RISC-V 32I(RV32I) 32비트 5단계 파이프라인 전체 흐름도

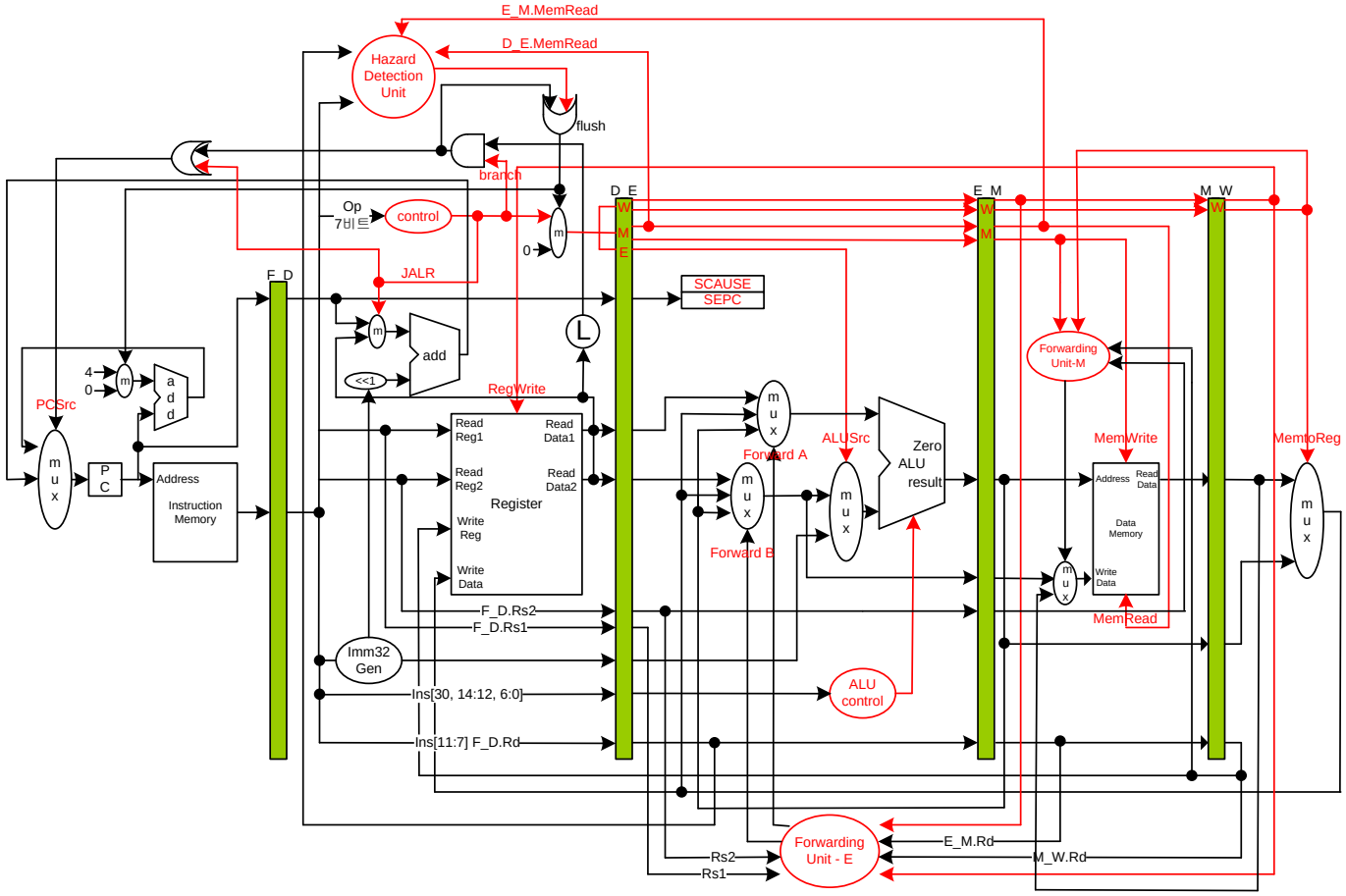

그림 1 RV32I 전체 흐름도

Fetch(F) : 명령어 인출

Decoder(D) : 명령어 해독 및 레지스터 파일 읽기

Execute(E) : 명령어 실행 또는 피 연산자 주소 계산

Memory(M) : 데이터 메모리 접근

Write(W) : 레지스터 파일에 쓰기

5단계 파이프라인의 구분은 그림 1에서 긴 녹색 직사각형의 레지스터로 구분한다. 이 레지스터의 이름은 각 파이프라인 단계를 연결한다는 의미로 각 단계의 첫 단어로 이름 짓는다. 예로서 Fetch와 Decoder 단계를 연결하는 레지스터를 F_D이며, 연속해서 D_E, E_M, M_W로 이름 지었다. F_D, D_E, E_M, M_W 레지스터는 clock 신호에 따라 각 파이프라인 단계의 결과를 입력 받아 잠시 저장 후 다음 단계의 파이프라인으로 출력한다. 이 레지스터들은 각 파이프라인 단계를 구분하는 구분자가 아니라 실제로는 각 파이프라인 단계를 연결하는 연결자 역할을 한다. 그림 1의 설명 전에 그림에서 사용된 사각형 모듈은 메모리가 포함된 순서회로이고, 원 또는 타원은 메모리가 없는 조합회로를 나타낸다. 각 파이프라인 단계별 그림 1의 설명은 다음과 같다.

F 단계

- mux : 입력(각각 32비트)

  현재 PC + 4, PC

  D 단계에 있는 branch 주소,

  1비트 선택 신호(PCSrc)

  1비트 선택 신호는 D 단계의 branch와 L 모듈의 결과를 and 한 결과.

- mux : PC + 0, PC + 4 입력 중 하나를 선택, 선택 신호는 D 단계 flush 신호
- PC : mux로부터 32비트 주소 저장
- add : 현재 PC, 또는 PC + 4를 수행하는 32비트 덧셈기
- Instruction Memory : PC 주소의 32비트 명령어를 출력
- F_D Reg : 32비트 현재 PC와 32비트 명령어 저장.

D 단계

