

コンピュータ科学実験レポート

坪井正太郎 (101830245)

2020 年 12 月 3 日

はじめに

この実験では、一部の命令が実装されていないプロセッサに、適切な命令を実装して、関数実行、乗算除算を実行できるようにすることを目的とする。

また、各実験では、シュミレータや論理合成のソフトウェアを使うために、以下の設定を行う。端末を終了した場合、再度 source コマンドを実行する。

ソースコード 1 設定の読み込み

```
1 $ ln -s /pub1/jikken/eda3/cadsetup.bash.altera ~/
2 $ source ~/cadsetup.bash.altera
```

HDL のコンパイルには Quartus Prime を、機能レベルシュミレーションには Model Sim を使用した。

各実験

1 実験 5-1

1.1 実験の目的、概要

この実験では、実験 4-2 で作成したプロセッサで画面上に文字を表示する C プログラムを実行するその中で関数呼び出しを行い、動作を予想し、結果を確認する。

これによって、現在足りていない機能を確認することを目的とする。

1.2 実験方法

以下のプログラムを配置した。

ソースコード 2 my_print.c

```
1 #define EXTIO_PRINT_STROKE (*(volatile unsigned int *) 0x0300)
2 #define EXTIO_PRINT_ASCII (*(volatile unsigned int *) 0x0304)
3
4 void my_print();
5
6 main()
```

```

7 {
8     unsigned int string[64];
9
10    string[0] = 'H';
11    string[1] = 'E';
12    string[2] = 'L';
13    string[3] = 'L';
14    string[4] = '0';
15    string[5] = '!';
16    string[6] = '!';
17    string[7] = '\0';
18
19    my_print(string);
20
21    string[0] = 'B';
22    string[1] = '\0';
23 }
24
25 void my_print(str)
26     unsigned int *str;
27 {
28     while (*str != '\0') {
29         EXTIO_PRINT_STROKE = (unsigned int)0x00000000;
30
31         if ((*str >= 'A') && (*str <= 'Z')) {
32             EXTIO_PRINT_ASCII = *str - 'A' + 1;
33         } else if ((*str >= 'a') && (*str <= 'z')) {
34             EXTIO_PRINT_ASCII = *str - 'a' + 1;
35         } else if ((*str >= '0') && (*str <= '9')) {
36             EXTIO_PRINT_ASCII = *str - '0' + 48;
37         } else {
38             if (*str == '@') {
39                 EXTIO_PRINT_ASCII = (unsigned int)0;
40             } else if (*str == '[') {
41                 EXTIO_PRINT_ASCII = (unsigned int)27;
42             } else if (*str == ']') {
43                 EXTIO_PRINT_ASCII = (unsigned int)29;
44             } else if ((*str >= ' ' ) && (*str <= '/')) {
45                 EXTIO_PRINT_ASCII = *str - ' ' + 32;
46             } else if (*str == '?') {
47                 EXTIO_PRINT_ASCII = (unsigned int)58;
48             } else if (*str == '=') {
49                 EXTIO_PRINT_ASCII = (unsigned int)59;
50             } else if (*str == ';') {
51                 EXTIO_PRINT_ASCII = (unsigned int)60;
52             } else if (*str == ':') {

```

```

53     EXTIO_PRINT_ASCII = (unsigned int)61;
54 } else if (*str == '\n') {
55     EXTIO_PRINT_ASCII = (unsigned int)62;
56 } else {
57     EXTIO_PRINT_ASCII = (unsigned int)0x00000000;
58 }
59 }
60
61     EXTIO_PRINT_STROKE = (unsigned int)0x00000001;
62     str++;
63 }
64 }

```

1.2.1 クロスコンパイル、メモリイメージファイルの作成

以下の操作でクロスコンパイルし、メモリイメージファイルを作成した。

ソースコード 3 クロスコンパイル、メモリイメージファイルの作成

```

1 $ cross_compile.sh my_print.c
2 $ bin2v my_print.bin

```

1.2.2 命令列の確認、動作予想

生成された、rom8x1024.mif を確認して、以下の点について結果を予測した。

- 最初に PC=0x0074 を実行した直後の REG[31] の値
- 最初に PC=0x0074 を実行した直後の PC の値

1.2.3 論理合成、ダウンロード

以下の操作で、論理合成し、FPGA にダウンロードした。

ソースコード 4 論理合成、ダウンロード

```

1 $ cp rom8x1024.mif ./mips_de10-lite/
2 $ cd ./mips_de10-lite/
3 $ quartus_sh --flow compile MIPS_Default
4 $ quartus_pgm MIPS_Default.cdf

```

クロックを手動モードで送り、70 個ほどの命令を実行、予想した点と、ディスプレイに表示されるはずの文字について確認した。

1.3 実験結果

1.3.1 命令列の確認、動作予想

メモリイメージファイルが生成された。

命令列を確認して、このような予想をたてた。

- 最初に PC=0x0074 を実行した直後の REG[31] の値
 - REG[31]=0x0078
- 最初に PC=0x0074 を実行した直後の PC の値
 - PC=0x00a0

1.3.2 FPGA での実行結果

予想した点について結果は、以下のようになった。

- 最初に PC=0x0074 を実行した直後の REG[31] の値
 - REGWRITED=00000000, WEN=0 であり、レジスタへの書き込みは発生していない
- 最初に PC=0x0074 を実行した直後の PC の値
 - PC=0x0078 であり、ジャンプはしていない

ディスプレイに文字は表示されなかった。

1.4 考察

このプロセッサには、jal 命令が実装されていないので、プログラムが正しく動作しなかった。jal 命令が正しく実装されている場合、31 番目のレジスタに次の戻り先プログラムカウンタの位置が退避されて、プログラムカウンタの位置が命令で指定された場所に更新されるはずである。

2 実験 5-2

2.1 実験の目的、概要

実験 4-2 で作成したプロセッサに jal 命令が足りないことが、実験 5-1 で確認できた。本実験では、プロセッサに jal 命令を追加実装し、動作を確認する。その際、実験 5-1 での予想と実際を比較する。

これによって、関数呼び出し時のプロセッサの動作、レジスタに保存されるデータなどを確認することを目指すとする。

2.2 実験方法

2.3 追加設計

main_ctrl.v に以下の変更を加え、jal 命令のオペコードと、実行されるとき制御信号を定義した。

ソースコード 5 main_ctrl.v の追加設計

```

1 オペコードの定義
2 'define JAL 6'b000011 // jump and link (J 形式)
3
4 jp_sel モジュールへの制御信号の記述
5 +=4の値ではなく、命令文中のアドレスを選択する
6 assign jp = ((op_code == 'J) || (op_code == 'JAL)) ? 1'b1 : 1'b0;
7
```

```
8 レジスタのwrite_enable 制御信号の追加
9 'JAL: reg_write_enable_tmp = 1'b1;
10
11 レジスタに流すデータのセレクト信号の追加
12 REG[31]に次のPC 値が保存されるようになる
13 'JAL: link_tmp = 1'b1;
```

2.3.1 論理合成、ダウンロード

以下の操作で、論理合成し、FPGA にダウンロードした。

ソースコード 6 論理合成、ダウンロード

```
1 $ quartus_sh --flow compile MIPS_Default
2 $ quartus_pgm MIPS_Default.cdf
```

実験 5-1 で予想した点と、ディスプレイに表示されるはずの文字について確認した。

2.4 実験結果

予想した点について結果は、以下のようになった。

- 最初に PC=0x0074 を実行した直後の REG[31] の値
 - REGWRITED=00400078,index=0x1f(=31),write_enable=1 が読み取れた
 - 予想通り、次に実行する PC の値が、REG[31]
- 最初に PC=0x0074 を実行した直後の PC の値
 - 予想通り PC=0x=0x00a0 であり、ジャンプしている

画面上には、"HELLO!!"という文字列が表示された。

2.5 考察

jal 命令を実装したことで、\$ra に PC の値が退避され、即値で指定した値にジャンプできるようになった。そのため、予想した点について正しい動作を確認することができた。これは、my_print 関数の呼び出しに成功しているということであると考えられる。実際に、画面上に文字列が表示されることが分かった。

