Embedded Processors

Dongbing Gu

School of Computer Science and Electronic Engineering
University of Essex
UK

Spring 2018

Outline

Section 1

Microprocessors

Introduction

- In general-purpose computing, the Intel x86 architecture overwhelmingly dominating all.
- There is no such dominance in embedded computing.
- On the contrary, the variety of processors can be daunting to a system designer.
- We provide some brief descriptions on microprocessors, DSPs, and FPGAs.

Microprocessors

- A microprocessors is a small computer on a single integrated circuit consisting of a relatively simple central processing unit (CPU) combined with peripheral devices such as memories, I/O devices, and timers.
- The simplest microprocessors operate on 8-bit words.
- They may consume extremely small amounts of energy, and often include a sleep mode that reduces the power consumption to nanowatts.
- Embedded components such as sensor network nodes and surveillance devices can operate on a small battery for several years.

Microprocessors

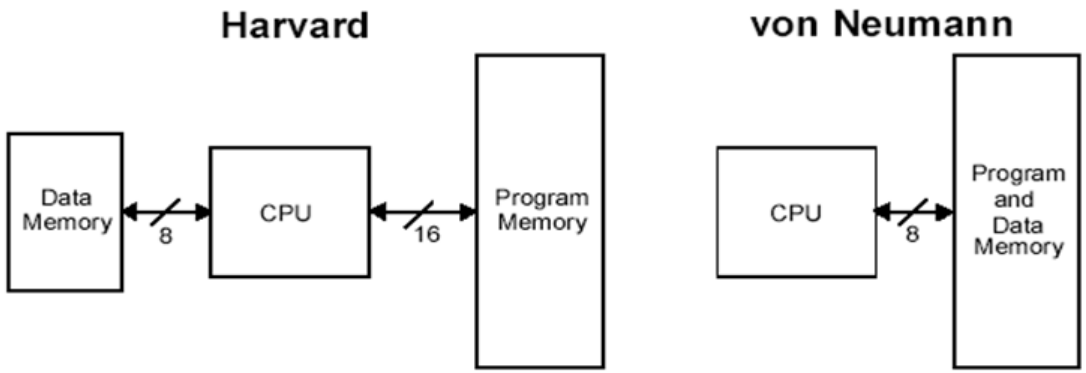
- Some of the architectures are quite old. The Motorola 6800 and Intel 8080 are 8-bit microcontrollers that appeared on the market in 1974.
- Descendants of these architectures survive today, for example, in the form of the Freescale 6811.
- Another very popular architecture is the Intel 8051, an 8-bit microcontroller developed by Intel in 1980.
- The 8051 ISA is today supported by many vendors (Atmel, Infineon Technologies, Dallas Semiconductor, NXP, ST Microelectronics, Texas Instruments, and Cypress Semiconductor)

Harvard Microprocessors

- Early days, instruction set is made as advanced and sophisticated as possible.
- The computer hardware is more complex, expensive and slower. Complex Instruction Set Computer (CISC)
- One characteristic is that instructions have different levels of complexity.
- Simple instructions could one byte long and complex ones could be several bytes long.

Harvard Microprocessors

- Harvard architecture has the program memory and data memory as separate memories which are accessed from separate buses.
- This improves bandwidth over traditional von Neumann architecture in which program and data are fetched from the same memory using the same bus.
- RISC (Reduced Instruction Set Computer) processors are based on the insight that simplified instructions can provide higher performance.
- They have fixed and wide instruction size with few formats. Data processing instructions operate only on registers, and can be very fast (compared to CISC).



RISC Processor Features

- One cycle execution time: RISC processors have a CPI (clock per instruction) of one cycle. This is due to the optimisation of each instruction on the CPU and a technique called pipelining. Each instruction is contained within a single binary word.
- Pipelining: a technique that allows for simultaneous execution of parts, or stages, of instructions to more efficiently process instructions. Every instruction takes the same amount of time to execute.
- Large number of registers: the RISC design philosophy generally incorporates a larger number of registers to prevent in large amounts of interactions with memory.

Microcontrollers

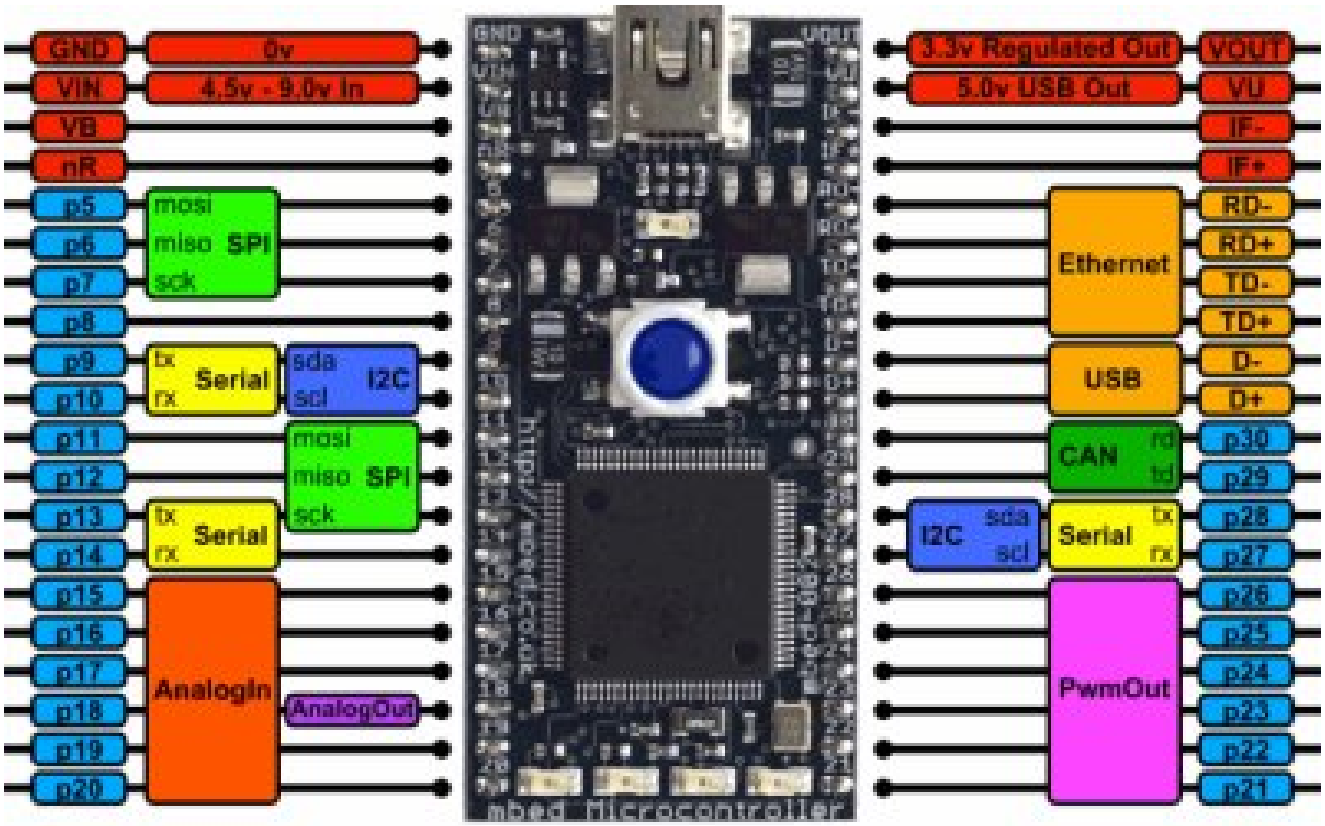
- Many 32-bit microcontrollers implement some variant of an ARM instruction set. (ARM originally stood for Advanced RISC Machine.)
- Processors that implement the ARM ISA are widely used in mobile phones and other systems.
- Semiconductor vendors license the instruction set from ARM Limited and produce their own chips, including Alcatel, Atmel, Broadcom, Cirrus Logic, Freescale, LG, Marvell Technology Group, NEC, NVIDIA, NXP, Samsung, Sharp, ST Microelectronics, Texas Instruments, VLSI Technology, Yamaha, and others.

