2012年度
計算機システム演習
2012/06/11 第7回

計算機システムTA　福田圭祐（松岡研究室）

連絡事項

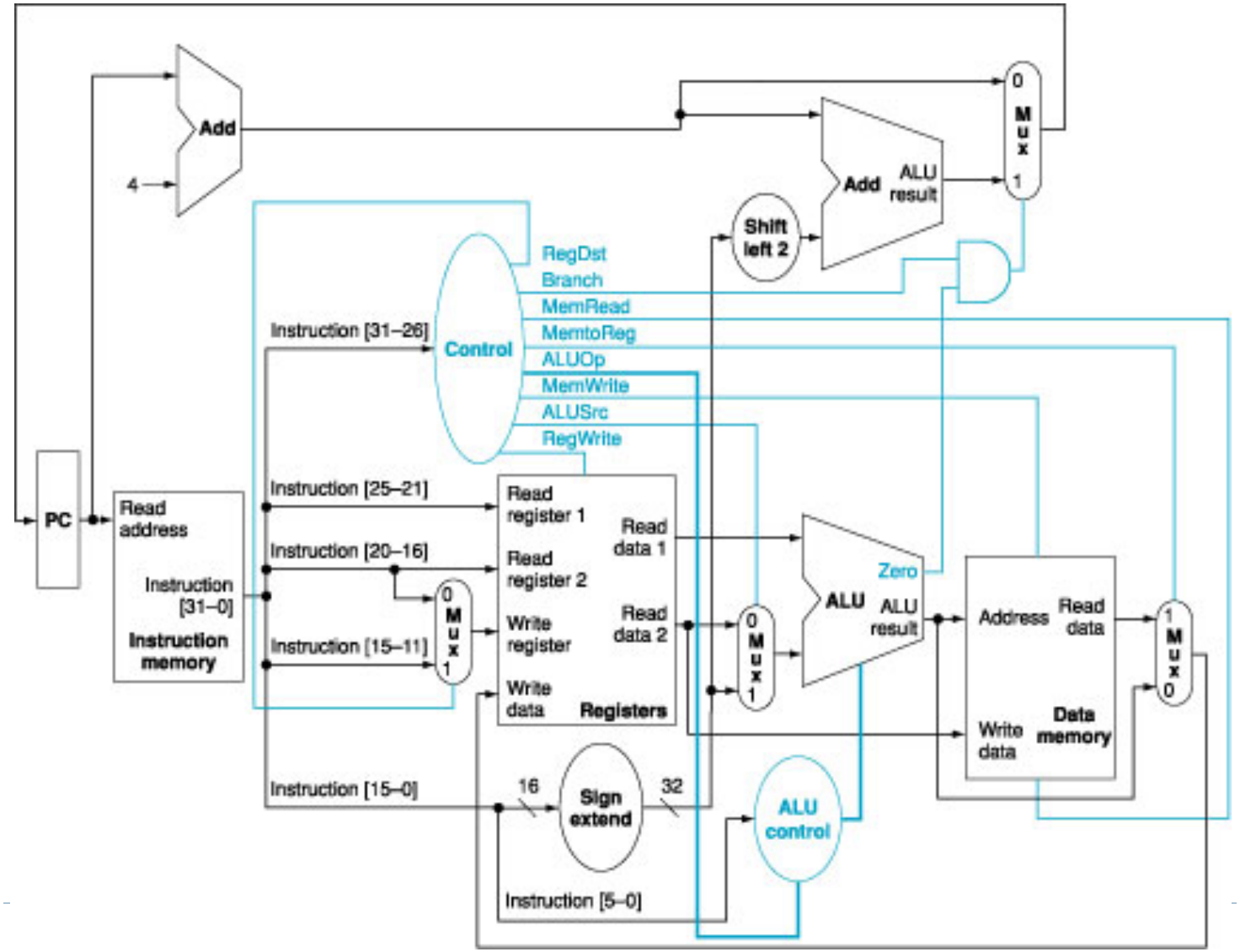
- いくつか日程変更があるのでWikiを参照

- 7月20日（金）に、組み立て演習を行います。
  - 可能な限り参加 （出席点+10～+20点）
  - 事情がある場合は個別対応します。事前に連絡ください

補足：

- SPIMにおける例外ハンドラの指定方法

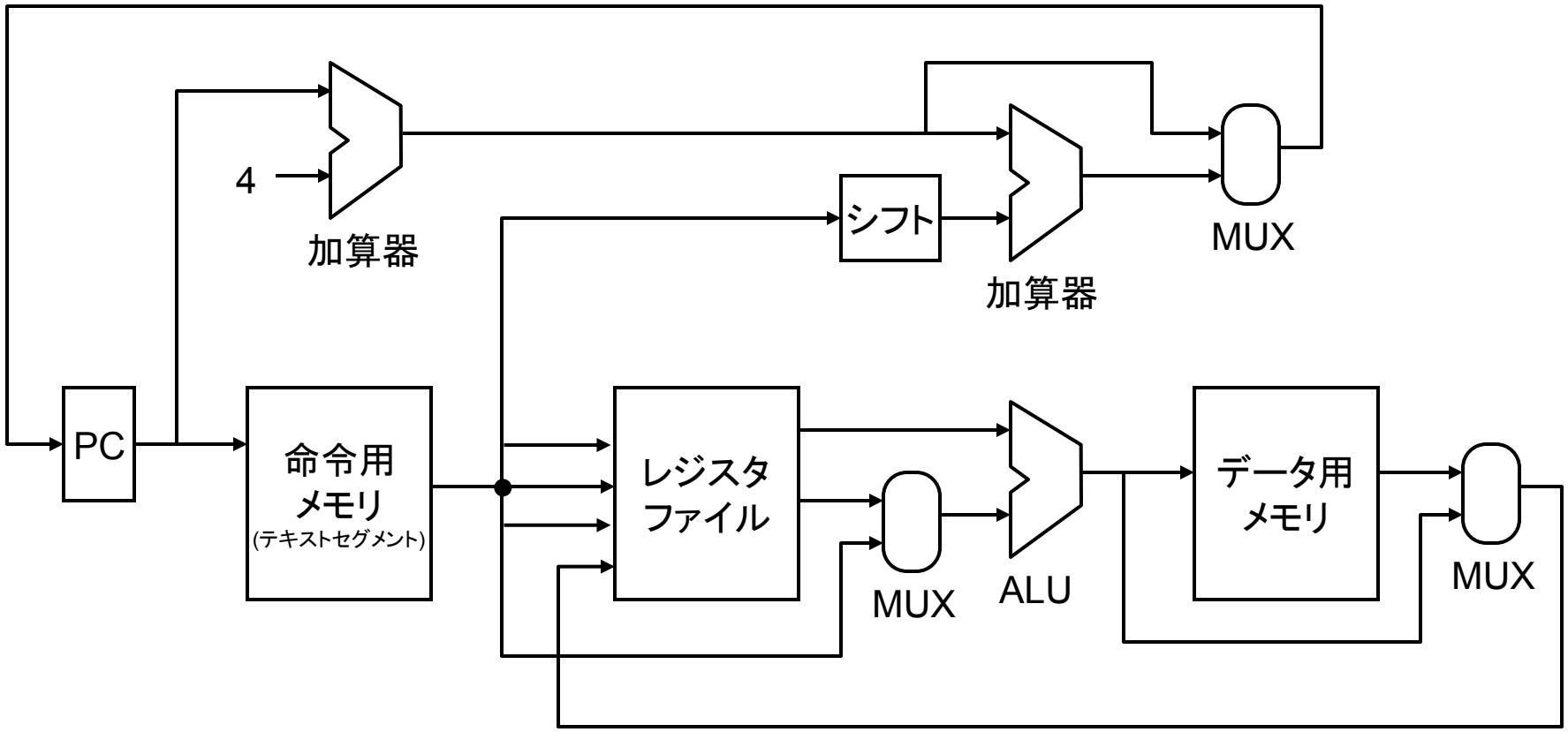
MIPSシミュレータ 完成図



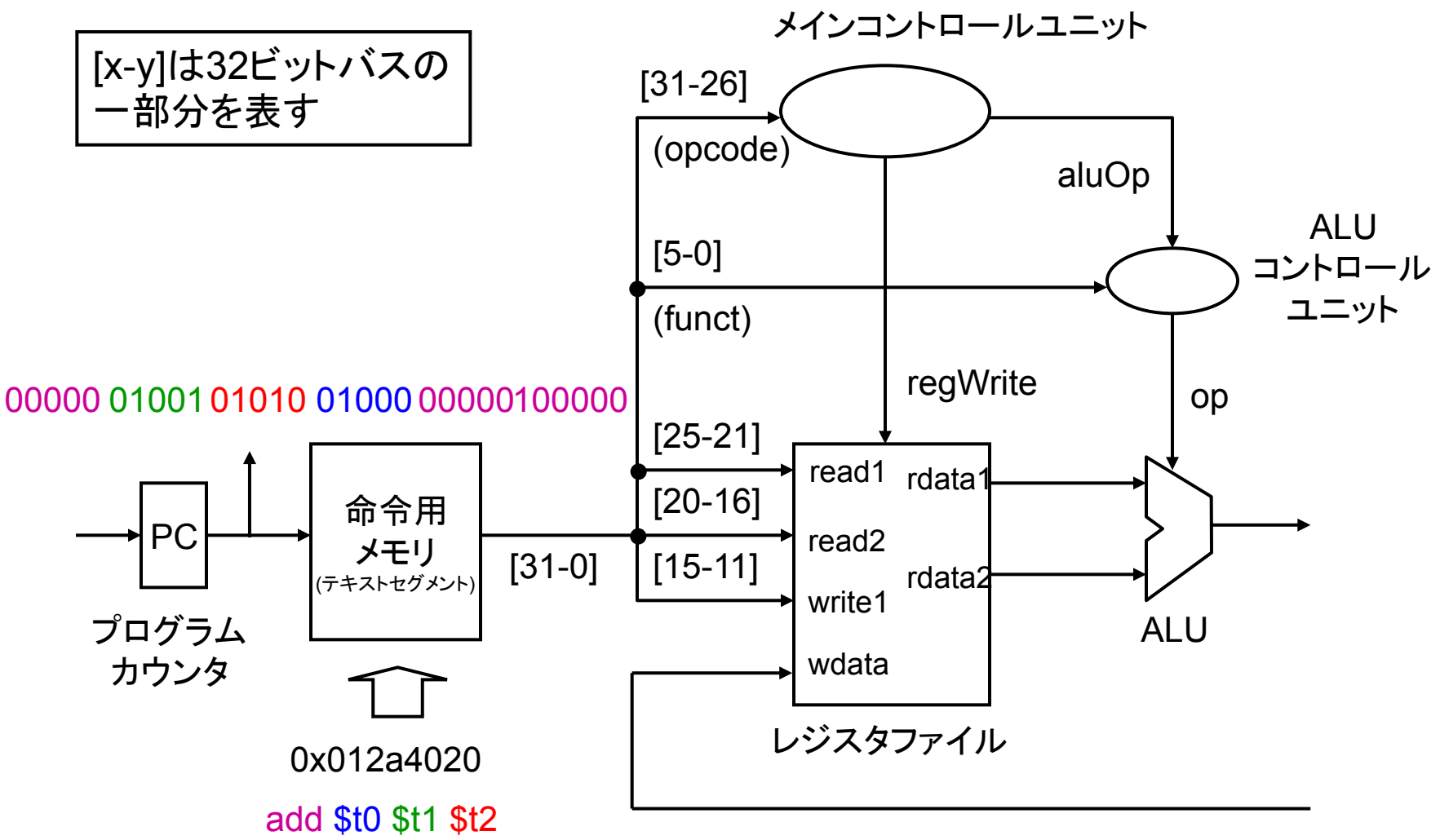