この実験で、関数呼び出し時のプロセッサの動作、レジスタに保存されるデータを確認する事ができた。

3 実験 6-1

3.1 実験の目的、概要

実験 5-2 で jal 命令を追加したプロセッサ上で、キーボードからの入力を受け取る C プログラムを実行する。その中で関数からの復帰を行い、それらの動作を予想し、結果を確認する。

これによって、現在足りていない機能を確認することを目的とする。

3.2 実験方法

以下のプログラムを配置した。

ソースコード 7 my_scan.c

```
1 #define EXTIO_SCAN_ASCII (*(volatile unsigned int *)0x0310)
2 #define EXTIO_SCAN_REQ (*(volatile unsigned int *)0x030c)
3 #define EXTIO_SCAN_STROKE (*(volatile unsigned int *)0x0308)
4
5 #define SCAN_STRORING (unsigned int)0xffffffff
6
7 #define EXTIO_PRINT_STROKE (*(volatile unsigned int *)0x0300)
8 #define EXTIO_PRINT_ASCII (*(volatile unsigned int *)0x0304)
9
10 void my_print();
11 void my_scan();
12
13 main() {
14     unsigned int string1[32];
15     unsigned int string2[32];
16
17     string1[0] = 'H';
18     string1[1] = 'E';
19     string1[2] = 'L';
20     string1[3] = 'L';
21     string1[4] = '0';
22     string1[5] = '!';
23     string1[6] = '!';
24     string1[7] = '\n';
25     string1[8] = '\0';
26
27     my_print(string1);
28
29     while (1) {
30         string1[0] = 'S';
31         string1[1] = 'T';
32         string1[2] = 'R';
33         string1[3] = 'I';
34         string1[4] = 'N';
35         string1[5] = 'G';
36         string1[6] = '=';
37         string1[7] = '\0';
38
39         my_print(string1);
40
```

```

41     my_scan(string2);
42
43     string1[0] = 'E';
44     string1[1] = 'C';
45     string1[2] = 'H';
46     string1[3] = '0';
47     string1[4] = ' ';
48     string1[5] = '\0';
49
50     my_print(string1);
51
52     my_print(string2);
53
54     string1[0] = '\n';
55     string1[1] = '\0';
56
57     my_print(string1);
58 }
59 }
60
61 void my_scan(str) unsigned int *str;
62 {
63     EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
64     EXTIO_SCAN_REQ = (unsigned int)0x00000001;
65     EXTIO_SCAN_STROKE = (unsigned int)0x00000001;
66
67     EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
68     EXTIO_SCAN_STROKE = (unsigned int)0x00000001;
69     while (EXTIO_SCAN_ASCII == SCAN_STRORING) {
70         EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
71         EXTIO_SCAN_STROKE = (unsigned int)0x00000001;
72     }
73
74     while ((*str = EXTIO_SCAN_ASCII) != (unsigned int)0x3e) { // 0x3e=RETURN
75         if ((*str >= 1) && (*str <= 26)) {
76             *str = 'A' + *str - 1;
77         } else if ((*str >= 48) && (*str <= 57)) {
78             *str = '0' + *str - 48;
79         } else {
80             if (*str == 0) {
81                 *str = '@';
82             } else if (*str == 27) {
83                 *str = '[';
84             } else if (*str == 29) {
85                 *str = ']';
86             } else if ((*str >= 32) && (*str <= 47)) {

```

```

87     *str = ' ' + *str - 32;
88 } else if (*str == 58) {
89     *str = '?';
90 } else if (*str == 59) {
91     *str = '=';
92 } else if (*str == 60) {
93     *str = ';';
94 } else if (*str == 61) {
95     *str = ':';
96 } else if (*str == 62) {
97     *str = '\n';
98 } else {
99     *str = '@';
100 }
101 }
102 EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
103 EXTIO_SCAN_STROKE = (unsigned int)0x00000001;
104 str++;
105 }
106 *str = '\0';
107
108 EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
109 EXTIO_SCAN_REQ = (unsigned int)0x00000000;
110 EXTIO_SCAN_STROKE = (unsigned int)0x00000001;
111
112 EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
113 }
114
115 void my_print(str) unsigned int *str;
116 {
117     // 省略
118 }

```

3.2.1 クロスコンパイル、メモリイメージファイルの作成

以下の操作でクロスコンパイルし、メモリイメージファイルを作成した。

ソースコード 8 クロスコンパイル、メモリイメージファイルの作成

```

1 $ cross_compile.sh my_scan.c
2 $ bin2v my_scan.bin

```

3.2.2 命令列の確認、動作予想

生成された、rom8x1024.mif を確認して、以下の点について結果を予測した。

- 最初に PC=0x007c を実行したときの、REG[31] の値

- 最初に PC=0x0804 を実行した直後の , PC の値

3.2.3 キーボードの接続

今回実行するプログラムでは、キーボードからの入力が必要とするため、FPGA に変換回路を接続して、キーボードを接続した。

3.2.4 論理合成、ダウンロード

以下の操作で、論理合成し、FPGA にダウンロードした。

ソースコード 9 論理合成、ダウンロード

```
1 $ cp rom8x1024.mif ./mips_de10-lite/
2 $ cd ./mips_de10-lite/
3 $ quartus_sh --flow compile MIPS_Default
4 $ quartus_pgm MIPS_Default.cdf
```

クロックを手動モードで送り、70 個ほどの命令を実行、予想した点と、ディスプレイに表示されるはずの文字について確認した。

3.3 実験結果

3.3.1 命令列の確認、動作予想

メモリーイメージファイルが生成された。

生成された命令列を確認して、このような予想をたてた。

- 最初に PC=0x007c を実行したときの , REG[31] の値
 - PC=0x004004d8
- 最初に PC=0x0804 を実行した直後の , PC の値
 - PC=0x00400080

3.3.2 FPGA での実行結果

予想した点について結果は、以下ようになった。

- 最初に PC=0x0074 を実行した直後の REG[31] の値
 - REGWRITED=0x00400080,IDX=0x1f,WEN=1 となり、REG[31] に PC の値が退避された
- 最初に PC=0x0804 を実行した直後の , PC の値
 - PC=0x00400080 となり、ジャンプしなかった

画面上には、”HELLO !!”のみ表示された。

3.4 考察

予想した点について、PC=0x0074 の命令は正しく動作しているが、PC=0x0804 の命令については正しく実行できていないということが分かる。これは、プロセッサに jr 命令が実装されていないためであると考え

られる。

このプロセッサには、jal 命令は実装されているが、jr 命令が実装されていないので、関数に入ることではできても、関数から戻ることができない。そのため、プログラム中の my_print 関数の実行はできるが、戻ることができないため、次の手続きに進むことができないと考えられる。

4 実験 6-2

4.1 実験の目的、概要

実験 5-2 で作成したプロセッサに jr 命令が足りないことが、実験 6-1 で確認できた。本実験では、プロセッサに jr 命令を追加実装し、動作を確認する。その際、実験 6-1 での予想と実際を比較する。

