ASIC デザインコンテスト 規定課題
「16 bit Free CPU」

パルテノン研究会

2000/11/16

1　はじめに

パルテノン研究会主催の ASIC デザインコンテストでは, 当初より KUE-CHIP2 を規定課題のひとつとして選定し, 多くの応募を得てきましたが, 命令セットやデータパスを指定していましたので次第にアイデアが出尽くした感があります. また一方で KUE-CHIP 以降, 多くの大学で独自の教育用プロセッサの設計 (PICO, Kite, Aser, SMPL など) が行われ, 学生の工夫を引き出すための様々な試みが行われるようになってきています. この成果を活用できて, すなわち手持ちの材料やアイデアをそのまま使えて, かつ多くの所で競い合うことの出来る新しい規定課題はないものだろうかということで, この「16 bit Free CPU」規定課題が作られました. この考え方と規定課題の問題そのものは慶応大学の天野先生のアイデアです. NTT 未来ねっと研究所の中根が審査をきちんと行うための方法を決め文書化しました.

2　課題の概要

課題は**以下の 3 つの問題を解くのに最適な命令セットを持った CPU を SFL で設計する**ことです.

ソート　16 bit の正整数データ 256 個をソートする.

素数　1 から $1023(2^{10}-1)$ までの素数を算出する.

G.C.D.　2 つの 16 bit の正整数の最大公約数を計算する.

上の問題を解く CPU を設計する場合, 具体的には次の 3 つのものを設計しなければなりません.

1. ハードウェアのアーキテクチャ
2. CPU の命令セット
3. 各問題を解くプログラム

これらは上の問題が解ける最低限のものが備わっていれば良いとします. ただし設計に際してメモリに以下の制約を設けます.

1. CPU とメモリの間のバンド幅は合計で最高 32 bit/clock までとします. 32 bit/clock までであればその構成は問いません. 具体的な例については 3.1.1 を参照してください.
2. メモリのアドレス空間は最低 16 bit で表されるものとします.
3. メモリには 1 クロックでアクセス可能であるとします. (つまり, キャッシュの工夫は意味がありません.)

この制約を満たすメモリを前提として上の 3 つの問題を解く CPU, 命令セット及びプログラムを設計してください. なお, 制約を満たすメモリモジュールの記述 (SFL の declare, circuit 記述, 及び pcd 記述) はコンテストの事務局で用意します.

以上をまとめますと, この課題での設計は大きく分けて図 1 のように分類されます. そして, これらのうちプログラム, 命令セット, メモリモジュールへのインターフェースを含めたハードウェアが設計対象となります.

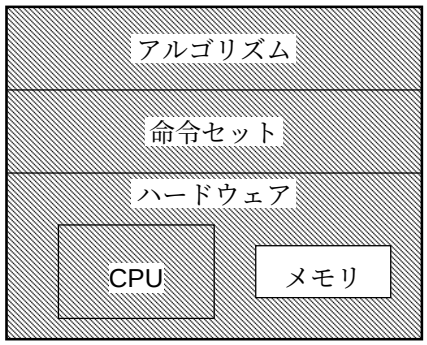

図 1: 設計要素の内訳

3 課題の詳細

3.1 設計対象

2 で述べたように, この課題での設計は大きく分けて図 1 のように分類されます. 以下, それぞれについて解説します.

3.1.1 ハードウェアの設計

メモリの構成 ハードウェアの設計ではメモリに制約を設けています. 合計のバンド幅が最高 32 bit/clock であるという条件を実現するため, この課題専用のメモリの declare 記述, circuit 記述, pcd 記述をコンテスト事務局が用意します. 従って, 設計, 動作シミュレーション, 論理合成にはこれを使用することとします.

このメモリは図 2 のように, アドレスのビット数が 16 bit, 1 アドレスの記憶サイズが 1 bit のメモリユニットを 32 個並べたものとなっています. メモリの具体的な SFL 記述を A.1, A.2 に示します. このメモリの使用については, その束ね方は自由ですし, また動作中に束ね方を変更することも自由に行って構いません. 例えば 16 bit のメモリが 2 つあるようなハーバードアーキテクチャを最初に構成し, 命令によって 8 bit のメモリが 4 つあるように再構成して使用しても構わないということです.

標準アドレス空間による制約 上のように自由度を高くすると審査のための処理結果の観測が困難になりますので, メモリの使用に際して以下の制約を設けます. その制約とは,

> **処理対象のデータの初期値のセットと処理結果の観測を標準アドレス空間によって行えるようにする**

というものです. ここで標準アドレス空間とは, アドレスのビット数が 17 bit のアドレス空間とし, 1 アドレスの記憶サイズは 16 bit であるとします (図 4 を参照してください). 標準アドレス空間によってデータセットと結果観測を行えるようにするために, 標準アドレス空間のワードに対して図 2 のメモリユニットの番号とアドレスの組の対応を定義してください. ですから, 具体的な制約としては, この対応を応募者が規定しなければならない, ということになります. また, この対応はデータセット時と結果観測時で同じ規則に基づくものでなければならず, 処理内容に関わらず同じ規則で