MIPSシミュレータ構築の流れ

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

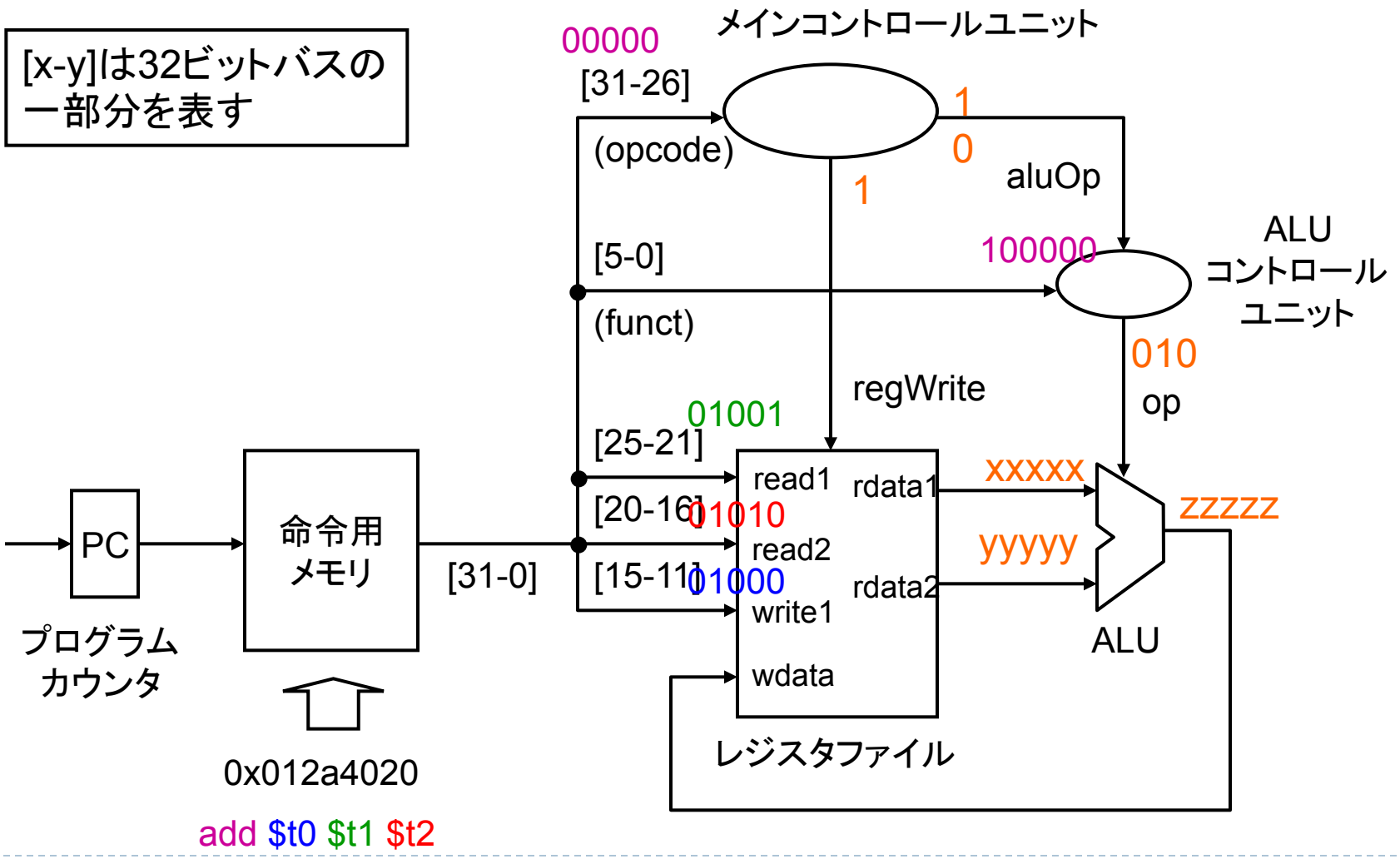
MIPSシミュレータ 概略図



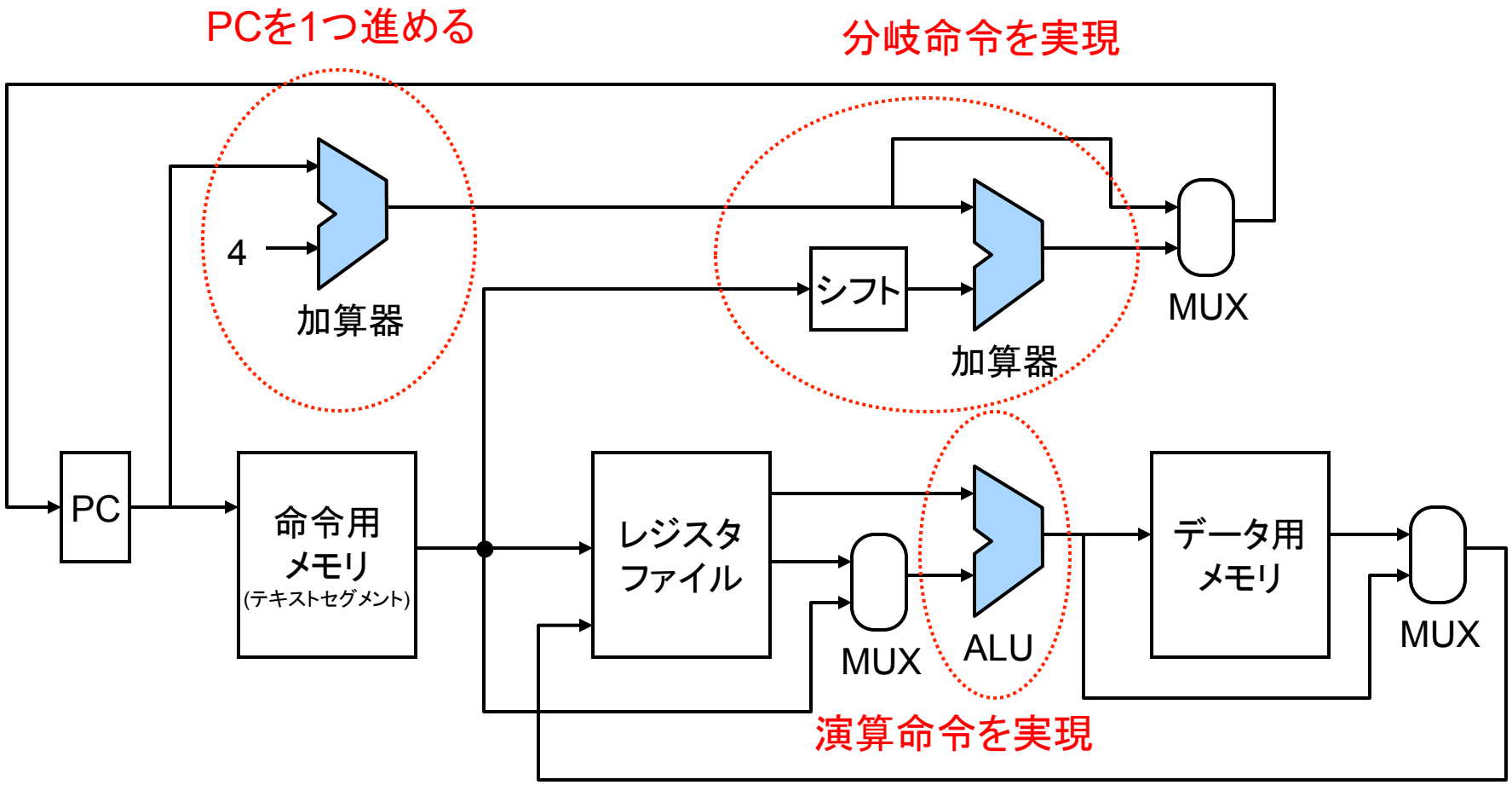
動作の流れ（一部）


[x-y]は32ビットバスの
一部分を表す
メインコントロールユニット
[31-26]
(opcode)
aluOp
ALU
コントロール
ユニット
[5-0]
(funct)
regWrite
op
00000 01001 01010 01000 00000100000
[25-21]
