情報通信実験3

RISC-Vプロセッサ基本設計編

一色 剛

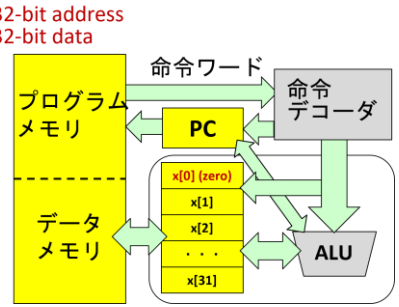
工学院情報通信系

isshiki@ict.e.titech.ac.jp

資料概要

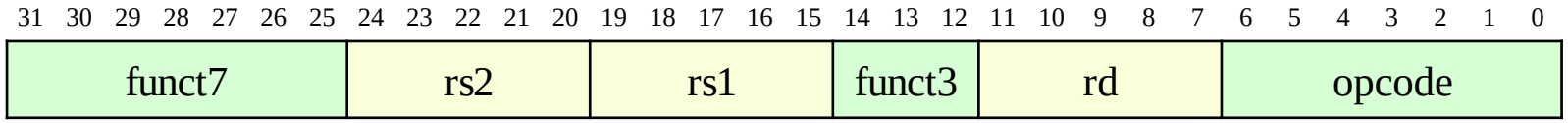
1. RV32-I命令セット仕様
2. RV32-Iハードウェアアーキテクチャ（ARCH-1）
3. 命令デコード動作
4. メモリ読出し･書込み動作
5. 命令フェッチサイクル
6. 命令実行サイクルのレジスタ転送記述
7. データパス制御論理設計
8. 算術論理演算器設計

RV32-I命令セット仕様



- **オペランド**：演算対象データ、格納データ
  - レジスタオペランド：5ビットでx[k] (k = 0, 1, … 31)を指定 (x[0] = 0に固定)
  - 格納レジスタがx[0]の場合は、演算結果は格納されない
  - メモリオペランド：レジスタへのロード命令、ストア命令のみ
  - 即値オペランド：命令ワードに埋め込まれた定数
- **即値（定数）オペランド**：様々なビット長と符号なし・付きの組合せ
  - simm[11:0] , simm[12:1] , simm[20:1] ：符号付き即値（32ビットへ符号拡張）
  - uimm[4:0] , uimm[31:12] ：符号なし即値（32ビットへ0拡張）
- **整数データ型**：32-bit (Word), 16-bit (Half), 8-bit (Byte)
  - 符号なし（unsigned）、符号付き（signed）
- **演算命令**：レジスタ格納、レジスタ・即値オペランド
  - 算術演算（加減算）、シフト演算（右シフト、左シフト）、論理演算（AND/OR/EXOR）、比較演算
- **プログラム制御命令**：サブルーチン呼び出し, 条件分岐
- **メモリアクセス命令**：
  - ロード(読出し)：Byte (signed/unsigned), Half (signed/unsigned), Word
  - ストア(書込み)：Byte, Half, Word (signed/unsignedの区別はない)
- **その他の命令（本実験では未使用）**：
  - FENCE命令：前後のメモリアクセス命令の実行順序制御
  - ECALL, EBREAK命令：システム制御

RV32-I命令セット：演算命令（レジスタオペランド）

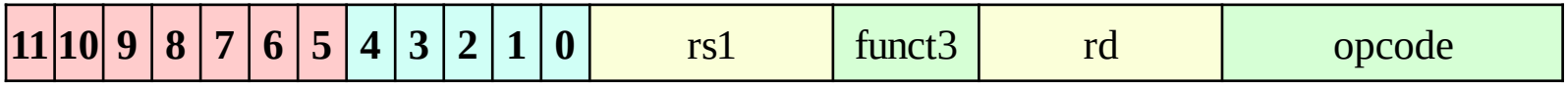


R-type

種別	アセンブリ命令	レジスタ転送式	funct7	funct3	opcode
R-type	ADD rd, rs1, rs2	$rd \leftarrow rs1 + rs2$	0000000	000	0110011
	SUB rd, rs1, rs2	$rd \leftarrow rs1 - rs2$	0100000	000	0110011
	SLT rd, rs1, rs2	$rd \leftarrow rs1 < rs2$ (signed比較)	0000000	010	0110011
	SLTU rd, rs1, rs2	$rd \leftarrow rs1 < rs2$ (unsigned比較)	0000000	011	0110011
	AND rd, rs1, rs2	$rd \leftarrow rs1 \,\&\, rs2$ (ビット毎の論理積)	0000000	111	0110011
	OR rd, rs1, rs2	$rd \leftarrow rs1 \mid rs2$ (ビット毎の論理和)	0000000	110	0110011
	XOR rd, rs1, rs2	$rd \leftarrow rs1 \,\hat{}\, rs2$ (ビット毎の排他的論理和)	0000000	100	0110011
	SLL rd, rs1, rs2	$rd \leftarrow rs1 \ll rs2[4:0]$ (論理左シフト)	0000000	001	0110011
	SRL rd, rs1, rs2	$rd \leftarrow rs1 \gg rs2[4:0]$ (論理右シフト)	0000000	101	0110011
	SRA rd, rs1, rs2	$rd \leftarrow rs1 \gg rs2[4:0]$ (算術右シフト)	0100000	101	0110011

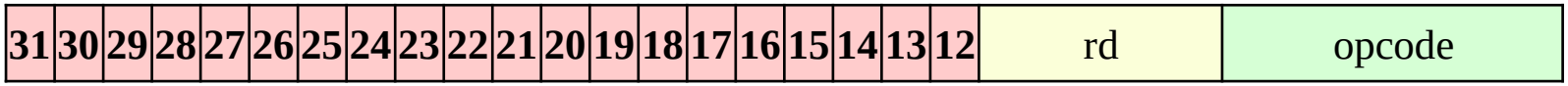
シフト演算（SLL, SRL, SRA）：rs2の下位5ビットでシフト量（0～31）が決まり、上位27ビットは無視される

RV32-I命令セット：演算命令（即値オペランド）



I-type

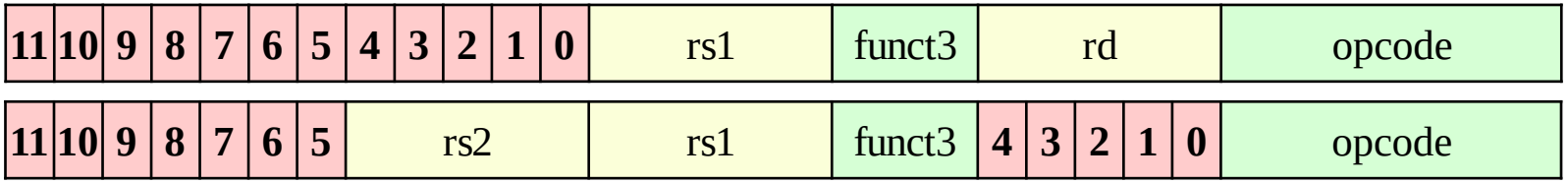
$simm_{[11:0]}$, $uimm_{[4:0]}$



U-type

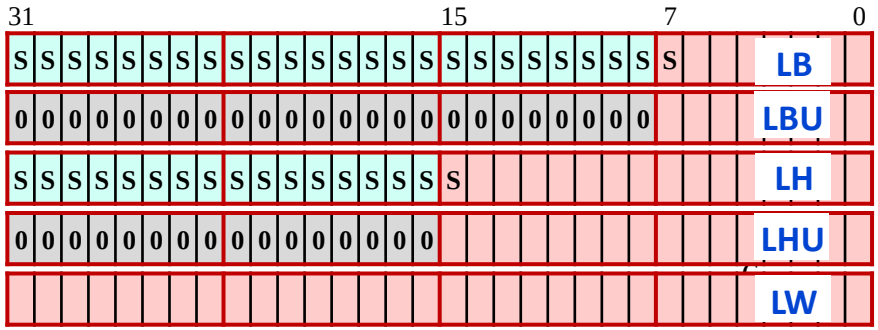
種別	アセンブリ命令	レジスタ転送式	imm[11:5]	funct3	opcode
I-type	ADDI rd, rs1, simm	$rd \leftarrow rs1 + simm[11:0]$		000	0010011
	SLTI rd, rs1, simm	$rd \leftarrow rs1 < simm[11:0]$ (signed比較)		010	0010011
	SLTIU rd, rs1, simm	$rd \leftarrow rs1 < simm[11:0]$ (unsigned比較)		011	0010011
	ANDI rd, rs1, simm	$rd \leftarrow rs1 \,\&\, simm[11:0]$ (ビット毎の論理積)		111	0010011
	ORI rd, rs1, simm	$rd \leftarrow rs1 \mid simm[11:0]$ (ビット毎の論理和)		110	0010011
	XORI rd, rs1, simm	$rd \leftarrow rs1 \,\hat{}\, simm[11:0]$ (ビット毎の排他的論理和)		100	0010011
	SLLI rd, rs1, uimm	$rd \leftarrow rs1 \ll uimm[4:0]$ (論理左シフト)	0000000	001	0010011
	SRLI rd, rs1, uimm	$rd \leftarrow rs1 \gg uimm[4:0]$ (論理右シフト)	0000000	101	0010011
	SRAI rd, rs1, uimm	$rd \leftarrow rs1 \gg uimm[4:0]$ (算術右シフト)	0100000	101	0010011
U-type	LUI rd, uimm	$rd \leftarrow uimm[31:12]$			0110111
	AUIPC rd, uimm	$rd \leftarrow PC + uimm[31:12]$			0010111

RV32-I命令セット：メモリアクセス命令

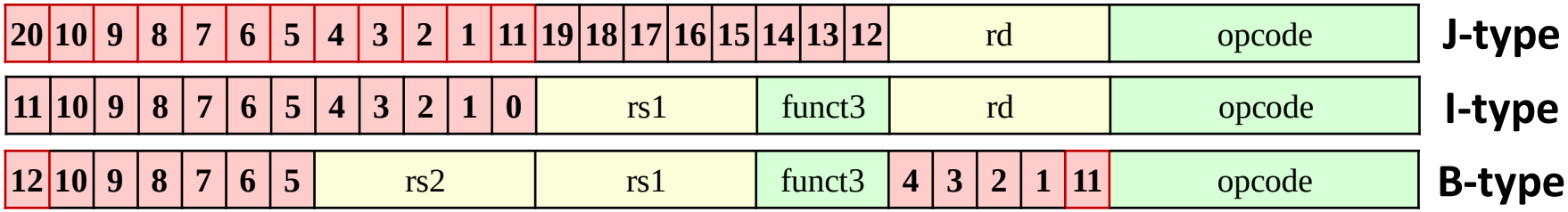


種別	アセンブリ命令	レジスタ転送式	funct3	opcode
I-type	LB rd, simm (rs1)	$rd \leftarrow SignExt\ (M_8[rs1 + simm[11:0]])$	000	0000011
	LBU rd, simm (rs1)	$rd \leftarrow ZeroExt(M_8[rs1 + simm[11:0]])$	100	0000011
	LH rd, simm (rs1)	$rd \leftarrow SignExt(M_{16}[rs1 + simm[11:0]])$	001	0000011
	LHU rd, simm (rs1)	$rd \leftarrow ZeroExt(M_{16}[rs1 + simm[11:0]])$	101	0000011
	LW rd, simm (rs1)	$rd \leftarrow M_{32}[rs1 + simm[11:0]]$	010	0000011
S-type	SB rs2, simm (rs1)	$M_8[rs1 + simm[11:0]] \leftarrow rs2[7:0]$	000	0100011
	SH rs2, simm (rs1)	$M_{16}[rs1 + simm[11:0]] \leftarrow rs2[15:0]$	001	0100011
	SW rs2, simm (rs1)	$M_{32}[rs1 + simm[11:0]] \leftarrow rs2[31:0]$	010	0100011

$M_8[addr]$: メモリ上のaddr番地の8ビットデータ
$M_{16}[addr]$: メモリ上のaddr番地の16ビットデータ
$M_{32}[addr]$: メモリ上のaddr番地の32ビットデータ
$SignExt\ (X)$: Xの符号拡張（signedビット長拡張）
$ZeroExt\ (X)$: Xの0拡張（unsignedビット長拡張）



RV32-I命令セット：プログラム制御命令



種別	アセンブリ命令	レジスタ転送式	funct3	opcode
J-type	JAL rd, *paddr*	$rd \leftarrow PC+4, PC \leftarrow PC+simm[20:1]$		1101111
I-type	JALR rd, simm(rs1)	$rd \leftarrow PC+4, PC \leftarrow ((rs1+simm[11:0])\&(\sim 1))$	000	1100111
B-type	BEQ rs1, rs2, *paddr*	$\mathbf{if}(rs1==rs2)\ PC \leftarrow PC+simm[12:1]$	000	1100011
	BNE rs1, rs2, *paddr*	$\mathbf{if}(rs1!=rs2)\ PC \leftarrow PC+simm[12:1]$	001	1100011
	BLT rs1, rs2, *paddr*	$\mathbf{if}(rs1<rs2)\ PC \leftarrow PC+simm[12:1]$ (signed比較)	100	1100011
	BLTU rs1, rs2, *paddr*	$\mathbf{if}(rs1<rs2)\ PC \leftarrow PC+simm[12:1]$ (unsigned比較)	110	1100011
	BGE rs1, rs2, *paddr*	$\mathbf{if}(rs1\geq rs2)\ PC \leftarrow PC+simm[12:1]$ (signed比較)	101	1100011
	BGEU rs1, rs2, *paddr*	$\mathbf{if}(rs1\geq rs2)\ PC \leftarrow PC+simm[12:1]$ (unsigned比較)	111	1100011

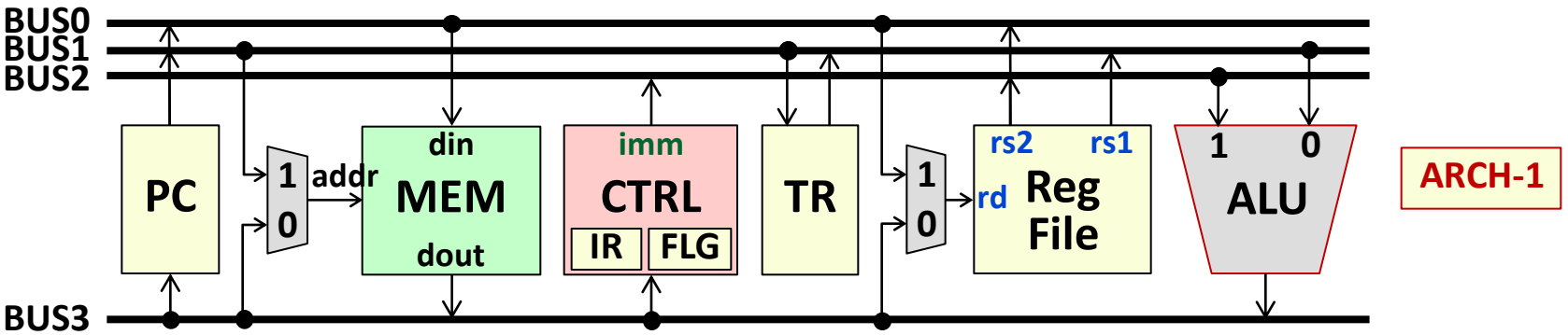
paddr：ジャンプ先のPC → 現PCの相対オフセット値$simm[20:1], simm[12:1]$で指定

- $((rs1+simm[11:0])\&(\sim 1))$：最下位ビットを0にクリア($(\sim 1)=$ 0xfffffffe)
- PCの値（命令アドレス）は2の倍数：命令長が4バイト、2バイト（C拡張の場合）

資料概要

1. RV32-I命令セット仕様
2. RV32-Iハードウェアアーキテクチャ（ARCH-1）
3. 命令デコード動作
4. メモリ読出し･書込み動作
5. 命令フェッチサイクル
6. 命令実行サイクルのレジスタ転送記述
7. データパス制御論理設計
8. 算術論理演算器設計

RV32-Iハードウェアアーキテクチャ（ARCH-1）



→ 命令実行制御設計のための計算機詳細構成図の一例（ARCH-1）

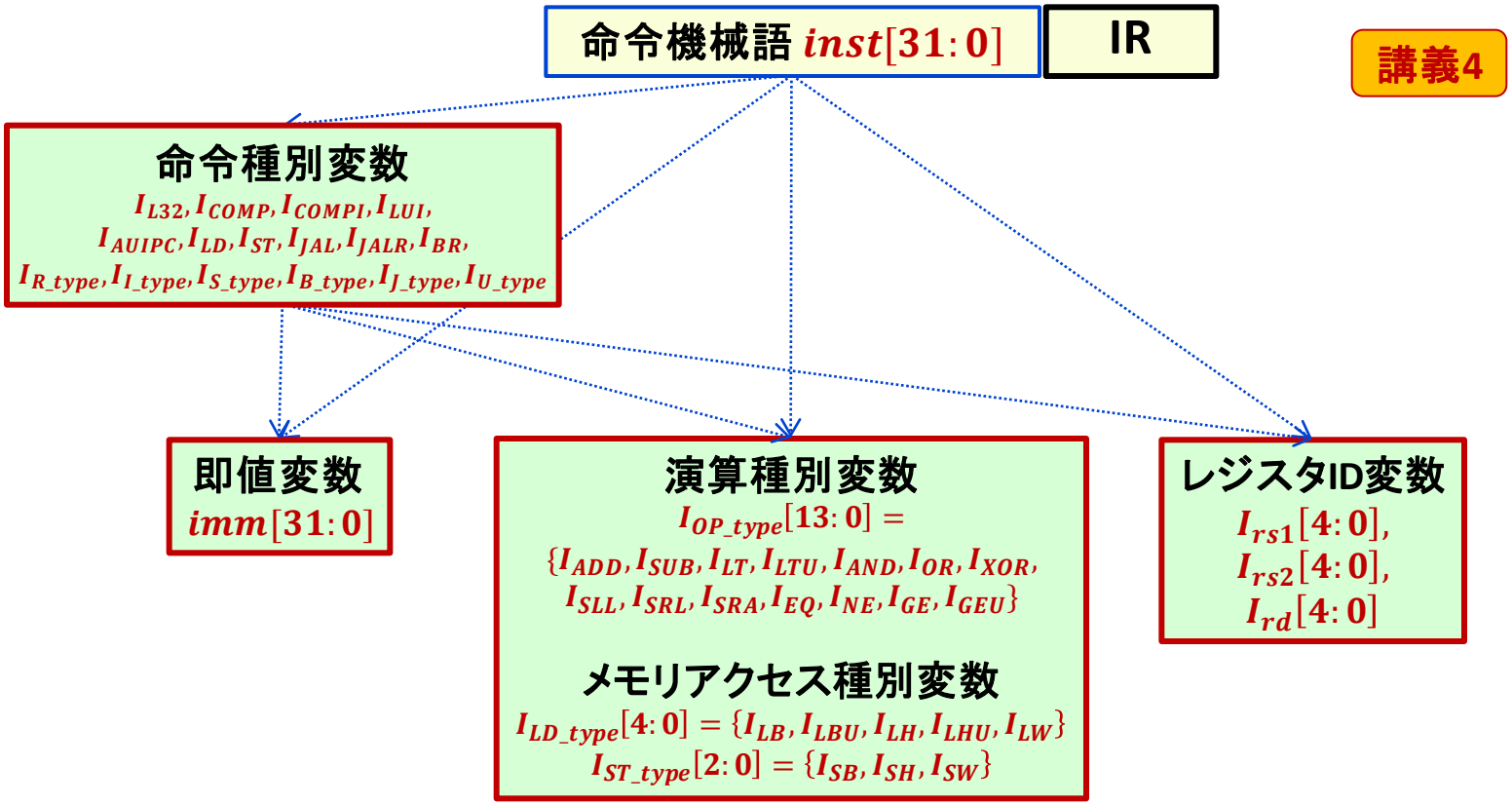
- MEM (メモリ) : in {addr:{BUS1, BUS3}, din:BUS0}, out {dout:BUS3}
- ALU : in {0:BUS1, 1:BUS2}, out {BUS3}
- RegFile (x[0]～x[31]) : in {rd:{BUS0, BUS3}}, out {rs1:BUS1, rs2:{BUS0, BUS2}}
- PC (プログラムカウンタ) : in {BUS3}, out {BUS0, BUS1}
- TR (一時レジスタ : PCの値を保持) : in {BUS1}, out {BUS1}
- CTRL (制御回路) : in {BUS3}, out {imm:BUS2}
  - 命令レジスタIR（inst[31:0]）、1ビットフラグレジスタFLG（比較結果）
  - out : 即値データ出力（imm[31:0]）
  - 制御出力（上図では省略）: ALU_{CTRL} , MEM_{CTRL}, $RegFile_{CTRL}$, PC_{CTRL}, TR_{CTRL}, . . .

