目次

1　実験の目的

コンピュータで扱う数値の表現方法、CPU の動作、各マシン命令の機能、アセンブリ言語とマシン語の関係。およびアドレッシングモードなどを理解する。

2　実装した関数の説明

実習２で作成したプログラムにおいて関数を複数作成した。ここで一度、１関数の書式、２関数の役割の説明、３用いる引数、４返される戻り値またはポインタ渡しの詳細、５関数の使用例、６関数の内部コードをまとめた。関数のドキュメントを作製した。

2.1　GetaFun

2.1.1　書式

Uword *GetaFun(int num, Cpub *cpub);

2.1.2　説明

整数 num によってオペランド A の場所を判定する。

2.1.3　引数

int num　：オペランドを指定する命令コード Cpub *cpub　：cpub のポインタ

2.1.4　戻り値

オペランド A のポインタ

2.1.5　使用例

```
1  Uword *opa;
2  opa = GetaFun(cpub->ir, cpub);
```

2.1.6　内部コード

```
1  Uword* GetaFun(int num, Cpub *cpub) { // Decode opa by addressing mode
2          switch (num % 16) {
3          case 0 ... 7:
4                  return &cpub->acc;
5                  break;
6          case 8 ... 0xF:
7                  return &cpub->ix;
8                  break;
9          default:
10                 return NULL;
11                 break;
```

```
12          }
13  }
```

2.2　GetbFun

2.2.1　書式

Uword* GetbFun(int num, Cpub *cpub);

2.2.2　説明

整数 num によってオペランド B の場所を判定する。

2.2.3　引数

int num　：オペランドを指定する命令コード Cpub *cpub　：cpub のポインタ

2.2.4　戻り値

オペランド B のポインタ

2.2.5　使用例

```
1  Uword *opb;
2  opb = GetbFun(cpub->ir, cpub);
```

2.2.6　内部コード

```
1  Uword* GetbFun(int num, Cpub *cpub) { // Decode opb by addressing mode
2          switch (num % 8) {
3          case 0:
4                  return &cpub->acc;
5                  break;
6          case 1:
7                  return &cpub->ir;
8                  break;
9          case 2:
10                 return &cpub->mem[cpub->pc++];
11                 break;
12         case 4:
13                 return &cpub->mem[cpub->mem[cpub->pc++]];
14                 break;
15         case 5:
16                 return &cpub->mem[cpub->mem[cpub->pc++] + IMEMORY_SIZE];
17                 break;
18         case 6:
19                 return &cpub->mem[cpub->mem[cpub->pc++] + cpub->ix];
20                 break;
21         case 7:
```

```
22                  return &cpub->mem[cpub->mem[cpub->pc++] + IMEMORY_SIZE + cpub->ix];
23                  break;
24          default:
25                  return NULL;
26                  break;
27          }
28  }
```

2.3　NF

2.3.1　書式

void NF(Uword result, Bit *nf);

2.3.2　説明

NF(Negative Flag) をセットする。演算結果が 0 より小さいとき NF は 1 にセットされる。

2.3.3　引数

Uword result：演算結果 Bit *nf：セットしたい nf のポインタ

2.3.4　戻り値

ポインタ渡し。0 または 1 がセットされた nf のポインタ

2.3.5　使用例

```
1  Uword result = 1;
2  NF(result, &cpub->nf);
```

2.3.6　内部コード

```
1  void NF(Uword result, Bit *nf) {
2          *nf = ((result | 0xBF) != 0xFF) ? 0 : 1;
3  }
```

2.4　ZF

2.4.1　書式

void ZF(Uword result, Bit *zf);

2.4.2　説明

ZF(Zero Flag) をセットする。演算結果が 0 のとき zF は 1 にセットされる。

2.4.3　引数

Uword result：演算結果 Bit *zf：セットしたい zf のポインタ

2.4.4　戻り値

ポインタ渡し。0 または 1 がセットされた zf のポインタ

2.4.5　使用例

```
1  Uword result = 1;
2  ZF(result, &cpub->nf);
```

2.4.6　内部コード

```
1  void ZF(Uword result, Bit *zf) {
2          *zf = (result == 0) ? 1 : 0;
3  }
```

2.5　VF

2.5.1　書式

void VF(Uword *a, Uword *b, Bit *vf);

2.5.2　説明

VF(Overflowr Flag) をセットする。オーバーフローが発生したとき VF が 1 にセットされる。

2.5.3　引数

Uword *a, *b : 演算に使用するデータのポインタ Uword *vf : セットしたい vf のポインタ

2.5.4　戻り値

ポインタ渡し。0 または 1 がセットされた vf のポインタ

2.5.5　使用例

```
1  Uword *a = &cpub->acc;
2  Uword +b = &cpub->ix;
3  CF(a, b, &cpub->cf);
```

2.5.6　内部コード

```
1  void VF(Uword *a, Uword *b, Bit *vf) {
2          Uword result = *a + *b;
3          if ((result >= 0) && (*a < 0) && (*b < 0)) {
4                  *vf = 1;
5          } else if ((result <= 0) && (*a > 0) && (*b > 0)) {
6                  *vf = 1;
7          } else {
```

```
8                   *vf = 0;
9           }
10  }
```

2.6　CF

2.6.1　書式

void CF(Uword *a, Uword *b, Bit *cf);

2.6.2　説明

CF(Carry Flag) をセットする。キャリーが発生したとき CF が 1 にセットされる。

2.6.3　引数

Uword *a, *b：演算に使用するデータのポインタ Uword *cf：セットしたい cf のポインタ

2.6.4　戻り値

ポインタ渡し。0 または 1 がセットされた cf のポインタ

2.6.5　使用例

```
1  Uword *a = &cpub->acc;
2  Uword *b = &cpub->ix;
3  CF(a, b, &cpub->cf);
```

2.6.6　内部コード

```
1  void CF(Uword *a, Uword *b, Bit *cf) {
2          Uword result = *a + *b;
3          *cf = *a < result ? 0 : 1;
4  }
```

2.7　CVFlagFun

2.7.1　書式

void CVFlagFun(Uword *a, Uword *b, Cpub *cpub);

2.7.2　説明

CF と VF をセット・リセットする。

2.7.3　引数

Uword *a, *b：演算に使用する整数のポインタ Cpub *cpub：cpub のポインタ

2.7.4　戻り値

ポインタ渡し。0 または 1 がセットされた vf のポインタと cf のポインタ

2.7.5　使用例

```
1  Uword *a = &cpub->acc;
2  Uword +b = &cpub->ix;
3  CVFlagFun(a,b,cpub);
```

2.7.6　内部コード

```
1  void CVFlagFun(Uword *a, Uword *b, Cpub *cpub) {
2          Uword *vf = &cpub->vf;
3          CF(a, b, &cpub->cf);
4          *vf = cpub->cf;
5  }
```

2.8　NZFlagFun

2.8.1　書式

void NZFlagFun(Uword result, Cpub *cpub);

2.8.2　説明

NF と ZF をセット・リセットする。

2.8.3　引数

Uword result：演算結果 Cpub *cpub：cpub のポインタ

2.8.4　戻り値

ポインタ渡し。0 または 1 がセットされた nf のポインタと zf のポインタ

2.8.5　使用例

```
1  Uword result = 1;
2  NF(result, cpub);
```

2.8.6　内部コード

```
1  void NZFlagFun(Uword result, Cpub *cpub) {
2          NF(result, &cpub->nf);
3          ZF(result, &cpub->zf);
4  }
```

2.9　BbcFlag

2.9.1　書式

void BbcFlag(Bit f, Cpub *cpub, int condtion);

2.9.2　説明

フラグによる分岐条件を処理をする。フラグの値によって次命令実行すべき場所の値を PC に代入する。

2.9.3　引数

Bit f：分岐条件に使用するフラグ Cpub *cpub：cpub のポインタ int condtion：フラグの条件の値

2.9.4　戻り値

ポインタ渡し。フラグの値によって次命令実行すべき場所の値を PC に代入する。

2.9.5　使用例

```
1  case 0x31: //BNZ
2    BbcFlag(cpub->zf, cpub, 0);
```

2.9.6　内部コード

```
1  void BbcFlag(Bit f, Cpub *cpub, int condtion) {
2          Uword *pc = &cpub->pc;
3          Uword b = cpub->mem[cpub->pc];
4          *pc = f == condtion ? b : *pc + 1;
5  }
```

2.10　SLA

2.10.1　書式

void SRA(Cpub *cpub, Uword *a);

2.10.2　説明

整数*a を算術左シフト SRA(shift left arithmetic) する。

2.10.3　引数

Cpub *cpub：cpub のポインタ Uword *a：オペランド A のポインタ

2.10.4　戻り値

ポインタ渡し。整数 a を左にシフトさせる。msb の値を cf と vf に代入する。

2.10.5　使用例

```
case 0x40: //SRA
        case 0x48:
                SRA(cpub, opa);
```

2.10.6　内部コード

```
void SLA(Cpub *cpub, Uword *a) {
        Bit msb, lsb;
        Uword *cf = &cpub->cf;
        Uword *vf = &cpub->vf;
        MsbLsbFun(a, &msb, &lsb);
        *cf = msb;
        *vf = msb;
        *a = *a << 1;
        NZFlagFun(*a,cpub);
}
```

