Department of Computer Science
Course number: CSC.T341

コンピュータ論理設計
Computer Logic Design

8. 命令セットアーキテクチャ：ロードストア命令と分岐命令
Instruction Set Architecture: Load/Store and Branch Instructions

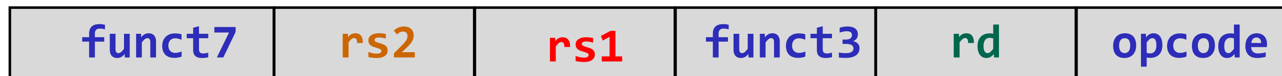
吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp　www.arch.cs.titech.ac.jp/lecture/CLD/
講義： 月曜日 10:45-12:25, 木曜日 10:45-12:25

Ver. 2023-05-08a

Machine Language - Add Instruction

- Instructions, like registers and words of data, are 32 bits long
- Arithmetic Instruction Format (R-type):

```
add  x7, x8, x9
```



R-type

opcode 7-bits *opcode* that specifies the operation

rs1 5-bits register file address of the first source operand

rs2 5-bits register file address of the second source operand

rd 5-bits register file address of the result's destination

funct3 and **funct7** 10-bits select the type of operation (function)

RISC-V Immediate Instructions

- Small constants are used often in typical code
- Possible approaches?
  - put "typical constants" in memory and load them
  - create hard-wired registers (like x0) for constants like 1
  - have special instructions that contain constants !

```
addi  x7, x8, -2   # x7 = x8 + (-2)
```

- Machine format (I format):

I-type

- The constant is kept inside the instruction itself
  - Immediate format limits values to the range $+2^{11}-1$ to -2^{11}

Two major ISA types: RISC vs CISC

- RISC (Reduced Instruction Set Computer) philosophy
  - fixed instruction lengths
  - load-store instruction sets
  - limited addressing modes
  - limited operations
  - RISC: MIPS, Alpha, ARM, RISC-V, …
- CISC (Complex Instruction Set Computer) philosophy
  - ! fixed instruction lengths
  - ! load-store instruction sets
  - ! limited addressing modes
  - ! limited operations
  - CISC : DEC VAX11, Intel 80x86, …

RISC-V の32ビット基本命令セット RV32I

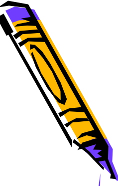
RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
				rd	0010011	SLLI
				rd	0010011	SRLI
				rd	0010011	SRAI
				rd	0110011	ADD
				rd	0110011	SUB
				rd	0110011	SLL
				rd	0110011	SLT
				rd	0110011	SLTU
				rd	0110011	XOR
				rd	0110011	SRL
				rd	0110011	SRA
				rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm pred	succ	rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 ($>32b$)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	$48b$
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	$64b$
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	$48b$
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq 80b$

Table 24.1: RISC-V base opcode map, inst[1:0]=11

RISC-V Memory Access Instructions



- RISC-V has two basic data transfer instructions for accessing memory

```
lw   x5, 24(x7)  # load word from memory
sw   x3, 28(x9)  # store word to memory
```

- The data is loaded into (lw) or stored from (sw) a register in the register file
- The memory address – a 32 bit address – is formed by adding the contents of the base address register to the offset value

Machine Language - Load Instruction

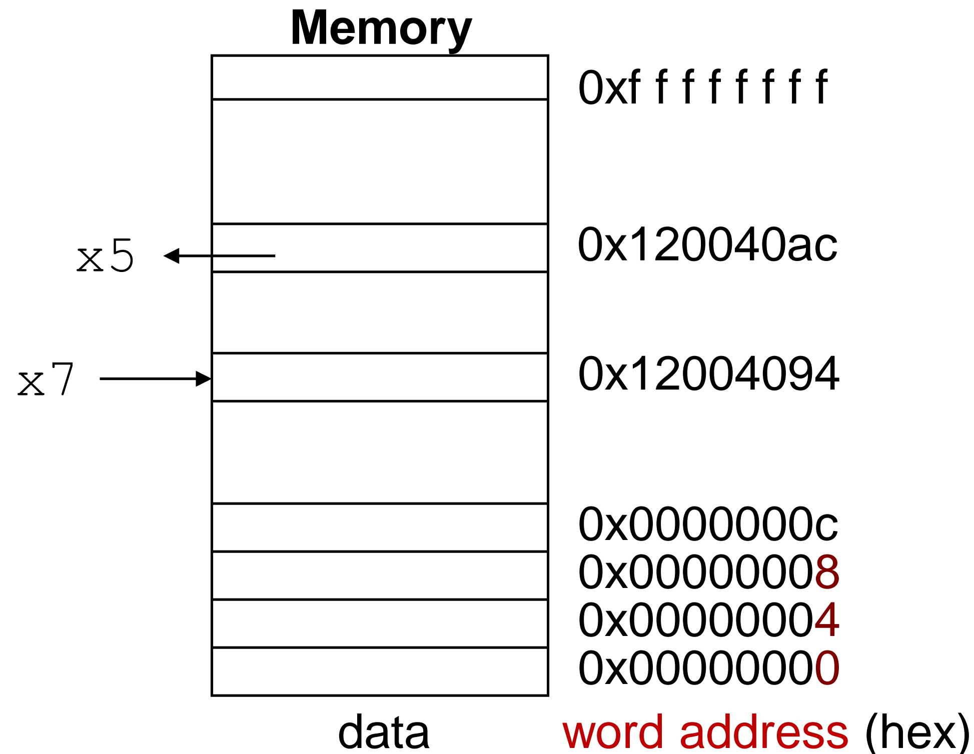
- Load Instruction Format (I-type):

```
lw x5, 24(x7)
```

