

Ain Shams University

Faculty of Engineering

Computer and Systems Engineering Department

CSE 311: Computer Organization (2)

**MIPS PROCESSOR**

**Submitted By**

Eslam Samir Ali Abo El-Ala

Mohamed Ahmed Anwer Abdelhalim

Nourhan Essam Ahmed Shiba El-Hamd

Shaza Ismail Kaoud

**Group 13**

**Submitted To**

Dr. Cherif Salama

Eng. Diaa El-Din Mohamed

**Cairo 2015**

Table of Contents

[**List of Figures** v](#_Toc439003971)

[1. IMPLEMENTATION DESCRIPTION 1](#_Toc439003972)

[1.1. The MIPS Processor Specs 1](#_Toc439003973)

[1.2. Supported Instructions 1](#_Toc439003974)

[1.3. Languages Used For Implementation 1](#_Toc439003975)

[1.4. Tools Used For Implementation And Simulation 1](#_Toc439003976)

[1.5. Additional Tools 1](#_Toc439003977)

[2. Datapath 2](#_Toc439003978)

[3. Test Programs and their outputs 2](#_Toc439003979)

[3.1. A program testing addition and subtraction 2](#_Toc439003980)

[3.1.1. Assembly Code 3](#_Toc439003981)

[3.1.2. Machine Code in binary 3](#_Toc439003982)

[3.1.3. Clock cycles taken 4](#_Toc439003983)

[3.1.4. Simulation 4](#_Toc439003984)

[3.2. A program testing If condition 5](#_Toc439003985)

[3.2.1. C++ Code 5](#_Toc439003986)

[3.2.2. Assembly Code 5](#_Toc439003987)

[3.2.3. Machine Code in binary 6](#_Toc439003988)

[3.2.4. Clock cycles taken 6](#_Toc439003989)

[3.2.5. Simulation 6](#_Toc439003990)

[3.3. A program testing Multiplication using SLL 7](#_Toc439003991)

[3.3.1. C++ Code 8](#_Toc439003992)

[3.3.2. Assembly Code 8](#_Toc439003993)

[3.3.3. Machine code in binary 8](#_Toc439003994)

[3.3.4. Clock cycles taken 8](#_Toc439003995)

[3.3.5. Simulation 9](#_Toc439003996)

[3.4. A program testing logic operations 9](#_Toc439003997)

[3.4.1. C++ Code 9](#_Toc439003998)

[3.4.2. Assembly Code 10](#_Toc439003999)

[3.4.3. Machine code in binary 11](#_Toc439004000)

[3.4.4. Clock cycles taken 11](#_Toc439004001)

[3.4.5. Simulation 11](#_Toc439004002)

[3.5. A program testing nested loops 13](#_Toc439004003)

[3.5.1. C++ Code 13](#_Toc439004004)

[3.5.2. Assembly Code 13](#_Toc439004005)

[3.5.3. Machine code in binary 14](#_Toc439004006)

[3.5.4. Clock cycles taken 14](#_Toc439004007)

[3.5.5. Simulation 15](#_Toc439004008)

[3.6. A program testing jr instruction 15](#_Toc439004009)

[3.6.1. C++ Code 15](#_Toc439004010)

[3.6.2. Assembly Code 15](#_Toc439004011)

[3.6.3. Machine code in binary 16](#_Toc439004012)

[3.6.4. Clock cycles taken 16](#_Toc439004013)

[3.6.5. Simulation 16](#_Toc439004014)

[3.7. A program initializing an array using loops 17](#_Toc439004015)

[3.7.1. C++ Code 17](#_Toc439004016)

[3.7.2. Assembly Code 17](#_Toc439004017)

[3.7.3. Machine code in binary 17](#_Toc439004018)

[3.7.4. Clock cycles taken 18](#_Toc439004019)

[3.7.5. Simulation 18](#_Toc439004020)

[3.8. A program testing looping and summation 21](#_Toc439004021)

[3.8.1. C++ Code 21](#_Toc439004022)

[3.8.2. Assembly Code 22](#_Toc439004023)

[3.8.3. Machine code in binary 22](#_Toc439004024)

[3.8.4. Clock cycles taken 23](#_Toc439004025)

[3.8.5. Simulation 23](#_Toc439004026)

[3.9. A program testing checking values and branching 25](#_Toc439004027)

[3.9.1. C++ Code 25](#_Toc439004028)

[3.9.2. Assembly Code 26](#_Toc439004029)

[3.9.3. Machine code in binary 26](#_Toc439004030)

[3.9.4. Clock cycles taken 26](#_Toc439004031)

[3.9.5. Simulation 26](#_Toc439004032)

[3.10. A program testing 2D arrays 27](#_Toc439004033)

[3.10.1. C++ Code 27](#_Toc439004034)

[3.10.2. Assembly Code 27](#_Toc439004035)

[3.10.3. Initial data memory in binary 28](#_Toc439004036)

[3.10.4. Machine code in binary 28](#_Toc439004037)

[3.10.5. Clock cycles taken 28](#_Toc439004038)

[3.10.6. Simulation 29](#_Toc439004039)

[3.11. A program testing reading initial values from memory & simple operations 33](#_Toc439004040)

[3.11.1. Assembly Code 33](#_Toc439004041)

[3.11.2. Initial data memory in binary 33](#_Toc439004042)

[3.11.3. Machine code in binary 33](#_Toc439004043)

[3.11.4. Clock cycles taken 33](#_Toc439004044)

[3.11.5. Simulation 34](#_Toc439004045)

[3.12. A program testing load and compare 34](#_Toc439004046)

[3.12.1. Assembly Code 34](#_Toc439004047)

[3.12.2. Initial data memory in binary 35](#_Toc439004048)

[3.12.3. Machine code in binary 35](#_Toc439004049)

[3.12.4. Clock cycles taken 35](#_Toc439004050)

[3.12.5. Simulation 35](#_Toc439004051)

[4. Contribution 36](#_Toc439004052)

[5. Assembler Manual 37](#_Toc439004053)

[5.1. Usage 37](#_Toc439004054)

[5.2. Supported Instructions 37](#_Toc439004055)

[5.3. Supported Data Types 37](#_Toc439004056)

[5.4. Example of Assembly File Format 37](#_Toc439004057)

[6. REFERENCES 38](#_Toc439004058)

**List of Figures**

[Figure 1 MIPS Processor Datapath 2](#_Toc439003630)

[Figure 2 Program 1 Data Memory 4](#_Toc439003631)

[Figure 3 Program 1 Register File 5](#_Toc439003632)

[Figure 4 Program 2 Data Memory 7](#_Toc439003633)

[Figure 5 Program 2 Register File 7](#_Toc439003634)

[Figure 6 Program 3 Data Memory 9](#_Toc439003635)

[Figure 7 Program 3 Register File 9](#_Toc439003636)

[Figure 8 Program 4 Data Memory 12](#_Toc439003637)

[Figure 9 Program 4 Register File 13](#_Toc439003638)

[Figure 10 Program 5 Data Memory 15](#_Toc439003639)

[Figure 11 Program 5 Register File 15](#_Toc439003640)

[Figure 12 Program 6 Data Memory 16](#_Toc439003641)

[Figure 13 Program 6 Register File 17](#_Toc439003642)

[Figure 14 Program 7 Data Memory 20](#_Toc439003643)

[Figure 15 Program 7 Register File 21](#_Toc439003644)

[Figure 16 Program 8 Data Memory 24](#_Toc439003645)

[Figure 17 Program 8 Register File 25](#_Toc439003646)

[Figure 18 Program 9 Data Memory 26](#_Toc439003647)

[Figure 19 Program 9 Register File 27](#_Toc439003648)

[Figure 20 Program 10 Data Memory 32](#_Toc439003649)

[Figure 21 Program 10 Register File 32](#_Toc439003650)

[Figure 22 Program 11 Data Memory 34](#_Toc439003651)

[Figure 23 Program 11 Register File 34](#_Toc439003652)

[Figure 24 Program 12 Data Memory 35](#_Toc439003653)

[Figure 25 Program 12 Register File 36](#_Toc439003654)

# IMPLEMENTATION DESCRIPTION

## The MIPS Processor Specs

* Single cycle MIPS processor alongside an assembler.
* 256 Bytes instruction memory (byte addressable).
* 2 Kbytes data memory (byte addressable).
* 32-bits 32 registers Register File.
* A clock with a clock period 80 ns.

## Supported Instructions

* Arithmetic: **add, addi, sub**
* Load/Store: **lw, sw**
* Logic: **sll, and, andi, nor**
* Control flow: **beq, jal, jr**
* Comparison: **slt**

## Languages Used For Implementation

* Verilog HDL for implementing the MIPS processor.
* Ruby for implementing the assembler. (bonus)
* Assembly and machine code for testing.

## Tools Used For Implementation And Simulation

* ModelSim.
* Active HDL.
* Sublime text.

## Additional Tools

* [OCRA](https://github.com/larsch/ocra/blob/master/bin/ocra) gem for packaging assembler to exe.
* Git and Github.

# Datapath

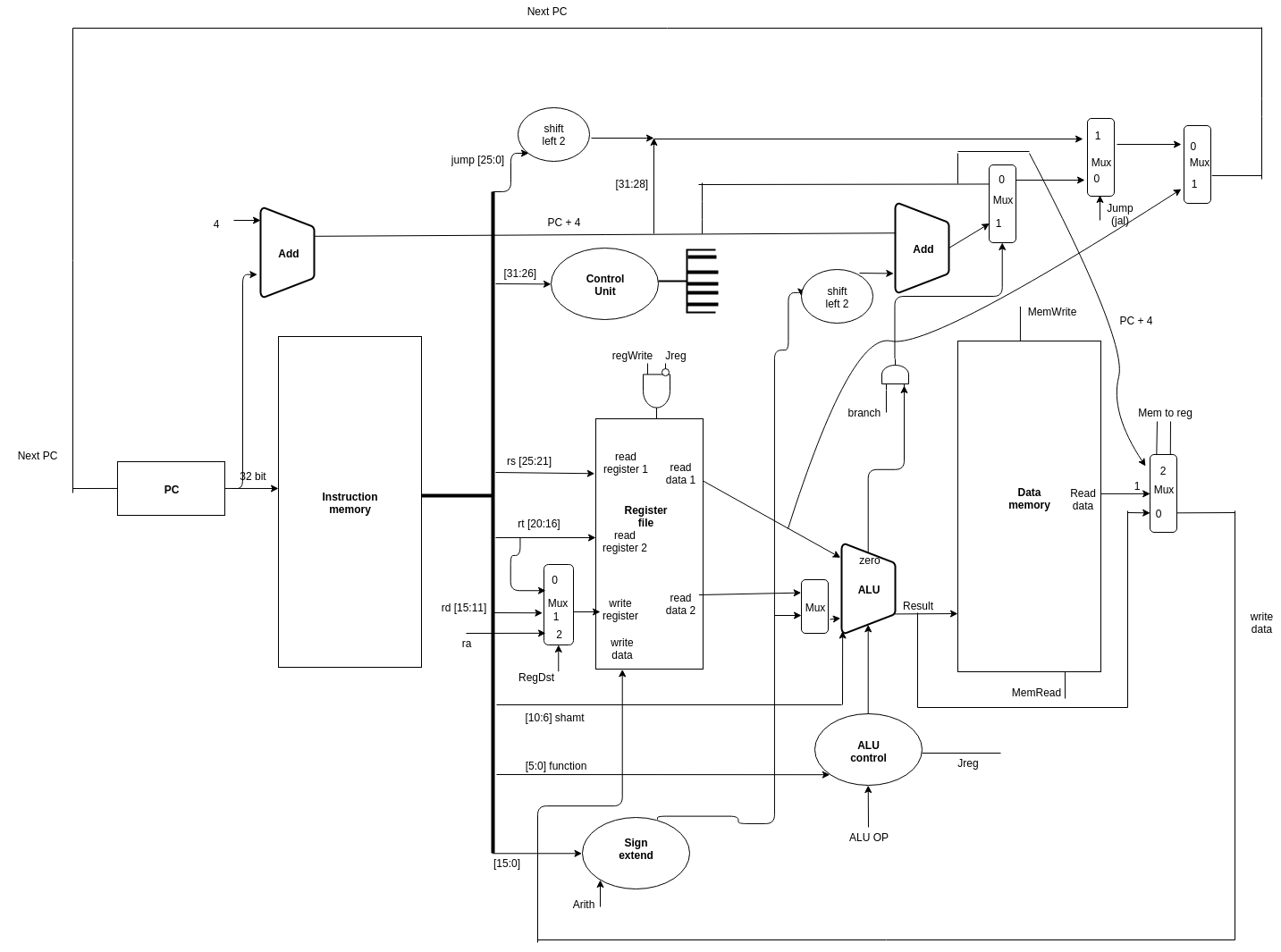


Figure 1 MIPS Processor Datapath

# Test Programs and their outputs

## A program testing addition and subtraction

Description: This program adds and subtracts certain values and stores these values in the memory.