これによって、関数からの復帰時のプロセッサの動作、レジスタから読み取れるデータなどを確認することを目的とする。

4.2 実験方法

4.3 追加設計

cpu.v に、以下の変更を加え、jpr_sel モジュールを追加した。

ソースコード 10 cpu.v の追加設計

```
1 jpr_sel の入出力ワイヤを定義
2 wire [31:0] jpr_sel_d0; // jpr 選択回路モジュール データ 1
3 wire [31:0] jpr_sel_d1; // jpr 選択回路モジュール データ 2
4 wire [31:0] jpr_sel_s; // jpr 選択回路モジュール セレクト信号
5 wire [31:0] jpr_sel_y; // jpr 選択回路モジュール 出力
6
7 32bit マルチプレクサモジュールを jpr_sel として宣言する
8 入出力はさっき定義したワイヤを使う
9 mux32_32_32 jpr_sel(jpr_sel_d0, jpr_sel_d1, jpr_sel_s, jpr_sel_y);
10
11 セレクトの出力をPCに接続する
12 assign pc_next = jpr_sel_y;
13 代わりに、割り当てられていたjp_sel_yをコメントアウトする
14 //assign pc_next = jp_sel_y;
15
16 セレクトの入力を割り当てる
17 assign jpr_sel_d0 = jp_sel_y;
18 assign jpr_sel_d1 = alu_ram_sel_y;
19 assign jpr_sel_s = jpr;
```

main_ctrl.v に以下の変更を加え、jr 命令が実行されときの制御信号を定義した。

ソースコード 11 main_ctrl.v の追加設計

```
1 jpr 信号を出力する条件として、jr のファンクションコードを追加する。
```

```

2 || Rfunc == 6'b001000)) ? 1'b1 : 1'b0;
3
4 レジスタの書き込み制御信号の出力条件を追加
5 if (Rfunc == 6'b001000) begin
6   reg_write_enable_tmp = 1'b0;
7 end else begin
8   reg_write_enable_tmp = 1'b1;
9 end

```

以下の操作で、論理合成し、FPGA にダウンロードした。

ソースコード 12 論理合成、ダウンロード

```

1 $ quartus_sh --flow compile MIPS_Default
2 $ quartus_pgm MIPS_Default.cdf

```

実験 6-1 で予想した点と、キーボードからの入力に対する反応について確認した。

4.4 実験結果

予想した点について結果は、以下のようになった。

- 最初に PC=0x0074 を実行した直後の REG[31] の値
 - REGWRITED=0x00400080,IDX=0x1f,WEN=1 となり、REG[31] に PC の値が退避された
- 最初に PC=0x0804 を実行した直後の、PC の値
 - PC=0x00400080 と、予想通りの結果になった

画面上には、"HELLO!!" という文字列が表示された。キーボードとの入出力の結果、以下のように、キーボードで入力した文字列がそのまま出力された。("STRING=" の後の文字列がキーボードから入力した文字列である。)

ソースコード 13 実行結果

```

1 HELLO!!
2 STRING=HELLO
3 ECHO HELLO
4 STRING=BYE
5 ECHO BYE

```

4.5 考察

予想した点の jr 命令は、最初の jal 命令で退避した PC の値を復元して、PC=0x00400080 にジャンプした。jr 命令を実装したことによって、関数から戻って手続きをすすめることができた。これにより、画面への出力と、キーボードからの入力を受け取る関数を正しく実行することができた。

この実験で、関数からの復帰時のプロセッサの動作を確認することができた。

5 実験 7

5.1 実験の目的、概要

本実験では、実験 6-2 で作成したプロセッサ上で、キーボード入力に応答を行うプログラムを動作させる。このプログラムは、プロセッサに実装されていない命令を使用するため、正常に動作しない。

本実験では、このプログラムを、正常に動作させることを目的とする。

5.2 実験方法 1

以下のようなプログラムを配置した。

ソースコード 14 sosuu.c

```
1 #define EXTIO_SCAN_ASCII (*(volatile unsigned int *)0x0310)
2 #define EXTIO_SCAN_REQ (*(volatile unsigned int *)0x030c)
3 #define EXTIO_SCAN_STROKE (*(volatile unsigned int *)0x0308)
4
5 #define SCAN_STRORING (unsigned int)0xffffffff
6
7 #define EXTIO_PRINT_STROKE (*(volatile unsigned int *)0x0300)
8 #define EXTIO_PRINT_ASCII (*(volatile unsigned int *)0x0304)
9
10 #define TRUE 0x1
11 #define FALSE 0x0
12
13 unsigned int sosuu_check(unsigned int kouho);
14 unsigned int my_a2i();
15 void my_i2a();
16 void my_print();
17 void my_scan();
18
19 main() {
20     unsigned int i;
21     unsigned int k;
22     unsigned int str1[16];
23     unsigned int str2[16];
24
25     /* "HELLO" を print */
26     str1[0] = 'H';
27     str1[1] = 'E';
28     str1[2] = 'L';
29     str1[3] = 'L';
30     str1[4] = 'O';
31     str1[5] = '\n';
32     str1[6] = '\0';
```

```

33  my_print(str1);
34
35  while (1) {
36      /* "NUM=" を print */
37      str1[0] = 'N';
38      str1[1] = 'U';
39      str1[2] = 'M';
40      str1[3] = '=';
41      str1[4] = '\0';
42      my_print(str1);
43
44      /* キーボードから入力された文字列(数字)を str2[] に記憶 */
45      my_scan(str2);
46
47      /* "ECHO " を print */
48      str1[0] = 'E';
49      str1[1] = 'C';
50      str1[2] = 'H';
51      str1[3] = '0';
52      str1[4] = ' ';
53      str1[5] = '\0';
54      my_print(str1);
55
56      /* str2[] を print */
57      my_print(str2);
58
59      /* '\n' を print */
60      str1[0] = '\n';
61      str1[1] = '\0';
62      my_print(str1);
63
64      /* 文字列(数字) str2[] を unsigned int に変換 */
65      k = my_a2i(str2);
66
67      for (i = 3; i <= k; i++) {
68          /* 素数判定 */
69          if (sosuu_check(i)) {
70              /* unsigned int i を文字列(数字)に変換して print */
71              my_i2a(i);
72          }
73      }
74
75      /* '\n' を print */
76      str1[0] = '\n';
77      str1[1] = '\0';
78      my_print(str1);

```

```

79     }
80 }
81
82 /* unsigned int kouho の素数判定を行う関数 */
83 /* 素数なら TRUE を返す */
84 /* 素数でないなら FALSE を返す */
85 unsigned int sosuu_check(unsigned int kouho) {
86     unsigned int t, tester, result;
87
88     if ((kouho % 2) == 0) {
89         /* kouho は偶数である == TRUE */
90         return FALSE;
91     } else {
92         result = TRUE;
93         for (tester = 3; tester < kouho / 2; tester += 2) {
94             /* kouho が本当に素数かどうかをチェック */
95             if ((kouho % tester) == 0) {
96                 /* kouho は tester の倍数である */
97                 result = FALSE;
98             }
99         }
100         return result;
101     }
102 }
103
104 /* 文字列(数字) srt[] を unsigned int に変換する関数 */
105 /* unsigned int result を返す */
106 unsigned int my_a2i(str) unsigned int *str;
107 {
108     // 省略
109 }
110
111 /* unsigned int i を文字列(数字)に変換して print する関数 */
112 void my_i2a(unsigned int i) {
113     // 省略
114 }

```

5.2.1 クロスコンパイル、メモリイメージファイルの作成

以下の操作でクロスコンパイルし、メモリイメージファイルを作成した。

ソースコード 15 クロスコンパイル、メモリイメージファイルの作成

```

1 $ cross_compile.sh sosuu.c
2 $ bin2v sosuu.bin

```

5.2.2 命令列の確認、動作予想

生成された、rom8x1024.mif を確認して、以下の点について結果を予測した。

- 命令メモリの 0x082 の命令はどのような命令か
- 命令メモリの 0x082 の命令は sosuu.check() のどの記述に対応するか

5.2.3 論理合成、ダウンロード

以下の操作で、論理合成し、FPGA にダウンロードした。

ソースコード 16 論理合成、ダウンロード

```
1 $ cp rom8x1024.mif ./mips_de10-lite/  
2 $ cd ./mips_de10-lite/  
3 $ quartus_sh --flow compile MIPS_Default  
4 $ quartus_pgm MIPS_Default.cdf
```

FPGA 上で、HELLO, NUM=と表示されたら、”20”を入力し、その結果を確認した。
このプログラムは正しく動作しない。この問題について、解決する方法を 2 つ考えた。

