システムプログラミング

第1回　イントロダクション

乃村能成 （のむらよしなり）
連絡先： 4号館 206号室

本演習で学んで欲しいこと

1. MIPSのアセンブリ言語
2. メモリの扱い(スタック，セグメント)
3. 入出力の扱い，文字と文字列
4. 手続き呼出しの仕組み
5. Cコンパイラとアセンブラの連携
   - レジスタ使用規則
   - 引数渡し規約(可変引数)
   - スタックフレーム．auto と static の違い
   - 配列，ポインタとアセンブリ言語

本演習でやること

1. 文字の入出力を行う関数(ライブラリ)作成
   - SPIMというMIPSエミュレータを使います
   - アセンブラで直接ハードウェアを操作します
   - システムコールを利用します
2. 1のライブラリをC言語から呼べるようにする
   - いわゆるシステムコールライブラリ相当
3. ライブラリを使用してprintf,gets相当を作成
   - いわゆる libc 相当
4. printf等を使って応用プログラム作成
   - これまでの演習相当

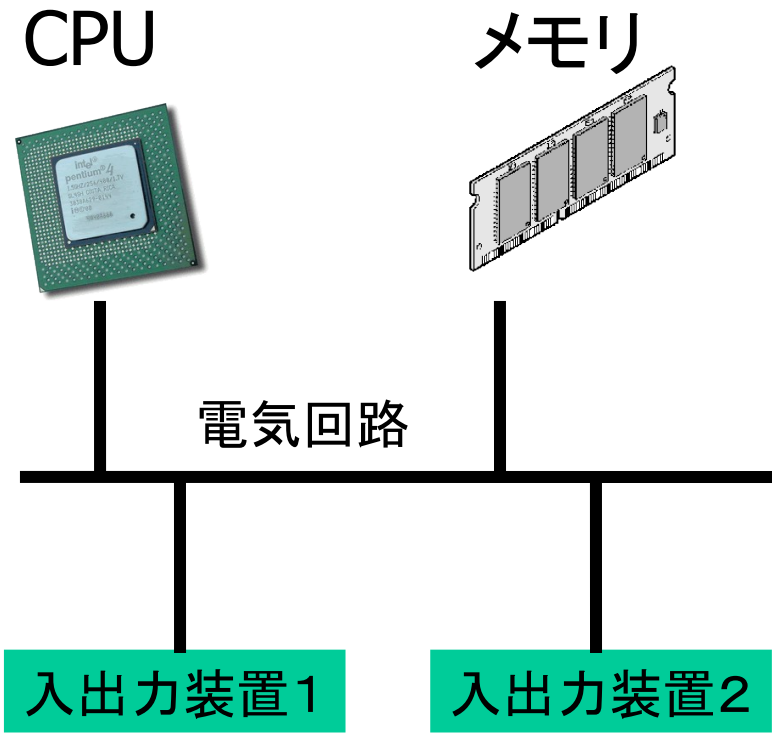
はじめに　―　CPUと機械語

コンピューター【computer】(コンピュータとも)
計算機。主に電子計算機をいう。(広辞苑）

？
＝

コンピュータとは

＝



3つの要素だけを考えればどちらも同じ数を扱う装置

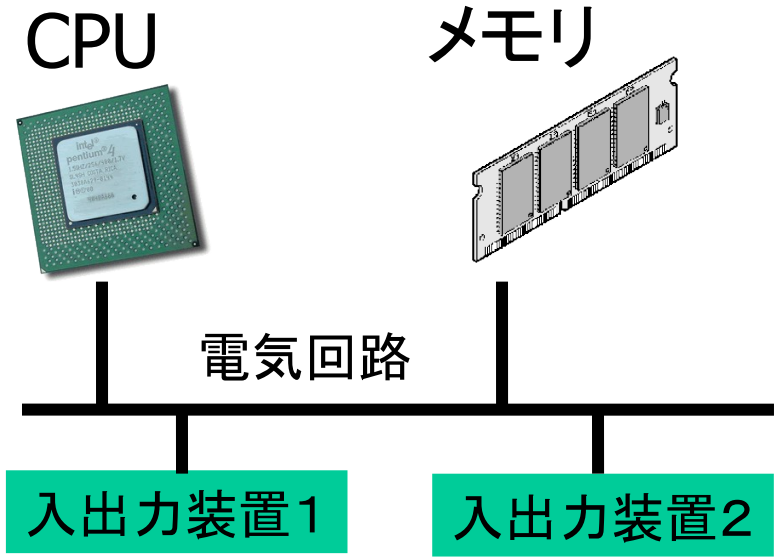
コンピュータの構成要素(CPU)

CPU (Central Processing Unit):

コンピュータの心臓部.
メモリや入出力装置に接続

仕事: メモリや入出力装置と
数の受け渡し.
数を使った計算.

