TESTING

**Opcode\_size Test**

Purpose  
To determine that the opcode\_size function returns a correct opcode\_size; i.e the number of bytes to fetch to get the operands associated with an opcode.

Set Up  
The test is set up by running the Z8 emulator using #define OPC\_ARRAY\_TEST, the size in bytes for every opcode will be displayed to the screen.

Input  
Run Program by initializing the opc-sz array.

Expected Output  
We expect an output of 0 bytes for all the instructions with a low nibble of F, 2 bytes for instructions with low nibble 5,6,7,D and 4 except for instruction D4 which is 1byte along with the rest of the instructions

Actual Output

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 1 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

1 1 1 1 2 2 2 2 1 1 1 1 1 2 1 0

Results  
Actual results is the same as what we expected, therefore the test was successful.

**IF Test 1**

Purpose

The purpose of this test is to determine if the IF instruction successfully check the condition code to determine if true or false, and executes all its true part statements without changing the system state for a true scenario.

Set Up  
To set up this test we load s19 recs from the assembly file 1 unto the emulator, and watch the program execute by defining IF\_TEST.

Input  
S0 Test of all ADC modes with IF

S10C000C1F79FFFF3101DF1F7FA2

S1140015120113021414101515101610201717FF6F5A

S30B00100F011381801610108A

S9000c

Expected Output

We expect the Emulator to execute the 1 false instruction i.e SRP #1 when the condition is false and skip the true part after which it executes the true instruction and skips the false part when the condition is true. Also during execution, we expect the flags to stay the same.

Actual Output

Test of all ADC modes with IF

Program counter holds : c

dst : 2, src: 9

If encountered, tcount 2, fcount: 1 : cond : 0

skipping true part, tcount: 2

skipping true part, tcount: 1

start: 10 f 1 13 81 80 16 10 10 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Program counter holds : 10

RP Is set to : 1

executing false part, fcount: 1

start: 10 f 1 13 81 80 16 10 10 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0

Program counter holds : 12

carry is set

start: 10 f 1 13 81 80 16 10 10 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 80 1 0 0

Program counter holds : 13

dst : 3, src: f

If encountered, tcount 3, fcount: 3 : cond : 1

executing true part, tcount: 3

start: 10 f 1 13 81 80 16 10 10 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 80 1 0 0

Program counter holds : 15

executing true part, tcount: 2

start: 10 11 1 13 81 80 16 10 10 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 80 1 0 0

Program counter holds : 17

executing true part, tcount: 1

start: 10 92 1 13 81 80 16 10 10 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 80 1 0 0

Program counter holds : 19

Enters!!

skipping false part, fcount: 3

skipping false part, fcount: 2

skipping false part, fcount: 1

start: 10 12 1 13 81 80 16 10 10 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 80 1 0 0

Program counter holds : 25

start: 10 12 1 13 81 80 16 10 10 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 80 1 0 0

Result  
The expected output was the same as the actual output therefore the test was successful

**IF\_Test 2**

Purpose  
The purpose of this test is to determine if the IF instruction completely hands over flow of control to a JP/ CALL instruction that is executed as one of the conditional statement.

Set Up  
The test can be set up by loading s19 records from assembly file 2 in appendix A unto the Emulator machine, the #define JUMP and #define IF\_TEST is used to watch the machine as it runs.

Input  
S0 Jump and condition code + IF test

S10A000C7D2000DF7D2000D0

S11720001610011F65FF6B062610021F666B04FFFFFFFF6F16

S9000C

Expected Output  
We expect that when a jump instruction takes place in a conditionally executed statement, the flow of control will be owned by the JP instruction. We expect the fcount and tcount values to be forced to zero and for the machine to continue execution after the jump is taken without any effect from the conditionally executed statements.

