情報実験第三 課題 2

15_03602 柿沼 建太郎
情報工学科 15_10588 中田 光

2017 年 7 月 28 日

作成したプログラムの概要

今回の課題では、課題 1 で作成された電卓プログラムの改造を行った。Verilog コードの改変と EX3 のコードの改変をともに行った。追加・変更する予定であった機能は以下の通り。

Verilog コードの追加/変更

- CPU の 32bit 化
- 命令数の追加
- 固定小数を実装
- メモリ参照命令を追加 (減算, 乗算, 除算, 剰余)

EX3 コードの追加/変更

- 小数の入力受付
- 表示の 10 進化
- 乗算, 除算, 剰余, 平方根などの新たな演算の実装

しかしこの内「減算・剰余命令の追加」と「固定小数の実装」及びそのソフトウェア側での対応は時間が足りず間に合わなかった。

各作業の担当者

Verilog コードの追加/改変

担当者 柿沼

CPU の 32bit 化

与えられた Verilog コードで実装された CPU は 16bitCPU であったが、固定小数による計算をするにあたって 16bit では精度が足りないと判断し、AC レジスタなどの容量を 32bit まで拡張した。具体的には以下

作業名	担当者
CPU の 32bit 化	柿沼
命令数の追加	柿沼
メモリ参照命令の追加	柿沼
小数の入力受付	中田
表示の 10 進化	中田
乗算, 除算, 剰余, 平方根などの新たな演算の実装	中田

のように改変した。

- ALU の 32bit 計算対応
- RAM と ROM の 32bit 化
- DR,AC レジスタの容量の 32bit 化
- バス容量の 32bit 化

前回の課題で AC レジスタの中身を 7 セグメント表示用レジスタに転送する命令を作成したが、7 セグメントディスプレイには 16bit ぶんしか表示できなかったため、下位 16bit のみしか表示されていない。LED での表示についても同様の対処がされている。また、与えられた EX3 シミュレータは出力するファイルが 16bit のデータとなっていたため、32bit 出力するように対応した。

命令数の追加とメモリ参照命令の追加

掛け算命令や割り算命令などを追加したほうがソフトウェア側での高機能な計算が楽になると思い、Verilog コード側で実装することにした。具体的にはメモリ参照命令で掛け算と割り算を行う「MUL」と「DIV」を実装した。掛け算や割り算命令はメモリ参照命令でありオペコードに空きがなかったため、オペコードを 16bit から 17bit に変更した。オペコードの割り当てについては変更を最小限にするために IR レジスタの 16,14,13,12bit 目の計 4bit を見て命令種別を判別することとした。そのため掛け算命令「MUL」のオペコードは 10XXX と 18XXX、割り算命令「DIV」のオペコードは 11XXX と 19XXX となっている。

感想

柿沼

CPU の 32bit 化の作業中にバグが大量に発生し、ほとんどその対応に時間を使ってしまった。数字を変えるだけだと侮っていたと思う。結局何度か修正していたら直ったが、原因はよくわからなかった。命令の拡張などはすぐにできたので、バグが大量発生した時点で早めに打ち切り、固定小数の実装に移っておくべきだったと今では思う。固定小数の実装が間に合わなかったため根号計算などが作れなかったのが痛手だと感じる。また、Verilog シミュレータをうまく使いこなせなかったのも時間のロスの原因だと思う。入出力の部分が見にくかったので結局ほとんどのデバッグを FPGA でやっていたのだが、ビルドに時間が掛かる上に授業中しかデバッグができない。Verilog シミュレータを使いこなせばもう少し効率的に作業ができたのではないかと思う。

test_calc1 の機能拡張：ソフトウェアプログラム

担当:中田

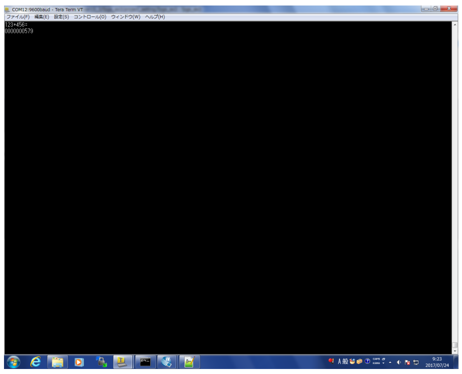
概要

計算機プログラムのアセンブリコード test_calc1.asm を改良し、入出力の 10 進数化、計算機能の拡張を行った。計画当初では固定小数を実装する予定であったが、ハードウェアとソフトウェアの双方で完成の目途が立たず、残された日数を考慮し、今回は実装を断念した。それに伴い実装予定だった平方根の機能拡張についても整数値しか扱うことができず、ニュートン法を用いた方法では精度が期待できなかったため他の計算機能を拡張した。以下に拡張した機能と主なラベルを記す。また、入力は全て中置記法で入力する。

機能	機能の説明
乗算	乗算を計算する。演算子は’*’。
除算	除算を計算する。剰余は切り捨て。演算子は’/’。
指数計算	指数計算をする。演算子は’^’。
組み合わせ	組み合わせを計算する。演算子は’c’。
総乗	P_Y から P_X までの積を計算する。$xp1 =$ とすると x の階乗の計算となる。演算子は’p’。

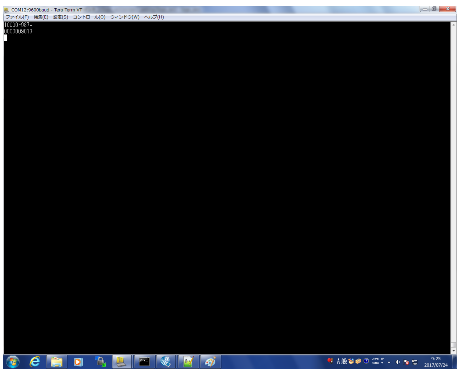
ラベル	ラベルの説明
READ_DX	入力が 0〜9 かを判断する。
H2D	Z の 16 進の値を出力用のレジスタ ZHMG に 10 進数で格納する。
P	総乗を計算をする。$\prod_{i=P\_Y}^{P\_X} i$ を P_RES に格納する。
C	組み合わせを計算をする。${}_{P\_X}C_{P\_Y}$ を P_RES に格納する。
EXP	指数を計算する。$EXP\_X^{EXP\_Y}$ を EXP_RES に格納する。

実行結果


COM12:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

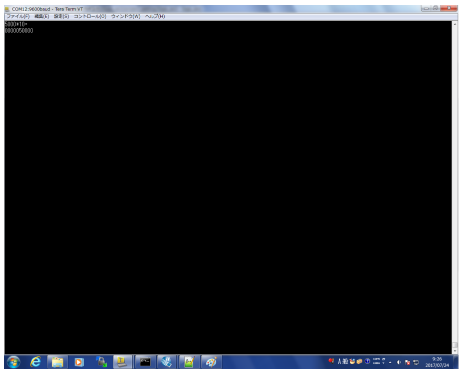
```
123+456=
0000000579
```

図 1　”123 + 456 =”


COM12:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

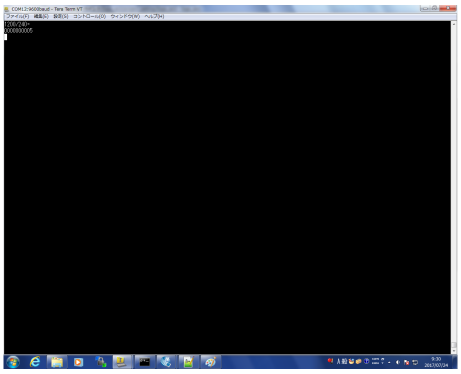
```
10000-987=
0000009013
```

図 2　”10000 − 987 =”


COM12:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

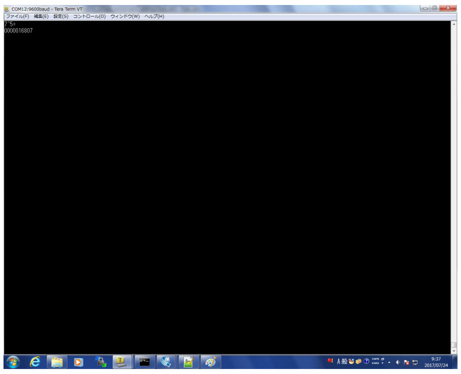
```
5000*10=
0000050000
```

図 3　”5000 × 10 =”


COM12:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

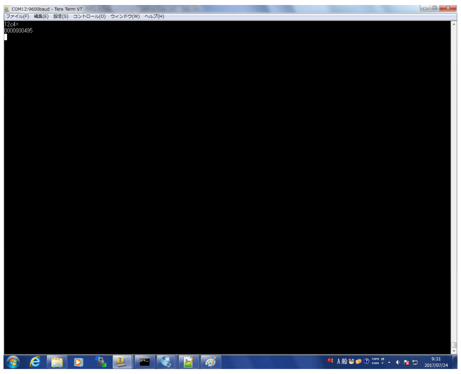
```
1200/240=
0000000005
```

図 4　”1200 ÷ 240 =”


COM12:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

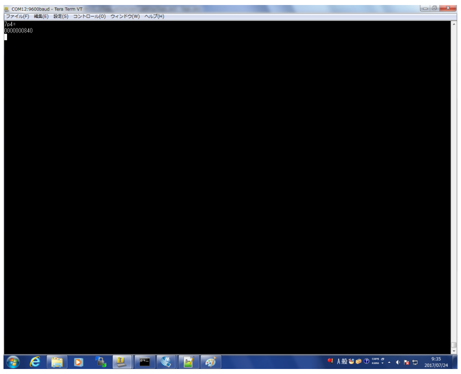
```
7^5=
0000016807
```

図 5　"$7^5 =$"


COM12:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

```
12c4=
0000000495
```

図 6　"${}_{12}C_4 =$"


COM12:9600baud - Tera Term VT
ファイル(F) 編集(E) 設定(S) コントロール(O) ウィンドウ(W) ヘルプ(H)

```
7p4=
0000000840
```

図 7　"$\prod_{i=4}^{7} i =$"

各サブルーチンの計算方法について

指数

まず、サブルーチンの呼び出し前に X の値を EXP_X、Y の値を EXP_Y に代入しておく。サブルーチン内では MUL 命令によって EXP_X の値を EXP_Z にかけるという処理を EXP_Y 回行い、その結果を EXP_RES に格納する。かける回数は EXP_A をインクリメントすることでカウントし、EXP_Y との差が 0 になったらループを抜ける。計算終了後、EXP_A,EXP_Z を 0,1 に初期化して、処理を終了する。

総乗

同様に、呼び出し前に X,Y の値を P_X,P_Y に代入しておく。サブルーチン内では MUL 命令によって P_Y の値を P_Z にかけていき、かけるごとに P_Y の値をインクリメントしている。P_Y の値が P_X に等しくな

るとループを抜ける。計算終了後、P_Z を 1 に初期化して処理を終了する。

組み合わせ

同様に、呼び出し前に X,Y の値を C_X,C_Y に代入しておく。サブルーチン内では総乗を計算するサブルーチン P を呼び出している。P_A には $\prod_{i=P\_X-P\_Y+1}^{P\_X} i$ を代入し、P_B には $P\_Y!$ を代入し、$P\_A \div P\_B$ を P_RES に格納する。初期化する必要のある変数はないため、計算終了後、処理を終了する。

工夫点

各計算を処理するサブルーチンの実装において、グローバルに存在するレジスタを利用せず、それぞれのサブルーチンに入力、出力レジスタを用意し、モジュール化を行った。サブルーチンの外部とのデータのやり取りは入出力レジスタのみで行い、内部の計算に使用するレジスタはそのサブルーチン内でのみ利用される。それによって、サブルーチン内で利用するレジスタの初期化をサブルーチン内で行い、再利用可能にすることで、計算を複数回行う場合や他のサブルーチン内で呼び出しを行えるようにした。以前作成したプログラムでもサブルーチンをモジュール化することで、容易に再利用可能なサブルーチンを作るべきだったと感じた。

感想

最初に 10 進数の入出力を実装する段階で大きく時間を割いてしまい、後半時間が足りなくなってしまった。前回作成した素数プログラムにおいて 10 進数の出力を実装していたので、それを利用すれば簡単に実装できると思ったが、実際は 32 ビットに対応させたり、整合をとるためにレジスタを変更したりと、計画以上に時間がかかってしまった。計算機能の追加については、ハードウェア側で乗算と除算を実装してもらえたため、コードを簡潔に書くことができ、実装しやすかった。また、今回はハードウェアとソフトウェアで担当を分けて行ったため、ハードウェア側の仕様が変わるごとにバグが発生してしまい、デバッグが大変だった。今回は二人だったが、人数が増えてプロジェクトが大きくなった場合、今回の計画では対応できないと感じた。

EX3 シミュレータのコード

ソースコード 1 common_asm.h

```
#if !defined _COMMON_ASM_H_INCLUDED_

#define _COMMON_ASM_H_INCLUDED_

#ifdef WIN32
#include <conio.h>
#else   /// LINUX
#include <termios.h>
#endif

#define COMMENT '/'