資料概要

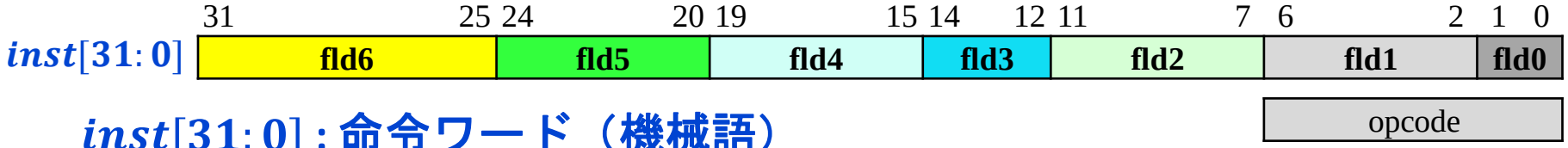
1. RV32-I命令セット仕様
2. RV32-Iハードウェアアーキテクチャ（ARCH-1）
3. 命令デコード動作
4. メモリ読出し･書込み動作
5. 命令フェッチサイクル
6. 命令実行サイクルのレジスタ転送記述
7. データパス制御論理設計
8. 算術論理演算器設計

命令デコード動作

- 命令フェッチ終了後、命令デコード回路によって、命令情報変数（命令種別、即値、演算種別、メモリアクセス種別、レジスタID変数、等）が制御回路（CTRL）から出力される



命令種別変数の論理式



$inst[31:0]$: 命令ワード（機械語）

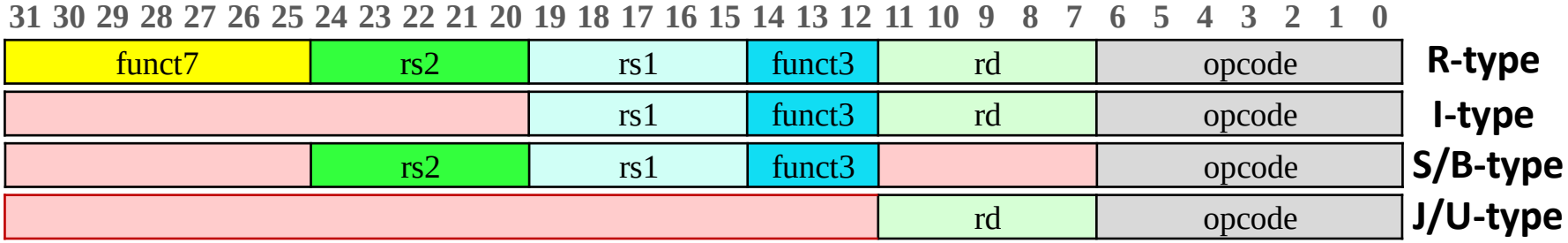
命令種別	type	opcode	命令種別変数の論理式
			$I_{L32} = inst[1] \cdot inst[0]$（32ビット命令判定項）
演算命令（レジスタ）	R-type	0110011 (0x33)	$I_{COMP} = \overline{inst[6]} \cdot inst[5] \cdot inst[4] \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
演算命令（即値）	I-type	0010011 (0x13)	$I_{COMPI} = \overline{inst[6]} \cdot \overline{inst[5]} \cdot inst[4] \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
LUI 命令	U-type	0110111 (0x37)	$I_{LUI} = \overline{inst[6]} \cdot inst[5] \cdot inst[4] \cdot \overline{inst[3]} \cdot inst[2] \cdot I_{L32}$
AUIPC 命令	U-type	0010111 (0x17)	$I_{AUIPC} = \overline{inst[6]} \cdot \overline{inst[5]} \cdot inst[4] \cdot \overline{inst[3]} \cdot inst[2] \cdot I_{L32}$
ロード命令	I-type	0000011 (0x03)	$I_{LD} = \overline{inst[6]} \cdot \overline{inst[5]} \cdot \overline{inst[4]} \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
ストア命令	S-type	0100011 (0x23)	$I_{ST} = \overline{inst[6]} \cdot inst[5] \cdot \overline{inst[4]} \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
JAL命令	J-type	1101111 (0x6f)	$I_{JAL} = inst[6] \cdot inst[5] \cdot \overline{inst[4]} \cdot inst[3] \cdot inst[2] \cdot I_{L32}$
JALR命令	I-type	1100111 (0x67)	$I_{JALR} = inst[6] \cdot inst[5] \cdot \overline{inst[4]} \cdot \overline{inst[3]} \cdot inst[2] \cdot I_{L32}$
条件分岐命令	B-type	1100011 (0x63)	$I_{BR} = inst[6] \cdot inst[5] \cdot \overline{inst[4]} \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
			各typeの命令種別変数
			$I_{R\_type} = I_{COMP}$
			$I_{I\_type} = I_{COMPI} + I_{LD} + I_{JALR}$
			$I_{S\_type} = I_{ST}$
			$I_{B\_type} = I_{BR}$
			$I_{J\_type} = I_{JAL}$
			$I_{U\_type} = I_{LUI} + I_{AUIPC}$

$I_{COMP}, I_{COMPI}, I_{LUI}, I_{AUIPC}, I_{LD}, I_{ST}, I_{JAL}, I_{JALR}, I_{BR}$

$I_{R\_type}, I_{I\_type}, I_{S\_type}, I_{B\_type}, I_{J\_type}, I_{U\_type}$

互いに排他的な論理変数集合
（同時に１にならない）

レジスタID変数の論理設計



レジスタIDフィールド：

- $I_{rs1}[4:0] = inst[19:15]$
- $I_{rs2}[4:0] = inst[24:20]$
- $I_{rd}[4:0] = inst[11:7]$

レジスタファイル：$x[32]$ → 32個 x 32ビットレジスタ

- $rs1 \leftarrow I_{rs1}\ ?\ x[I_{rs1}] : 32'b0$
- $rs2 \leftarrow I_{rs2}\ ?\ x[I_{rs2}] : 32'b0$
- $\text{if}(I_{rd})\ x[I_{rd}] \leftarrow rd$

- x[0] = 0に固定
- x[0]への書込みは無視

レジスタID条件式	条件式の含意
$I_{rs1\_type} = I_{R\_type} + I_{I\_type} + I_{S\_type} + I_{B\_type}$	(rs1) フィールドが存在する
$I_{rs2\_type} = I_{R\_type} + I_{S\_type} + I_{B\_type}$	(rs2) フィールドが存在する
$I_{rd\_type} = I_{R\_type} + I_{I\_type} + I_{J\_type} + I_{U\_type}$	(rd) フィールドが存在する

R-type	$rd \leftarrow rs1\ (op)\ rs2$
I-type	$rd \leftarrow rs1\ (op)\ imm$
……	……

$k'b0$: kビット幅の「0」値

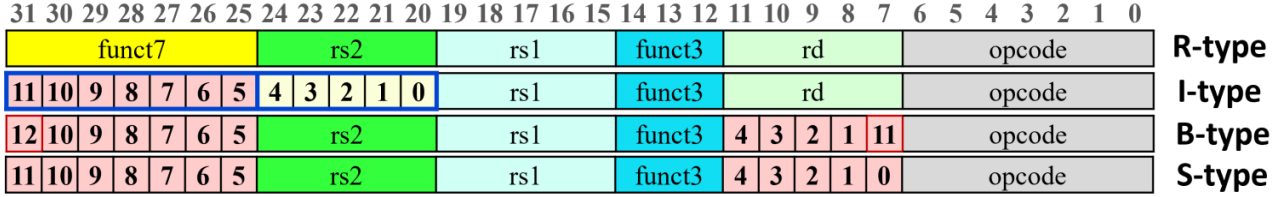
ビット範囲	IDフィールド変数の論理式	IDフィールド変数のレジスタ転送式
$0 \le n \le 4$	$I_{rs1}[n] = I_{rs1\_type} \cdot inst[15+n]$	$I_{rs1}[4:0] = I_{rs1\_type}\ ?\ inst[19:15] : 5'b0$
$0 \le n \le 4$	$I_{rs2}[n] = I_{rs2\_type} \cdot inst[20+n]$	$I_{rs2}[4:0] = I_{rs2\_type}\ ?\ inst[24:20] : 5'b0$
$0 \le n \le 4$	$I_{rd}[n] = I_{rd\_type} \cdot inst[7+n]$	$I_{rd}[4:0] = I_{rd\_type}\ ?\ inst[11:7] : 5'b0$

条件演算子（3項演算子）：

a = (b) ? c : d; → if (b != 0) a = c; else a = d;

演算種別とメモリアクセス種別

命令種別	type	opcode	命令種別変数の論理式
			$I_{L32} = inst[1] \cdot inst[0]$（32ビット命令判定項）
演算命令（レジスタ）	R-type	0110011 (0x33)	$I_{COMP} = \overline{inst[6]} \cdot inst[5] \cdot inst[4] \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
演算命令（即値）	I-type	0010011 (0x13)	$I_{COMPI} = \overline{inst[6]} \cdot \overline{inst[5]} \cdot inst[4] \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
ロード命令	I-type	0000011 (0x03)	$I_{LD} = \overline{inst[6]} \cdot \overline{inst[5]} \cdot \overline{inst[4]} \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
ストア命令	S-type	0100011 (0x23)	$I_{ST} = \overline{inst[6]} \cdot inst[5] \cdot \overline{inst[4]} \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
条件分岐命令	B-type	1100011 (0x63)	$I_{BR} = inst[6] \cdot inst[5] \cdot \overline{inst[4]} \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$



アセンブリ命令 (R-type : I_{COMP})	funct3 inst[14:12]	funct7 inst[31:25]
ADD rd, rs1, rs2	000	0000000
SUB rd, rs1, rs2	000	0100000
SLT rd, rs1, rs2	010	0000000
SLTU rd, rs1, rs2	011	0000000
AND rd, rs1, rs2	111	0000000
OR rd, rs1, rs2	110	0000000
XOR rd, rs1, rs2	100	0000000
SLL rd, rs1, rs2	001	0000000
SRL rd, rs1, rs2	101	0000000
SRA rd, rs1, rs2	101	0100000

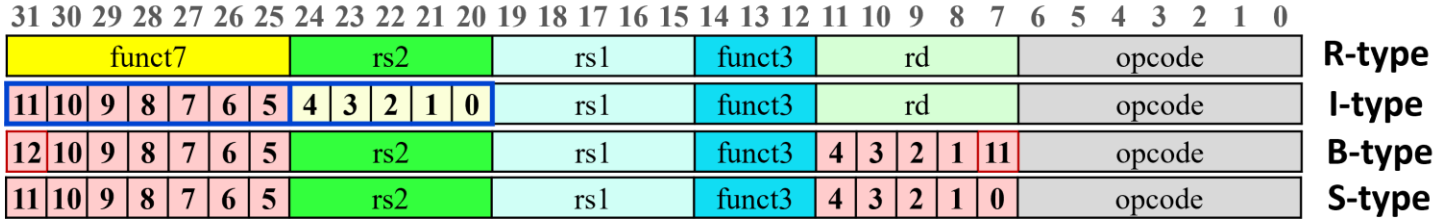
アセンブリ命令 (I-type : I_{COMPI})	funct3 inst[14:12]	imm[11:5] inst[31:25]
ADDI rd, rs1, simm	000	
SLTI rd, rs1, simm	010	
SLTIU rd, rs1, simm	011	
ANDI rd, rs1, simm	111	
ORI rd, rs1, simm	110	
XORI rd, rs1, simm	100	
SLLI rd, rs1, uimm	001	0000000
SRLI rd, rs1, uimm	101	0000000
SRAI rd, rs1, uimm	101	0100000

アセンブリ命令 (I_{LD}, I_{ST})	funct3 inst[14:12]
LB rd, simm (rs1)	000
LBU rd, simm (rs1)	100
LH rd, simm (rs1)	001
LHU rd, simm (rs1)	101
LW rd, simm (rs1)	010
SB rs2, simm (rs1)	000
SH rs2, simm (rs1)	001
SW rs2, simm (rs1)	010

アセンブリ命令 (B-type : I_{BR})	funct3 inst[14:12]
BEQ rs1, rs2, *paddr*	000
BNE rs1, rs2, *paddr*	001
BLT rs1, rs2, *paddr*	100
BLTU rs1, rs2, *paddr*	110
BGE rs1, rs2, *paddr*	101
BGEU rs1, rs2, *paddr*	111

演算種別とメモリアクセス種別

命令種別	type	opcode	命令種別変数の論理式
			$I_{L32} = inst[1] \cdot inst[0]$（32ビット命令判定項）
演算命令（レジスタ）	R-type	0110011 (0x33)	$I_{COMP} = \overline{inst[6]} \cdot inst[5] \cdot inst[4] \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
演算命令（即値）	I-type	0010011 (0x13)	$I_{COMPI} = \overline{inst[6]} \cdot \overline{inst[5]} \cdot inst[4] \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
ロード命令	I-type	0000011 (0x03)	$I_{LD} = \overline{inst[6]} \cdot \overline{inst[5]} \cdot \overline{inst[4]} \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
ストア命令	S-type	0100011 (0x23)	$I_{ST} = \overline{inst[6]} \cdot inst[5] \cdot \overline{inst[4]} \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$
条件分岐命令	B-type	1100011 (0x63)	$I_{BR} = inst[6] \cdot inst[5] \cdot \overline{inst[4]} \cdot \overline{inst[3]} \cdot \overline{inst[2]} \cdot I_{L32}$



$$I_{F7\_Z} = \overline{inst[31]} \cdot \overline{inst[29]} \cdot \overline{inst[28]} \cdot \overline{inst[27]} \cdot \overline{inst[26]} \cdot \overline{inst[25]}$$
$$I_{F7\_0} = I_{F7\_Z} \cdot \overline{inst[30]}$$
$$I_{F7\_32} = I_{F7\_Z} \cdot inst[30]$$

$$I_{COMP\_0} = I_{F7\_0} \cdot I_{COMP} + I_{COMPI}$$
$$I_{COMP\_L} = I_{F7\_0} \cdot (I_{COMP} + I_{COMPI})$$
$$I_{COMP\_A} = I_{F7\_32} \cdot (I_{COMP} + I_{COMPI})$$

$$I_{F3\_0} = \overline{inst[14]} \cdot \overline{inst[13]} \cdot \overline{inst[12]}$$
$$I_{F3\_1} = \overline{inst[14]} \cdot \overline{inst[13]} \cdot inst[12]$$
$$I_{F3\_2} = \overline{inst[14]} \cdot inst[13] \cdot \overline{inst[12]}$$
$$I_{F3\_3} = \overline{inst[14]} \cdot inst[13] \cdot inst[12]$$
$$I_{F3\_4} = inst[14] \cdot \overline{inst[13]} \cdot \overline{inst[12]}$$
$$I_{F3\_5} = inst[14] \cdot \overline{inst[13]} \cdot inst[12]$$
$$I_{F3\_6} = inst[14] \cdot inst[13] \cdot \overline{inst[12]}$$
$$I_{F3\_7} = inst[14] \cdot inst[13] \cdot inst[12]$$

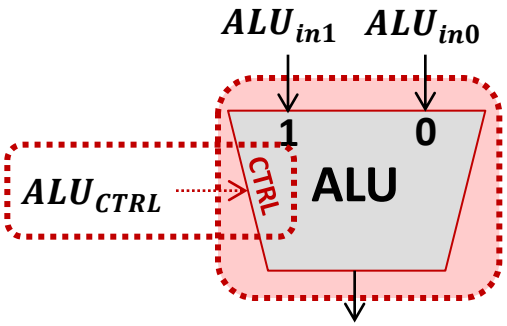
$$I_{LB} = I_{LD} \cdot I_{F3\_0}$$
$$I_{LBU} = I_{LD} \cdot I_{F3\_4}$$
$$I_{LH} = I_{LD} \cdot I_{F3\_1}$$
$$I_{LHU} = I_{LD} \cdot I_{F3\_5}$$
$$I_{LW} = I_{LD} \cdot I_{F3\_2}$$
$$I_{SB} = I_{ST} \cdot I_{F3\_0}$$
$$I_{SH} = I_{ST} \cdot I_{F3\_1}$$
$$I_{SW} = I_{ST} \cdot I_{F3\_2}$$

メモリアクセス種別変数

$$I_{ADD} = I_{F3\_0} \cdot I_{COMP\_0}$$
$$I_{SUB} = I_{F3\_0} \cdot I_{F7\_32} \cdot I_{COMP}$$
$$I_{LT} = I_{F3\_2} \cdot I_{COMP\_0} + I_{F3\_4} \cdot I_{BR}$$
$$I_{LTU} = I_{F3\_3} \cdot I_{COMP\_0} + I_{F3\_6} \cdot I_{BR}$$
$$I_{AND} = I_{F3\_7} \cdot I_{COMP\_0}$$
$$I_{OR} = I_{F3\_6} \cdot I_{COMP\_0}$$
$$I_{XOR} = I_{F3\_4} \cdot I_{COMP\_0}$$
$$I_{SLL} = I_{F3\_1} \cdot I_{COMP\_L}$$
$$I_{SRL} = I_{F3\_5} \cdot I_{COMP\_L}$$
$$I_{SRA} = I_{F3\_5} \cdot I_{COMP\_A}$$
$$I_{EQ} = I_{F3\_0} \cdot I_{BR}$$
$$I_{NE} = I_{F3\_1} \cdot I_{BR}$$
$$I_{GE} = I_{F3\_5} \cdot I_{BR}$$
$$I_{GEU} = I_{F3\_7} \cdot I_{BR}$$

演算種別変数

ALU演算制御論理



$$ALU(ALU_{in0}[31:0], ALU_{in1}[31:0], ALU_{CTRL}[13:0])$$

演算命令と条件分岐命令の演算種別変数

$$I_{OP\_type}[13:0] = \{I_{ADD}, I_{SUB}, I_{LT}, I_{LTU}, I_{AND}, I_{OR}, I_{XOR}, I_{SLL}, I_{SRL}, I_{SRA}, I_{EQ}, I_{NE}, I_{GE}, I_{GEU}\}$$

One-Hot符号：「1」になるビットが高々1つ

命令種別変数	レジスタ転送式
I_{COMP}	$rd \leftarrow rs1\ (op)\ rs2$
I_{COMPI}	$rd \leftarrow rs1\ (op)\ imm$
I_{LUI}	$rd \leftarrow imm$
I_{AUIPC}	$rd \leftarrow PC + imm$
I_{LD}	$rd \leftarrow Load(rs1 + imm, I_{LD\_type})$
I_{ST}	$Store(rs2, rs1 + imm, I_{ST\_type})$
I_{JAL}	$rd \leftarrow PC + 4, PC \leftarrow PC + imm$
I_{JALR}	$rd \leftarrow PC + 4, PC \leftarrow (rs1 + imm)\&(\sim 1)$
I_{BR}	$\text{if}(rs1\ (op)\ rs2)\ PC \leftarrow PC + imm$

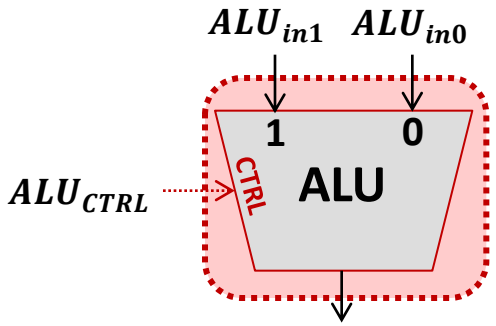
加算(ADD)をALUで実行

条件分岐命令は、
比較演算と加算を2段階で実行

その他の命令での加算実行時のALU制御入力値

$$I_{OP\_ADD}[13:0] = \{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$$

ALU演算制御論理



$$ALU(ALU_{in0}[31:0], ALU_{in1}[31:0], ALU_{CTRL}[13:0])$$

$$I_{OP\_type}[13:0] = \{I_{ADD}, I_{SUB}, I_{LT}, I_{LTU}, I_{AND}, I_{OR}, I_{XOR}, I_{SLL}, I_{SRL}, I_{SRA}, I_{EQ}, I_{NE}, I_{GE}, I_{GEU}\}$$

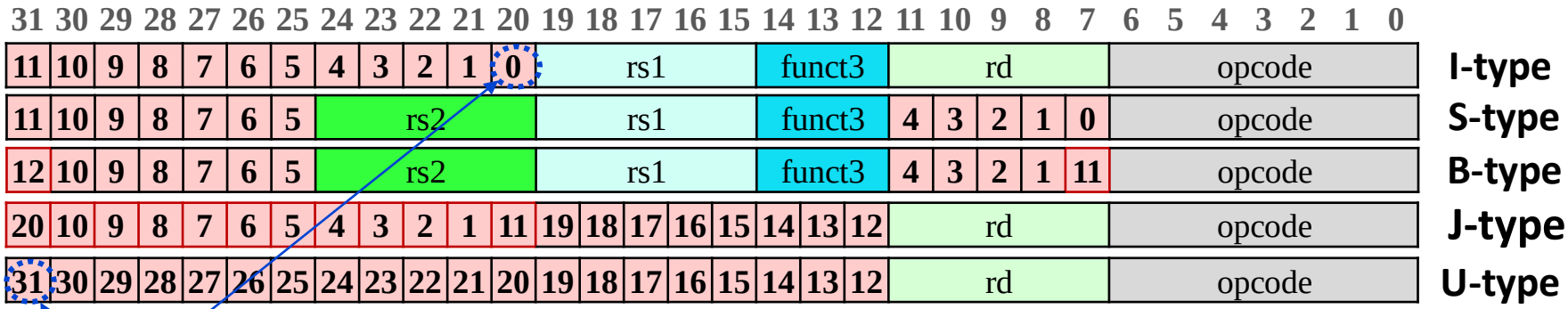
$$I_{OP\_ADD}[13:0] = \{1,0,0,0,0,0,0,0,0,0,0,0,0,0\}$$

命令種別変数	レジスタ転送式（概要構成）	レジスタ転送式（詳細構成）
I_{COMP}	$rd \leftarrow rs1\ (op)\ rs2$	$rd \leftarrow ALU(rs1, rs2, I_{OP\_type})$
I_{COMPI}	$rd \leftarrow rs1\ (op)\ imm$	$rd \leftarrow ALU(rs1, imm, I_{OP\_type})$
I_{LUI}	$rd \leftarrow imm$	$rd \leftarrow ALU(zero, imm, I_{OP\_ADD})$
I_{AUIPC}	$rd \leftarrow PC + imm$	$rd \leftarrow ALU(PC, imm, I_{OP\_ADD})$
I_{LD}	$rd \leftarrow Load(rs1 + imm, I_{LD\_type})$	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD})$
I_{ST}	$Store(rs2, rs1 + imm, I_{ST\_type})$	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD})$
I_{JAL}	$rd \leftarrow PC + 4;\ PC \leftarrow PC + imm$	$PC \leftarrow ALU(PC, imm, I_{OP\_ADD})$
I_{JALR}	$rd \leftarrow PC + 4;\ PC \leftarrow (rs1 + imm)\&(\sim 1)$	$PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$
I_{BR}	$\text{if}(rs1\ (op)\ rs2)\ PC \leftarrow PC + imm$	$FLG \leftarrow ALU(rs1, rs2, I_{OP\_type})$ $\text{if}(FLG)\ PC \leftarrow ALU(PC, imm, I_{OP\_ADD})$

$(PC + 4)$の演算をどこで実行するか？

条件分岐命令は、
比較演算と加算を2段階で実行

命令機械語から即値データへの変換



命令機械語 $inst[31:0]$ のフォーマット

$0 \leq n \leq 31$: 即値データのビット位置 $imm[n]$

$k'b0$: kビット幅の「0」値
$k\{X\}$: Xをk個複製
$\{X, Y\}$: XとYをビット連結

```
imm_i = (I_I_type) ? {20{inst[31]}, inst[31:20]} : 32'b0;
imm_s = (I_S_type) ? {20{inst[31]}, inst[31:25], inst[11:7]} : 32'b0;
imm_b  = (I_B_type) ? {20{inst[31]}, inst[7], inst[30:25], inst[11:8], 1'b0} : 32'b0;
imm_j = (I_J_type) ? {12{inst[31]}, inst[19:12], inst[20], inst[30:21], 1'b0} : 32'b0;
imm_u = (I_U_type) ? {inst[31:20], 12'b0} : 32'b0;
imm = imm_i | imm_s | imm_b | imm_j | imm_u;
```

