

コンピュータの動作入門

和田 知久

琉球大学・工学部・情報工学科教授

wada@ie.u-ryukyu.ac.jp

<http://bw-www.ie.u-ryukyu.ac.jp/~wada/>



アウトライン

1. ソフトウェアとハードウェア
2. ソフトウェアはどのようにハードウェアで実行されるか
3. コンピュータの基本構成
4. シーケンスのコントロール
5. ノイマン形コンピュータ
6. 一般的RISCコンピュータの構成
7. ミニミップスマシン
8. 実際のPCの構成
9. キャッシュメモリの導入(メモリの階層化)

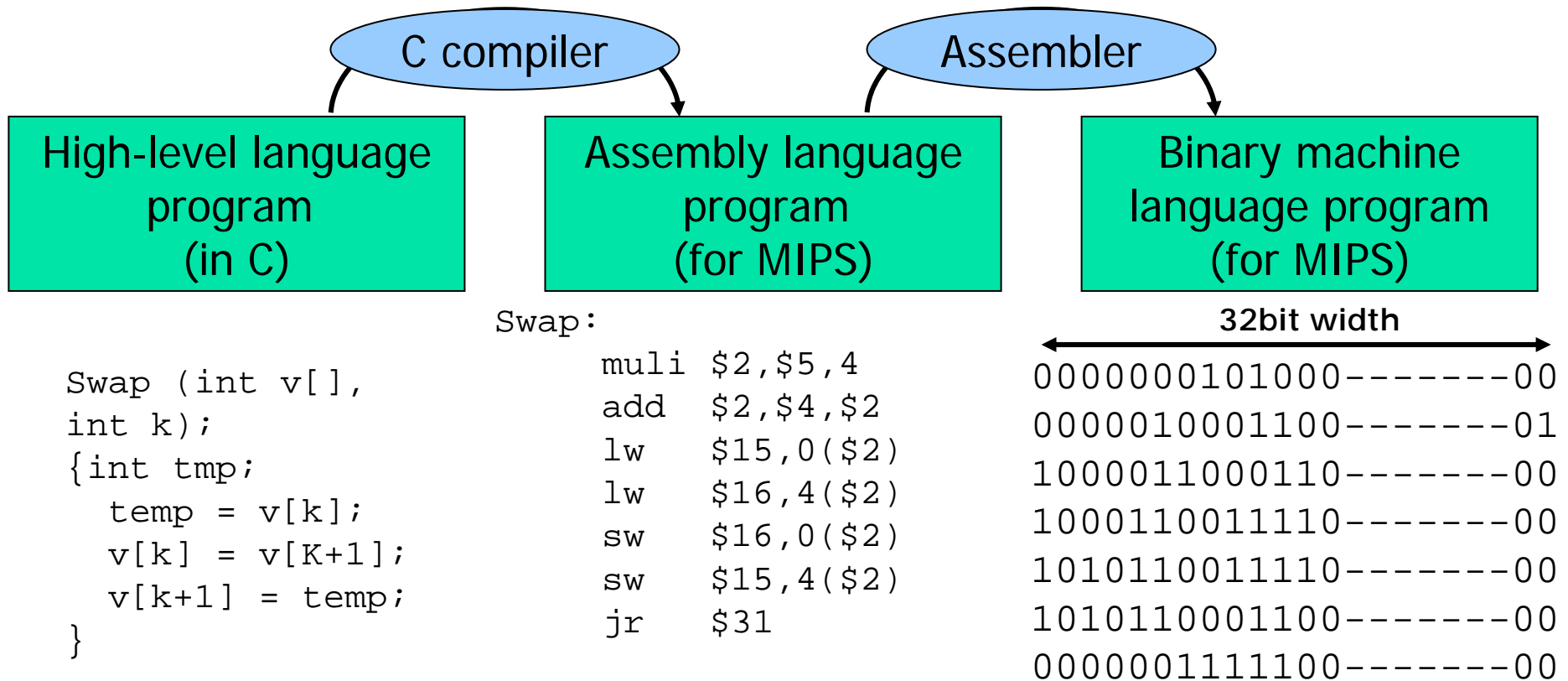


ソフトウェアとハードウェア

- コンピュータ関連には2種類のエンジニアがいる
ソフトウェア VS ハードウェア
- ソフトウェア:
 - C, Java, HTML, PARL, PASCAL...
- すべてのソフトウェアはハードウェア上で実行される
 - PC, DSP board, PDA...
- 普通、ソフトウェア人はハードウェアを理解せず、またその逆も真である
- しかし、本当に凄いコンピュータシステムを実現したいなら、H/WとS/Wを単独に最適化しても無意味である。
- 本当に凄いものをやりたいなら、両方を最適化する必要がある。

