2012年度
計算機システム演習 第11回

計算機システムTA　福田圭祐

日程確認

- PC組み立て演習
  - 日時：7/20（金）15:05~　(2時間程度)
  - 場所：西7演習室
    - （福田は出張中につきTAは別の人です）
- 試験
  - 日時：8/6（月）3,4限
  - 場所：W833（授業と異なるので注意）

補足事項 (1)

- 狙い通りのビット列(を表すlong整数)を作る方法

10000….000 となるような整数値は何か？

32ビット

単純に(符号付きintで)表すと、
Javaの整数演算と干渉する恐れがある
特に、整数 ⇔ ビット列 の変換時に支障

Javaには符号無し整数型が存在しない

0x 80000000 L

- 16進数一桁が、2進数四桁にぴったり対応する
  - $0_{16} = 0000_2$　$8_{16} = 1000_2$　$F_{16} = 1111_2$ など

補足事項 (2)

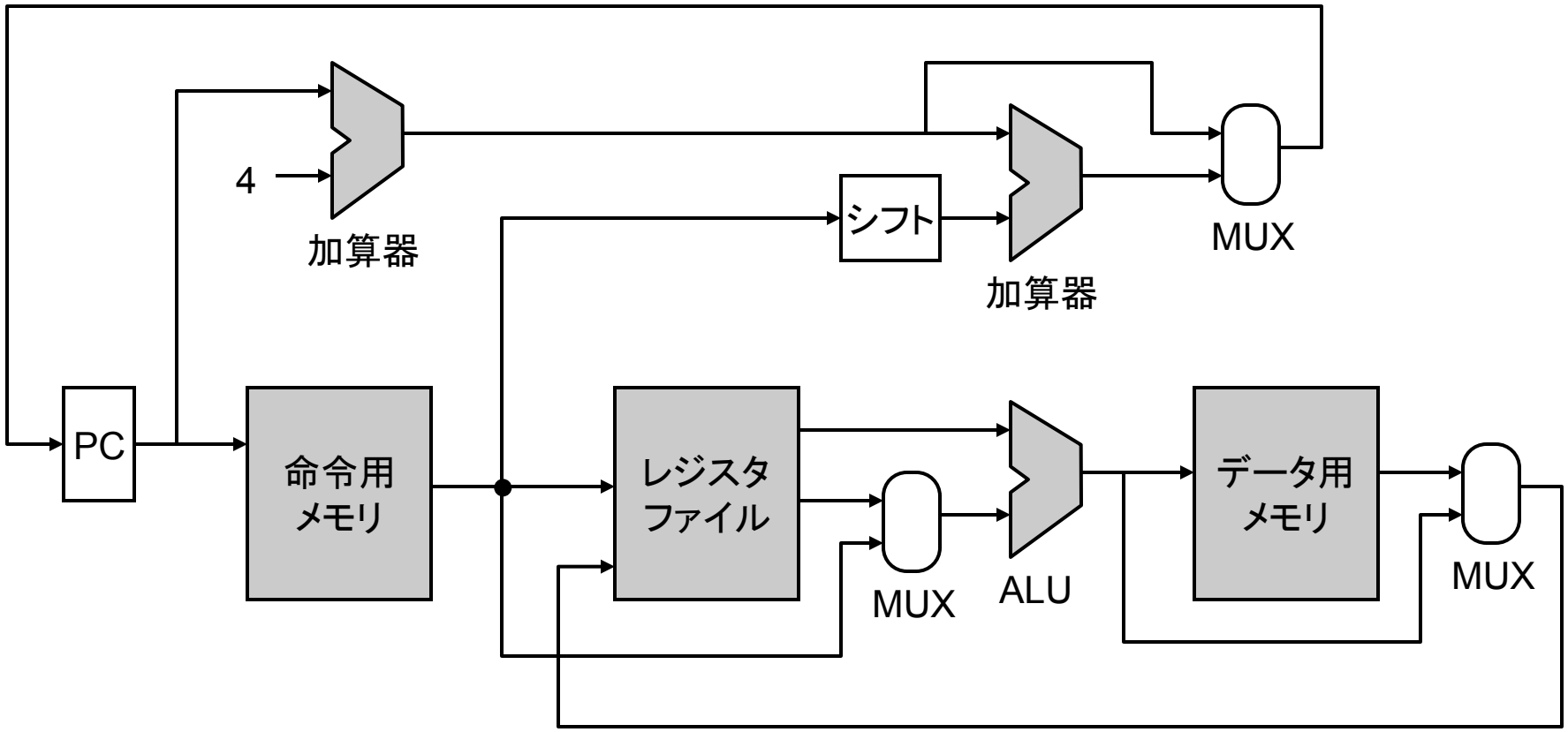
- 同様に・・・
- $11111\ldots111_2$ は？
- $11111\ldots110_2$ は？
- $1010\ldots1010_2$ は？