- Hazard detection unit : 적재 명령어 바로 다음 명령어가 적재 명령어의 레지스터를 읽는 명령어 이면 1clock 지연이 필요. 입력으로 F_D Reg 의 32비트 명령어와 E 단계의 적재 신호 D_E.MemRead 그리고 목적지 레지스터 번호 D_E.rd 를 입력 받아, 아래의 조건이 맞으면, 1 clock 지연.

  if(D_E.MemRead && ((D_E.rd == F_D.rs1) | (D_E.rd == F_D.rs2)))

  1 clock 지연

  적재 명령어 바로 다음에 조건부 점프 명령어가 적재 명령어의 레지스터를 읽는 명령어 이면 2clock 지연이 필요

  if(D_E.MemRead && branch && ((D_E.rd == F_D.rs1) | (D_E.rd == F_D.rs2)))

  1 clock 지연

  if(E_M.MemRead && branch && ((E_M.rd == F_D.rs1) | (E_M.rd == F_D.rs2)))

  1 clock 지연

  1 clock 지연은 먼저 명령어를 삭제하고 같은 명령어를 다시 수행 시키는 것.

- Control : 32비트 명령어 중 op 7비트를 입력으로 받아 D, E, M, W 단계에서 필요한 제어신호 7비트를 만든다.

op	D 단계		E 단계	M 단계		W 단계	
	JALR	branch	ALUSrc	MemRead	MemWrite	RegWrite	MemtoReg
0110111(55)	0	0	1	0	0	1	1
0010111(23)	0	0	1	0	0	1	1
1101111(111)	0	0	1	0	0	1	1
1100111(103)	1	0	1	0	0	1	1
1100011(99)	0	1	0	0	0	0	0
0000011(3)	0	0	1	1	0	1	0
0100011(35)	0	0	1	0	1	0	0
0010011(19)	0	0	1	0	0	1	1
0110011(51)	0	0	0	0	0	1	1
0001111(15)	0	0	1	0	0	1	1
1110011(114)	0	0	1	0	0	1	1

