CS466 Chapter 3

Microprocessors & Microcontrollers

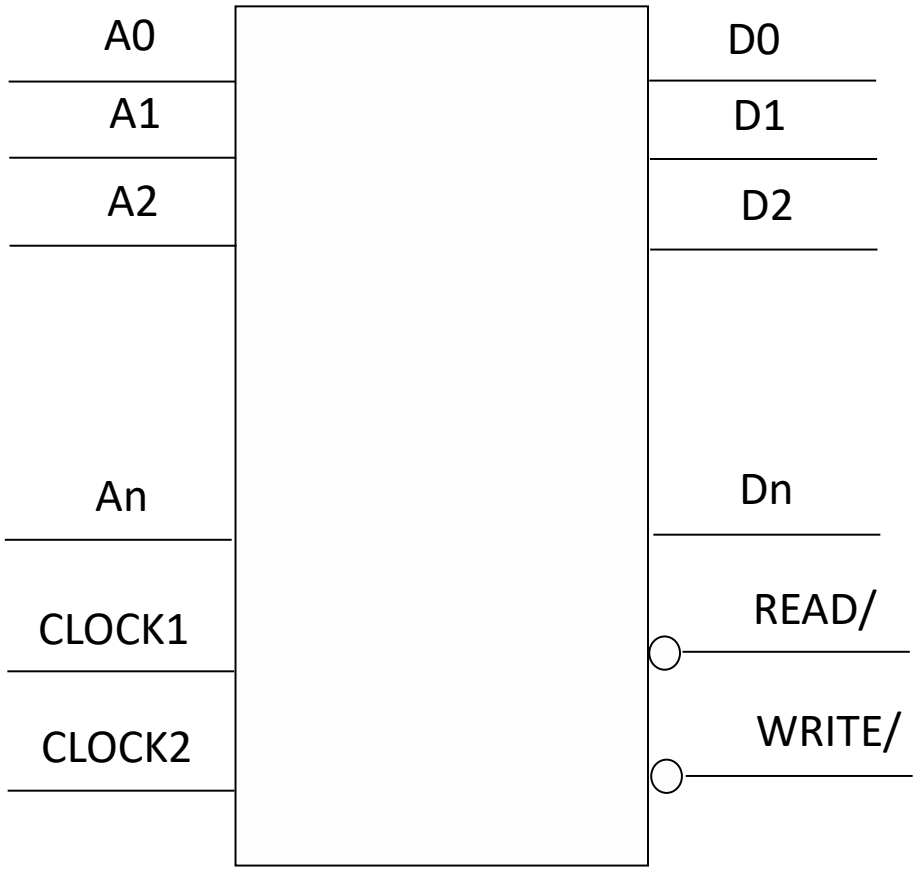
3.1 – Classic Microprocessors

- Microprocessors come in all kinds of varieties from the very simple to the very complex (atTiny8, ARM-A17)
- In this class we use a modern variant if the classic microprocessor. Modern devices are better described as MicroControllers or System on Chip (SoC) devices .

3.1 – Classic Microprocessors (cont)

- For instructional purposed we will discuss one so simple that no one is made that is quite this simple. However, it shares characteristics with every other microprocessor including:
  - A collection of address signals – the address bus
  - A collection of data signals – the data bus
  - A READ/ line, which it strobes low to get data, and a WRITE line, which it strobes low to write data out.
  - A clock signal input, which paces all of the work that the microprocessor does, and, as a consequence, paces the work in the rest of the system.
- In our TI processor (TM4C123GH6PM) all of these signals are buried in the device and generally inaccessible.

A very basic microprocessor



ARM Core

- The ARM Cortex-M4F is a processing core in our microcontroller.
- ARM does not actually sell any devices, they make all their income by selling the ARM core in software form to device manufactures.
- Nearly all chip manufacturers use ARM cores in some of their product lines. It is far above the most common microcontroller in use today.
- The ARM core is a 32 bit microprocessor, with 4GB of address space, The 32 bits means that the data bus is 32 bits wide.
- ARM Cores vary from the small Cortex-M0 for small bare metal or RTOS controlled devices all the way up to the Cortex-A78
- Many vendors load more than a single core into a single SOC to handle separate concurrent functions

Buses

- We will show on the next slide a very simple microprocessor system with some ROM and some RAM with the following:
  - All three parts have eight data signals D0-D7
  - The processor can address 64k of memory and therefore has 16 address lines, A0-A15.
  - The ROM and RAM each have 32k and thus have 15 address lines each, A0 – A14