2.11　SRA

2.11.1　書式

void SRA(Cpub *cpub, Uword *a);

2.11.2　説明

整数*a を算術右シフト SRA(shift right arithmetic) する。

2.11.3　引数

Cpub *cpub：cpub のポインタ Uword *a：オペランド A のポインタ

2.11.4　戻り値

ポインタ渡し。整数 a を右にシフトさせる。msb の値を cf と vf に代入する。

2.11.5　使用例

```
case 0x40: //SRA
        case 0x48:
                SRA(cpub, opa);
```

2.11.6　内部コード

```
void SRA(Cpub *cpub, Uword *a) {
        Bit msb, lsb;
```

```
3         Uword *cf = &cpub->cf;
4         Uword *vf = &cpub->vf;
5         *vf = 0;
6         MsbLsbFun(a, &msb, &lsb);
7         *cf = lsb;
8         *a = *a >> 1;
9         *a = (*cf == 1) ? *a | 0x80 : *a | 0x00;
10        NZFlagFun(*a,cpub);
11    }
```

2.12　SRL

2.12.1　書式

SRL(Cpub *cpub, Uword *a);

2.12.2　説明

整数*a を算術右シフト SRL(shift right logical) する。

2.12.3　引数

Cpub *cpub：cpub のポインタ Uword *a：オペランド A のポインタ

2.12.4　戻り値

ポインタ渡し。整数 a を右にシフトさせる。msb の値を cf に代入し vf を 0 にする。

2.12.5　使用例

```
1   case 0x42: //SRL
2         case 0x4A:
3                   SRL(cpub, opa);
```

2.12.6　内部コード

```
1   void SLL(Cpub *cpub, Uword *a) {
2         Bit msb, lsb;
3         Uword *cf = &cpub->cf;
4         Uword *vf = &cpub->vf;
5         *vf = 0;
6         MsbLsbFun(a, &msb, &lsb);
7         *cf = msb;
8         *a = *a << 1;
9         NZFlagFun(*a,cpub);
10  }
```

2.13　SLL

2.13.1　書式

SLL(Cpub *cpub, Uword *a);

2.13.2　説明

整数*a を算術左シフト SLL(shift left logical) する。

2.13.3　引数

Cpub *cpub：cpub のポインタ Uword *a：オペランド A のポインタ

2.13.4　戻り値

ポインタ渡し。整数 a を左にシフトさせる。msb の値を cf に代入し vf を 0 にする。

2.13.5　使用例

```
1  case 0x43: //SLL
2          case 0x4B:
3                  SLL(cpub, opa);
```

2.13.6　内部コード

```
1  void SLL(Cpub *cpub, Uword *a) {
2          Bit msb, lsb;
3          Uword *cf = &cpub->cf;
4          Uword *vf = &cpub->vf;
5          *vf = 0;
6          MsbLsbFun(a, &msb, &lsb);
7          *cf = msb;
8          *a = *a << 1;
9          NZFlagFun(*a,cpub);
10 }
```

2.14　RRA

2.14.1　書式

void RRA(Cpub *cpub, Uword *a);

2.14.2　説明

整数*a を算術右回転 RRA(Rotate Right Athmetic) する。

2.14.3　引数

Cpub *cpub：cpub のポインタ Uword *a：オペランド A のポインタ

2.14.4　戻り値

ポインタ渡し。整数 a を右にシフトさせ、最下位ビットを最上位ビットに挿入する。

2.14.5　使用例

```
case 0x44: //RRA
        case 0x4C:
                RRA(cpub, opa);
```

2.14.6　内部コード

```
void RRA(Cpub *cpub, Uword *a) {
        Bit msb, lsb;
        Uword *cf = &cpub->cf;
        Uword *vf = &cpub->vf;
        *vf = 0;
        MsbLsbFun(a, &msb, &lsb);
        *cf = lsb;
        *a = *a >> 1;
        *a = (*cf == 1) ? *a | 0x80 : *a | 0x00;
        NZFlagFun(*a,cpub);
}
```

2.15　RLA

2.15.1　書式

void RLA(Cpub *cpub, Uword *a);

2.15.2　説明

整数*a を算術左回転 RLA(Rotate Left Athmetic) する。

2.15.3　引数

Cpub *cpub：cpub のポインタ Uword *a：オペランド A のポインタ

2.15.4　戻り値

ポインタ渡し。整数 a を左にシフトさせ、最上位ビットを最下位ビットに挿入する。

2.15.5　使用例

```
case 0x45: //RLA
```

```
2         case 0x4D:
3                 RLA(cpub, opa);
```

2.15.6　内部コード

```
1  void RLA(Cpub *cpub, Uword *a) {
2         Bit msb, lsb;
3         Uword *cf = &cpub->cf;
4         Uword *vf = &cpub->vf;
5         MsbLsbFun(a, &msb, &lsb);
6         *cf = msb;
7         *vf = msb;
8         *a = *a << 1;
9         *a = (*cf == 1) ? *a | 0x01 : *a | 0x00;
10        NZFlagFun(*a,cpub);
11 }
```

2.16　RRL

2.16.1　書式

void RRL(Cpub *cpub, Uword *a);

2.16.2　説明

整数*a を論理的右回転 RRL(Rotate Right Athmetic) する。

2.16.3　引数

Cpub *cpub：cpub のポインタ Uword *a：オペランド A のポインタ

2.16.4　戻り値

ポインタ渡し。整数 a を右にシフトさせ、最上位ビットを最下位ビットに挿入する。

2.16.5　使用例

```
1  case 0x46: //RRL
2         case 0x4E:
3                 RRL(cpub, opa);
```

2.16.6　内部コード

```
1  void RRL(Cpub *cpub, Uword *a) {
2         Bit msb, lsb;
3         Uword *cf = &cpub->cf;
4         Uword *vf = &cpub->vf;
5         *vf = 0;
```

```
6         MsbLsbFun(a, &msb, &lsb);
7         *cf = lsb;
8         *a = *a >> 1;
9         *a = (*cf == 1) ? *a | 0x80 : *a | 0x00;
10        NZFlagFun(*a,cpub);
11    }
```

2.17　RLL

2.17.1　書式

void RLL(Cpub *cpub, Uword *a);

2.17.2　説明

整数*a を論理的左回転 RLL(Rotate Left Athmetic) する。

2.17.3　引数

Cpub *cpub：cpub のポインタ Uword *a：オペランド A のポインタ

2.17.4　戻り値

ポインタ渡し。整数 a を左にシフトさせ、最上位ビットを最下位ビットに挿入する。

2.17.5　使用例

```
1   case 0x47: //RLL
2         case 0x4F:
3                   RLL(cpub, opa);
```

2.17.6　内部コード

```
1   void RLL(Cpub *cpub, Uword *a) {
2         Bit msb, lsb;
3         Uword *cf = &cpub->cf;
4         Uword *vf = &cpub->vf;
5         *vf = 0;
6         MsbLsbFun(a, &msb, &lsb);
7         *cf = msb;
8         *a = *a << 1;
9         *a = (*cf == 1) ? *a | 0x01 : *a | 0x00;
10        NZFlagFun(*a,cpub);
11    }
```

2.18　MsbLsbFun

2.18.1　書式

void MsbLsbFun(Uword *a, Bit *msb, Bit *lsb);

2.18.2　説明

整数 a の最上位ビットと最下位ビットをセットする。

2.18.3　引数

Uword *a：整数 a のポインタ Bit *msb：最上位ビットのポインタ Bit *lsb：最下位ビットのポインタ

2.18.4　戻り値

ポインタ渡し。整数 a の 0

2.18.5　使用例

```
1  Bit msb, lsb;
2  Uword *a;
3  MsbLsbFun(a, &msb, &lsb);
```

2.18.6　内部コード

```
1  void MsbLsbFun(Uword *a, Bit *msb, Bit *lsb) {
2          *msb = ((*a & 0x80) == 0x80) ? 1 : 0;
3          *lsb = ((*a & 0x01) == 0x01) ? 1 : 0;
4  }
```

2.19　LD

2.19.1　書式

void LD(Uword *a, Uword *b);

2.19.2　説明

整数 b の値を a に代入する命令 LD を行う。

2.19.3　引数

Uword *a, *b：オペランド A と B のポインタ

2.19.4　戻り値

ポインタ渡し。整数 b の値を a に代入する。

2.19.5　使用例

```
case 0x60 ... 0x6F: //LD
                    LD(opa, opb);
```

2.19.6　内部コード

```
void LD(Uword *a, Uword *b) {
        *a = *b;
}
```

2.20　ST

2.20.1　書式

void ST(Uword *a, Uword *b);

2.20.2　説明

整数 a の値を b に代入する命令 ST を行う。

2.20.3　引数

Uword *a, *b : オペランド A と B のポインタ

2.20.4　戻り値

ポインタ渡し。整数 a の値を b に代入する。

2.20.5　使用例

```
case 0x74 ... 0x7F: //ST
                    ST(opa, opb);
```

2.20.6　内部コード

```
void ST(Uword *a, Uword *b) {
        *b = *a;
}
```

2.21　EOR

2.21.1　書式

void EOR(Uword *a, Uword *b, Cpub *cpub);

