Chapter 3 :

Embedded processors/Microcontrollers Memory and I/O

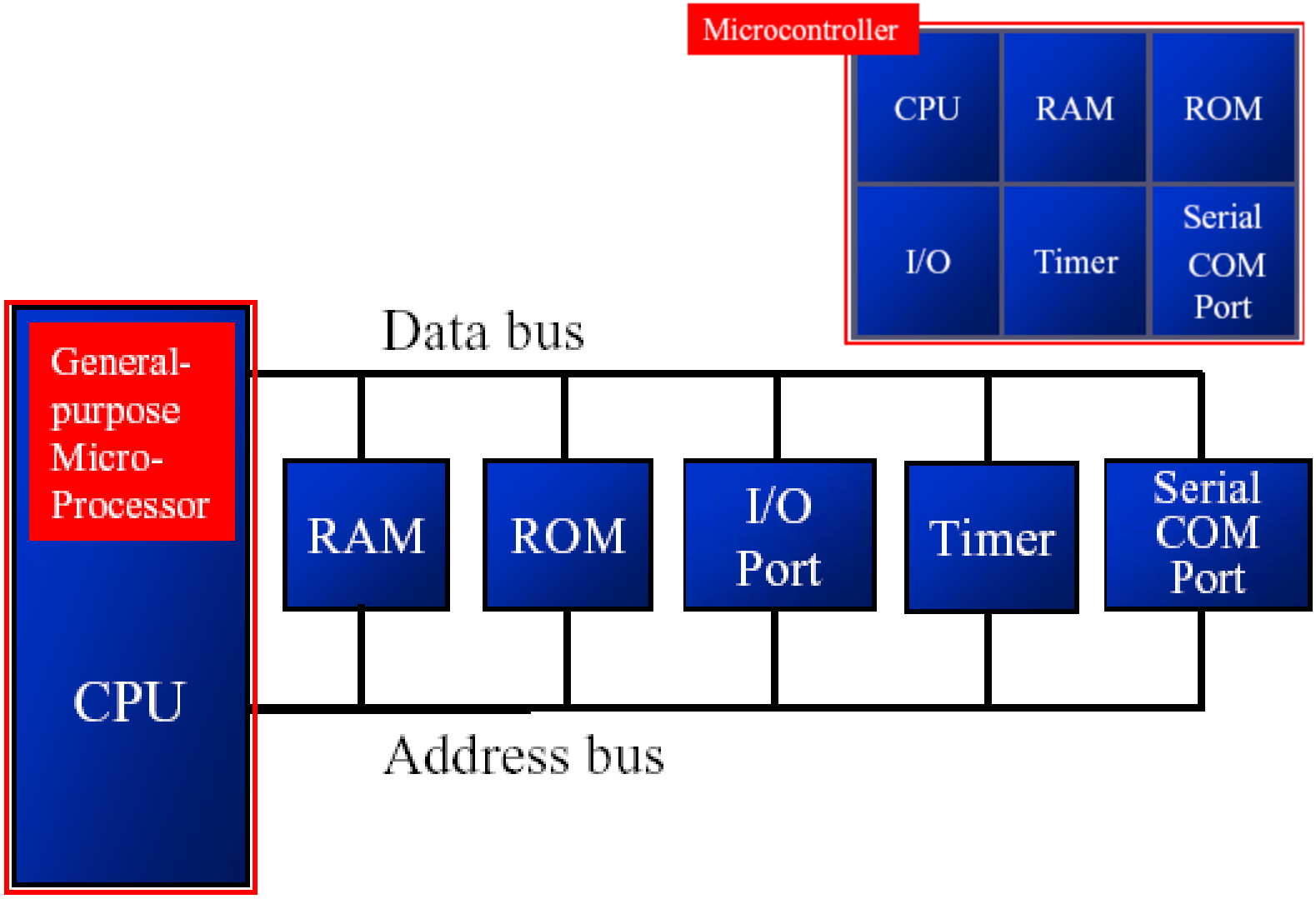
Department of Computer Engineering

Textbook : Tammy Noergaard, “Embedded Systems Architecture: A Comprehensive Guide for Engineers and Programmers”, 2nd Edition, Newnes, ISBN: 9780123821966, 2012. pp129-206

CPU vs. MCU

- Microprocessor
  - No RAM, ROM, I/O Ports on chip itself
  - e.g. Pentium (Intel), PowerPC (Motorola), Spark (SUN)
  - Applications: Desktop, Laptop, workstations, Servers
- Microcontroller
  - Microprocessor, RAM, ROM, timers, I/O Ports on a single chip.
  - 8051(Intel), 68HC08 (Motorola), AVR (Atmel), Z8 (Zilog), PIC (Microchip), ARM (Advanced RISC Machine).
  - Application: TV Remote Control, video games, robots,

CPU vs. MCU


Microcontroller
CPU
RAM
ROM
I/O
Timer
Serial COM Port
General-purpose Micro-Processor
CPU
Data bus
RAM
ROM
I/O Port
Timer
Serial COM Port
Address bus

CPU vs. MCU

- Microprocessor:
  - Faster.
  - General purpose.
  - Expensive
  - CPU is stand-alone, RAM, ROM, I/O, timer are separate.
  - Designer can control the amount of ROM, RAM and I/O ports.

CPU vs. MCU

- Microcontroller:
  - **Slower**, **Stand alone operation**
  - **Cheaper**.
  - **CPU, RAM, ROM, I/O and timer** are all on a **single chip**(but fixed amount).
  - for applications in which cost, power and space are critical: A microcontroller is often small and **low cost**. **A low-power** device: a battery-operated microcontroller might consume as little as 50mW.
  - **Single Operation**: Dedicated to one task and run one specific program. The program is stored in ROM and generally does not change.
  - A microcontroller may take an input from the device it is controlling and controls the device by sending signals to different components in the device.
  - A typical low-end microcontroller chip might have 1000 bytes of ROM and 20 bytes of RAM on the chip, along with eight I/O pins.

Memory types

There are several types of memory which is divided into RAM (Volatile) and ROM and Flash memory (Non-Volatile) and each type is divided into sub types as follows:

RAM(Random Access Memory):

- **SRAM(Static RAM):** Very expensive, Very high performance, Constructed of Flip-Flops
- **DRAM(Dynamic RAM):** Cheap, Constructed from Capacitors so it needs to be refreshed periodically.

Memory types

- **ROM(Read Only Memory):**
  - **ROM(ROM)**
  - **PROM(Programmable ROM).**
  - **EPROM (Erasable PROM).**
  - **EEPROM(Electrically Erasable PROM).**
- **Flash Memory:**
  - It is non-volatile computer memory that can be electrically erased and reprogrammed. It is a technology that is primarily used in memory cards and USB flash drives for general storage and transfer of data between computers and other digital products. It is a specific type of EEPROM (Electrically Erasable Programmable Read-Only Memory) that is erased and programmed in large blocks

Microcontrollers for Embedded Systems

- An embedded product uses a microprocessor (or microcontroller) to do one task and one task only
  - There is only one application software that is typically burned into ROM
- A PC, in contrast with the embedded system, can be used for any number of applications
  - It has RAM memory and an operating system that loads a variety of applications into RAM and lets the CPU run them
  - A PC contains or is connected to various embedded products
    - Each one peripheral has a microcontroller inside it that performs only one task

x86 PC Embedded Applications

- Many manufactures of general-purpose microprocessors have targeted their microprocessor for the high end of the embedded market
  - There are times that a microcontroller is inadequate for the task
- When a company targets a general-purpose microprocessor for the embedded market, it optimizes the processor used for embedded systems
- Very often the terms *embedded processor* and *microcontroller* are used interchangeably

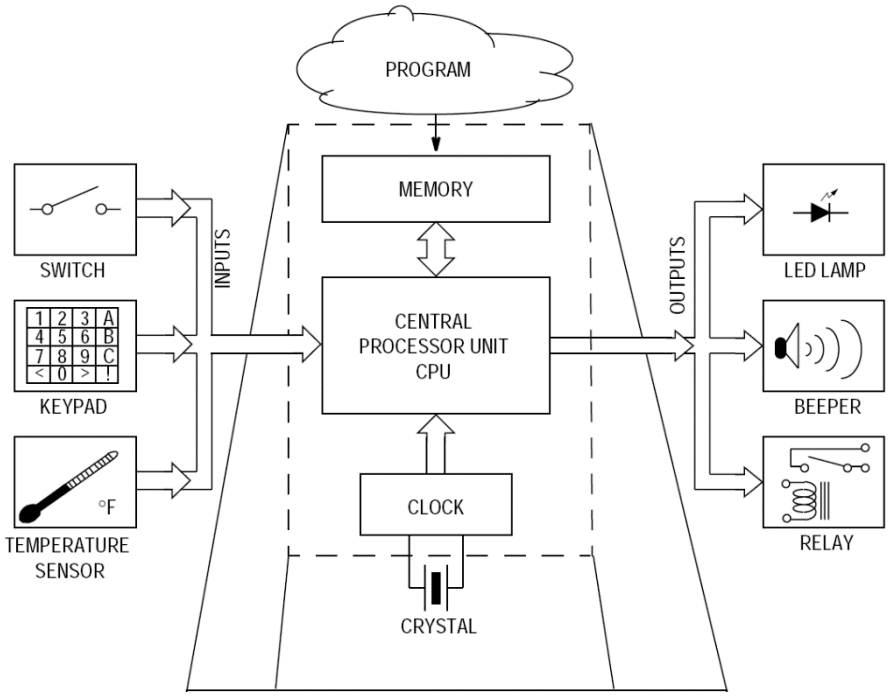
x86 PC Embedded Applications

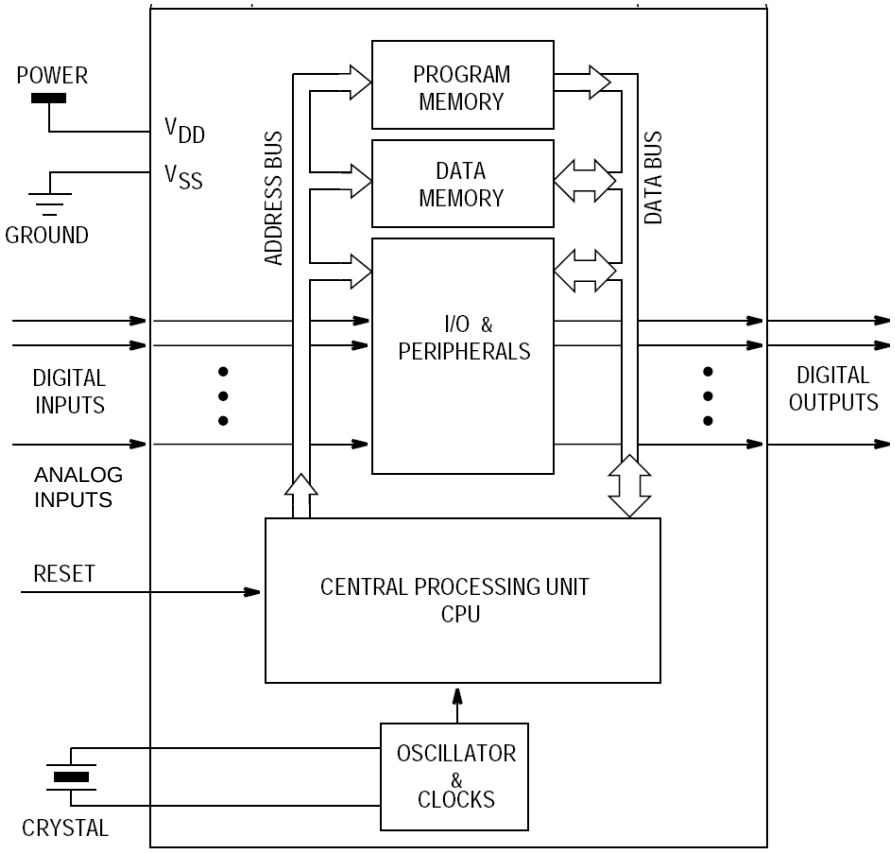
- One of the most critical needs of an embedded system is to decrease power consumption and space
- In high-performance embedded processors, the trend is to integrate more functions on the CPU chip and let designer decide which features he/she wants to use
- In many cases using x86 PCs for the high-end embedded applications
  - Saves money and shortens development time
    - A vast library of software already written
    - Windows is a widely used and well understood platform

Most common MCU/CPU

- 8-bit MCU
  - AVR
  - PIC
  - 8051
- 16-bit MCU/CPU
  - 8086/186/96/196
  - 68HC12/16
- 32-bit MCU
  - ARM
  - PIC32

What is a Microcontroller?





What is the difference between a '*Digital* Input' and an '*Analog* Input'?

MicroController

A microcontroller is a computer with most of the necessary support chips onboard. 'Embedded' inside some other device (often a consumer product) so that they can control the features or actions of the product. Another name for a microcontroller is therefore an 'embedded controller'. All computers have several things in common, namely:

- A central processing unit (CPU) that 'executes' programs.
- Some random-access memory (RAM) where it can store data that is variable.
- Some read only memory (ROM) where programs to be executed can be stored.
- Input and output (I/O) devices that enable communication to be established with the outside world i.e. connection to devices such as keyboard, mouse, monitors and other peripherals.
- There are a number of other common characteristics that define microcontrollers. If a computer matches a majority of these characteristics, then it can be classified as a 'microcontroller'.

Choosing a microcontroller

- **Speed**: What is the highest speed a microcontroller supports?
- **Packaging**: Is it DIP (dual inline package) or a QFP (quad flat package) or some other type?
- **Power Consumption**: Critical for battery powered products: How easy to upgrade to higher performance or lower power-consumption versions.
- The amount of **RAM** and **ROM** on chip
- The number of **timers** and **I/O pins** on chip
- **Cost** per unite
- Availability of **Compiler**, **Simulator**, **Debugger**: Availability of software development tools, such as compilers, assemblers, debuggers, C compilers, emulator, simulator, technical support
- Availability of chip in **market**

RISC and CISC

- **CISC (Complex Instruction Set Computer)**
  - A large number of instructions, typically from 100 to 250 instructions. Some instructions that perform specialized task and are used infrequently. Variable-length instruction formats.
  - A large variety of addressing modes, typically from 5 to 20 different modes.
  - Example: **MULT** is what is known as a "complex instruction." Instruction **does'nt** complete in one cycle execution. Processor hardware that is capable of understanding and executing a series of operations.
- **RISC (Reduced Instruction Set Computer)**
  - Relatively few instructions, Relatively few addressing modes.
  - Fixed-length, easily decoded instruction format.
  - Architecture which reduces the chip complexity by simpler processing instructions: RISC architecture CPUs capable of executing only a very limited (simple) set of instructions.

