DREXEL UNIVERSITY
Electrical and Computer Engineering
College of Engineering

Drexel University

Electrical and Computer Engineering Dept.

Computer Architecture , ECE-C355

TITLE: **Term Project Report**

Team Members: **Avik Bag, Kunal Malik**

Instructor: **Cem Sahin**

Design Principles

The MIPS instruction set illustrates four underlying principles of hardware design:

1. Simplicity favors regularity.
2. Smaller is faster.
3. Good design demands compromise.
4. Make the common case fast.

Most Instructions in MIPS has *exactly* three operands. MIPS has 32 32-bit registers, *$v0,...$v31*, a very large number would increase the clock cycle time as reflected by (2) above. The compromise represented by the MIPS design, was to make all the instructions the *same* length, thereby requiring different instruction formats(3).

The MIPS instruction set addresses this principal by making constants part of arithmetic instructions. Furthermore, by loading small constants into the upper 16-bits of a register(4).

Part A

Mips Code :

```
        lw   $s0, 0($t0)
        lw   $s1, 4($t0)
        beq  $s0, $s1, L
        add  $s3, $s4, $s5
        j    EXIT
L:      sub  $s3, $s4, $s5
EXIT:   sw   $s3, 8($t0)
```

Machine Code (Hex)

```
8d100000
8d110004
12110002
02959820
08000024
02959822
ad130008
```

Line 1: 0x00000000:0x8d100000 [lw $s0 0($t0) => I(op:35(lw) rs:8(t0) rt:16(s0) immed:0x00000000)]

Line 2: 0x00000004:0x8d110004 [lw $s1 4($t0) => I(op:35(lw) rs:8(t0) rt:17(s1) immed:0x00000004)]

Line 3: 0x00000008:0x12110002 [beq $s0 $s1 L => I(op:4(beq) rs:16(s0) rt:17(s1) immed:0x00000002)]

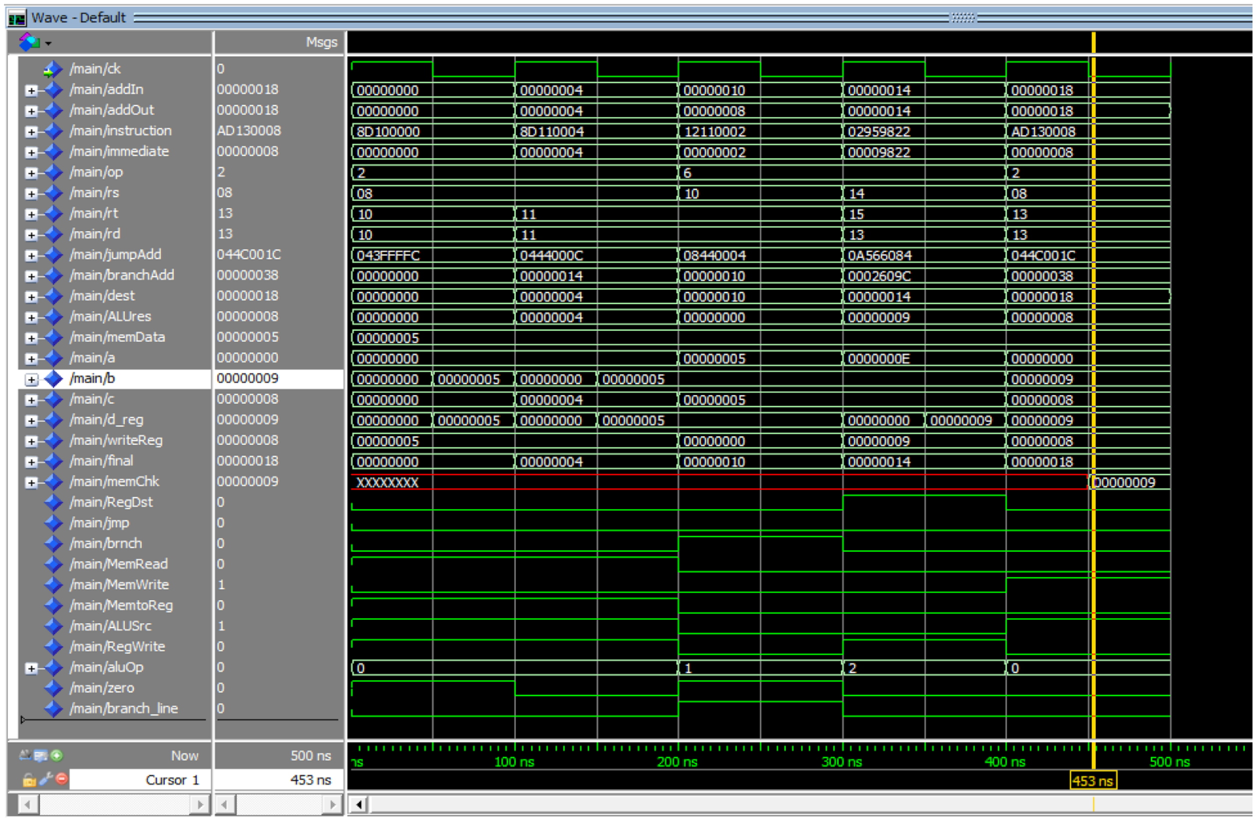
Line 4: 0x0000000c:0x02959820 [add $s3 $s4 $s5 => R(op:0(add) rs:20(s4) rt:21(s5) rd:19(s3) sh:0 func:32)]