例: 入出力装置1から
数を受け取って,
計算を加えた結果を
メモリに送る.



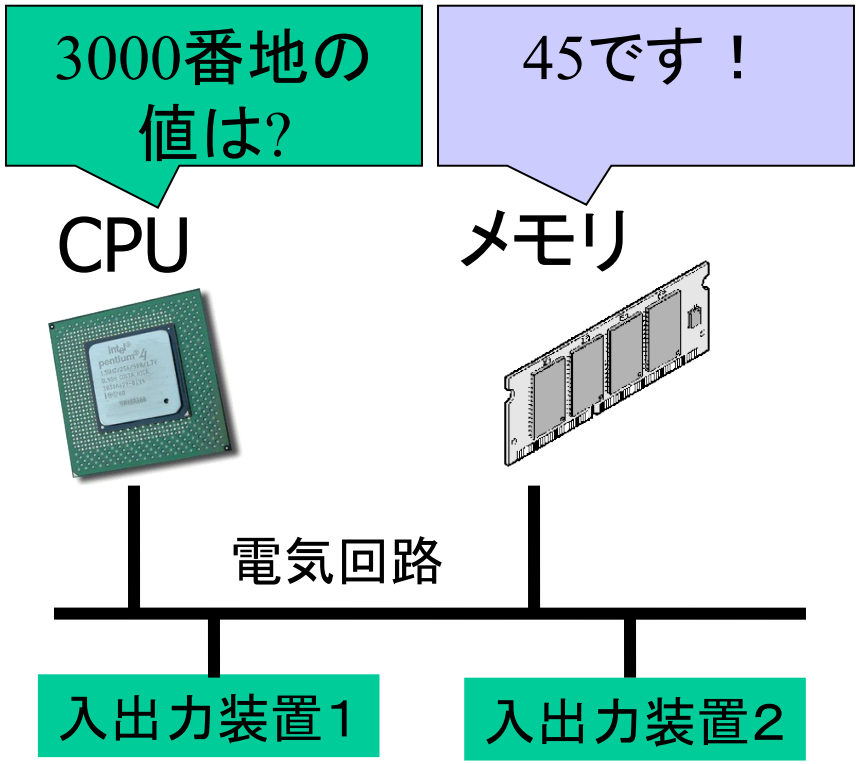
コンピュータの構成要素(メモリ)

メモリ (Memory):

番地で区切られた場所に,
多数のデータを記憶する

仕事: CPUから受取った
数を記憶する.
CPUからの求めに
応じて, 数を読出す.

例: 8000番地に5を記憶.
3000番地の数を
CPUに渡す.



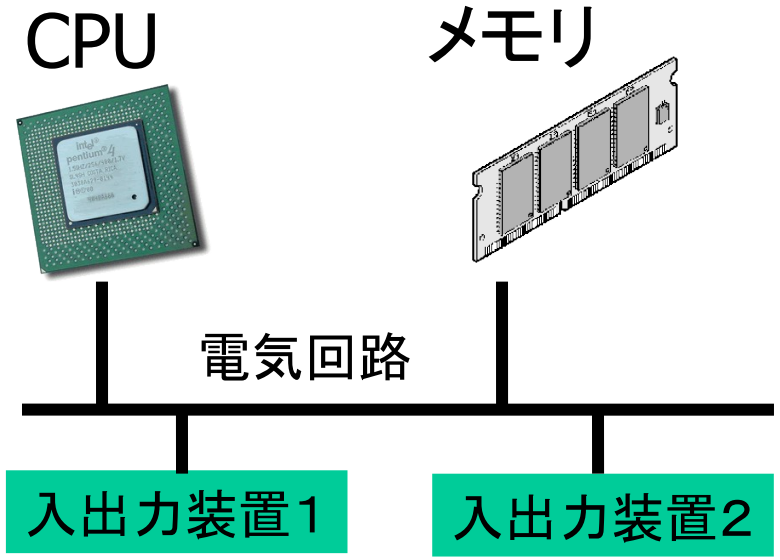
コンピュータの構成要素(入出力装置)

