マイクロプロセッサ演習

2004 年度

第 7 回

1 MIPS のアセンブリ言語の機械語への翻訳

今回の演習では、MIPS のアセンブリ言語の機械語への翻訳を取り扱う。もちろん、機械語そのものを覚えるのが目的ではなく、コンピュータがハードウェアとして働く仕組みを理解するための足掛かりとすることを目指している。今回の演習の内容は、教科書の第五章 (データパスと制御) を理解するためには必須と言えるだろう。

今までの演習で、MIPS CPU 上で動作するソフトウェアが `add`, `lw`, `sw` などの**命令** (instruction) の系列として実現されることをみて来た。これらの命令はアセンブラによって機械語に翻訳される。機械語とは、命令を数値で表現したものである。

MIPS R2000 CPU においては、一命令は全て 32 ビットで表現される。今まで見た通り、MIPS においてはレジスタやデータの 1 語のデータ長は 32 ビット (4 バイト) であったから、1 命令は全て 1 語で表されることになる。さらにメモリアドレスも 32 ビットで表現されている。このように MIPS R2000 CPU は非常に規則的な命令体系および構造をしているのが特徴である。

一方、Intel の 80x86 系 CPU はその歴史背景から非常に複雑な構造をしている。例えばレジスタのサイズが 16 ビットのものとと 32 ビットのものが混在している、などである。

とはいえ、MIPS R2000 CPU の構造を理解することは Intel や PowerPC 等、他の CPU を理解する助けになるだろう。

2 [SPIM] R 形式 (レジスタ・アドレシング)

`and`, `sub`, `slt` 等の命令は、図 1 に示すような R 形式で表される。この形式では 32 ビットを 6 つに分割し、以下のように命令コードやオペランドに割り当てる。

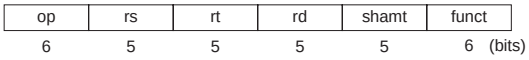

図 1: R 形式

- **op:** 命令操作コード。 6 ビット。
- **rs:** 第 1 のソースオペランドのレジスタ。 5 ビット。
- **rt:** 第 2 のソースオペランドのレジスタ。 5 ビット。
- **rd:** デスティネーション・オペランド (結果出力) のレジスタ。5 ビット。
- **shamt:** シフト量。これまでの範囲では扱われず、0 である。 5 ビット。
- **funct:** 機能。命令操作フィールド (op) のバリエーションを表す。6 ビット。

このように、命令の数字列をビット数の分割法で分類したものを**命令形式**と呼ぶ。すでに見たように、図 1 は R 形式である。また、引数の表現形式 (**アドレシング**) で分類することもできる。その場合、この形式は引数が全てレジスタであるため、レジスタ・アドレシングと呼ばれる。

なお、各レジスタに 5 ビットが割り当てられているが、$2^5 = 32$ であるため、32 個のレジスタを表すのに必要十分であることに注意しよう。

例として「add $t0, $s0, $s1」という命令を機械語に変換した様子を図 2 に示す。足される数 $s0, $s1 と結果 $t0 の順番が変わっているが、基本的に命令が素直に数字に変換されていることがわかるだ

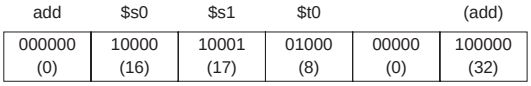

図 2: 命令「add $t0, $s0, $s1」を機械語に変換した様子。かっこ内の数字は 2 進数を 10 進数に変換したもの。

ろう。なお、レジスタの数字での表現は付録 A を参照。

また、このように命令を数値で表すと「(有限な) レジスタ」、「(非常に大きな) メモリ」を持った現在のコンピュータが、チューリング・マシンの良い近似になっていることがわかる (付録 B)。

問題

1. 演習用の Web ページより data07.zip をダウンロードせよ。中の AddSub.asm を実行して中身を理解せよ。

2. このプログラム中に、図 2 と同じ命令 (add ···) がある。この命令に対応する機械語が、確かに図 2 と同じになることを確認せよ。(ヒント：Windows 版 SPIM のテキストウインドウで、上記命令に対応する機械語が 0x02114020 であることを見つける (main はメモリアドレス 0x00400020 から始まることに注意)。この 16 進数の機械語を 2 進数に変換し (付録 C 参照)、R 形式のビットごとに区切ってやればよい。

3. 上記プログラムには引き算の命令「sub $t1, $s0, $s1」も含まれている。この命令の機械語を 2 進数で求め、命令コード sub に対応する op フィールドと funct フィールドの値を求めよ。また、教科書などで命令 sub の op コードを調べて正しい結果が得られているか確かめよ。

