

# Real-Time Embedded Systems

## CpE-450 Spring 07

Class 2

Bruce McNair

[bmcnair@stevens.edu](mailto:bmcnair@stevens.edu)

# Syllabus - by week

## 1. Introduction

- Definition of embedded system
- Constraints on embedded vs. standalone systems
- Concept of real-time design
- Time scales for real-time systems
- Applications

## 2. Hardware/software functional partitioning

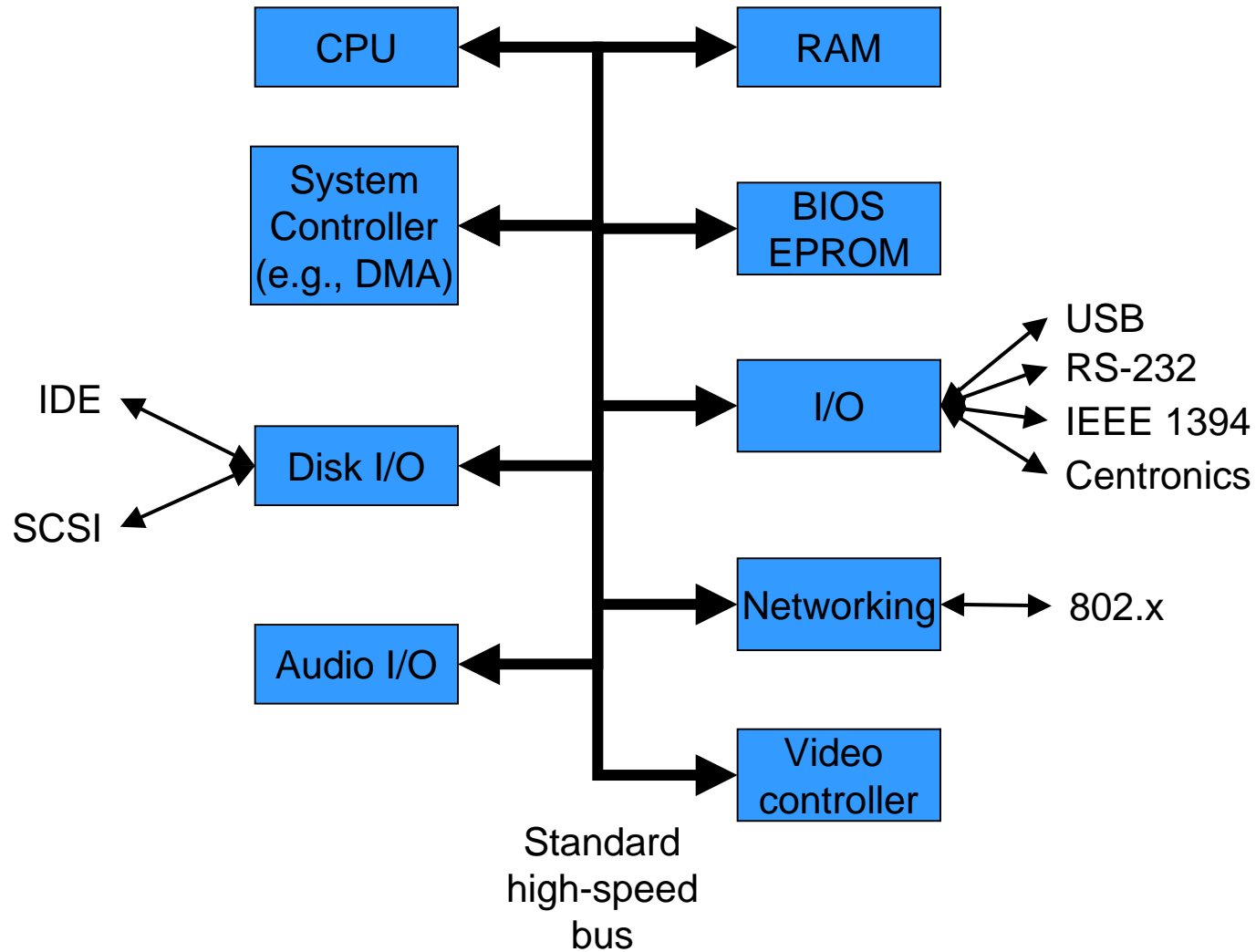
- Relevant hardware technologies:
  - Discrete logic
  - CPLDs, FPGAs, ASICs
- Software environments
  - HLL vs. assembly coding
  - DSP vs. general purpose vs. RISC

