컴퓨터구조 프로젝트 보고서

Project 1

MIPS Single Cycle CPU Implementation

수업 명: 컴퓨터구조

담당 교수: 이성원 교수님

학과: 컴퓨터정보공학부

학번: 2023202070

이름: 최현진

제출일: 2025.04.

1. Introduction

기본 명령어 lw, sw, ori, j, lui, llo, lhi는 구현되어 있고, 추가로 AND, NOR, ADDI, SLTU, SRL, SH, LB, BNE, BGEZ, JALR을 설계해야 한다. 이를 위해 PLA(Programmable Logic array) 구조를 이용해서 명령어를 해석하고, 제어 신호를 생성해야 한다. PLA\_AND.txt에는 각 명령어에 대해 opcode, func, regimm를 설정한다. PLA\_OR.txt에는 설정한 줄에 맞춰 해당 명령이 작동할 제어 신호를 채워 넣는다.

2. Assignment

|  |
| --- |
| <초기 M\_TEXT\_SEG.txt>  001111\_00000\_00010\_0001001000110100 // lui $2, 0x1234  001101\_00010\_00011\_0101011001111000 // ori $3, $2, 0x5678  001111\_00000\_00100\_0001000100100010 // lui $4, 0x1122  001101\_00100\_00101\_0011001101000100 // ori $5, $4, 0x3344  <실행 결과>  $2 = 0x12340000  $3 = $2 | 0x5678 = 0x12340000 | 0x00005678 = 0x12345678  $4 = 0x11220000  $5 = 0x11220000 | 0x00003344 = 0x11223344 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. AND  두 레지스터의 비트 단위 AND 연산 결과를 레지스터에 저장한다, R-type  Syntax: f $d, $s, $t  Operation: $d = $s & $t   |  |  |  | | --- | --- | --- | | Op | Func | Regimm | | 000000 | 100100 | xxxxx |   000000\_100100\_xxxxx // 0x15 : and   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | RegDst | RegDatSel | RegWrite | SEUmode | ALUsrcB | ALUctrl | | 01 | 00 | 1 | x | 00 | 0x | | ALUop | DataWidth | MemWrite | MemtoReg | Branch | Jump | | 00000 | xxx | 0 | 0 | 000 | 00 |   01\_00\_1\_x\_00\_0x\_00000\_xxx\_0\_0\_000\_00\_xxxxx // 0x15 : and $d = $s & $t  RegDst: 01, 결과값을 rd에 쓸 것이므로 목적지 레지스터로 rd를 선택한다.  RegDatSel: 00, ALU 결과값을 register file에 쓰도록 선택한다.  RegWrite: 1, register file에 값을 쓴다.  SEUmode: x, imm값을 extension할 필요가 없다.  ALUsrcB: 00, ALU 입력으로 register file port B를 사용한다.  ALUctrl: 00, ALUctrl[1]=0 (normal ALU input), ALUctrl[0]= x (shift 수행하지 않음)  ALUop: 00000, bitwise AND  DataWidth: xxx, 메모리 접근하지 않음.  MemWrite: 0, 메모리에 쓰지 않음.  MemtoReg: 0, register file에 ALU값을 쓴다.  Branch: 000, branch 수행하지 않음.  Jump: 00, jump 수행하지 않음. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2. NOR  두 레지스터의 비트 단위 NOR 연산 결과를 레지스터에 저장한다, R-type  Syntax: f $d, $s, $t  Operation: $d = ~($s | $t)   |  |  |  | | --- | --- | --- | | Op | Func | Regimm | | 000000 | 100111 | xxxxx |   000000\_100111\_xxxxx // 0x18 : nor   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | RegDst | RegDatSel | RegWrite | SEUmode | ALUsrcB | ALUctrl | | 01 | 00 | 1 | x | 00 | 0x | | ALUop | DataWidth | MemWrite | MemtoReg | Branch | Jump | | 00010 | xxx | 0 | 0 | 000 | 00 |   01\_00\_1\_x\_00\_0x\_00010\_xxx\_0\_0\_000\_00\_xxxxx // 0x18 : nor $d = ~($s | $t)  RegDst: 01, 결과값을 rd에 쓸 것이므로 목적지 레지스터로 rd를 선택한다.  