I-type

24_{10} + x7 =

. . . 0001 1000
\+ . . . 1001 0100
. . . 1010 1100 =
0x120040ac

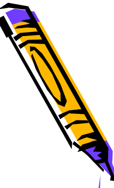


Example（例題）

- g = h + A[8]

100語から成る配列Aがあるとする．また，コンパイラは変数g, h にレジスタ x5, x6 を割り付ける．さらに配列の開始アドレスは x7 に納められているとする．
上のステートメントをコンパイルせよ．

Answer



- g = h + A[8]

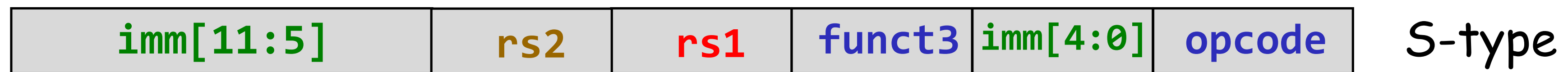
  100語から成る配列Aがあるとする．また，コンパイラは変数g, h にレジスタ x5, x6 を割り付ける．さらに配列の開始アドレスは x7 に納められているとする．
  上のステートメントをコンパイルせよ．

```
lw  x9, 32(x7)     # x9 = A[8]
add x5, x6, x9     # g = h + x9
```

Machine Language - Store Instruction

- Load Instruction Format (S-type):

