

CSE 311: COMPUTER ORGANIZATION (2) MIPS PROCESSOR

Single-cycle implementation

Group: 22

# Table of Content

[1 Table of Content 1](#_Toc407056097)

[2 Table of figures 2](#_Toc407056098)

[3 Brief description 3](#_Toc407056099)

[4 MIPS map 4](#_Toc407056100)

[5 Team work responsibility distribution 5](#_Toc407056101)

[5.1 Project team members 5](#_Toc407056102)

[5.2 Team members Role 5](#_Toc407056103)

[6 User guide 7](#_Toc407056104)

[6.1 Steps to start simulation 7](#_Toc407056105)

[6.2 Restrictions 7](#_Toc407056106)

[6.3 Preferable notes 8](#_Toc407056107)

[7 Test programs step by step 10](#_Toc407056108)

[7.1 Program 1: comparing 4 numbers and get their max 10](#_Toc407056109)

[7.1.1 Step by step execution 10](#_Toc407056110)

[7.2 Program2: arithmetic operation function 34](#_Toc407056111)

[7.2.1 C-code 34](#_Toc407056112)

[7.2.2 Step by step execution 34](#_Toc407056113)

[7.3 Program 3: Array summation 55](#_Toc407056114)

[7.3.1 C-code 55](#_Toc407056115)

[7.3.2 Step by step execution 55](#_Toc407056116)

# Table of figures

[Figure 1: MIPS SINGLE-CYCLE IMPLEMENTATION 4](file:///D:\CSE%203%20projects\Comp%20org\report\MIPS%20report.docx#_Toc407044461)

[Figure 2: Register file display setting1 8](file:///D:\CSE%203%20projects\Comp%20org\report\MIPS%20report.docx#_Toc407044462)

[Figure 3: Register file display setting2 8](file:///D:\CSE%203%20projects\Comp%20org\report\MIPS%20report.docx#_Toc407044463)

# Brief description

* MIPS Processor is a single-cycle processor which has a clock cycle of 8 units.
* For the implementation we used Verilog language to describe the processor behaviorally and structurally.
* Modelsim student edition is used to simulate the MIPS.
* Binary description for the instruction format is used as an input for the MIPS and is loaded from a file under the name of “memory.list”.
* Memory supported by the MIPS is 1 byte wide and 1024 depth which is 1Kb memory.
* Data memory and instruction memory are separated modules and have the same size.
* **Bonus features :**
  + Instruction set architecture (ISA) - with bonus instructions supported - :
    - Arithmetic: add, addi, sub, mul.
    - Load/Store: lw, sw, lh, lhu, lb, lbu, sh, sb, lui.
    - Logic: sll, and, andi, or, ori, nor, srl.
    - Control flow: beq, jal, jr, bne, j.
    - Comparison: slt, slti, sltu, sltui.
  + Set of test files (17 test) are included to test all the instruction supported by the MIPS.
  + Assembler is used to compile the assembly instruction into binary code and saved directly into the input file for the MIPS.
  + C++ language and Visual studio 2010 are used to build the Assembler.
  + Program monitoring with snapshots in each clock cycle for 2 programs is included in this report.

# MIPS map

Figure 1: MIPS SINGLE-CYCLE IMPLEMENTATION

# Team work responsibility distribution

## Project team members

|  |  |
| --- | --- |
| Names | Section |
| 1. Arwa soliman 2. Ramy Mohamed Farid 3. Reem Medhat Ernest 4. Raymon Mina Yossef 5. Mohamed Ahmed Abas 6. Fady fares Abdel Ahad | 1  2  2  2  2  2 |