$I_{I\_type}, I_{S\_type}, I_{B\_type}, I_{J\_type}, I_{U\_type}$ が互いに排他的なため

即値データ $imm[31:0]$ のレジスタ転送式

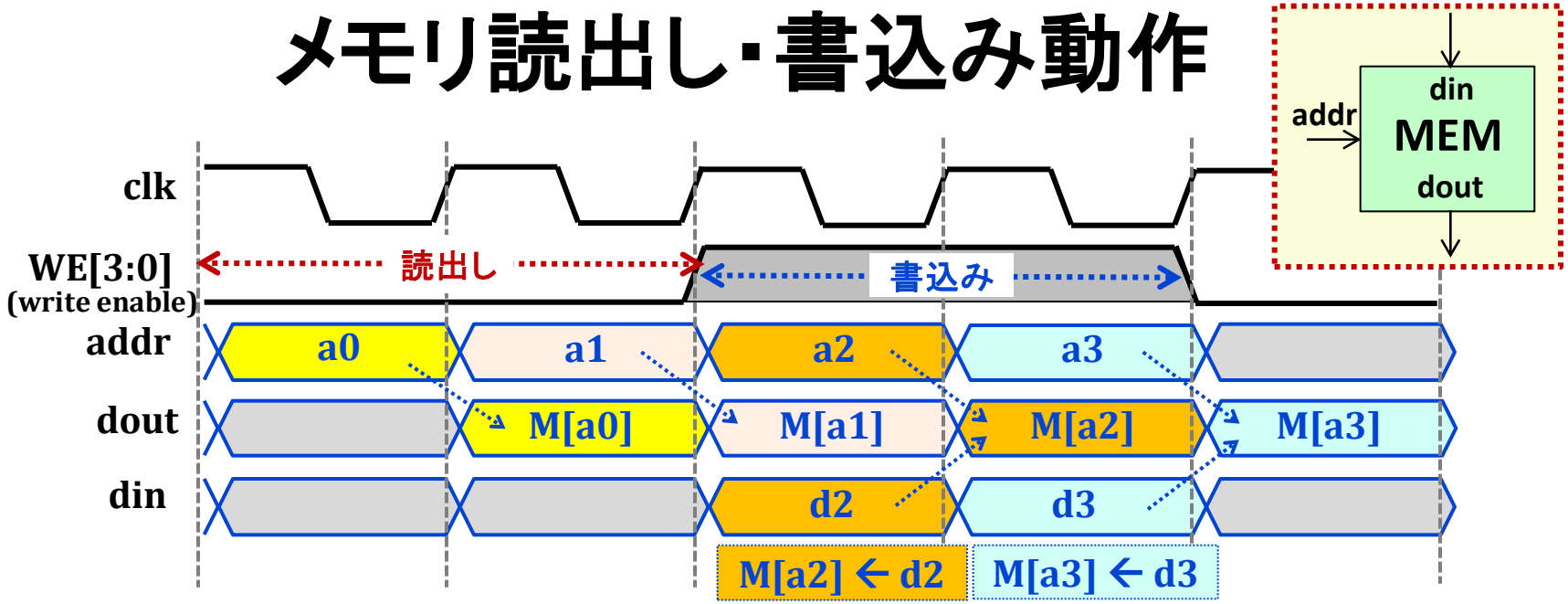
条件演算子(3項演算子) :
a = (b) ? c : d; → if (b != 0) a = c; else a = d;

“|” → 論理OR

資料概要

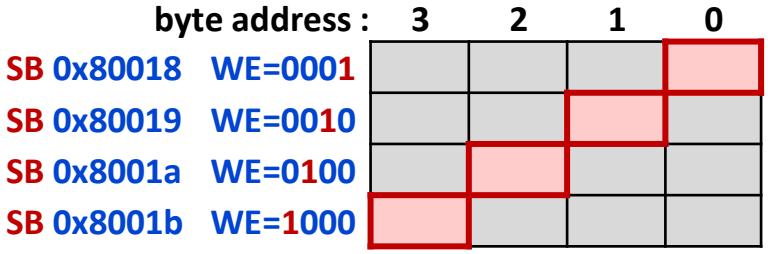
1. RV32-I命令セット仕様
2. RV32-Iハードウェアアーキテクチャ（ARCH-1）
3. 命令デコード動作
4. メモリ読出し･書込み動作
5. 命令フェッチサイクル
6. 命令実行サイクルのレジスタ転送記述
7. データパス制御論理設計
8. 算術論理演算器設計

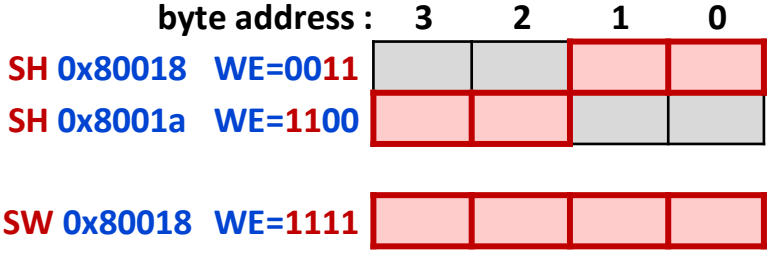
メモリ読出し・書込み動作



- メモリ読出し動作（WE = 4'b0000）：addr入力の次のサイクルでdoutが出力
  - → 命令機械語読出し（命令フェッチ）、ロード命令
- メモリ書込み動作（WE != 4'b0000）：addrとdinが同サイクルで入力
  - → ストア命令：WE[3:0]はバイト単位で書込み許可を制御

講義2





メモリアクセス動作記述

ロード命令	$rd \leftarrow Load(rs1 + imm, I_{LD\_type})$
ストア命令	$Store(rs2, rs1 + imm, I_{ST\_type})$

$I_{LD\_type}[4:0] = \{I_{LB}, I_{LBU}, I_{LH}, I_{LHU}, I_{LW}\}$
$I_{ST\_type}[2:0] = \{I_{SB}, I_{SH}, I_{SW}\}$

One-Hot符号：「1」になるビットが高々1つ

■ ロード命令動作：addr入力の次のサイクルで、doutが出力

1. $MEM.addr \leftarrow rs1 + imm, MEM.SetRead(I_{LD\_type})$
2. $rd \leftarrow MEM.dout$

I_{LB}, I_{LH}：符号拡張(SignExt)
I_{LBU}, I_{LHU}：0拡張(ZeroExt)
I_{LW}：ビット拡張なし

■ ストア命令動作：addrとdinが同じサイクルで入力

1. $MEM.addr \leftarrow rs1 + imm, MEM.din \leftarrow rs2, MEM.SetWrite(I_{ST\_type})$

■ 命令フェッチ動作：addr入力(PC)の次のサイクルで、命令inst[31:0]が出力

1. $MEM.addr \leftarrow PC, MEM.SetRead(I_{LD\_INST})$
2. $IR \leftarrow MEM.dout$

$I_{LD\_type}[4:0] = \{I_{LB}, I_{LBU}, I_{LH}, I_{LHU}, I_{LW}\}$
$I_{LD\_INST}[4:0] = \{0,0,0,0,1\}$ ← I_{LW}を指定

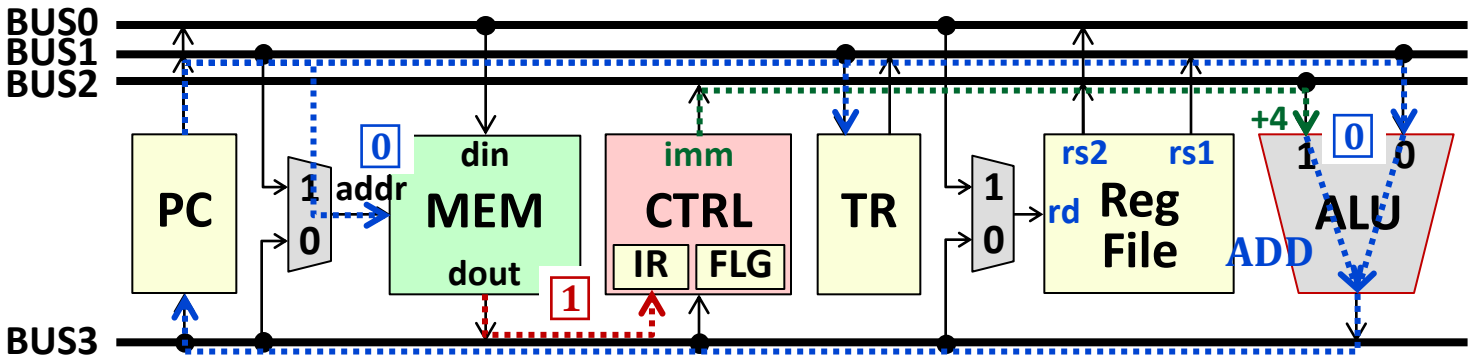
■ メモリアクセスなし：$I_{LD\_NOP}[4:0] = \{0,0,0,0,0\}, I_{ST\_NOP}[2:0] = \{0,0,0\}$

		31															15								7							0
LB	$SignExt\ (M_8[addr])$	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S							
LBU	$ZeroExt\ (M_8[addr])$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0							
LH	$SignExt\ (M_{16}[addr])$	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S	S															
LHU	$ZeroExt\ (M_{16}[addr])$	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0															
LW	$M_{32}[addr]$																															

資料概要

1. RV32-I命令セット仕様
2. RV32-Iハードウェアアーキテクチャ（ARCH-1）
3. 命令デコード動作
4. メモリ読出し･書込み動作
5. 命令フェッチサイクル
6. 命令実行サイクルのレジスタ転送記述
7. データパス制御論理設計
8. 算術論理演算器設計

命令フェッチサイクル



ARCH-1

$\boxed{0}\ MEM.addr \leftarrow PC, MEM.SetRead(I_{LD\_INST}), TR \leftarrow PC, PC \leftarrow PC + 4$

$\boxed{1}\ IR \leftarrow MEM.dout$

$I_{LD\_Inst}[4:0] = \{0,0,0,0,1\}$

$ALU(ALU_{in0}, ALU_{in1}, ALU_{CTRL})$
→ ALU動作の詳細記述

実行サイクル番号

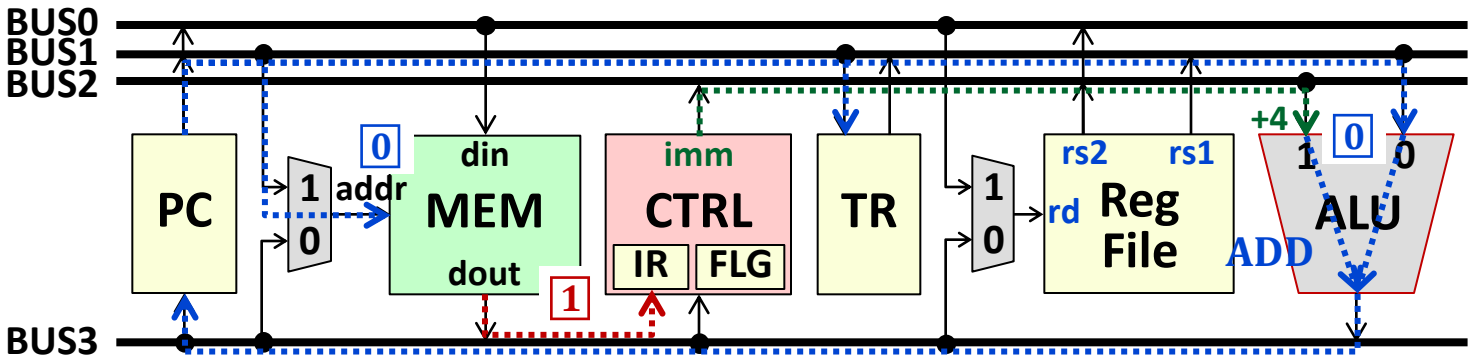
$\boxed{0}\ MEM.addr \leftarrow PC, MEM.SetRead(I_{LD\_INST}), TR \leftarrow PC, PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$

$\boxed{1}\ IR \leftarrow MEM.dout$

$I_{OP\_ADD}[13:0] = \{1,0,0,0,0,0,0,0,0,0,0,0,0,0\}$

命令フェッチ後：現命令アドレス→TR、次命令アドレス→PC += 4

命令フェッチサイクル



ARCH-1

$$PC \leftarrow PC + 4$$

$ALU(ALU_{in0}, ALU_{in1}, ALU_{CTRL})$
→ ALU動作の詳細記述

$$PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$$

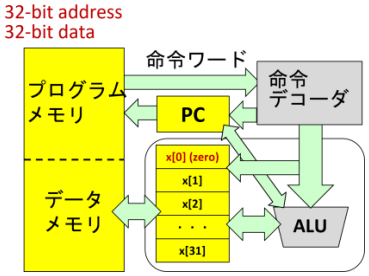
$ALU_{in0} \leftarrow PC, ALU_{in1} \leftarrow +4,$
$ALU_{CTRL} \leftarrow I_{OP\_ADD},$
$ALU_{out} \leftarrow PC + 4,$
$PC \leftarrow ALU_{out}$

ALUの入出力ポート間転送の詳細記述

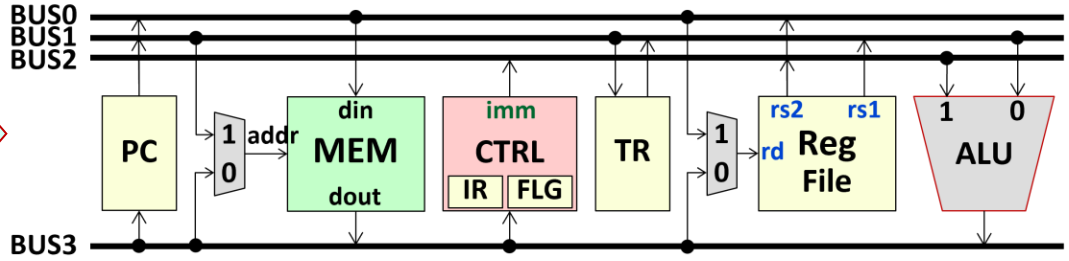
資料概要

1. RV32-I命令セット仕様
2. RV32-Iハードウェアアーキテクチャ（ARCH-1）
3. 命令デコード動作
4. メモリ読出し･書込み動作
5. 命令フェッチサイクル
6. 命令実行サイクルのレジスタ転送記述
7. データパス制御論理設計
8. 算術論理演算器設計

命令実行サイクルのレジスタ転送記述



RV32-I概要計算機構成



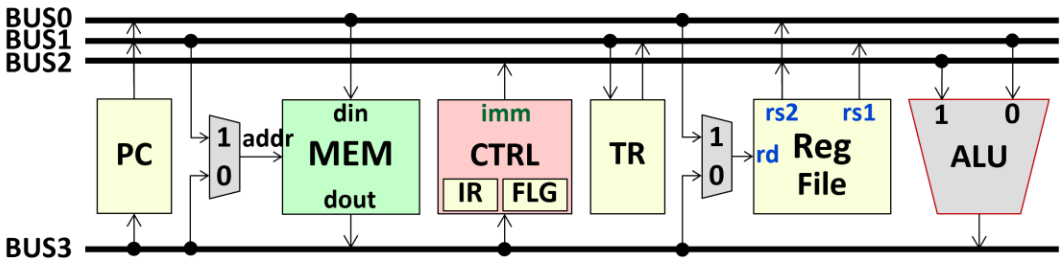
RV32-I詳細計算機構成（ARCH-1）

命令種別変数	レジスタ転送式（概要構成）	レジスタ転送式（詳細構成：ARCH-1）
【命令フェッチ】		0 $MEM.addr \leftarrow PC, MEM.SetRead(I_{LD\_INST}),$ $TR \leftarrow PC, PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$
		1 $IR \leftarrow MEM.dout$
I_{COMP}	$rd \leftarrow rs1\ (op)\ rs2$	2 $rd \leftarrow ALU(rs1, rs2, I_{OP\_type})$
I_{COMPI}	$rd \leftarrow rs1\ (op)\ imm$	2 $rd \leftarrow ALU(rs1, imm, I_{OP\_type})$
I_{LUI}	$rd \leftarrow imm$	2 $rd \leftarrow ALU(zero, imm, I_{OP\_ADD})$
I_{AUIPC}	$rd \leftarrow PC + imm$	2 $rd \leftarrow ALU(TR, imm, I_{OP\_ADD})$
I_{LD}	$rd \leftarrow Load(rs1 + imm, I_{LD\_type})$	2 $MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.SetRead(I_{LD\_type})$
		3 $rd \leftarrow MEM.dout$
I_{ST}	$Store(rs2, rs1 + imm, I_{ST\_type})$	2 $MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}),$ $MEM.din \leftarrow rs2, MEM.SetWrite(I_{ST\_type})$
I_{JAL}	$rd \leftarrow PC + 4, PC \leftarrow PC + imm$	2 $rd \leftarrow PC, PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$
I_{JALR}	$rd \leftarrow PC + 4, PC \leftarrow (rs1 + imm)\&(\sim 1)$	2 $rd \leftarrow PC, PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$
I_{BR}	$\text{if}(rs1\ (op)\ rs2)\ PC \leftarrow PC + imm$	2 $FLG \leftarrow ALU(rs1, rs2, I_{OP\_type})$
		3 $\text{if}(FLG)\ PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$

命令フェッチ後：
・現命令アドレス→ TR
・次命令アドレス→ PC

$(PC + 4)$の演算をどこで実行するか?→ 命令フェッチサイクル

命令実行サイクルのレジスタ転送記述

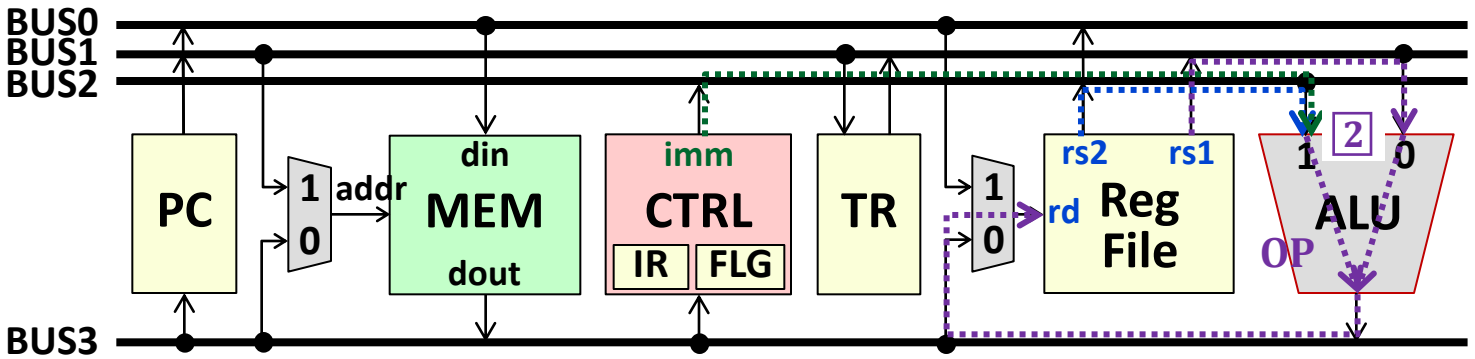


ARCH-1

src : 転送元
dst : 転送先

命令種別変数	レジスタ転送式（詳細構成 : ARCH-1）	BUS0 (src)	BUS1 (src)	BUS2 (src)	BUS3 (dst)
【命令フェッチ】	0 $MEM.addr \leftarrow PC, MEM.SetRead(I_{LD\_INST}), TR \leftarrow PC,$ $PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$		PC	+4	PC
	1 $IR \leftarrow MEM.dout$				IR
I_{COMP}	2 $rd \leftarrow ALU(rs1, rs2, I_{OP\_type})$		rs1	rs2	rd
I_{COMPI}	2 $rd \leftarrow ALU(rs1, imm, I_{OP\_type})$		rs1	imm	rd
I_{LUI}	2 $rd \leftarrow ALU(zero, imm, I_{OP\_ADD})$		zero	imm	rd
I_{AUIPC}	2 $rd \leftarrow ALU(TR, imm, I_{OP\_ADD})$		TR	imm	rd
I_{LD}	2 $MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.SetRead(I_{LD\_type})$		rs1	imm	addr
	3 $rd \leftarrow MEM.dout$				rd
I_{ST}	2 $MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.din \leftarrow rs2,$ $MEM.SetWrite(I_{ST\_type})$	rs2	rs1	imm	addr
I_{JAL}	2 $rd \leftarrow PC, PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$	PC	TR	imm	PC
I_{JALR}	2 $rd \leftarrow PC, PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	PC	rs1	imm	PC
I_{BR}	2 $FLG \leftarrow ALU(rs1, rs2, I_{OP\_type})$		rs1	rs2	FLG
	3 $\mathrm{if}(FLG)\ PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$		TR	imm	PC

演算命令実行サイクル



OP rd, rs1, rs2 OP rd, rs1, imm	rd ← rs1 OP rs2 rd ← rs1 OP simm[11:0]	算術・論理・シフト演算

$$I_{OP\_type}[13:0] = \{I_{ADD}, I_{SUB}, I_{LT}, I_{LTU}, I_{AND}, I_{OR}, I_{XOR}, I_{SLL}, I_{SRL}, I_{SRA}, I_{EQ}, I_{NE}, I_{GE}, I_{GEU}\}$$

OP rd, rs1, rs2 :

2 $rd \leftarrow ALU(rs1, rs2, I_{OP\_type})$

→ 0 命令フェッチへ戻る

$$ALU_{in0} \leftarrow rs1, ALU_{in1} \leftarrow rs2,$$

OP rd, rs1, imm :