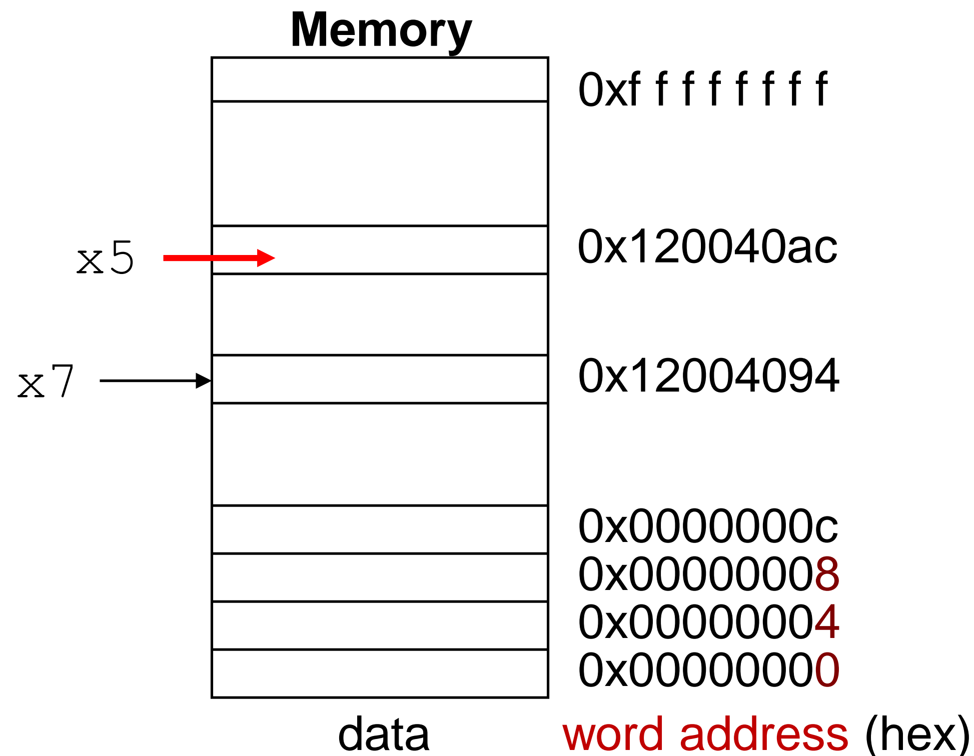
```
sw x5, 24(x7)
```

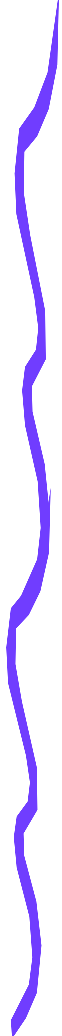


S-type

24_{10} + x7 =

. . . 0001 1000
\+ . . . 1001 0100
. . . 1010 1100 =
0x120040ac





Example（例題）

- A[12] = h + A[8]

  100語から成る配列Aがあるとする．また，コンパイラは変数 h にレジスタ x6 を割り付ける．さらに配列の開始アドレスは x7 に納められているとする．
  上のステートメントをコンパイルせよ．

Answer

- A[12] = h + A[8]

  100語から成る配列Aがあるとする．また，コンパイラは変数 h にレジスタ x6 を割り付ける．さらに配列の開始アドレスは x7 に納められているとする．
  上のステートメントをコンパイルせよ．

```
lw  x9, 32(x7)     # x9 = A[8]
add x9, x6, x9     # x9 = h + x9
sw  x9, 48(x7)     # A[12] = x9
```

RISC-V の32ビット基本命令セット RV32I の分岐命令

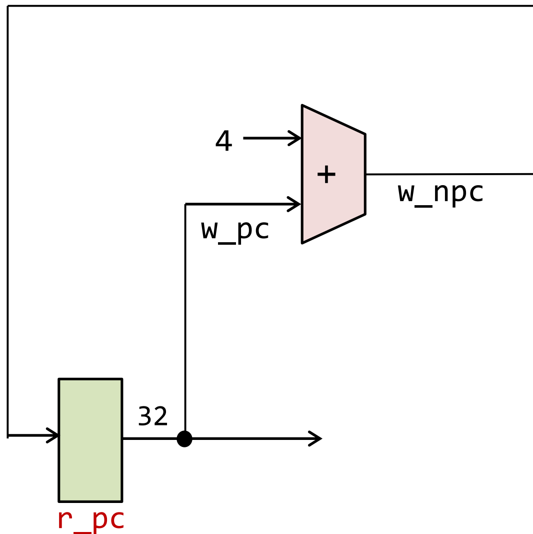
RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU
imm[11:0]		rs1	100	rd	0010011	XORI
imm[11:0]		rs1	110	rd	0010011	ORI
imm[11:0]		rs1	111	rd	0010011	ANDI
				rd	0010011	SLLI
				rd	0010011	SRLI
				rd	0010011	SRAI
				rd	0110011	ADD
				rd	0110011	SUB
				rd	0110011	SLL
				rd	0110011	SLT
				rd	0110011	SLTU
				rd	0110011	XOR
				rd	0110011	SRL
				rd	0110011	SRA
				rd	0110011	OR
0000000	rs2	rs1	111	rd	0110011	AND
fm pred	succ	rs1	000	rd	0001111	FENCE
000000000000		00000	000	00000	1110011	ECALL
000000000001		00000	000	00000	1110011	EBREAK

inst[4:2] / inst[6:5]	000	001	010	011	100	101	110	111 (> 32b)
00	LOAD	LOAD-FP	*custom-0*	MISC-MEM	OP-IMM	AUIPC	OP-IMM-32	48b
01	STORE	STORE-FP	*custom-1*	AMO	OP	LUI	OP-32	64b
10	MADD	MSUB	NMSUB	NMADD	OP-FP	*reserved*	*custom-2/rv128*	48b
11	BRANCH	JALR	*reserved*	JAL	SYSTEM	*reserved*	*custom-3/rv128*	$\geq$ 80b