RegDatSel: 00, ALU 결과값을 register file에 쓰도록 선택한다.  RegWrite: 1, register file에 값을 쓴다.  SEUmode: x, imm값을 extension할 필요가 없다.  ALUsrcB: 00, ALU 입력으로 register file port B를 사용한다.  ALUctrl: ALUctrl[1]=0 (normal ALU input), ALUctrl[0]= x (shift 수행하지 않음)  ALUop: 00010, bitwise NOR  DataWidth: xxx, 메모리 접근하지 않음.  MemWrite: 0, 메모리에 쓰지 않음.  MemtoReg: 0, register file에 ALU값을 쓴다.  Branch: 000, branch 수행하지 않음.  Jump: 00, jump 수행하지 않음. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 3. ADDI  imm값을을 sign extend하여 $s와 더한 결과를 $t에 저장한다, I-type  Syntax: o $t, $s, i  Operation: $t = $s + SE(i)   |  |  |  | | --- | --- | --- | | Op | Func | Regimm | | 001000 | xxxxxx | xxxxx |   001000\_xxxxxx\_xxxxx // 0x23 : addi   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | RegDst | RegDatSel | RegWrite | SEUmode | ALUsrcB | ALUctrl | | 00 | 00 | 1 | 1 | 01 | 0x | | ALUop | DataWidth | MemWrite | MemtoReg | Branch | Jump | | 00100 | xxx | 0 | 0 | 000 | 00 |   00\_00\_1\_1\_01\_0x\_00100\_xxx\_0\_0\_000\_00\_xxxxx // 0x23 : addi $t = $s + SE(i)  RegDst: 00, 결과값을 rt에 쓸 것이므로 목적지 레지스터로 rt를 선택한다.  RegDatSel: 00, ALU 결과값을 register file에 쓰도록 선택한다.  RegWrite: 1, register file에 값을 쓴다.  SEUmode: 1, imm값을 sign extension한다.  ALUsrcB: 01, ALU 입력으로 imm값을 사용한다.  ALUctrl: 00, ALUctrl[1]=0 (normal ALU input), ALUctrl[0]=x (shift 수행하지 않음)  ALUop: 00100, a + b  DataWidth: xxx, 메모리 접근하지 않음.  MemWrite: 0, 메모리에 쓰지 않음.  MemtoReg: 0, register file에 ALU값을 쓴다.  Branch: 000, branch 수행하지 않음.  Jump: 00, jump 수행하지 않음. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 4. SLTU  Set Less Than Unsigned, unsigned 기준으로 $s < $t 비교 결과를 $d에 저장한다, R-type  Syntax: f $d, $s, $t  Operation: $d = ($s < $t)   |  |  |  | | --- | --- | --- | | Op | Func | Regimm | | 000000 | 101011 | xxxxx |   000000\_101011\_xxxxx // 0x1a : sltu   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | RegDst | RegDatSel | RegWrite | SEUmode | ALUsrcB | ALUctrl | | 01 | 00 | 1 | x | 00 | 0x | | ALUop | DataWidth | MemWrite | MemtoReg | Branch | Jump | | 10001 | xxx | 0 | 0 | 000 | 00 |   01\_00\_1\_x\_00\_0x\_10001\_xxx\_0\_0\_000\_00\_xxxxx // 0x1a : sltu $d = ($s < $t)  RegDst: 01, 결과값을 rd에 쓸 것이므로 목적지 레지스터로 rd를 선택한다.  RegDatSel: 00, ALU 결과값을 register file에 쓰도록 선택한다.  