```

```
#define ABORT_PROGRAM(msg)      msg; printf("hit return : "); getchar(); exit(-1)

#if !defined(WIN32)
class GetChSetting
{
public:
        termios oldTermios, newTermios;
        GetChSetting(){
                tcgetattr(0, &oldTermios);
                newTermios = oldTermios;
                newTermios.c_lflag &= ~ICANON;  /// disable buffered i/o
                tcsetattr(0, TCSANOW, &newTermios);
        }
        ~GetChSetting(){ tcsetattr(0, TCSANOW, &oldTermios);}
};
extern int _getche(void);
#endif

class Label
{
public:
        class Element
        {
        public:
                char * name;
                int nlen;
                unsigned int address;
                Element(){ name = 0; nlen = 0; address = 0;}
                void Set(const char * n, int len, unsigned int addr){
                        if(name){ delete [] name;}
                        nlen = len;
                        name = new char[nlen + 1];
                        strncpy(name, n, nlen);
                        name[nlen] = 0;
                        address = addr;
                }
                ~Element(){ if(name) delete [] name;}
                void PrintInfo(FILE * fp, int maxlen){ if(fp) fprintf(fp, "label(%*s) :
                     %5d(0x%04x)\n", maxlen, name, address, address);}
        };
        enum AnnotationLabel{
                AL_Null       = 0x0,
                AL_Breakpoint = 0x1,
                AL_Monitor    = 0x2,
        };
        class AnnotationStatus
        {
        public:
                int curAddr, annotation;
                AnnotationStatus(){ curAddr = 0; annotation = AL_Null;}
                AnnotationLabel CheckAnnotationLabel(const char * p, int len){
                        if(len != 3 || p[0] != '_' || p[2] != '_'){ return AL_Null;}
                        switch(p[1]){
```

```
                        case 'B': return AL_Breakpoint;
                        case 'M': return AL_Monitor;
                        default : return AL_Null;
                        }
                }
                bool AddAnnotation(const char * p, int len, unsigned int addr){
                        if(curAddr != addr){ annotation = AL_Null; curAddr = addr;}
                        AnnotationLabel al = CheckAnnotationLabel(p, len);
                        annotation |= al;
                        return (al != AL_Null);
                }
        } annotation;
#define MAX_LABEL_COUNT 1000
        Element element[MAX_LABEL_COUNT];
        int count, maxLabelLength;
        Label() : count(0), maxLabelLength(0){}
        Element * AddLabel(const char * n, int len, unsigned int addr){
                Element * lb;
                if(count >= MAX_LABEL_COUNT)    { printf("ERROR: labelCount exceeds
                    limit %d\n", MAX_LABEL_COUNT); return 0;}
                if((lb = GetLabel(n, len)))             { printf("ERROR: label(%s)
                    already exists!!\n", lb->name); return 0;}
                lb = &element[count ++];
                lb->Set(n, len, addr);
                if(maxLabelLength < len){ maxLabelLength = len;}
                return lb;
        }
        Element * GetLabel(const char * p, int len){
                int i;
                for(i = 0; i < count; i ++){ if(element[i].nlen == len && strncmp(
                    element[i].name, p, len) == 0) return &element[i];}
                return 0;
        }
        Element * GetLabel(int addr){
                int i;
                for(i = 0; i < count; i ++){ if(element[i].address == addr) return &
                    element[i];}
                return 0;
        }
        void PrintLabels(FILE * fp){
                if(fp){
                        int i;
                        for(i = 0; i < count; i ++){ element[i].PrintInfo(fp,
                            maxLabelLength);}
                }
        }
};

class CPU;
class Memory
{
public:
        enum Status
```

```
{        ///      bit[0] : 1 (used as instruction or data), 0 (unused)
         MS_Null                          = 0x0,  ///      00
         MS_Used                          = 0x1,  ///      01
         MS_Monitor           = 0x2,      ///      for ISS and Verilog simulations
         MS_Breakpoint           = 0x4,   ///      for ISS (breakpoint set)
};
class Word
{
public:
         unsigned char status;
         unsigned char insnID;
         unsigned int value;
         char * headComment, * tailComment;
         Word(){ status = MS_Null; insnID = 0; value = 0; headComment = 0;
             tailComment = 0;}
         ~Word(){
                 if(headComment){ delete [] headComment;}
                 if(tailComment){ delete [] tailComment;}
         }
         void SetStatus(Label::AnnotationStatus * labelAnnotation, int addr){
                 status = MS_Used;
                 if(labelAnnotation->curAddr == addr){
                         if(labelAnnotation->annotation & Label::AL_Monitor)
                                          { status |= MS_Monitor;}
                         if(labelAnnotation->annotation & Label::AL_Breakpoint)
                               { status |= MS_Breakpoint;}
                 }
         }
         void SetComment(const char * c, int len, int headFlag){
                 if(headFlag){
                         int len0 = (headComment) ? strlen(headComment) + 2: 0;
                         int len2 = len0 + len;
                         char * comment2 = new char[len2 + 1];
                         char * p = comment2;
                         if(headComment){ sprintf(p, "%s\n/", headComment); p +=
                              strlen(headComment) + 2;}
                         strncpy(p, c, len);
                         comment2[len2] = 0;
                         if(headComment){ delete [] headComment;}
                         headComment = comment2;
                 }
                 else if(tailComment == 0){
                         char * comment2 = new char[len + 1];
                         char * p = comment2;
                         strncpy(p, c, len);
                         comment2[len] = 0;
                         tailComment = comment2;
                 }
         }
};
Word * word, * curCode, bogusWord;      ///      bogusWord : for returning
    memory word upon access error
int size, maxAddr;
```

```
        unsigned int errorFlag;
        Memory(int sz){
                size = sz;
                maxAddr = 0;
                errorFlag = 0;
                curCode = 0;
                word = new Word[size];
        }
        ~Memory(){ delete [] word;}
        bool IsValidAddress(int addr){ return addr >= 0 && addr < size;}
        Word * GetWord(unsigned int addr){
                if(!IsValidAddress(addr) || errorFlag){ errorFlag = 1; return &
                    bogusWord;}
                word[addr].status |= MS_Used;
                return &word[addr];
        }
        void ProbeInsn(unsigned int pc){ curCode = GetWord(pc);}
        void FetchInsn(unsigned int & pc){ curCode = GetWord(pc); pc ++;}
        unsigned int Address(){
                Word * m = (IsIndirectMemoryAccess(curCode)) ? GetWord(GetAddressField(
                    curCode)) : curCode;
                return GetAddressField(m);
        }
        unsigned int & Operand(){ return GetWord(Address())->value;}

        ///     virtual abstract functions : must be defined in actual class
        virtual bool IsIndirectMemoryAccess(Word * w)                    = 0;
        virtual unsigned int GetAddressField(Word * m)          = 0;
};

class InsnSet
{
public:
        class Insn
        {
        public:
                unsigned char ID;
                unsigned char type;
                unsigned char nlen;
                unsigned char showMemFlag;
                const char * name;
                unsigned int code;
                void (* operation)(CPU *);
                Insn(){ ID = 0; type = 0; nlen = 0; showMemFlag = 0; name = 0;
                    operation = OP_DUMMY;}
                void Set(int id, const char * n, int t, int showMem, void (* op)(CPU
                    *), unsigned int c){
                        ID = id; name = n; type = t; showMemFlag = showMem;
                        nlen = strlen(name);
                        operation = op;
                        code = c;
                }
        };
        Insn * SearchInsn(const char ** iname_p){
                const char * iname = *iname_p;
                int i;
                for(i = 0; i < ICount; i ++){
                        Insn * ii = &insn[i];
                        if(ii->nlen > 0 && strncmp(ii->name, iname, ii->nlen) == 0){ *
                            iname_p += ii->nlen; return ii;}
                }
                return 0;
        }
        Insn * insn;
        int ICount;
        enum InsnID
        {
                I_INVALID, /// valid InsnID is non-zero
                I_ORG, I_END, I_DEC, I_HEX, I_CHR, I_SYM,
                I_TAIL,
        };
        InsnSet(int icount){
                ICount = icount;
                insn = new Insn[ICount];
#define NI(i)   insn[I_##i].Set(I_##i, #i, 0, 0, OP_DUMMY, 0)
                NI(ORG); NI(END); NI(DEC); NI(HEX); NI(CHR); NI(SYM);
#undef  NI
        }
        ~InsnSet(){ delete [] insn;}
        static void OP_DUMMY(CPU * cpu){}
};

class Debugger;

#define ECHO    (Debugger::globalDBG->Echo())
#define FDMP    (Debugger::globalDBG->FileLog())

#define EMIT_MESSAGE_0(func, fp)
            if(FDMP) func(fp);                                                  if(
    ECHO) func(stdout)
#define EMIT_MESSAGE_1(func, fp, arg0)                                          if(FDMP
    ) func(fp, arg0);                                           if(ECHO) func(stdout, arg0
    )
#define EMIT_MESSAGE_2(func, fp, arg0, arg1)                             if(FDMP) func(
    fp, arg0, arg1);                            if(ECHO) func(stdout, arg0, arg1)
#define EMIT_MESSAGE_3(func, fp, arg0, arg1, arg2)                       if(FDMP) func(
    fp, arg0, arg1, arg2);              if(ECHO) func(stdout, arg0, arg1, arg2)
#define EMIT_MESSAGE_4(func, fp, arg0, arg1, arg2, arg3)         if(FDMP) func(fp, arg0,
     arg1, arg2, arg3);       if(ECHO) func(stdout, arg0, arg1, arg2, arg3)
#define EMIT_MESSAGE_5(func, fp, arg0, arg1, arg2, arg3, arg4)  \
        if(FDMP) func(fp, arg0, arg1, arg2, arg3, arg4);        if(ECHO) func(stdout,
            arg0, arg1, arg2, arg3, arg4)

#define PROGRAM_ENTRY_POINT     0x10
class CPU
{
```

```
public:
        Label label;
        Memory * mem;
        InsnSet * isa;
        unsigned int _S, _PC, _oldPC, _INPR, _OUTR, _FGI, _FGO;
        int cycle_count, appl_cycle_count, intr_cycle_count, intr_pending;
        FILE * fplog;
        Debugger * dbg;
        CPU(const char * logfilename) : mem(0), isa(0), dbg(0){ fplog = fopen(
            logfilename, "w"); Reset();}
        ~CPU(){
                if(mem)         { delete mem;}
                if(isa)         { delete isa;}
                if(fplog)       { fclose(fplog);}
        }
        void PrintSeparator(FILE * fp){ fprintf(fp, "----------------\n");}
        virtual void Reset(){
                cycle_count                     = 0;
                appl_cycle_count        = 0;
                intr_cycle_count        = 0;
                intr_pending            = 0;
                _S                                      = 1;
                _oldPC                          = PROGRAM_ENTRY_POINT;
                _PC                                     = PROGRAM_ENTRY_POINT;
                _INPR                           = 0;
                _OUTR                           = 0;
                _FGI                            = 0x0; _FGO
                            = 0x3;
        }
        void PrintMemory(FILE * fp, int printMode){
                int i;
                for(i = 0; i < mem->maxAddr; i ++){ PrintMemoryWord(&mem->word[i], fp,
                    i, printMode);}
        }
        bool IsInputReady(){ return _GetFGI() == 0;}
        bool IsOutputReady(){ return _GetFGO() == 0;}
        void SetInput(unsigned char val){
                if(IsInputReady())              { _SetFGI(1); _INPR = val;}
                else                                    { printf("ERROR!!! SetInput()
                    called when input is not ready...\n");}
        }
        unsigned char GetOutput(){
                if(IsOutputReady())             { _SetFGO(1); return (unsigned char)
                    _OUTR;}
                else                                    { printf("ERROR!!! GetInput()
                    called when output is not ready...\n"); return -1;}
        }
        ///     virtual abstract functions : must be defined in actual class
        virtual int GetSelectedPortID() = 0;
        virtual int Execute() = 0;
        virtual bool IsWaitingForInput() = 0;
        virtual void _SetFGI(int val) = 0;
        virtual void _SetFGO(int val) = 0;
        virtual int _GetFGI() = 0;
        virtual int _GetFGO() = 0;
        virtual void PrintStatus(FILE * fp, bool intr_cycle) = 0;
        enum PrintMemoryWordMode{
                PM_Null            = 0x0,
                PM_Verilog         = 0x1,
                PM_Program         = 0x2,
                PM_Monitor         = 0x4,
                PM_SkipHeadComment = 0x8,
        };
        virtual void PrintMemoryWord(Memory::Word * m, FILE * fp, int addr, int
            printMode) = 0;
};

class Debugger
{
public:
        enum BreakpointType
        {
                BT_PC,          ///     break at PC
                BT_ICOUNT,      /// break at insnCount
                BT_MEM,         ///     break at memory change
        };
        class Breakpoint
        {
        public:
                BreakpointType btype;
                int val, ID, addr;
                unsigned int * mem;
                void Set(int id, BreakpointType bt, int value, unsigned int * mem0){
                        ID = id, btype = bt;
                        mem = 0; val = 0;
                        if(btype == BT_PC)                      { addr = value;}
                        else if(btype == BT_ICOUNT)     { val = value;}
                        else                                            { addr = value;
                             mem = mem0;}
                }
                const char * GetBreakpointTypeName(){
                        switch(btype){
                        case BT_PC:             return "PC";
                        case BT_ICOUNT: return "ICOUNT";
                        case BT_MEM:    return "MEM";
                        }
                }
                void PrintInfo(Debugger * dbg = 0){
                        printf("Breakpoint(%d) at ", ID);
                        if(btype == BT_PC)                      { printf("PC (0x%03x)",
                             addr);}
                        else if(btype == BT_ICOUNT)     { printf("insnCount (%d)", val
                            );}
                        else                                            { printf("MEM
                            (0x%03x) = (%04x : %04x)", addr, val, *mem);}
                        if(dbg) { printf("\n"); dbg->ShowCPUStatus();}
                        else    { printf("\n");}
                }
        };
        enum CommandType
        {
                CT_Start,                                       /// initial state
                CT_Run,                                         /// 'r'
                CT_Step,                                        /// 's'
        };
#define BREAKPOINT_COUNT 100
#define MONITOR_COUNT 100
        Breakpoint breakpoints[BREAKPOINT_COUNT], monitors[MONITOR_COUNT];
        int breakpointCount, monitorCount, bpID;
        int verboseMode, fileLogMode;
        CommandType com;
        CPU * cpu;
        static Debugger * globalDBG;
        Debugger(CPU * cpu0){
                if(globalDBG){ ABORT_PROGRAM(printf("globalDBG != 0\n"));}
                globalDBG = this;
                cpu = cpu0; breakpointCount = 0; monitorCount = 0; bpID = 0; com =
                    CT_Start; verboseMode = 0; fileLogMode = 0;
        }
        ~Debugger(){ globalDBG = 0;}
        bool Echo(){ return (com == CT_Step || verboseMode || DetectMonitor());}
        bool FileLog(){ return (com == CT_Start || fileLogMode || DetectMonitor());}
        void ShowCommand(){
                printf( "r : run\n"
                                "s : step\n"
                                "b : show breakpoints\n"
                                "d : delete breakpoint\n"
                                "p : set PC breakpoint\n"
                                "i : set InsnCount breakpoint\n"
                                "m : set Memory Monitor breakpoint\n"
                                "l : toggle verbose CPU-log mode (verboseMode = %d)\n"
                                "f : toggle file CPU-log mode (fileLogMode = %d)\n"
                                "w : show CPU status\n"
                                "n : show PC monitors\n"
                                "h : show commands\n"
                                "q : quit\n"
                                , verboseMode, fileLogMode
                                );
        }
        int Debug(){
                UpdateMonitor();
                switch(com){
                case CT_Run: if(!DetectBreakpoint())    return 0;
                case CT_Step:                                                   break;
                case CT_Start: ShowCommand();                   break;
                }
                return Command();
        }
        int Command(){
```

```
        while(1){
                printf("EX3-DBG > ");
                int ch = _getche();
                printf("\n");
                switch(ch){
                case 'r': com = CT_Run;
                            return 0;
                case '\n':
                case 's': com = CT_Step;
                            return 0;
                case 'b': ShowBreakpoints();                            break;
                case 'd': DeleteBreakpoint();                           break;
                case 'p': InsertBreakpoint(BT_PC);                      break;
                case 'i': InsertBreakpoint(BT_ICOUNT);          break;
                case 'm': InsertBreakpoint(BT_MEM);                     break;
                case 'n': ShowMonitors();
                            break;
                case 'l': ToggleVerboseMode();                          break;
                case 'f': ToggleFileLogMode();                          break;
                case 'w': ShowCPUStatus();
                            break;
                case 'h': ShowCommand();
                            break;
                case 'q': return 1;
                default : ShowCommand();
                            break;
                }
        }
        return 0;
}
void ToggleVerboseMode(){ verboseMode = 1 - verboseMode; printf("verboseMode =
    %d\n", verboseMode);}
void ToggleFileLogMode(){ fileLogMode = 1 - fileLogMode; printf("fileLogMode =
    %d\n", fileLogMode);}
void EnableAllLogs(){ verboseMode = 1; fileLogMode = 1;}
void ShowBreakpoints(){
        printf("%d breakpoints\n", breakpointCount);
        int i;
        for(i = 0; i < breakpointCount; i ++){ breakpoints[i].PrintInfo();}
}
void GetValue(const char * msg, const char * field, int * value){ printf(msg);
    scanf(field, value);}
void DeleteBreakpoint(){
        int ID;
        GetValue("delete breakpoint ID = ", "%d", &ID);
        int i;
        for(i = 0; i < breakpointCount; i ++){
                if(breakpoints[i].ID == ID){
                        breakpointCount --;
                        for(; i < breakpointCount; i ++){ breakpoints[i] =
                            breakpoints[i + 1];}
                        break;
                }
        }
        ShowBreakpoints();
}
void InsertBreakpoint(BreakpointType bt){
        if(breakpointCount >= BREAKPOINT_COUNT){ printf("Too many breakpoints
            ... (delete some to add new ones)\n"); return;}
        int value;
        unsigned int * mem0 = 0;
        if(bt == BT_ICOUNT){ GetValue("insn count (decimal) = ", "%d", &value
            );}
        else{
                GetValue("address (hex) = ", "%x", &value);
                if(value >= cpu->mem->size){ printf("address is out of range
                    ...\n"); return;}
                if(bt == BT_MEM){ mem0 = &cpu->mem->word[value].value;}
        }
        breakpoints[breakpointCount ++].Set(bpID ++, bt , value , mem0);
        ShowBreakpoints();
}
void InsertMonitorOrBreakpoint(int status , int addr , bool isInsn){
        if(addr < 0 || addr >= cpu->mem->size){ ABORT_PROGRAM("Error in
            InsertMonitorOrBreakpoint!!!\n");}
        BreakpointType bt = (isInsn) ? BT_PC : BT_MEM;
        unsigned int * mem0 = 0;
        if(bt == BT_MEM){ mem0 = &cpu->mem->word[addr].value;}
        if(status & Memory::MS_Monitor){
                if(monitorCount >= MONITOR_COUNT){ printf("[
                    InsertMonitorOrBreakpoint] Too many monitors... \n");
                    return;}
                monitors[monitorCount].Set(monitorCount , bt , addr , mem0);
                monitorCount ++;
        }
        if(status & Memory::MS_Breakpoint){
                if(breakpointCount >= BREAKPOINT_COUNT){ printf("[
                    InsertMonitorOrBreakpoint] Too many breakpoints... (delete
                    some to add new ones)\n"); return;}
                breakpoints[breakpointCount ++].Set(bpID ++, bt , addr , mem0);
        }
}
bool DetectBreakpoint(){
        int i;
        Breakpoint * bp = breakpoints;
        for(i = 0; i < breakpointCount; i ++, bp ++){
                switch(bp->btype){
                case BT_PC:             if(cpu->_PC == bp->addr)
                            { bp->PrintInfo(this); return true;} break;
                case BT_ICOUNT: if(cpu->cycle_count == bp->val) { bp->PrintInfo
                    (this); return true;} break;
                case BT_MEM:    if(*bp->mem != bp->val)                 { bp->
                    PrintInfo(this); bp->val = *bp->mem; return true;} break;
                }
        }
        return false;
}
void ShowMonitors(){
        int i;
        Breakpoint * mon = monitors;
        printf("monitors = {");
        for(i = 0; i < monitorCount; i ++, mon ++){
                if(i){ printf(", ");}
                printf("0x%03x(%s)", mon->addr, (mon->btype == BT_PC) ? "PC" :
                    "MEM");
        }
        printf("}\n");
}
bool DetectMonitor(){
        int i;
        Breakpoint * mon = monitors;
        for(i = 0; i < monitorCount; i ++, mon ++){
                switch(mon->btype){
                case BT_PC:             if(cpu->_PC == mon->addr)
                            { return true;} break;
                case BT_MEM:    if(*mon->mem != mon->val)               {
                    return true;} break;
                }
        }
        return false;
}
void UpdateMonitor(){
        int i;
        Breakpoint * mon = monitors;
        for(i = 0; i < monitorCount; i ++, mon ++){ if(mon->mem){ mon->val = *
            mon->mem;}}
}
void ShowCPUStatus(){ cpu->PrintStatus(stdout, 0);}
};

#define MAX_LINE 1000
#define VERILOG_DIR "../verilog/"
class ASMParser
{
public:
CPU * cpu;
FILE * fp;
const char * asm_name;
static const char * tool_name;
int asmLineNum, blockCommentPending;
ASMParser(CPU * cpu0, const char * fname){ cpu = cpu0; fp = 0; asm_name = fname
    ;}
void Close(){
        if(fp){ fclose(fp);}
        fp = 0;
        WriteVerilogMemProbeFile();
}
~ASMParser(){ if(fp) fclose(fp);}
int Open(){
        if(strcmp(asm_name + strlen(asm_name) - 4, ".asm") != 0){ printf("
            Incorrect ASM file extension : %s\n", asm_name); return -1;}
        fp = fopen(asm_name, "r");
        if(fp == 0){ printf("Cannot open (%s)\n", asm_name); return -1;}
        ///     1st pass : extract labels
        if(Parse(1) != 0){ return -1;}
        EMIT_MESSAGE_0(cpu->PrintSeparator, cpu->fplog);
        EMIT_MESSAGE_0(cpu->label.PrintLabels, cpu->fplog);
        rewind(fp);
        ///     2nd pass : extract instructions
        if(Parse(2) != 0){ return -1;}
        EMIT_MESSAGE_0(cpu->PrintSeparator, cpu->fplog);
        EMIT_MESSAGE_1(cpu->PrintMemory, cpu->fplog, CPU::PM_Program);
        EMIT_MESSAGE_0(cpu->PrintSeparator, cpu->fplog);
        fclose(fp);
        fp = 0;
        if(WriteVerilogMemFile() != 0 || WriteVerilogMonitorFile() != 0){
            return -1;}
        return 0;
}
FILE * OpenFile(const char * extname){
        FILE * fp0 = 0;
        char * mem_fname = new char[strlen(asm_name) + strlen(VERILOG_DIR) +
            strlen(extname) + 1];
        sprintf(mem_fname, "%s%s", VERILOG_DIR, asm_name);
        char * p = mem_fname + strlen(mem_fname) - 4;
        if(*p != '.')   { printf("sorry, something is wrong... (please contact
            admin)\n");}
        else                    { strcpy(p, extname); fp0 = fopen(mem_fname, "w
            ");}
        delete [] mem_fname;
        return fp0;
}
int WriteVerilogMemFile(){
        FILE * fp0 = OpenFile(".mem");
        if(fp0){
                fprintf(fp0, "/// Verilog Memory Initialization File (.mem)
                    generated by %s\n\n"
                                          "/// 12-bit address\n"
                                          "/// 16-bit data\n\n"
                                          , tool_name);
                cpu->PrintMemory(fp0, CPU::PM_Verilog | CPU::PM_Program);
                fclose(fp0);
                return 0;
        }
        else{ printf("Cannot open memory output file (skipped memory output)\n"
            ); return -1;}
}
int WriteVerilogMemProbeFile(){
        FILE * fp0 = OpenFile(".prb");
        if(fp0){
                fprintf(fp0, "/// Verilog Memory Probe File (.prb) generated by
                     %s\n\n"
```

```
                                         "/// bit[31:28] : memory type (0000 :
                                             data, 1111 : end-of-memory\n"
                                         "/// bit[27:16] : address\n"
                                         "/// bit[15:0]  : data\n\n"
                                         , tool_name);
                cpu->PrintMemory(fp0, CPU::PM_Verilog);
                fprintf(fp0, "%1x%03x%04x\t///\t%s\n", 0xf, 0, 0, "end-of-
                    memory");
                fclose(fp0);
                return 0;
        }
        else{ printf("Cannot open memory output file (skipped memory output)\n"
            ); return -1;}
}
int WriteVerilogMonitorFile(){
        FILE * fp0 = OpenFile(".mon");
        if(fp0){
                fprintf(fp0, "/// Verilog Monitor File (.mon) generated by %s\n
                    \n"
                                         "/// bit[31:28] : memory type (0000 :
                                             prog, 0001 : data, 1111 : end-of-
                                             memory\n"
                                         "/// bit[27:16] : address\n"
                                         "/// bit[15:0]  : data\n\n"
                                         , tool_name);
                cpu->PrintMemory(fp0, CPU::PM_Verilog | CPU::PM_Monitor);
                fprintf(fp0, "%08x\t///\t%s\n", (unsigned int)(-1), "end-of-
                    memory");
                fclose(fp0);
                return 0;
        }
        else{ printf("Cannot open memory output file (skipped memory output)\n"
            ); return -1;}
}
void SkipWhiteSpace(const char ** pp){
        const char * p = *pp;
        while(*p == ' ' || *p == '\t'){ p ++;}
        *pp = p;
}
int GetNum(const char ** pp, int insnType){
        if(insnType == InsnSet::I_CHR){ return **pp;}
        const char * p = *pp;
        int val = 0;
        int first = 1;
        int base = 0;
        switch(insnType){
                case InsnSet::I_ORG:
                case InsnSet::I_HEX:    base = 16;      break;
                case InsnSet::I_DEC:    base = 10;      break;
                default:                                return -1;
        }
        int sign = 0;
        if(*p == '-'){ sign = 1; p ++;}
```

```
                while(*p){
                        if(*p == ' ' || *p == '\t' || *p == '\n' || *p == '\r' || *p ==
                             COMMENT){ break;}
                        val *= base;
                        int num = *p - '0';
                        if(num >= 0 && num <= 9){ val += num;}
                        else if(base == 16){
                                int upper = *p - 'A';
                                int lower = *p - 'a';
                                if              (upper >= 0 && upper <= 5)      { val
                                    += upper + 10;}
                                else if (lower >= 0 && lower <= 5)      { val += lower
                                    + 10;}
                                else
                                                    { return -1;}   ///     error
                        }
                        else{ return -1;}       /// error
                        p ++;
                        first = 0;
                }
                *pp = p;
                if(sign){ val = -val;}
                return (first) ? -1 : val;
        }
        int ParseLabel(const char * p){
                int nlen = 0;
                while(*p){
                        switch(*p){
                                case ' ':
                                case '\t':      return 0;
                                case ',':       return nlen;
                                default:        break;
                        }
                        nlen ++;
                        p ++;
                }
                return 0;
        }
        int GetLabelLength(const char * p){
                int nlen = 0;
                while(*p){
                        switch(*p){
                                case ' ':
                                case '\r':
                                case '\t':
                                case '\n':
                                case ',':       return nlen;
                                default:        break;
                        }
                        nlen ++;
                        p ++;
                }
                return 0;
        }
        int ExtractComment(const char * p, int addr, int headFlag){
                if(addr >= 0 && addr < cpu->mem->size){
                        Memory::Word * m = &cpu->mem->word[addr];
                        SkipWhiteSpace(&p);
                        if(*p == '/'){
                                const char * pp = p;
                                while(*pp && *pp != '\r' && *pp != '\n'){ pp ++;}
                                m->SetComment(p, pp - p, headFlag);
                                return 1;
                        }
                }
                return 0;
        }
        void PrintErrorLocation(){ printf("Error occurred on line %d\n", asmLineNum);}
        int ParseLabel(int passNum, const char ** p, int & addr){
                if(passNum == 2 && ExtractComment(*p, addr, 1)){ return 1;}
                SkipWhiteSpace(p);
                if(**p == '/'){ return 1;}      ///     comment...
                int labelLength = ParseLabel(*p);
                if(labelLength > 0){
                        if(cpu->label.annotation.AddAnnotation(*p, labelLength, (
                            unsigned int) addr)){}
                        else if(passNum == 1){  ///     create label
                                if(cpu->label.AddLabel(*p, labelLength, (unsigned int)
                                    addr) == 0){ PrintErrorLocation(); return -1;}
                        }
                        else if(cpu->label.GetLabel(*p, labelLength) == 0)
                        { printf("ERROR: label(%s) not found in 1st pass (bug in the
                            parser...)\n", p); PrintErrorLocation(); return -1;}
                        *p += labelLength + 1;
                        SkipWhiteSpace(p);
                        if(passNum == 2){ ExtractComment(*p, addr, 1);}
                        if(**p == '/'){ return 1;}
                }
                return (**p == '\n' || **p == '\r') ? 1 : 0;
        }
        int ParseNonInsn(int passNum, const char * p, int insnID, int & addr){
                SkipWhiteSpace(&p);
                int param = GetNum(&p, insnID);
                Memory::Word * m = &cpu->mem->word[addr];
                int endReached = 0;
                switch(insnID){
                        case InsnSet::I_ORG:
                                addr = param;
                                if(!cpu->mem->IsValidAddress(addr))
                                { printf("ERROR: ORG has invalid address %d (0x%x)\n",
                                    addr, addr); PrintErrorLocation(); return -1;}
                                if(passNum == 2){ ExtractComment(p, addr, 1);}
                                break;
                        case InsnSet::I_END:
                                endReached = 1;
                                break;
                        case InsnSet::I_SYM:
                                if(passNum == 2){
                                        SkipWhiteSpace(&p);
                                        int len = GetLabelLength(p);
                                        if(len == 0)
                                        { printf("ERROR: label(%s) length = 0 (bug in
                                            parser)\n", p); PrintErrorLocation();
                                            return -1;}
                                        Label::Element * lb = cpu->label.GetLabel(p,
                                            len);
                                        if(lb == 0)
                                        { printf("ERROR: label(%s) not found in the 1st
                                             pass (bug in the program...)\n", p);
                                            PrintErrorLocation(); return -1;}
                                        if(!cpu->mem->IsValidAddress(lb->address))
                                        { printf("ERROR: label(%s) has invalid address
                                            (%x)\n", lb->name, lb->address);
                                            PrintErrorLocation(); return -1;}
                                        param = lb->address;
                                }
                        case InsnSet::I_CHR:
                        case InsnSet::I_HEX:
                        case InsnSet::I_DEC:
                                if(passNum == 2){
                                        m->SetStatus(&cpu->label.annotation, addr);
                                        cpu->dbg->InsertMonitorOrBreakpoint(m->status,
                                            addr, false);
                                        m->value = param;
                                }
                                if(passNum == 2){ ExtractComment(p, addr, 0);}
                                addr ++;
                                break;
                        default:
                                break;
                }
                return (endReached) ? 1 : 0;
        }
        int ParseBlockCommentStart(const char ** p){
                if(!blockCommentPending){
                        SkipWhiteSpace(p);
                        if(strncmp(*p, "/*", 2) == 0){ blockCommentPending = 1; (*p) +=
                             2; return 1;}
                }
                return 0;
        }
        int ParseBlockCommentEnd(const char ** p){
                if(blockCommentPending){
                        while(**p){
                                if(strncmp(*p, "*/", 2) == 0){ (*p) += 2;
                                    blockCommentPending = 0; return 1;}
                                else{ (*p) ++;}
                        }
                }
                return 0;
        }
        int ParseBlockComment(const char ** p){
                while(ParseBlockCommentStart(p) || ParseBlockCommentEnd(p)){}
                return (blockCommentPending != 0);
        }
        int Parse(int passNum){
                if(passNum != 1 && passNum != 2){ printf("passNum = %d is invalid\n",
                    passNum); return -1;}
                char buf[MAX_LINE];
                int addr = 0, endReached = 0;
                blockCommentPending = 0;
                asmLineNum = 0;
                while(!endReached){
                        if(fgets(buf, MAX_LINE, fp) == 0){ printf("ERROR: End of file
                            reached before END\n"); return -1;}
                        asmLineNum ++;
                        const char * p = buf;
                        if(ParseBlockComment(&p)){ continue;}
                        switch(ParseLabel(passNum, &p, addr)){
                                case -1:        return -1;
                                case  1:        continue;
                                default:        break;
                        }
                        InsnSet::Insn * insn = cpu->isa->SearchInsn(&p);
                        if(insn == 0){ printf("ERROR: Invalid instruction (%s)\n", p);
                            PrintErrorLocation(); return -1;}
                        switch(ParseInsn(passNum, p, insn, addr)){
                                case -1:        return -1;
                                case  1:        continue;
                                default:        break;
                        }
                        switch(ParseNonInsn(passNum, p, insn->ID, addr)){
                                case -1:        return -1;
                                case  1:        endReached = 1;
                                default:        break;
                        }
                }
                if(passNum == 1){
                        if(!cpu->mem->IsValidAddress(addr)){ printf("ERROR: program
                            extends to invalid address %d (0x%x)\n", addr, addr);
                            return -1;}
                        cpu->mem->maxAddr = addr;
                }
                return 0;
        }
        ///     virtual abstract functions : must be defined in actual class
        virtual int ParseInsn(int passNum, const char * p, InsnSet::Insn * insn, int &
            addr) = 0;
};

class String
{
```

```
public:
        char * string;
        int length, allocLength;
#define DEFAULT_ALLOC_LENGTH 100
        void Allocate(){
                char * s = new char[allocLength];
                if(string){
                        strncpy(s, string, length);
                        delete [] string;
                }
                s[length] = 0;
                string = s;
        }
        String(){
                allocLength = DEFAULT_ALLOC_LENGTH;
                string = 0;
                length = 0;
                Allocate();
        }
        ~String(){ delete [] string;}
        void Insert(int c){
                if(c == 0){ Insert('\\'); Insert('0'); return;}
                if(allocLength - length <= 1){
                        allocLength = allocLength + (allocLength >> 1) + 1;
                        Allocate();
                }
                string[length ++] = (char) c;
                string[length] = 0;
        }
        void Reset(){ length = 0; string[0] = 0;}
};

class System
{
public:
#define MAX_INTERVAL 50
        class RandomPeripheral
        {
        public:
                CPU * cpu;
                const char * name;
                int type;       /// type = 0 : output, type = 1 : input
                int interval;
                FILE * fp;
                RandomPeripheral(CPU * cpu0, const char * n, int t) : cpu(cpu0), name(n
                    ), type(t), interval(-1), fp(0){}
                virtual ~RandomPeripheral(){ if(fp){ fclose(fp);}}
                enum TraceType{ TT_Interval = 0, TT_Data = 1 };
                void Open(FILE * fp0){
                        fp = fp0;
                        if(fp){
                                fprintf(fp, "/// RandomPeripheral trace (%s) :\n", name
                                    );
                                fprintf(fp, "/// bit[17]   (port ID) : 0 = gpio, 1 =
                                    uart\n");
                                fprintf(fp, "/// bit[16]   (trace type) : 0 = interval
                                    cycles, 1 = data\n");
                                fprintf(fp, "/// bit[15:0] (value)\n\n");
                        }
                }
                void Close(){
                        if(type == 0 && IsPortReady())  { AccessPort();}        ///
                                 access output ready port before closing...
                        if(fp)                                                  {
                            fclose(fp);}
                        fp = 0;
                }
                int GetPortID(){ return cpu->GetSelectedPortID();}
                const char * GetPortName(){ return (GetPortID()) ? "SIO" : "PIO";}
                void RecordTrace(int rtype, unsigned char value){ if(fp){ fprintf(fp, "
                    %x%04x\n", (GetPortID() << 1) | rtype, value);}}
                virtual int GetInterval(){ return rand() % MAX_INTERVAL;}
                void SetRandomInterval(){
                        interval = GetInterval();
                        EMIT_MESSAGE_4(fprintf, cpu->fplog,
                                "-------------------------------------\n"
                                "%s[%s].interval = %d\n"
                                "-------------------------------------\n"
                                , name, GetPortName(), interval);
                        RecordTrace(TT_Interval, interval);
                }
                int Execute(void (* accessPortHook)(int) = 0){
                        if(interval < 0 && IsPortReady())       { SetRandomInterval();}
                        if(interval == 0)                                       { (
                            accessPortHook && AccessHookEnabled()) ? accessPortHook(
                            type) : AccessPort();}
                        if(interval >= 0)                                       {
                            interval --;}
                        return 0;
                }
                ///     virtual abstract functions : must be defined in actual class
                virtual bool IsPortReady() = 0;
                virtual void AccessPort() = 0;
                virtual bool AccessHookEnabled() = 0;
        };
        class TerminalViewer
        {
                void Reset(){ str->Reset();}
        public:
                String * str;
                TerminalViewer(){ str = new String;}
                ~TerminalViewer(){ if(str) delete str;}
                void PrintView(){
                        printf("------------ TERMINAL VIEWER -------------\n");
                        printf("%s\n", str->string);
                        printf("------------------------------------------\n");
                        Reset();
                }
        };
        class InputTerminal : public RandomPeripheral
        {
        public:
                TerminalViewer * termView;
                InputTerminal(CPU * cpu0) : RandomPeripheral(cpu0, "in", 1){ termView =
                     0;}
                bool IsPortReady(){ return cpu->IsWaitingForInput();}
                virtual void AccessPort(){
                        termView->PrintView();
                        printf("IN-term [%s] > ", GetPortName());
                        int value = _getche();
                        printf("\n");
                        switch(value){
                        case 0x03:      /// ctrl-C
                                ABORT_PROGRAM(printf("Program terminated on Input
                                    terminal (key = %02x)\n", value););
                        case 0x1B:      /// ESC
                                cpu->dbg->Command();
                                return;
                        }
                        PutInput(value);
                }
                void PutInput(int value){
                        termView->str->Insert(value);
                        cpu->SetInput((unsigned char) value);
                        EMIT_MESSAGE_5(fprintf, cpu->fplog,
                                "-------------------------------------\n"
                                "cpu->input[%s] = 0x%02x (%2d : '%c')\n"
                                "-------------------------------------\n"
                                , GetPortName(), value, value, value);
                        RecordTrace(TT_Data, value);
                }
                virtual bool AccessHookEnabled(){ return true;}
        };
        class OutputTerminal : public RandomPeripheral
        {
        public:
                TerminalViewer * termView;
                OutputTerminal(CPU * cpu0) : RandomPeripheral(cpu0, "out", 0){ termView
                     = 0;}
                virtual int GetInterval(){ return 0;}
                bool IsPortReady(){ return cpu->IsOutputReady();}
                virtual void AccessPort(){
                        int value = GetOutput();
                        RecordTrace(TT_Data, value);
                        termView->str->Insert(value);
                }
                int GetOutput(){
                        int value = cpu->GetOutput();
                        printf("value = %d\n",value);
```

```
                        EMIT_MESSAGE_5(fprintf, cpu->fplog,
                                "---------------------------------------\n"
                                "cpu->output[%s] = 0x%02x (%2d : '%c')\n"
                                "---------------------------------------\n"
                                , GetPortName(), value, value, value);
                        return value;
                }
                virtual bool AccessHookEnabled(){ return cpu->IsOutputReady();}
        };
};

#endif /// _COMMON_ASM_H_INCLUDED_
```