- Add : 32비트 PC + 32비트로 부호 확장한 변위값 또는 레지스터 + 32비트로 부호 확장한 변위값, 이 결과는 다음 PC 값으로 사용할 수 있음.
- <<1 : 32비트 Imm32Gen 출력 결과를 왼쪽 1비트 시프트.
- L : 2개의 레지스터 값을 비교하는 branch 명령어의 6가지 ==, !=, >, >=, <, <=, 결과를 0, 1로 판별하는 모듈
- Imm32Gen : 각 명령어에서 즉치값을 32비트로 부호 확장.
- Register : 32비트 32개의 레지스터 파일, 1/2 clock 으로 레지스터 쓰기를 먼저 수행하며 쓸 레지스터 번호는 WriteReg, 쓸 데이터는 WriteData, 쓰기 신호는 RegWrite로 3개의 입력이 필요함. 나머지 1/2 clock 으로 2개의 레지스터를 동시에 읽기 수행, 첫번째 읽을 레지스터는 ReadReg1, 두번째 읽을 레지스터는 ReadReg2 를 입력 받으면 레지스터로부터 읽은 32비트 데이터를 ReadData1, ReadData2를 통해 D_E Reg에 저장.
- Mux : control 모듈에서 생성된 E, M, W 단계에 필요한 제어신호와 flush 신호 중 하나를 선택, 선택 신호는 Hazard detection unit와 PCSrc 신호를 OR 하여 생성. Mux 의 출력은 E, M, W 단계로 구분하여 D_E Reg에 저장.
- Mux : 32비트 PC 또는 레지스터 값 중 하나를 선택, 선택 신호는 control 신호의 JALR.
- D_E Reg : 레지스터 파일로부터 2개의 32비트 값 저장, 모든 명령어 마다 2개의 32비트 값을 저장하는 것은 아니고, 즉치값이 있는 명령어는 1개의 32비트 레지스터만 저장함. 부호 확장된 32비트 즉치값 저장, 32비트 PC 저장, 목적지 레지스터와 소스 레지스터 2개의 레지스터 번호 저장, 각각은 5비트로 Rd, Rs1, Rs2, E 단계 ALU의 제어 신호를 만드는 ALUcontrol 모듈에 사용할 32비트 명령어 중 op 7비트, fun3 3비트, 명령어 ins[30]번째 비트 저장, Mux 의 출력 E 단계 ALUSrc 1비트, M 단계 MemWrite 1비트, MemRead 1비트, W 단계 RegWrite 1비트, MemtoReg 1비트 제어 신호를 저장.

E 단계

- SCAUSE : 예외 원인을 기록하는 레지스터.
- SEPC : 예외를 발생시킨 명령어의 32비트 주소(PC)를 저장 후 예외 처리를 위해 OS로 제어를 넘김
- ALU : 32비트 산술논리연산장치, 2개의 32비트 입력은 D 단계로부터 2개의 32비트 레지스터 값 또는 부호 확장된 32비트 즉치값 또는 M 단계의 ALU 결과 32비트 또는 W 단계의 32비트 결과를 입력으로 받아서 +, -, &, |, ^, <<, >>, 산술>>, 크기 비교 연산 <, 부호 없는 크기 비교 연산 <, 연산 결과를 출력한다.
- Mux : ALU 입력을 선택하는 3개의 입력 중 1개를 선택하는 mux 2개는 각각 Forwarding Unit – E 로부터 2비트 선택 신호를 받아 32비트 입력 3개 중 1개를 선택한다. 2개의 입력으로부터 1개를 선택하는 mux 3개는 각각 D 단계의 Control 모듈의 E.flush 신호와 D_E Reg의 ALUSrc 신호를 선택 신호로 데이터를 선택한다.
- ALUcontrol : D_E Reg로부터 32비트 명령어 중 op 7비트, fun3 3비트, 명령어의 31번째 비트 ins[30], 11비트를 입력으로 받아 ALU 연산 +, -, &, |, ^, <<, >>, 산술>>, 크기 비교 연산 <, 부호 없는 크기 비교 연산 < 중 하나를 선택하는 4비트 신호를 만든다.
- Forwarding Unit - E : 데이터 종속성으로 데이터 해저드(Hazard) 발생시 전방 전달을 통해 지연을 방지하기 위한 모듈, 입력으로 D_E Reg의 소스 레지스터1, 2번 각각의 번호, E_M Reg의 목적지 레지스터 번호와 RegWrite 신호, M_W Reg의 목적지 레지스터 번호와 RegWrite 신호를 입력으로 받아 E 단계의 ALU 32비트 입력 데이터를 결정하는 2개의 mux의 선택 신호를 생성한다.