Section 2

mbed NXP LPC1768

mbed NXP LPC1768

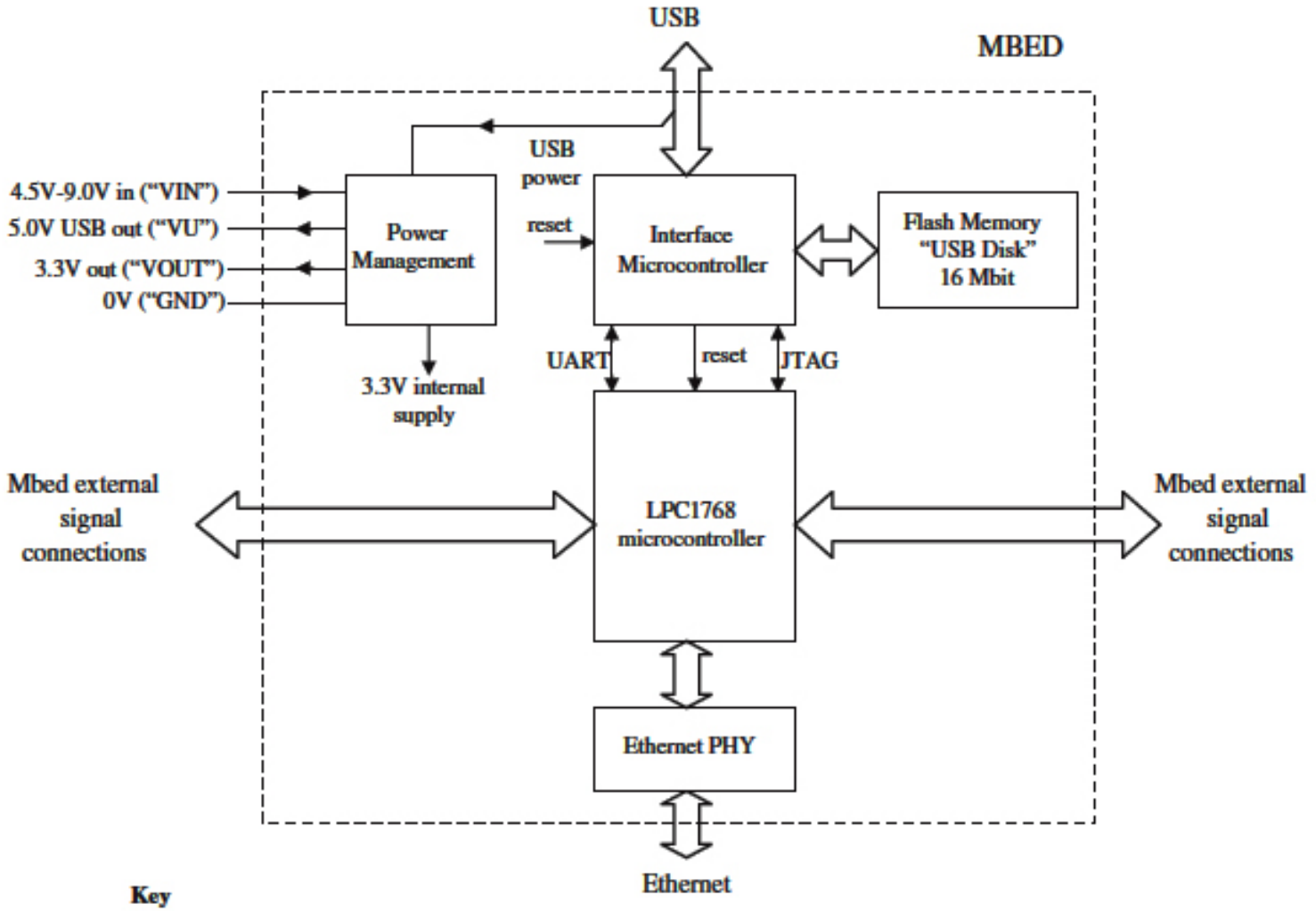
- NXP LPC1768 is an ARM Cortex-M3 based microcontroller.
- Designed for prototyping all sorts of devices, including Ethernet, USB, and the flexibility of lots of peripheral interfaces.
- Includes a built-in USB FLASH programmer.