ソースコード 2　ex3_asm.cpp

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "ex3_asm.h"

#define INSN_FUNC_DEFINITION
#include "ex3_asm_def.h"
#undef INSN_FUNC_DEFINITION

Debugger * Debugger::globalDBG = 0;

#if !defined(WIN32)
int _getche(void){
        GetChSetting getchSetting;
        return getchar();
}
#endif

const char * ASMParser::tool_name = "ex3_asm";

#define SYS_MODE 0
/// SYS_MODE = 0 : RunCPUModel();
/// SYS_MODE = 1 : RunSoftModel();                         --> need tic_tac_toe.cpp
/// SYS_MODE = 2 : RunCPU_vs_SoftModel();        --> need tic_tac_toe.cpp

int main(int argc, char ** argv)
{
//#define CUR_VERSION "ver0.0 (10/05/2011)"
//#define CUR_VERSION "ver0.1 (10/29/2012)"
//#define CUR_VERSION "ver0.2 (12/08/2012)"
//#define CUR_VERSION "ver0.3 (01/16/2013)"
#define CUR_VERSION "ver0.4 (01/17/2013)"
        printf("EX3 Instruction-Set Simulator : %s\n", CUR_VERSION);
        if(argc != 2){
                printf("usage: %s <asm filename>\n", argv[0]);
                printf("\nhit return : ");
                getchar();
                return -1;
        }
        ASMParser::tool_name = argv[0];
        printf("asmfile = %s\n", argv[1]);
        EX3_TerminalSystem ex3_sys(argv[1]);
        if(ex3_sys.parser.Open() != 0){
                printf("ERROR during parsing\n", argv[0]);
                printf("\nhit return : ");
                getchar();
                return -1;
        }
#if (SYS_MODE == 1)
        ex3_sys.RunSoftModel();
#elif (SYS_MODE == 2)
        ex3_sys.RunCPU_vs_SoftModel();
#else
        ex3_sys.RunCPUModel();
#endif
        printf("\nhit return : ");
        getchar();
        return (ex3_sys.cpu.mem->errorFlag) ? -1 : 0;
}
```

ソースコード 3　ex3_asm.h

```
#if !defined _EX3_ASM_H_INCLUDED_

#define _EX3_ASM_H_INCLUDED_

#include "common_asm.h"

class EX3_Memory : public Memory
{
public:
        EX3_Memory(int sz) : Memory(sz){}
        bool IsIndirectMemoryAccess(Word * w){ return (w->value & 0x8000) != 0;}
        unsigned int GetAddressField(Word * m){ return m->value & 0xfff;}
};

class EX3_CPU;
class EX3_InsnSet : public InsnSet
{
public:
        enum InsnType{ NON_INSN, MEM_INSN, REG_INSN,};
        enum InsnID
        {
                I_ANCHOR = InsnSet::I_TAIL - 1,
#define INSN_ENUM
#include "ex3_asm_def.h"
#undef INSN_ENUM
```

```
                I_COUNT
        };

#define INSN_FUNC_DECLARATION
#include "ex3_asm_def.h"
#undef INSN_FUNC_DECLARATION

        EX3_InsnSet(int icount) : InsnSet(icount){
#define INSN_CREATE
#include "ex3_asm_def.h"
#undef INSN_CREATE
        }
};

class EX3_CPU : public CPU
{
public:
        unsigned int _AC, _E, _R, _IEN, _IMSK, _IOT, inputPending;
        EX3_CPU(int mem_size) : CPU("ex3_cpu.log"){
                mem = new EX3_Memory(mem_size);
                isa = new EX3_InsnSet(EX3_InsnSet::I_COUNT);
                dbg = new Debugger(this);
        }
        void Reset(){
                CPU::Reset();
                _AC             = 0;
                _E              = 0;
                _R              = 0;    ///     interrupt detection
                _IEN    = 0;    ///     interrupt enable
                _IMSK   = 0;    ///     interrupt mask (4 bits) : {UART_RX, UART_TX,
                    GP_IN, GP_OUT}
                _IOT    = 0;    ///     io select : 1 (SIO), 0 (PIO)
                inputPending = 0;
        }
        virtual int GetSelectedPortID(){ return (_IOT) & 1;}
        virtual bool IsWaitingForInput(){
                bool inputEnabled = (_IMSK & 0x8) || (_IMSK & 0x2) ;
                return IsInputReady() && (_IEN && inputEnabled || inputPending && !
                    intr_pending);}
        void SetFG(unsigned int * fg, int val){
                int mask = (1 << GetSelectedPortID());
                if(mask == 2){ val <<= 1;}
                *fg = (*fg & (~mask)) | val;
        }
        virtual void _SetFGI(int val){ SetFG(&_FGI, val);}
        virtual void _SetFGO(int val){ SetFG(&_FGO, val);}
        virtual int _GetFGI(){ return (_FGI & (1 << GetSelectedPortID())) != 0;}
        virtual int _GetFGO(){ return (_FGO & (1 << GetSelectedPortID())) != 0;}
        void PrintDataMemory(){
                EMIT_MESSAGE_0(PrintSeparator, fplog);
                EMIT_MESSAGE_1(PrintMemory, fplog, CPU::PM_Null);
        }
        void PrintStatus(FILE * fp, bool intr_cycle){
```