- $11111\ldots111_2$ は、具体的にどんな数を表すだろうか？
- $10000\ldots000_2$ は ？

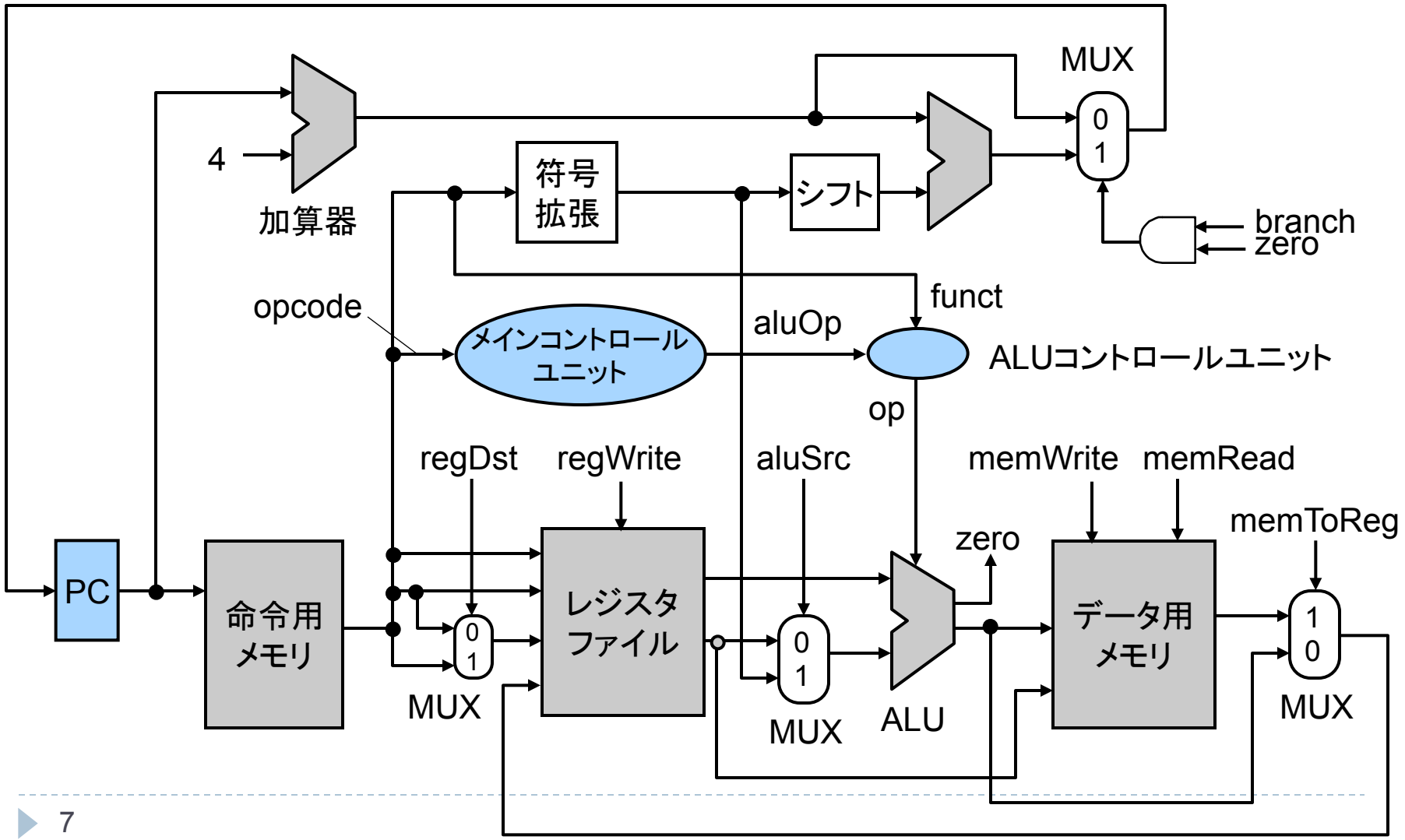
MIPSシミュレータ構築の流れ

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

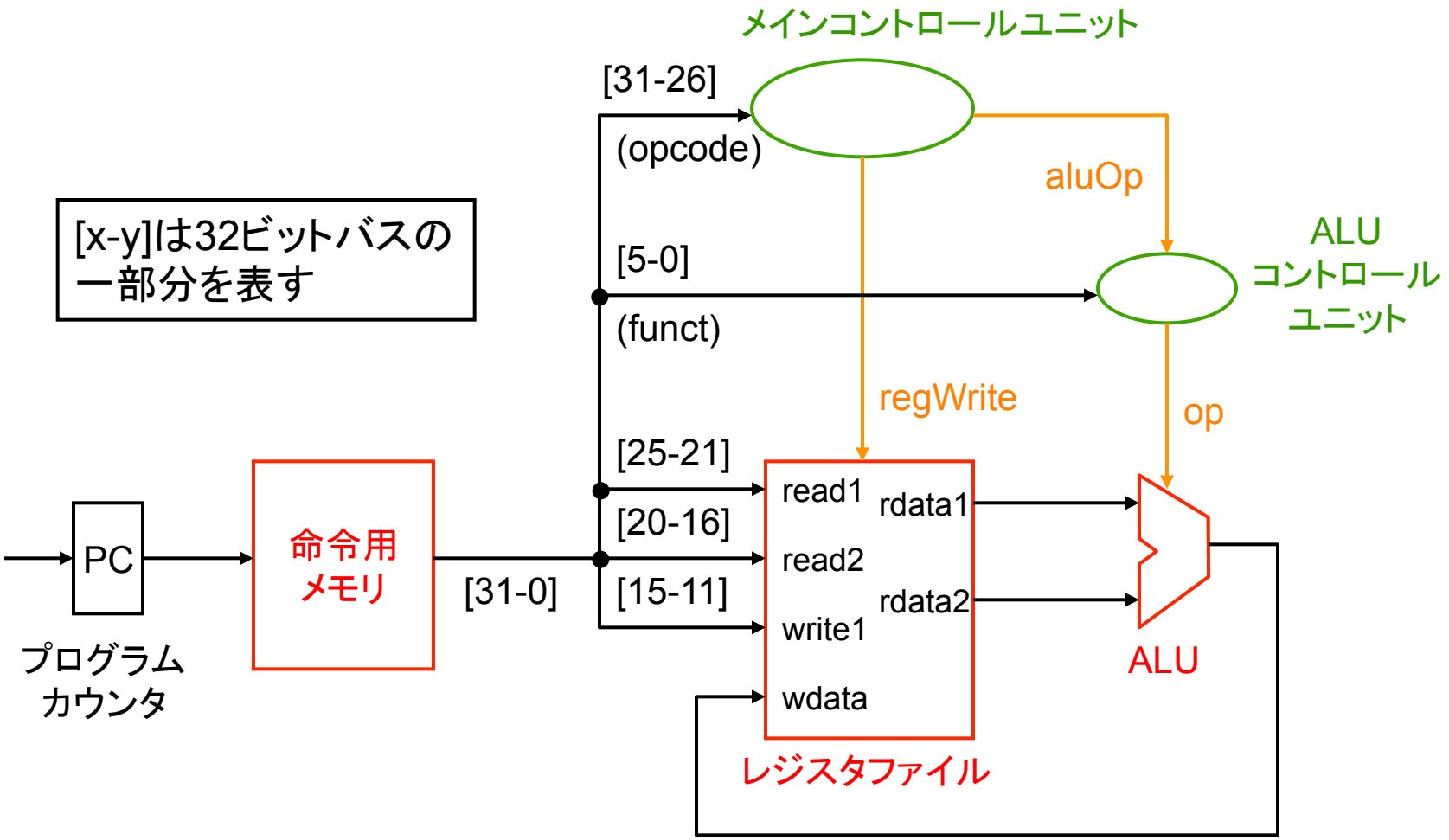
MIPSシミュレータの概要



MIPSシミュレータの完成図

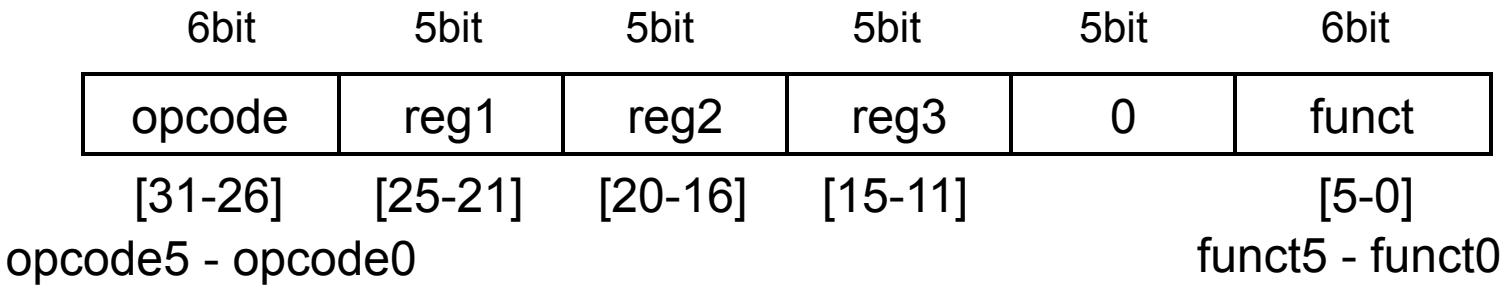


本日は「算術論理演算回路」を作る

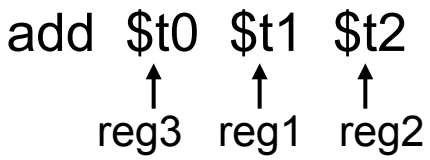


算術論理演算命令のフォーマット

- 32ビットの命令の中身は以下のようになっている



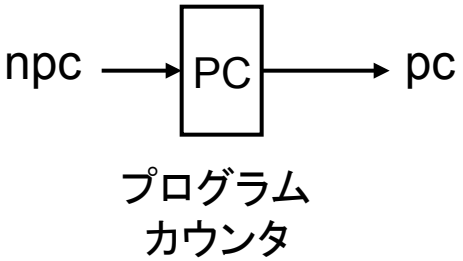
- funct が演算の内容を決める
- opcodeで制御信号の値を決める
- 入力レジスタ番号：reg1, reg2
- 出力レジスタ番号：reg3



funct	命令
100000	add
100010	sub
100100	and
100101	or
101010	slt

プログラムカウンタ

- 実行しているアドレスを保持するレジスタ
  - 0x04000000 から実行を開始
  - 入力：次の命令のアドレス
  - 出力：実行する命令のアドレス
  - Register クラスの wctl を常に 1 にしておけばよい
    - 実行毎（クロック毎）に実行アドレスを更新するため




```
public class PC {
  Register reg;
  public PC(Bus npc, Bus pc) {
    Path wctl1 = new Path();
    wctl1.setSignal(new Signal(true));
    reg = new Register(wctl1, npc, pc);
  }
}
  //初期アドレスをセット
  public void setValue(int addr);
```

Bus クラス（メソッド追加）

```
public class Bus {
    :
  // index番目からnum本の配線を取り出したバスのサブセットを返す
  public Bus getSubset(int index, int num) {
    Bus subBus = new Bus(num);
    for (int i = 0; i < num; i++)
      subBus.paths[i] = this.paths[index + i];
    return subBus;
  }
    :
}
```

例

```
Bus funct = inst.getSubset(0, 6);   // 0～5番の6本
Bus opcode = inst.getSubset(26, 6);  // 26～31番の6本
```

