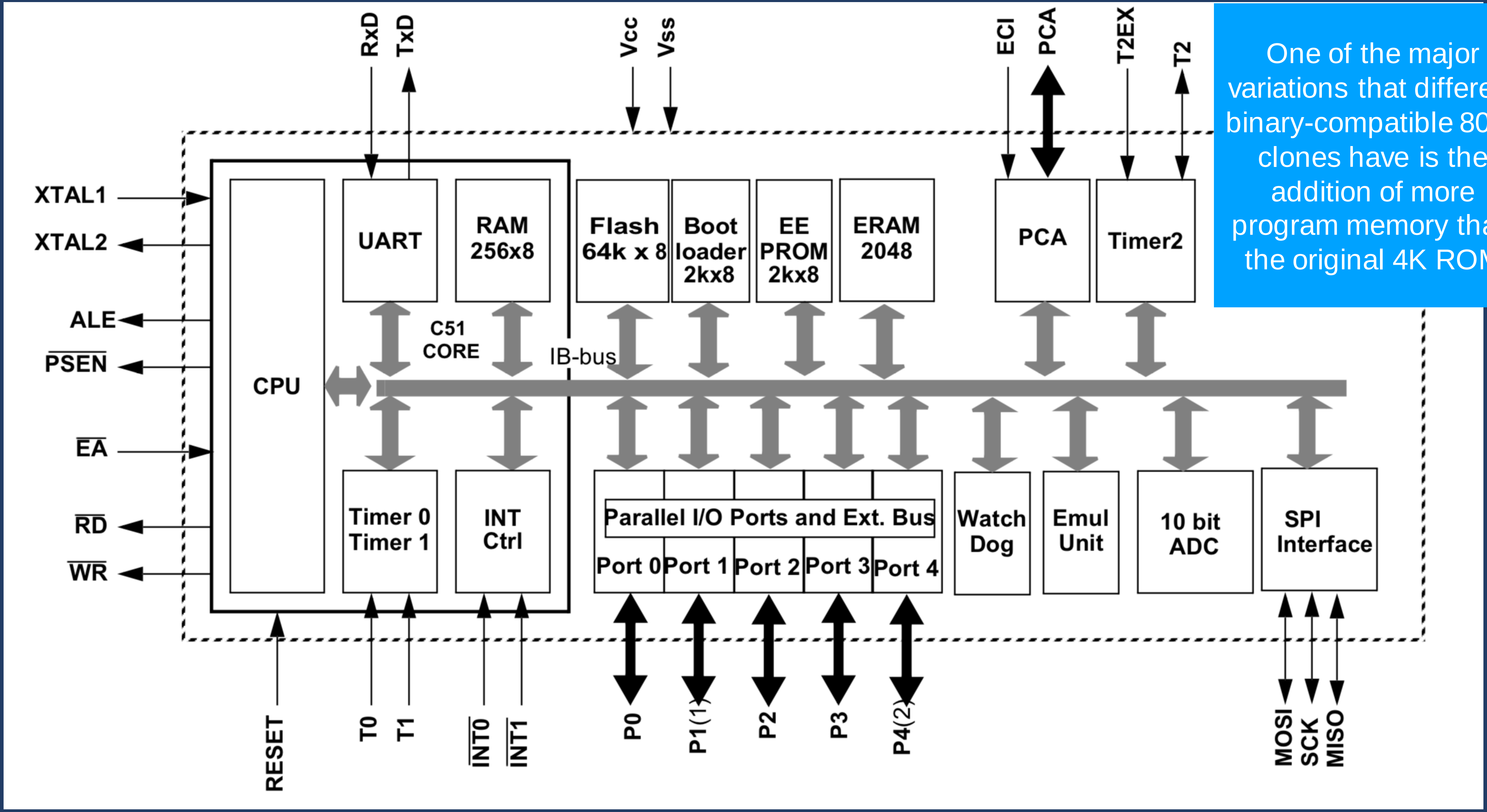
- Turn in: a commented Hex file at start of your Lab1.
  - This needn’t have many additional notes. 1 or 2 lines up at the top explaining the changes that you have made.
    - A brief comment on each line explaining the line-by-line changes.

MEMORY: RAM & STORAGE



- Computers with a Harvard Architecture have separate program and data memories.
  - Microcontrollers have a ‘volatile’ data memory.
    - RAM, loses state when the system resets.
  - They have a non-volatile program memory.
    - Retains state in power-off conditions.
    - Historically, this was some form of ROM (read only memory), originally programmable only once.
      - Modern microcontrollers (including the C8051F020) use flash memory for program memory.
        - Flash memory may be reprogrammed a relatively large number of times, but not during program execution.
        - Program memory is often embedded on the microcontroller, but may also consist of external memory modules.