入出力装置(I/O System): よくI/Oと略す

外部との接点．CPUやメモリと接続．(キー，画面，プリンタ…)

仕事: CPUと外界の情報(数)の受け渡し．CPUの代わりに高度な計算をすることもある．

仕事例: CPUから受け取った数を画面に表示．



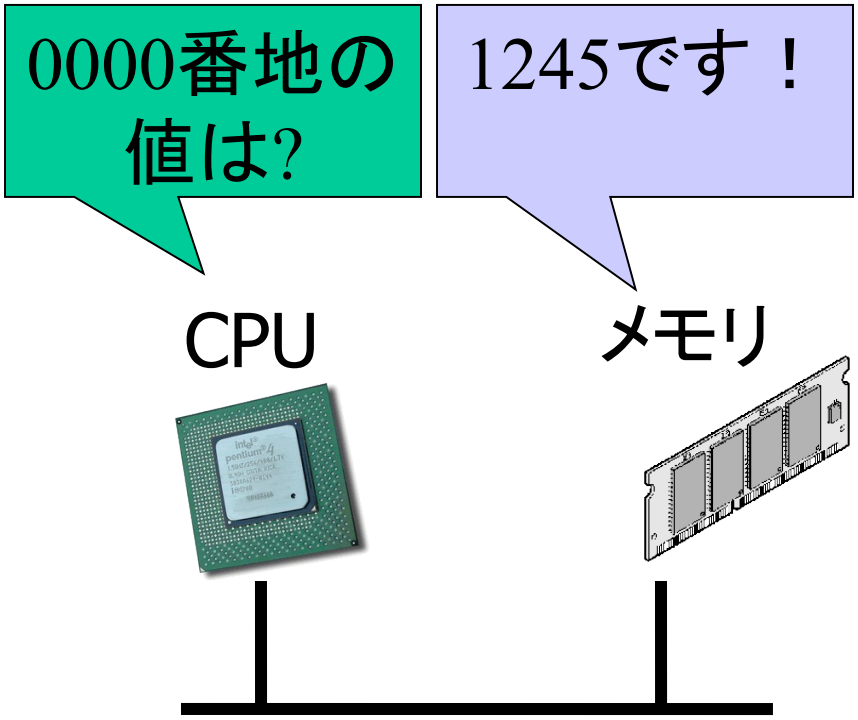
ソフトウェア（プログラム）とは

＝CPUの計算手順を書いた数字の列

1. メモリ中に数の列（プログラム）を配置
2. CPU がメモリから数を受取る(ロード)
3. 数に対応する命令を理解(1:加算,2:減算)
4. 計算して，結果をメモリに記憶(ストア)
5. 時々，入出力装置と数のやりとり