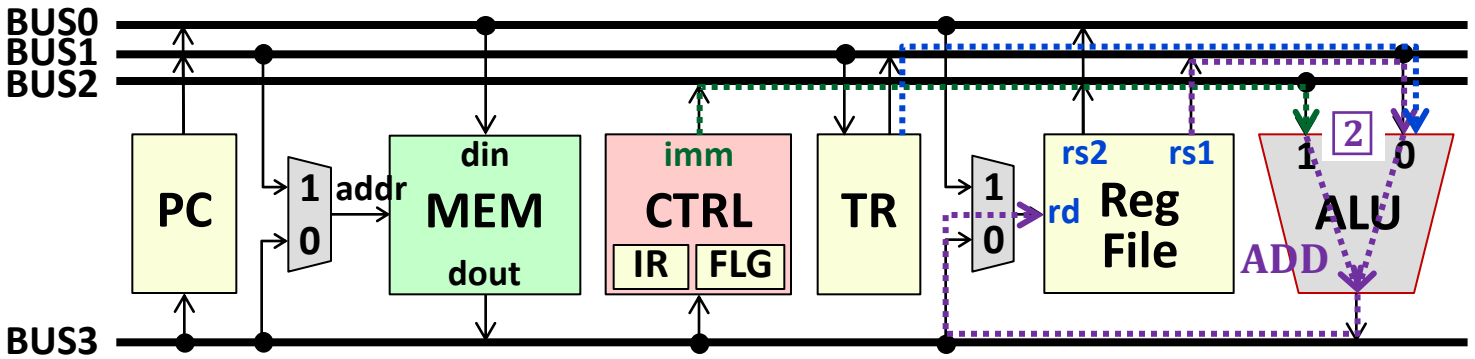
2 $rd \leftarrow ALU(rs1, imm, I_{OP\_type})$

→ 0 命令フェッチへ戻る

$$ALU_{in0} \leftarrow rs1, ALU_{in1} \leftarrow imm,$$

$$ALU_{CTRL} \leftarrow I_{OP\_type},$$
$$ALU_{out} \leftarrow ALU_{in0}\ (OP)\ ALU_{in1},$$
$$rd \leftarrow ALU_{out}$$

LUI/AUIPC命令実行サイクル



LUI rd, uimm AUIPC rd, uimm	rd ← uimm[31:12] rd ← TR + uimm[31:12]	即値ロード

LUI rd, uimm :

2 $rd \leftarrow ALU(zero, imm, I_{OP\_ADD})$

→ 0 命令フェッチへ戻る

$zero = x[0]$ (0値)

$ALU_{in0} \leftarrow zero, ALU_{in1} \leftarrow imm,$

AUIPC rd, uimm :

2 $rd \leftarrow ALU(TR, imm, I_{OP\_ADD})$

→ 0 命令フェッチへ戻る

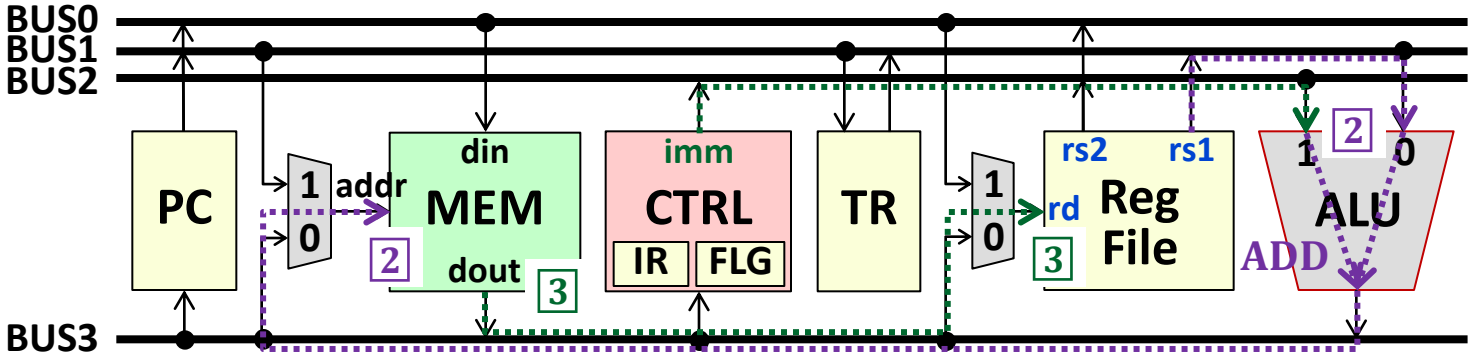
$ALU_{in0} \leftarrow TR, ALU_{in1} \leftarrow imm,$

$ALU_{CTRL} \leftarrow I_{OP\_ADD},$

$ALU_{out} \leftarrow ALU_{in0} + ALU_{in1},$

$rd \leftarrow ALU_{out}$

ロード命令実行サイクル



LD rd, imm(rs1)	rd ← M[rs1+ simm[11:0]]	ロード命令

2 $MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.SetRead(I_{LD\_type})$

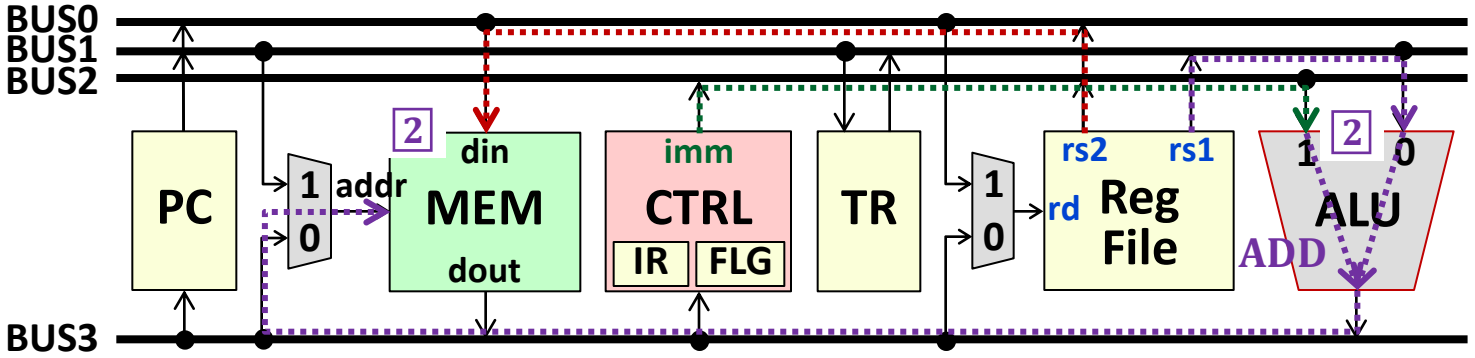
3 $rd \leftarrow MEM.dout$

→ 0 命令フェッチへ戻る

$I_{LD\_type}[4:0] = \{I_{LB}, I_{LBU}, I_{LH}, I_{LHU}, I_{LW}\}$

LB rd, simm (rs1)	$rd \leftarrow SignExt\ (M_8[rs1 + simm[11:0]])$	s s s s s s s s s s s s s s s s s s s s s s s s LB
LBU rd, simm (rs1)	$rd \leftarrow ZeroExt(M_8[rs1 + simm[11:0]])$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 LBU
LH rd, simm (rs1)	$rd \leftarrow SignExt(M_{16}[rs1 + simm[11:0]])$	s s s s s s s s s s s s s s s s LH
LHU rd, simm (rs1)	$rd \leftarrow ZeroExt(M_{16}[rs1 + simm[11:0]])$	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 LHU
LW rd, simm (rs1)	$rd \leftarrow M_{32}[rs1 + simm[11:0]]$	LW

ストア命令実行サイクル



ST sr2, imm(sr1)	M[rs1+ simm[11:0]] ← rs2	ストア命令

2 $MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.din \leftarrow rs2,$
$MEM.SetWrite(I_{ST\_type})$

→ 0 命令フェッチへ戻る

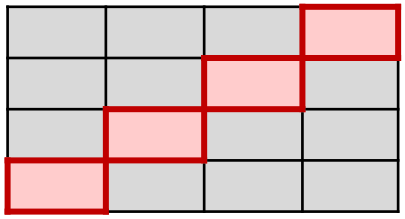
$I_{ST\_type}[2:0] = \{I_{SB}, I_{SH}, I_{SW}\}$

SB rs2, simm (rs1)	$M_8[rs1 + simm[11:0]] \leftarrow rs2[7:0]$
SH rs2, simm (rs1)	$M_{16}[rs1 + simm[11:0]] \leftarrow rs2[15:0]$
SW rs2, simm (rs1)	$M_{32}[rs1 + simm[11:0]] \leftarrow rs2[31:0]$

WE[3:0](write enable)はバイト単位で書込み許可を制御をする

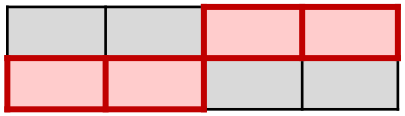
byte address : 3 2 1 0

SB 0x80018 WE=0001
SB 0x80019 WE=0010
SB 0x8001a WE=0100
SB 0x8001b WE=1000



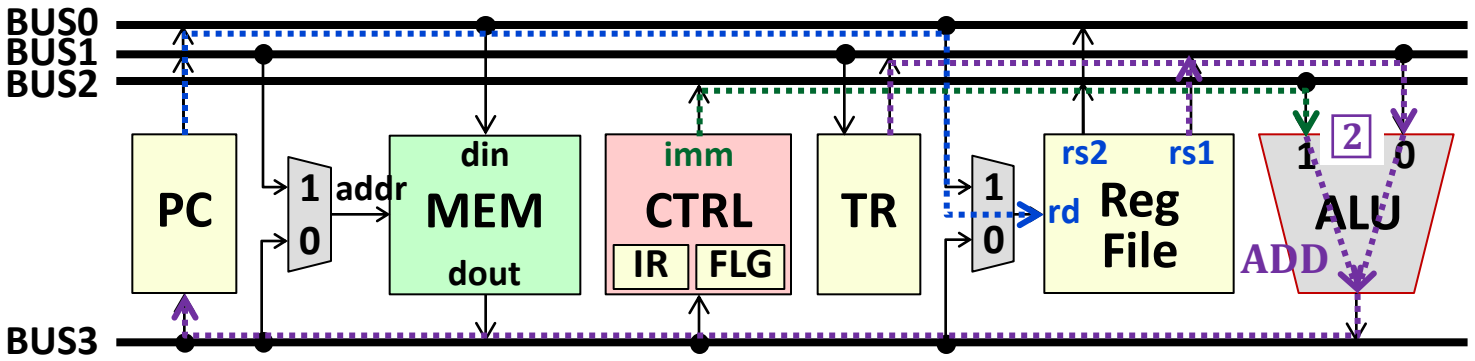
byte address : 3 2 1 0

SH 0x80018 WE=0011
SH 0x8001a WE=1100



SW 0x80018 WE=1111

JAL/JALR命令実行サイクル



JAL rd, imm JALR rd, imm(rs1)	rd ← PC, PC ← (TR + simm[20:1]) rd ← PC, PC ← (rs1 + simm[11:0])&(~1)	Jump and Link命令

JAL rd, imm :

2 $rd \leftarrow PC, PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$

→ 0 命令フェッチへ戻る

$ALU_{in0} \leftarrow TR, ALU_{in1} \leftarrow imm,$
$ALU_{CTRL} \leftarrow I_{OP\_ADD},$
$ALU_{out} \leftarrow TR + imm,$
$MEM.addr \leftarrow ALU_{out}$

最下位ビットを0にクリア $((\sim 1) = 0xfffffffe)$
→ PC入力制御で実装する（次回講義）

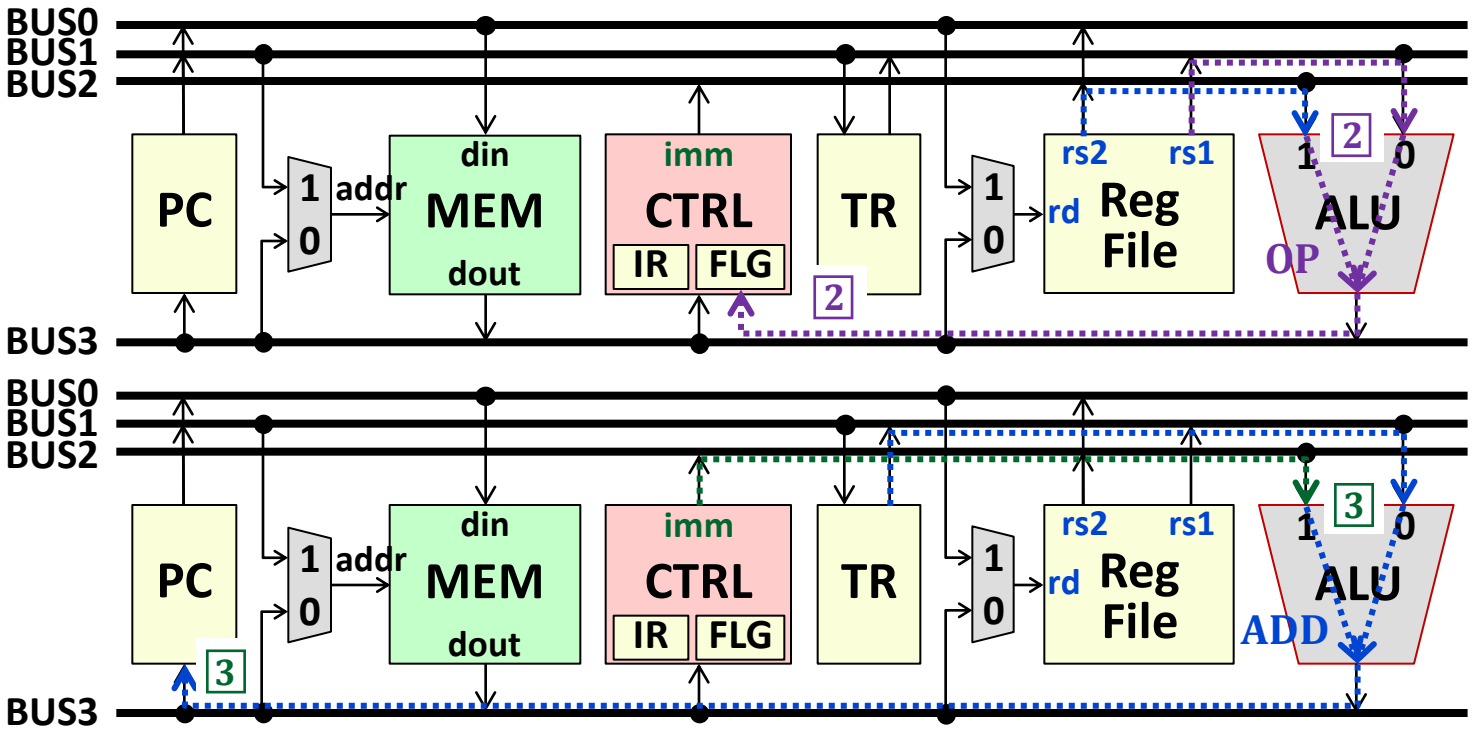
JALR rd, imm(rs1) :

2 $rd \leftarrow PC, PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$

→ 0 命令フェッチへ戻る

$ALU_{in0} \leftarrow rs1, ALU_{in1} \leftarrow imm,$
$ALU_{CTRL} \leftarrow I_{OP\_ADD},$
$ALU_{out} \leftarrow rs1 + imm,$
$MEM.addr \leftarrow ALU_{out}$

条件分岐命令実行サイクル



BR rs1, rs2, imm	if (rs1 *OP* rs2) PC ← PC + simm[11:0]	条件分岐命令

2 $FLG \leftarrow ALU(rs1, rs2, I_{OP\_type})$

3 if(FLG) $PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$

→ 0 命令フェッチへ戻る

$$I_{OP\_type}[13:0] = \{I_{ADD}, I_{SUB}, I_{LT}, I_{LTU}, I_{AND}, I_{OR}, I_{XOR}, I_{SLL}, I_{SRL}, I_{SRA}, I_{EQ}, I_{NE}, I_{GE}, I_{GEU}\}$$

条件分岐命令は、
比較演算と加算を2段階で実行

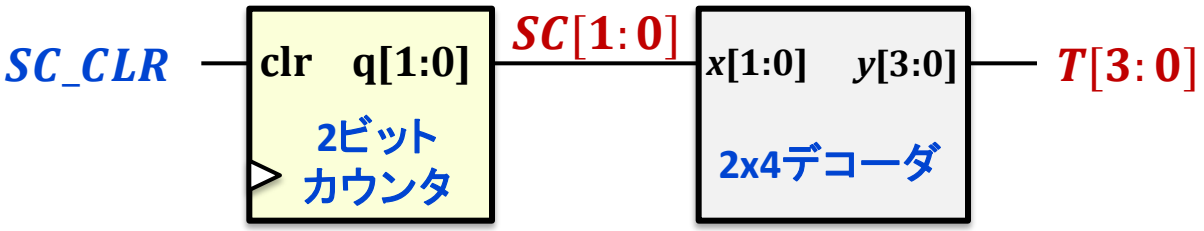
命令実行状態表現

一般的な順序回路の状態遷移よりも簡単な遷移構造のため、順序回路で一般に使われる「状態変数」ではなく、「タイミング情報」と「命令情報」を表す論理変数の「組」として命令実行状態を表現し、論理回路設計を単純化することを考える

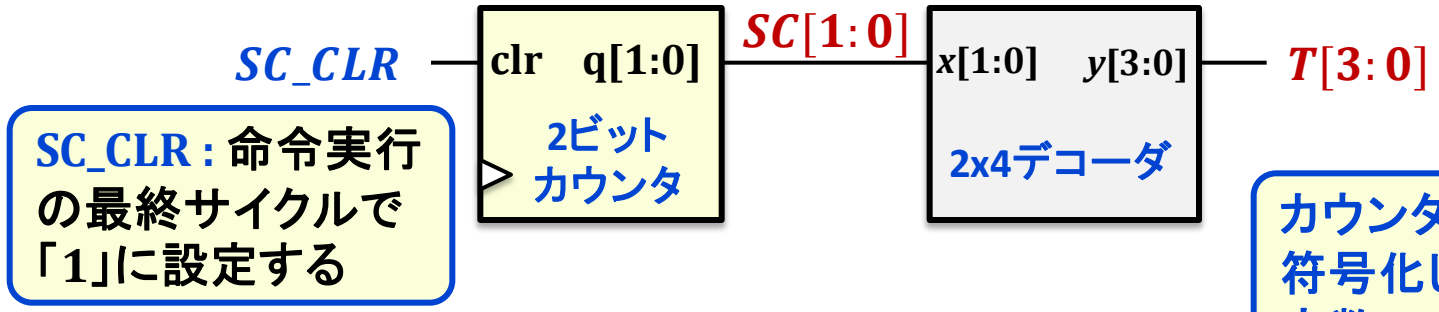
- **タイミング情報**：命令フェッチサイクル開始時からのクロック数
  - 命令フェッチサイクル：0, 1サイクル
  - 命令実行サイクル：2, 3サイクル

→ タイミング生成回路

- **命令情報**：命令種別変数
  - $I_{COMP}, I_{COMPI}, I_{LUI}, I_{AUIPC}, I_{LD}, I_{ST}, I_{JAL}, I_{JALR}, I_{BR}$

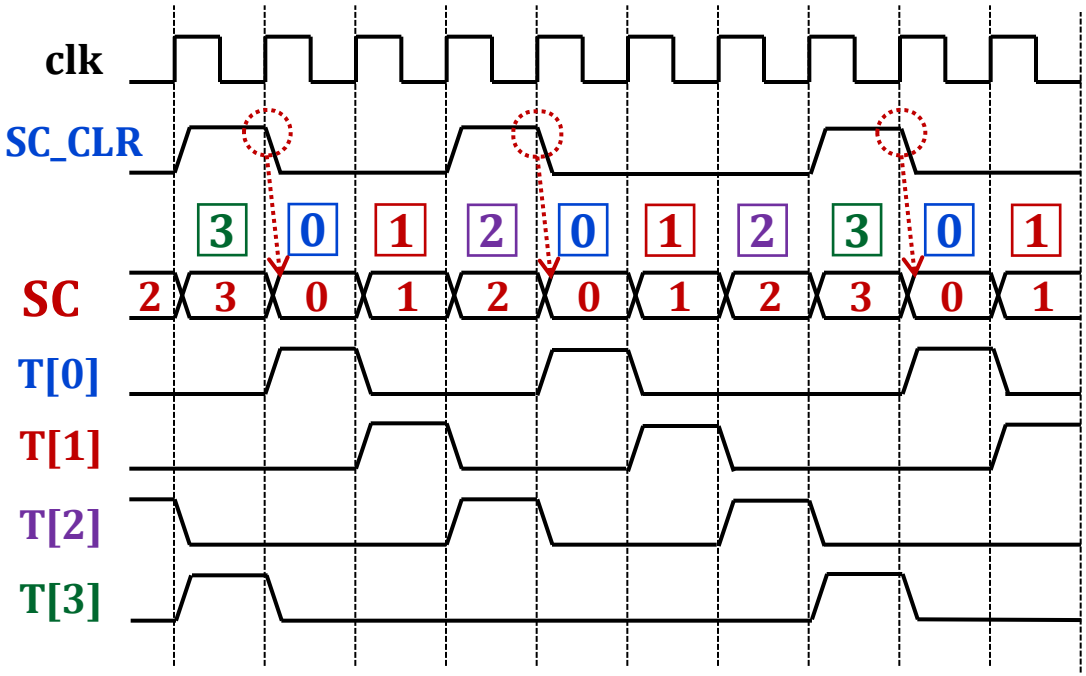


タイミング情報：2ビットカウンタ → 2x4デコーダ



SC_CLR：命令実行の最終サイクルで「1」に設定する

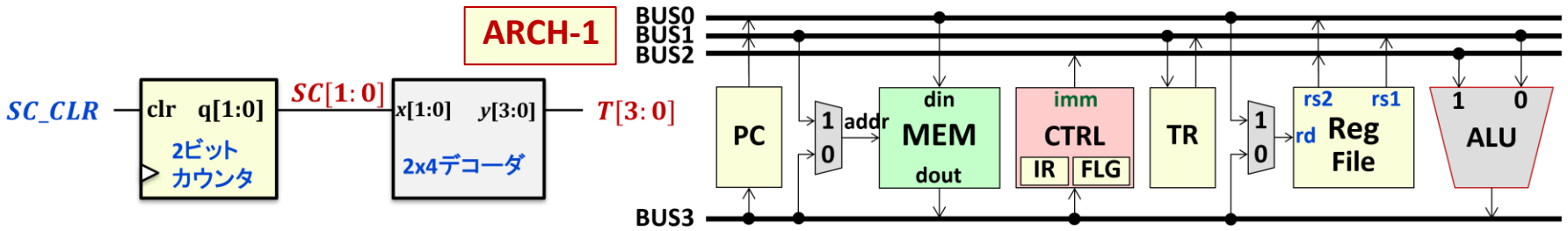
カウンタ値 SC[1:0] を符号化し4つの論理変数 T[3:0] に変換



入力 SC[1:0]	出力 T[3:0]	2x4デコーダ論理関数
00	0001	$T[0] = \overline{SC[1]} \cdot \overline{SC[0]}$
01	0010	$T[1] = \overline{SC[1]} \cdot SC[0]$
10	0100	$T[2] = SC[1] \cdot \overline{SC[0]}$
11	1000	$T[3] = SC[1] \cdot SC[0]$