RegWrite: 1, register file에 값을 쓴다.  SEUmode: x, imm값을 extension할 필요가 없다.  ALUsrcB: 00, ALU 입력으로 register file port B를 사용한다.  ALUctrl: ALUctrl[1]=0 (normal ALU input), ALUctrl[0]=x (shift 수행하지 않음)  ALUop: 10001, Unsigned SLT  DataWidth: xxx, 메모리 접근하지 않음.  MemWrite: 0, 메모리에 쓰지 않음.  MemtoReg: 0, register file에 ALU값을 쓴다.  Branch: 000, branch 수행하지 않음.  Jump: 00, jump 수행하지 않음. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 5. SRL  Shift Right Logical: shamt만큼 logical shift한 값을 레지스터에 저장한다. R-type  Syntax: f $d, $t, sa  Operation: $d = $t >> a   |  |  |  | | --- | --- | --- | | Op | Func | Regimm | | 000000 | 000010 | xxxxx |   000000\_000010\_xxxxx // 0x01 : srl   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | RegDst | RegDatSel | RegWrite | SEUmode | ALUsrcB | ALUctrl | | 01 | 00 | 1 | x | 00 | 00 | | ALUop | DataWidth | MemWrite | MemtoReg | Branch | Jump | | 01110 | Xxx | 0 | 0 | 000 | 00 |   01\_00\_1\_x\_00\_00\_01110\_xxx\_0\_0\_000\_00\_xxxxx // 0x01 : srl $d = $t >> a  RegDst: 01, 결과값을 rd에 쓸 것이므로 목적지 레지스터로 rd를 선택한다.  RegDatSel: 00, ALU 결과값을 register file에 쓰도록 선택한다.  RegWrite: 1, register file에 값을 쓴다.  SEUmode: x, imm값을 extension할 필요가 없다.  ALUsrcB: 00, ALU 입력으로 register file port B를 사용한다.  ALUctrl: 00, ALUctrl[1]=0 (normal ALU input), ALUctrl[0]=0 (shift = shift amount)  ALUop: 01110, b >> a  DataWidth: xxx, 메모리 접근하지 않음.  MemWrite: 0, 메모리에 쓰지 않음.  MemtoReg: 0, register file에 ALU값을 쓴다.  Branch: 000, branch 수행하지 않음.  Jump: 00, jump 수행하지 않음. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 6. SH  Store Halfword: MEM [$s + i]에 레지스터 $t의 하위 2바이트 값을 쓴다, I-type  Syntax: o $t, i ($s)  Operation: MEM [$s + i]:2 = LH ($t)   |  |  |  | | --- | --- | --- | | Op | Func | Regimm | | 101001 | xxxxxx | xxxxx |   101001\_xxxxxx\_xxxxx // 0x31 : sh   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | RegDst | RegDatSel | RegWrite | SEUmode | ALUsrcB | ALUctrl | | xx | xx | 0 | 1 | 01 | 0x | | ALUop | DataWidth | MemWrite | MemtoReg | Branch | Jump | | 00100 | 010 | 1 | x | 000 | 00 |   xx\_xx\_0\_1\_01\_0x\_00100\_010\_1\_x\_000\_00\_xxxxx // 0x31 : sh MEM [$s + i]:2 = LH ($t)  RegDst: xx, register file에 쓰지 않음.  RegDatSel: xx, register file에 쓰지 않음  RegWrite: 0, register file에 값을 쓰지 않음..  SEUmode: 1, imm값을 sign extension한다.  ALUsrcB: 01, ALU 입력으로 imm값을 사용한다.  ALUctrl: 00, ALUctrl[1]=0 (normal ALU input), ALUctrl[0]=x (shift 수행하지 않음)  ALUop: 00100, a + b  DataWidth: 010, 16-bit Halfword  MemWrite: 1, 메모리에 씀..  