E 단계 해저드

if(E_M.RegWrite && (E_M.rd != 0) && (E_M.rd == D_E.rs1))

Forward A = 2

if(E_M.RegWrite && (E_M.rd != 0) && (E_M.rd == D_E.rs2))

Forward B = 2

M 단계 해저드

if(M_W.RegWrite && (M_W.rd != 0) && !(E_M.RegWrite && (E_M.rd != 0) && (E_M.rd == D_E.rs1)) && (M_W.rd == D_E.rs1))

ForwardA = 1

if(M_W.RegWrite && (M_W.rd != 0) && !(E_M.RegWrite && (E_M.rd != 0) && (E_M.rd == D_E.rs2)) && (M_W.rd == D_E.rs2))

ForwardB = 1

- E_M Reg : D_E Reg로부터 M, W 단계에서 사용할 제어신호 각각 2비트씩 저장, 32비트 ALU 결과 저장, 32비트 레지스터 값과 그 레지스터 번호 5비트 저장, 목적지 레지스터 번호 5비트 저장.

M 단계

- Forwarding Unit - M : E_M Reg의 MemWrite, 소스2 레지스터 번호 Rs2 그리고 M_W Reg의 MemtoReg, 목적지 레지스터 번호 Rd를 입력으로 받아 메모리에 저장할 32비트 데이터의 선택을 의한 mux 선택 신호를 만든다.

if(E_M.MemWrite && (M_W.MemtoReg == 0) && (E_M.rs2 == M_W.rd))

Forwarding Unit – M = 1

- Data memory : 메모리 읽기는 E_M Reg 로부터 ALU 결과값 32비트를 메모리 Address로 그리고 MemRead 신호가 인가되면 32비트 Address 위치의 메모리 32비트 데이터를 읽어 ReadData를 거쳐 M_W Reg로 읽은 32비트 데이터를 저장, 메모리 쓰기는 E_M Reg 로부터 ALU 결과값 32비트를 메모리 Address로 그리고 E_M Reg 로부터 메모리에 저장할 32비트 데이터를 WriteData로 그리고 MemWrite 신호가 인가 되면 WriteData에 저장된 32비터 데이터를 메모리 Address 위치에 저장.
- Mux : 메모리에 저장할 32비트 데이터 2개중 1개를 선택하는 mux 32비트 입력 2개는 E_M Reg와 M_W Reg 이고 선택 신호는 Forwarding Unit - M 모듈에서 생성한다.
- M_W Reg : E_M Reg로부터 W 단계에서 사용할 제어 신호 2비트 저장, 데이터 메모리에서 읽은 32비트 데이터 저장, E_M Reg의 32비트 ALU 결과 저장, 목적지 레지스터 번호 5비트 저장.