One-Hot符号：「1」になるビットが高々1つ

命令実行サイクルのレジスタ転送記述



条件式	レジスタ転送式	BUS0 (src)	BUS1 (src)	BUS2 (src)	BUS3 (dst)
命令フェッチ $T[0]$:	$MEM.addr \leftarrow PC, MEM.SetRead(I_{LD\_INST}), TR \leftarrow PC,$ $PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$		PC	+4	PC
命令フェッチ $T[1]$:	$IR \leftarrow MEM.dout$				IR
$T[2] \cdot I_{COMP}$:	$rd \leftarrow ALU(rs1, rs2, I_{OP\_type})$		rs1	rs2	rd
$T[2] \cdot I_{COMPI}$:	$rd \leftarrow ALU(rs1, imm, I_{OP\_type})$		rs1	imm	rd
$T[2] \cdot I_{LUI}$:	$rd \leftarrow ALU(zero, imm, I_{OP\_ADD})$		zero	imm	rd
$T[2] \cdot I_{AUIPC}$:	$rd \leftarrow ALU(TR, imm, I_{OP\_ADD})$		TR	imm	rd
$T[2] \cdot I_{LD}$:	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.SetRead(I_{LD\_type})$		rs1	imm	addr
$T[3] \cdot I_{LD}$:	$rd \leftarrow MEM.dout$				rd
$T[2] \cdot I_{ST}$:	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.din \leftarrow rs2,$ $MEM.SetWrite(I_{ST\_type})$	rs2	rs1	imm	addr
$T[2] \cdot I_{JAL}$:	$rd \leftarrow PC, PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$	PC	TR	imm	PC
$T[2] \cdot I_{JALR}$:	$rd \leftarrow PC, PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	PC	rs1	imm	PC
$T[2] \cdot I_{BR}$:	$FLG \leftarrow ALU(rs1, rs2, I_{OP\_type})$		rs1	rs2	FLG
$T[3] \cdot I_{BR}$:	$\text{if}(FLG)\ PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$		TR	imm	PC
$T[2] \cdot \overline{I_{LD\_BR}}$:	$SC \leftarrow 0$				
$T[3] \cdot I_{LD\_BR}$:	$SC \leftarrow 0$				

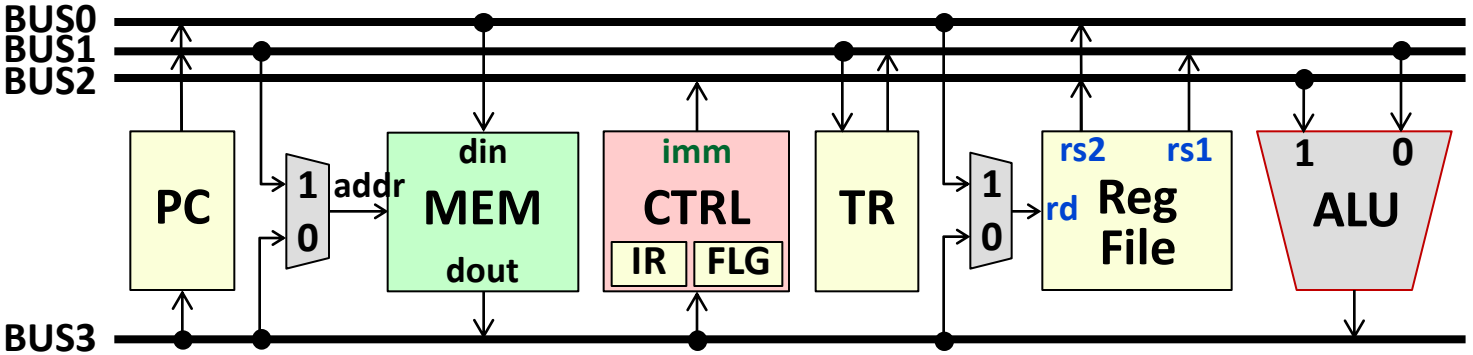
src : 転送元
dst : 転送先

$I_{LD\_BR} = I_{LD} + I_{BR}$ → 2サイクル命令（他の命令は1サイクル）

資料概要

1. RV32-I命令セット仕様
2. RV32-Iハードウェアアーキテクチャ（ARCH-1）
3. 命令デコード動作
4. メモリ読出し･書込み動作
5. 命令フェッチサイクル
6. 命令実行サイクルのレジスタ転送記述
7. データパス制御論理設計
8. 算術論理演算器設計

データパス制御論理設計



→ 下記の制御信号を「タイミング情報」と「命令情報」で表現

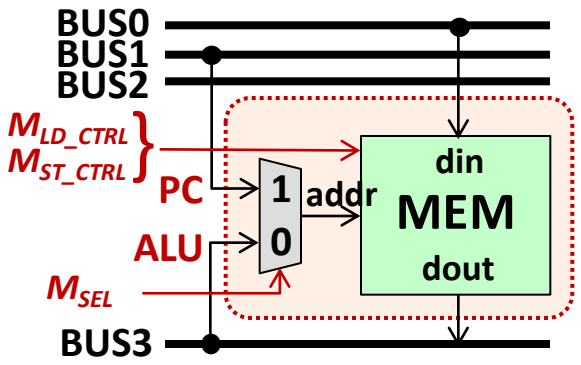
- MEM_{CTRL} : 読出し・書込み制御、addr入力選択（BUS1, BUS3）
- ALU_{CTRL} : 演算制御
- $RegFile_{CTRL}$: 読出し制御（rs1, rs2）、書込み制御（rd : BUS0, BUS3）
- $BUS0_{CTRL}$, $BUS1_{CTRL}$, $BUS2_{CTRL}$, $BUS3_{CTRL}$: バス選択制御
- PC_{LD} : 書込み制御
- TR_{LD} : 書込み制御
- IR_{LD} , FLG_{LD} : 書込み制御
- $CTRL_{imm}$: 即値データ出力

データパス制御論理設計 (1) : メモリ制御論理

- $MEM.SetRead(M_{LD\_CTRL})$
- $MEM.SetWrite(M_{ST\_CTRL})$

$MEM_{CTRL}[8:0] = \{M_{LD\_CTRL}[4:0], M_{ST\_CTRL}[2:0], M_{SEL}\}$

- $M_{LD\_CTRL}[4:0]$: ロード種別変数
- $M_{ST\_CTRL}[2:0]$: ストア種別変数
- M_{SEL} : addr入力のバス選択（BUS1, BUS3）



条件式	レジスタ転送式	M_{LD_CTRL}	M_{ST_CTRL}	M_{SEL}
$T[0]$:	$MEM.addr \leftarrow PC, MEM.SetRead(I_{LD\_INST}), TR \leftarrow PC,$ $PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$	I_{LD_INST}	I_{ST_NOP}	1
$T[2] \cdot I_{LD}$:	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.SetRead(I_{LD\_type})$	I_{LD_type}	I_{ST_NOP}	0
$T[2] \cdot I_{ST}$:	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.din \leftarrow rs2,$ $MEM.SetWrite(I_{ST\_type})$	I_{LD_NOP}	I_{ST_type}	0
（上記以外）:		I_{LD_NOP}	I_{ST_NOP}	*

$I_{LD\_type}[4:0] = \{I_{LB}, I_{LBU}, I_{LH}, I_{LHU}, I_{LW}\}$
$I_{LD\_INST}[4:0] = \{0,0,0,0,1\}$（命令フェッチ）
$I_{LD\_NOP}[4:0] = \{0,0,0,0,0\}$
$I_{ST\_type}[2:0] = \{I_{SB}, I_{SH}, I_{SW}\}$
$I_{ST\_NOP}[2:0] = \{0,0,0\}$

One-Hot符号：「1」になるビットが高々1つ

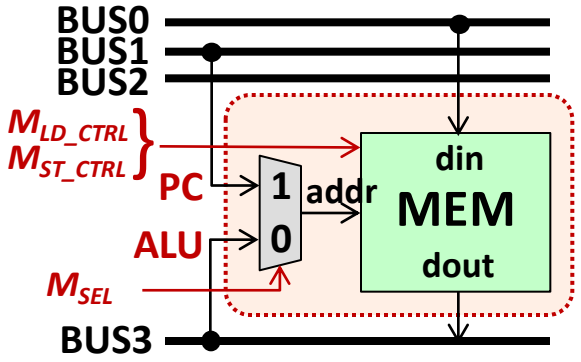
$I_{LB} = I_{LD} \cdot I_{F3\_0}$
$I_{LBU} = I_{LD} \cdot I_{F3\_4}$
$I_{LH} = I_{LD} \cdot I_{F3\_1}$
$I_{LHU} = I_{LD} \cdot I_{F3\_5}$
$I_{LW} = I_{LD} \cdot I_{F3\_2}$
$I_{SB} = I_{ST} \cdot I_{F3\_0}$
$I_{SH} = I_{ST} \cdot I_{F3\_1}$
$I_{SW} = I_{ST} \cdot I_{F3\_2}$

ドントケア

メモリ読出し制御論理

■ $M_{LD\_CTRL}[4:0]$: ロード種別変数

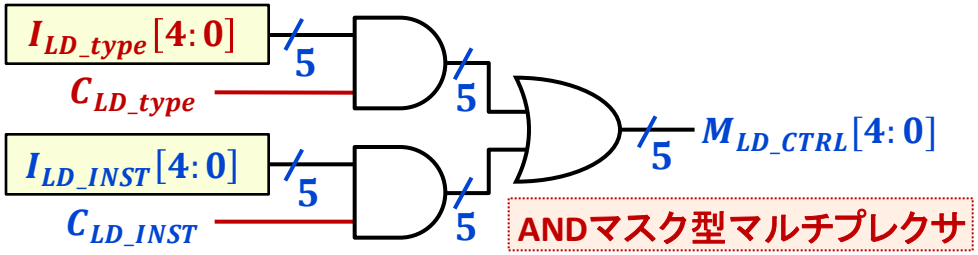
条件式	レジスタ転送式	M_{LD_CTRL}
$C_{LD\_INST} = T[0]$:	$MEM.SetRead(I_{LD\_INST})$	I_{LD_INST}
$C_{LD\_type} = T[2] \cdot I_{LD}$:	$MEM.SetRead(I_{LD\_type})$	I_{LD_type}
(上記以外):		I_{LD_NOP}

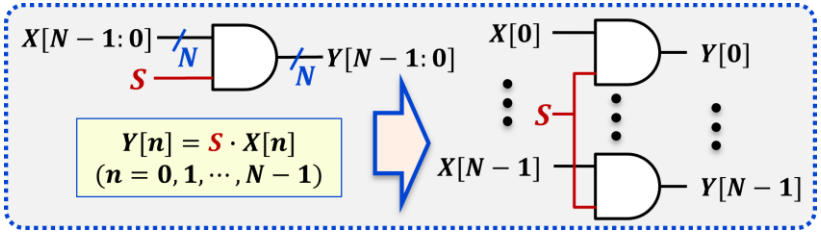


条件演算子(3項演算子):
a = (b) ? c : d; → if (b != 0) a = c; else a = d;

$$A_{LD\_type} = C_{LD\_type} \; ? \; I_{LD\_type} : 5'b0 ;$$
$$A_{LD\_INST} = C_{LD\_INST} \; ? \; I_{LD\_INST} : 5'b0 ;$$
$$M_{LD\_CTRL} = A_{LD\_Type} \mid A_{LD\_INST} ;$$

$$I_{LD\_type}[4:0] = \{I_{LB}, I_{LBU}, I_{LH}, I_{LHU}, I_{LW}\}$$
$$I_{LD\_INST}[4:0] = \{0,0,0,0,1\}$$
$$I_{LD\_NOP}[4:0] = \{0,0,0,0,0\}$$





LB	$M_{LD\_CTRL}[4] = C_{LD\_type} \cdot I_{LB}$
LBU	$M_{LD\_CTRL}[3] = C_{LD\_type} \cdot I_{LBU}$
LH	$M_{LD\_CTRL}[2] = C_{LD\_type} \cdot I_{LH}$
LHU	$M_{LD\_CTRL}[1] = C_{LD\_type} \cdot I_{LHU}$
LW	$M_{LD\_CTRL}[0] = C_{LD\_type} \cdot I_{LW} + C_{LD\_INST}$

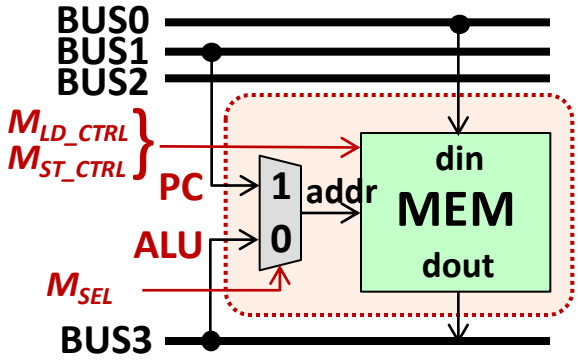
冗長項の削除

LB	$M_{LD\_CTRL}[4] = T[2] \cdot I_{LB}$
LBU	$M_{LD\_CTRL}[3] = T[2] \cdot I_{LBU}$
LH	$M_{LD\_CTRL}[2] = T[2] \cdot I_{LH}$
LHU	$M_{LD\_CTRL}[1] = T[2] \cdot I_{LHU}$
LW	$M_{LD\_CTRL}[0] = T[2] \cdot I_{LW} + T[0]$

メモリ書込み制御論理

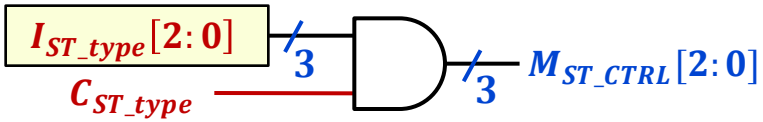
■ $M_{ST\_CTRL}[2:0]$：ストア種別変数

条件式	レジスタ転送式	M_{ST_CTRL}
$C_{ST\_type} = T[2] \cdot I_{ST}$：	$MEM.SetWrite(I_{ST\_type})$	I_{ST_type}
（上記以外）:		I_{ST_NOP}



$M_{ST\_CTRL} = C_{ST\_type}\ ?\ I_{ST\_type} : 5'b0;$

$I_{ST\_type}[2:0] = \{I_{SB}, I_{SH}, I_{SW}\}$
$I_{ST\_NOP}[2:0] = \{0, 0, 0\}$



SB	$M_{ST\_CTRL}[2] = C_{ST\_type} \cdot I_{SB}$
SH	$M_{ST\_CTRL}[1] = C_{ST\_type} \cdot I_{SH}$
SW	$M_{ST\_CTRL}[0] = C_{ST\_type} \cdot I_{SW}$

冗長項の削除

証明せよ！

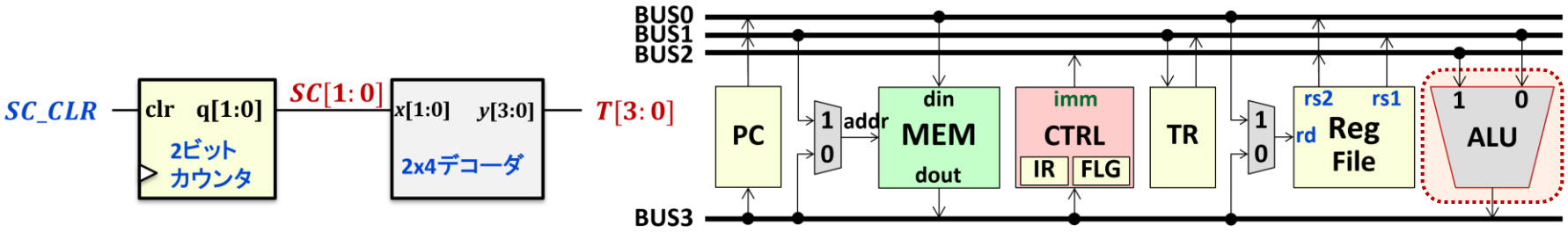
SB	$M_{ST\_CTRL}[2] = T[2] \cdot I_{SB}$
SH	$M_{ST\_CTRL}[1] = T[2] \cdot I_{SH}$
SW	$M_{ST\_CTRL}[0] = T[2] \cdot I_{SW}$

■ M_{SEL}：addr入力のバス選択（BUS1, BUS3）

条件式	レジスタ転送式	M_{SEL}
$T[0]$：	$MEM.addr \leftarrow PC$	1
$T[2] \cdot I_{LD}$：	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	0
$T[2] \cdot I_{ST}$：	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	0
（上記以外）:		*

$M_{SEL} = T[0]$

データパス制御論理設計 (2) : ALU制御論理



条件式	レジスタ転送式	BUS0 (src)	BUS1 (src)	BUS2 (src)	BUS3 (dst)
$T[0]:$	$MEM.addr \leftarrow PC, MEM.SetRead(I_{LD\_INST}), TR \leftarrow PC,$ $PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$		PC	+4	PC
$T[2] \cdot I_{COMP}:$	$rd \leftarrow ALU(rs1, rs2, I_{OP\_type})$		rs1	rs2	rd
$T[2] \cdot I_{COMPI}:$	$rd \leftarrow ALU(rs1, imm, I_{OP\_type})$		rs1	imm	rd
$T[2] \cdot I_{LUI}:$	$rd \leftarrow ALU(zero, imm, I_{OP\_ADD})$		zero	imm	rd
$T[2] \cdot I_{AUIPC}:$	$rd \leftarrow ALU(TR, imm, I_{OP\_ADD})$		TR	imm	rd
$T[2] \cdot I_{LD}:$	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.SetRead(I_{LD\_type})$		rs1	imm	addr
$T[2] \cdot I_{ST}:$	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.din \leftarrow rs2,$ $MEM.SetWrite(I_{ST\_type})$	rs2	rs1	imm	addr
$T[2] \cdot I_{JAL}:$	$rd \leftarrow PC, PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$	PC	TR	imm	PC
$T[2] \cdot I_{JALR}:$	$rd \leftarrow PC, PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	PC	rs1	imm	PC
$T[2] \cdot I_{BR}:$	$FLG \leftarrow ALU(rs1, rs2, I_{OP\_type})$		rs1	rs2	FLG
$T[3] \cdot I_{BR}:$	$\text{if}(FLG)\ PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$		TR	imm	PC

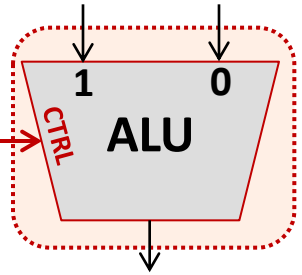
$I_{OP\_type}[13:0] = \{I_{ADD}, I_{SUB}, I_{LT}, I_{LTU}, I_{AND}, I_{OR}, I_{XOR}, I_{SLL}, I_{SRL}, I_{SRA}, I_{EQ}, I_{NE}, I_{GE}, I_{GEU}\}$

$I_{OP\_ADD}[13:0] = \{1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0\}$

ALU制御論理

条件式	レジスタ転送式	ALU_{CTRL}
$T[0]:$	$PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$	I_{OP_ADD}
$T[2] \cdot I_{COMP}:$	$rd \leftarrow ALU(rs1, rs2, I_{OP\_type})$	I_{OP_type}
$T[2] \cdot I_{COMPI}:$	$rd \leftarrow ALU(rs1, imm, I_{OP\_type})$	I_{OP_type}
$T[2] \cdot I_{LUI}:$	$rd \leftarrow ALU(zero, imm, I_{OP\_ADD})$	I_{OP_ADD}
$T[2] \cdot I_{AUIPC}:$	$rd \leftarrow ALU(TR, imm, I_{OP\_ADD})$	I_{OP_ADD}
$T[2] \cdot I_{LD}:$	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	I_{OP_ADD}
$T[2] \cdot I_{ST}:$	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	I_{OP_ADD}
$T[2] \cdot I_{JAL}:$	$PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$	I_{OP_ADD}
$T[2] \cdot I_{JALR}:$	$PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	I_{OP_ADD}
$T[2] \cdot I_{BR}:$	$FLG \leftarrow ALU(rs1, rs2, I_{OP\_type})$	I_{OP_type}
$T[3] \cdot I_{BR}:$	$\text{if}(FLG)\ PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$	I_{OP_ADD}

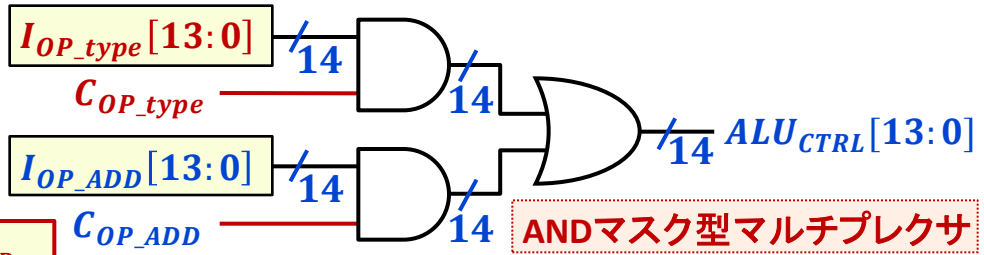
$ALU(ALU_{in0}, ALU_{in1}, ALU_{CTRL})$



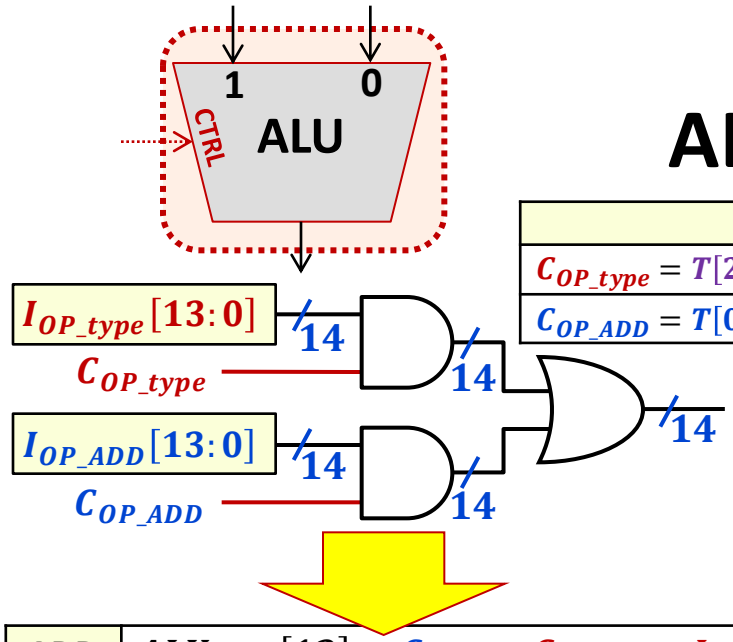
制御入力条件	ALU_{CTRL}
$C_{OP\_type} = T[2] \cdot (I_{COMP} + I_{COMPI} + I_{BR})$	I_{OP_type}
$C_{OP\_ADD} = T[0] + T[2] \cdot (I_{LUI} + I_{AUIPC} + I_{LD} + I_{ST} + I_{JAL} + I_{JALR}) + T[3] \cdot I_{BR}$	I_{OP_ADD}

