2012年度
計算機システム演習 第10回

計算機システムTA　福田圭祐（松岡研究室）

連絡事項

- 組み立て演習について（再）
  - 7/20（金）に組み立て演習を行います
  - 機材の準備都合上、なるべく参加人数を正確に把握したいので、ご協力ください
- 次回～学期末まで、Wiki上で演習になっている回のうち、数回は講義に振り替わる予定です。
- 課題の最終提出日は、期末試験前後になる予定です（詳細は未定）

MIPSシミュレータ構築の流れ

1. ALUの作成
2. レジスタファイル
3. メモリ領域
   - 命令用メモリ
   - データ用メモリ
4. PCの作成
5. メインコントロールユニット
6. ALUコントロールユニット
7. 機能拡張
   - メモリアクセス命令
   - 分岐命令

課題1

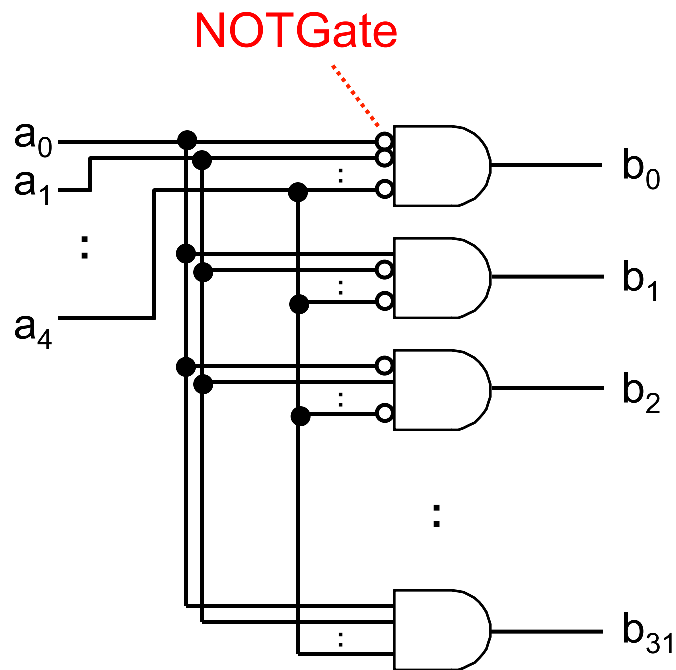
- レジスタファイルを完成させよ
  - Register、Decoder5、MUX32_bus、RegisterFile
  - RegisterFileDriver クラスを作り、テストを行うこと
    - 同じレジスタへの書き込みと読み出しを行うテストすること
      - run()を2回実行する必要がある

課題2

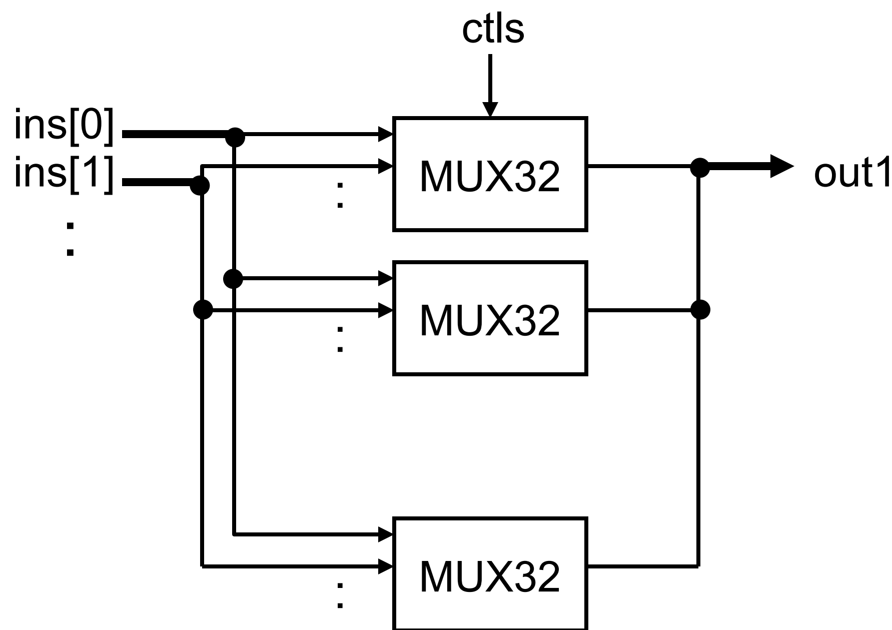
- メモリを完成させよ
  - InstMemory、DataMemory
  - InstMemoryDriver、DataMemoryDriver を作り、テストを行うこと
    - データを書き込んだ後に正しく読み出せることを確認すること