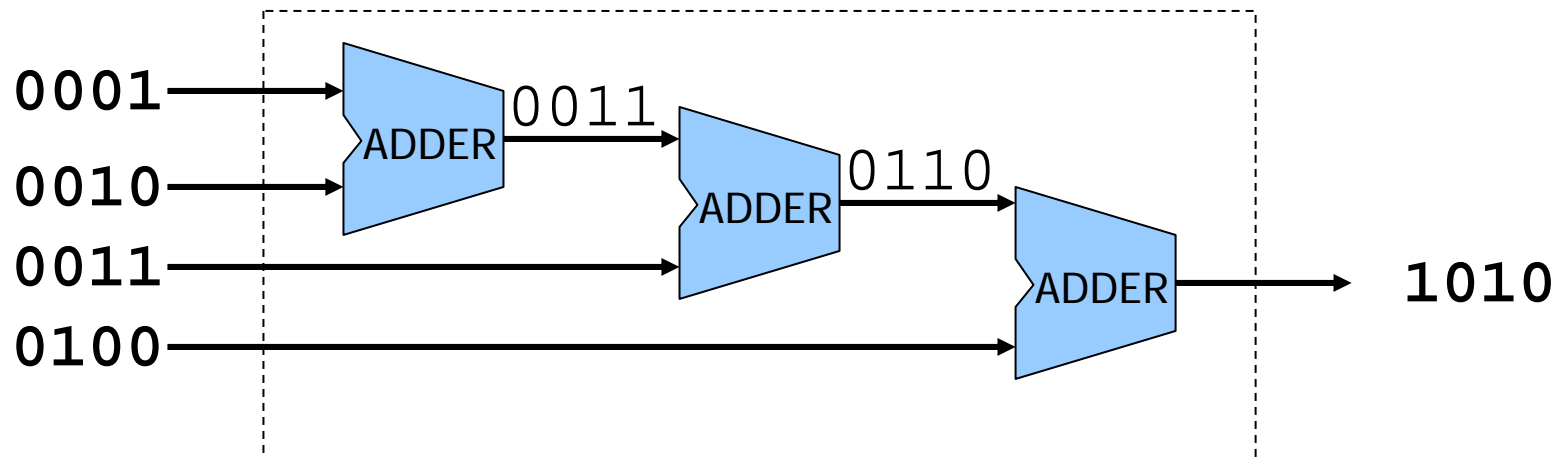
さて、S/WはどのようにH/Wで実行される

- デジタルコンピュータは‘1’か‘0’のみ取り扱う



ケース1: $1+2+3+4$ をはどのように計算できるか？

今、H/W(LSIチップ)があるとすれば、以下のような回路を作れば良い！



この場合には、ソフトウェアは必要ない！



もし、加算機 (ADDER) が1つしかない場合どうする？

PROBLEM #1



ケース2: $1+2+3+4$ をはどのように計算できるか？

今、加算器が1つしかないとする....

A) 同じ加算器を3回使用する

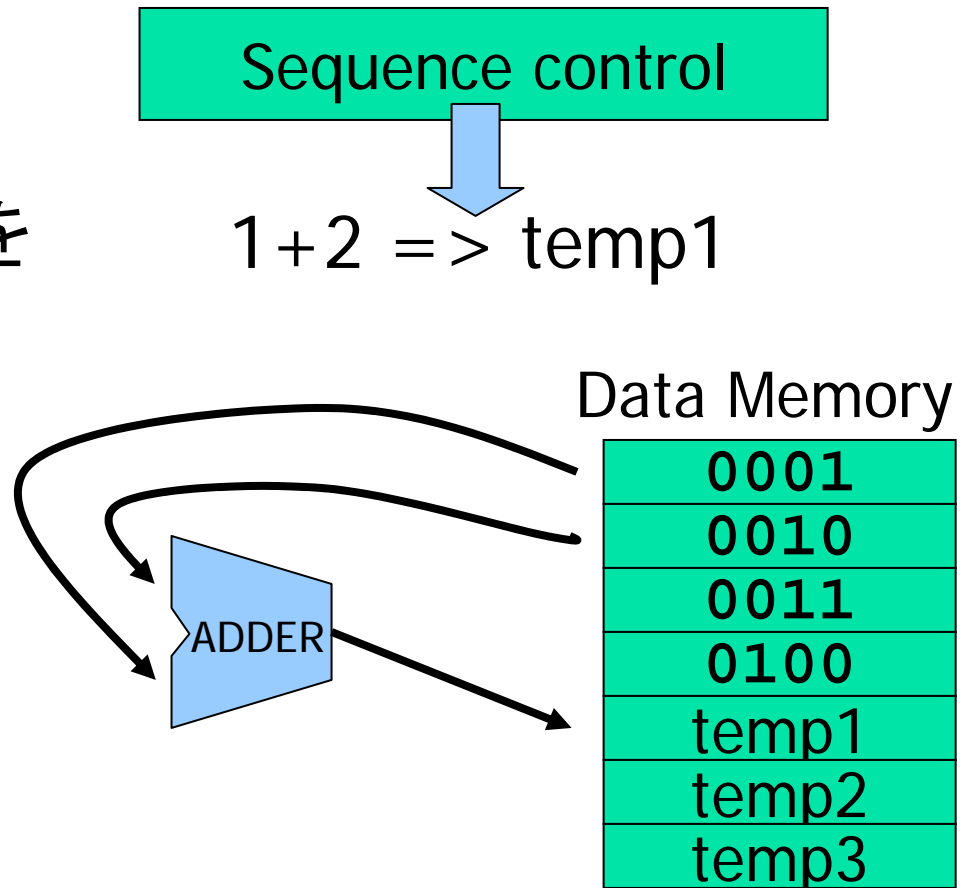
1. $1+2 \Rightarrow \text{temp1}$
2. $\text{temp1} + 3 \Rightarrow \text{temp2}$
3. $\text{temp2} + 4 \Rightarrow \text{temp3} \Rightarrow \text{答え}$

B) 中間の計算値をメモリ領域に記憶する必要がある

C) 上記3つの手順は順序どおりに実行される必要がある

ケース2: $1+2+3+4$ をはどのように計算できるか?