RISC and CISC

- **RISC (Reduced Instruction Set Computer)**
  - RISC processors only use simple instructions that can be executed within one clock cycle. "MULT" command divided into three separate commands:

```
LOAD   A, 2:3
LOAD   B, 5:2
PROD   A, B
STORE  2:3, A
```

Single Cycle Execution

- Fast Execution of Instructions due to simple instructions for CPU.
- RISC chips require fewer transistors, which makes them cheaper to design and produce.
- Emphasis on software
- Single-clock,reduced instruction only
- Register to register: "LOAD" and "STORE" are independent instructions
- Spends more transistors on memory registers

Harvard and Von Neumann Architecture

- **Von Neumann (Princeton) architecture.**
  - The same bus is used for accessing both the code and data.
  - Pentium Processor is based on von Neumann Architecture.
  - CPU can Read an instruction or data from/to the memory.
  - Read, Write can`t occur at the same time due to same memory and signal pathway for data and instructions.

Harvard and Von Neumann Architecture

Harvard architecture

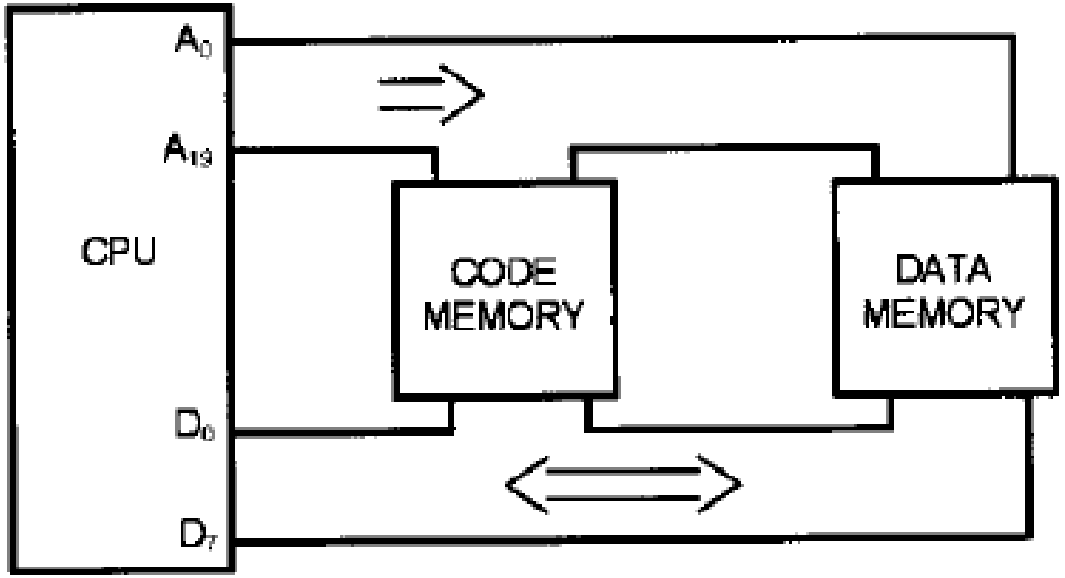
- Separate buses are used for accessing the code and data memory. Computer architectures that used physically separate storage and signal pathways for their instructions and data.
- That means that we need four sets of buses:
  1. **A set of data buses for carrying data into and out of the CPU,**
  2. **A set of address buses for accessing the data,**
  3. **A set of data buses for carrying code into the CPU, and**
  4. **An address bus for accessing the code**

Harvard and Von Neumann Architecture

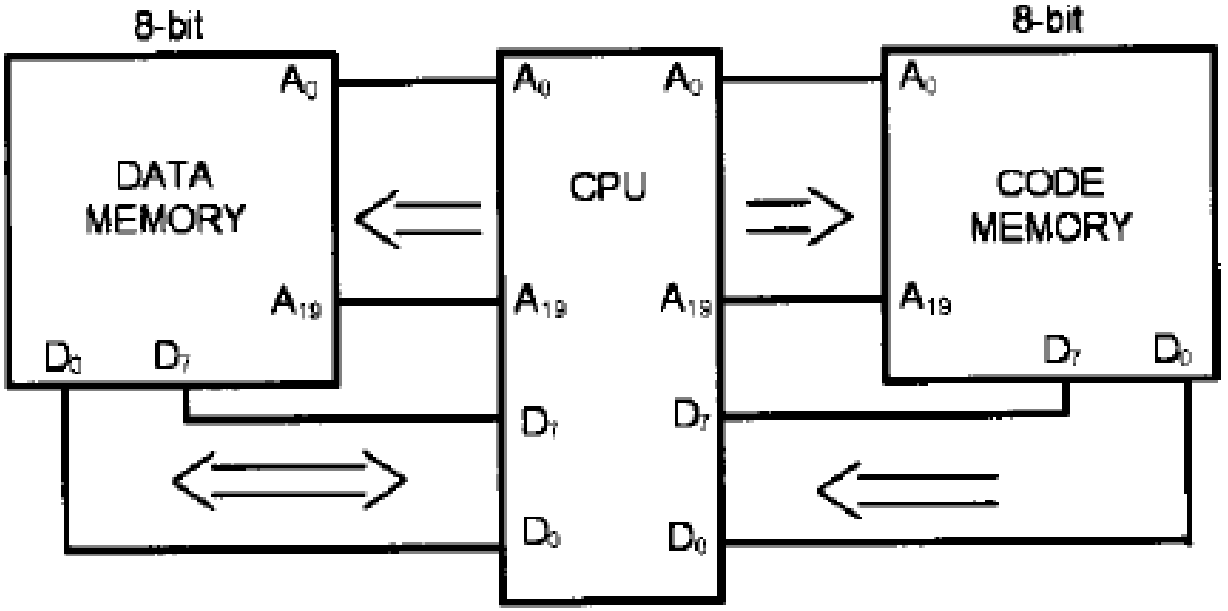
Harvard architecture

- This is easy to implement inside an IC chip such as a microcontroller where both ROM code and data RAM are internal (on-chip) and distances are on the micron and millimeter scale.
- CPU can read both an instruction and data from memory at the same time that makes it faster.

von Neumann Architecture



Harvard Architecture



Overview of 8051 Microcontroller

- Made by intel in 1981: developed by Intel in 1981 for use in embedded systems.
- An 8 bit, single-chip microcontroller optimized for control applications.
  - Harvard architecture (separate instruction/data memories)
  - today largely superseded by a vast range of faster and/or functionally enhanced 8051-compatible devices manufactured by more than 20 independent manufacturers.
- 128 bytes RAM, 4096 bytes (4KB) ROM, 2 timers, 1 serial port, 4 I/O ports.
- 40 pins in dual in dual in-line package (DIP) layout.

Specific Features

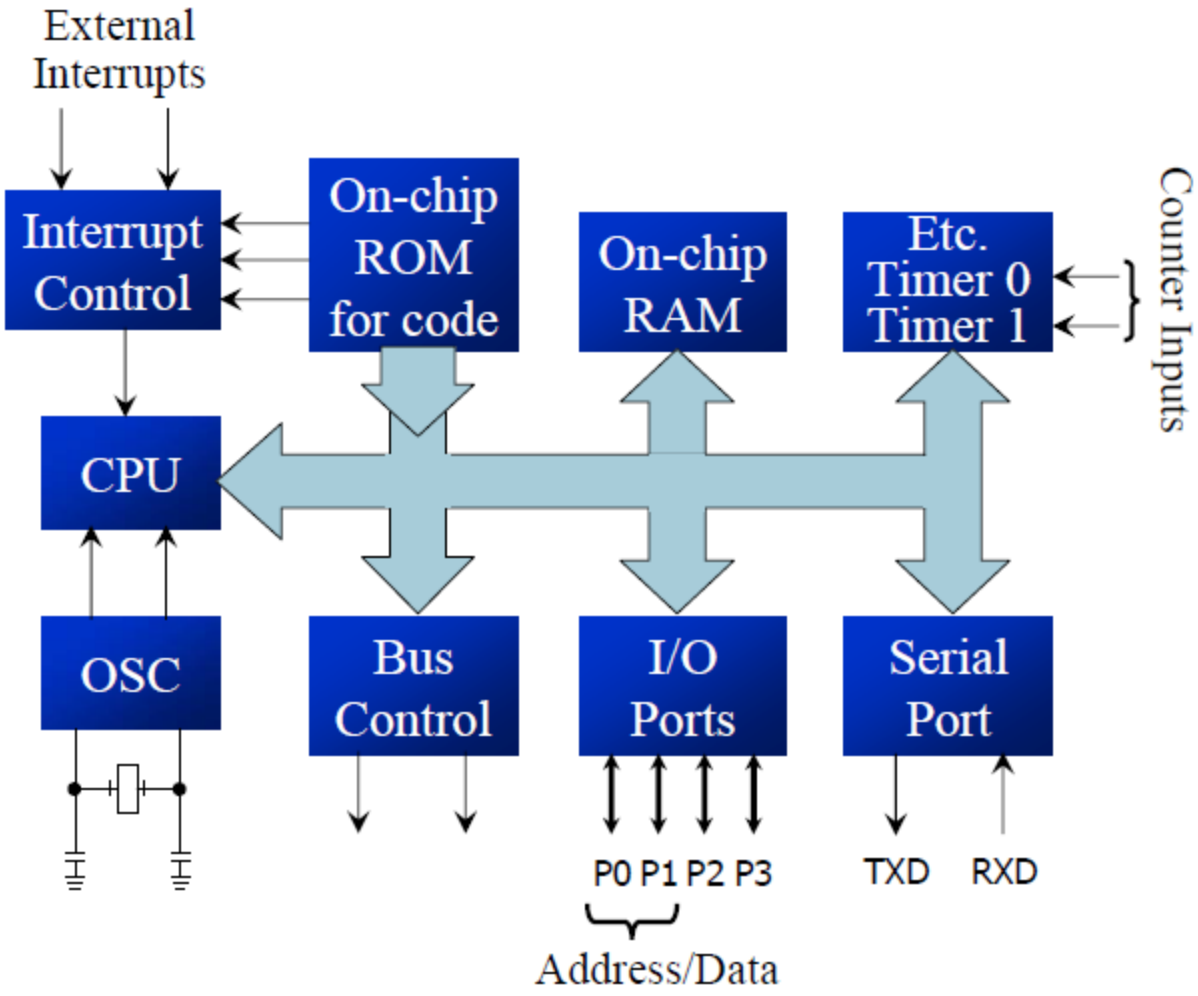
- 8 bit CPU with registers A and B: *Currently*, the most widely used microcontroller.
- 16 bit PC and DPTR(data pointer).
- 8 bit program status word(PSW)
- 8 bit Stack Pointer
- 2 distinct separately addressable memory areas.
  - Maximum of 64K on-chip ROM.
    - Usually 0 to 4K.
  - Maximum of 64K external data memory.
  - Maximum of 64K external code memory.
  - Basic version (8051) contains:
    - 4K Bytes of on-chip ROM instruction memory.
    - 128 Bytes of on-chip RAM for temporary data storage and the stack.
- 4K Internal ROM

Specific Features

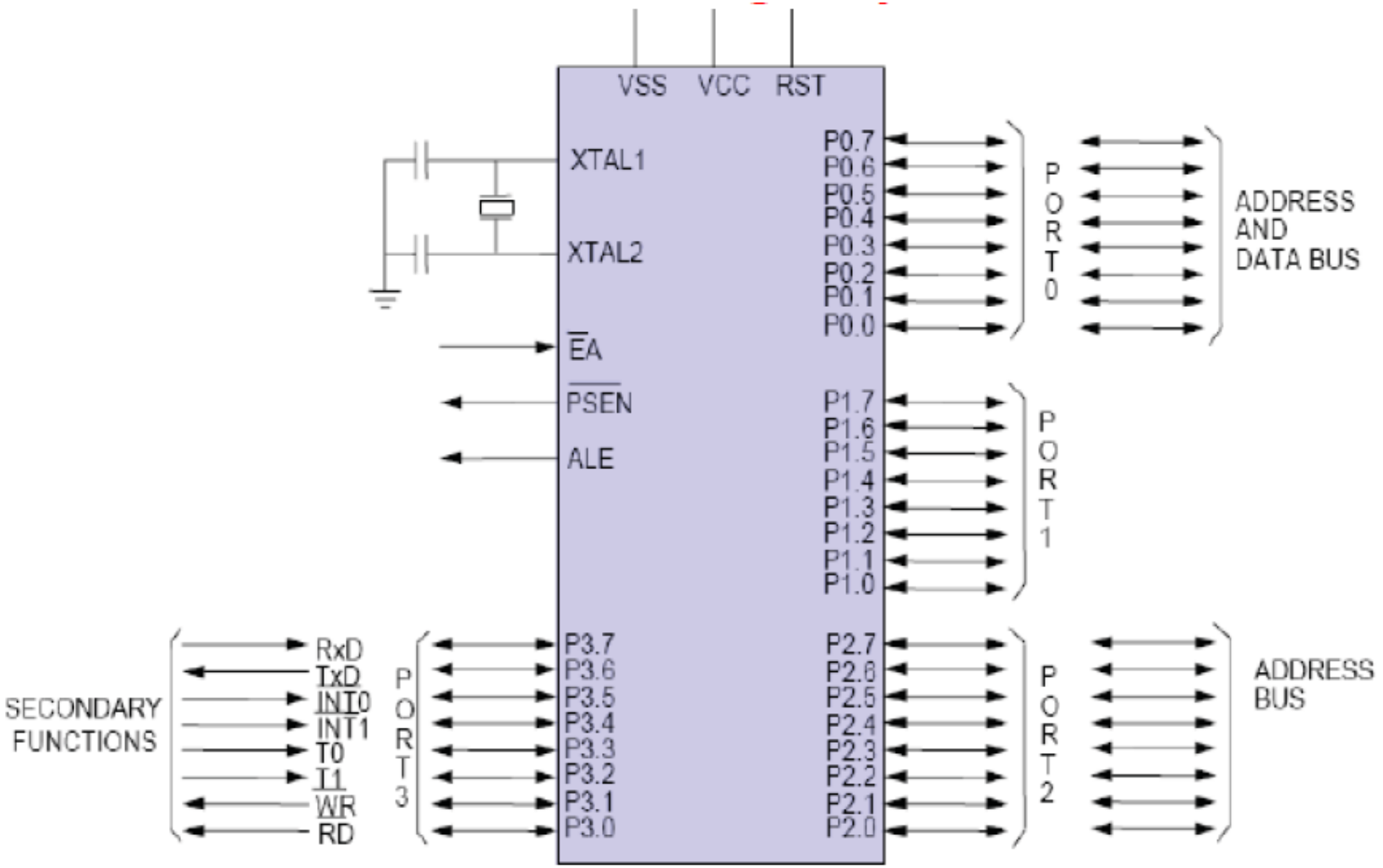
- 128bytes Internal RAM
  - 4 register banks each having 8 registers (Each bank has R0-R7 Selectable by psw.2,3 [RS1,RS0] bit.)
  - 16 bytes, which may be addressed at the bit level:
  - 80 bytes of general purpose data memory
- 32 i/o pins arranged as 4-8 bit ports:P0 to P3
- Two 16 bit timer/counters:T0(TH0 and TL0) and T1 (TH1 and TL1).
- Full duplex serial data receiver/transmitter
- Control registers:TCON,TMOD,SCON,PCON,IP and IE
- Two external and Three internal interrupt sources.
- Oscillator and Clock Circuits.
- Boolean Instructions work with 1 bit at a time
- Assume clock Frequency =11.0592MHz, it takes about 4.34µs (i.e. 4.34×10^{-6}s) to carry out a 8 bit multiplication instruction.

