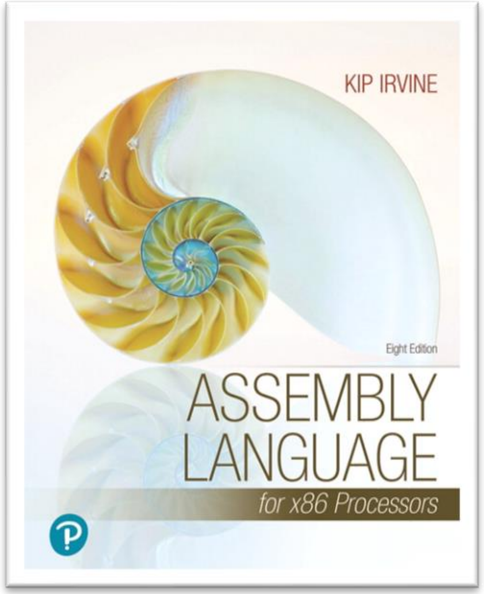
Assembly Language for x86 Processors



Lab 1

Basic Concepts

❑ Lab Objectives:

1. Core concepts relating to assembly language programming
   - ➢ How assembly language fits into the wide spectrum of languages & applications
2. Underlying hardware associated with x86 assembly language
   - ➢ Assembly language is the ideal software tool for communicating directly with a machine
3. Basic operations that take place inside the processor when instructions are executed
4. How programs are loaded and executed by the operating system.

Why am I learning Assembly Language?

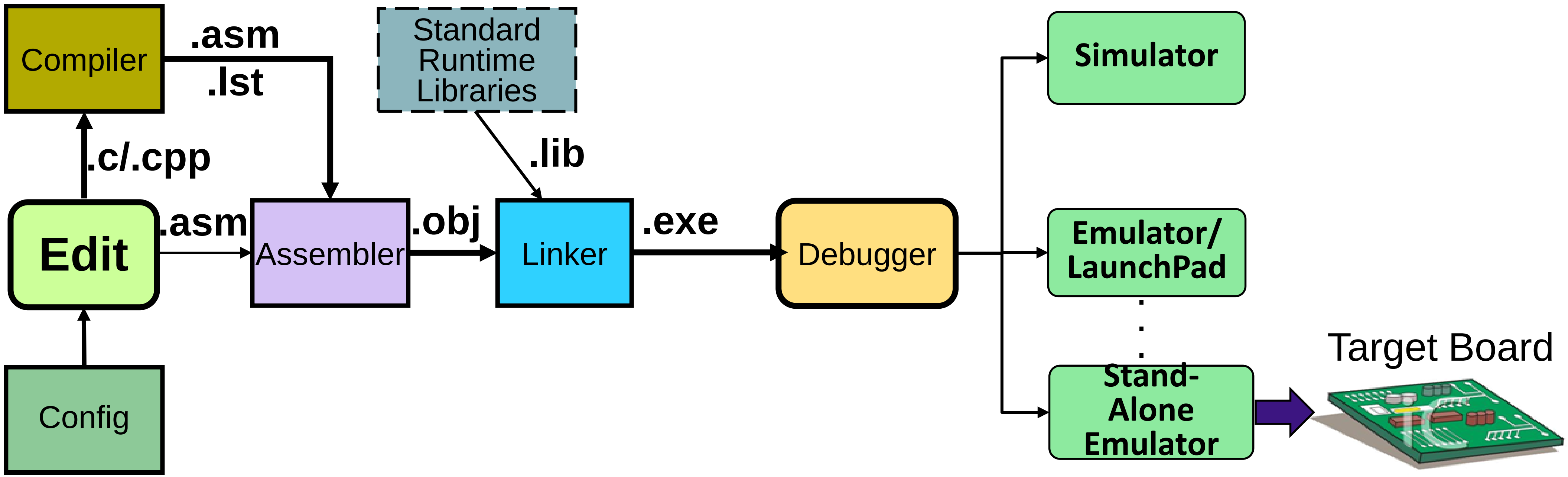
1. Assembly language is a low-level programming language

2. Assembly language provides direct access to computer hardware, requiring you to understand much about the computer’s architecture and operating system.
   - As a HL (C, C++, etc…) developer, you need to develop an understanding of how memory, address, and instructions work at a low level

3. Many programming errors are not easily recognized at the high-level language level
   - In some situations, you many need to “drill down” into your program’s internals to find out why it isn’t working

What Are Assemblers and Linkers?

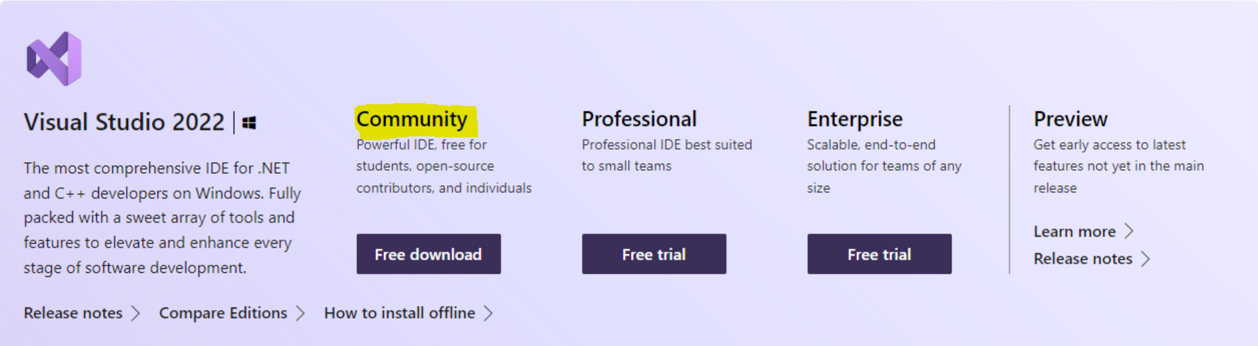
- An **assembler** is a utility program that converts source code programs from assembly language into machine language.
- A **linker** is a utility program that combines individual files created by an assembler into a single executable program.
- A related utility, called a **debugger**, lets you to step through a program while it's running and examine registers and memory.

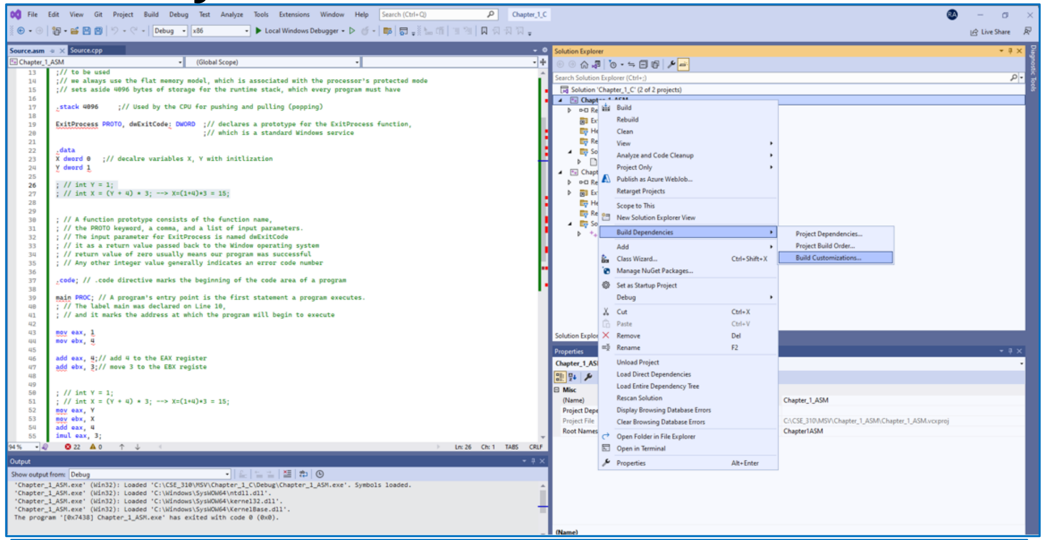


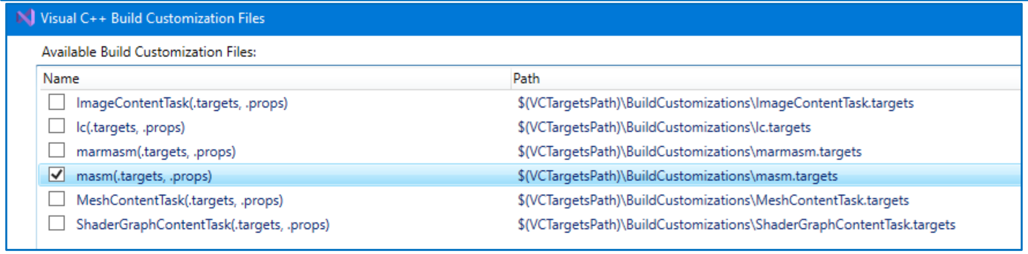
What Hardware and Software Do I Need?

- You need a computer that runs a 32-bit or 64-bit version of Microsoft Windows, along with one of the recent versions of Microsoft Visual Studio (visual studio 2022).
- Integrated Development Environment (IDE) software by Microsoft used for the creating and running of programs