※ 32bit CLA (オプション課題)を作成した人は既に実装済み

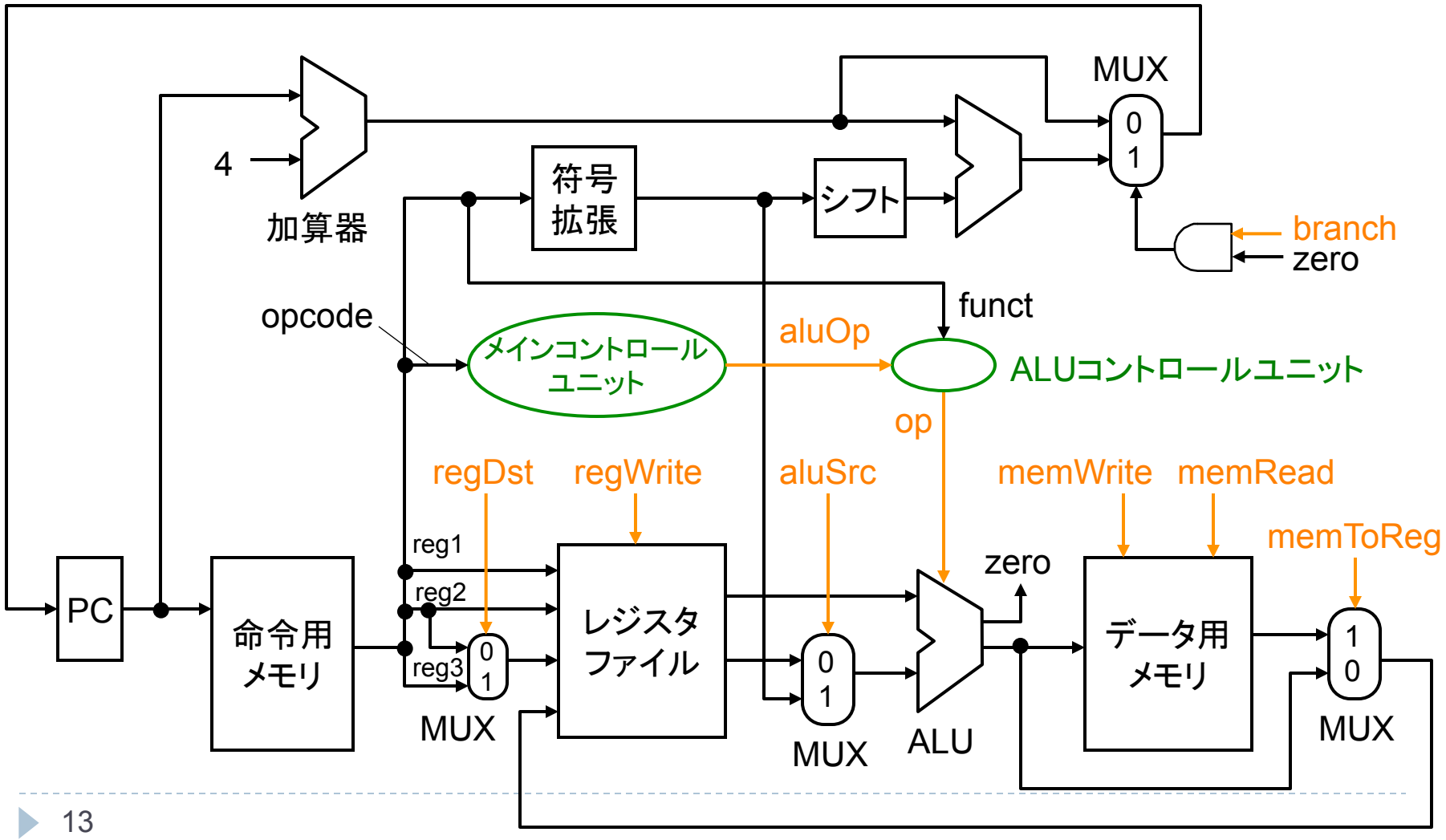
メインコントロールユニット

- 命令の内容（opcode）から制御信号の値を決める
  - 各回路に制御入力を接続する

制御パス	意味
aluOp（2bit）	命令の種類（演算、メモリアクセス、分岐）
regWrite	レジスタへの書き込みを制御
memRead	データ用メモリの読み出しを制御
memWrite	データ用メモリへの書き込みを制御
regDst	命令中の書き込みレジスタ番号の位置を制御
memToReg	計算結果とメモリ値のどちらをレジスタに書くかを制御
aluSrc	レジスタ値と命令中の値のどちらをALUに入力するかを制御
branch	分岐を制御

完成版のためにこれらも実装

回路全体を制御


MUX
0
1
4
加算器
符号
拡張
シフト
branch
zero
opcode
funct
メインコントロール
ユニット
aluOp
ALUコントロールユニット
op
regDst
regWrite
aluSrc
memWrite
memRead
memToReg
reg1
reg2
reg3
zero
PC
命令用
メモリ
レジスタ
ファイル
データ用
メモリ
1
0
MUX
MUX
ALU
MUX

制御信号の値

- 命令の種類によって以下のように決定される
  - X：使用されない (任意のXで動作は同じ)

命令	regDst	aluSrc	memTo Reg	reg Write	mem Read	mem Write	branch	aluOp1	aluOp0
演算	1	0	0	1	0	0	0	1	0
lw	0	1	1	1	1	0	0	0	0
sw	X	1	X	0	0	1	0	0	0
beq	X	0	X	0	0	0	1	0	1

これらも実装　　これらも実装

R 形式

opcode	reg1	reg2	reg3	0	funct
[31-26]	[25-21]	[20-16]	[15-11]		[5-0]

I 形式

opcode	reg1	reg2	offset
[31-26]	[25-21]	[20-16]	[15-0]