19

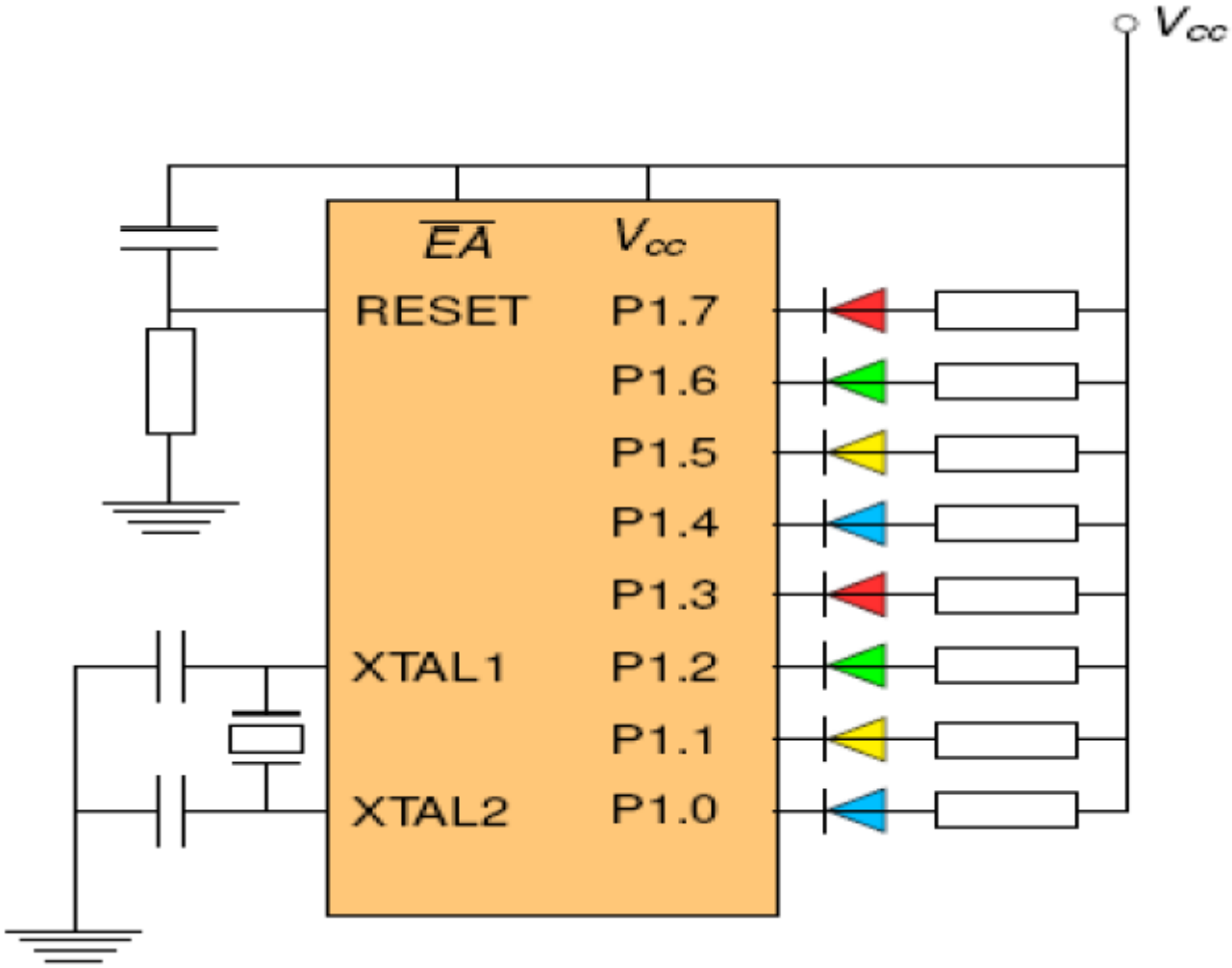
8051 Microcontroller architecture



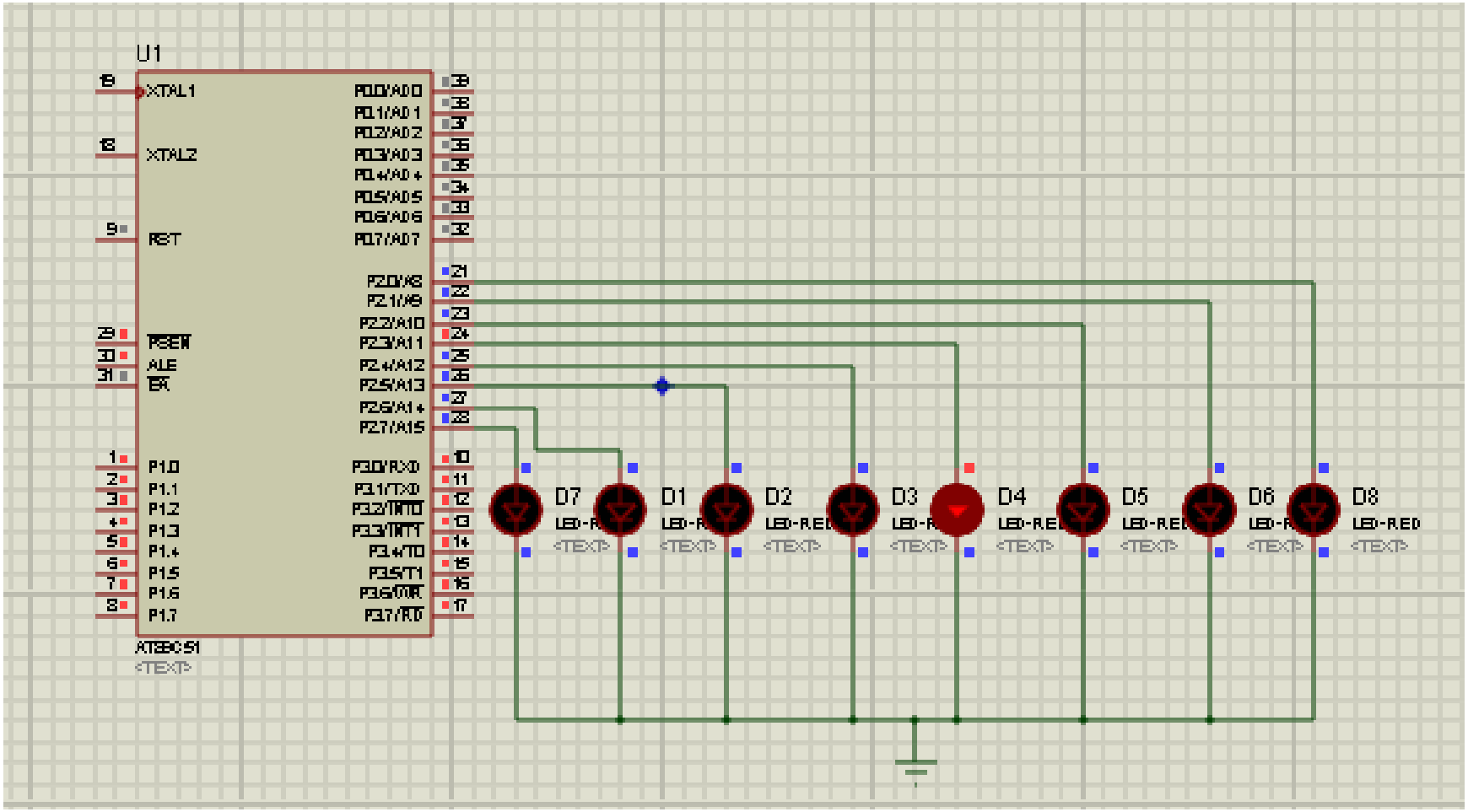
8051 Microcontroller architecture



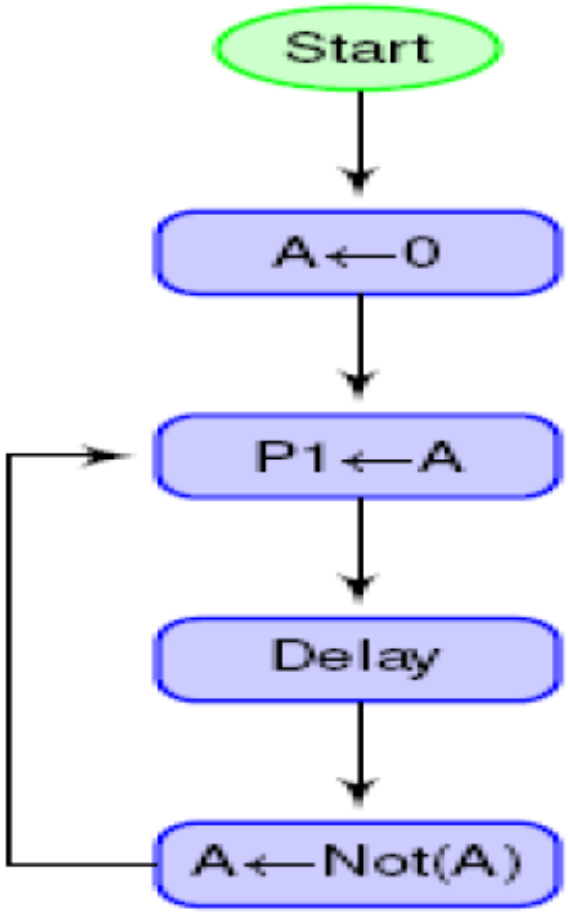
Microcontroller 8051: Application 1



Microcontroller 8051: Application 1



Microcontroller 8051: Application 1




```
        ORG 0000
        CLR A
LOOP :  MOV P1, A
        CPL A
        ACALL DELAY ;100 ms
        AJMP LOOP
DELAY:  MOV R6, #250
DL1 :   MOV R7, #200
DL2 :   DJNZ R7,DL2
        DJNZ R6,DL1
        RET
END
```

Microcontroller 8051: Application 1

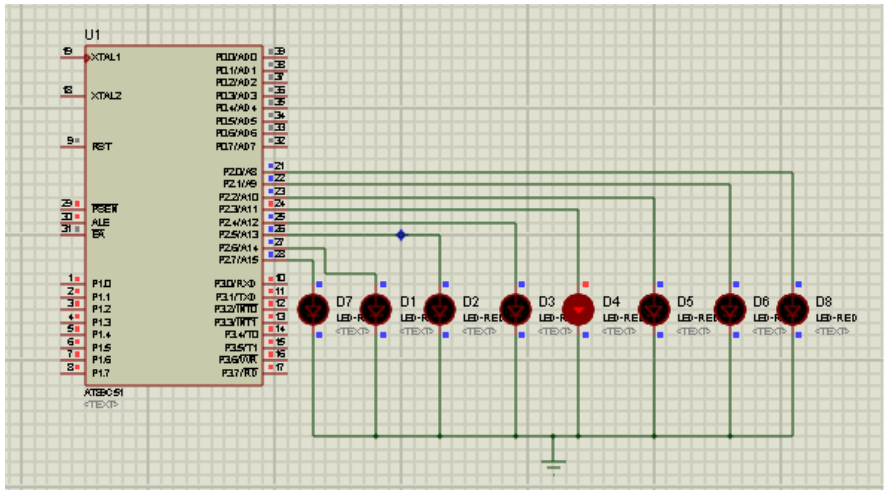
```
#include<reg52.h>
#include<stdio.h>
#define relayport P2
void Delay_ms(int);
void main()
{
relayport = 0x00; //All relays Off
do
{
relayport = 0xFF; //All relays On
Delay_ms(1000);
relayport = 0x00; //All relays Off
Delay_ms(1000);
}while(1);
}
```

```
void Delay_ms(int k)
{
int j;
int i;
for(i=0;i<k;i++)
{
for(j=0;j<1000;j++)
}
}
```

Microcontroller 8051: Application 2

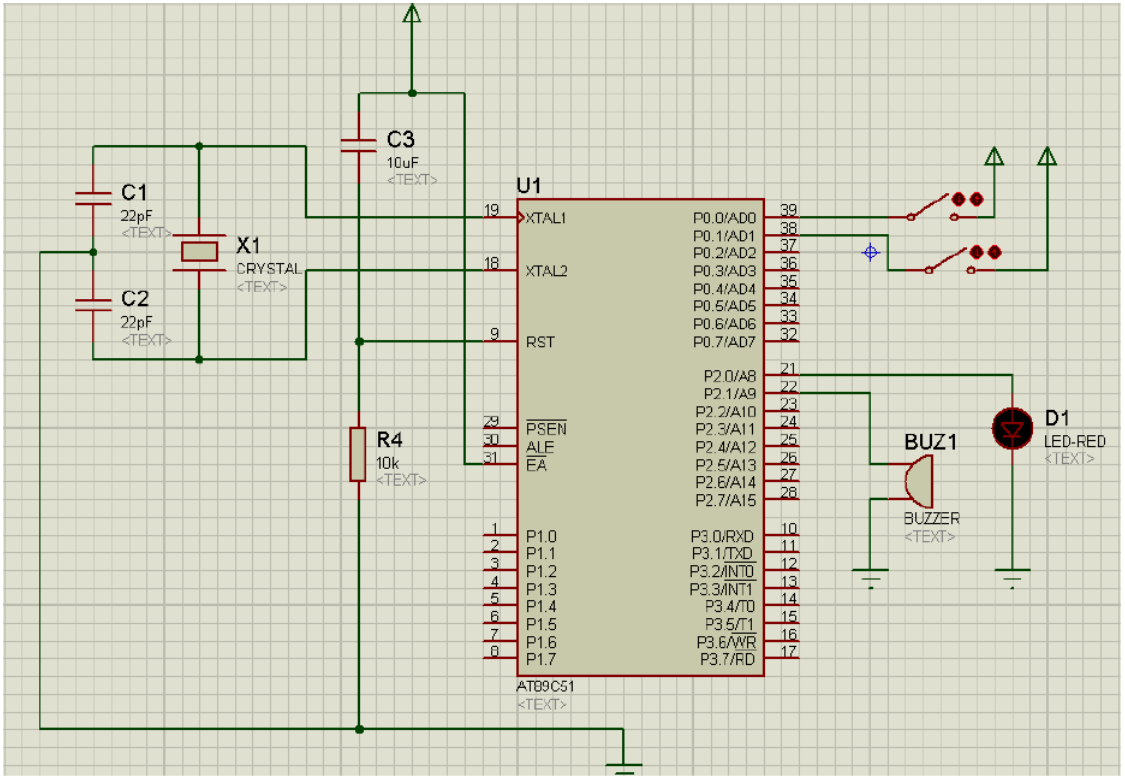
```
#include<reg52.h>
#include<stdio.h>
#define relayport P2
void Delay_ms(int);
void main()
{ int i;
int value=0x01;
relayport = 0x00;
do
{
value=0x01;
for (i=0;i<=7;i++){
relayport = value;
Delay_ms(100);
value=value <<1;
}
}while(1);
}
```

```
void Delay_ms(int k)
{
int j;
int i;
for(i=0;i<k;i++)
{
for(j=0;j<1000;j++)
}
}
```