https://visualstudio.microsoft.com/downloads/





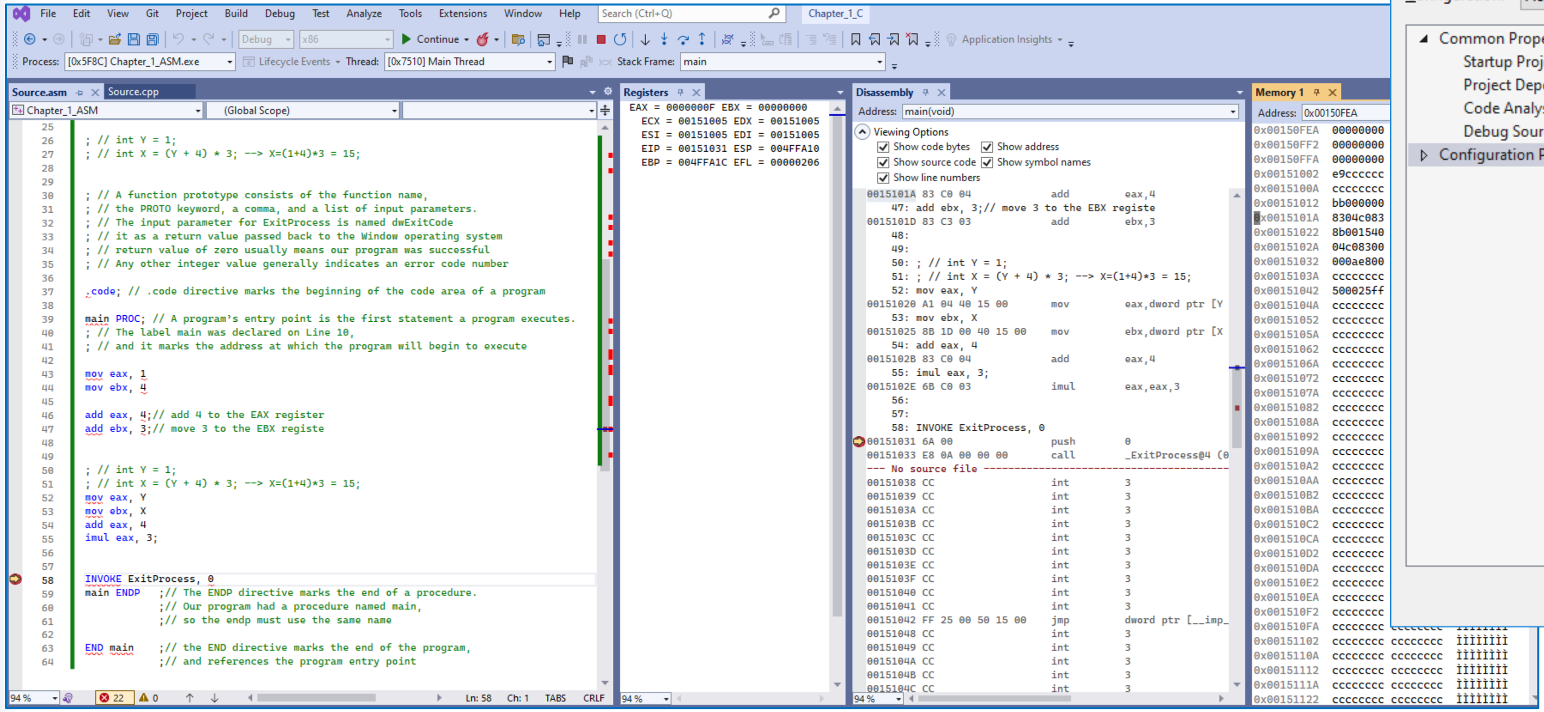


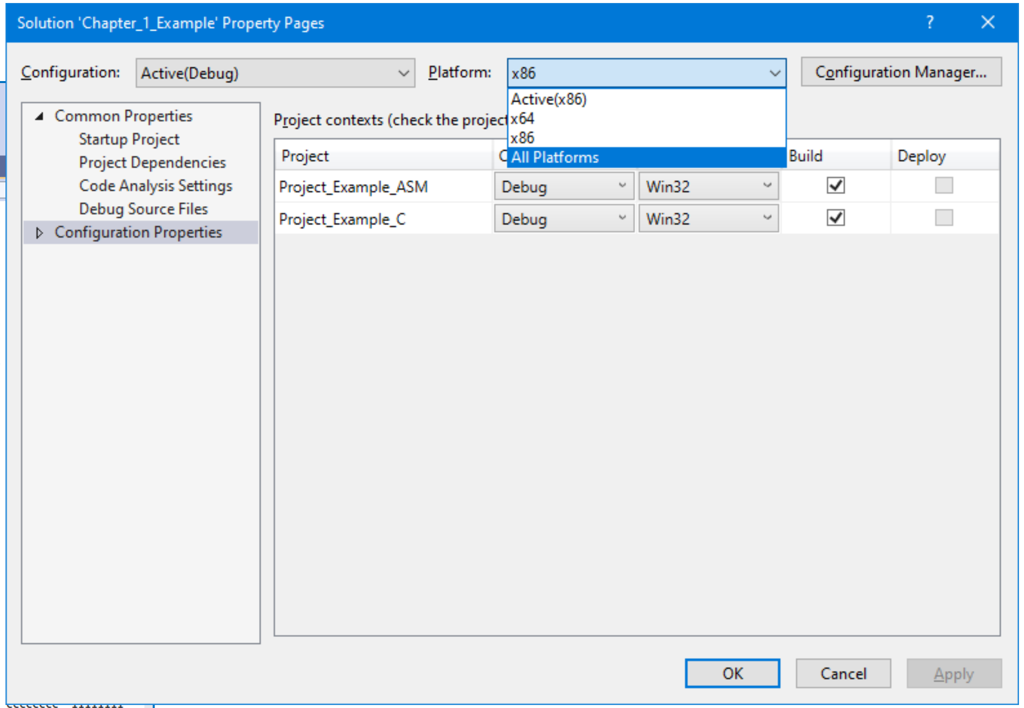
- Need to enable MASM in VS:
- Microsoft Macro Assembler (MASM)
- The Microsoft utility program that translates entire assembly language source programs into machine language

What Types of Programs Can Be Created

1. **32-Bit Protected Mode**: 32-bit protected mode programs run under all 32-bit and 64-bit versions of Microsoft Windows.

2. **64-Bit Mode:** 64-bit programs run under all 64-bit versions of Microsoft Windows.

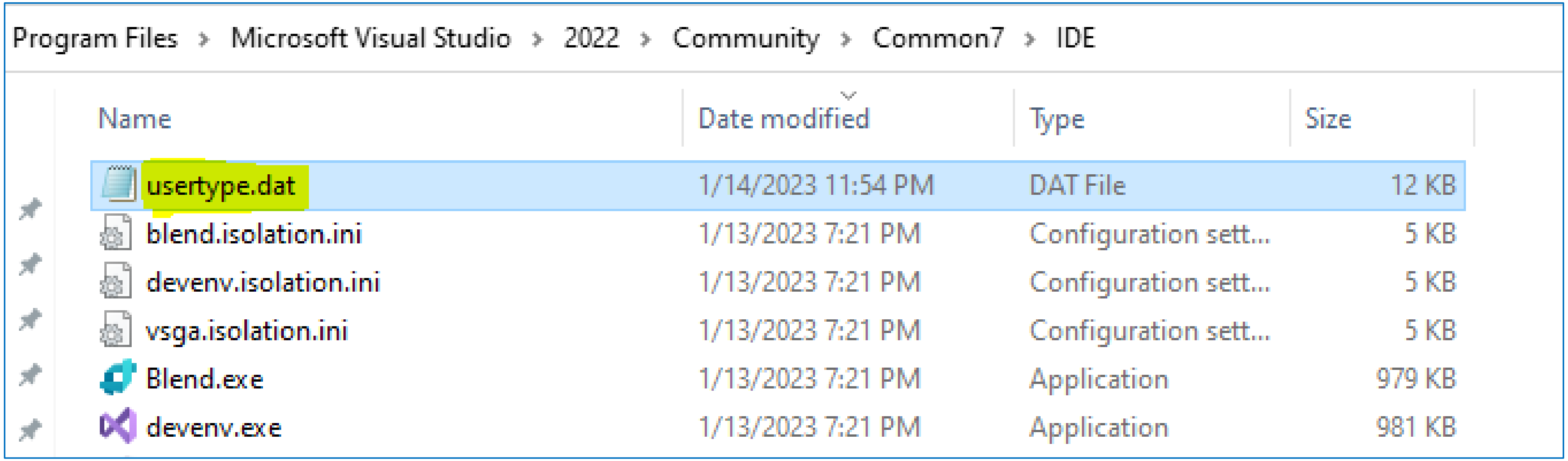




Some Settings for MASM
Will be covered also in chapter 3

- The file [usertype.dat] uploaded to CANVAS, should be copied to:

  C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\IDE



- Restart VS after the above step
- More steps will be covered in the video and next videos.

What Will I Learn?

1. Principles of computer architecture as applied to x86 processors
2. Boolean logic and how it applies to programming and computer hardware
3. How x86 processors manage memory, using protected mode and virtual mode
4. How high-level language compilers (such as C++) translate statements from their language into assembly language and native machine code
5. How high-level languages implement arithmetic expressions, loops, and logical structures at the machine level Data representation
6. How to debug programs at the machine level
7. How application programs communicate with the computer's operating system via interrupt handlers and system calls
8. How to interface assembly language code to C++ programs
9. How to create assembly language application programs

Welcome to Assembly Language

- How does Assembly Language (AL) relate to Machine Language (ML)?
  - Assembly Language consists of statements written with short mnemonics such as ADD, MOV, SUB, and CALL
  - Machine Language is a numeric language specifically understood by a computer's processor (the CPU).

Instruction set

- Instructions include: [in brief]
  1. Logical instructions (AND, OR, XOR, etc.)
  2. Move and branching instructions (allow one to move data from and to registers and conditional and unconditional branching)
  3. bit instructions (operations on single bits in an operand)
  4. Arithmetic instructions such as add and subtract,
  5. Subroutine calls
  6. Other instructions that have to do with the performance of the CPU.

Instruction set (example of Microcomputer)

Instructions	Examples	notes
Logical instructions	AND, OR, XOR, 2's COMPLEMENT	Some generate a carry as well as set other flags for subsequent use
Integer math instructions	Add, Subtract	Carry and other flags are generated
Counting and conditional branching	Increment, decrement, decrement/skip, Bit test/skip	The main means of creating loops and branching out of loops. Skipping is based on some detectable condition
Clear and Set operations	CLEAR register, CLEAR Watchdog timer, CLEAR bit, SET bit,	Allow manipulation of registers and bits within registers
Unconditional branch	GOTO, Return, Return From Interrupt	GOTO is a general branch instruction, Return is used to return from a subroutine after its execution
Move operations	Move to/from register	Allows placing of data for operations
Other instructions	No operation, Sleep	
Shift operations	Shift left, Shift right	Digits are shifted left or right (through carry)

- Some instructions are bit oriented, some are byte (register) oriented, some are literal and control operations
- Many more instructions and specialized use of instructions exist depending on the processor.