課題3（オプション）

- Decoder5 クラスおよび MUX32_bus クラスを回路を使って実現せよ



Decoder5



MUX32_bus

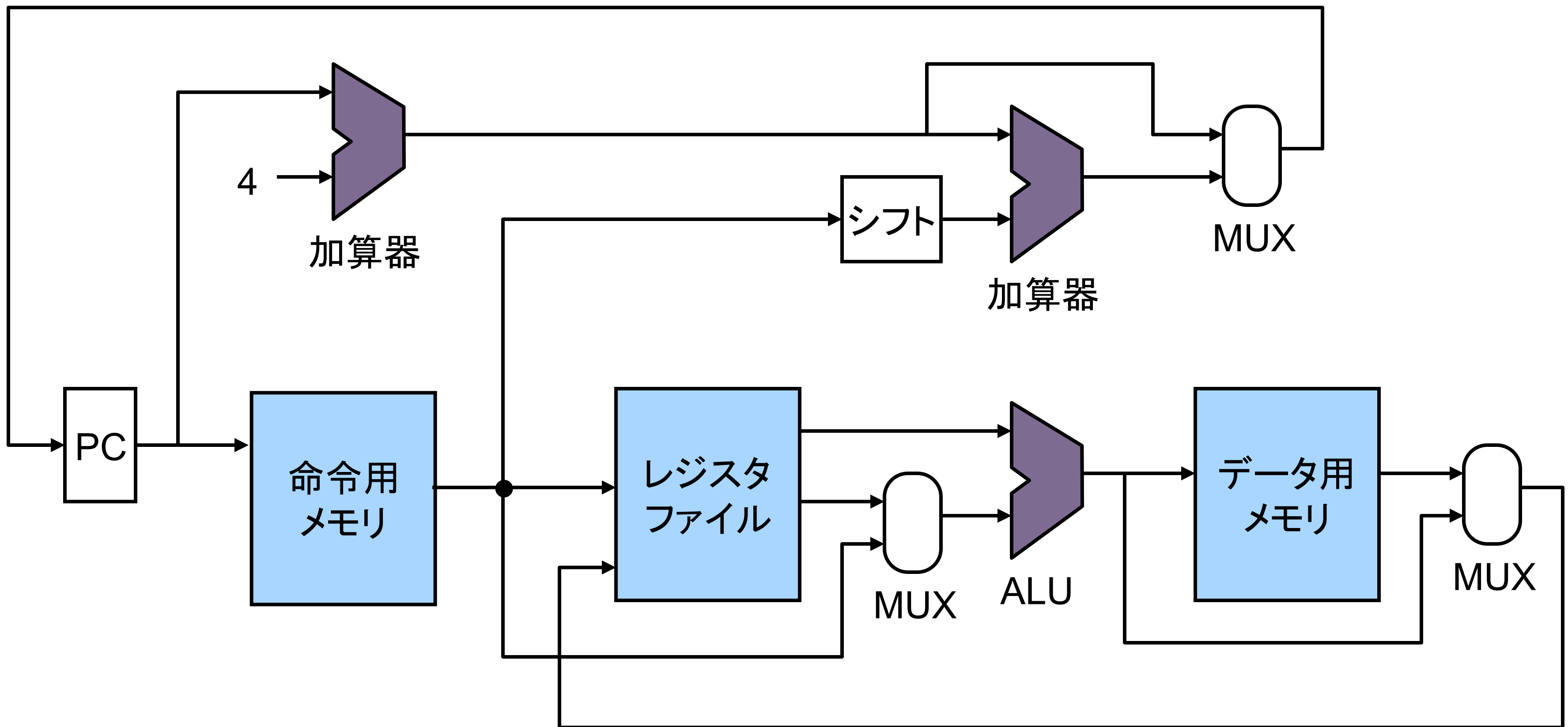
課題4（オプション）

- レジスタをフリップフロップを使って作成し直してみよ
  - D-フリップフロップを配列として作成
    - 1ビットの情報を保持する
    - D-フリップフロップはD-ラッチの組み合わせ回路
  - 制御入力としてクロックを入れる

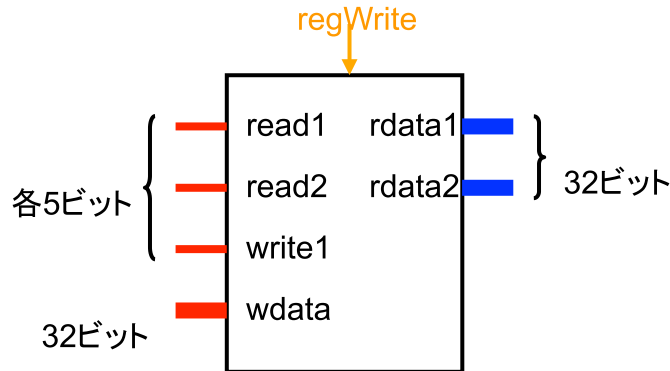
課題4（オプション）

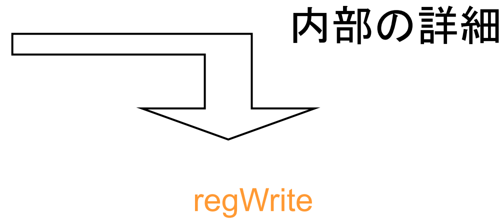
- レジスタをフリップフロップを使って作成し直してみよ
  - D-フリップフロップを配列として作成
    - 1ビットの情報を保持する
    - D-フリップフロップはD-ラッチの組み合わせ回路
  - 制御入力としてクロックを入れる