3 [SPIM] I 形式

32 ビットの命令が、6 · 5 · 5 · 16 に分かれている形式を I 形式とよぶ。この形式は、アドレシングによって以下の 3 つに分類される。

[**ベース相対アドレシング**]

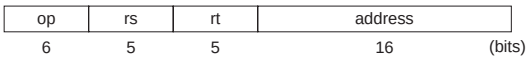

図 3: I 形式 (ベース相対アドレシング)

`lw`, `sw` などが図 3 に示す I 形式である。例として命令「lw $t0, 4($s0)」の 機械語による表現を図 4 に示した。(やはり $t0 と $s0 の順番が変わる。とはいえ、この順番の変化は重要ではない)

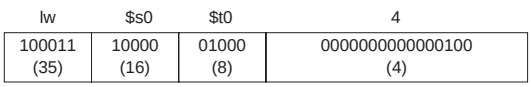

図 4: 命令「lw $t0, 4($s0)」の機械語による表現。

この時、対応するメモリのアドレスは、レジスタ $s0 に格納されたメモリアドレス (ベース・アドレス) に “address フィールド” に格納された値を加えることによって表現される。それゆえ、このアドレシング方式をベース相対アドレシングとよぶ。

この時、アドレスフィールドは 16 ビットあるため、ベースアドレスから 2^{16} バイトの範囲の値をロード・ストアできることに注意しよう。(実際には、正負の両方があるため、$\pm 2^{15} = \pm 32768$ の範囲である)。

[**即値アドレシング**]

命令「addi $t0, $s0, 4」はレジスタ $s0 と即値 4 の和を計算する (**即値** (immediate) とは定数のことである)。この命令は機械語では図 5 に示す即値アドレシングで表現される。即値には 16 ビットが割り

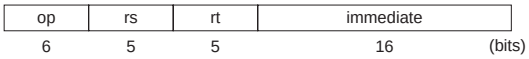

図 5: I 形式 (即値アドレシング)

当てられるため、表現できる即値は $\pm 2^{15} = \pm 32768$ の範囲の値である (正確には、**2 の補数表現**した場合、$-2^{15} \sim 2^{15} - 1$ の即値を表現できる)。定数を含んだ計算は、定数が小さい場合に多く用いられるため、この範囲で十分である。(大きい数を含んだ計算は、R 形式を用いればレジスタの 32 ビットをフルに使える)

[**PC 相対アドレシング**]

図 6 は、レジスタ $t0 と $t1 の値が等しければ $t2 に 1 を格納し、異なれば 0 を格納するプログ

ラムである。(説明のためのプログラムであり、機能的には大した意味はない)

```
0x400028 | bne $t0,$t1,ELSE
0x40002c |    li $t2,1
0x400030 |    j ENDIF
         | ELSE:
0x400034 |    li $t2,0
         | ENDIF:
0x400038 |    jr $ra
```

図 6: レジスタ $t0 と $t1 の値が等しければ $t2 に 1 を格納し、異なれば 0 を格納するプログラム。左の数字は、プログラムが格納されているメモリアドレスである。

この中で、「bne $t0,$t1,ELSE」 という命令は、図 7 に示すような PC 相対アドレシングで表現される。

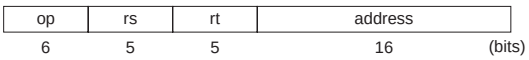

図 7: I 形式 (PC 相対アドレシング)

この「bne $t0,$t1,ELSE」という命令を機械語に変換する際に問題になるのは、ラベル ELSE のアドレス (ここでは 0x400034) をどのように表現するか、ということである。

もし、ELSE を絶対的なメモリアドレス (すなわち 0x400034) で表現するとすると、「I形式」の “address フィールド” は 16 ビットしか割り当てられていないため、プログラム全体は 2^{16} バイトで表されねばならなくなる。今日ではこれはプログラムのサイズとしてはあまりに小さい。

そこで ELSE は、プログラムの現在の命令の位置 (Program Counter: PC) からの相対位置で表すことにする。これが **PC 相対アドレシング**である。

具体的には、命令「bne $t0,$t1,ELSE」において、プログラムの現在の位置は、PC = 0x400028 である。ELSE の絶対的な位置は 0x400034 であるが、PC からの相対的な位置で表すと、12 バイト先である。さらに、バイトでなくワード数 (語数) で表すと、ELSE は bne 命令から 3 ワード先、ということになる。