Example of an Assembly Program

- The following C++ code carries out two arithmetic operations and assigns the result to a variable. Assume X and Y are integers:

```
1. int Y;
2. int X = (Y + 4) * 3;
```

The translation requires multiple statements because each assembly language statement corresponds to a single machine instruction:

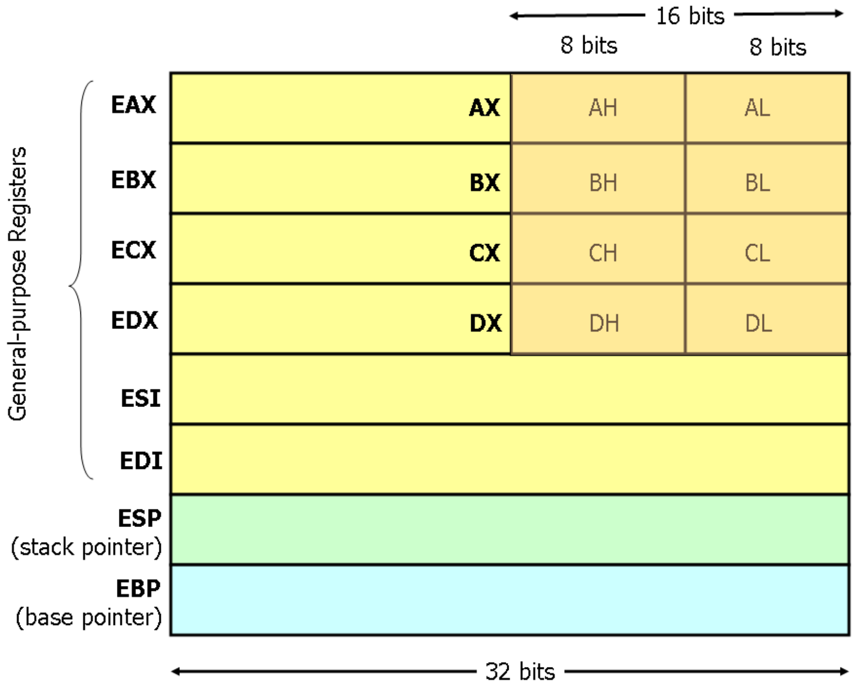
```
1. mov eax,Y    ; move Y to the EAX register

2. add eax,4    ; add 4 to the EAX register

3. mov ebx,3    ; move 3 to the EBX register

4. imul ebx     ; multiply EAX by EBX

5. mov X,eax    ; move EAX to X
```



Is Assembly Language Portable?

– A language whose source programs can be compiled and run on a wide variety of computer systems is said to be portable.

– A C++ program, for example, will compile and run on just about any computer, **unless** it makes specific references to library functions that exist under a single operating system.

– Assembly language is not portable, because it is designed for a specific processor family

Why learn AL?

1. In the early days of programming, most applications were written partially or entirely in assembly language [some loops and algorithms are still written in AL)

2. As a computer engineering major, you may likely be asked to write embedded programs, which are short programs stored in a small amount of memory in single-purpose devices

3. Real-time applications dealing with simulation and hardware monitoring require precise timing and responses

4. Computer game consoles require their software to be highly optimized for small code size and fast execution

Real Time Operating System (RTOS) for ARM Processor

```c
int OS_AddThreads(void(*task0)(void), void(*task1)(void),
void(*task2)(void))
{
  int32_t status;
  status = StartCritical();

  tcbs[0].next = &tcbs[1]; // 0 points to 1
  tcbs[1].next = &tcbs[2]; // 1 points to 2
  tcbs[2].next = &tcbs[0]; // 2 points to 0

  SetInitialStack(0); Stacks[0][STACKSIZE-2] = (int32_t)(task0); // PC
  SetInitialStack(1); Stacks[1][STACKSIZE-2] = (int32_t)(task1); // PC
  SetInitialStack(2); Stacks[2][STACKSIZE-2] = (int32_t)(task2); // PC
  RunPt = &tcbs[0];       // thread 0 will run first
  EndCritical(status);
  return 1;               // successful
}

///******** OS_Launch ***************
// start the scheduler, enable interrupts
// Inputs: number of 20ns clock cycles for each time slice
// Outputs: none (does not return)
void OS_Launch(uint32_t theTimeSlice)
{
  NVIC_ST_RELOAD_R = theTimeSlice - 1; // reload value
  NVIC_ST_CTRL_R = 0x00000007;         // enable core ARM
  StartOS();                           // start on the first task
}
```

```c
void Task1(void)
{
  Count1 = 0;
  for(;;)
  {
     Count1++;
  }
}
```

```c
void Task2(void)
{
  Count2 = 0;
  for(;;)
  {
     Count2++;
  }
}
```

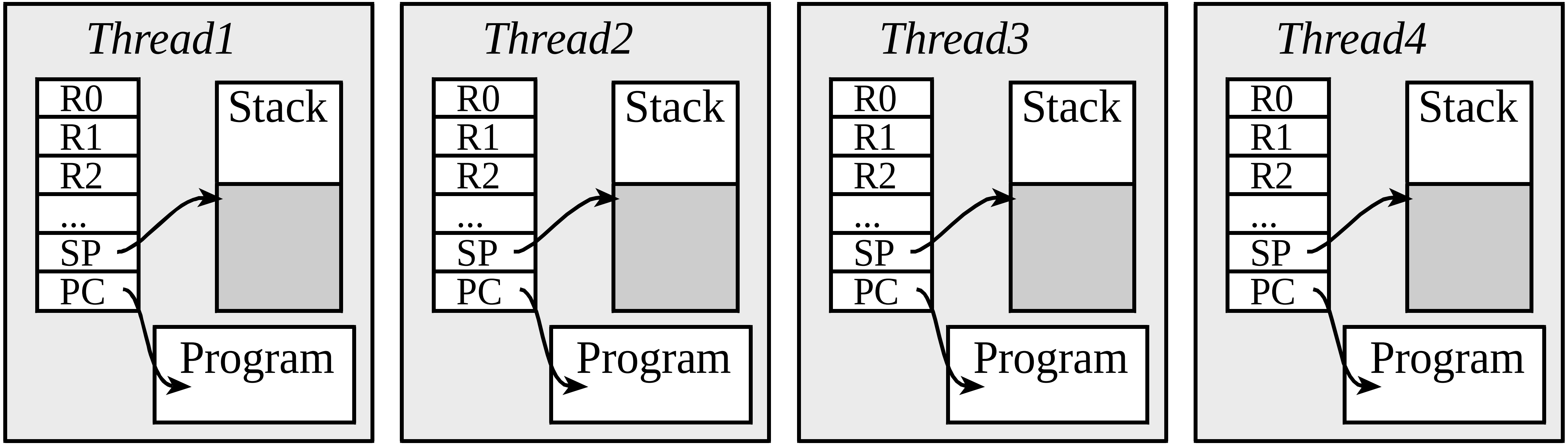
Embedded Systems: Real-Time Operating Systems for ARM Cortex-M Microcontrollers, Jonathan Valvano Vol. 3, Third Edition, 2014 (ISBN-13: 978-1466468863).


```asm
SysTick_Handler                ; 1) Saves R0-R3,R12,LR,PC,PSR
    CPSID   I                  ; 2) Prevent interrupt during switch
    PUSH    {R4-R11}           ; 3) Save remaining regs r4-11
    LDR     R0, =RunPt         ; 4) R0=pointer to RunPt, old thread
    LDR     R1, [R0]           ;    R1 = RunPt
    STR     SP, [R1]           ; 5) Save SP into TCB
    LDR     R1, [R1,#4]        ; 6) R1 = RunPt->next
    STR     R1, [R0]           ;    RunPt = R1
    LDR     SP, [R1]           ; 7) new thread SP; SP = RunPt->sp;
    POP     {R4-R11}           ; 8) restore regs r4-11
    CPSIE   I                  ; 9) tasks run with interrupts enabled
    BX      LR                 ; 10) restore R0-R3,R12,LR,PC,PSR

StartOS
    LDR     R0, =RunPt         ; currently running thread
    LDR     R2, [R0]           ; R2 = value of RunPt
    LDR     SP, [R2]           ; new thread SP; SP = RunPt->stackPointer;
    POP     {R4-R11}           ; restore regs r4-11
    POP     {R0-R3}            ; restore regs r0-3
    POP     {R12}
    POP     {LR}               ; discard LR from initial stack
    POP     {LR}               ; start location
    POP     {R1}               ; discard PSR
    CPSIE   I                  ; Enable interrupts at processor level
    BX      LR                 ; start first thread
```

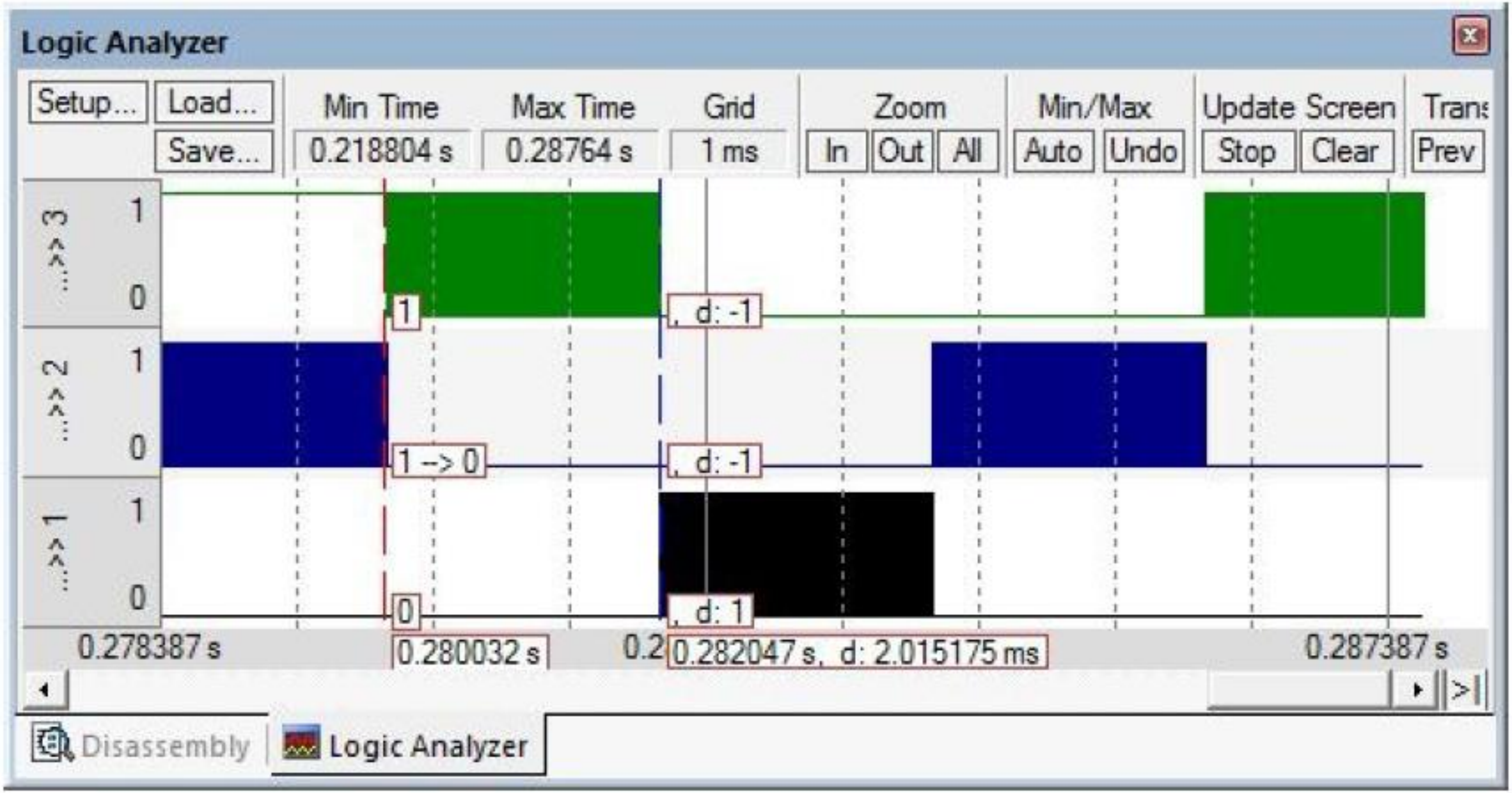
Multi-Threading / Multi-Tasking

- Threads appear they are running simultaneously, where in fact only one thread is running at any time
- Each thread executes independently (program),
- Thread Switching is handled by ASM program