## 3. Development environments, course project definition

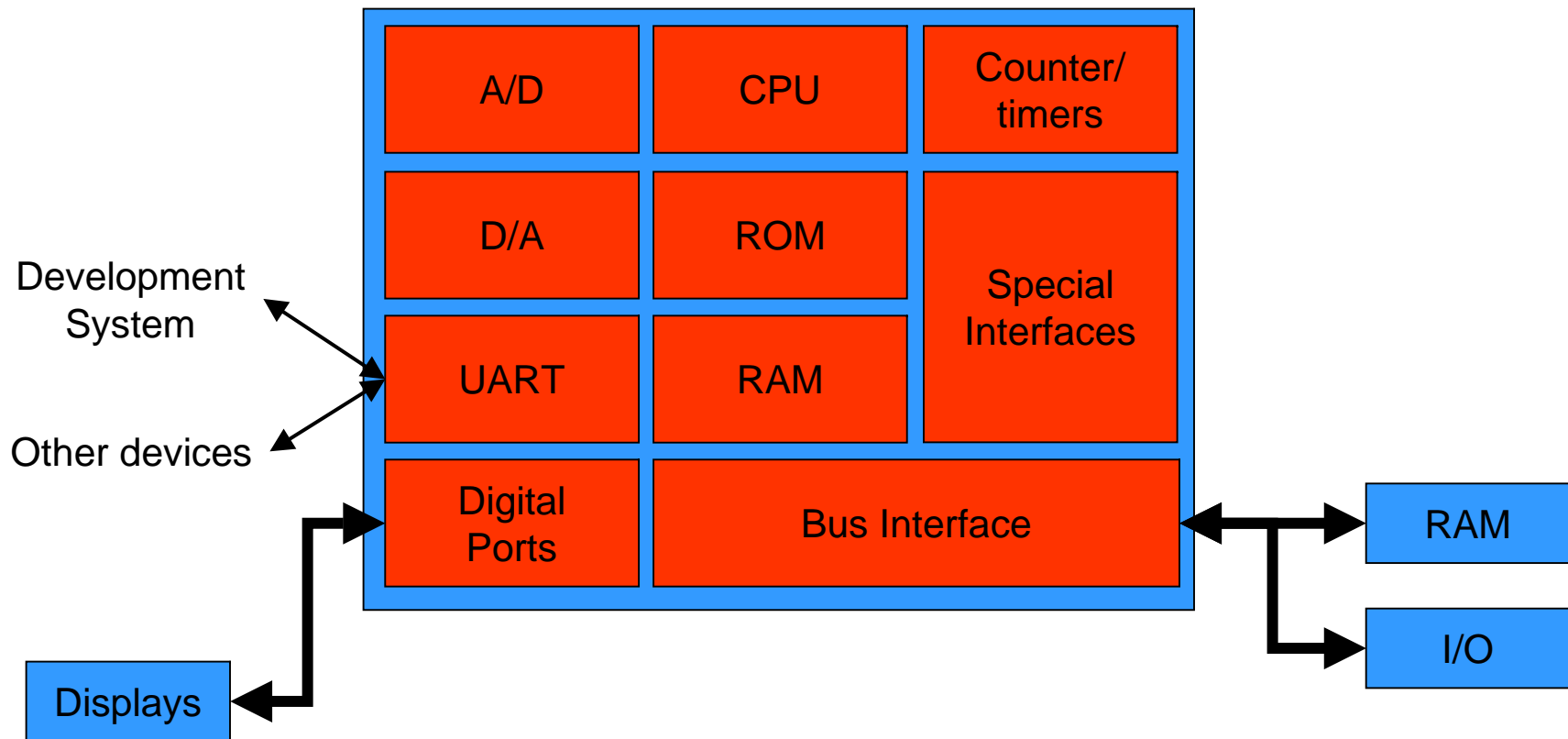
## 4. System architectures

## 5. Pipelining, interrupt service routines

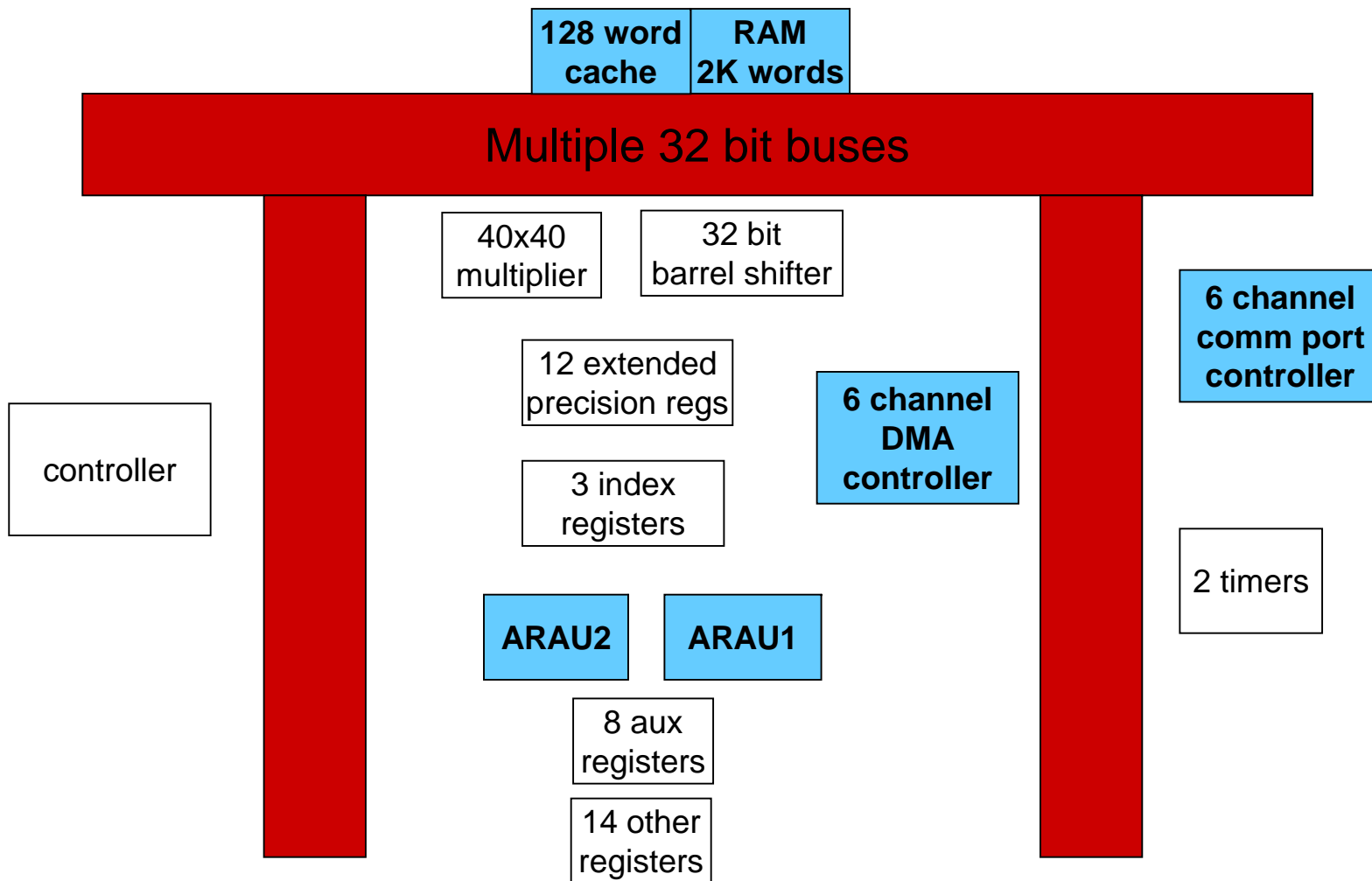
# Review of Computer Architecture



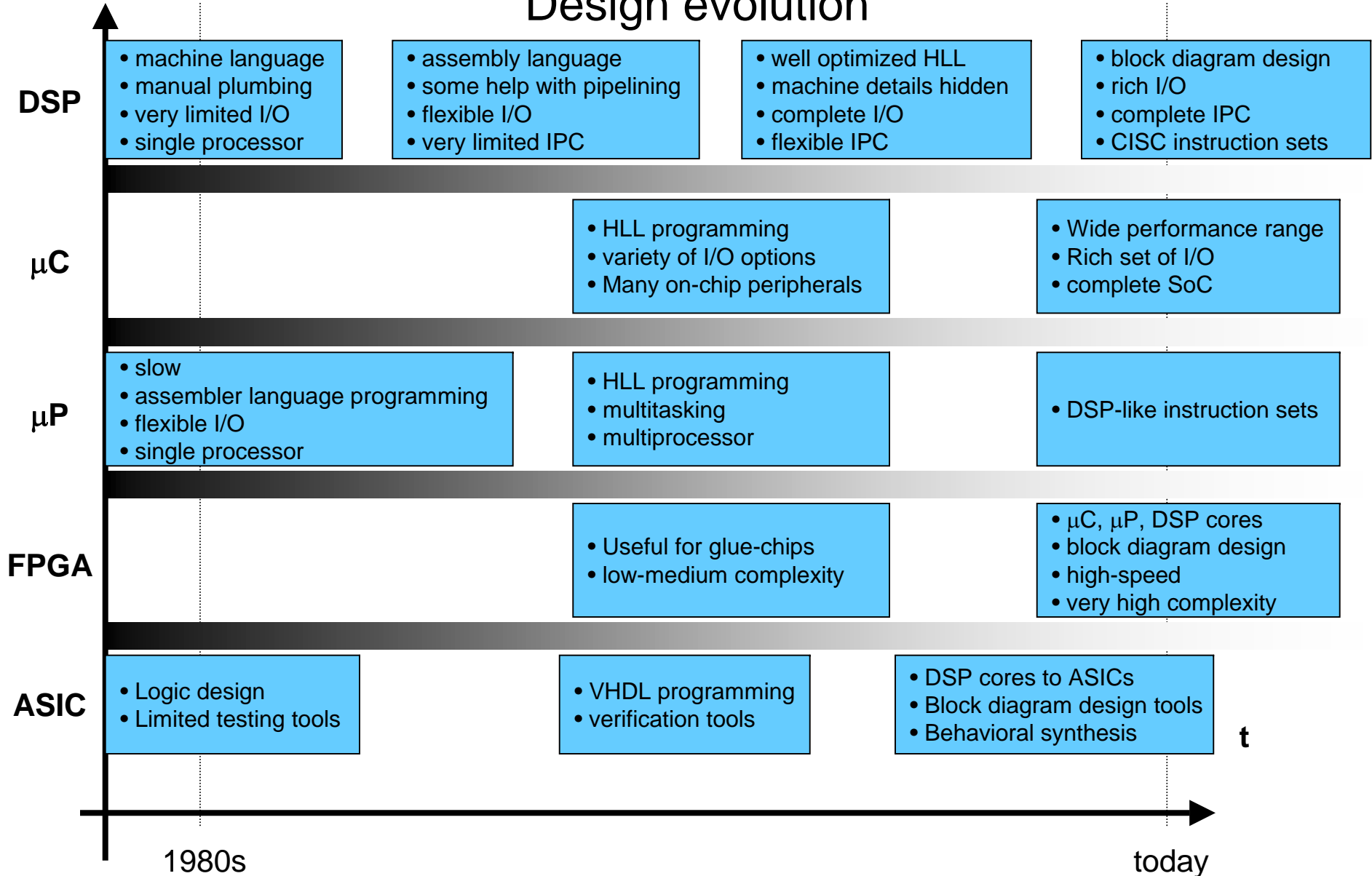
# Embedded System Architecture



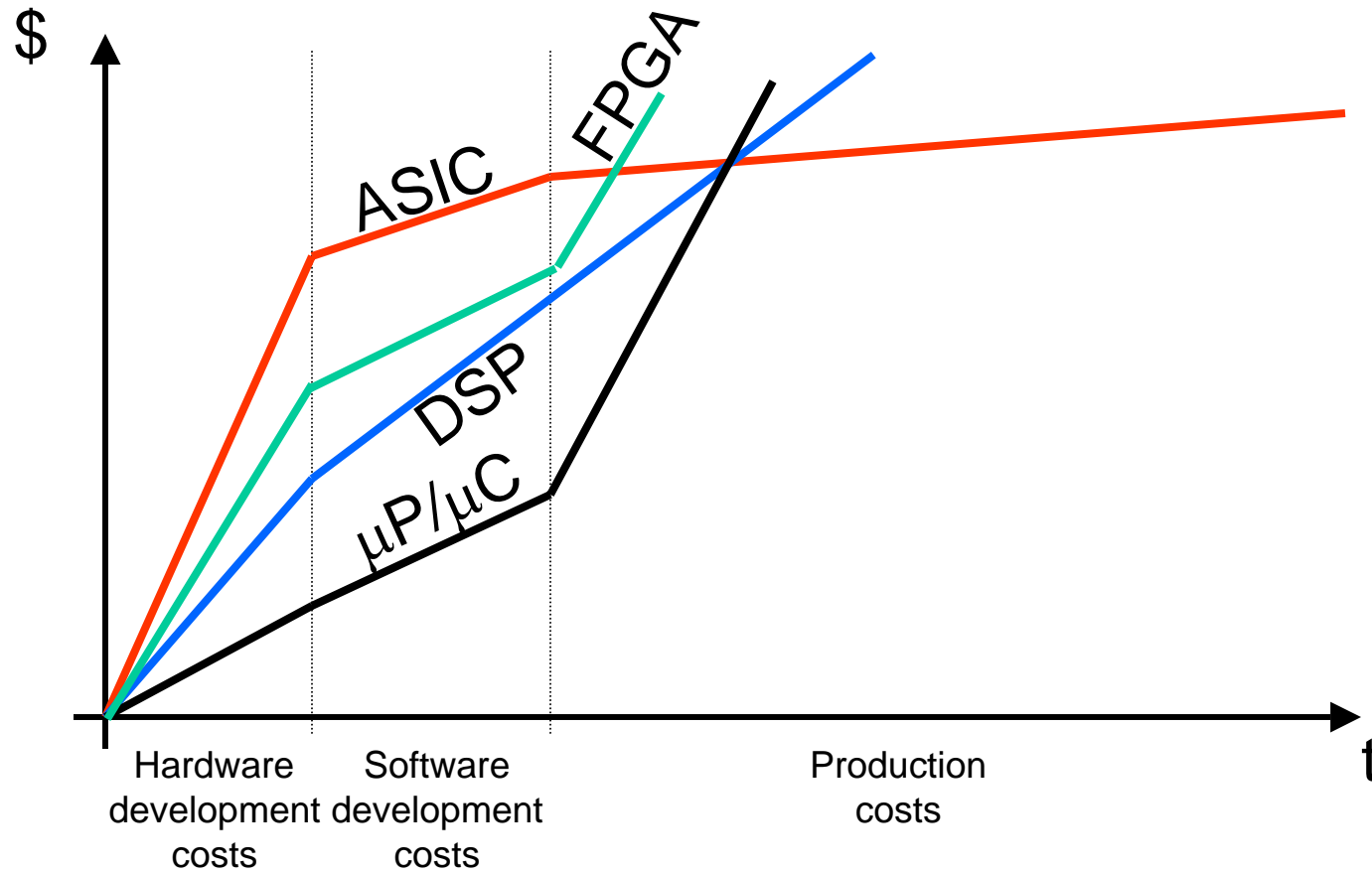
# DSPs – the Ultimate Embedded Processors: TMS320C40 Architecture