この “3” を命令「bne $t0,$t1,ELSE」の “ELSE” のコーディングとしてと採用するのが、PC 相対アドレシングである。これは、分岐先 (ここでは ELSE) と 分岐元 (ここでは bne) の間が $\pm 2^{15}$ 語である限りにおいてうまく行く。この制限は、先程の「プログラムのサイズが 2^{16} バイト」という制限よりは現実的であり、実際に採用されている。

なお、現実の MIPS CPU では、分岐先を PC (ここでは 0x400028) からの相対位置ではなく、PC+4 (0x40002c) からの相対位置で表すが (教科書や授業参照)、上で見たように、MIPS シミュレータ SPIM では PC を起点として計算しているようである。

問題

1. data07.zip に含まれていた IfElse.asm を実行する。図 6 と同じプログラムである。このうち「bne $t0,$t1,ELSE」の機械語を調べ、PC 相対アドレシングが行なわれていることを確かめよ。

4 [SPIM] J 形式 (疑似直接アドレシング)

図 6 の命令「j ENDIF」は図 8 の J 形式 (疑似直接アドレシング) で表される。

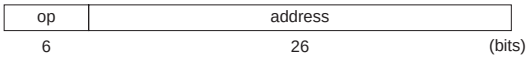

図 8: J 形式

ここで、“address フィールド” にはメモリアドレスを 4 で割った数字が直接代入される。(すなわち、“address フィールド” には「メモリの先頭から何ワードずれているか」が入る。

問題

1. 前節と同じく IfElse.asm を実行する。命令「j ENDIF」の機械語を調べ、J 形式の表現を確認せよ。

A [補足] レジスタの数値による表現

32 個のレジスタを数値で表現すると、以下のようになる。これは Windows SPIM のレジスタ・ウィンドウでも確認できる。

これらは、例えば $t0 であれば $8 などとも記述できるのであるが、人間が理解しやいように名前 ($t0 など) がついているのである。

レジスタ名	番号	機能
$zero	0	定数ゼロ
$at	1	アセンブラ用
$v0 ∼ $v1	2∼3	関数の戻り値
$a0 ∼ $a3	4∼7	関数の引数
$t0 ∼ $t7	8∼15	一時的な変数
$s0 ∼ $s7	16∼23	変数
$t8 ∼ $t9	24∼25	予備の一時的変数
$k0 ∼ $k1	26∼27	OS 用
$gp	28	グローバル・ポインタ
$sp	29	スタック・ポインタ
$fp	30	フレーム・ポインタ
$ra	31	戻りアドレス

B [補足] チューリング・マシン

1930 年代に計算可能性の概念を導入するためにチューリングによって考案された抽象的な機械。

- **チューリング・マシン：** 有限個数の「内部状態」をもち、大きさに制限のない入力 (「テープ」) を受取るような機械。マシンはテープの内容を読むことができ、演算操作の一部としてテープを前後に動かすこともできる。さらに、テープに新しいマークを書き込むことや、消すこともできる。

この数学的に定義された (上の説明は全然数学的ではないですが…) 機械で実現可能な演算のことを「アルゴリズム」と呼ぶ。

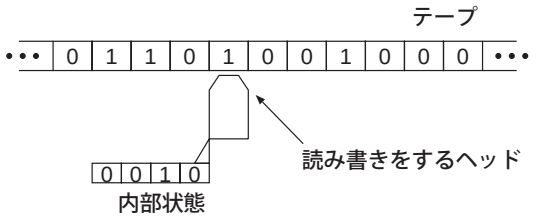

図 9: チューリング・マシン

このチューリングマシンに何ができるか (計算可能性の問題) を考えて行くと「あるクラスの数学的問題の全てに答えるアルゴリズムは存在しない」という結論が導き出される。(チューリング・マシンの停止問題)

なお、レジスタ、メモリをそれぞれチューリング・マシンの「内部状態」および「テープ」に対応させて考えると、現在のコンピュータがこのチューリング・マシンの非常に良い近似になっていることがわかる。

C [補足] 2 進数と 16 進数の変換

教科書 4.1 章に書かれているように、2 進数と 16 進数を相互に変換するには図 10 のような手続きに従えばよい。

まず、16 進数から 2 進数へ変換するには 16 進数の数字 (0,1, ··· e,f) を一桁ずつ 4 ビットの 2 進数に変換し、それを順に並べてゆけば良い。

逆に 2 進数から 16 進数へ変換するには、2 進数の数字を 4 ビットづつのグループに区切り、各グループごとに 16 進数に変換して並べてゆけば良い。

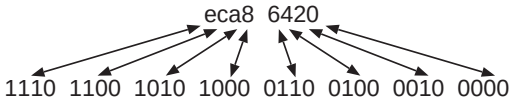

図 10: 2 進数と 16 進数の相互変換