Daniel Benjamin

HW3 - COMP 3350 02/03/2021

1. Explain two ways of generating a clock for a CPU, explain which is preferable and state reasons why.

· Clock signals can either be generated using a Quartz crystal or by a resonating circuit with a resistance, capacitance, and inductance RLC Circuit.

· However, since resistance, capacitance, and inductance are temperature dependent, their values change with temperature. Also, impurities in the manufactoring process can contribute to inaccurate clock frequency readings. Correctors are used to correct inaccurrate clock frequencies.

Quartz crystal:

Quartz crystals provide high-accurracy frequency readings. A Quartz crystal is cut into the shape of a cube using a diamond. A Quartz crystal oscillates based on its natural frequency of vibration which is a function of one side of a cube. Quartz crystals are made out of sand and are highly accessible and cheap. For all these reasons, Quartz crystals are prefered over resonating circuits when clock signals are to be generated.

2. Discuss a sychronous memory read cycle:

Info on buses:

Multiple machine cycles are needed when reading from memory, because it responds much more slowly than the CPU. There are three types of buses such as the address bus, control bus, and data bus. The address bus is unidirectional from the processor to the memory. The control bus is bidirectional from the memory controller to the processor. The data bus is bidirectional from the processor to memory.

Reading from Memory (How memory is read):

First, an address is placed on the address bus. Then with the Read Line (RD) set low the CPU now waits one clock cycle for the memory to respond. When the memory responds and the Read Line (RD) is set to 1, the data is now on the data bus. The cycle where the CPU is waiting for the memory to respond after an address is placed on the address bus is called the wait cycle.

3.

1. An uninitialized data declaration for a 16-bit signed and unsigned integer.

```
16-bit signed:
var1 SWORD ?

16-bit unsigned:
var1 WORD ?
```

2. An uninitialized data declaration each for an 64-bit signed and unsigned integer

```
64-bit signed:

var1 SQWORD ?

64-bit unsigned:

var1 QWORD ?
```

3. An initialized data declaration for a 16-bit unsigned integer with the value 1298h

```
var1 WORD 1298h
```

4. A null-terminated string variable with the value "Computer Organization"

```
str1 BYTE "Computer Organization", 0
```

3. 5. A symbolic constant named "Area of a circle" using the equal sign directive and assign it an arithmetic expression that calculates the circumference in terms of Pi and diameter, D of the circle.

Area Of a circle = 0.25 * (3.1415927)*(D * D)

4. Show the order of individual bytes in memory, lowest to highest, for the following variables using little endian order:

Rose WORD $\overbrace{67}^{B_1}\overbrace{9A}^{B_0}$

Magnolia DWORD $\underbrace{12}_{B_3}\underbrace{9B}_{B_2}\underbrace{C7}_{B_1}\underbrace{4E}_{B_0}$h

Rose	9A
Rose +1	67
Magnolia = Rose +2	4E
Magnolia+1	C7
Magnolia+2	9B
Magnolia+3	12

5. Use assembler directives to declare a signed DWORD array of five elements and initialize it with the following values: 5, 25, -125, 250, -500.

dwList SDWORD 5, 25, -125, 250, -500

Show how to calculate the number of elements in this array and assign that value to a symbolic constant named "Number Of Elements". To calculat the number of elements in a SDWORD array, find the the total number of bytes and divide the result by 4, which ~~#s~~ is the size of one SDWORD.

5. dwList SDWORD 5, 25, -125, 250, -500

4 bytes 4 bytes

Each SDWORD is 4 bytes where there are five of them in dwList:

TotalBytesIndwList = 4 + 4 + 4 + 4 + 4 = 20 B

Number of Elements = ($ - dwList) / 4 = 20/4 = 5

current address within the data segment at thes beginning of this line

size of each DWORD

address of dwList

total number of elements

* the difference ($ - dwList) will result in the total number of bytes

6.

```
.data
val1 SWORD F448h
val2 SWORD 07D0h
val3 SWORD 03E8h
finalVal SWORD ?
.code
mov eax, val1
add eax, val2
sub eax, val3
mov finalVal, eax
```