

Team project 보고서

-Mu0 processor-

20160458 김지우

20160521 유영미

20170629 정소연

<목차>

1. Introduction

-roles of each team member

2. Processor block diagram

-components and detailed signals

3. Verification of instructions

-Simulation plan

-Modelsim simulation results and analysis

-Analysis of assembly code and the number of operation cycles

4. Synthesis results

5. Conclusion

1. Introduction

-roles of each team member

20160458 김지우

- FSM 설계
- Mu0 processor 설계
- 확장된 Mu0 processor 설계 및 datapath 통합
- Mu0 processor와 확장된 Mu0 processor 오류 검사 및 보안

20160521 유영미

- Mu0 processor 설계(datapath)
- 확장된 Mu0 processor 설계(datapath)
- Mu0 processor와 확장된 Mu0 processor 오류 검사 및 보안
- PPT 작성

20170629 정소연

- Mu0 processor 설계
- 확장된 Mu0 processor 설계
- Instruction 및 sigma simulation 분석
- 보고서 작성

2. Processor block diagram

2.1 메모리 동작

메모리와 Mu0 processor를 연결하는 databus를 inout port로 사용하였다. Mu0 processor에서 acc_oe가 1이 될 때만 databus는 output으로 사용되고, 나머지는 high impedance로 동작된다. Acc_oe가 1이 되는 경우는 opcode가 0001 즉, STO 명령어일 때 databus가 output이 되고, acc 레지스터 값을 메모리에 저장한다.

2.2 기존 Mu0 processor

-입출력 포트

- inout port

databus[15:0]: STO 명령어일 때만 output port로 사용되며 메모리에 값을 저장한다. 나머지 명령어일 때는 input port로 사용되며 memory에서 나와서 register에 들어간다.

- input port

clk: Mu0 processor의 기준이 되는 clk 값으로 1bit scalar값이다.

rst: Mu0 processor를 reset하는 signal

- output port (Memory control signals)

mem_rq: memory request

r n w: memory read(1)/write(0)

- output port (address pass)

a_out[11:0]: memory의 주소 값(memory에서는 addr[11:0]으로 input으로 사용된다.)

- Datapath control signals

a_sel: a_mux(address) select control

b_sel: b_mux(alu's b operand) select control

acc_ce: accumulator change enable

pc_ce: pc change enable

ir_ce: ir change enable

acc_oe: accumulator output enable

alu_fs[2:0]: alu function select

Operation	alu_fs[2:0]
0	000
A+B	001
A-B	010
B	011
B+1	100

- Control logic state

EX/FT: execute(0)/fetch(1)

- Control logic state inputs from datapath

opcode: operation code(databus의 상위 4bit)

acc_msb: acc에 저장된 값이 음수면 1, 양수면 0

acc_z: acc에 저장된 값이 0이면 1, 아니면 0

- Register

alu[15:0]: performs a number of operations on binary operands, such as add, subtract, increment, ...

acc[15:0]: holds a data value while it is worked upon

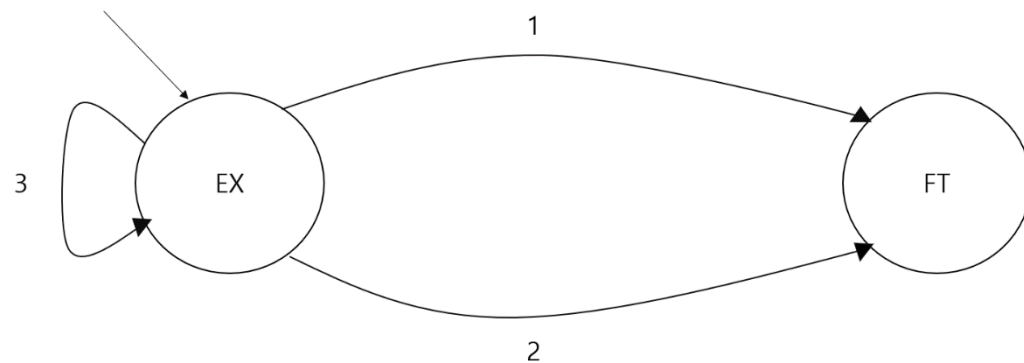
ir[15:0]: holds the current instruction while it is executed

pc[11:0]: holds the address of the current instruction

-FSM 설계

Input: opcode, acc_z, acc_msb

Output: a_sel, b_sel, acc_ce, pc_ce, ir_ce, acc_oe, alu_fs, mem_rq, rnw



1) rst=1일 때 나머지 input은 don't care

output {a_sel, b_sel, acc_ce, pc_ce, ir_ce, acc_oe, alu_fs, mem_rq, rnw}

=11'b0_0_1_1_1_1_0_000_1_1

2) rst=0일 때

① EX->FT

input				Output									
opcode	acc_z	acc_msb	comp_sn	a_sel	b_sel	acc_ce	pc_ce	ir_ce	acc_oe	alu_fs		mem_rq	rnw
0000(LDA)	x	x	x	1	1	1	0	0	0	011	B	1	1
0001(STO)	x	x	x	1	x	0	0	0	1	x	x	1	0
0010(ADD)	x	x	x	1	1	1	0	0	0	001	A+B	1	1
0011(SUB)	x	x	x	1	1	1	0	0	0	010	A-B	1	1