[20-16]
[15-11]
read1
read2
write1
wdata
rdata1
rdata2
PC
命令用
メモリ
(テキストセグメント)
[31-0]
プログラム
カウンタ
ALU
0x012a4020
レジスタファイル
add $t0 $t1 $t2

動作の流れ（一部）


[x-y]は32ビットバスの
一部分を表す
00000
メインコントロールユニット
[31-26]
(opcode)
1
0
aluOp
1
[5-0]
100000
ALU
コントロール
ユニット
(funct)
010
op
regWrite
01001
[25-21]
[20-16]
01010
[15-11]
01000
read1
rdata1
read2
rdata2
write1
wdata
xxxxx
yyyyy
zzzzz
ALU
PC
プログラム
カウンタ
命令用
メモリ
[31-0]
0x012a4020
レジスタファイル
add $t0 $t1 $t2

MIPSシミュレータ 概略図

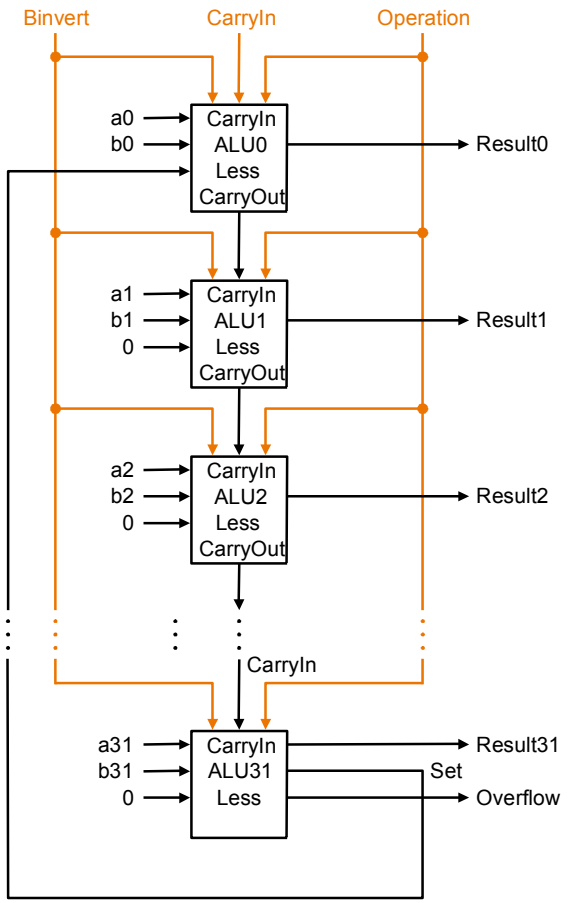


本日の内容 (Outline)

- MIPSシミュレータの概要
- ALUの作成
  - 1ビット加算器の作成
    - Signal, Pathを作成
    - 1ビット論理演算ユニットの作成
      - AND, OR, NOTゲート etc …
  - 32ビット加算器の作成
  - 1ビットALU
  - 32ビットALU