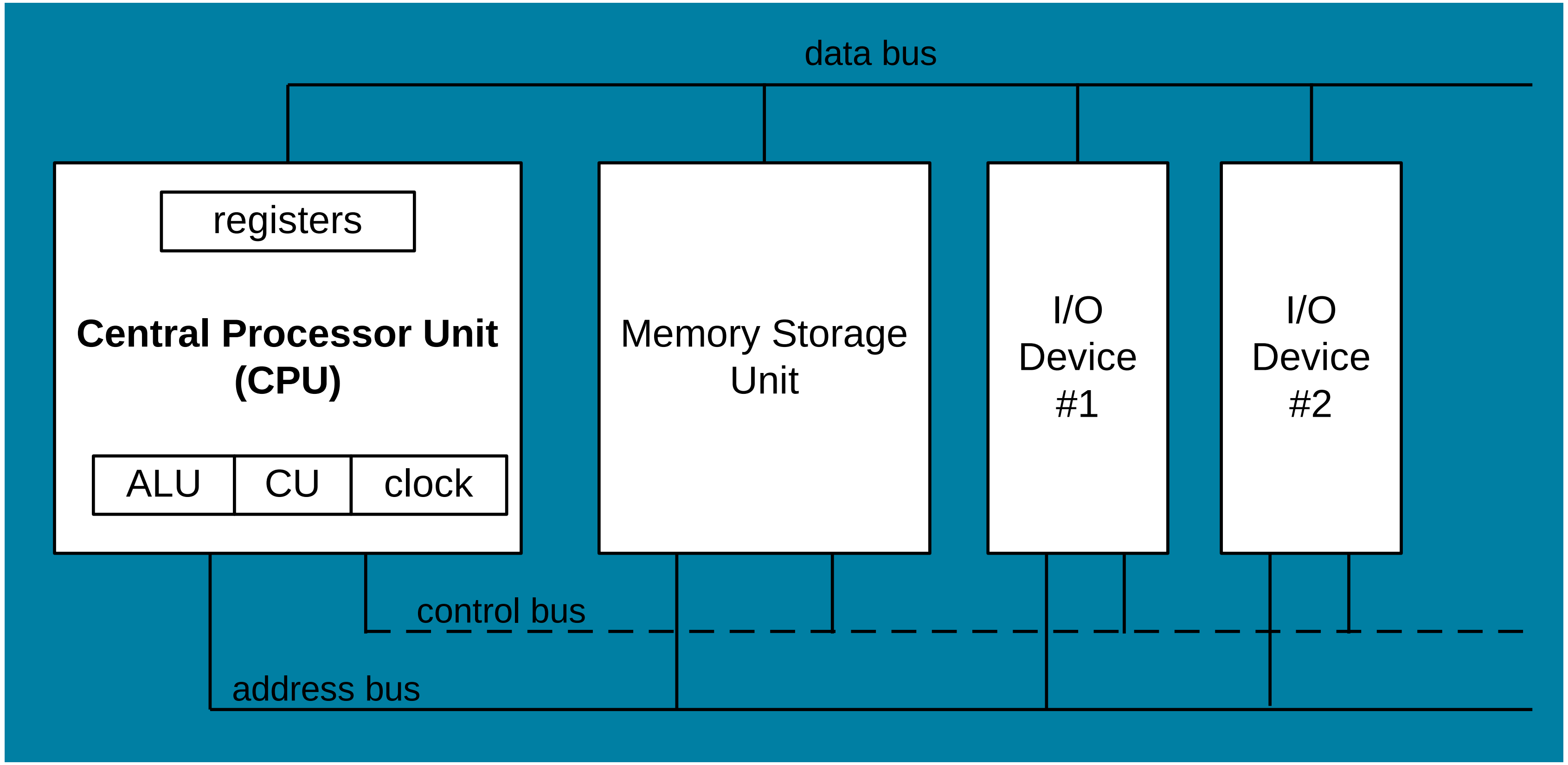
Embedded Systems: Real-Time Operating Systems for ARM Cortex-M Microcontrollers, Jonathan Valvano Vol. 3, Third Edition, 2014 (ISBN-13: 978-1466468863).

OS Profiling



Basic Microcomputer Design

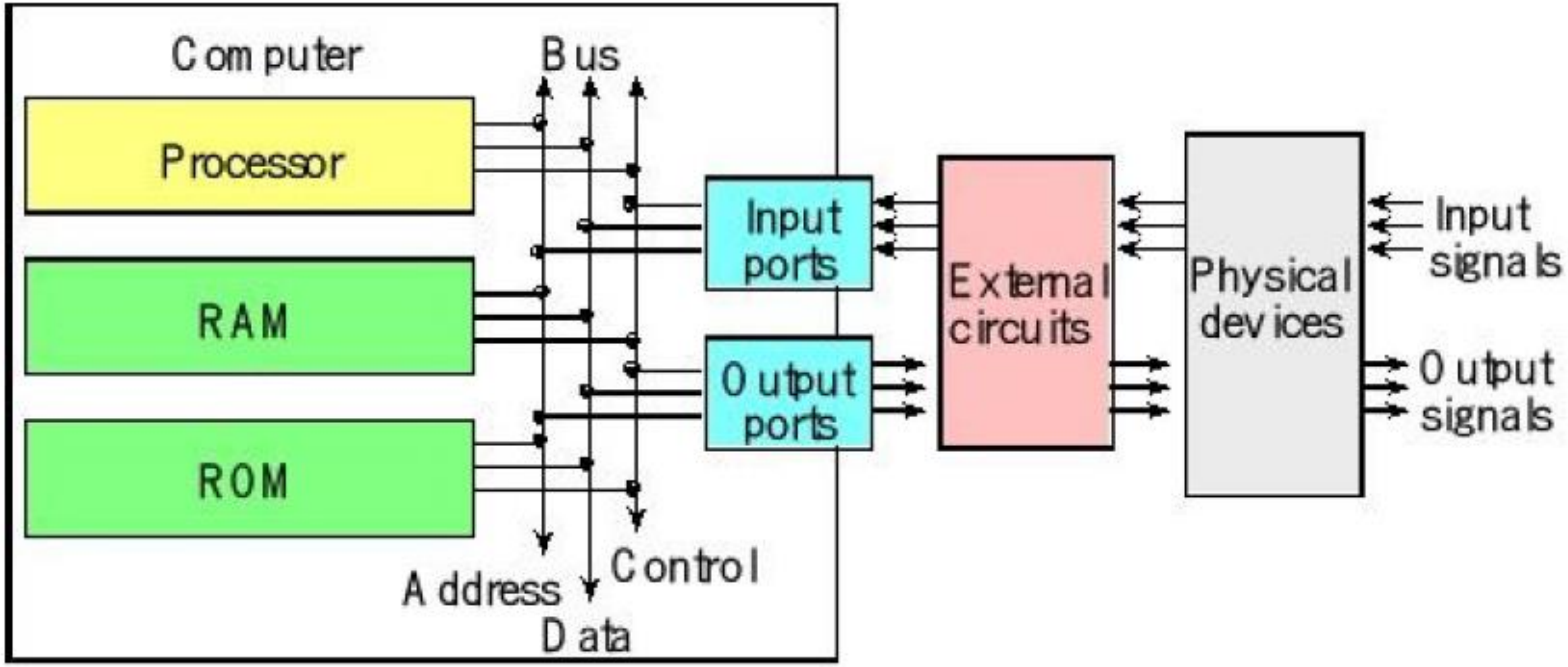
1. Control unit (CU) coordinates sequence of execution steps
2. ALU performs arithmetic and bitwise processing
3. Clock synchronizes CPU operations



Computers, processors, and microcontrollers

- A computer combines a central processing unit (CPU), random access memory (RAM), read only memory (ROM), and input/output (I/O) ports

Memory will be discussed later

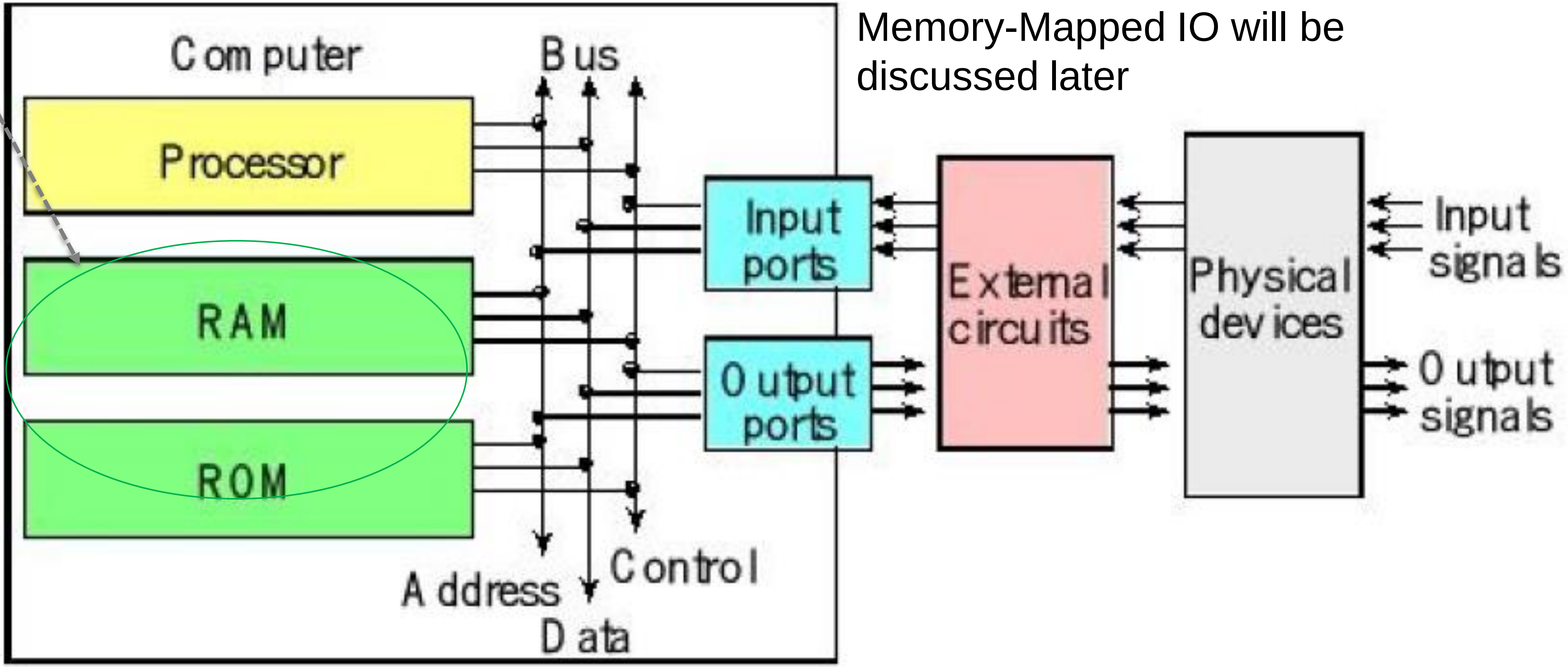


The basic components of a computer system include processor, memory and I/O. (Valvano book)