```
        if(fp){
                if(mem->curCode == 0){ mem->ProbeInsn(_PC);}
                if(intr_cycle)
                        fprintf(fp, "[%4d] INTR [%4d:A] + [%4d:I] : PC(%04x),
                            oldPC(%04x), R(%x), IEN(%x), IMSK(%x), FGI/O(%x:%x
                            )\n",
                                cycle_count, appl_cycle_count, intr_cycle_count
                                    , _PC, _oldPC, _R, _IEN, _IMSK, _FGI, _FGO
                                    );
                else if(mem->curCode == 0){ fprintf(fp, " ERROR!!!! curCode =
                    0... \n");}
                else{
                        if(mem->curCode->headComment){ fprintf(fp, "/%s\n", mem
                            ->curCode->headComment);}
                        if(isa->insn[mem->curCode->insnID].showMemFlag){
                                fprintf(fp, "[%4d] S(%d),R(%x),IO(%x:%x:%x|%x:%
                                    x|%x:%x),MEM(%04x),E(%x),PC(%03x): ",
                                        cycle_count, _S, _R, _IEN, _IMSK, _IOT,
                                             _FGI, _INPR, _FGO, _OUTR,
                                        mem->Operand(), _E, _PC);
                        }
                        else{
                                fprintf(fp, "[%4d] S(%d),R(%x),IO(%x:%x:%x|%x:%
                                    x|%x:%x), AC(%04x),E(%x),PC(%03x): ",
                                        cycle_count, _S, _R, _IEN, _IMSK, _IOT,
                                             _FGI, _INPR, _FGO, _OUTR,
                                        _AC, _E, _PC);
                        }
                        PrintMemoryWord(mem->curCode, fp, _oldPC, CPU::
                            PM_Program | CPU::PM_SkipHeadComment);
                }
                if(mem->errorFlag){ printf("Memory error occurred!!!\n");}
        }
}
int Execute(){
        cycle_count ++;
        bool intr_cycle = (_R == 1);
        intr_pending   |= (_R == 1);
        if(intr_cycle){ ///     interrupt cycle
                mem->word[0].value = _PC;       ///     MEM[0] <- PC
                _PC = 1;                                        ///     PC
                          <- 1 : 1st instruction of interrupt handler
                _IEN = 0;                                       ///     IEN <-
                    0 : disable interrupt upon entering interrupt handler
                _R = 0;                                         ///     R
                           <- 0 : disable interrupt detection flag
        }
        else{   ///     instruction cycle
                int IO_ready = (((_FGI & 0x2) != 0) << 3) | (((_FGO & 0x2) !=
                    0) << 2) | (((_FGI & 0x1) != 0) << 1) | (((_FGO & 0x1) !=
                    0));
                if(_IEN && (_IMSK & IO_ready)){ _R = 1;}
                mem->FetchInsn(_PC);
```

```
                        if(mem->curCode)        { isa->insn[mem->curCode->insnID].
                            operation(this);}
                }
                if(intr_pending)        { intr_cycle_count ++;}
                else                            { appl_cycle_count ++;}
                /// 0xc000 : BUN 0 I (indirect jump at addr 0) --> this is a return
                    from interrupt handler
                if(!intr_cycle && mem->curCode && mem->curCode->value == 0xc000){
                    intr_pending = false; inputPending = false;}
                EMIT_MESSAGE_1(PrintStatus, fplog, intr_cycle);
                _oldPC = _PC;
                return (mem->errorFlag || _S == 0) ? -1 : 0;
        }
#define VERILOG_READ_MEM_SEPARATOR " "
        void PrintMemoryWord(Memory::Word * m, FILE * fp, int addr, int printMode)
        {
                bool verilogFlag = (printMode & CPU::PM_Verilog) != 0;
                bool showProgram = (printMode & CPU::PM_Program) != 0;
                bool showMonitor = (printMode & CPU::PM_Monitor) != 0;
                bool skipHeadComment = (printMode & CPU::PM_SkipHeadComment) != 0;
                if(fp){
                        InsnSet::Insn * insn = &isa->insn[m->insnID];
                        Label::Element * lb = label.GetLabel(addr);
                        if(lb == 0 && insn->type == EX3_InsnSet::NON_INSN && m->status
                            == 0 && m->value == 0)   { return;}
                        if(showMonitor){ if((m->status & Memory::MS_Monitor) == 0)

                                    { return;}}
                        else if(insn->type != EX3_InsnSet::NON_INSN && !showProgram)
                                                                                {
                            return;}
                        if(!skipHeadComment && m->headComment){ fprintf(fp, "/%s\n", m
                            ->headComment);}
                        if(verilogFlag){
                                if(showProgram)                 { fprintf(fp, "@%03x%s
                                    %08x\t///", addr, VERILOG_READ_MEM_SEPARATOR, m->
                                    value);}
                                else if(showMonitor)    { fprintf(fp, "%1x%03x%08x\t
                                    ///\t", (insn->type == EX3_InsnSet::NON_INSN), addr
                                    , m->value);}
                                else                                    { fprintf(fp, "
                                    %1x%03x%08x\t///\t", 0, addr, m->value);}
                        }
                        fprintf(fp, "%*s%s", label.maxLabelLength, (lb) ? lb->name : ""
                            , (lb) ? ": " : "  ");
                        if(insn->type == EX3_InsnSet::MEM_INSN){
                                int addrValue = mem->GetAddressField(m);
                                Label::Element * target_lb = label.GetLabel(addrValue);
                                fprintf(fp, "%04x [%08x]: %s %03x %s (%*s)", addr, m->
                                    value, insn->name, addrValue,
                                        (mem->IsIndirectMemoryAccess(m)) ? "I" : " ",
                                            label.maxLabelLength, (target_lb) ?
                                            target_lb->name : "???");
                        }
                        else if(insn->type == EX3_InsnSet::REG_INSN)
                        { fprintf(fp, "%04x [%08x]: %s%*s", addr, m->value, insn->name,
                             label.maxLabelLength + 9, "");}
                        else if(m->status != 0){
                                const char * field = (m->value >= 0x20 && m->value <= 0
                                    x7e) ? "%04x [%08x]: (%5d:  '%c')%*s" : "%04x [%08x
                                    ]: (%5d:%5d)%*s";
                                fprintf(fp, field, addr, m->value, (signed int) m->
                                    value, m->value, label.maxLabelLength + 1, "");
                        }
                        else if(m->value != 0)
                        { EMIT_MESSAGE_3(fprintf, fp, "%04x [%08x]: ERROR!!! this
                            memory location is unused but has non-zero value!!!\n",
                            addr, m->value);}
                        if(m->tailComment){ fprintf(fp, "\t/%s", m->tailComment);}
                        fprintf(fp, "\n");
                }
        }
};

class EX3_ASMParser : public ASMParser
{
public:
        EX3_ASMParser(CPU * cpu0, const char * fname) : ASMParser(cpu0, fname){}
        int ParseInsn(int passNum, const char * p, InsnSet::Insn * insn, int & addr){
                if(insn->type == EX3_InsnSet::REG_INSN || insn->type == EX3_InsnSet::
                    MEM_INSN){
                        if(passNum == 2){
                                Memory::Word * m = &cpu->mem->word[addr];
                                m->insnID = insn->ID;
                                m->value = insn->code;
                                m->SetStatus(&cpu->label.annotation, addr);
                                cpu->dbg->InsertMonitorOrBreakpoint(m->status, addr,
                                    true);
                                if(insn->type == EX3_InsnSet::MEM_INSN){
                                        SkipWhiteSpace(&p);
                                        int len = GetLabelLength(p);
                                        if(len == 0){ printf("ERROR: label(%s) length =
                                             0 (bug in parser)\n", p);
                                            PrintErrorLocation(); return -1;}
                                        Label::Element * lb = cpu->label.GetLabel(p,
                                            len);
                                        if(lb == 0){ printf("ERROR: label(%s) not found
                                             in the 1st pass (bug in the program...)\n"
                                            , p); PrintErrorLocation(); return -1;}
                                        if(!cpu->mem->IsValidAddress(lb->address)){
                                            printf("ERROR: label(%s) has invalid
                                            address (%x)\n", lb->name, lb->address);
                                            PrintErrorLocation(); return -1;}
                                        m->value |= lb->address;
                                        p += lb->nlen;
                                        SkipWhiteSpace(&p);
                                        if(*p == 'I'){ m->value |= 0x8000; p ++;}
                                                   ///     set bit15 to 1
                                }
                        }
                        if(passNum == 2){ ExtractComment(p, addr, 0);}
                        addr ++;
                        return 1;
                }
                else{ return 0;}
        }
};

class EX3_TerminalSystem
{
public:
        EX3_CPU cpu;
        EX3_ASMParser parser;
        System::TerminalViewer termView;
        System::InputTerminal inTerm;
        System::OutputTerminal outTerm;
        EX3_TerminalSystem(const char * fname) : cpu(0x1000), parser(&cpu, fname),
            inTerm(&cpu), outTerm(&cpu){
                inTerm.Open(parser.OpenFile("_in.log"));
                outTerm.Open(parser.OpenFile("_out.log"));
                inTerm.termView = &termView;
                outTerm.termView = &termView;
        }
        ~EX3_TerminalSystem(){ cpu.dbg->EnableAllLogs(); Close();}
        void AccessInPort(){ inTerm.AccessPort(); cpu._FGI = 0;}
        void AccessOutPort(){ cpu._FGO = 0; outTerm.AccessPort();}
        void PrintString(const char * s){ while(*s){ cpu._OUTR = *(s ++); AccessOutPort
            ();}}
        void Close(){ cpu.PrintDataMemory(); parser.Close();}
        void RunCPUModel(){
                cpu.Reset();
                int inFlag, outFlag, cpuFlag;
                do{
                        if(cpu.dbg->Debug()){ break;}
                        ///     1. simulate components : inTerm, outTerm, cpu
                        inFlag = inTerm.Execute();
                        outFlag = outTerm.Execute();
                        cpuFlag = cpu.Execute();
                } while(inFlag + cpuFlag + outFlag == 0);
                inTerm.Close();
                outTerm.Close();
                termView.PrintView();
    printf("step count = %d\n", cpu.cycle_count);
        }
        ///     below functions are defined in tic_tac_toe.cpp:
        void RunSoftModel();
        void RunCPU_vs_SoftModel();
};
#endif /// _EX3_ASM_H_INCLUDED_
```

ソースコード 4　ex3_asm_def.h

```

#if defined INSN_ENUM
#define INSN(i, code, showMem, func) I_##i,
/// example: INSN(AND, 0x0000, 0, AC &= MEM;) ==> I_AND,
#elif defined INSN_CREATE
#define INSN(i, code, showMem, func) insn[I_##i].Set(I_##i, #i, ((code & 0x7000) != 0
    x7000) ? MEM_INSN : REG_INSN, showMem, OP_##i, code);
/// example: INSN(AND, 0x0000, 0, AC &= MEM;) ==> insn[I_AND].Set(I_AND, "AND", MemInsn
    , 0, OP_AND, 0x0000);
#elif defined INSN_FUNC_DECLARATION
#define INSN(i, code, showMem, func) static void OP_##i(CPU * cpu);
/// example: INSN(AND, 0x0000, 0, AC &= MEM;) ==> static void OP_AND(CPU * cpu);
#elif defined INSN_FUNC_DEFINITION
#define INSN(i, code, showMem, func) void EX3_InsnSet::OP_##i(CPU * cpu){ func }
/// example: INSN(AND, 0x0000, 0, AC &= MEM;) ==> void MY_InsnSet::OP_AND(CPU * cpu){
    AC &= MEM; }
#else
#define INSN(i, code, showMem, func)            ///     empty string
#endif

#define S                         (cpu->_S)
#define PC                        (cpu->_PC)
#define MEM                       (cpu->mem->Operand())
#define AR                        (cpu->mem->Address())
#define INPR              (cpu->_INPR)
#define OUTR              (cpu->_OUTR)
#define SetFGI(n)         (cpu->_SetFGI(n))
#define SetFGO(n)         (cpu->_SetFGO(n))
#define FGI                       (cpu->_GetFGI())
#define FGO                       (cpu->_GetFGO())
#define AC                                (((EX3_CPU *) cpu)->_AC)
#define E                                 (((EX3_CPU *) cpu)->_E)
#define R                                 (((EX3_CPU *) cpu)->_R)
#define IEN                               (((EX3_CPU *) cpu)->_IEN)
#define IMSK                      (((EX3_CPU *) cpu)->_IMSK)
#define IOT                               (((EX3_CPU *) cpu)->_IOT)
#define INPUT_PENDING     (((EX3_CPU *) cpu)->inputPending)

INSN(AND, 0x0000, 0, AC &= MEM;)
INSN(ADD, 0x1000, 0, int tmp = AC + MEM; E = (tmp >> 16) & 1; AC = (tmp & 0xffff);)
INSN(LDA, 0x2000, 0, AC = MEM;)
INSN(STA, 0x3000, 0, MEM = AC;)
INSN(BUN, 0x4000, 0, PC = AR;)
INSN(BSA, 0x5000, 0, MEM = PC; PC = AR + 1;)
INSN(ISZ, 0x6000, 1, MEM ++; if(MEM == 0)       PC = PC + 1;)
INSN(MUL, 0x10000, 0, ;)
INSN(DIV, 0x11000, 0, AC /= MEM;)

INSN(CLA, 0x7800, 0, AC = 0;)
INSN(CLE, 0x7400, 0, E = 0;)
INSN(CMA, 0x7200, 0, AC = ~AC;)
INSN(CME, 0x7100, 0, E = !E;)
INSN(CIR, 0x7080, 0, int tmp = E; E = AC & 1; AC = (AC >> 1) | (tmp << 15);)
INSN(CIL, 0x7040, 0, int tmp = E; E = (AC >> 15) & 1; AC = (AC << 1) | tmp;)
INSN(INC, 0x7020, 0, AC ++;)
INSN(SPA, 0x7010, 0, if((AC & 0x8000) == 0)      PC = PC + 1;)
INSN(SNA, 0x7008, 0, if(AC & 0x8000)             PC = PC + 1;)
INSN(SZA, 0x7004, 0, if(AC == 0)                         PC = PC + 1;)
INSN(SZE, 0x7002, 0, if(E == 0)                          PC = PC + 1;)
INSN(HLT, 0x7001, 0, S = 0;)

INSN(INP, 0xf800, 0, AC = (AC & ~0xff) | (INPR & 0xff); SetFGI(0); INPUT_PENDING = 0;)
INSN(OUT, 0xf400, 0, OUTR = AC & 0xff;                                SetFGO(0);)
INSN(SKI, 0xf200, 0, if(FGI)    PC = PC + 1; INPUT_PENDING = 1;)
INSN(SKO, 0xf100, 0, if(FGO)    PC = PC + 1;)
INSN(ION, 0xf080, 0, IEN = 1;)
INSN(IOF, 0xf040, 0, IEN = 0;)
INSN(SIO, 0xf020, 0, IOT = 1;)
INSN(PIO, 0xf010, 0, IOT = 0;)
INSN(IMK, 0xf008, 0, IMSK = AC & 0xf;)
INSN(SEG, 0xf004, 0, ;)

#undef S
#undef PC
#undef MEM
#undef AR
#undef INPR
#undef OUTR
#undef FGI
#undef FGO
#undef SetFGI
#undef SetFGO
#undef AC
#undef E
#undef R
#undef IEN
#undef IMSK
#undef IOT

#undef INSN
```

Verilog コード

ソースコード 5　cpu_ex3.v

```

`include "def_ex3.v"

module cpu_ex3 (clk, com_ctl, com_addr,
             fgi_bsy, fgi, inpr_in, /// input ports
             fgo_bsy, fgo, outr,    /// output ports
             ien, imsk, iot,
             s, r, e, bus_data, mem_data, dr, ac,
             ir, ar, pc, sc, sc_clr, seg);

     input          clk;
     input   [1:0]  com_ctl;
     input   [11:0] com_addr;

     input   [1:0]  fgi_bsy;
     output  [1:0]  fgi;
     input   [7:0]  inpr_in;

     input   [1:0]  fgo_bsy;
     output  [1:0]  fgo;
     output  [7:0]  outr;

     output         ien, iot;
     output  [3:0]  imsk;

     output         s, r, e;

     output  [31:0] mem_data, bus_data;
     output  [19:0] ir;
     output  [31:0] dr, ac;
     output  [11:0] ar, pc;
     output  [2:0]  sc;
     output         sc_clr;
     output  [15:0] seg;

     wire           com_rst    = (com_ctl == `COM_RST); /// reset (pc, sc, ar, 1-bit FFs)
     wire           com_start  = (com_ctl == `COM_RUN); /// release all resets
     wire           com_stop   = (com_ctl == `COM_STP); /// stop (freeze all)

     wire    [2:0]  bus_ctl;
     wire    [31:0] ac_nxt;

     wire           e_nxt, i15, r_nxt, ien_nxt, s_nxt, iot_nxt;
     wire    [3:0]  imsk_nxt;
     wire    [1:0]  fgi_nxt, fgo_nxt, fgi_set, fgo_set;
     wire    [15:0] seg_nxt;
         wire    [7:0]  inpr;

     wire           ar_ld, ar_inr, ar_clr;
     wire           pc_ld, pc_inr, pc_clr;
     wire           dr_ld, dr_inr, dr_clr;
     wire           ac_ld, ac_inr, ac_clr;
     wire           ac_and, ac_add, ac_dr, ac_inpr, ac_cmp, ac_shr, ac_shl, ac_mult,
         ac_div;
     wire           e_clr, e_cmp;

     wire           mem_we;

     wire           ir_ld;
     wire           outr_ld;
```

```
    wire [11:0]   ar_nxt;    /// ar_nxt is the value of ar at next clock (needed for
        synchronous RAM)
    wire [11:0]   mem_addr = (com_stop) ? com_addr : (mem_we) ? ar : ar_nxt;
    wire [31:0]   mem_din  = bus_data;

    ram_sync_4kx32 MEM   (clk, mem_we & ~com_stop, mem_addr, mem_din, mem_data);

    reg_lci_nxt #12 AR   (clk, ~com_stop, bus_data[11:0], ar, ar_nxt, ar_ld, ar_clr,
        ar_inr);
    reg_lci     #12 PC   (clk, ~com_stop, bus_data[11:0], pc, pc_ld | com_rst, pc_clr,
        pc_inr);

    reg_lci     #32 DR   (clk, ~com_stop, bus_data, dr, dr_ld, dr_clr, dr_inr);
    reg_lci     #32 AC   (clk, ~com_stop, ac_nxt,   ac, ac_ld, ac_clr, ac_inr);
    reg_lci     #20 IR   (clk, ~com_stop, bus_data[19:0], ir, ir_ld, 1'b0,   1'b0);

    reg_lci     #8  OUTR (clk, ~com_stop, bus_data[7:0], outr, outr_ld, com_rst, 1'b0);
    reg_dff     #8  INPR (clk, ~com_stop, inpr_in, inpr);

    reg_lci     #3  SC   (clk, ~com_stop, 3'b0, sc, 1'b0, sc_clr | com_rst, ~sc_clr);
        /// if(sc_clr == 0) sc ++;

    reg_dff     #1  I    (clk, ~com_stop, ir[15]                   , i15);
    reg_dff     #1  E    (clk, ~com_stop, e_nxt & ~com_rst         , e);    /// reset
        value = 0;
    reg_dff     #1  R    (clk, ~com_stop, r_nxt & ~com_rst         , r);    /// reset
        value = 0;
    reg_dff     #1  S    (clk, ~com_stop, s_nxt | com_rst          , s);    /// reset
        value = 1;
    reg_dff     #1  IEN  (clk, ~com_stop, ien_nxt & ~com_rst       , ien);  /// reset
        value = 0;
    reg_dff     #2  FGI  (clk, ~com_stop, fgi_nxt & {2{~com_rst}}  , fgi);  /// reset
        value = 00;
    reg_dff     #2  FGO  (clk, ~com_stop, fgo_nxt | {2{com_rst}}   , fgo);  /// reset
        value = 11;
    reg_dff     #1  IOT  (clk, ~com_stop, iot_nxt & ~com_rst       , iot);  /// reset
        value = 0;
    reg_dff     #4  IMSK (clk, ~com_stop, imsk_nxt & {4{~com_rst}} , imsk); /// reset
        value = 0000;
    reg_dff     #16 SEG  (clk, ~com_stop, seg_nxt & {16{~com_rst}} , seg);  /// reset
        value = 0x0000

        edge_to_pulse #(4,0) FGP (clk, {fgi_bsy, fgo_bsy}, {fgi_set, fgo_set});

    alu    ALU (dr, inpr, ac, e, ac_nxt, e_nxt, ac_and, ac_add, ac_dr, ac_inpr, ac_cmp,
         ac_shr, ac_shl, ac_mult, ac_div, e_clr, e_cmp);

    bus  #32  BUS (bus_ctl, {16'b0, `PROGRAM_ENTRY_POINT}, {20'b0, ar}, {20'b0, pc}, dr
        , ac, {12'b0, ir}, 32'b0, mem_data, bus_data);

    wire  [7:0]  t;
    wire  [15:0] d;
    dec_3to8  DEC_T  (sc, t, 1'b1);          /// (t[k] == 1) implies sc = k;
    dec_4to16  DEC_D  ({ir[16], ir[14:12]}, d, 1'b1);  /// (d[k] == 1) implies ir
        [14:12] = k;
                                             /// (d[7] == 1) implies non-memory-insn
                                                 type

    wire         rt = d[7] & ~i15 & t[3]; /// @ t[3] : implies register-insn type
    wire         pt = d[7] &  i15 & t[3]; /// @ t[3] : implies IO register-insn type