第1目標：ALU( )の作成

- ALU: Arithmetic Logic Unit
  - 演算論理装置
- 組み合わせ回路
  - AND, OR, NOT ゲートを使って作成
- 作成の順番
  1. 1ビット加算器 ⇦ 本日の内容
  2. 32ビット加算器
  3. 1ビットALU
  4. 32ビットALU



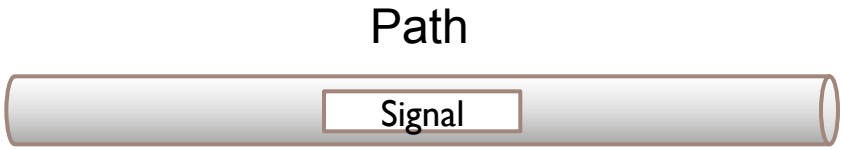
Signal クラス

- 回路内の信号を表現
  - 信号は 0, 1 の値を持つ
    - 0 は false で表現
    - 1 は true で表現
- 信号の値の表示
  - System.out.println(sig)
    - sig は Signal オブジェクト

```
public class Signal {
  boolean val;
  public Signal(boolean val) {
    this.val = val;
  }
  // 信号の値の読み出し
  public boolean getValue() {
    return val;
  }
  public String toString() {
    if (val)
      return "1";
    else
      return "0";
  }
}
```

Path クラス

- 配線を表現
  - 配線上に信号が流れている
    - Signal オブジェクトを持つ



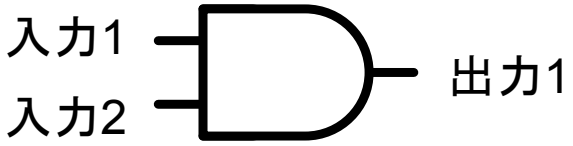

```
public class Path {
  Signal sig = new Signal(false); // 信号の初期値は0

  // 流れる信号を変更
  public void setSignal(Signal sig) {
    this.sig = sig;
  }
  // 流れている信号を読み出す
  public Signal readSignal() {
    return sig;
  }
}
```

ゲート

- 基本論理演算を実装した電子素子
- 3種類
  - AND, OR, NOT
- 入力から出力が決まる
  - 入力配線から信号を読む
  - 出力する信号を論理演算で決定する
  - 出力配線に新しい信号を設定する
- 配線(pathクラス)で他のゲートと接続される

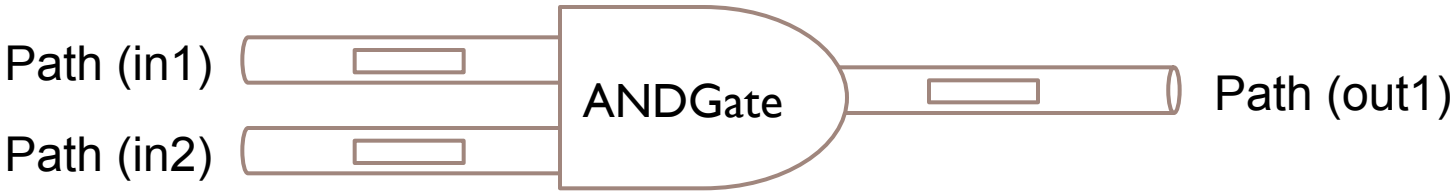
ANDGate クラス

- AND ゲートを表現
  - 論理積:A・B
  - boolean の論理積は && 演算子で実現
  - コンストラクタ（new した時点）
    - 引数は入出力配線
    - 入出力配線を接続
  - run メソッド
    - 入力信号から出力信号を決定



入力1	入力2	出力1
0	0	0
0	1	0
1	0	0
1	1	1

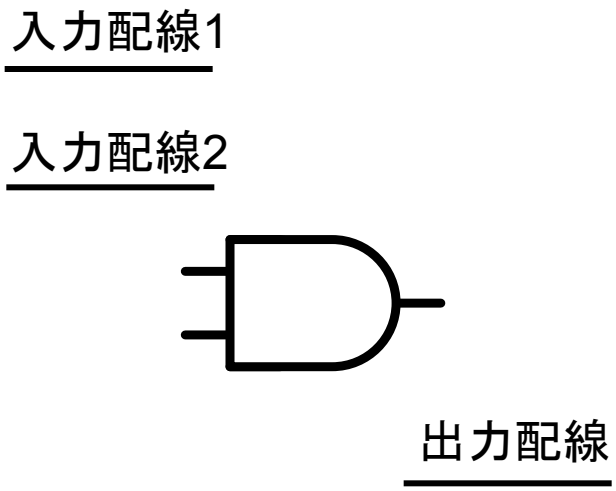
ANDGate クラスのコード例




```
public class ANDGate {
  Path in1, in2, out1;
  public ANDGate(Path in1, Path in2, Path out1) {
    this.in1 = in1;  // 入力1を接続
    this.in2 = in2;  // 入力2を接続
    this.out1 = out1;  // 出力1を接続
  }
  public void run() {
    Signal sig1 = in1.readSignal();  // 入力1の信号
    Signal sig2 = in2.readSignal();  // 入力2の信号
    // 論理積
    boolean val = sig1.getValue() && sig2.getValue();
    out1.setSignal(new Signal(val));  // 出力1に信号を設定
  }
}
```

