情報連携学部

INIAD

INIAD Computer Architecture Practice

第2回 命令のシミュレータを作る
Make an instruction simulator

1. この実習の概要
Outline of this course

このコースの目標 Goal of this course

- CPUのマシンコードを書いて実行する
  Write and execute CPU machine code.
  - CPUの命令セットを確認する
    Check the CPU instruction set.
  - 命令のシミュレータを作る
    Make an instruction simulator.
  - 命令コードの簡易アセンブラを作る
    Make a simple assembler of instruction codes.
  - マシンコードを書いてそのコードの実行をシミュレーションする
    Write machine code and simulate the execution of the code.

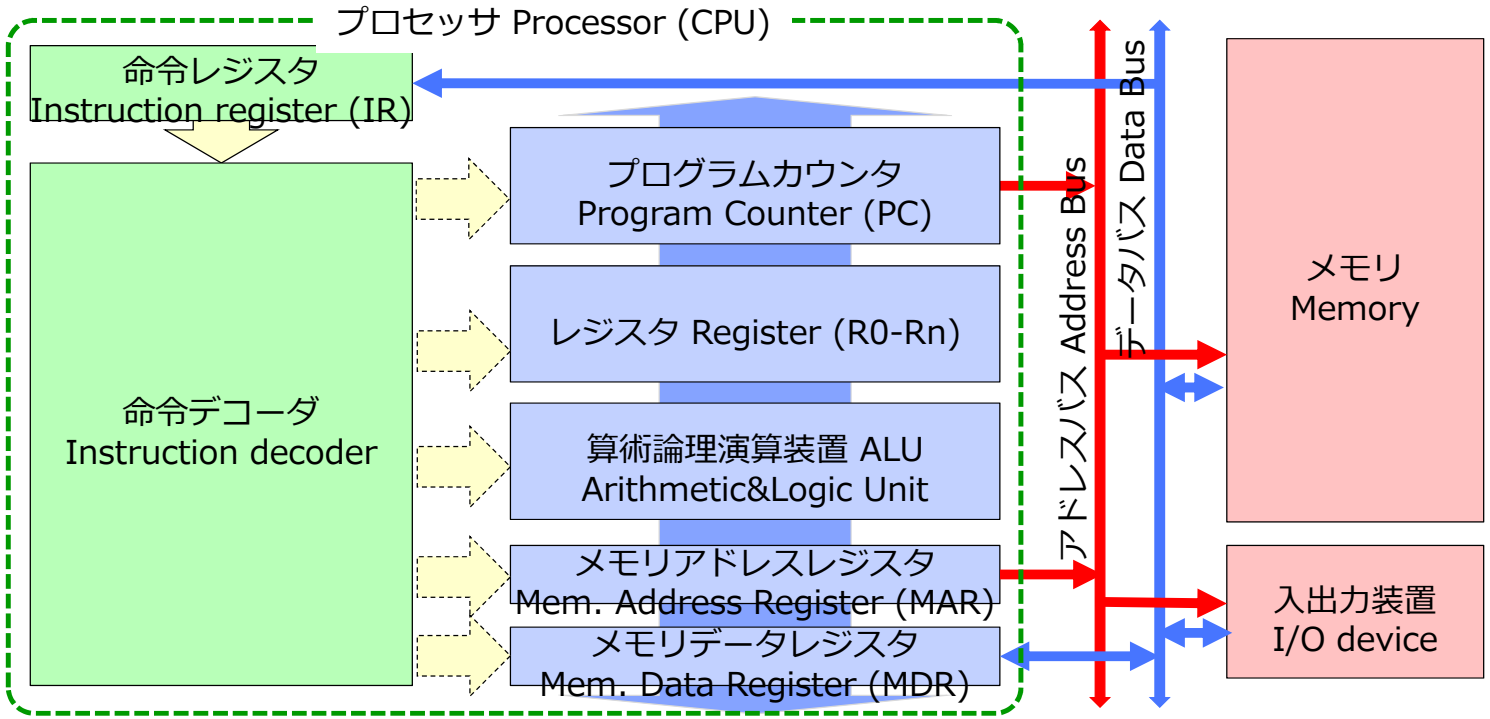
開発対象の命令セット Instruction set to be developed

- M32R命令セット M32R instruction set
  - CA講座のCPU16命令セットの出典
    It is the source of the CPU16 instruction set in CA course.
    - CPU16は16ビットCPUだったが、M32Rは32ビットCPU
      CPU16 was a 16-bit CPU, but M32R is a 32-bit CPU.
  - 参考資料 Reference documents
    - 日本語:
      https://www.renesas.com/jp/ja/doc/products/mpumcu/002/rjj09b0107_32fpusm.pdf
      •この資料の3-112ページ、「②STH Rsrc1,@Rsrc2+ [M32R-FPU拡張ニーモニック]」は、このコースでの開発の対象外です。無視してください。
      •Page 3-112 of this document, “②STH Rsrc1, @ Rsrc2 + [M32R-FPU Extended Mnemonic]”, is not covered by this course. Please ignore.
    - English:
      https://1jzmerc.com/files/pdf/mcu/m3217x-e32fpu.pdf

2. 命令シミュレータとは？
What is an instruction simulator?