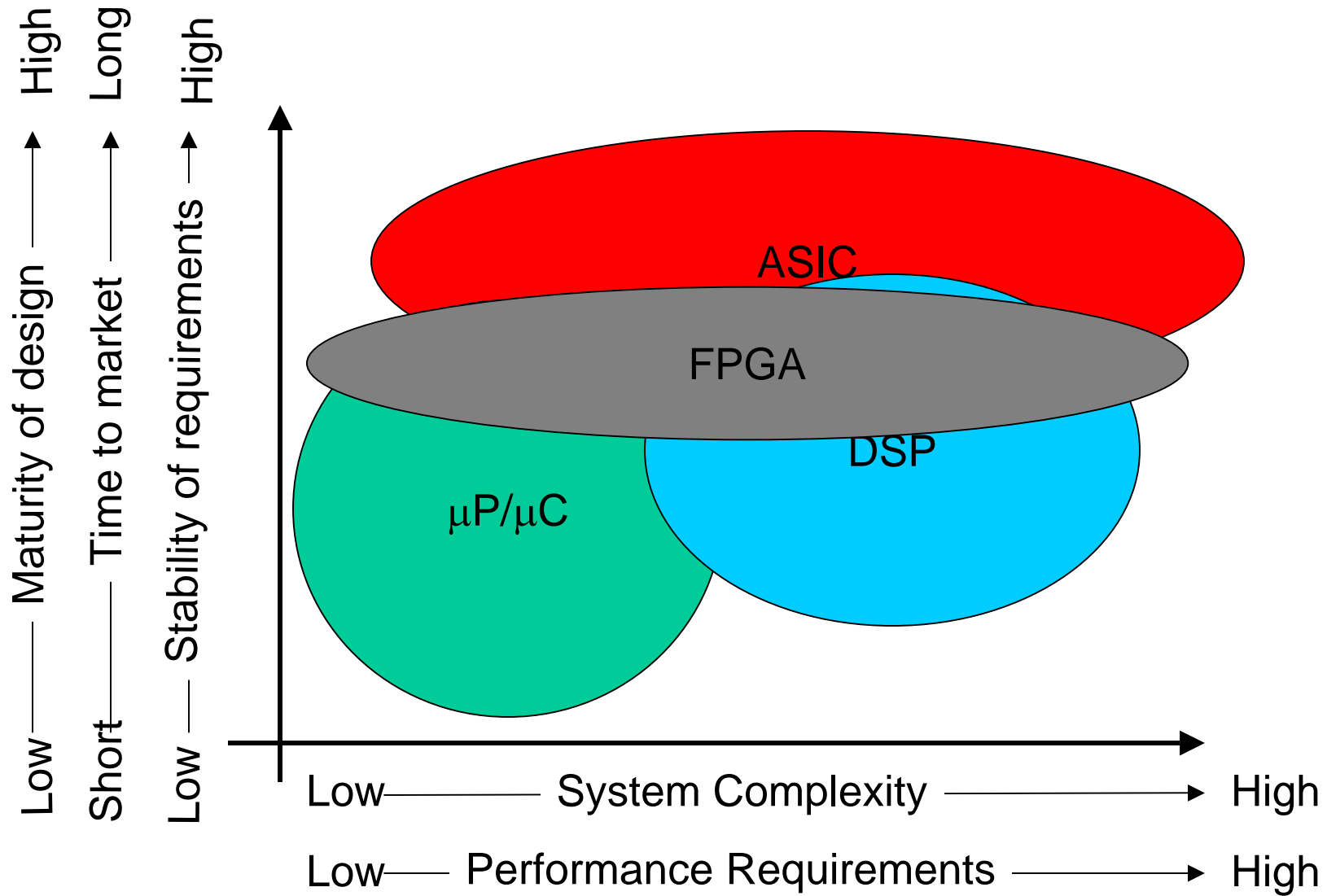
# Design evolution



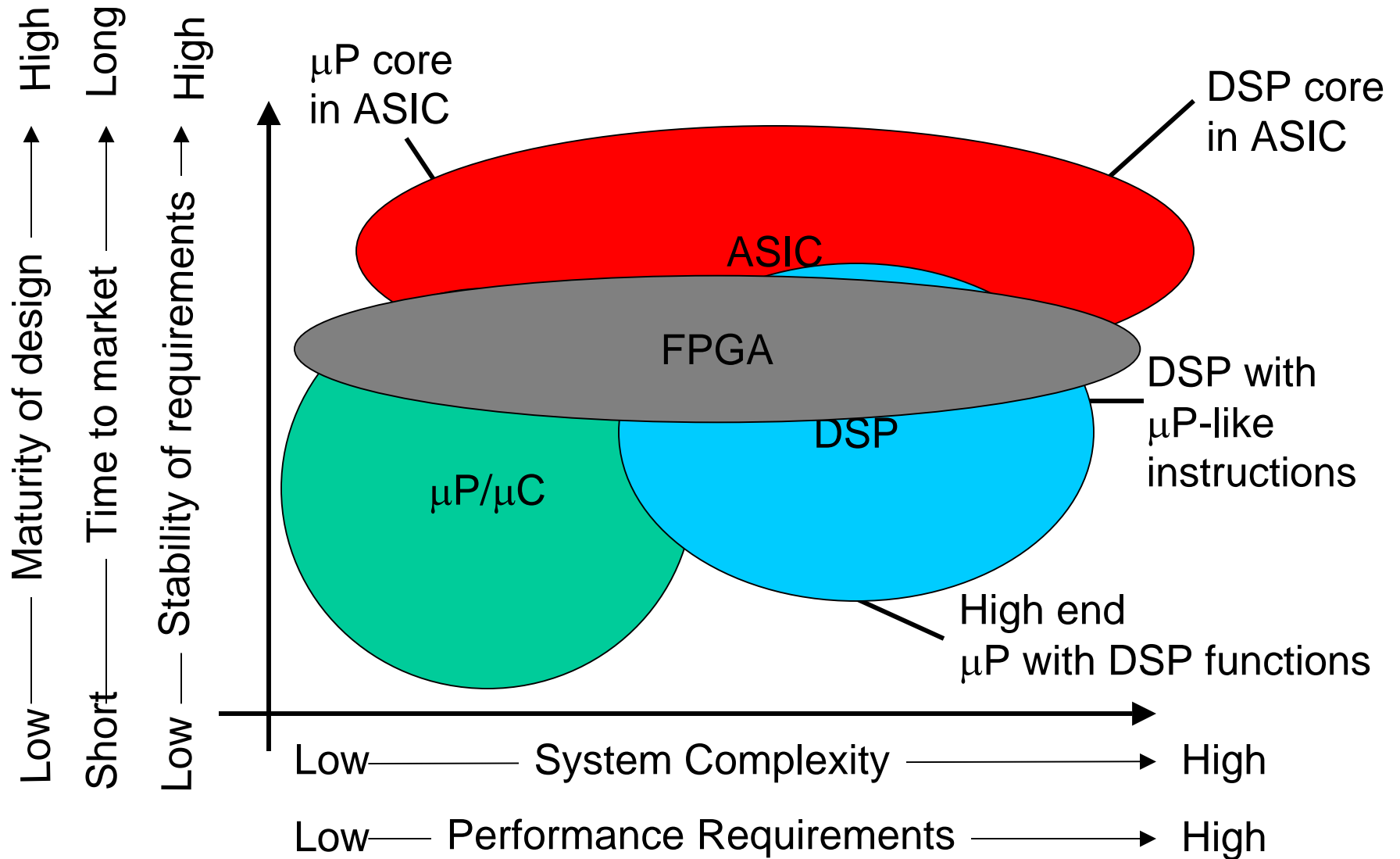
# DSP versus ASIC versus $\mu P/\mu C$



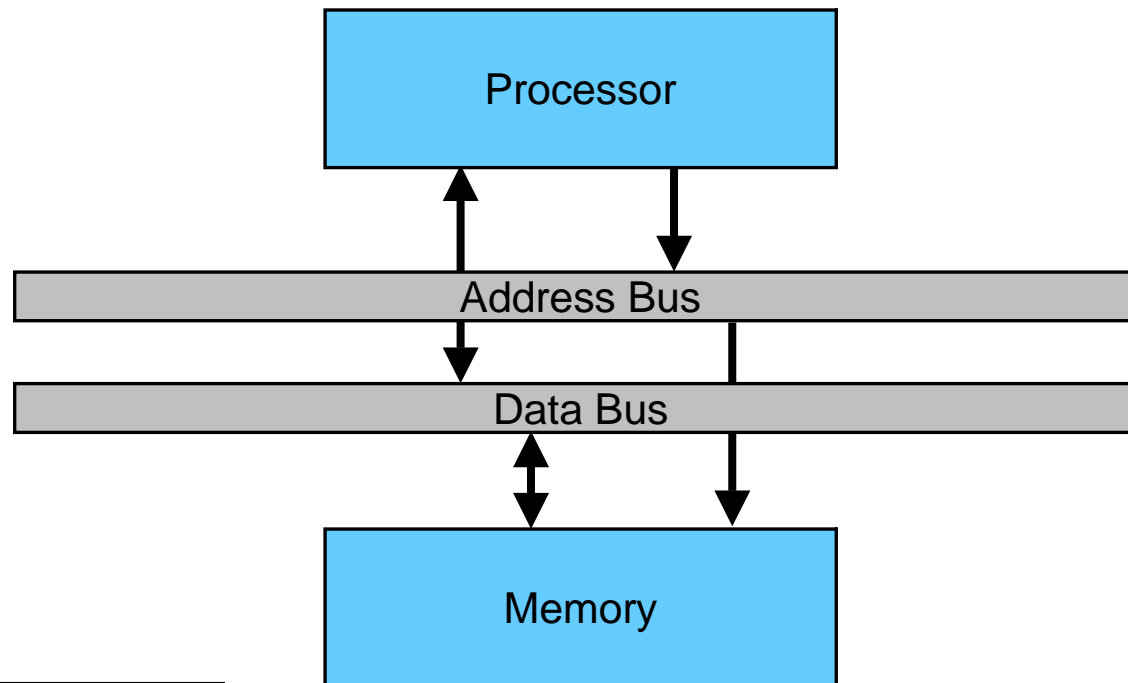
# DSP vs. ASIC vs. FPGA vs. $\mu$ P/ $\mu$ C