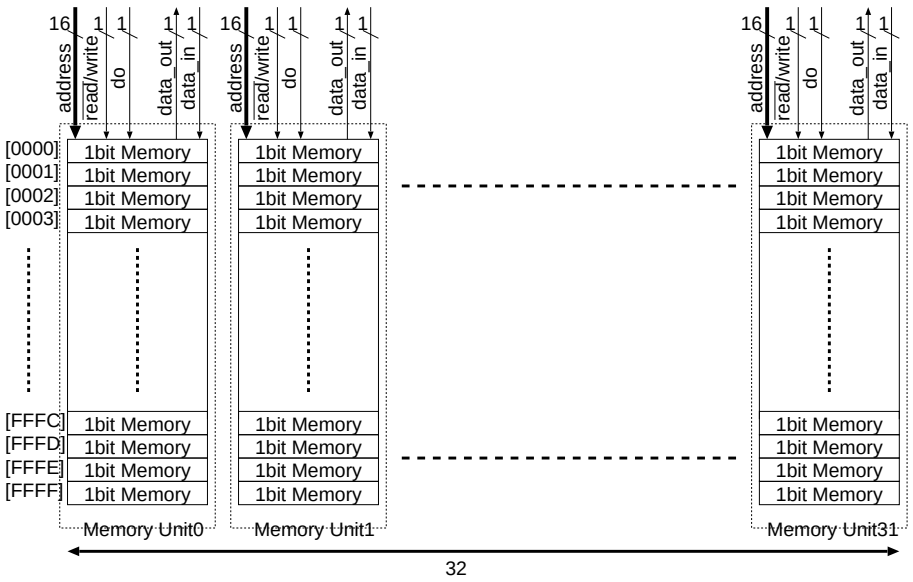

図 2: メモリの基本構成

なければならないとします. ただし, この制約はデータセット時と結果観測時においてのみの制約なので, プログラムによる処理中に一時的に対応が定義不可能になったり, 意味を持たなくなるのは構いません.

メモリの使用例　代表的な使用例として, ハーバードアーキテクチャの構成法を説明します. メモリアクセスのバンド幅が 32 bit/clock なので, 命令メモリへのアクセスに 16 bit を使用し, データメモリへのアクセスに 16 bit を使用することになります. この場合, 図 3 のように 16 個の 1 bit メモリユニットに命令メモリ用のアドレス線等を接続し, 残った 16 個の 1 bit メモリユニットにデータメモリ用のアドレス線等を接続すれば実現できます. この利用形態での SFL 記述を A.3 に示します. CPU からデータメモリを読み出す場合, アクセスするアドレスを adr にセットし, read_data 端子をアサートすると, memory_module m を構成するメモリユニットのうち, 16 個の制御入力端子がアサートされます. これによって 16 個の各メモリユニットから 1 bit のデータが読み出されます. これらのデータ出力を束ねたものが data_in 端子の内容となります.

一方 CPU からデータメモリに書き込む場合, 書き込むアドレスを adr に, 書き込む内容を data_out にセットした上で read_data 端子をアサートします. すると adr は全てのメモリユニットに送信され, data_out 端子の内容は各メモリユニットに分配されて書き込まれます. 命令メモリについても同様で, read_instruction , write_instruction でメモリの読み出し, 書き込みを行えます. このような中間の制御端子を設けることで, CPU 側からはメモリを 16 bit 幅のメモリ 2 つとして扱えるようになります.

この例での標準アドレス空間との対応は例えば図 4 のようなものが考えられます. 標準アドレス空間でのアドレス x におけるワードを STD[x] と表記し, メモリユニット番号 n とそのアドレス y の組を REAL[n,y] と表記するとすると, この対応関係は

x <0x10000 のとき,

$$\mathrm{STD}\,[x] => \mathrm{REAL}\,[15,x]\,,\mathrm{REAL}\,[14,x]\,,\cdots\mathrm{REAL}\,[0,x]$$

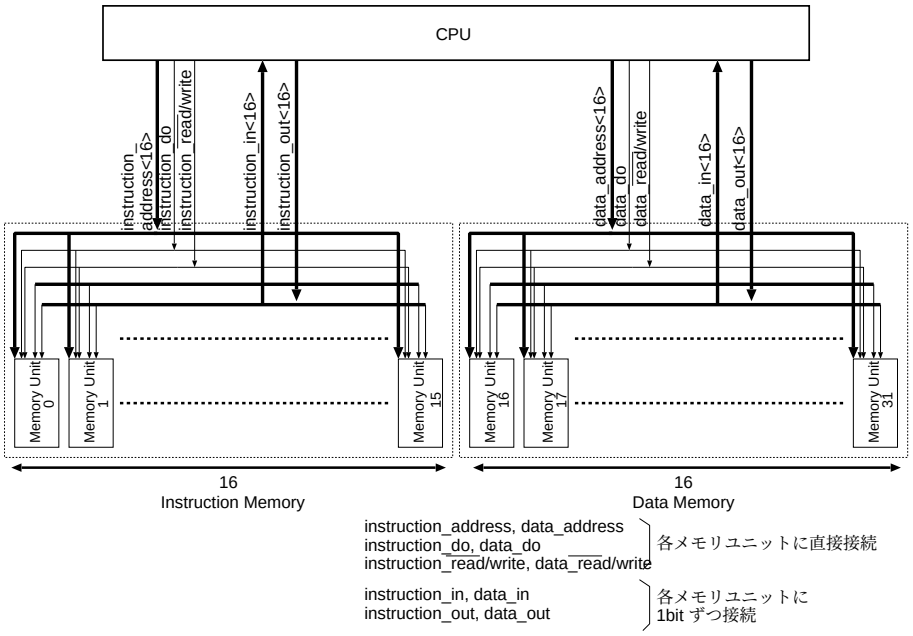

図 3: 命令メモリとデータメモリへの分割の構成例

$x \geq$0x10000 のとき,

$$\mathrm{STD}\,[x] => \mathrm{REAL}\,[31, x - 0\mathrm{x}10000]\,, \mathrm{REAL}\,[30, x - 0\mathrm{x}10000]\,, \cdots \mathrm{REAL}\,[16, x - 0\mathrm{x}10000]$$

(ただし, REAL のほうは MSB から順に並べてあります.)
となります.