/// interrupt cycle:
/// r & t[0]  : ar <- 0;
/// r & t[1]  : mem[ar] <- pc; pc <- 0;
/// r & t[2]  : pc <- pc + 1; r <- 0; ien <- 0; sc <- 0;

/// instruction fetch cycle:
/// ~r & t[0] : ar <- pc;
/// ~r & t[1] : ir <- mem[ar]; pc <- pc + 1;
/// ~r & t[2] : ir15 <- ir[15]; ar <- ir[11:0];

/// indirect memory access:
/// ~d[7] & i15 & t[3] : ar <- mem[ar];

/// memory instruction cycle:
/// AND :   d[0] & t[4] : dr <- mem[ar];
///         d[0] & t[5] : ac <- ac & dr; sc <- 0;
/// ADD :   d[1] & t[4] : dr <- mem[ar];
///         d[1] & t[5] : ac <- ac + dr; sc <- 0;
/// LDA :   d[2] & t[4] : dr <- mem[ar];
///         d[2] & t[5] : ac <- dr;          sc <- 0;
/// STA :   d[3] & t[4] : mem[ar] <- ac; sc <- 0;
/// BUN :   d[4] & t[4] : pc <- ar;      sc <- 0;
/// BSA :   d[5] & t[4] : mem[ar] <- pc; ar <- ar + 1;
///         d[5] & t[5] : pc <- ar;      sc <- 0;
/// ISZ :   d[6] & t[4] : dr <- mem[ar];
///         d[6] & t[5] : dr <- dr + 1;
///         d[6] & t[6] : mem[ar] <- dr; (dr == 0) ? pc <- pc + 1; sc <- 0;

/// register instruction cycle:
/// rt = d[7] & ~i15 & t[3];            /// implies normal register-insn type
/// pt = d[7] &  i15 & t[3];            /// implies IO register-insn type

/// rt : sc <- 0;
/// pt : sc <- 0;

/// CLA :   rt & ir[11] : ac <- 0;
/// CLE :   rt & ir[10] :  e <- 0;
/// CMA :   rt & ir[9]  : ac <- ~ac;
/// CME :   rt & ir[8]  :  e <- ~e;
/// CIR :   rt & ir[7]  : ac[14:0] <- ac[15:1]; ac[15] <- e; e <- ac[0];
/// CIL :   rt & ir[6]  : ac[15:1] <- ac[14:0]; ac[0]  <- e; e <- ac[15];
/// INC :   rt & ir[5]  : ac <- ac + 1;
/// SPA :   rt & ir[4]  : (ac[15] == 0) ? pc <- pc + 1;
/// SNA :   rt & ir[3]  : (ac[15] == 1) ? pc <- pc + 1;
/// SZA :   rt & ir[2]  : (ac == 0)     ? pc <- pc + 1;
/// SZE :   rt & ir[1]  : (e == 0)      ? pc <- pc + 1;
/// HLT :   rt & ir[0]  : s <- 0;

/// INP :   pt & ir[11] : ac[7:0] <- inpr;
/// OUT :   pt & ir[10] : outr <- ac[7:0];
/// SKI :   pt & ir[9]  : (fgi[iot] == 1) ? pc <- pc + 1;
/// SKO :   pt & ir[8]  : (fgo[iot] == 1) ? pc <- pc + 1;
/// ION :   pt & ir[7]  : ien <- 1;
/// IOF :   pt & ir[6]  : ien <- 0;

/// SIO :   pt & ir[5]  : iot <- 1;
/// PIO :   pt & ir[4]  : iot <- 0;
/// IMK :   pt & ir[3]  : imsk <- ac[3:0];
/// SEG :   pt & ir[2]  : seg <- ac

    assign mem_we   =    r & t[1] |   /// interrupt @ t[1] : mem[ar] <- pc;
                      d[3] & t[4] |   /// STA @ t[4]       : mem[ar] <- ac;
                      d[5] & t[4] |   /// BSA @ t[4]       : mem[ar] <- pc;
                      d[6] & t[6];    /// ISZ @ t[6]       : mem[ar] <- dr;

    assign ac_and   = d[0] & t[5];    /// AND @ t[5] : ac <- ac & dr;
    assign ac_add   = d[1] & t[5];    /// ADD @ t[5] : ac <- ac + dr;
    assign ac_dr    = d[2] & t[5];    /// LDA @ t[5] : ac <- dr;
    assign ac_mult  = d[8] & t[5];    /// MUL @ t[5] : ac <- ac * dr;
    assign ac_div   = d[9] & t[5];    /// DIV @ t[5] : ac <- ac / dr;
    assign ac_inpr  = pt & ir[11];    /// INP : ac[7:0] <- inpr;
    assign ac_cmp   = rt & ir[9];     /// CMA : ac <- ~ac;
    assign ac_shr   = rt & ir[7];     /// CIR : ac[14:0] <- ac[15:1], ac[15] <- e, e_nxt
         <- ac[0];
    assign ac_shl   = rt & ir[6];     /// CIL : ac[15:1] <- ac[14:0], ac[0]  <- e, e_nxt
         <- ac[15];

/// ac_ld : all of above operations
    assign ac_ld    = ac_and | ac_add | ac_dr | ac_inpr | ac_cmp | ac_shr | ac_shl |
        ac_mult | ac_div;
    assign ac_inr   = rt & ir[5];     /// INC : ac <- ac + 1;
    assign ac_clr   = rt & ir[11];    /// CLA : ac <- 0;

    assign e_clr    = rt & ir[10];    /// CLE : e <- 0;
    assign e_cmp    = rt & ir[8];     /// CME : e <- ~e

    assign ar_ld    = ~r & t[0]          |   /// fetch @ t[0] : ar <- pc;
                      ~r & t[2]          |   /// memory address : ar <- ir[11:0];
                      ~d[7] & i15 & t[3];    /// indirect memory access : ar <- mem[ar];
    assign ar_inr   = d[5] & t[4];           /// BSA @ t[4] : mem[ar] <- pc, ar <- ar +
        1;
    assign ar_clr   = r & t[0];              /// interrupt @ t[0] : ar <- 0;

    assign pc_ld    = d[4] & t[4] |                 /// BUN @ t[4] : pc <- ar;
                      d[5] & t[5];                  /// BSA @ t[5] : pc <- ar;
    assign pc_inr   = ~r & t[1]               |     /// fetch @ t[1]
```

```
                     r & t[2]                    |   /// interrupt @ t[2]
                     d[6] & t[6] & (dr == 16'b0)|   /// ISZ @ t[6] & (dr == 0)
                     rt & ir[4] & ~ac[15]        |   /// SPA @ (ac[15] == 0)
                     rt & ir[3] & ac[15]         |   /// SNA @ (ac[15] == 1)
                     rt & ir[2] & (ac == 16'b0) |   /// SZA @ (ac == 0)
                     rt & ir[1] & ~e             |   /// SZE @ (e == 0)
                     pt & ir[9] & fgi[iot]       |   /// SKI @ (fgi[iot] == 1)
                     pt & ir[8] & fgo[iot];          /// SKO @ (fgo[iot] == 1)
    assign pc_clr  = r & t[1];                       /// interrupt @ t[1]

    assign dr_ld   = d[0] & t[4] |  /// AND @ t[4] : dr <- mem[ar];
                     d[1] & t[4] |  /// ADD @ t[4] : dr <- mem[ar];
                     d[2] & t[4] |  /// LDA @ t[4] : dr <- mem[ar];
                     d[8] & t[4] |  /// MUL @ t[4] : dr <- mem[ar];
                     d[9] & t[4] |  /// DIV @ t[4] : dr <- mem[ar];
                     d[6] & t[4];   /// ISZ @ t[4] : dr <- mem[ar];
    assign dr_inr  = d[6] & t[5];   /// ISZ @ t[5] : dr <- dr + 1;
    assign dr_clr  = 0;

    assign sc_clr  = r & t[2]     |                                              /// end
        of interrupt cycle
                     d[0] & t[5] | d[1] & t[5] | d[2] & t[5] | d[3] & t[4] |   /// end
                         of AND/ADD/LDA/STA
                     d[4] & t[4] | d[5] & t[5] | d[6] & t[6] | d[8] & t[5] |
                     d[9] & t[5] |                   /// end of BUN/BSA/ISZ
                     rt | pt |                                                  /// end
                         of reg-insn/io-insn
                     ~s;                                                        /// halt

    assign ir_ld   = ~r & t[1];      /// fetch @ t[1]

    assign outr_ld = pt & ir[10]; /// OUT : outr <- ac[7:0]

    wire bus_ar    = d[4] & t[4]          |   /// BUN @ t[4] : pc <- ar;
                     d[5] & t[5];             /// BSA @ t[5] : pc <- ar;
    wire bus_pc    = ~r & t[0] & ~com_rst |   /// fetch @ t[0] : ar <- pc; (com_rst =
        0)
                     r & t[1]             |   /// interrupt @ t[1] : mem[ar] <- pc;
                     d[5] & t[4];             /// BSA @ t[4] : mem[ar] <- pc;
    wire bus_dr    = d[6] & t[6];             /// ISZ @ t[6] : mem[ar] <- dr;
    wire bus_ac    = d[3] & t[4]          |   /// STA @ t[4] : mem[ar] <- ac;
                     pt & ir[10];             /// OUT : outr <- ac[7:0]
    wire bus_ir    = ~r & t[2];               /// fetch @ t[2] : ar <- ir[11:0];
    wire bus_mem   = ~r & t[1]            |   /// fetch @ t[1] : ir <- mem[ar];
                     ~d[7] & i15 & t[3]   |   /// indirect : ar <- mem[ar];
                     d[0] & t[4]          |   /// AND @ t[4] : dr <- mem[ar];
                     d[1] & t[4]          |   /// ADD @ t[4] : dr <- mem[ar];
                     d[8] & t[4]          |   /// MUL @ t[4] : dr <- mem[ar];
                     d[9] & t[4]          |   /// DIV @ t[4] : dr <- mem[ar];
                     d[2] & t[4]          |   /// LDA @ t[4] : dr <- mem[ar];
                     d[6] & t[4];             /// ISZ @ t[4] : dr <- mem[ar];

    assign bus_ctl[0]  = bus_ar | bus_dr | bus_ir | bus_mem;  /// b1 | b3 | b5 | b7
    assign bus_ctl[1]  = bus_pc | bus_dr | bus_mem;          /// b2 | b3 | b7
    assign bus_ctl[2]  = bus_ac | bus_ir | bus_mem;          /// b4 | b5 | b7

    wire   detect_intr = ~t[0] & ~t[1] & ~t[2] & ien &
                         ((fgi[1] & imsk[3]) | (fgo[1] & imsk[2]) |
                         (fgi[0] & imsk[1]) | (fgo[0] & imsk[0]));
    assign r_nxt   = (detect_intr) ? 1 :  /// detect interrupt
                     (r & t[2])    ? 0 :  /// disable interrupt
                     r;                   /// unchanged

    assign ien_nxt = (pt & ir[7]) ? 1                 :  /// ION
                            : ien <- 1
                     ((pt & ir[6]) | (r & t[2])) ? 0  :  /// IOF | (end of interrupt
                         cycle) : ien <- 0
                     ien;                                 /// unchanged

    assign fgi_nxt[0] = (fgi_set[0]) ? 1          :  /// fgi_set[0] : fgi[0] <- 1
                        (pt & ir[11] & ~iot) ? 0 :  /// INP        : fgi[0] <- 0
                        fgi[0];                      /// unchanged
    assign fgi_nxt[1] = (fgi_set[1]) ? 1          :  /// fgi_set[1] : fgi[1] <- 1
                        (pt & ir[11] &  iot) ? 0 :  /// INP        : fgi[1] <- 0
                        fgi[1];                      /// unchanged

    assign fgo_nxt[0] = (fgo_set[0]) ? 1          :  /// fgo_set[0] : fgo[0] <- 1
                        (pt & ir[10] & ~iot) ? 0 :  /// OUT        : fgo[0] <- 0
                        fgo[0];                      /// unchanged
    assign fgo_nxt[1] = (fgo_set[1]) ? 1          :  /// fgo_set[1] : fgo[1] <- 1
                        (pt & ir[10] &  iot) ? 0 :  /// OUT        : fgo[1] <- 0
                        fgo[1];                      /// unchanged

    assign iot_nxt = (pt & ir[5]) ? 1 :              /// SIO : iot  <- 1
                         (pt & ir[4]) ? 0 :              /// PIO : iot  <- 0
                     iot;                            /// unchanged

    assign imsk_nxt = (pt & ir[3]) ? ac[3:0] : imsk;  /// IMK : imsk <- ac[3:0]

    assign seg_nxt = (pt & ir[2]) ? ac[15:0] : seg;       /// SEG : seg <- ac

    assign s_nxt = (rt & ir[0]) ? 0 : s;             /// HLT : s <- 0;

endmodule
```

ソースコード 6　cpu_module.v

```

`include "def_ex3.v"



/*********************** synchronous ROM model ************************/

module rom_sync_4kx32 (clk, addr, dout);

```

```
/// mem_size = 4096;
/// addr_width = 12;
/// data_width = 32;

        input  clk;         /// clock
        input  [11:0] addr; /// address input
        output [31:0] dout; /// data output

        reg  [31:0] dout, mem[0:4095];

        initial $readmemh(`PROB_FILE, mem);

        always @ (posedge clk) begin
                dout <= mem[addr];
        end
endmodule

/*********************** synchronous RAM model ************************/

module ram_sync_4kx32 (clk, we, addr, din, dout);

/// mem_size = 4096;
/// addr_width = 12;
/// data_width = 16;

    input  clk;         /// clock
    input  we;          /// write enable
    input  [11:0] addr; /// address input
    input  [31:0] din;  /// data input
    output [31:0] dout; /// data output

    reg    [31:0] dout, mem[0:4095];

    initial $readmemh(`MEM_INIT_FILE, mem);

    always @ (posedge clk) begin
        if(we) mem[addr] <= din;
        dout <= mem[addr];
    end
endmodule


/*********************** load/clear/increment/clock-enable register model
    ************************/

module reg_lci (clk, en, din, dout, ld, clr, inr);

parameter DATA_WIDTH = 16;

    input clk;          /// clock
    input en;           /// clock enable
    input ld;           /// load
    input clr;          /// clear
    input inr;          /// increment
    input  [DATA_WIDTH - 1:0] din;   /// data input
    output [DATA_WIDTH - 1:0] dout;  /// data output

    wire   [DATA_WIDTH - 1:0] dout_nxt; /// not connected to output

    reg_lci_nxt #(DATA_WIDTH) R0 (clk, en, din, dout, dout_nxt, ld, clr, inr);

endmodule

/// reg_lci_nxt has same functionality as reg_lci, but also outputs dout_nxt (dout at
    next clock)

module reg_lci_nxt (clk, en, din, dout, dout_nxt, ld, clr, inr);

parameter DATA_WIDTH = 16;

    input clk;          /// clock
    input en;           /// clock enable
    input ld;           /// load
    input clr;          /// clear
    input inr;          /// increment
    input  [DATA_WIDTH - 1:0] din;      /// data input
    output [DATA_WIDTH - 1:0] dout;     /// data output
    output [DATA_WIDTH - 1:0] dout_nxt; /// data output at the next cycle

    assign dout_nxt = (clr) ? 0 : (ld) ? din : (inr) ? dout + 1 : dout;

    reg_dff #(DATA_WIDTH) R0 (clk, en, dout_nxt, dout);
endmodule

/*********************** clock-enable register model ************************/

module reg_dff (clk, en, din, dout);

parameter DATA_WIDTH = 16;

    input clk;                        /// clock
    input en;                         /// enable
    input  [DATA_WIDTH - 1:0] din;  /// data input
    output [DATA_WIDTH - 1:0] dout; /// data output

    reg    [DATA_WIDTH - 1:0] dout;

    always @ (posedge clk)
        if(en) dout <= din;           /// update dout only when (en == 1)
endmodule

/*********************** edge-to-pulse converter ************************/

module edge_to_pulse (clk, din, dout);

    parameter DATA_WIDTH = 1;
```

```
        parameter EDGE_TYPE  = 0;  /// 0 : negative-edge (1->0), 1 : positive-edge
            (0->1)

    input                          clk;
        input   [DATA_WIDTH - 1:0]   din;
    output  [DATA_WIDTH - 1:0]   dout;

    reg     [DATA_WIDTH - 1:0]   prev_din2;

        wire    [DATA_WIDTH - 1:0]   din2 = (EDGE_TYPE == 1) ? din : ~din;
    always @ (posedge clk) prev_din2 <= din2;

    assign dout = ~prev_din2 & din2;

endmodule

/*********************** 8-master bus model ************************/

module bus (bus_ctl, b0, b1, b2, b3, b4, b5, b6, b7, bout);

parameter DATA_WIDTH = 16;

    input  [2:0]  bus_ctl;
    input  [DATA_WIDTH - 1:0] b0, b1, b2, b3, b4, b5, b6, b7;
    output [DATA_WIDTH - 1:0] bout;

    reg    [DATA_WIDTH - 1:0] bout; /// bout is reg-type but is actually combinational

    always @ (bus_ctl or b0 or b1 or b2 or b3 or b4 or b5 or b6 or b7) begin
        case (bus_ctl)
            3'b000 : bout = b0;
            3'b001 : bout = b1;
            3'b010 : bout = b2;
            3'b011 : bout = b3;
            3'b100 : bout = b4;
            3'b101 : bout = b5;
            3'b110 : bout = b6;
            3'b111 : bout = b7;
        endcase
    end
endmodule

