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

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1. AND  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  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  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, 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 shift한다. 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 |