A basic microprocessor system

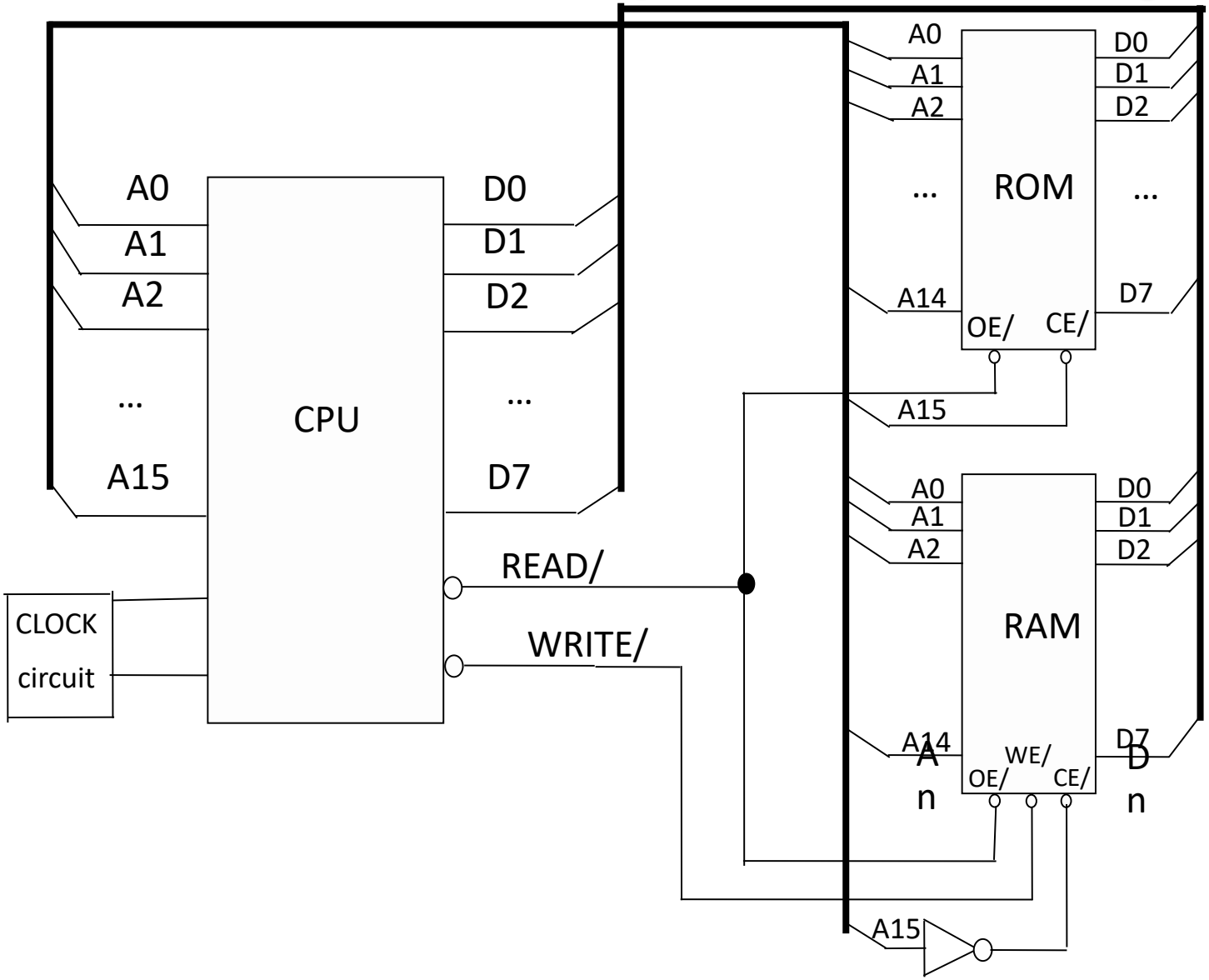


Figure 2.1. Cortex-M4 block diagram

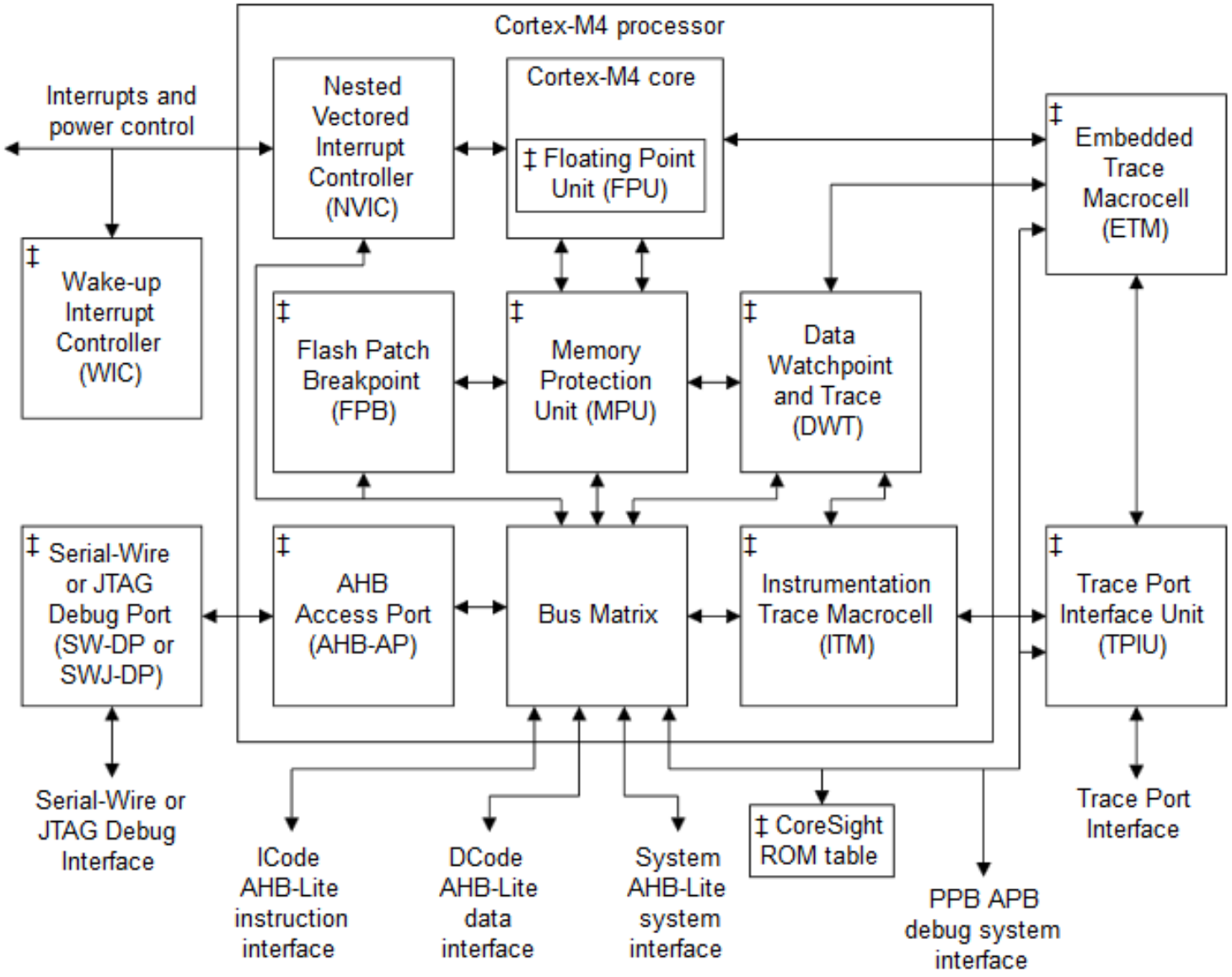


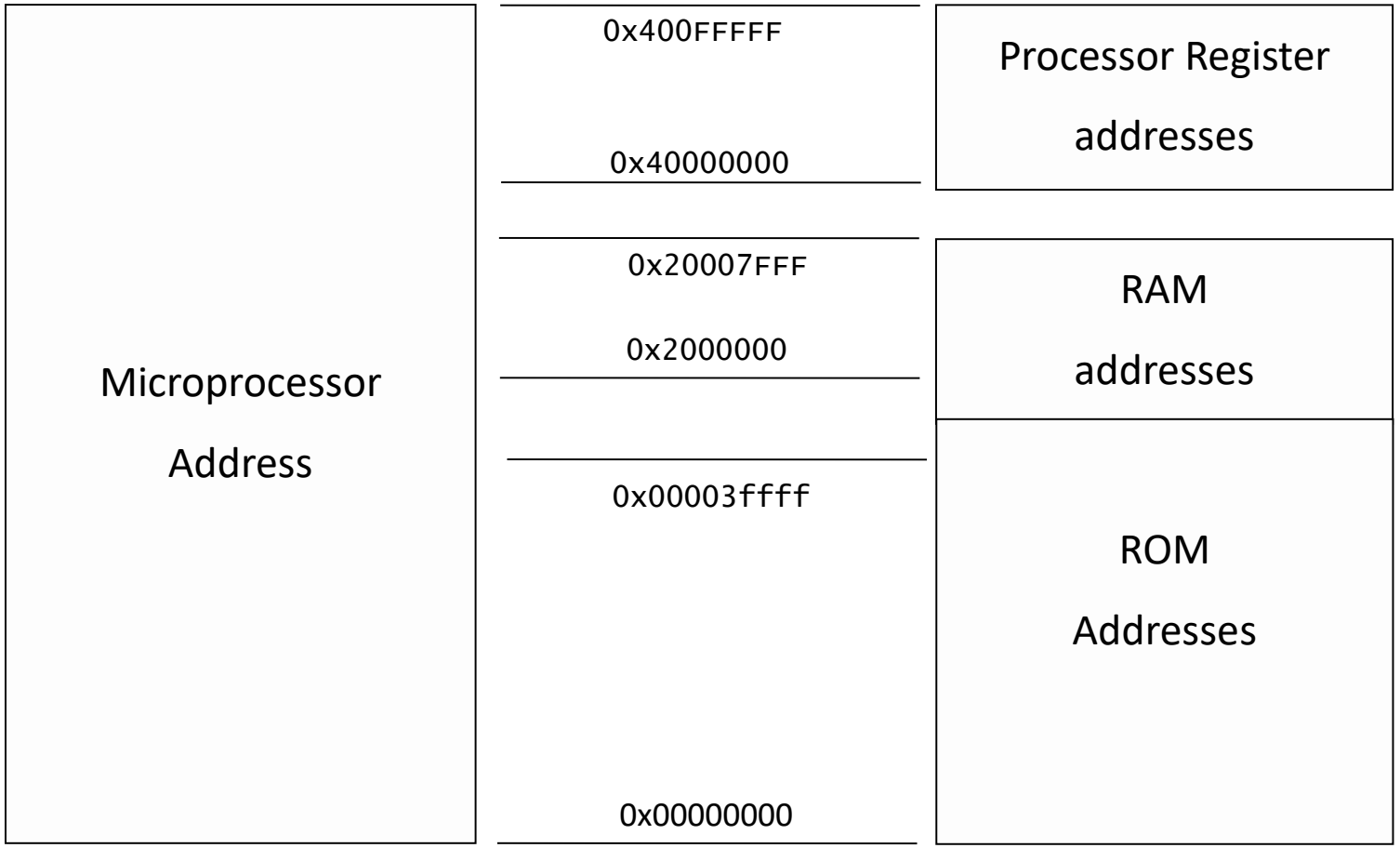
Table 3.1 TI TM4C123G Memory Map

Start	End	Size	Description Size
0x00000000	0x0003FFFF	0x00040000	On Chip Flash (ROM)
0x20000000	0x20007FFF	0x00007FFF	Bit-Banded In-Chip SRAM (RAM)
0x22000000	0x220FFFFF	0x00007FFF	Bit-Banded SRAM Alias
0x40000000	0x400FFFFF	0x00100000	Peripheral control Registers

For a more detailed memory map see the 2.4 Memory Model section of the processor data sheet

Another Representation of Memory Map