CPUによる命令実行 Instruction execution by CPU

- CPUの構成 CPU configuration



CPUによる命令実行 Instruction execution by CPU

- クロックの立ち上がり後 After clock rising edge
  - 更新されたPC値とレジスタデータに基づいて、
  - 命令の読み込み Instruction fetch
  - デコード Decode
  - オペランドレジスタの読み出し Read operand registers
  - ALU演算の実行 Execute ALU operation
  - 次命令アドレスを計算 Calculate next instruction address
- 次のクロック立ち上がりで At the next rise of the clock
  - レジスタへのデータ書き込み Write data to register
    PC値の更新 Update PC value

命令シミュレータとは？

What is an instruction simulator?

- CPUの動作をプログラムでシミュレーションする
  Simulate the CPU operation by a program.
  - 更新されたPC値とレジスタデータに基づいて、
  - 命令の読み込み Instruction fetch
  - デコード Decode
  - オペランドレジスタの読み出し Read operand registers
  - ALU演算の実行 Execute ALU operation
  - 次命令アドレスを計算 Calculate next instruction address
  - レジスタへのデータ書き込み Write data to register
    PC値の更新 Update PC value

命令シミュレータとは？
What is an instruction simulator?

- 正確度に応じた様々なシミュレータ
  Various simulators depending on the accuracy.
  - 回路動作を正しくシミュレートする
    Simulate circuit behavior correctly
  - CPU動作をクロックサイクル毎に正しくシミュレートする
    Simulate CPU operation correctly every clock cycle.
  - 命令動作を正しくシミュレートする
    Simulate instruction behavior correctly.

命令シミュレータとは？
What is an instruction simulator?

- 命令動作のためのリソースは変数として準備
  Resources for instruction operation are prepared as variables.
  - レジスタ Register
  - 条件ビット Condition bit
  - メモリ Memory
  - プログラムカウンタ Program counter

命令シミュレータとは？
What is an instruction simulator?

- 命令動作はプログラムでシミュレーション
  Instruction operation is simulated by a program.
  - 命令フェッチ Instruction fetch
  - デコード Decode
  - 実行 Execute
- 命令コードやデータは2進数として正確に表現される
  Instruction codes and data are accurately represented as binary numbers.

3. 留意点
Points to remember

メモリ中のデータ Data in memory

- ビッグエンディアンとリトルエンディアン
  Big endian and little endian
  - ワード内のバイトのアドレス割り付けには2つの方法がある
    There are two ways to address the bytes in a word.

ビッグエンディアン
Big endian (M32R)

	MSB	バイトアドレス Byte address		LSB
Address 4a	4a	4a+1	4a+2	4a+3
Address 4a+4	4a+4	4a+5	4a+6	4a+7
Address 4a+8	4a+8	4a+9	4a+10	4a+11
Address 4a+12	4a+12	4a+13	4a+14	4a+15

リトルエンディアン
Little endian

Address 4b	4b+3	4b+2	4b+1	4b
Address 4b+4	4b+7	4b+6	4b+5	4b+4
Address 4b+8	4b+11	4b+10	4b+9	4b+8
Address 4b+12	4b+15	4b+14	4b+13	4b+12

メモリ中のデータ Data in memory

- エンディアンは予めCPUによって決められている
  The endian is predetermined by the CPU.
  - ビッグエンディアンのCPUでアドレス4aに32’h01234567を書き込んだとする
    Suppose that 32'h01234567 is written to address 4a by a big endian CPU.
  - アドレス4aのバイト値は8’h01になる

ビッグエンディアン
Big endian

	MSB	バイトアドレス Byte address		LSB
Address 4a	8’h01	8’h23	8’h45	8’h67
Address 4a+4	4a+4	4a+5	4a+6	4a+7
Address 4a+8	4a+8	4a+9	4a+10	4a+11
Address 4a+12	4a+12	4a+13	4a+14	4a+15

メモリ中のデータ Data in memory

- リトルエンディアンのCPUでアドレス4bに32’h01234567を書き込んだとする
  Suppose that 32'h01234567 is written to address 4b by a little endian CPU.
- アドレス4bのバイト値は8’h67になる