### Assembly Code

|  |
| --- |
| #initializing data  addi $s0, $zero, 4 #$s0 = 4  addi $s1, $s0, 6 #$s1 = 10  addi $s2, $zero, 16 #$s2 = 16  addi $s3, $zero, 68 #$s3 = 68  addi $s4, $zero, 40 #$s4 = 40  #adding and subtracting  add $s0, $s3, $s4 #$s0 = (40+68) = 108  add $s4, $s1, $s3 #$s4 = (68+10) = 78  addi $s1, $s2, -10 #$s1 = (16-10) = 6  slt $s3, $s4, $s1 #$s3 = 0  slt $s2, $s3, $s0 #$s2 = 1  #storing data into memory  sw $s0, 4($s3) #$s0 in loc 4 in memory (0+4) .. loc 4 contains 108  sw $s1, 8($s3) #$s1 in loc 8 in memory (0+8) .. loc 8 contains 6  sw $s3, 4($s0) #$s3 in loc 112 in memory (108+4) .. loc 112 contains 0  lw $s5, 8($s3) #$s5 = 6  addi $s5, $s5, 2 #$s5 = 8  sw $s2, 12($s5) #$s2 in loc 20 in memory (12+8) .. loc 20 contains 1  sw $s4, 20($s5) #$s4 in loc 28 in memory (20+8) .. loc 28 contains 78 |

\*we expect that location 4 in memory contains 108 in decimal or 6C in hexadecimal, location 8 contains 6, location 20 contains 1, location 28 contains 78, and location 112 contains 0.

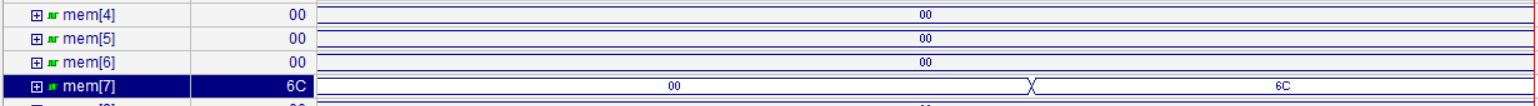
### Machine Code in binary

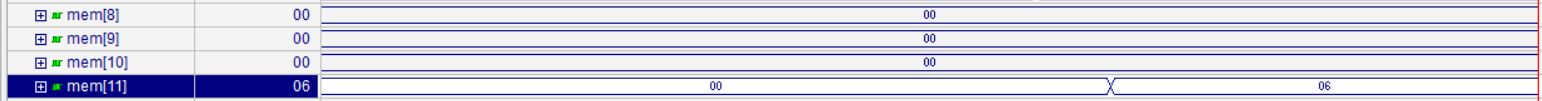
|  |
| --- |
| 00100000 00010000 00000000 00000100  00100010 00010001 00000000 00000110  00100000 00010010 00000000 00010000  00100000 00010011 00000000 01000100  00100000 00010100 00000000 00101000  00000010 01110100 10000000 00100000  00000010 00110011 10100000 00100000  00100010 01010001 11111111 11110110  00000010 10010001 10011000 00101010  00000010 01110000 10010000 00101010  10101110 01110000 00000000 00000100  10101110 01110001 00000000 00001000  10101110 00010011 00000000 00000100  10001110 01110101 00000000 00001000  00100010 10110101 00000000 00000010  10101110 10110010 00000000 00001100  10101110 10110100 00000000 00010100 |

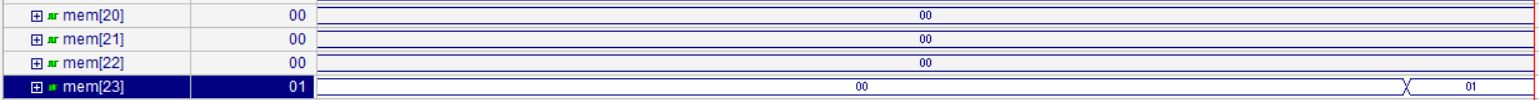
### Clock cycles taken

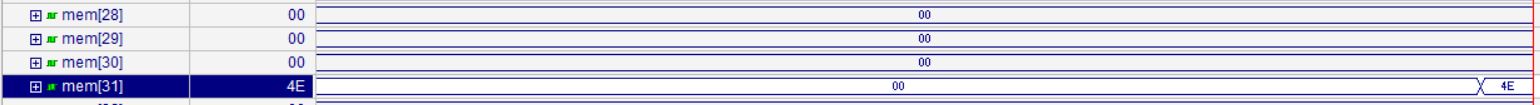
|  |
| --- |
| 17 Clock Cycles |

### Simulation









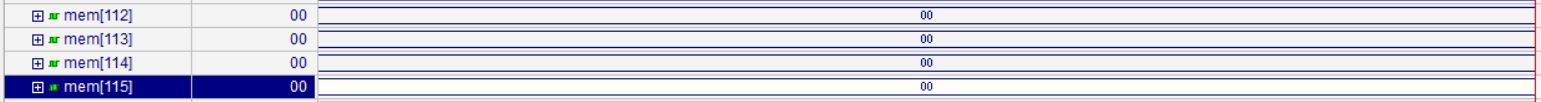
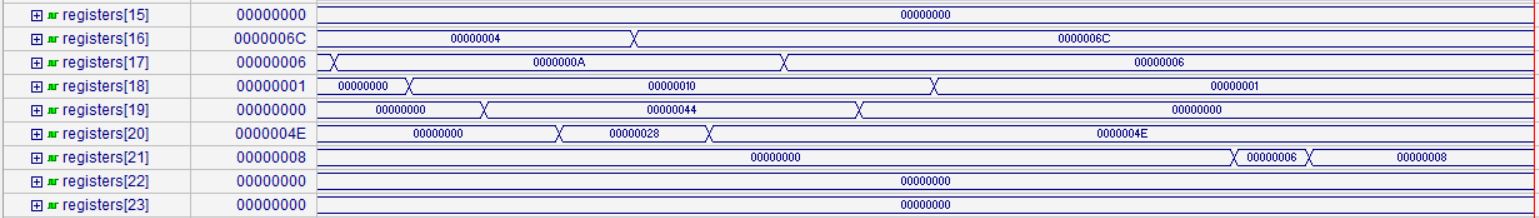


Figure 2 Program 1 Data Memory

\*The memory is byte addressable, so location 4 contains 0x6C (128 in decimal), location 8 contains 0x06, location 20 contains 0x01, location 28 contains 0x4E (78 in decimal), and location 112 contains 0x00. those values are similar to the values expected before.



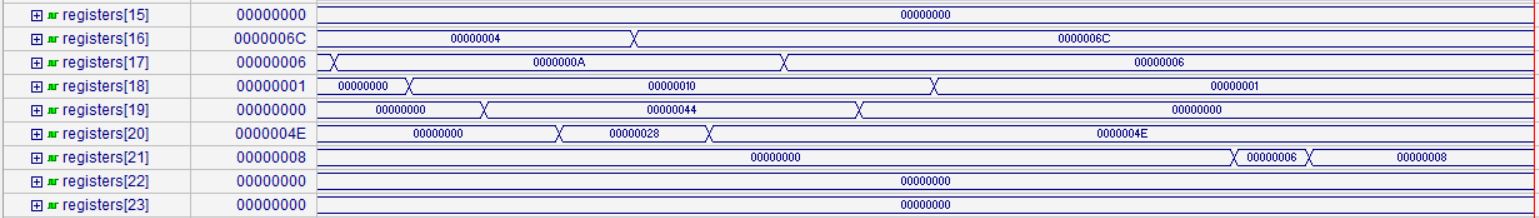


Figure 3 Program 1 Register File

## A program testing If condition

Description: This program tests if condition and branch.

### C++ Code

|  |
| --- |
| i=0;  j=14;  if (i<j) {  A[i] = 1000;  }  i=20;  if (i<j) {  A[i] = 2000;  } |

### Assembly Code

|  |
| --- |
| #initializing data  addi $s0, $zero, 4 #$s0 = 4  addi $s1, $zero, 0 #$s1 = 0  addi $s2, $s0, 10 #$s2 = 14  sw $s1, 0($s0) #$s1 stored in location 4 .. 0 in loc 4  sw $s2, 4($s0) #$s2 stored in location 8 .. 14 in loc 8  #if condition  lw $t0, 0($s0) #$t0 = 0  lw $t1, 4($s0) #$t1 = 14  slt $t2, $t0, $t1 #$t2 = 1  beq $t2, $zero, exit #wont branch  sll $t0, $t0, 2 #$t0\*4 = 0  add $t0, $t0, $s0 #$t0 = 4  addi $s5, $zero, 1000 #$s5 = 1000  sw $s5, 0($t0) #put 1000 in loc 4  addi $t0, $t0, 20 #$t0 = 24  slt $t2, $t0, $t1 #$t2 = 0  beq $t2, $zero, exit #Branch  sll $t0, $t0, 2 #$t0\*4 = 96 .. wont be executed  add $t0, $t0, $s0 #$t0 = 100 .. wont be executed  sw $s5, 0($t0) #put 1000 in loc 100 .. wont be executed  exit: |

\*we expect location 4 in memory to have 1000 in decimal (0x3E8), location 8 to have 14 (0xE), and location 100 to have 0.