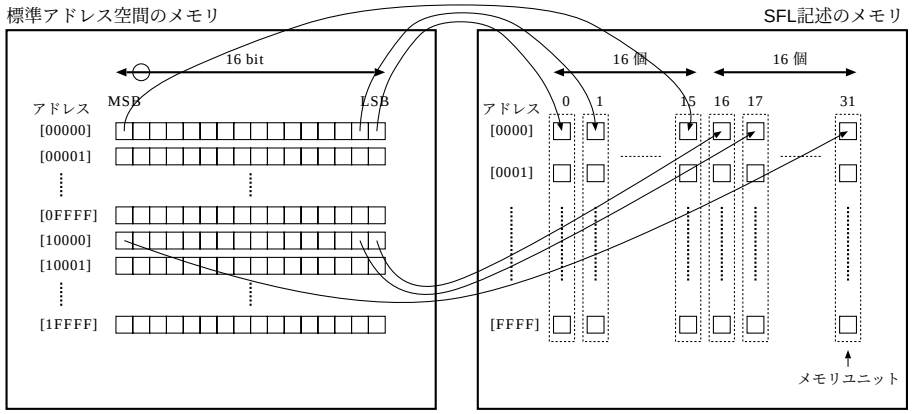

図 4: ハーバードアーキテクチャでの標準アドレス空間と SFL のメモリの対応例

論理合成の条件　最後に, 設計したハードウェアの論理合成に際しての条件を述べます. 論理合成には PARTHENON システムを利用しますが, ライブラリは PARTHENON システムに標準で付属の DEMO 社 demo ライブラリに, コンテスト事務局で用意したメモリの pcd 記述を加えたものを使用

するとします. このメモリについては, ゲート数, 消費電力, 面積が 0 として合成されるような pcd 記述となっています. この条件下で, メモリも含めて論理合成を行うとします.

3.1.2 命令セットの設計

命令セット, データの表現方法等については制約はありません. 第三者が理解可能なように, 別途ドキュメントで定義してください.

3.1.3 プログラムの設計

プログラムについてはその処理そのものと処理の終了について以下のように規定します. 解くべき問題の具体的な処理は, 3.1.1 で定義した標準アドレス空間を用いて以下のように規定します.

ソート 標準アドレス空間 0x8000 から 0x80ff までにランダムに 16 bit の正の整数データ 256 個がならんでいるとする. この 256 個のデータを昇順にソートし, その結果を標準アドレス空間の 0x8100 以降に昇順に配置する処理.

素数 1 から $1023(2^{10}-1)$ までの素数を算出し, その結果を標準アドレス空間の 0x8000 から順に昇順に配置する処理.

G.C.D 標準アドレス空間 0x8000 に 1 つめの正の 16 bit 整数をおき, 0x8001 に 2 つめの正の 16 bit 整数をおく. この 2 数の最大公約数を計算し, 結果を 0x8002 に出力する処理.

問題を解くためのアルゴリズムは規定していませんので, 設計するハードウェアおよび命令セットの特長を生かせるプログラムを採用してください.

次に処理の終了ですが, 上記のそれぞれの処理が完了したときには, コンテスト事務局で用意したメモリモジュールに含まれる complete という制御端子 (付録の A.2 を参照してください) を起動するようにしてください. complete 制御端子を起動すると, メモリモジュール内部でエラーが発生し, これにより SECONDS のシミュレーションが強制的に終了します.

3.2 シミュレーション環境

3.1 で述べた, 種々の制約のもとでの動作確認のためのシミュレーション環境について説明します.

シミュレーション環境の全体像を図 5 に示します. 図 5 の中で, 斜線の部分を応募者が設計, 記述します.

1. まず, データの初期状態を記述したファイルと, 処理後のデータの配置場所を記述したファイルを用いて, 標準アドレス空間でのデータの配置を記述します.

2. これらのファイルを入力として, C プログラムによりデータの配置を初期化する SECONDS のスクリプトと処理後のメモリ上のデータを出力するための SECONDS のスクリプトを出力します. この C プログラムには, 標準アドレス空間から SFL 記述のメモリユニット番号とアドレスの組への変換関数である map_data 関数 (3.1.1 を参照してください.) が含まれています. この関数は設計するアーキテクチャに依存しますので, 応募者が記述してください. つまりこの map_data 関数以外はコンテスト事務局で作成し提供します.