MemtoReg: x, register file에 쓰지 않음  Branch: 000, branch 수행하지 않음.  Jump: 00, jump 수행하지 않음. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 7. LB  Load Byte: MEM [$s + i]의 1바이트 데이터를 가져와서 sign extend 후 $t에 저장한다, I-type  Syntax: o $t, i ($s)  Operation: $t = SE (MEM [$s + i]:1)   |  |  |  | | --- | --- | --- | | Op | Func | Regimm | | 100000 | xxxxxx | xxxxx |   100000\_xxxxxx\_xxxxx // 0x2b : lb   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | RegDst | RegDatSel | RegWrite | SEUmode | ALUsrcB | ALUctrl | | 00 | 00 | 1 | 1 | 01 | 0x | | ALUop | DataWidth | MemWrite | MemtoReg | Branch | Jump | | 00100 | 111 | 0 | 1 | 000 | 00 |   00\_00\_1\_1\_01\_0x\_00100\_111\_0\_1\_000\_00\_xxxxx // 0x2b : lb $t = SE (MEM [$s + i]:1)  RegDst: 00, 결과값을 rt에 쓸 것이므로 목적지 레지스터로 rt를 선택한다.  RegDatSel: 00, MEM 결과값을 register file에 쓰도록 선택한다.  RegWrite: 1, register file에 값을 쓴다.  SEUmode: 1, imm값을 sign extension한다.  ALUsrcB: 01, ALU 입력으로 imm값을 사용한다.  ALUctrl: 00, ALUctrl[1]=0 (normal ALU input), ALUctrl[0]=x (shift 수행하지 않음)  ALUop: 00100, a + b  DataWidth: 111, 8-bit Byte with Sign Ext  MemWrite: 0, 메모리에 쓰지 않음.  MemtoReg: 1, register file에 메모리 값을 쓴다.  Branch: 000, branch 수행하지 않음.  Jump: 00, jump 수행하지 않음. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 8. BNE  Branch Not Equal: $s != $t이면 branch한다, I-type  Syntax: o $s, $t, label  Operation: l if ($s != $t) pc += i << 2   |  |  |  | | --- | --- | --- | | Op | Func | Regimm | | 000101 | xxxxxx | xxxxx |   000101\_xxxxxx\_xxxxx // 0x20 : bne   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | RegDst | RegDatSel | RegWrite | SEUmode | ALUsrcB | ALUctrl | | xx | xx | 0 | 1 | 00 | 0x | | ALUop | DataWidth | MemWrite | MemtoReg | Branch | Jump | | 00110 | xxx | 0 | x | 101 | 00 |   xx\_xx\_0\_1\_00\_0x\_00110\_xxx\_0\_x\_101\_00\_xxxxx // 0x20 : bne if ($s != $t) pc += i << 2  RegDst: xx, register file에 쓰지 않음.  RegDatSel: xx, register file에 쓰지 않음  RegWrite: 0, register file에 값을 쓰지 않음..  SEUmode: 1, imm값을 sign extension한다.  ALUsrcB: 00, ALU 입력으로 register file port B를 사용한다.  ALUctrl: 00, ALUctrl[1]=0 (normal ALU input), ALUctrl[0]=x (shift 수행하지 않음)  ALUop: 00110, a – b (두 레지스터 값 비교)  DataWidth: xxx, 메모리 접근하지 않음  MemWrite: 0, 메모리에 쓰지 않음.  MemtoReg: x, register file에 쓰지 않음  Branch: 101, Branch if not zero  Jump: 00, jump 수행하지 않음. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 9. BGEZ  I-type, RegImm-type: opcode=000001 고정하고, rt(=regimm)로 명령어를 구분한다. $s >=이면 branch한다.  