MIPSシミュレータの全体像



レジスタファイルの概要





regWrite

read1

read2

デコーダ

0
1
:
31

write1

wdata

0

レジスタ0

レジスタ1

:

レジスタ31

:

MUX

rdata1

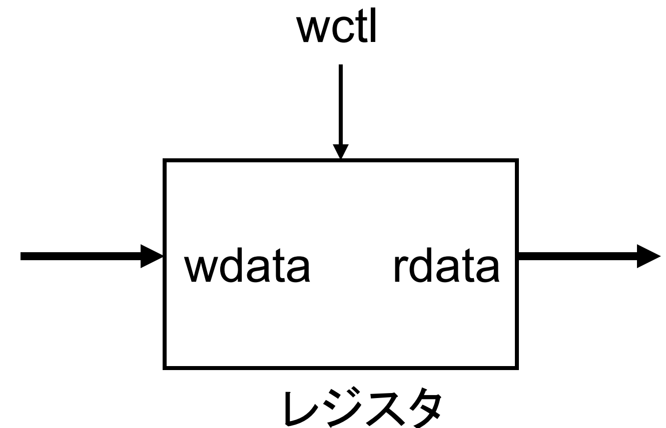
:

MUX

rdata2

レジスタ

- 状態回路で構成される
  - 内部状態を持ち、入力と状態により出力が決まる回路
    - 組み合わせ回路では入力のみから出力が決まった
- 入力
  - wctl : 書き込み制御フラグ(1bit)
  - wdata : 書き込むデータ(32bit)
- 出力
  - rdata : 読み出したデータ(32bit)