THE 8051'S STORAGE



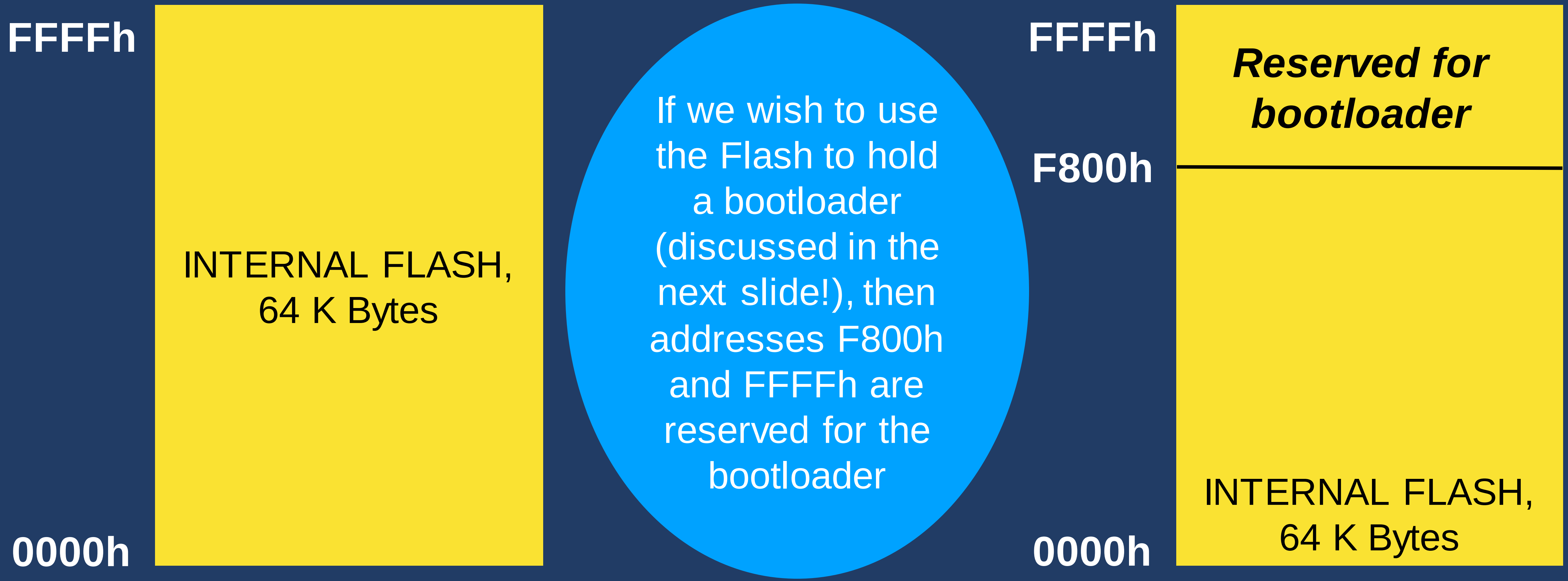
One of the major variations that different binary-compatible 8051 clones have is the addition of more program memory than the original 4K ROM

C8051F020 variant of 8051:
256 bytes of RAM

C8051F020 variant of 8051:
64KB of Flash ProgMem

ROM & FLASH: PROGRAM MEMORY

- The C8051F020 has 64 KB of internal flash.
  - See page 24 of the data sheet (C8051F02X.pdf) for *much* more information.
- While most programs are stored to this in-system-programmable flash…
  - …the C8051F020 has 2KBytes of EEPROM
    - The EEPROM may be edited programatically, and is sometimes used to store variables that need to be retained after a reboot cycle.



FFFFh

INTERNAL FLASH, 64 K Bytes

0000h

If we wish to use the Flash to hold a bootloader (discussed in the next slide!), then addresses F800h and FFFFh are reserved for the bootloader