- コンピュータの基本要素(部品)
 1. データパス(計算を行うところ)
 2. シーケンス(順序)のコントロール
 3. 記憶領域(メモリ)

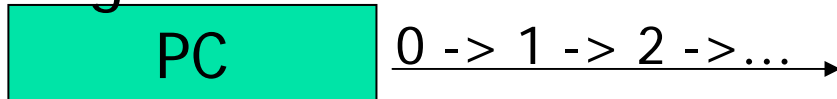


実在コンピュータでの、シーケンスコントロールとは

Sequence control



Program Counter



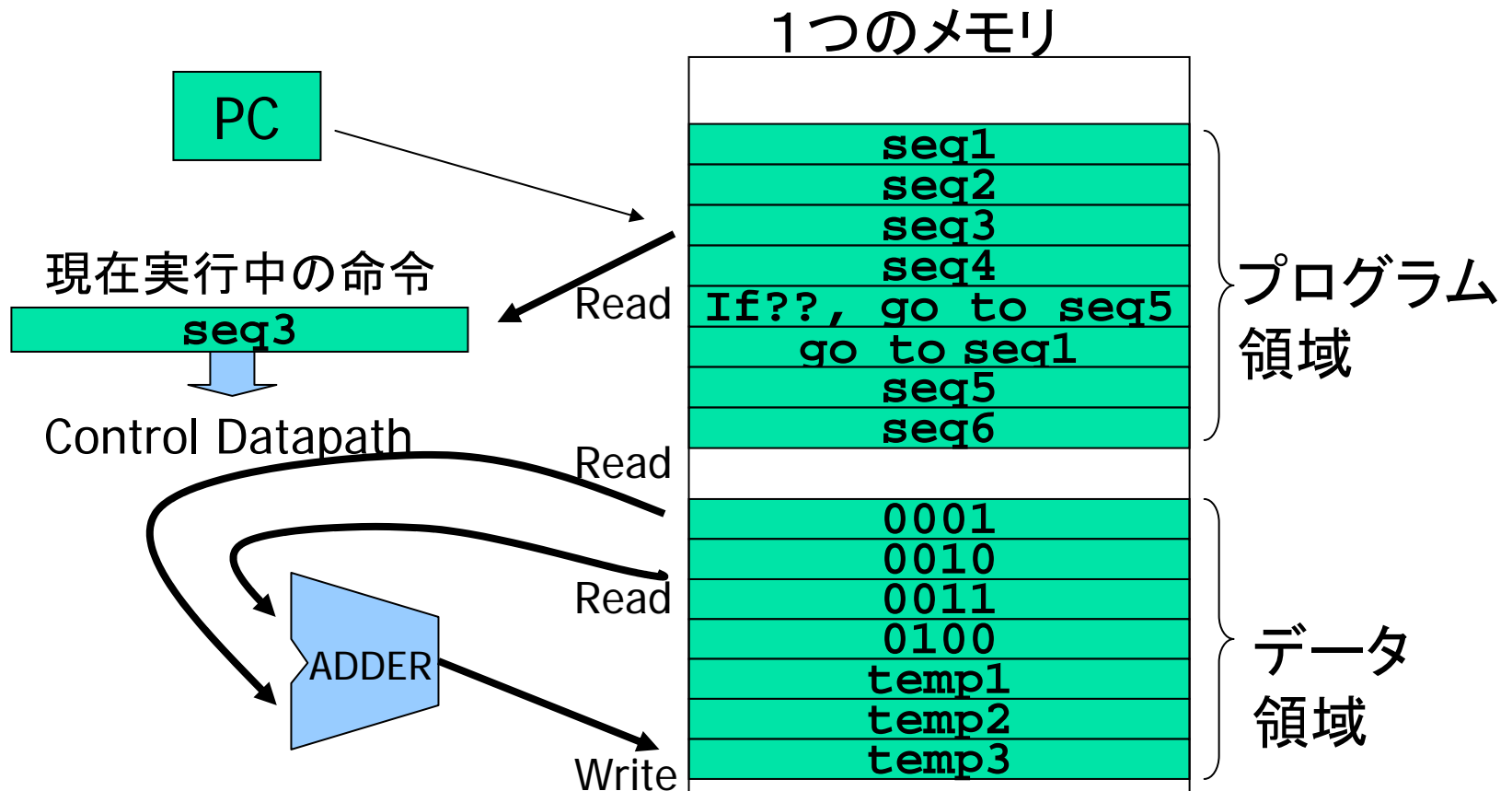
- 2進数で書かれたプログラムが、プログラムメモリに記憶されている。

Program Memory

0	seq1
1	seq2
2	seq3
3	seq4
4	If??, go to seq5
5	go to seq1
6	seq5
7	seq6

プログラムもデータも同じメモリ上にある -ノイマンマシン-