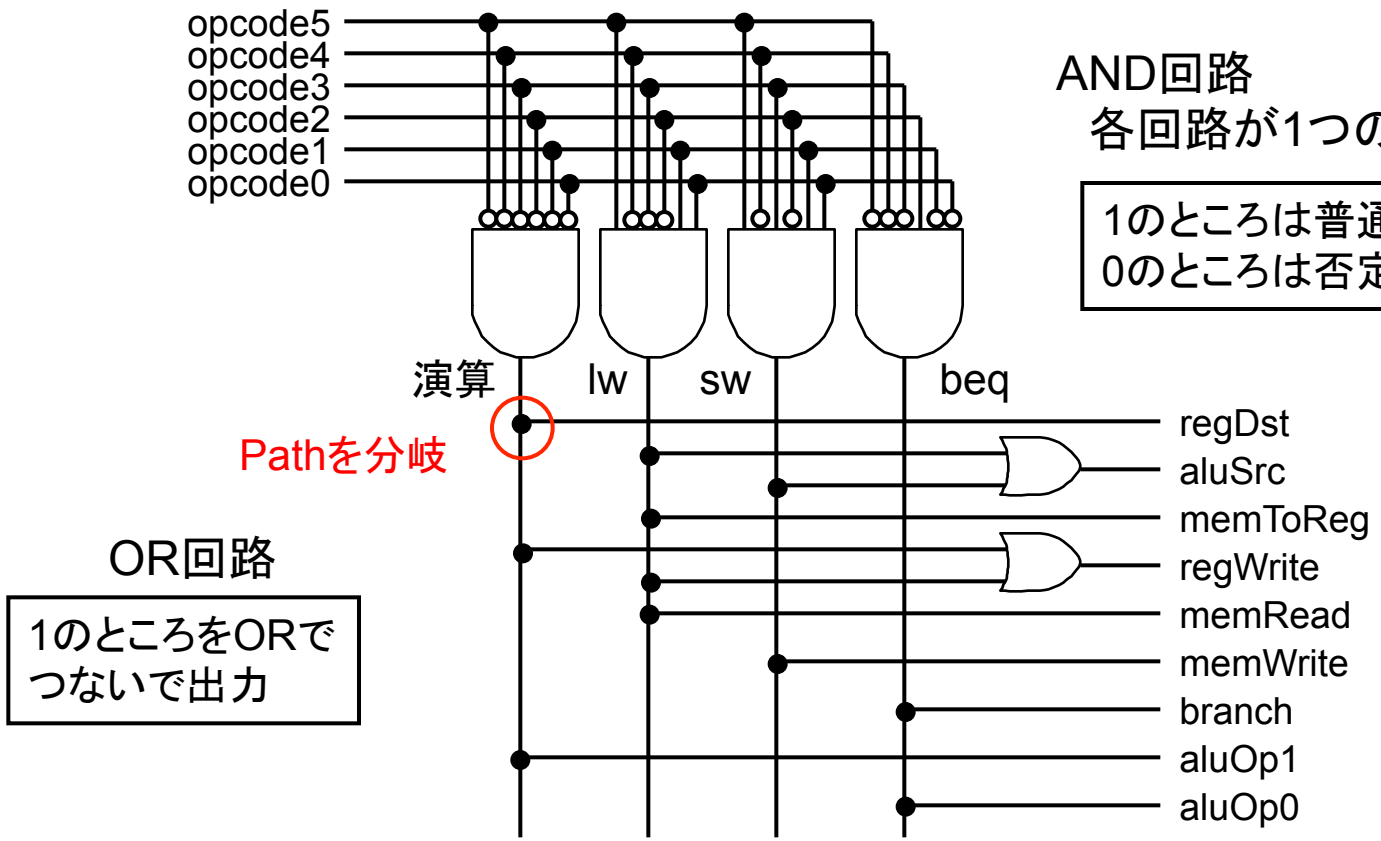
メインコントロールユニットの真理値表

入出力	配線名	演算	lw	sw	beq
入力	opcode5	0	1	1	0
	opcode4	0	0	0	0
	opcode3	0	0	1	0
	opcode2	0	0	0	1
	opcode1	0	1	1	0
	opcode0	0	1	1	0
出力	regDst	1	0	X	X
	aluSrc	0	1	1	0
	memToReg	0	1	X	X
	regWrite	1	1	0	0
	memRead	0	1	0	0
	memWrite	0	0	1	0
	branch	0	0	0	1
	aluOp1	1	0	0	0
	aluOp0	0	0	0	1

命令の種類を
決めるビット列
（命令に含まれる）

メインコントロールユニットの回路

- PLA（Programmable Logic Array）で作成



AND回路
各回路が1つの演算タイプを表す

1のところは普通の入力
0のところは否定を入力

OR回路

1のところをORで
つないで出力

ControlUnit クラス

```
public class ControlUnit {
  public ControlUnit(Bus opcode,
                Path regDst, Path aluSrc, Path memToReg,
                Path regWrite, Path memWrite, Path memRead,
                Path branch, Path[] aluOp) {
     :
}
  public void run() {
     :
  }
}
```

DUP（duplicate） クラス

- 1つの配線を2つに分岐

```
public class DUP {
  public DUP(Path in1, Path out1, Path out2) {
     :
  }
  public void run() {
    Signal sig = in1.readSignal();
    out1.setSignal(sig);
    out2.setSignal(sig);
  }
}
```

例

```
... = new ANDGateN(..., beqLine);
... = new DUP(beqLine, branch, aluOp0);
```

ALUコントロールユニット

- ALU の使い方に関する制御
  - メインコントロールユニットから分離

命令	命令の種類（メインコントロールユニットから）		演算の種類（命令のビット列([5-0])から）						[2][1][0]
	aluOp1	aluOp0	funct5	funct4	funct3	funct2	funct1	funct0	op
lw, sw	0	0	X	X	X	X	X	X	0 1 0
beq	X	1	X	X	X	X	X	X	1 1 0
add	1	X	X	X	0	0	0	0	0 1 0
sub	1	X	X	X	0	0	1	0	1 1 0
and	1	X	X	X	0	1	0	0	0 0 0
or	1	X	X	X	0	1	0	1	0 0 1
slt	1	X	X	X	1	0	1	0	1 1 1

op毎の真理値表

op[2]: Binvert
op[1][0]: Operation(MUX)

op[2]=1

aluOp1	aluOp0	funct5	funct4	funct3	funct2	funct1	funct0
X	1	X	X	X	X	X	X
1	X	X	X	X	X	1	X