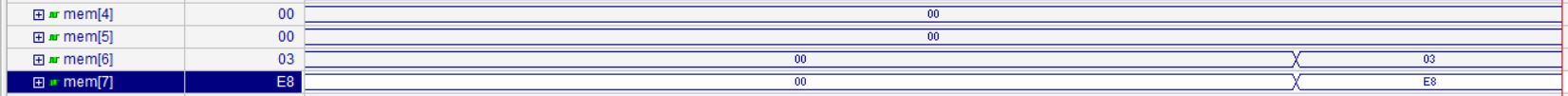
### Machine Code in binary

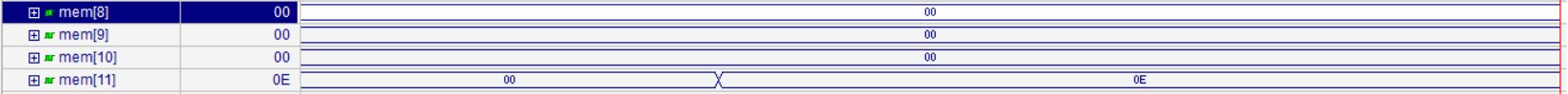
|  |
| --- |
| 00100000 00010000 00000000 00000100  00100000 00010001 00000000 00000000  00100010 00010010 00000000 00001010  10101110 00010001 00000000 00000000  10101110 00010010 00000000 00000100  10001110 00001000 00000000 00000000  10001110 00001001 00000000 00000100  00000001 00001001 01010000 00101010  00010000 00001010 00000000 00001010  00000001 00000000 01000000 10000000  00000001 00010000 01000000 00100000  00100000 00010101 00000011 11101000  10101101 00010101 00000000 00000000  00100001 00001000 00000000 00010100  00000001 00001001 01010000 00101010  00010000 00001010 00000000 00000011  00000001 00000000 01000000 10000000  00000001 00010000 01000000 00100000  10101101 00010101 00000000 00000000 |

### Clock cycles taken

|  |
| --- |
| 16 Clock Cycles |

### Simulation





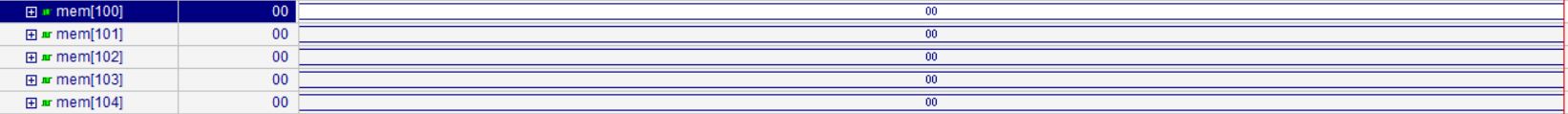
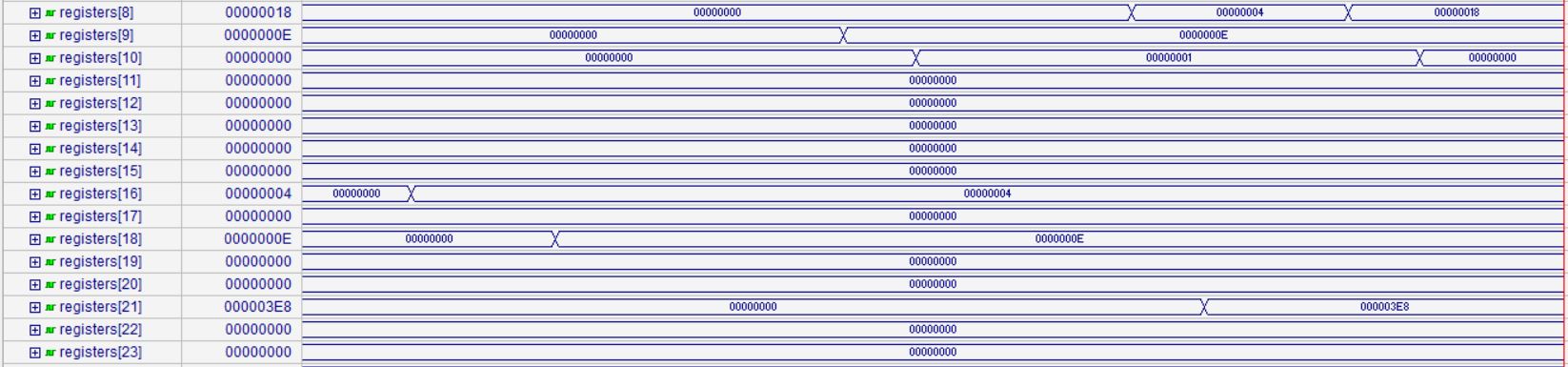


Figure 4 Program 2 Data Memory

\*The memory is byte addressable, so location 4 contains 0x3E8 (1000 in decimal), location 8 contains 0xE, location 100 contains 0x0. those values are similar to the values expected before.



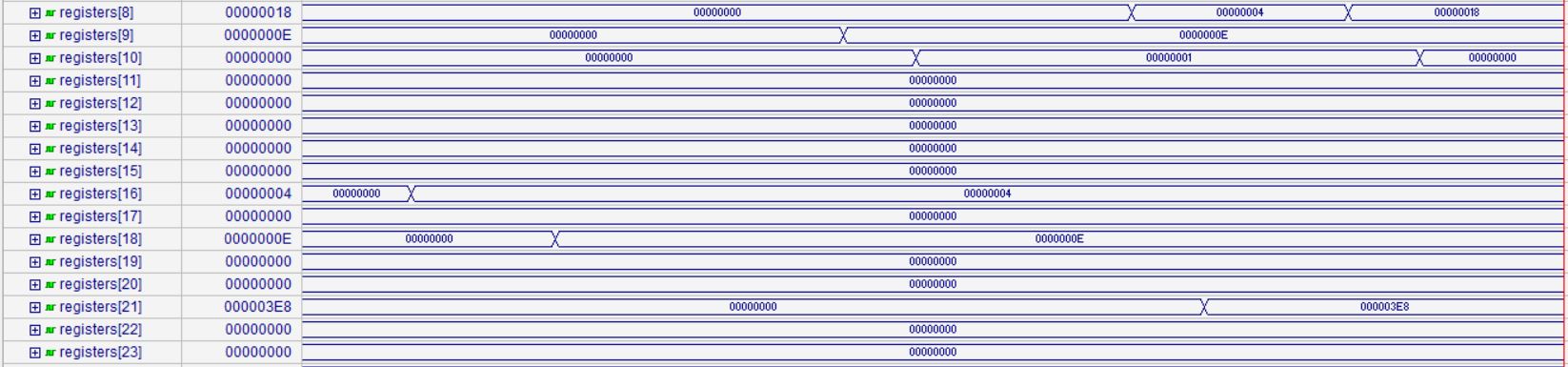


Figure 5 Program 2 Register File

## A program testing Multiplication using SLL

Description: This program loops 10 times and multiplies A by 2 each loop, initially A equals 2.

### C++ Code

|  |
| --- |
| A = 2;  for (i=0 ; i<10 ; i++) {  A = A\*2;  } |

### Assembly Code

|  |
| --- |
| #initializing data    addi $s0, $zero, 2 #$s0 = 2 -> A  addi $s1, $zero, 0 #$s1 = 0 -> i  addi $s2, $zero, 10 #$s2 = 10  addi $s3, $zero, 0 #$s3 = 0    # Multiply and Divide    loop: sll $s0, $s0, 1 #$s0 = $s0\*2  addi $s1, $s1, 1  beq $s1, $s2, end  jal loop  end: sw $s0, 0($s3) #A is stored in location 0 .. loc 0 contains 2048 (0x800) |

\*Initially A was equal to 2, the loop loops 10 times and each time it multiplies A by 2, hence, the final expected result is 2\*(2^10) = 2048 (0x800) which is stored in location 0 in memory.

### Machine code in binary

|  |
| --- |
| 00100000 00010000 00000000 00000010  00100000 00010001 00000000 00000000  00100000 00010010 00000000 00001010  00100000 00010011 00000000 00000000  00000000 00010000 10000000 01000000  00100010 00110001 00000000 00000001  00010010 01010001 00000000 00000001  00001100 00000000 00000000 00000100  10101110 01110000 00000000 00000000 |

### Clock cycles taken

|  |
| --- |
| 44 clock cycles. |

### 

### Simulation

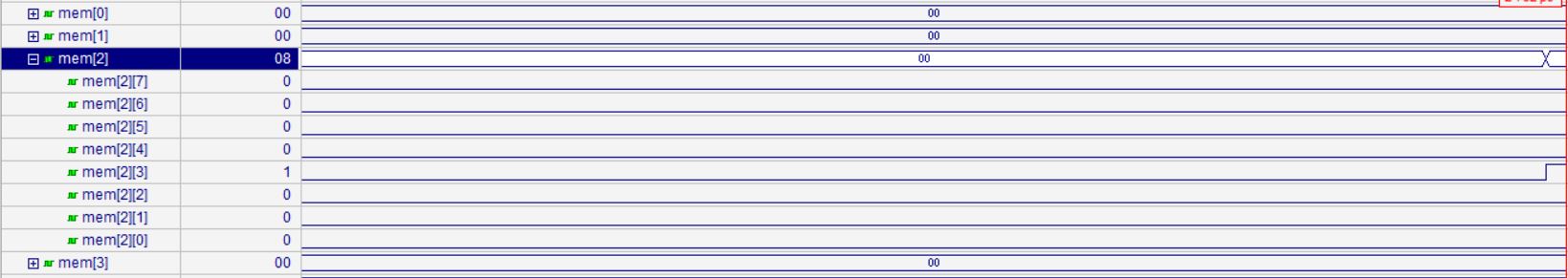
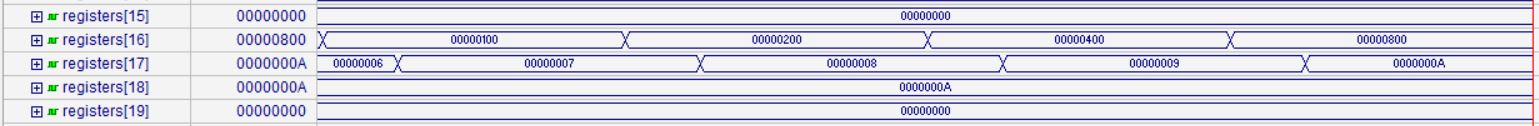
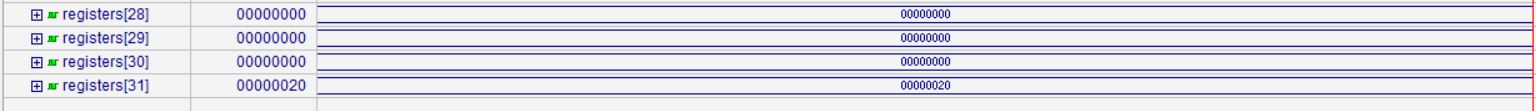
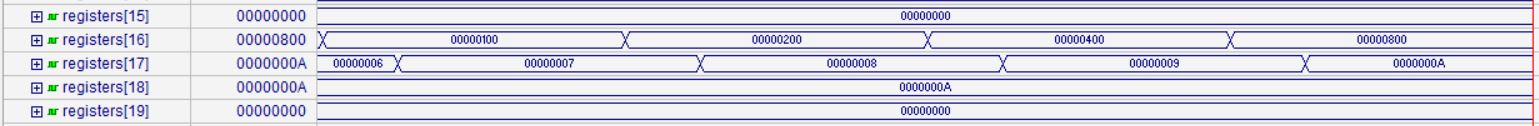


Figure 6 Program 3 Data Memory

\*the data memory contains 0x800 in location 4 as expected before (memory is byte addressable).







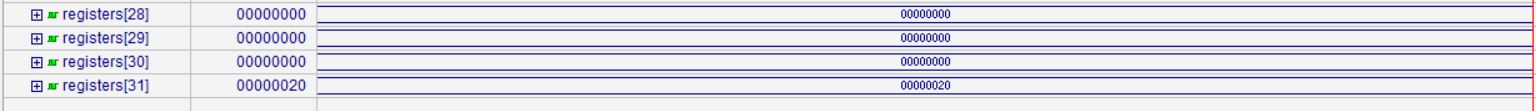


Figure 7 Program 3 Register File

## A program testing logic operations

Description: this program does some logic operations on data stored in registers.