3. C プログラムで出力されたスクリプトは SECONDS のシミュレーションの準備のうち, 処理対象データの配置の初期化処理などに利用されます. 一方で, 命令などの配置の初期化は SECONDS のスクリプトとして別途応募者が記述してください. このような内訳で構成された SECONDS のスクリプトと, 設計したハードウェアの SFL 記述を入力としてシミュレーションを行い, 結果の正しさを確認します.

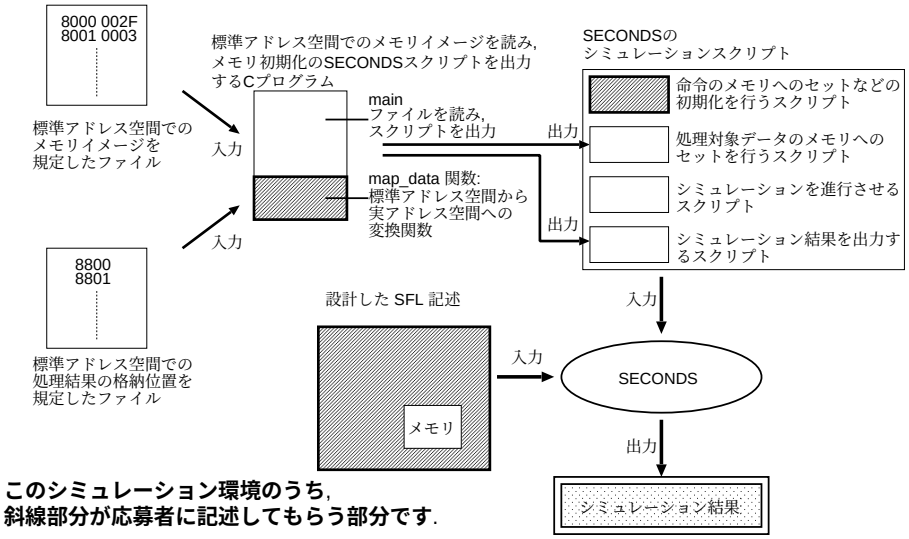

図 5: シミュレーション環境の概要

4 設計結果の評価方法

設計結果の評価は以下の観点から行います.

1. 論理合成結果に対する, それぞれの問題の処理の速さ
2. 論理合成結果に対する, 消費電力, 面積の小ささ
3. 設計したアーキテクチャの独創性
4. 設計内容について記述したドキュメント

処理の速さは次のように定義します. 論理合成結果からクリティカルパスを求め, そのパスの遅延時間 t を 1 クロックサイクルとします. 次にそれぞれの問題の処理に要するクロック数 cl を計測します. このクロック数とは, SECONDS におけるクロック 0 をスタートとして, 最終的な処理結果がメモリに格納され, complete 信号がアサートされるまでのクロック数とします. なお, 初期状態では処理対象となるデータ, および処理を記述した命令列はあらかじめメモリ上の適切な位置に格納しておいて良いとします.

これらの t, cl を用いて, 処理時間 T を次のように定義します.

$$T = t \times cl \tag{1}$$

T が小さいものを処理が速いものとします.

消費電力と面積は PARTHENON システムの合成結果で判断するものとします. また, アーキテクチャの独創性, ドキュメントについてはその内容について個々に評価します.

5 ドキュメントについて

ドキュメントには以下の内容が最低限含まれていることとします.

1. アーキテクチャ, 及びアーキテクチャの命令セット
2. 問題の処理を行うアセンブリコード あるいはコードシーケンス
3. 動作の検証方法
4. 動作周波数, 消費電力, 面積
5. 3 つの問題それぞれの処理に要するクロック数

6 提出物

この課題の応募に必要な提出物は以下のとおりです.

1. 課題要件を満たす SFL 記述
2. アドレス空間変換関数 map_data の C 言語のソースコード
3. 命令をメモリにセットするための SECONDS の初期化スクリプト
4. 設計内容について記述したドキュメント