2.21.2　説明

整数 a の値と b の排他的演算命令 EOR を行う。

2.21.3　引数

Uword *a, *b：オペランド A と B のポインタ

2.21.4　戻り値

ポインタ渡し。整数 a と b の排他的演算の結果を a に代入する。

2.21.5　使用例

```
1  case 0xC0 ... 0xCF: //EOR
2                  EOR(opa, opb, cpub);
```

2.21.6　内部コード

```
1  void EOR(Uword *a, Uword *b, Cpub *cpub) {
2          Uword *vf = &cpub->vf;
3          *vf = 0;
4          *a = *a ^ *b;
5          NZFlagFun(*a, cpub);
6  }
```

2.22　ADD

2.22.1　書式

void ADD(Uword *a, Uword *b, Cpub *cpub);

2.22.2　説明

整数 a の値と b の足し算命令 ADD を行う。

2.22.3　引数

Uword *a, *b：オペランド A と B のポインタ

2.22.4　戻り値

ポインタ渡し。整数 a と b の足し算の結果を a に代入する。

2.22.5　使用例

```
1  case 0xB0 ... 0xBF: //ADD
2                  ADD(opa, opb, cpub);
```

2.22.6　内部コード

```
1  void ADD(Uword *a, Uword *b, Cpub *cpub) {
2          VF(a,b,&cpub->vf);
3          *a += *b;
4          NZFlagFun(*a, cpub);
5  }
```

2.23　ADC

2.23.1　書式

void ADC(Uword *a, Uword *b, Cpub *cpub)；

2.23.2　説明

整数 a の値と b とキャリーフラグの足し算命令 ADC を行う。

2.23.3　引数

Uword *a, *b：オペランド A と B のポインタ

2.23.4　戻り値

ポインタ渡し。整数 a と b とキャリーフラグの足し算の結果を a に代入する。

2.23.5　使用例

```
1  case 0x90 ... 0x9F: //ADC
2                      ADC(opa, opb, cpub);
```

2.23.6　内部コード

```
1  void ADC(Uword *a, Uword *b, Cpub *cpub) {
2          Bit addcf = cpub->cf;
3          CVFlagFun(a,b,cpub);
4          *a = *a + *b + addcf;
5          NZFlagFun(*a,cpub);
6  }
```

2.24　SUB

2.24.1　書式

void SUB(Uword *a, Uword *b, Cpub *cpub);

2.24.2　説明

整数 a の値と b の引き算命令 SUB を行う。

2.24.3　引数

Uword *a, *b：オペランド A と B のポインタ

2.24.4　戻り値

ポインタ渡し。整数 a と b の引き算の結果を a に代入する。

2.24.5　使用例

```
case 0xA0 ... 0xAF: //SUB
                    SUB(opa, opb, cpub);
```

2.24.6　内部コード

```
void SUB(Uword *a, Uword *b, Cpub *cpub) {
        VF(a,b,&cpub->vf);
        *a -= *b;
        NZFlagFun(*a, cpub);
}
```

2.25　SBC

2.25.1　書式

void SBC(Uword *a, Uword *b, Cpub *cpub);

2.25.2　説明

整数 a の値と b とキャリーフラグの引き算命令を行う。

2.25.3　引数

Uword *a, *b：オペランド A と B のポインタ

2.25.4　戻り値

ポインタ渡し。整数 a と b の引き算の結果を a に代入する。

2.25.5　使用例

```
case 0x80 ... 0x8F: //SBC
                    SBC(opa, opb, cpub);
```

2.25.6　内部コード

```
void SBC(Uword *a, Uword *b, Cpub *cpub) {
        Bit addcf = cpub->cf;
```

```
3         CVFlagFun(a,b,cpub);
4         *a = *a - *b - addcf;
5         NZFlagFun(*a,cpub);
6   }
```

2.26　OR

2.26.1　書式

void OR(Uword *a, Uword *b, Cpub *cpub);

2.26.2　説明

整数 a の値と b の論理和命令 OR を行う。

2.26.3　引数

Uword *a, *b：オペランド A と B のポインタ Cpub *cpub：cpub のポインタ

2.26.4　戻り値

ポインタ渡し。整数 a と b の論理和演算の結果を a に代入する。

2.26.5　使用例

```
1   case 0xD0 ... 0xDF: //OR
2                   OR(opa, opb, cpub);
```

2.26.6　内部コード

```
1   void OR(Uword *a, Uword *b, Cpub *cpub) {
2         Uword *vf = &cpub->vf;
3         *vf = 0;
4         *a = (*a | *b);
5         NZFlagFun(*a, cpub);
6   }
```

2.27　AND

2.27.1　書式

void AND(Uword *a, Uword *b, Cpub *cpub);

2.27.2　説明

整数 a の値と b の論理積命令 AND を行う。

2.27.3　引数

Uword *a, *b：オペランド A と B のポインタ Cpub *cpub：cpub のポインタ

2.27.4　戻り値

ポインタ渡し。整数 a と b の論理積演算の結果を a に代入する。

2.27.5　使用例

```
1  case 0xE0 ... 0xEF: //AND
2                  AND(opa, opb, cpub);
```

2.27.6　内部コード

```
1  void AND(Uword *a, Uword *b, Cpub *cpub) {
2          Uword *vf = &cpub->vf;
3          *vf = 0;
4          *a = (*a & *b);
5          NZFlagFun(*a, cpub);
6  }
```

2.28　CMP

2.28.1　書式

void CMP(Uword *a, Uword *b, Cpub *cpub);

2.28.2　説明

整数 a の値と b の比較命令 CMP を行う。

2.28.3　引数

Uword *a, *b：オペランド A と B のポインタ Cpub *cpub：cpub のポインタ

2.28.4　戻り値

ポインタ渡し。整数 a と b の比較を行う。

2.28.5　使用例

```
1  case 0xF0 ... 0xFF: //CMP
2                  CMP(opa, opb, cpub);
```

2.28.6　内部コード

```
1  void CMP(Uword *a, Uword *b, Cpub *cpub) {
2          VF(a,b,&cpub->vf);
3          Uword result = *a - *b;
4          NZFlagFun(result, cpub);
5  }
```

3　実習 2 シミュレータを作成せよ

実習に引き続き step 関数の中身を加えることで複数の命令を実行できるようにした。具体的には実習 1 では $cpub->ir$ の値を switch 文で case に分けて処理をした。この $cpub->ir$ の値を他の命令文でも対応できるようにした。switch 文の case でひとつひとつ命令コードの処理を書くのは何度も使用するものが出て冗長であり、また非常に長い行数となり読みにくいものとなる。そこで、よく使う形のものであったり、処理が長くて複雑なものは関数化した。関数を作成することで効率性や可読性を上げることができた。

3.1　cpub.h について

図 1 は cpuboard.h に書き加えたものである。ヘッダーファイルは定数や宣言などを書くためのファイルである。そこで、cpuboard.h には新たに cpuboard.c で加えた関数を cpuboard.h で宣言した。特出すべき関数をいくつかあげみてみる。

3.1.1　GetaFun()、GetbFun について (2.1,2.2)

```
Uword *GetaFun(int num, Cpub *cpub);
Uword *GetbFun(int num, Cpub *cpub);
```

GetaFun()、GetbFun() では命令コードの 1 桁目を見てオペランド A、B を返す関数である。表 3 の (a) データ移動/算術演算命令/論理演算命令に注目すると、縦の列の一桁目はすべて一緒である。例えば、オペランド A が ACC でオペランド B が [d] のとき、命令コードは 64H,74H,84H,94H..F4H である。このように命令コードによらず、一桁目は 4 である。つまり、一桁目が 4 であるときは、オペランド A、オペランド B は ACC、[d] と分かる。このように一桁目を見て、オペランド A、オペランド B の場所を Uword のポインタで返す関数が GetaFun()、GetbFun() である。
まず、GetaFun() についてみてみる。表 3 の (a) より、一桁目が 0 から 7 のときオペランド A は acc である。また一桁目が 8 から F のとき ix である。これを switch 文で分岐しさせ、戻り値としてポインタを返す。次に GetbFun() をみてみる。表 3 の (a) より、一桁目が 0 のときオペランド B は acc で、一桁目が 1 のときは ir である。GetaFun() と同様に switch 文で分岐させ、戻り値としてポインタを返す。

3.1.2　フラグ関数について (2.3,2.4,2.5,2.6,2.7,2.8)

```
void NF(Uword resultValue, Bit *nf);
void ZF(Uword resultValue, Uword *zf);
void CF(Uword *a, Uword *b, Bit *cf);
void VF(Uword *a, Uword *b, Bit *of);
void CVFlagFun(Uword *a, Uword *b, Cpub *cpub);
void NZFlagFun(Uword result, Cpub *cpub);
```