② FT->EX

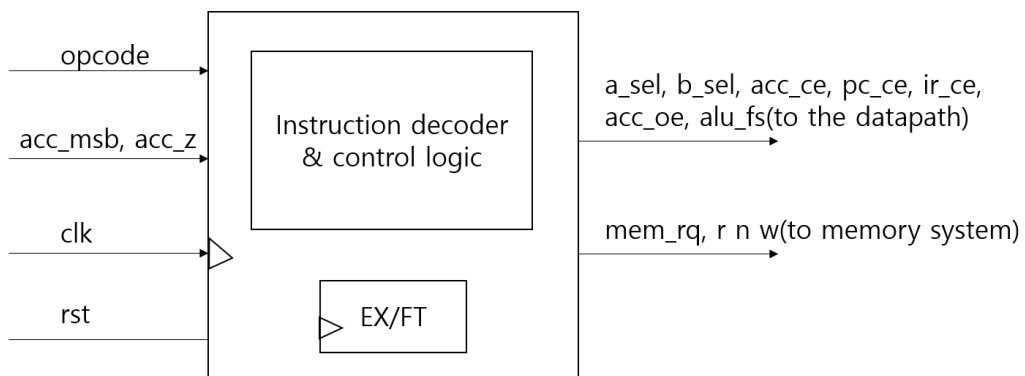
input				Output									
opcode	acc_z	acc_msb	comp_sn	a_sel	b_sel	acc_ce	pc_ce	ir_ce	acc_oe	alu_fs		mem_rq	rnw
0000(LDA)	x	x	x	0	0	0	1	1	0	100	B+1	1	1
0001(STO)	x	x	x	0	0	0	1	1	0	100	B+1	1	1
0010(ADD)	x	x	x	0	0	0	1	1	0	100	B+1	1	1
0011(SUB)	x	x	x	0	0	0	1	1	0	100	B+1	1	1

③ EX->EX

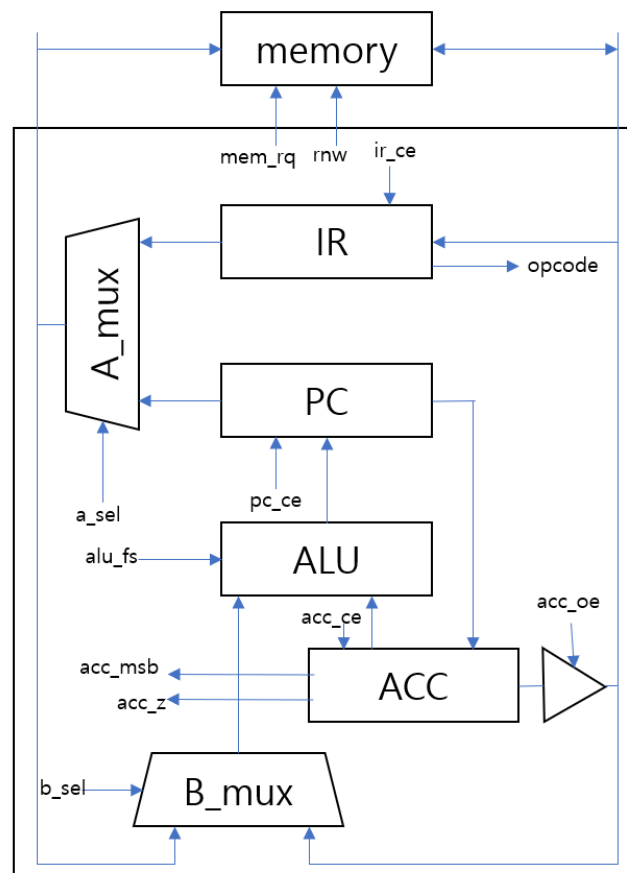
input				Output									
opcode	acc_z	acc_msb	comp_sn	a_sel	b_sel	acc_ce	pc_Ce	ir_Ce	acc_oe	alu_fs		mem_rq	rnw
0100(JMP)	x	x	x	1	0	0	1	1	0	100	B+1	1	1
0101(JGE)	x	0	x	1	0	0	1	1	0	100	B+1	1	1
0101(JGE)	x	1	x	0	0	0	1	1	0	100	B+1	1	1
0110(JNE)	0	x	x	1	0	0	1	1	0	100	B+1	1	1
0110(JNE)	1	x	x	0	0	0	1	1	0	100	B+1	1	1
0111(STP)	x	x	x	1	x	0	0	0	0	x	x	0	1

-시스템 블록 정의

Control logic



Datapath



2.3 확장된 Mu0 processor

-입출력포트

- inout port

`databus[15:0]`: STO 명령어일 때만 output port로 사용되며 메모리에 값을 저장한다. 나머지 명령어일 때는 input port로 사용되며 memory에서 나와서 register에 들어간다.

- input port

`clk`: Mu0 processor의 기준이 되는 clk 값으로 1bit scalar값이다.

`rst`: Mu0 processor를 reset하는 signal

- output port (Memory control signals)

mem_rq: memory request

r n w: memory read(1)/write(0)

- output port (address pass)

a_out[11:0]: memory의 주소 값(memory에서는 addr[11:0]으로 input으로 사용된다.)

- Datapath control signals

a_sel: a_mux(address) select control

b_sel: b_mux(alu's b operand) select control

acc_ce: accumulator change enable

pc_ce: pc change enable

ir_ce: ir change enable

acc_oe: accumulator output enable

alu_fs[2:0]: alu function select

(추가된 Datapathr control signal)

s_en: s register에 data를 저장하라는 신호

inc_en: s register에 저장된 값을 1증가하라는 신호

n_en: n register에 data를 저장하라는 신호

sum_rst: sum register에 저장된 값을 0으로 초기화하라는 신호

sum_en: sum register에 저장된 값과 s register에 저장된 값을 더하라는 신호

sum_oe: sum register에 저장된 값을 출력하라는 신호

- Control logic state

EX/FT: execute(0)/fetch(1)