# DSP vs. ASIC vs. FPGA vs. $\mu$ P/ $\mu$ C



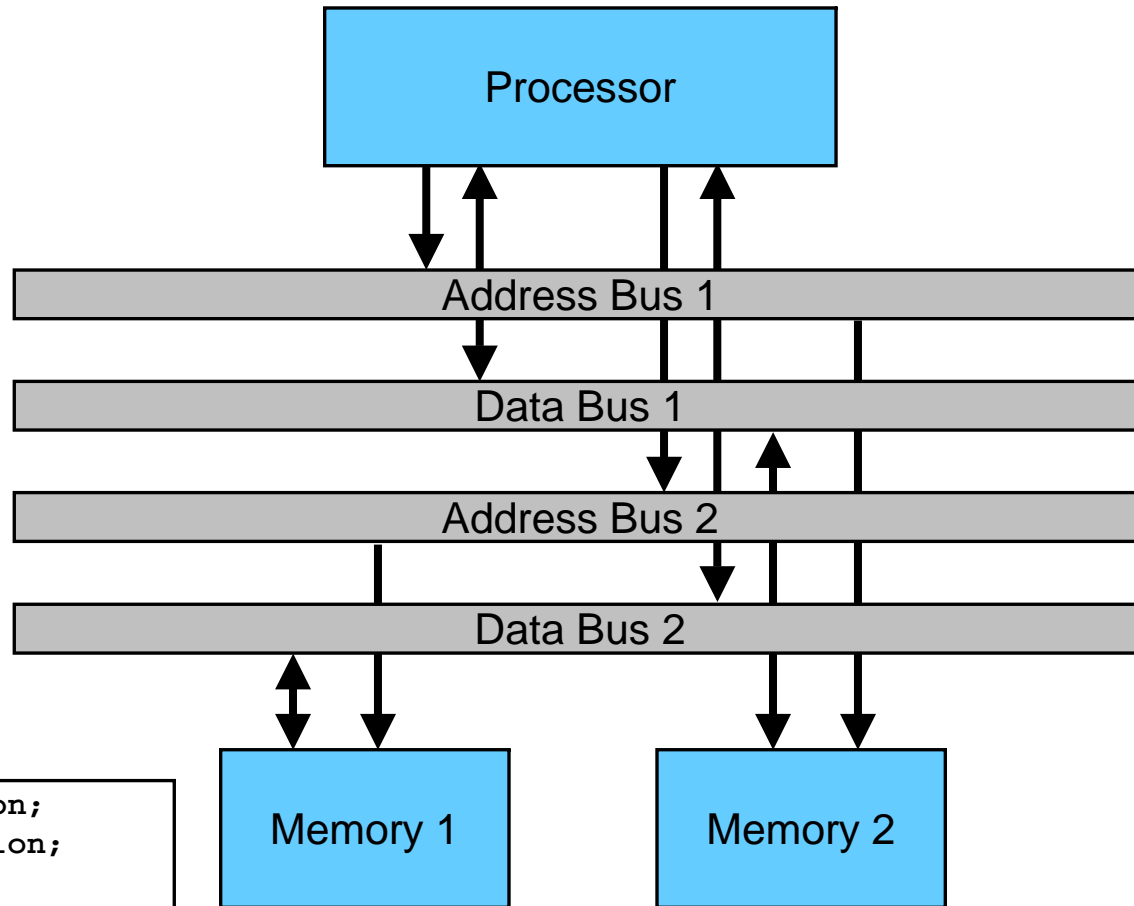
# Von Neumann Architecture



```
fetch instruction;  
decode instruction;  
fetch data1;  
fetch data2;  
multiply/accumulate;  
write data;
```

Single buses create bottleneck

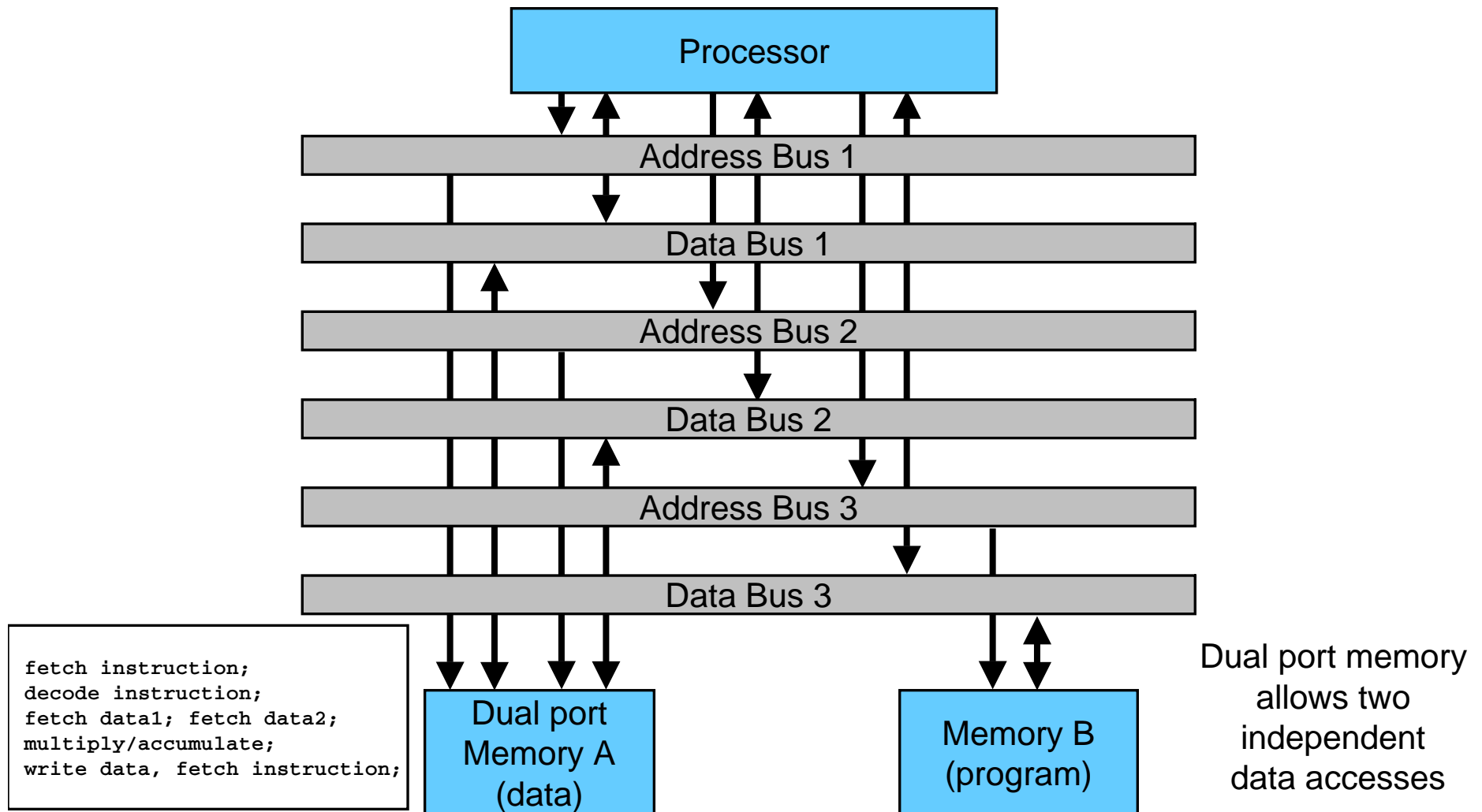
# Harvard Architecture



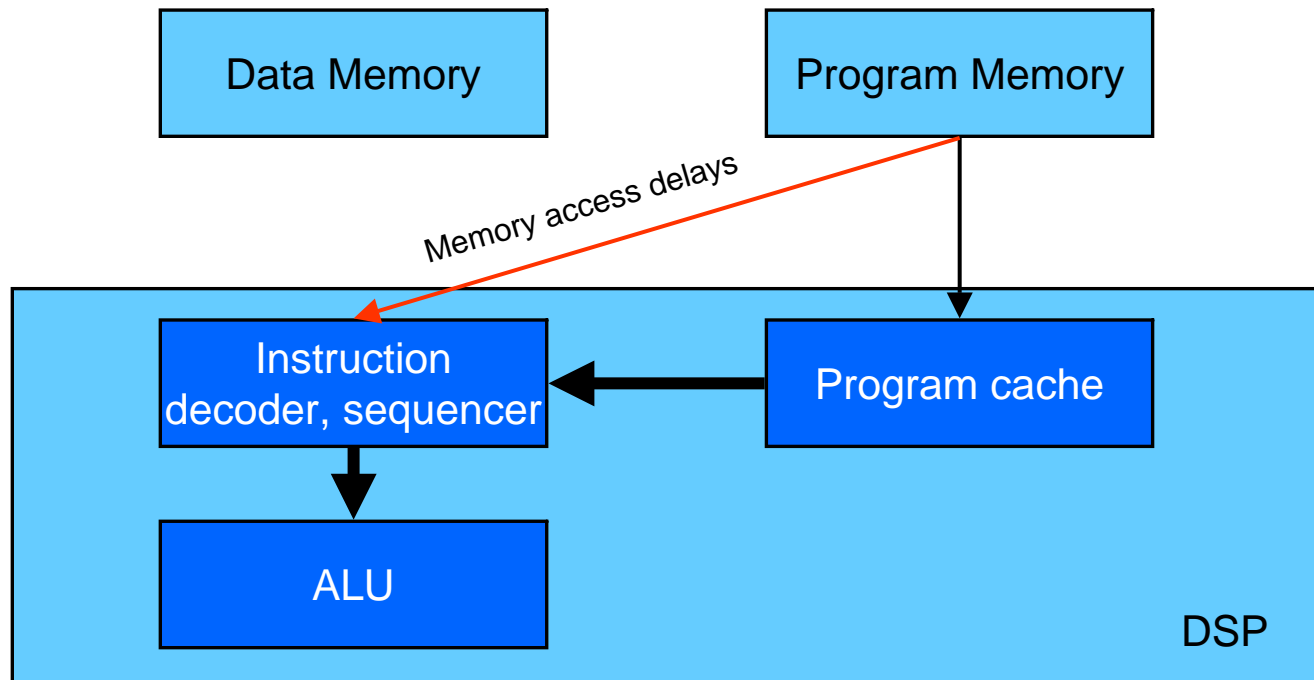
```
fetch instruction;  
decode instruction;  
fetch data1;  
fetch data2;  
multiply/accumulate;  
write data, fetch instr;
```

Multiple buses allow (limited) parallel operations

# Harvard Architecture with enhanced memory access

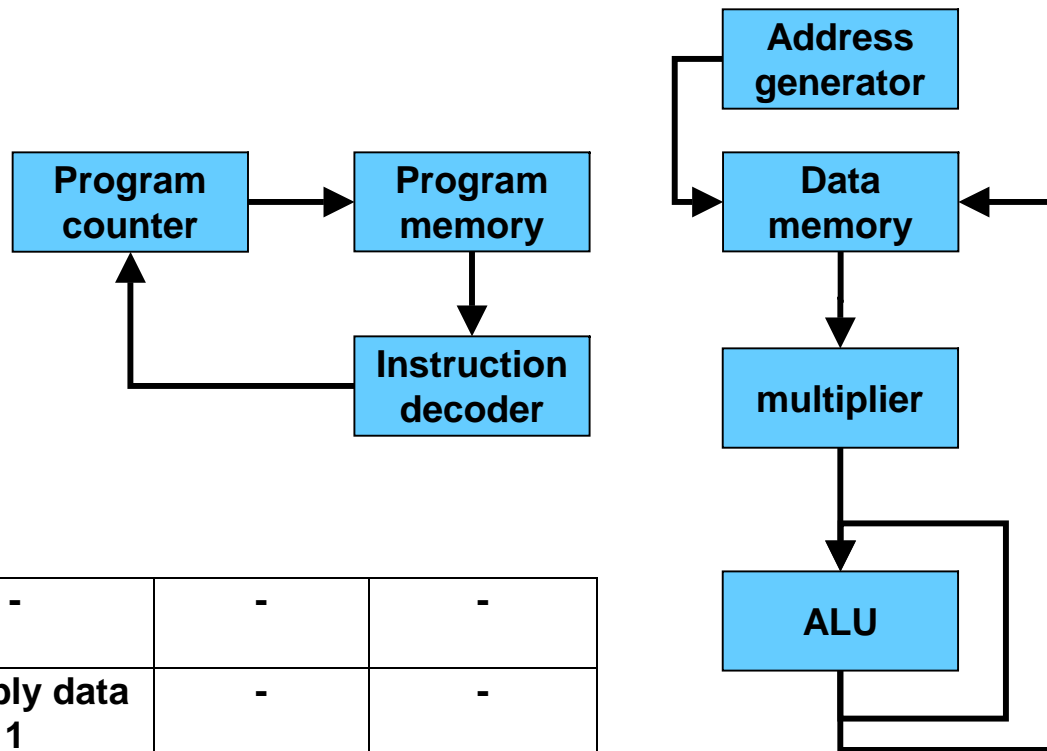


# Caching



Most programs have the property of “locality”  
- a small group of routines are exercised frequently  
These instructions are candidates for “caching”  
- storing on-chip for fast access