$A_{OP\_type} = C_{OP\_type}\ ?\ I_{OP\_type} : 14'b0;$
$A_{OP\_ADD} = C_{OP\_ADD}\ ?\ I_{OP\_ADD} : 14'b0;$
$ALU_{CTRL} = A_{OP\_type} \mid A_{OP\_ADD};$

$ALU_{CTRL}[13:0] = \{C_{ADD}, C_{SUB}, C_{LT}, C_{LTU}, C_{AND}, C_{OR}, C_{XOR}, C_{SLL}, C_{SRL}, C_{SRA}, C_{EQ}, C_{NE}, C_{GE}, C_{GEU}\}$



ALU制御論理



制御入力転送条件	ALU_{CTRL}
$C_{OP\_type} = T[2] \cdot (I_{COMP} + I_{COMPI} + I_{BR})$	I_{OP_type}
$C_{OP\_ADD} = T[0] + T[2] \cdot (I_{LUI} + I_{AUIPC} + I_{LD} + I_{ST} + I_{JAL} + I_{JALR}) + T[3] \cdot I_{BR}$	I_{OP_ADD}

$$ALU_{CTRL}[13:0] = \{C_{ADD}, C_{SUB}, C_{LT}, C_{LTU}, C_{AND}, C_{OR}, C_{XOR}, C_{SLL}, C_{SRL}, C_{SRA}, C_{EQ}, C_{NE}, C_{GE}, C_{GEU}\}$$

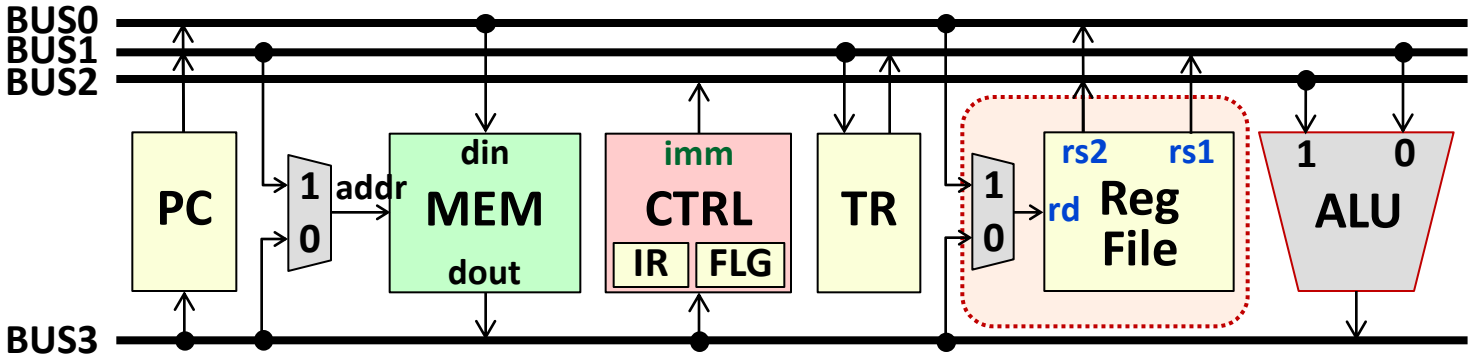
$$I_{OP\_type}[13:0] = \{I_{ADD}, I_{SUB}, I_{LT}, I_{LTU}, I_{AND}, I_{OR}, I_{XOR}, I_{SLL}, I_{SRL}, I_{SRA}, I_{EQ}, I_{NE}, I_{GE}, I_{GEU}\}$$

ADD	$ALU_{CTRL}[13] = C_{ADD} = C_{OP\_type} \cdot I_{ADD} + C_{OP\_ADD}$
SUB	$ALU_{CTRL}[12] = C_{SUB} = C_{OP\_type} \cdot I_{SUB}$
LT	$ALU_{CTRL}[11] = C_{LT} = C_{OP\_type} \cdot I_{LT}$
LTU	$ALU_{CTRL}[10] = C_{LTU} = C_{OP\_type} \cdot I_{LTU}$
AND	$ALU_{CTRL}[9] = C_{AND} = C_{OP\_type} \cdot I_{AND}$
OR	$ALU_{CTRL}[8] = C_{OR} = C_{OP\_type} \cdot I_{OR}$
XOR	$ALU_{CTRL}[7] = C_{XOR} = C_{OP\_type} \cdot I_{XOR}$
SLL	$ALU_{CTRL}[6] = C_{SLL} = C_{OP\_type} \cdot I_{SLL}$
SRL	$ALU_{CTRL}[5] = C_{SRL} = C_{OP\_type} \cdot I_{SRL}$
SRA	$ALU_{CTRL}[4] = C_{SRA} = C_{OP\_type} \cdot I_{SRA}$
EQ	$ALU_{CTRL}[3] = C_{EQ} = C_{OP\_type} \cdot I_{EQ}$
NE	$ALU_{CTRL}[2] = C_{NE} = C_{OP\_type} \cdot I_{NE}$
GE	$ALU_{CTRL}[1] = C_{GE} = C_{OP\_type} \cdot I_{GE}$
GEU	$ALU_{CTRL}[0] = C_{GEU} = C_{OP\_type} \cdot I_{GEU}$

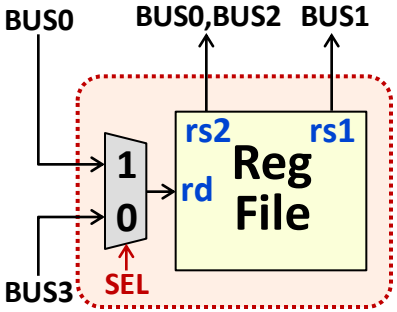
冗長項の削除

$ALU_{CTRL}[13] = C_{ADD} = T[2] \cdot I_{ADD} + C_{OP\_ADD}$
$ALU_{CTRL}[12] = C_{SUB} = T[2] \cdot I_{SUB}$
$ALU_{CTRL}[11] = C_{LT} = T[2] \cdot I_{LT}$
$ALU_{CTRL}[10] = C_{LTU} = T[2] \cdot I_{LTU}$
$ALU_{CTRL}[9] = C_{AND} = T[2] \cdot I_{AND}$
$ALU_{CTRL}[8] = C_{OR} = T[2] \cdot I_{OR}$
$ALU_{CTRL}[7] = C_{XOR} = T[2] \cdot I_{XOR}$
$ALU_{CTRL}[6] = C_{SLL} = T[2] \cdot I_{SLL}$
$ALU_{CTRL}[5] = C_{SRL} = T[2] \cdot I_{SRL}$
$ALU_{CTRL}[4] = C_{SRA} = T[2] \cdot I_{SRA}$
$ALU_{CTRL}[3] = C_{EQ} = T[2] \cdot I_{EQ}$
$ALU_{CTRL}[2] = C_{NE} = T[2] \cdot I_{NE}$
$ALU_{CTRL}[1] = C_{GE} = T[2] \cdot I_{GE}$
$ALU_{CTRL}[0] = C_{GEU} = T[2] \cdot I_{GEU}$

データパス制御論理設計 (3) : レジスタファイル制御論理



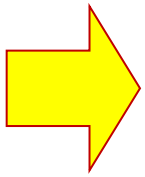
- $RegFile_{CTRL}[15:0] = \{RF_{rs1}[4:0], RF_{rs2}[4:0], RF_{rd}[4:0], RF_{SEL}\}$
  - $RF_{rs1}[4:0]$: rs1のレジスタ番号（読出し時のみ有効）
  - $RF_{rs2}[4:0]$: rs2のレジスタ番号（読出し時のみ有効）
  - $RF_{rd}[4:0]$: rdのレジスタ番号（書込み時のみ有効）
  - RF_{SEL} : rd入力のバス選択（BUS0, BUS3）



- x[0] = 0に固定
- x[0]への書込みは無視

レジスタファイル : $x[32]$
- $rs1 \leftarrow I_{rs1}\ ?\ x[I_{rs1}] : 32'b0$
- $rs2 \leftarrow I_{rs2}\ ?\ x[I_{rs2}] : 32'b0$
- $\text{if}(I_{rd})\ x[I_{rd}] \leftarrow rd$

I_{rs1}, I_{rs2}, I_{rd} : 命令情報のみ



レジスタファイル : $x[32]$
- $rs1 \leftarrow RF_{rs1}\ ?\ x[RF_{rs1}] : 32'b0$
- $rs2 \leftarrow RF_{rs2}\ ?\ x[RF_{rs2}] : 32'b0$
- $\text{if}(RF_{rd})\ x[RF_{rd}] \leftarrow rd$

$RF_{rs1}, RF_{rs2}, RF_{rd}$: タイミング情報を追加

レジスタファイル出力(rs1, rs2)制御論理

IDフィールド変数（命令情報のみ）
$I_{rs1}[4:0] = I_{rs1\_type}\ ?\ inst[19:15] : 5'b0$
$I_{rs2}[4:0] = I_{rs2\_type}\ ?\ inst[24:20] : 5'b0$
$I_{rd}[4:0] = I_{rd\_type}\ ?\ inst[11:7] : 5'b0$

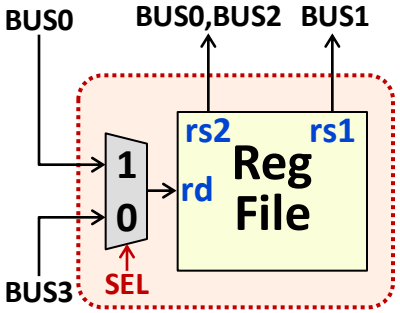
レジスタファイル：$x[32]$

- $rs1 \leftarrow RF_{rs1}\ ?\ x[RF_{rs1}] : 32'b0$
- $rs2 \leftarrow RF_{rs2}\ ?\ x[RF_{rs2}] : 32'b0$
- $\mathrm{if}(RF_{rd})\ x[RF_{rd}] \leftarrow rd$

src：転送元
dst：転送先

条件式	レジスタ転送式	BUS0 (src)	BUS1 (src)	BUS2 (src)	BUS3 (dst)
$T[2] \cdot I_{COMP}$ ：	$rd \leftarrow ALU(rs1, rs2, I_{OP\_type})$		rs1	rs2	rd
$T[2] \cdot I_{COMPI}$ ：	$rd \leftarrow ALU(rs1, imm, I_{OP\_type})$		rs1	imm	rd
$T[2] \cdot I_{LD}$ ：	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD})$		rs1	imm	addr
$T[2] \cdot I_{ST}$ ：	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.din \leftarrow rs2$	rs2	rs1	imm	addr
$T[2] \cdot I_{JALR}$ ：	$rd \leftarrow PC, PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	PC	rs1	imm	PC
$T[2] \cdot I_{BR}$ ：	$FLG \leftarrow ALU(rs1, rs2, I_{OP\_type})$		rs1	rs2	FLG

制御入力	出力IDフィールド変数の論理式
rs1のレジスタ番号	$RF_{rs1}[4:0] = T[2] \cdot (I_{COMP} + I_{COMPI} + I_{LD} + I_{ST} + I_{JALR} + I_{BR})\ ?\ I_{rs1}[4:0] : 5'b0$
rs2のレジスタ番号	$RF_{rs2}[4:0] = T[2] \cdot (I_{COMP} + I_{ST} + I_{BR})\ ?\ I_{rs2}[4:0] : 5'b0$



（冗長項の削除）→ 証明せよ！

制御入力	IDフィールド変数の論理式
rs1のレジスタ番号	$RF_{rs1}[4:0] = T[2]\ ?\ I_{rs1}[4:0] : 5'b0$
rs2のレジスタ番号	$RF_{rs2}[4:0] = T[2]\ ?\ I_{rs2}[4:0] : 5'b0$

レジスタファイル入力(rd)制御論理

IDフィールド変数（命令情報のみ）
$I_{rs1}[4:0] = I_{rs1\_type}\ ?\ inst[19:15] : 5'b0$
$I_{rs2}[4:0] = I_{rs2\_type}\ ?\ inst[24:20] : 5'b0$
$I_{rd}[4:0] = I_{rd\_type}\ ?\ inst[11:7] : 5'b0$

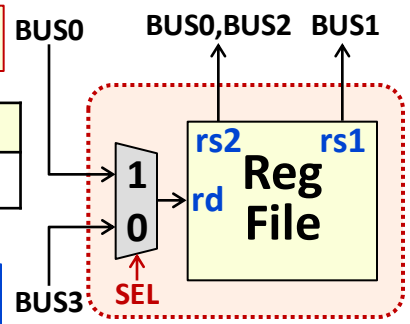
レジスタファイル：$x[32]$

- $rs1 \leftarrow RF_{rs1}\ ?\ x[RF_{rs1}] : 32'b0$
- $rs2 \leftarrow RF_{rs2}\ ?\ x[RF_{rs2}] : 32'b0$
- $\mathrm{if}(RF_{rd})\ x[RF_{rd}] \leftarrow rd$

条件式	レジスタ転送式（src：転送元 dst：転送先）	BUS0 (src)	BUS0 (dst)	BUS1 (src)	BUS2 (src)	BUS3 (dst)
$T[2] \cdot I_{COMP}$ ：	$rd \leftarrow ALU(rs1, rs2, I_{OP\_type})$			rs1	rs2	rd
$T[2] \cdot I_{COMPI}$ ：	$rd \leftarrow ALU(rs1, imm, I_{OP\_type})$			rs1	imm	rd
$T[2] \cdot I_{LUI}$ ：	$rd \leftarrow ALU(zero, imm, I_{OP\_ADD})$				imm	rd
$T[2] \cdot I_{AUIPC}$ ：	$rd \leftarrow ALU(TR, imm, I_{OP\_ADD})$			TR	imm	rd
$T[3] \cdot I_{LD}$ ：	$rd \leftarrow MEM.dout$					rd
$T[2] \cdot I_{JAL}$ ：	$rd \leftarrow PC, PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$	PC	rd	TR	imm	PC
$T[2] \cdot I_{JALR}$ ：	$rd \leftarrow PC, PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	PC	rd	rs1	imm	PC

rd入力IDフィールド変数の論理式

$$RF_{rd}[4:0] = \left(T[2] \cdot \left(I_{COMP} + I_{COMPI} + I_{LUI} + I_{AUIPC} + I_{JAL} + I_{JALR}\right) + T[3] \cdot I_{LD}\right)\ ?\ I_{rd}[4:0] : 5'b0$$



rd入力のバス選択の論理式

$$RF_{SEL} = T[2] \cdot \left(I_{JAL} + I_{JALR}\right)$$

$MEM.dout, ALU.out$

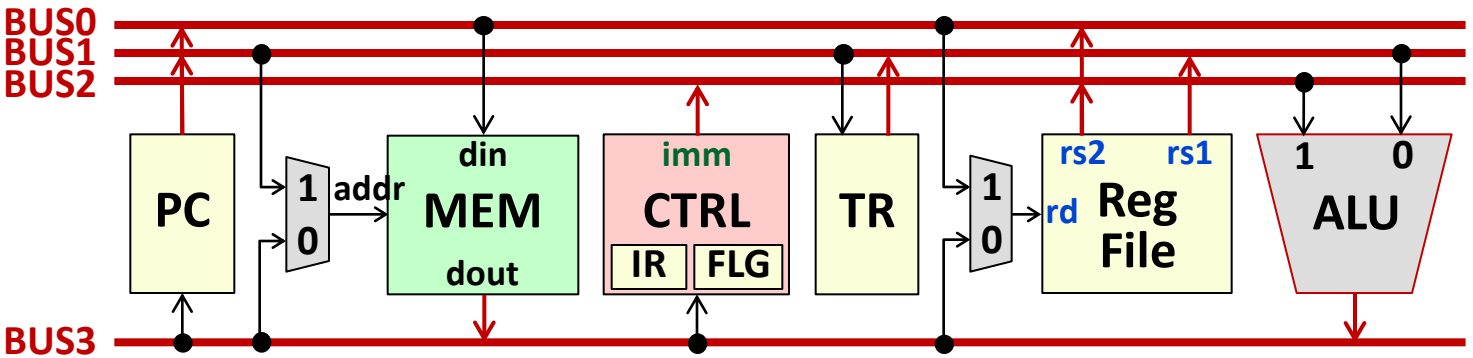
（冗長項の削除）→ 証明せよ！

rd入力IDフィールド変数の論理式

$$RF_{rd}[4:0] = \left(T[2] \cdot \overline{I_{LD}} + T[3] \cdot I_{LD}\right)\ ?\ I_{rd}[4:0] : 5'b0$$

ヒント：$x \cdot y = 0$ ならば $x \cdot \bar{y} = x$

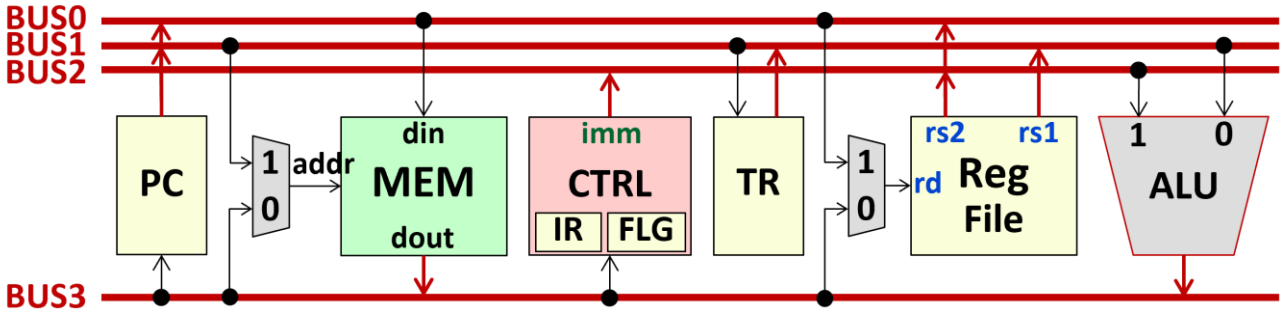
データパス制御論理設計 (4) : バス制御論理



条件式	レジスタ転送式	BUS0 (src)	BUS1 (src)	BUS2 (src)	BUS3 (dst)	BUS3 (src)
$T[0]$:	$MEM.addr \leftarrow PC, MEM.SetRead(I_{LD\_INST}), TR \leftarrow PC,$ $PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$		PC	+4	PC	ALU.out
$T[1]$:	$IR \leftarrow MEM.dout$				IR	MEM.dout
$T[2] \cdot I_{COMP}$:	$rd \leftarrow ALU(rs1, rs2, I_{OP\_type})$		rs1	rs2	rd	ALU.out
$T[2] \cdot I_{COMPI}$:	$rd \leftarrow ALU(rs1, imm, I_{OP\_type})$		rs1	imm	rd	ALU.out
$T[2] \cdot I_{LUI}$:	$rd \leftarrow ALU(zero, imm, I_{OP\_ADD})$		zero	imm	rd	ALU.out
$T[2] \cdot I_{AUIPC}$:	$rd \leftarrow ALU(TR, imm, I_{OP\_ADD})$		TR	imm	rd	ALU.out
$T[2] \cdot I_{LD}$:	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.SetRead(I_{LD\_type})$		rs1	imm	addr	ALU.out
$T[3] \cdot I_{LD}$:	$rd \leftarrow MEM.dout$				rd	MEM.dout
$T[2] \cdot I_{ST}$:	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD}), MEM.din \leftarrow rs2,$ $MEM.SetWrite(I_{ST\_type})$	rs2	rs1	imm	addr	ALU.out
$T[2] \cdot I_{JAL}$:	$rd \leftarrow PC, PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$	PC	TR	imm	PC	ALU.out
$T[2] \cdot I_{JALR}$:	$rd \leftarrow PC, PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	PC	rs1	imm	PC	ALU.out
$T[2] \cdot I_{BR}$:	$FLG \leftarrow ALU(rs1, rs2, I_{OP\_type})$		rs1	rs2	FLG	ALU.out
$T[3] \cdot I_{BR}$:	$\text{if}(FLG)\ PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$		TR	imm	PC	ALU.out
	データ転送元(src)	PC, RF.out1	PC, TR, RF.out0	CTRL.imm, RF.out1		ALU.out, MEM.dout

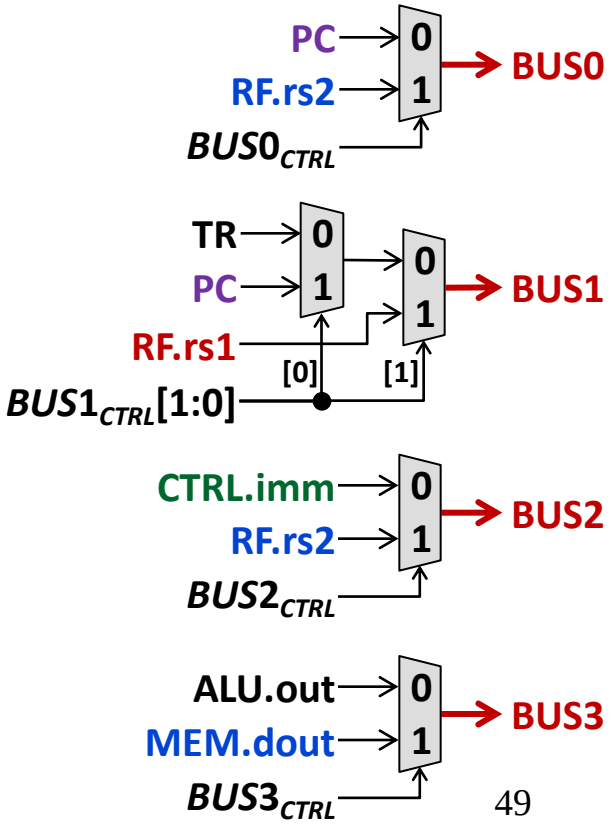
rs1 ← RF.rs1
rs2 ← RF.rs2
imm, +4 ← CTRL.imm