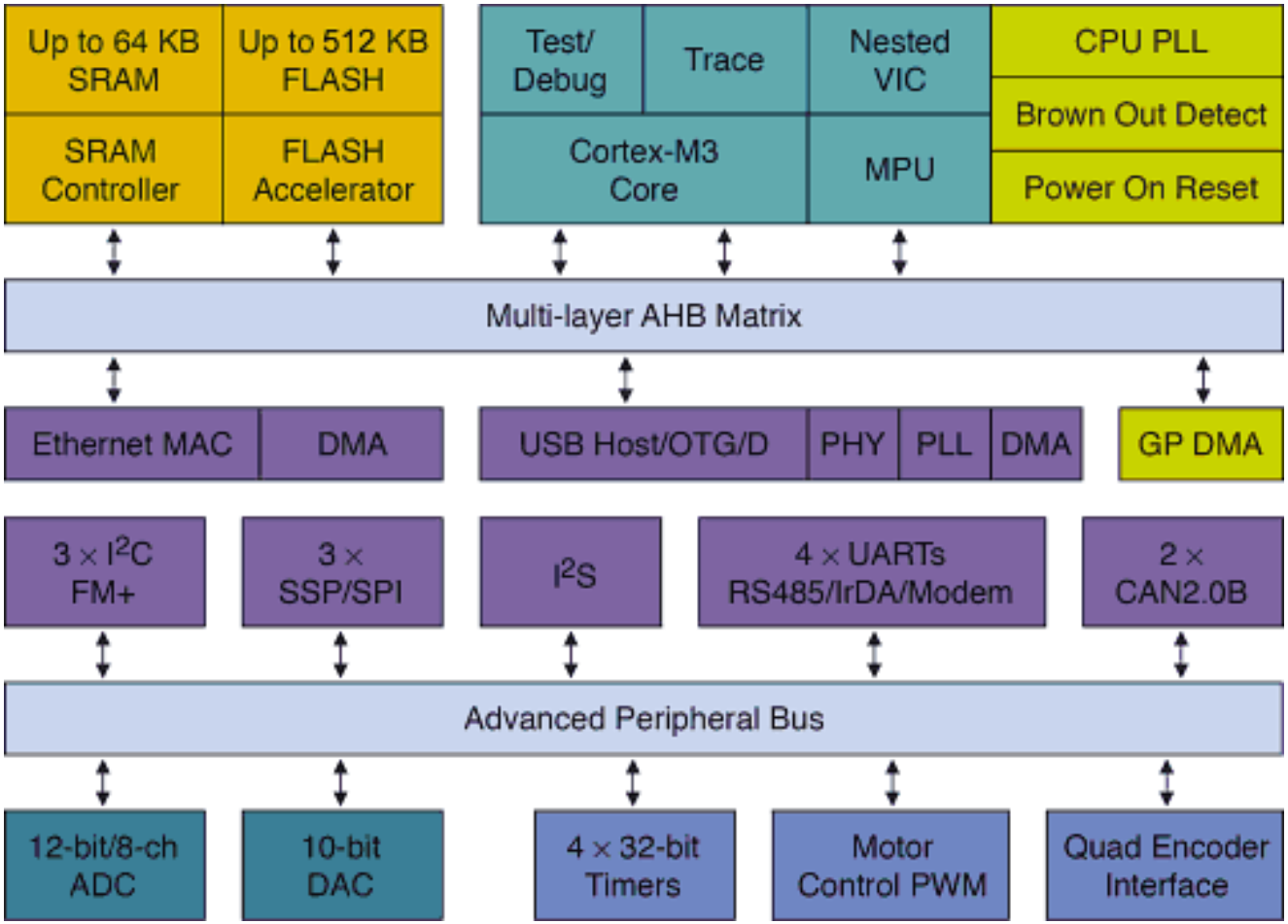
ARM microcontroller development boards

- A 32-bit ARM Cortex-M3 core running at 96MHz.
- 512KB flash, 64KB RAM
- built-in Ethernet, USB Host and Device, CAN, SPI, I2C, ADC, DAC, PWM and other I/O interfaces.
- Provide experienced embedded developers a powerful and productive platform for building proof-of-concepts.

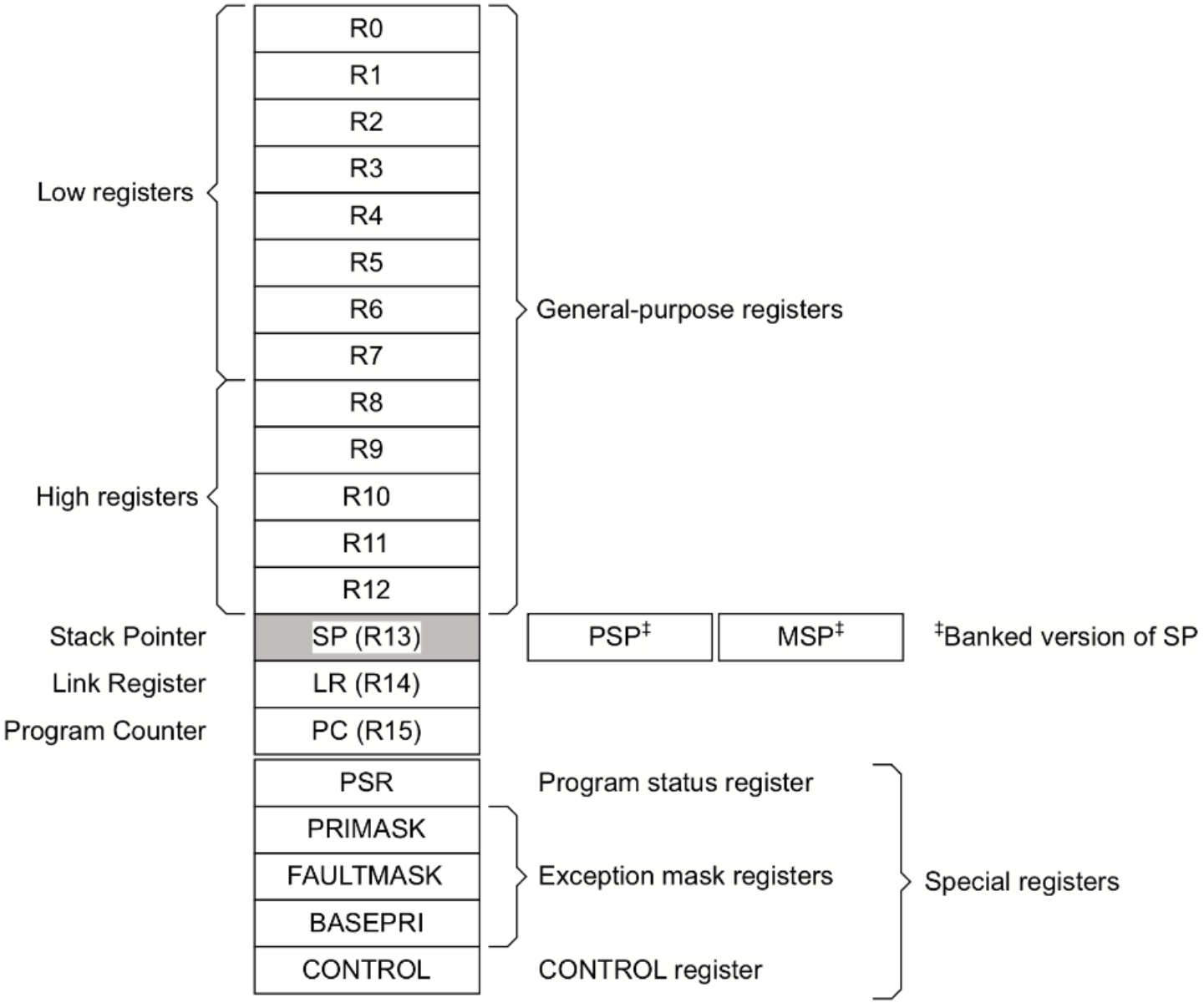
Block Diagram of mbed Architecture



The LPC1768 Microcontroller



Cortex M3 Core Registers

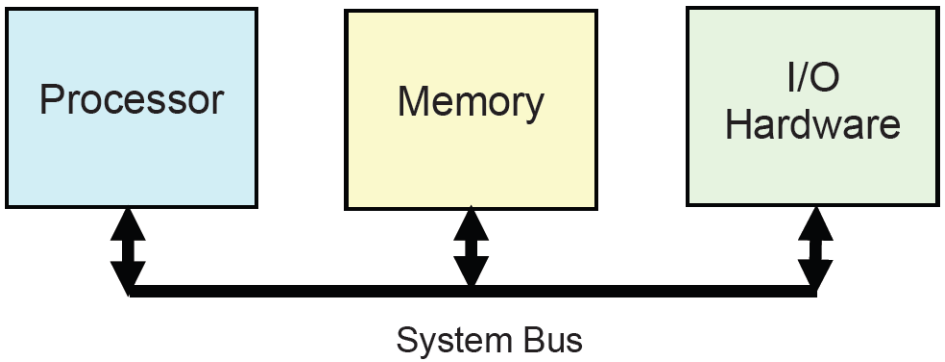


Section 3

I/O Architectures

Buses

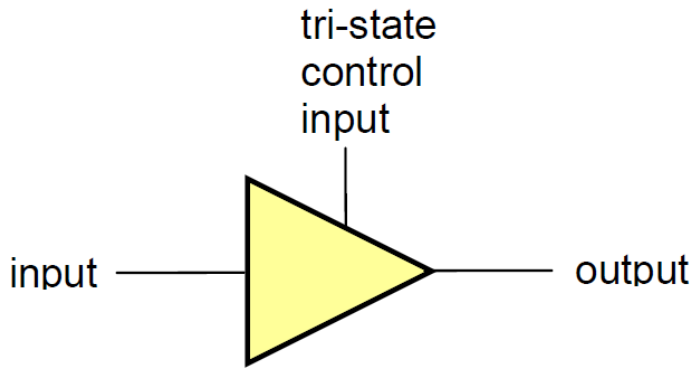
- Processor and I/O devices exchange data over a bus.
- The Industry Standard Architecture (ISA) bus was introduced on the first PCs by IBM in the early 1980s.
- The ISA bus supports 8-bit and 16-bit data transfers.
- PC/104 is an industrial version of this bus that is still used in some embedded systems.
- Later versions extended the ISA bus to a 32-bit bus called the Extended Industry Standard Architecture (EISA).