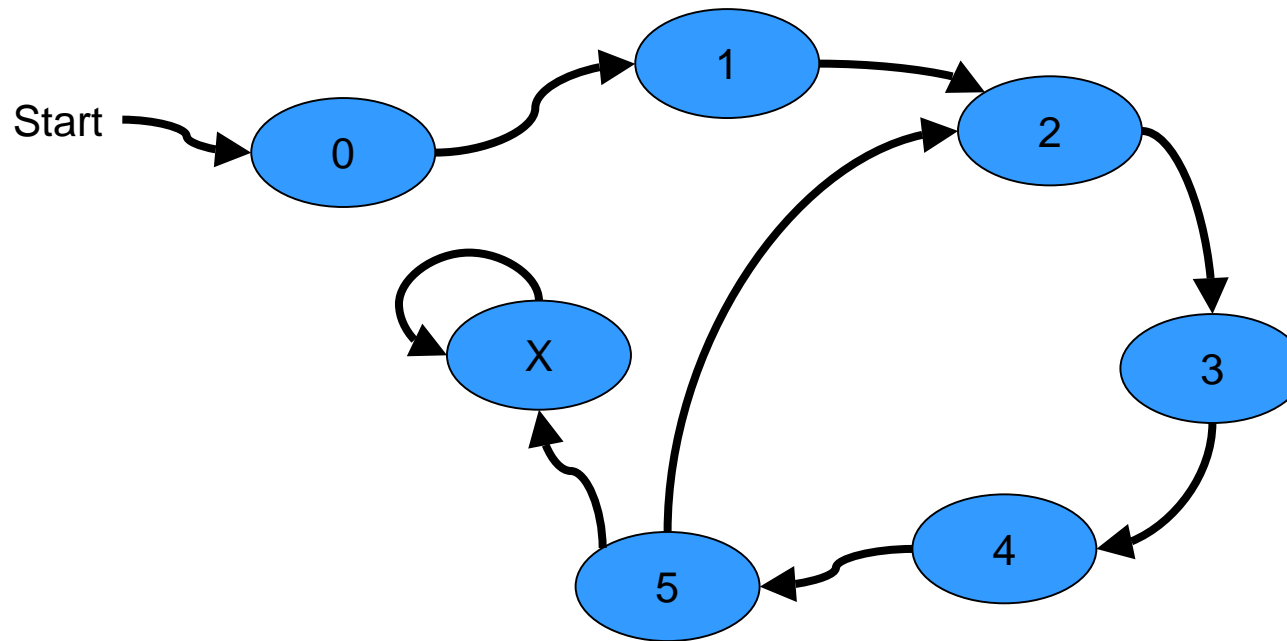
# Pipelining instructions



Fetch data 1	-	-	-
Fetch data 2	Multiply data 1	-	-
Fetch data 3	Multiply data 2	Add data 1	-
Fetch data 4	Multiply data 3	Add data 2	Store data 1

# Hardware considerations in Embedded Systems

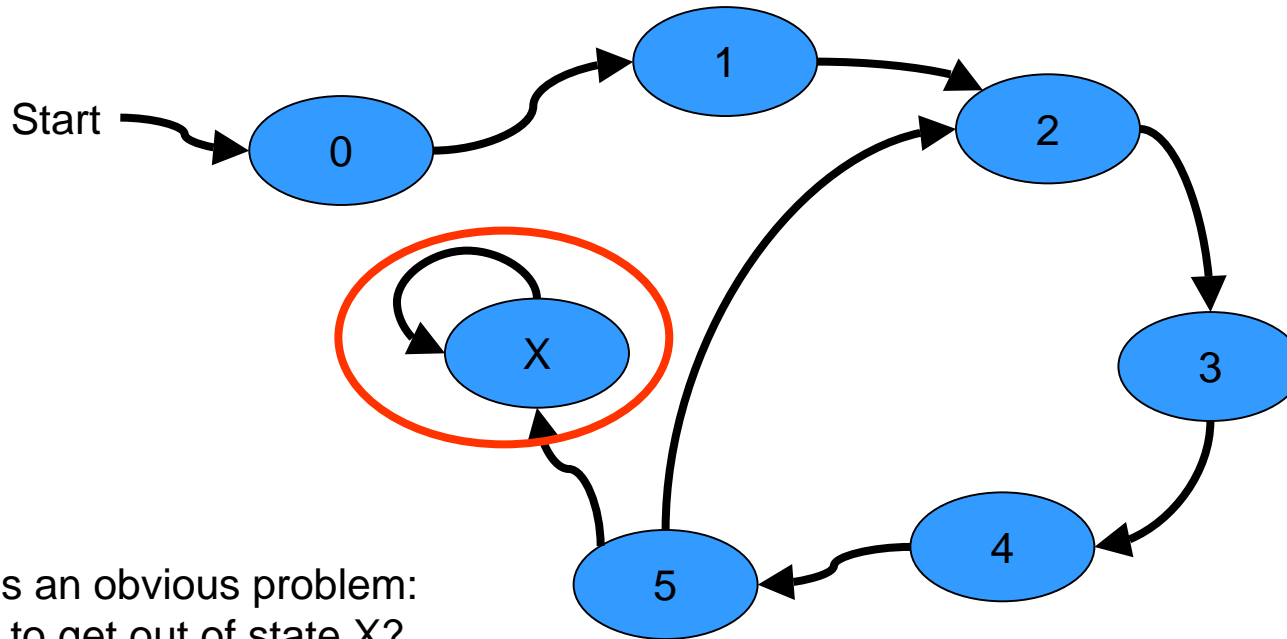
- Consider this state flow diagram:





# Hardware considerations in Embedded Systems

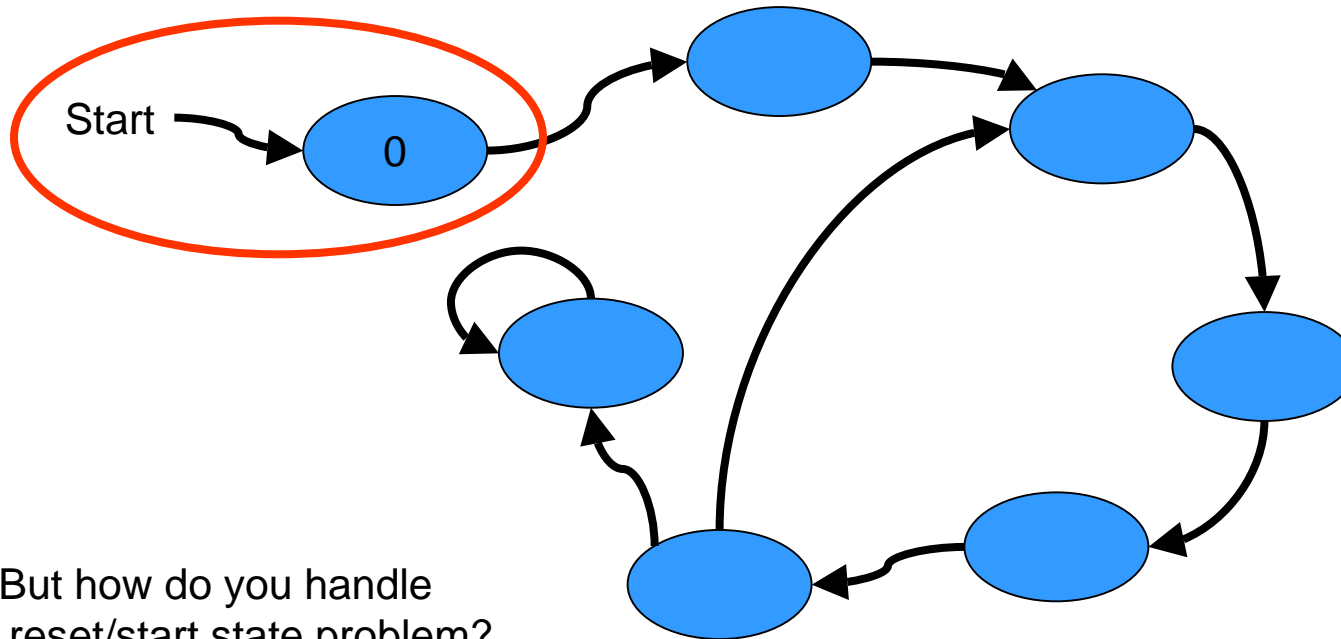
- Consider this state flow diagram:



This is an obvious problem:  
How to get out of state X?  
Detect the event with W.D.T. and  
reset

# Hardware considerations in Embedded Systems

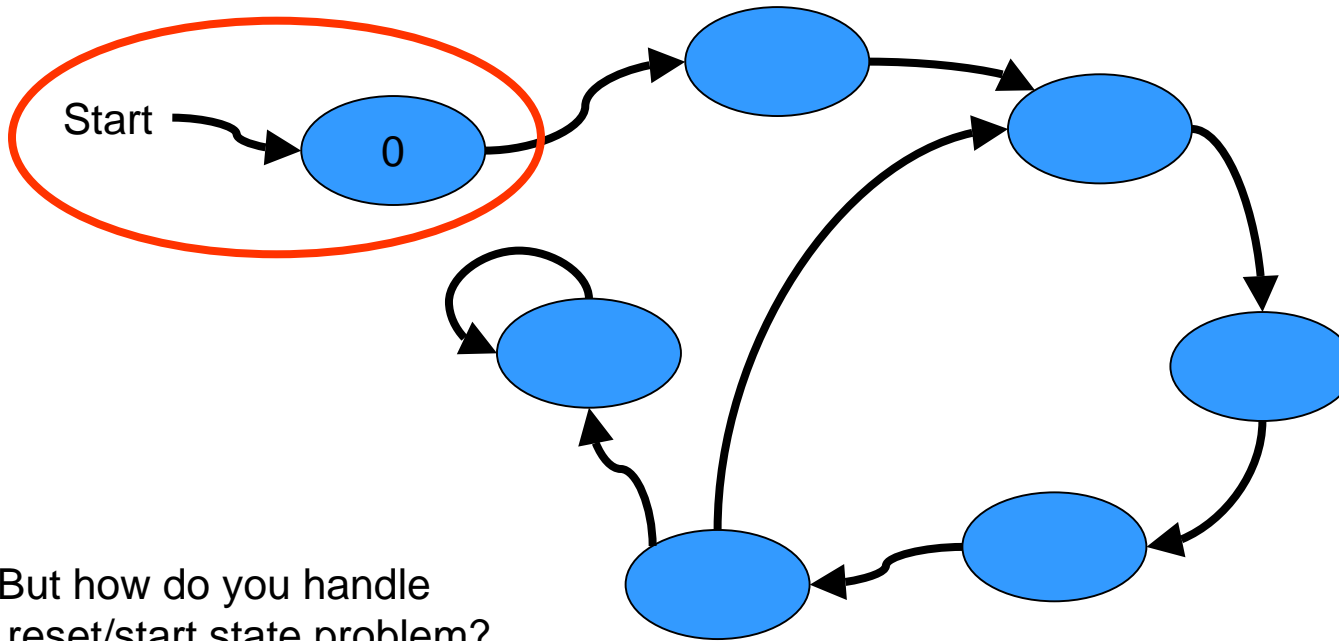
- Consider this state flow diagram:



But how do you handle  
the reset/start state problem?