- Control logic state inputs from datapath

opcode: operation code(databus의 상위 4bit)

acc_msb: acc에 저장된 값이 음수면 1, 양수면 0

acc_z: acc에 저장된 값이 0이면 1, 아니면 0

comp_sn: s와 n에 저장된 값을 비교해서 같으면 1, 다르면 0

- Register

alu[15:0]: performs a number of operations on binary operands, such as add, subtract, increment, ...

acc[15:0]: holds a data value while it is worked upon

ir[15:0]: holds the current instruction while it is executed

pc[11:0]: holds the address of the current instruction

(sigma 연산을 위해 추가된 register)

s[15:0]: LDSA 명령어가 실행되면 acc register에 저장되는 것과 같은 값이 저장된다. 또한 SUM 명령어가 실행되면 n register에 저장된 값과 다르다면 s register에 저장되어 있는 값이 1씩 증가한다.

n[15:0]: LDN 명령어가 실행되면 acc register에 저장되는 것과 같은 값이 저장된다.

sum[15:0]: LDSA 명령어가 실행되면 이 register를 0으로 초기화 해서 시그마 연산을 할 준비를 한다. 그리고 SUM 명령어가 실행되면 s, n register의 값이 같지 않으면 시그마 연산을 한 값이 계속 여기에 저장된다.

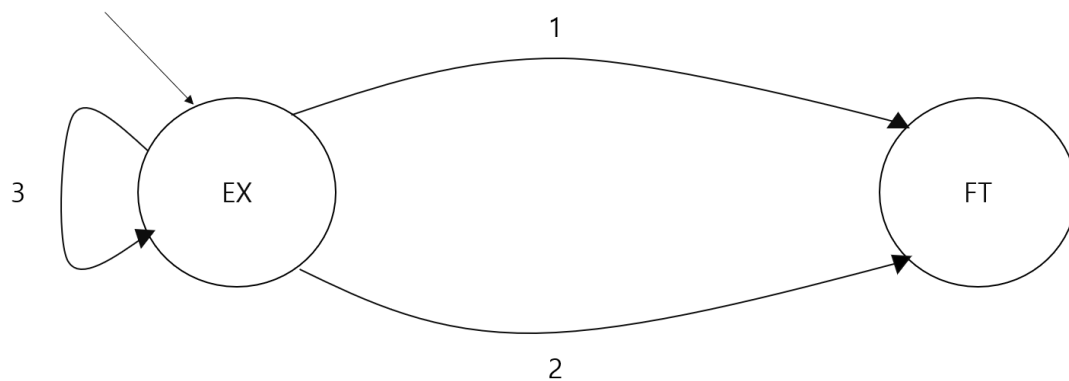
-추가된 명령어

Instructions	opcode	Effect
LDSA S	1000	s:=mem[S], sum:=0
LDN S	1001	n:=mem[S]
SUM	1010	sum:=sum+s, s:=s+1 If s!=n pc:=S
JLT S	1011	If s!=n pc:=S
STS S	1100	mem[S]:=sum

-FSM 설계

Input: opcode, acc_z, acc_msb

Output: a_sel, b_sel, acc_ce, pc_ce, ir_ce, acc_oe, alu_fs, mem_rq, rnw
s_en, n_en, inc_en, sum_rst, sum_en, sum_oe



1) rst=1일 때 나머지 input은 don't care

output {a_sel, b_sel, acc_ce, pc_ce, ir_ce, acc_oe, alu_fs, mem_rq, r n w}

=11'b0_0_1_1_1_1_0_000_1_1

2) rst=0일 때

① EX->FT

input				Output									
opcode	acc_z	acc_msb	comp_sn	a_sel	b_sel	acc_ce	pc_ce	ir_ce	acc_oe	alu_fs		mem_rq	rnw
0000(LDA)	x	x	x	1	1	1	0	0	0	011	B	1	1
0001(STO)	x	x	x	1	x	0	0	0	1	x	x	1	0
0010(ADD)	x	x	x	1	1	1	0	0	0	001	A+B	1	1
0011(SUB)	x	x	x	1	1	1	0	0	0	010	A-B	1	1
1000(LDSA)	x	x	x	1	1	1	0	0	0	011	B	1	1
1001(LDN)	x	x	x	1	1	1	0	0	0	011	B	1	1
1100(STS)	x	x	x	1	x	0	0	0	0	x	x	1	0

<추가된 output(datapath control signal)>

input				output					
opcode	acc_z	acc_msb	comp_sn	s_en	n_en	inc_en	sum_rst	sum_en	sum_oe
0000(LDA)	x	x	x	0	0	0	0	0	0
0001(STO)	x	x	x	0	0	0	0	0	0
0010(ADD)	x	x	x	0	0	0	0	0	0
0011(SUB)	x	x	x	0	0	0	0	0	0
1000(LDSA)	x	x	x	1	0	0	1	0	0
1001(LDN)	x	x	x	0	1	0	0	0	0
1100(STS)	x	x	x	0	0	0	0	0	1

② FT->EX

input				Output									
opcode	acc_z	acc_msb	comp_sn	a_sel	b_sel	acc_ce	pc_ce	ir_ce	acc_oe	alu_fs		mem_rq	rnw
0000(LDA)	x	x	x	0	0	0	1	1	0	100	B+1	1	1
0001(STO)	x	x	x	0	0	0	1	1	0	100	B+1	1	1
0010(ADD)	x	x	x	0	0	0	1	1	0	100	B+1	1	1
0011(SUB)	x	x	x	0	0	0	1	1	0	100	B+1	1	1
1000(LDSA)	x	x	x	0	0	0	1	1	0	100	B+1	1	1
1001(LDN)	x	x	x	0	0	0	1	1	0	100	B+1	1	1
1100(STS)	x	x	x	0	0	0	1	1	0	100	B+1	1	1