■ 現実のコンピュータでは、データ用の記憶領域とプログラム用の記憶領域は同じメモリ上にある

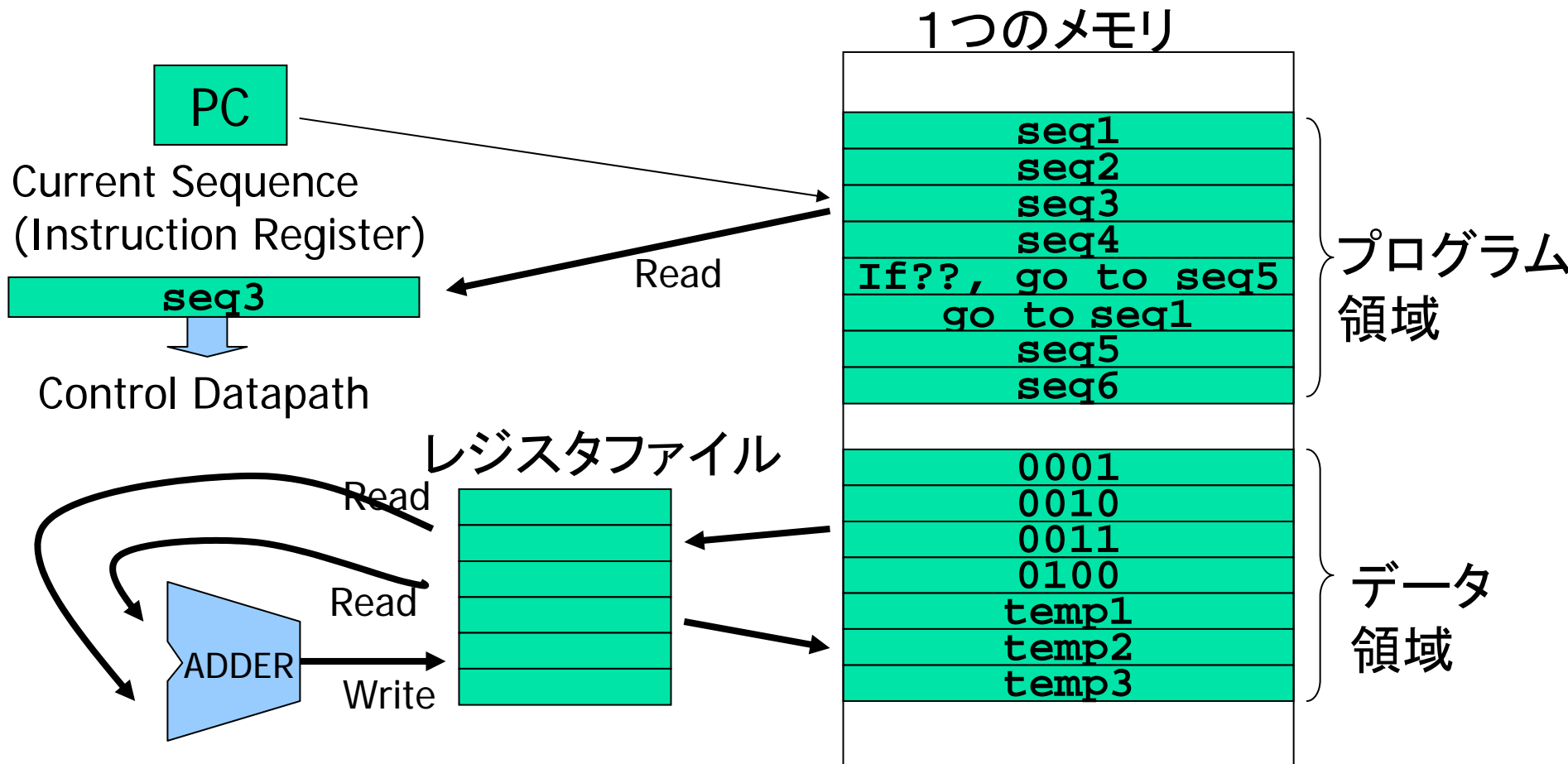




前ページのコンピュータの問題点

- 1つの命令を実行する時に、
命令をメモリから1回読み出す、
2つのデータをメモリから読み出す、
計算結果をメモリに書き込む
- メモリの使用回数が多く、メモリが低速であれば性能が
上がらない。
- そこで、データを一時的に記憶するレジスタファイルを設けた
 - メモリ: たとえば128メガバイト
 - レジスタファイル: たとえば128バイト
- それにより、新しい命令が必要となった
 - Load Word (LW) : データ領域からレジスタファイルにデータ転送
 - Store Word (SW) : レジスタファイルからデータ領域にデータ転送