Typically only interesting bits are shown



Additional devices on the bus

- Virtually all embedded systems have the three aforementioned devices, unless the ROM and RAM is in turn embedded inside the microprocessor chip itself.
- Other hardware may also be connected to the address and data buses…
  - A/D and D/A converters
  - UARTS…etc

TM4C123GH6PM devices on the bus

0x4000.6000	0x4000.6FFF	GPIO Port C
0x4000.7000	0x4000.7FFF	GPIO Port D
0x4000.8000	0x4000.8FFF	SSI0
0x4000.9000	0x4000.9FFF	SSI1
0x4000.A000	0x4000.AFFF	SSI2
0x4000.B000	0x4000.BFFF	SSI3
0x4000.C000	0x4000.CFFF	UART0
0x4000.D000	0x4000.DFFF	UART1
0x4000.E000	0x4000.EFFF	UART2
0x4000.F000	0x4000.FFFF	UART3
0x4001.0000	0x4001.0FFF	UART4

…there are a few pages of peripheral addresses in the data-sheet

Additional devices (cont)

- A common way to handle the additional devices is to assign each of them a (contiguous) block of memory that is not used by the memory chips.
- This scheme is known as **memory mapping**. The size that is needed depends upon what that chip's needs are. The external device appears as just memory to the microprocessor and it is written to or read from by using standard C code memory access as shown on the next slide.