Tri-State

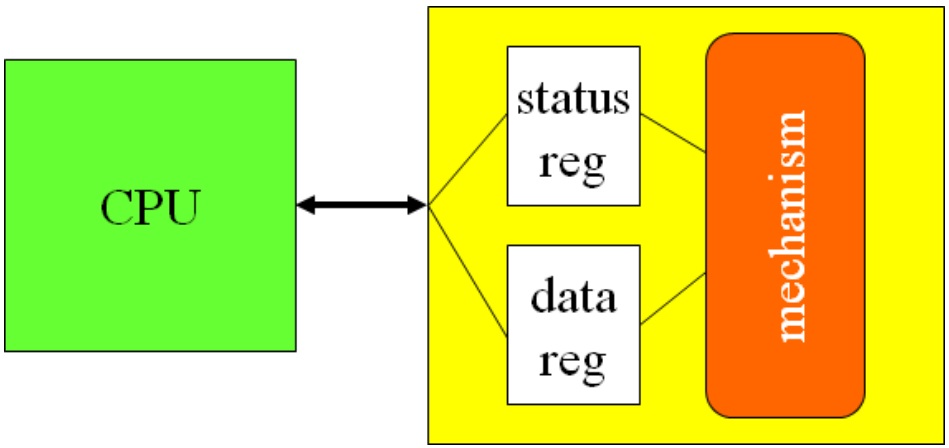
- Most bus signals are driven using tri-state logic devices.
- Recall that tri-state logic has a third state that is high impedance or not connected
- A tri-state buffer has an extra control line that makes it act just like a normal buffer or go to a high impedance (Z) state (i.e. disconnected).
- This allows multiple devices to drive the bus, but only one at a time.
- Only one device at a time ever turns on its tri-state control to force its outputs to drive the bus signals high or low.

control	input	output
0	0	High Z
0	1	High Z
1	0	0
1	1	1



I/O Registers

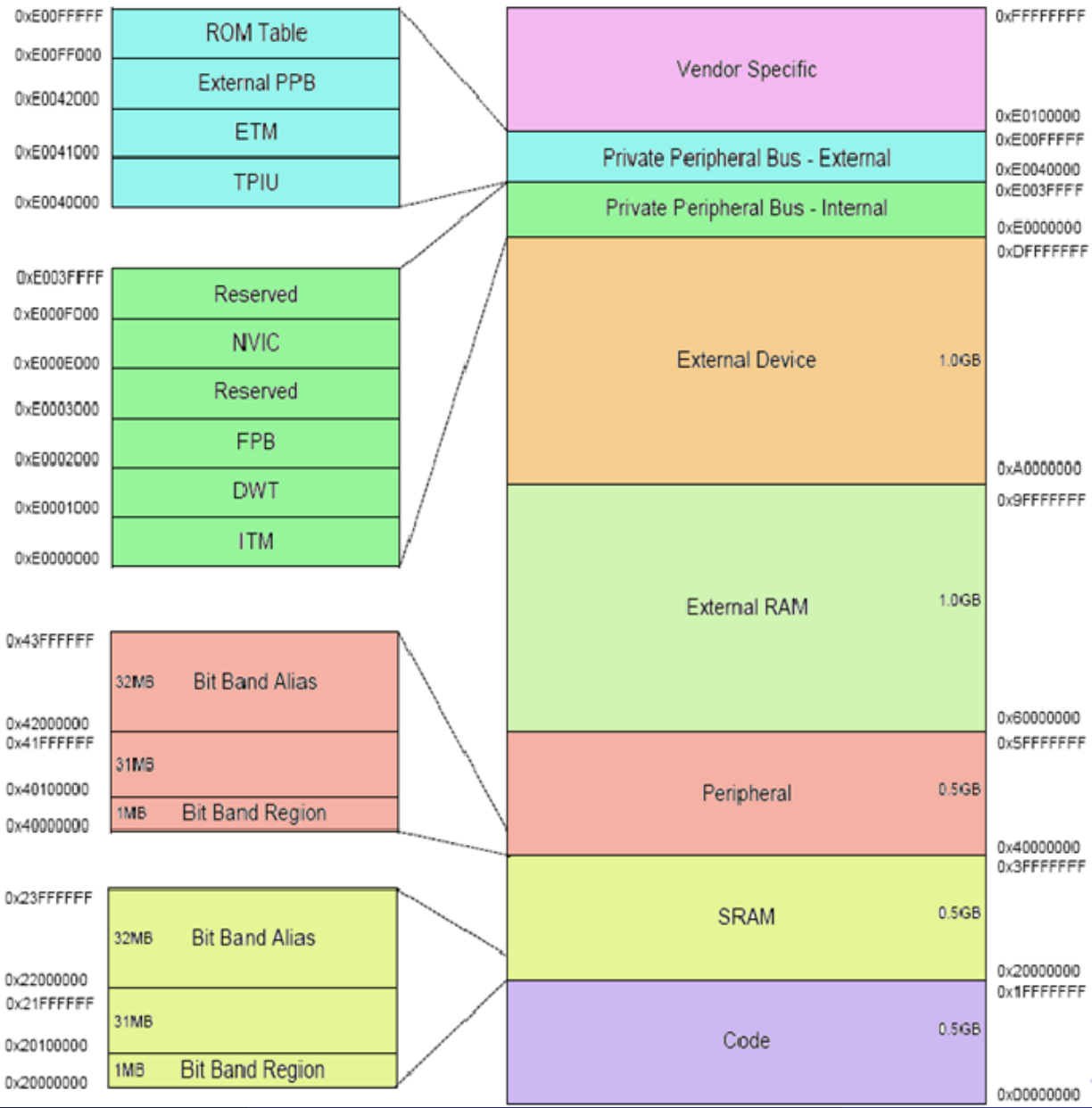
- Usually includes a CONTROL REGISTER that is used to initialise the peripheral's mode of operation.
- and a DATA REGISTER that is used to send the data to bus or read the data from bus.



Programming I/O

- Peripheral device registers usually look like memory locations to the program. This is called memory-mapped I/O.
  - memory-mapped load/store instructions.
  - I/O device registers are memory locations.
- A section of the memory map is reserved for I/O devices
- Most CPUs use memory-mapped I/O.
- But Intel x86 CPU provides in, out instructions.
  - special-purpose I/O instructions (OUT $< port >, < reg >$)
  - I/O device registers have device identifier "address".

Memory map on the mbed



Memory-mapped I/O

- The on-chip I/O registers can be accessed via the mbed header file **mbed.h**.
- The mbed API DigitalOut

```
#include "mbed.h"
DigitalOut myled(LED1);
int main(){
    while (1){
        myled=1;
        wait(0.2);
        myled=0;
        wait(0.2);
    }
}
```

DigitalOut API

- The DigitalOut API component creates a C++ class, called DigitalOut.
- The class member functions include: DigitalOut, write, read, operator =, and operator int().
- myled=1; replace mylead.write(1);
- External pins: 26 pins (pins 5-30) can be configured as digital inputs.