<추가된 output(datapath control signal)>

{s_en, n_en, inc_en, sum_rst, sum_en, sum_oe}<=6'b000000

③ EX->EX

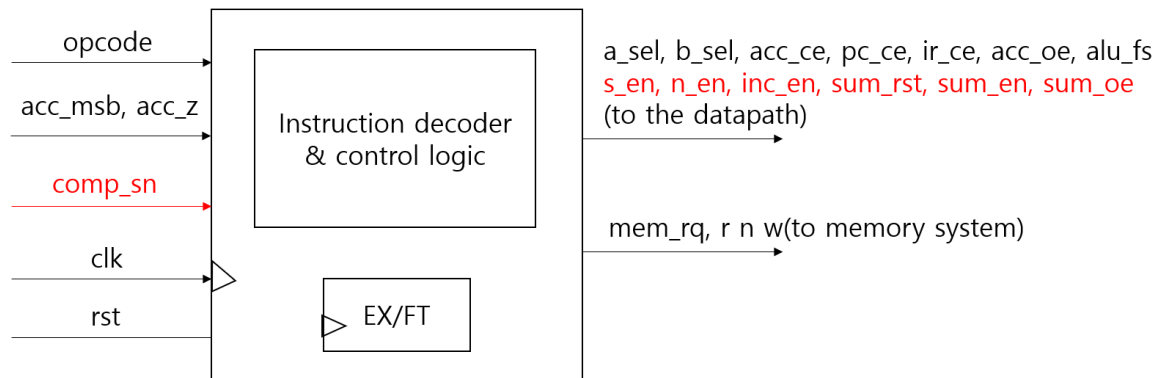
input				Output									
opcode	acc_z	acc_msb	comp_sn	a_sel	b_sel	acc_ce	pc_Ce	ir_Ce	acc_oe	alu_fs		mem_rq	rnw
0100(JMP)	x	x	x	1	0	0	1	1	0	100	B+1	1	1
0101(JGE)	x	0	x	1	0	0	1	1	0	100	B+1	1	1
0101(JGE)	x	1	x	0	0	0	1	1	0	100	B+1	1	1
0110(JNE)	0	x	x	1	0	0	1	1	0	100	B+1	1	1
0110(JNE)	1	x	x	0	0	0	1	1	0	100	B+1	1	1
0111(STOP)	x	x	x	1	x	0	0	0	0	x	x	0	1
1010(SUM)	x	x	1	0	0	0	1	1	0	100	B+1	1	1
1010(SUM)	x	x	0	1	0	0	1	1	0	100	B+1	1	1
1011(JLT)	x	x	1	0	0	0	1	1	0	100	B+1	1	1
1011(JLT)	x	x	0	1	0	0	1	1	0	100	B+1	1	1

<추가된 output(datapath control signal)>

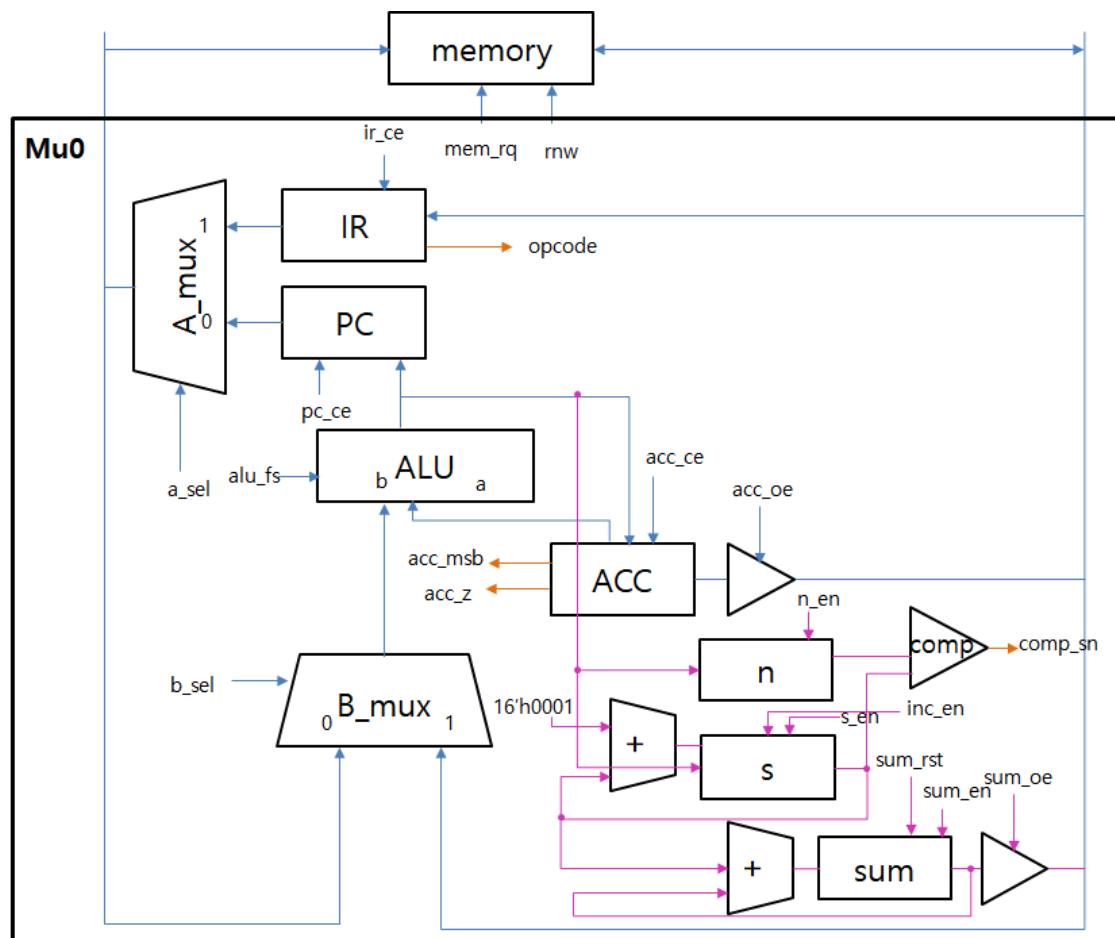
input				output					
opcode	acc_z	acc_msb	comp_sn	s_en	n_en	inc_en	sum_rst	sum_en	sum_oe
0100(JMP)	x	x	x	0	0	0	0	0	0
0101(JGE)	x	0	x	0	0	0	0	0	0
0101(JGE)	x	1	x	0	0	0	0	0	0
0110(JNE)	0	x	x	0	0	0	0	0	0
0110(JNE)	1	x	x	0	0	0	0	0	0
0111(STOP)	x	x	x	0	0	0	0	0	0
1010(SUM)	x	x	1	0	0	0	0	0	0
1010(SUM)	x	x	0	0	0	1	0	1	0
1011(JLT)	x	x	1	0	0	0	0	0	0
1011(JLT)	x	x	0	0	0	0	0	0	0