Computers, processors, and microcontrollers

- **Software** is an ordered sequence of very specific instructions that are stored in memory, defining exactly what and when certain tasks are to be performed.

Memory will be discussed later



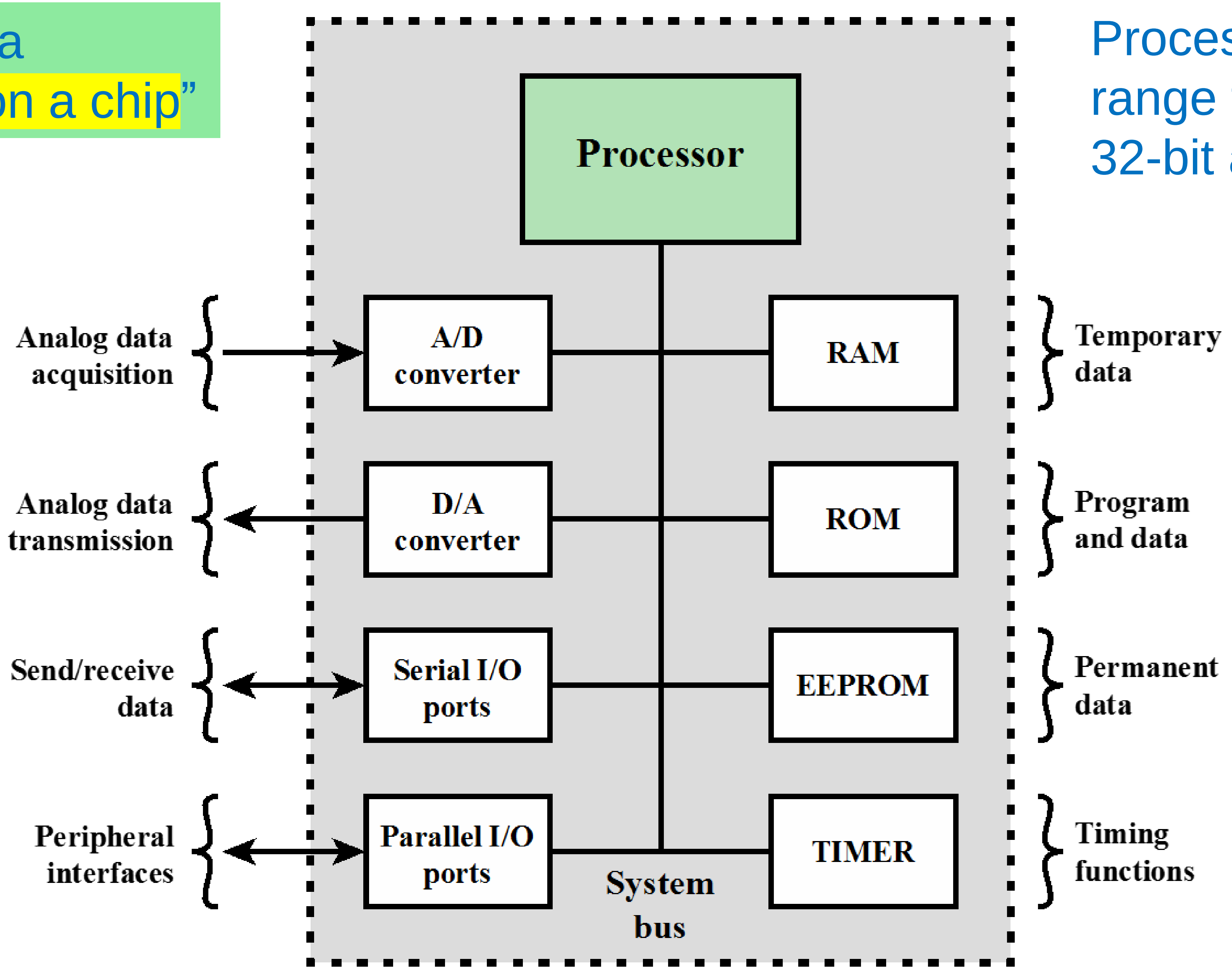
Memory-Mapped IO will be discussed later

Example: Typical Microcontroller Chip Elements

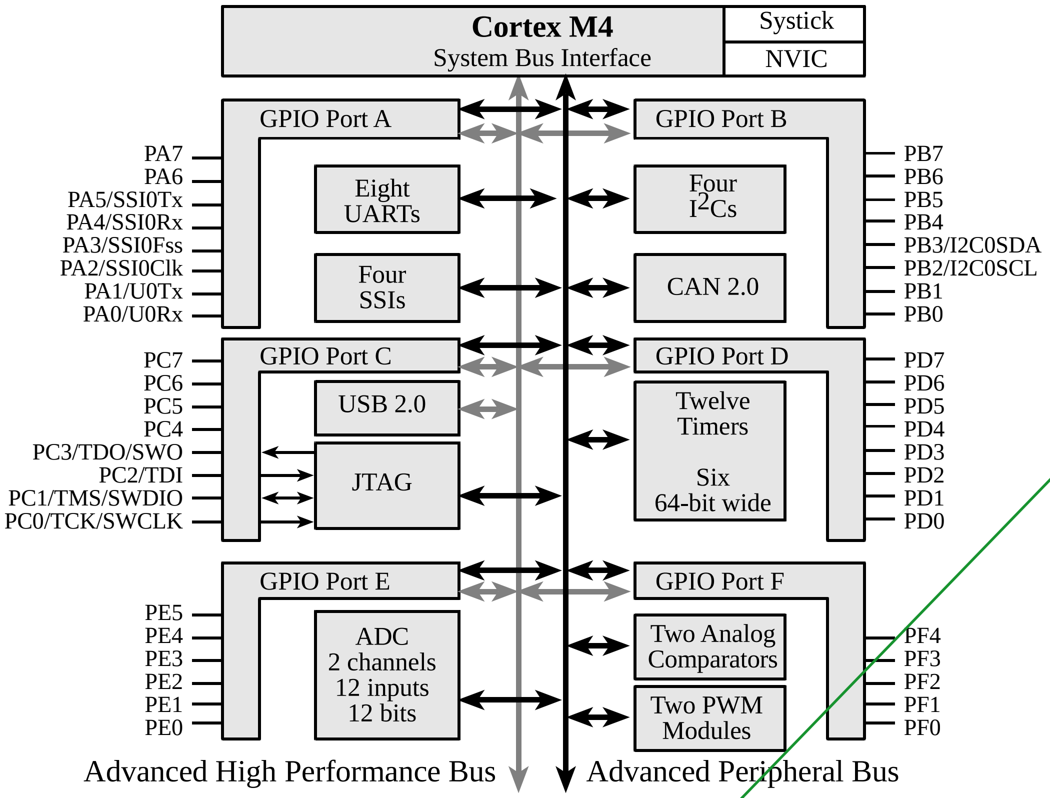
Also called a "computer on a chip"

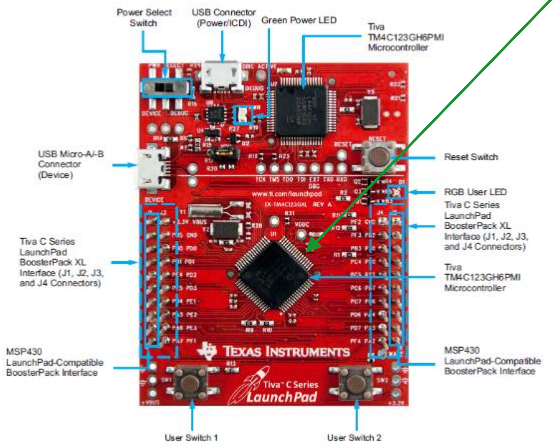
Processors range from 4-bit to 32-bit architectures