ANDGateクラス の使用方法

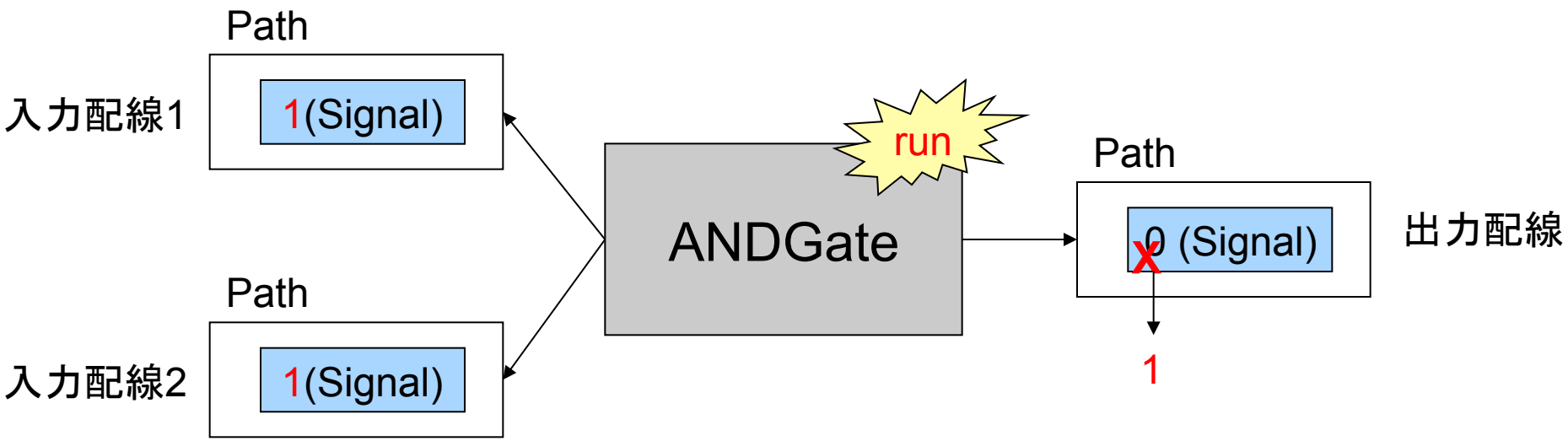
- 3つの配線を作成
  - new Path()
- ANDGate を作成
  - コンストラクタに配線を渡す
  - new ANDGate(in1, in2, out1)
- 入力配線に信号を設定
- ANDGate の run メソッドを実行
- 出力配線から信号を読み出し



ANDDriver クラスのコード例
(ANDGateクラスのテストプログラム)

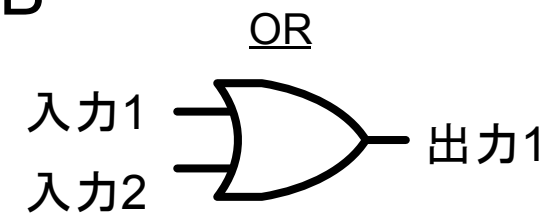
```
public class ANDDriver {
  public static void main(String[] args) {
    Path in1 = new Path();  // 入力配線1
    Path in2 = new Path();  // 入力配線2
    Path out1 = new Path();  // 出力配線
    ANDGate and1 = new ANDGate(in1, in2, out1);

    in1.setSignal(new Signal(true));  // 信号(1)を設定
    in2.setSignal(new Signal(true));  // 信号(1)を設定

    and1.run();  // 1 and 1

    System.out.println(out1.readSignal());//1が出力
  }
}
```

ANDGateクラスの動作のイメージ



ORGate, NOTGate

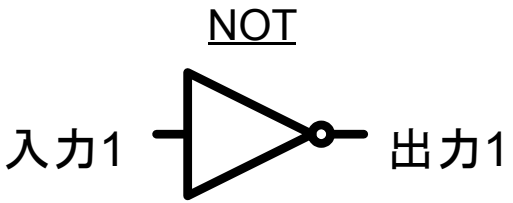
A+B



論理和
（||）

入力1	入力2	出力1
0	0	0
0	1	1
1	0	1
1	1	1

$\overline{A}$



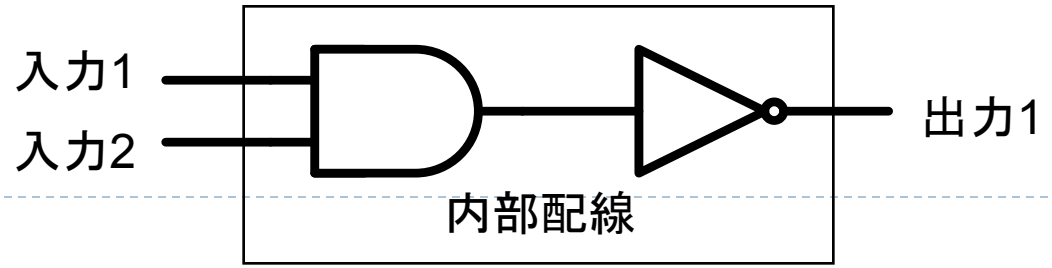
否定
（!）

入力1	出力1
0	1
1	0

NANDCircuit クラス

- 組み合わせ回路
  - ANDGate と NOTGate の組み合わせ
    - 内部状態を持たない
    - 出力は入力のみに依存
  - コンストラクタ
    - 入出力配線を接続
    - 回路を構成
      - 内部配線でゲート同士を接続
  - run メソッド
    - 各ゲートの run メソッドを実行