-시스템 블록 정의

Control logic



Datapath



3. Verification of instruction

3.1 기존 Mu0 processor

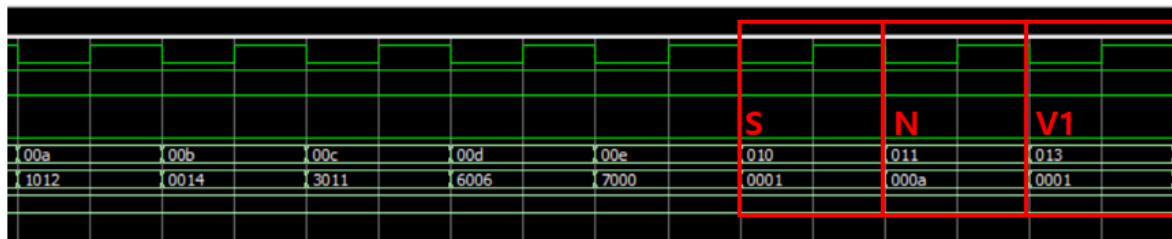
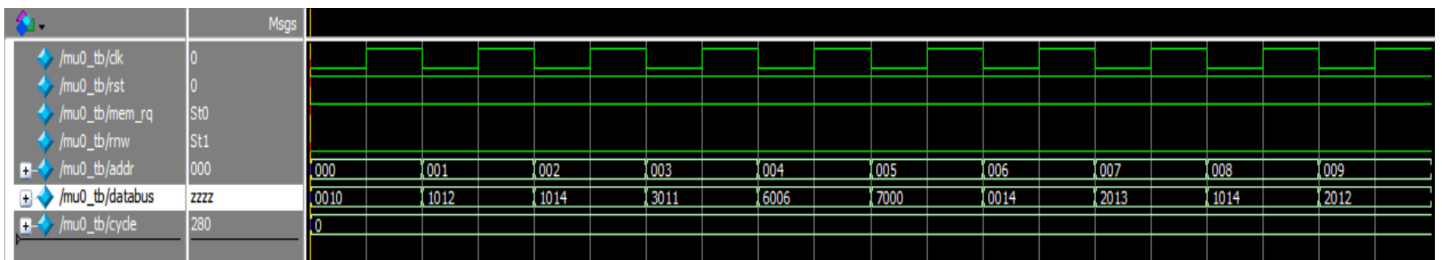
-simulation plan

address		opcode	S
0	LDA S	0000	0000_0001_0000
1	STO sum	0001	0000_0001_0010
2	STO I	0001	0000_0001_0100
3	SUB N	0011	0000_0001_0001
4	JNE loop1	0110	0000_0000_0110
5	STP	0111	0000_0000_0000
6(loop1)	LDA I	0000	0000_0001_0100
7	ADD V1	0010	0000_0001_0011
8	STO I	0001	0000_0001_0100
9	ADD sum	0010	0000_0001_0010
10	STO sum	0001	0000_0001_0010
11	LDA I	0000	0000_0010_0100
12	SUB N	0011	0000_0001_0011
13	JNE loop1	0110	0000_0000_0110
14	STP	0111	0000_0000_0000
15		xxxx	x
16(S)		0000	0000_0000_0001
17(N)		0000	0000_0000_1010
18(sum)		xxxx	x
19(I)		xxxx	x
20(V1)		0000	0000_0000_0001
....		xxxx	x
31		xxxx	x

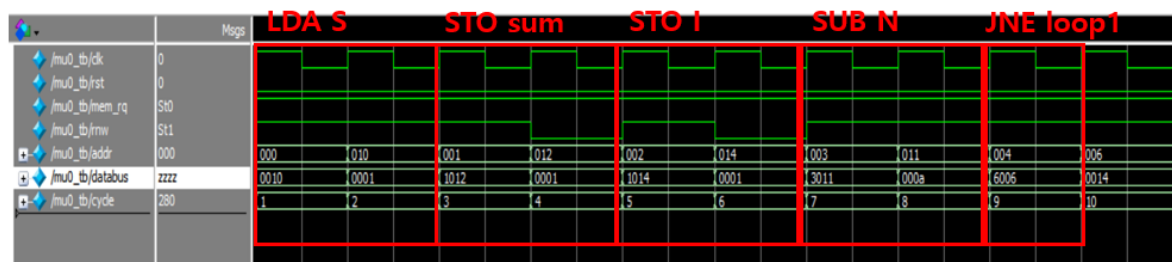
테스트 벤치를 통해 위와 같이 데이터를 넣어주었다. 여기서 S, N, sum, I, V1은 주소 값으로 12'h010, 12'h011, 12'h012, 12'h013, 12'h014로 정해주었다. 또한 각각의 주소가 가리키는 메모리에는 S=16'h0001, N=16'h000A, V1=16'h0001을 넣어주었다. rst가 0이 되면서 프로세서가 실행되면 mem[S]에 있는 값을 acc에 저장해준 다음 mem[sum]과 mem[I]에 acc에 저장되어 있는 mem[S]값인 16'h0001가 저장된다. 그 다음 acc에 저장되어 있는 16'h0001와 mem[N]에 있는 값을 빼서 acc에