DigitalIn API

- The DigitalIn API component creates a C++ class, called DigitalIn.
- The class member functions include: DigitalIn, read, mode, and operator int().
- mode PullUp, PullDown, PullNone

BusOut API

- The BusOut API class allows you group a set of digital outputs into one bus, so that you can write a digital word to it.
- The BusOut Interface can be used on any pin with a blue label, and also with the on-board LEDs (LED1-LED4).
- The class member functions include: BusOut, write, read, operator =, and operator int().

```
#include "mbed.h"
BusOut display(p5,p6,p7,p8,p9,p10,p11,p12);
int main(){
    while (1){
        for(int i=0;i<4;i++){
          case 0 display=0x3F;break;
          ...
        }
    }
}
```

Port API

- The PortOut API is used to write to an GPIO port as one value.
- The class member functions include: PortOut, write, read, operator =, and operator int().
- Port0-Port5 can be used.

```
#include "mbed.h"

#define LED_MASK 0x00B40000
// LED1 = P1.18  LED2 = P1.20  LED3 = P1.21  LED4 = P1.23
PortOut ledport(Port1, LED_MASK);

int main() {
    while(1) {
        ledport = LED_MASK;
        wait(1);
        ledport = 0;
        wait(1);
    }
}
```

Timers

- Four general purpose 32-bit timers.
- One Interrupt Timer
- One System Tick Timer
- The class member functions include: start(),stop(), reset(), read(), read_ms(), read_us()

```
#include "mbed.h"

Timer t;

int main() {
    t.start();
    printf("Hello World!\n");
    t.stop();
    printf("The time taken was %f seconds\n", t.read());
}
```

Timer Interrupts

- Timer interrupt is set up by the Timeout class.
- It allows an interrupt to call a function (ISR) after a specified delay

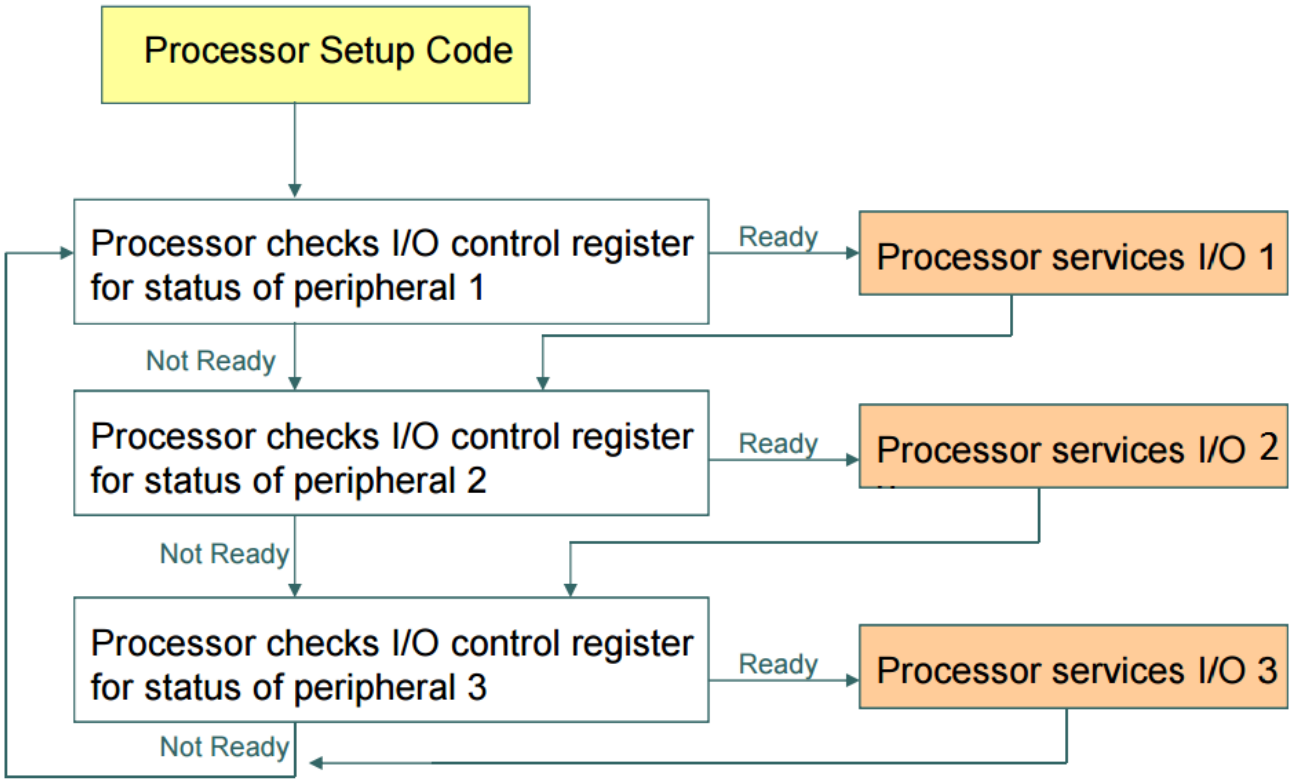
```
#include "mbed.h"
Timeout flipper;
DigitalOut led1(LED1);
DigitalOut led2(LED2);
void flip() {
    led2 = !led2;
}
int main() {
    led2 = 1;
    flipper.attach(&flip, 2.0); // setup flipper to call flip ↩
        after 2 seconds
    while(1) {
        led1 = !led1;
        wait(0.2);
    }
}
```

Section 4

Interruption Mechanisms

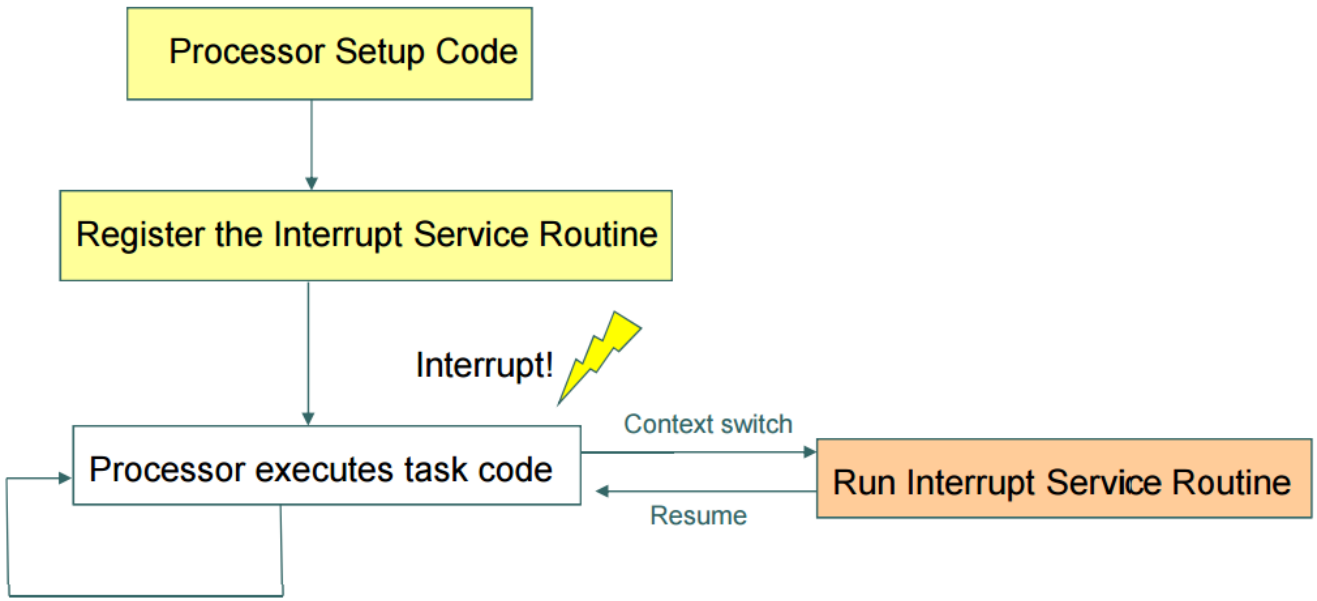
Pin Polling

- Polling
  - Main loop uses each I/O device periodically.
  - If output is to be produced, produce it.
  - If input is ready, read it.

