

# 1.20 Processor Technology

EE302 – Real time and embedded systems

1

## Overview

2

### □ Aims

介绍与嵌入式系统最相关的处理器技术概念

- Introduce the processor technology concepts most relevant to embedded systems

### □ Learning outcomes – you should be able to...

- Differentiate the 3 principle processor types by their advantages and disadvantages
- Select the appropriate processor type for an embedded system

根据它们的优点和缺点来区分3种主要的处理器类型

为嵌入式系统选择适当的处理器类型

29 September 2020

2

## Considerations when selecting a processor

3

- Power consumption
- Performance
- Cost – per unit vs. NRE
- Development support and ease of development
- Time to prototype/time to market
- Flexibility and upgrade path
- And, last but not least, the developer's familiarity with processor

29 September 2020

3

## We consider 3 types of processor

4

1. **General purpose processor (GPP)**
  - Programmable for any application
  - usually requires external support hardware
  - Example: Motorola 68000, Intel Pentium
2. **Application Specific Instruction Processor (ASIP)**
  - Optimized for specific application areas, but still software programmable
  - Example: Microcontroller, Digital Signal Processor (DSP)
3. **Single purpose processor (SPP)**
  - Optimized for one specific task only with no flexibility
  - Algorithm “hard coded” in finite state machine, not software programmable
  - Example: UART, JPEG encoder, GSM codec

*A complete system might include all 3 processor types.*

29 September 2020

4

# 1. General Purpose Processor (GPP)

5

- A GPP is designed for (almost any) general purpose computation task
- carefully designed by manufacturer
  - Manufacturer's large NRE cost is offset by large volume sales
  - Programmable, maximum flexibility, general purpose, not specialised to any one task
  - Support hardware is generally not integrated – must be provided externally
- Examples: x86 (Intel or AMD) processors, ARM CPUs, older processors such as Z80, 68000, etc.



29 September 2020

5

## GPP Contd.

6

- For the embedded system designer...
  - ~ Moderate to high unit cost
  - ✓ No NRE to design processor/FSM required
  - ✗ NRE required for support hardware and I/O
  - ✓ Easiest and most flexible for software (minimizing software NRE)
  - ✓ Often provides the highest performance for applications that are dominated by software
  - ✓ Flexible for later upgrades
  - ✓ Can be short time-to-market/prototype (depending on hardware needs)
  - ✗ Power consumption often (much) higher than other processors

29 September 2020

6

## 2. Application Specific Instruction Processor (ASIP)

7

- An ASIP is designed for a specific application area or domain
  - Example domains: embedded control, digital signal processing, video processing, network processing, telecommunications, etc.
  - The ASIP contains architectural features specific to the domain
    - integrated support hardware
    - optimized instructions and operations
  - It is still programmable with software, but is not as flexible as a GPP (or at least not as well suited to certain application types)
- ASIP Examples:
  - Microcontroller (as we'll be using in this module)
  - Digital Signal Processor

29 September 2020

7

## ASIP Example 1: Microcontroller

8

- A microcontroller is optimized for embedded control applications
- In such applications...
  - Typical tasks are reading sensors, setting actuators (digital, analog), communications
  - Mostly dealing with events (bits): data is present, but not in huge amounts
- Typical microcontroller features
  - On-chip peripherals
  - On-chip program and data memory
  - Direct programmer access to many of the chip's pins
  - Specialized instructions for bit-manipulation and other low-level operations



*For information on common microcontroller families, see  
<https://predictabledesigns.com/how-to-select-the-microcontroller-for-your-new-product/>*

29 September 2020

8

## ASIP Example 2: Digital Signal Processor (DSP)

9

- A DSP chip is optimized for signal processing applications
  - e.g. mobile phone voice coding, DVD decoding, digital camcorder, music synthesizer
  - Large amounts of digitized data, often streaming
  - Data transformations must be applied fast
- Typical DSP features
  - Useful I/O such as ADC and DAC on board
  - Several instruction execution units
  - Special purpose instructions
    - Full multiply-accumulate (MAC) in one instruction cycle
      - $a = a + (b * c)$
    - Efficient vector operations – e.g., add two arrays
      - Vector ALUs, loop buffers, etc.
  - Additionally has common GPP instructions