入力1	入力2	出力1
0	0	1
0	1	1
1	0	1
1	1	0



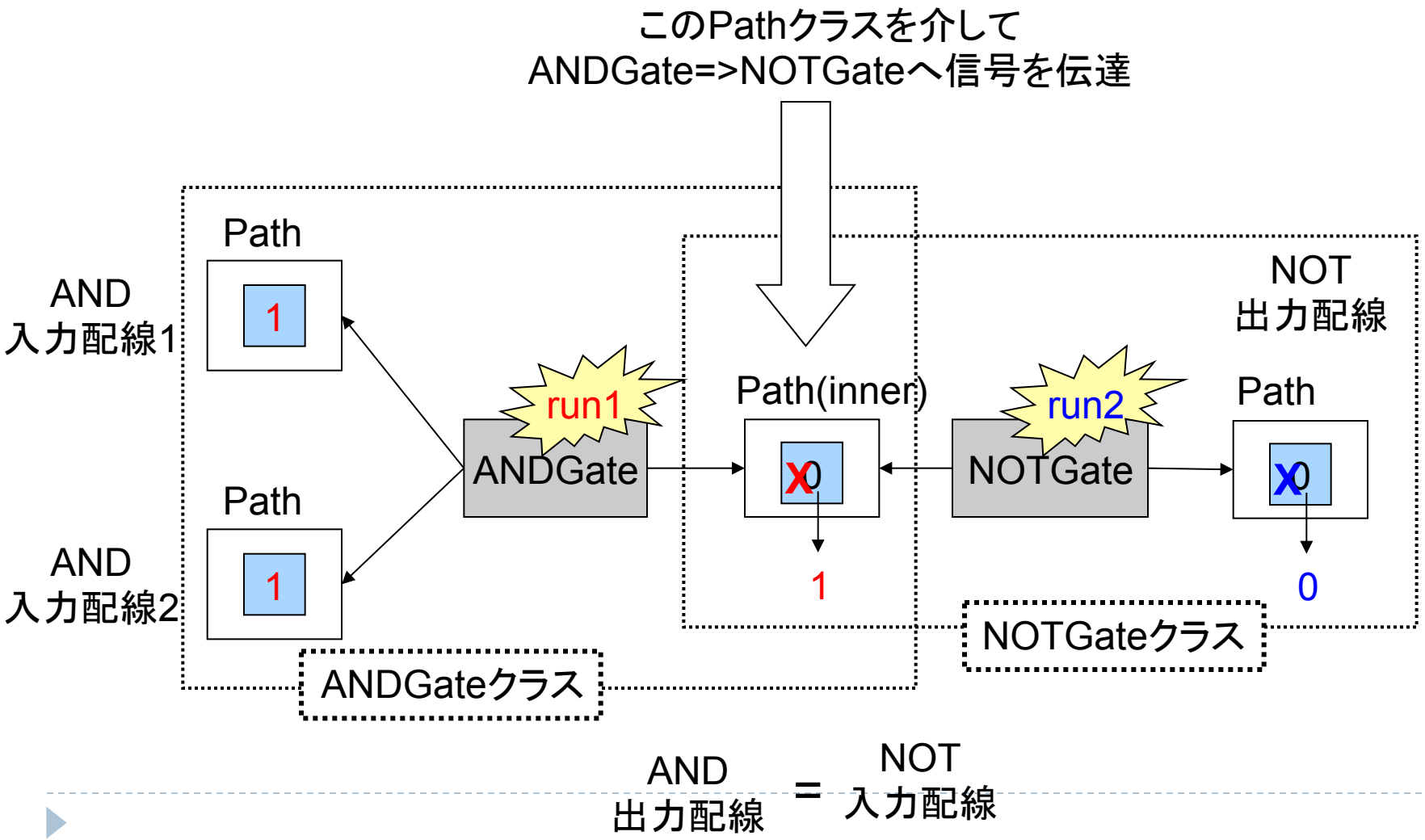
NANDCircuit クラスのコード例

```
class NANDCircuit {
  ANDGate and1;
  NOTGate not1;
  public NANDCircuit(Path in1, Path in2, Path out1) {
    Path inner = new Path();  // 内部配線
    // 内部配線をつかってANDGateの出力とNOTGateの入力を接続
    and1 = new ANDGate(in1, in2, inner);
    not1 = new NOTGate(inner, out1);
  }
  public void run() {
    // 各ゲートを実行
    and1.run();
    not1.run();
  }
}
```

重要

回路の上流からrunを実行

NANDCircuitクラスの動作イメージ



NANDDriver クラス
(NANDCircuitクラスのテストプログラム)

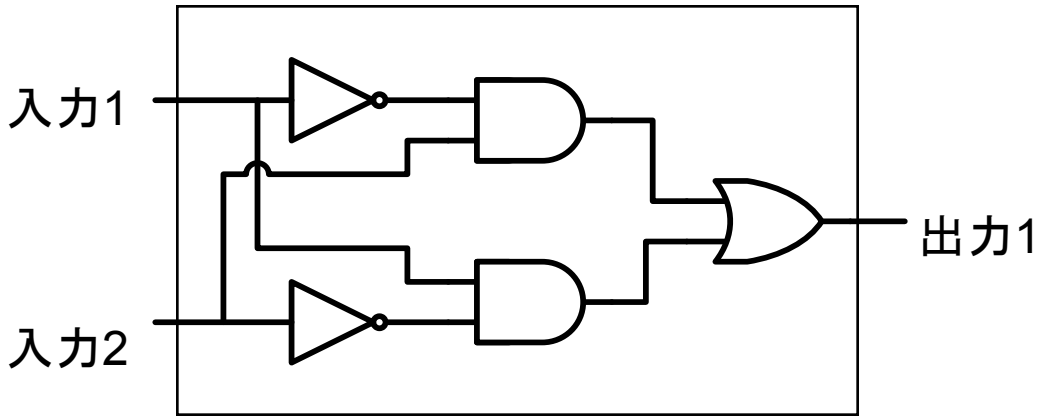
- 組み合わせ回路もゲートと同じように使える
  - コンストラクタの引数は入出力配線
  - run メソッドで入力信号から出力信号を決定

```
public class NANDDriver {
  public static void main(String[] args) {
      :
    NANDCircuit nand = new NANDCircuit(in1, in2, out1);
      :
    nand.run();
      :
  }
}
```