# Hardware considerations in Embedded Systems

- Consider this state flow diagram:



But how do you handle  
the reset/start state problem?

Most embedded processors  
provide a power-on reset function

# Debugging Embedded Systems

- The classic “first program” for a standalone system:

```
main()  
{  
    printf( 'hello world\n' );  
}
```

# Debugging Embedded Systems

- The classic “first program” for a standalone system:

```
main()  
{  
    printf( 'hello world\n' );  
}
```

- What is the equivalent for an embedded system?

# Debugging Embedded Systems

- The classic “first program” for a standalone system:

```
main()  
{  
    printf( 'hello world\n' );  
}
```

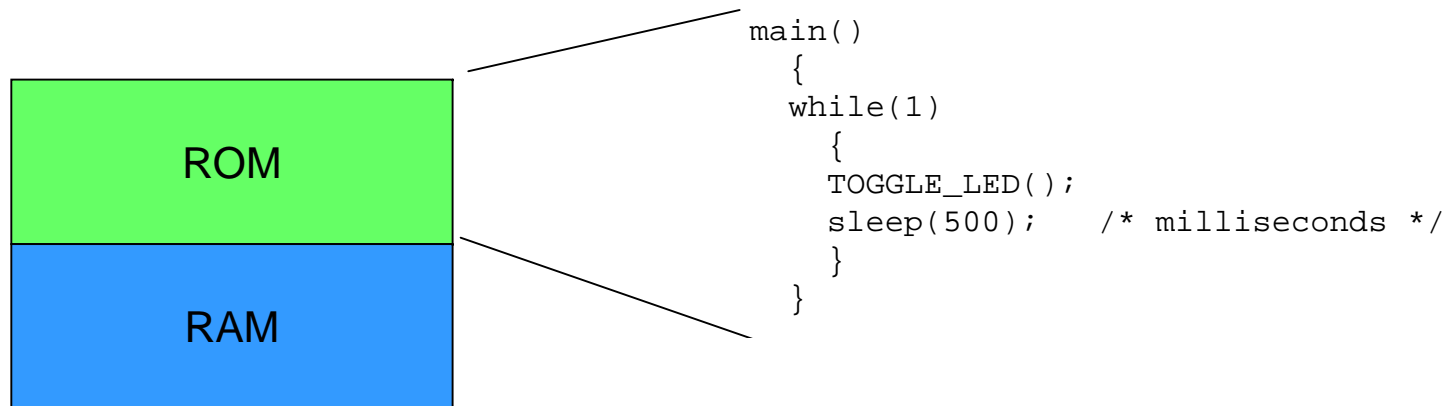
- What is the equivalent for an embedded system?

```
main()  
{  
    while(1)  
    {  
        TOGGLE_LED();  
        sleep(1);  
    }  
}
```

# A Few C Language References

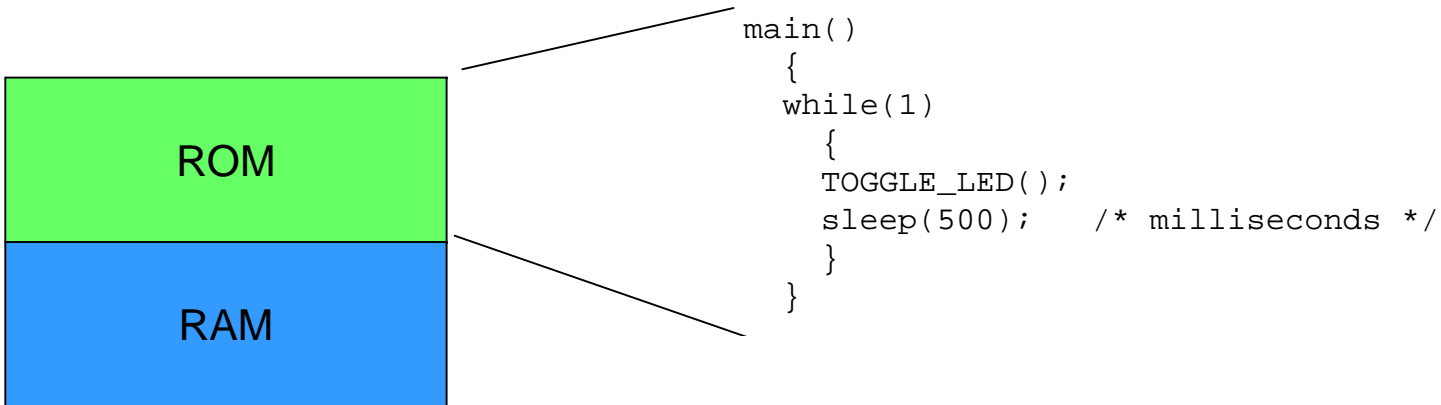
- Brian Kernighan and Dennis Richie, “The C Programming Language, 2<sup>nd</sup> Edition,” Prentice Hall, 1988, ISBN 0-13-110362-8
- Paul Chirlian, “Introduction to C,” Matrix Publishers, 1984, ISBN 0-916460-37-1

# Memory and other practical issues



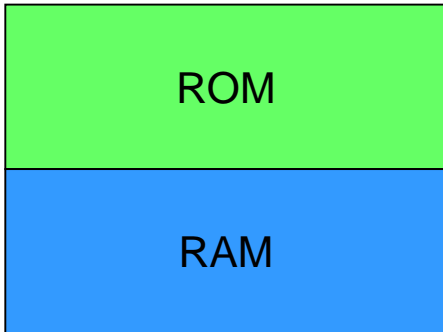
- Program is loaded into ROM

# Memory and other practical issues



- Program is loaded into ROM
  - How?
  - Where?
  - Where does it begin?

# Memory and other practical issues

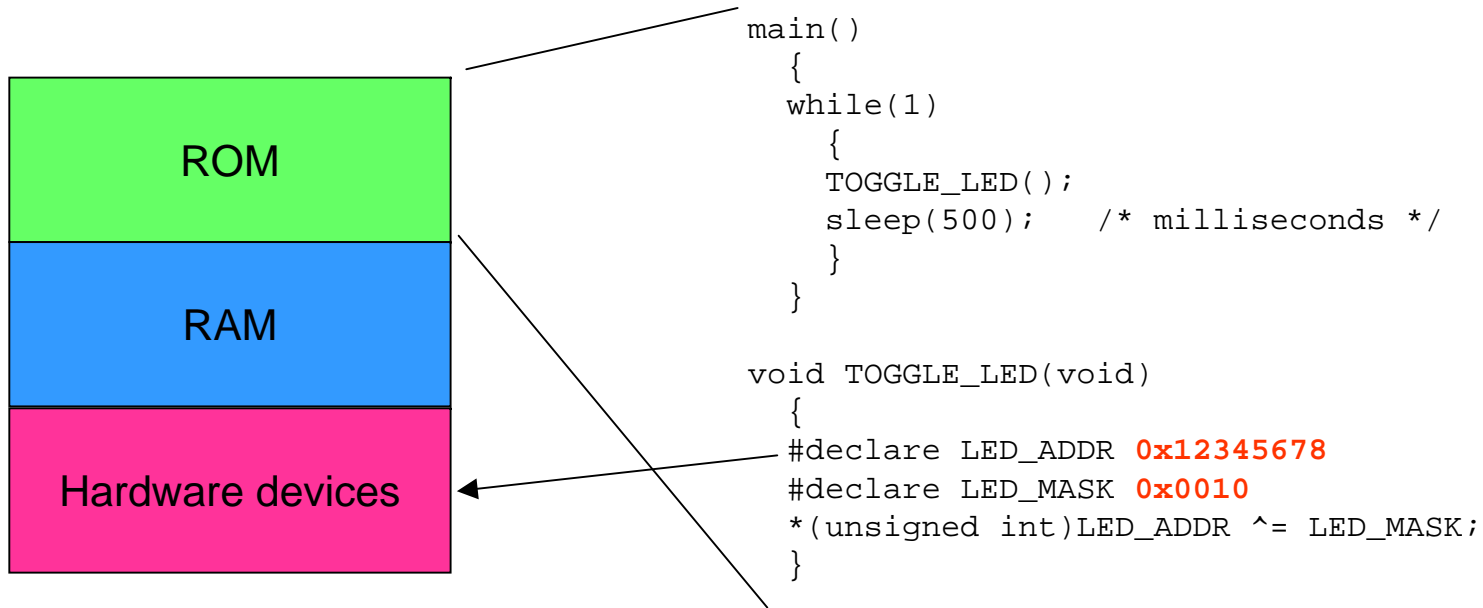


```
main()
{
  while(1)
  {
    TOGGLE_LED();
    sleep(500); /* milliseconds */
  }
}

void TOGGLE_LED(void)
{
  #declare LED_ADDR 0x12345678
  #declare LED_MASK 0x0010
  *(unsigned int)LED_ADDR ^= LED_MASK;
}
```