5.3 実験結果 1

5.3.1 命令列からの予想

メモリイメージファイルから、以下のような予想をたてた。

- 命令メモリの 0x082 の命令はどのような命令か
 - 2 つのレジスタの内容を符号なし整数と解釈して除算し、商は LO、余りは HI に格納する、divu 命令
 - rs に 3, rt に 2 を指定しているため、LO=REG[3]/REG[2]、LO=REG[3]%REG[2]
- 命令メモリの 0x082 の命令は sosuu.check() のどの記述に対応するか
 - ソースコード 14 の 88 行目 if ((kouho % tester) == 0) に対応する
 - RAM に 0(false) を代入しており、変数を lw して ++2 していることから分かる

5.3.2 プログラムの実行

以下のような表示となり、素数が表示されず、正しく実行されなかった。

ソースコード 17 sosuu.c 実行結果 1

```
1 HELLO  
2 NUM=20  
3 ECHO 20  
4 03 05 08  
5 NUM=
```

この問題について、以下のような解決方法を考えた。

- プロセッサが実行できるようにプログラムを変更
- プロセッサに命令を足りない命令を追加

5.4 実験方法 2

考案した解決方法のうち、今回は、「プロセッサが実行できるようにプログラムを変更」という方法で解決した。

5.4.1 C プログラムの変更

sosuu.c のプログラムを以下のように変更した。

ソースコード 18 sosuu.c

```

1 // 省略
2
3 main() {
4     // 省略
5 }
6
7 /* unsigned int kouho の素数判定を行う関数 */
8 /* 素数なら TRUE を返す */
9 /* 素数でないなら FALSE を返す */
10 unsigned int sosuu_check(unsigned int kouho) {
11     unsigned int t, tester;
12
13     for (t = 2; t <= kouho; t += 2) {
14         if (t == kouho) {
15             return FALSE;
16         }
17     }
18
19     for (t = 3; t < kouho; t += 2) {
20         for (tester = t; tester <= kouho; tester += t) {
21             if (tester == kouho) {
22                 return FALSE;
23             }
24         }
25     }
26
27     return TRUE;
28 }
29
30 // 省略

```

5.4.2 論理合成、ダウンロード

以下の操作で、論理合成し、FPGA にダウンロードした。

ソースコード 19 論理合成、ダウンロード

```
1 $ cp rom8x1024.mif ./mips_de10-lite/  
2 $ cd ./mips_de10-lite/  
3 $ quartus_sh --flow compile MIPS_Default  
4 $ quartus_pgm MIPS_Default.cdf
```

FPGA 上で実行し、プログラムの動作を確認した。

5.4.3 実験結果 2

画面上には、以下のような結果が表示され、正しく素数を判定できていた。

ソースコード 20 実行結果 2

```
1 HELLO  
2 NUM=20  
3 ECHO 20  
4 03 05 07 11 13 17 19  
5 NUM=
```

5.5 考察

このプロセッサには、divu 命令が実装されていない。そのため、正しくプログラムを実行することができなかったが、プログラムに変更することで、正しく動作させることに成功した。

必要な変更は、剰余を用いずに kouho を tester で割り切ることができるか判定することであった。ここでは、tester に tester を加えていき、kouho を超える前に、kouho==tester となるかどうかを判定した。これによって、kouho が tester で割り切ることができるか判定できると考えた。実際に、判定プログラムは正しく動作した。

6 実験 8

6.1 実験の目的、概要

本実験では、プロセッサからステッピングモータを扱う。最終的に、キーボードからモータを制御するプログラムを作成、実行する。

これによって、このプロセッサで提供されている機能を使用することで実現できるプログラムを作成することを目的とする。

6.2 実験方法

以下のようなプログラムを配置した。

ソースコード 21 motor.c

```
1 #define EXTIO_SCAN_ASCII (*(volatile unsigned int *)0x0310)
2 #define EXTIO_SCAN_REQ (*(volatile unsigned int *)0x030c)
3 #define EXTIO_SCAN_STROKE (*(volatile unsigned int *)0x0308)
4
5 #define SCAN_STRORING (unsigned int)0xffffffff
6
7 #define EXTIO_PRINT_STROKE (*(volatile unsigned int *)0x0300)
8 #define EXTIO_PRINT_ASCII (*(volatile unsigned int *)0x0304)
9
10 #define TRUE 0x1
11 #define FALSE 0x0
12
13 #define GPIO0 (*(volatile unsigned int *)0x0320)
14
15 void my_motor();
16 void ext_out();
17
18 main() {
19     while (1) {
20         my_motor();
21     }
22 }
23
24 void my_motor() {
25     ext_out(8);
26     ext_out(4);
27     ext_out(2);
28     ext_out(1);
29 }
30
31 void ext_out(unsigned int num) { GPIO0 = num; }
```

6.2.1 クロスコンパイル、メモリイメージファイルの作成

以下の操作でクロスコンパイルし、メモリイメージファイルを作成した。

ソースコード 22 クロスコンパイル、メモリイメージファイルの作成

```
1 $ cross_compile.sh motor.c
2 $ bin2v motor.bin
```

6.2.2 モーターの接続

このプログラムでは、GPIO ピンを経由して、ステッピングモータを使用するため、プログラムのダウンロード前に、制御回路と GPIO ピンを、モータと制御回路を接続した。

6.2.3 論理合成、ダウンロード

以下の操作で、論理合成し、FPGA にダウンロードした。

ソースコード 23 論理合成、ダウンロード

```
1 $ cp rom8x1024.mif ./mips_de10-lite/  
2 $ cd ./mips_de10-lite/  
3 $ quartus_sh --flow compile MIPS_Default  
4 $ quartus_pgm MIPS_Default.cdf
```

FPGA 上でプログラムを実行し、結果を確認した。

6.2.4 ステッピングモータ制御プログラムの作成

コンパイラと、プロセッサの対応する機能に注意して、motor.c を変更した。

ソースコード 24 motor.c'

```
1 // 省略  
2 main() {  
3     unsigned int output[32];  
4     unsigned int input[32];  
5  
6     unsigned int steps, direction;  
7  
8     while (1) {  
9         output[0] = 'S';  
10        output[1] = 'T';  
11        output[2] = 'E';  
12        output[3] = 'P';  
13        output[4] = '=';  
14        output[5] = '\\0';  
15        my_print(output);  
16  
17        my_scan(input);  
18        steps = my_a2i(input);  
19  
20        output[0] = 'D';  
21        output[1] = 'I';  
22        output[2] = 'R';  
23        output[3] = '?';  
24        output[4] = '=';  
25        output[5] = '\\0';  
26        my_print(output);  
27  
28        my_scan(input);  
29        direction = my_a2i(input);  
30
```

```

31     if (direction == 0) {
32         my_motor(0x1248, steps);
33     } else {
34         my_motor(0x8421, steps);
35     }
36
37     output[0] = 'D';
38     output[1] = '0';
39     output[2] = 'N';
40     output[3] = 'E';
41     output[4] = '!';
42     output[5] = '\n';
43     output[6] = '\0';
44     my_print(output);
45 }
46 }
47
48 void my_motor(unsigned int table, unsigned int steps) {
49     int cnt;
50     int shifter = 0;
51     for (cnt = 0; cnt < steps; cnt++) {
52         shifter = (shifter == 12 ? 4 : shifter + 4);
53         ext_out((table >> shifter) & 0b1111);
54     }
55 }
56
57 void ext_out(unsigned int num) { GPIO0 = num; }
58
59 unsigned int my_a2i(str) unsigned int *str;
60 {
61     // 省略
62 }
63
64 void my_scan(str) unsigned int *str;
65 {
66     // 省略
67 }
68
69 void my_print(str) unsigned int *str;
70 {
71     // 省略
72 }

```

6.2.5 クロスコンパイル、メモリイメージファイルの作成

以下の操作でクロスコンパイルし、メモリイメージファイルを作成した。

ソースコード 25 クロスコンパイル、メモリイメージファイルの作成

```
1 $ cross_compile.sh motor.c
2 $ bin2v motor.bin
```