XORCircuit クラス

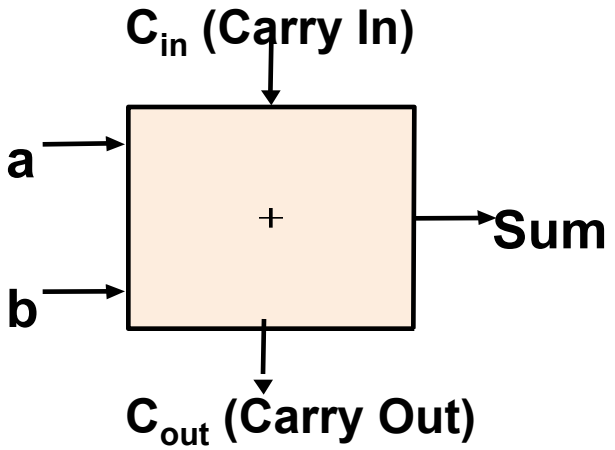
- XOR
  - 排他的論理和：exclusive or
  - Not Equal
    - 2つの入力値が異なる場合に真
  - $\overline{A}\cdot B + A\cdot\overline{B}$ と等価

入力1	入力2	出力1
0	0	0
0	1	1
1	0	1
1	1	0

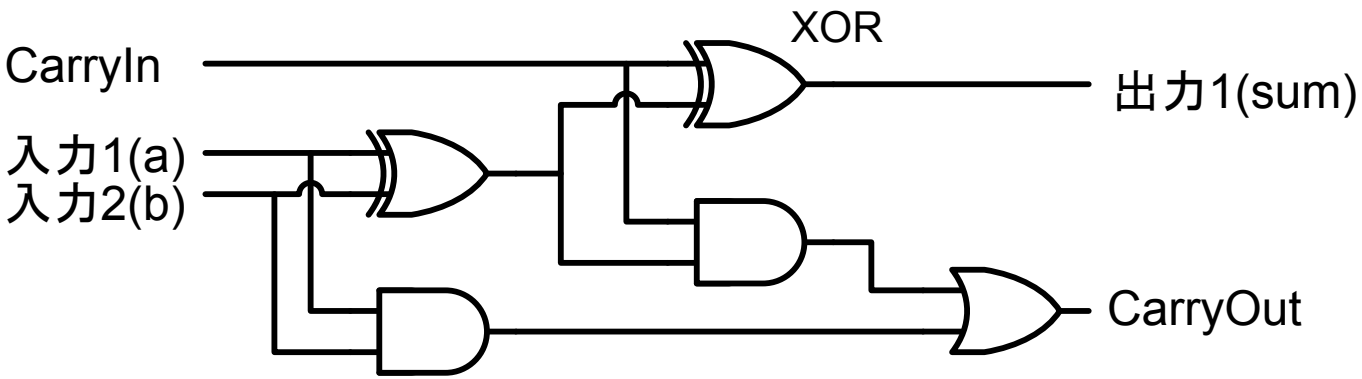


1ビット全加算器（FA）クラス（1/2）

- 全加算器（Full Adder）
  - 桁上がり（Carry In）を入力に受け付ける
    - c.f) 半加算器（Half Adder）：桁上がりを入力に取らない
- 講義スライドの論理式
  - sum = A xor B xor C_{in}
  - C_{out} = A･B + A･C_{in} + B･C_{in}
- 実装に使用する論理式
  - sum = A xor B xor C_{in}
  - C_{out} = A･B + （A xor B）･C_{in}



1ビット全加算器（FA）クラス（2/2）



入力1	入力2	Carry In	出力1	Carry Out
0	0	0	0	0
0	1	0	1	0
1	0	0	1	0
1	1	0	0	1

入力1	入力2	Carry In	出力1	Carry Out
0	0	1	1	0
0	1	1	0	1
1	0	1	0	1
1	1	1	1	1

課題

遅れても採点します

課題1

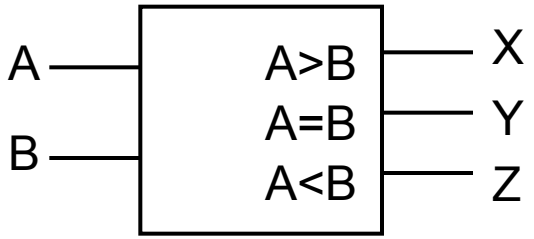
- ORGate, NOTGate, XORCircuit クラスを作成せよ
  - ORGateとNOTGateは、Java言語の演算子等を使ってよい
  - XORCircitは、既存の各種Gateを組み合わせて作ること
  - ORDriver, NOTDriver, XORDriver クラスを作成してテストすること
    - すべての入力についてそれぞれが正しい結果を返すことを確認すること
      - n入力の論理回路に対して、出力は 2^n パターンある
  - レポートには各 Driver クラスおよびその実行結果も含めること
    - クラスファイル（ORGate.classなど）は送らない

課題2

- 1ビット全加算器を作成せよ（FA クラス）
  - FADriver クラスを作成し、すべての入力に対して正しく動作することを確認すること
  - レポートには FADriver クラスおよびその実行結果も含めること
    - 8入力パターンすべて試すこと
    - クラスファイルは送らない

課題3 （オプション）

- 1ビットの2つの入力の大小関係を出力する”回路を設計”し、CompCircuit クラスを作ってみよ
  - AND, OR, NOTゲートを組み合わせて実装
  - X,Y, Z のいずれかが 1 になる
  - ヒント:
    - A>B になるのは A が 1、B が 0 の時のみ
    - A=B の判定は XOR に似ている



A	B	X	Y	Z
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

課題提出

- ∝切：6/22 (金) 23:59
- 提出物：以下のファイルを圧縮したもの
  - プログラムソース一式 (ソースコードのみ、.classファイルは含めない)
    - 必ず…Driver.javaクラスも含める
    - 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
  - (必要であれば) ドキュメント
    - 感想等
- 提出方法：Webから提出
  - 授業のページからリンク
  - パスワードを忘れてしまった方は福田まで