ビッグエンディアン
Big endian

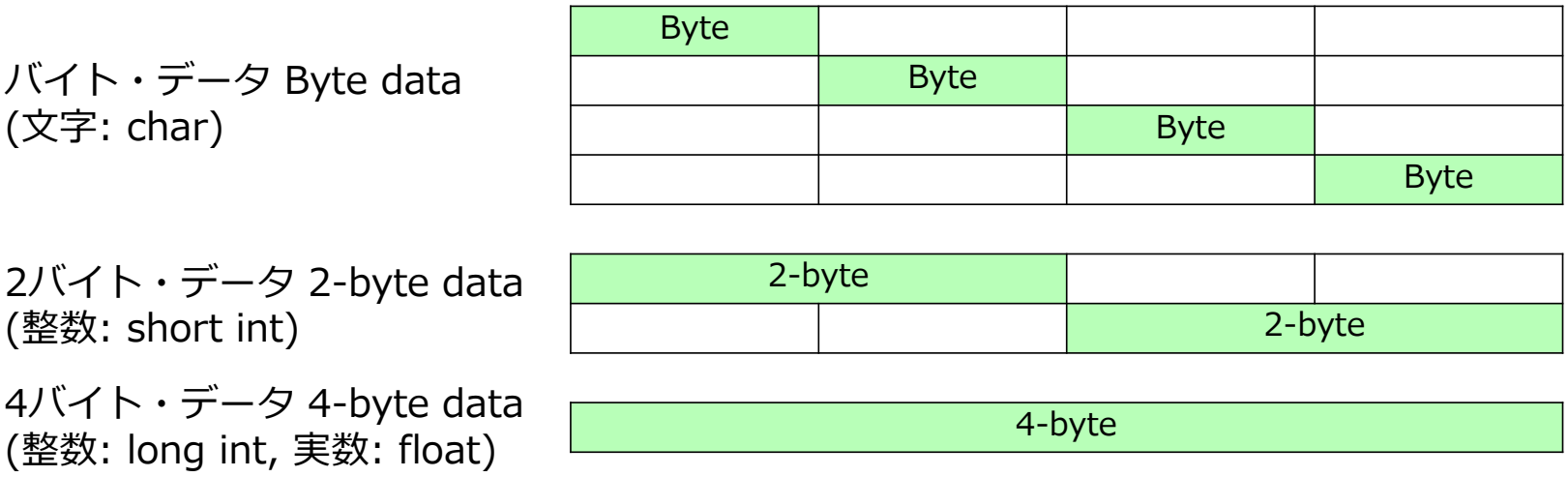
	MSB	バイトアドレス Byte address		LSB
Address 4a	8’h01	8’h23	8’h45	8’h67
Address 4a+4	4a+4	4a+5	4a+6	4a+7
Address 4a+8	4a+8	4a+9	4a+10	4a+11
Address 4a+12	4a+12	4a+13	4a+14	4a+15

リトルエンディアン
Little endian

Address 4b	8’h01	8’h23	8’h45	8’h67
Address 4b+4	4b+7	4b+6	4b+5	4b+4
Address 4b+8	4b+11	4b+10	4b+9	4b+8
Address 4b+12	4b+15	4b+14	4b+13	4b+12

メモリ中のデータ整列 Data alignment in memory

- データサイズ境界をベースにしたデータ配置
  Data placement based on data size boundaries.
  - 1ワード＝4バイトとする Suppose 1 word = 4-byte.



メモリ中のデータ整列 Data alignment in memory

- M32Rではデータはアライメントを取らなければならない
  In M32R, data must be aligned.
- M32Rシミュレータでは、アライメントを取っていないデータもサポートしている
  The M32R simulator also supports unaligned data.

メモリ中のデータ整列 Data alignment in memory

- M32Rシミュレータ M32R simulator
  - ビッグエンディアンのバイトアドレッシング
    Big endian byte addressing.
  - アライメントを取っていないデータもサポート
    Supports unaligned data.

```
#define get2bytes(a) ((mem[a]<<8)|mem[(a)+1])

#define get4bytes(a) ((mem[a]<<24)|(mem[(a)+1]<<16)|(mem[(a)+2]<<8)|mem[(a)+3])

#define put2bytes(a,d) ((mem[a]=(((d)>>8)&0xff)), (mem[(a)+1]=((d)&0xff)))

#define put4bytes(a,d) ((mem[a]=(((d)>>24)&0xff)), (mem[(a)+1]=(((d)>>16)&0xff)), (mem[(a)+2]=(((d)>>8)&0xff)), (mem[(a)+3]=((d)&0xff)))
```

メモリ中の命令整列 Instruction alignment in memory

- M32Rでは命令もアライメントを取らなければならない
  In M32R, instruction must be aligned.
- M32Rシミュレータでは、アライメントを取っていない32-ビット命令もサポートしている
  The M32R simulator also supports unaligned 32-bit instructions.