Table 24.1: RISC-V base opcode map, inst[1:0]=11

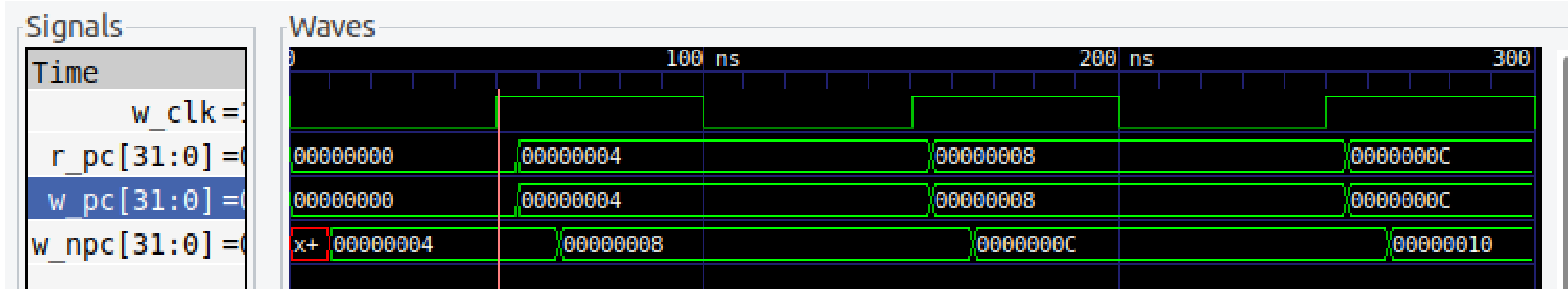
m_proc01 プロセッサの設計と実装に向けた一歩



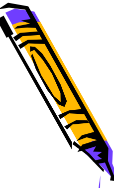
code102.v

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [31:0] w_pc;
  m_main m_main0 (r_clk, w_pc);
  always@(*) #1 $write("%3d %x¥n", $time, w_pc);
  initial $dumpfile("main.vcd"); /* file name for GTKWave */
  initial $dumpvars(0, m_top);   /* module    for GTKWave */
  initial #300 $finish();
endmodule

module m_main (w_clk, w_pc);
  input  wire w_clk;
  output wire [31:0] w_pc;

  reg [31:0] r_pc = 0;
  assign w_pc = r_pc;
  wire [31:0] #10 w_npc = w_pc + 4;
  always@(posedge w_clk) #5 r_pc <= w_npc;
endmodule
```

```
  1 00000000
 56 00000004
156 00000008
256 0000000c
```



RISC-V Control Flow Instructions



- RISC-V conditional branch instructions:

```
beq x4, x5, Lbl  # go to Lbl if x4==x5
bne x4, x5, Lbl  # go to Lbl if x4!=x5
```

```
Ex:    if (i==j) h = i + j;

       bne x4, x5, Lbl1    # if (i!=j) goto Lbl1
       add x6, x4, x5      # h = i + j;
Lbl1:  ...
```

- Instruction Format (B-type):

imm[12]	imm[10:5]	rs2	rs1	funct3	imm[4:1]	imm[11]	opcode	B-type

- How is the branch destination address specified?

RISC-V Control Flow Instructions

- B形式のimmを適切に並べ替えて連結することで得られる12ビットのimm[12:1]の下位に1ビットの0を連結して，13ビットの即値を得る．
- これを符号拡張することで32ビットの即値に変換する．
- このように得られた32ビットの即値にPCの値を加算することで，分岐先のアドレスを得る．

imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU

```
beq  (branch if equal)
bne  (branch if not equal)
blt  (branch if less than)
bge  (branch if greater than or equal)
bltu (branch if less than, unsigned)
bgeu (branch if greater than or equal, unsigned)
```

RISC-V の命令長

The RISC-V Instruction Set Manual
Volume I: Unprivileged ISA
Document Version 20191214-*draft*

Editors: Andrew Waterman[1], Krste Asanović[1,2]
[1]SiFive Inc.,
[2]CS Division, EECS Department, University of California, Berkeley
andrew@sifive.com, krste@berkeley.edu
November 12, 2021

Figure 1.1: RISC-V instruction length encoding. Only the 16-bit and 32-bit encodings are considered frozen at this time.

RISC-V の命令形式と即値

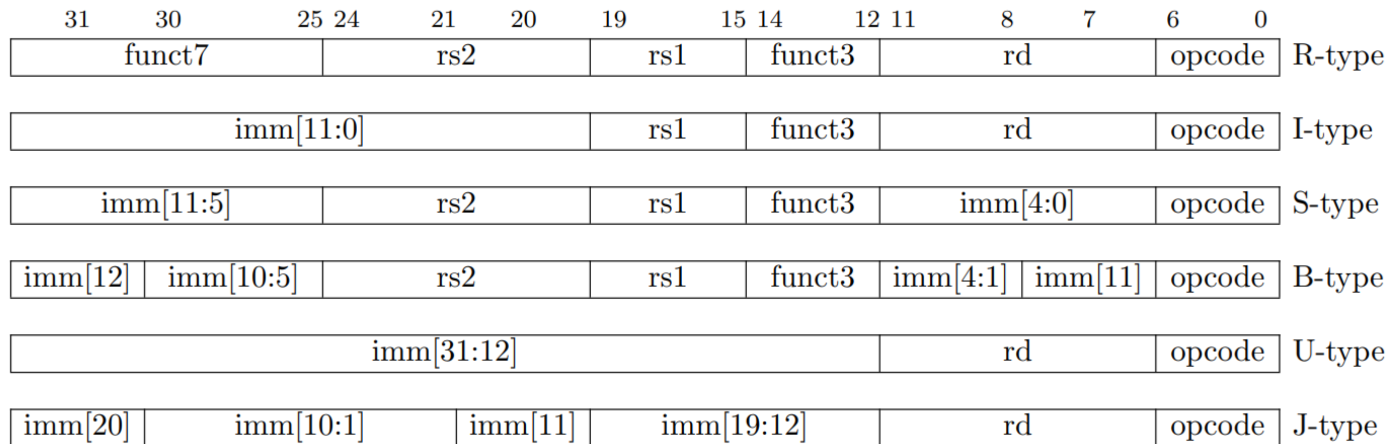
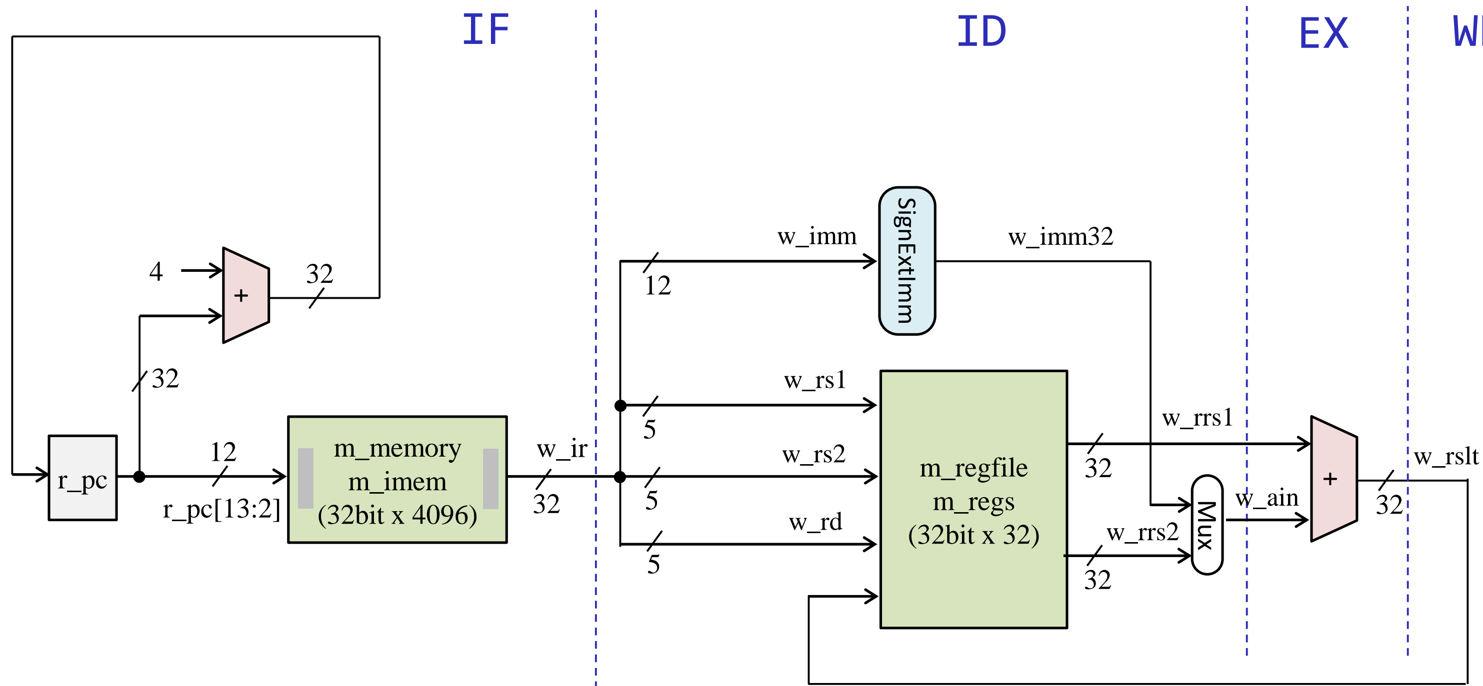


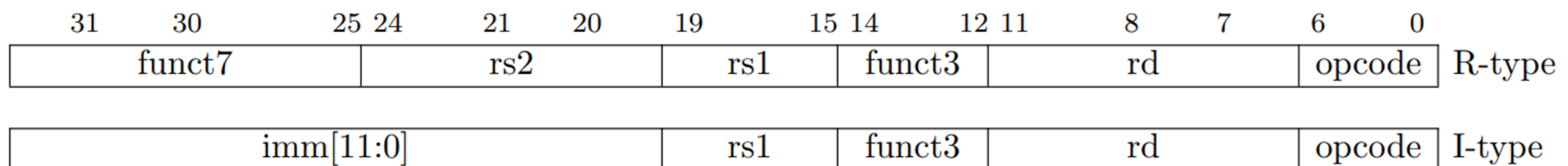
Figure 2.3: RISC-V base instruction formats showing immediate variants.

Figure 2.4: Types of immediate produced by RISC-V instructions. The fields are labeled with the instruction bits used to construct their value. Sign extension always uses inst[31].

m_proc03 add と addi を処理するプロセッサ

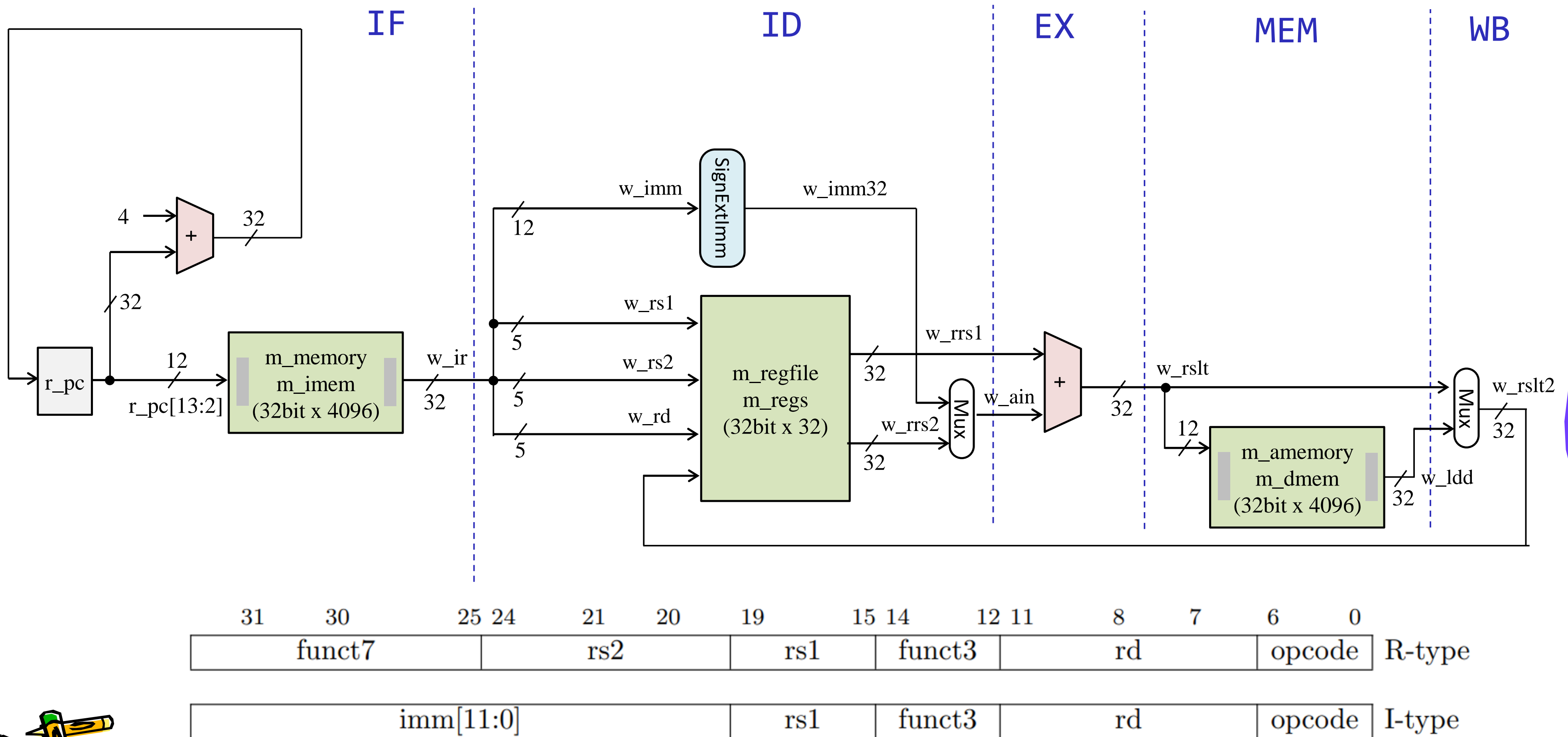
- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), ライトバック(WB) の処理をおこなう加算命令(add, addi)に対応したプロセッサのブロック図