Memory-Mapped IO will be discussed later

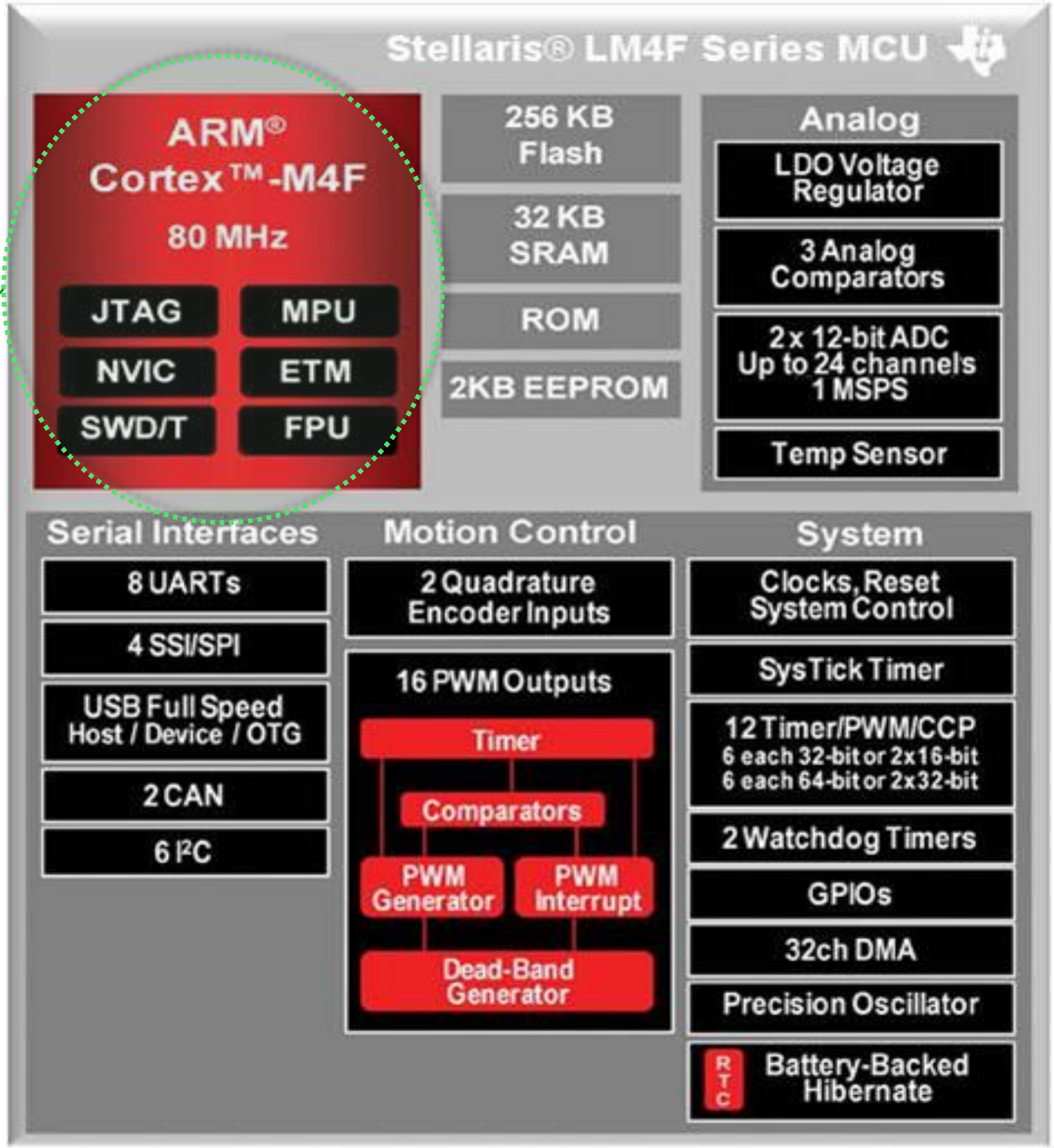


Example: Tiva™ TM4C123G Microcontroller

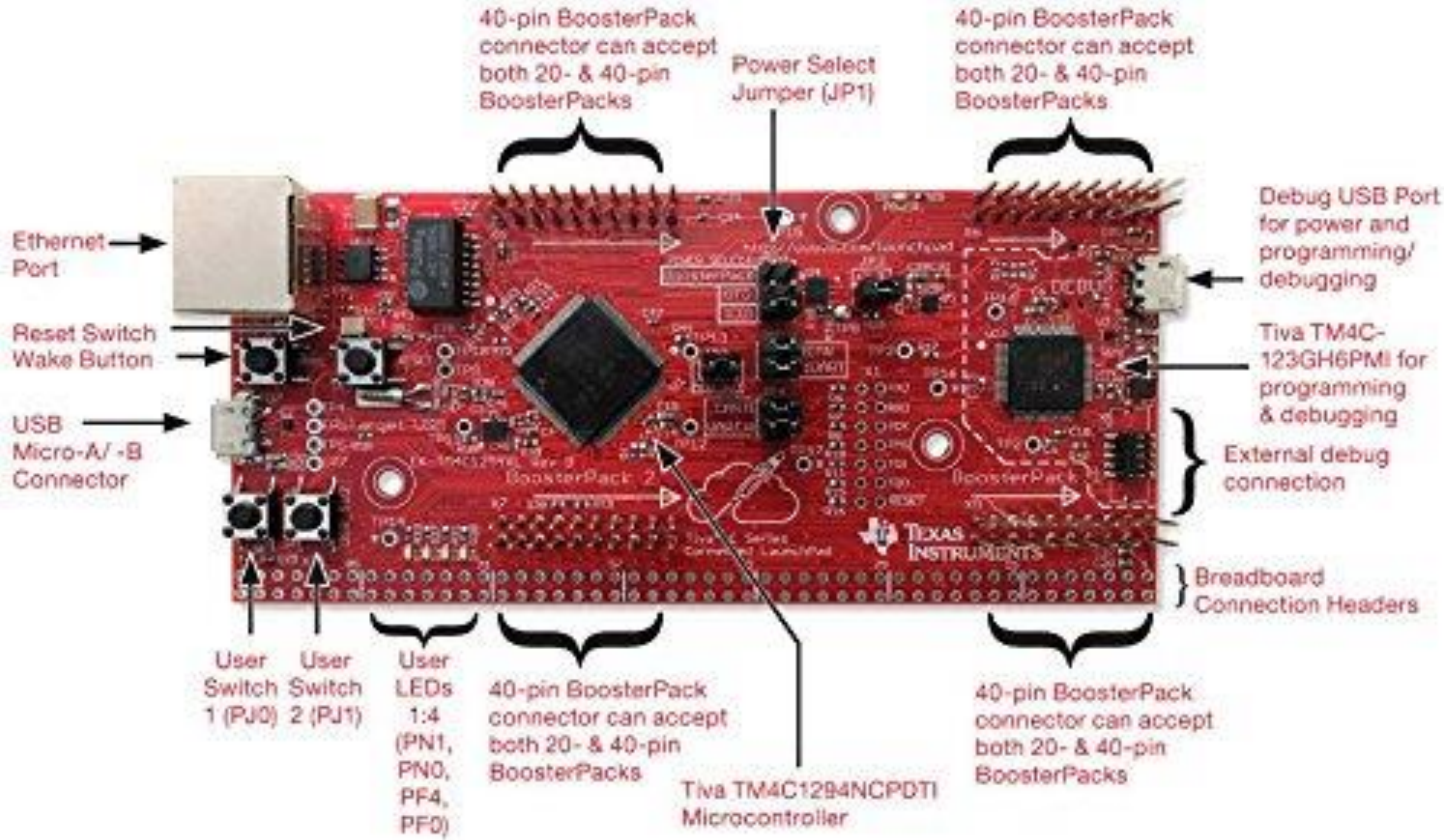




Tiva C Series TM4C123G LaunchPad Evaluation Board



Example: TEXAS INSTRUMENTS EK-TM4C1294XL EVALUATION BOARD, TIVA C LAUNCHPAD, TM4C1294



Example: Freescale (NXP) HCS12 Microcontroller

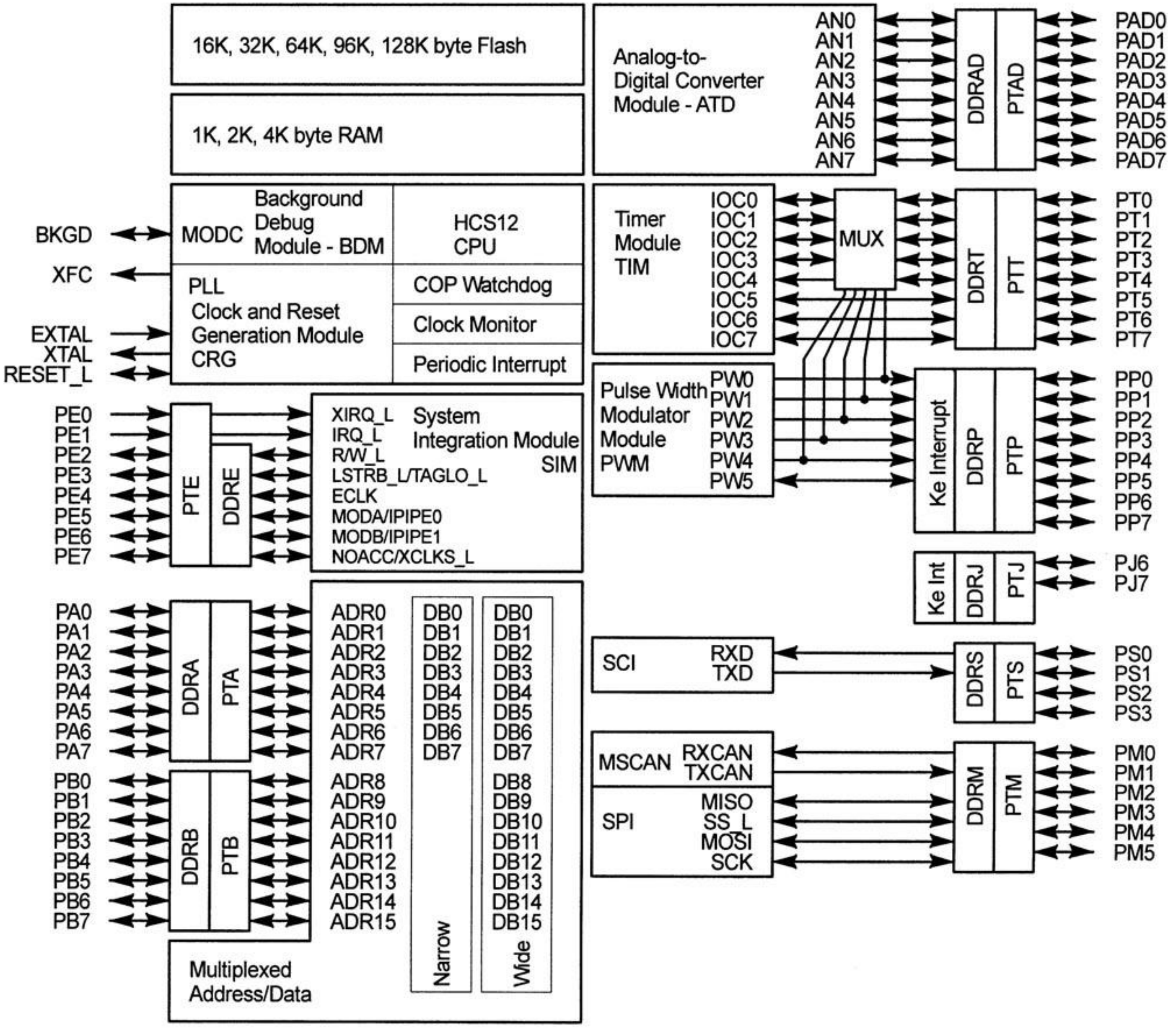


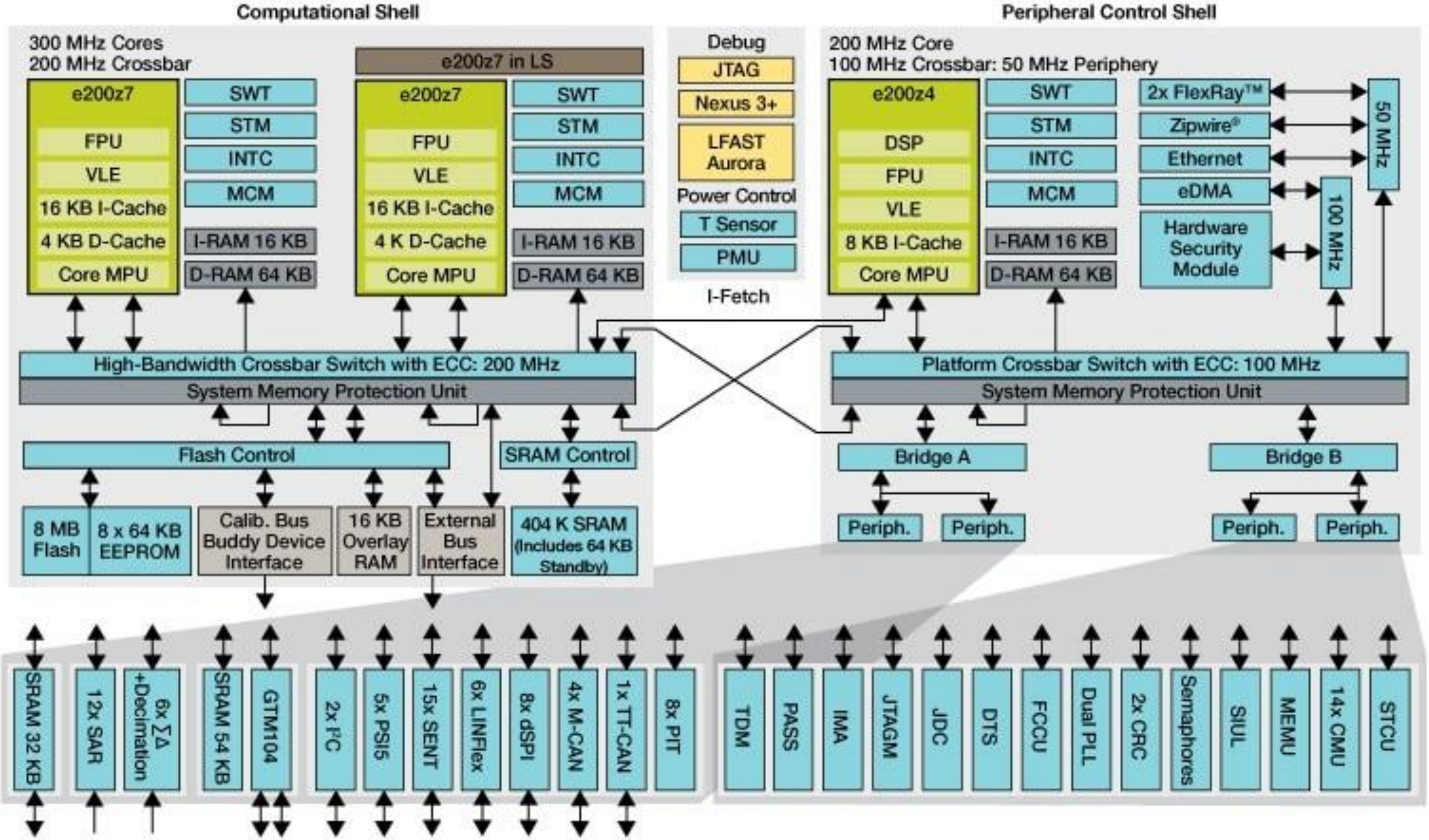
Figure 4-1 MC9S12C family block diagram.

Memory-Mapped IO will be discussed later

Memory-Mapped IO will be discussed later

MPC5777M 32-bit Multicore MCU for Powertrain Applications

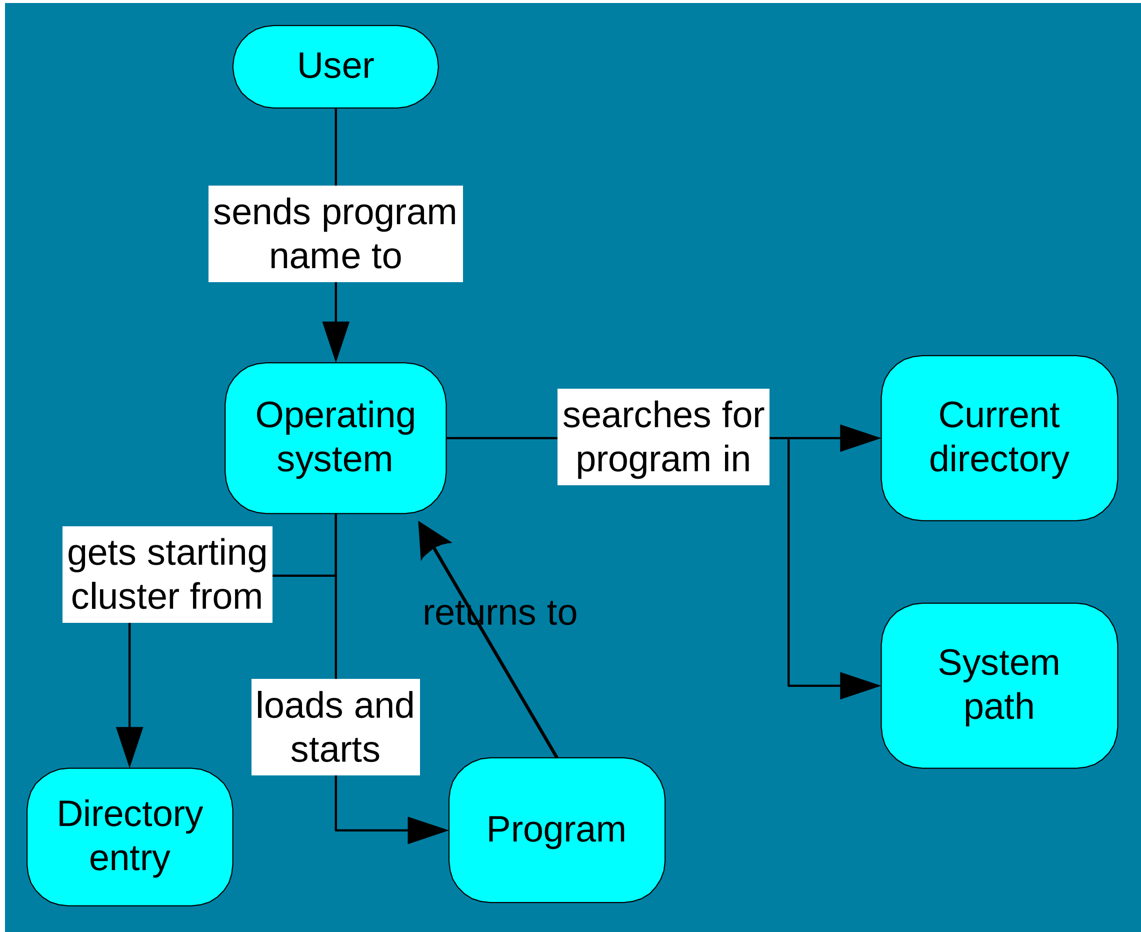
- MPC5777M Power Architecture® MCU targets high-end industrial and powertrain applications that meet next-generation advanced engine control, functional safety, and security requirements



How a Program Runs

- Before a program can run, it must be loaded into memory by a utility known as a program loader.