バス制御論理

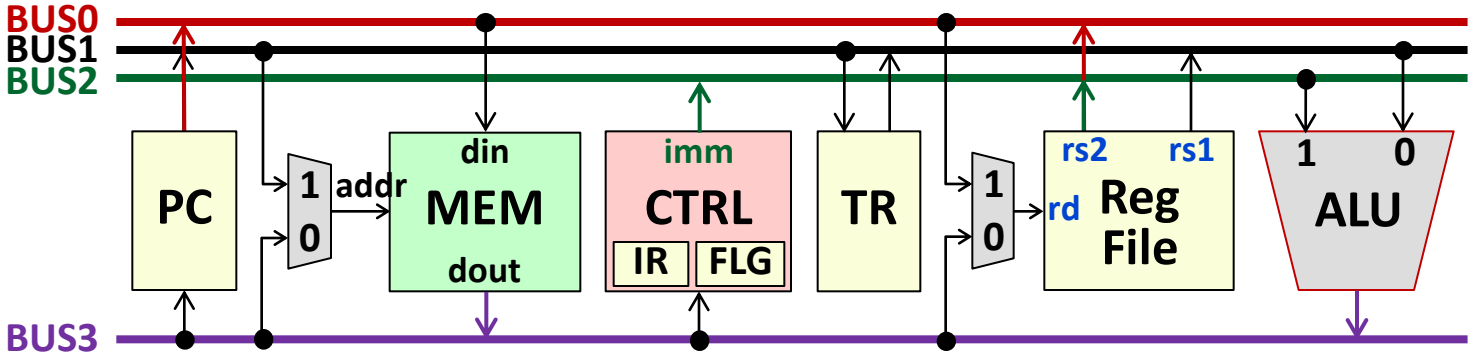


rs1 ← RF.rs1
rs2 ← RF.rs2
imm, +4 ← CTRL.imm

条件式	BUS0 (src)	BUS1 (src)	BUS2 (src)	BUS3 (src)
$T[0]$:	*	01: PC	0: +4	0: ALU.out
$T[1]$:	*	**	*	1: MEM.dout
$T[2] \cdot I_{COMP}$:	*	1*: rs1	1: rs2	0: ALU.out
$T[2] \cdot I_{COMPI}$:	*	1*: rs1	0: imm	0: ALU.out
$T[2] \cdot I_{LUI}$:	*	1*: zero	0: imm	0: ALU.out
$T[2] \cdot I_{AUIPC}$:	*	00: TR	0: imm	0: ALU.out
$T[2] \cdot I_{LD}$:	*	1*: rs1	0: imm	0: ALU.out
$T[3] \cdot I_{LD}$:	*	**	*	1: MEM.dout
$T[2] \cdot I_{ST}$:	1: rs2	1*: rs1	0: imm	0: ALU.out
$T[2] \cdot I_{JAL}$:	0: PC	00: TR	0: imm	0: ALU.out
$T[2] \cdot I_{JALR}$:	0: PC	1*: rs1	0: imm	0: ALU.out
$T[2] \cdot I_{BR}$:	*	1*: rs1	1: rs2	0: ALU.out
$T[3] \cdot I_{BR}$:	*	00: TR	0: imm	0: ALU.out
データ転送元 (src)	PC(0), RF.rs2(1)	TR(00), PC(01), RF.rs1(1*)	CTRL.imm(0), RF.rs2(1)	ALU.out(0), MEM.dout(1)

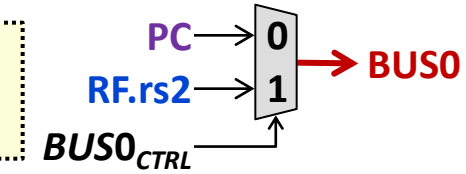


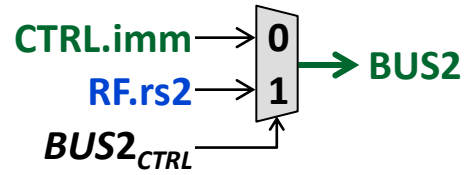
バス制御論理（BUS0,BUS2,BUS3）

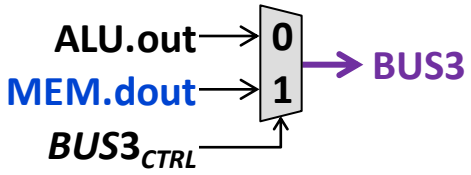


条件式	BUS0	BUS2	BUS3
$T[0]$:	*	0: +4	0: ALU.out
$T[1]$:	*	*	1: MEM.dout
$T[2] \cdot I_{COMP}$:	*	1: rs2	0: ALU.out
$T[2] \cdot I_{COMPI}$:	*	0: imm	0: ALU.out
$T[2] \cdot I_{LUI}$:	*	0: imm	0: ALU.out
$T[2] \cdot I_{AUIPC}$:	*	0: imm	0: ALU.out
$T[2] \cdot I_{LD}$:	*	0: imm	0: ALU.out
$T[3] \cdot I_{LD}$:	*	*	1: MEM.dout
$T[2] \cdot I_{ST}$:	1: rs2	0: imm	0: ALU.out
$T[2] \cdot I_{JAL}$:	0: PC	0: imm	0: ALU.out
$T[2] \cdot I_{JALR}$:	0: PC	0: imm	0: ALU.out
$T[2] \cdot I_{BR}$:	*	1: rs2	0: ALU.out
$T[3] \cdot I_{BR}$:	*	0: imm	0: ALU.out
データ転送元（src）	PC(0), RF.rs2(1)	CTRL.imm(0), RF.rs2(1)	ALU.out(0), MEM.dout(1)

rs1 ← RF.rs1
rs2 ← RF.rs2
imm, +4 ← CTRL.imm





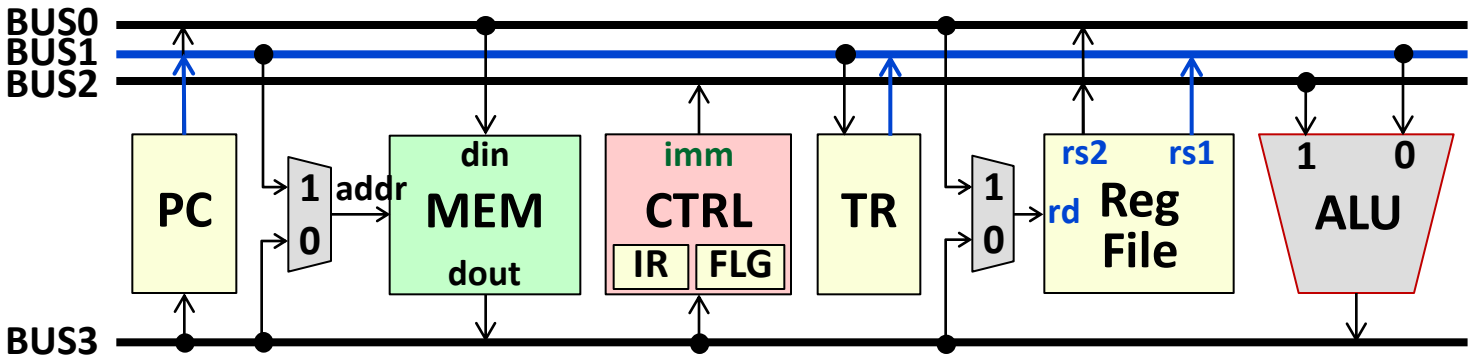


$BUS0_{CTRL} = I_{ST}$

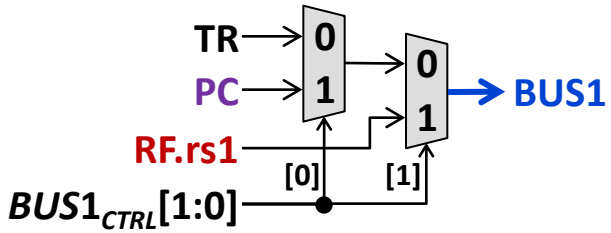
$BUS2_{CTRL} = T[2] \cdot (I_{COMP} + I_{BR})$

$BUS3_{CTRL} = T[1] + T[2] \cdot I_{LD}$

バス制御論理（BUS1）



条件式	$BUS1_{CTRL}[1:0]$ TR(00), PC(01), RF.rs1(1*)	$BUS1_{CTRL}[0]$ TR(0), PC(1)	$BUS1_{CTRL}[1]$ TR/PC(0), RF.rs1(1)
$T[0]$:	01: PC	1: PC	0: PC
$T[1]$:	**	*	*
$T[2] \cdot I_{COMP}$:	1*: rs1	*	1: rs1
$T[2] \cdot I_{COMPI}$:	1*: rs1	*	1: rs1
$T[2] \cdot I_{LUI}$:	1*: zero	*	1: zero
$T[2] \cdot I_{AUIPC}$:	00: TR	0: TR	0: TR
$T[2] \cdot I_{LD}$:	1*: rs1	*	1: rs1
$T[3] \cdot I_{LD}$:	**	*	*
$T[2] \cdot I_{ST}$:	1*: rs1	*	1: rs1
$T[2] \cdot I_{JAL}$:	00: TR	0: TR	0: TR
$T[2] \cdot I_{JALR}$:	1*: rs1	*	1: rs1
$T[2] \cdot I_{BR}$:	1*: rs1	*	1: rs1
$T[3] \cdot I_{BR}$:	00: TR	0: TR	0: TR

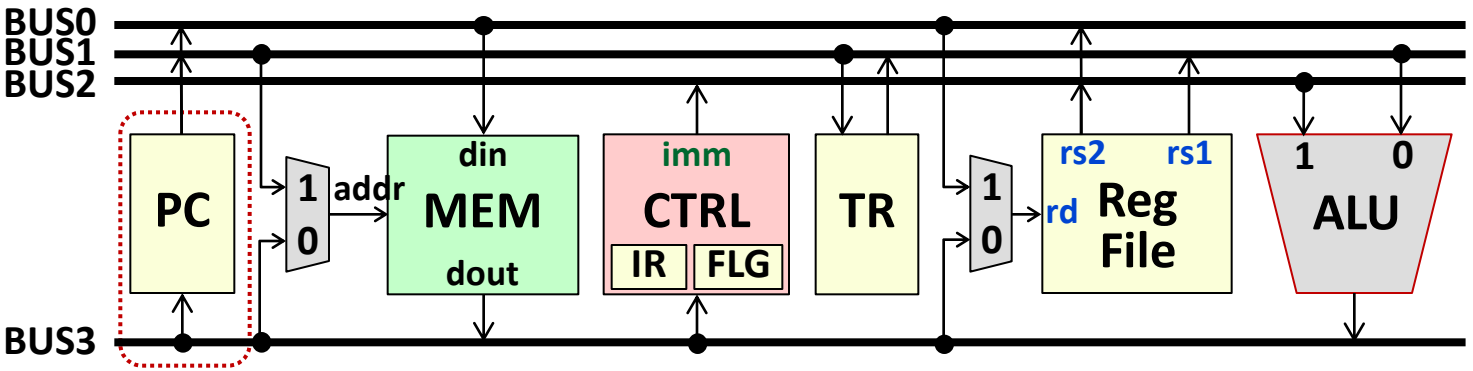


rs1 ← RF.rs1
rs2 ← RF.rs2
imm, +4 ← CTRL.imm

$$BUS1_{CTRL}[0] = T[0]$$

$$BUS1_{CTRL}[1] = T[2] \cdot (I_{COMP} + I_{COMPI} + I_{LUI} + I_{LD} + I_{ST} + I_{JALR} + I_{BR})$$

データパス制御論理設計 (5)：PC書込み制御論理



条件式	レジスタ転送式（PC更新）	書込み制御変数（PC_{LD}）
$T[0]$：	$PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$	$PC_{LD} = T[0] + T[2] \cdot (I_{JAL} + I_{JALR}) + T[3] \cdot I_{BR} \cdot FLG$
$T[2] \cdot I_{JAL}$：	$PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$	
$T[2] \cdot I_{JALR}$：	$PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$	
$T[3] \cdot I_{BR}$：	if(FLG) $PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$	

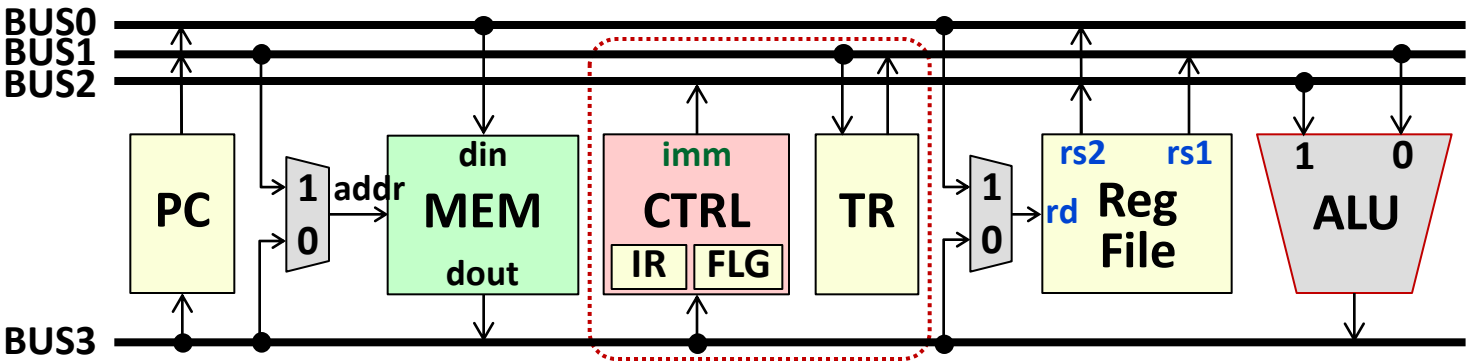
条件付きレジスタ代入式の場合、条件項をレジスタ転送式の条件式にANDで結合

条件式	レジスタ転送式（PC更新）
PC_{LD}：	$PC \leftarrow \{BUS3[31:1], 1'b0\}$

RISC-V命令ビット長は32ビット/16ビットなので、命令アドレスは2の倍数に制限
→最下位ビット（ビット0）を強制的に「0」にする

JALR rd, simm(rs1)	$rd \leftarrow PC + 4, PC \leftarrow ((rs1 + simm[11:0])\&(\sim 1))$

データパス制御論理設計 (6)：TR/IR/FLG書込み制御

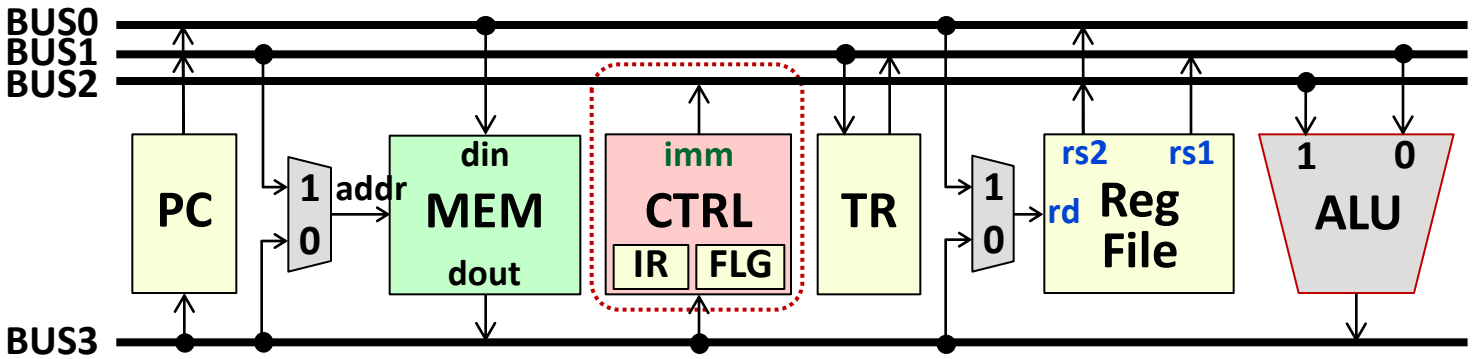


条件式	レジスタ転送式（TR/IR/FLG更新）	書込み制御変数
$T[0]$:	$TR \leftarrow PC$	$TR_{LD} = T[0]$
$T[1]$:	$IR \leftarrow MEM.dout$	$IR_{LD} = T[1]$
$T[2] \cdot I_{BR}$:	$FLG \leftarrow ALU(rs1, rs2, I_{OP\_type})$	$FLG_{LD} = T[2] \cdot I_{BR}$

条件式	レジスタ転送式（TR/IR/FLG更新）
TR_{LD} :	$TR \leftarrow BUS1$
IR_{LD} :	$IR \leftarrow BUS3$
FLG_{LD} :	$FLG \leftarrow BUS3[0]$

1ビットフラグレジスタ
FLG（比較結果）

データパス制御論理設計 (7) : 即値データ出力



条件式	レジスタ転送式（即値データ転送）
$T[0]$:	$PC \leftarrow ALU(PC, +4, I_{OP\_ADD})$
$T[2] \cdot I_{COMPI}$:	$rd \leftarrow ALU(rs1, imm, I_{OP\_type})$
$T[2] \cdot I_{LUI}$:	$rd \leftarrow ALU(zero, imm, I_{OP\_ADD})$
$T[2] \cdot I_{AUIPC}$:	$rd \leftarrow ALU(TR, imm, I_{OP\_ADD})$
$T[2] \cdot I_{LD}$:	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD})$
$T[2] \cdot I_{ST}$:	$MEM.addr \leftarrow ALU(rs1, imm, I_{OP\_ADD})$
$T[2] \cdot I_{JAL}$:	$rd \leftarrow PC, PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$
$T[2] \cdot I_{JALR}$:	$rd \leftarrow PC, PC \leftarrow ALU(rs1, imm, I_{OP\_ADD})$
$T[3] \cdot I_{BR}$:	$\text{if}(FLG)\ PC \leftarrow ALU(TR, imm, I_{OP\_ADD})$

$$C_{IMM\_4} = T[0]\ ;$$
$$A_{IMM\_4} = C_{IMM\_4}\ ?\ 32'h4 : 32'h0\ ;$$
$$C_{IMM\_IR} = \overline{T[0]}\ ;$$
$$A_{IMM\_IR} = C_{IMM\_IR}\ ?\ imm : 32'h0\ ;$$
$$CTRL_{imm} = A_{IMM\_IR} \mid A_{IMM\_4}\ ;$$

k'h<16進定数>: kビット幅の16進表記の定数

ビット範囲	即値ビット論理式
$n = 2$	$CTRL_{imm}[2] = imm[2] + T[0]$
$0 \le n \le 1$ $3 \le n \le 31$	$CTRL_{imm}[n] = \overline{T[0]} \cdot imm[n]$

$$I_{ISBJ\_type} = I_{I\_type} + I_{S\_type} + I_{B\_type} + I_{J\_type}$$
$$I_{ISB\_type} = I_{I\_type} + I_{S\_type} + I_{B\_type}$$
$$I_{JU\_type} = I_{J\_type} + I_{U\_type}$$
$$I_{IS\_type} = I_{I\_type} + I_{S\_type}$$
$$I_{IJ\_type} = I_{I\_type} + I_{J\_type}$$
$$I_{SB\_type} = I_{S\_type} + I_{\_type}$$
$$I_{ISBJ\_msb} = I_{ISBJ\_type} \cdot inst[31]$$
$$I_{ISB\_msb} = I_{ISB\_type} \cdot inst[31]$$

ビット範囲	即値ビット論理式
$20 \le n \le 31$	$imm[n] = I_{ISBJ\_msb} + I_{U\_type} \cdot inst[n]$
$12 \le n \le 19$	$imm[n] = I_{ISB\_msb} + I_{JU\_type} \cdot inst[n]$
$n = 11$	$imm[11] = I_{IS\_type} \cdot inst[31] + I_{B\_type} \cdot inst[7] + I_{J\_type} \cdot inst[20]$
$5 \le n \le 10$	$imm[n] = I_{ISBJ\_type} \cdot inst[20+n]$
$1 \le n \le 4$	$imm[n] = I_{IJ\_type} \cdot inst[20+n] + I_{SB\_type} \cdot inst[7+n]$
$n = 0$	$imm[0] = I_{I\_type} \cdot inst[20] + I_{S\_type} \cdot inst[7]$

データパス制御論理設計：まとめ

メモリ制御論理：$MEM_{CTRL}[8:0] = \{M_{LD\_CTRL}[4:0], M_{ST\_CTRL}[2:0], M_{SEL}\}$			
LB	$M_{LD\_CTRL}[4] = T[2] \cdot I_{LB}$	SB	$M_{ST\_CTRL}[2] = T[2] \cdot I_{SB}$
LBU	$M_{LD\_CTRL}[3] = T[2] \cdot I_{LBU}$	SH	$M_{ST\_CTRL}[1] = T[2] \cdot I_{SH}$
LH	$M_{LD\_CTRL}[2] = T[2] \cdot I_{LH}$	SW	$M_{ST\_CTRL}[0] = T[2] \cdot I_{SW}$
LHU	$M_{LD\_CTRL}[1] = T[2] \cdot I_{LHU}$		$M_{SEL} = T[0]$
LW	$M_{LD\_CTRL}[0] = T[2] \cdot I_{LW} + T[0]$		

レジスタファイル制御論理： $RegFile_{CTRL}[8:0] = \{RF_{rs1}[4:0], RF_{rs2}[4:0],, RF_{rd}[4:0], RF_{SEL}\}$	
rs1レジスタ番号	$RF_{rs1}[4:0] = T[2]\ ?\ I_{rs1}[4:0] : 5'b0$
rs2レジスタ番号	$RF_{rs2}[4:0] = T[2]\ ?\ I_{rs2}[4:0] : 5'b0$
rdレジスタ番号	$RF_{rd}[4:0] = (T[2] \cdot \overline{I_{LD}} + T[3] \cdot I_{LD})\ ?\ I_{rd}[4:0] : 5'b0$
rd選択制御	$RF_{SEL} = T[2] \cdot (I_{JAL} + I_{JALR})$