### C++ Code

|  |
| --- |
| A = 215;  B = 255;  C = 157;  D = 0;  E = A&B;  F = A&C;  G = A&D;  H = A!|B;  I = A!|C;  J = A!|D;  H = A&5; |

### Assembly Code

|  |
| --- |
| #initializing data    addi $s0, $zero, 215 #$s0 = 215 -> A  addi $s1, $zero, 255 #$s1 = 255 -> B  addi $s2, $zero, 157 #$s2 = 157 -> C  addi $s3, $zero, 0 #$s3 = 0 -> D    #Logic operations    and $t0, $s0, $s1 #$t0 = 215 = 00000000000000000000000011010111 (0x000000D7)  and $t1, $s0, $s2 #$t1 = 149 = 00000000000000000000000010010101 (0x00000095)  and $t2, $s0, $s3 #$t2 = 0 = 00000000000000000000000000000000 (0x00000000)  nor $t3, $s0, $s1 #$t3 = 32 = 11111111111111111111111100100000 (0xFFFFFF00)  nor $t4, $s0, $s2 #$t4 = 40 = 11111111111111111111111100101000 (0xFFFFFF20)  nor $t5, $s0, $s3 #$t5 = 40 = 11111111111111111111111100101000 (0xFFFFFF28)  andi $t6, $s0, 5 #$t6 = 05 = 00000000000000000000000000000101 (0x00000005)    #Storing data    sw $t0, 0($s3)  sw $t1, 4($s3)  sw $t2, 8($s3)  sw $t3, 16($s3)  sw $t4, 20($s3)  sw $t5, 24($s3)  sw $t6, 28($s3) |

\*As we can see from the commented code, the expected value in location 0 in memory is 0x000000D7, in location 4 is 0x00000095, in location 8 is 0x00000000, in location 16 is 0xFFFFFF00, in location 20 is 0xFFFFFF20, in location 24 is 0xFFFFFF28, and in location 28 is 0x00000005.

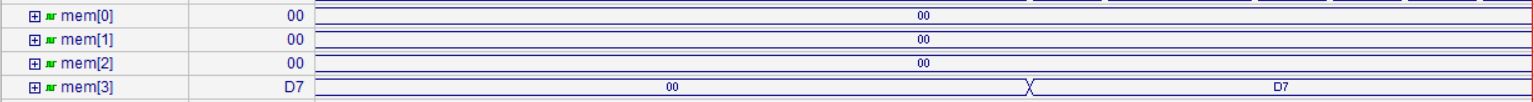
### Machine code in binary

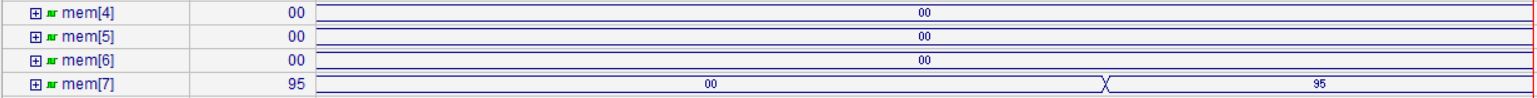
|  |
| --- |
| 00100000 00010000 00000000 11010111  00100000 00010001 00000000 11111111  00100000 00010010 00000000 10011101  00100000 00010011 00000000 00000000  00000010 00010001 01000000 00100100  00000010 00010010 01001000 00100100  00000010 00010011 01010000 00100100  00000010 00010001 01011000 00100111  00000010 00010010 01100000 00100111  00000010 00010011 01101000 00100111  00110010 00001110 00000000 00000101  10101110 01101000 00000000 00000000  10101110 01101001 00000000 00000100  10101110 01101010 00000000 00001000  10101110 01101011 00000000 00010000  10101110 01101100 00000000 00010100  10101110 01101101 00000000 00011000  10101110 01101110 00000000 00011100 |

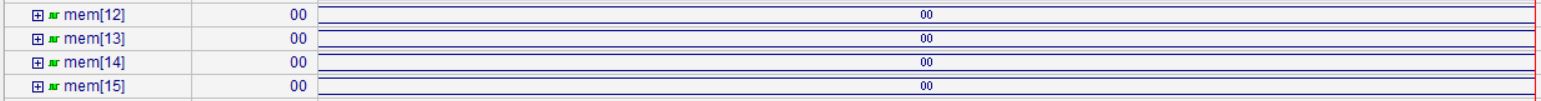
### Clock cycles taken

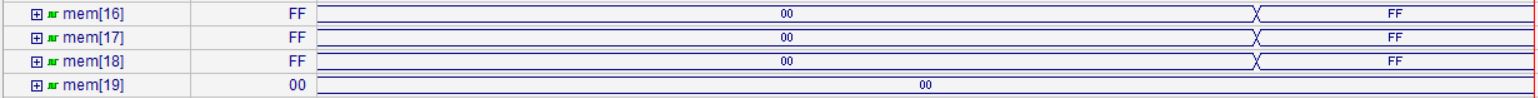
|  |
| --- |
| 18 clock cycles |

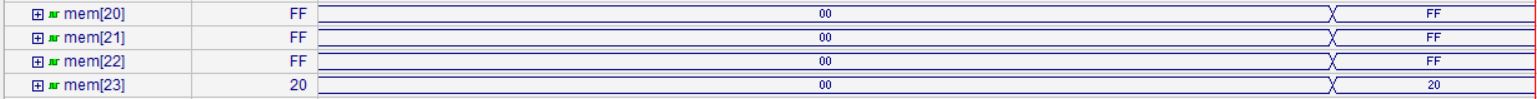
### Simulation

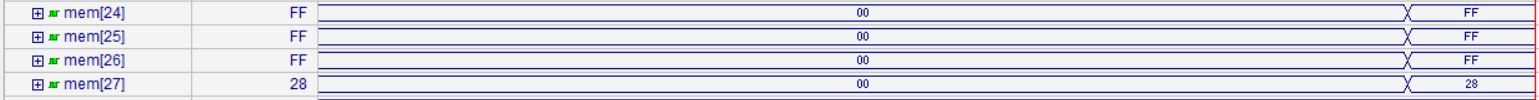


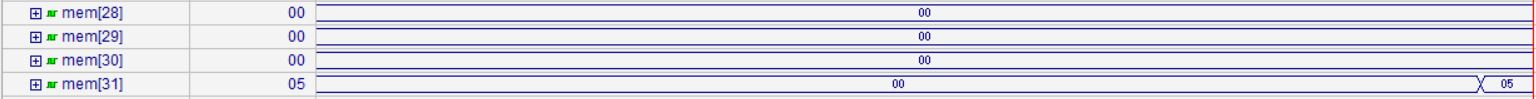












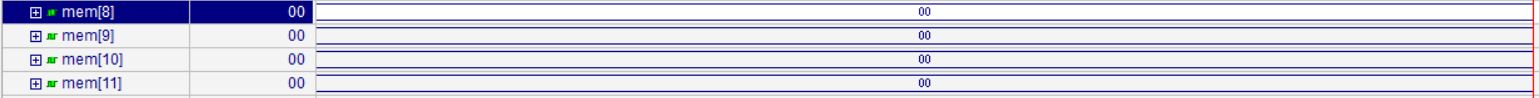
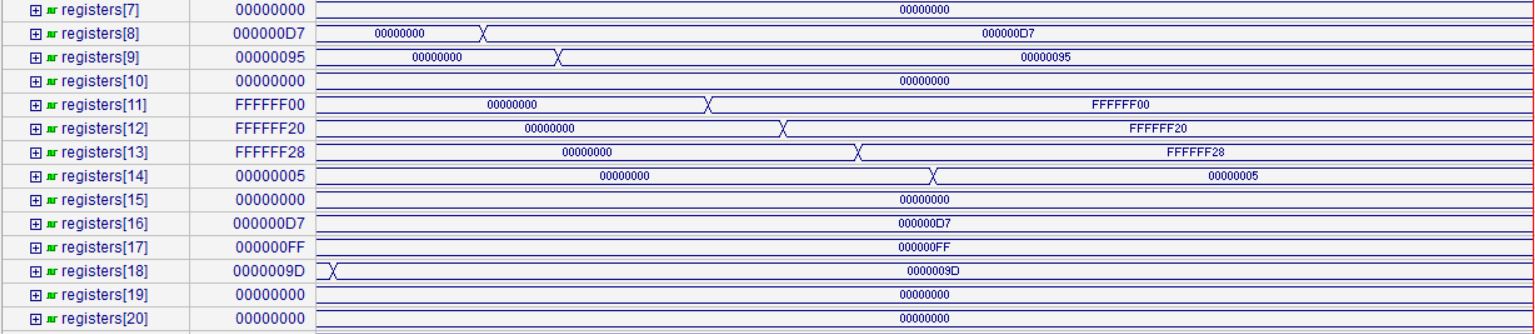


Figure 8 Program 4 Data Memory

\*the data memory contains values 0x800 in location 0 in memory is 0x000000D7, in location 4 is 0x00000095, in location 8 is 0x00000000, in location 16 is 0xFFFFFF00, in location 20 is 0xFFFFFF20, in location 24 is 0xFFFFFF28, and in location 28 is 0x00000005, as expected before(memory is byte addressable).



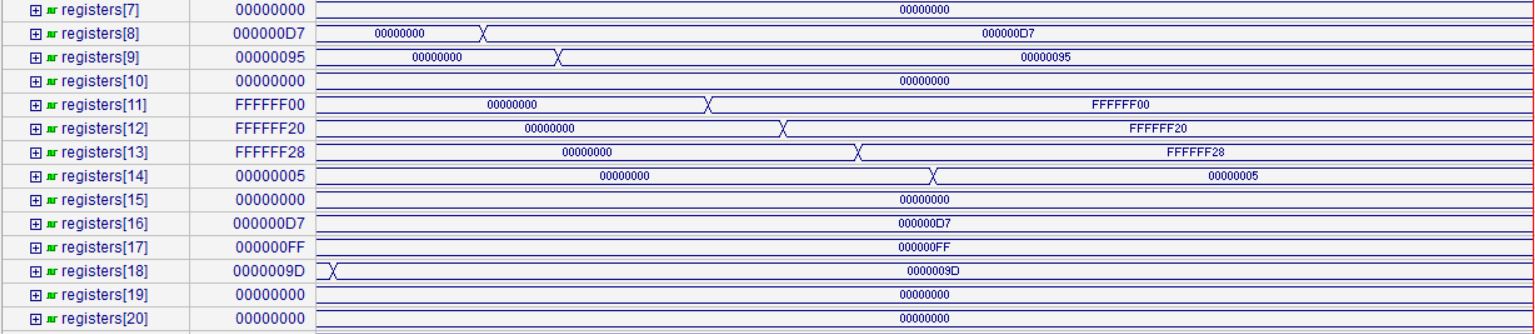


Figure 9 Program 4 Register File

## A program testing nested loops

Description: This program has to loops, each loops from 0 till 5, the program counts number of equal elements in both loops.

### C++ Code

|  |
| --- |
| int n=0;  for (int i=0;i<5;i++)  {  for(int j=0;j<5;j++)  {  if(i==j)  {  n++;  }  }  } |

### Assembly Code

|  |
| --- |
| #initializing data    addi $s0, $zero, 0 #n=0  addi $s1, $zero, 0 #i=0  addi $s2, $zero, 0 #j=0  addi $s3, $zero, 5  addi $s4, $zero, 0    L1: addi $s2, $zero, 0 #j=0  L2: beq $s1, $s2, equal  jal skip    equal: addi $s0, $s0, 1    skip: addi $s2, $s2, 1  beq $s2, $s3, L3  jal L2    L3: addi $s1, $s1, 1  beq $s1, $s3, exit  jal L1  exit: sw $s0, 0($s4) #loc 0 contains 5 |

\*We expect location 0 in memory to have value 5 (0x05), which is the number of times I was equal to j.