저장해준다. 그 값이 0과 같지 않으면 loop1이 가리키는 주소로 가게 된다. Mem[I]에 있는 값을 acc에 저장한다. acc에 acc에 저장된 값과 mem[V1]에 있는 값을 더한 후 저장한다. 그 값을 mem[I]에 저장한다. 다음 acc에 저장된 값과 mem[sum]의 값을 더한 후 acc에 저장한다. 그 값을 mem[sum]에 저장한다. 그 후에 다시 LDA I를 하면 acc에 mem[I]값이 저장되는데 이 때 mem[I]의 값은 16'h0001+16'h0001의 값인 16'h0002이다. 이 값과 mem[N]의 값을 뺀 후 0과 같지 않으면 또 loop1이 가리키는 곳으로 간다. 따라서 이 동작이 mem[I]의 값이 mem[N]의 값과 같을 때까지 반복된다.

-simulation 결과 및 분석

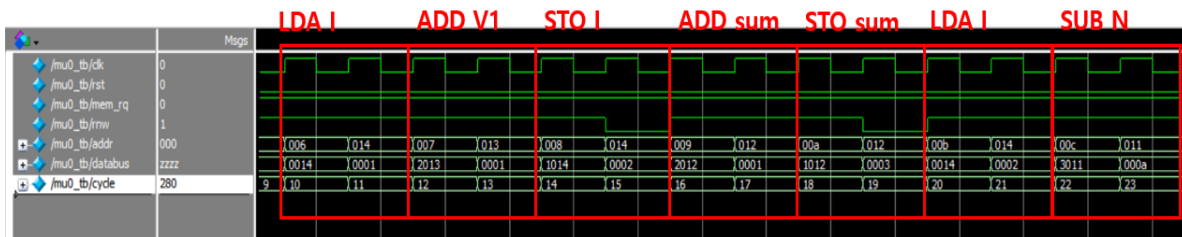


처음에 주소 값이 12'h000인 곳부터 12'h00e인 곳과, S, N, V1으로 정해진 12'h010, 12'h011, 12'h013에 테스트 벤치에서 지정해준 데이터가 잘 들어갔음을 확인할 수 있다.

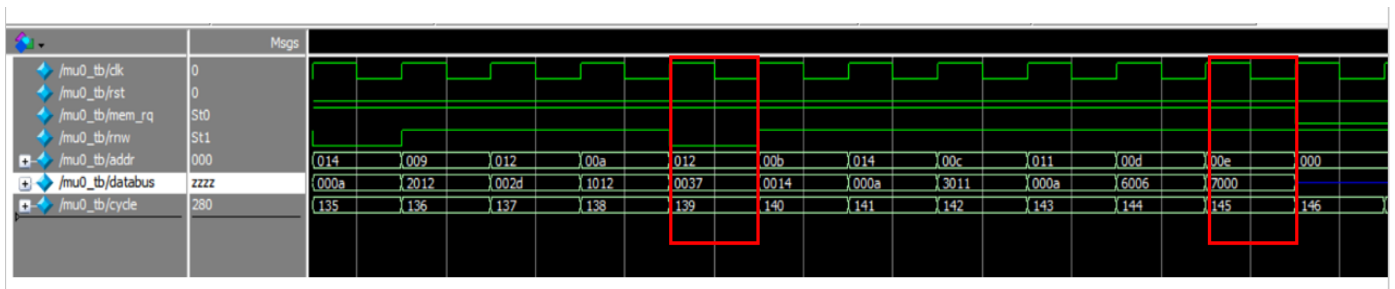


rst가 0이고, clk의 rising edge일 때 cycle이 증가하기 시작하고, LDA S, STO sum,

STO I, SUB N, JNE loop1이 잘 실행되었음을 확인할 수 있다.



JNE loop1일 때 acc register에 저장된 값이 0이 아니기 때문에 loop1번지인



12'h006으로 점프했고, 그 다음부터는 시그마 연산을 하기 위한 반복이 계속된다.

1부터 10까지 더하는 summation을 하고, 139cycle에서 다 더한 결과값인 16'h0037 즉 55가 나왔고, 145cycle이 지나면 STOP되는 것을 확인할 수 있다.

-Analysis of assembly code and the number of operation cycles

LDA, STO, STO, SUB 각각 2cycle씩 걸리고, 1과 10을 로드해서 비교하는 JNE는 1cycle 걸려 총 9cycle 걸린다. JNE에 의해서 loop1번지로 점프하기 때문에 STP는 실행되지 않는다. Loop1번지에 가서는 LDA, ADD, STO, ADD, STO, LDA, SUB 각 2cycle씩 걸리고, JNE는 1cycle걸려서 한번 루프를 돌 때 15cycle걸리는 것이 총 9번 반복되어 135cycle 걸린다. 따라서 마지막 STP 1cycle까지 계산하면 총 145cycle이 걸립니다. 이 계산 결과는 우리가 설계한 Mu0 processor로 simulation한 결과와 같음을 확인할 수 있다.