m_proc04 add, addi, lw を処理するプロセッサ

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), メモリアクセス(MEM), ライトバック(WB) の処理をおこなうadd, addi, lw命令に対応したプロセッサ



31	30	25	24	21	20	19	15	14	12	11	8	7	6	0	

funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type

m_proc05 add, addi, lw, sw を処理するプロセッサ

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), メモリアクセス(MEM), ライトバック(WB) の処理をおこなうadd, addi, lw, sw命令に対応したプロセッサ

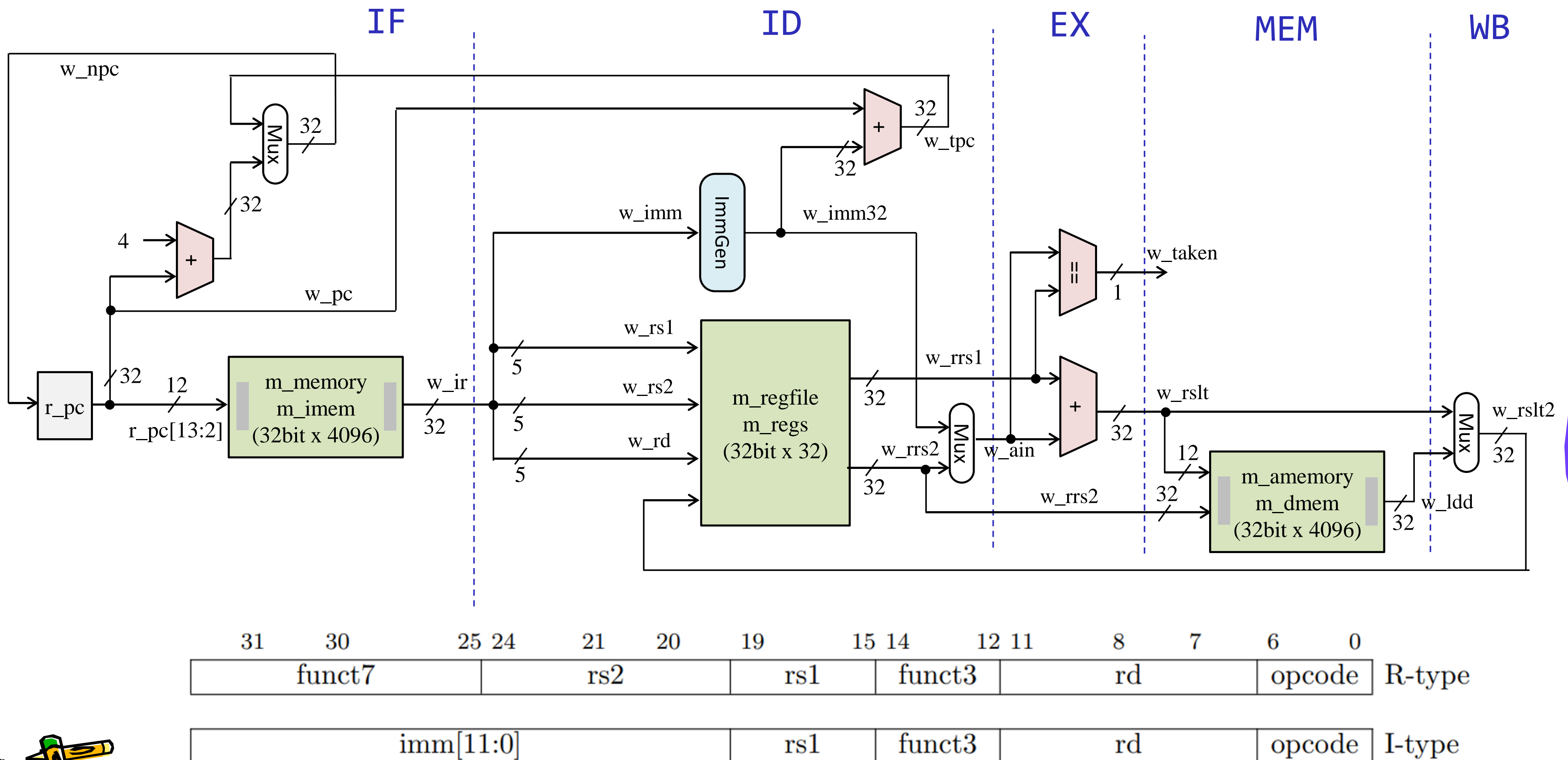
IF ID EX MEM WB

4
+
32
r_pc
12
r_pc[13:2]
m_memory
m_imem
(32bit x 4096)
w_ir
32
w_imm
ImmGen
w_imm32
w_rs1
5
w_rs2
5
w_rd
5
m_regfile
m_regs
(32bit x 32)
32
w_rrs1
w_rrs2
32
Mux
w_ain
+
32
w_rslt
12
32
w_rrs2
m_amemory
m_dmem
(32bit x 4096)
32
w_ldd
Mux
w_rslt2
32

31 30 25	24 21 20	19 15	14 12	11 8 7	6 0	
funct7	rs2	rs1	funct3	rd	opcode	R-type
imm[11:0]		rs1	funct3	rd	opcode	I-type

m_proc06 add, addi, lw, sw, beq を処理するプロセッサ

- 命令フェッチ(IF), デコードとオペランドフェッチ(ID), 実行(EX), メモリアクセス(MEM), ライトバック(WB) の処理をおこなうadd, addi, lw, sw, beq命令に対応したプロセッサ



RISC-V の32ビット基本命令セット RV32I

- LB (load byte)
- LH (load halfword)
- LW (load word)
- LBU (load byte, unsigned)
- LHU (load halfword, unsigned)