FFFFh

Reserved for bootloader

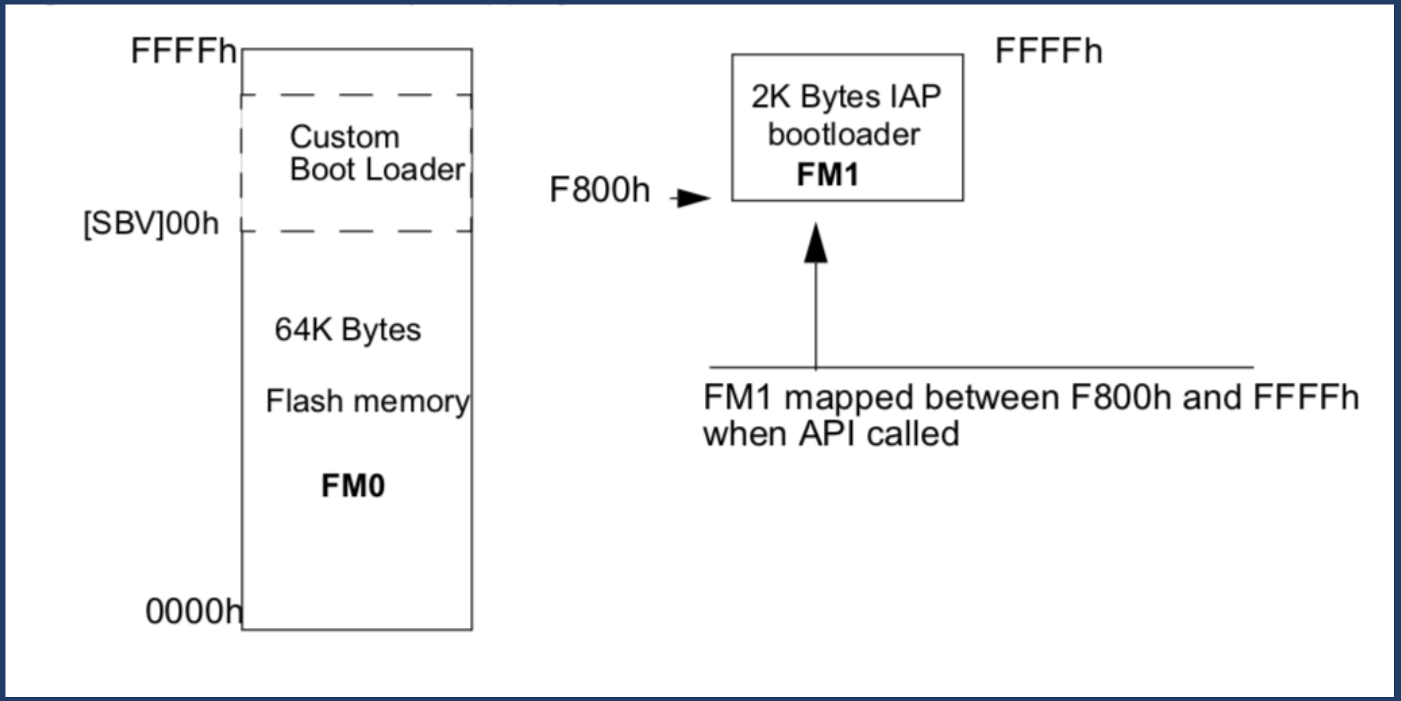
F800h

INTERNAL FLASH, 64 K Bytes

0000h

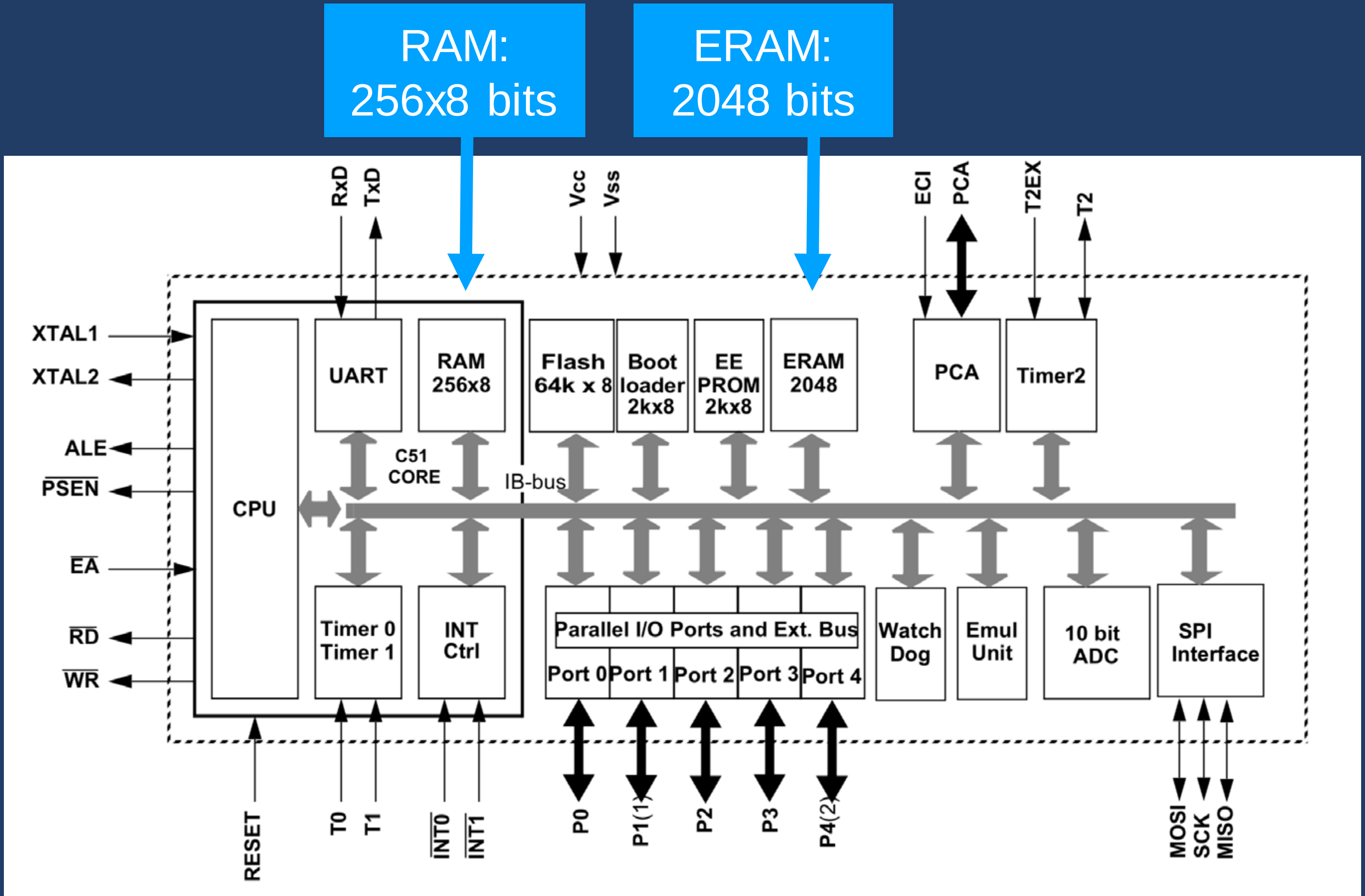
FLASH-BASED BOOTLOADER

- Early microcontrollers (and some contemporary basic/specialised ones) were programmed using custom programmers.
  - These required the microcontroller (or the microcontroller's data ROM) to be removed from the circuit and programmed with high voltages.
- Contemporary microcontrollers can be programmed 'in-system,' allowing for simple rapid development and iteration/revision of firmware.
  - As flash memory requires some specific steps to be programmed, a specific 'serial bootloader' may be used to allow the flash to be programmed in-system via the microcontroller's serial port.