- After loading, the operating system must point the CPU to the program's entry point , which is the address at which the program is to begin execution

1. The OS determines the next available location in memory and loads the program file into memory
2. The OS begins execution of the program's first machine instruction (its entry point).
3. The process runs by itself. It is the OS's job to track the execution of the process and to respond to requests for system resources



Modes of Operation

1. Protected mode
   - native mode (Windows, Linux) → all instructions and features are available
   - Programs are given separate memory areas (prevented from referencing outside this area)

2. Real-address mode
   - native MS-DOS (access to system memory and hardware devices)
   - Not supported by new windows OS

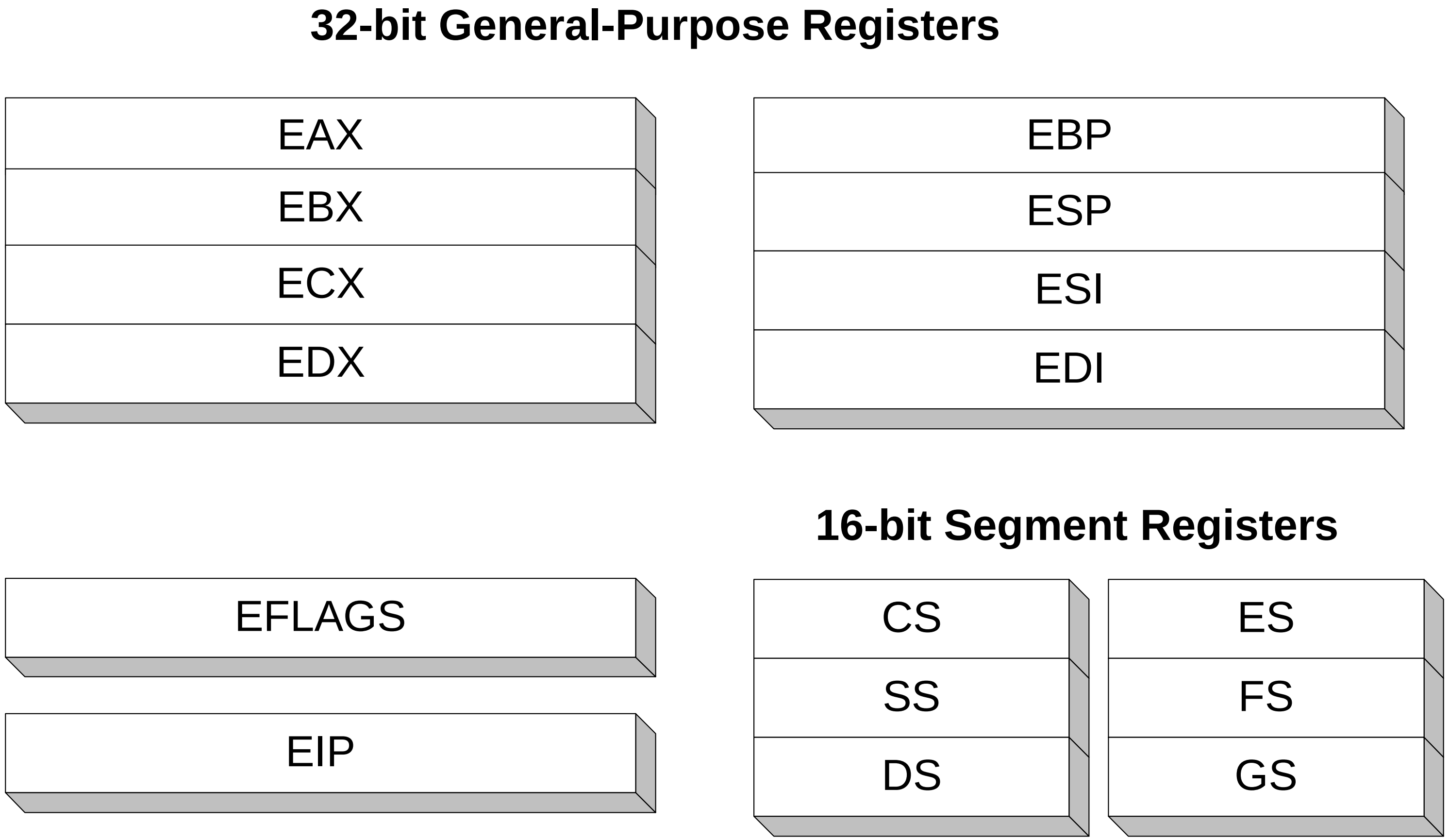
3. System management mode
   - power management, system security, diagnostics