29 September 2020

9

## ASIP contd.

10

- For the embedded system designer...
  - ✓ Very low unit cost (often much less than GPP + support hardware)
  - ✓ No NRE to design processor/FSM required
  - ✓ Useful support hardware often integrated, so little or no additional hardware NRE required
  - ✓ Very short time-to-market/prototype
  - ✓ Dedicated instructions/operations for domain specific tasks
  - ~ Application details still defined by software and fairly flexible, though not as flexible as GPP
  - ✗ Resource constrained and may require more software “tricks” (can affect algorithms, data limits, etc.)

29 September 2020

10

### 3. Single Purpose Processors (SPP)

11

- Might have features of GPP or ASIP, but the key difference is that it has a custom designed control unit
  - No “program code” in memory
  - Instead the control unit usually implements a (hard coded / fixed) finite state machine (FSM) where state transitions depend on current state and current inputs
  - The FSM may be simple or complex
  - Off the shelf SPPs are available for many useful tasks
  - You might design your own SPP if you are developing a new algorithm or heavily optimising something (e.g. creating a very low power AI accelerator)
- Example (off the shelf) SPPs:
  - Standalone Universal Asynchronous Receiver/Transmitter (UART) for serial communications
  - Direct Memory Access (DMA) unit for high speed large data input/output without main processor intervention
  - Memory management unit to handle memory caching
  - H.264 or HEVC video hardware decoder

29 September 2020

11

### SPP: Custom SPPs

12

- To design a custom SPP
  - Normal “sequential” algorithm must be converted to a FSM with data
  - Resulting states & logic must be optimized
  - Specialized tools and techniques available
- Possible implementations
  - Application Specific Integrated Circuit (ASIC)
  - Programmable logic devices (PLDs) such as Field Programmable Gate Array (FPGA)
- ASIC vs. FPGA
  - FPGA more expensive per unit, higher power consumption, but more flexible (can be programmed many times) and easier to use (so lower NRE)
  - ASICs may be faster, have lower power consumption, and be cheaper per unit (in bulk)

29 September 2020

12

- For the embedded system designer...
  - ✗ Processor/FSM design required so high NRE cost
  - ✓ Necessary support hardware usually integrated in the design
  - ✗ Longer time-to-market/prototype
  - ✗ Not flexible – requires a h/w redesign
  - ✓ Main advantages (usually not all at once)
    - Cheap unit cost (depends on NRE/num units)
    - Simple
    - Small
    - Fast
    - Low power

## Example questions

- Q1. When is a GPP the best processor to use?*
- Q2. When is an ASIP the best processor to use?*
- Q3. When is an SPP the best processor to use?*

### *Examples to consider:*

- *Ticket collection machine in cinema*
- *Broadband router*
- *SIM card*
- *Computer keyboard*
- *Digital camera*

## Inside the processor

15

29 September 2020

15

## Key properties of a processor

16

In labs we will use the Microchip PIC 16F877 microcontroller.

This device is a RISC processor, which uses the Harvard architecture, with 8 bit data and 14 bit opcodes. It supports 8K opcodes in programme memory and just 368 bytes of RAM.

Most smart phones are based on ARM processors

The ARM family are much more sophisticated than the PIC (though still relatively simple). Again these are RISC processors, but they utilise a multiple stage pipeline to speed code execution.

