Noah Jones

COMP-3350

Dr. Baskiyar

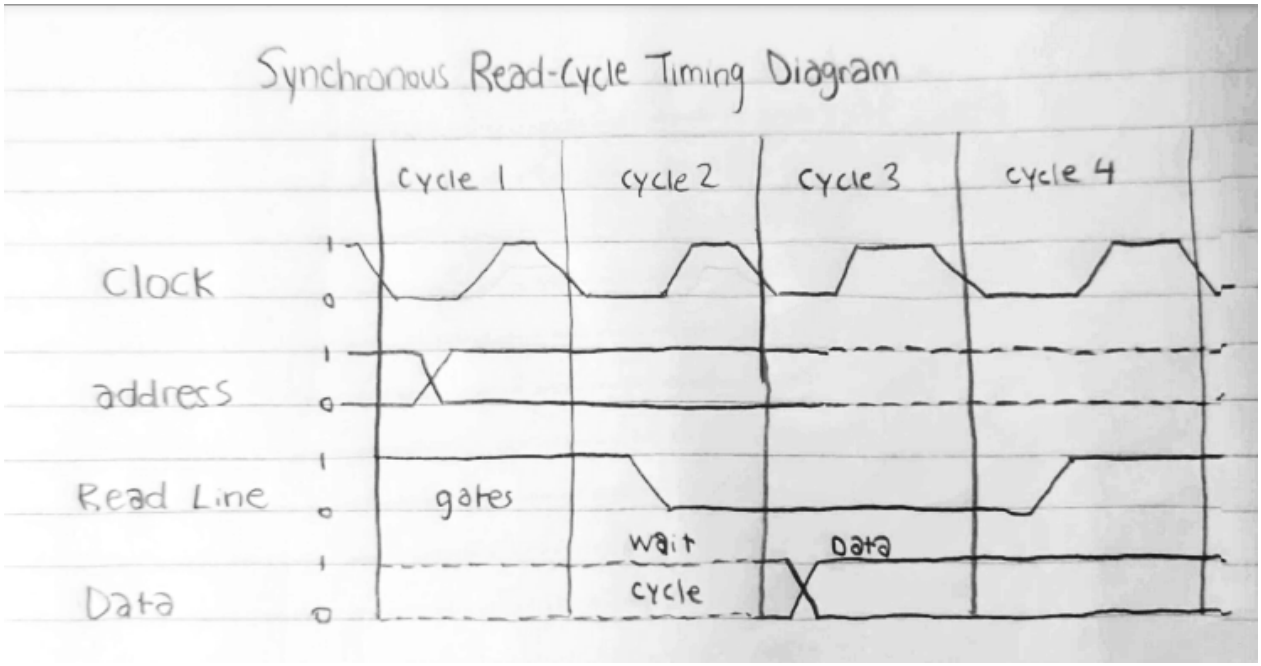
Homework #3

1. **Explain two ways of generating a clock for a CPU, as discussed in the class and explain which is preferable and state reasons.**

1. Quartz Crystal Oscillator:
    - The quartz crystal oscillator provides a very precise and reliable frequency. It enables the CPU to reach higher speeds, especially the CPUs with overclocking capabilities. The downside to using a quartz crystal oscillator is that they are larger and more expensive than the alternative.
2. RLC Circuit
    - The RLC Circuit consists of a resistor(R) , an inductor (L), and a capacitor (C). The circuit can be used to create an oscillation and function as a clock with low-resistance. The downside to this is reduced accuracy in comparison to a quartz crystal oscillator

.

2. **Draw a synchronous memory read cycle timing diagram and explain it with a few sentences.**



This timing diagram represents the sequence and timing of 4 signals when reading from memory. The clock is always at the top of the timing diagram, it stays at a steady rate between 1 and 0 to keep the timing of the operations. The address signal specifies the memory location in which we are reading data from. The memory chip interprets the data's address, and prepares the data to be read. The address line turns off during cycle 3 when the data is received. The read line sets to 0 during cycle 2 to notify memory that a value is about to be read. The read line then sets to 1 in cycle 4 when the data is on the bus, signaling the CPU to read it. The data line is inactive while it is waiting for the data to be put on the bus, which happens during cycle 3.

3. **Declare the following:**

1. **A data variable each for a 16-bit signed and an unsigned integer with initial values of 0xC4F2 and 0x1DA7**

- Signed 0xC4F2

  .data

  myVar SWORD 0C4F2h
- Unsigned 0xC4F2

  .data

  myVar WORD 0C4F2h
- Signed 0x1DA7

  .data

  myVar SWORD 01DA7h
- Unsigned 0x1DA7

  .data

  myVar WORD 01DA7h

2. **A data variable each for a 32-bit signed and an unsigned integer with no initial values.**

- Signed

  .data

  val1 SDWORD ?
- Unsigned

  .data

  val1 DWORD ?

3. **A null-terminated string variable with the value "Quantum Computing"**

- str1 BYTE "Quantum Computing",0

4. **A symbolic constant named "*Perimeter of a circle*" using the equal-sign directive and assign it an arithmetic expression that calculates the area in terms of Pi and radius, R of the circle.**

- Pi EQU <3.1416>

  R EQU <5>

  .data

  Perimeter_of_a_circle EQU $((Pi*R)^2)$

4. **Show the order of individual bytes in memory *(lowest to highest)* for the following variables *(use little endian order):***

Summer SWORD 0812Bh (16 Bits/ 2 Bytes)

b_0 = 2B
b_1 = 81
b_2 = 00

Spring DWORD 394BE451h (32 Bits / 4 Bytes)

b_0 = 51
b_1 = E4
b_2 = 4B
b_3 = 39

5. **Use assembler directives to declare a signed DWORD array of five elements and initialize it with the following values: C11AF, 1, A689, 4115, D31B. Show how to calculate the number of elements in this array and assign that value to a symbolic constant named "*NumberOfElements*"**

```
.data
  myArray DWORD C11AFh, 1, 1689h, 4115h, D31Bh          // (not sure if '1'
.data                                                    is supposed to be dec or hex?)
  NumberOfElements =  ($ - myArray) / 4;calculate num of elements and assign to constant
```

6. Using the *AddTwo.asm* program from the textbook as a reference, write a program *Add2Sub.asm* that adds two unsigned word sized integers, and then subtracts one word sized integer. Hand write the code. You do not need to assemble/execute at this time.

```
TITLE Add2Sub           (Add2Sub.asm)
INCLUDE Irvine32.inc
main PROC
.data
  num1 SWORD 1240h   ; First operand
  num2 SWORD 1361h   ; Second operand
  num3 SWORD 9876h   ; Third operand for subtracting
.code
  mov ax, [num1]          ; load num 1 into ax register
  add ax, [num2]          ; add num 2 to ax register
  sub ax, [num3]          ; subtract num 3 from ax
  mov bx, ax              ; store the result from ax to bx
  call DumpRegs           ; display registers
exit
main ENDP
END main
```