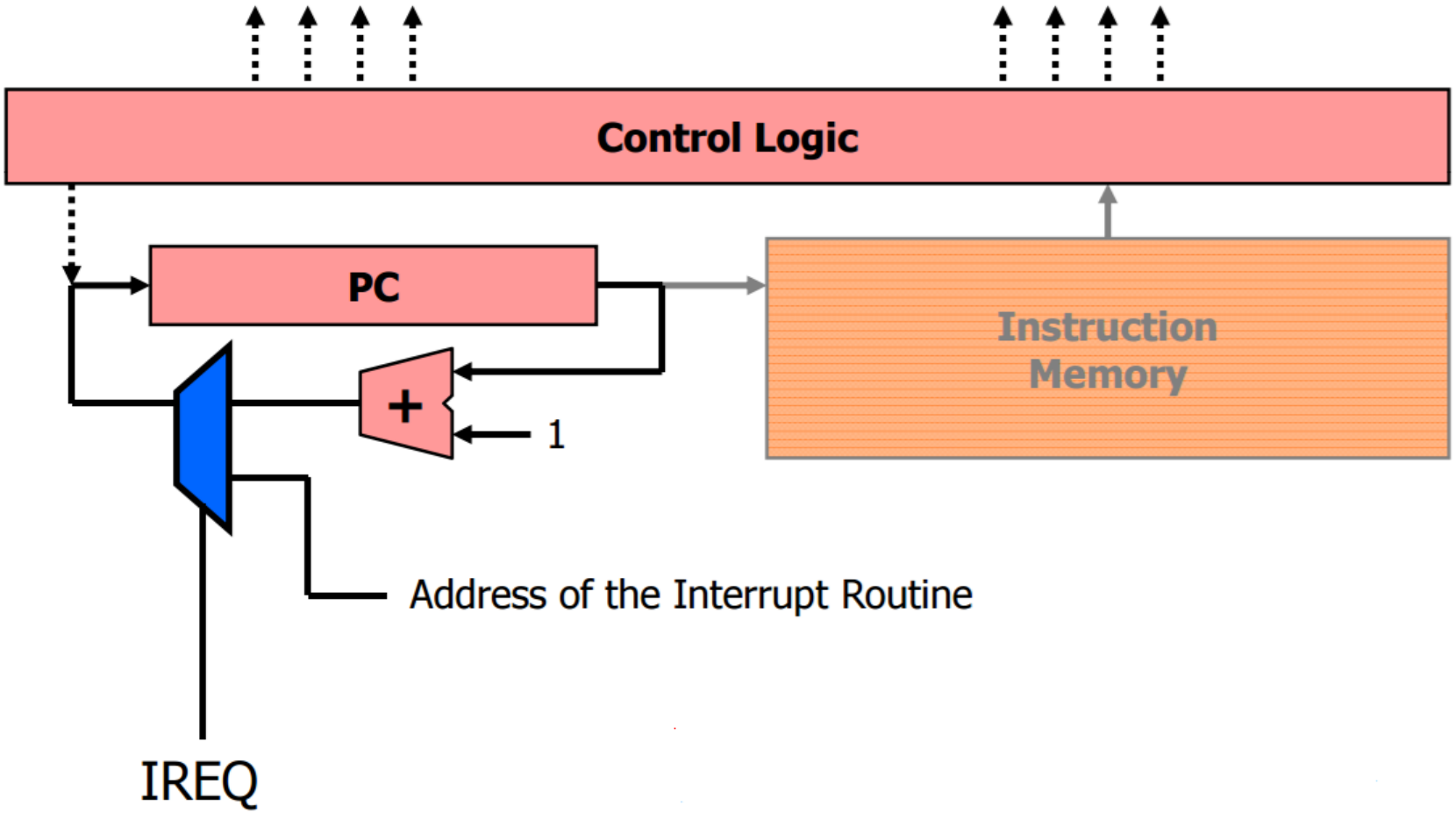


Interrupt I/O

- Interrupts
  - External hardware alerts the processor that input is ready.
  - Processor suspends what it is doing.
  - Processor invokes an interrupt service routine (ISR).
  - ISR interacts with the application.



Interrupt I/O



Interrupt I/O

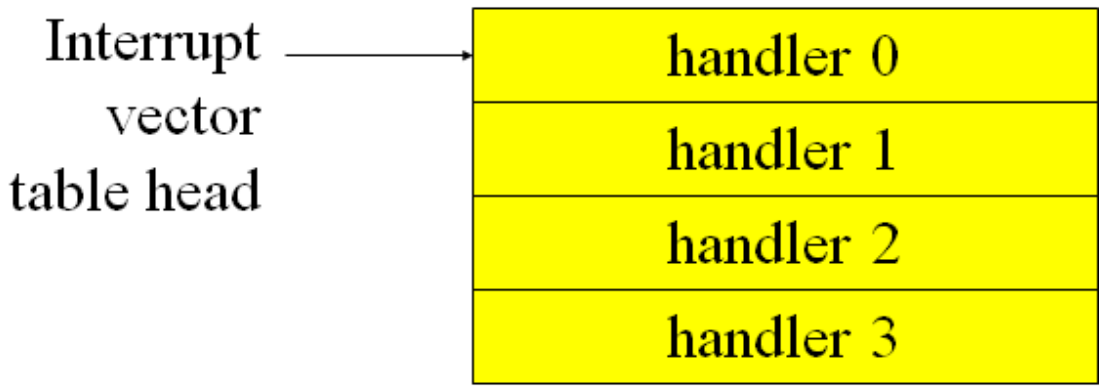
- Port or pin "Polling" is very inefficient.
  - CPU can't do other work while testing device;
  - Hard to do simultaneous I/O.
- Interrupts allow a device to change the flow of control in the CPU.
  - Causes subroutine call to handle device.
- When status register sets, it calls interrupt
  - CPU ends current instruction,
  - and then executes the device's predefined subroutine.
- In ISR you should avoid any call to wait, infinitive while loop, or blocking calls in general.

Interrupt I/O

- Triggers:
  - A level change on an interrupt request pin.
  - Software interrupts, such as timer.
- Responses:
  - Disable interrupts
  - Push the current program counter onto the stack.
  - Execute the instruction at a designated address in the program memory.
- Design of interrupt service routine:
  - Save and restore any registers it uses.
  - Re-enable interrupts before returning from interrupt.

Interrupt Vectors

- Allow different devices to be handled by different codes.
- Each device number is an OFFSET in this vector of addresses.
- So the vector may be stored anywhere in main memory and the device offset added to the vector (table)'s base address.
- Interrupt vector table:



Interrupt on the LPC1768

- Management of all the interrupts is undertaken by the Nested Vectored Interrupt Controller (NVIC).
- NVIC offers very fast interrupt handling and provides the vector table as a set of real vectors (addresses).
- Saves and restores automatically a set of the CPU registers (R0-R3, R12, PC, PSR, and LR).
- For Cortex M3, it has 33 interrupt sources and 32 possible priority levels

InterruptIn API

- The InterruptIn interface is used to trigger an event when a digital input pin changes.
- mbed NXP LPC1768: Any of the numbered mbed pins (p5-p30) can be used as an InterruptIn, except p19 and p20.
- The class member functions include InterruptIn, rise, fall, enable_irq (), disable_irq ()

```
#include "mbed.h"
InterruptIn button(p5);
DigitalOut led(LED1);
DigitalOut flash(LED4);
void flip() {
    led = !led;
}
int main() {
    button.rise(&flip);
    while(1) {
        flash = !flash;
        wait(0.25);
    }
}
```