Line 5: 0x00000010:0x08000024 [j EXIT => JFormat(op:2(j) target:0x00000018 >> 2 = 0x00000006)]

Line 6: 0x00000014:0x02959822 [L: sub $s3 $s4 $s5 => R(op:0(sub) rs:20(s4) rt:21(s5) rd:19(s3) sh:0 func:34)]

Line 7: 0x00000018:0xad130008 [EXIT: sw $s3 8($t0) => I(op:43(sw) rs:8(t0) rt:19(s3) immed:0x00000008)]

Final state of your data memory



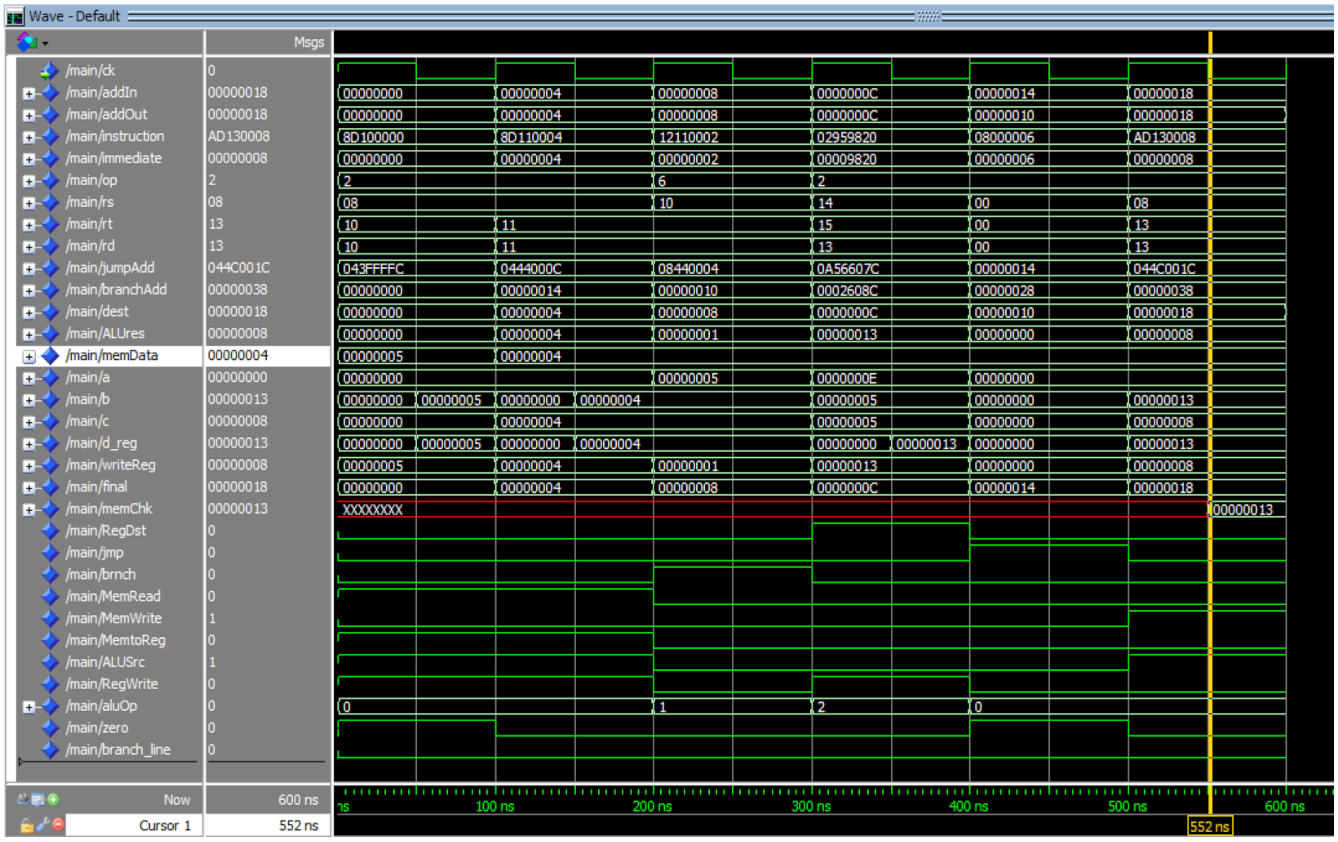
```
Wave - Default
                  Msgs
/main/ck          0
/main/addIn       00000018
/main/addOut      00000018
/main/instruction AD130008
/main/immediate   00000008
/main/op          2
/main/rs          08
/main/rt          13
/main/rd          13
/main/jumpAdd     044C001C
/main/branchAdd   00000038
/main/dest        00000018
/main/ALUres      00000008
/main/memData     00000005
/main/a           00000000
/main/b           00000009
/main/c           00000008
/main/d_reg       00000009
/main/writeReg    00000008
/main/final       00000018
/main/memChk      00000009
/main/RegDst      0
/main/jmp         0
/main/brnch       0
/main/MemRead     0
/main/MemWrite    1
/main/MemtoReg    0
/main/ALUSrc      1
/main/RegWrite    0
/main/aluOp       0
/main/zero        0
/main/branch_line 0

Now               500 ns
Cursor 1          453 ns
```