*Q. What are the practical consequences of the underlined properties for the embedded systems engineer?*

29 September 2020

16



## Processor instruction set

17

- Processors are usually classified as CISC or RISC
- **CISC** (complex instruction set computing), E.g. Intel x86 family
  - The main goal is to complete a task with as few instructions as possible
  - To achieve it, the processor hardware is more complex to handle a wide variety of instructions, each of which potentially has a series of underlying operations
- **RISC** (reduced instruction set computing), E.g. ARM family
  - The focus is on really simple instructions that can be executed in one instruction cycle
  - To achieve this the hardware is simplified (and perhaps speeded up) and the software tools such as the compiler take on more work to convert high level code to an appropriate sequence of simple instructions

29 September 2020

17

## RISC vs CISC contd

18

- **CISC**
  - Program needs fewer instructions than RISC, because instructions can be more expressive and vary in complexity
  - Instructions are not orthogonal (regular) – instead many special rules about which registers and addressing modes to use
  - Instruction timing depends greatly on the instruction
  - Can write denser code and require less program memory
  - Processor is more complex and more transistors = more power consumed
- **RISC**
  - Instructions are simple, orthogonal, and all of similar complexity
  - The program machine code may need to be more verbose (since more simple instructions may be required to accomplish something) – requires more work from the compiler and usually requires larger program memory
  - Processor can be much simpler: fewer transistors = less power
  - The few simple instructions can be highly optimised, so execution may be faster (for a given clock speed)

29 September 2020

18

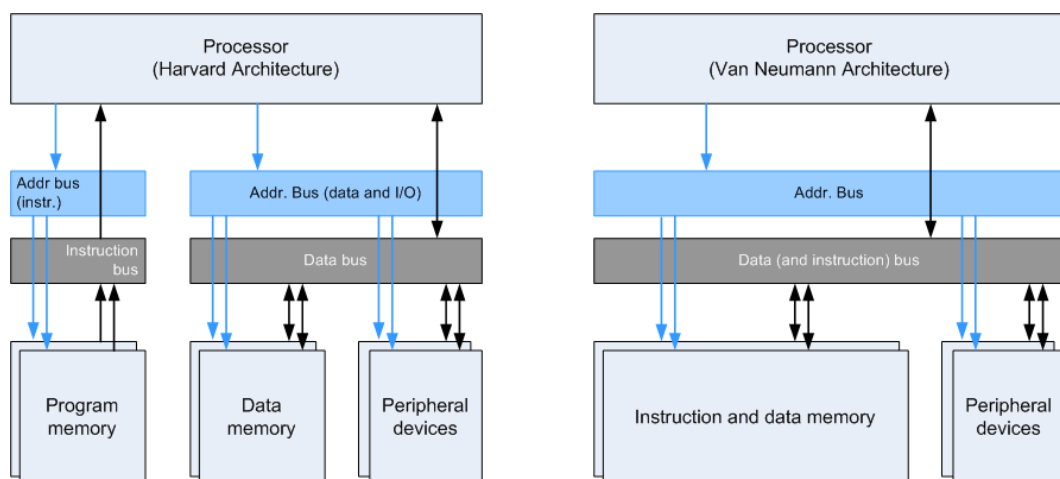
### □ Relevance to embedded designer

- Although high level languages (e.g. C) hide the instruction set details, program size, power consumption, and execution speed matter
- Additionally, certain embedded applications require some functions to be programmed in assembly language (human readable machine language instructions) to achieve the very best performance or timing accuracy

29 September 2020

19

## Memory architecture



29 September 2020

20

### □ Relevance to embedded designer

- At a given clock speed, if there is no memory cache (typical on low end processors), the Harvard architecture will allow faster execution because instructions and data can be read simultaneously
- The program memory and data memory can be optimised specifically for the processor, e.g.
  - The PIC 16 data memory is 8 bits wide in each location
  - the PIC 16 uses program memory that is 14 bits wide in each location because opcodes are 14 bits wide
  - 14 is not a multiple of 8, so it avoids wasted bits if we use exactly 14 bits of storage per opcode rather than 16 (if a Van Neumann architecture were used). This saves transistors and power, e.g. on PIC which can store 32Ki instructions could save  
 $32768 \text{ instructions} * 2 \text{ bits per instruction} * 1 \text{ transistor per bit of ROM} = 65536 \text{ transistors}$