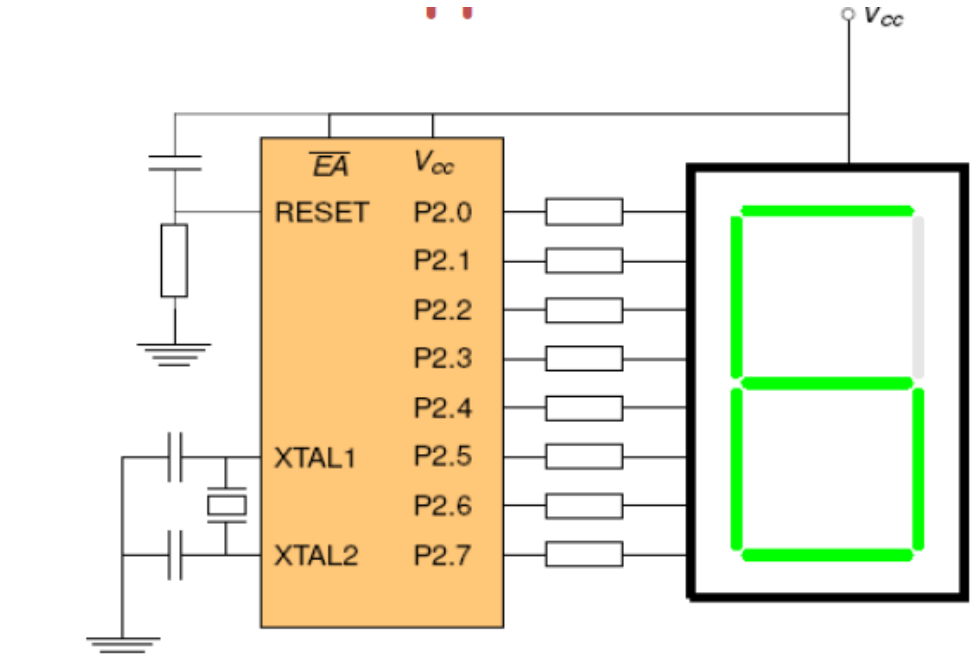


Microcontroller 8051: Application 3

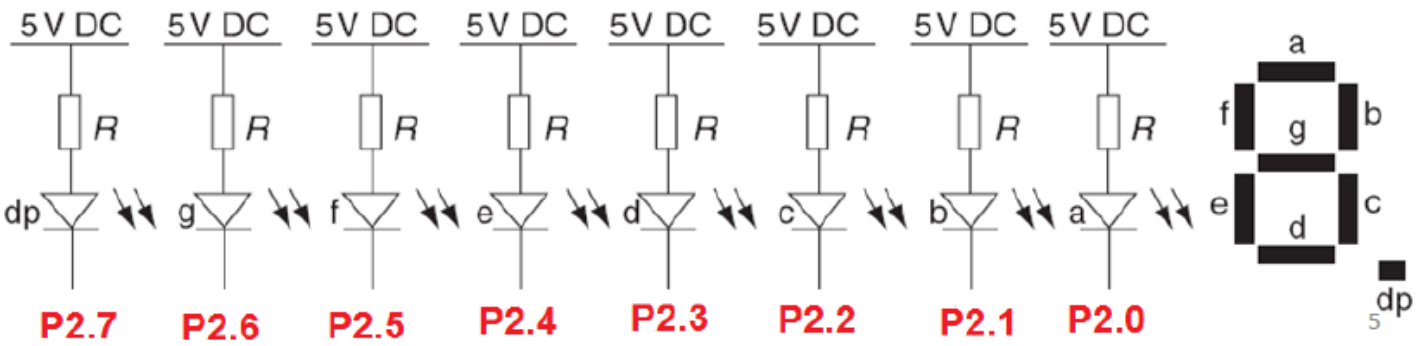
```
#include <reg52.h> /* special function register declarations */
sbit LED_pin = P2^0; //Defining LED PIN
sbit buzzer = P2^1;
sbit switch_pin1 = P0^0; //Defining Switch PIN1
sbit switch_pin2 = P0^1; //Defining Switch PIN2
void main (void)
{
if(switch_pin1 == 1)
{ LED_pin = 1;
}
else {
LED_pin = 0;
}
if( switch_pin2 == 1)
{ buzzer = 1;
}
else
{
buzzer = 0;
}
}
```



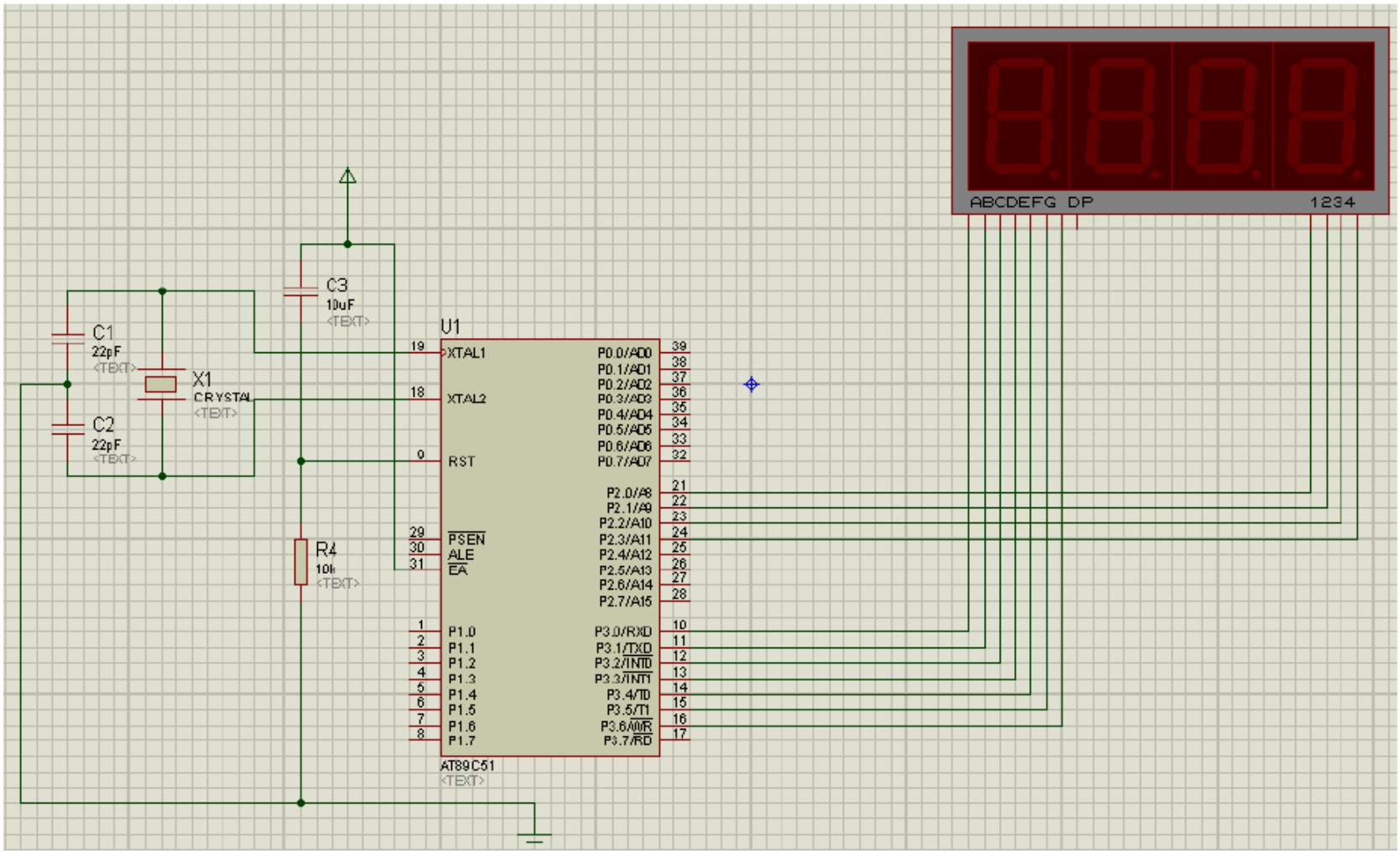
Microcontroller 8051: Application 4



Write the decimal value 4 on the SSD in the following figure. Switch the decimal point off.



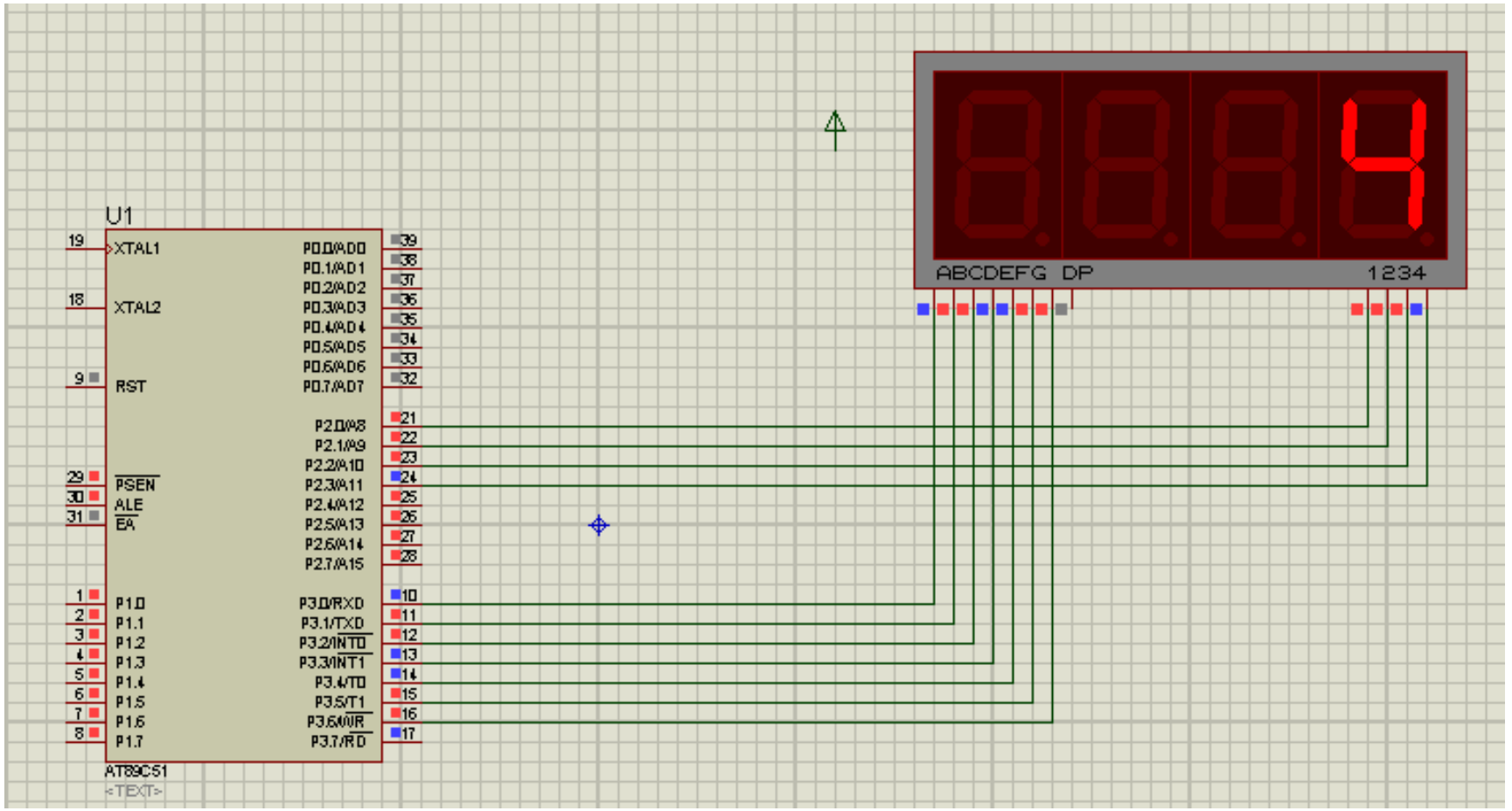
Microcontroller 8051: Application 4



Microcontroller 8051: Application 4

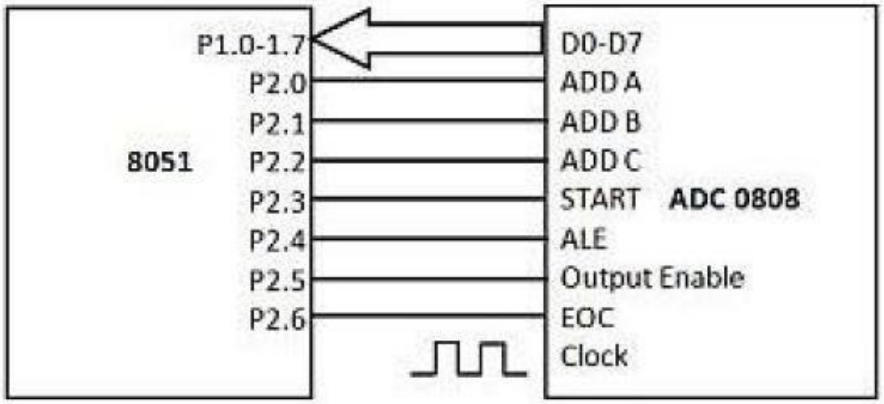
```
#include <REGX52.H>
#define Seg_Data P3 /*port initialized for 7 segment*/
#define High 1
#define Low 0
sbit SegmentACntrl = P2^0;
sbit SegmentBCntrl = P2^1;
sbit SegmentCCntrl = P2^2;
sbit SegmentDCntrl = P2^3;
/******** Delay ********/
void Delay(unsigned int wait)
{
unsigned int var1,var2;
for(var1=0;var1<=wait;var1++)
for(var2=0;var2<=1275;var2++);
}
/*Segment_Disp: This function provide digit 0 to 9 */
void Segment_Disp()
{
Seg_Data=0x3F; Delay(100);
Seg_Data=0x06; Delay(100);
Seg_Data=0x5B; Delay(100);
Seg_Data=0x4F; Delay(100);
Seg_Data=0x66; Delay(100);
Seg_Data=0x6D; Delay(100);
Seg_Data=0x7D; Delay(100);
Seg_Data=0x07; Delay(100);
Seg_Data=0x7F; Delay(100);
Seg_Data=0x6F; Delay(100);
}
/********Main Function ******************/
void main()
{
while(1) /* Forever loop*/
{
SegmentACntrl=High; /* Control Pins for
Seven segment */
SegmentACntrl=High;
SegmentBCntrl=High;
SegmentCCntrl=High;
SegmentDCntrl=Low;
Segment_Disp();
}
}
```

Microcontroller 8051: Application 4



Microcontroller 8051: Application 5

- In lot of embedded systems microcontrollers need to use analog input because most of the sensors AND transducers such as temperature, humidity, pressure, are analog.
- For interfacing these sensors to microcontrollers we require to convert the analog output of these sensors to digital so that the microcontroller can process it.
- One of the most commonly used ADC is ADC0808 (8-bit data output).
- ADC 0808 is a successive approximation type with 8 channels i.e. it can directly access 8 single ended analog signals.