NF()、ZF()、CF()、VF() はそれぞれのフラグをセットする関数である。nf は実行結果が負かどうかみる。zf は実行結果が 0 かどうかみる。cf はキャリーが生じたかどうかみる。キャリーが生じたとき、実行結果は実行前に比べ CVFlagFun() は cf と vf を設定する関数である。CF() 関数と VF() 関数をそれぞれ呼び出すのは大変なので、一回の引数で 2 つのフラグが設定されるようにした。一方で NZFlagFun() は nf と zf を設定する関数である。

3.1.3　shift と rotate 関数について (2.12,2.10,2.13, 2.14,2.15,2.16,2.17, 2.18)

```
void SRA(Cpub *cpub,Uword *a);
void SLA(Cpub *cpub,Uword *a);
void SRL(Cpub *cpub,Uword *a);
void SLL(Cpub *cpub,Uword *a);
void RRA(Cpub *cpub,Uword *a);
void RLA(Cpub *cpub,Uword *a);
void RRL(Cpub *cpub,Uword *a);
void RLL(Cpub *cpub,Uword *a);
void MsbLsbFun(Uword *a,Bit *msb,Bit *lsb);
```

SRA(),SLA(),SRL(),SLL(),RRA(),RLA(),RRL(),RLL() はそれぞれの方式にあった shift または rotate する関数である。MsbLsbFun() は最上位ビットと最下位ビットをセットする関数である。この最上位ビットまたは最下位ビットをもちいて、シフト演算したあとの値に代入することでこの演算を実装した。

3.1.4　BbcFlag 関数について (2.9)

```
void BbcFlag(Bit f,Cpub *cpub,int condtion)
```

分岐命令にはいくつかの種類がある。(表 1) たとえば BVF は VF が 1 のとき分岐を行う。また、BNZ は ZF が 0 のとき分岐を行う。このようにあるフラグをみてそれが 1 又は 0 のとき分岐を行うという命令が複数ある。そこで、第一引数にフラグ、第三引数に分岐条件を与え、分岐処理を行う。例えば、

```
BbcFlag(cpub->vf,cpub,1);
BbcFlag(cpub->zf,cpub,0);
```

また成立条件が満たされたとき PC は B' の値が代入され、成立しないときは PC をインクリメントし、次の命令へと移る。

3.1.5　命令コマンド関数について (2.19,2.20,2.22, 2.23,2.24,2.25,2.28,2.27, 2.26,2.21)

```
void LD(Uword *a, Uword *b);
void ST(Uword *a, Uword *b);
void ADD(Uword *a, Uword *b, Cpub *cpub);
void ADC(Uword *a, Uword *b, Cpub *cpub);
```

表 1: Branc Condition

$\spadesuit^{\ddagger} bc$: Branch Condition

A	0	0	0	0	常に成立
VF	1	0	0	0	桁あふれ　$VF = 1$
NZ	0	0	0	1	$\neq 0$　$ZF = 0$
Z	1	0	0	1	$= 0$　$ZF = 1$
ZP	0	0	1	0	≥ 0　$NF = 0$
N	1	0	1	0	< 0　$NF = 1$
P	0	0	1	1	> 0　$(NF \vee ZF) = 0$
ZN	1	0	1	1	≤ 0　$(NF \vee ZF) = 1$
NI	0	1	0	0	$IBUF\_FLG\_IN = 0$
NO	1	1	0	0	$OBUF\_FLG\_IN = 1$
NC	0	1	0	1	$CF = 0$
C	1	1	0	1	$CF = 1$
GE	0	1	1	0	≥ 0　$(VF \oplus NF) = 0$
LT	1	1	1	0	< 0　$(VF \oplus NF) = 1$
GT	0	1	1	1	> 0　$((VF \oplus NF) \vee ZF) = 0$
LE	1	1	1	1	≤ 0　$((VF \oplus NF) \vee ZF) = 1$

```
void SUB(Uword *a, Uword *b, Cpub *cpub);
void SBC(Uword *a, Uword *b, Cpub *cpub);
void CMP(Uword *a, Uword *b, Cpub *cpub);
void AND(Uword *a, Uword *b, Cpub *cpub);
void OR(Uword *a, Uword *b, Cpub *cpub);
void EOR(Uword *a, Uword *b, Cpub *cpub);
```

LD(),ST(),ADD(),ADC(),SUB(),CMP(),AND(),OR(),EOR() はそれぞれ命令コマンドの処理を行う関数である。引数は Uword *a　Uword　*b が使われているが、これらはオペランド A と B のポインタである。また、フラグは実行への影響及び、実行後の状態に関わってくる。そこで、フラグをセットする関数 (VF(),CF().ZF(),NF()) を関数内で呼び出しフラグを調整した。

表 2: フラグ機能

略記号	実行への影響 †				実行後の状態 ‡			
	CF	VF	NF	ZF	CF	VF	NF	ZF
NOP/HLT	—	—	—	—	—	—	—	—
OUT/IN	—	—	—	—	—	—	—	—
RCF	—	—	—	—	0	—	—	—
SCF	—	—	—	—	1	—	—	—
LD/ST	—	—	—	—	—	—	—	—
ADD	—	—	—	—	—	V	N	Z
ADC	c	—	—	—	C	V	N	Z
SUB	—	—	—	—	—	V	N	Z
SBC	c	—	—	—	C	V	N	Z
CMP	—	—	—	—	—	V	N	Z
AND	—	—	—	—	—	0	N	Z
OR	—	—	—	—	—	0	N	Z
EOR	—	—	—	—	—	0	N	Z
SRA	—	—	—	—	$b0$	0	N	Z
SLA	—	—	—	—	$b7$	V	N	Z
SRL	—	—	—	—	$b0$	0	N	Z
SLL	—	—	—	—	$b7$	0	N	Z
RRA	$b7$	—	—	—	$b0$	0	N	Z
RLA	$b0$	—	—	—	$b7$	V	N	Z
RRL	—	—	—	—	$b0$	0	N	Z
RLL	—	—	—	—	$b7$	0	N	Z
BA	—	—	—	—	—	—	—	—
BVF	—	VF	—	—	—	—	—	—
BNZ	—	—	—	$\overline{ZF}$	—	—	—	—
BZ	—	—	—	ZF	—	—	—	—
BZP	—	—	$\overline{NF}$	—	—	—	—	—
BN	—	—	NF	—	—	—	—	—
BP	—	—	$\overline{NF \vee ZF}$		—	—	—	—
BZN	—	—	$NF \vee ZF$		—	—	—	—
BNI	—	—	—	—	—	—	—	—
BNO	—	—	—	—	—	—	—	—
BNC	$\overline{CF}$	—	—	—	—	—	—	—
BC	CF	—	—	—	—	—	—	—
BGE	—	$\overline{VF \oplus NF}$		—	—	—	—	—
BLT	—	$VF \oplus NF$		—	—	—	—	—
BGT	—	$\overline{(VF \oplus NF) \vee ZF}$			—	—	—	—
BLE	—	$(VF \oplus NF) \vee ZF$			—	—	—	—

フラグ略称
- CF: Carry Flag
- VF: oVerflow Flag
- NF: Negative Flag
- ZF: Zero Flag

† 実行への影響
- c: 最下位への carry/borrow 入力となる
- $b0$: オペランド A の第 0 ビットとなる
- $b7$: オペランド A の第 7 ビットとなる
- 式: 分岐の成立する条件（論理）を示す
- —: 影響なし

‡ 実行後の状態
- 0: 0 にリセット
- 1: 1 にセット
- C: carry/borrow の発生によりセット/リセット
- V: オーバフローの発生によりセット/リセット
- N: 演算結果の第 7 ビットの値に設定
- Z: 演算結果が 0 ならセット，0 以外ならリセット
- $b0$: オペランド A の第 0 ビットの値に設定
- $b7$: オペランド A の第 7 ビットの値に設定
- —: 変化なし

```
1   //Get Opeland A or B
2   Uword *GetaFun(int num, Cpub *cpub);
3   Uword *GetbFun(int num, Cpub *cpub);
4
5   //Set Each Flagment
6   void NF(Uword resultValue, Bit *nf);
7   void ZF(Uword resultValue, Uword *zf);
8   void CF(Uword *a, Uword *b, Bit *cf);
9   void VF(Uword *a, Uword *b, Bit *of);
10  void AllFlagFun(Uword *a, Uword *b, Cpub *cpub);
11  void VNZFlagFun(Uword *a, Uword *b, Cpub *cpub);
12  void NZFlagFun(Uword result, Cpub *cpub);
13
14  //Execute Each Shift or Rotate command
15  void SRA(Cpub *cpub,Uword *a);
16  void SLA(Cpub *cpub,Uword *a);
17  void SRL(Cpub *cpub,Uword *a);
18  void SLL(Cpub *cpub,Uword *a);
19  void RRA(Cpub *cpub,Uword *a);
20  void RLA(Cpub *cpub,Uword *a);
21  void RRL(Cpub *cpub,Uword *a);
22  void RLL(Cpub *cpub,Uword *a);
23  void MsbLsbFun(Uword *a,Bit *msb,Bit *lsb);
24
25  //Execute Branch Condtion of Flag
26  void BbcFlag(Bit f,Cpub *cpub,int condtion);
27
28  //Execute DataMove, Athmetic or Logical command
29  void LD(Uword *a, Uword *b);
30  void ST(Uword *a, Uword *b);
31  void ADD(Uword *a, Uword *b, Cpub *cpub);
32  void ADC(Uword *a, Uword *b, Cpub *cpub);
33  void SUB(Uword *a, Uword *b, Cpub *cpub);
34  void SBC(Uword *a, Uword *b, Cpub *cpub);
35  void CMP(Uword *a, Uword *b, Cpub *cpub);
36  void AND(Uword *a, Uword *b, Cpub *cpub);
37  void OR(Uword *a, Uword *b, Cpub *cpub);
38  void EOR(Uword *a, Uword *b, Cpub *cpub);
```