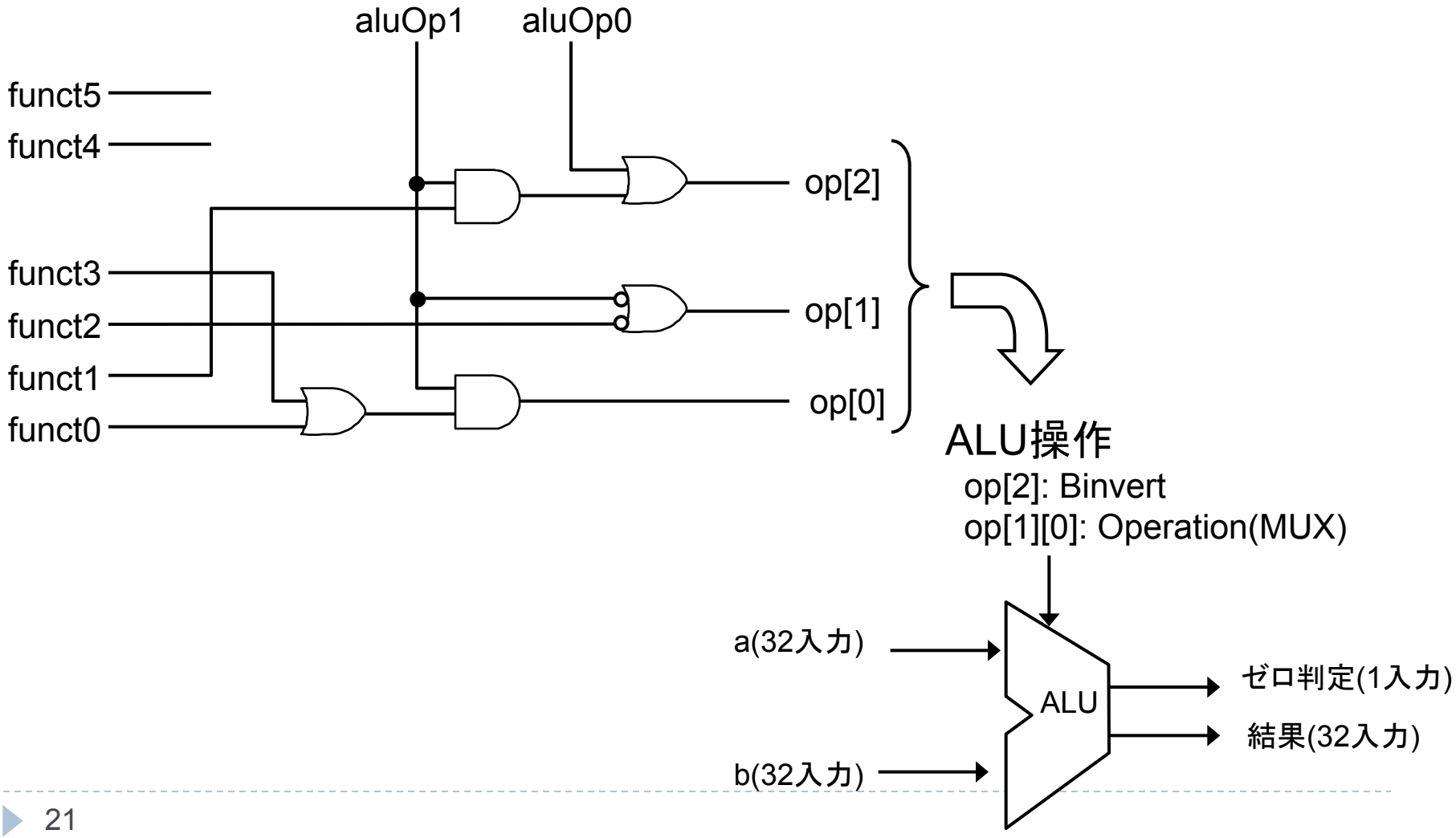
op[1]=1

aluOp1	aluOp0	funct5	funct4	funct3	funct2	funct1	funct0
0	X	X	X	X	X	X	X
X	X	X	X	X	0	X	X

op[0]=1

aluOp1	aluOp0	funct5	funct4	funct3	funct2	funct1	funct0
1	X	X	X	X	X	X	1
1	X	X	X	1	X	X	X

ALUコントロールユニットの回路



ALUControlUnit クラス

```
public class ALUControlUnit {
  public ALUControlUnit(Bus funct, Path[] aluOp,
                        Path[] op) {
    :
  }
  public void run() {
    :
  }
}
```

MIPS クラス（1/2）

```
public class MIPS {
  // PC, RegisterFile, InstMemory, ALU32,
  // ControlUnit, ALUControlUnit が必要
  public MIPS() {
    // ---回路作成---
    pc = new PC(...);
    iMem = new InstMemory(...);
    regFile = new RegisterFile(...);
    // -------------
    pc.setValue(0x04000000);  // PCに実行開始アドレスを設定
    iMem.setInst(0x04000000, 機械語命令);  // メモリに命令を格納
                              //例） add $t0, $t1, $t2 =>  0x012a4020
    regFile.setValue(9, 0x100);  // $t1（9番レジスタ）に代入
    regFile.setValue(10, 0x300);  // $t2（10番レジスタ）に代入
  }
    :
```

MIPS クラス（2/2）

```
    :
  public void run() {
      :
    regFile.run();
    alu.run();
    regFile.run();   // ALUの計算結果を書き込むためにもう一度
      :
    // $t0(8番レジスタ)の内容を表示する場合
    System.out.println(regFile.getValue(8));
  }
  public static void main(String[] args) {
    new MIPS().run();   // 実行を開始
  }
}
```

Register クラス（メソッド追加）

- レジスタに値を設定したり読み出したりできるようにする
  - PC クラスにも同様のメソッドを追加
  - テスト、初期値設定のために使用

```
public class Register {
     :
  public void setValue(int val) {
    this.val = val;
  }
  public int getValue() {
    return val;
  }
     :
}
```

RegisterFile クラス（メソッド追加）

- レジスタファイル内のレジスタの値を操作できるようにする
  - テスト、初期値設定のために使用すること

```
public class RegisterFile {
     :
  public void setValue(int regNum, int val) {
    regs[regNum].setValue(val);
  }
  public int getValue(int regNum) {
    return regs[regNum].getValue();
  }
     :
}
```

