EE312 Computer architecture

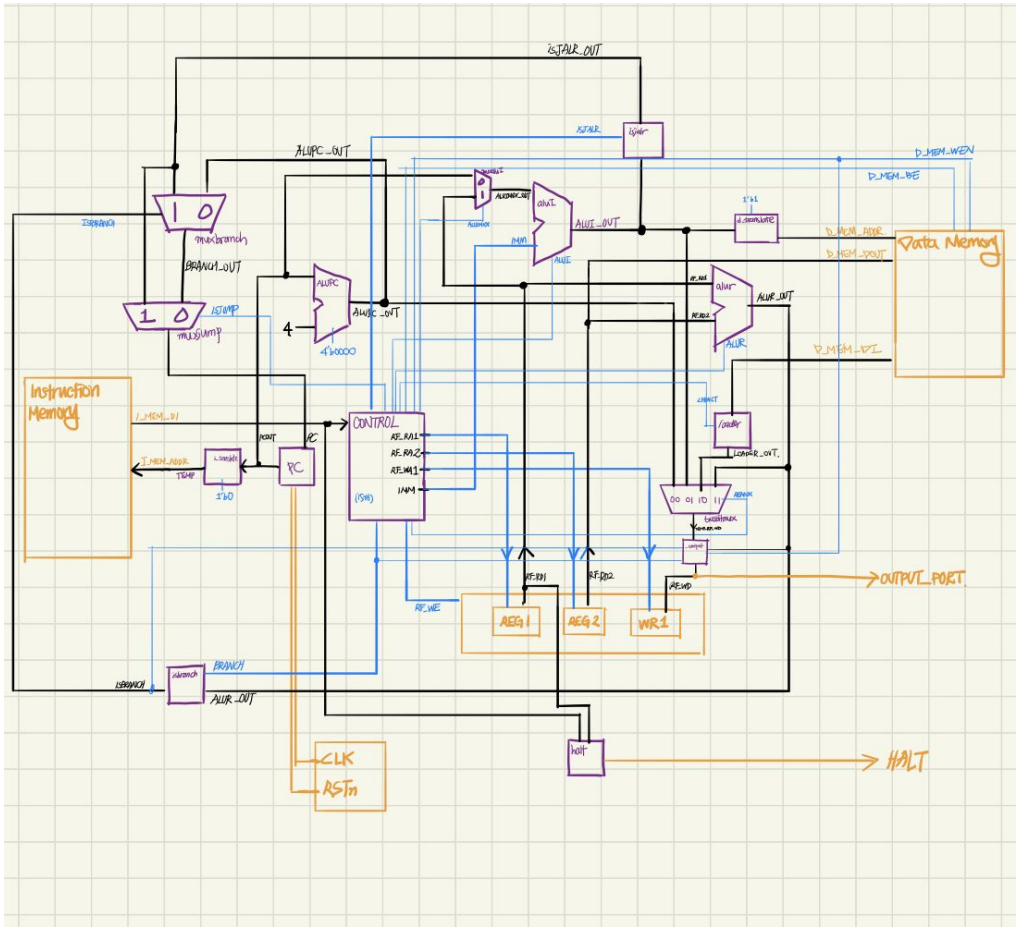
LAB3 report

20160542 이현지 / 20160707 형준하

INTRODUCTION

- 이번 랩의 목적은 single-cycle CPU of risc-v version을 Verilog을 통해 implement하는 것이었다. 주어진 instruction들을 risc-v 가 수행하게 정해놓은 바와 같이 실행할 수 있도록 코드를 짰다. Data path, control path를 나누어서 하나의 모듈에서 모든 control path를 관리할 수 있도록 하였고, 이전 lab들과는 달리 여러가지 모듈로 나누어서 이를 RISCV_TOP.v에서 input and output을 알맞게 이어주어 이를 연결만 해주는 방식으로 코드를 구성하였다.

DESIGN

- 

- 위와 같이 회로를 디자인하였다. 보라색은 만든 module을, 파란선은 control path를, 검은선은 data path를 그리고 주황색은 RISCV_TOP에서 해주지 않고 주어진 파일에서 만들어진 메모리, register 그리고 clocking을 나타낸다. 그리고 외부와 연결되어 나가는 halt, output_port 또한 주황색으로 그렸다.
- CONTROL이라는 module안에서 instruction을 해석하고, opcode에 따라서 instruction을 알맞게 decode해준다. decode한 것을 기반으로 아래의 엑셀 파일에 알맞게 신호를 주었고, 이를 통해 operation이 진행이 되고 instruction이 실행되게 만들었다.