### Machine code in binary

|  |
| --- |
| 00100000 00010000 00000000 00000000  00100000 00010001 00000000 00000000  00100000 00010010 00000000 00000000  00100000 00010011 00000000 00000101  00100000 00010100 00000000 00000000  00100000 00010010 00000000 00000000  00010010 01010001 00000000 00000001  00001100 00000000 00000000 00001010  00100010 00010000 00000000 00000001  00100010 01010010 00000000 00000001  00010010 01110010 00000000 00000001  00001100 00000000 00000000 00000111  00100010 00110001 00000000 00000001  00010010 01110001 00000000 00000001  00001100 00000000 00000000 00000101  10101110 10010000 00000000 00000000 |

### Clock cycles taken

|  |
| --- |
| 145 |

### Simulation

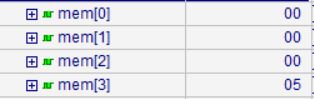
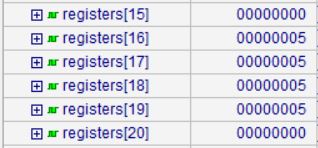


Figure 10 Program 5 Data Memory

\*the data memory contains value 0x05 in location 0 in memory, as expected before(memory is byte addressable).



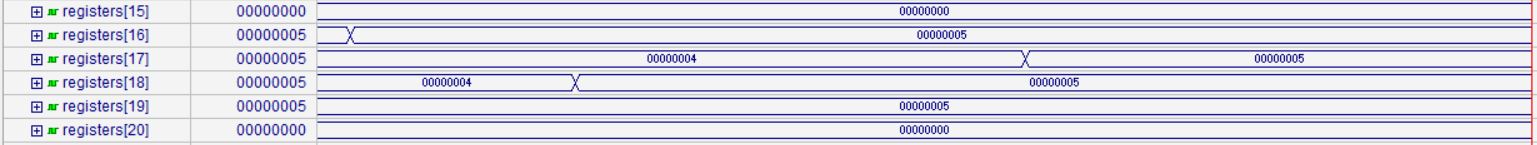


Figure 11 Program 5 Register File

## A program testing jr instruction

Description: This program loops, and uses jr instruction in jumping.

### C++ Code

|  |
| --- |
| for(int i=0;i<5;i++)  {  j++;  } |

### Assembly Code

|  |
| --- |
| addi $s0, $zero, 0 #j=0  addi $s1, $zero, 0 #i=0  addi $s2, $zero, 5  addi $s3, $zero, 16  addi $s4, $zero, 0    addi $s0, $s0, 1 #j++  addi $s1, $s1, 1 #i++  beq $s1, $s2, exit  jr $s3 #jump to instruction number 4  exit: sw $s0,0($s4) |

\*The data memory is expected to contain 0x05 (as the loop loops 5 times so j = 5) in location 0.

### Machine code in binary

|  |
| --- |
| 00100000 00010000 00000000 00000000  00100000 00010001 00000000 00000000  00100000 00010010 00000000 00000101  00100000 00010011 00000000 00010000  00100000 00010100 00000000 00000000  00100010 00010000 00000000 00000001  00100010 00110001 00000000 00000001  00010010 01010001 00000000 00000001  00000010 01100000 00000000 00001000  10101110 10010000 00000000 00000000 |

### Clock cycles taken

|  |
| --- |
| 29 clock cycles. |

### Simulation

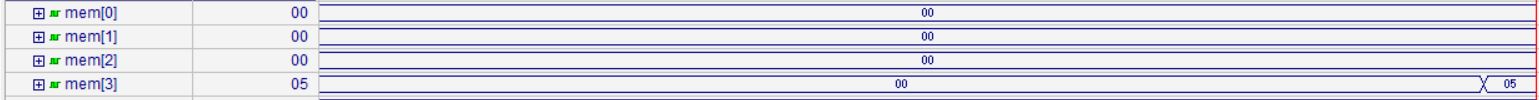
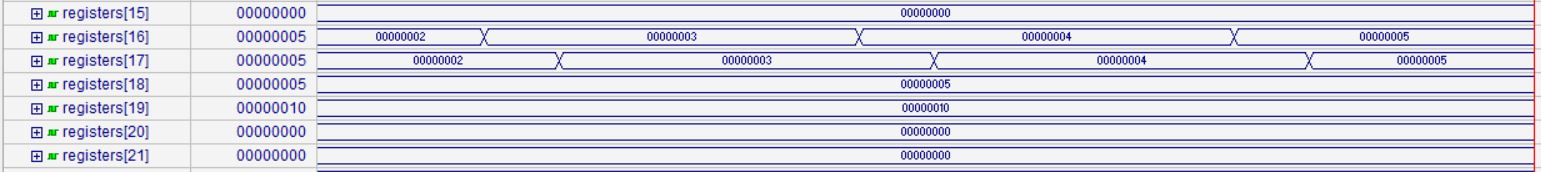


Figure 12 Program 6 Data Memory

\*the data memory contains value 0x05 in location 0 in memory, as expected before(memory is byte addressable).



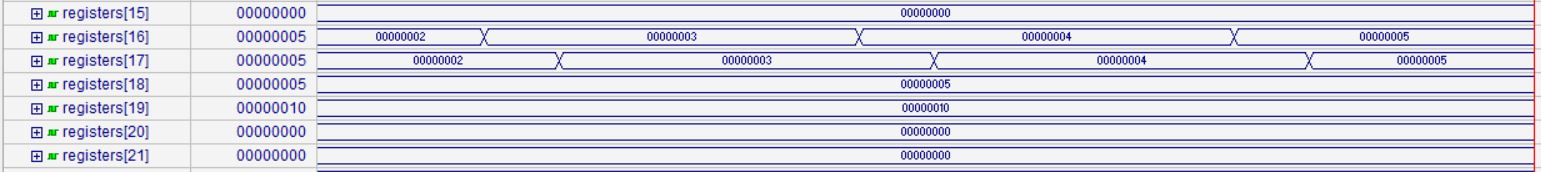


Figure 13 Program 6 Register File

## A program initializing an array using loops

Description: this program initializes an array from the second element with value = first element + 100.

### C++ Code

|  |
| --- |
| for (int i=1;i<100;i++)  {  A[i] = h+B[0];  } |

### Assembly Code

|  |
| --- |
| addi $s0, $zero, 12 #base address of A =12  addi $s5, $zero, 5 #A[0] = 5  addi $s2, $zero, 100 #h = 100 (0x64)  addi $s3, $zero, 1 #i=1  addi $s4, $zero, 100  addi $s1, $zero, 0    sw $s2, 0($s1) #h = 100 in loc 0  sw $s5, 0($s0) #A[0] = 5 in loc 12    loop: lw $t0, 0($s0) #$t0 = A[0] = 5  add $t0, $t0, $s2 #$t0 = A[0] +h = 105  sll $t1, $s3, 2 #$t1 = i\*4  add $t1, $t1, $s0 #$t1 = i\*4 + base of A  sw $t0, 0($t1) #store 105 in loc $t1  addi $s3, $s3, 1 #i++  beq $s3, $s4, exit #if(i==100) {goto exit;}  jal loop #jump to loop  exit: |

\*The expected values in memory are: location 0 contains 100 (0x64), location 12 contains 5 (0x05) (first element in the array), and the array will be stored starting from the second element from location 16 till location 408 each containing (0x69).

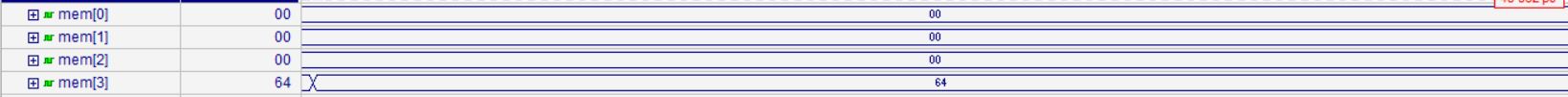
### Machine code in binary

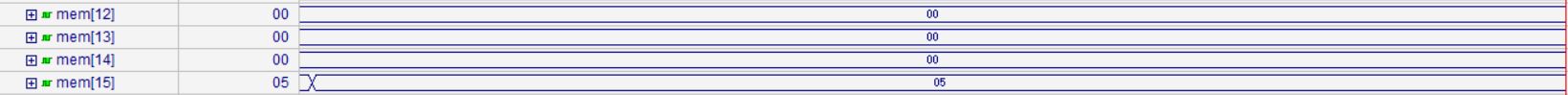
|  |
| --- |
| 00100000 00010000 00000000 00001100  00100000 00010101 00000000 00000101  00100000 00010010 00000000 01100100  00100000 00010011 00000000 00000001  00100000 00010100 00000000 01100100  00100000 00010001 00000000 00000000  10101110 00110010 00000000 00000000  10101110 00010101 00000000 00000000  10001110 00001000 00000000 00000000  00000001 00010010 01000000 00100000  00000000 00010011 01001000 10000000  00000001 00110000 01001000 00100000  10101101 00101000 00000000 00000000  00100010 01110011 00000000 00000001  00010010 10010011 00000000 00000001  00001100 00000000 00000000 00001000 |