6.2.6 論理合成、ダウンロード

以下の操作で、論理合成し、FPGA にダウンロードした。

ソースコード 26 論理合成、ダウンロード

```
1 $ cp rom8x1024.mif ./mips_de10-lite/
2 $ cd ./mips_de10-lite/
3 $ quartus_sh --flow compile MIPS_Default
4 $ quartus_pgm MIPS_Default.cdf
```

ダウンロード後、FPGA で実行して動作を確認した。

6.3 実験結果

最初の motor.c を実行した結果、モータは 1 方向に回転し続けた。

motor.c を変更し、実行した結果、以下のような結果になった。(//でコメントアウトした部分で、モータの様子を記述した)

ソースコード 27 変更した motor.c の実行結果

```
1 STEP=200
2 DIR?=1
3 //最初の実行と同じ方向に、200ステップ分回転した
4 DONE!
5 STEP=100
6 DIR?=0
7 //最初の実行と逆の方向に、200ステップ分回転した
8 DONE!
```

6.4 考察

変更前の motor.c から、GPIO ピンに、4bit のフラグを入力し、コイルの相を指定して励磁させていると考えられる。プログラムでは、「1000 → 0100 → 0010 → 0001」の順で繰り返し励磁されていた。

変更後のプログラムでは、順方向の他に、逆方向への回転も実現するために、テーブルを定義した。1248 と、8421 の 2 つのテーブルを用意し、4bit シフトして下位 4bit をとることで励磁するビット数を決定した。また、回転するステップ数を指定するため、ループ数を入力から指定することにした。

この実験でのプログラムの作成を通して、プロセッサに搭載されている機能の中でプログラムを作成することができた。

7 実験 9

7.1 実験の目的、概要

実験 8 で使用したプロセッサには、乗算命令 `mult` が実装されておらず、乗算を使用したプログラムは動作しない。この実験では、乗算結果を決められた `hi,lo` レジスタに保存する `mult` 命令と、`lo` レジスタから乗算結果の下位半分の `bit` を取得する `mflo` 命令を実装する。

これによって、`hi,lo` レジスタを使用した演算の動作を確認することを目的とする。

7.2 実験方法

以下のプログラムを配置した。

ソースコード 28 `mult.c`

```
1 #define EXTIO_SCAN_ASCII (*(volatile unsigned int *)0x0310)
2 #define EXTIO_SCAN_REQ (*(volatile unsigned int *)0x030c)
3 #define EXTIO_SCAN_STROKE (*(volatile unsigned int *)0x0308)
4
5 #define SCAN_STRORING (unsigned int)0xffffffff
6
7 #define EXTIO_PRINT_STROKE (*(volatile unsigned int *)0x0300)
8 #define EXTIO_PRINT_ASCII (*(volatile unsigned int *)0x0304)
9
10 #define TRUE 0x1
11 #define FALSE 0x0
12
13 unsigned int sosuu_check(unsigned int kouho);
14 unsigned int my_a2i();
15 void my_i2a();
16 void my_print();
17 void my_scan();
18
19 main() {
20     unsigned int k;
21     unsigned int str1[16];
22     unsigned int str2[16];
23
24     /* "HELLO" を print */
25     str1[0] = 'H';
26     str1[1] = 'E';
27     str1[2] = 'L';
28     str1[3] = 'L';
29     str1[4] = 'O';
30     str1[5] = '\n';
31     str1[6] = '\0';
```

```

32  my_print(str1);
33
34  while (1) {
35      /* "NUM=" を print */
36      str1[0] = 'N';
37      str1[1] = 'U';
38      str1[2] = 'M';
39      str1[3] = '=';
40      str1[4] = '\0';
41      my_print(str1);
42
43      /* キーボードから入力された文字列(数字)を str2[] に記憶 */
44      my_scan(str2);
45
46      /* "ECHO " を print */
47      str1[0] = 'E';
48      str1[1] = 'C';
49      str1[2] = 'H';
50      str1[3] = '0';
51      str1[4] = ' ';
52      str1[5] = '\0';
53      my_print(str1);
54
55      /* str2[] を print */
56      my_print(str2);
57
58      /* '\n' を print */
59      str1[0] = '\n';
60      str1[1] = '\0';
61      my_print(str1);
62
63      /* 文字列(数字) str2[] を unsigned int に変換 */
64      k = my_a2i(str2);
65
66      /* k × k を計算 */
67      k = k * k;
68
69      /* unsigned int k を文字列(数字)に変換して print */
70      my_i2a(k);
71
72      /* '\n' を print */
73      str1[0] = '\n';
74      str1[1] = '\0';
75      my_print(str1);
76  }
77 }

```

```

78
79 /* 文字列(数字) srt[] を unsigned int に変換する関数 */
80 /* unsigned int result を返す */
81 unsigned int my_a2i(str) unsigned int *str;
82 {
83     // 省略
84 }
85
86 /* unsigned int i を文字列(数字)に変換して print する関数 */
87 void my_i2a(unsigned int i) {
88     // 省略
89 }
90
91 /* キーボードから入力された文字列を str[] に記憶する関数 */
92 void my_scan(str) unsigned int *str;
93 {
94     // 省略
95 }
96
97 /* 文字列 str[] を表示する関数 */
98 void my_print(str) unsigned int *str;
99 {
100     // 省略
101 }

```

7.2.1 bin2v の拡張

bin2v.tar.gz を配置して、tar xvfz ./bin2v.tar.gz で、bin2v のソースコードと Makefile を取得した。
bin2v に以下のような変更を行った。

ソースコード 29 bin2v の拡張

```

1 // コメント追加、funct コード追加、output のためのコメント追加
2
3 // コメント追加
4 'define R 6 b000000 R 形式 (add, addu, sub, subu, and, or, slt, jalr, jr, mult, mflo)
5
6 // コメント追加
7     MULT(op = 000000, func = 011000)
8     MULT {Hi, Lo} <= REG[rs] * REG[rt]; MULT rs,rt
9
10 // コメント追加
11     MFLO(op = 000000, func = 010010)
12         MFLO REG[rd] <= Lo; MFLO rd
13
14 // funct コードの定義追加
15     #define MULT 24

```



```

16  #define MFLO 18
17
18  // funct が mult のものだった場合の、rom8x1024_sim.v 生成用の記述
19  case MULT:
20      fprintf(outfp, "10'h%03x: data = 32'h%08x; // %08x: MULT, ", rom_addr, wd,
                cmt_addr);
21      fprintf(outfp, "{Hi, Lo}<=REG[%d]*REG[%d];\n", rs, rt);
22      break;
23
24  // funct が mflo のものだった場合の、rom8x1024_sim.v 生成用の記述
25  case MFLO:
26      fprintf(outfp, "10'h%03x: data = 32'h%08x; // %08x: MFLO, ", rom_addr, wd,
                cmt_addr);
27      fprintf(outfp, "REG[%d]<=Lo;\n", rd);
28      break;
29
30  // funct が mult のものだった場合の、rom8x1024_sim.v 生成用の記述
31  case MULT:
32      fprintf(outfp, "%03x : %08x ; %% %08x: MULT, ", rom_addr, wd, cmt_addr);
33      fprintf(outfp, "{Hi, Lo}<=REG[%d]*REG[%d]; %%\n", rs, rt);
34      break;
35
36  // funct が mflo のものだった場合の、rom8x1024_sim.v 生成用の記述
37  case MFLO:
38      fprintf(outfp, "%03x : %08x ; %% %08x: MFLO, ", rom_addr, wd, cmt_addr);
39      fprintf(outfp, "REG[%d]<=Lo; %%\n", rd);
40      break;

```

変更後、make コマンドで make した。

7.2.2 クロスコンパイル、メモリイメージファイルの作成

以下の操作でクロスコンパイルし、メモリイメージファイルを作成した。

ソースコード 30 クロスコンパイル、メモリイメージファイルの作成

```

1  $ cross_compile.sh mult.c
2  $ ./bin2v/bin2v mult.bin

```

7.2.3 cpu.v の追加設計

ソースコード 31 cpu.v の変更

```

1  alu インスタンスの入出力を同期回路として定義し直した。
2  alu alua(alu_a, alu_b, alu_ctrl, alu_y, alu_comp);
3
4  alu alua(clock, reset, alu_a, alu_b, alu_ctrl, alu_y, alu_comp);