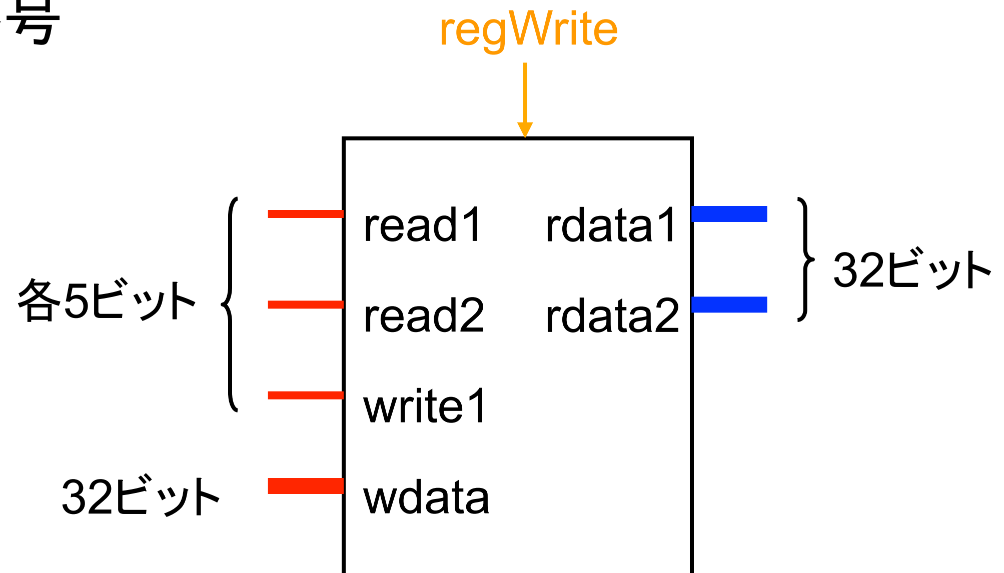


Register クラス

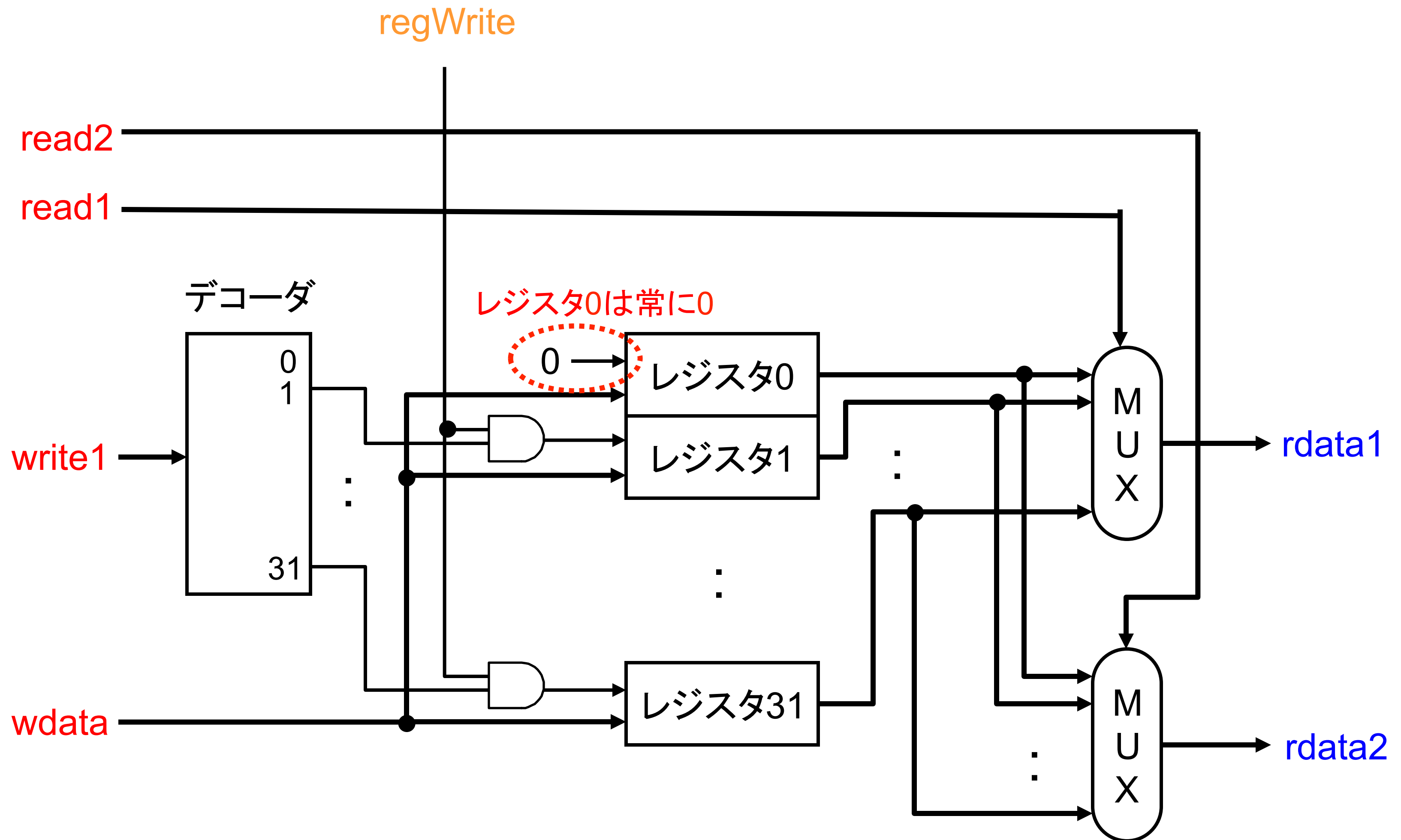
```
// 回路を用いずに作成してよい
public class Register {
  int val;  // レジスタの値
  //必要であれば他のフィールドも宣言
  public Register(Path wctl,     // 制御入力
                  Bus wdata,     // 入力
                  Bus rdata) {   // 出力
     :
  }
  public void run() {
    // 1. レジスタの値をrdataに出力
    // 2. wctlの信号が1ならwdataの値を書き込む
    //    (if文を使ってよい)
    //    書き込む前の値を読み出せるように、読み出し、書き込みの
    //    順番で行う必要がある(同時に読み書きする場合への対策)
  }
}
```

レジスタファイル

- レジスタの集合
  - 機能：同時に2つのレジスタを読み、1つのレジスタに書き込める
    - 例：`add $t0, $t1, $t2`
- 入力
  - read1, read2, write1
    - 読み書きするレジスタ番号
  - wdata
    - 書き込むデータ
  - regWrite
    - 書き込み制御信号
- 出力
  - rdata1, rdata2
    - 読んだデータ

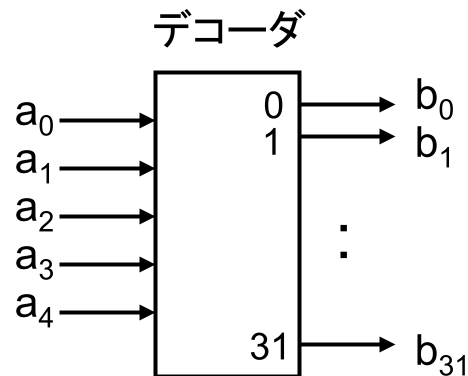


レジスタファイルの回路図



デコーダ

- 入力された数値に対応するビットを1にして出力
  - マルチプレクサの制御入力の部分に似ている
    - n 本の制御入力を使って 2^n 本の入力から1つ選ぶ
  - 例: a = 00011 (10進数で 3) が入力されたら b_3 = 1 になる（他の b_i は 0）

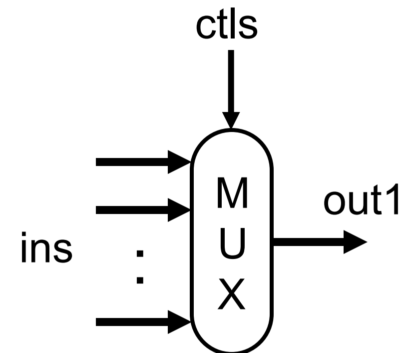


Decoder5 クラス

```
// 5ビット(a)から32ビット(b)へのデコーダ
// 回路を用いずに作成してよい(if文使用可)
public class Decoder5 {
  public Decoder5(Bus a, Bus b) {
        :
  }
  public void run() {
    // バスbの内、aの値によって選ばれる
    // 配線の値だけを1に設定する(後の配線は0)
  }
}
```

