Organization of Digital Computers Lab
EECS 112L

Lab 3

Kyle Zyler Cayanan
80576955

02/13/2022

1 - Objective

For this experiment we were tasked with translating MIPS assembly code into machine code, high level language code to assembly then machine code, and lastly converting assembly back into C code. After retrieving the machine code from our 4 problems, we load it into the instruction memory module for our MIPS processor from Lab 2. The following programs is what we have to translate.

3.1 Program 1

```
f = (g + h) - (i + j);
```

3.2 Program 2

```
if (i ≠ j) f = g + h; else f = g - h;
```

3.3 Program 3

```
while (save[i] ≠ k)
    i += 1;
```

3.4 Program 4

```
sll $t0,$a1,2
add $t0,$a0,$t0
lw $t1,0($t0)
lw $t2,4($t0)
sw $t2,0($t0)
sw $t1,4($t0)
```

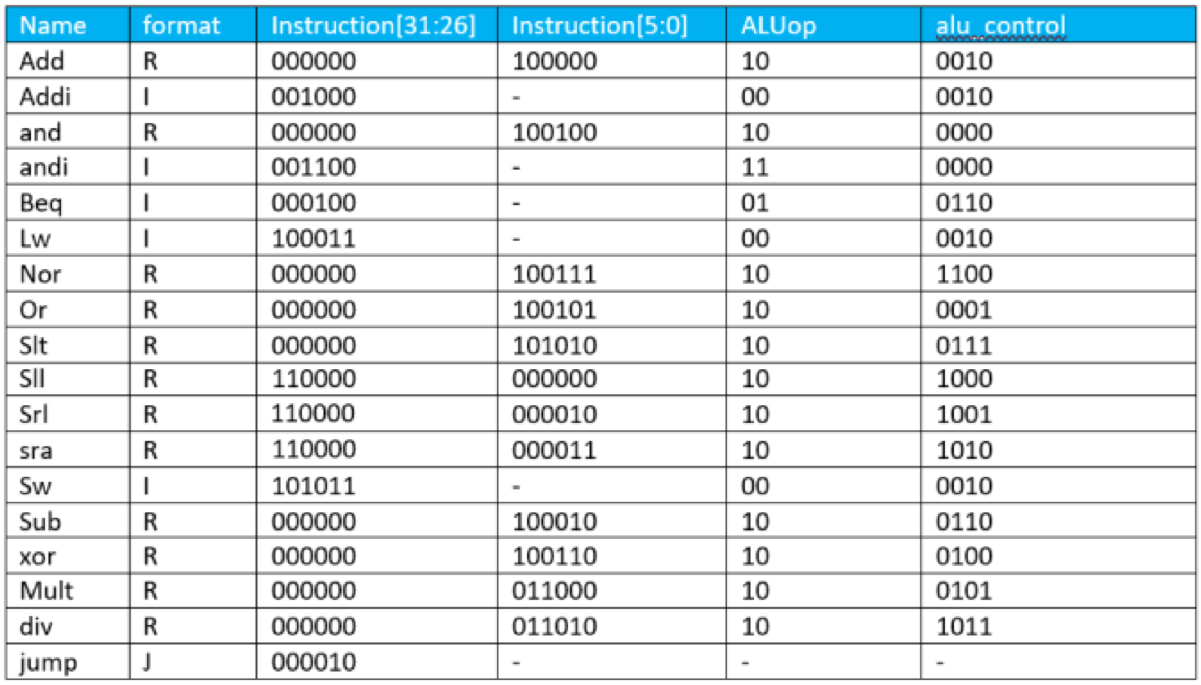
2 - Procedure

Each program's translation will answer the following questions:

1. What MIPS assembly code might a C compiler produce?
2. From the assembly code, what machine code might a MIPS assembler produce?
3. What does this program do? (For program 4 only).

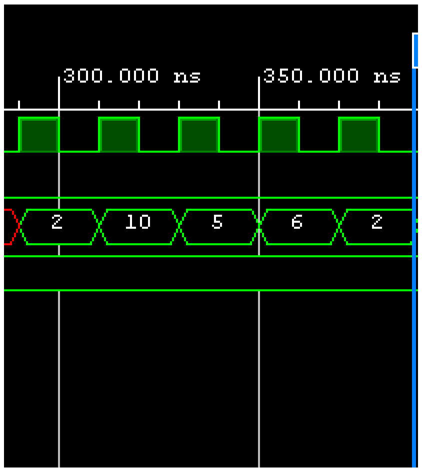
Each program is decoded from assembly code to machine code using the instruction set tables from lab 2 to get the opcode and function code