```

7.2.4 main_ctrl.v の追加設計

ソースコード 32 main_ctrl.v の変更

```
1 // mult, mflo 命令に関するコメントの追加
2 R 形式
3 MULT(op = 000000, func = 011000)
4 MULT {Hi, Lo} <= REG[rs] * REG[rt]; MULT rs,rt
5 MFLO(op = 000000, func = 010010)
6 MFLO REG[rd] <= Lo; MFLO rd
```

7.2.5 alu.v の追加設計

ソースコード 33 alu.v の変更

```
1 // コメントの追加
2 // mult(multiply)
3 // mflo(move from Lo)
4
5 // alu 制御コードのコメントの追加
6 // 1011, mult
7 // 1100, mflo
8
9 // ALU 制御コードの定義
10 `define ALU_MULT 4 `b1011
11 `define ALU_MFLO 4 `b1100
12
13 // ALU モジュールの入力ポートの拡張
14 // clock,reset 信号の追加
15 module alu (alu_a, alu_b, alu_ctrl, alu_y, alu_comp);
16 // を
17 module alu (clock, reset, alu_a, alu_b, alu_ctrl, alu_y, alu_comp);
18 // に変更
19
20 // mult 命令実行時に alu_a * alu_b の結果を {hi, lo}に格納する記述の追加
21 input clock, reset; // 入力 クロック, リセット
22 reg [31:0] hi; //上位
23 reg [31:0] lo; //下位
24 always @(posedge clock or negedge reset) begin
25     if (reset == 1 `b0) begin
26         hi <= 32 `h00000000;
27         lo <= 32 `h00000000;
28     end else begin
29         {hi, lo} <= (alu_ctrl == `ALU_MULT) ? alu_a * alu_b : {hi, lo};
30     end
31 end
```

```

32
33 // mflo 命令実行時に {hi, lo} の lo を result に出力する記述の追加
34 `ALU_MFLO: begin
35     result <= lo;
36 end

```

7.2.6 alu_ctrl.v の追加設計

ソースコード 34 alu_ctrl.v

```

1 // mult,mflo 命令用の ALU 制御コードの定義
2 `define ALU_CTRL_MULT 4'b1011
3 `define ALU_CTRL_MFLO 4'b1100
4
5 // mult,mflo 命令用の ALU 制御コードについてのコメント追加
6 // mult(multiply), 010, 011000, 1011
7 // mflo(move from lo), 010, 010010, 1100
8
9 // 実行する命令がmult,mflo 命令のとき、mult,mflo 命令用の ALU 制御コードを生成する処理の追加
10 end else if (func == 6'b011000) begin // func=MULT
11     y = `ALU_CTRL_MULT;
12 end else if (func == 6'b010010) begin // func=MFLO
13     y = `ALU_CTRL_MFLO;

```

7.2.7 論理合成、ダウンロード

以下の操作で、論理合成し、FPGA にダウンロードした。

ソースコード 35 論理合成、ダウンロード

```

1 $ cp rom8x1024.mif ./mips_de10-lite/
2 $ cd ./mips_de10-lite/
3 $ quartus_sh --flow compile MIPS_Default
4 $ quartus_pgm MIPS_Default.cdf

```

FPGA 上でプログラムを実行し、結果を確認した。

7.3 実験結果

7.3.1 メモリイメージファイルの生成結果

生成されたメモリイメージファイルのうち、追加実装した部分に関わるものは以下ようになった。

ソースコード 36 メモリイメージファイルの追加実装に関わる部分

```

1 050 : 00620018 ; % 00400140: MULT, {Hi, Lo}<=REG[3]*REG[2]; %
2 051 : 00001012 ; % 00400144: MFLO, REG[2]<=Lo; %

```

7.3.2 mult.c の実行結果

mult.c を実行した結果、以下のようになった。

ソースコード 37 mult.c の実行結果

```
1  HELLO
2  NUM=4
3  echo 4
4  16
5  NUM=6
6  echo 6
7  36
```

7.4 考察

7.4.1 bin2v の追加実装

bin2v の追加実装ではコメント追加のほか、命令コードの機能コードが mult,mflo 命令のものだった場合のイメージファイルの生成コードを追加した。結果に示したように、適切なコメントを挿入していた。

7.4.2 プロセッサの追加実装

cpu.v の追加実装では、クロックとリセット信号を ALU に追加した。これは、演算結果を保持する Hi,Lo レジスタの更新、消去のためであると考えられる。

alu.v の追加実装では、コントローラから入力される制御コードとして mult,mflo 命令のものを追加した。alu 内部では、クロック信号の立ち上がりで、制御コードが mult だった場合、乗算の結果を hi と lo の接続に入力し、その他のときは両方のレジスタを同じ値でリライトしている。また、リセット信号の立ち下がりでは、Hi,Lo 両方を 0 でリセットしている。

mflo 命令の制御コードを受けた場合、結果には Lo レジスタの値を入力している。

main_ctrl.v、alu_ctrl.v では、命令に対応した制御コードを出力している。

7.4.3 mult.c の実行結果

mult.c を実行した結果から、プログラムは乗算命令を正しく実行できていることが分かる。

8 実験 10

8.1 実験の目的、概要

実験 9 までで作成したプロセッサには、除算を行う命令と、その結果を利用する命令が実装されておらず、除算を利用したプログラムは正しく動作しない。この実験では、除算命令の divu 命令と、結果を取得するための mfhi 命令を実装する。また、この命令を使用して、実験 7 で動作させた素数判定のプログラムを改良させて動作させる。

これによって、プロセッサに足りない機能に対して正しい変更を行い、機能を実現できることを確認する。

8.2 実験方法

以下のプログラムを配置した。

ソースコード 38 mult.c

```
1 #define EXTIO_SCAN_ASCII (*(volatile unsigned int *)0x0310)
2 #define EXTIO_SCAN_REQ (*(volatile unsigned int *)0x030c)
3 #define EXTIO_SCAN_STROKE (*(volatile unsigned int *)0x0308)
4
5 #define SCAN_STRORING (unsigned int)0xffffffff
6
7 #define EXTIO_PRINT_STROKE (*(volatile unsigned int *) 0x0300)
8 #define EXTIO_PRINT_ASCII (*(volatile unsigned int *) 0x0304)
9
10 #define TRUE 0x1
11 #define FALSE 0x0
12
13 unsigned int sosuu_check(unsigned int kouho);
14 unsigned int my_a2i();
15 void my_i2a();
16 void my_print();
17 void my_scan();
18
19 main() {
20     unsigned int i;
21     unsigned int k;
22     unsigned int str1[16];
23     unsigned int str2[16];
24
25     /* "HELLO" を print */
26     str1[0] = 'H'; str1[1] = 'E';
27     str1[2] = 'L'; str1[3] = 'L';
28     str1[4] = 'O'; str1[5] = '\n';
29     str1[6] = '\0';
30     my_print(str1);
31
32     while (1) {
33         /* "NUM=" を print */
34         str1[0] = 'N'; str1[1] = 'U';
35         str1[2] = 'M'; str1[3] = '=';
36         str1[4] = '\0';
37         my_print(str1);
38
39         /* キーボードから入力された文字列(数字)を str2[] に記憶 */
40         my_scan(str2);
```

```

41
42  /* "ECHO " を print */
43  str1[0] = 'E'; str1[1] = 'C';
44  str1[2] = 'H'; str1[3] = 'O';
45  str1[4] = ' '; str1[5] = '\0';
46  my_print(str1);
47
48  /* str2[] を print */
49  my_print(str2);
50
51  /* '\n' を print */
52  str1[0] = '\n'; str1[1] = '\0';
53  my_print(str1);
54
55  /* 文字列(数字) str2[] を unsigned int に変換 */
56  k = my_a2i(str2);
57
58  for (i = 3; i <= k; i++) {
59      /* 素数判定 */
60      if (sosuu_check(i)) {
61          /* unsigned int i を文字列(数字)に変換して print */
62          my_i2a(i);
63      }
64  }
65
66  /* '\n' を print */
67  str1[0] = '\n'; str1[1] = '\0';
68  my_print(str1);
69  }
70 }
71
72 /* unsigned int kouho の素数判定を行う関数 */
73 /* 素数なら TRUE を返す */
74 /* 素数でないなら FALSE を返す */
75 unsigned int sosuu_check(unsigned int kouho) {
76     unsigned int t, tester, result;
77     // my_print('K');
78     // my_i2a(kouho);
79     if ((kouho % 2) == 0) {
80         /* kouho は偶数である == TRUE */
81         return FALSE;
82     } else {
83         result = TRUE;
84         my_i2a(kouho / 2);
85         for (tester = 3; tester < kouho / 2; tester += 2) {
86             // my_print('T');