MUX32_bus クラス

```
// 32入力のマルチプレクサ(バス版)
// 回路を用いずに作成してよい(if文使用可)
public class MUX32_bus {
  public MUX32_bus(Bus ctls,  // 5ビット
                   Bus[] ins,  // 32本
                   Bus out1) {
    :
  }
  public void run() {
    // ctlsの値によって選択される入力バスの値を
    // 出力バスに設定する
  }
}
```



RegisterFile クラス

```
// デコーダ、レジスタ、32ビットMUXを接続する
public class RegisterFile {
  Register[] regs = new Register[32];
  Decode5 dec;
  MUX32_bus mux1, mux2;
  public RegisterFile(Path regWrite,                // 制御入力
                      Bus read1, Bus read2,         // 入力
                      Bus write1, Bus wdata,
                      Bus rdata1, Bus rdata2) {     // 出力
      :
  }
  public void run() {
      :
  }
}
```

RegisterFile クラスの使い方

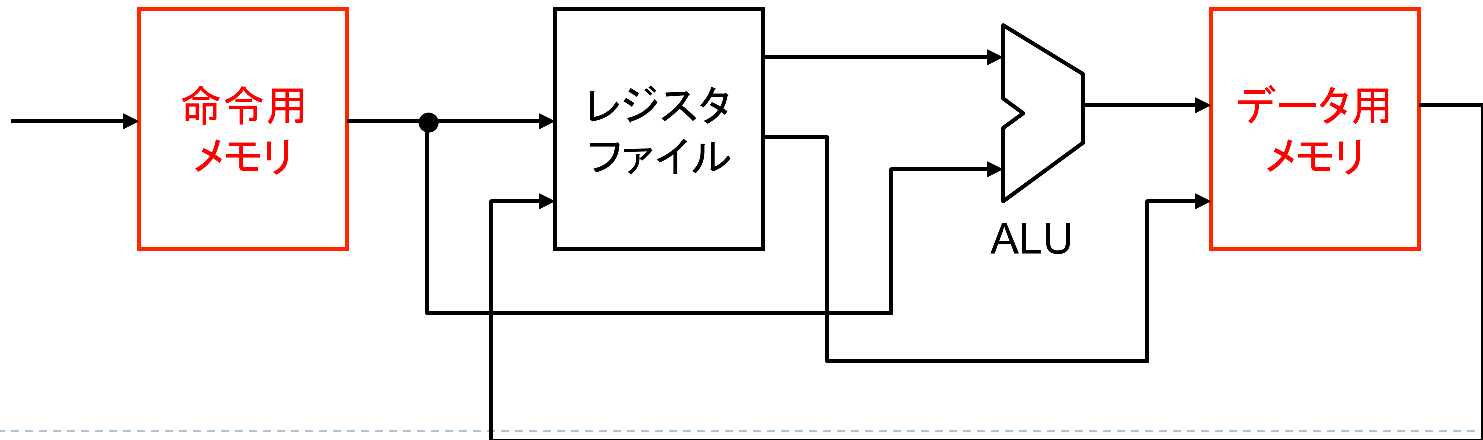
```
RegisterFile regFile =
  new RegisterFile(regWrite, read1, read2, write1, wdata,
                   rdata1, rdata2);

// 1番レジスタへの書き込みと読み出しを同時に行う
regWrite.setSignal(new Signal(true));
read1.setValue(1);  // 読み出すレジスタ番号
write1.setValue(1);  // 書き込むレジスタ番号
wdata.setValue(100);  // 書き込む値
regFile.run();
System.out.println(rdata1.getValue());  // 古い値(0)

// もう一度同じアクセス
regFile.run();
System.out.println(rdata1.getValue());  // 新しい値(100)
```

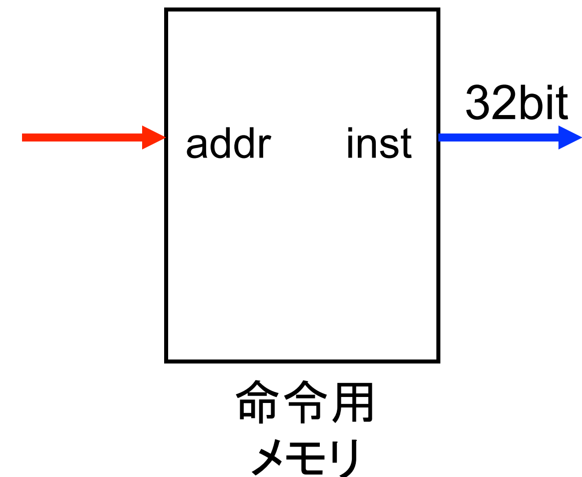
メモリ

- メモリを命令用とデータ用に分離し、個別に実装
  - 通常は区別はない
  - 制御を簡単にするため
- メモリは回路を用いずに実装する
  - 回路を用いると巨大になるため



