INIAD Computer Architecture Practice

# 第3回 命令コードの簡易アセンブラを作る

# Make a simple assembler of instruction code

# 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 code.
  - 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
    - English:
      https://1jzmerc.com/files/pdf/mcu/m3217x-e32fpu.pdf

# 2. 前回の課題から
# From the previous challenge

# 前回の課題 The previous challenge

- sim32r開発を完成させる Complete the development of sim32r
  - 提供された元々のバージョン Original version provided:
    - 表 “isa32r.xlsx”の中の、太字で示した命令をシミュレーションすることができる。
      The instructions shown in bold in the table “isa32r.xlsx” can be simulated.
  - やって欲しいこと What you should do:
    - 表 “isa32r.xlsx”に含まれる他の命令のシミュレーションコードを追加して命令シミュレータを完成する。
      Complete the instruction simulator by adding simulation codes for other instructions included in the table “isa32r.xlsx”.

# 前回の課題 The previous challenge

- 命令シミュレータ sim32r.c について About instruction simulator sim32r.c
  - 使い方 Usage:
    - > sim32r “コードファイル名” “シミュレーションする命令数 (省略可能)”
      > sim32r “code file name” “number of instructions to simulate (optional)”
    - コードファイル名：コードを内蔵したファイルの名前。
      Code file name: Name of the file containing the code.
    - シミュレーションする命令数：sim32rが何命令シミュレーションしたら停止するか。この指定を省略すると10になる。
      Number of instructions to simulate: How many instructions sim32r will simulate before stopping. If this specification is omitted, it will be 10.
  - 補足コメント Supplementary comments:
    - R12とR13に、本来のCPUには無い特別な機能を持たせている。
      R12 and R13 have special functions not found in the original CPU.
    - R12の値は、シミュレーション終了時に表示される。シミュレーションの履歴の確認に使える。
      The value of R12 is displayed at the end of the simulation. It can be used to check the simulation history.
    - R13に0以外がセットされると、シミュレーションが終了する。R13=1はシミュレーション成功、それ以外の値はシミュレーション失敗を示す。
      When a value other than 0 is set in R13, the simulation ends. R13 = 1 indicates simulation success, and other values indicate simulation failure.

# 前回の課題 The previous challenge

- サンプルコード asmcode2.txt (sample2.txt) について
  About sample code asmcode2.txt (sample2.txt)
  - このコードは、命令をテストするコードを集めたもの
    This is a collection of code that tests instructions.
  - 例えば、ADD3をテストしているコードは次のとおり
    For example, the code testing ADD3 is:
    - LDI16(R12,0x1); // テスト番号1をR12にセット Set test number 1 to R12.
      LDI(R0,11); // R0に11をセット Set 11 to R0.
      ADD3(R1,R0,22); // ADD3を実行して結果をR1にセット Execute ADD3 and set the result to R1.
      LDI(R2,33); // 結果の期待値33をR2にセット Set expected result value 33 to R2.
      BNE(R1,R2,L_NG);
      // 結果が期待値に合わなければL_NGにジャンプ
      // If the result does not match the expected value, jump to L_NG.
      // 結果が期待値に合えば次のテストに進む
      // If the result meets the expected value, proceed to the next test.

# 2. 簡易アセンブラとは？
# What is a simple assembler?

# アセンブラ Assembler

- 1行が1命令に対応
  One line corresponds to one instruction.
  - アドレス Address
  - 2進コード Binary code
  - ニモニック記述 Mnemonic description
    - ニモニック記述の為のプログラミング言語を「アセンブリ言語」と呼ぶ
      The programming language for mnemonic description is called “assembly language”.
    - アセンブリ言語で書かれた記述をマシンコードに変換するツールを「アセンブラ」と呼ぶ
      A tool that converts a description written in assembly language into machine code is called an “assembler”.

```
00000000 <_factorial>:
int factorial(int n)
{
   0:   55                      push   %ebp
   1:   89 e5                   mov    %esp,%ebp
   3:   83 ec 18                sub    $0x18,%esp
 if (n == 1)
   6:   83 7d 08 01             cmpl   $0x1,0x8(%ebp)
   a:   75 07                   jne    13 <_factorial+0x13>
    return 1;
   c:   b8 01 00 00 00          mov    $0x1,%eax
  11:   eb 12                   jmp    25 <_factorial+0x25>
 return n*factorial(n - 1);
  13:   8b 45 08                mov    0x8(%ebp),%eax
  16:   83 e8 01                sub    $0x1,%eax
  19:   89 04 24                mov    %eax,(%esp)
  1c:   e8 df ff ff ff          call   0 <_factorial>
  21:   0f af 45 08             imul   0x8(%ebp),%eax
}
  25:   c9                      leave
  26:   c3                      ret
  27:   90                      nop
```

