**CS321 Lab12**

**Single Cycle CPU using HDL(Verilog)**

A microarchitecture that executes instructions in a single cycle is shown in figure. The structure shows the datapath , connecting the state elements with combinational logic that can execute the various instructions. Control signals determine which specific instruction is carried out by the datapath at any given time. The controller contains combinational logic that generates the appropriate control signals based on the current instruction.



**Reference: Digital Design and Computer Architecture-** [**David Harris**](http://portal.acm.org/author_page.cfm?id=81100117009&coll=DL&dl=ACM&trk=0&cfid=111188109&cftoken=31800392)[**Sarah Harris**](http://portal.acm.org/author_page.cfm?id=81100116734&coll=DL&dl=ACM&trk=0&cfid=111188109&cftoken=31800392)

**Your Assignment 12 (50 points)**

**Submission** Your submission must contain:



Study the above architecture and simulate using modelsim and then write a brief report on the architecture.

Report should contain timing diagram of all signals related to a given instruction. Simulate (add, sub , LW , SW, BEQ, ADDI, J , etc).

Implement two new instruction and prepare a brief report including timing diagram.

**Appendix: Encoding of the Sample Machine Code**

|  |  |
| --- | --- |
| Description: | Adds a register and a sign-extended immediate value and stores the result in a register |
| Operation: | $t = $s + imm; advance\_pc (4); |
| Syntax: | addi $t, $s, imm |
| Encoding: | 0010 00ss ssst tttt iiii iiii iiii iiii |

### 

addi $t, $s, imm

Addi $2 $0 5

**0010 00 0000 0 00010 0005 = 2002 0005 (hex)**

Addi $3 $0 12

**0010 00 0000 0 00011 0005 = 2003 000c (hex)**

Addi $7 $3 12

**0010 00 00011 00111 1111 1111 1111 0111 = 2067 fffc (hex)**

### OR -- *Bitwise or*

|  |  |
| --- | --- |
| Description: | Bitwise logical ors two registers and stores the result in a register |
| Operation: | $d = $s | $t; advance\_pc (4); |
| Syntax: | or $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0101 |

### or $d, $s, $t

**0r $4, $7, $2**

0000 00ss ssst tttt dddd d000 0010 0101

0000 00**00 111**0 0010 0010 0000 0010 0101 =00e22025

And $5, $3, $4

### AND -- *Bitwise and*

|  |  |
| --- | --- |
| Description: | Bitwise ands two registers and stores the result in a register |
| Operation: | $d = $s & $t; advance\_pc (4); |
| Syntax: | and $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0100 |

**and $d, $s, $t**

**and $5, $3, $4**

0000 0000 0110 0100 0010 1000 0010 0100 = 00642824 (hex)

|  |  |
| --- | --- |
| Description: | Adds two registers and stores the result in a register |
| Operation: | $d = $s + $t; advance\_pc (4); |
| Syntax: | add $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0000 |

### 

add $d, $s, $t

add $5 $5 $4

0000 0000 1010 0100 0010 1000 0010 0000 =00a42820

|  |  |
| --- | --- |
| Description: | Branches if the two registers are equal |
| Operation: | if $s == $t advance\_pc (offset << 2)); else advance\_pc (4); |
| Syntax: | beq $s, $t, offset |
| Encoding: | 0001 00ss ssst tttt iiii iiii iiii iiii |

beq $s, $t, offset

beq $5, $7, 44

**0001 0000 1010 0111 0000 0000 0000 1010 = 10a7000a (hex)**

### SLT -- *Set on less than*

|  |  |
| --- | --- |
| Description: | If $s is less than $t, $d is set to one. It gets zero otherwise. |
| Operation: | if $s < $t $d = 1; advance\_pc (4); else $d = 0; advance\_pc (4); |
| Syntax: | slt $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 1010 |