- SB (store byte(8-bit))
- SH (store halfword(16-bit))
- SW (store word(32-bit))

RV32I Base Instruction Set

imm[31:12]				rd	0110111	LUI
imm[31:12]				rd	0010111	AUIPC
imm[20|10:1|11|19:12]				rd	1101111	JAL
imm[11:0]		rs1	000	rd	1100111	JALR
imm[12|10:5]	rs2	rs1	000	imm[4:1|11]	1100011	BEQ
imm[12|10:5]	rs2	rs1	001	imm[4:1|11]	1100011	BNE
imm[12|10:5]	rs2	rs1	100	imm[4:1|11]	1100011	BLT
imm[12|10:5]	rs2	rs1	101	imm[4:1|11]	1100011	BGE
imm[12|10:5]	rs2	rs1	110	imm[4:1|11]	1100011	BLTU
imm[12|10:5]	rs2	rs1	111	imm[4:1|11]	1100011	BGEU
imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW
imm[11:0]		rs1	000	rd	0010011	ADDI
imm[11:0]		rs1	010	rd	0010011	SLTI
imm[11:0]		rs1	011	rd	0010011	SLTIU

imm[11:0]		rs1	000	rd	0000011	LB
imm[11:0]		rs1	001	rd	0000011	LH
imm[11:0]		rs1	010	rd	0000011	LW
imm[11:0]		rs1	100	rd	0000011	LBU
imm[11:0]		rs1	101	rd	0000011	LHU
imm[11:5]	rs2	rs1	000	imm[4:0]	0100011	SB
imm[11:5]	rs2	rs1	001	imm[4:0]	0100011	SH
imm[11:5]	rs2	rs1	010	imm[4:0]	0100011	SW

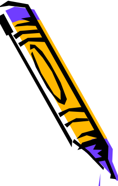
0000000		rs2	rs1	111	rd	0110011	AND
fm	pred	succ	rs1	000	rd	0001111	FENCE
000000000000			00000	000	00000	1110011	ECALL
000000000001			00000	000	00000	1110011	EBREAK

little-endian, big-endian

In a little-endian configuration, multibyte stores write the least-significant register byte at the lowest memory byte address, followed by the other register bytes in ascending order of their significance. Loads similarly transfer the contents of the lesser memory byte addresses to the less-significant register bytes.

In a big-endian configuration, multibyte stores write the most-significant register byte at the lowest memory byte address, followed by the other register bytes in descending order of their significance. Loads similarly transfer the contents of the greater memory byte addresses to the less-significant register bytes.

References



- Computer Logic Design support page
  - https://www.arch.cs.titech.ac.jp/lecture/CLD/
- ACRi Room
  - https://gw.acri.c.titech.ac.jp
- ACRi Blog
  - https://www.acri.c.titech.ac.jp/wordpress/
- 情報工学系計算機室
  - http://www.csc.titech.ac.jp/
- Xilinx Vivado Design Suite
  - https://japan.xilinx.com/products/design-tools/vivado.html
- Digilent Arty A7-35T
  - https://reference.digilentinc.com/reference/programmable-logic/arty-a7/start
- Verilog HDL
  - https://ja.wikipedia.org/wiki/Verilog