VOLATILE MEMORY: RAM

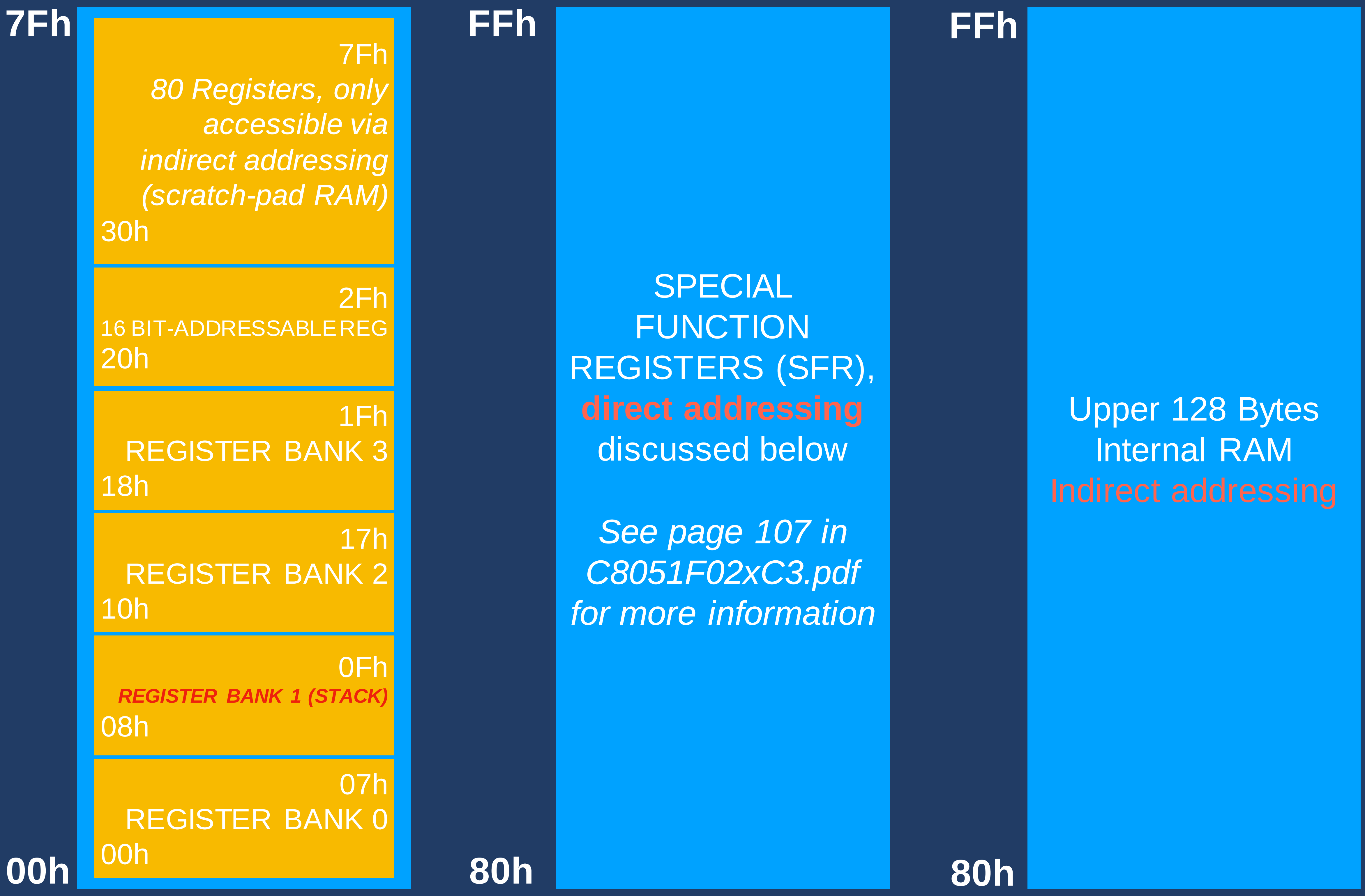
- Originally, the 8051 had 128 Bytes of volatile RAM.
  - The AT89C51AC3 has a whopping 256 Bytes alongside 2 KBytes of additional RAM (called the “expanded RAM segment”, ERAM).
  - This RAM is subdivided into a number of blocks, some general purpose and some with very specific functions.



RAM:
256x8 bits

ERAM:
2048 bits

8051 DATA MEMORY MAP



7Fh

7Fh
80 Registers, only accessible via indirect addressing (scratch-pad RAM)
30h

2Fh
16 BIT-ADDRESSABLE REG
20h

1Fh
REGISTER BANK 3
18h

17h
REGISTER BANK 2
10h

0Fh
REGISTER BANK 1 (STACK)
08h

07h
REGISTER BANK 0
00h

00h

FFh

SPECIAL FUNCTION REGISTERS (SFR), **direct addressing** discussed below

See page 107 in C8051F02xC3.pdf for more information

80h

FFh

Upper 128 Bytes Internal RAM
Indirect addressing

80h

LOWER 128 BYTES, NOT TO SCALE

8051 ADDRESSING MODES

- A key part of computer operation involves the accessing of memory; this may be done on the 8051 using five main approaches.

- **IMMEDIATE ADDRESSING MODE**
- The data is included in the 8051 instruction.
- MOV A,#48H
- The # shows that the data is 'immediate'
- In a sense, this data is hard-coded into the instruction. Fast but less flexible.

- **REGISTER ADDRESSING MODE**
- The data operand is in a specified register.
- Only some registers may be used: R0 through R7 of each of the 8051's banks.
- MOV A,R7
  - Contents of R7 are copied to ACC.

- **DIRECT ADDRESSING MODE**
- The address of a location in RAM is specified, and its contents are operated upon. Only works with internal RAM & SFR's
- MOV A,10H
  - Contents of address are copied to ACC.

- **INDIRECT ADDRESSING MODE**
- Slower: the contents of a location of the address stored in a register are fetched.
- MOV A,@R7
  - The @ indicates an address
- Upper 128 bytes of RAM are accessible this way.

- **INDEXED ADDRESSING MODE**
- Used to step through data (as in lookup tables).
- We won't be exploring this in depth (and you won't be tested on it!), but see details about the MOVC instruction in C8051F02xC3.pdf