Table 4

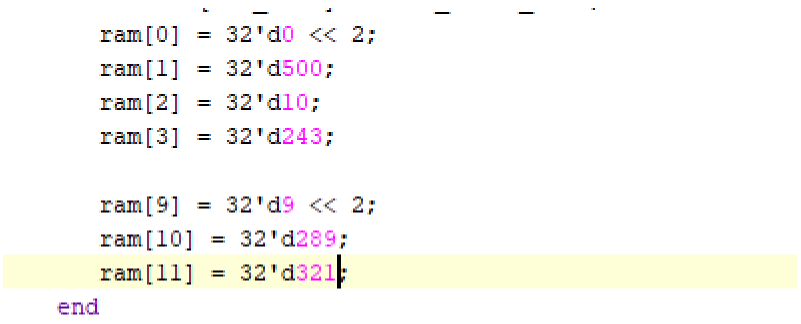


Name	format	Instruction[31:26]	Instruction[5:0]	ALUop	alu_control
Add	R	000000	100000	10	0010
Addi	I	001000	-	00	0010
and	R	000000	100100	10	0000
andi	I	001100	-	11	0000
Beq	I	000100	-	01	0110
Lw	I	100011	-	00	0010
Nor	R	000000	100111	10	1100
Or	R	000000	100101	10	0001
Slt	R	000000	101010	10	0111
Sll	R	110000	000000	10	1000
Srl	R	110000	000010	10	1001
sra	R	110000	000011	10	1010
Sw	I	101011	-	00	0010
Sub	R	000000	100010	10	0110
xor	R	000000	100110	10	0100
Mult	R	000000	011000	10	0101
div	R	000000	011010	10	1011
jump	J	000010	-	-	-

For all programs, we use the following data for each register and the arrays unless shown otherwise:



Here we have $s0-$s4 respectively



```
    ram[0] = 32'd0 << 2;
    ram[1] = 32'd500;
    ram[2] = 32'd10;
    ram[3] = 32'd243;

    ram[9] = 32'd9 << 2;
    ram[10] = 32'd289;
    ram[11] = 32'd321;
end
```

ram[0:3] is the array save[]

ram[9:11] is the array v[]

Program 1

The following is the machine and assembly code for program 1

```
# Program 1
program_1_1: # this is the label for the assembly program
# -------- begin your MIPS assembly code --------
addi $s0, $zero, 2 #all addi instructions load constants for all
programs
addi $s1, $zero, 10
addi $s2, $zero, 5
addi $s3, $zero, 6
addi $s4, $zero, 2

add $t0, $s2, $s3 # g + h
add $t1, $s4, $s0 # i + j
```

```
sub $s1, $t0, $t1 # f = (g + h) - (i + j)
# -------- end ----------------------------------


# Program 1.2
# -------- begin your MIPS Machine code ---------
00100000000100000000000000000010
00100000000100010000000000001010
00100000000100100000000000000101
00100000000100110000000000000110
00100000000101000000000000000010
00000010010100110100000000100000
00000010100100000100100000100000
00000001000010011000100000100010
# -------- end ----------------------------------
```

Program 2

The following is the machine code and assembly code for program 2:

```
# Program 2
program_2_1:
# -------- begin your MIPS assembly code --------
beq $s4, $s0, else #else { f = g - h }
add $s1, $s2, $s3 #if (i!=j) { f = g + h }
j exit
else:
     sub $s1, $s2, $s3
exit: # this will end up jumping to program 3
# -------- end ----------------------------------


# Program 2.2
# -------- begin your MIPS Machine code ---------
00010010000101000000000000000010
00000010010100111000100000100000
00001000000000000000000000010110
00000010010100111000100000100010
# -------- end ----------------------------------
```

<u>Program 3</u>

The following is the machine code and assembly code for program 3:

```
# Program 3
program_3_1:
# -------- begin your MIPS assembly code --------
addi $s1, $zero, 10 #loading k
addi $t0, $zero, 1 #initialize i to 0
addi $t1, $zero, 0 # temp to store calculated address
lw   $s2, 0(0x0) #base address for array
loop:
     sll $t1, $t0, 2 #temp reg $t1 = i*4
     add $t1, $t1, $s2 #add base address
     lw $t2, 0x0($t1) #temp reg $t2 = save[i]
     beq $t2, $s1, exit
     addi $t0, $t0, 1
     j loop
exit: # in this case we will exit to program 4
# -------- end ----------------------------------


# Program 3.2
# -------- begin your MIPS Machine code ---------
00100000000100010000000000001010
00100000000010000000000000000001
00100000000010010000000000000000
10001100000010010000000000000000
11000001000000000100100010000000
00000010010010010100100000100000
10001101001010100000000000000100
00010001010100010000000000000010
00100001000010000000000000000001
00001000000000000000000000011010
# -------- end ----------------------------------
```

