

/////////////////define

마스킹을 위한 상수 정의

#define FMT1 0b00000011111000000000000000000000 //25-21

#define FMT2 0b00000000000111110000000000000000 //20-16

#define FMT3 0b00000000000000001111100000000000 //15-11

#define FMT4 0b00000000000000001111111111111111 //15-0

#define FMT5 0b00000000000000000000000000111111 //5-0

#define FMT6 0b11111111111111110000000000000000 //sign\_msb == 1

#define FMT7 0b00000000000000001111111111111111 //sing\_msb == 0

#define FMT8 0b00000011111111111111111111111111 //25-0

MUX switch에 사용하기 위한 상수 정의

//MUX mode

#define REGDST 0

#define BRANCH 1

#define MEMREAD 2

#define MEMTOREG 3

#define ALUOP 4

#define MEMWRITE 5

#define ALUSRC 6

#define REGWRITE 7

#define PCSRC 8

#define JUMP 9

Signal 변수 정의

int RegDst, Branch, MemRead, MemtoReg, ALUOp, MemWrite, ALUSrc, RegWrite, PCSrc, Jump, Jal, Jr; //signals

long ALUcontrol; //signal

/////////////////Fetch

Read hex 🡪 long 이 32bit 자료형이므로

위 6비트가 opcode 저장(31-26)

25-21 inst2521저장

25-00 inst2500저장

20-16 inst2016저장

15-11 inst1511저장

15-0 inst1500저장

5-0 inst0500저장

변수 extended = Instruction[15-0] sign extend

//////////////////Decode

Read data 1 = Read register 1

Read data 2 = Read register 2

//////////////////EX

ALUcontroller(): ALUcontrol 시그널 결정

ALU(): 결정된 ALUcontrol시그널에 의해 ALUoutput 도출

////////////////////MEM

MemRead 및 MemWrite 시그널을 확인하여 Datamemory 로부터 read 또는 write

////////////////////wb

RegWrite == 1이면 MUX(REGDST) index에 해당하는 regs배열에 MUX(MEMTOREG)의 결과 저장

////////////////////update\_pc

Jr 인경우 pc = readData1로 설정 (readData1 = regs[inst2521])

그 외경우 PCSrc = (Branch & Zero) 를 통해 PCSrc 시그널 결정하여

pc에 MUX(JUMP) 결과 저장 ( MUX(JUMP)를 통해 j인 경우 j target을 pc로 , 그 외에는 내부에서 MUX(PCSRC)를 호출하여 pc+4또는 readMem의 값을 pc로 전달)

////////////////////ALU 구현

void ALU()

{

int src; //ALUsrc

src = MUX(ALUSRC);

switch (ALUcontrol)

{

case 0: //AND

ALUoutput = src & readData1;

break;

case 1: //OR

ALUoutput = src | readData1;

break;

case 2: //add

ALUoutput = src + readData1;

break;

case 6: //sub

if (src >= readData1)

ALUoutput = src - readData1;

else

ALUoutput = readData1 - src;

if (ALUoutput == 0)

Zero = 1;

else

Zero = 0;

break;

case 7: //slti

if (readData1 < src)

ALUoutput = 1;

else

ALUoutput = 0;

break;

}

}

////////////////////ALUcontroller 구현

void ALUcontroller()

{

switch (ALUOp)

{

case 0://lw, sw

ALUcontrol = 2;

break;

case 1: //beq

ALUcontrol = 6;

break;

case 2: //R-type, slti

if (inst0500 == 32) // add

{

ALUcontrol = 2;

}

else if (inst0500 == 34) //sub

{

ALUcontrol = 6;

}

else if (inst0500 == 36) //AND

{

ALUcontrol = 0;

}

else if (inst0500 == 37) //OR

{

ALUcontrol = 1;

}

else if (opcode == 10) //slti

{

ALUcontrol = 7;

}

break;

}

}

////////////////////MUX 구현

int MUX(int n)

{

switch(n)

{

case REGDST:

if (RegDst == 0)

return inst2016;

else if (RegDst == 2)

{

return 31;

}

else//RegDst 1

return inst1511;

case ALUSRC:

if (ALUSrc == 0)

return readData2;

else //from sign-extend

return extended;

case PCSRC:

if (PCSrc == 0)// pc + 4

return pc+4;

else //branch

return (extended<<2) + (pc+4) ;

case MEMTOREG:

if (MemtoReg == 0)

return ALUoutput;

else if (MemtoReg == 2)

return pc + 4;

else

return readMem;

case JUMP:

if (Jump == 0)

return MUX(PCSRC);

else

return ((pc + 4) & 0b11110000000000000000000000000000) | (inst2500 << 2);

}

}

////////////////////Main control 구현

switch (opcode)

{

case 0: //add,sub,jr

if (inst0500 == 8) //jr

{

RegDst = 0;

Branch = 0;

MemRead = 0;

MemtoReg = 0;

ALUOp = 0;

MemWrite = 0;

ALUSrc = 0;

RegWrite = 0;

Jump = 0;

Jal = 0;

Jr = 1;

}

else

{

RegDst = 1;

Branch = 0;

MemRead = 0;

MemtoReg = 0;

ALUOp = 2;

MemWrite = 0;

ALUSrc = 0;

RegWrite = 1;

Jump = 0;

Jal = 0;

Jr = 0;

}

break;

case 8: //addi

RegDst = 0;

Branch = 0;

MemRead = 0;

MemtoReg = 0;

ALUOp = 0;

MemWrite = 0;

ALUSrc = 1;

RegWrite = 1;

Jump = 0;

Jal = 0;

Jr = 0;

break;

case 3: //jal

RegDst = 2; //for jal

Branch = 0;

MemRead = 0;

MemtoReg = 2; //for jal

ALUOp = 0;

MemWrite = 0;

ALUSrc = 0;

RegWrite = 1;

Jump = 1;

Jal = 1;

Jr = 0;

break;

case 2: //j

RegDst = 0;

Branch = 0;

MemRead = 0;

MemtoReg = 0;

ALUOp = 0;

MemWrite = 0;

ALUSrc = 0;

RegWrite = 0;

Jump = 1;

Jal = 0;

Jr = 0;

break;

case 35: //lw

RegDst = 0;

Branch = 0;

MemRead = 1;

MemtoReg = 1;

ALUOp = 0;

MemWrite = 0;

ALUSrc = 1;

RegWrite = 1;

Jump = 0;

Jal = 0;

Jr = 0;

break;

case 43: //sw

RegDst = 0;

Branch = 0;

MemRead = 0;

MemtoReg = 0;

ALUOp = 0;

MemWrite = 1;

ALUSrc = 1;

RegWrite = 0;

Jump = 0;

Jal = 0;

Jr = 0;

break;

case 10: //slti

RegDst = 0;

Branch = 0;

MemRead = 0;

MemtoReg = 0;

ALUOp = 2;

MemWrite = 0;

ALUSrc = 1;

RegWrite = 1;

Jump = 0;

Jal = 0;

Jr = 0;

break;

case 4: //beq

RegDst = 0;

Branch = 1;

MemRead = 0;

MemtoReg = 0;

ALUOp = 1;

MemWrite = 0;

ALUSrc = 0;

RegWrite = 0;

Jump = 0;

Jal = 0;

Jr = 0;

break;

}