	ALUIMUX	ALUI	ALUR	D_MEM_BE	D_MEM_WEN	BRANCH	RF_WE	REMUX	LFUNCT	RF_RA1	RF_RA2	RF_WA1	IMM	EXT	ISJALR	ISJUMP
LUI	z	z	z	z	1	0	1	2b01	z	z	z	11~7	31~12	lowest 12 bit to zero	0	0
AUIPC	0	4'b0000	z	z	1	0	1	2b01	z	z	z	11~7	31~12	lowest 12 bit to zero	0	0
JAL	0	4'b0000	z	z	1	0	1	2b00	z	z	z	11~7	31,19~12,20,30~21, lsb에 0	sign extend	0	1
JALR	1	4'b0000	z	z	1	0	1	2b00	z	19~15	z	11~7	31~20	sign extend	1	1
BEQ	0	4'b0000	4b1010	z	1	1	0	z	z	19~15	24~20	z	31,7,30~25,11~8, lsb에 0	sign extend	0	0
BNE	0	4'b0000	4b1011	z	1	1	0	z	z	19~15	24~20	z	31,7,30~25,11~8, lsb에 0	sign extend	0	0
BLT	0	4'b0000	4b1100	z	1	1	0	z	z	19~15	24~20	z	31,7,30~25,11~8, lsb에 0	sign extend	0	0
BGE	0	4'b0000	4b1101	z	1	1	0	z	z	19~15	24~20	z	31,7,30~25,11~8, lsb에 0	sign extend	0	0
BLTU	0	4'b0000	4b1110	z	1	1	0	z	z	19~15	24~20	z	31,7,30~25,11~8, lsb에 0	sign extend	0	0
BGEU	0	4'b0000	4b1111	z	1	1	0	z	z	19~15	24~20	z	31,7,30~25,11~8, lsb에 0	sign extend	0	0
LB	1	4'b0000	z	z	1	0	1	2b10	14~12	19~15	z	11~7	31~20	sign extend	0	0
LH	1	4'b0000	z	z	1	0	1	2b10	14~12	19~15	z	11~7	31~20	sign extend	0	0
LW	1	4'b0000	z	z	1	0	1	2b10	14~12	19~15	z	11~7	31~20	sign extend	0	0
LBU	1	4'b0000	z	z	1	0	1	2b10	14~12	19~15	z	11~7	31~20	sign extend	0	0
LHU	1	4'b0000	z	z	1	0	1	2b10	14~12	19~15	z	11~7	31~20	sign extend	0	0
SB	1	4'b0000	z	4'b0001	0	0	0	z	z	19~15	24~20	z	31~25,11~7	sign extend	0	0
SJ	1	4'b0000	z	4'b0011	0	0	0	z	z	19~15	24~20	z	31~25,11~7	sign extend	0	0
SW	1	4'b0000	z	4b1111	0	0	0	z	z	19~15	24~20	z	31~25,11~7	sign extend	0	0
ADDI	1	4'b0000	z	z	1	0	1	2b01	z	19~15	z	11~7	31~20	sign extend	0	0
SLTI	1	4b0011	z	z	1	0	1	2b01	z	19~15	z	11~7	31~20	sign extend	0	0
SLTIU	1	4b0100	z	z	1	0	1	2b01	z	19~15	z	11~7	31~20	sign extend	0	0
XORI	1	4b0101	z	z	1	0	1	2b01	z	19~15	z	11~7	31~20	sign extend	0	0
ORI	1	4b1000	z	z	1	0	1	2b01	z	19~15	z	11~7	31~20	sign extend	0	0
ANDI	1	4b1001	z	z	1	0	1	2b01	z	19~15	z	11~7	31~20	sign extend	0	0
SLLI	1	4b0010	z	z	1	0	1	2b01	z	19~15	z	11~7	31~20	sign extend	0	0
SRLI	1	4b0110	z	z	1	0	1	2b01	z	19~15	z	11~7	31~20	sign extend	0	0
SRAI	1	4b0111	z	z	1	0	1	2b01	z	19~15	z	11~7	31~20	sign extend	0	0
ADD	z	z	4b0000(add)	z	1	0	1	2b11	z	19~15	24~20	11~7	z	z	0	0
SUB	z	z	4b0001(sub)	z	1	0	1	2b11	z	19~15	24~20	11~7	z	z	0	0
SLL	z	z	4b0010(sll)	z	1	0	1	2b11	z	19~15	24~20	11~7	z	z	0	0
SLT	z	z	4b0011(slt)	z	1	0	1	2b11	z	19~15	24~20	11~7	z	z	0	0
STLU	z	z	4b0100(stlu)	z	1	0	1	2b11	z	19~15	24~20	11~7	z	z	0	0
XOR	z	z	4b0101(xor)	z	1	0	1	2b11	z	19~15	24~20	11~7	z	z	0	0
SRL	z	z	4b0110(srl)	z	1	0	1	2b11	z	19~15	24~20	11~7	z	z	0	0
SRA	z	z	4b0111(sra)	z	1	0	1	2b11	z	19~15	24~20	11~7	z	z	0	0
OR	z	z	4b1000(or)	z	1	0	1	2b11	z	19~15	24~20	11~7	z	z	0	0
AND	z	z	4b1001(and)	z	1	0	1	2b11	z	19~15	24~20	11~7	z	z	0	0