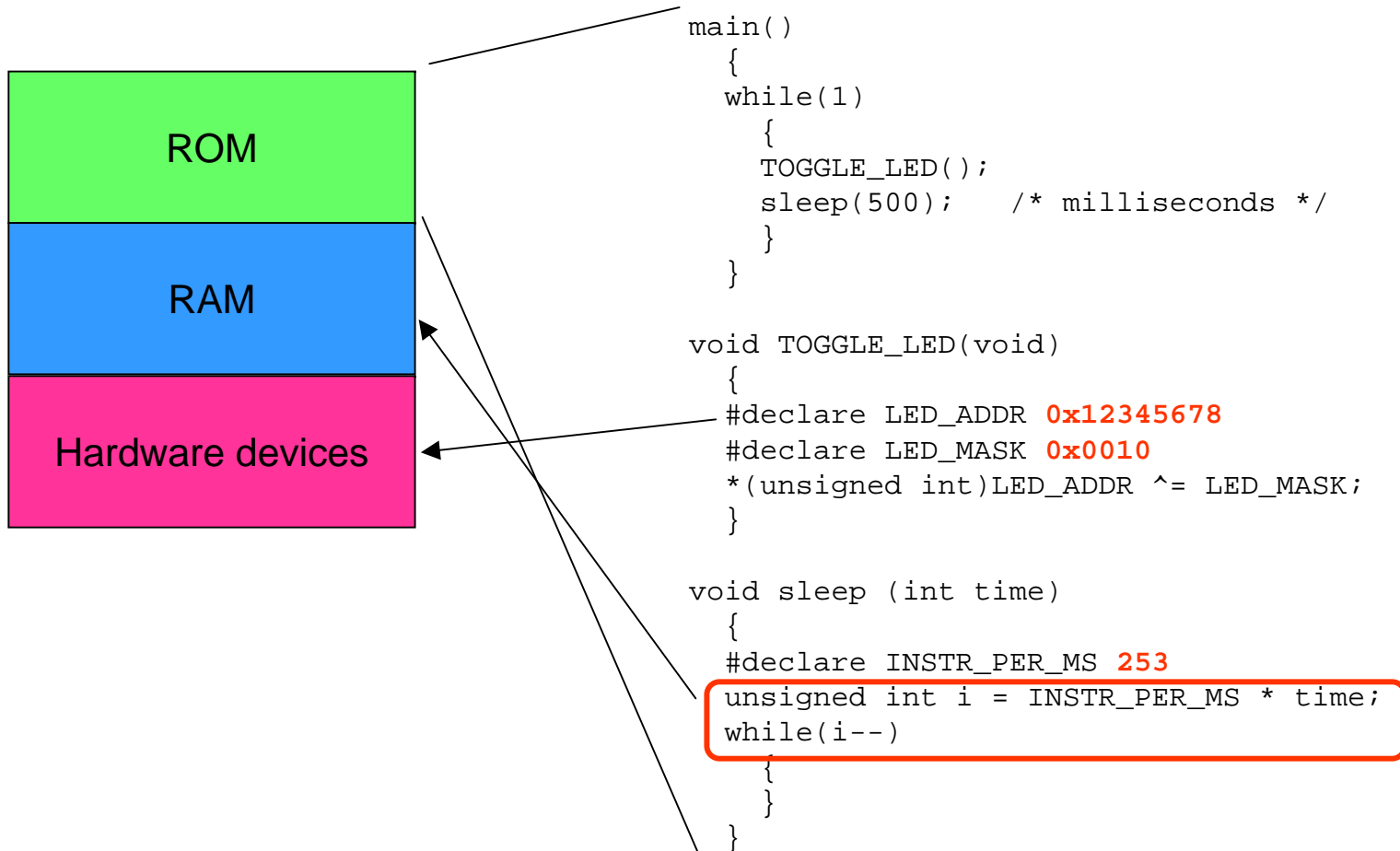
- We can access hardware in C, but where is it?

# Memory and other practical issues



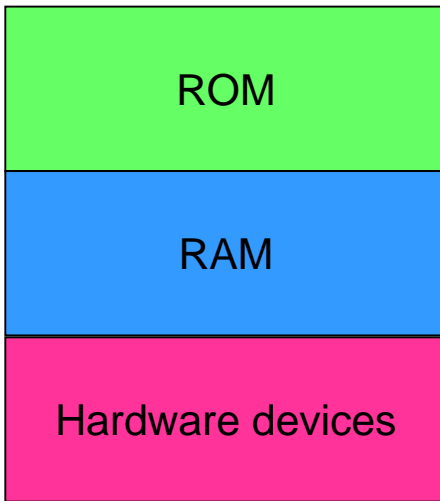
- We can access hardware in C, but where is it?
  - Probably elsewhere in address space of machine

# Memory and other practical issues



- What about variables?

# Memory and other practical issues



Entry →

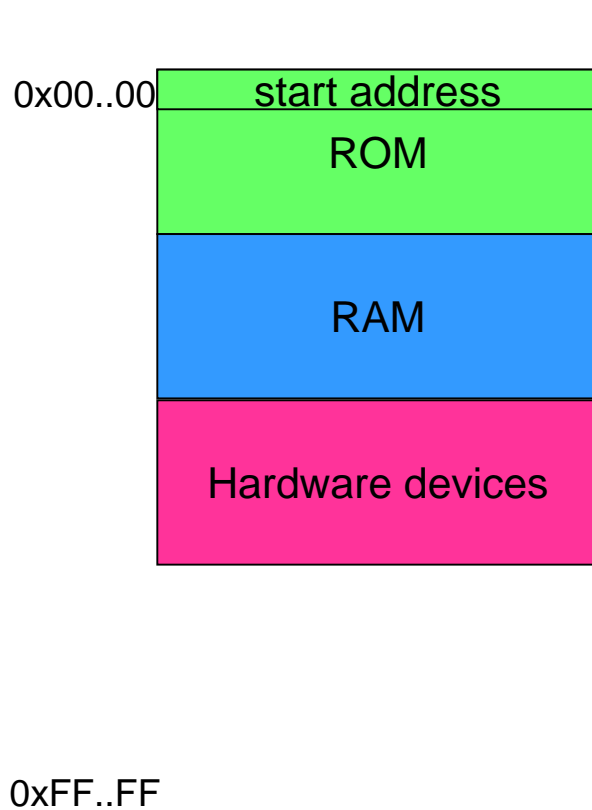
```
main()
{
  while(1)
  {
    TOGGLE_LED();
    sleep(500); /* milliseconds */
  }
}

void TOGGLE_LED(void)
{
  #declare LED_ADDR 0x12345678
  #declare LED_MASK 0x0010
  *(unsigned int)LED_ADDR ^= LED_MASK;
}

void sleep (int time)
{
  #declare INSTR_PER_MS 253
  unsigned int i = INSTR_PER_MS * time;
  while(i--)
  {
  }
}
```

- How do you define where the program starts?

# Memory and other practical issues



Entry →

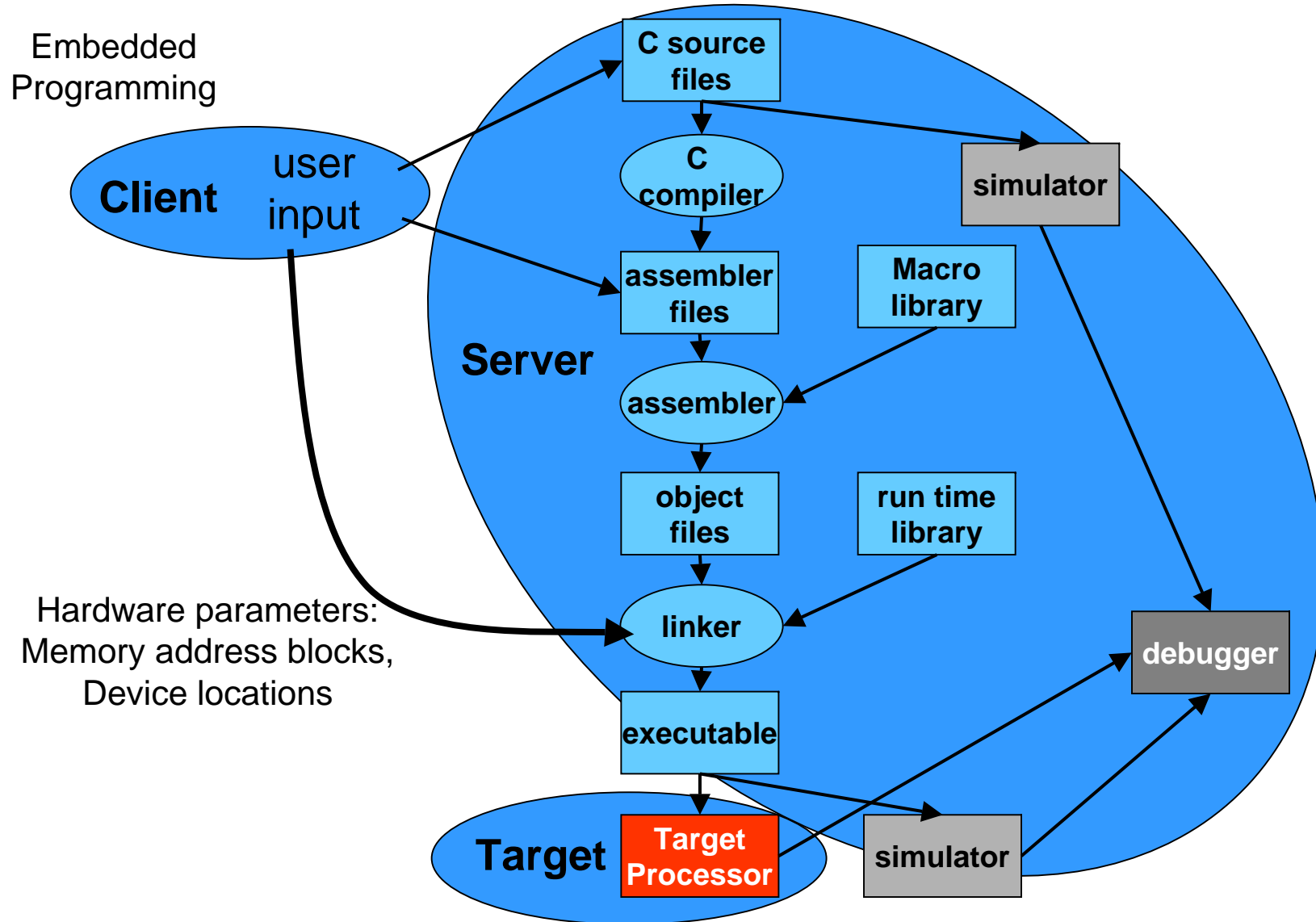
```
main()
{
  while(1)
  {
    TOGGLE_LED();
    sleep(500); /* milliseconds */
  }
}

void TOGGLE_LED(void)
{
  #declare LED_ADDR 0x12345678
  #declare LED_MASK 0x0010
  *(unsigned int)LED_ADDR ^= LED_MASK;
}

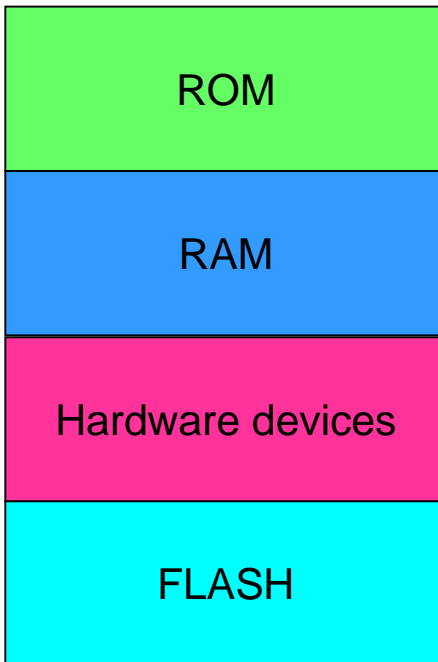
void sleep (int time)
{
  #declare INSTR_PER_MS 253
  unsigned int i = INSTR_PER_MS * time;
  while(i--)
  {
  }
}
```

- How do you define where the program starts?

