**BYOC course**

**Homework exercise #3 – Disassembly**

**346172604**

**207201211**

**316473958**

**205713969**

Here is the MIPS binary code you need to dis-assembly:

x"00400000" => x"20010001": addi $1,$0,1

x"00400004" => x"20020002": addi $2,$0,2

x"00400008" => x"20030003": addi $3,$0,3

x"0040000C" => x"20040004": addi $4,$0,4

x"00400010" => x"20050005": addi $5,$0,5

x"00400014" => x"20060006": addi $6,$0,6

x"00400018" => x"20070007": addi $7,$0,7

x"0040001C" => x"20080008": addi $8,$0,8

x"00400020" => x"20090009": addi $9,$0,9

x"00400024" => x"200A000A": addi $10,$0,10

x"00400028" => x"200B000B": addi $11,$0,11

x"0040002C" => x"200C000C": addi $12,$0,12

x"00400030" => x"200D000D": addi $13,$0,13

x"00400034" => x"200E000E": addi $14,$0,14

x"00400038" => x"200F000F": addi $15,$0,15

x"0040003C" => x"20100010": addi $16,$0,16

x"00400040" => x"20110001": addi $17,$0,1

x"00400044" => x"20120002": addi $18,$0,2

x"00400048" => x"20130004": addi $19,$0,4

x"0040004C" => x"20140008": addi $20,$0,8

x"00400050" => x"20150010": addi $21,$0,16

x"00400054" => x"20160020": addi $22,$0,32

x"00400058" => x"20170040": addi $23,$0,64

x"0040005C" => x"20180080": addi $24,$0,128

x"00400060" => x"20190100": addi $25,$0,256

x"00400064" => x"201B0200": addi $27,$0,512

x"00400068" => x"201C0400": addi $28,$0,1024

x"0040006C" => x"201D0800": addi $29,$0,2048

x"00400070" => x"201E1000": addi $30,$0,4096

x"00400074" => x"201F2000": addi $31,$0,8192

x"00400078" => x"02018020": add $16,$16,$1

x"0040007C" => x"02028020": add $16,$16,$2

x"00400080" => x"02038020": add $16,$16,$3

x"00400084" => x"02048020": add $16,$16,$4

x"00400088" => x"00000000": sll $0,$0,0x0 (nop)

x"0040008C" => x"00000000": sll $0,$0,0x0 (nop)

x"00400090" => x"00000000": sll $0,$0,0x0 (nop)

x"00400094" => x"00220820": add $1,$1,$2

x"00400098" => x"00000000": sll $0,$0,0x0 (nop)

x"0040009C" => x"00000000": sll $0,$0,0x0 (nop)

x"004000A0" => x"00230820": add $1,$1,$3

x"004000A4" => x"00000000": sll $0,$0,0x0 (nop)

x"004000A8" => x"00000000": sll $0,$0,0x0 (nop)

x"004000AC" => x"00810820": add $1,$4,$1

x"004000B0" => x"00000000": sll $0,$0,0x0 (nop)

x"004000B4" => x"00000000": sll $0,$0,0x0 (nop)

x"004000B8" => x"00A10820": add $1,$5,$1

x"004000BC" => x"00000000": sll $0,$0,0x0 (nop)

x"004000C0" => x"00000000": sll $0,$0,0x0 (nop)

x"004000C4" => x"00260820": add $1,$1,$6

x"004000C8" => x"00000000": sll $0,$0,0x0 (nop)

x"004000CC" => x"00000000": sll $0,$0,0x0 (nop)

x"004000D0" => x"00E83820": add $7,$7,$8

x"004000D4" => x"00000000": sll $0,$0,0x0 (nop)

x"004000D8" => x"00000000": sll $0,$0,0x0 (nop)

x"004000DC" => x"00E90820": add $1,$7,$9

x"004000E0" => x"00000000": sll $0,$0,0x0 (nop)

x"004000E4" => x"00000000": sll $0,$0,0x0 (nop)

x"004000E8" => x"002A0820": add $1,$1,$10

x"004000EC" => x"00000000": sll $0,$0,0x0 (nop)

x"004000F0" => x"00000000": sll $0,$0,0x0 (nop)