### Example question

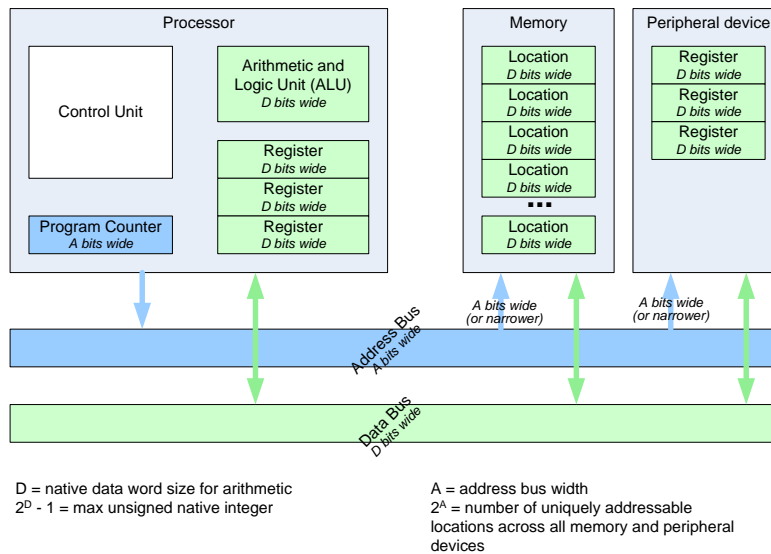
*The PIC family are RISC processors based on the Harvard architecture*

*Q1. Briefly explain each of these properties*

*Q2. What is the practical consequence of each of these properties*

## Data and Address Width

23



Note: this picture is for a processor using the Van Neumann architecture. The Harvard architecture would require separate programme memory.

29 September 2020

23

## Contd.

24

- Data bus, data width (in bits), D
  - 8 bit, 16 bit, and 32 bit processors are common for embedded systems
  - Integer arithmetic
  - Internal data bus vs. external data bus (usually same, but not always)
- Address bus, address width (in bits), A
  - Values on address bus determine which location will be accessed next (e.g. to fetch an instruction, or load/store data to memory)
  - Native size of the address space ( $2^A$  locations)
  - External logic can increase limits using segmented or paged addressing but this is less convenient



29 September 2020

24

- Relevance to embedded designer
  - **D** determines the native integer size which determines the kind of arithmetic that can be done with single instructions
  - E.g. an 8 bit processor will require several instructions just to add two 10 bit numbers (such as 1000 + 1000) which will in turn make is slower than a 16 bit processor in this situation
  - **A** determines the max addressable memory
  - For flexibility it is beneficial for A to be large, but this requires extra pins on the processor package (sometimes an issue for very cheap devices)

29 September 2020

25

- Each machine language instruction for the processor is stored in memory as an opcode
  - A certain number of bits ,fixed in value, determine the actual instruction to execute 
  - The remaining variable bits (called the operand(s)) qualify the instruction, e.g. by specifying the registers to use
- The size of the opcode determines...
  - The number of unique instructions
  - The number of unique registers which can be specified
- E.g. the PIC 16 has 14 bit opcodes, of which 
  - 6 bits determine the instruction – so how many instructions?
  - 1 bit determines the destination of the instruction result – how many destinations?
  - 7 bits determine the source register to use – how many unique registers?

29 September 2020

26

### □ Relevance to embedded system designer

- The size of the opcode determines the number of opcodes and number of data sources/destinations
- A C compiler hides these details from you, but they can have an effect on how quickly your programme exceeds the limits of the device.
  - E.g. I wrote a relatively simple programme for the PIC16xxx processor which exceeded available resources just because it incorporated some string display features based on the C printf function.
  - This may limit what you can do with a particular device
- An opcode that supports few instructions (e.g. PIC16xxx supports just 35) also implies that more instructions will be required for basic tasks.
  - E.g. the PIC has no hardware multiply, so multiplication take much longer than addition/subtraction. Division and Modulo take longer again.