```

```

87     // my_i2a(tester);
88     /* kouho が本当に素数かどうかをチェック */
89     my_i2a(kouho % tester);
90     if ((kouho % tester) == 0) {
91         // my_print('F');
92         /* kouho は tester の倍数である */
93         result = FALSE;
94     }
95 }
96 return result;
97 }
98 }
99
100 /* 文字列(数字) srt[] を unsigned int に変換する関数 */
101 /* unsigned int result を返す */
102 unsigned int my_a2i(str)
103     unsigned int *str;
104 {
105     unsigned int *str_tmp;
106     unsigned int k;
107     unsigned int result;
108
109     str_tmp = str;
110     for (k = 0; *str_tmp != '\0'; k++) {
111         str_tmp++;
112     }
113
114     result = 0;
115     str_tmp = str;
116
117     if (k == 1) {
118         result = *str_tmp - '0';
119     } else if (k == 2) {
120         for (k = 0; k < (*str_tmp - '0'); k++) {
121             result = result + 10;
122         }
123         str_tmp++;
124         result = result + (*str_tmp - '0');
125     } else if (k == 3) {
126         for (k = 0; k < (*str_tmp - '0'); k++) {
127             result = result + 100;
128         }
129         str_tmp++;
130         for (k = 0; k < (*str_tmp - '0'); k++) {
131             result = result + 10;
132         }

```

```

133     str_tmp++;
134     result = result + (*str_tmp - '0');
135 }
136
137 return result;
138 }
139
140 /* unsigned int i を文字列(数字)に変換して print する関数 */
141 void my_i2a(unsigned int i) {
142     unsigned int counter;
143     unsigned int s[4];
144
145     for (counter = 0; i >= 10; counter++) {
146         i -= 10;
147     }
148     s[0] = counter + '0';
149     s[1] = i + '0';
150     s[2] = ' ';
151     s[3] = '\0';
152
153     my_print(s);
154 }
155
156 /* キーボードから入力された文字列を str[] に記憶する関数 */
157 void my_scan(str)
158     unsigned int *str;
159 {
160     EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
161     EXTIO_SCAN_REQ = (unsigned int)0x00000001;
162     EXTIO_SCAN_STROKE = (unsigned int)0x00000001;
163
164     EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
165     EXTIO_SCAN_STROKE = (unsigned int)0x00000001;
166     while (EXTIO_SCAN_ASCII == SCAN_STRORING) {
167         EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
168         EXTIO_SCAN_STROKE = (unsigned int)0x00000001;
169     }
170
171     while ((*str = EXTIO_SCAN_ASCII) != (unsigned int)0x3e) { // 0x3e=RETURN
172         if ((*str >= 1) && (*str <= 26)) {
173             *str = 'A' + *str - 1;
174         } else if ((*str >= 48) && (*str <= 57)) {
175             *str = '0' + *str - 48;
176         } else {
177             if (*str == 0) {
178                 *str = '@';

```



```

179     } else if (*str == 27) {
180         *str = '[';
181     } else if (*str == 29) {
182         *str = ']';
183     } else if ((*str >= 32) && (*str <= 47)) {
184         *str = ' ' + *str - 32;
185     } else if (*str == 58) {
186         *str = '?';
187     } else if (*str == 59) {
188         *str = '=';
189     } else if (*str == 60) {
190         *str = ',';
191     } else if (*str == 61) {
192         *str = ':';
193     } else if (*str == 62) {
194         *str = '\n';
195     } else {
196         *str = '@';
197     }
198 }
199 EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
200 EXTIO_SCAN_STROKE = (unsigned int)0x00000001;
201 str++;
202 }
203 *str = '\0';
204
205 EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
206 EXTIO_SCAN_REQ = (unsigned int)0x00000000;
207 EXTIO_SCAN_STROKE = (unsigned int)0x00000001;
208
209 EXTIO_SCAN_STROKE = (unsigned int)0x00000000;
210 }
211
212 /* 文字列 str[] を表示する関数 */
213 void my_print(str)
214     unsigned int *str;
215 {
216     while (*str != '\0') {
217         EXTIO_PRINT_STROKE = (unsigned int)0x00000000;
218
219         if ((*str >= 'A') && (*str <= 'Z')) {
220             EXTIO_PRINT_ASCII = *str - 'A' + 1;
221         } else if ((*str >= 'a') && (*str <= 'z')) {
222             EXTIO_PRINT_ASCII = *str - 'a' + 1;
223         } else if ((*str >= '0') && (*str <= '9')) {
224             EXTIO_PRINT_ASCII = *str - '0' + 48;

```

```

225     } else {
226         if (*str == '@') {
227             EXTIO_PRINT_ASCII = (unsigned int)0;
228         } else if (*str == '[') {
229             EXTIO_PRINT_ASCII = (unsigned int)27;
230         } else if (*str == ']') {
231             EXTIO_PRINT_ASCII = (unsigned int)29;
232         } else if ((*str >= ' ' && (*str <= '/')) {
233             EXTIO_PRINT_ASCII = *str - ' ' + 32;
234         } else if (*str == '?') {
235             EXTIO_PRINT_ASCII = (unsigned int)58;
236         } else if (*str == '=') {
237             EXTIO_PRINT_ASCII = (unsigned int)59;
238         } else if (*str == ';') {
239             EXTIO_PRINT_ASCII = (unsigned int)60;
240         } else if (*str == ':') {
241             EXTIO_PRINT_ASCII = (unsigned int)61;
242         } else if (*str == '\\n') {
243             EXTIO_PRINT_ASCII = (unsigned int)62;
244         } else {
245             EXTIO_PRINT_ASCII = (unsigned int)0x00000000;
246         }
247     }
248
249     EXTIO_PRINT_STROKE = (unsigned int)0x00000001;
250     str++;
251 }
252 }

```

8.2.1 bin2v の拡張

bin2v に以下のような変更を行った。

ソースコード 39 bin2v の拡張

```

1 // コメント追加、funct コード追加、output のためのコメント追加
2
3 // コメント追加
4 #define R 6'b000000 R 形式 (add, addu, sub, subu, and, or, slt, jalr, jr, mult, mflo,
   divu, mfhi)
5
6 // コメント追加
7 DIVU(op = 000000, func = 00011011)
8 DIVU Lo <= REG[rs] / REG[rt], Hi <= REG[rs] % REG[rt]; DIVU rs, rt
9
10 // コメント追加
11 MFHI(op = 000000, func = 00010000)