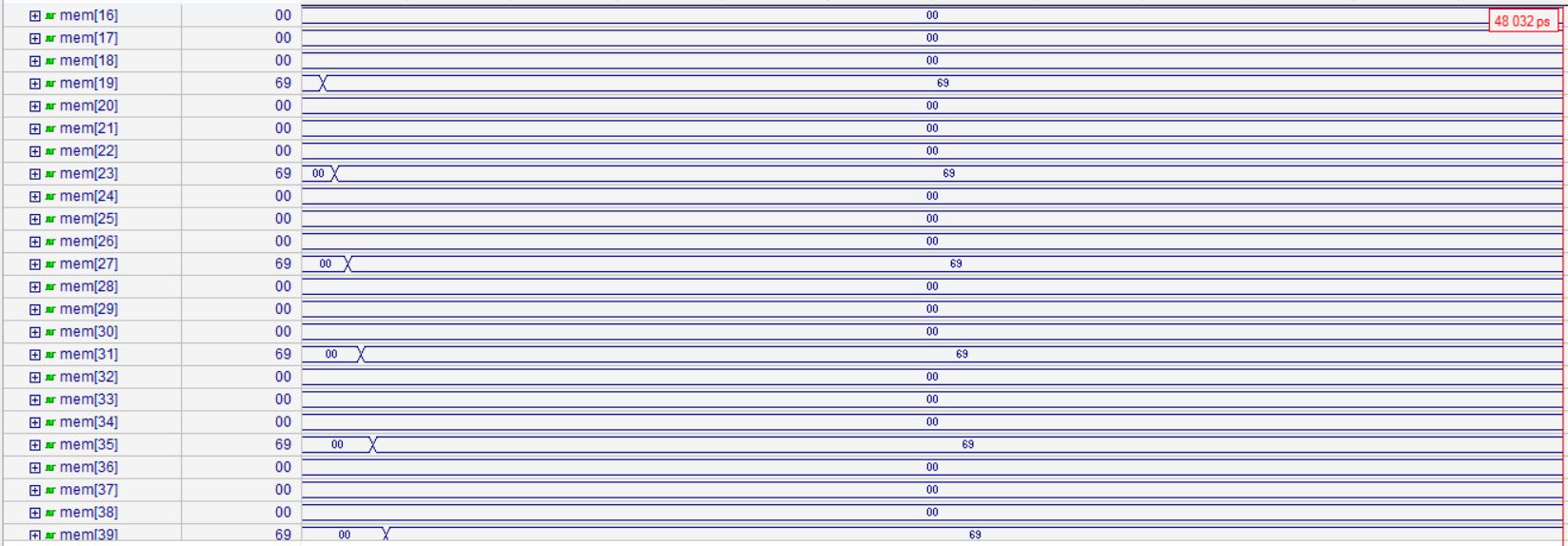
### Clock cycles taken

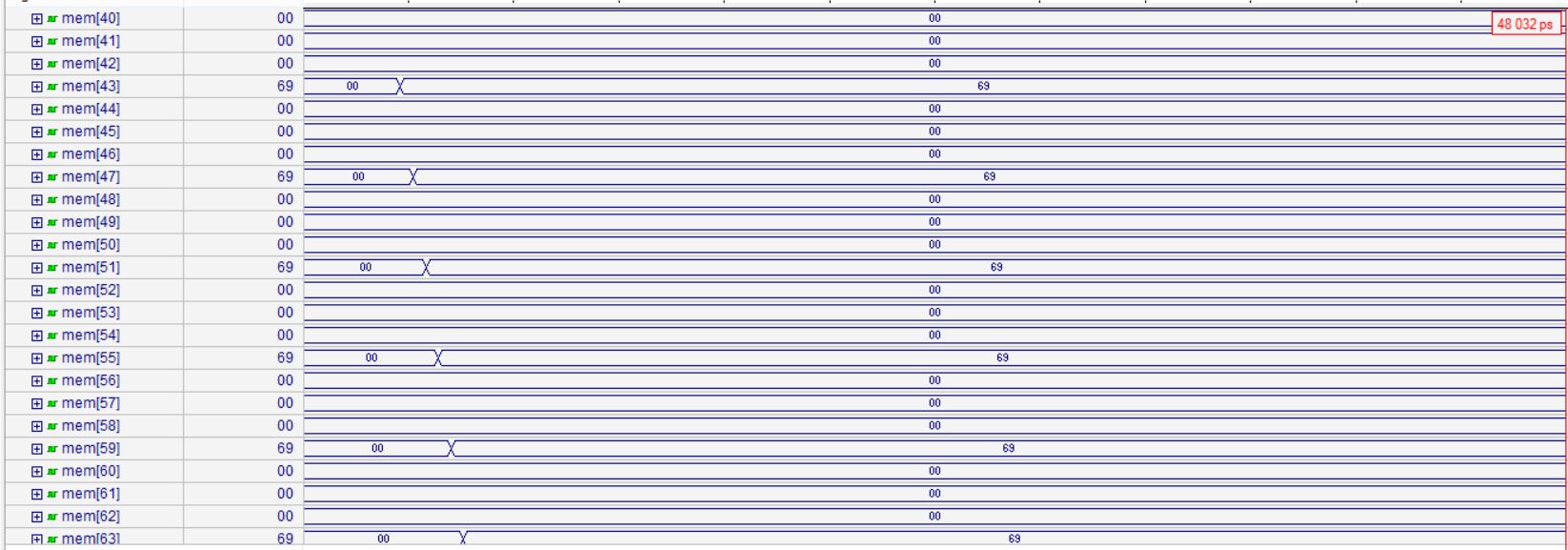
|  |
| --- |
| 799 clock cycles. |

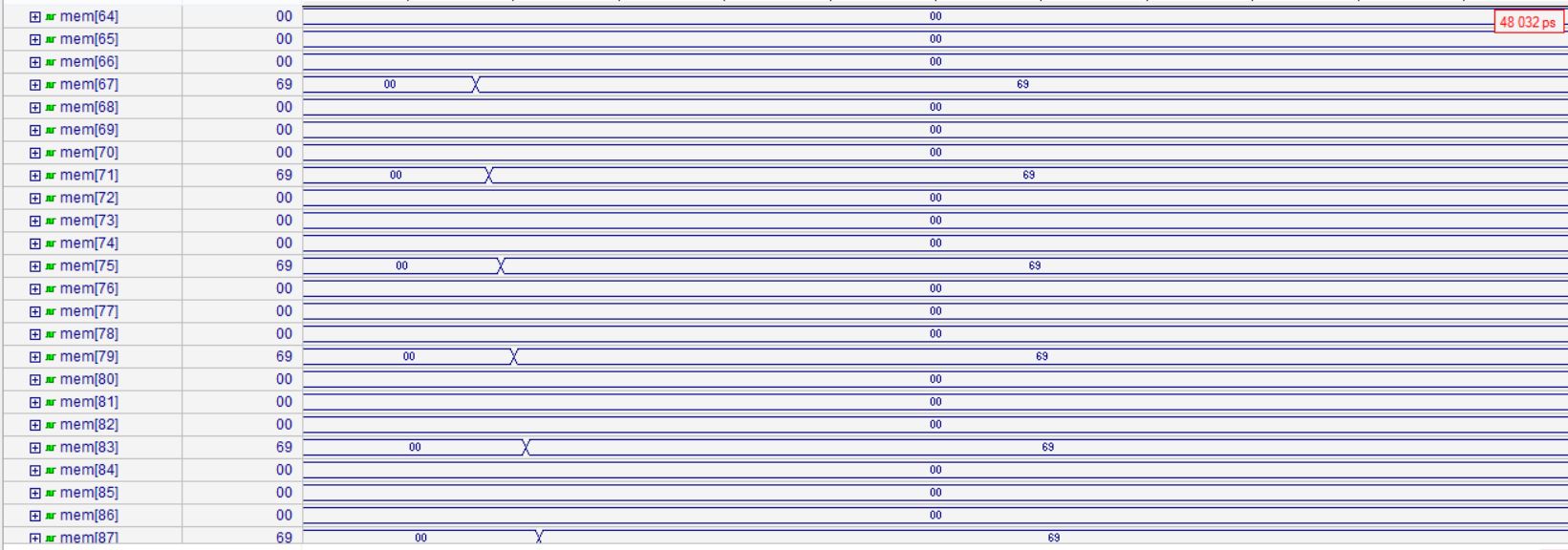
### Simulation











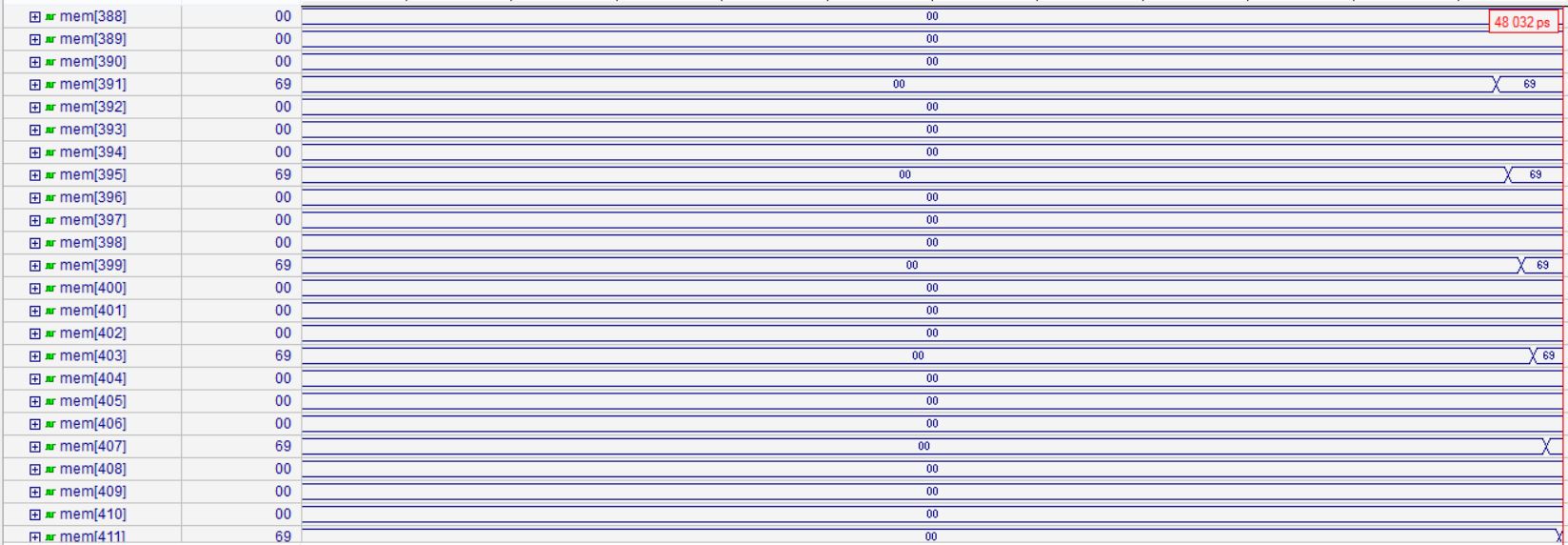
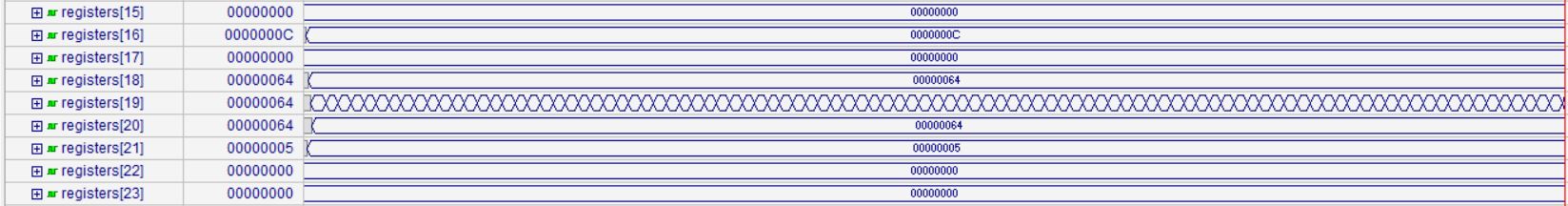
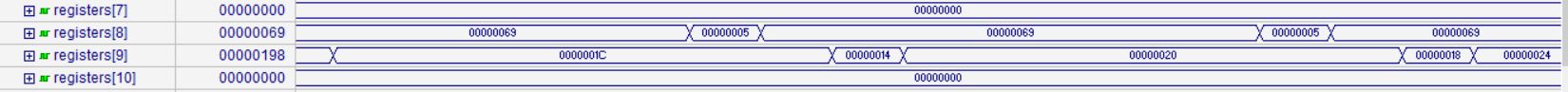


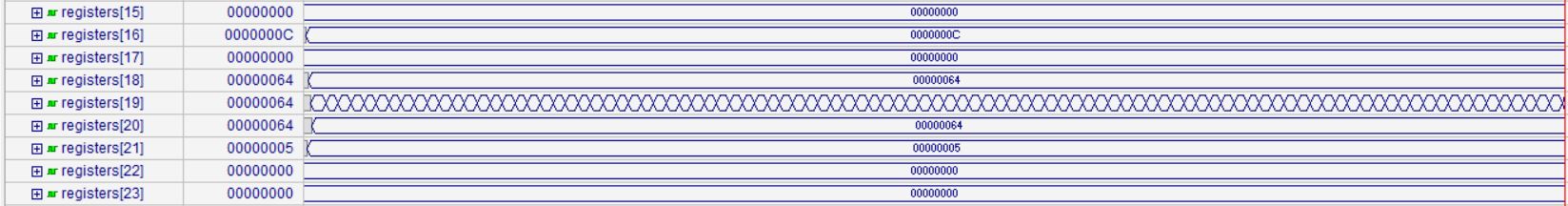
Figure 14 Program 7 Data Memory

\*the data memory contains values: location 0 contains 100 (0x64), location 12 contains 5 (0x05) (first element in the array), and the array will be stored starting from the second element from location 16 till location 408 each containing (0x69), as expected before(memory is byte addressable).