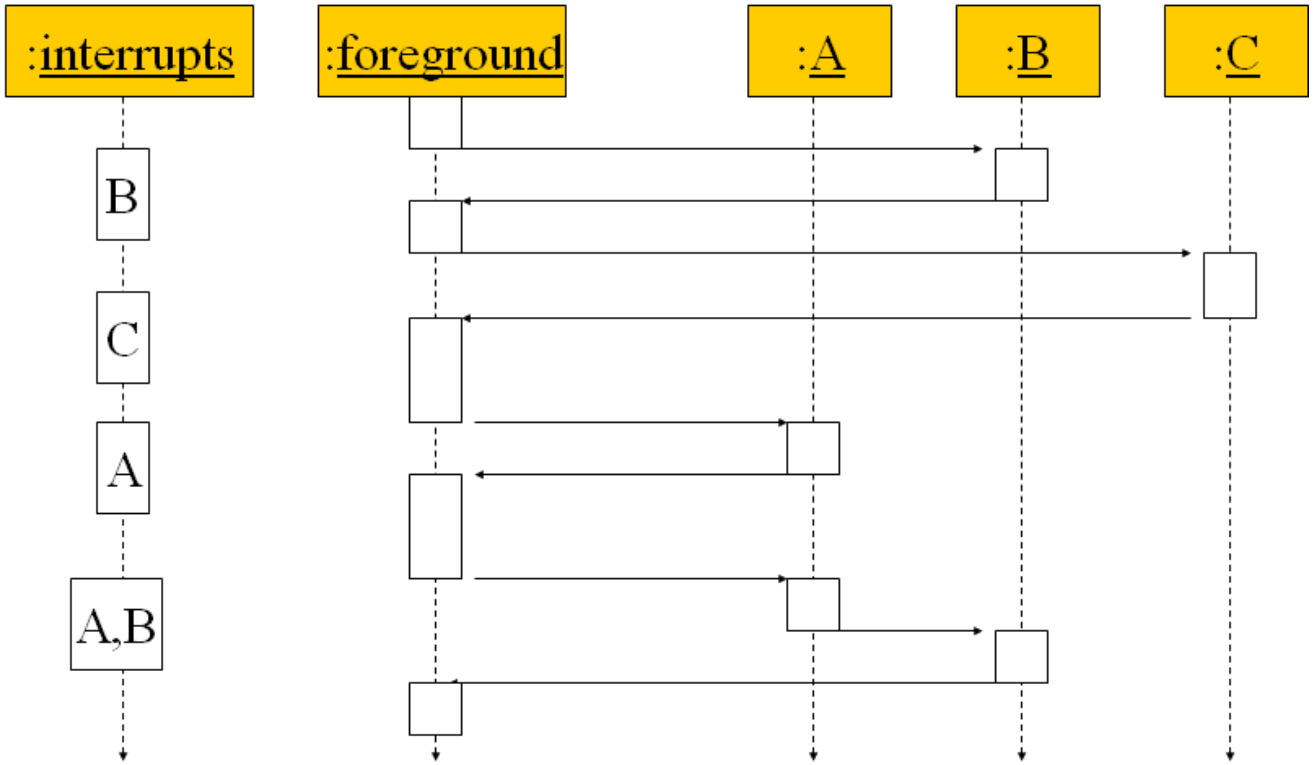
Debugging interrupt code

- Enabling interrupts could lead to problems in program execution.
- ISR must save and restore any CPU registers it uses. If this is not done properly then the foreground program can exhibit mysterious bugs.
- Bugs will be hard to repeat - depend on interrupt timing.
- An advantage in using C is that the compiler can automatically produce code to save and restore registers (so your ISR does not need to).
- You do need to reset the interrupt flag for the interrupting device, to avoid continuous IRQs.

Priorities and vectors

- Two mechanisms allow us to make interrupts more specific:
  - Priorities determine which interrupt gets CPU first if several sources request at the same time.
  - Vector determines what code is called for each type of interrupt.
- Most CPUs provide both.

Prioritized I/O



Concurrency in Computing

- Interrupt Handling
  - Reacting to external events (interrupts)
  - Exception handling (software interrupts)
- Processes
  - Creating the illusion of simultaneously running different programs (multitasking)
- Threads
  - How is a thread different from a process?
- Multiple processors (multi-cores)

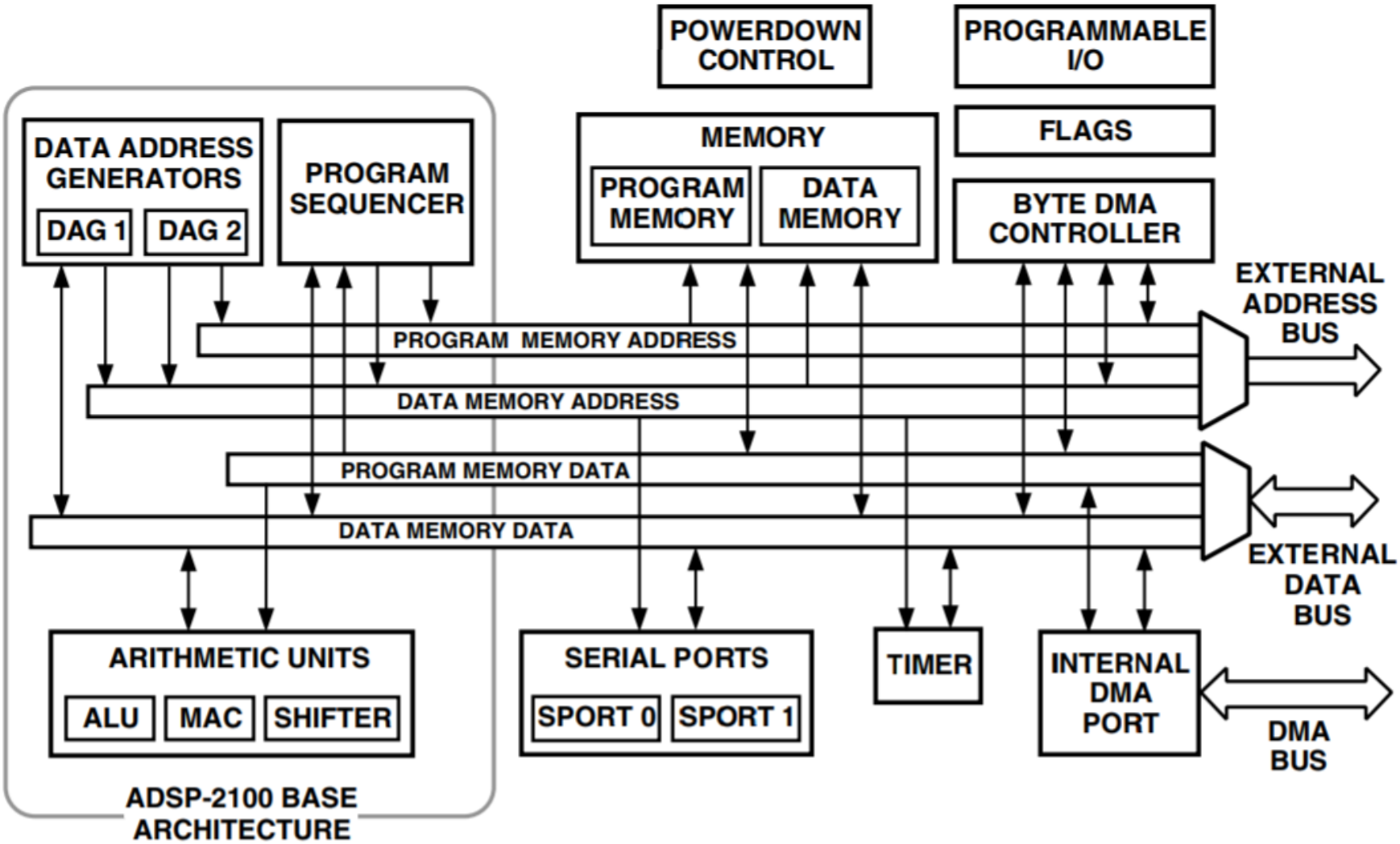
Section 5

DSP Processors

DSP Processors

- Processors designed specifically to support numerically intensive signal processing applications are called DSPs (digital signal processors).
- They deal with large amounts of data.
- They typically perform sophisticated mathematical operations on the data, including filtering, system identification, frequency analysis, machine learning, and feature extraction.
- The combination of design elements: arithmetic operators, memory handling, instruction set, parallelism, data addressing that provide this ability forms the key difference between DSPs and other kinds of processors.