Challenges Faced

The implementation of the PC took a while to understand. Getting the PC to increment at every clock rising edge was where the problem occurred. So the way it was solved was by wiring the clock input and checking for every clock rising event.

Part B

Final state of your data memory



Wave - Default

	Msgs
/main/ck	0
/main/addIn	00000018
/main/addOut	00000018
/main/instruction	AD130008
/main/immediate	00000008
/main/op	2
/main/rs	08
/main/rt	13
/main/rd	13
/main/jumpAdd	044C001C
/main/branchAdd	00000038
/main/dest	00000018
/main/ALUres	00000008
/main/memData	00000004
/main/a	00000000
/main/b	00000013
/main/c	00000008
/main/d_reg	00000013
/main/writeReg	00000008
/main/final	00000018
/main/memChk	00000013
/main/RegDst	0
/main/jmp	0
/main/brnch	0
/main/MemRead	0
/main/MemWrite	1
/main/MemtoReg	0
/main/ALUSrc	1
/main/RegWrite	0
/main/aluOp	0
/main/zero	0
/main/branch_line	0

Now 600 ns

Cursor 1 552 ns

Challenges Faced

The next problem was with the branch statement. The thing about the branch statement is that it brings it to the exact line address in the instruction memory. Now before the start of any cycle, the PC component adds 4 to the incoming PC from the previous cycle. Therefore, to balance it out, the branch address after the entire branch operation is complete, we reduced the value by 4 to ensure that the right line number was reached. Same applied to the implementation of the jump component.