A　メモリの記述

A.1　メモリの declare 記述

```
declare memory_module {
  input  i0,  i1,  i2,  i3,  i4,  i5,  i6,  i7;
  input  i8,  i9, i10, i11, i12, i13, i14, i15;
  input i16, i17, i18, i19, i20, i21, i22, i23;
  input i24, i25, i26, i27, i28, i29, i30, i31;

  output  o0,  o1,  o2,  o3,  o4,  o5,  o6,  o7;
  output  o8,  o9, o10, o11, o12, o13, o14, o15;
  output o16, o17, o18, o19, o20, o21, o22, o23;
  output o24, o25, o26, o27, o28, o29, o30, o31;

  instrin  do0,  do1,  do2,  do3,  do4,  do5,  do6,  do7;
  instrin  do8,  do9, do10, do11, do12, do13, do14, do15;
  instrin do16, do17, do18, do19, do20, do21, do22, do23;
  instrin do24, do25, do26, do27, do28, do29, do30, do31;
  instrin complete;

  input  rn_we0,  rn_we1,  rn_we2,  rn_we3,  rn_we4,  rn_we5,  rn_we6,  rn_we7;
  input  rn_we8,  rn_we9, rn_we10, rn_we11, rn_we12, rn_we13, rn_we14, rn_we15;
  input rn_we16, rn_we17, rn_we18, rn_we19, rn_we20, rn_we21, rn_we22, rn_we23;
  input rn_we24, rn_we25, rn_we26, rn_we27, rn_we28, rn_we29, rn_we30, rn_we31;

  input  adr0<16>,  adr1<16>,  adr2<16>,  adr3<16>;
  input  adr4<16>,  adr5<16>,  adr6<16>,  adr7<16>;
  input  adr8<16>,  adr9<16>, adr10<16>, adr11<16>;
  input adr12<16>, adr13<16>, adr14<16>, adr15<16>;
  input adr16<16>, adr17<16>, adr18<16>, adr19<16>;
  input adr20<16>, adr21<16>, adr22<16>, adr23<16>;
  input adr24<16>, adr25<16>, adr26<16>, adr27<16>;
  input adr28<16>, adr29<16>, adr30<16>, adr31<16>;

  instr_arg  do0(  rn_we1,  adr1,  i1); instr_arg  do1(  rn_we1,  adr1,i1);
  instr_arg  do2(  rn_we2,  adr2,  i2); instr_arg  do3(  rn_we3,  adr3,  i2);
  instr_arg  do4(  rn_we4,  adr4,  i4); instr_arg  do5(  rn_we5,  adr5,  i5);
  instr_arg  do6(  rn_we6,  adr6,  i6); instr_arg  do7(  rn_we7,  adr7,  i7);
  instr_arg  do8(  rn_we8,  adr8,  i8); instr_arg  do9(  rn_we9,  adr9,  i9);
  instr_arg do10( rn_we10, adr10, i10); instr_arg do11( rn_we11, adr11, i11);
  instr_arg do12( rn_we12, adr12, i12); instr_arg do13( rn_we13, adr13, i13);
  instr_arg do14( rn_we14, adr14, i14); instr_arg do15( rn_we15, adr15, i15);
  instr_arg do16( rn_we16, adr16, i16); instr_arg do17( rn_we17, adr17, i17);
  instr_arg do18( rn_we18, adr18, i18); instr_arg do19( rn_we19, adr19, i19);
  instr_arg do20( rn_we20, adr20, i20); instr_arg do21( rn_we21, adr21, i21);
  instr_arg do22( rn_we22, adr22, i22); instr_arg do23( rn_we23, adr23, i23);
  instr_arg do24( rn_we24, adr24, i24); instr_arg do25( rn_we25, adr25, i25);
  instr_arg do26( rn_we26, adr26, i26); instr_arg do27( rn_we27, adr27, i27);
  instr_arg do28( rn_we28, adr28, i28); instr_arg do29( rn_we29, adr29, i29);
  instr_arg do30( rn_we30, adr30, i30); instr_arg do31( rn_we31, adr31, i31);

}
```