/*********************** ALU model ************************/

module alu (dr, inpr, ac, e, ac_nxt, e_nxt, ac_and, ac_add, ac_dr, ac_inpr, ac_cmp,
    ac_shr, ac_shl, ac_mult, ac_div, e_clr, e_cmp);

    input  [31:0] dr, ac;
    input  [7:0]  inpr;
    input         e, ac_and, ac_add, ac_dr, ac_inpr, ac_cmp, ac_shr, ac_shl, ac_mult,
        ac_div, e_clr, e_cmp;

    output [31:0] ac_nxt;
    output        e_nxt;

    wire   [31:0] o_and  = (ac_and)  ? (dr & ac)                 : 32'b0;
    wire   [32:0] o_add  = (ac_add)  ? ({1'b0, dr} + {1'b0, ac}) : 33'b0;
    wire   [31:0] o_dr   = (ac_dr)   ? (dr)                      : 32'b0;
    wire   [31:0] o_inpr = (ac_inpr) ? ({ac[31:8], inpr})        : 32'b0;
    wire   [31:0] o_cmp  = (ac_cmp)  ? (~ac)                     : 32'b0;
    wire   [31:0] o_shr  = (ac_shr)  ? ({e, ac[31:1]})           : 32'b0;
    wire   [31:0] o_shl  = (ac_shl)  ? ({ac[30:0], e})           : 32'b0;
    wire   [31:0] o_mul  = (ac_mult) ? (ac * dr)                 : 32'b0;
    wire   [31:0] o_div  = (ac_div)  ? (ac / dr)                 : 32'b0;

    assign ac_nxt = o_and | o_add[31:0] | o_dr | o_inpr | o_cmp | o_shr | o_shl | o_mul
         | o_div;

    assign e_nxt =  (ac_add) ? o_add[32] :
                    (ac_shr) ? ac[0]     :
                    (ac_shl) ? ac[31]    :
                    (e_clr)  ? 1'b0      :
                    (e_cmp)  ? ~e        : e;
endmodule

/*********************** 2to4 decoder model ************************/

module dec_2to4 (din, dout, en);

    input  [1:0] din;
    input        en;
    output [3:0] dout;

    assign dout[0]  = en & (din == 2'b00);
    assign dout[1]  = en & (din == 2'b01);
    assign dout[2]  = en & (din == 2'b10);
    assign dout[3]  = en & (din == 2'b11);
endmodule

/*********************** 3to8 decoder model ************************/

module dec_3to8 (din, dout, en);

    input  [2:0] din;
    input        en;
    output [7:0] dout;

    dec_2to4 D0 (din[1:0], dout[3:0], en & ~din[2]);
    dec_2to4 D1 (din[1:0], dout[7:4], en & din[2]);
endmodule

/*********************** 4to16 decoder model ************************/

module dec_4to16 (din, dout, en);

    input  [3:0] din;
    input        en;
    output [15:0] dout;

    dec_3to8 D0 (din[2:0], dout[7:0], en & ~din[3]);
    dec_3to8 D1 (din[2:0], dout[15:8], en & din[3]);
endmodule

/*********************** 7-segment encoder/decoder ************************/

module seg7_enc (data, enc_out);
    input  [3:0] data;
    output [6:0] enc_out;
    reg    [6:0] enc_out_n;

    assign enc_out = ~enc_out_n;

    always @(data)
    case(data)
        4'h0:    enc_out_n = `SEG_7_0;
        4'h1:    enc_out_n = `SEG_7_1;
        4'h2:    enc_out_n = `SEG_7_2;
        4'h3:    enc_out_n = `SEG_7_3;
        4'h4:    enc_out_n = `SEG_7_4;
        4'h5:    enc_out_n = `SEG_7_5;
        4'h6:    enc_out_n = `SEG_7_6;
        4'h7:    enc_out_n = `SEG_7_7;
        4'h8:    enc_out_n = `SEG_7_8;
        4'h9:    enc_out_n = `SEG_7_9;
        4'ha:    enc_out_n = `SEG_7_A;
        4'hb:    enc_out_n = `SEG_7_B;
        4'hc:    enc_out_n = `SEG_7_C;
        4'hd:    enc_out_n = `SEG_7_D;
        4'he:    enc_out_n = `SEG_7_E;
        4'hf:    enc_out_n = `SEG_7_F;
        default: enc_out_n = `SEG_7_UNDEFINED;
    endcase
endmodule

module seg7_dec (hex, dec_out);
    input  [6:0] hex;
    output [4:0] dec_out;
    reg    [4:0] dec_out;

    always @(hex)
    case(~hex)
        `SEG_7_0 : dec_out = 5'h00;
        `SEG_7_1 : dec_out = 5'h01;
        `SEG_7_2 : dec_out = 5'h02;
        `SEG_7_3 : dec_out = 5'h03;
        `SEG_7_4 : dec_out = 5'h04;
        `SEG_7_5 : dec_out = 5'h05;
        `SEG_7_6 : dec_out = 5'h06;
        `SEG_7_7 : dec_out = 5'h07;
        `SEG_7_8 : dec_out = 5'h08;
        `SEG_7_9 : dec_out = 5'h09;
        `SEG_7_A : dec_out = 5'h0a;
        `SEG_7_B : dec_out = 5'h0b;
        `SEG_7_C : dec_out = 5'h0c;
        `SEG_7_D : dec_out = 5'h0d;
        `SEG_7_E : dec_out = 5'h0e;
        `SEG_7_F : dec_out = 5'h0f;
        default  : dec_out = 5'h1f;     ///     error
    endcase
endmodule

/*********************** UART parameter ***********************/

    /// CLK_FREQ = 27000000;              /// 27MHz
    /// BAUD_RATE = 9600;                 /// 9.6KHz

/*********************** UART parity generator ************************/

module uart_parity (xor_parity, parity_bit);

    input        xor_parity;
    output       parity_bit;

    assign parity_bit    = (`UART_PARITY_DEFAULT == `UART_PARITY_ODD) ? ~xor_parity :
                           (`UART_PARITY_DEFAULT == `UART_PARITY_EVEN) ? xor_parity :
                           (`UART_PARITY_DEFAULT == `UART_PARITY_STICK_1) ? 1       :
                           (`UART_PARITY_DEFAULT == `UART_PARITY_STICK_0) ? 0       :
                                                   1'bx;
endmodule

/*********************** UART RX model ***********************/

module uart_rx (clk, reset, rx_din, rx_byte_out, rx_error, rx_rdy);

    parameter      ID0 = 0;
    wire[2:0]      ID = ID0;

    input          clk, reset;
    input          rx_din;      /// serial input
    output  [7:0]  rx_byte_out;
    output         rx_error, rx_rdy;

    reg     [7:0]  rx_byte_out;
    reg            rx_error, rx_rdy;

    parameter HALF_BAUD_PHASE = 2;              /// 2 phases per 0.5 bit
    parameter ONE_BAUD_PHASE = HALF_BAUD_PHASE * 2;  /// 4 phases per 1 bit
    parameter BAUD_PERIOD = 703; /* 703.125 */  /// = CLK_FREQ / (BAUD_RATE *
        ONE_BAUD_PHASE)

    reg            rx_start;                    /// 0 : IDLE, 1 : RX_ACTIVE
    reg     [1:0]  rx_din_synced;               /// synchronized rx_din (using 2 DFFs)
    reg     [1:0]    baud_phase;                       /// 0 to ONE_BAUD_PHASE - 1
    reg     [`UART_BIT_COUNT - 1:0]   rx_shift;     /// start-bit, 8-bit data, parity-bit,
         end-bit
    reg     [9:0]    baud_tick;                        /// 0 to BAUD_PERIOD - 1
    reg     [3:0]    bit_count;                        /// 0 to BIT_COUNT - 1

    wire    rx_xor_parity = ^rx_byte_out;              /// XOR all bits in rx_byte_out[7:0]
    wire    rx_parity;
    uart_parity UP (rx_xor_parity, rx_parity);

    wire    rx_parity_error = `UART_PARITY_ON & (rx_parity != rx_shift[1]);
    wire    [7:0] rx_byte = rx_shift[`UART_BIT_COUNT - 3 : `UART_BIT_COUNT - 10];

    always@(posedge clk)
        ////////////// reset //////////////
        if(reset) begin
            rx_start       <= 0;
            rx_byte_out    <= 0;
            rx_rdy         <= 0;
            rx_din_synced  <= 2'b11;
            rx_error       <= 0;
            baud_phase     <= 2'b0;
            rx_shift       <= 0;
            baud_tick      <= 10'b0;
            bit_count      <= 4'b0;
        end
        else begin
            rx_din_synced <= {rx_din, rx_din_synced[1]};  /// rx_din synchronization (
                right shift)
            if(~rx_start) begin
                if(~rx_din_synced[0]) begin  /// start bit = 0
                    rx_start  <= 1;           /// start receiving data
                    rx_rdy  <= 0;             /// now busy...
`ifdef ENABLE_UART_MONITORING
                    if(ID == 0) $display($stime, " : UART_RX[%d] (start!)", ID);
`endif
                end
            end
            ////////////// baud_tick : count until BAUD_PERIOD - 1 //////////////
            else if(baud_tick < BAUD_PERIOD - 1)
                baud_tick    <= baud_tick + 1;
            else begin  /// end of baud cycle
                baud_tick    <= 10'b0;
                ////////////// baud_phase : count until ONE_BAUD_PHASE - 1
                    //////////////
                if(baud_phase == HALF_BAUD_PHASE - 1) /// sample rx_din_synced[0] at
                    middle of baud_phase
                    rx_shift  <= {rx_din_synced[0], rx_shift[`UART_BIT_COUNT - 1 : 1]};
                        /// right shift, MSB = rx_din_synced[0]
                if(baud_phase < ONE_BAUD_PHASE - 1)
                    baud_phase <= baud_phase + 1;
                else begin  /// end of baud phase
                    baud_phase <= 2'b0;
                    ////////////// bit_count : count until UART_BIT_COUNT - 1
                        //////////////
                    if(bit_count < `UART_BIT_COUNT - 1)
                        bit_count <= bit_count + 1;
                    else begin /// all bits shifted
                        bit_count    <= 4'b0;
                        rx_rdy       <= 1;
                        rx_byte_out  <= rx_byte;
                        rx_error     <= ~rx_shift[`UART_BIT_COUNT - 1] | rx_parity_error
                            ; /// end-bit must be 1
                        rx_start     <= 0; /// goto IDLE state
`ifdef ENABLE_UART_MONITORING
                        if(ID == 0)
                            $display($stime, " : UART_RX[%d] (stop!!) : rx_shift = %b_%
                                b_%b (%h:%s)",
                                ID, rx_shift[`UART_BIT_COUNT - 1:`UART_BIT_COUNT - 2],
                                rx_byte, rx_shift[0], rx_byte, `GET_CHAR(rx_byte));
`endif
                    end
                end
            end
        end
    /// end always
endmodule

/*********************** UART TX model ***********************/

module uart_tx (clk, reset, tx_dout, tx_byte_in, fgo, tx_rdy);

    parameter       ID0 = 0;
    wire[2:0]       ID = ID0;

    input           clk, reset;
    output          tx_dout;        /// serial input
    input  [7:0]    tx_byte_in;
    input           fgo;
    output          tx_rdy;

    parameter BAUD_PERIOD = 2812; /* 2812.5 */   /// = CLK_FREQ / BAUD_RATE

    reg             tx_start;                     /// 0 : IDLE, 1 : TX_ACTIVE
    reg     [`UART_BIT_COUNT - 1:0]  tx_shift;    /// start-bit, 8-bit data, parity-bit,
        end-bit
    reg     [11:0]  baud_tick;                    /// 0 to BAUD_PERIOD - 1
    reg     [3:0]   bit_count;                    /// 0 to BIT_COUNT - 1

    assign tx_dout = tx_shift[0];
    assign tx_rdy = ~tx_start & ~reset;
    wire   tx_xor_parity = ^tx_byte_in;           /// XOR all bits in tx_byte_in[7:0]
    wire   tx_parity;
    uart_parity UP (tx_xor_parity, tx_parity);

        wire   tx_en;
        edge_to_pulse #(1,0) TX_EN (clk, fgo, tx_en);

    always@(posedge clk)
        /////////////// reset ///////////////
        if(reset) begin
            tx_start      <= 0;
            tx_shift      <= {(`UART_BIT_COUNT){1'b1}}; /// all 1s
            baud_tick     <= 12'b0;
            bit_count     <= 4'b0;
        end
        else begin
            if(~tx_start) begin
                if(tx_en) begin
                    tx_start <= 1;
                    tx_shift <= {1'b1, tx_parity, tx_byte_in, 1'b0};
`ifdef ENABLE_UART_MONITORING
                    if(ID == 0) $display($stime, " : UART_TX[%d] (start!) : tx_shift =
                        %b_%b_%b (%h:%s)",
                             ID, {1'b1, tx_parity}, tx_byte_in, 1'b0, tx_byte_in,
                                 `GET_CHAR(tx_byte_in));
`endif
                end
            end
            /////////////// baud_tick : count until BAUD_PERIOD - 1 ///////////////
            else if(baud_tick < BAUD_PERIOD - 1)
                baud_tick    <= baud_tick + 1;
            else begin
                baud_tick    <= 10'b0;
                tx_shift <= {1'b1, tx_shift[`UART_BIT_COUNT - 1 : 1]}; /// right shift,
                     MSB = 1
                /////////////// bit_count : count until UART_BIT_COUNT - 1
                    ///////////////
                if(bit_count < `UART_BIT_COUNT - 1)
                    bit_count <= bit_count + 1;
                else begin /// all bits shifted
                    bit_count <= 4'b0;
                    tx_start <= 0; /// goto IDLE state
`ifdef ENABLE_UART_MONITORING
                    if(ID == 0) $display($stime, " : UART_TX[%d] (stop!!)", ID);
`endif
                end
            end
        end
    /// end always
endmodule
```

ソースコード 7　def_ex3.v

```

`define PROGRAM_ENTRY_POINT     16'h0010

//`define ENABLE_CPU_MONITORING
```

```
//`define ENABLE_MONITOR_FILE
//`define ENABLE_UART_MONITORING

/// simulation stops at these values
/// modify this if your program runs longer than this...
`define MAX_CYCLE       32'hb00000
`define MAX_INSN_COUNT  32'h400000

/*********************** Input files ************************/

`define PROGRAM_NAME        "test_calc2"
`define MEM_INIT_FILE       {`PROGRAM_NAME, ".mem"}
`define PROB_FILE           {`PROGRAM_NAME, ".prb"}
`define MONITOR_FILE        {`PROGRAM_NAME, ".mon"}
`define INPUT_VECTOR_FILE   {`PROGRAM_NAME, "_in.log"}
`define OUTPUT_VECTOR_FILE  {`PROGRAM_NAME, "_out.log"}

`define MAX_FILENAME_LENGTH 20

`define INPUT_VECTOR_SIZE  256
`define OUTPUT_VECTOR_SIZE 1024 /// 256

`define MONITOR_VECTOR_SIZE  256

/************************************************************/

/*********************** CPU control ************************/
`define COM_RST 2'b00
`define COM_RUN 2'b01
`define COM_STP 2'b10

/*********************** memory probe ***********************/
`define MEM_DATA  4'h0
`define MEM_END   4'hf

/*********************** 7-segment LED ***********************
7-segment LED bit assignment :
      00000
     5     1
     5     1
     5     1
      66666
     4     2
     4     2
     4     2
      33333
*************************************************************/
`define SEG_7_0 7'h3f                   ///      543210 : 0
`define SEG_7_1 7'h06                   ///          21  : 1
`define SEG_7_2 7'h5b                   ///     6 43 10 : 2
`define SEG_7_3 7'h4f                   ///     6  3210 : 3
`define SEG_7_4 7'h66                   ///     65   21  : 4
`define SEG_7_5 7'h6d                   ///     65 32 0 : 5
```

```
`define SEG_7_6 7'h7d                   ///     65432 0 : 6
`define SEG_7_7 7'h07                   ///         210 : 7
`define SEG_7_8 7'h7f                   ///     6543210 : 8
`define SEG_7_9 7'h6f                   ///     65 3210 : 9
`define SEG_7_A 7'h77                   ///     654 210 : A
`define SEG_7_B 7'h7c                   ///     65432   : b
`define SEG_7_C 7'h39                   ///      543  0 : C
`define SEG_7_D 7'h5e                   ///     6 4321  : d
`define SEG_7_E 7'h79                   ///     6543  0 : E
`define SEG_7_F 7'h71                   ///     654   0 : F
`define SEG_7_UNDEFINED 7'h40   ///     6       : -

/*********************** FPGA SW options ************************/
///////////       SW[9:8] : system GPIO mode       //////////////
`define G0  2'b00   /// disable GPIO
`define G1  2'b01   /// enable GPIO (GPIO[35:18] = GP_OUT[17:0], GPIO[17:0] = GP_IN
    [17:0])
`define G2  2'b10   /// enable GPIO (GPIO[35:18] = GP_IN[17:0],  GPIO[17:0] = GP_OUT
    [17:0])
///////////       SW[7:6] : system operation mode  //////////////
`define M0  2'b00   /// run continuous
`define M1  2'b01   /// run until interrupt
`define M2  2'b10   /// step single clock
`define M3  2'b11   /// step single insn
///////////       SW[5:2] : system probe mode       //////////////
`define P0  4'b0000 /// probe ac[15:0]
`define P1  4'b0001 /// probe e[0]
`define P2  4'b0010 /// probe pc[11:0]
`define P3  4'b0011 /// probe ir[15:0]
`define P4  4'b0100 /// probe ar[11:0]
`define P5  4'b0101 /// probe dr[15:0]
`define P6  4'b0110 /// probe mem_dout[15:0]
`define P7  4'b0111 /// probe bus[15:0]
`define P8  4'b1000 /// probe sc[3:0]
`define P9  4'b1001 /// probe {iosel, ien, imask[1:0]}
`define P10 4'b1010 /// probe {fgi, inpr[7:0]}
`define P11 4'b1011 /// probe {fgo, outr[7:0]}

/*********************** UART parity options ************************/
`define UART_PARITY_NONE      3'b000
`define UART_PARITY_ODD       3'b001   /// XNOR all bits (1 if even number of 1s)
`define UART_PARITY_EVEN      3'b011   /// XOR  all bits (1 if odd number of 1s)
`define UART_PARITY_STICK_1   3'b101   /// 1
`define UART_PARITY_STICK_0   3'b111   /// 0

`define UART_PARITY_DEFAULT       `UART_PARITY_EVEN  /// change this to switch to other
     parity types
`define UART_PARITY_ON        (`UART_PARITY_DEFAULT != `UART_PARITY_NONE)
`define UART_BIT_COUNT        (1 + 8 + `UART_PARITY_ON + 1) /// start-bit, 8-bit data (
    LSB-first!), parity-bit, end-bit

`define GET_CHAR(ch)          ((ch != 8'h0a) ? { 8'h20, ch } : { 8'h5c, 8'h6e })
```

ソースコード 8　fpga_ex3.v