- 위의 엑셀 파일은 instruction에 알맞게 어떤 control signal을 넣어줄 지 정한 파일이다.

IMPLEMENTATION

- 총 11개의 module을 만들었습니다. 각각의 module에 대한 간략한 설명을 드리겠습니다.
- [ALU] alul, aluPC, alur을 만드는데 사용하였고, OP라는 control signal을 받아 이 신호에 알맞게 operation을 input A, B에 알맞게 진행하고 결과값을 Out으로 내보냈다. OP의 값에 대해서는 자체적으로 3개의 alu에 알맞게 지정해주고 instruction에 알맞게 controller에서 보내주었다.
- [HALT] halt signal을 관리하는 module로, Instruction이 32'h00008067이고 RF_RD1이 32'h0000000c이면 halt signal을 1로 만들어 종료하게 하였고, 다른 경우에 대해서는 그냥 0을 내보내었다.
- [CONTROL] 총 15개의 signal을 관리하는데, 우선 Instruction을 Instruction memory로부터 받아 case문을 통해 opcode인 [6:0] 부분에 따라 알맞는 control signal을 내보내준다. 어떤 control signal을 내보내주는지는 riscv와 동일하게 되도록 위에 엑셀 파일로 정리하였다.
- [isBRANCH] BRANCH라는 control signal과 ALUR_OUT의 lsb를 and 하여 나온 output을 보내준다. BRANCH는 instruction이 branch인지 여부를 알려주는 control signal이고, isBRANCH의 output은 instruction이 branch이면서 사실이기에 pc+4가 아닌 알맞는 값이 들어가야하는 경우이다. 이 특수 경우를 제외하고는 isBRANCH의 값이 0이므로 mux에서 pc+4의 값이 들어가게 하였다.

- [isJALR] JALR instruction의 경우에는 연산을 진행하고 마지막 bit을 0으로 만들어야하는데, 이를 진행하기 위해 만든 module이다. control signal에서 JALR instruction이면 1을 아니면 0을 내보내주고, 이에 알맞게 1이면 lsb를 0으로 아니면 원래대로 나가게 해준다.
- [LOADER] Load instruction 의 경우, LB, LH, LW 등 무슨 instruction인지에 따라 몇 bit가 load되는지 그리고 어떻게 그 나머지 부분들이 채워지는지가 결정된다. 이를 위해 만든 모듈로, 어떤 load instruction인지 control signal로 받고, 그에 맞게 load 되어야하는 값을 변환하여 output으로 내보낸다.
- [ONEBITMUX] signal로 0인지 1인지 넣어주어 2개의 input 중에 어떤 output이 나갈지 결정해주는 mux module이다.
- [TWOBITMUX] signal로 2bit input을 넣어주어 (00, 01, 10, 11) 4개의 input 중에 어떤 output이 나갈지 결정해주는 mux module이다.
- [OUTPUT] output으로 무엇이 나갈지 결정해주었다. 우선적으로 ISBRANCH가 1이면 branch instruction이고 참이라는 것이므로 output을 1로 해주었고, ISBRANCH = 0이지만 BRANCH의 값이 1이면, branch instruction이지만 true가 아닌 것이므로 output=0으로 해주었다. D_MEM_WEN이 0이면 store instruction이므로, data memory address가 나가게 하였고 이외의 경우에 대해서는 RFWD의 값이 나가게 하였다.
- [PC] CLK의 positive edge에 마다 RSTn이 0이면 output을 0으로 아니면 들어온 pc 값이 그대로 나가게 하였다.
- [TRANSLATE] effective address를 target address로 변환해주어 physical memory에 접근하는 address의 값이 알맞게 나오도록 변환해주는 과정을 해주는 모듈이다. WHICH signal 값에 의해 instruction memory인지 data memory인지 결정해주고, 각자에 알맞게 address 값을 넣어주었다.
- [RISCV_TOP.v] 여기서는 위에 만든 모듈들을 필요한 만큼 선언하고 만들어서 wire로 연결시키게 하였다. 외부에 선언된 memory, register 그리고 clocking과도 연결시켜준다.