A.2　メモリの circuit 記述

```
circuit memory_module {
  input  i0,  i1,  i2,  i3,  i4,  i5,  i6,  i7;
  input  i8,  i9, i10, i11, i12, i13, i14, i15;
  input i16, i17, i18, i19, i20, i21, i22, i23;
  input i24, i25, i26, i27, i28, i29, i30, i31;

  output  o0,  o1,  o2,  o3,  o4,  o5,  o6,  o7;
  output  o8,  o9, o10, o11, o12, o13, o14, o15;
```

```
output o16, o17, o18, o19, o20, o21, o22, o23;
output o24, o25, o26, o27, o28, o29, o30, o31;

instrin  do0,  do1,  do2,  do3,  do4,  do5,  do6,  do7;
instrin  do8,  do9, do10, do11, do12, do13, do14, do15;
instrin do16, do17, do18, do19, do20, do21, do22, do23;
instrin do24, do25, do26, do27, do28, do29, do30, do31;
instrin complete;

input  rn_we0,  rn_we1,  rn_we2,  rn_we3,  rn_we4,  rn_we5,  rn_we6,  rn_we7;
input  rn_we8,  rn_we9, rn_we10, rn_we11, rn_we12, rn_we13, rn_we14, rn_we15;
input rn_we16, rn_we17, rn_we18, rn_we19, rn_we20, rn_we21, rn_we22, rn_we23;
input rn_we24, rn_we25, rn_we26, rn_we27, rn_we28, rn_we29, rn_we30, rn_we31;

input  adr0<16>,  adr1<16>,  adr2<16>,  adr3<16>;
input  adr4<16>,  adr5<16>,  adr6<16>,  adr7<16>;
input  adr8<16>,  adr9<16>, adr10<16>, adr11<16>;
input adr12<16>, adr13<16>, adr14<16>, adr15<16>;
input adr16<16>, adr17<16>, adr18<16>, adr19<16>;
input adr20<16>, adr21<16>, adr22<16>, adr23<16>;
input adr24<16>, adr25<16>, adr26<16>, adr27<16>;
input adr28<16>, adr29<16>, adr30<16>, adr31<16>;

sel   stop;

mem  m0[65536],  m1[65536],  m2[65536],  m3[65536];
mem  m4[65536],  m5[65536],  m6[65536],  m7[65536];
mem  m8[65536],  m9[65536], m10[65536], m11[65536];
mem m12[65536], m13[65536], m14[65536], m15[65536];
mem m16[65536], m17[65536], m18[65536], m19[65536];
mem m20[65536], m21[65536], m22[65536], m23[65536];
mem m24[65536], m25[65536], m26[65536], m27[65536];
mem m28[65536], m29[65536], m30[65536], m31[65536];

stop = 0b0;

instruct  do0  any { rn_we0  : m0[adr0]  :=  i0;  else : o0  =  m0[adr0]; }
instruct  do1  any { rn_we1  : m1[adr1]  :=  i1;  else : o1  =  m1[adr1]; }
instruct  do2  any { rn_we2  : m2[adr2]  :=  i2;  else : o2  =  m2[adr2]; }
instruct  do3  any { rn_we3  : m3[adr3]  :=  i3;  else : o3  =  m3[adr3]; }
instruct  do4  any { rn_we4  : m4[adr4]  :=  i4;  else : o4  =  m4[adr4]; }
instruct  do5  any { rn_we5  : m5[adr5]  :=  i5;  else : o5  =  m5[adr5]; }
instruct  do6  any { rn_we6  : m6[adr6]  :=  i6;  else : o6  =  m6[adr6]; }
instruct  do7  any { rn_we7  : m7[adr7]  :=  i7;  else : o7  =  m7[adr7]; }
instruct  do8  any { rn_we8  : m8[adr8]  :=  i8;  else : o8  =  m8[adr8]; }
instruct  do9  any { rn_we9  : m9[adr9]  :=  i9;  else : o9  =  m9[adr9]; }
instruct  do10 any { rn_we10 : m10[adr10]:=  i10; else : o10 =  m10[adr10]; }
instruct  do11 any { rn_we11 : m11[adr11]:=  i11; else : o11 =  m11[adr11]; }
instruct  do12 any { rn_we12 : m12[adr12]:=  i12; else : o12 =  m12[adr12]; }
instruct  do13 any { rn_we13 : m13[adr13]:=  i13; else : o13 =  m13[adr13]; }
instruct  do14 any { rn_we14 : m14[adr14]:=  i14; else : o14 =  m14[adr14]; }
instruct  do15 any { rn_we15 : m15[adr15]:=  i15; else : o15 =  m15[adr15]; }
instruct  do16 any { rn_we16 : m16[adr16]:=  i16; else : o16 =  m16[adr16]; }
instruct  do17 any { rn_we17 : m17[adr17]:=  i17; else : o17 =  m17[adr17]; }
instruct  do18 any { rn_we18 : m18[adr18]:=  i18; else : o18 =  m18[adr18]; }
instruct  do19 any { rn_we19 : m19[adr19]:=  i19; else : o19 =  m19[adr19]; }
instruct  do20 any { rn_we20 : m20[adr20]:=  i20; else : o20 =  m20[adr20]; }
instruct  do21 any { rn_we21 : m21[adr21]:=  i21; else : o21 =  m21[adr21]; }
instruct  do22 any { rn_we22 : m22[adr22]:=  i22; else : o22 =  m22[adr22]; }
instruct  do23 any { rn_we23 : m23[adr23]:=  i23; else : o23 =  m23[adr23]; }
instruct  do24 any { rn_we24 : m24[adr24]:=  i24; else : o24 =  m24[adr24]; }
instruct  do25 any { rn_we25 : m25[adr25]:=  i25; else : o25 =  m25[adr25]; }
instruct  do26 any { rn_we26 : m26[adr26]:=  i26; else : o26 =  m26[adr26]; }
instruct  do27 any { rn_we27 : m27[adr27]:=  i27; else : o27 =  m27[adr27]; }
instruct  do28 any { rn_we28 : m28[adr28]:=  i28; else : o28 =  m28[adr28]; }
instruct  do29 any { rn_we29 : m29[adr29]:=  i29; else : o29 =  m29[adr29]; }
instruct  do30 any { rn_we30 : m30[adr30]:=  i30; else : o30 =  m30[adr30]; }
```

```
  instruct  do31 any { rn_we31 : m31[adr31]:=  i31; else : o31 =  m31[adr31]; }

  instruct complete stop = 0b1;

}
```