# Building an executable module



# Memory and other practical issues



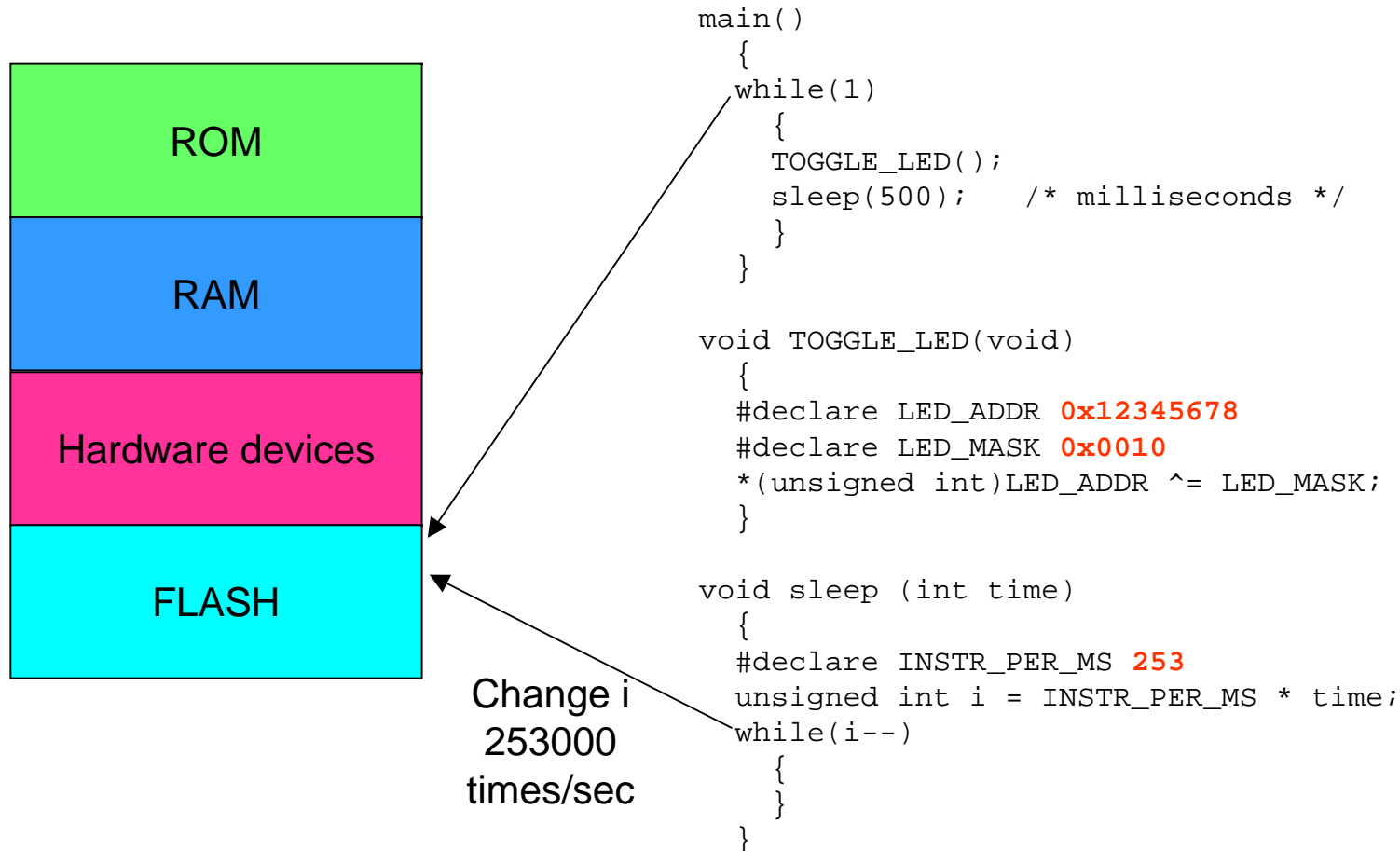
```
main()
{
  while(1)
  {
    TOGGLE_LED();
    sleep(500); /* milliseconds */
  }
}

void TOGGLE_LED(void)
{
  #declare LED_ADDR 0x12345678
  #declare LED_MASK 0x0010
  *(unsigned int)LED_ADDR ^= LED_MASK;
}

void sleep (int time)
{
  #declare INSTR_PER_MS 253
  unsigned int i = INSTR_PER_MS * time;
  while(i--)
  {
  }
}
```

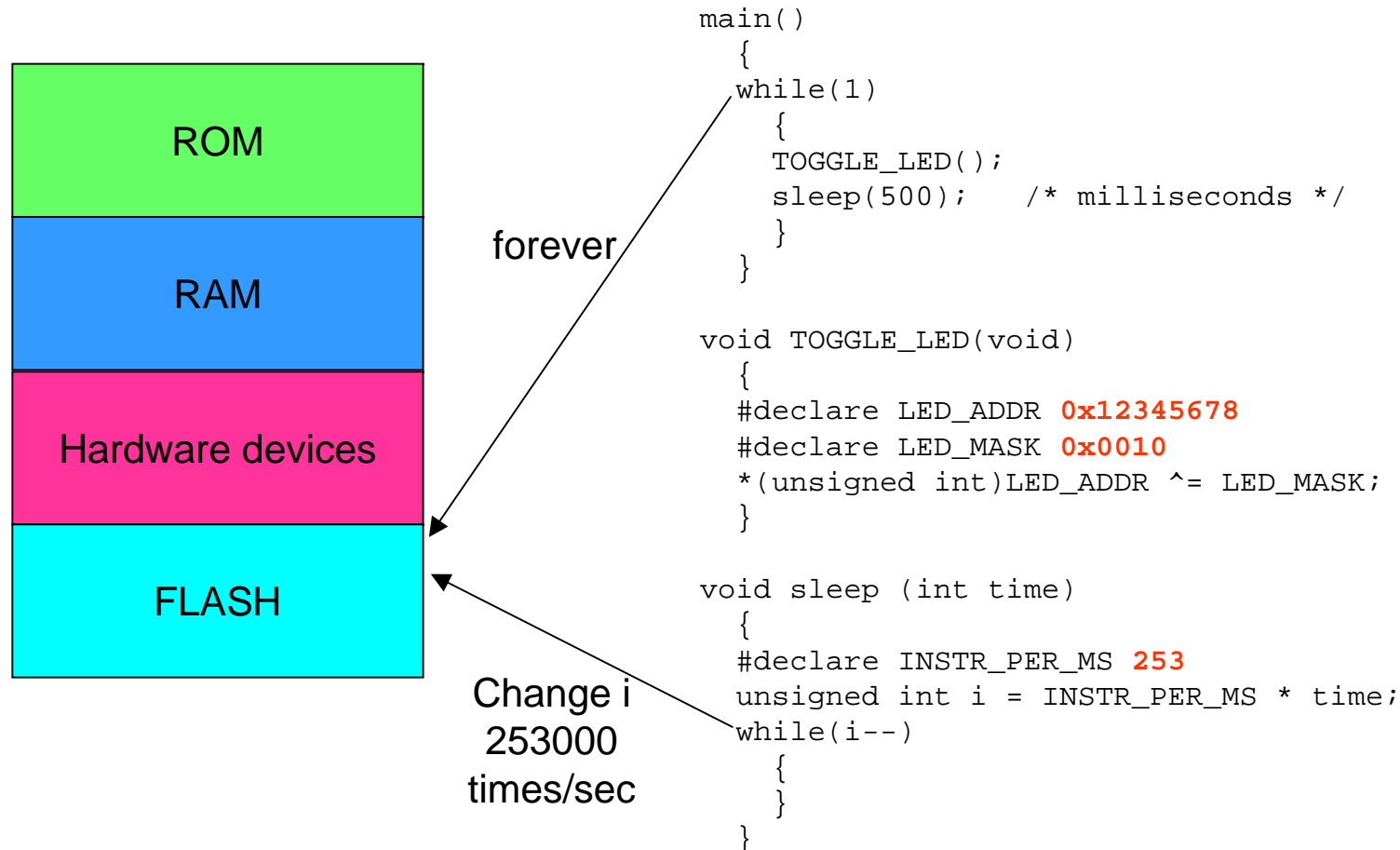
- FLASH (and other read-mostly memory technologies)
  - Is it RAM? Is it ROM?

# Memory and other practical issues



- FLASH (and other read-mostly memory technologies)
  - Is it RAM? Is it ROM?
- FLASH and many other nonvolatile memory technologies have a finite write cycle lifetime

# Memory and other practical issues



- FLASH (and other read-mostly memory technologies)
  - Is it RAM? Is it ROM?
- FLASH and many other nonvolatile memory technologies have a finite write cycle lifetime

## Assignment 2

- You are designing an embedded system realize a “Floating Display” digital clock/ caller-ID/ message display as shown below. A set of LEDs rapidly flash on a rotating arm to form the characters



- Besides automatically generated caller-ID messages and time/date displays, this device can be programmed to display announcements or user selected messages.
- Define a set of parameters and variables that might be needed to design this system and map them into various memory areas. Assume that you have as much memory and as many memory types as needed to design this system.
- (Optional) – Consider how you might control and synchronize this system to allow a stable and accurate information display

Image from Sharper Image web site: [sharperimage.com](http://sharperimage.com)

**Info Globe Scrolling LED Message Center with Caller ID #WI701**