W 단계

- Mux : 레지스터 파일에 저장할 32비트 데이터 선택을 위한 선택기, 2개의 32비트 입력은 M_W Reg로부터 메모리에서 읽은 데이터와 ALU 결과이고 mux 선택 신호는 M_W Reg의 MemtoReg 신호 이다. 그리고 RegWrite 신호는 D 단계의 레지스터 파일의 쓰기 신호로 사용 된다.

imm[31:12]				rd	0110111(55)	LUI	U
imm[31:12]				rd	0010111(23)	AUIPC	U
imm[20|10:1|11|19:12]				rd	1101111(111)	JAL	J
imm[11:0]		rs1	000	rd	1100111(103)	JALR	I
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011(99)	BEQ	B
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011(99)	BNE	B
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011(99)	BLT	B
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011(99)	BGE	B
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011(99)	BLTU	B
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011(99)	BGEU	B
imm[11:0]		rs1	000	rd	0000011(3)	LB	I
imm[11:0]		rs1	001	rd	0000011(3)	LH	I
imm[11:0]		rs1	010	rd	0000011(3)	LW	I
imm[11:0]		rs1	100	rd	0000011(3)	LBU	I
imm[11:0]		rs1	101	rd	0000011(3)	LHU	I
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011(35)	SB	S
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011(35)	SH	S
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011(35)	SW	S
imm[11:0]		rs1	000	rd	0010011(19)	ADDI	I
imm[11:0]		rs1	010	rd	0010011(19)	SLTI	I
imm[11:0]		rs1	011	rd	0010011(19)	SLTIU	I
imm[11:0]		rs1	100	rd	0010011(19)	XORI	I
imm[11:0]		rs1	110	rd	0010011(19)	ORI	I
imm[11:0]		rs1	111	rd	0010011(19)	ANDI	I
0000000	shamt	rs1	001	rd	0010011(19)	SLLI	I
0000000	shamt	rs1	101	rd	0010011(19)	SRLI	I
0100000	shamt	rs1	101	rd	0010011(19)	SRAI	I
0000000	rs2	rs1	000	rd	0110011(51)	ADD	R
0100000	rs2	rs1	000	rd	0110011(51)	SUB	R
0000000	rs2	rs1	001	rd	0110011(51)	SLL	R
0000000	rs2	rs1	010	rd	0110011(51)	SLT	R
0000000	rs2	rs1	011	rd	0110011(51)	SLTU	R
0000000	rs2	rs1	100	rd	0110011(51)	XOR	R
0000000	rs2	rs1	101	rd	0110011(51)	SRL	R
0100000	rs2	rs1	101	rd	0110011(51)	SRA	R
0000000	rs2	rs1	110	rd	0110011(51)	OR	R
0000000	rs2	rs1	111	rd	0110011(51)	AND	R
fm	pred	succ	rs1	000	rd	0001111(15)	FENCE	I
imm[11:0]			rs1	001	rd	0001111(15)	FENCE.I	I
000000000000			00000	000	00000	1110011(114)	ECALL	I

000000000001	00000	000	00000	1110011(114)	EBREAK	I
csr	rs1	001	rd	1110011(114)	CSRRW	I
csr	rs1	010	rd	1110011(114)	CSRRS	I
csr	rs1	011	rd	1110011(114)	CSRRC	I
csr	uimm	101	rd	1110011(114)	CSRRWI	I
csr	uimm	110	rd	1110011(114)	CSRRSI	I
csr	uimm	111	rd	1110011(114)	CSRRCI	I

위 표는 RV32I의 47개의 기본 명령어 표이다. 각 명령어 설명은 순서대로 다음과 같다.