3.2 확장 Mu0 processor

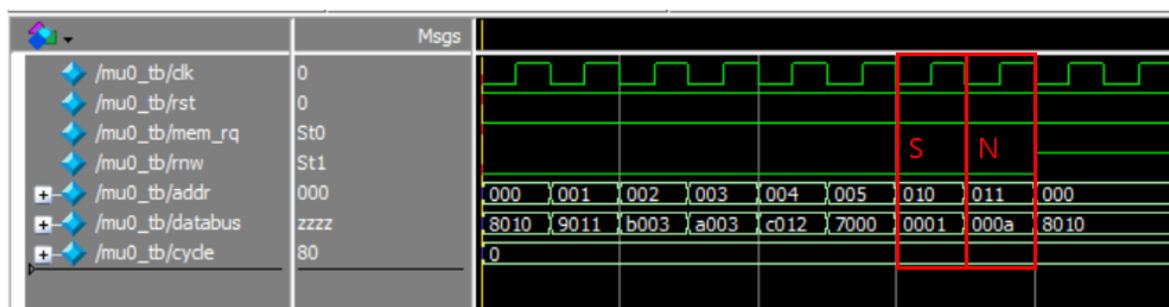
-simulation plan

address		opcode	S
0	LDSA S	1000	0000_0001_0000
1	LDN N	1001	0000_0001_0001
2	JLT LOOP1	1011	0000_0000_0011
3(LLOOP1)	SUM LOOP1	1010	0000_0000_0011
4	STS RESULT	1100	0000_0001_0010
5	STP	0111	0000_0000_0000
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16(S)		0000	0000_0000_0001
17(N)		0000	0000_0000_1010
18(RESULT)		x	x
19			
20			
....			
31			

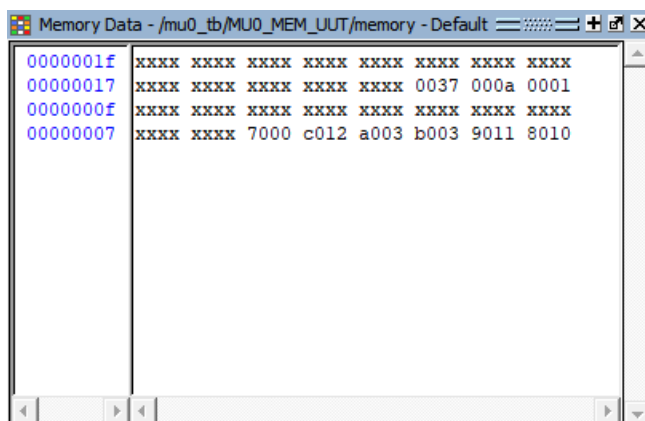
메모리 주소 0번지에 LDSA S(12'h010), 1번지에 LDN N(12'h011), 2번지에 JLT LOOP1(12'h003), 3번지에 SUM LOOP1(12'h003), STS RESULT(12'h013), 5번지에 STP를 넣어줬다. 그리고 S번지에 16'h0001, N번지에 16'h000A를 넣어줬다. 따라서 rst값이 1이되면 0번지에 있는 명령어 LDSA가 실행되면서 메모리 S번지에 있는 값을 s register에 저장하고, sum register의 값을 0으로 초기화한다. 다음 명령어 LDN이 실행되면 메모리 N번지에 있는 값을 n register에 저장하고, JLT 명령어를

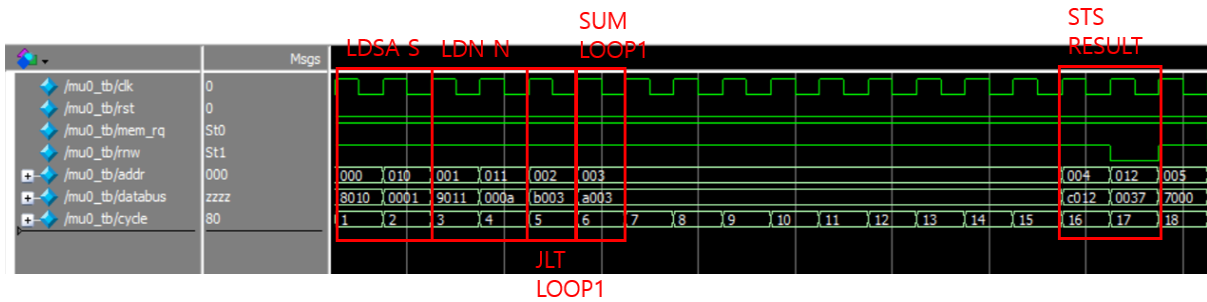
만나서 s register 값과 n register 값이 다르기 때문에 LOOP1번지로 점프한다. SUM 명령어를 만나면 s와 n값을 비교해서 다르면, s값을 1 증가하고, sum에는 s 값을 더한다. 그리고 LOOP1번지로 다시 점프한다. 이 동작을 10번 반복하게 된다. 왜냐하면 처음에 s값이 1이고, n값은 10이기 때문에 s값이 증가해서 n값과 같아지려면 10번 반복해야 하기 때문이다. 따라서 sum register에는 1부터 10까지의 값이 더해진 55가 저장되어 있을 것이다. 10번 반복 후 만난 명령어 STS는 ACC에 sum register값을 저장하라는 명령어이기 때문에 ACC에 55값이 저장되고 프로세서는 종료될 것이다.

-simulation 결과 및 분석



Test bench에서 작성한 대로 값이 잘 들어갔음을 확인할 수 있다. 메모리의 S번지에는 1, N번지에는 10이 잘 들어가 있음을 확인할 수 있다.