```
`include "def_ex3.v"

module fpga_ex3
    (
    ////////////////////        Clock Input             ////////////////////
    CLOCK_27,                       //  27 MHz
    ////////////////////        Push Button             ////////////////////
    KEY,                            //  Pushbutton[3:0]
    ////////////////////        DPDT Switch             ////////////////////
    SW,                             //  Toggle Switch[9:0]
   //////////////////// 7-SEG Display   ////////////////////
    HEX0,                           //  Seven Segment Digit 0
    HEX1,                           //  Seven Segment Digit 1
    HEX2,                           //  Seven Segment Digit 2
    HEX3,                           //  Seven Segment Digit 3
    ////////////////////////    LED             ////////////////////////
    LEDG,                           //  LED Green[7:0]
    LEDR,                           //  LED Red[9:0]
    ////////////////////////    UART    ////////////////////////
    UART_TXD,                       //  UART Transmitter
    UART_RXD,                       //  UART Receiver
    ////////////////////        GPIO    ////////////////////////////
    GPIO_0,                         //  GPIO Connection 0
    GPIO_1                          //  GPIO Connection 1
    );

    input           CLOCK_27;               //  27 MHz
    input   [3:0]   KEY;                    //  Pushbutton[3:0]
    input   [9:0]   SW;                     //  Toggle Switch[9:0]
    output  [6:0]   HEX0, HEX1, HEX2, HEX3; //  Seven Segment Digit 0,1,2,3
    output  [7:0]   LEDG;                   //  LED Green[7:0]
    output  [9:0]   LEDR;                   //  LED Red[9:0]
    output          UART_TXD;               //  UART Transmitter
    input           UART_RXD;               //  UART Receiver

    inout   [35:0]  GPIO_0; // (inpr, outr, fgi_set_n, fgo_set_n, fgi, fgo)
    output  [35:0]  GPIO_1; // (used for cpu monitoring)

    wire    [3:0]   gm, sm;
    wire    [15:0]  sp;

    dec_2to4  DEC_G (SW[9:8], gm, 1'b1);
    dec_2to4  DEC_M (SW[7:6], sm, 1'b1);
    dec_4to16 DEC_P (SW[5:2], sp, 1'b1);
    wire            intr_detect_fgi = SW[1];
    wire            intr_detect_fgo = SW[0];


    wire    [17:0]  sys_probe_data;
    assign LEDG = sys_probe_data[7:0];
    assign LEDR = sys_probe_data[17:8];

    wire            clk = CLOCK_27;

    wire    [17:0]  GP_IN;  //  GPIO Input 0
    wire    [17:0]  GP_OUT; //  GPIO Output 0

    wire    [3:0]   PREV_KEY;
    wire    [1:0]   cpu_state;
    wire    [11:0]  com_addr_reg, probe_idx;
    wire    [31:0]  probe_info;
    wire    insn_end, intr_detected, halted;

    wire    [7:0]   inpr, outr, sin_byte, pin_byte;
    wire            sout_rdy, sin_error, sin_rdy, pout_bsy, pin_bsy;
    wire    [1:0]   fgi, fgi_bsy, fgo, fgo_bsy;
    wire    [31:0]  bus, mem;
    wire    [19:0]  ir;
    wire    [31:0]  dr, ac;
    wire    [15:0]  seg;
    wire    [11:0]  ar, pc;
    wire    [2:0]   sc;
    wire            s, r, e, ien, sc_clr, iot;
    wire    [3:0]   imsk;

    rom_sync_4kx32 ROM (clk, probe_idx, probe_info);

    fpga_ex3_fsm FSM (clk, insn_end, intr_detected, halted, probe_info,
                     KEY, sm, cpu_state, com_addr_reg, probe_idx);

/*********************************************************************
                       GPIO connection:
board A(G1)                                board B(G2)
=============================              ============================
GPIO_0[17:0]   = GP_IN [17:0]   <-----  GPIO_0[17:0]   = GP_OUT[17:0]
GPIO_0[35:18]  = GP_OUT[17:0]   ----->  GPIO_0[35:18]  = GP_IN [17:0]
-----------------------------              ----------------------------
GP_IN [7:0]    = pin_byte       <-----  GP_OUT[7:0]    = outr[7:0]
GP_IN [8]      = pin_bsy        <-----  GP_OUT[8]      = fgo[0]
GP_OUT[9]      = fgi[0]         ----->  GP_IN [9]      = pout_bsy
GP_OUT[9]      = fgi[0]         ----->  GP_IN [9]      = pout_bsy

GP_OUT[7:0]    = outr[7:0]      ----->  GP_IN [7:0]    = pin_byte
GP_OUT[8]      = fgo[0]         ----->  GP_IN [8]      = pin_bsy
GP_IN [9]      = pout_bsy       <-----  GP_OUT[9]      = fgi[0]

*********************************************************************/

    assign  GPIO_0   =   (gm[1]) ? {GP_OUT, 18'hzzzzz} :  /// G1
                         (gm[2]) ? {18'hzzzzz, GP_OUT} :  /// G2
                         36'hzzzzzzzzz;                   /// G0

    assign  GP_IN    =    (gm[1]) ? GPIO_0[17:0]         :  /// G1
                          (gm[2]) ? GPIO_0[35:18]        :  /// G2
//                        18'h0;                            /// G0
                          {8'h0, fgo[0], 9'b0};             /// G0

    assign pin_byte    = GP_IN[7:0];
    assign pin_bsy     = GP_IN[8];
    assign pout_bsy    = GP_IN[9];
    assign inpr        = (iot) ? sin_byte : pin_byte;
    assign fgi_bsy     = { ~sin_rdy,  pin_bsy  };
    assign fgo_bsy     = { ~sout_rdy, pout_bsy };
    assign GP_OUT[7:0] = outr;
    assign GP_OUT[9:8] = { fgi[0], fgo[0] };

    wire cpu_reset = (cpu_state == `COM_RST);

    assign intr_detected = r & sc_clr & (intr_detect_fgi & fgi[iot] | intr_detect_fgo &
         fgo[iot]);

    assign insn_end = sc_clr;

    wire [11:0] com_addr = (cpu_state == `COM_STP) ? com_addr_reg : pc;
    wire [15:0] seg_out = (SW[7:6] == 2'b11) ? {4'b0, com_addr} : seg;

    seg7_enc SEG7_E0 (seg_out[15:12],HEX3);
    seg7_enc SEG7_E1 (seg_out[11:8], HEX2);
    seg7_enc SEG7_E2 (seg_out[7:4],  HEX1);
    seg7_enc SEG7_E3 (seg_out[3:0],  HEX0);

    assign halted = (s == 1'b0);
    assign GPIO_1 = {1'b0, s, ien, r, sc_clr, sc, pc, ir};

    assign sys_probe_data[15:0] = (sp[0])  ? ac[15:0] :
                                  (sp[1])  ? {15'b0, e} :
                                  (sp[2])  ? {4'b0, pc} :
                                  (sp[3])  ? ir[15:0] :
                                  (sp[4])  ? {4'b0, ar} :
                                  (sp[5])  ? dr[15:0] :
                                  (sp[6])  ? mem[15:0] :
                                  (sp[7])  ? bus[15:0] :
                                  (sp[8])  ? {13'b0, sc} :
                                  (sp[9])  ? {10'b0, ien, imsk, iot} :
                                  (sp[10]) ? {4'b0, fgi_bsy, fgi, inpr} :
                                  (sp[11]) ? {4'b0, fgo_bsy, fgo, outr} :
                                  16'hffff;
    assign sys_probe_data[17:16] = cpu_state;

    cpu_ex3 CPU_EX3 (clk, cpu_state, com_addr,
            fgi_bsy, fgi, inpr, /// input port
            fgo_bsy, fgo, outr, /// output port
            ien, imsk, iot,
            s, r, e, bus, mem, dr, ac, ir, ar, pc, sc, sc_clr, seg);

    uart_rx S_IN  (clk, cpu_reset, UART_RXD, sin_byte, rx_error,  sin_rdy);
    uart_tx S_OUT (clk, cpu_reset, UART_TXD, outr,     fgo[1],    sout_rdy);

endmodule

module fpga_ex3_fsm (clk, insn_end, intr_detected, halted, probe_info,
                     KEY, sm, cpu_state, com_addr_reg, probe_idx);

    input           clk, insn_end, intr_detected, halted;
    input   [3:0]   KEY, sm;
    input   [31:0]  probe_info;
    output  [1:0]   cpu_state;
    output  [11:0]  com_addr_reg, probe_idx;

    reg     [3:0]   PREV_KEY;
    reg     [1:0]   cpu_state;
    reg     [11:0]  com_addr_reg, probe_idx;

    wire    [3:0]   probe_header = probe_info[31:28];
    wire    [11:0]  probe_addr = probe_info[27:16];

    wire    KEY0    = ~KEY[0] & PREV_KEY[0]; /// detect 1->0 transition on KEY[0]
    wire    KEY1    = ~KEY[1] & PREV_KEY[1]; /// detect 1->0 transition on KEY[1]
    wire    KEY2    = ~KEY[2] & PREV_KEY[2]; /// detect 1->0 transition on KEY[2]
    wire    KEY3    = ~KEY[3] & PREV_KEY[3]; /// detect 1->0 transition on KEY[3]

    wire    stop_cond = (sm[2] | (sm[3] & insn_end) |
                        (sm[1] & intr_detected) | (halted & KEY1));

    initial cpu_state = `COM_RST; ///   cpu_state = S0 after power-on
    initial PREV_KEY  = 4'b0;     ///   PREV_KEY = 0000 after power-on

////////////       system finite state machine       //////////////
    always @ (posedge clk) begin
        PREV_KEY <= KEY;
        case(cpu_state)
//////////// COM_RST : reset state (after power on) ////////////
            `COM_RST : begin
                if(KEY0)            cpu_state <= `COM_RUN;
                else if(KEY1)       cpu_state <= `COM_STP;
                com_addr_reg <= 12'b0;
                probe_idx    <= 12'b0;
            end
//////////// COM_RUN : run state ////////////
            `COM_RUN : begin
                if(KEY0)            cpu_state <= `COM_RST;
                else if(stop_cond)  cpu_state <= `COM_STP;
            end
//////////// COM_STP : stop state ////////////
            `COM_STP : begin
                if(KEY0)            cpu_state <= `COM_RST;
                else if(KEY1)       cpu_state <= `COM_RUN;
                if(halted) begin
```

```
                    if(KEY2 | (probe_header == `MEM_END)) probe_idx <= 12'b0;
                    else if(KEY3)                         probe_idx <= probe_idx + 1;
                    com_addr_reg  <= probe_addr;
                end
                else begin
                    if(KEY2)        com_addr_reg <= 12'b0;
                    else if(KEY3)   com_addr_reg <= com_addr_reg + 1;
                end
            end
        endcase
    end
endmodule
```

アセンブリコード

ソースコード 9　test_calc1.asm

```
                ORG 0           / interrupt entry point
ST0,HEX 0               / interrupt return address
        BUN I_HND       / goto I_HND (interrupt handler)

        ORG 10          / program entry point
INI, / initialize data
                LDA VH8                 / AC <- 1000
                IMK                             / IMSK <- 1000 (S_IN enabled)
                SIO                             / IOT <- 1 (serial-IO selected)
/ initialize
                CLA                             / AC <- 0
                STA BYE                 / M[BYE] <- 0
                STA NXT_BYE             / M[NXT_BYE] <- 0
                STA X                   / M[X] <- 0
                LDA CH_EQ               / AC <- '='
                STA OPR                 / M[OPR] <- '='
                BSA INI_ST              / call INI_ST (initialize state)
                ION                             / enable interrupt
/ wait until (M[BYE] = 1)
L0,
                LDA BYE                 / AC <- M[BYE]
                SZA                             / (M[BYE] == 0) ? skip next
                HLT
                BUN L0                  / goto L0

INI_ST, HEX 0
////////// subroutine (initialize state) ///////////
                CLA                             / AC     <- 0
                STA Y                   / M[Y]    <- 0
                STA Y_PD                / M[Y_PD] <- 0
                STA STT                 / M[STT]  <- 0
                STA OUT_STT             / M[OUT_STT] <- 0
                LDA VM10                        / AC     <- M[VM4] (-10)
                STA CNT                 / M[CNT]  <- -10
                BUN INI_ST I    / return from INI_ST
```

```
/////////// interrupt handler ///////////
I_HND,
/ store AC & E to memory
                STA BA                  / M[BA] <- AC   (store AC)
                CIL                             / AC[0] <- E    (AC[15:1] is not
                    important here...)
                STA BE                  / M[BE] <- AC   (store E)

/////////// state machine ///////////
/ M[OUT_STT] != 0  : output pending
/ M[STT] = 0  : read hex inputs (up to 4 hex digits)
/ M[STT] = 1  : read operator (+,-,= : compute, others : end program...)
/ M[STT] >= 2 : output message
/ check state :
                LDA OUT_STT             / AC <- M[OUT_STT]
                SZA                             / (M[OUT_STT] = 0) ? skip next
                BUN PUT_OUT             / goto PUT_OUT (process output)

/////////// process input ///////////
/ input mode (M[TMI] <- INPR)
                SKI                             / (FGI = 0) ? skip next
                BUN IRT                 / goto IRT (return from interrupt handler) -->
                    this should not happen...
                CLA                             / AC <- 0
                INP                             / AC[7:0] <- INPR
                STA TMI                 / M[TMI] <- INPR
                /BSA READ_HX            / call READ_HX (read hex value to M[HXI](3:0))
                BSA READ_DX
                SPA                             / (AC >= 0) ? skip next
                BUN STT_1               / goto STT_1 (non-hex input)
/ hex input :
/ check state 0 :
                LDA STT                 / AC <- M[STT]
                SZA                             / (AC = 0) ? skip next
                BUN ERR                 / goto ERR (error!!!)

/////////// state 0: read operand 1,2 ///////////
                LDA Y                   / AC <- M[Y]
                MUL VD10
                ADD DXI
                STA Y
                LDA VH1
                STA Y_PD
                ISZ CNT


/ operand digit pending
                BUN IRT                 / goto IRT (return from interrupt handler)
/ goto state 1 :
                ISZ STT                 / ++M[STT] (no skip)

/////////// return from interrupt handler ///////////
```

```
IRT,
                LDA BE                  / AC <- M[BE]
                CIR                             / E  <- AC[0]   (restore E)
                LDA BA                  / AC <- M[BA]   (restore AC)
                ION                             / IEN <- 1              (enable
                    interrupt)
                BUN STO I               / indirect return (return address stored in STO
                    )

/////////// error !!!! ///////////
ERR,
                CLA                             / AC <- 0
                STA X                   / M[X] <- 0
                STA Y                   / M[Y] <- 0
                LDA A_EMG               / AC <- M[A_EMG] (EMG)
                BSA SET_MSG             / call SET_MSG (set message info)

/////////// prepare output ///////////
PRP_OUT,
                LDA VH1                 / AC <- M[VH1] (2)
                STA OUT_STT             / M[OUT_STT] <- 1 (output state)
                LDA VH4                 / AC <- 0100
                IMK                             / IMSK <- 0100 (S_OUT enabled)
                BUN IRT                 / goto IRT (return from interrupt handler)

CHK_CH, HEX 0                   / return address
////////// subroutine (check character) ///////////
/ arg0 (AC) : character to identify
/ return AC = 1 : character matched
/ return AC = 0 : character not matched
                CMA                             / AC <- ~AC
                INC                             / AC <- AC + 1 (AC = - arg0)
                ADD TMI                 / AC <- AC + M[TMI] (M[TMI] - arg0)
                SZA                             / (M[TMI] = arg0) ? skip next
                LDA VM1                 / AC <- M[VM1] (-1) (no match)
                INC                             / AC <- AC + 1
                BUN CHK_CH I    / return from CHK_CH

/////////// state 1: read operator ///////////
STT_1,  / cur-operator : M[TMI]
/ (cur-operator = ' ') ?
                LDA CH_WS               / AC <- M[CH_WS] (' ')
                BSA CHK_CH              / call CHK_CH (check character)
                SZA                             / (AC = 0) ? skip next (not white-space
                    )
                BUN STT_WS              / goto STT_WS (handle white-space)
/ (cur-operator = '=') ?
                LDA CH_EQ               / AC <- M[CH_EQ] ('=')
                BSA CHK_CH              / call CHK_CH (check character)
                SZA                             / (AC = 0) ? skip next (not '=')
                BUN STT_COMP    / goto STT_COMP (compute on prev-operator)
/ (cur-operator = '+') ?
                LDA CH_PL               / AC <- M[CH_PL] ('+')
                BSA CHK_CH              / call CHK_CH (check character)
                SZA                             / (AC = 0) ? skip next (not '+')
                BUN STT_COMP    / goto STT_COMP (compute on prev-operator)
/ (cur-operator = '-') ?
                LDA CH_MN               / AC <- M[CH_MN] ('-')
                BSA CHK_CH              / call CHK_CH (check character)
                SZA                             / (AC = 0) ? skip next (not '-')
                BUN STT_COMP    / goto STT_COMP (compute on prev-operator)
/ (cur-operator = '*') ?
                LDA CH_MUL              / AC <- M[CH_MN] ('*')
                BSA CHK_CH              / call CHK_CH (check character)
                SZA                             / (AC = 0) ? skip next (not '*')
                BUN STT_COMP    / goto STT_COMP (compute on prev-operator)
/ (cur-operator = '/') ?
                LDA CH_DIV              / AC <- M[CH_MN] ('/')
                BSA CHK_CH              / call CHK_CH (check character)
                SZA                             / (AC = 0) ? skip next (not '/')
                BUN STT_COMP    / goto STT_COMP (compute on prev-operator)
/ (cur-operator = 'p') ?
                LDA CH_P                / AC <- M[CH_MN] ('p')
                BSA CHK_CH              / call CHK_CH (check character)
                SZA                             / (AC = 0) ? skip next (not 'p')
                BUN STT_COMP    / goto STT_COMP (compute on prev-operator)
/ (cur-operator = 'c') ?
                LDA CH_C                / AC <- M[CH_MN] ('c')
                BSA CHK_CH              / call CHK_CH (check character)
                SZA                             / (AC = 0) ? skip next (not 'c')
                BUN STT_COMP    / goto STT_COMP (compute on prev-operator)
/ (cur-operator = '^') ?
                LDA CH_EXP              / AC <- M[CH_MN] ('^')
                BSA CHK_CH              / call CHK_CH (check character)
                SZA                             / (AC = 0) ? skip next (not '^')
                BUN STT_COMP    / goto STT_COMP (compute on prev-operator)
/ (cur-operator is unsupported... : prepare to terminate this program)
                LDA VH1                 / AC <- M[VH1] (1)
                STA NXT_BYE             / M[NXT_BYE] <- 1
                LDA A_BMG               / AC <- M[A_BMG] (BMG)
                BSA SET_MSG             / call SET_MSG (set message info)
                BUN PRP_OUT
/ current input is white-space : check (M[Y_PD] = 1) ?
STT_WS,
                LDA Y_PD                / AC <- M[Y_PD]
                ADD VM1                 / AC <- M[Y_PD] - 1
                SZA                             / (M[Y_PD] - 1 = 0) ? skip next
/ no pending input
                BUN IRT                 / goto IRT (return from interrupt handler)
/ set STT <- 1 on white-space on pending input (M[Y_PD] = 1)
                LDA VH1                 / AC <- M[VH1] (1)
                STA STT                 / M[STT] <- 1
                BUN IRT                 / goto IRT (return from interrupt handler)
/ compute on prev-operator
STT_COMP,
/ swap M[OPR] (prev-operator) <-> M[TMI] (cur-operator)
```