図 1: 実習 2 における cpubord.h に追加した関数

3.2　cpub.c について

cpub.c は 6 のように実装した。まず、命令フェッチの部分を見てみる。まず、pc の値を mar にコピーする。次に、memory から mar の値の番地の値を取り出し、ir にコピーする。そして、GetaFun() を呼び出す。GetaFun() は命令コードから A のオペランドをとってポインタとして返す。その A のオペランドのポイン

表 3: 命令語コード早見表

(a) データ移動命令／算術演算命令／論理演算命令

オペランド $B \Rightarrow$		ACC	IX	d	[d]	(d)	[IX+d]	(IX+d)
LD	ACC/IX,	60/68	61/69	62/6A	64/6C	65/6D	66/6E	67/6F
ST	ACC/IX,	− −	− −	− −	74/7C	75/7D	76/7E	77/7F
ADD	ACC/IX,	B0/B8	B1/B9	B2/BA	B4/BC	B5/BD	B6/BE	B7/BF
ADC	ACC/IX,	90/98	91/99	92/9A	94/9C	95/9D	96/9E	97/9F
SUB	ACC/IX,	A0/A8	A1/A9	A2/AA	A4/AC	A5/AD	A6/AE	A7/AF
SBC	ACC/IX,	80/88	81/89	82/8A	84/8C	85/8D	86/8E	87/8F
CMP	ACC/IX,	F0/F8	F1/F9	F2/FA	F4/FC	F5/FD	F6/FE	F7/FF
AND	ACC/IX,	E0/E8	E1/E9	E2/EA	E4/EC	E5/ED	E6/EE	E7/EF
OR	ACC/IX,	D0/D8	D1/D9	D2/DA	D4/DC	D5/DD	D6/DE	D7/DF
EOR	ACC/IX,	C0/C8	C1/C9	C2/CA	C4/CC	C5/CD	C6/CE	C7/CF

(b) 制御命令

NOP	00
HLT	0F
OUT	10
IN	1F
RCF	20
SCF	2F

(c) シフト演算命令

SRA	ACC/IX	40/48
SLA	ACC/IX	41/49
SRL	ACC/IX	42/4A
SLL	ACC/IX	43/4B
RRA	ACC/IX	44/4C
RLA	ACC/IX	45/4D
RRL	ACC/IX	46/4E
RLL	ACC/IX	47/4F

(d) 分岐命令

BA	30	BVF	38
BNZ	31	BZ	39
BZP	32	BN	3A
BP	33	BZN	3B
BNI	34	BNO	3C
BNC	35	BC	3D
BGE	36	BLT	3E
BGT	37	BLE	3F

タを opa にコピーする。同様にオペランド B も GetbFun() を使ってセットする。GetbFun() では ir の値によって pc をインクリメントするので、ir が 60h から ffh までの場合オペランド B をセットするようにする。

```
1   int step(Cpub *cpub) {
2       Uword *opa, *opb;
3
4   //Fetch Instruction
5       cpub->mar = cpub->pc++;
6       cpub->ir = cpub->mem[cpub->mar];
7
8   //Fetch Opeland
9       opa = GetaFun(cpub->ir, cpub);
10      if (cpub->ir >= 0x60 && cpub->ir <= 0xFF) {
11              opb = GetbFun(cpub->ir, cpub);
12      }
```

図 2: 実習 2 における cpubord.c の命令フェッチの部分

次に命令コードを解読して実行する部分を見てみる。まず、表 3 の (b) の制御命令の部分を見てみる。0x00 は NOP である。No OPeration（何もしない）。なにもしないで、switch 文を抜ける。
0x0F は HLT である。HaLT（停止）。exit() 関数を用いて、プログラムを停止させる。引数 0 で成功終了を示す。
0x10 は OUT である。OUTput(ACC → OBUF) であるから、acc の値を obuf の buf にコピーする。

0x1F は IN である。INput(IBUF → ACC) であるから ibuf の buf の値を acc にコピーする。
0x20 は RCF である。ResetCarryFlag（0 → CF）であるから、0 を cf にコピーする。
0x2F は SCF である。SetCarryFlag(1 → CF) であるから、1 を cf にコピーする。

```
1   //Execute Instruction
2       switch (cpub->ir) {
3       case 0x00: //NOP
4               break;
5       case 0x0F: //HLT
6               exit(RUN_HALT);
7               break;
8       case 0x10: //OUT
9               cpub->obuf.buf = cpub->acc;
10              break;
11      case 0x1F: //IN
12              cpub->acc = cpub->ibuf->buf;
13              break;
14      case 0x20: //RCF
15              cpub->cf = 0;
16              break;
17      case 0x2F: //SCF
18              cpub->cf = 1;
19              break;
```

図 3: 実習 2 における cpubord.c の制御命令

次に、表 3 の (d) の分岐命令の部分を見てみる。0x30 は BA である。Branch Condition Always(常に成立) である。命令コード　数字 の数字の番地に移動する。pc の値の番地の memory を pc にコピーする。
0x31〜0x3F はある flag がある値のとき命令コードの 2 語目の値へ PC を移動する命令である。BbcFlag 関数を用いて、第一引数に分岐条件のフラグを、第二引数に cpub を、第三引数に、分岐条件のフラグの値をいれる。たとえば、BNZ は Branch Non ZeroFlag（zf = 0 のとき分岐）であるから

```
BbcFlag(cpub->zf,cpub,0)
```

となる

```
case 0x30: //BA
  cpub->pc = cpub->mem[cpub->pc];
  break;
case 0x31: //BNZ
  BbcFlag(cpub->zf, cpub, 0);
  break;
case 0x32: //BZP
  BbcFlag(cpub->nf, cpub, 0);
  break;
case 0x33: //BP
  BbcFlag(cpub->nf, cpub, 1);
  break;
case 0x34: //BNI
  BbcFlag(cpub->ibuf->flag, cpub, 0);
  break;
case 0x35: //BNC
  BbcFlag(cpub->cf, cpub, 0);
  break;
case 0x36: //BGE
  BbcFlag(cpub->vf ^ cpub->nf, cpub, 0);
  break;
case 0x37: //BGT
  BbcFlag((cpub->vf ^ cpub->nf) | cpub->zf, cpub, 0);
  break;
case 0x38: //BVF
  BbcFlag(cpub->vf, cpub, 1);
  break;
case 0x39: //BZ
  BbcFlag(cpub->zf, cpub, 1);
  break;
case 0x3A:  //BN
  BbcFlag(cpub->nf, cpub, 1);
  break;
case 0x3B:  //BZN
  BbcFlag(cpub->nf, cpub, 0);
  break;
case 0x3C: //BNO
  BbcFlag(cpub->obuf.flag, cpub, 1);
  break;
case 0x3D: //BC
  BbcFlag(cpub->cf, cpub, 1);
  break;
case 0x3E: //BLT
  BbcFlag(cpub->vf ^ cpub->nf, cpub, 1);
  break;
case 0x3F: //BLE
  BbcFlag(cpub->vf ^ cpub->nf, cpub, 1);
  break;
```

図 4: 実習 2 における cpubord.c の制御命令

次に、表 3 の (c) のシフト演算命令の部分を見てみる。0x40〜0x4F は Ssm と Rsm である。シフト演算には SRA、SLA、SRL、SLL がある。それぞれの関数の第一引数は cpub で第二引数には opa をいれる。オペランド A は acc と ix の 2 種類あるので、それぞれ対応する命令コードも 2 種類ある。例えば、SRA なら 0x40 と 0x48 の 2 種類ある。
回転演算には RRA、RLA、RRL、RLL がある。シフト演算の関数と同様に、第一引数は cpub で第二引数には opa をいれる。オペランド A は acc と ix の 2 種類あるので、それぞれ対応する命令コードも 2 種類ある。例えば、RRA なら 0x44 と 0x4C の 2 種類ある。

```
1        case 0x40: //SRA
2        case 0x48:
3                SRA(cpub, opa);
4                break;
5        case 0x41: //SLA
6        case 0x49:
7                SLA(cpub, opa);
8                break;
9        case 0x42: //SRL
10       case 0x4A:
11               SRL(cpub, opa);
12               break;
13       case 0x43: //SLL
14       case 0x4B:
15               SLL(cpub, opa);
16               break;
17       case 0x44: //RRA
18       case 0x4C:
19               RRA(cpub, opa);
20               break;
21       case 0x45: //RLA
22       case 0x4D:
23               RLA(cpub, opa);
24               break;
25       case 0x46: //RRL
26       case 0x4E:
27               RRL(cpub, opa);
28               break;
29       case 0x47: //RLL
30       case 0x4F:
31               RLL(cpub, opa);
32               break;
```