命令用メモリ

- 命令用メモリは読み出し専用
  - 入力: 16進数アドレス(バイトアドレッシング)
    - 命令サイズ: 4byte (32bit)
    - 4byte (32bit) 毎に命令を格納
  - 出力: 命令
- アドレス 0x04000000 から始める
  - spim と同じ開始アドレス
- Spimの「.textセグメント」に相当

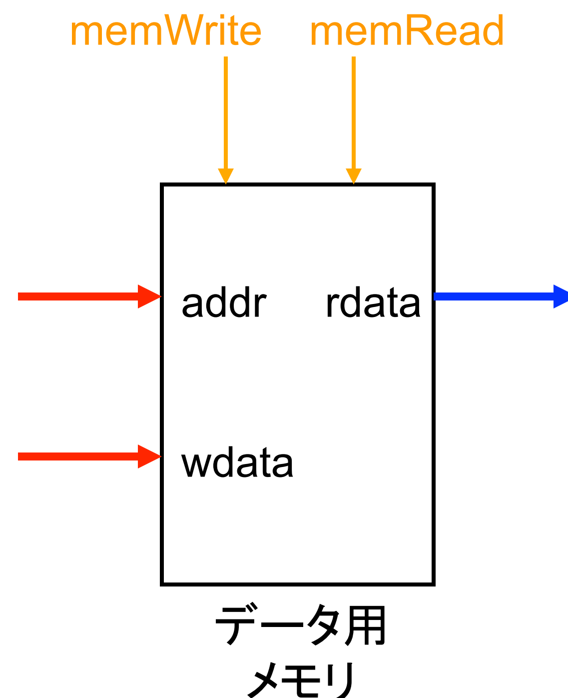


InstMemory クラス

```
public class InstMemory {
  int[] mem = new int[1024];  // 1024ワード
  //mem[i]: 1ワード(32bit)=>1命令分の情報を格納
  public InstMemory(Bus addr, Bus inst) {
    :
  }
  public void run() {
    // instバスにデータを送る
    // 命令アドレスは0x04000000から始まるので、
    // 0x04000000をmem[0]に対応させると効率がよい
    // memはワード単位なので4で割る
    long val = addr.getValue();
    int offset = (int)(val - 0x04000000) / 4;
    // mem[offset]の値をinstに設定
  }
  public void setInst(int addr, int inst) {
    // 命令をメモリに書き込む処理を書く
  }
}
```

データ用メモリ

- データ用メモリは読み書き可
  - 制御入力：memWrite、memRead
    - 読み書きを制御
  - 入力：アドレス、書き込むデータ
  - 出力：読み出したデータ
- アドレス 0x10000000 から始める
  - spim と同じ開始アドレス
- Spimの「.dataセグメント」、スタック等に相当



DataMemory クラス

```
public class DataMemory {
  int[] mem = new int[1024];  // 1024ワード
  public DataMemory(Path memRead, Path memWrite,  // 制御入力
                    Bus addr, Bus wdata,          // 入力
                    Bus rdata) {                  // 出力
    :
  }
  public void run() {
    // 本来ならmemReadとmemWriteが1かどうかはANDGateを
    // 使って判定するが、ここでは回路を用いないのでif文を使ってよい
    if (memRead.readSignal().getValue()) {
      // 読み出し処理(メモリの値をrdataに設定)
    }
    else if (memWrite.readSignal().getValue()) {
      // memRead, memWriteが同時に1になることはない。=> else if
      // 書き込み処理(wdataの値をメモリに設定)
    }
  }
}
```

課題

課題1（再掲）

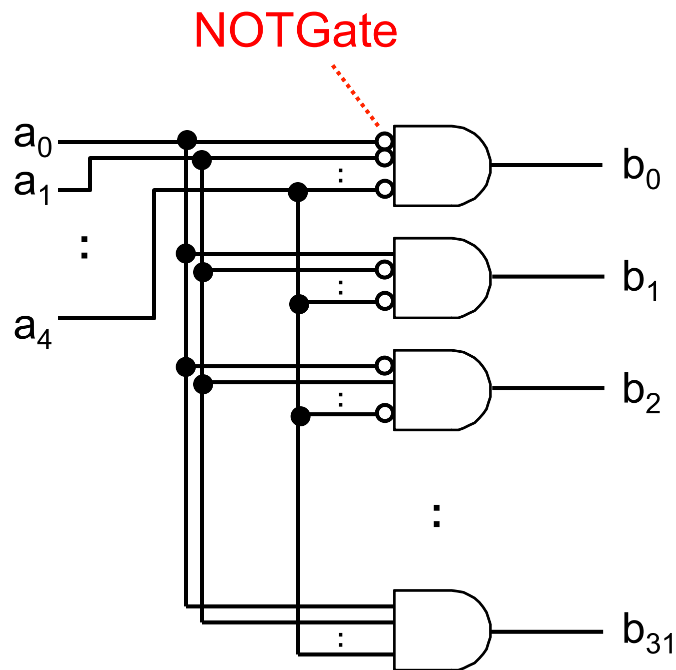
- レジスタファイルを完成させよ
  - Register、Decoder5、MUX32_bus、RegisterFile
  - RegisterFileDriver クラスを作り、テストを行うこと
    - 同じレジスタへの書き込みと読み出しを行うテストすること
      - run()を2回実行する必要がある