```

```

12  MFHI REG[rd] <= Hi; MFHI rd
13
14  // funct コードの定義追加
15  #define DIVU 27
16  #define MFHI 16
17
18  // funct が mult のものだった場合の、rom8x1024_sim.v 生成用の記述
19  case DIVU:
20      fprintf(outfp, "10'h%03x: data = 32'h%08x; // %08x: DIVU, ", rom_addr, wd,
                cmt_addr);
21      fprintf(outfp, "Lo<=REG[%d]/REG[%d], Hi<=REG[%d]%REG[%d];\n", rs, rt, rs, rt);
22      break;
23
24  // funct が mflo のものだった場合の、rom8x1024_sim.v 生成用の記述
25  case MFHI:
26      fprintf(outfp, "10'h%03x: data = 32'h%08x; // %08x: MFHI, ", rom_addr, wd,
                cmt_addr);
27      fprintf(outfp, "REG[%d]<=Hi;\n", rd);
28      break;
29
30  // funct が mult のものだった場合の、rom8x1024_sim.v 生成用の記述
31  case DIVU:
32      fprintf(outfp, "%03x : %08x ; %% %08x: DIVU, ", rom_addr, wd, cmt_addr);
33      fprintf(outfp, "Lo<=REG[%d]*REG[%d], Hi<=REG[%d]\%REG[%d]; %%\n", rs, rt, rs, rt
                );
34      break;
35
36  // funct が mflo のものだった場合の、rom8x1024_sim.v 生成用の記述
37  case MFHI:
38      fprintf(outfp, "%03x : %08x ; %% %08x: DIVU, ", rom_addr, wd, cmt_addr);
39      fprintf(outfp, "REG[%d]<=Hi; %%\n", rd);
40      break;

```

変更後、make コマンドで make した。

8.2.2 alu.v の追加設計

ソースコード 40 alu.v の変更

```

1  // コメントの追加
2  // divu(divide unsigned)
3  // mfhi(move from Hi)
4
5  // alu 制御コードのコメントの追加
6  // 0101, divu
7  // 1101, mfhi
8

```

```

9 // ALU 制御コードの定義
10 `define ALU_DIVU 4'b0101
11 `define ALU_MFHI 4'b1101
12
13 // divu 命令実行時に 商と剰余 の結果を {hi, lo}に格納する記述の追加
14 // mult のものもまとめて if 文で制御
15 if (alu_ctrl == 'ALU_MULT) begin
16     {hi, lo} <= alu_a * alu_b;
17 end else if (alu_ctrl == 'ALU_DIVU) begin
18     hi <= alu_a % alu_b;
19     lo <= alu_a / alu_b;
20 end
21
22 // mfhi 命令実行時に {hi, lo} の hi を result に出力する記述の追加
23 `ALU_MFHI: begin
24     result <= hi;
25 end

```

8.2.3 alu_ctrl.v の追加設計

ソースコード 41 alu_ctrl.v

```

1 // divu,mfhi 命令用の ALU 制御コードの定義
2 `define ALU_CTRL_DIVU 4'b0101
3 `define ALU_CTRL_MFHI 4'b1101
4
5 // divu,mfhi 命令用の ALU 制御コードについてのコメント追加
6 // divu(divide unsigned), 010, 011011, 0101
7 // mfhi(move from hi), 010, 01000, 1101
8
9 // 実行する命令がdivu,mfhi 命令のとき、divu,mfhi 命令用の ALU 制御コードを生成する処理の追加
10 end else if (func == 6'b011011) begin // func=DIVU
11     y = 'ALU_CTRL_DIVU;
12 end else if (func == 6'b010000) begin // func=MFHI
13     y = 'ALU_CTRL_MFHI;

```

8.2.4 main_ctrl.v の追加設計

ソースコード 42 main_ctrl.v の変更

```

1 // mult, mflo 命令に関するコメントの追加
2 DIVU(op = 000000, func = 011011)
3 DIVU Lo <= REG[rs] / REG[rt], Hi <= REG[rs] % REG[rt]; DIVU rs, rt
4 MFHI(op = 000000, func = 010000)
5 MFHI REG[rd] <= Hi; MFHI rd

```

8.2.5 アセンブリの出力

以下の操作で `sosuu.c` から `sosuu.s` を出力した。

ソースコード 43 アセンブリの出力

```
1 $ mips-rtems-gcc -O0 -S sosuu.c
```

8.2.6 `sosuu.s` の変更

ソースコード 44 `sosuu.s` の変更

```
1 $L14:
2     lw $3,32($fp)
3     lw $2,4($fp)
4     nop
5     bne $2,$0,1f
6     divu $0,$3,$2
7     break 7
8 1:
9     mfhi $2
10    bne $2,$0,$L15
11    nop
12
13    を
14
15 $L14:
16    lw $3,32($fp)
17    lw $2,4($fp)
18    nop
19    divu $0,$3,$2
20    bne $2,$0,1f
21    nop
22    break 7
23 1:
24    mfhi $2
25    bne $2,$0,$L15
26    nop
27
28    に変更
```

8.2.7 バイナリ出力、`bin2v`

以下の操作でバイナリファイルを生成し、

ソースコード 45 アセンブリの出力、クロスコンパイル

```
1 $ mips-rtems-as -mips1 -g -o sosuu.o sosuu.s
```

```
2 $ mips-rtems-ld -N sosuu.o -o sosuu.bin -e main
3 $ mips-rtems-objcopy -O binary sosuu.bin
4 $ ./bin2v/bin2v mult.bin
```

8.2.8 論理合成、ダウンロード

以下の操作で、論理合成し、FPGA にダウンロードした。

ソースコード 46 論理合成、ダウンロード

```
1 $ cp rom8x1024.mif ./mips_de10-lite/
2 $ cd ./mips_de10-lite/
3 $ quartus_sh --flow compile MIPS_Default
4 $ quartus_pgm MIPS_Default.cdf
```

FPGA 上でプログラムを実行し、結果を確認した。

8.3 実験結果

8.3.1 メモリイメージファイルの生成結果

生成されたメモリイメージファイルのうち、追加実装した部分に関わるものは以下ようになった。

ソースコード 47 メモリイメージファイルの追加実装に関わる部分

```
1 089 : 0062001b ; % 00400224: DIVU, Lo<=REG[3]*REG[2], Hi<=REG[3]%REG[2]; %
2 ...
3 08b : 00001010 ; % 0040022c: MFHI, REG[2]<=Hi; %
```

8.3.2 divt.c の実行結果

プログラムを FPGA で実行した結果、以下ようになった。

ソースコード 48 sosuu.c の実行結果

```
1 HELLO
2 NUM=20
3 ECHO 20
4 03 05 07 11 13 17 19
5 NUM=
```

8.4 考察

8.4.1 bin2v の追加実装

この実験では、まず実験 9 と同様、bin2v で適切なコメント付きでメモリイメージファイルを作成するための実装をした。その結果、該当部分にコメントが挿入されたことがソースコード 47 から分かる。

8.4.2 プロセッサの追加実装

また、プロセッサは `mult` 命令の実装時と同様、ALU 内部で `divu` の演算結果をクロックの立ち上がりで `Hi,Lo` 別々に入力している。また、命令コードに対する制御コードは `alu_ctrler.v` で定義し、出力した。

8.4.3 アセンブリの修正

実験 9 までで使用したクロスコンパイラは、パイプラインでの処理での遅延実行を行うように命令を並び替える。したがって、今回のプロセッサで正しく実行させる事ができない。

そこで、実際の手続き通り、`bne` 命令を `divu` 命令の後に差し替え、`nop` 命令を `bne` 命令の後ろに配置した。そのために、`cat /pub1/jikken/eda/linux/bin/cross_compile.sh` でクロスコンパイラのアセンブリ出力前後のコマンドを主導で実行した。

8.4.4 `sosuu.c` の実行結果

`sosuu.c` では、本来のアルゴリズムにしたがって、剰余を使った倍数判定を行なった。実行結果は正しいものになっているのが確認され、実行時には実験 7 よりも早く結果が表示された。これは、倍数判定にかかるステップ数がなくなったためであると考えられる。

この実験では、これまでのプロセッサに対する変更をもとに、未実装な命令を正しく実行できるように変更した。これによって、この実験の目的を達成できたと考えた。

考察

この実験で、FPGA 上のプロセッサは関数の呼び出し、関数内部からの `return`、乗算、除算を行えるようになった。したがって、キーボードからの入力、乗算を使った比較的高速な倍数判定による素数を出力するプログラムを実行することができた。

実装を通して、関数実行時のプロセッサの動作、`alu` 内部で `Hi,Lo` レジスタを使った演算について理解することができた。