```
||                LDA  OPR                    / AC      <- M[OPR]
||                STA  TMA                    / M[TMA] <- M[OPR]
||                LDA  TMI                    / AC      <- M[TMI]
||                STA  OPR                    / M[OPR] <- M[TMI]
||                LDA  TMA                    / AC      <- M[TMA]
||                STA  TMI                    / M[TMI] <- M[TMA]
||/ (M[Y_PD] = 0) ?
||                LDA  Y_PD                   / AC <- M[Y_PD]
||                SZA                                 / (M[Y_PD] = 0) ? skip next
||                BUN  CHK_OP                 / goto CHK_OP (check prev-operator)
||/ no input at M[Y] : copy M[X] to M[Y]
||                LDA  X                      / AC    <- M[X]
||                STA  Y                      / M[Y] <- M[X]
||CHK_OP,
||/ skip-output flag = 0
||                CLA                                 / AC      <- 0
||                STA  TMA                    / M[TMA] <- 0 (skip-output flag = 0)
||/ (prev-operator = '=') ?
||                LDA  CH_EQ                  / AC <- M[CH_EQ] ('=')
||                BSA  CHK_CH                 / call CHK_CH
||                SZA                                 / (AC = 0) ? skip next
||                BUN  C_EQ                   / goto C_EQ (compute EQUAL)
||/ (prev-operator = '+') ?
||                LDA  CH_PL                  / AC <- M[CH_PL] ('+')
||                BSA  CHK_CH                 / call CHK_CH
||                SZA                                 / (AC = 0) ? skip next
||                BUN  C_ADD                  / goto C_ADD (compute ADD)
||/ (prev-operator = '-') ?
||                LDA  CH_MN                  / AC <- M[CH_PL] ('-')
||                BSA  CHK_CH                 / call CHK_CH
||                SZA                                 / (AC = 0) ? skip next
||                BUN  C_SUB                  / goto C_SUB (compute SUB)
||/ (prev-operator = '*') ?
||                LDA  CH_MUL                 / AC <- M[CH_MUL] ('*')
||                BSA  CHK_CH                 / call CHK_CH
||                SZA                                 / (AC = 0) ? skip next
||                BUN  C_MUL                  / goto C_SUB (compute MUL)
||/ (prev-operator = '/') ?
||                LDA  CH_DIV                 / AC <- M[CH_PL] ('/')
||                BSA  CHK_CH                 / call CHK_CH
||                SZA                                 / (AC = 0) ? skip next
||                BUN  C_DIV                  / goto C_SUB (compute DIV)
||/ (prev-operator = 'p') ?
||                LDA  CH_P                   / AC <- M[CH_PL] ('p')
||                BSA  CHK_CH                 / call CHK_CH
||                SZA                                 / (AC = 0) ? skip next
||                BUN  C_P            / goto C_P (compute xPy)
||/ (prev-operator = 'c') ?
||                LDA  CH_C                   / AC <- M[CH_PL] ('c')
||                BSA  CHK_CH                 / call CHK_CH
||                SZA                                 / (AC = 0) ? skip next
||                BUN  C_C            / goto C_C (compute xCy)
||/ (prev-operator = '^') ?
```

```
                LDA CH_EXP              / AC <- M[CH_PL] ('^')
                BSA CHK_CH              / call CHK_CH
                SZA                             / (AC = 0) ? skip next
                BUN C_EXP               / goto C_EXP (compute EXP)
/ (prev-operator is unsupported) ?
                BUN C_NONE              / goto C_NONE (unsupported operator)
C_EQ,   / EQUAL : M[Z] <- M[Y]
                ISZ TMA                 / ++M[TMA] (no skip) : skip-output flag = 1
                LDA Y                   / AC    <- M[Y]
                BUN STA_Z               / goto STA_Z
C_ADD,  / ADD : M[Z] <- M[X] + M[Y]
                LDA X                   / AC <- M[X]
                ADD Y                   / AC <- M[X] + M[Y]
                BUN STA_Z               / goto STA_Z
C_SUB,  / SUB : M[Z] <- M[X] - M[Y]
                LDA Y                   / AC <- M[Y]
                CMA                             / AC <- ~AC
                INC                             / AC <- AC + 1 (-M[Y])
                ADD X                   / AC <- M[X] - M[Y]
                BUN STA_Z               / goto STA_Z
C_MUL,  / MUL : M[Z] <- M[X] * M[Y]
                LDA Y                   / AC <- M[Y]
                MUL X                   / AC <- M[X] * M[Y]
                BUN STA_Z               / goto STA_Z
C_DIV,  / DIV : M[Z] <- M[X] / M[Y]
                LDA X                   / AC <- M[Y]
                DIV Y                   / AC <- M[X] / M[Y]
                BUN STA_Z               / goto STA_Z
C_P,    / P
                LDA X
                STA P_X
                LDA Y
                STA P_Y
                BSA P
                BUN STA_Z               / goto STA_Z
C_C,    / C
                LDA X
                STA C_X
                LDA Y
                STA C_Y
                BSA C
                BUN STA_Z               / goto STA_Z
C_EXP,  / EXP
                LDA X
                STA EXP_X
                LDA Y
                STA EXP_Y
                BSA EXP
                BUN STA_Z               / goto STA_Z
C_NONE,
                CLA                             / AC <- 0 (just for now...)
STA_Z,
                STA Z                   / M[Z] <- AC
                STA X                   / M[X] <- M[Z]
                CLA                             / AC <- 0
                STA Y                   / M[Y] <- 0
                STA Y_PD                / M[Y_PD] <- 0
/ check skip-output flag
                LDA TMA                 / AC <- M[TMA] (skip-output flag)
                SZA                             / (AC = 0) ? skip next (prev-operator
                    != '=')
                BUN SKIP_OUT    / goto SKIP_OUT (prev-operator = '=')
/ write Z to Z_MSG
                /BSA WRITE_Z            / call WRITE_Z (write Z to Z_MSG)
                /LDA A_ZMG              / AC <- M[A_ZMG] (ZMG)
                BSA H2D
                LDA A_ZHMG
                BSA SET_MSG             / call SET_MSG (set message info)
                BUN PRP_OUT
SKIP_OUT, / prev-operator is '=' : skip output
                BSA INI_ST              / call INI_ST (initialize state)
                BUN IRT                 / goto IRT (return from interrupt handler)

SET_MSG,HEX 0
////////// subroutine (set message info) //////////
/ arg0 (AC) : message address
                STA PTR_MG              / M[PTR_MG] <- arg0 (message address)
                ADD VM1                 / AC <- arg0 - 1
                STA TMA                 / M[TMA] <- arg0 - 1
                LDA TMA I               / AC <- M[M[TMA]] (M[arg0 - 1] = message count)
                STA CNT                 / M[CNT] <- message count
                BUN SET_MSG I   / return from SET_MSG

READ_DX,HEX     0
                LDA CH_0                / AC <- M[CH_0] ('0')
                BSA CHK_DGT             / call CHK_DGT (check digit character)
                DEC 0                   / 2nd argument to CHK_DGT (offset)
                DEC 9                   / 3rd argument to CHK_DGT (upper bound)
                SNA                             / (AC < 0) ? skip next
                BUN READ_DX I   / return from RHX (M[HXI](3:0) = {0 to 9})
/ not dec value --> convert new-line (\n) and carrage-return (\r) to equal (=)
                LDA CH_NL               / AC <- M[CH_NL] ('\n')
                BSA CHK_CH              / call CHK_CH
                SZA                             / (AC = 0) ? skip next
                BUN CONV_EQ_D           / goto CONV_EQ_D (convert to EQUAL)
                LDA CH_CR               / AC <- M[CH_CR] ('\r')
                BSA CHK_CH              / call CHK_CH
                SZA                             / (AC = 0) ? skip next
                BUN CONV_EQ_D           / goto CONV_EQ_D (convert to EQUAL)
R_READ_DX,
                LDA VM1                 / AC <- M[VM1] (-1)
                BUN READ_DX I   / return from RHX (not hex value)
CONV_EQ_D,
                LDA CH_EQ               / AC <- M[CH_EQ] ('=')
                STA TMI                 / M[TMI] <- '='
                BUN R_READ_DX   / goto R_READ_DX (return : not hex value)
```

```


CHK_DGT,HEX 0                    / return address
////////// subroutine (check digit character) //////////
/ arg0 (AC) : lower bound character
/ arg1 (M[M[CHK_DGT]]) : offset
/ arg2 (M[M[CHK_DGT]+1]) : upper bound value
/ return AC >= 0 : valid digit value in M[HXI](3:0)
/ return AC < 0  : not valid digit
/ check (M[TMI] >= lower bound)
                CMA                             / AC <- ~AC
                INC                             / AC <- AC + 1 (- arg0)
                ADD     TMI                     / AC <- AC + M[TMI] (M[TMI] - arg0)
                SPA                             / (AC = M[TMI] - arg0 >= 0) ? skip next
                BUN R_CHK1              / goto R_CHK1 (return : AC < 0)
                STA TMA                 / M[TMA] <- M[TMI] - arg0
                ADD CHK_DGT I   / AC <- M[TMI] - arg0 + arg1
                STA DXI                 / M[HXI] <- M[TMI] - arg0 + arg1 (actual hex
                    value)
                ISZ CHK_DGT             / ++M[CHK_DGT]
/ check (M[TMI] <= upper bound)
                LDA TMA                 / AC <- M[TMA] (M[TMI] - arg0)
                CMA                             / AC <- ~AC
                INC                             / AC <- AC + 1 (arg0 - M[TMI])
                ADD CHK_DGT I   / AC <- arg2 - arg0 - M[TMI] (if (AC < 0) then not
                    within bound)
                BUN R_CHK2              / goto R_CHK2
R_CHK1,
                ISZ CHK_DGT             / ++M[CHK_DGT]
R_CHK2,
                ISZ CHK_DGT             / ++M[CHK_DGT]
                BUN CHK_DGT I   / return from CHK_DGT


////////// process output //////////
PUT_OUT,
                SKO                             / (FGO = 0) ? skip next
                BUN IRT                 / goto IRT (return from interrupt handler) -->
                    this should not happen...
/ here, AC = M[OUT_STT] :
                ADD VM1                 / AC <- M[OUT_STT] - 1
                SZA                             / (M[OUT_STT] = 1) ? skip next
                BUN PUT_OUT_2   / goto PUT_O2
/ M[OUT_STT] = 1 : put 1st newline
                LDA CH_NL               / AC <- M[CH_NL] ('\n')
                OUT                             / OUTR <- AC(7:0)
                ISZ OUT_STT             / ++M[OUT_STT] (no skip)
                BUN IRT                 / goto IRT (return from interrupt handler)
/ check (M[OUT_STT] = 2) ?
PUT_OUT_2,
                ADD VM1                 / AC <- M[OUT_STT] - 1 - 1
                SZA                             / (M[OUT_STT] = 2) ? skip next
                BUN PUT_NL2             / goto PUT_NL2
```

```
/ M[OUT_STT] = 2 : put message
                LDA PTR_MG I    / AC <- M[M[PTR_MG]]
                OUT                             / OUTR <- AC(7:0)
                ISZ PTR_MG              / ++M[PTR_MG] (no skip)
                ISZ CNT                 / (++M[CNT])= 0) ? skip next
                BUN IRT                 / goto IRT (return from interrupt handler)
                ISZ OUT_STT             / ++M[OUT_STT] (no skip)
                BUN IRT                 / goto IRT (return from interrupt handler)
/ M[OUT_STT] = 3 : put 2nd newline (process output ends here...)
PUT_NL2,
                LDA CH_NL               / AC <- M[CH_NL] ('\n')
                OUT                             / OUTR <- AC(7:0)
                BSA INI_ST              / call INI_ST (initialize state)
                LDA NXT_BYE             / AC <- M[NXT_BYE]
                STA BYE                 / M[BYE] <- M[NXT_BYE]
                SZA                             / (M[NXT_BYE] == 0) ? skip next
                BUN EXT                 / goto EXT (disable all interrupts)
                LDA VH8                 / AC <- 1000
                IMK                             / IMK <- 1000 (S_IN enabled)
                BUN IRT                 / goto IRT (return from interrupt handler)
EXT,
                CLA                             / AC <- 0
                IMK                             / IMK <- 0000 (all interrupts disabled)
                BUN IRT                 / goto IRT (return from interrupt handler)


H2D, HEX 0
        LDA A_ZHMG
        ADD VH9
        STA A_ZHMG
        LDA VM10
        STA CNT
H2DINI,
        CLA
        STA A_ZHMG I
        LDA A_ZHMG
        ADD VM1
        STA A_ZHMG
        ISZ CNT
        BUN H2DINI
        LDA A_ZHMG
        ADD VD10
        STA A_ZHMG
        LDA VM10
        STA CNT
/ ZHMG(CHR)[] <- Z(HEX)
LHD,
        ISZ CNT_ZHMG
        LDA W1  / AC <- M[W1]
        STA W   / M[W] <- AC
        CLA     / AC <- 0
        LDA Z   / AC <- M[Z]
        ADD VM10 / AC -= 10
        CME     / E <- ~E
        SZE     / (E == 0) ? skip next
        BUN LHD5 / goto LHD5
LHD0,
        LDA W   / AC <- M[W]
        SNA     / (AC < 0) ? skip next
        BUN LHD4 / goto LHD4
LHD1,
        CLE     / E <- 0
        CLA     / AC <- 0
        LDA Z   / AC <- M[Z]
        CIL     / {E,AC[15:0]} <- {AC[15:0],E}
        STA Z   / M[Z] <- AC
        CLA     / AC <- 0
        LDA A_ZHMG I / AC <- M[M[A_ZHMG]]
        CIL     / {E,AC[15:0]} <- {AC[15:0],E}
        STA A_ZHMG I / M[M[A_ZHMG]] <- AC
        LDA VHA   / AC <- M[M]
        CMA     / AC <- ~AC
        INC     / ++AC
        ADD A_ZHMG I / AC += M[M[A_ZHMG]]
        SZE     / (E == 0) ? skip next
        BUN LHD2 / goto LHD2
LHD3,
  CLA     / AC <- 0
        LDA W   / AC <- M[W]
        INC     / ++AC
        STA W   / M[W] <- AC
        CLA     / AC <- 0
        BUN LHD0 / goto LHD0
LHD2,
        STA A_ZHMG I / M[M[A_ZHMG]] <- AC
        CLA     / AC <- 0
        LDA Z   / AC <- M[Z]
        INC     / ++AC
        STA Z   / AC <- M[Z]
        BUN LHD3 / goto LHD3
LHD4,
        LDA A_ZHMG I/ AC <- M[M[A_ZHMG]]
        ADD CH_0 / AC += 48
        STA A_ZHMG I / AC <- M[M[A_ZHMG]]
        LDA A_ZHMG
        ADD VM1
        STA A_ZHMG
        BUN LHD / goto M[LHD]
LHD5,
        CLA     / AC <- 0
        LDA Z   / AC <- M[Z]
        ADD CH_0 / AC += 48
        STA A_ZHMG I / M[M[A_ZHMG]] <- AC
        LDA A_ZHMG
        LDA CNT_ZHMG
        SNA
        BUN LHD7
LHD6,
        LDA A_ZHMG
        ADD VM1
        STA A_ZHMG
        LDA CH_0
        STA A_ZHMG I
        ISZ CNT_ZHMG
        BUN LHD6
LHD7,
        LDA VM10
        STA CNT_ZHMG
        CLA
        STA W
        BUN H2D I



P, HEX 0
P_1,
        LDA P_Z
        MUL P_Y
        STA P_Z
        LDA P_Y
        CMA
        ADD VH1
        ADD P_X
        SZA
        BUN P_2
        LDA P_Z
        STA P_RES
        CLA
        ADD VH1
        STA P_Z
        LDA P_RES
        BUN P I
P_2,
        ISZ P_Y
        BUN P_1

C, HEX 0
        LDA C_X
        STA P_X
        LDA C_Y
        CMA
        ADD VH2
        ADD C_X
        STA P_Y
        BSA P
        STA C_A
        LDA C_Y
        STA P_X
        LDA VH1
        STA P_Y
        BSA P
        STA C_B
        LDA C_A
        DIV C_B
        STA C_RES
        BUN C I

EXP, HEX 0
EXP_1,
        LDA EXP_A
        CMA
        ADD VH1
        ADD EXP_Y
        SZA
        BUN EXP_2
        LDA EXP_Z
        STA EXP_RES
        CLA
        ADD VH1
        STA EXP_Z
        CLA
        STA EXP_A
        LDA EXP_RES
        BUN EXP I
EXP_2,
        LDA EXP_Z
        MUL EXP_X
        STA EXP_Z
        ISZ EXP_A
        BUN EXP_1


/ data (no initialization)
Z,              DEC 0       / result
TMA,    DEC 0           / temporal
TMB,    DEC 0           / temporal
TMI,    DEC 0           / char (raw) input
HXI,    DEC 0           / hex input
BA,             DEC 000         / backup storage for AC during interrupt handling
BE,             DEC 000         / backup storage for  E during interrupt handling
PTR_MG, HEX 0           / message pointer
/ data (need initialization code : one-time)
BYE,    DEC 0           / (init: 0) bye
NXT_BYE,DEC 0           / (init: 0) next bye
OPR,    DEC 0           / (init: 0) operator
X,              DEC 0       / (init: 0) X operand
/ data (need initialization code : after every output -> INI_ST)
Y,              DEC 0       / (init: 0) Y operand
Y_PD,   DEC 0           / (init: 0) Y pending
CNT,    DEC 0           / (init: -4) digit count
STT,    DEC 0           / (init: 0) 0: read operand, 1: read operator
OUT_STT,DEC 0           / (init: 0) 0: output 1st newline, 1: output ans, 2: output 2nd
```

```
     newline
/ data (read-only)
AMK,    HEX FFF0        / AMK = FFF0 (and mask)
AMKN,   HEX 000F        / AMKN = 000F (and mask negated)
VH1,    HEX 1           / VH1 = 1
VH2,    HEX 2           / VH2 = 2
VH3,    HEX 3           / VH3 = 3
VH4,    HEX 4           / VH4 = 4
VH8,    HEX 8           / VH8 = 8
VH9,  HEX 9
VHA,    HEX A           / VHA = A
VM1,    DEC -1          / VM1 = -1
VM2,    DEC -2          / VM2 = -2
VM4,    DEC -4          / VM2 = -4
VM5,  DEC -5
VM10,   DEC -10         / VM10 = -10
CH_0,   HEX 30          / '0'
CH_P, HEX 70
CH_C, HEX 63
CH_EXP, HEX 5E
CH_UA,  HEX 41          / 'A'
CH_LA,  HEX 61          / 'a'
CH_NL,  HEX 0A          / '\n' (newline : line feed)
CH_CR,  HEX 0D          / '\r' (carrage return : appears on DOS)
CH_WS,  HEX 20          / ' ' (white space)
CH_EQ,  HEX 3D          / '=' (equal)
CH_PL,  HEX 2B          / '+' (plus)
CH_MN,  HEX 2D          / '-' (minus)
CH_MUL, HEX 2A          / '*' (mul)
CH_DIV, HEX 2F          / '/' (div)
A_ZMG,  SYM ZMG
CNT_ZMG,DEC -4          / CNT_ZMG = -4
ZMG,    HEX 0           / hex digit 3
                HEX 0           / hex digit 2
                HEX 0           / hex digit 1
                HEX 0           / hex digit 0
A_EMG,  SYM EMG
CNT_EMG,DEC -6          / CNT_EMG = -6
EMG,    HEX 65          / 'e'
                HEX 72          / 'r'
                HEX 72          / 'r'
                HEX 6F          / 'o'
                HEX 72          / 'r'
                HEX 21          / '!'
A_BMG,  SYM BMG
CNT_BMG,DEC -4          / CNT_BMG = -4
BMG,    HEX 62          / 'b'
                HEX 79          / 'y'
                HEX 65          / 'e'
                HEX 21          / '!'
/nakata
A_ZHMG, SYM ZHMG
CNT_ZHMG,DEC -10                / CNT_ZHMG = -10
```

```
ZHMG,
                HEX 0   / hex digit 9
                HEX 0           / hex digit 8
                HEX 0           / hex digit 7
                HEX 0           / hex digit 6
                HEX 0           / hex digit 5
                HEX 0   / hex digit 4
                HEX 0           / hex digit 3
                HEX 0           / hex digit 2
                HEX 0           / hex digit 1
                HEX 0           / hex digit 0
W, HEX 0
W1, DEC -32
DXI, DEC 0
VD10, DEC 10
VD100, DEC 100
VD1000, DEC 1000
VD10000, DEC 10000
P_X, DEC 0
P_Y, DEC 0
P_Z, DEC 1
P_RES, DEC 0
C_X, DEC 0
C_Y, DEC 0
C_A, DEC 1
C_B, DEC 1
C_RES, DEC 0
EXP_X, DEC 0
EXP_Y, DEC 0
EXP_A, DEC 0
EXP_Z, DEC 1
EXP_RES, DEC 0

END
```