- LUI : 12비트를 왼쪽 시프트한 20비트 즉치값을 rd 레지스터로 저장. 예 LUI x5, 0x12345 결과는 x5 = 0x12345000 이 x5에 저장 됨
- AUIPC : 12비트 왼쪽 시프트한 20비트 즉치값과 PC를 더한 결과를 rd 레지스터로 저장. 예 AUIPC x5, 0x12345 결과는 x5 = PC + 0x12345000 이 x5에 저장됨
- JAL : PC 상대 주소 변환으로 무조건 점프. 예 JAL x1, 100 결과는 x1 = PC+100 을 저장 하고 PC+100번지로 무조건 점프
- JALR : 레지스터 상대 주소 변환으로 무조건 점프. 예 JALR x1, 100(x5) 결과는 x1 = PC+4 을 저장 하고 x5+100번지로 무조건 점프
- BEQ : 2개의 레지스터 값이 같으면 PC 상대 주소로 점프. 예 BEQ x5, x6, 100 의 결과는 if(x5 == x6) 이면 PC+100번지로 점프
- BNE : 2개의 레지스터 값이 다르면 PC 상대 주소로 점프. 예 BNE x5, x6, 100 의 결과는 if(x5 != x6) 이면 PC+100번지로 점프
- BLT : 2번째 레지스터 값이 더 크면 PC 상대 주소로 점프. 예 BLT x5, x6, 100 의 결과는 if(x5 < x6) 이면 PC+100번지로 점프
- BGE : 1번째 레지스터 값이 같거나 크면 PC 상대 주소로 점프. 예 BGE x5, x6, 100 의 결과는 if(x5 >= x6) 이면 PC+100번지로 점프
- BLTU : 부호 없는 정수로 2번째 레지스터 값이 더 크면 PC 상대 주소로 점프. 예 BLTU x5, x6, 100 의 결과는 if(x5 < x6) 이면 PC+100번지로 점프
- BGEU : 부호 없는 정수로 1번째 레지스터 값이 같거나 크면 PC 상대 주소로 점프. 예 BGEU x5, x6, 100 의 결과는 if(x5 >= x6) 이면 PC+100번지로 점프
- LB : 레지스터 상대 주소 메모리로부터 1바이트를 rd 레지스터에 저장. 예 LB x5, 100(x6) 결과 x5 = mem[x6+100]
- LH : 레지스터 상대 주소 메모리로부터 1/2워드를 rd 레지스터에 저장. 예 LH x5, 100(x6) 결과 x5 = mem[x6+100]
- LW : 레지스터 상대 주소 메모리로부터 1워드를 rd 레지스터에 저장. 예 LW x5, 100(x6) 결과 x5 = mem[x6+100]

- LBU : 레지스터 상대 주소 메모리로부터 부호 없는 정수 1바이트를 rd 레지스터에 저장. 예 LBU x5, 100(x6) 결과 x5 = mem[x6+100]
- LHU : 레지스터 상대 주소 메모리로부터 부호 없는 정수 1/2워드를 rd 레지스터에 저장. 예 LHU x5, 100(x6) 결과 x5 = mem[x6+100]
- SB : 레지스터의 1바이트를 레지스터 상대 주소 메모리에 저장. 예 SB x5, 100(x6) 결과는 mem[x6+100] = x5
- SH : 레지스터의 1/2워드를 레지스터 상대 주소 메모리에 저장. 예 SH x5, 100(x6) 결과는 mem[x6+100] = x5
- SW : 레지스터의 1워드를 레지스터 상대 주소 메모리에 저장. 예 SW x5, 100(x6) 결과는 mem[x6+100] = x5
- ADDI : rs1 레지스터와 12비트 즉치값을 더하여 rd 레지스터에 저장. 예 ADDI x5, x6, 100 의 결과는 x5 = x6+100
- SLTI : rs1 부호 있는 레지스터 값보다 부호 확장된 12비트 측치 값이 크면 rd 레지스터에 1, 그렇지 않으면 0을 저장. 예 SLTI x5, x6, 0x123 의 결과는 if(x6 < 0x12300000) x5 = 1 else x5 = 0
- SLTIU : rs1 부호 없는 레지스터 값보다 부호 없이 확장된 12비트 측치 값이 크면 rd 레지스터에 1, 그렇지 않으면 0을 저장. 예 SLTIU x5, x6, 0x123 의 결과는 if(x6 < 0x12300000) x5 = 1 else x5 = 0
- XORI : rs1 레지스터와 12비트 즉치값을 XOR 하여 rd 레지스터에 저장. 예 XORI x5, x6, 100 의 결과는 x5 = x6 ^ 100
- ORI : rs1 레지스터와 12비트 즉치값을 OR 하여 rd 레지스터에 저장. 예 ORI x5, x6, 100 의 결과는 x5 = x6 | 100
- ANDI : rs1 레지스터와 12비트 즉치값을 AND 하여 rd 레지스터에 저장. 예 ANDI x5, x6, 100 의 결과는 x5 = x6 & 100
- SLLI : rs1 레지스터를 12비트 즉치값으로 왼쪽 시프트하여 rd 레지스터에 저장. 예 SLLI x5, x6, 10 의 결과는 x5 = x6 << 10
- SRLI : rs1 레지스터를 12비트 즉치값으로 오른쪽 시프트하여 rd 레지스터에 저장. 예 SRLI x5, x6, 10 의 결과는 x5 = x6 >> 10
- SRAI : rs1 레지스터를 12비트 즉치값으로 오른쪽 산술 시프트하여 rd 레지스터에 저장. 예 SRAI x5, x6, 10 의 결과는 x5 = x6 >> 10
- ADD : rs1, rs2 레지스터를 더하여 rd 레지스터에 저장. 예 ADD x5, x6, x7 의 결과는 x5 = x6 + x7
- SUB : rs1, rs2 레지스터를 빼서 rd 레지스터에 저장. 예 SUB x5, x6, x7 의 결과는 x5 = x6 - x7
- SLL : rs1 레지스터를 rs2 레지스터 값으로 왼쪽 시프트하여 rd 레지스터에 저장. 예 SLL x5, x6, x7의 결과는 x5 = x6 << x7
- SLT : rs1 부호 있는 레지스터 값보다 rs2 부호 있는 레지스터 값이 크면 rd 레지스터에 1, 그렇지 않으면 0을 저장. 예 SLT x5, x6, x7 의 결과는 if(x6 < x7) x5 = 1 else x5 = 0