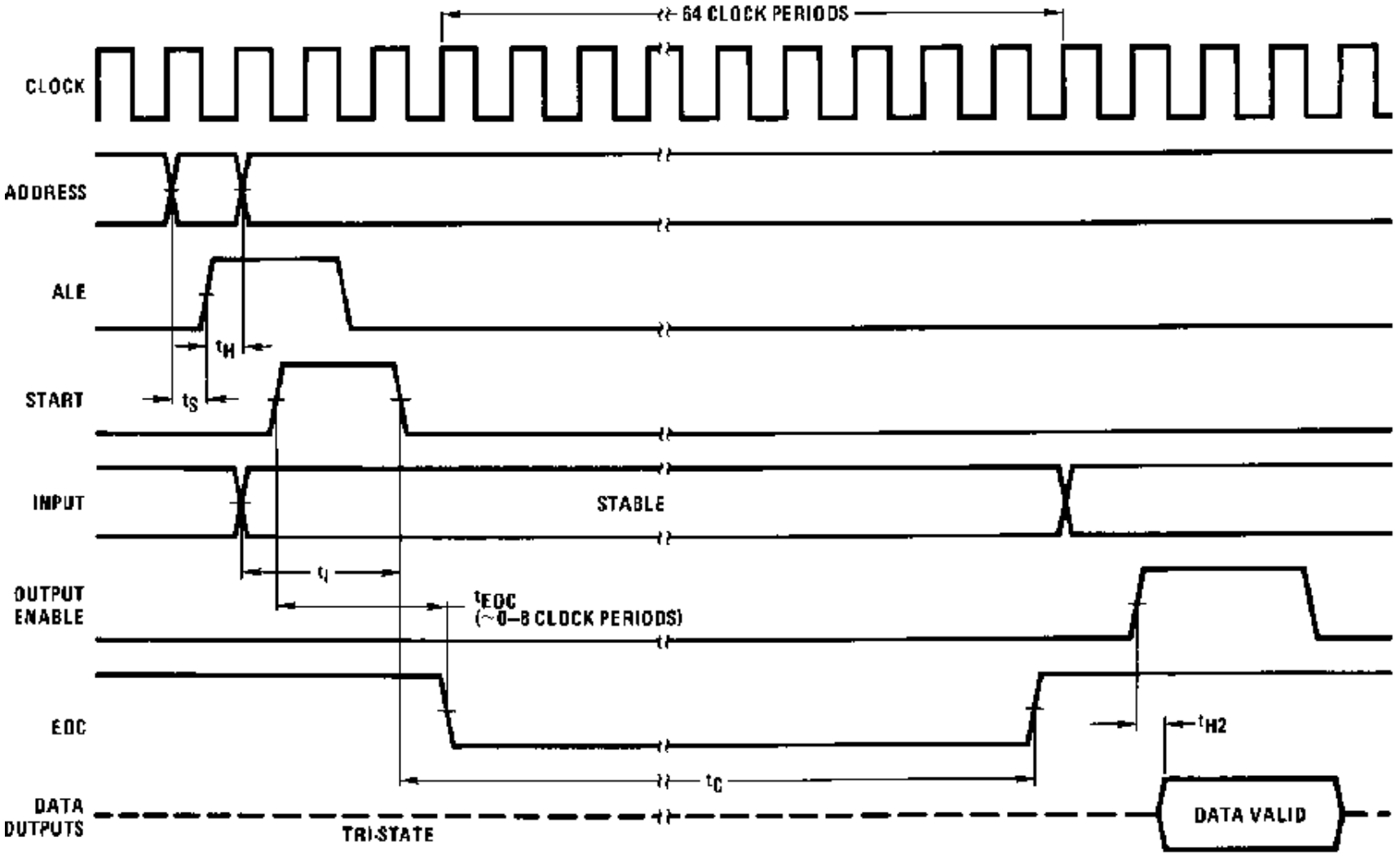
Microcontroller 8051: Application 5

- **Input pins (INT0-INT7):** The ADC 0808 has eight input analog pins. These pins are multiplexed together, and only one of them can be selected using three select lines.
- **Select lines and ALE :** It has three select lines, namely A, B, and C, that are used to select the desired input lines. The ALE pin also needs to be activated by a low to high pulse to select a particular input.
- **Output pins (D0-D7) :** The ADC has eight output pins that give the binary equivalent of a given analog value.
- **VCC and Ground :** These two pins are used to provide the required voltage to power the microcontroller. In most cases, the ADC uses 5V DC to power up.
- **Clock :** The 0808 does not have an internal clock and needs an external clock signal to operate. It uses a clock frequency of 20Mhz, and using this clock frequency it can perform one conversion in 100 microseconds.

Microcontroller 8051: Application 5

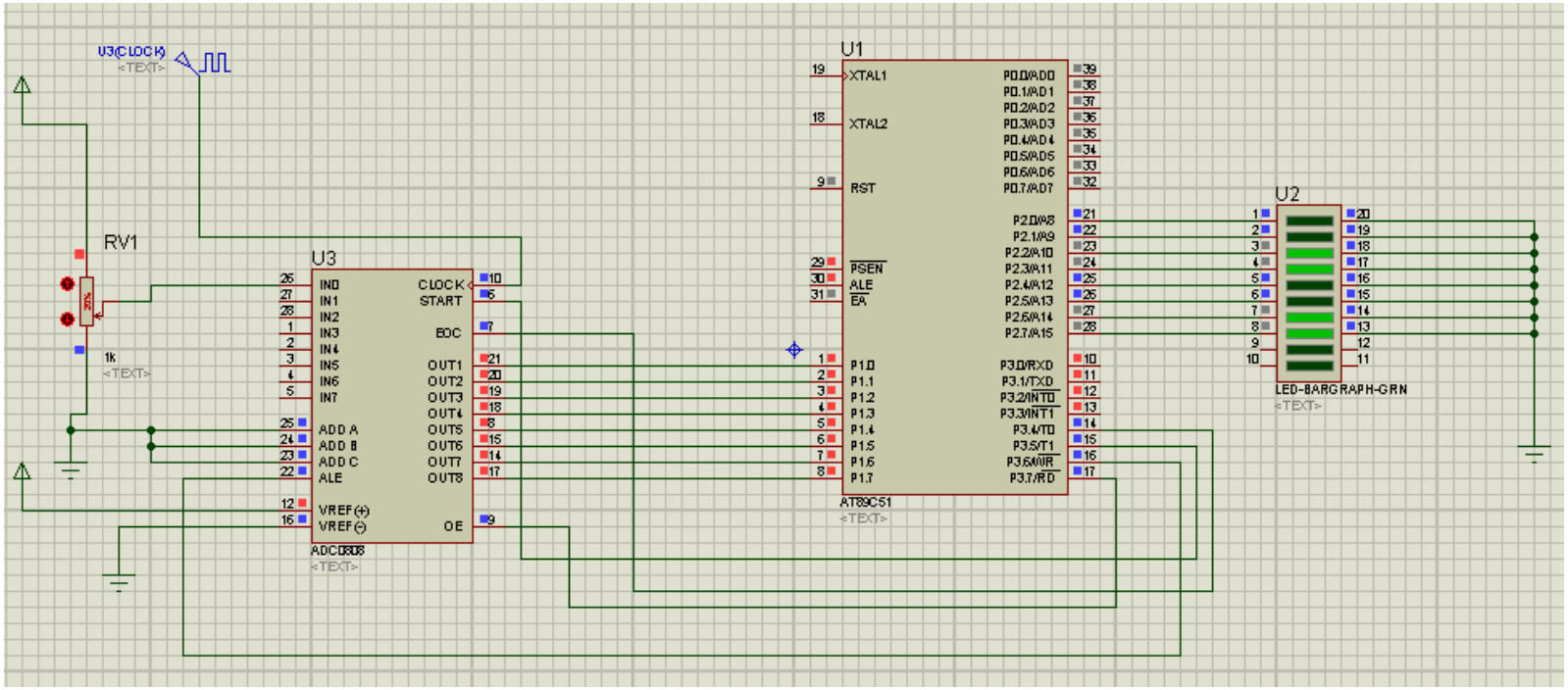
- **VREF (+) and VREF (-):** These two pins are used to provide the upper and the lower limit of voltages which determine the step size for the conversion. Here Vref(+) has a higher voltage, and Vref(-) has the lower voltage. If Vref(+) has an input voltage 5v and Vref(-) has a voltage of 0v then the step size will be 5v-0v/28= 19.53 mv.
- **Start conversion :** This pin is used to tell the ADC to start the conversion. When the ADC receives a low to high pulse on this pin, it starts converting the analog voltage on the selected pin to its 8-bit digital equivalent.
- **End of conversion:** Once the conversion is complete, the ADC sends low to high signal to tell a microcontroller that the conversion is complete and that it can extract the data from the 8 data pins.
- **Output enable :** This pin is used to extract the data from the ADC. A microcontroller sends a low to high pulse to the ADC to extract the data from its data buffers

Microcontroller 8051: Application 5



Microcontroller 8051: Application 5

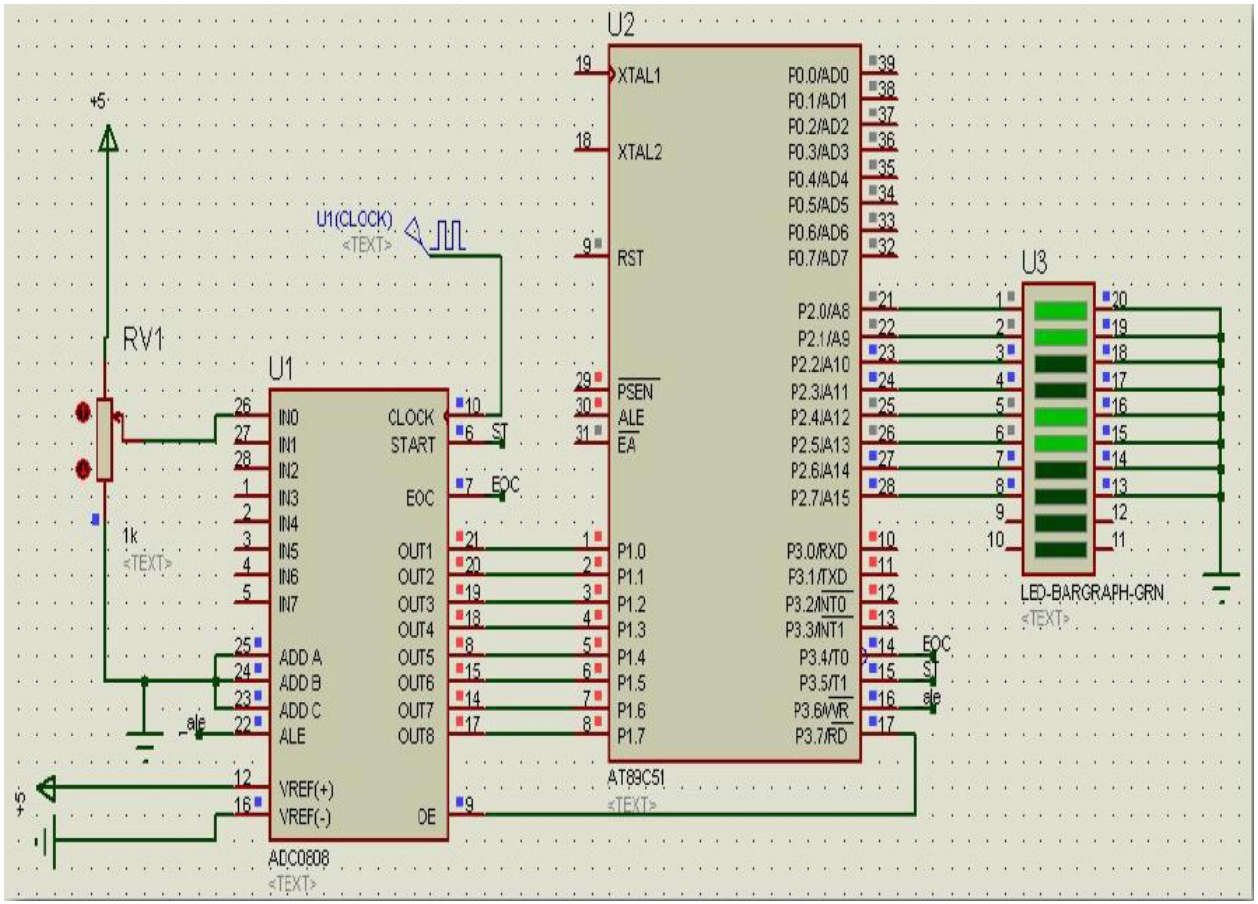
- Analog input (IN0) is converted to digital value (Out1-Out8) connected to P1 of the microcontroller then displayed on LEDs through port P2.
- The potentiometer is used instead of sensor and it allows to vary voltage.



Microcontroller 8051: Application 5

- Analog value changed, so digital value also.