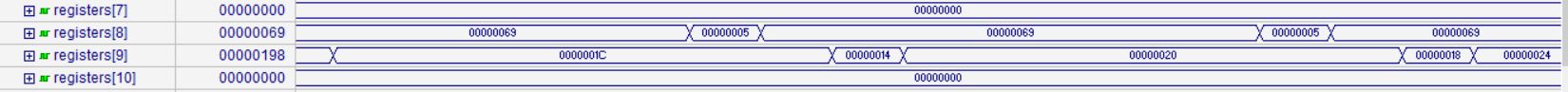


Figure 15 Program 7 Register File

## A program testing looping and summation

Description: this program initializes an array with ones then adds the elements of the array to each other.

### C++ Code

|  |
| --- |
| i=0;  sum = 0;  for(int i=0;i<10;i++)  {  sum += A[i];  } |

### Assembly Code

|  |
| --- |
| #initializing array    addi $s0, $zero, 12 #base address of A =12  addi $s5, $zero, 1 #A[0] = 1  addi $s3, $zero, 1 #i=1  addi $s4, $zero, 10    sw $s5, 0($s0) #A[0] = 1 in loc 12    loop: lw $t0, 0($s0) #$t0 = A[0] = 1  sll $t1, $s3, 2 #$t1 = i\*4  add $t1, $t1, $s0 #$t1 = i\*4 + base of A  sw $t0, 0($t1) #store 1 in loc $t1  addi $s3, $s3, 1 #i++  beq $s3, $s4, endini #if(i==10) {goto endini;}  jal loop #jump to loop    #summation    endini: addi $s0, $zero, 0 #i=0  addi $s1, $zero, 0 #sum=0  addi $s2, $zero, 12 #base address of A =12  addi $s3, $zero, 9    label: sll $t0, $s0, 2 #$t0 = i\*4  add $t0, $t0, $s2 #$t0 = i\*4 + base  lw $t0, 0($t0)  add $s1, $s1, $t0 #sum = sum + A[i]  addi $s0, $s0, 1 #i++  slt $t0, $s3, $s0  beq $t0, $zero, label |

\*We expect to find 1 in location 12, 1 in each location after that from location 16 till 48 (10 elements of the array each of value 1), the sum is not stored in memory so we expect to find it in register $s1, where register $s1 is expected to hold 10 in decimal (0xA).

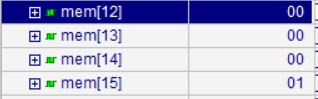
### Machine code in binary

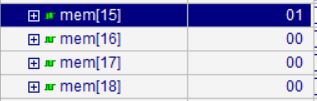
|  |
| --- |
| 00100000 00010000 00000000 00001100  00100000 00010101 00000000 00000001  00100000 00010011 00000000 00000001  00100000 00010100 00000000 00001010  10101110 00010101 00000000 00000000  10001110 00001000 00000000 00000000  00000000 00010011 01001000 10000000  00000001 00110000 01001000 00100000  10101101 00101000 00000000 00000000  00100010 01110011 00000000 00000001  00010010 10010011 00000000 00000001  00001100 00000000 00000000 00000101  00100000 00010000 00000000 00000000  00100000 00010001 00000000 00000000  00100000 00010010 00000000 00001100  00100000 00010011 00000000 00001001  00000000 00010000 01000000 10000000  00000001 00010010 01000000 00100000  10001101 00001000 00000000 00000000  00000010 00101000 10001000 00100000  00100010 00010000 00000000 00000001  00000010 01110000 01000000 00101010  00010000 00001000 11111111 11111000 |

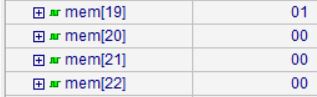
### Clock cycles taken

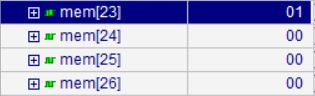
|  |
| --- |
| 150 |

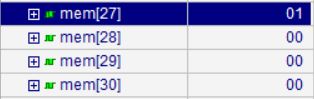
### Simulation

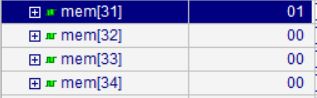


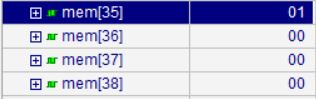


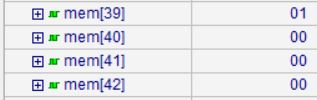


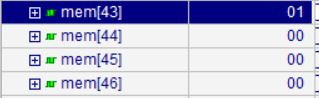


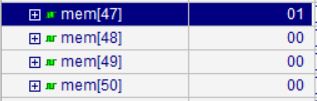












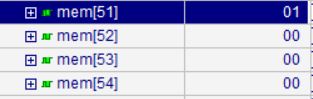


Figure 16 Program 8 Data Memory

\*The data memory is as expected before, we can see that it contains 1from location 12 till location 48.



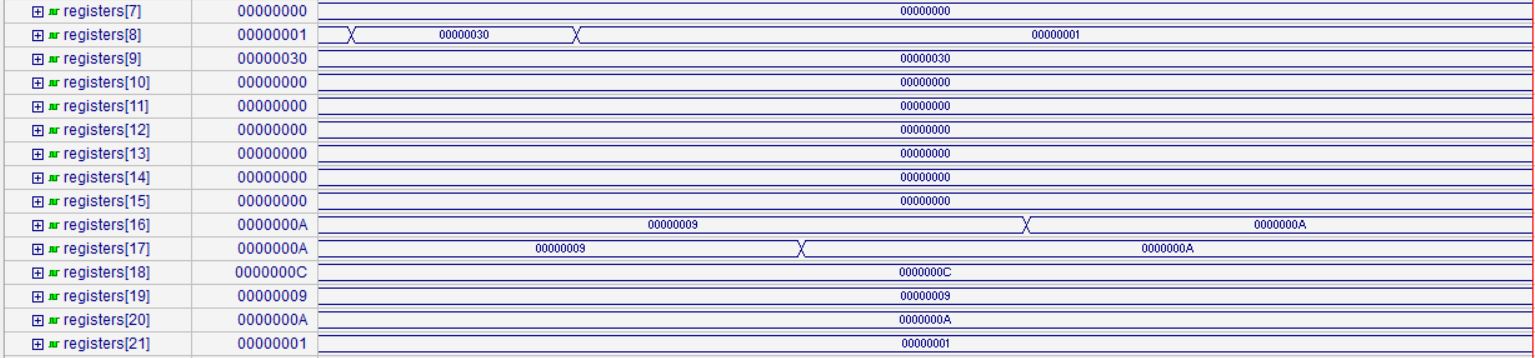


Figure 17 Program 8 Register File

\*The register file is as expected before, we can see that register $s1 (register number 17) contains 0xA which is 10 in decimal (summation of the 10 elements of the array).

## A program testing checking values and branching

Description: this program checks a value using if condition and sets another value based on this check.

### C++ Code

|  |
| --- |
| A=5;  B=5;    if(A<=B)  {  A=10;  } |

### Assembly Code

|  |
| --- |
| addi $s1, $zero, 5 #A=5  addi $s2, $zero, 5 #B=5  addi $s3, $zero, 0    slt $s0, $s2, $s1 #if(A<=B) #{$s0 = 0;}  beq $s0, $zero, label  jal exit  label: addi $s1, $zero, 10 #A=10  sw $s1, 0($s3) #loc 0 contains 10  exit: |

\*Location 0 is expected to have 10 (0x0A).

### Machine code in binary

|  |
| --- |
| 00100000 00010001 00000000 00000101  00100000 00010010 00000000 00000101  00100000 00010011 00000000 00000000  00000010 01010001 10000000 00101010  00010000 00010000 00000000 00000001  00001100 00000000 00000000 00001001  00100000 00010001 00000000 00001010  10101110 01110001 00000000 00000000 |

### Clock cycles taken

|  |
| --- |
| 7 clock cycles. |

### Simulation

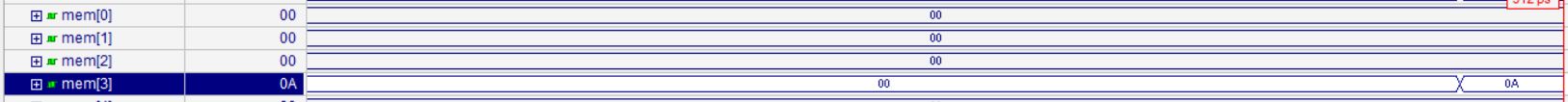
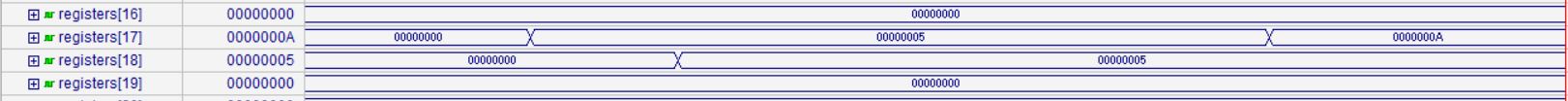


Figure 18 Program 9 Data Memory

\*Location 0 contains 0x0A as expected, (Memory is byte addressable).



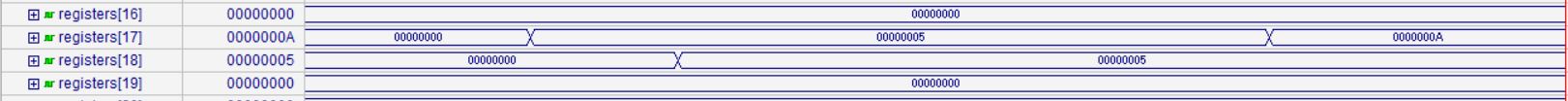


Figure 19 Program 9 Register File

## A program testing 2D arrays

### C++ Code

|  |
| --- |
| sum = 0;  for(int i = 0; i < 4; i++)  {  for(int j = 0; j < 5; j++)  {  A[i][j] = B[j][i];  sum = sum + A[i][j];  }  } |

### Assembly Code

|  |
| --- |
| #Test Program #10 - 2D arrays  .data  .word 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20  .text  addi $s0, $zero, 0 # sum = 0  addi $t0, $zero, 0 # i = 0  addi $t1, $zero, 0 # j = 0  addi $s1, $zero, 0 # for B-array starting from first memory location  addi $s2, $zero, 100 # to start saving array A from byte no. 100  addi $s3, $zero, 3  addi $s4, $zero, 4  L1: addi $t1, $zero, 0  L2: lw $t2, 0($s1)  sw $t2, 0($s2)  addi $s1, $s1, 4  addi $s2, $s2, 4  add $s0, $s0, $t2  addi $t1, $t1, 1  slt $t3, $s4, $t1  beq $t3, $zero, L2  addi $t0, $t0, 1  slt $t4, $s3, $t0  beq $t4, $zero, L1 |

Program is expected to copy values from first 20 word to words starting from 100 and keep the sum in a register.