図 5: 実習 2 における cpubord.c のシフト演算命令

次に、表 3 の (a) のデータ移動、算術演算、論理演算命令の部分を見てみる。

```
        case 0x60 ... 0x6F: //LD
                LD(opa, opb);
                break;
        case 0x74 ... 0x7F: //ST
                ST(opa, opb);
                break;
        case 0x80 ... 0x8F: //SBC
                SBC(opa, opb, cpub);
                break;
        case 0x90 ... 0x9F: //ADC
                ADC(opa, opb, cpub);
                break;
        case 0xA0 ... 0xAF: //SUB
                SUB(opa, opb, cpub);
                break;
        case 0xB0 ... 0xBF: //ADD
                ADD(opa, opb, cpub);
                break;
        case 0xC0 ... 0xCF: //EOR
                EOR(opa, opb, cpub);
                break;
        case 0xD0 ... 0xDF: //OR
                OR(opa, opb, cpub);
                break;
        case 0xE0 ... 0xEF: //AND
                AND(opa, opb, cpub);
                break;
        case 0xF0 ... 0xFF: //CMP
                CMP(opa, opb, cpub);
                break;
        default:
                printf("␣%x␣:␣no␣instruction␣code", cpub->ir);
                cpub->pc++;
                break;
    }
    return RUN_HALT;
    }
```

図 6: 実習 2 における cpubord.c の命令演算

4　実習 3　シュミレータの動作の確認

サンプルコード（図 7）を実行してみた。まず、acc を 4、ix を 3 にセットし、prog.txt を読み込む (図 8) 次に 1 行ずつ実行していく。

```
75 03
```

Address	Obj. Code	Source Code		
00	75 03	START:	ST	ACC,(03H)
02	C0		EOR	ACC,ACC
03	B5 03	LOOP:	ADD	ACC,(03H)
05	AA 01		SUB	IX,1
07	31 03		BNZ	LOOP
09	0F		HLT	
			END	

図 7: サンプルプログラム

これは acc の値を data 領域の 03 番地に格納する ST 命令である。図 9 より実行結果はデータ領域 03 番地に acc の値 04 が入っていることが分かり確かに ST が実行されたことが分かる。

```
1  C0
```

これは ACC と ACC の排他的論理和である EOR 命令である。同じ値を排他的論理和したところ値は 0 にリセットされる。図 10 より ACC は 0x00 にリセットされていて確かに EOR　ACC　ACC が実行されたことが分かる。

```
1  B5 03
```

これは ACC と data 領域 03 番地の値の和を acc にコピーする ADD 命令である。図 11 より実行前は acc が 0 であり、data 領域 03 番地の値は 4 であるが、実行後は ACC は 4 の値にセットされていて確かに ADD が実行されていることが分かる。

```
1  AA 01
```

これは IX から即値アドレス 1 を引いた値を IX に格納する SUB 命令である。実図 12 より実行前は ix が 3 であるが、実行後は ix は 2 の値にセットされていて確かに SUB が実行されていることが分かる。

```
1  31 03
```

これは zf が 0 のとき program 領域 03 番地に分岐する BNZ 命令である。図 13 より実行前は zf が 0 であるので、実行後は pc の値が 0x03 の値にセットされていて確かに BNZ が実行されていることが分かる。一方図 14 より実行前は zf が 1 であるとき、実行後は pc の値が 0x09 の値にセットされていて次の pc にインクリメントされている。確かに BNZ が実行されていることが分かる。

```
1  0F
```

これはプログラムを停止する HLT 命令である。図 15 より HLT 命令を実行するとプログラムが終了され、確かに HLT が実行されたことが分かる。

```
CPU0,PC=0x0> s acc 4
CPU0,PC=0x0> s ix 3
CPU0,PC=0x0> r prog.txt
acc=0x04(4,4) ix=0x03(3,3) cf=0 vf=0 nf=0 zf=0
ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
```

図 8: 実習 3 におけるプログラムの準備

```
CPU0,PC=0x0> i
Program Halted.
CPU0,PC=0x2> m
    | 000: 75 03 c0 b5 03 aa 01 31 | 008: 03 0f 00 00 00 00 00 00
    | 010: 00 00 00 00 00 00 00 00 | 018: 00 00 00 00 00 00 00 00

    | 0f0: 00 00 00 00 00 00 00 00 | 0f8: 00 00 00 00 00 00 00 00
    | 100: 00 00 00 04 00 00 00 00 | 108: 00 00 00 00 00 00 00 00
```

図 9: 実習 3 における ST

```
CPU0,PC=0x2> i
Program Halted.
CPU0,PC=0x3> d
      acc=0x00(0,0) ix=0x03(3,3) cf=0 vf=0 nf=0 zf=1
      ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
```

図 10: 実習 3 における EOR

```
CPU0,PC=0x3> d
      acc=0x00(0,0) ix=0x03(3,3) cf=0 vf=0 nf=0 zf=1
      ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
CPU0,PC=0x3> i
Program Halted.
CPU0,PC=0x5> d
      acc=0x04(4,4) ix=0x03(3,3) cf=0 vf=0 nf=0 zf=0
      ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
```

図 11: 実習 3 における ADD　ACC　03

```
CPU0,PC=0x5> d
    acc=0x04(4,4) ix=0x03(3,3) cf=0 vf=0 nf=0 zf=0
    ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
CPU0,PC=0x5> i
Program Halted.
CPU0,PC=0x7> d
    acc=0x04(4,4) ix=0x02(2,2) cf=0 vf=0 nf=0 zf=0
    ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
```

図 12: 実習 3 における SUB IX 1

```
CPU0,PC=0x7> d
    acc=0x04(4,4) ix=0x02(2,2) cf=0 vf=0 nf=0 zf=0
    ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
CPU0,PC=0x7> i
Program Halted.
CPU0,PC=0x3>
```

図 13: 実習 3 における BNZ LOOP

```
CPU0,PC=0x7> d
    acc=0x0c(12,12) ix=0x00(0,0) cf=0 vf=0 nf=0 zf=1
    ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
CPU0,PC=0x7> i
Program Halted.
CPU0,PC=0x9>
```

図 14: 実習 3 における BNZ LOOP

```
CPU0,PC=0x9> i
```

図 15: 実習 3 における HLT

5 参考文献

参考文献

[1] コンピュータアーキテクチャの基礎, 柴山潔 著, 2.2 基本命令セットアーキテクチャ, p43

[2]　コンピュータアーキテクチャの基礎, 柴山潔 著, 6.1 固定小数点数の算術演算装置, p164
[3]　コンピュータアーキテクチャの基礎, 柴山潔 著, 2.2 基本命令セットアーキテクチャ, p55