これらの繰り返しでプログラムが実行される

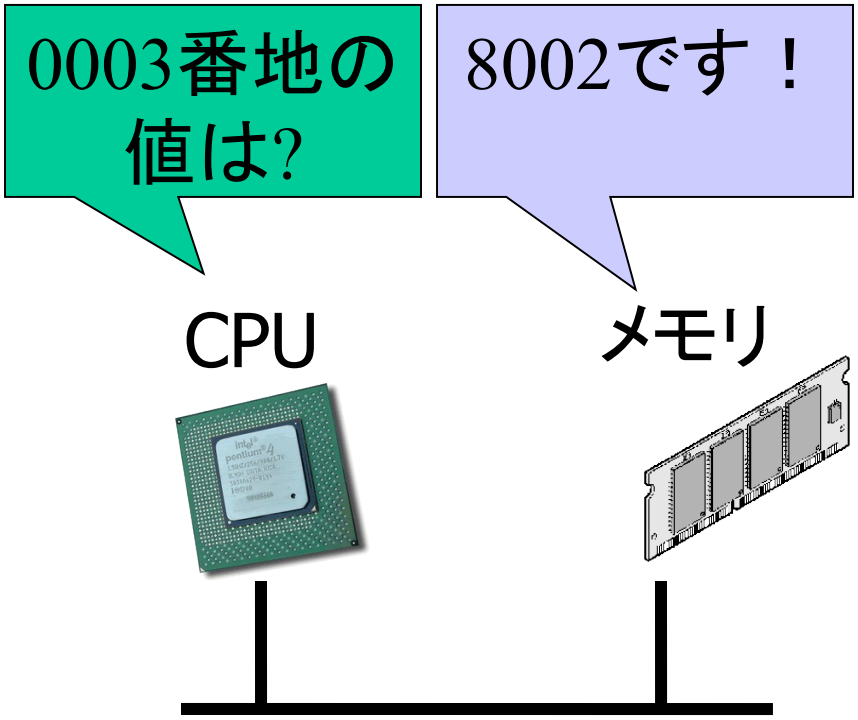
プログラムの動作 1/4



番地	値	意味
0000	1245	メモリの中身同士加算
0001	8000	8000番地の値を使え
0002	8001	8001番地の値も使え
0003	8002	8002番地に答を入れろ
:	:	:
8000	1	1+2の1(足される数)
8001	2	1+2の2(足す数)
8002	?	答を入れる場所

CPU: メモリの加算…あと3つ数を読まなきゃ

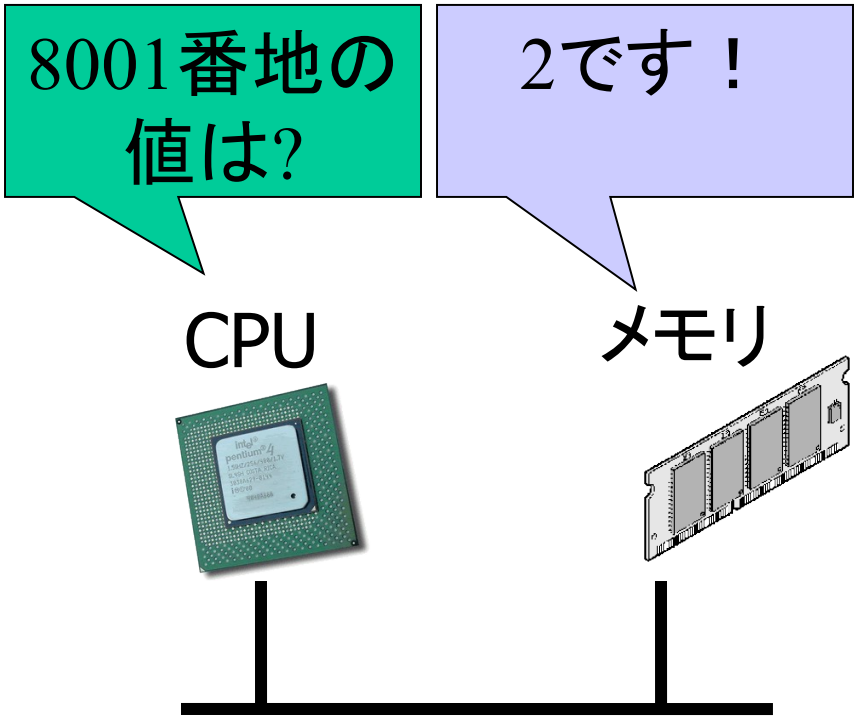
プログラムの動作 2/4



番地	値	意味
0000	1245	メモリの中身同士加算
0001	8000	8000番地の値を使え
0002	8001	8001番地の値も使え
0003	8002	8002番地に答を入れろ
:	:	:
8000	1	1+2の1(足される数)
8001	2	1+2の2(足す数)
8002	?	答を入れる場所