一般的なRISC型コンピュータの構成



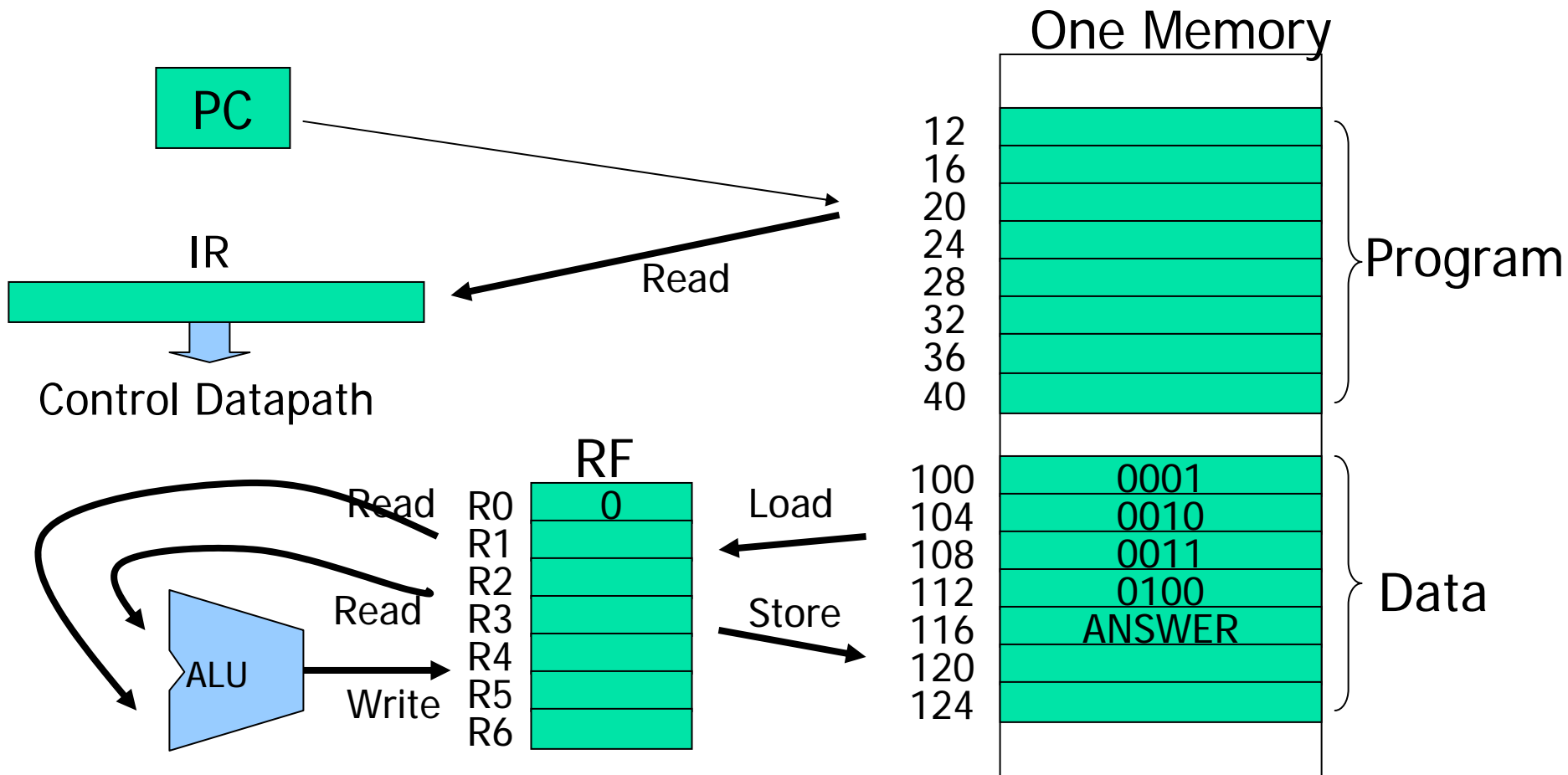


例: ミニ ミップス マシン

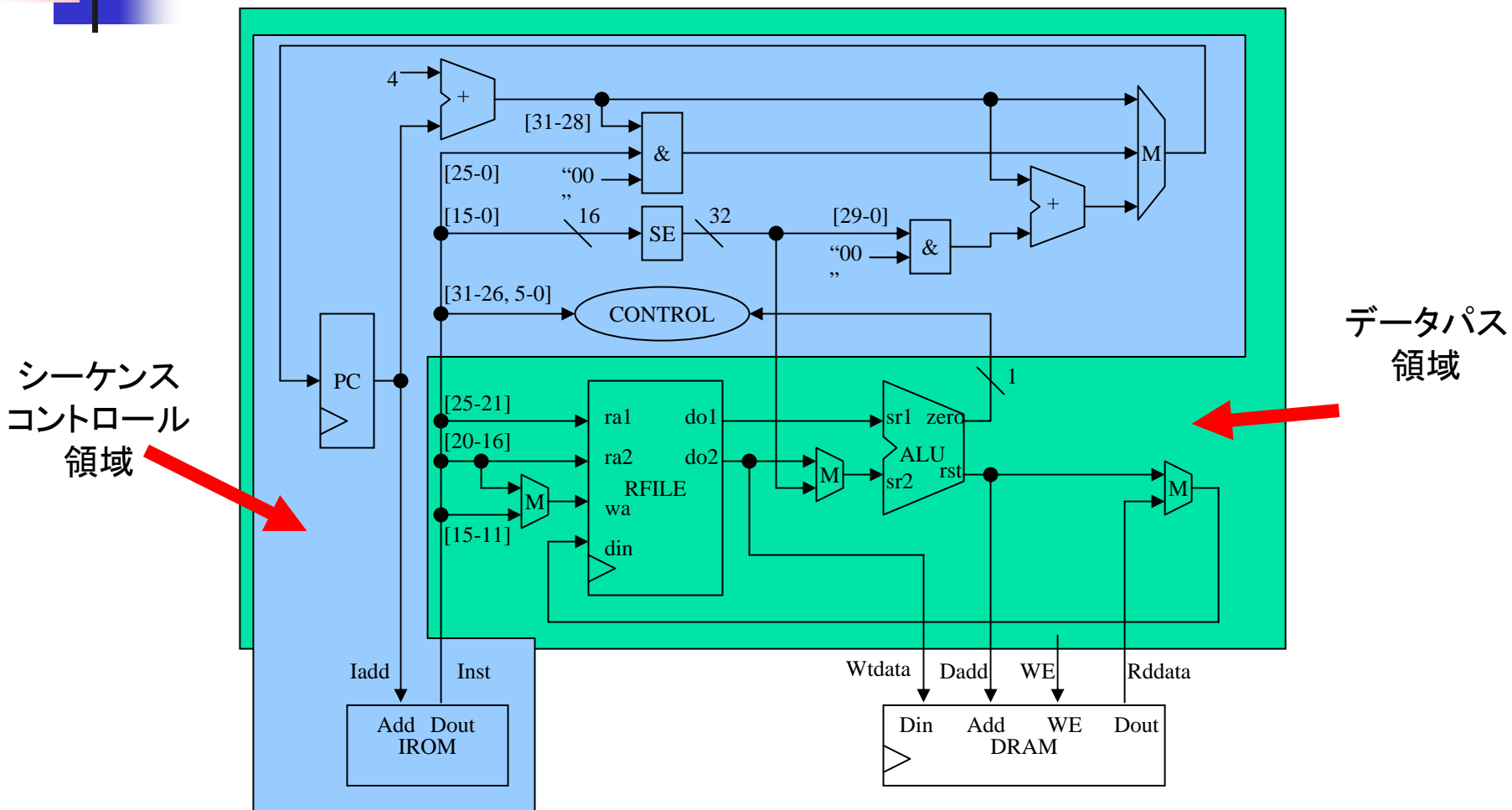
32個のレジスタR0,R1,...R31を持つRISCコンピュータ

Category	Instruction	Example	Meaning	Comments
算術命令	add	add R1,R2,R3	$R1 \leftarrow R2 + R3$	3 operands
	subtract	sub R1,R2,R3	$R1 \leftarrow R2 - R3$	3 operands
論理命令	and	and R1,R2,R3	$R1 \leftarrow R2 \text{ and } R3$	3 operands, bit-wise logical and
	or	or R1,R2,R3	$R1 \leftarrow R2 \text{ or } R3$	3 operands, bit-wise logical or
データ転送	load word	lw R1, 100(R2)	$R1 \leftarrow \text{Memory}[R2+100]$	Data from memory to register
	store word	sw R1, 100(R2)	$\text{Memory}[R2+100] \leftarrow R1$	Data from register to Memory
条件分岐	branch on equal	beq R1,R2,25	if (R1=R2) go to PC+4+100	Equal Test; PC relative branch
	set on less than	slt R1,R2,R3	if (R2<R3) R1<=1 else R1<=0	
無条件分岐	jump	j 2500	go to 10000	Jump to target address