6　ソースコード

ソースコード 1: cpuboard.c

```
1  /*
2   * Project-based Learning II (CPU)
3   *
4   * Program: instruction set simulator of the Educational CPU Board
5   * File Name: cpuboard.c
6   * Descrioption: simulation(emulation) of an instruction
7   */
8
9  #include "cpuboard.h"
10 #include <stdio.h>
11 #include <stdlib.h>
12
13 /*=============================================================================
14  * Simulation of a Single Instruction
15  *============================================================================*/
16
17 int step(Cpub *cpub) {
18         Uword *opa, *opb;
19
20 //Fetch Instruction
21         cpub->mar = cpub->pc++;
22         cpub->ir = cpub->mem[cpub->mar];
23
24 //Fetch Opeland
25         opa = GetaFun(cpub->ir, cpub);
26         if (cpub->ir >= 0x60 && cpub->ir <= 0xFF) {
27                 opb = GetbFun(cpub->ir, cpub);
28         }
29
30 //Execute Instruction
31         switch (cpub->ir) {
32 /*=============================================================================
33 * Control command (00H ... 2FH)
34 *============================================================================*/
35         case 0x00: //NOP
36                 break;
37         case 0x0F: //HLT
38                 exit(0);
39                 break;
40         case 0x10: //OUT
```

```
41                  cpub->obuf.buf = cpub->acc;
42                  break;
43          case 0x1F: //IN
44                  cpub->acc = cpub->ibuf->buf;
45                  break;
46          case 0x20: //RCF
47                  cpub->cf = 0;
48                  break;
49          case 0x2F: //SCF
50                  cpub->cf = 1;
51                  break;
52  /*=============================================================================
53  * Branch command (30H ... 3FH)
54  *============================================================================*/
55          case 0x30: //BA
56                  cpub->pc = cpub->mem[cpub->pc];
57                  break;
58          case 0x31:  //BNZ
59                  BbcFlag(cpub->zf, cpub, 0);
60                  break;
61          case 0x32: //BZP
62                  BbcFlag(cpub->nf, cpub, 0);
63                  break;
64          case 0x33: //BP
65                  BbcFlag(cpub->nf, cpub, 1);
66                  break;
67          case 0x34: //BNI
68                  BbcFlag(cpub->ibuf->flag, cpub, 0);
69                  break;
70          case 0x35: //BNC
71                  BbcFlag(cpub->cf, cpub, 0);
72                  break;
73          case 0x36: //BGE
74                  BbcFlag(cpub->vf ^ cpub->nf, cpub, 0);
75                  break;
76          case 0x37: //BGT
77                  BbcFlag((cpub->vf ^ cpub->nf) | cpub->zf, cpub, 0);
78                  break;
79          case 0x38: //BVF
80                  BbcFlag(cpub->vf, cpub, 1);
81                  break;
82          case 0x39: //BZ
83                  BbcFlag(cpub->zf, cpub, 1);
84                  break;
85          case 0x3A:  //BN
86                  BbcFlag(cpub->nf, cpub, 1);
87                  break;
88          case 0x3B:  //BZN
89                  BbcFlag(cpub->nf, cpub, 0);
```

```
90                  break;
91          case 0x3C: //BNO
92                  BbcFlag(cpub->obuf.flag, cpub, 1);
93                  break;
94          case 0x3D: //BC
95                  BbcFlag(cpub->cf, cpub, 1);
96                  break;
97          case 0x3E: //BLT
98                  BbcFlag(cpub->vf ^ cpub->nf, cpub, 1);
99                  break;
100         case 0x3F: //BLE
101                 BbcFlag(cpub->vf ^ cpub->nf, cpub, 1);
102                 break;
103 /*============================================================================
104 * Shift command (40H ... 4FH)
105 *============================================================================*/
106         case 0x40: //SRA
107         case 0x48:
108                 SRA(cpub, opa);
109                 break;
110         case 0x41: //SLA
111         case 0x49:
112                 SLA(cpub, opa);
113                 break;
114         case 0x42: //SRL
115         case 0x4A:
116                 SRL(cpub, opa);
117                 break;
118         case 0x43: //SLL
119         case 0x4B:
120                 SLL(cpub, opa);
121                 break;
122         case 0x44: //RRA
123         case 0x4C:
124                 RRA(cpub, opa);
125                 break;
126         case 0x45: //RLA
127         case 0x4D:
128                 RLA(cpub, opa);
129                 break;
130         case 0x46: //RRL
131         case 0x4E:
132                 RRL(cpub, opa);
133                 break;
134         case 0x47: //RLL
135         case 0x4F:
136                 RLL(cpub, opa);
137                 break;
138 /*============================================================================
```

```
139 * Data movement/ Arithmetic/ Logical command (60H ... FFH)
140 *================================================================================*/
141         case 0x60 ... 0x6F: //LD
142                 LD(opa, opb);
143                 break;
144         case 0x74 ... 0x7F: //ST
145                 ST(opa, opb);
146                 break;
147         case 0x80 ... 0x8F: //SBC
148                 SBC(opa, opb, cpub);
149                 break;
150         case 0x90 ... 0x9F: //ADC
151                 ADC(opa, opb, cpub);
152                 break;
153         case 0xA0 ... 0xAF: //SUB
154                 SUB(opa, opb, cpub);
155                 break;
156         case 0xB0 ... 0xBF: //ADD
157                 ADD(opa, opb, cpub);
158                 break;
159         case 0xC0 ... 0xCF: //EOR
160                 EOR(opa, opb, cpub);
161                 break;
162         case 0xD0 ... 0xDF: //OR
163                 OR(opa, opb, cpub);
164                 break;
165         case 0xE0 ... 0xEF: //AND
166                 AND(opa, opb, cpub);
167                 break;
168         case 0xF0 ... 0xFF: //CMP
169                 CMP(opa, opb, cpub);
170                 break;
171         default:
172                 printf("␣%x␣:␣no␣instruction␣code", cpub->ir);
173                 cpub->pc++;
174                 break;
175         }
176         return RUN_HALT;
177 }
178
179 /*================================================================================
180  * Function definition
181  *================================================================================*/
182
183 Uword* GetaFun(int num, Cpub *cpub) { // Decode opa by addressing mode
184         switch (num % 16) {
185         case 0 ... 7:
186                 return &cpub->acc;
187                 break;
```

```
188         case 8 ... 0xF:
189                 return &cpub->ix;
190                 break;
191         default:
192                 return NULL;
193                 break;
194         }
195 }
196
197 Uword* GetbFun(int num, Cpub *cpub) { // Decode opb by addressing mode
198         switch (num % 8) {
199         case 0:
200                 return &cpub->acc;
201                 break;
202         case 1:
203                 return &cpub->ir;
204                 break;
205         case 2:
206                 return &cpub->mem[cpub->pc++];
207                 break;
208         case 4:
209                 return &cpub->mem[cpub->mem[cpub->pc++]];
210                 break;
211         case 5:
212                 return &cpub->mem[cpub->mem[cpub->pc++] + IMEMORY_SIZE];
213                 break;
214         case 6:
215                 return &cpub->mem[cpub->mem[cpub->pc++] + cpub->ix];
216                 break;
217         case 7:
218                 return &cpub->mem[cpub->mem[cpub->pc++] + IMEMORY_SIZE + cpub->ix];
219                 break;
220         default:
221                 return NULL;
222                 break;
223         }
224 }
225 void CF(Uword *a, Uword *b, Bit *cf) {
226         Uword result = *a + *b;
227         *cf = *a < result ? 0 : 1;
228 }
229 void VF(Uword *a, Uword *b, Bit *vf) {
230         Uword result = *a + *b;
231         if ((result >= 0) && (*a < 0) && (*b < 0)) {
232                 *vf = 1;
233         } else if ((result <= 0) && (*a > 0) && (*b > 0)) {
234                 *vf = 1;
235         } else {
236                 *vf = 0;
```

```
237         }
238 }
239 void NF(Uword result, Bit *nf) {
240         *nf = ((result | 0xBF) != 0xFF) ? 0 : 1;
241 }
242 void ZF(Uword result, Bit *zf) {
243         *zf = (result == 0) ? 1 : 0;
244 }
245
246 void CVFlagFun(Uword *a, Uword *b, Cpub *cpub) {
247         Uword *vf = &cpub->vf;
248         CF(a, b, &cpub->cf);
249         *vf = cpub->cf;
250 }
251
252 void NZFlagFun(Uword result, Cpub *cpub) {
253         NF(result, &cpub->nf);
254         ZF(result, &cpub->zf);
255 }
256
257 /*void BNZ(Uword *zf, Uword *pc, Uword *b) {
258  (*pc) = ((*zf) != 0) ? (*b) : (*pc) + 1;
259  }
260  void FlagCondtion(Uword *f, Uword *pc, Uword *b, int condtion) {
261  (*pc) = ((*f) == condtion) ? (*pc) + 1: (*b);
262  }*/
263
264 void BbcFlag(Bit f, Cpub *cpub, int condtion) {
265         Uword *pc = &cpub->pc;
266         Uword b = cpub->mem[cpub->pc];
267         *pc = f == condtion ? b : *pc + 1;
268 }
269 void SRA(Cpub *cpub, Uword *a) {
270         Bit msb, lsb;
271         Uword *cf = &cpub->cf;
272         Uword *vf = &cpub->vf;
273         *vf = 0;
274         MsbLsbFun(a, &msb, &lsb);
275         *cf = lsb;
276         *a = *a >> 1;
277         *a = (*cf == 1) ? *a | 0x80 : *a | 0x00;
278         NZFlagFun(*a,cpub);
279 }
280 void SLA(Cpub *cpub, Uword *a) {
281         Bit msb, lsb;
282         Uword *cf = &cpub->cf;
283         Uword *vf = &cpub->vf;
284         MsbLsbFun(a, &msb, &lsb);
285         *cf = msb;
```

```
286         *vf = msb;
287         *a = *a << 1;
288         NZFlagFun(*a,cpub);
289 }
290 void SRL(Cpub *cpub, Uword *a) {
291         Bit msb, lsb;
292         Uword *cf = &cpub->cf;
293         Uword *vf = &cpub->vf;
294         *vf = 0;
295         MsbLsbFun(a, &msb, &lsb);
296         *cf = lsb;
297         *a = *a >> 1;
298         NZFlagFun(*a,cpub);
299 }
300 void SLL(Cpub *cpub, Uword *a) {
301         Bit msb, lsb;
302         Uword *cf = &cpub->cf;
303         Uword *vf = &cpub->vf;
304         *vf = 0;
305         MsbLsbFun(a, &msb, &lsb);
306         *cf = msb;
307         *a = *a << 1;
308         NZFlagFun(*a,cpub);
309 }
310 void RRA(Cpub *cpub, Uword *a) {
311         Bit msb, lsb;
312         Uword *cf = &cpub->cf;
313         Uword *vf = &cpub->vf;
314         *vf = 0;
315         MsbLsbFun(a, &msb, &lsb);
316         *cf = lsb;
317         *a = *a >> 1;
318         *a = (*cf == 1) ? *a | 0x80 : *a | 0x00;
319         NZFlagFun(*a,cpub);
320 }
321 void RLA(Cpub *cpub, Uword *a) {
322         Bit msb, lsb;
323         Uword *cf = &cpub->cf;
324         Uword *vf = &cpub->vf;
325         MsbLsbFun(a, &msb, &lsb);
326         *cf = msb;
327         *vf = msb;
328         *a = *a << 1;
329         *a = (*cf == 1) ? *a | 0x01 : *a | 0x00;
330         NZFlagFun(*a,cpub);
331 }
332 void RRL(Cpub *cpub, Uword *a) {
333         Bit msb, lsb;
334         Uword *cf = &cpub->cf;
```

```
335         Uword *vf = &cpub->vf;
336         *vf = 0;
337         MsbLsbFun(a, &msb, &lsb);
338         *cf = lsb;
339         *a = *a >> 1;
340         *a = (*cf == 1) ? *a | 0x80 : *a | 0x00;
341         NZFlagFun(*a,cpub);
342 }
343 void RLL(Cpub *cpub, Uword *a) {
344         Bit msb, lsb;
345         Uword *cf = &cpub->cf;
346         Uword *vf = &cpub->vf;
347         *vf = 0;
348         MsbLsbFun(a, &msb, &lsb);
349         *cf = msb;
350         *a = *a << 1;
351         *a = (*cf == 1) ? *a | 0x01 : *a | 0x00;
352         NZFlagFun(*a,cpub);
353 }
354 void MsbLsbFun(Uword *a, Bit *msb, Bit *lsb) {
355         *msb = ((*a & 0x80) == 0x80) ? 1 : 0;
356         *lsb = ((*a & 0x01) == 0x01) ? 1 : 0;
357 }
358 void LD(Uword *a, Uword *b) {
359         *a = *b;
360 }
361 void ST(Uword *a, Uword *b) {
362         *b = *a;
363 }
364 void EOR(Uword *a, Uword *b, Cpub *cpub) {
365         Uword *vf = &cpub->vf;
366         *vf = 0;
367         *a = *a ^ *b;
368         NZFlagFun(*a, cpub);
369 }
370 void ADD(Uword *a, Uword *b, Cpub *cpub) {
371         VF(a,b,&cpub->vf);
372         *a += *b;
373         NZFlagFun(*a, cpub);
374 }
375 void ADC(Uword *a, Uword *b, Cpub *cpub) {
376         Bit addcf = cpub->cf;
377         CVFlagFun(a,b,cpub);
378         *a = *a + *b + addcf;
379         NZFlagFun(*a,cpub);
380 }
381 void SUB(Uword *a, Uword *b, Cpub *cpub) {
382         VF(a,b,&cpub->vf);
383         *a -= *b;
```

```
384         NZFlagFun(*a, cpub);
385 }
386 void SBC(Uword *a, Uword *b, Cpub *cpub) {
387         Bit addcf = cpub->cf;
388         CVFlagFun(a,b,cpub);
389         *a = *a - *b - addcf;
390         NZFlagFun(*a,cpub);
391 }
392 void OR(Uword *a, Uword *b, Cpub *cpub) {
393         Uword *vf = &cpub->vf;
394         *vf = 0;
395         *a = (*a | *b);
396         NZFlagFun(*a, cpub);
397 }
398 void AND(Uword *a, Uword *b, Cpub *cpub) {
399         Uword *vf = &cpub->vf;
400         *vf = 0;
401         *a = (*a & *b);
402         NZFlagFun(*a, cpub);
403 }
404 void CMP(Uword *a, Uword *b, Cpub *cpub) {
405         VF(a,b,&cpub->vf);
406         Uword result = *a - *b;
407         NZFlagFun(result, cpub);
408 }
```