<u>Program 4</u>

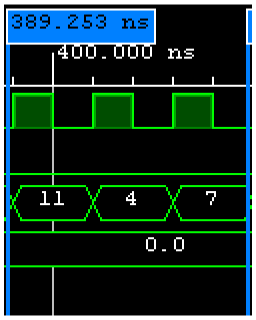
The following is the machine code and C code for program 4:

```
# Program 4.3
void program_4_3(int v[], int k){
```

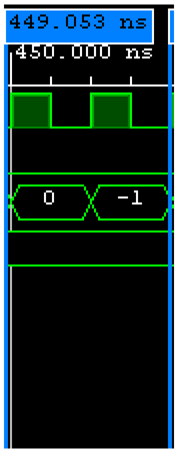
```
# -------- begin your C code --------------------
int temp;
temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;
# -------- end ----------------------------------
    return;
}

# Program 4
# -------- begin your MIPS Machine code ---------
00100000000001010000000000000001 #k = 1
10001100000001000000000000100100 #a0 = mem[9]
11000000101000000100000010000000 #sll $t0, $a1, 2 (k+1)
00000000100010000100000000100000 #add base address
10001101000010010000000000000000
10001101000010100000000000000100
10101101000010100000000000000000
10101101000010010000000000000100
# -------- end ----------------------------------
```

3 - Simulation Results

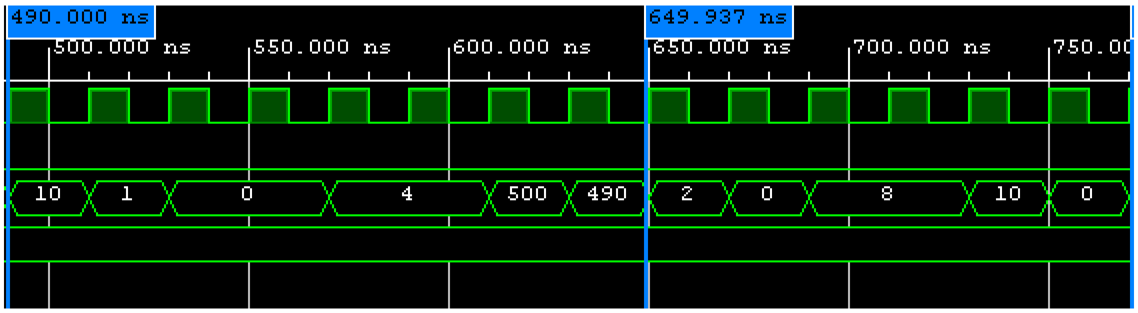
The following are screenshots of the results of the programs above



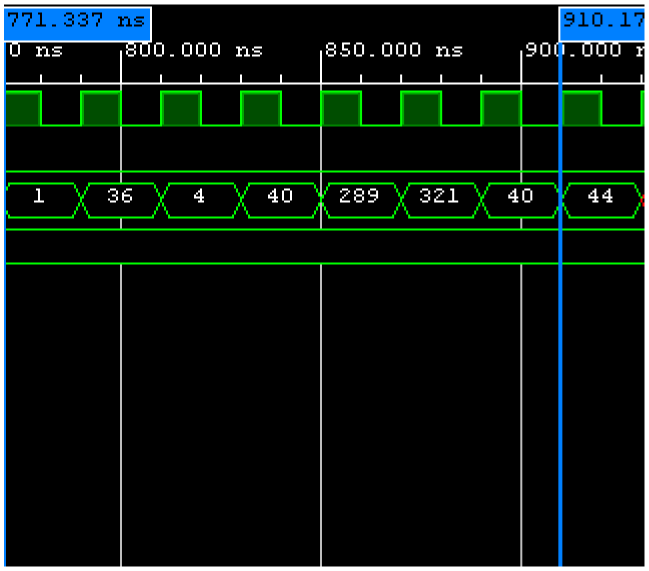
These are the results of program 1,
11 corresponds to g + h,
4 corresponds to i + j,
7 corresponds to f = (g+h) - (i +j);



Above is the result of program 2 when branch is taken
The first output is showing that branch was equal to zero
and then computed the second output $s2 - $s3 or 5 - 6



Program 3:
Here we are indexing through the array save[3] = {500, 10, 243}
First we are checking index i = 1*4 and checking if save[i] = k, if this is true then we exit the program. In this case we have k = 10, and we can see that at index 2 (assuming indexing on 1 for this program) is equal to 10 and the 0 indicates a jump we have taken to exit to the next program.



Program 4:

Here program 4 performs a simple swap of the content of two registers

First we get the index of our first argument k and k + 1 and calculate their respective addresses in the array by multiplying by 4.

The array v here has 289 and 321 as elements. In the output waveform we see the data being loaded in then saved in swapped memory locations.