課題2（再掲）

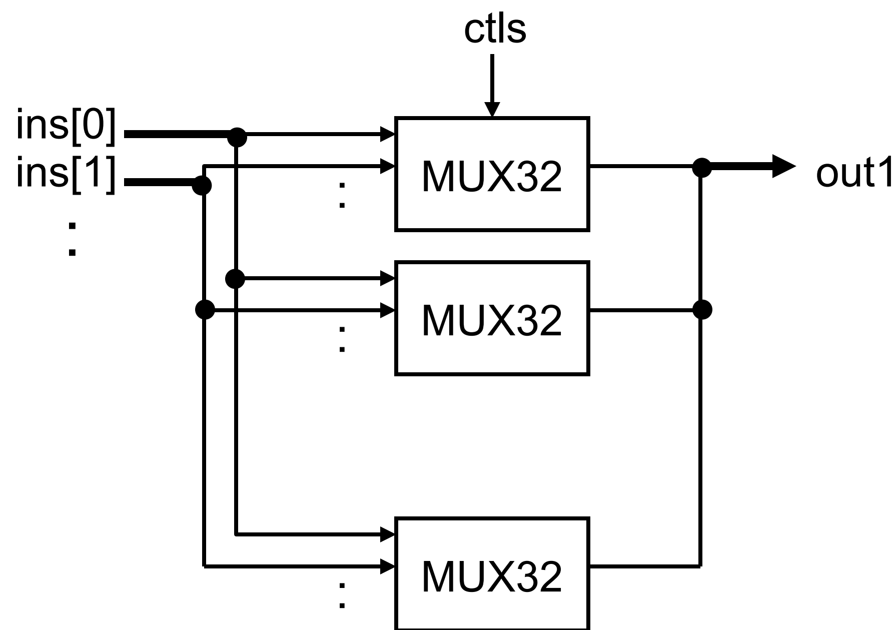
- メモリを完成させよ
  - InstMemory、DataMemory
  - InstMemoryDriver、DataMemoryDriver を作り、テストを行うこと
    - データを書き込んだ後に正しく読み出せることを確認すること

課題3（オプション）（再掲）

- Decoder5 クラスおよび MUX32_bus クラスを回路を使って実現せよ



Decoder5



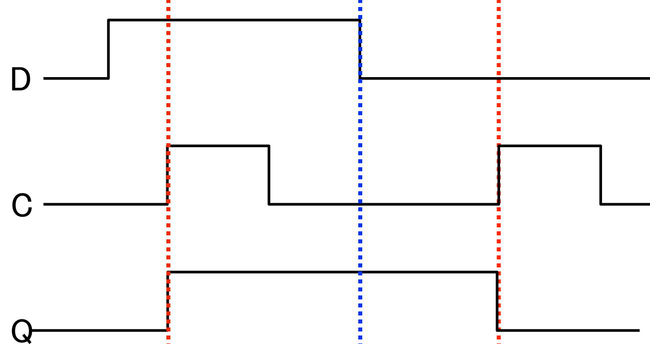
MUX32_bus

課題4（オプション）（再掲）

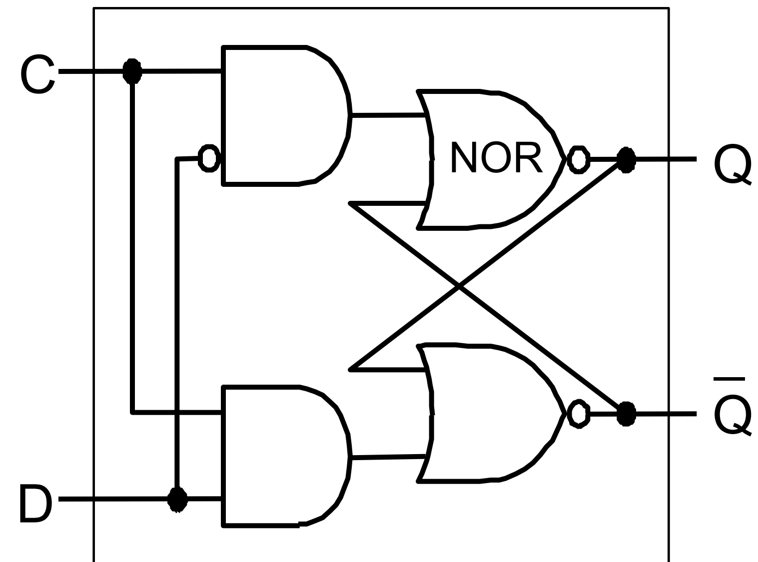
- レジスタをフリップフロップを使って作成し直してみよ
  - D-フリップフロップを配列として作成
    - 1ビットの情報を保持する
    - D-フリップフロップはD-ラッチの組み合わせ回路
  - 制御入力としてクロックを入れる