ソースコード 2: cpuboard.h

```
/*
 * Project-based Learning II (CPU)
 *
 * Program: instruction set simulator of the Educational CPU Board
 * File Name: cpuboard.h
 * Descrioption: resource definition of the educational computer board
 */

/*=============================================================================
 * Architectural Data Types
 *============================================================================*/
typedef signed char Sword;
typedef unsigned char Uword;
typedef unsigned short Addr;
typedef unsigned char Bit;

/*=============================================================================
 * CPU Board Resources
 *============================================================================*/
#define MEMORY_SIZE 256*2
#define IMEMORY_SIZE 256
#define UWORD_SIZE 255

typedef struct iobuf {
        Bit flag;
        Uword buf;
} IOBuf;

typedef struct cpuboard {
        Uword pc;
        Uword mar;
        Uword ir;
        Uword acc;
        Uword ix;
        Bit cf, vf, nf, zf;
        IOBuf *ibuf;
        IOBuf obuf;
        /*
         * [ add here the other CPU resources if necessary ]
         */
        Uword mem[MEMORY_SIZE]; /* 0XX:Program, 1XX:Data */
} Cpub;

/*=============================================================================
 * Top Function of an Instruction Simulation
 *============================================================================*/
#define RUN_HALT 0
```

```
48 #define RUN_STEP 1
49 int step(Cpub *);
50
51 //Get Opeland A or B
52 Uword *GetaFun(int num, Cpub *cpub);
53 Uword *GetbFun(int num, Cpub *cpub);
54
55 //Set Each Flagment
56 void NF(Uword resultValue, Bit *nf);
57 void ZF(Uword resultValue,Bit *zf);
58 void CF(Uword *a, Uword *b, Bit *cf);
59 void VF(Uword *a, Uword *b, Bit *of);
60 void CVFlagFun(Uword *a, Uword *b, Cpub *cpub);
61 void NZFlagFun(Uword result, Cpub *cpub);
62
63 //Execute Each Shift command
64 void SRA(Cpub *cpub,Uword *a);
65 void SLA(Cpub *cpub,Uword *a);
66 void SRL(Cpub *cpub,Uword *a);
67 void SLL(Cpub *cpub,Uword *a);
68 void RRA(Cpub *cpub,Uword *a);
69 void RLA(Cpub *cpub,Uword *a);
70 void RRL(Cpub *cpub,Uword *a);
71 void RLL(Cpub *cpub,Uword *a);
72 void SLL(Cpub *cpub,Uword *a);
73 void MsbLsbFun(Uword *a,Bit *msb,Bit *lsb);
74
75 //Execute Branch Condtion of Flag
76 void BbcFlag(Bit f,Cpub *cpub,int condtion);
77
78 //Execute DataMove, Athmetic or Logical command
79 void LD(Uword *a, Uword *b);
80 void ST(Uword *a, Uword *b);
81 void ADD(Uword *a, Uword *b, Cpub *cpub);
82 void ADC(Uword *a, Uword *b, Cpub *cpub);
83 void SUB(Uword *a, Uword *b, Cpub *cpub);
84 void SBC(Uword *a, Uword *b, Cpub *cpub);
85 void CMP(Uword *a, Uword *b, Cpub *cpub);
86 void AND(Uword *a, Uword *b, Cpub *cpub);
87 void OR(Uword *a, Uword *b, Cpub *cpub);
88 void EOR(Uword *a, Uword *b, Cpub *cpub);
```

ソースコード 3: cpuboard.h

```
CPU0,PC=0x0> r prog.txt
CPU0,PC=0x0> s acc 4
        acc=0x04(4,4) ix=0x00(0,0) cf=0 vf=0 nf=0 zf=0
        ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
CPU0,PC=0x0> s ix 3
        acc=0x04(4,4) ix=0x03(3,3) cf=0 vf=0 nf=0 zf=0
        ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
CPU0,PC=0x0> i
Program Halted.
CPU0,PC=0x2> m
    | 000: 75 03 c0 b5 03 aa 01 31 | 008: 03 0f 00 00 00 00 00 00
    | 010: 00 00 00 00 00 00 00 00 | 018: 00 00 00 00 00 00 00 00

    | 0f0: 00 00 00 00 00 00 00 00 | 0f8: 00 00 00 00 00 00 00 00
    | 100: 00 00 00 04 00 00 00 00 | 108: 00 00 00 00 00 00 00 00
    | 110: 00 00 00 00 00 00 00 00 | 118: 00 00 00 00 00 00 00 00

CPU0,PC=0x2> i
Program Halted.
CPU0,PC=0x3> d
        acc=0x00(0,0) ix=0x03(3,3) cf=0 vf=0 nf=0 zf=1
        ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
CPU0,PC=0x3> i
Program Halted.
CPU0,PC=0x5> d
        acc=0x04(4,4) ix=0x03(3,3) cf=0 vf=0 nf=0 zf=0
        ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
CPU0,PC=0x5> i
Program Halted.
CPU0,PC=0x7> d
        acc=0x04(4,4) ix=0x02(2,2) cf=0 vf=0 nf=0 zf=0
        ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
CPU0,PC=0x7> i
Program Halted.
CPU0,PC=0x3> i
Program Halted.
CPU0,PC=0x5> i
Program Halted.
CPU0,PC=0x7> i
Program Halted.
CPU0,PC=0x3> i
Program Halted.
CPU0,PC=0x5> i
Program Halted.
CPU0,PC=0x7> d
        acc=0x0c(12,12) ix=0x00(0,0) cf=0 vf=0 nf=0 zf=1
        ibuf=0:0x00(0,0) obuf=0:0x00(0,0)
```

```
48 CPU0,PC=0x7> i
49 Program Halted.
50 CPU0,PC=0x9> i
```

ソースコード 4: cpuboard.h

```
1 75 03
2 C0
3 B5 03
4 AA 01
5 31 03
6 0F
```