Actual Output

Jump and condition code + IF test

Program counter holds : c

condition is false

start: 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Program counter holds : f

carry is set

start: 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 80 0 0 0

Program counter holds : 10

condition is true

JUMP TAKEN, NEW PC = 2000

flow of control tcount: 0 fcount: 0 cexec: 0

start: 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 80 0 0 0

Program counter holds : 2000

start: 10 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Program counter holds : 2003

condition is false

dst : 1, src: 5

If encountered, tcount 1, fcount: 1 : cond : 0

skipping true part, tcount: 1

start: 10 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Program counter holds : 2006

condition is false

executing false part, fcount: 1

start: 10 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Program counter holds : 2008

start: 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 cc 0 0 0

Program counter holds : 200b

condition is true

dst : 1, src: 6

If encountered, tcount 1, fcount: 2 : cond : 1

executing true part, tcount: 1

start: 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 cc 0 0 0

Program counter holds : 200d

condition is true

pc before jump is : 200f

contents of prog mem(pc) is ff

JUMP TAKEN: dst = 4 , pc = 2013

contents of prog mem(pc) is 6f

flow of control tcount: 0 fcount: 0 cexec: 0

start: 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 cc 0 0 0

Program counter holds : 2013

start: 10 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

start: f0 0 0 0 0 0 0 0 0 0 0 0 0 cc 0 0 0

Results  
The expected output was the same as the actual output therefore the test was successful

APPENDIX A –ASSEMBLY FILES

**Assembly file 1**  
All ADC addressing modes

SECTION PROGRAM

ORIGIN %000C

!carry is initially clear

IF C, 2,1

NOP

NOP

!condition is false, so it won’t do the true part NOP's

SRP

#1

SCF

!set the carry flag

IF C, 3,3

! true part is executed, false part is skipped

ADC

R0,R1 !Add 0x01 in R1 to 0x0F in R0

ADC

R0,@R2

!R2 holds 0x13 (R3) - Add 0x81 in R3 to R0

ADC,

%10,%14

!Add 0x80 in R4 to R0

ADC

%10,@%15

!R5 holds 0x16 (R6) - Add 0x10 in R6 to R0

ADC

%10,#%20

!Add 0x20 to R0

ADC

@%17,#%FF

!R7 holds 0x10 (R0) - Add 0xFF to R0

STOP

SECTION REGISTER

ORIGIN

%0010

BVAL

%0F

BVAL

%01

BVAL

%13

BVAL

%81

BVAL

%80

BVAL

%16

BVAL

%10

BVAL

%10

END

S0 Test of all ADC modes with IF

S10C000C1F79FFFF3101DF1F7FA2

S1140015120113021414101515101610201717FF6F5A

S30B00100F011381801610108A

S9000c

**Assembly file 2**

JP/JR and Condition Codes

SECTION PROGRAM

ORIGIN %000C

!Carry flag is initially clear

JP

C,%2000

!Jump to 0x2000 if Carry flag is set

!This jump should NOT be taken

SCF

!Set the carry flag

JP

C,%2000

!Jump to 0x2000 if Carry flag is set

!This time the jump SHOULD be taken

ORIGIN

%2000

!When the JP instruction is executed, the program should run from here

ADC

%10,#1

!Add 1 to R0 - the Zero flag will NOT be set

IF Z, 1,1

!The false part will be executed since the zero flag is not set

JR

Z,%04

!This jump should NOT be taken

NOP

SUB

%10,#2

!Subtract 2 from R0 - should SET Zero flag

IF Z, 1,2

! Zero flag is set so true part should be executed

JR

Z,%04

!IF Zero flag set, jump to PC + 4

!This jump SHOULD be taken

!Jump take over control of flow

!terminates execution of conditionally executed statements

NOP

NOP

NOP

NOP

STOP

!Jump should skip NOPs and execute STOP

END

S0 Jump and condition code + IF test

S10A000C7D2000DF7D2000D0

S11720001610011F65FF6B062610021F666B04FFFFFFFF6F16

S9000C

APPENDIX B – SOFTWARE

/\* returns no of bytes occupied by operands of a particular opcode with

MSN = hi and LSN = lo \*/

BYTE op\_size(BYTE hi, BYTE lo)