EVALUATION

- 3개의 file에 대해서 다 통과하였습니다.

```
emplate ?iverilog -o SINGLECYCLE.vvp *.v ../testbench/TB_RISCV_inst.v
 amy_hyunji@ihyeonjiui-MacBookPro ?~/Documents/GitHub/Computer_Architecture/lab3/code/t
emplate ?vvp SINGLECYCLE.vvp
WARNING: Mem_Model.v:18: $readmemh(/Users/amy_hyunji/Documents/GitHub/Computer_Architect
ure/lab3/code/testcase/hex/inst.hex): Not enough words in the file for the requested ran
ge [0:1023].
Test #    1 has been passed
Test #    2 has been passed
Test #    3 has been passed
Test #    4 has been passed
Test #    5 has been passed
Test #    6 has been passed
Test #    7 has been passed
Test #    8 has been passed
Test #    9 has been passed
Test #   10 has been passed
Test #   11 has been passed
Test #   12 has been passed
Test #   13 has been passed
Test #   14 has been passed
Test #   15 has been passed
Test #   16 has been passed
Test #   17 has been passed
Test #   18 has been passed
Test #   19 has been passed
Test #   20 has been passed
Test #   21 has been passed
Test #   22 has been passed
Test #   23 has been passed
Finish:          24 cycle
Success.
 amy_hyunji@ihyeonjiui-MacBookPro ?~/Documents/GitHub/Computer_Architecture/lab3/code/t
emplate ?iverilog -o SINGLECYCLE.vvp *.v ../testbench/TB_RISCV_forloop.v
 amy_hyunji@ihyeonjiui-MacBookPro ?~/Documents/GitHub/Computer_Architecture/lab3/code/t
emplate ?vvp SINGLECYCLE.vvp
WARNING: Mem_Model.v:18: $readmemh(/Users/amy_hyunji/Documents/GitHub/Computer_Architect
ure/lab3/code/testcase/hex/forloop.hex): Not enough words in the file for the requested
range [0:1023].
Finish:          73 cycle
Success.
 amy_hyunji@ihyeonjiui-MacBookPro ?~/Documents/GitHub/Computer_Architecture/lab3/code/t
emplate ?iverilog -o SINGLECYCLE.vvp *.v ../testbench/TB_RISCV_sort.v
 amy_hyunji@ihyeonjiui-MacBookPro ?~/Documents/GitHub/Computer_Architecture/lab3/code/t
emplate ?vvp SINGLECYCLE.vvp
WARNING: Mem_Model.v:18: $readmemh(/Users/amy_hyunji/Documents/GitHub/Computer_Architect
ure/lab3/code/testcase/hex/sort.hex): Not enough words in the file for the requested ran
ge [0:1023].
Finish:       9408 cycle
Success.
```

- debugging을 위해 주어진 comment의 instruction을 해석하고 그 instruction의 path를 역으로 추정하여 결과를 display하는 방식과 testcase를 개별적으로 만들어서 해보는 방식으로 진행했습니다.

DISCUSSION

- 이전 2개의 랩은 디버깅하는 게 어렵지 않았는데, 이번 경우에는 module도 많아지고 이를 같이 엮은 결과를 보는 것이었기 때문에 어디서 에러가 났는지 찾는데 어려움을 겪었습니다.

CONCLUSION

- 이번 랩도 디버깅하고 코드를 짜는데 어려움이 많았는데, 앞으로의 랩들에 대한 걱정이 생기는 시간이었습니다. 클라썸에 질문 답변 빨리 해주셔서 감사합니다!