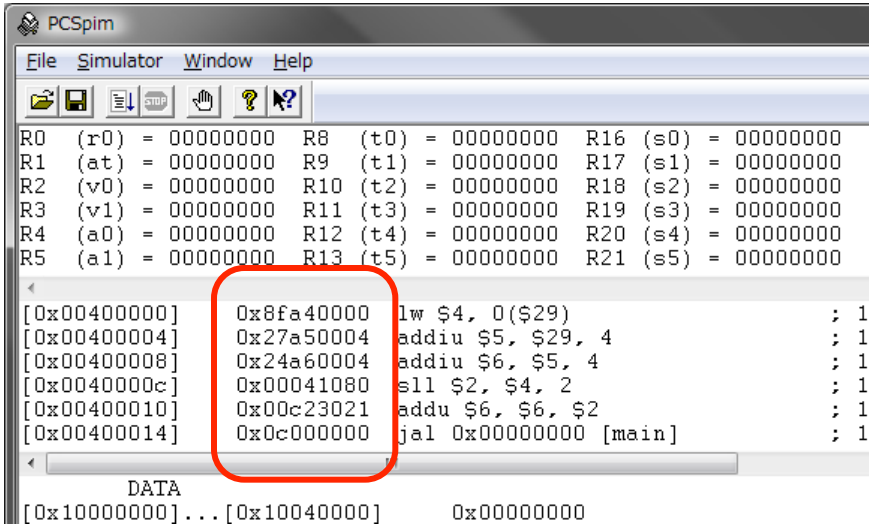
課題

遅れても採点します。

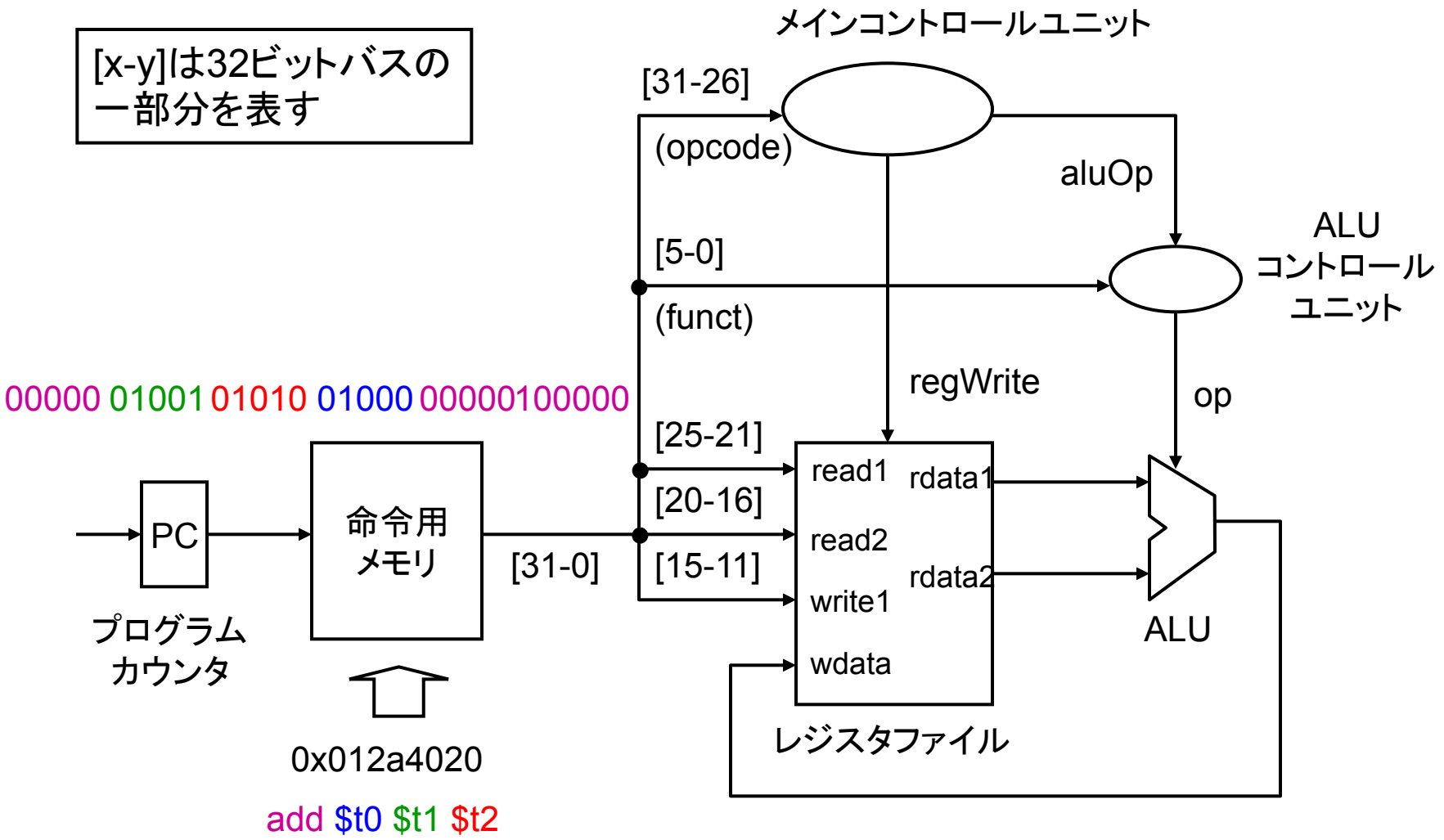
課題

- 「算術論理演算回路」を作成せよ
  - add, sub, and, or, slt をそれぞれ実行し、結果をまとめる
    - 機械語命令は qtspim の Text Segments の2列目を見れば分かる