x"004000F4" => x"002B0820": add $1,$1,$11

x"004000F8" => x"00000000": sll $0,$0,0x0 (nop)

x"004000FC" => x"00000000": sll $0,$0,0x0 (nop)

x"00400100" => x"002C0820": add $1,$1,$12

x"00400104" => x"00000000": sll $0,$0,0x0 (nop)

x"00400108" => x"00000000": sll $0,$0,0x0 (nop)

x"0040010C" => x"002D0820": add $1,$1,$13

x"00400110" => x"00000000": sll $0,$0,0x0 (nop)

x"00400114" => x"00000000": sll $0,$0,0x0 (nop)

x"00400118" => x"002E0820": add $1,$1,$14

x"0040011C" => x"00000000": sll $0,$0,0x0 (nop)

x"00400120" => x"00000000": sll $0,$0,0x0 (nop)

x"00400124" => x"002F0820": add $1,$1,$15

x"00400128" => x"00000000": sll $0,$0,0x0 (nop)

x"0040012C" => x"00000000": sll $0,$0,0x0 (nop)

x"00400130" => x"00008020": add $16,$0,$0

x"00400134" => x"200F0003": addi $15,$0,3

x"00400138" => x"00000000": sll $0,$0,0x0 (nop)

x"0040013C" => x"00000000": sll $0,$0,0x0 (nop)

x"00400140" => x"00000000": sll $0,$0,0x0 (nop)

x"00400144" => x"22100001": addi $16,$16,1

x"00400148" => x"00000000": sll $0,$0,0x0 (nop)

x"0040014C" => x"21EFFFFF": addi $15,$15,-1

x"00400150” => x”11E00004": beq $15,$0,0x400164

x"00400154” => x”00000000": sll $0,$0,0x0 (nop)

x"00400158” => x”00000000": sll $0,$0,0x0 (nop)

x"0040015C” => x”08100051": j 0x400144

x"00400160” => x”00000000": sll $0,$0,0x0 (nop)

x"00400164” => x”08100059": j 0x400164

x"00400168” => x”00000000": sll $0,$0,0x0 (nop)

x"0040016C” => x”00000000": sll $0,$0,0x0 (nop)

See questions below:

1. What does the code in addresses 0x400000-0x40003C do?

The code initializes registers $1 through $16 with the values 1 through 16, respectively, by computing the sum of $0, which equals zero, and an immediate.

1. What does the code in addresses 0x400040-0x400074 do?

The code initializes registers $17 through $31 with values that are powers of two, ranging from 1 to 8192, just like the code in addresses 0x400000-0x40003C. Each register holds a number with one bit set. For example, register $17 has the first bit set, register $18 has the second bit set, and this pattern continues until register $31, which has the 14th bit set.

1. What does the code in addresses 0x400078-0x400090 do and what will be the contents of register $16 when we reach address 0x400094?

The code calculates the sum of the values in registers $1 (which is 1), $2 (which is 2), $3 (which is 3), and the initial value of $16 (which is 16), and stores the result in $16. The code is followed by a series of no-operation instructions. Thus, when reaching address 0x400094, the value of $16 will be 1 + 2 + 3 + 4 + 16 = 26.

1. What does the code in addresses 0x400144-0x400168 do and what will be the contents of register $16 at the end of this code section?

The code in addresses 0x400144 to 0x400168 implements a loop that increments the value in register $16 and decrements the value in register $15 until $15 reaches zero. Since $16 is initialized to zero at 0x00400130, and $15 is initialized to 3 at 0x00400134, the code will increment $16 three times. Once $15 reaches zero, the loop will enter an infinite loop at 0x400164. Thus, the final value of $16 will be 3.

1. Does this code tests the GPR\_file and ALU parts of a MIPS CPU? How? What is not covered?

The code tests the GPR (General Purpose Register) file by reading from and writing to it. It covers writing to all registers in the addresses 0x400000-0x400074. Additionally, it tests reading from both read ports simultaneously in the addresses 0x00400078-0x00400084. However, not all registers are tested for reading on both ports.

The code tests the ALU (Arithmetic Logic Unit) by performing arithmetic operations. It tests the add and beq (branch if equal) operations. Other operations like sub (subtract), and, or, xor, and slt (set on less than) are not covered.