### slt $d, $s, $t

### slt $4, $3, $4

0000 0000 0110 0100 0010 0000 0010 1010 =0064202a (hex)

beq $s, $t, offset

beq $4, $0, round

**0001 0000 1000 0000 0000 0000 0000 0001 = 10800001 (hex)**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

addi $t, $s, imm

Addi $5 $0 0

**0010 0000 0000 0101 0000 0000 0000 0000 = 20050000 (hex)**

### slt $d, $s, $t

### slt $4, $7, $2

|  |  |
| --- | --- |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 1010 |

|  |  |
| --- | --- |
| Encoding: | 0000 0000 1110 0010 0010 0000 0010 1010 |

=00e2202a( hex)

add $d, $s, $t

add $7 $4 $5

|  |  |
| --- | --- |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0000 |

### 

|  |  |
| --- | --- |
| Encoding: | 0000 0000 1000 0101 0011 1000 0010 0000 |

### =00853820 (hex)

### Subtract

|  |  |
| --- | --- |
| Description: | Subtracts two registers and stores the result in a register |
| Operation: | $d = $s - $t; advance\_pc (4); |
| Syntax: | sub $d, $s, $t |
| Encoding: | 0000 00ss ssst tttt dddd d000 0010 0010 |

sub $d, $s, $t

sub $7, $7, $2

|  |  |
| --- | --- |
| Encoding: | 0000 0000 1110 0010 0011 1000 0010 0010 |

=00e23822 (hex)

**Store word**

|  |  |
| --- | --- |
| Description: | The contents of $t is stored at the specified address. |
| Operation: | MEM[$s + offset] = $t; advance\_pc (4); |
| Syntax: | sw $t, offset($s) |
| Encoding: | 1010 11ss ssst tttt iiii iiii iiii iiii |

sw $t, offset($s)

sw $7, 68($3)

|  |  |
| --- | --- |
| Encoding: | 1010 1100 0110 0111 0000 0000 0100 0100 |

=ac670044 (hex)

### LW -- *Load word*

|  |  |
| --- | --- |
| Description: | A word is loaded into a register from the specified address. |
| Operation: | $t = MEM[$s + offset]; advance\_pc (4); |
| Syntax: | lw $t, offset($s) |
| Encoding: | 1000 11ss ssst tttt iiii iiii iiii iiii |

lw $t, offset($s)

lw $2, 80($0)

|  |  |
| --- | --- |
| Encoding: | 1000 11ss ssst tttt iiii iiii iiii iiii |

|  |  |
| --- | --- |
| Encoding: | 1000 1100 0000 0010 0000 0000 0101 0000 |

=8c020050 (hex)

### J -- *Jump*

|  |  |
| --- | --- |
| Description: | Jumps to the calculated address |
| Operation: | PC = nPC; nPC = (PC & 0xf0000000) | (target << 2); |
| Syntax: | j target |
| Encoding: | 0000 10ii iiii iiii iiii iiii iiii iiii |

j target

j end

|  |  |
| --- | --- |
| Encoding: | 0000 1000 0000 0000 0000 0000 0001 0001 |

=08000011 (hex)

|  |  |
| --- | --- |
| Encoding: | 0010 00ss ssst tttt iiii iiii iiii iiii |

### 

addi $t, $s, imm

Addi $2 $0 1

|  |  |
| --- | --- |
| Encoding: | 0010 0000 0000 0010 iiii iiii iiii iiii |

**Store word**

|  |  |
| --- | --- |
| Description: | The contents of $t is stored at the specified address. |
| Operation: | MEM[$s + offset] = $t; advance\_pc (4); |
| Syntax: | sw $t, offset($s) |
| Encoding: | 1010 11ss ssst tttt iiii iiii iiii iiii |

sw $t, offset($s)

sw $2, 84($0)

|  |  |
| --- | --- |
| Encoding: | 1010 1100 0000 0010 0000 0000 0101 0100 |

=ac020054 (hex)