ミニ ミップス命令をもちいて1+2+3+4を計算するプログラムを作る！

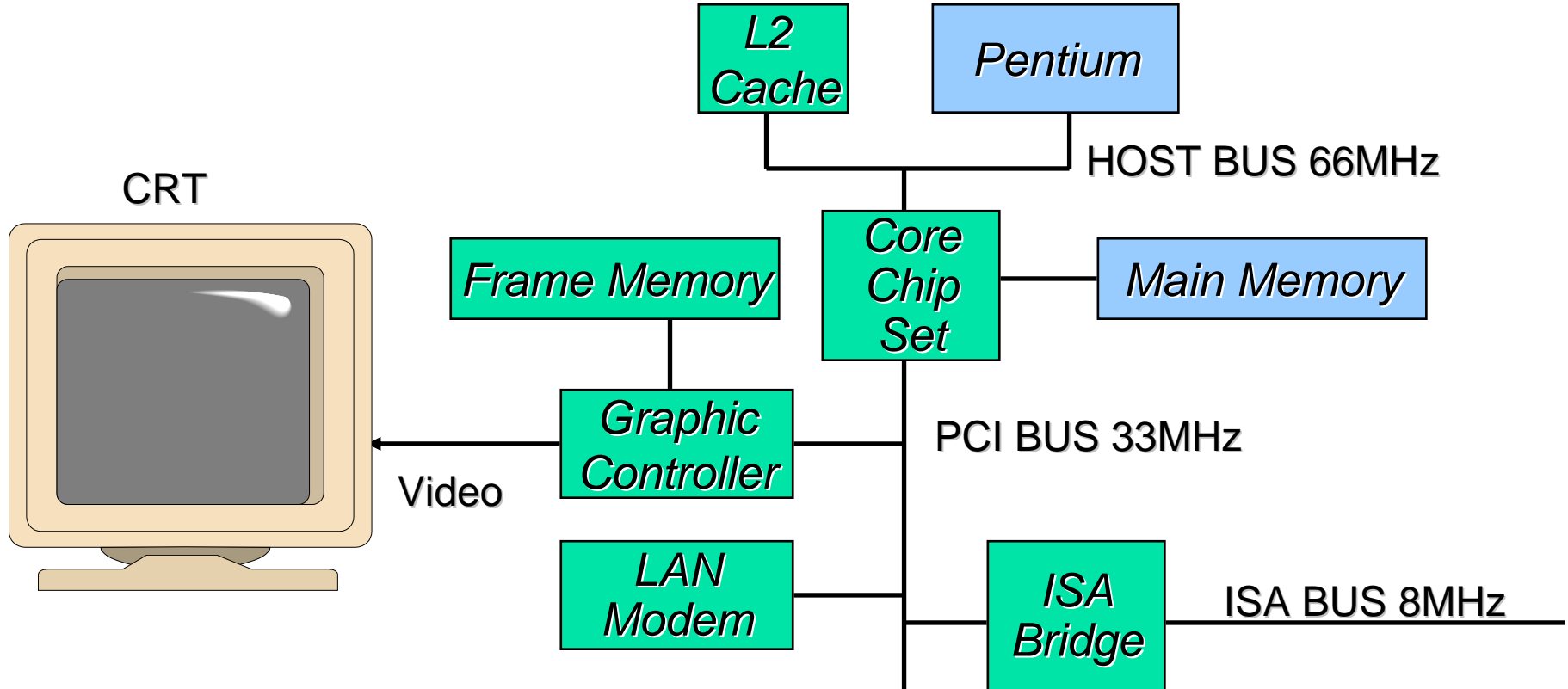


現実のミニミップスマシンのブロック図



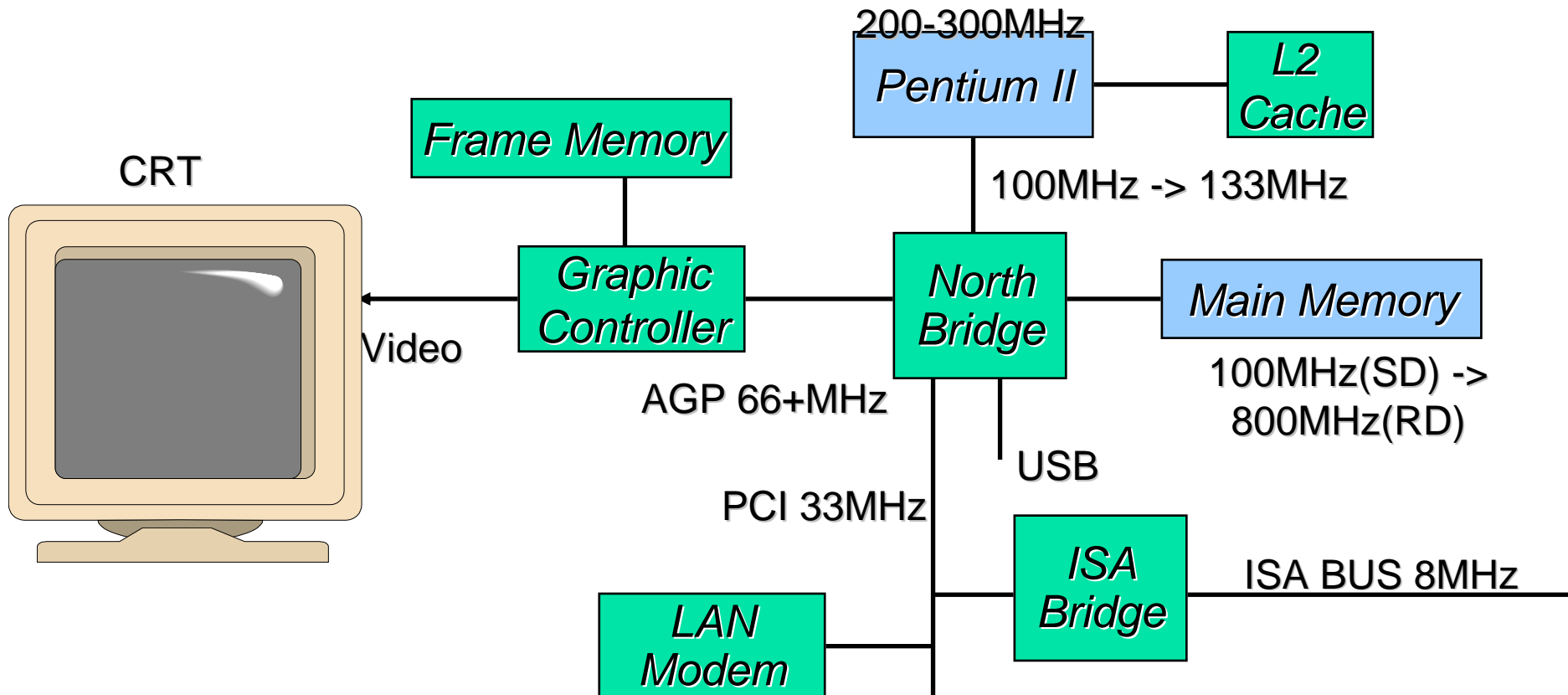
現実の Pentium PC の構成 @ 1995

- 各四角形はLSIを示す