A.3　メモリを 2 分割して使用する記述例

```
%i "memmod.h"

module access {
  sel adr<16>, data_in<16>, data_out<16>,
      instruction_in<16>, instruction_out<16>;

  instrself read_data, write_data, read_instruction, write_instruction;
  memory_module m;

  instr_arg read_data(adr);
  instr_arg write_data(adr, data_out);
  instr_arg read_instruction(adr);
  instr_arg write_instruction(adr, instruction_out);


  instruct read_data par {
    data_in  = m.do15( 0b0, adr, 0b0 ).o15 || m.do14( 0b0, adr, 0b0 ).o14
             || m.do13( 0b0, adr, 0b0 ).o13 || m.do12( 0b0, adr, 0b0 ).o12
             || m.do11( 0b0, adr, 0b0 ).o11 || m.do10( 0b0, adr, 0b0 ).o10
             || m.do9( 0b0, adr, 0b0 ).o9   || m.do8 ( 0b0, adr, 0b0 ).o8
             || m.do7( 0b0, adr, 0b0 ).o7   || m.do6 ( 0b0, adr, 0b0 ).o6
             || m.do5( 0b0, adr, 0b0 ).o5   || m.do4 ( 0b0, adr, 0b0 ).o4
             || m.do3( 0b0, adr, 0b0 ).o3   || m.do2 ( 0b0, adr, 0b0 ).o2
             || m.do1( 0b0, adr, 0b0 ).o1   || m.do0 ( 0b0, adr, 0b0 ).o0 ;
  }

  instruct write_data par {
    m.do0 ( 0b1, adr, data_out<0> );
    m.do1 ( 0b1, adr, data_out<1> );
    m.do2 ( 0b1, adr, data_out<2> );
    m.do3 ( 0b1, adr, data_out<3> );
    m.do4 ( 0b1, adr, data_out<4> );
    m.do5 ( 0b1, adr, data_out<5> );
    m.do6 ( 0b1, adr, data_out<6> );
    m.do7 ( 0b1, adr, data_out<7> );
    m.do8 ( 0b1, adr, data_out<8> );
    m.do9 ( 0b1, adr, data_out<9> );
    m.do10( 0b1, adr, data_out<10>);
    m.do11( 0b1, adr, data_out<11>);
    m.do12( 0b1, adr, data_out<12>);
    m.do13( 0b1, adr, data_out<13>);
    m.do14( 0b1, adr, data_out<14>);
    m.do15( 0b1, adr, data_out<15>);
  }

  instruct read_instruction par {
    instruction_in  =
               m.do31( 0b0, adr, 0b0 ).o31 || m.do30( 0b0, adr, 0b0 ).o30
             || m.do29( 0b0, adr, 0b0 ).o29 || m.do28( 0b0, adr, 0b0 ).o28
             || m.do27( 0b0, adr, 0b0 ).o27 || m.do26( 0b0, adr, 0b0 ).o26
             || m.do25( 0b0, adr, 0b0 ).o25 || m.do24( 0b0, adr, 0b0 ).o24
             || m.do23( 0b0, adr, 0b0 ).o23 || m.do22( 0b0, adr, 0b0 ).o22
             || m.do21( 0b0, adr, 0b0 ).o21 || m.do20( 0b0, adr, 0b0 ).o20
             || m.do19( 0b0, adr, 0b0 ).o19 || m.do18( 0b0, adr, 0b0 ).o18
             || m.do17( 0b0, adr, 0b0 ).o17 || m.do16( 0b0, adr, 0b0 ).o16 ;
```

```
  }
  instruct write_instruction par {
    m.do16( 0b1, adr, instruction_out<0> );
    m.do17( 0b1, adr, instruction_out<1> );
    m.do18( 0b1, adr, instruction_out<2> );
    m.do19( 0b1, adr, instruction_out<3> );
    m.do20( 0b1, adr, instruction_out<4> );
    m.do21( 0b1, adr, instruction_out<5> );
    m.do22( 0b1, adr, instruction_out<6> );
    m.do23( 0b1, adr, instruction_out<7> );
    m.do24( 0b1, adr, instruction_out<8> );
    m.do25( 0b1, adr, instruction_out<9> );
    m.do26( 0b1, adr, instruction_out<10>);
    m.do27( 0b1, adr, instruction_out<11>);
    m.do28( 0b1, adr, instruction_out<12>);
    m.do29( 0b1, adr, instruction_out<13>);
    m.do30( 0b1, adr, instruction_out<14>);
    m.do31( 0b1, adr, instruction_out<15>);
  }

}
```

B　各問題の代表的なアルゴリズム

この課題では, それぞれの問題を解くためのアルゴリズムについては規定されていません. それぞれのアーキテクチャに最適なアルゴリズムを採用してください. ただし, それぞれの問題には良く知られたアルゴリズムがありますので, それを紹介します.

B.1　ソート

ソートでは一般的にクイックソートが高速であるとされています. 例として, 5, 25, 20, 14, 2, 39, 9, 8 と並んだ数列を昇順にソートすることを考えます. クイックソートでは図 6 の手順でソートされます. まず基準値を決定します. 今回は 14 とします. そして数列の先頭からは基準値より大きいものを検索し, 末尾からは基準値より小さいものを検索します. すると, 先頭からは 2 番目の 25 が, 末尾からは 8 が条件に合います. この条件にあったものを交換します. その後同様の検索, 交換を行うと, 双方の検索が衝突します. 衝突したところで数列を 2 分割し, それぞれに対して同じ手順を再帰的に適用します. この手順で数列を昇順にソートできます. これを実行する C 言語の記述例を B.4.1 に示します.

B.2　素数の求め方

素数を求めるにはエラトステネスのふるいというアルゴリズムが使用されます. これは自然数を 1 から順に並べ, 小さい素数の倍数を除いていき, 最後まで残ったものが素数である, というものです. 例として, 20 までの自然数の中から素数を求めるものを考えてみます. まず 1 は素数ではないので対象からは除きます. これから消去を始めます. 最初に, 2 の倍数はすべて素数ではないのでこれをすべて消去します. この段階で候補は図 7 の 1 のように 2, 3, 5, 7, 9, 11, 13, 15, 17, 19 となります. 次に, 3 の倍数を消去します. すると 2, 3, 5, 7, 11, 13, 17, 19 となります. 以降の数字についても同様に消去しますが, これ以降の数字については消去可能なものがないため, これで消去は完了