CPU: 8000,8001番地の数を読まなきゃ…

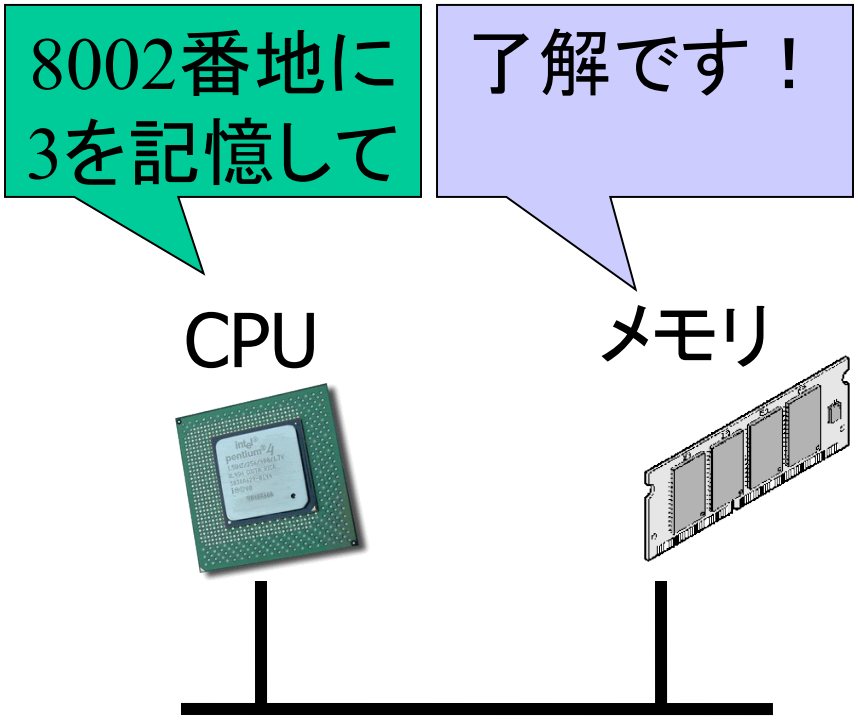
プログラムの動作 3/4



番地	値	意味
0000	1245	メモリの中身同士加算
0001	8000	8000番地の値を使え
0002	8001	8001番地の値も使え
0003	8002	8002番地に答を入れろ
:	:	:
8000	1	1+2の1(足される数)
8001	2	1+2の2(足す数)
8002	?	答を入れる場所

CPU: 1+2を8002番地にいれればいいんだ

プログラムの動作 4/4



番地	値	意味
0000	1245	メモリの中身同士加算
0001	8000	8000番地の値を使え
0002	8001	8001番地の値も使え
0003	8002	8002番地に答を入れろ
:	:	:
8000	1	1+2の1(足される数)
8001	2	1+2の2(足す数)
8002	3	答を入れる場所

CPU: 0004から次の命令を読まなきゃ…つづく

Practice1-1

CPUの気持ちになって,
計算を机上でしてみよう.

CPUの動きは，とても
単純な動作の繰り返しです.

P1-1 : 解答(step1)

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	00 →05	計算結果
81	05	繰り返し回数

1. 00～03番地を読んで命令を解釈
2. 80番地と81の値を加算して80番地へ
   80番地の値が00→05に変化

P1-1 : 解答(step2)

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	05	計算結果
81	05 →04	繰り返し回数

1. 04～07番地を読んで命令を解釈
2. 81番地の値から1を減じて81番地へ
   81番地の値が05→04に変化

P1-1 : 解答(step3)

次回 ➡

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	05	計算結果
81	04	繰り返し回数

1. 08～10番地を読んで命令を解釈
2. 81番地の値が0でなければ
   次の命令は0番地から読め　4なので0でない

次命令は0番地から

P1-1 : 解答(step4)

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	05→09	計算結果
81	04	繰り返し回数

1. 00～03番地を読んで命令を解釈
2. 80番地と81の値を加算して80番地へ
   80番地の値が05→09に変化

P1-1 : 解答(step5)

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	09	計算結果
81	04→03	繰り返し回数

1. 04～07番地を読んで命令を解釈
2. 81番地の値から1を減じて81番地へ
   81番地の値が04→03に変化

P1-1 : 解答(step6)

次回 ➡

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	09	計算結果
81	03	繰り返し回数

1. 08～10番地を読んで命令を解釈
2. 81番地の値が0でなければ
   次の命令は0番地から読め

次命令は0番地から

P1-1 : 解答(step7)