# ここでのアセンブリ言語 The assembly language here

- C言語の関数コールとステートメントがアセンブリ言語になる
  C function calls and statements become assembly language.
  - 大文字の名前の関数は命令コードを生成
    A function with an uppercase name generates an instruction code.
  - 小文字の名前の関数はそれ以外のコマンド
    Lowercase name functions are other commands.
  - 代入文でラベルを宣言
    Declare a label using an assignment statement.

```
        org(0);
        LDI(R12,0);
        LDI(R13,0);

        LDI16(R12,0x1);
        LDI(R0,11);
        ADD3(R1,R0,22); // ADD3
        LDI(R2,33);
        BNE(R1,R2,L_NG);

        LDI16(R12,0x4);
        LDI(R3,5);
        BEQZ(R3,L_NG); // BEQZ
        LDI(R4,0);
        BEQZ(R4,L1);
        BRA24(L_NG);
L1 = addr;
```

# 簡易アセンブラ A simple assembler, asm32r

- 使い方 Usage:
  - アセンブリ言語で書かれたソースコードをファイル”asmcode.txt”に用意する
    Prepare the source code written in assembly language in the file “asmcode.txt”.
  - “asmcode.txt”を”asm32r.c”と同じフォルダに置く
    Place “asmcode.txt” in the same folder as “asm32r.c”.
  - Gccで”asm32r.c”をコンパイルする
    Compile “asm32r.c” with Gcc.
    - gcc asm32r.c –o asm32r
  - “asm32r”を実行すると生成されたコードが表示される
    When “asm32r” is executed, the generated code is displayed.
  - 生成されたコードをファイルにセーブする
    Save the generated code to a file.
    - asm32r > code.txt

“asm32r.c” main function:

```
int main(int ac, char **av)
{
    int L_OK, L_NG,
        L1, L2, L3, L4, L5, L6, L7, L8, L9, L10,
        L11, L12, L13, L14, L15, L16, L17, L18, L19, L20;

switch( ac ) {
    case 1: // 命令名を出力 Output instruction name.
        option = 0;
        break;
    case 2:
        if( strcmp(av[1],“-”) == 0 ) // 命令名無し No instruction name
            option = 1;
        else { // 使い方エラー Usage error.
            printf("Usage: %s [-]¥n", av[0]);
            return -1;
        }
        break;
    default: // 使い方エラー Usage error.
        printf("Usage: %s [-]¥n", av[0]);
        return -1;
    }

    for( pass = 0; pass < 2; pass++ ) {
        addr = 0;
#include "asmcode.txt"
    }
    return 0;
}
```

# アセンブラのプログラミング Assembler programming

- 変数 Variables
  - addr: 生成コードを置くアドレス Address to place generated code.
  - pass: 処理パスの制御変数 Processing pass control variable.

# アセンブラのプログラミング Assembler programming

- 関数 Functions
  - rr(): ADD型のコード生成 ADD type code generation.
  - ri(): ADDI型のコード生成 ADDI type code generation.
  - bc(): BC型のコード生成 BC type code generation.
  - bc24(): BC24型のコード生成 BC24 type code generation.
  - beq(): BEQ型のコード生成 BEQ type code generation.
  - rrd(): LDR型のコード生成 LDR type code generation.
  - ld24(): LD24型のコード生成 LD24 type code generation.
  - ri5(): SLLI型のコード生成 SLLI type code generation.

# アセンブラのプログラミング Assembler programming

- マクロ関数 Macro functions
  - org(): アドレスをセット Set address.
  - byte(): 8ビットデータを配置 Place 8-bit data.
  - half(): 16ビットデータを配置 Place 16-bit data.
  - word(): 32ビットデータを配置 Place 32-bit data.

# アセンブラのプログラミング Assembler programming

- 2パスのアセンブラ 2-pass assembler
  - Pass 1: ラベルのアドレス値を決定 Determine the address value of the label.
  - Pass 2: コードを生成する Generate code.
  - 前に定義されたラベルを後でジャンプ命令が参照する場合（後方参照）は、1パスで処理できるが、ジャンプ命令が参照しているラベルが更に後で定義されている場合（前方参照）は、2パス必要になる。
    If a jump instruction refers to a previously defined label (backward reference), it can be processed in one pass, but if a label referenced by a jump instruction is further defined (forward reference) 2 passes are required.

# アセンブラのプログラミング Assembler programming

- 出力形式 Output format
  - 各行は「アドレス〈空白〉バイト列 // 命令名」
    Each line is “address byte-string // instruction-name”.
  - アドレス、バイトは16進表記
    Address and byte are expressed in hexadecimal.

```
0000 6c00 // LDI
0002 6d00 // LDI
0004 9cf00001 // LDI16
0008 600b // LDI
000a 81a00016 // ADD3
000e 6221 // LDI
0010 b112007a // BNE
0014 9cf00002 // LDI16
0018 90f000ff // LDI16
001c 81c00ff0 // AND3
…
```

# 3. 本日の課題
# Today’s challenge

# “asm32r”開発を完成させる
# Complete “asm32r” development

- “asm32r”のコードは次のフォルダに置かれている
  “asm32r” code is placed in the following folder:
  - https://drive.google.com/drive/u/1/folders/1G6_Ma4Wy1ld4bjC64KEaXkaKgWDUUJ6q
  - アセンブラコード Assembler code
    asm32r.c
  - アセンブリ言語のサンプルコード An assembly language sample code
    asmcode3.txt
    - 生成されたコードを確認するためだけのもので、機能的な意味はない。
      It is only for checking the generated code and has no functional meaning.