Syntax: r $s, label  Operation: if ($s >= 0) pc += i << 2   |  |  |  | | --- | --- | --- | | Op | Func | Regimm | | 000001 | xxxxxx | 00001 |   000001\_xxxxxx\_00001 // 0x1c : bgez   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | RegDst | RegDatSel | RegWrite | SEUmode | ALUsrcB | ALUctrl | | Xx | Xx | 0 | 1 | 10 | 0x | | ALUop | DataWidth | MemWrite | MemtoReg | Branch | Jump | | 10000 | Xxx | 0 | X | 011 | 00 |   xx\_xx\_0\_1\_10\_0x\_10000\_xxx\_0\_x\_011\_00\_xxxxx // 0x1c : bgez if ($s >= 0) pc += i << 2  RegDst: xx, register file에 쓰지 않음.  RegDatSel: xx, register file에 쓰지 않음  RegWrite: 0, register file에 값을 쓰지 않음..  SEUmode: 1, imm값을 sign extension한다.  ALUsrcB: 10, ALU 입력으로 0을 사용한다.  ALUctrl: 0x, ALUctrl[1]=0 (normal ALU input), ALUctrl[0]=x (shift 수행하지 않음)  ALUop: 10000, set less than  DataWidth: xxx, 메모리 접근하지 않음  MemWrite: 0, 메모리에 쓰지 않음.  MemtoReg: x, register file에 쓰지 않음  Branch: 011, Unconditional Branch to PC+imm16 (ALU에서 조건 검사 수행하므로)  Jump: 00, jump 수행하지 않음. |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 10. JALR  Jump And Link Register: PC를 $31에 저장하고 $s 레지스터가 가리키는 주소로 점프한다, R-type  Syntax: f labelR  Operation: $31 = pc; pc = $s   |  |  |  | | --- | --- | --- | | Op | Func | Regimm | | 000000 | 001001 | xxxxx |   000000\_001001\_xxxxx // 0x07 : jalr   |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | | RegDst | RegDatSel | RegWrite | SEUmode | ALUsrcB | ALUctrl | | 10 | 11 | 1 | x | Xx | Xx | | ALUop | DataWidth | MemWrite | MemtoReg | Branch | Jump | | xxxxx | Xxx | 0 | x | xxx | 10 |   10\_11\_1\_x\_xx\_xx\_xxxxx\_xxx\_0\_x\_000\_10\_xxxxx // 0x07 : jalr $31 = pc; pc = $s  RegDst: 10, PC를 $31에 쓸 것이므로 목적지 레지스터로 $31를 선택한다.  RegDatSel: 11, PC를 register file에 쓴다.  RegWrite: 1, register file에 값을 쓴다.  SEUmode: x, imm값을 extension할 필요가 없다.  ALUsrcB: xx, ALU 연산 수행하지 않음.  ALUctrl: xx, ALU 연산 수행하지 않음.  ALUop: xxxxx, ALU 연산 수행하지 않음.  DataWidth: xxx, 메모리 접근하지 않음.  MemWrite: 0, 메모리에 쓰지 않음.  MemtoReg: x, PC를 바로 레지스터 파일에 씀. (alu값과 메모리값을 쓰지 않음)  Branch: xxx, branch 수행하지 않음.  Jump: 10, Use $rs  주어진 회로 그림의 control signal에는 표시되지 않았지만, RegDatSel을 11로 설정하여 레지스터 파일에 쓸 값이 PC로 선택되었음이 동작하고 있다. 해당 동작에 맞추어, 좌측 위에서 계산된 PC+4가 레지스터 파일의 Write data로 들어가는 선을 추가하였다. 또한 RegDst가 write register로 $31를 선택하는 것을 표현했다.  마찬가지로 control signal인 Jump를 10으로 설정하여 $rs를 사용하는 동작이 수행되고 있다. 따라서 Jump 신호가 다음 PC를 선택하는 mux로 들어가는 선택선과, 읽은 $rs의 값을 해당 mux의 입력선으로 사용하기 위해 해당 선들을 추가로 그려서 위의 동작을 표현했다. |