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	09→12	計算結果
81	03	繰り返し回数

1. 00～03番地を読んで命令を解釈
2. 80番地と81の値を加算して80番地へ
   80番地の値が09→12に変化

P1-1 : 解答(step8)

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	12	計算結果
81	03→02	繰り返し回数

1. 04～07番地を読んで命令を解釈
2. 81番地の値から1を減じて81番地へ
   81番地の値が03→02に変化

P1-1 : 解答(step9)

次回 ➡

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	12	計算結果
81	02	繰り返し回数

1. 08～10番地を読んで命令を解釈
2. 81番地の値が0でなければ
   次の命令は0番地から読め

次命令は0番地から

P1-1：解答(step10)

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	12→14	計算結果
81	02	繰り返し回数

1. 00～03番地を読んで命令を解釈
2. 80番地と81の値を加算して80番地へ
   80番地の値が12→14に変化

P1-1：解答(step11)

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	14	計算結果
81	02→01	繰り返し回数

1. 04～07番地を読んで命令を解釈
2. 81番地の値から1を減じて81番地へ
   81番地の値が02→01に変化

P1-1：解答(step12)

次回 ➡

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	14	計算結果
81	01	繰り返し回数

1. 08～10番地を読んで命令を解釈
2. 81番地の値が0でなければ
   次の命令は0番地から読め

次命令は0番地から

P1-1 : 解答(step13)

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	14→15	計算結果
81	01	繰り返し回数

1. 00～03番地を読んで命令を解釈
2. 80番地と81の値を加算して80番地へ
   80番地の値が14→15に変化

P1-1：解答(step14)

CPU ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	15	計算結果
81	01→00	繰り返し回数

1. 04～07番地を読んで命令を解釈
2. 81番地の値から1を減じて81番地へ
   81番地の値が01→00に変化

P1-1 : 解答(step15)

CPU ➡
次回 ➡

番地	値	意味
00	01 80 81 80	(80) ＋ (81) → (80)
04	04 81 01 81	(81) － 1 → (81)
08	05 81 00	JUMP TO 0 IF (81)≠0
11	06	終了
:		
80	15	計算結果
81	00	繰り返し回数

1. 08～10番地を読んで命令を解釈
2. 81番地の値が0でなければ
   次の命令は0番地から読め 0なので条件合致

次命令は11番地から

P1-1 : 解答(step16)

	番地	値	意味
	00	01 80 81 80	(80) ＋ (81) → (80)
	04	04 81 01 81	(81) － 1 → (81)
	08	05 81 00	JUMP TO 0 IF (81)≠0
CPU ⇨	11	06	終了
	:		
	80	15	計算結果
	81	00	繰り返し回数

1. 11番地を読んで命令を解釈
2. 終了

プログラム = 数字の列

プログラムの例

```
0061160   1  40 192 232 172 195   3   0 104 192   1   0   0 104   1 252
0061200  39 192 232 157 195   3   0 104   1   5   0   0 104 128  33   0
0061220   0 104  67 253  39 192 232  25 193   3   0 106   0  15 183   5
0061240  64  18  53 192  80 104   6 253  39 192 232 121 152  34   0 131
```

複雑な処理をするプログラム:

何百万，何千万個の数字の列

人間が理解・作成するのは困難

高級言語の世界

人間に理解容易な形でプログラムを作りたい

```
:
6   int i;
7
8   /* num で指定回数だけ実行 */
9   for (i = 0; i < num; i++) {
10
11      SUM = SUM + i;
12  }
:
```

高級言語 (C言語の例)

翻訳

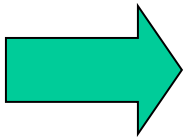
```
0061160   1  40 192 232 172 195   3
  0 104 192   1   0   0 104   1 252
0061200  39 192 232 157 195   3   0
104   1   5   0   0 104 128  33   0
0061220   0 104  67 253  39 192
232  25 193   3   0 106   0  15 183
50061240  64  18  53 192
 80 104   6 253  39 192 232 121 152
 34   0 131 …
```

機械語(あるCPUの例)