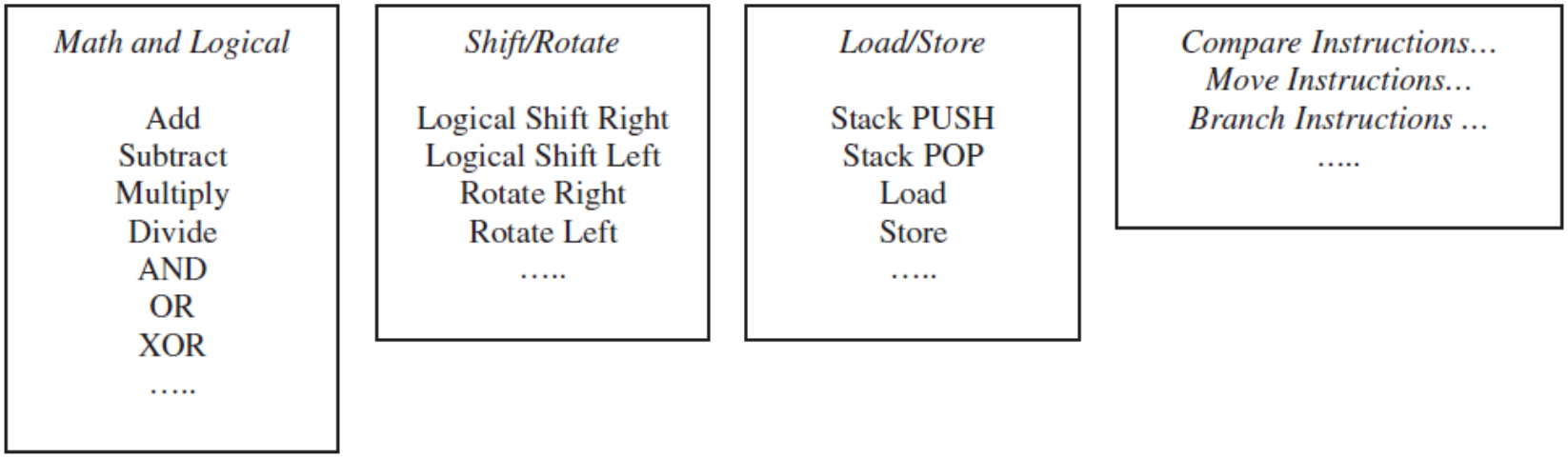
```
#include <reg51.h>
sbit OE = P3 ^ 7 ;
sbit ALE = P3 ^ 6 ;
sbit ST = P3 ^ 5 ;
sbit EOC = P3 ^ 4 ;
void main()
{
ALE = 1 ;
ALE = 0 ;
while(1)
{
ST = 1 ;
ST = 0 ;
while ( ! EOC ) ;
OE = 1 ;
P2 = P1 ;
OE = 0 ;
 }
 }
```



ISA Architecture Models

- The *features* that are built into an architecture's instruction set are commonly referred to as the ***Instruction Set Architecture*** or ***ISA.***
- The ISA defines such features as the *operations* that can be used by programmers to create programs for that architecture, the *operands* (data) that are accepted and processed by an architecture, *storage*, *addressing modes* used to gain access to and process operands, and the handling of *interrupts*.
- These features are described in more detail next, because an ISA implementation is a determining factor in defining important characteristics of an embedded design, such as performance, design time, available functionality, and cost.

ISA Architecture : Operations



Samples ISA operations

- Different processors can have similar types of operations, but usually have different overall instruction sets.
- What is also important to note is that different architectures can have operations with the same purpose (add, subtract, compare, etc.), but the operations may have different names or internally operate much differently.

Operands

- Operands are the data that operations manipulate.
- An ISA defines the types and formats of operands for a particular architecture.
- For example, in the case of the MPC823 (Motorola/Freescale PowerPC), SA-1110 (Intel StrongARM), and many other architectures, the ISA defines simple operand types of bytes (8 bits), halfwords (16 bits), and words (32 bits).
- An ISA also defines the operand formats (how the data looks) that a particular architecture can support, such as binary, decimal and hexadecimal.
- **MOV registerX, 10d ; Move decimal value 10 into register X**
- **MOV registerX, $0Ah ; Move hexadecimal value A (decimal 10) to register X**
- **MOV registerX, 00001010b ; Move binary value 00001010 (decimal 10) to register X**

Storage

- One of many ways processors can be referenced is according to the *size* (in bits) of **data** that can be processed and the *size* (in bits) of the **memory space** that can be addressed in a single instruction by that processor.
- Commonly used embedded processors support 4-bit, 8-bit, 16-bit, 32-bit, and/or 64-bit processing.
- Some processors can process larger amounts of data and can access larger memory spaces in a single instruction, such as 128-bit architectures, but they are not commonly used in embedded designs.

"x"-Bit	Architecture
4	Intel 4004, …
8	Mitsubishi M37273, 8051, 68HC08, Intel 8008/8080/8086, …
16	ST ST10, TI MSP430, Intel 8086/286, …
32	68K, PowerPC, ARM, x86 (386+), MIPS32, …

Addressing mode

Addressing modes define how the processor can access operand storage. In fact, the usage of registers is partly determined by the ISA's ***Memory Addressing Modes.*** The two most common types of addressing mode models are the:

- *Load-Store Architecture*, which only allows operations to process data in registers, not anywhere else in memory. For example, the PowerPC architecture has only one addressing mode for load and store instructions: register plus displacement (supporting register indirect with immediate index, register indirect with index, etc.).

- *Register-Memory Architecture*, which allows operations to be processed both within registers and other types of memory. Intel's i960 Jx processor is an example of an addressing mode architecture based upon the register-memory model (supporting absolute, register indirect, etc.).

Interrupts and exception handling

- *Interrupts* (also referred to as exceptions or traps depending on the type) are mechanisms that stop the standard flow of the program in order to execute another set of code in response to some event, such as problems with the hardware, resets, and so forth.
- The ISA defines what if any type of hardware support a processor has for interrupts.

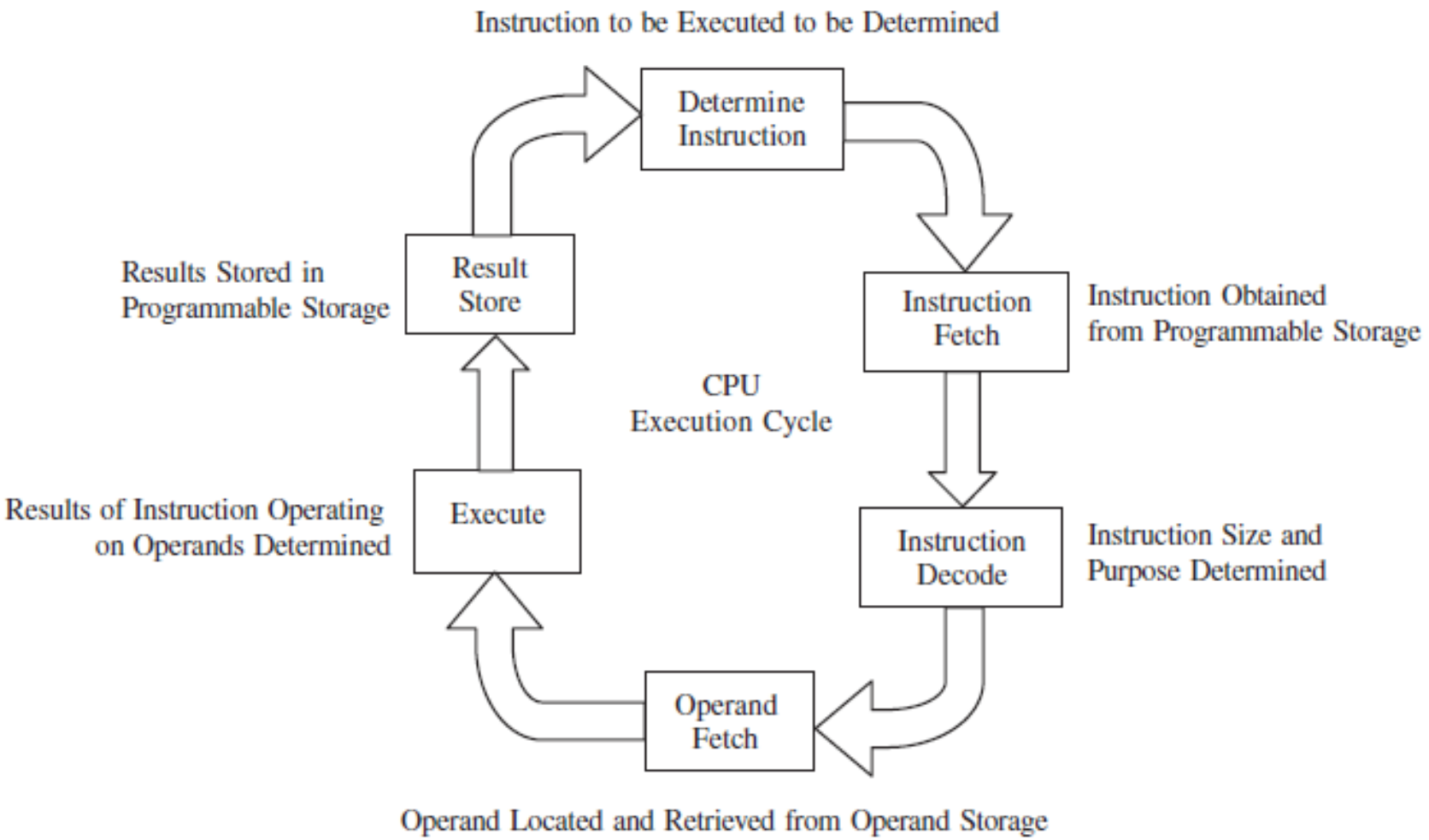
Internal Processor Design

- The ISA defines ***what*** a processor can do, and it is the processor's internal interconnected hardware components that physically implement the ISA's features.
- Interestingly, the fundamental components that make up an embedded board are the same as those that implement an ISA's features in a processor: a CPU, memory, input components, output components, and buses.
- These components are the basis of the von Neumann model.

Central Processing Unit (CPU)

- The semantics of this section can be a little confusing, because processors themselves are commonly referred to as CPUs, but it is actually the *processing unit* within a processor that is the CPU.
- The CPU is responsible for executing the cycle of fetching, decoding, and executing instructions.
- This three-step process is commonly referred to as a three stage *pipeline*, and most recent CPUs are pipelined designs.

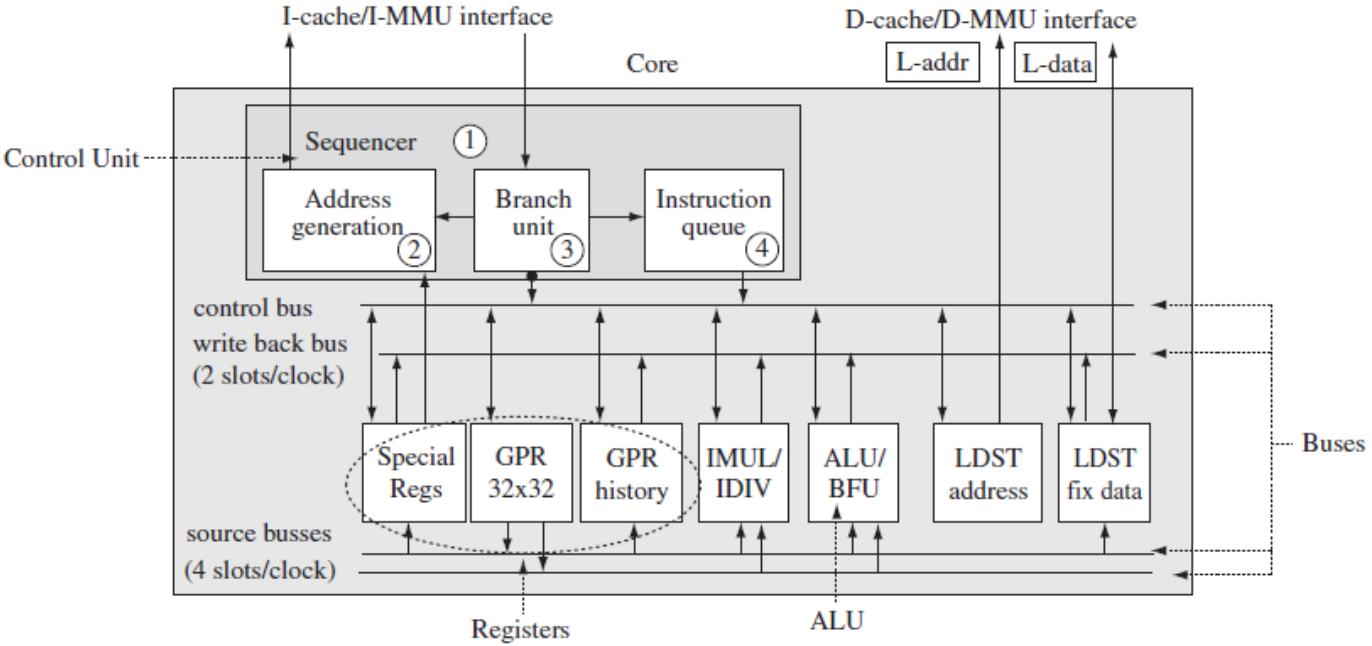
Central Processing Unit (CPU)



Central Processing Unit (CPU)

As defined by the von Neumann model, this cycle is implemented through some combination of four major CPU components:

- the arithmetic logic unit (ALU) – *implements the ISA's operations*
- registers – *a type of fast memory*
- the control unit (CU) – manages the entire fetching and execution cycle
- the internal CPU buses – interconnect the ALU, registers, and the CU



Internal CPU Buses

- The CPU buses are the mechanisms that interconnect the CPU's other components: the ALU, the CU, and registers.
- Buses are simply wires that interconnect the various other components within the CPU.
- Each bus's wire is typically divided into logical functions, such as data (which carries data, bi-directionally, between registers and the ALU), address (which carries the locations of the registers that contain the data to be transferred), control (which carries control signal information, such as timing and control signals, between the registers, the ALU, and the CU), and so on.

Arithmetic Logic Unit (ALU)

- The arithmetic logic unit (ALU) implements the comparison, mathematical and logical operations defined by the ISA.
- The format and types of operations implemented in the CPU's ALU can vary depending on the ISA.
- Considered the core of any processor, the ALU is responsible for accepting multiple *n*-bit binary operands and performing any logical (AND, OR, NOT, etc.), mathematical (+, –, *, etc.), and comparison (=, <, >, etc.) operations on these operands.
- The ALU is a combinational logic circuit that can have one or more inputs and only one output.
- An ALU's output is dependent only on inputs applied at that instant, as a function of time, and "no" past conditions.
- The basic building block of most ALUs (from the simplest to the multifunctional) is considered the ***full adder***, a logic circuit that takes three 1-bit numbers as inputs and produces two 1-bit numbers.