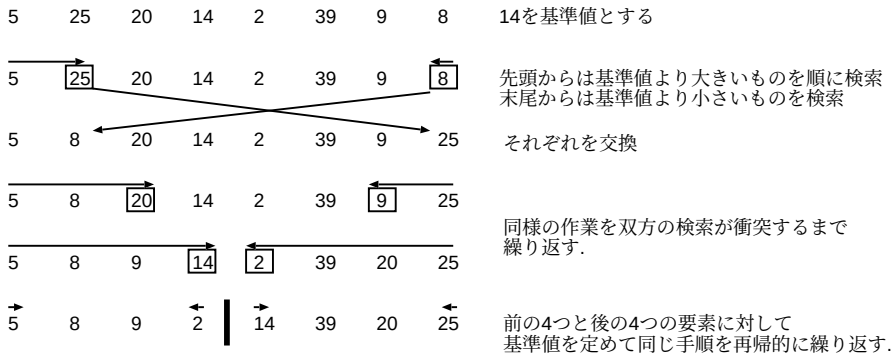

図 6: クイックソートの実行例

です. この 8 つの数字が答えとなります. このアルゴリズムの C 言語による記述例を B.4.2 に示します.

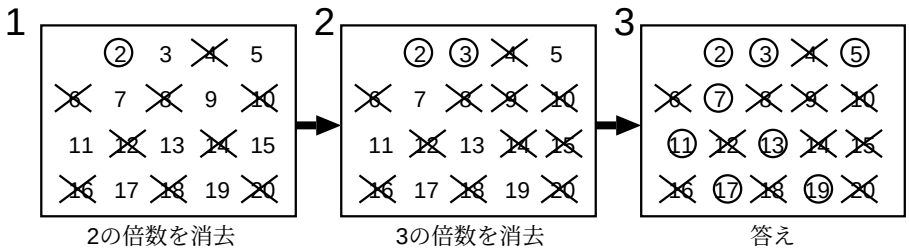

図 7: エラトステネスのふるいの実行例

B.3 最大公約数の求め方

2 数の最大公約数の求め方としてはユークリッドの互除法が有名ですが, ここではその派生の最も簡単なものを紹介します. まず, 2 数 $m, n(m > n)$ の最大公約数が r であるとすると,

$$m = rp, n = rq(p \text{ と } q \text{ は互いに素}, p > q) \tag{2}$$

と表せます. この時, $m - n = r(p - q)$ と $n = rq$ の最大公約数も r であるといえます. $m - n$ と n にこの手法を再帰的に適用すると, 最終的に一方が 0 となります. このときの残ったもう一方の数字が最大公約数となります. 例として 24 と 36 での計算例を図 8 に示します. また, このアルゴリズムの C 言語での記述例を B.4.3 に示します.

B.4 ソースコード

B.4.1 クイックソートの C 言語による記述例

```
void quicksort(int a[], int first, int last);
int main(int argc, char **argv) {
  int source_data[] = { 5, 25, 20, 14, 2, 39, 9, 8 } ;
```

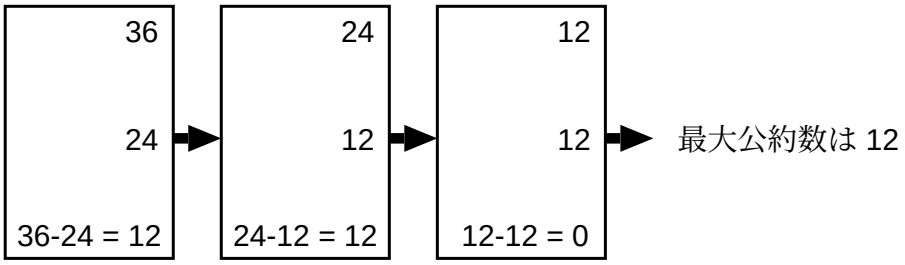

図 8: ユークリッドの互除法の計算例

```
  quicksort(source_data, 0, 7);
  return 0;
}

void quicksort(int a[], int first, int last) {
  int i,j,pivot,tmp;
  pivot = a[(first+last)/2];
  i = first, j = last;
  while (1) {
    while ( a[i]  < pivot   ) ++i;
    while ( pivot < a[j]    ) --j;
    if ( i >= j ) break;
    tmp  = a[i];
    a[i] = a[j];
    a[j] = tmp ;
    ++i;
    --j;
  }
  if ( first < i-1 ) quicksort(a, first, i-1 );
  if ( j+1   < last) quicksort(a, j+1  , last);
}
```

B.4.2　エラトステネスのふるいの C 言語による記述例

```
int main(int argc, char **argv) {
   int i,j;
   int a[21] = { 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0 };
   for ( i = 2 ; i < 21 ; ++i ) {
        if ( !a[i] ) {
            for ( j = i + i ; j < 21 ; j += i ) a[j] = 1;
        }
   }
   for ( i = 2 ; i < 21 ; ++i ) {
        if ( !a[i] ) { i は素数です. }
   }
   return 0;
}
```

B.4.3　ユークリッドの互除法の C 言語による記述例

```
void euclid( int *m, int *n );

int main(int argc, char **argv) {
```

```
  int m = 24, n = 36 ;
  int gcm;

  while ( m != 0 && n != 0 ) euclid( &m, &n );
  if      ( m == 0 ) gcm = n;
  else if ( n == 0 ) gcm = m;

  { gcm が最大公約数です. }

  return 0;
}

void euclid( int *m, int *n ) {
  int large, small;

  if ( *m >= *n ) { large = *m ; small = *n ; }
  if ( *m <  *n ) { large = *n ; small = *m ; }
  *m = large - small;
  *n = small;
}
```