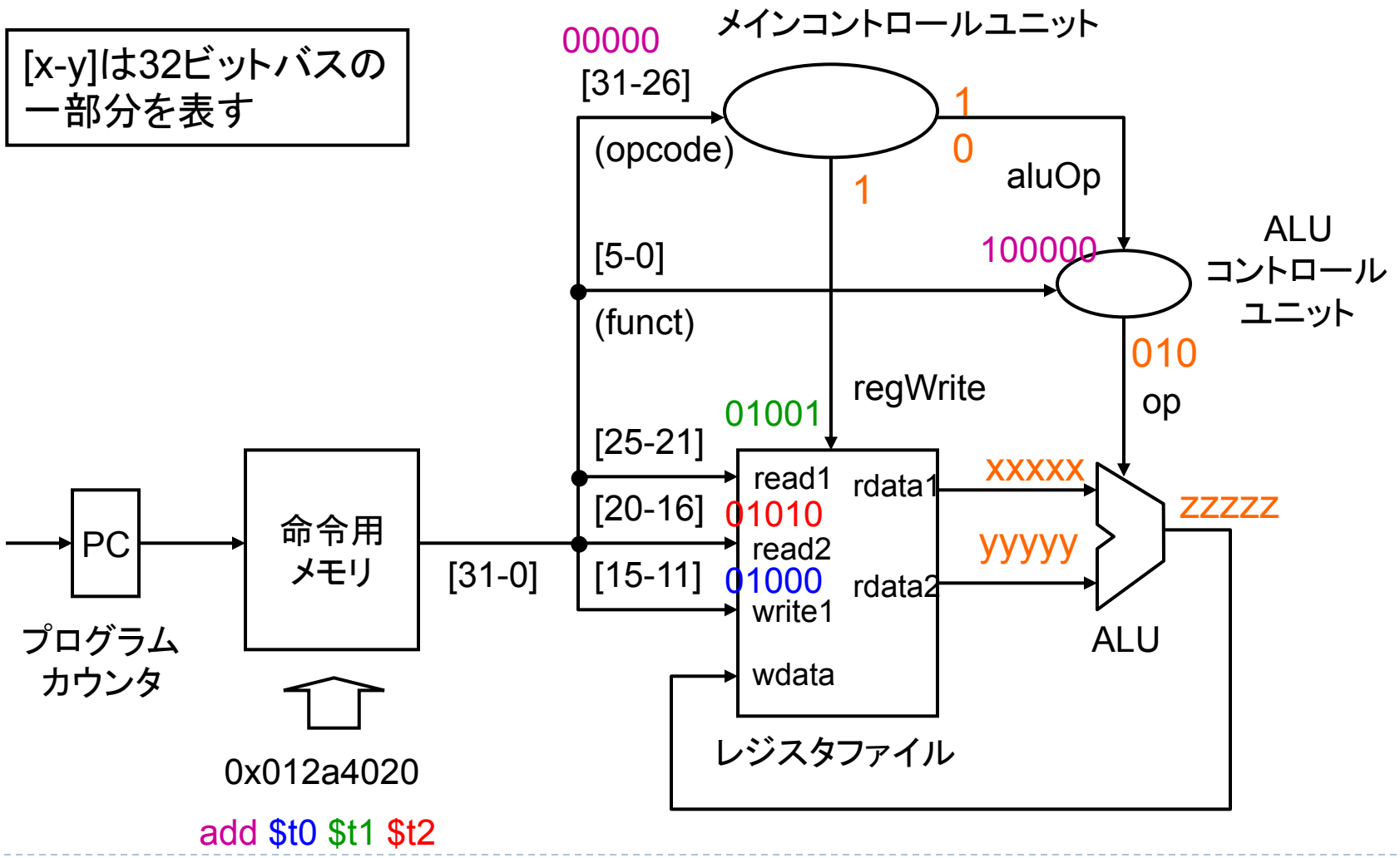
アセンブリ命令	機械語命令
`add $t0, $t1, $t2`	`0x012a4020`
`sub $t0, $t1, $t2`	`0x012a4022`
`and $t0, $t1, $t2`	`0x012a4024`
`or  $t0, $t1, $t2`	`0x012a4025`
`slt $t0, $t1, $t2`	`0x012a402a`



Appendix: 動作の流れ



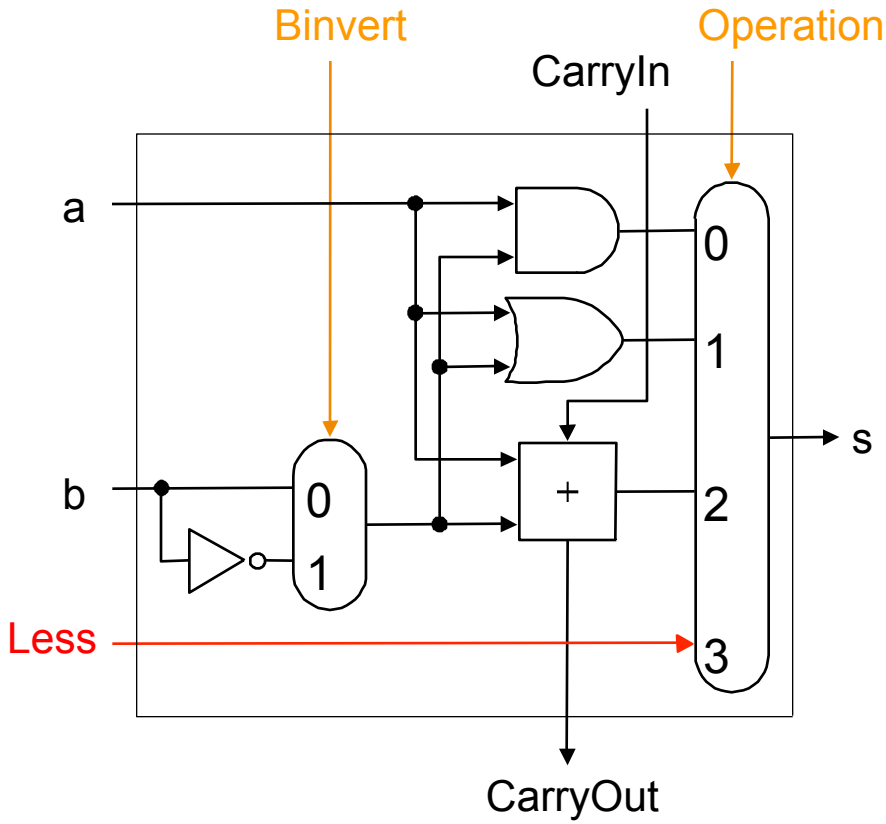
Appendix: 動作の流れ



Appendix: 1ビットALU （ver. 3）

- sltのサポート
- 入力 Less を追加
  - MUX で選択されると Less の値をそのまま出力

Binvert	Operation	演算
0	00	AND
0	01	OR
0	10	加算
1	10	減算
1	11	slt



課題提出

- 〆切：7/23（20日は組み立て演習のため）
- 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
    - 必要なソースファイルを一揃い出してください（前回からの差分のみだと採点が大変なので）
- 提出方法：Webから提出