Registers

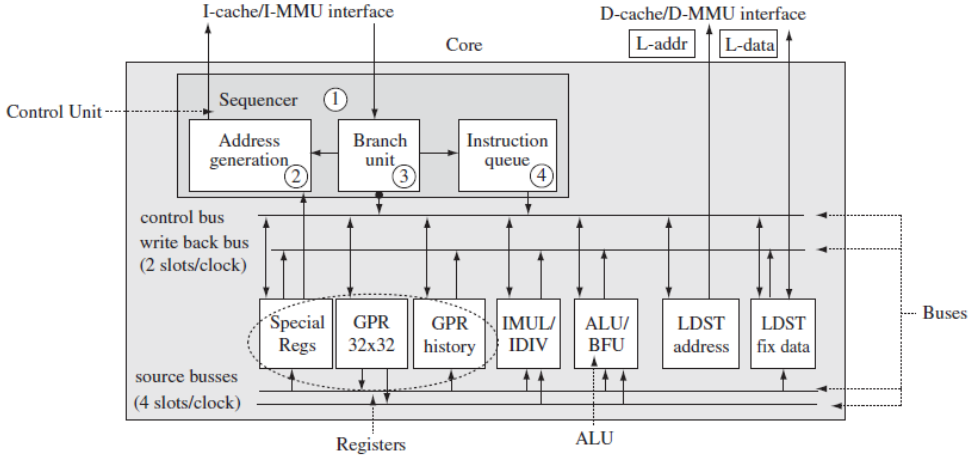
- Registers are simply a combination of various flip-flops that can be used to temporarily store data or to delay signals.
- A ***storage register*** is a form of fast programmable internal processor memory usually used to temporarily store, copy, and modify operands that are immediately or frequently used by the system.
- ***Shift registers*** delay signals by passing the signals between the various internal flip-flops with every clock pulse.
- Registers are made up of a set of flip-flops that can be activated either individually or as a set.
- In fact, it is *the number of flip-flops in each register* that is actually used to describe a processor (for example, a 32-bit processor has working registers that are 32 bits wide containing 32 flip-flops, a 16-bit processor has working registers that are 16 bits wide containing 16 flipflops, and so on).
- The number of flip-flops within these registers also determines the width of the data buses used in the system.

Control Unit (CU)

- The control unit (CU) is primarily responsible for generating timing signals, as well as controlling and coordinating the fetching, decoding, and execution of instructions in the CPU.
- After the instruction has been fetched from memory and decoded, the control unit then determines what operation will be performed by the ALU, and selects and writes signals appropriate to each functional unit within or outside of the CPU (i.e., memory, registers, ALU, etc.).
- To better understand how a processor's control unit functions, let's examine more closely the control unit of a PowerPC processor.

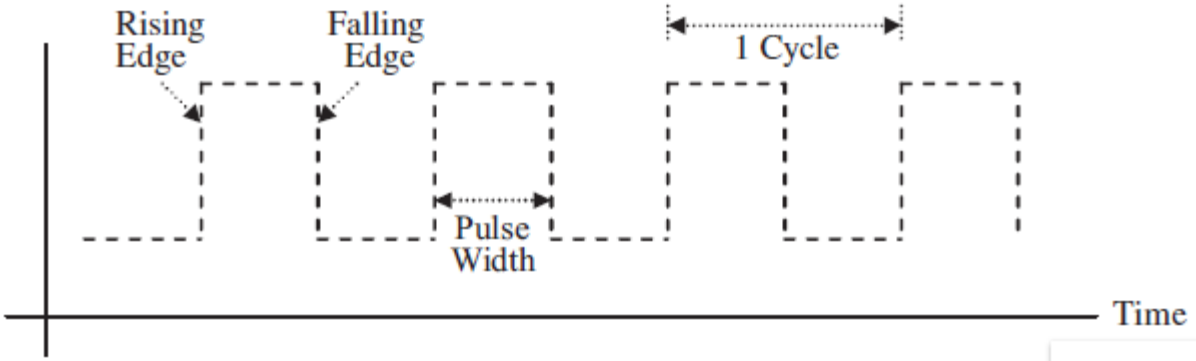
Control Unit (CU)

- The PowerPC core's CU is called a "sequencer unit," and is the heart of the PowerPC core.
- The sequencer unit is responsible for managing the continuous cycle of fetching, decoding, and executing instructions while the PowerPC has power, including such tasks as:
  - Provides the central control of the data and instruction flow among the other major units within the PowerPC core (CPU), such as registers, ALU and buses.
  - Implements the basic instruction pipeline.
  - Fetches instructions from memory to issue these instructions to available execution units.
  - Maintains a state history for handling exceptions.



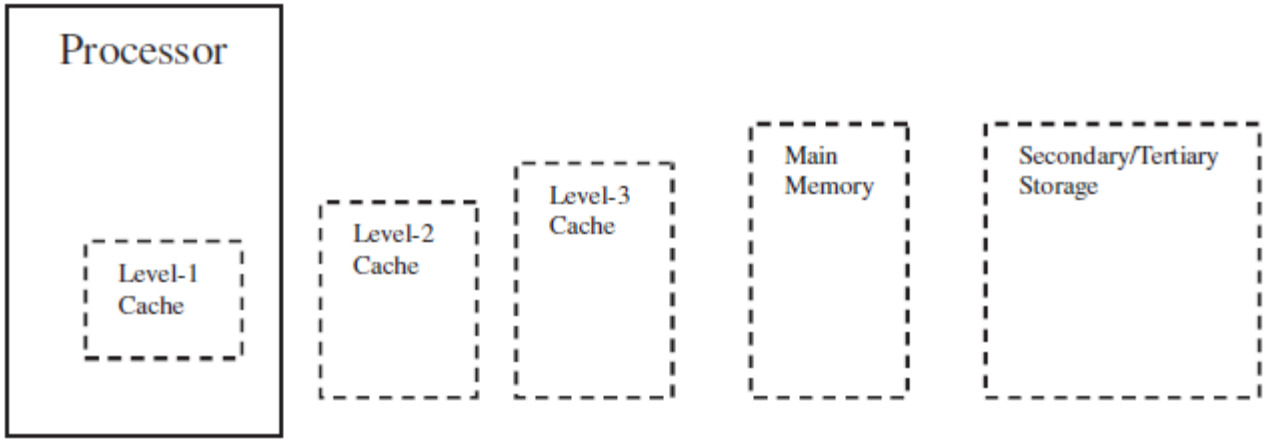
The CPU and the System (Master) Clock

- A processor's execution is ultimately synchronized by an external *system* or *master clock,* located on the board.
- The master clock is an oscillator which produces a fixed frequency sequence of regular on/off pulse signals (square waves).
- Components are driven by either the actual level of the signal (a "0" or a "1"), the rising edge of a signal (the transition from "0" to "1"), and/or the falling edge of the signal (the transition from "1" to "0").
- Different master clocks, depending on the circuitry, can run at a variety of frequencies, but typically must run so the slowest component on the board has its timing requirements met.
- In some cases, the master clock signal is divided by the components on the board to create other clock signals for their own use.



On-Chip Memory

- The CPU goes to memory to get what it needs to process, because it is in memory that all of the data and instructions to be executed by the system are stored.
- Embedded platforms have a *memory hierarchy*, a collection of different types of memory, each with unique speeds, sizes, and usages.
- Some of this memory can be physically integrated on the processor, such as registers, read-only memory (ROM), certain types of random access memory (RAM), and level-1 cache.

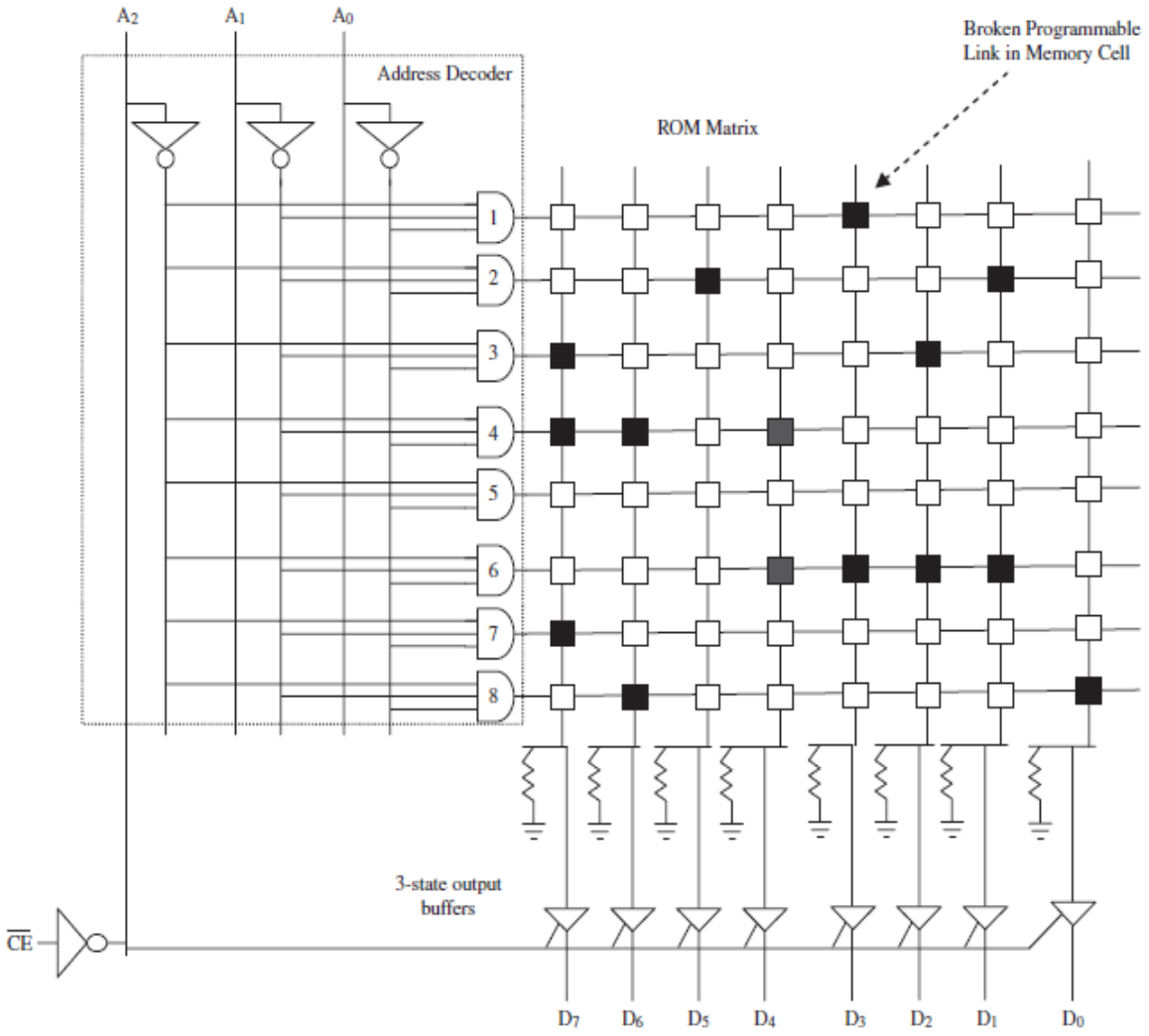


Read-Only Memory (ROM)

- On-chip ROM is memory integrated into a processor that contains data or instructions that remain even when there is no power in the system, due to a small, longer-life battery, and therefore is considered to be *nonvolatile memory*.
- ROM size specifications are represented in the real world (8 × 8, 16k × 32, and so on) reflects the actual size of ROM where the first number, preceding the "×", is the number of addresses, and the second number, after the "×", reflects the size of the data (number of bits) at each address location—8 = byte, 16 = half word, 32 = word, and so on.
- Also, note that in some design documentation, the ROM matrix size may be summarized.
- For example, 16 kB (kBytes) of ROM is 16K × 8 ROM, 32 MB of ROM is 32 M × 8 ROM, and so on.

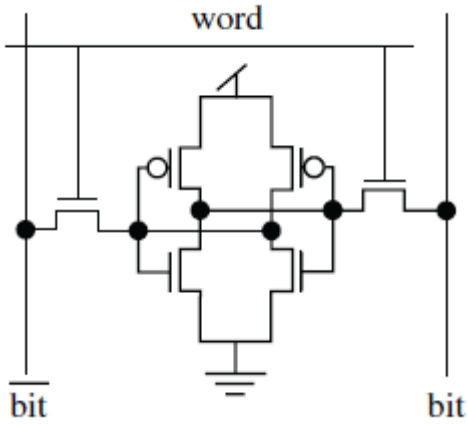
Read-Only Memory (ROM)

Gate	A2	A1	A0	D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	1	1	1	1	0	1	1	1
2	0	0	1	1	1	0	1	1	1	0	1
3	0	1	0	0	1	1	1	1	0	1	1
4	0	1	1	0	0	1	0	1	1	1	1
5	1	0	0	1	1	1	1	1	1	1	1
6	1	0	1	1	1	1	0	0	0	0	1
7	1	1	0	0	1	1	1	1	1	1	1
8	1	1	1	1	0	1	1	1	1	1	0



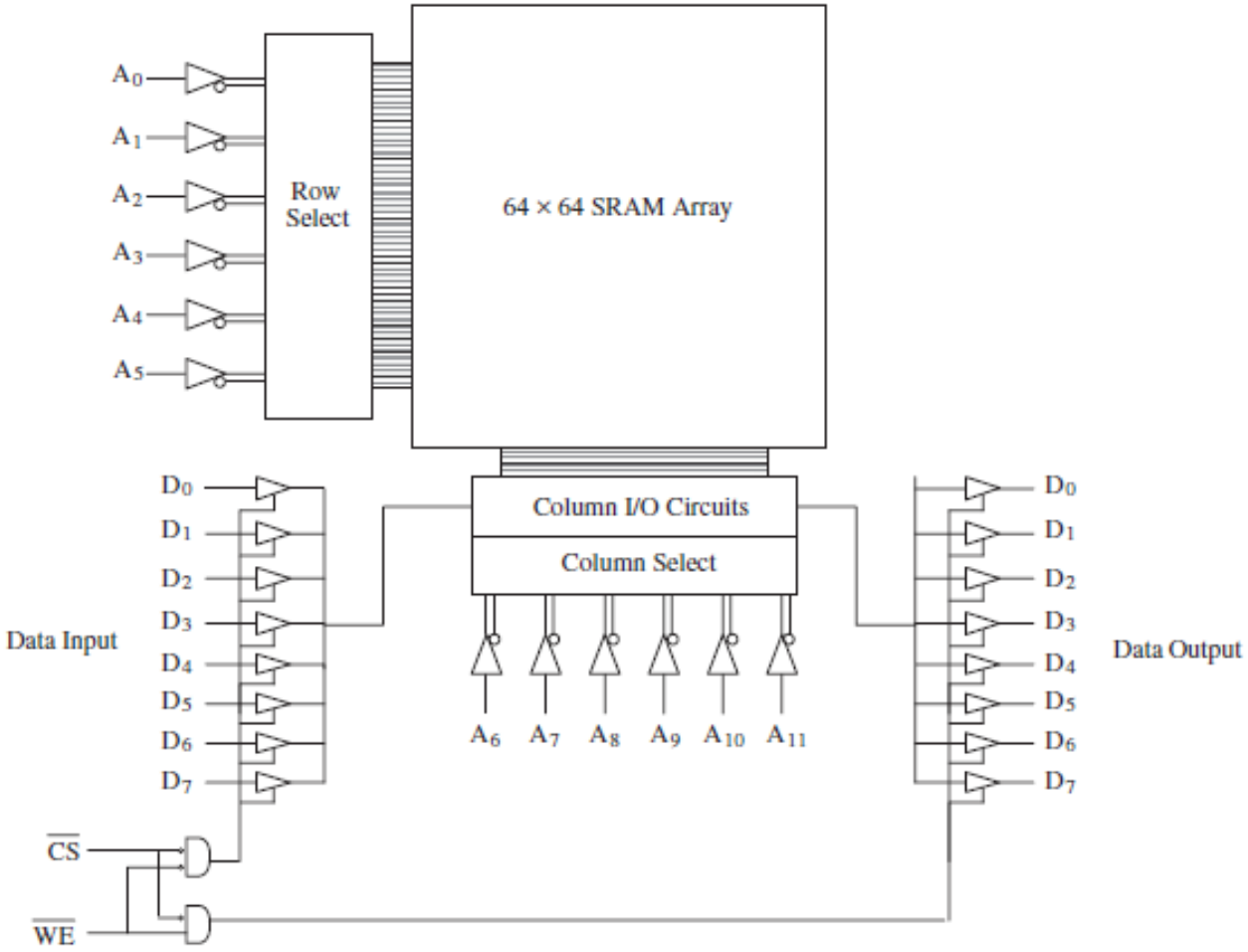
Random-Access Memory (RAM)