Additional devices (memory mapped)

```
#define GPIO_PORT_F_ADDR     ( (uint32_t *) 0x400253FC)
#define GPIO_PORTF_DATA_R    (*((volatile uint32_t *)0x400253FC))

void vFunction()
{
    uint32_t fData;
    uint32_t *p_byHardware;

    /* set up a pointer to the network chip */
    p_byHardware = GPIO_PORT_F_ADDR;

   //* read the status from the network chip
    fData = *p_byHardware;

  // or using the direct define as our lab headers use.
  fDAta = GPIO_PORTF_DATA_R,
}
```

Bus Handshaking

- In addition to the logic problems of hooking up the address and data busses correctly, we have the issue of **timing**.
- The ROM and RAM chips have certain timing requirements which must be met. The address lines must be stable for a certain specified period of time and the read enable and chip enable lines must be stable for a certain period of time before the device can place its data on the bus.
- The microprocessor is in control of all of these signals.
- The entire sequence is called a **bus cycle**.
- There are various schemes by which this timing can be accomplished and it is generally referred to collectively as **handshaking**.

No Handshake

- If there is no bus handshaking, the micro-processor just drives the bus at whatever speed suits it.
- In this scenario, the hw engineer must supply parts that can keep up, or buy a slower micro-processor (or run with a slower clock).
- As you might expect, the cost of memory is inversely proportional to its speed.