高級言語から機械語への翻訳(コンパイル)を行う道具をコンパイラという

機械語とアセンブリ言語

```
        blez    $a1,L10
        move    $v1,$zero
        lw      $a0,_ret
L6:
        addu    $a0,$a0,$v1
        addu    $v1,$v1,1
        slt     $v0,$v1,$a1
        bne     $v0,$zero,L6
          :
```




```
0061160   1  40 192 232 172 195   3
  0 104 192   1   0   0 104   1 252
0061200  39 192 232 157 195   3   0
104   1   5   0   0 104 128  33   0
0061220   0 104  67 253  39 192
232  25 193   3   0 106   0  15 183
50061240  64  18  53 192
 80 104   6 253  39 192 232 121 152
 34   0 131 …
```

機械語と逐語的に対応。機械語よりは理解しやすい

でも、コンパイラを使えば、アセンブリ言語は不要？

アセンブリ言語の使いどころ

- どうしても高級言語で記述できない部分
  CPU固有命令（割込，システムコール等）
  - 高速化やメモリ効率を重視する部分
  - CPUに対応するコンパイラがない
  新規に設計されたCPU

CPUと高級言語の相互の関係の深い理解
　高級言語においても、効率的記述を会得

必要最低限の部分をアセンブラ。後は高級言語が賢い手法　→　共存手法を学ぶ必要がある

Practice1-2

MIPS CPU のシミュレータである
SPIMを使って、
サンプルプログラムを動作させてみよう。

特殊な模擬コンピュータを
使います

Practice1-2

```
        .text                   #  text セグメントに配置する指定
       .align  2                #  4バイト境界に配置する指定
main:                           # main (ここから実行開始)
        move    $a0,$zero       # $a0 : SUM   = 0
        li      $v1,1           # $v1 : COUNT = 1
loop:
        addu    $a0,$a0,$v1     # SUM   += COUNT
        addu    $v1,$v1,1       # COUNT++
        slt     $v0,$v1,6       #  $v0 = COUNT < 6 ? 1 : 0
        bnez    $v0,loop        #  if ($v0 != 0) goto loop
        move    $t0,$a0         #  $t0 = SUM
        j       $ra             #  return
```

Practice1-2

疑問

(1) textセグメントって？ align？

(2) 個々の命令の意味は？

(3) ＄a0ってなに？

(4) 結果はどこに出る？ printf は？

→ 教科書の2章及び付録A

アセンブラから見たMIPS CPU とは？

MIPS　CPUの概要

- 32個の32ビット汎用レジスタをもつ
- RISC型（命令数が少ない）
- ロード＆ストアアーキテクチャ
- ゲーム機，組込機器などに多く採用実績

Intel Pentium などに比べて、命令体系が単純

- 学習に向いている
- CPUを小さく作れる

MIPS　CPUのレジスタ

レジスタとは

CPUがメモリ以外に内部に持っている記憶領域
この領域を利用して演算を高速に行う

$0 ～ $31 のように表現できる

```
add  $4, $5, $6        reg4  = reg5 + reg6
```

番号だと分かり辛い

MIPS　CPUのレジスタ

レジスタは、$0 ～ $31 のように表現されるが
人間に分かり辛い

慣例で使用用途を決めて、名前が付いている

教科書 A6節 図A6.1参照

```
add  $a0, $a1, $a2        reg4  = reg5 + reg6
```

少しはマシ

MIPS CPUのレジスター覧

名前	番号	用途
$zero	0	常にゼロ(書込み不可)
$at	1	アセンブラ(擬似命令)が使用
$v0 ～ $v1	2-3	戻り値
$a0 ～ $a3	4-7	引数
$t0 ～ $t7	8-15	一時変数用
$s0 ～ $s7	16-23	退避が必要な変数用
$t8 ～ $t9	24-25	一時変数用
$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	戻りアドレス記憶用

$2～$23を通常の演算で使用。詳細は教科書 A.6節参照

MIPS CPU の 命令

RISC (Reduced Instruction Set Computer)

命令数が非常に少ない
→覚えるのが簡単

教科書 まず 2.1-2.3, 2.6, 2.7 を理解
必要に応じて A.10 参照

- 全てを覚える必要はない。英単語のようなもの。
- 頻出単語は限られている。
- 本演習で1つも使用しない:
  浮動小数点命令、トラップ命令
  例外命令(syscallだけは使用)