- SLTU : rs1 부호 없는 레지스터 값보다 rs2 부호 없는 레지스터 값이 크면 rd 레지스터에 1, 그렇지 않으면 0을 저장. 예 SLTU x5, x6, x7 의 결과는 if(x6 < x7) x5 = 1 else x5 = 0
- XOR : rs1, rs2 레지스터를 XOR하여 rd 레지스터에 저장. 예 XOR x5, x6, x7 의 결과는 x5 = x6 ^ x7
- SRL : rs1 레지스터를 rs2 레지스터 값으로 오른쪽 시프트하여 rd 레지스터에 저장. 예 SRL x5, x6, x7의 결과는 x5 = x6 >> x7
- SRA : rs1 레지스터를 rs2 레지스터 값으로 오른쪽 산술 시프트하여 rd 레지스터에 저장. 예 SRA x5, x6, x7의 결과는 x5 = x6 >> x7
- OR : rs1, rs2 레지스터를 OR하여 rd 레지스터에 저장. 예 OR x5, x6, x7 의 결과는 x5 = x6 | x7
- AND : rs1, rs2 레지스터를 AND하여 rd 레지스터에 저장. 예 AND x5, x6, x7 의 결과는 x5 = x6 & x7
- FENCE : 멀티프로세싱과 입출력을 위한 데이터 메모리 접근 순서 제어
- FENCE.I : 프로세서에게 소프트웨어가 명령어 메모리를 수정 하였음을 알려주어 프로세서가 수정된 명령어를 인출하게 한다.
- ECALL : 환경 호출 예외를 발생시키고, OS를 호출
- EBREAK : 브레이크 포인트 예외를 발생시키고 디버거 호출
- CSRRW : csr을 0으로 확장하여 32비트를 rd에 저장, csr = rs1
- CSRRS : csr을 0으로 확장하여 32비트를 rd에 저장, csr = csr | rs1
- CSRRC : csr을 0으로 확장하여 32비트를 rd에 저장, csr = 0
- CSRRWI : csr을 0으로 확장하여 32비트를 rd에 저장, csr = 즉치값
- CSRRSI : csr을 0으로 확장하여 32비트를 rd에 저장, csr = csr | 즉치값
- CSRRCI : csr을 0으로 확장하여 32비트를 rd에 저장, csr = 0