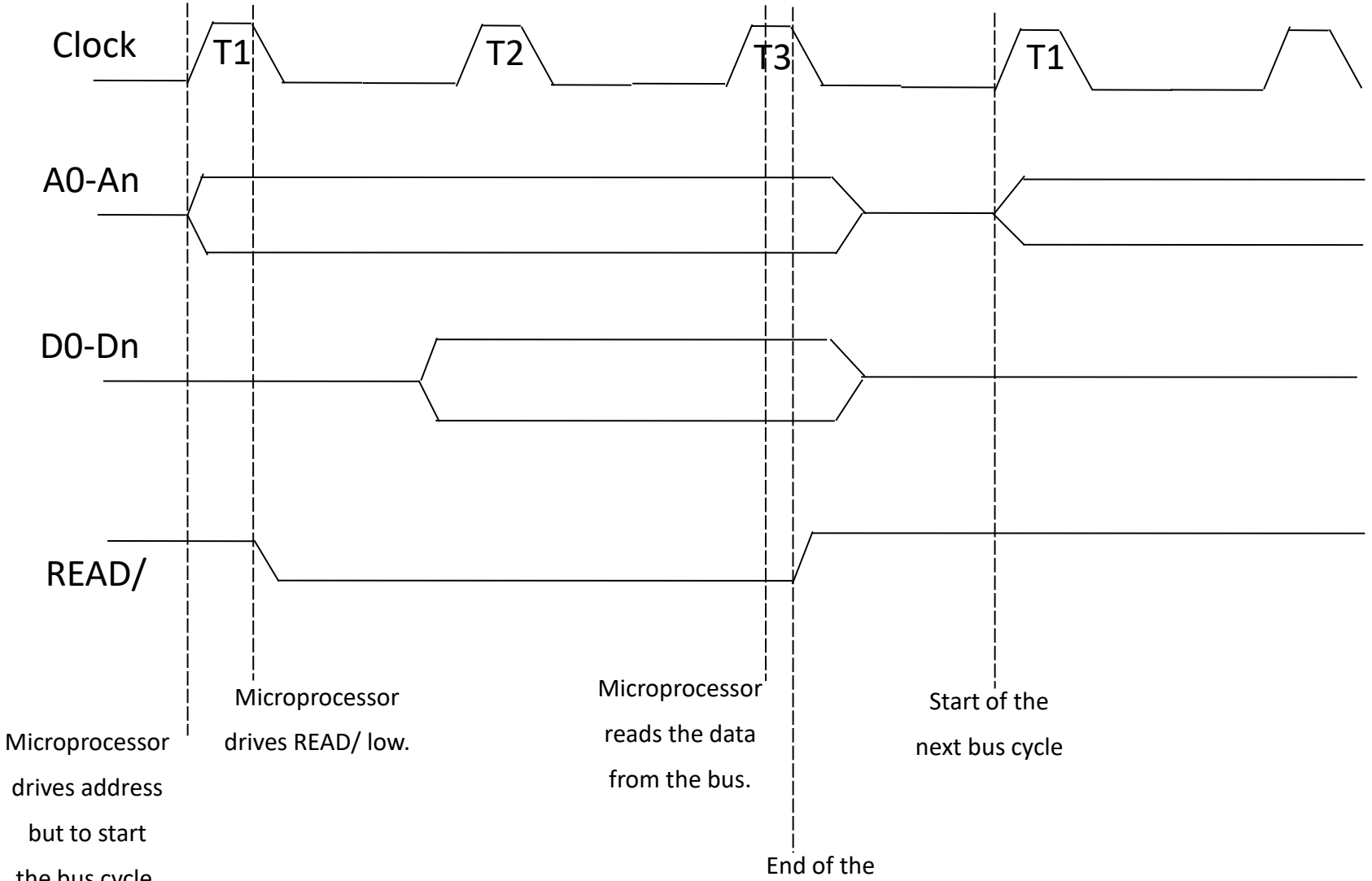
Wait Signals

- Some microprocessors (e.g. Motorola 68000) offer another alternative. They have a WAIT input which memory can use to extend the signals offered up to memory.
- This is a good technique, but has the disadvantage that some engineer has to design the WAIT state logic usually because ROMS and RAMS don't come with a wait signal.

Wait States

- Some micro-processors offer a third alternative for dealing with slower memory devices called **wait states**.
- We will look at a couple of timing diagrams which represent memory accesses with and without wait states inserted.

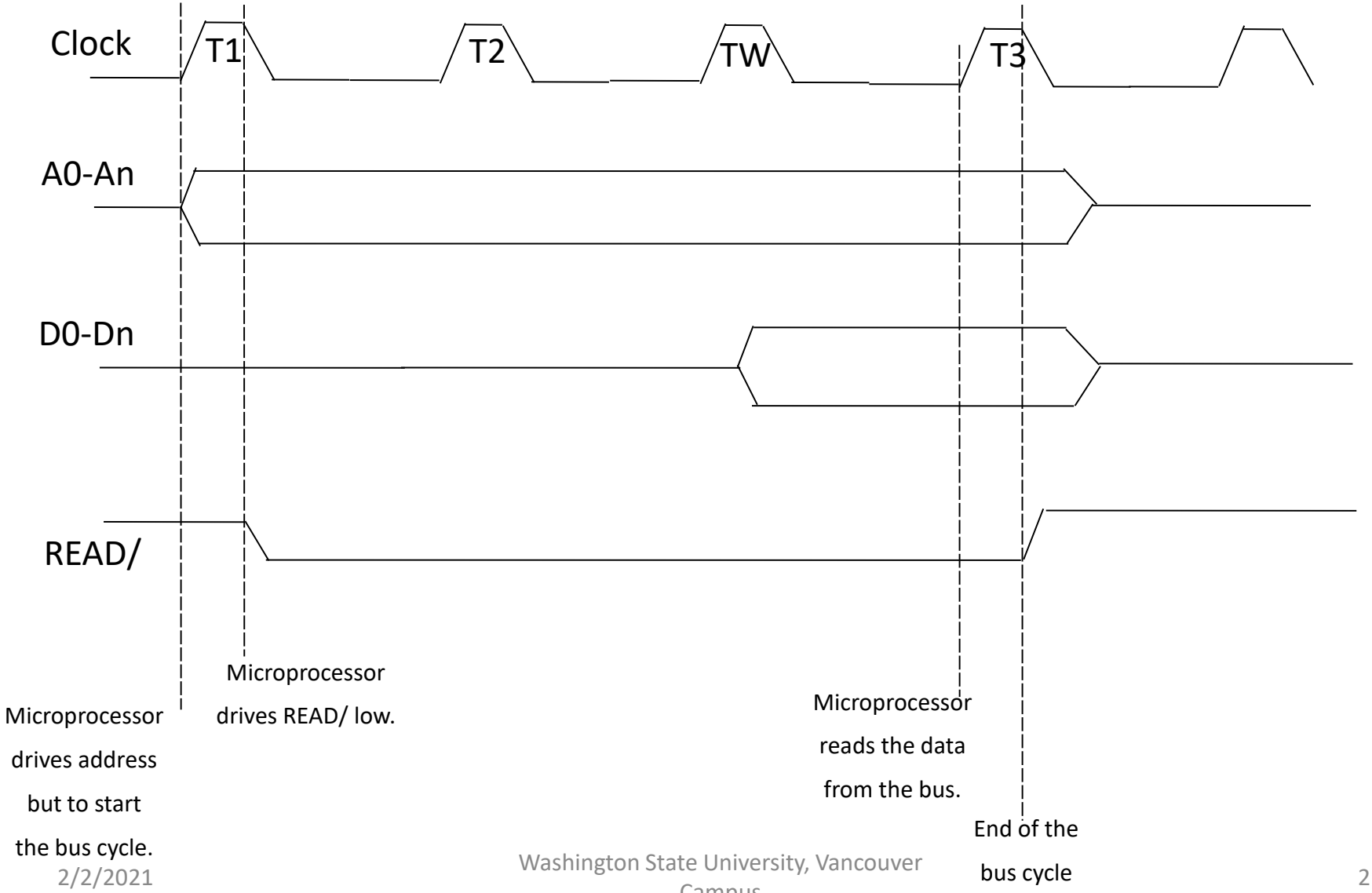
Wait States (Figure 3.6)