|  |
| --- |
| <Simulation>  먼저 기본 ALU 연산 동작을 하는 명령어들의 시뮬레이션을 진행했다. 레지스터 연산을 하는 R-type 명령어 and, nor, sltu, srl과 imm값을 사용하여 연산하는 I-type 명령어 addi의 검증을 수행했다. 다음은 사용한 명령어와 그 동작이다.  000000\_00011\_00101\_00110\_00000\_100100 // and $6, $3, $5  000000\_00011\_00101\_00111\_00000\_100111 // nor $7, $3, $5  001000\_00011\_01000\_0000000000000001 // addi $8, $3, 1 000000\_00010\_00101\_01001\_00000\_101011 // sltu $9, $2, $5  000000\_00000\_00011\_01010\_00100\_000010 // srl $10, $3, 4    먼저 add 명령어를 읽고 수행 결과, $3, $5 두 레지스터를 읽고 $6에 $3 & $5 연산 결과인 0x12345678 & 0x11223344 = 0x10201240의 값이 저장되었다. PC = PC + 4이다.    다음으로 nor 명령어를 읽고 수행 결과, $3, $5 두 레지스터를 읽고 $7에 ~($3 | $5) 연산 결과인 ~(0x1336777C) = 0xECC98883의 값이 저장되었다. PC = PC + 4이다.    addi 명령어를 읽고 수행한 결과, $3과 imm값인 1을 읽은 후 $8에 0x12345678 + 1 연산 결과인 0x12345679가 저장되었다. PC = PC + 4이다.    sltu 명령어를 읽고 수행 결과, $2와 $5를 읽고 비교하여 0x12340000 > 0x11223344이므로 $9에 unsigned 기준 연산 결과인 0이 저장되었다. PC = PC + 4이다.    srl 명령어를 읽고 연산 결과, $3와 shamt 값을 읽은 후 $10에 $3를 shamt만큼 shift한 $3 >> 4 연산 결과인 0x01234567이 저장되었다. PC = PC + 4이다.  모든 명령어 수행 후 저장된 reg\_dump.txt 파일이다.    다음으로 메모리 접근을 수행하는 두 명령어 sh, lb의 시뮬레이션을 수행하였다.  001111\_00000\_01011\_0000000000000000 // lui $11, 0x0000  001101\_01011\_01011\_0000100000000000 // ori $11, $11, 0x0800  101001\_01011\_00011\_0000000000000001 // sh $3, 1($11)  100000\_01011\_01100\_0000000000000001 // lb $12, 1($11)  먼저 위 명령어들 수행을 통해 $11에 0x00000800 값이 저장될 것이다.      sh 명령어를 읽고 수행한 결과, MEM[$11 + 1]에 $3의 하위 2바이트 값을 쓸 것이다. 0x0800에 imm 값인 1이 더해진다. $3=0x12345678 중 리틀 앤디언 방식에 따라 메모리 주소 0x0800에는 0x78이, 0x0801에는 0x56이 저장되었다. RegWrite는 0, MemWrite는 1이다. PC = PC + 4이다.    다음으로 lb 명령어를 읽고 수행 결과, MEM[$11 + 1]의 1바이트 데이터를 가져와 sign extend 후 $12에 저장할 것이다. 0x0800에 imm 값인 1이 더해진 결과인 0x0801에는 이전 sh 명령어 수행 결과 0x56이 저장되어 있다. write register인 $12에 0x56이 저장되었다. 따라서 RegWrite는 1, MemWrite는 0이다. PC = PC + 4이다.  모든 명령어 수행 후 저장된 reg\_dump.txt 파일이다.    다음으로 branch와 jump 명령어들인 bne, bgez, jalr을 각각 검증했다.      000101\_00010\_00101\_0000000000000001 // bne $2, $5, 1  먼저 bne 명령어를 읽고 수행한 결과, $2와 $5를 읽었고 ALU에서 a-b 연산을 수행한 결과 0이아니므로 branch조건인 0x12340000 =/ 0x11223344를 만족한다. 계산한 pc 주소는 pc = pc+4+(imm<<2)이므로, next\_pc 값이 0x0010 + 0x0004 + 0x0004 = 0x0018인 것을 확인했다.      001111\_00000\_00110\_0000000000000000 // lui $6, 0x0000  001101\_00110\_00111\_0000000000000001 // ori $7, $6, 0x0001  000001\_00111\_00001\_0000000000000001 // bgez $7, 1  다음으로 bgez 명령어 검증을 위해 일단 lui와 ori 명령어를 통해 $7에 0x0001을 저장했다. bgez 명령어를 읽고 수행한 결과, $7를 읽고 0과 set less than ALU 연산한 결과 조건인 0x00000001 >= 0을 만족하므로 branch한다. imm값은 1이고, 계산한 pc 주소는 pc = pc+4+(imm<<2)이므로, next\_pc 값이 decimal로 표현했을 때 24 + 4 + 4 = 32인 것을 확인했다.    000000\_00101\_00000\_11111\_00000\_001001 // jalr $31, $5  jalr 명령어를 읽고 수행 결과, $31에는 PC+4=0x0014가 저장되고, PC에는 레지스터에서 읽은$5=0x11223344가 저장된다.  jalr 명령어 수행 후 저장된 reg\_dump.txt 파일이다. |