Typical DSP architecture



DSP architecture - Numeric Section

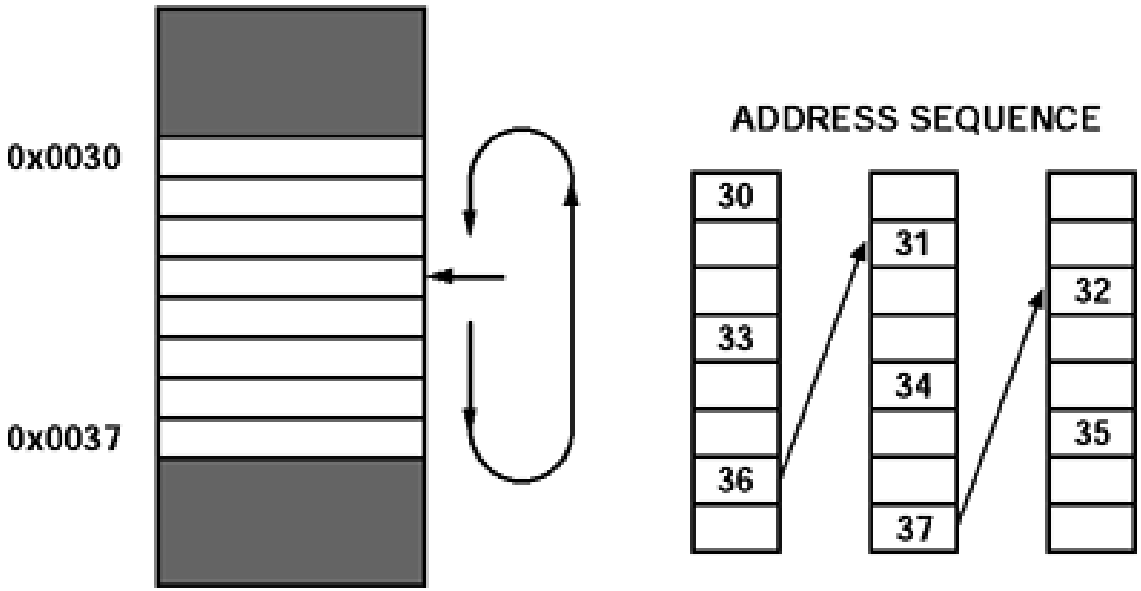
- DSPs must complete multiply/accumulate, add, subtract, and/or bit-shift operations in a single instruction cycle.
- It is this hardware that distinguishes DSPs from general-purpose microprocessors.
- The MAC (multiplier-accumulator) performs sum-of-products operations. ALU capabilities include addition, subtraction, and logical operations. Operations on bits and words occur within the shifter.

DSP architecture - Memory Section

- Data and instructions must flow into the numeric and sequencing sections of the DSP on every instruction cycle.
- There can be no delays, no bottlenecks. Everything about the design focuses on throughput.
- In DSPs, memory is divided into program and data memory-with separate busses for each - Harvard architecture.
- Data address-generators (DAGs): reduce overhead and automatically manage memory accesses.

DSP architecture - Memory Section

- Circular buffer: DSP algorithms usually require data in a range of addresses (a buffer) to be addressed so that the address pointer "wraps-around" from the end of the buffer back to the start of the buffer (buffer length).



DSP architecture - Sequencer Section

- The program sequencer needs to loop through the repeated code without incurring overhead while getting from the end of the loop back to the start of the loop.
- DSPs perform these test and branch functions in hardware, storing the needed addresses.

DSP Processors

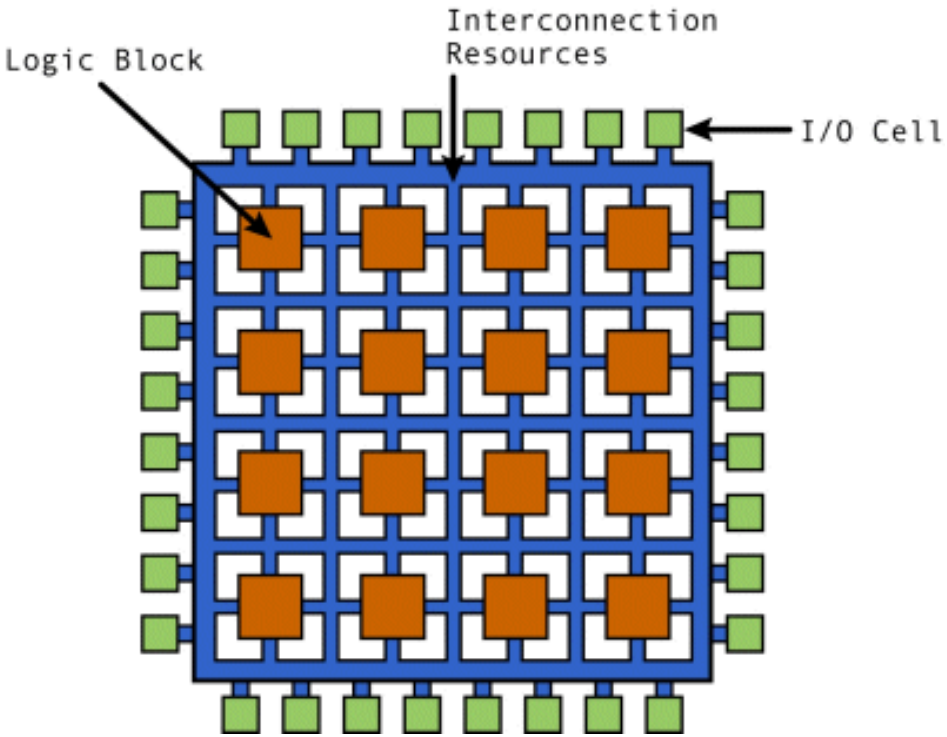
- Several variants of the Harvard architecture; and addressing modes supporting auto increment, circular buffers, and bit-reversed addressing.
- Most support fixed-point data precisions of 16-24 bits, typically with much wider accumulators (40-56 bits).
- A few DSPs have appeared with floating point hardware.
- DSPs are difficult to program compared to normal microprocessors, primarily because of complex specialised instructions and pipeline.
- Until the late 1990s, these devices were almost always programmed in assembly language. Even today, C programs make extensive use of libraries that are hand-coded in assembly language.

Section 6

FPGAs

FPGAs

- A field-programmable gate array (FPGA) is an integrated circuit designed to be configured by a customer or a designer after manufacturing.
- The FPGA configuration is generally specified using a hardware description language (HDL).



FPGAs

- FPGAs contain programmable logic components called "logic blocks", and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together"
- Logic blocks can be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR.
- In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory.
- The ability to update the functionality after shipping, partial re-configuration of a portion of the design and the low non-recurring engineering costs offer advantages for many applications.