- 線はバスと呼ばれ、多数の配線の束である。

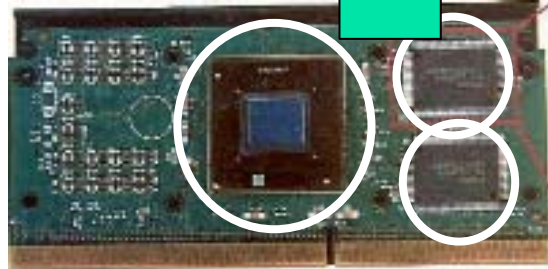


現実のPentiumII PCの構成 @ 1997

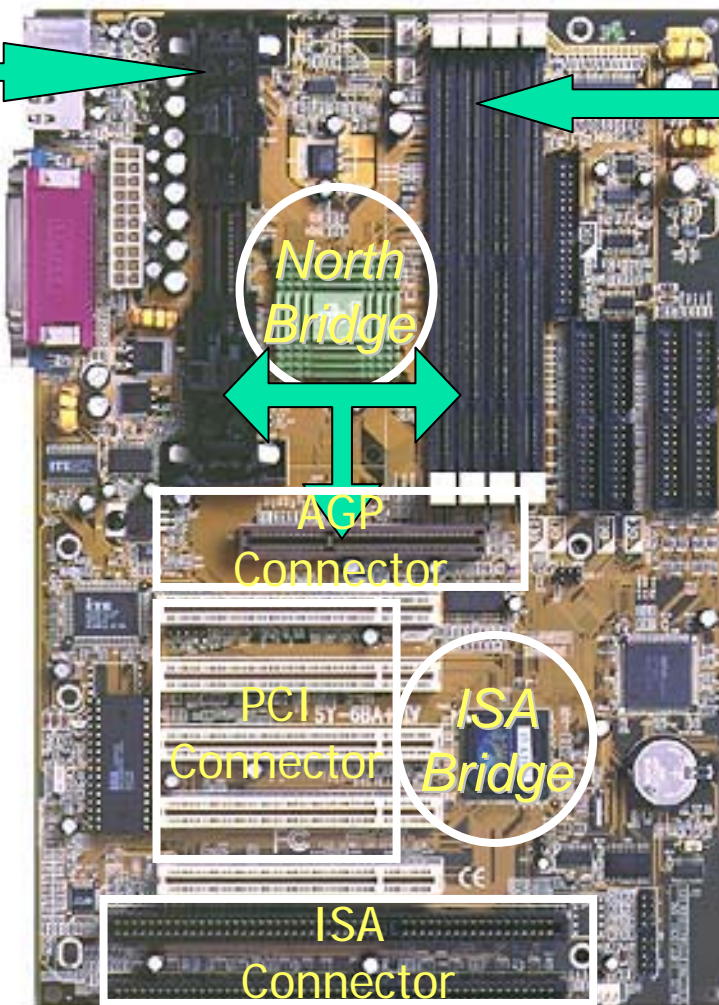
- 各四角形はLSIを示す
- 線はバスと呼ばれ、多数の配線の束である。



Real Pentium II Mother board