アライメントが取れた命令
Aligned instructions

16-bit instruction	16-bit instruction
32-bit instruction	

アライメントが取れていない32ビット命令
Unaligned 32-bit instruction

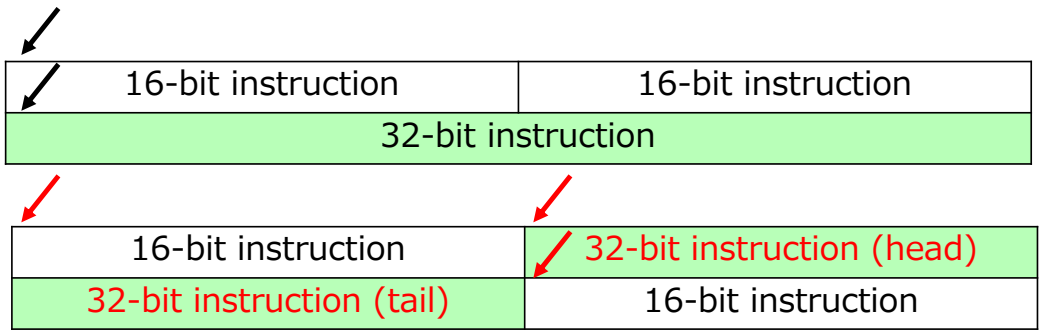
16-bit instruction	32-bit instruction (head)
32-bit instruction (tail)	16-bit instruction

メモリ中の命令整列 Instruction alignment in memory

- M32Rでは、ジャンプ先アドレスを4バイト境界に限定
  In M32R, jump destination address is limited to 4-byte boundary.
- M32Rシミュレータでは、ジャンプ先は2バイト境界も可
  In the M32R simulator, the jump destination can be a 2-byte boundary.

4バイトバウンダリのアドレス
4-byte boundary address

2バイトバウンダリのアドレス
2-byte boundary address



メモリ中の命令整列 Instruction alignment in memory

- M32Rシミュレータ M32R simulator
  - ジャンプ先は2バイト境界も可なので、分岐命令のPC相対ディスプレースメントは1ビットシフト（ジャンプ先アドレスは偶数）
    Since the jump destination can be a 2-byte boundary, the PC relative displacement of the branch instruction is shifted by 1 bit. (The jump destination address is an even number.)
    - 元のM32Rでは、ジャンプ先は4バイト境界なので、分岐命令のPC相対ディスプレースメントは2ビットシフト（ジャンプ先アドレスは4の倍数でなければならない）
      In the original M32R, the jump destination is a 4-byte boundary, so the PC relative displacement of the branch instruction is shifted by 2 bits. (The jump destination address must be a multiple of 4.)

メモリ中の命令整列 Instruction alignment in memory

- M32Rシミュレータ M32R simulator
  - ジャンプ先は2バイト境界も可なので、分岐命令のPC相対ディスプレースメントは1ビットシフト（ジャンプ先アドレスは偶数）
    Since the jump destination can be a 2-byte boundary, the PC relative displacement of the branch instruction is shifted by 1 bit. (The jump destination address is an even number.)

```
case 0x7f00: // BRA
        nextpc = pc+((char)(icode1&0x00ff)<<1);
        break;
case 0xff00: // BRA24
        nextpc = pc+((((char)(icode1&0x00ff)<<16)|(unsigned short)icode2)<<1);
        break;
```

入力ファイル形式 Input file format

- 各行は、「アドレス〈空白〉バイトデータ列」
  Each line is “address <blank> byte data string”.
  - アドレス、バイトデータは16進表記
    Address and byte data are expressed in hexadecimal.
- 例 Example

```
0000 6000 //    LDI R0,0
0002 610a //    LDI R1,10
0004 6201 //    LDI R2,1
0006 6300 //    LDI R3,0
                // L1:
0008 00a1 //    ADD R0,R1
000a 0122 //    SUB R1,R2
000c 0341 //    CMP R3,R1
000e 7cfd  //    BC L1
                // L2:
0010 7f00  //    BRA L2
```

4. 本日の課題
Today’s challenge

“sim32r”開発を完成させる
Complete “sim32r” development

- シミュレータのサンプルコードは次のフォルダに置かれている
  A sample code of the instruction simulator is placed in the following folder:
  - https://drive.google.com/drive/u/1/folders/1G6_Ma4Wy1ld4bjC64KEaXkaKgWDUUJ6q
  - シミュレータコード Simulator code
    sim32r.c
  - 入力サンプル Input sample
    sample1.txt