ロード＆ストア　アーキテクチャ

ロード命令：　メモリからレジスタへの読込
ストア命令：　レジスタからメモリへの書出

MIPSでは、メモリに直接演算する命令がない

1. メモリのデータは、まずレジスタにロード
2. レジスタを使って演算 (add, sub…)
3. 結果をメモリにストア

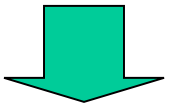
```
                                  # $sp が 8000 だとすると:
lw      $a0,    8($sp)     # 8008番地から$a0にロード
lw      $a1,    4($sp)     # 8004番地から$a1にロード
add     $a2, $a0, $a1      # $a2 = $a0 + $a1
sw      $a2,   0($sp)      # 8000番地に結果を保存
```

Practice2: 文字（列）表示

CPU：数字の羅列を扱うだけなのに
文字を表示できたりするのは何故?

- CPUは文字を認識している訳ではない
- ‘A’を65だと認識している

画面に`A’を表示する



CPUは65を表示装置に送っているだけ

文字と数字の変換表を使用

文字と数字の変換表(ASCIIコード表)

番号	文字	番号	文字	番号	文字	番号	文字
:		48	0	:		74	J
10	改行	49	1	65	A	75	K
:		50	2	66	B	76	L
32	空白	51	3	67	C	77	M
33	!	52	4	68	D	78	N
34		53	5	69	E	79	O
35	#	54	6	70	F	80	P
36	$	55	7	71	G	81	Q
37	%	56	8	72	H	82	R
:		57	9	73	I	83	:

表示装置に35を送る→＃が表示される

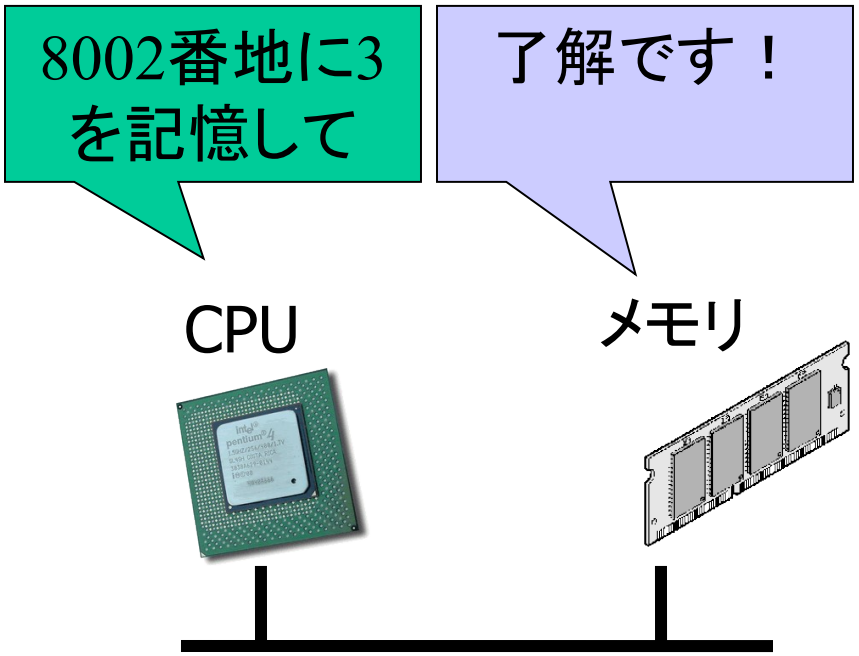
ソフトウェアから見た入出力装置

表示装置に35を送る→＃が表示

どうやって35を表示装置に送る?

CPU→メモリ:
番地を指定して
数を送る

表示装置も同じ方法:
装置専用の番地を
割り当てて送受信



仮想表示装置への出力

プリンタへの割り当て例:

0xffff0008番地: 状態監視用

0xffff000c番地: 印字用

- 0xffff0008番地を読む →

  奇数なら印字準備OK

- 0xffff000c番地に65を書く →

  A が印字される

P2-1: ポーリングによる表示

入出力機器に対応する番地を
直接読み書きして，文字出力を
してみましょう.

ASCIIコード表に対応する
数字を並べて、Hello
と表示してみよう

% man ascii　で調べて