EGRE 426 – Lab 4 Week 1 Report

Due: October 22, 2025, 11:59 PM

Team Members:

Macarios Atia

Ansh Patel

**1. Objective**

The goal of Week 1 is to design the instruction set and define the components and make sure that the control unit table for a 16-bit single-cycle RISC processor works. This design forms the basis for our CPU where the CPU will be simulated and integrated within our code.

**2. Instruction Set Architecture (ISA) Design**

**Design Overview**

|  |
| --- |
| **Word Size and Memory**: 16-bit machine = 512 bytes. PC starts at 0x0000 and increments by 2 each instruction (half-word aligned) |
| **Clocking:** Synchronous, rising edge triggered for the register file and PC. |
| **Register File** 8 general purpose registers ($r0-$r7), 16-bit each. Read happens on the falling edge and write |

**2.1 Instruction Formats**

|  |  |  |
| --- | --- | --- |
| Type | Format | Description |
| R-Type | Opcode (4 bits)] Rs (4 bits) Rt (4 bits)] [Rd (4 bits) | arithmetic/logic |
| I-Type | Opcode (4 bits)] [Rs (4 bits)] [Rt (4 bits) Immediate (4 bits) | immediate or memory access |
| J-Type | Opcode (4 bits) Address (12 bits) | for jumps |

**2.2 Instructions**

A table with numbers and symbols

AI-generated content may be incorrect.

**2.3 Assembly language and machine code for the test program (Pseudocode)**

Pseudocode for the test program (refer to the project handout):

while ($a1 > 0) {

$a1 = $a1 –1;

$t0 = Mem[$a0];

if ($t0 > 0100hex) then {

$v0 = $v0 ÷ 8;

$v1 = $v1 | $v0; //or

Mem[$a0] = FF00hex;

}

else {

$v2 = $v2 × 4;

$v3 = $v3 ⊕ $v2; //xor

Mem[$a0] = 00FFhex;

}

$a0 = $a0 + 2;

}

return;

|  |  |
| --- | --- |
| Assembly Language | Machine Code (Hex -> Formatted binary -> binary |
| addi $r2, $r2, -1; | 0x34BF → 0011 010 010 101111 → 0011 0100 1010 1111 |
| lw $r3, 0($r1); | 0x10C0 → 0001 001 011 000000 → 0001 0010 1100 0000 |
| lw $r6, 48($r0); | 0x118C → 0001 000 110 110000 → 0001 0001 1011 0000 |
| bgt $r3, $r6, +5; | 0x58A5 → 0101 011 110 000101 → 0101 1011 1100 0101 |
| sll $r6, $r6, 2; | 0x0D90 → 0000 110 110 110 100 → 0000 1101 1011 0100 |
| xor $r7, $r7, $r6; | 0x0FEF → 0000 111 110 111 111 → 0000 1111 1110 1111 |
| lw $r3, 52($r0); | 0x11D0 → 0001 000 011 110100 → 0001 0000 1111 0100 |
| sw $r3, 0($r1); | 0x20C0 → 0010 001 011 000000 → 0010 0010 1100 0000 |
| j ENDIF | 0x801E → 1000 000000011110 → 1000 0000 0001 1110 |
| srl $r4, $r4, 1; | 0x0D85 → 0000 100 100 100 101 → 0000 1001 0010 0101 |
| srl $r4, $r4, 1; | 0x0D85 → 0000 100 100 100 101 → 0000 1001 0010 0101 |
| srl $r4, $r4, 1; | 0x0D85 → 0000 100 100 100 101 → 0000 1001 0010 0101 |
| or $r5, $r5, $r4; | 0x0B6B → 0000 101 100 101 011 → 0000101100101011 |
| lw $r3, 50($r0); | 0x11B8 → 0001 000 011 110010 → 0001 0000 1111 0010 |
| sw $r3, 0($r1); | 0x20C0 → 0010 001 011 000000 → 0010 0010 1100 0000 |
| addi $r1, $r1, 2; | 0x3902 → 0011 100 100 000010 → 0011 1001 0000 0010 |
| bgt $r2, $r0, -17; | 0x541F → 0101 010 000 101111 → 0101 0100 0001 0111 |
| return (no encoding) ation) | — |

**3. Instruction and Control Signal Table**

A table with numbers and symbols

AI-generated content may be incorrect.

**4. Datapath Components and Description**

|  |  |  |
| --- | --- | --- |
| Component | Function | Notes |
| Program Counter (PC) | Holds address of next instruction | 16-bit register, increments by 2 |
| Instruction Memory | Stores machine instructions | 512 bytes total |
| Register File | Holds operands and results | 8 registers, each 16 bits |
| ALU | Performs arithmetic/logic operations | Supports add, sub, and, or, xor, slt |
| Data Memory | Stores data for lw/sw instructions | Addressed by ALU output |
| Multiplexers | Select data/control sources | Used for reg\_dst, alu\_src, mem\_to\_reg |
| Sign Extender | Extends immediate fields | 4 → 16 bits |
| Control Unit | make control signals based on opcode | Controls datapath flow |

**5. Register Definitions**

|  |  |  |  |
| --- | --- | --- | --- |
| Register | Symbol | Description | Size |
| R0 | $v0 | General-purpose (result register) | 16 bits |
| R1 | $v1 | General-purpose | 16 bits |
| R2 | $v2 | General-purpose | 16 bits |
| R3 | $v3 | General-purpose | 16 bits |
| R4 | $t0 | Temporary data register | 16 bits |
| R5 | $a0 | Address/data register | 16 bits |
| R6 | $a1 | Loop counter | 16 bits |
| R7 | R | Reserved register | 16 bits |

**6. Variable Initialization (from Pseudocode)**

|  |  |  |  |
| --- | --- | --- | --- |
| Variable | Register | Initial Value (hex) | Description |
| $v0 | R0 | 0040 | Used for division and OR operations |
| $v1 | R1 | 1010 | Used in OR logic |
| $v2 | R2 | 000F | Multiplied by 4 during loop |
| $v3 | R3 | 00F0 | XORed with $v2 |
| $t0 | R4 | 0000 | Temporary memory holder |
| $a0 | R5 | 0010 | Memory pointer (increments by 2) |
| $a1 | R6 | 0005 | Loop counter (decrements each iteration) |

**7. Memory Initialization**

|  |  |  |
| --- | --- | --- |
| Address | Data (hex) | Description |
| $a0 | 0101 | Data element 1 |
| $a0 + 2 | 0110 | Data element 2 |
| $a0 + 4 | 0011 | Data element 3 |
| $a0 + 6 | 00F0 | Data element 4 |
| $a0 + 8 | 00FF | Data element 5 |

**8. Control Unit Inputs and Outputs**

|  |  |
| --- | --- |
| Input | Description |
| Opcode [15–12] | instruction type |
| Output | Description |
| reg\_dst | Select destination register |
| jump | Jump control |
| branch | Branch control |
| mem\_read | Enables data memory read |
| mem\_to\_reg | Selects memory data for write-back |
| ALU\_OP | Finds ALU 5function |
| mem\_write | Enables data memory write |
| alu\_src | Selects ALU input (register or immediate) |
| reg\_write | Enables register file write |

**9. Summary**

The instruction formats, Datapath components, and control signals were fully defined. All registers, memory initializations, and variable mappings were clearly shown above in the tables. The setup we have right now makes sure that simulator and control logic can be implemented and tested in Week which should give us enough time to work on the pipelining of our CPU with the buffers added.