- Virtual-8086 mode
  - hybrid of Protected
  - each program has its own 8086 computer

General-Purpose Registers

- ❑Registers are storage locations inside the CPU, optimized for speed.
- ➢Designed to be accessed at much higher speed than conventional memory.

- Eight general-purpose registers
- Processor Status Flags register (EFLAGS)
- Instruction Pointer (EIP)
- Six segment registers

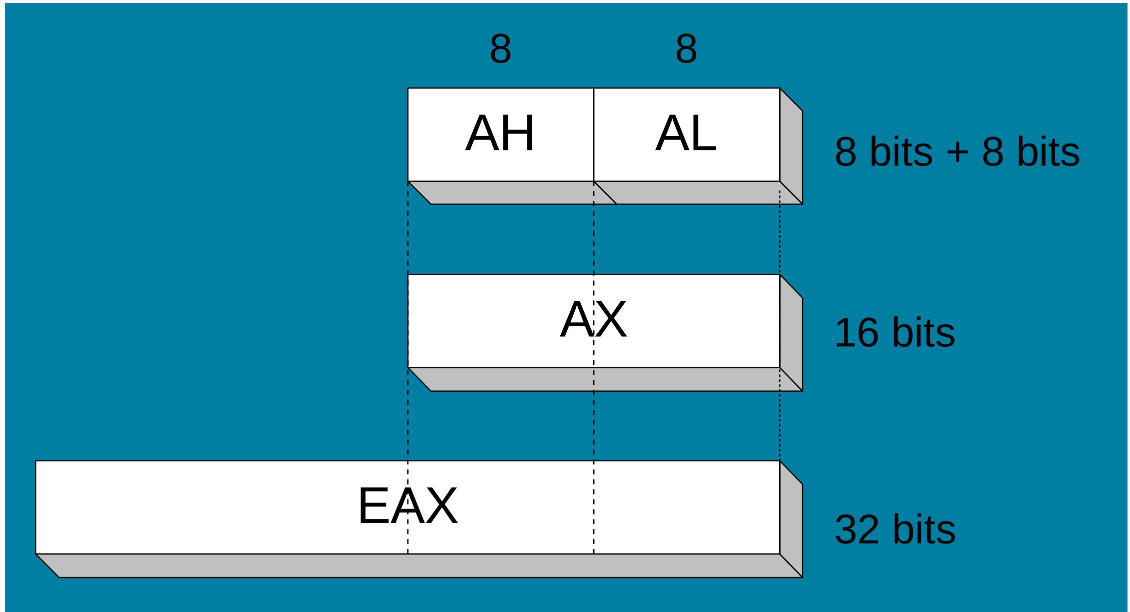


Accessing Parts of Registers

- General-purpose registers are primarily used for arithmetic and data movement

32-bit	16-bit	8-bit (high)	8-bit (low)
EAX	AX	AH	AL
EBX	BX	BH	BL
ECX	CX	CH	CL
EDX	DX	DH	DL

- Use 8-bit name, 16-bit name, or 32-bit name
- Applies to EAX, EBX, ECX, and EDX



Index and Base Registers

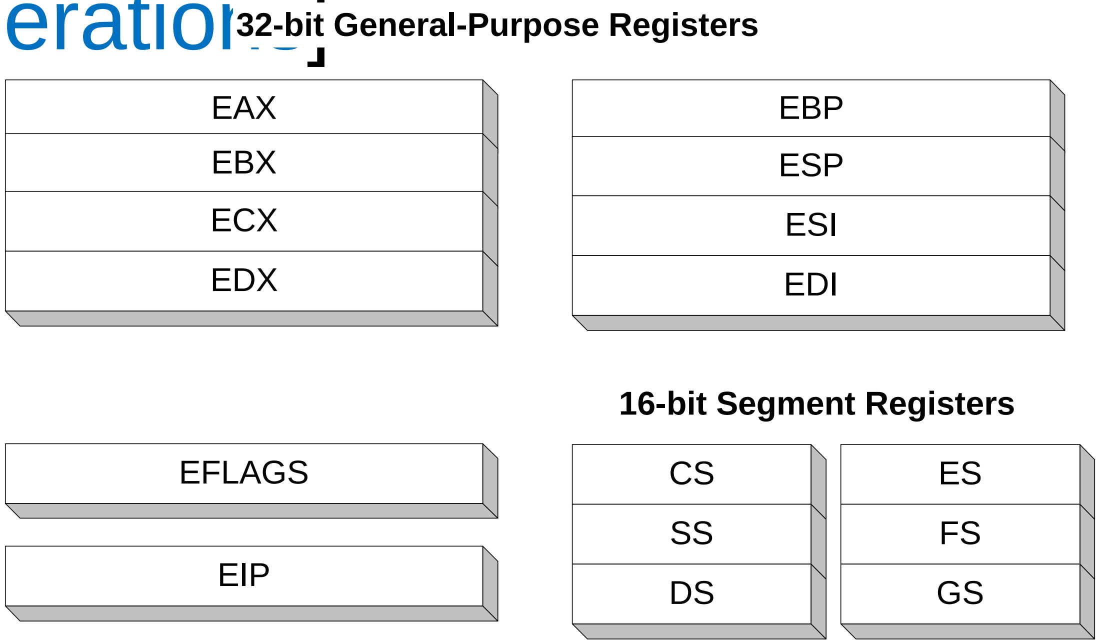
- Some registers have only a 16-bit name for their lower half:

32-bit	16-bit
ESI	SI
EDI	DI
EBP	BP
ESP	SP

Some Specialized Register Uses (2 of 2)

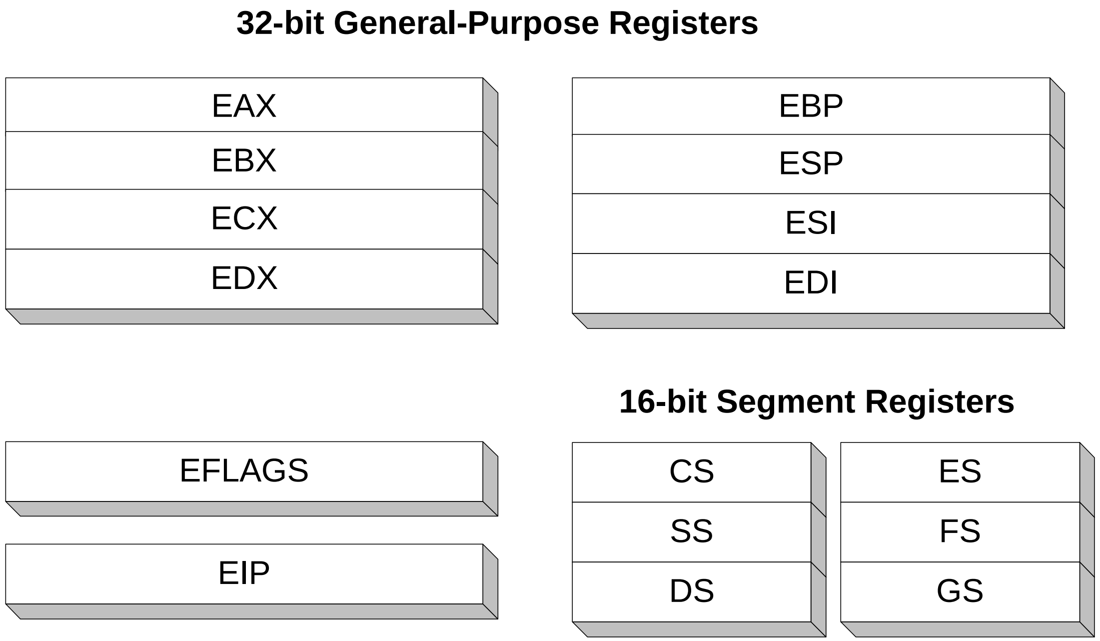
- EIP – Instruction Pointer: Contains the address of the next instruction to be executed

- EFLAGS [Contains Results of Operations]
  - Status and control flags
  - Each flag is a single binary bit

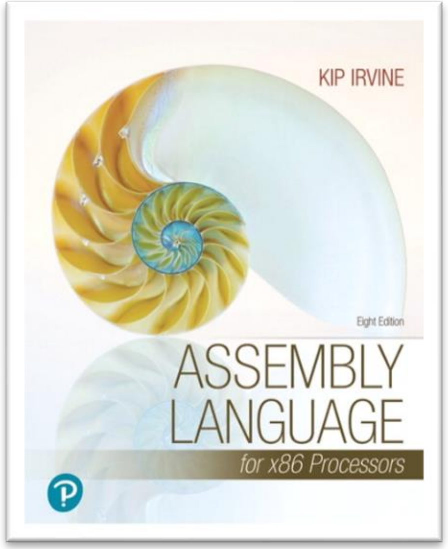


Status Flags

- Carry (CF)
  - unsigned arithmetic out of range
- Overflow (OF)
  - signed arithmetic out of range
- Sign (SF)
  - result is negative
- Zero (ZF)
  - result is zero
- Auxiliary Carry (AC)
  - carry from bit 3 to bit 4
- Parity (PF)
  - sum of 1 bits is an even number



Additional References



Prentice-Hall
(Pearson Education)