## Team members Role

|  |  |
| --- | --- |
| Arwa Soliman | * Responsible for the implementation of the Control unit. * Made 2 instruction programs. * Participated in MIPS testing. * Participated in MIPS report. |
| Ramy Mohamed Farid | * Responsible for the implementation of the ALU. * Responsible for designing the MIPS map including all instructions. * Made 2 instruction programs. * Participated in MIPS testing. |
| Reem Medhat Ernest | * Responsible for the implementation of Adders, Muxes, shifting units, sign extensions, PC. * Made 5 instruction programs. * Participated in MIPS merging. * Participated in MIPS testing. |
| Raymon Mina Yossef | * Responsible for the implementation of the register file. * Made 2 instruction programs. * Participated in MIPS merging. * Participated in MIPS testing. |
| Mohamed Ahmed Abas | * Responsible for the implementation of the ALU with ALU Control. * Responsible for the Assembler and comparing tools. * Made 3 instruction programs. * Participated in MIPS report. |
| Fady Fares Abdel Ahad | * Responsible for the implementation of the Instruction and Data memory. * Responsible for making the project report. * Made 3 instruction programs * Participated in MIPS testing. |

# User guide

## Steps to start simulation

1. Open “test” folder and open “Assembler.exe”.
2. Put the name of your assembly code file in the assembler to generate the binary code for you.
3. The output file is saved directly to “memory.list” in MIPS folder.
4. Binary test files are listed in “binary test codes” folder as test verification.
5. If you want to skip step from 1 to 2 you can write directly a binary code in the file “memory.list“ included in the MIPS folder ([see restrictions](#_Restrictions)).
6. Start MIPS simulation using ModelSim tool.
7. Open “MIPS.mpf” and from simulation menu press start simulation.
8. Make sure to put a proper runtime simulation then press run ([see preferable notes](#_preferable_notes)).
9. To see the results you can open the Memory List window and see the Register file memory and Data memory.
10. You can put the whole MIPS on a wave to see how it works in each clock cycle and the whole MIPS signals accompanied with the instruction.

## Restrictions

1. Writing in “memory.list”:
   * Must be in Binary format only.
   * 8 bits per line.
   * No spaces or extra enters are allowed.
   * Underscore and comments are allowed.
2. Using the Assembler:
   * You can write comments in your assembly code as they are neglected by the assembler. The comment must begin with double slash “//” and it’s a line comment, multiple lines are not supported.
   * The assembler is case insensitive so it can’t differentiate between the word LOOP and loop. Remember not to use the same words to jump to different instructions.
   * Any spaces are neglected by the assembler.
   * You have to write one instruction per line.
   * Supported instructions by the assembler are:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Instruction | Op code | Instruction | Op code | Instruction | Op code | Instruction | Op code | Instruction | Op code |
| Add | 000000 | Sub | 000000 | Addi | 001000 | Mul | 000000 | And | 000000 |
| Srl | 000000 | Slt | 000000 | Sltu | 000000 | Slti | 001010 | Sltiu | 001001 |
| Lui | 001111 | Lb | 100000 | Lbu | 100100 | Lh | 100001 | Lhu | 100101 |
| Or | 000000 | Nor | 000000 | Andi | 001100 | Ori | 001101 | Sll | 000000 |
| J | 000010 | Jal | 000011 | Jr | 001000 | Beq | 000100 | Bne | 000101 |
| Lw | 100011 | Sb | 101000 | Sh | 101001 | sw | 101011 |  |  |

Table 1: instruction opcode supported

* Immediate operands must be in decimal.
* Any errors the assembler won’t write the file and an error message will be displayed in the console window.

1. Using the ALU
   * Multiplication using “Mul” must have a result of 32bits or overflow will happen.
   * Addition or subtraction using “add” or “sub” must have 31bits operands or carry bit will be lost.
   * Any immediate instruction must take 16bits immediate operand.
   * 32'b0 instruction is Nop.

## Preferable notes

1. Use proper simulation time=8\*no. of instructions +2.
2. Data memory and register file are initially unknown.
3. For register file display:

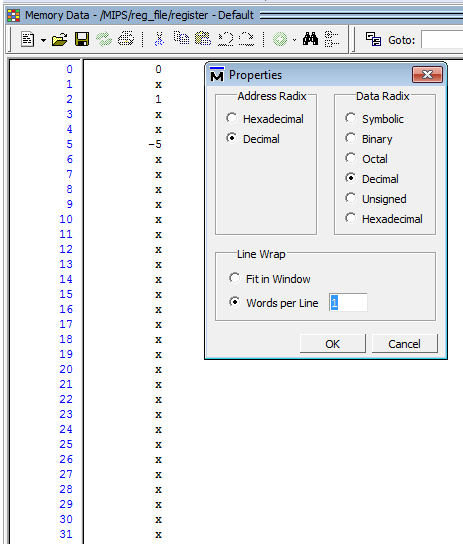
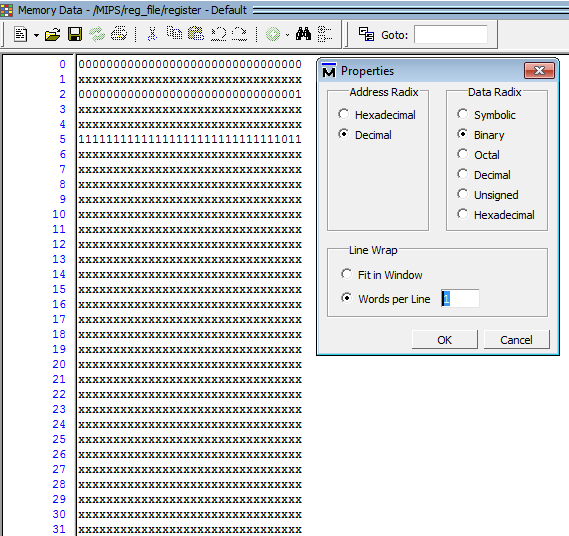
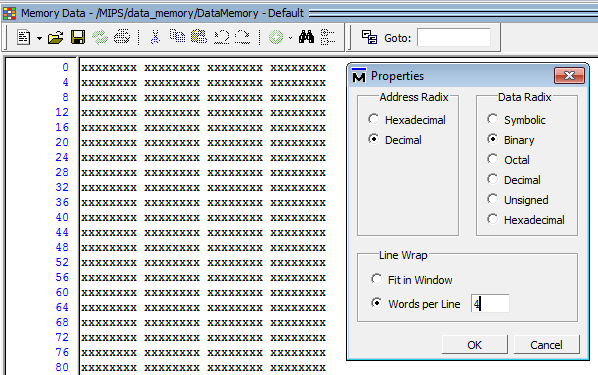


Figure 2: Register file display setting1

Or

Figure 3: Register file display setting2

1. For Data memory display:



# Test programs step by step

## Program 1: comparing 4 numbers and get their max

* Initially $s2=20, $s3=50, $s4=200.
* The program find their max ($s4) and put its value in $s7.
* Compare result in $s7 with immediate operand 100 and put the bigger in $t0.
* Final state of register used : $t0=200, s1=1, $s2=20, $s3=50, $s4=200,$s5=1, $s6=1, $s7=200

### Step by step execution

|  |  |  |
| --- | --- | --- |
| **Clock cycle** | **MIPS signals** | **Register file output** |
| Before run |  |  |
| 0 |  |  |
| 1 |  |  |
| 2 |  |  |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |
| 10 |  |  |
| 11 |  |  |
| 12 |  |  |
| 13 |  |  |
| 14 |  |  |
| 15 |  |  |
| 16 |  |  |
| 17 |  |  |
| 18 |  |  |
| 19 |  |  |
| 20 |  |  |
| 21 |  |  |
| 22  “Program end” |  |  |

## Program2: arithmetic operation function

### C-code

Int g=6;

Int h=4;

Int i=2;

Int j=3;

int f= 10 + proc (g,h,i,j);

int proc (int g, int h, int i , int j){

int f;

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

return f;

}

Final state $v0 will be 15.

### Step by step execution

|  |  |
| --- | --- |
| Clock cycle | MIPS signals and Register file |
| 0 |  |
| 1 |  |
| 2 |  |
| 3 |  |
| 4 |  |
| 5 |  |
| 6 |  |
| 7 |  |
| 8 |  |
| 9 |  |
| 10 |  |
| 11 |  |
| 12 |  |
| 13 |  |
| 14 |  |
| 15 |  |
| 16 |  |
| 17 |  |
| 18 |  |
| 19 |  |

## Program 3: Array summation

### C-code

int sum (int A,int n)

{ // where A is an array

int i,s;

s=0;

for (int i=0;i<n;i++)

s+=A[i];

return s;

}

Final state $v0 will be 27.

### Step by step execution

Snap shot are taken for each clock cycle see folder “Array summation”.