バス制御論理
$BUS0_{CTRL} = I_{ST}$
$BUS1_{CTRL}[0] = T[0]$
$BUS1_{CTRL}[1] = T[2] \cdot (I_{COMP} + I_{COMPI} + I_{LUI} + I_{LD} + I_{ST} + I_{JALR} + I_{BR})$
$BUS2_{CTRL} = T[2] \cdot (I_{COMP} + I_{BR})$
$BUS3_{CTRL} = T[1] + T[2] \cdot I_{LD}$

レジスタ書込み制御論理
$PC_{LD} = T[0] + T[2] \cdot (I_{JAL} + I_{JALR}) + T[3] \cdot I_{BR} \cdot FLG$
$TR_{LD} = T[0]$
$IR_{LD} = T[1]$
$FLG_{LD} = T[2] \cdot I_{BR}$

ALU制御論理：$ALU_{CTRL}[13:0]$
$ALU_{CTRL}[13] = C_{ADD} = T[2] \cdot I_{ADD} + C_{OP\_ADD}$
$ALU_{CTRL}[12] = C_{SUB} = T[2] \cdot I_{SUB}$
$ALU_{CTRL}[11] = C_{LT} = T[2] \cdot I_{LT}$
$ALU_{CTRL}[10] = C_{LTU} = T[2] \cdot I_{LTU}$
$ALU_{CTRL}[9] = C_{AND} = T[2] \cdot I_{AND}$
$ALU_{CTRL}[8] = C_{OR} = T[2] \cdot I_{OR}$
$ALU_{CTRL}[7] = C_{XOR} = T[2] \cdot I_{XOR}$
$ALU_{CTRL}[6] = C_{SLL} = T[2] \cdot I_{SLL}$
$ALU_{CTRL}[5] = C_{SRL} = T[2] \cdot I_{SRL}$
$ALU_{CTRL}[4] = C_{SRA} = T[2] \cdot I_{SRA}$
$ALU_{CTRL}[3] = C_{EQ} = T[2] \cdot I_{EQ}$
$ALU_{CTRL}[2] = C_{NE} = T[2] \cdot I_{NE}$
$ALU_{CTRL}[1] = C_{GE} = T[2] \cdot I_{GE}$
$ALU_{CTRL}[0] = C_{GEU} = T[2] \cdot I_{GEU}$

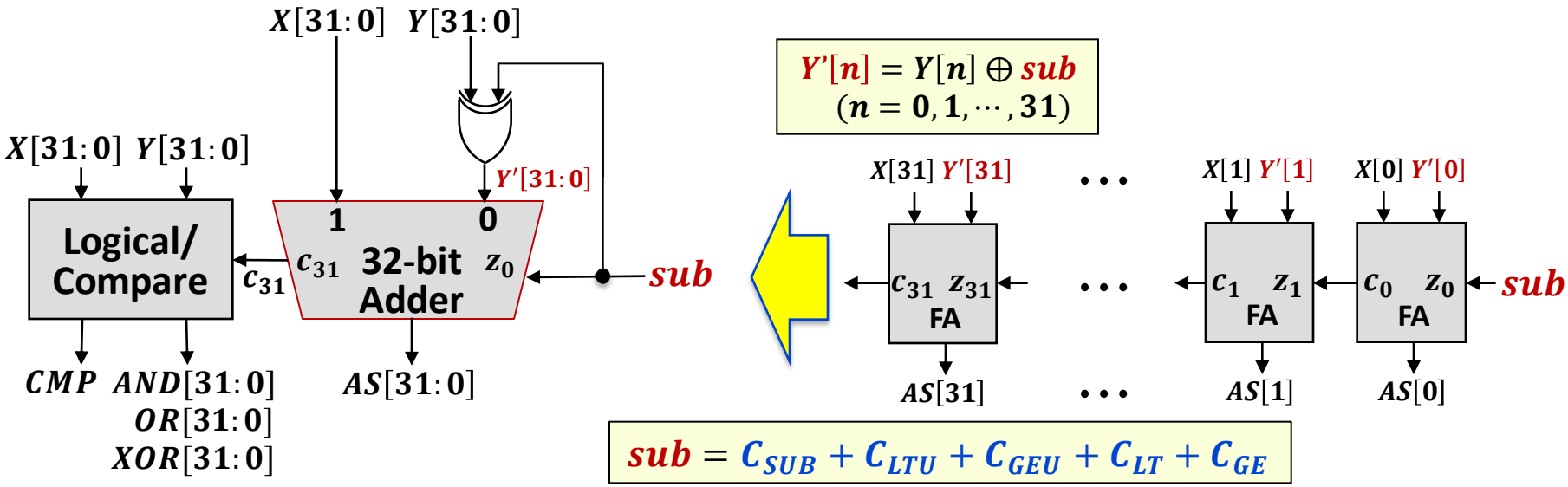
各制御信号を「タイミング情報」と「命令情報」で表現

即値データ出力論理：$CTRL_{imm}[31:0]$	
ビット範囲	即値ビット論理式
$n = 4$	$CTRL_{imm}[4] = imm[4] + T[0]$
$0 \le n \le 3$ $5 \le n \le 31$	$CTRL_{imm}[n] = \overline{T[0]} \cdot imm[n]$

資料概要

1. RV32-I命令セット仕様
2. RV32-Iハードウェアアーキテクチャ（ARCH-1）
3. 命令デコード動作
4. メモリ読出し･書込み動作
5. 命令フェッチサイクル
6. 命令実行サイクルのレジスタ転送記述
7. データパス制御論理設計
8. 算術論理演算器設計

算術演算器(加減算・大小比較)



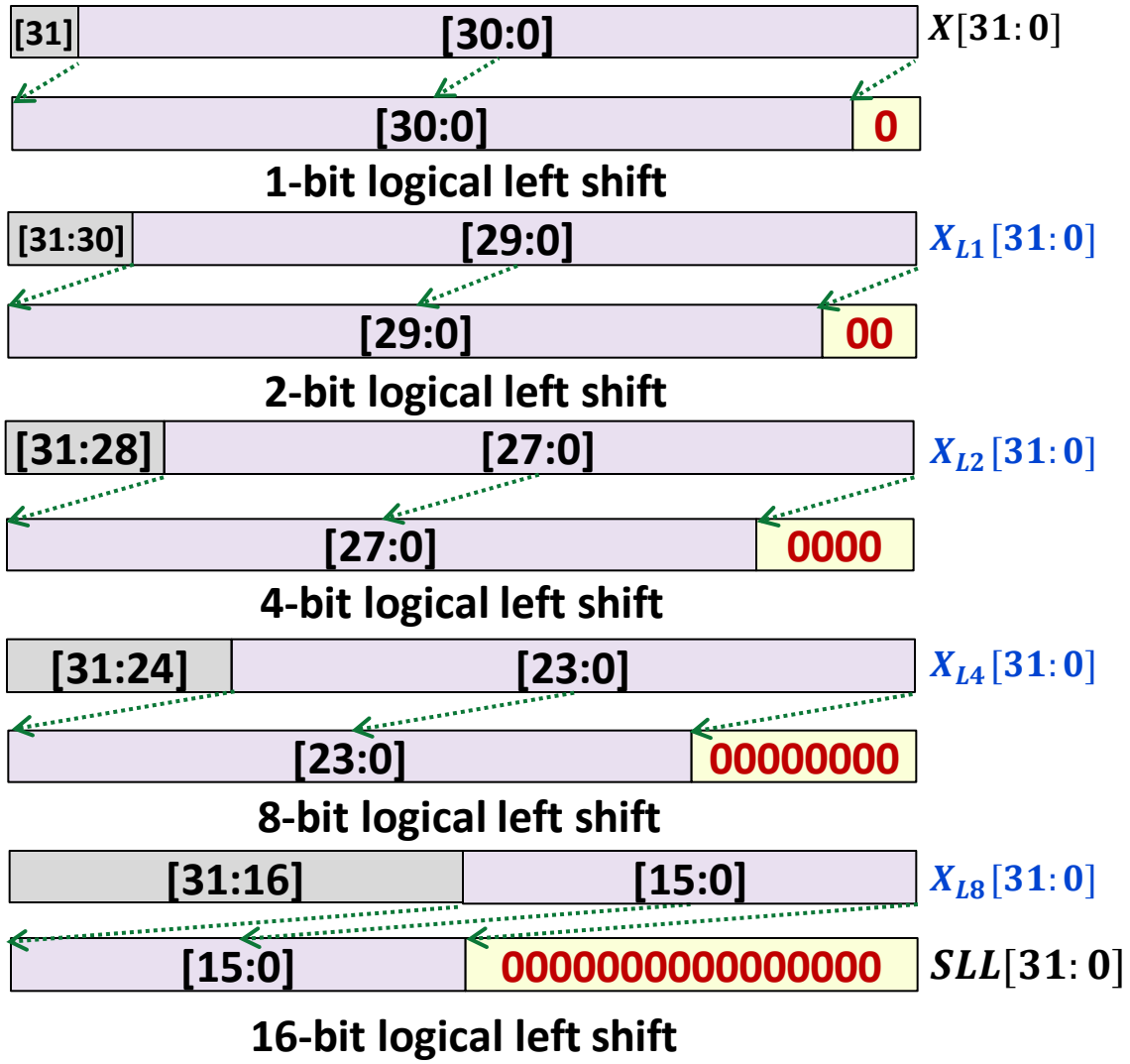
$$sub = C_{SUB} + C_{LTU} + C_{GEU} + C_{LT} + C_{GE}$$

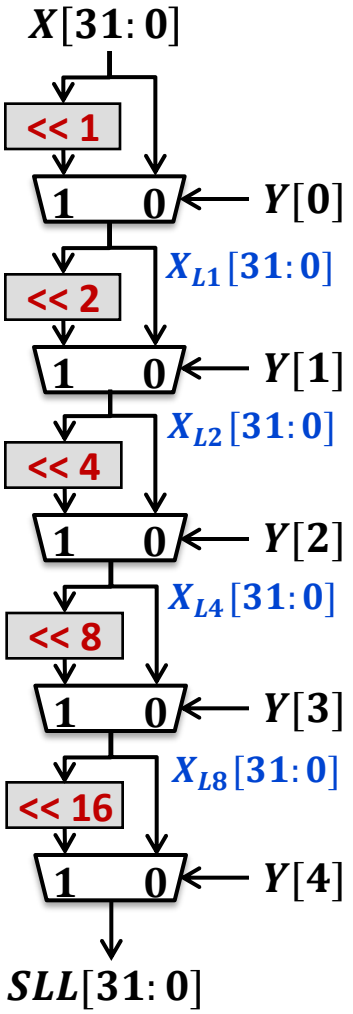
$$CMP = LTU \cdot C_{LTU} + \overline{LTU} \cdot C_{GEU} + LT \cdot C_{LT} + \overline{LT} \cdot C_{GE} + NE \cdot C_{NE} + \overline{NE} \cdot C_{EQ}$$

$AND[n] = X[n] \cdot Y[n] \ (0 \le n \le 31)$
$OR[n] = X[n] + Y[n] \ (0 \le n \le 31)$
$XOR[n] = X[n] \oplus Y[n] \ (0 \le n \le 31)$
$NE = XOR[31] + \cdots + XOR[1] + XOR[0]$
$EQ = \overline{NE}$
$LTU = \overline{c_{31}}$
$GEU = \overline{LTU}$
$LT = X[31] \oplus Y[31] \oplus \overline{c_{31}}$
$GE = \overline{LT}$

$ALU_{CTRL}[13] = C_{ADD} = T[2] \cdot I_{ADD} + C_{OP\_ADD}$
$ALU_{CTRL}[12] = C_{SUB} = T[2] \cdot I_{SUB}$
$ALU_{CTRL}[11] = C_{LT} = T[2] \cdot I_{LT}$
$ALU_{CTRL}[10] = C_{LTU} = T[2] \cdot I_{LTU}$
$ALU_{CTRL}[3] = C_{EQ} = C_{OP\_type} \cdot I_{EQ}$
$ALU_{CTRL}[2] = C_{NE} = C_{OP\_type} \cdot I_{NE}$
$ALU_{CTRL}[1] = C_{GE} = T[2] \cdot I_{GE}$
$ALU_{CTRL}[0] = C_{GEU} = T[2] \cdot I_{GEU}$

論理左シフト演算 : SLL



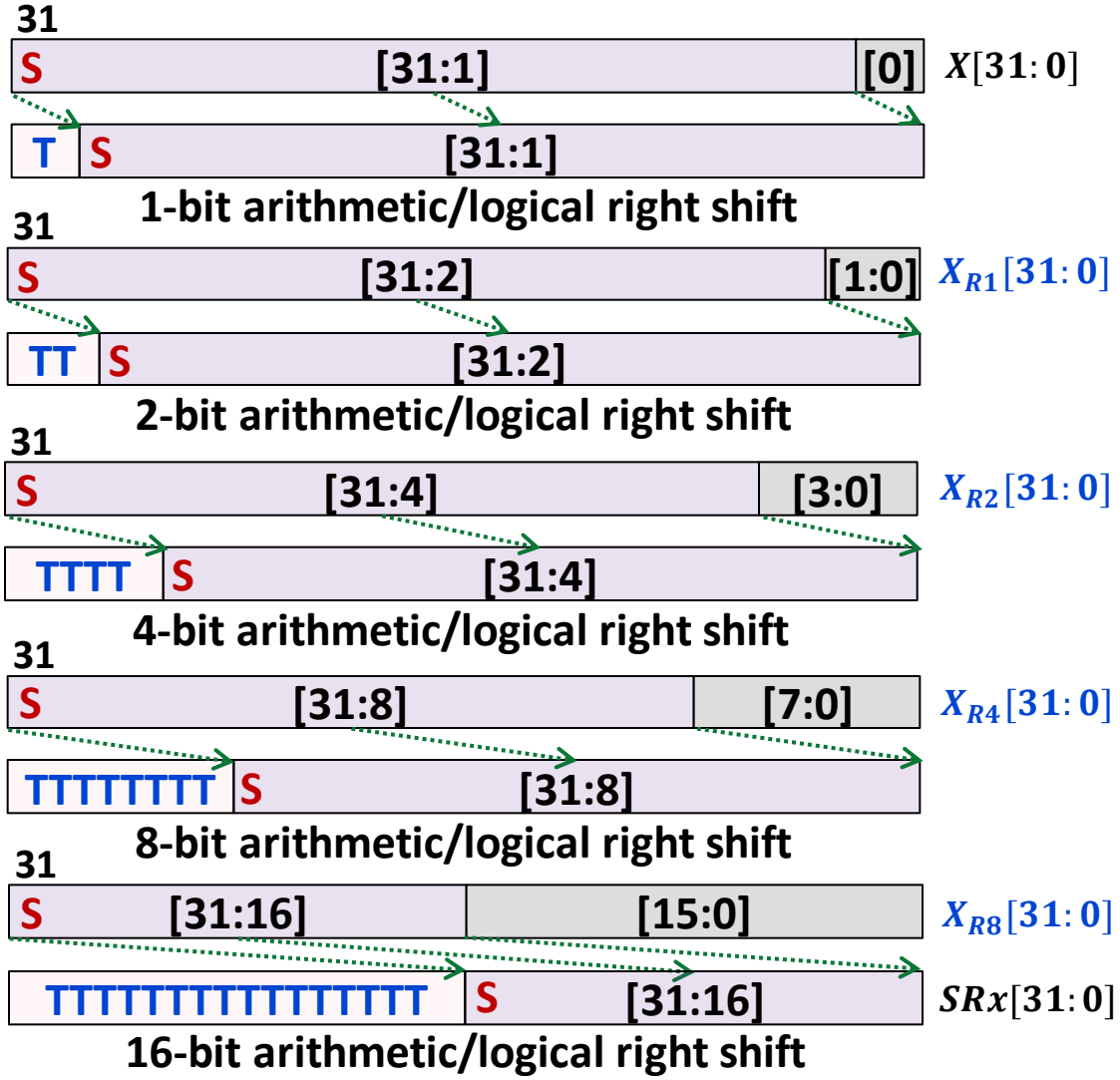


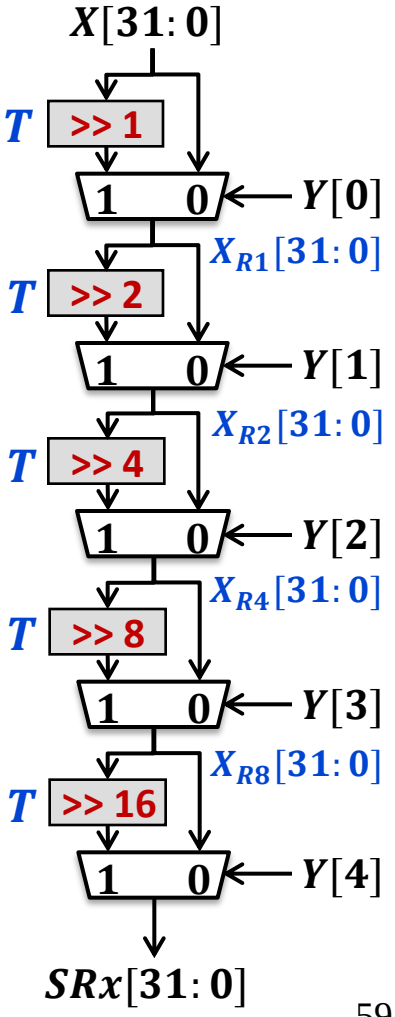
論理・算術右シフト演算: SRL/SRA

S : sign bit

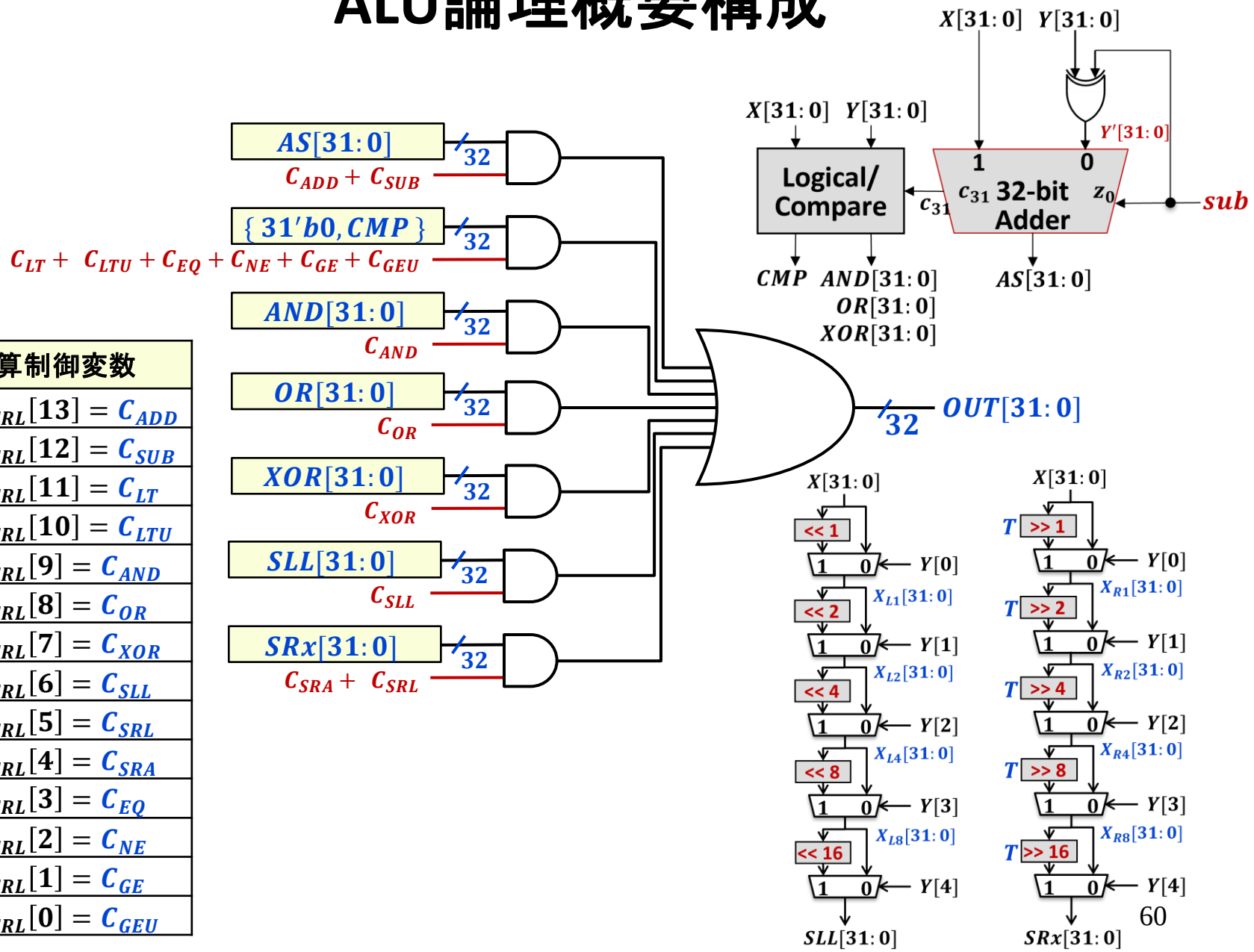
$T = S \cdot C_{SRA}$: extension bit

$ALU_{CTRL}[4] = C_{SRA} = T[2] \cdot I_{SRA}$





ALU論理概要構成



演算制御変数
$ALU_{CTRL}[13] = C_{ADD}$
$ALU_{CTRL}[12] = C_{SUB}$
$ALU_{CTRL}[11] = C_{LT}$
$ALU_{CTRL}[10] = C_{LTU}$
$ALU_{CTRL}[9] = C_{AND}$
$ALU_{CTRL}[8] = C_{OR}$
$ALU_{CTRL}[7] = C_{XOR}$
$ALU_{CTRL}[6] = C_{SLL}$
$ALU_{CTRL}[5] = C_{SRL}$
$ALU_{CTRL}[4] = C_{SRA}$
$ALU_{CTRL}[3] = C_{EQ}$
$ALU_{CTRL}[2] = C_{NE}$
$ALU_{CTRL}[1] = C_{GE}$
$ALU_{CTRL}[0] = C_{GEU}$