Wait States(cont)

- The microprocessor outputs the address on the rising edge of T1
- It asserts the READ/ line at the falling edge of T1
- It expects the data to be valid and actually takes the data in just a little after the rising edge of T3
- It de-asserts the READ/ line at the falling edge of T3 and shortly thereafter stops driving the address signals, thereby completing the bus transaction.

Add a Wait State (Figure 3.7)



Direct Memory Access

- Many processors also have provision to move multiple bytes to and from memory quickly without the processor having to move each byte individually.
- The processor still has to set the transaction and kick it off, and then the data is moved in parallel with instruction execution.
- Since there is only one set of address/data signals, designers must make sure that the CPU is not trying to drive these busses at the same time as the DMA circuitry.

Direct Memory Access(cont)

- The usual scheme is for the DMA signal to assert a bus request and the CPU honors it by giving up the bus(tri-stating it) and asserting this by way of a bus acknowledge signal.
- The transfer is then done and then the DMA circuitry gives up the bus by tri-stating it and releasing bus request.

Interrupts

- Even simple processors have the idea of an interrupt. In effect, it is told to stop whatever it is doing and respond by calling the interrupt service routine (isr).
- An easy way to think of an interrupt is an **asynchronous** call to an isr. It is the asynchronous nature of this signal that allows for quick response but also causes problems in that it must be handled carefully!

Watchdog Timers

- Watchdog timers are very common on embedded systems.
- A watchdog timer contains a timer that will expire after a certain amount of time and will cause an output pulse of some sort if it ever expires (times out).
- This output pulse is typically connected to the RESET input which means that it is quite catastrophic to the system but that it does allow the system to restart if the software goes “in the weeds”.
- The timer has some mechanism that allows software to restart it whenever it wishes. The idea is that the timer should never expire.
- These are many times required for safety reasons!

Micro-processor built-ins…

- Timers
- DMA
- I/O pins
- Address Decoding
- D/A and A/D circuits
- Comparators
- I2C, SPI, Quadrature Encoder, …and More
- Motor driver circuits
- Memory caches and instruction pipelines

Hardware Considerations

- Unlike software for which the engineering cost is virtually all of the cost, every copy of hardware costs money. Therefore, functionality is cheaper if it can be done in software.
- Every additional part takes up space either in a chip or on a PC board.
- Every additional part uses more power
- Every additional part turns power into heat
- Faster circuit components cost more, use more power, and generate more heat.

Interrupts

- The response problem in imbedded systems is the difficult one of providing a fast response to certain external events, even if in the middle of doing something else.
- If the underground tank monitoring system is busy calculating how much gasoline is in tank #6, it must still respond rapidly if someone presses a button requesting to know how much gasoline is in tank #2.