### Initial data memory in binary

|  |
| --- |
| 00000000 00000000 00000000 00000001  00000000 00000000 00000000 00000010  00000000 00000000 00000000 00000011  00000000 00000000 00000000 00000100  00000000 00000000 00000000 00000101  00000000 00000000 00000000 00000110  00000000 00000000 00000000 00000111  00000000 00000000 00000000 00001000  00000000 00000000 00000000 00001001  00000000 00000000 00000000 00001010  00000000 00000000 00000000 00001011  00000000 00000000 00000000 00001100  00000000 00000000 00000000 00001101  00000000 00000000 00000000 00001110  00000000 00000000 00000000 00001111  00000000 00000000 00000000 00010000  00000000 00000000 00000000 00010001  00000000 00000000 00000000 00010010  00000000 00000000 00000000 00010011  00000000 00000000 00000000 00010100 |

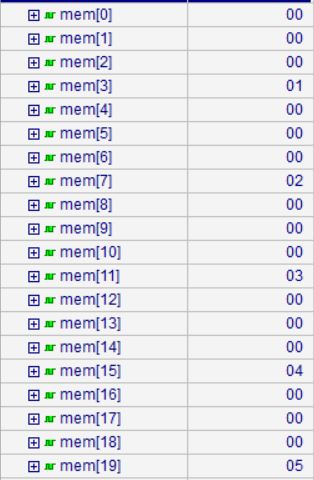
### Machine code in binary

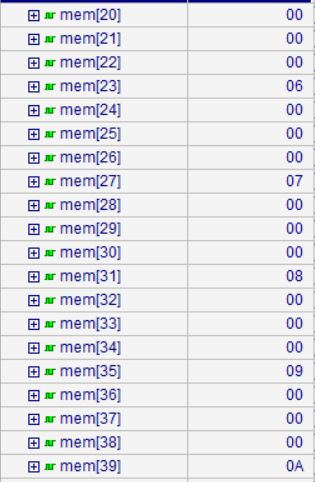
|  |
| --- |
| 00100000 00010000 00000000 00000000  00100000 00001000 00000000 00000000  00100000 00001001 00000000 00000000  00100000 00010001 00000000 00000000  00100000 00010010 00000000 01100100  00100000 00010011 00000000 00000011  00100000 00010100 00000000 00000100  00100000 00001001 00000000 00000000  10001110 00101010 00000000 00000000  10101110 01001010 00000000 00000000  00100010 00110001 00000000 00000100  00100010 01010010 00000000 00000100  00000010 00001010 10000000 00100000  00100001 00101001 00000000 00000001  00000010 10001001 01011000 00101010  00010000 00001011 11111111 11111000  00100001 00001000 00000000 00000001  00000010 01101000 01100000 00101010  00010000 00001100 11111111 11110100 |

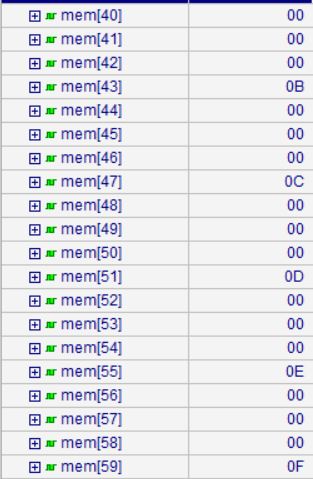
### Clock cycles taken

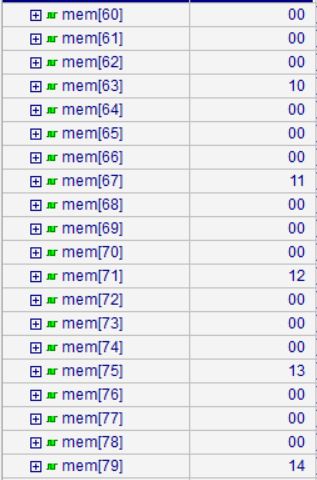
|  |
| --- |
| 183 clock cycles. |

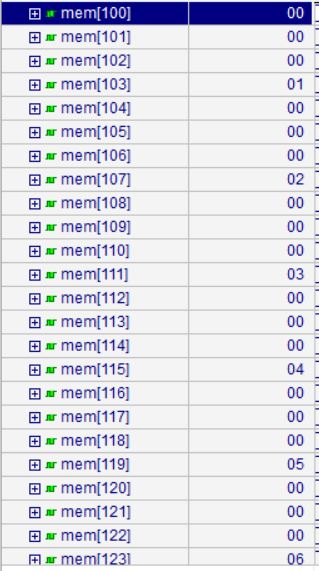
### Simulation

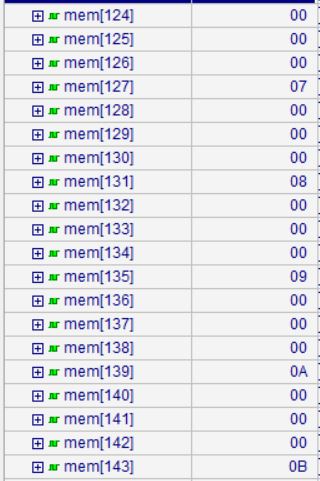


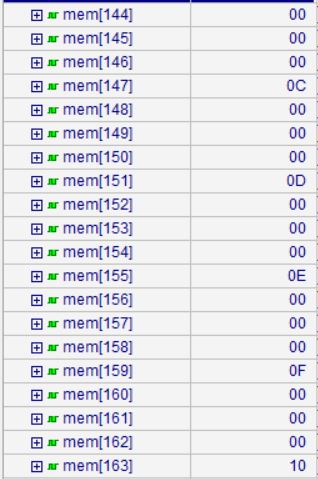












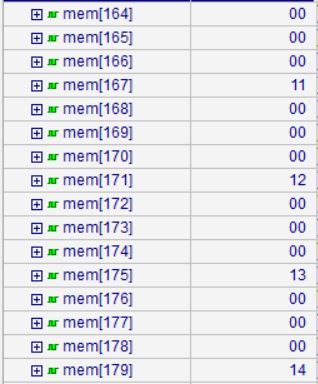


Figure 20 Program 10 Data Memory

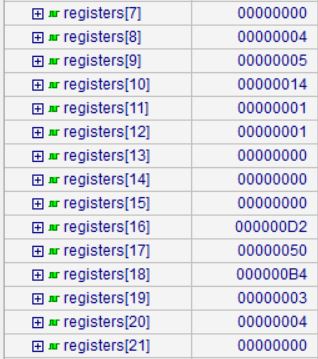


Figure 21 Program 10 Register File

register[16] is keeping the sum int(210) as expected and values in memory.

## A program testing reading initial values from memory & simple operations

### Assembly Code

|  |
| --- |
| #Test Program #11 - LW & operations  .data  .word 15, 16  .text  addi $t0, $zero, 0  addi $t1, $zero, 4  lw $s0, 0($t0)  lw $s1, 0($t1)  add $s3, $s0, $s1  nor $t3, $s0, $s1  sll $s4, $s0, 2 |

Program is expected to load 2 values from memory and in registers keep their sum, nor, and the first shifted left by 2 -multiplied by 4-.

### Initial data memory in binary

|  |
| --- |
| 00000000 00000000 00000000 00001111  00000000 00000000 00000000 00010000 |

### Machine code in binary

|  |
| --- |
| 00100000 00001000 00000000 00000000  00100000 00001001 00000000 00000100  10001101 00010000 00000000 00000000  10001101 00110001 00000000 00000000  00000010 00010001 10011000 00100000  00000010 00010001 01011000 00100111  00000000 00010000 10100000 10000000 |

### Clock cycles taken

|  |
| --- |
| 7 clock cycles. |

### Simulation

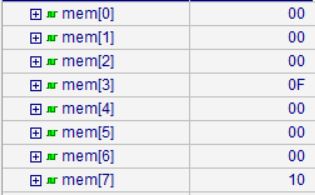


Figure 22 Program 11 Data Memory



Figure 23 Program 11 Register File

Results as expected.

## A program testing load and compare

### Assembly Code

|  |
| --- |
| #Test Program #12 - LW & compare  #pseudo code: Is b < a  .data  .word 5, 15  .text  addi $t0, $zero, 0  addi $t1, $zero, 4  lw $s0, 0($t0)  lw $s1, 0($t1)  slt $s2, $s1, $s0 |

Program is expected to load 2 values and compare them setting register $s2 by 0 if second is greater than first value, and by 1 if second is less than first.

### Initial data memory in binary

|  |
| --- |
| 00000000 00000000 00000000 00000101  00000000 00000000 00000000 00001111 |

### Machine code in binary

|  |
| --- |
| 00100000 00001000 00000000 00000000  00100000 00001001 00000000 00000100  10001101 00010000 00000000 00000000  10001101 00110001 00000000 00000000  00000010 00110000 10010000 00101010 |

### Clock cycles taken

|  |
| --- |
| 5 clock cycles. |

### Simulation

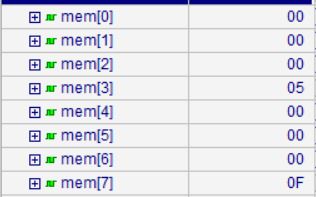


Figure 24 Program 12 Data Memory



Figure 25 Program 12 Register File

Registers values as expected after running the program.

# Contribution

Eslam Samir Ali Abo El-Ala [ID: 1200259]

* ALU Module.
* Adder, Subtractor, SltCircuit, And, Nor, and Shiftleft Modules.
* Mux and sign extender Modules.
* ALU Control Unit Module.

Mohamed Ahmed Anwer Abdelhalim [ID: 1101803]

* Instruction Memory Module
* Data Memory Module
* Assembler [Bonus]

Nourhan Essam Ahmed Shiba El-Hamd [ID: 1201605]

* Control unit Module
* Program counter Module
* 9 Test programs [Bonus]

Shaza Ismail Kaoud [ID: 1200717]

* Register file Module
* Clock Module
* Delays formatting
* 3 Test programs [Bonus]

MIPS module, report & design was collaborative effort in team meetings.

# Assembler Manual

the assembler is a simple command line tool written in Ruby and packaged to exe using “OCRA” gem.

## Usage

“assembler.exe ASSEMBLY\_FILE” **OR** “ruby assembler.rb ASSEMBLY\_FILE”

This will generate binary code file(s) in the same folder of the ASSEMBLY\_FILE

You can view the usage from the command line by calling:  
“assembler.exe” **OR** “ruby assembler.rb”

**Disclaimer: The exe package was packed for 64 bit Windows but not for 32 bit Windows, So if you encountered any errors during execution please download and install the latest stable ruby version from this** [**link**](http://rubyinstaller.org/) **and run “ruby assembler.rb ASSEMBLY\_FILE” from inside the “src/” folder**

## Supported Instructions

* Arithmetic: **add, addi, sub**
* Load/Store: **lw, sw**
* Logic: **sll, and, andi, nor**
* Control flow: **beq, jal, jr**
* Comparison: **slt**

## Supported Data Types

**.word**

## Example of Assembly File Format

|  |
| --- |
| .text  # this is an ignored comment  and $s0, $s1, $s2  label: addi $s1, $s2, -2  # another ignored comment  jal label  jr $ra # another allowed comment  sll $t0, $t1, 4  exit:  .data  .word 12, 14  .word -15 |

NOTE: labels are not supported in data segment, it will raise a syntax error

# REFERENCES

[1] “Computer Organization and Design”, 5th edition, David A. Patterson, John L. Hennessy.

[2] Lectures Slides, Dr. Cherif Salama, 2015.

[3] “Correct Methods For Adding Delays To Verilog Behavioral Models” <http://www.sunburst-design.com/papers/CummingsHDLCON1999_BehavioralDelays_Rev1_1.pdf>