LDSA S, LDN N이 실행되면서 s register와 n register에 1과 10이 잘 들어갔음을 확인할 수 있다. 그 다음 s와 n값을 비교하는 JLT 명령어를 만나서 두 수가 같지 않으므로 LOOP1로 점프했다. LOOP1에서 SUM명령어를 만나서 s와 n값을 비교하고 같지 않으므로 다시 LOOP1로 점프할 것이고, s를 1증가하고, sum register에 s값을 더하라고 명령하고 그것의 실행은 다음 cycle에서 이루어진다. 따라서 s값이 1이고, n값이 10일 때를 2번 비교하게 되고, 따라서 SUM이 10번 반복되는 것을 확인할 수 있다. 10번 반복 후 s와 n 값이 같으므로 다음 명령어 STS를 만나게 된다. 따라서 1부터 10까지 더한 결과값인 55가 RESULT번지에 저장되는 것을 확인할 수 있고, 이는 18cycle이 걸렸다. 기존 Mu0 processor는 145cycle만에 프로세서가 종료된 것과 비교하면 127cycle이 줄었고 따라서 효율은 12.41배 좋아졌다고 할 수 있다.

-Analysis of assembly code and the number of operation cycles

LDSA, LDN이 각 2cycle씩 걸리고, JLT는 1과 10을 비교하는 동작으로 1cycle 걸리므로 여기까지 총 5cycle 걸린다. JLT를 통해서 LOOP1번지로 점프해서 SUM명령어를 만난다. SUM은 한 번 동작할 때 1cycle 소요되는데 처음 한 번은 sum_en이 0인 상태, 그 다음 cycle부터는 sum_en이 1이어서 1이 더해지고, 총 SUM명령어의 반복은 10cycle이 소요된다. 이후 STS는 2cycle, 마지막으로 STP 명령어가 1cycle걸려서 총 18cycle이 걸린다. 이는 우리가 설계한 확장 Mu0 processor의 simulation 결과와 같다.


4. Synthesis results

Flow Summary		
 <<Filter>>		
Flow Status	Successful - Mon Nov 25 11:26:42 2019	
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition	
Revision Name	mu0_process	
Top-level Entity Name	mu0_process	
Family	Cyclone V	
Device	5CGXFC7C7F23C8	
Timing Models	Final	
Logic utilization (in ALMs)	N/A	
Total registers	95	
Total pins	32	
Total virtual pins	0	
Total block memory bits	0	
Total DSP Blocks	0	
Total HSSI RX PCSs	0	
Total HSSI PMA RX Deserializers	0	
Total HSSI TX PCSs	0	
Total HSSI PMA TX Serializers	0	
Total PLLs	0	
Total DLLs	0	

Type	ID	Message

>		Running Quartus Prime Analysis & Synthesis
		Command: quartus_map --read_settings_files=on --write_settings_files=off mu0_process -c mu0_process
	18236	Number of processors has not been specified which may cause overloading on shared machines. Set the global assignment
	20030	Parallel compilation is enabled and will use 2 of the 2 processors detected
>	12021	Found 1 design units, including 1 entities, in source file /modelsim/mu0_project/mu0_process.v
	12127	Elaborating entity "mu0_process" for the top level hierarchy
	286030	Timing-driven Synthesis is running
	144001	Generated suppressed messages file D:/quartus/mu0_process/output_files/mu0_process.map.smsg
>	16010	Generating hard_block partition "hard_block:auto_generated_inst"
>	21057	Implemented 254 device resources after synthesis - the final resource count might be different
>		Quartus Prime Analysis & Synthesis was successful. 0 errors, 1 warning

확장 Mu0 processor의 mu0_process 코드 합성 결과이다.

Flow Summary	
 <<Filter>>	
Flow Status	Successful - Mon Nov 25 11:32:11 2019
Quartus Prime Version	18.1.0 Build 625 09/12/2018 SJ Lite Edition
Revision Name	mu0_memory
Top-level Entity Name	mu0_memory
Family	Cyclone V
Device	5CGXFC7C7F23C8
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	512
Total pins	32
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Type	ID	Message
> i		Running Quartus Prime Analysis & Synthesis
> i		Command: quartus_map --read_settings_files=on --write_settings_files=off mu0_memory -c mu0_memory
> i	18236	Number of processors has not been specified which may cause overloading on shared machines. Set the gl
> i	20030	Parallel compilation is enabled and will use 2 of the 2 processors detected
> i	12021	Found 1 design units, including 1 entities, in source file /modelsim/mu0_project/mu0_memory.v
> i	12127	Elaborating entity "mu0_memory" for the top level hierarchy
> i	276014	Found 1 instances of uninferred RAM logic
> i	286030	Timing-Driven Synthesis is running
> i	144001	Generated suppressed messages file D:/quartus/mu0_memory/output_files/mu0_memory.map.smsg
> i	16010	Generating hard_block partition "hard_block:auto_generated_inst"
> i	21074	Design contains 8 input pin(s) that do not drive logic
> i	21057	Implemented 737 device resources after synthesis - the final resource count might be different
> i		Quartus Prime Analysis & Synthesis was successful. 0 errors, 10 warnings

Mu0_memoery 코드의 합성 결과이다.

5. Conclusion

이번 프로젝트를 통해서 Mu0 processor를 직접 verilog code로 구현해보면서 이 프로세서의 기본 구조와 원리에 대해서 알 수 있었다. 확장 Mu0 processor를 설계하면서 시그마 연산을 효율적으로 하기 위해 LDSA, LDN, SUM, STS 명령어를 기존 프로세서에 새롭게 추가했고, 이 명령어들을 제대로 실행시키기 위한 레지스터 3개와 덧셈기, 비교기를 추가했다. 따라서 기존 프로세서에서는 불필요하게 반복하던 동작들을 확연하게 줄일 수 있었다. 이를 cycle 수를 가지고 계산해보면 기존 프로세서에 비해 확장된 mu0 processor의 속도가 8배 빨라졌다는 것을 알 수 있다.