Part C

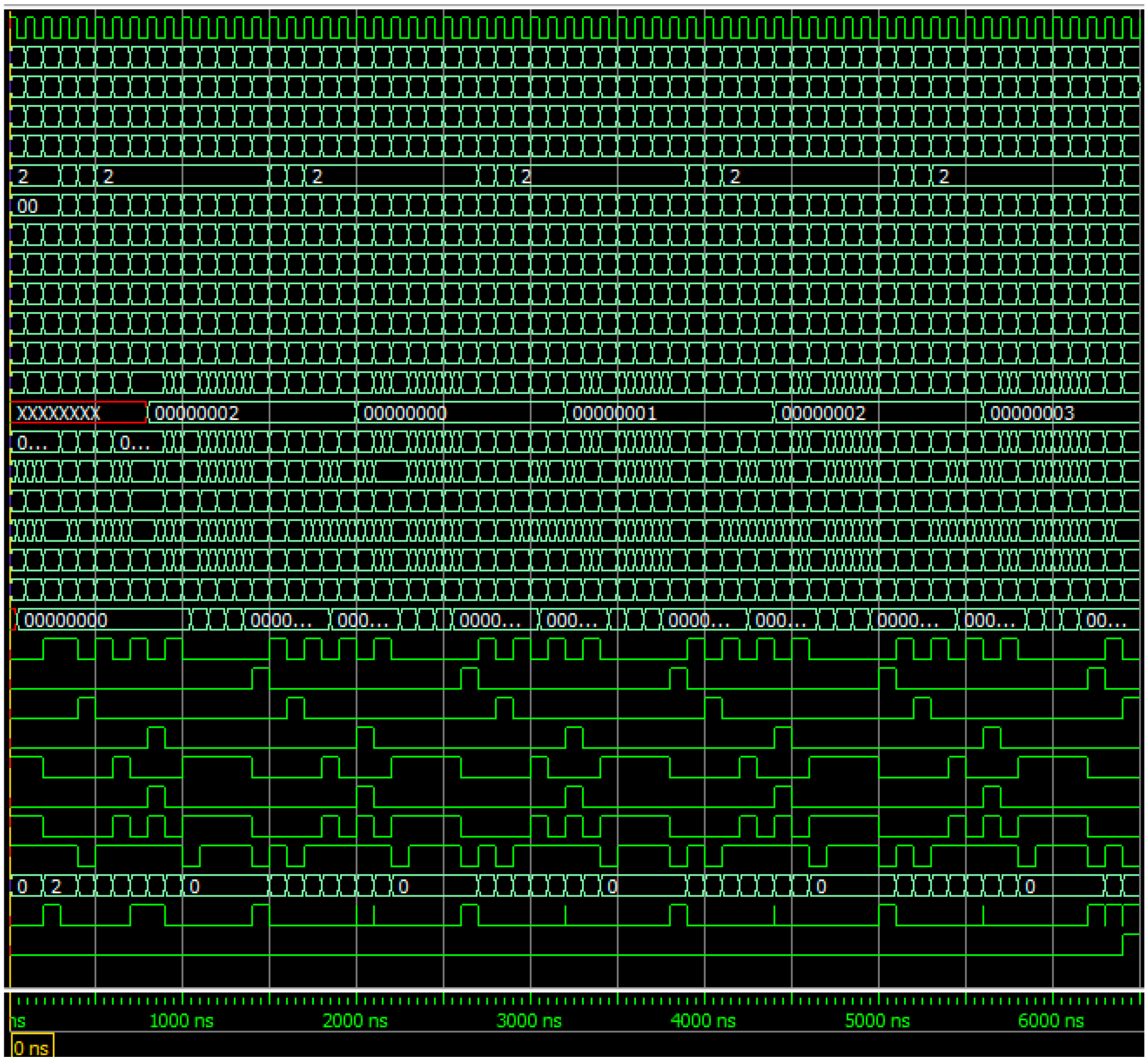
C code (Given)

```
x = 10;
y = 20;
i = 0;
while(y >= x){
      B[i] = A[i] + x + y;
      x = x - 1;
      y = y - 3;
      i = i + 1;
}
```

C to Mips

2008000a	| addi $t0, $zero, 10	| t0 = 10 = x
20090014	| addi $t1, $zero, 20	| t1 = 20 = y
00005020	| add $t2, $zero, $zero	| t2 = 0 = I
0128c02a	| slt $t8 $t1 $t0	| f y < x
1300000f	| beq $t8 $zero 15	| if true, exit
01285820	| add $t3, $t1, $t0	| x + y
214d0030	| addi $t5, $t2, 48	| t5 = 48+I
000a7020	| add $t6, $zero,$t2	| base address of A + I
8dcf0000	| lw $t7, 0($t6)	| load A
01eb7820	| add $t7, $t7, $t3	| A[i] + x + y
adaf0000	| sw $t7, 0($t5)	| B[i] = A[i] + x + y
2108ffff	| addi $t0, $t0, -1	| x = x - 1
2129fffd	| addi $t1, $t1, -3	| y = y-3
214a0001	| addi $t2, $t2, 1	| I = i+1
08000003	| j loop X	|
EXIT		

Final state of your data memory



Total Cycles : 65

Challenges Faced

The problem here was the addi, where we needed to initialize the values. This was also necessary for the incrementation of i which contained the array index. Another issue was trying to fix the slt function so that the desired output was achieved.