ヒント：D-ラッチ

- クロック（C）が 1 の間、入力 D の値が出力 Q に反映される
  - メモリへの書き込み
- クロックが 0 の間、D の値が変化しても Q は変化しない
  - メモリからの読み出し
  - 状態の保持



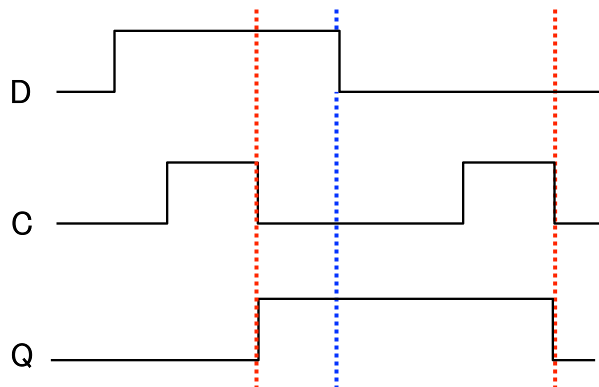
1クロックサイクル

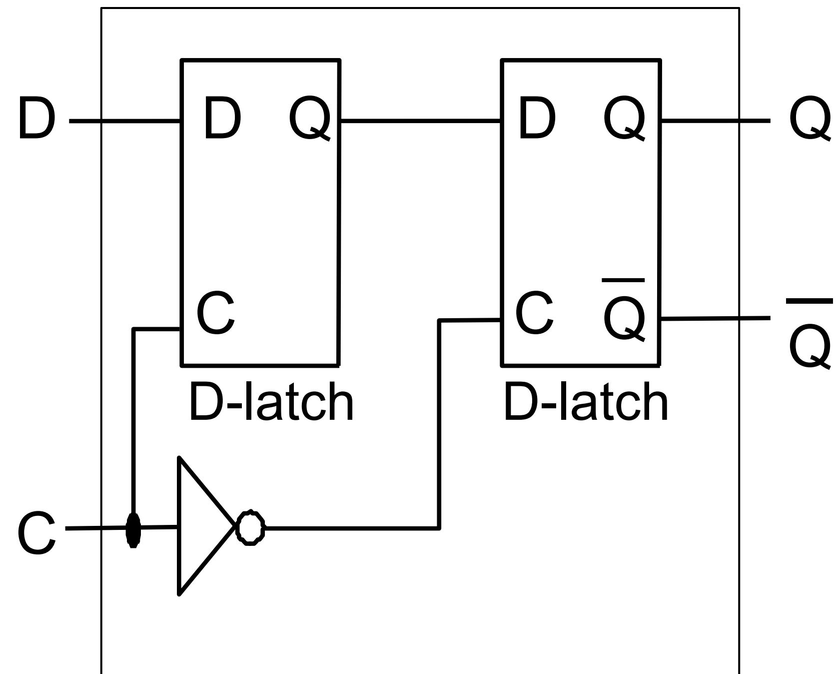


NOR：ORの出力を反転したゲート

ヒント：D-フリップフロップ

- 下降クロックエッジでのみ入力が出力に反映される
  - C=I になった時に1つ目のD-ラッチに反映
  - C=0 になった時に2つ目のD-ラッチに反映





ヒント: Latch, FlipFlop クラス

```
public class Latch {
  public Latch(Path clock, Path data, Path q, Path q2) {
      :
  }
  public void run() {
    // 出力を安定させるためにNOR回路を2回実行する必要がある
    // 実行結果のqとq2の値をNOR回路への入力とするため
  }
}

public class FlipFlop {
  public FlipFlop(Path clock, Path data, Path q, Path q2) {
    // 初期値を0にするには、2つのLatchの出力をq=0,q2=1に設定しておく
  }
  public void run() {
      :
  }
}
```

ヒント：Register クラスの使い方

```
Path regWrite = new Path();   // 書き込み制御フラグ
Path clock = new Path();   // クロック
Bus wdata = new Bus(32);   // 書き込むデータ
Bus rdata = new Bus(32);   // 読み出したデータ
Register reg = new Register(regWrite, clock, wdata, rdata);

// クロックが1の間に書き込みたい値を入力する
clock.setSignal(new Signal(true));
regWrite.setSignal(new Signal(true));
wdata.setValue(100);
reg.run();
System.out.println(rdata.readSignal());   // まだ読み出せない

// 下降クロックエッジ(1→0)で実際に書き込み
clock.setSignal(new Signal(false));
reg.run();
System.out.println(rdata.readSignal());   // 出力に反映される
```

課題提出

- 〆切: 7/13 (Fri) 23:59
- 提出物: 以下のファイルを圧縮したもの
  - ドキュメント(pdf,plain txt,wordなんでも可)
    - テスト結果　(テスト内容とその結果)
      - 注意: 作成したプログラムは今後も使用するため、十分にテストすること
    - 感想等
  - プログラムソース一式 (ソースコードのみ)
    - 必ず…Driver.javaクラスも含める
- 提出方法: Webから提出