29 September 2020

27

## For information: PIC Instruction Set Encoding

Mnemonic, Operands	Description	Cycles	14-Bit Opcode				Status Affected	Notes	
			MSb		LSb				
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	1fff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECf	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
DECFSZ	f, d	Decrement f, Skip if 0	1(2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
INCFSZ	f, d	Increment f, Skip if 0	1(2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	1fff	ffff		
NOP	-	No Operation	1	00	0000	0xxx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2

29 September 2020

28

## Instruction processing and pipelining

29

- A processor processes an instruction in several stages, e.g.
  - Fetch – from memory
  - Decode – figure out what to do
  - Execute – do it
- In a simple processor, these stages take place serially (i.e. one after another)
- A pipelined processor design recognises that many instruction processing stages are independent and can be performed in parallel
- An N stage pipeline can run N stages in parallel
  - A single instruction still takes N cycles to execute from start to finish
  - But the throughput may approach one instruction per cycle
  - How is this possible...?

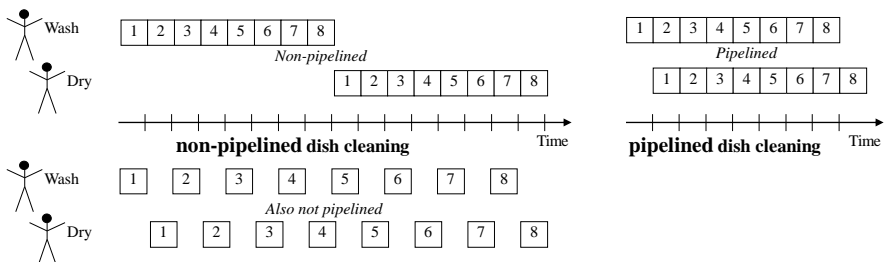
29 September 2020

29

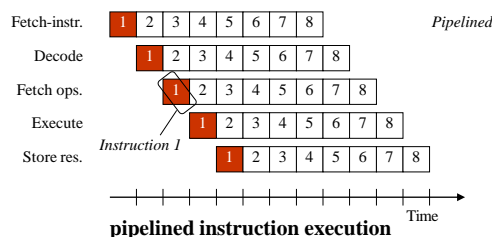
## Instruction Pipelining

30

### Pipelineing analogy



### Pipelineing in a processor



29 September 2020

30

## Instruction cycles vs. Clock Cycles

31

### □ Clock cycle

- Means clock period (i.e.  $1/\text{clock\_freq}$ )
- The minimum period (and hence maximum frequency) is limited by the **critical path** through the processor, (usually) the path from register through ALU back to register with longest delay

### □ Instruction cycle

- Base unit of time at which processor processes instructions
- Usually a multiple of the clock cycle
  - e.g. PIC16 instruction cycle = 4 clock cycles, other processors might have a different factor
  - Various transfers within processor take place on rising/falling edges of intermediate clock cycles within instruction cycle

### □ Simple instructions execute in 1 instruction cycle

### □ Complex instructions may take several

- In the PIC16 branch instructions might take 2 instruction cycles, but everything else is 1

Q. What are the practical consequences of all this?

29 September 2020

31

## Example question

32

The PIC16F877A processor is an 8 bit processor, with 14 bit addresses, 9 bit data registers, 13 bit instruction addresses, and no pipelining.

The ARM family are 32 bit processors with 32 bit addresses and a 3 or 5 stage pipeline.

For each processor...

Q1. Briefly explain each of these properties

Q2. What is the practical consequence of each of these properties

29 September 2020

32



Q. What can we recognise based on what we've spoken about in this lecture



29 September 2020