# マイクロプロセッサ演習

2004 年

第 3 回

## 1　今回の演習に必要な予備知識

[**レジスタ**]

MIPS CPU にはデータを保持する機構として 32 のレジスタがある。レジスタの 1 ワードは 4 バイト (32 ビット) である。32 個のレジスタの内 $t0 ··· $t7 は一時格納用に、$s0 ··· $s7 は変数格納用に主に用いられる。残りのレジスタについては随時紹介してゆく。

[**ロードとストアの概念**]

ロードとストアの概念図を図 1 に示した。メモリからレジスタに値をロードする際に命令 `lw` を、レジスタからメモリに値をストアする際に命令 `sw` を用いていることがわかる。これを C 言語に対応させると図 2 のような記述となるだろう。つまり、

```
int a=5;
int b=a;
```

図 2: 図 1 に対応する C 言語記述。

メモリアドレス 0x10010000 に対応する変数 a の値をメモリアドレス 0x10010004 に対応する変数 b にコピーしている.

なお、MIPS CPU においてはメモリの 1 ワードは 4 バイトであり、そのため、メモリアドレスは 4 刻みで増えて行く。

[**MIPS CPU のアセンブリ言語の命令**]

- `add` $\$R_i$, $\$R_j$, $\$R_k$
  $\$R_i = \$R_j + \$R_k$ (前回の復習)
- `lw` $\$R_i$, N($\$R_j$)
  $\$R_j$ + N のアドレスの内容をレジスタ $\$R_i$ にロード。(N は 4 の倍数。負の数でも良い)
  C 言語風に書くと $\$R_i = \text{Memory}[\$R_j + \text{N}]$
- `sw` $\$R_i$, N($\$R_j$)
  レジスタ $\$R_i$ の内容を $\$R_j$ + N のアドレスにストア。(N は 4 の倍数。負の数でも良い)
  C 言語風に書くと $\text{Memory}[\$R_j + \text{N}] = \$R_i$

## 2　[SPIM] 3 つの数の足し算

演習用の Web サイトより、data03.zip をダウンロードしよう。この中に、今日の演習で使うプログラムの雛形が入っている。

圧縮ファイルに Add3.asm というファイルが含まれている。このファイルにはプログラムの骨組みしか記述されておらず、プログラム本体は書かれていない。このファイルに以下の内容を MIPS CPU のアセンブリ言語で記述し、シミュレータで結果を確認せよ。なお、本問題は前回の “Add.asm” の復習である。

「レジスタ $s0 〜 $s2 にそれぞれ 1, 2, 3 という値が既に格納されている。レジスタ $s3 に $s0 + $s1 + $s2 の結果を格納せよ」

## 3　[SPIM] ロードとストア (1)

LoadStore1.asm というファイルがある。このファイルは完成されたプログラムなので、テキストエディタ (秀丸) で中身を確認した後、Windows 版の spim (pcspim.exe) で結果を確認してみよう。

1. レジスタ $t0 および $t1 にはどのような値が格納されているか？

2. $t0 と $t1 にはメモリアドレスが格納されている。対応するメモリには、どんな値が格納されているか？

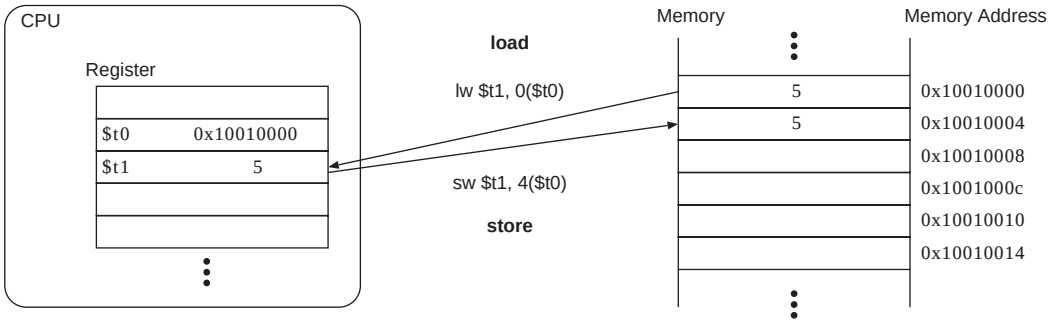


図 1: ロードとストアの概念図

## 4 [SPIM] ロードとストア (2)

LoadStore2.asm というファイルがある。このファイルは完成されたプログラムなので、テキストエディタ (秀丸) で中身を確認した後、Windows 版の spim (pcspim.exe) で結果を確認してみよう。

1. レジスタ $t0 には何が格納されているか？
2. array[0], array[1], array[2] の値はどこに表示されているか？

## 5 [SPIM] ロードとストア (3)

LoadStore3.asm というファイルがある。このファイルにはプログラムの骨組みしか記述されておらず、プログラム本体は書かれていない。このファイルに以下の内容を MIPS CPU のアセンブリ言語で記述し、シミュレータで結果を確認せよ。

1. array[0]、array[1]、array[2] のデータを変数 $s0, $s1, $s2 にロード
2. $s0 + $s1 + $s2 の結果を $s3 に格納
3. $s3 の値を array[3] に格納

## A [補足] C 言語との対応

本演習ではアセンブリ言語の記述を通してコンピュータの動作を理解するが、受講者の中にはアセンブリ言語と C 言語との対応について知りたい人もいるだろう。本資料では、「付録による補足」という形で C 言語との対応を必要に応じて解説してゆくことにする。

まず「ロードとストア (2)」で扱ったアセンブリ言語プログラムを C 言語に対応させると、以下のようになるだろう。

```
int array[3]; // 外部変数で配列を定義

int main(void){
    array[0] = 1; // 配列の要素に値を代入
    array[1] = 2;
    array[2] = 3;
    return(0); // プログラムの終了
}
```

図 3: LoadStore2.asm に対応する C 言語記述。

この C 言語プログラムでは要素数が 3 の配列 array[3] を外部変数として定義し、main 関数内で array の各要素に値を代入している。このとき、array[3] はメモリ上の**静的領域**に確保される。

このように、C 言語によるプログラミングはメモリ上における変数の配置に注意しながら進められる。表現を変えると、C 言語 (やその他の高級言語) によるプログラミングではメモリの状態だけに気をくばっていれば良いことになる。しかし、実際のコンピュータでは本資料で扱ったようにメモリと CPU が常に相互作用して値をやりとりしながら計算が進行してゆくのである。

なお、「ロードとストア (3)」では配列 array[4] を定義し、array[3] = array[0] + array[1] + array[2]; を計算するプログラムの記述になっていることに注意しておく。

なお、array[3] などの変数を main 関数内で宣言するには**スタック**の知識が必要であり、今後扱うことになるだろう。(図 2 ではその点をごまかして記述した)