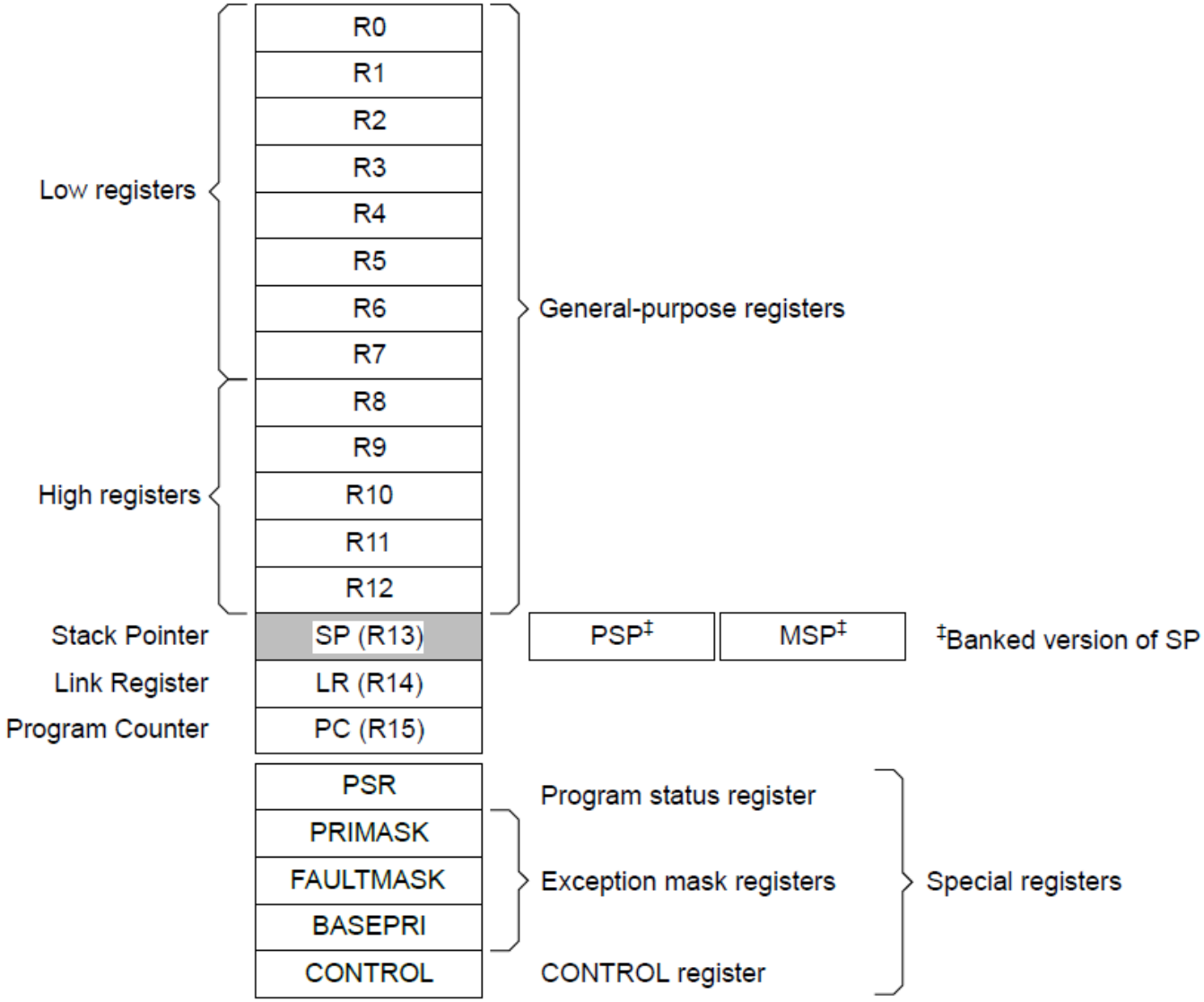
Microprocessor Architecture

- Assembly language is the human-readable form of the instructions that the microprocessor knows how to do. A program called an **assembler** translates the assembly language into essentially binary numbers before the microprocessor can execute them. Generally, one assembler instruction turns into just one machine instruction.
- Every family of microprocessors has its own assembler language…because each family has its own instructions. Assembler is NOT ‘as’ portable…C is.
- C as a language is still pretty portable but code can be tied to hardware or designed to be portable as well.

Microprocessor Architecture

- Usually microprocessors have what is called the programmers model, which is a set of general purpose registers. These may be named R1,R2, etc or in the case of the Rabbit, A, BC, DE, etc, similar to the Z180.
- Each microprocessor also has a PC or program counter which points to the address of the next instruction to be fetched.
- Most also have a stack pointer, which points to the address of the top of the stack.

ARM Cortex M4 Registers



Visit the ARM Info Center

- http://infocenter.arm.com/help/index.jsp
- Just go to the docs and peruse interesting documentation!
- You could spend a Career reading through all the information, don't start today but browse around.. It's a wealth of stuff.