{

BYTE ans;

switch (lo)

{

case 0x04:

if(hi==0x0D)

ans=1;

else

ans =2;

break;

case 0x05:

case 0x06:

case 0x07:

case 0x0D:

ans = 2;

break;

case 0x0F:

ans =0;

break;

default:

ans =1;

break;

}

return ans;

}

/\* initializes the opcode\_size array called by the mainline program \*/

opc\_size\_init()

{

BYTE h,l;

for (h=0; h<=0x0f; h++)

{

for(l=0; l<=0x0f; l++)

{

opcode\_size[h][l]=op\_size(h,l);

}

}

#ifdef OPC\_ARRAY\_TEST

for (h=0; h<=0x0f; h++)

{

for(l=0; l<=0x0f; l++)

{

printf( "%2x ",opcode\_size[h][l]);

}

printf("\n");

}

#endif

}

///////////////Instruction Decode ////////////////////////////////////////

lownib>=8

switch lownib

case 0x0F

switch highnib

case 0x01: /\* IF instuction \*/

/\* fetch next byte \*/

dst = prog\_mem\_fetch();

regval = MSN(dst); // condition code

src = LSN(dst);

dst = src>>2;

tcount = dst&0x03; // retrieve tt

fcount = src&0x03; // retrieve ff

cexec = cond\_handler(regval); // holds 1 if condition is true, and 0 if false

/\* store flags in temp variable \*/

tempflags = read\_rm(FLAGS);

#ifdef TEST\_IF

printf("If encountered, tcount %1x, fcount: %1x : cond : %x \n", tcount, fcount, cexec);

#endif

break;

/\* Inserted into JUMP sections of the JP, JR, DJNZ and CALL instructions \*/

/\*\*\*\*\*\*\*\*\*\*\*IF ADDITION \*\*\*\*\*\*\*\*\*\*\*\*\*/

tcount =0;

fcount=0;

cexec=0;

#ifdef IF\_TEST

printf("flow of control tcount: %x fcount: %x cexec: %x \n", tcount, fcount, cexec);

#endif

/\*\*\*\*\*\*\*\*\*\*\*\*END of instruction CYCLE \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*IF ADDITION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*IF ADDITION \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

int i;

if(cexec) // cexec is 1

{

if(tcount>0x00 && tcount<0x04) //sanity check

{

#ifdef IF\_TEST

printf("executing true part, tcount: %x \n", tcount);

#endif

/\* execute true part of the statement \*/

tcount--; // decrement the count of true instructions

/\* restore flags \*/

write\_rm(FLAGS, tempflags);

}

else if(fcount>0x00 && fcount<0x04)

{

/\* do not execute false part skip over false part

increment program counter by number of bytes required per instruction \*/

for(i=fcount; i>0; i--)

{

#ifdef IF\_TEST

printf("skipping false part, fcount: %x \n", fcount);

#endif

fcount--;

inst = prog\_mem\_fetch(); // fetch opcode

pc +=opcode\_size[MSN(inst)][LSN(inst)]; // increment pc by size of operands for opcode

}

/\* restore flags \*/

write\_rm(FLAGS, tempflags);

}

}

else { // cexec == 0, execute false part only

if(tcount>0x00 && tcount<0x04)

{

for(i=tcount; i>0; i--)

{

#ifdef IF\_TEST

printf("skipping true part, tcount: %x \n", tcount);

#endif

tcount--;

inst = prog\_mem\_fetch(); // fetch opcode

pc +=opcode\_size[MSN(inst)][LSN(inst)]; // increment pc by size of operands for opcode

}

}

else if(fcount>0x00 && fcount<0x04)

{

/\* execute false part \*/

#ifdef IF\_TEST

printf("executing false part, fcount: %x \n", fcount);

#endif

fcount--; //decrement false count

/\* restore Flags \*/

write\_rm(FLAGS, tempflags);

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*ADDITION TO MAIN \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

opc\_size\_init();