- RAM (random access memory), commonly referred to as *main memory*, is memory in which any location within it can be accessed directly (randomly, rather than sequentially from some starting point), and whose content can be changed more than once (the number depending on the hardware).
- Unlike ROM, contents of RAM are erased if RAM loses power, meaning RAM is *volatile*.
- The two main types of RAM are ***static RAM*** (SRAM) and ***dynamic RAM*** (DRAM).
- SRAM memory cells are made up of transistor-based flip-flop circuitry that typically holds its data due to a moving current being switched bi-directionally on a pair of inverting gates in the circuit, until power is cut off or the data is overwritten.



Random-Access Memory (RAM)

- Let us examine a sample logic circuit of 4K x8 SRAM



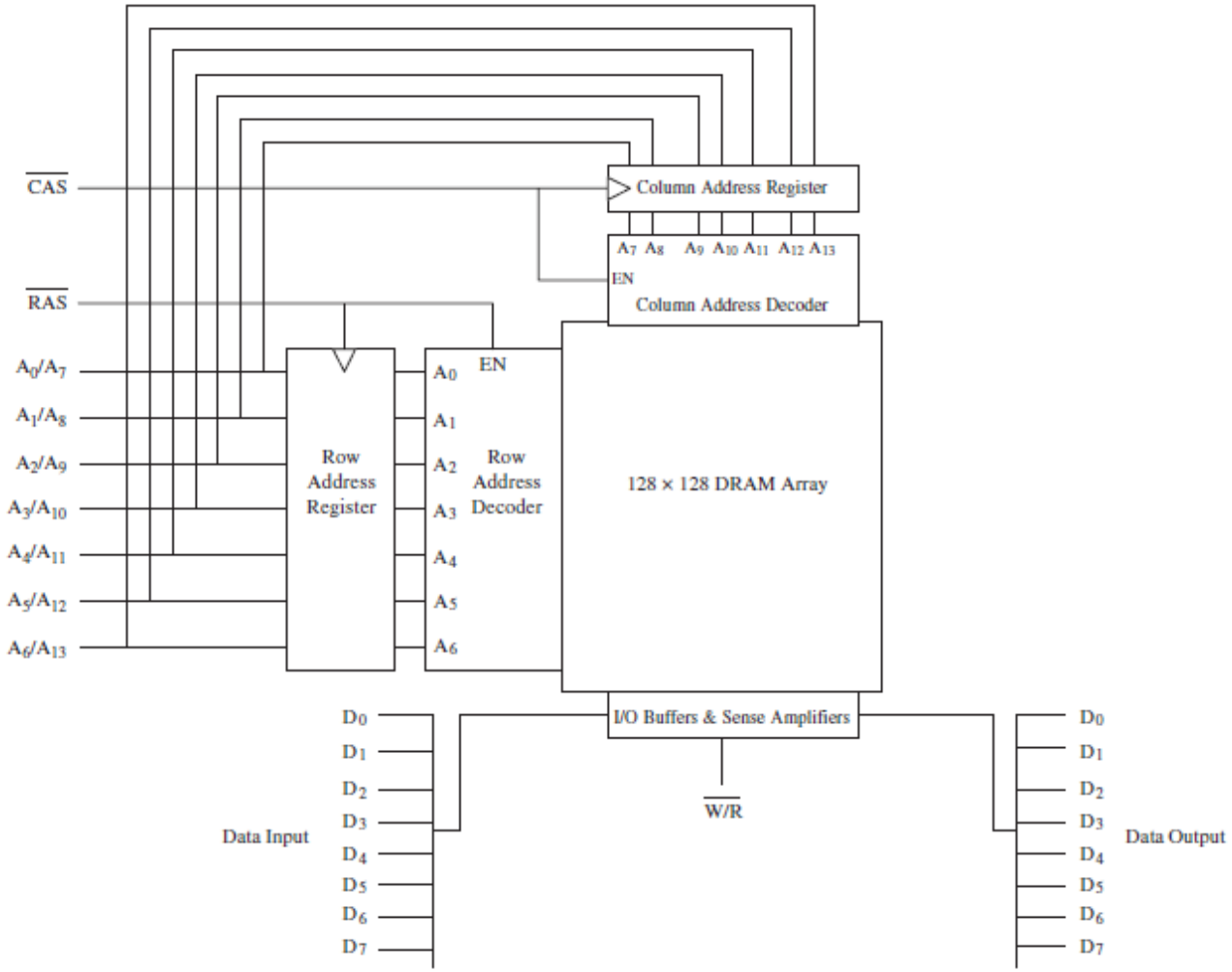
Random-Access Memory (RAM)

- In this example, the 4K × 8 SRAM is a 4K × 8 matrix, meaning it can store 4096 (4 × 1024) different 8-bit bytes, or 32768 bits of information.
- As shown in the figure, 12 address lines (A0–A11) are needed to address all 4096 (000000000000b–111111111111b) possible addresses, one address line for every address digit of the address. In this example, the 4K × 8 SRAM is set up as a 64 × 64 array of rows and columns where addresses A0–A5 identifying the row, and A6–A11 identifying the column.
- As with ROM, every intersection of a row and column in the SRAM matrix is a memory cell, and in the case of SRAM memory cells, they can contain flip-flop circuitry mainly based on semiconductor devices such as polysilicon load resistors, bipolar transistors, and/or CMOS transistors.
- There are eight output lines (D0–D7), a byte for every byte stored at an address.
- In this SRAM example, when the chip select (CS) is HIGH, then memory is in standby mode (no read or writes are occurring). When CS is toggled to LOW and write-enable input (WE) is LOW, then a byte of data is written through the data input lines (D0–D7) at the address indicated by the address lines.
- Given the same CS value (LOW) and WE is HIGH, then a byte of data is being read from the data output lines (D0–D7) at the address indicated by the address lines (A0–A7).

Random-Access Memory (RAM)

- Given a sample logic DRAM circuit of 16K x8, this RAM configuration is a two dimensional array of 128 rows and 128 columns, meaning it can store 16384 (16 × 1024) different 8-bit bytes, or 131072 bits of information.
- With this address configuration, larger DRAMs can either be designed with 14 address lines (A0–A13) needed to address all 16384 (00000000000b–11111111111b) possible addresses—one address line for every address digit of the address—or these address lines can be ***multiplexed***, or combined into fewer lines to share, with some type of data selection circuit managing the shared lines.
- The following figure demonstrates how a multiplexing of address lines could occur in this example.
- The 16K × 8 DRAM is set up with addresses A0–A6 identifying the row, and A7–A13 identifying the column. In this example, the ROW address strobe (RAS) line is toggled (from HIGH to LOW) for A0–A6 to be transmitted, and then the Column Address Strobe (CAS) line is toggled (from HIGH to LOW) for A7–A7 to be transmitted. After this point the memory cell is latched and ready to be written to or read from.
- There are eight output lines (D0–D7), a byte for every byte stored at an address. When the write enable (WE) input line is HIGH, data can be read from output lines D0–D7 and when WE is LOW, data can be written to input lines D0–D7.

Random-Access Memory (RAM)



Processor Input/Output (I/O)

- Input/output components of a processor are responsible for moving information to and from the processor's other components to any memory and I/O outside of the processor.
- Processor I/O can either be input components that only bring information into the master processor, output components that bring information out of the master processor, or components that do both.
- Virtually any electromechanical system, embedded and non embedded, conventional (keyboard, mouse, etc.) as well as unconventional (power plants, human limbs, etc.), can be connected to an embedded board and act as an I/O device.
- I/O is a high-level group that can be subdivided into smaller groups of either output devices, input devices, or devices that are both input and output devices.

Processor Input/Output (I/O)

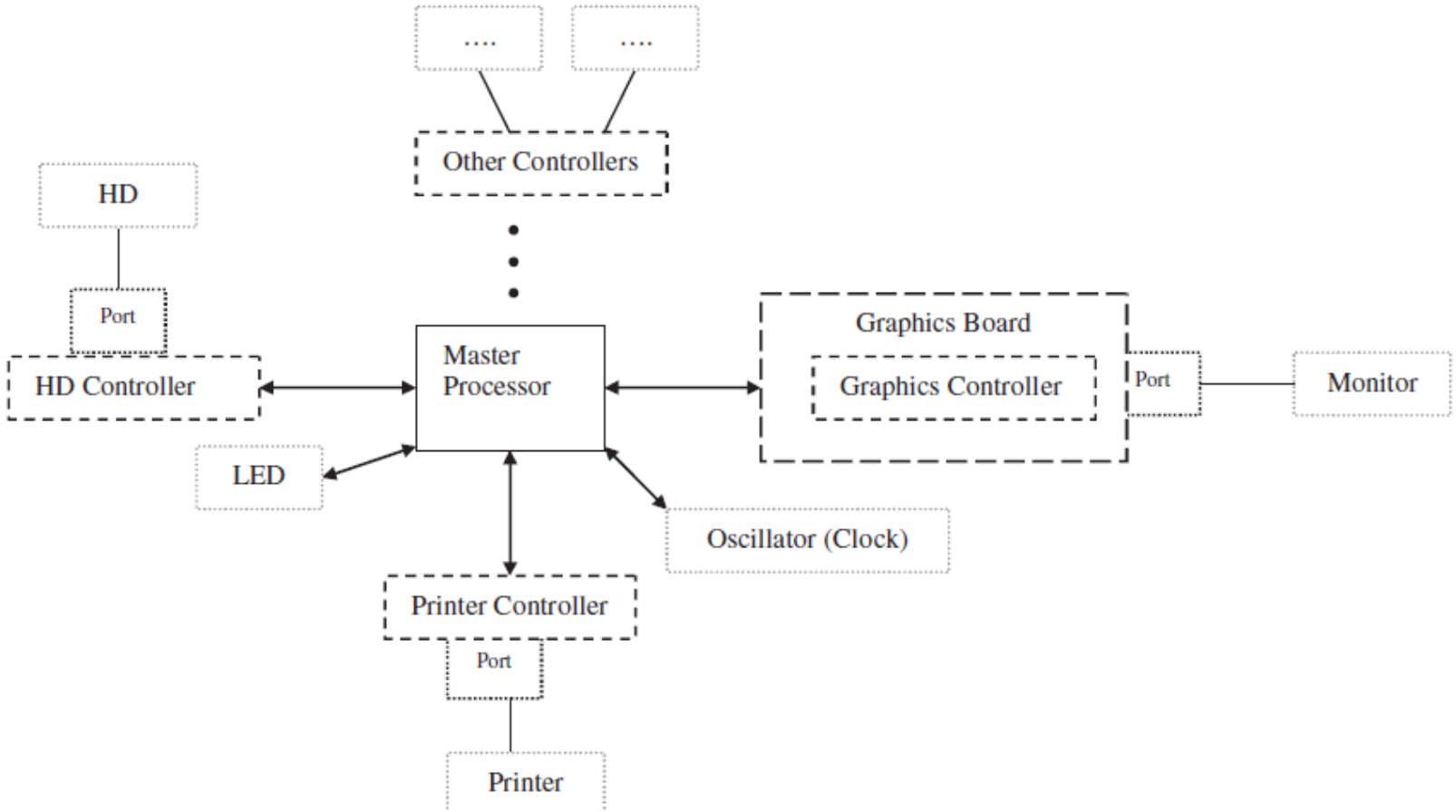
- Output devices can receive data from board I/O components and display that data in some manner, such as printing it to paper, to a disk, or to a screen, or a blinking LED light for a person to see.
- An input device transmits data to board I/O components, such as a mouse, keyboard, or remote control.
- I/O devices can do both, such as a networking device that can transmit data to and from the Internet, for instance.
- An I/O device can be connected to an embedded board via a wired or wireless data transmission medium, such as a keyboard or remote control, or can be located on the embedded board itself, such as an LED.

Processor Input/Output (I/O)

I/O hardware is typically made up of all or some combination of six main logical units:

- The *transmission medium*, wireless or wired medium connecting the I/O device to the embedded board for data communication and exchanges.
- *A communication port,* which is what the transmission medium connects to on the board, or if a wireless system, what receives the wireless signal.
- A *communication interface,* which manages data communication between master CPU and I/O device or I/O controller; also responsible for encoding data and decoding data to and from the logical level of an IC and the logical level of the I/O port. This interface can be integrated into the master processor, or can be a separate IC.
- An *I/O controller*, a slave processor that manages the I/O device.
- *I/O buses*, the connection between the board I/O and master processor
- The *master processor integrated I/O.*

Processor Input/Output (I/O)



Processor Buses

- A key feature of processor buses is their *width* which is (the number of bits that can be transmitted at any one time).
- This can vary depending on both the buses implemented within the processor—for example: x86 contains bus widths of 16/32/64, 68K has 8/16/32/ 64 bit buses, MIPS 32 has 32 bit buses, and so forth—as well as the ISA register size definitions.
- Each bus also has a bus *speed* (in MHz) that impacts the performance of the processor.
- Buses implemented in real-world processor designs include the U, peripheral, and CPM buses in the MPC8xx family of processors, and the C and X buses in the x86 Geode.