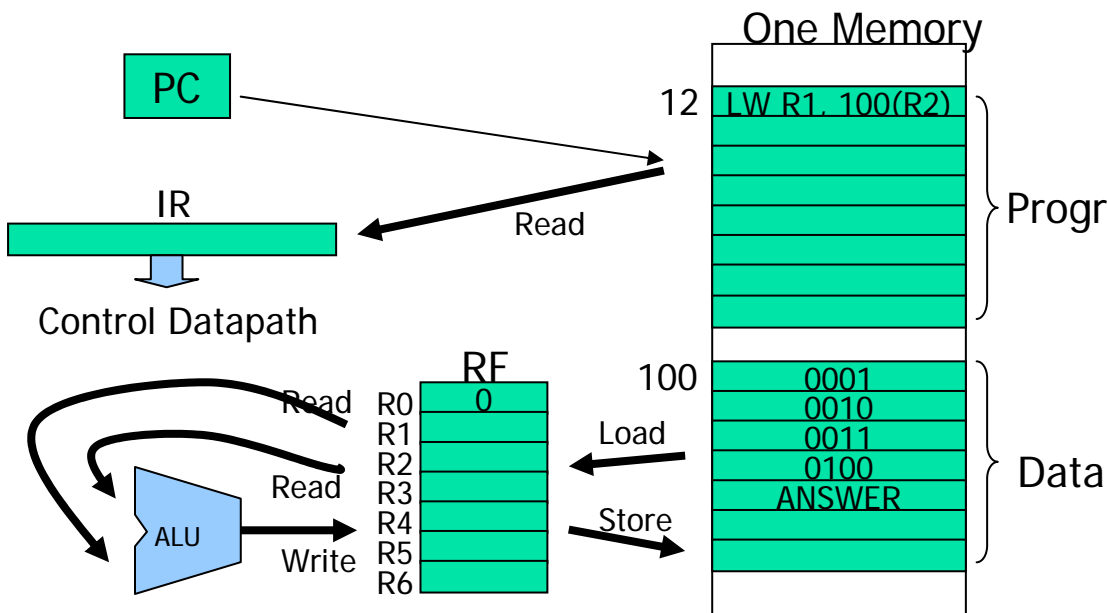
Pentium II and
Cache module



DIMM
Memory module

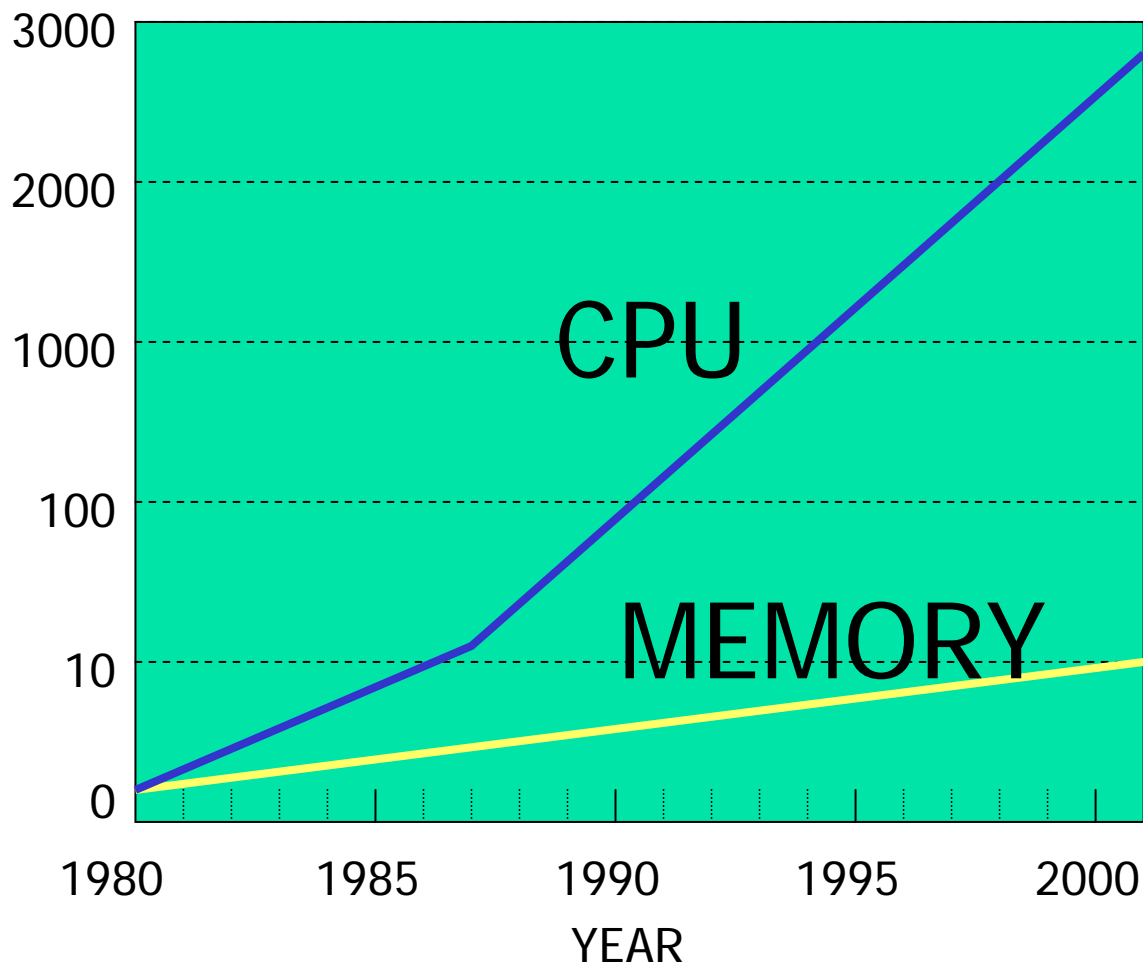
もう一度動作の確認！

LW R1, 100(R2) を実行すると実際にはどう動くか？



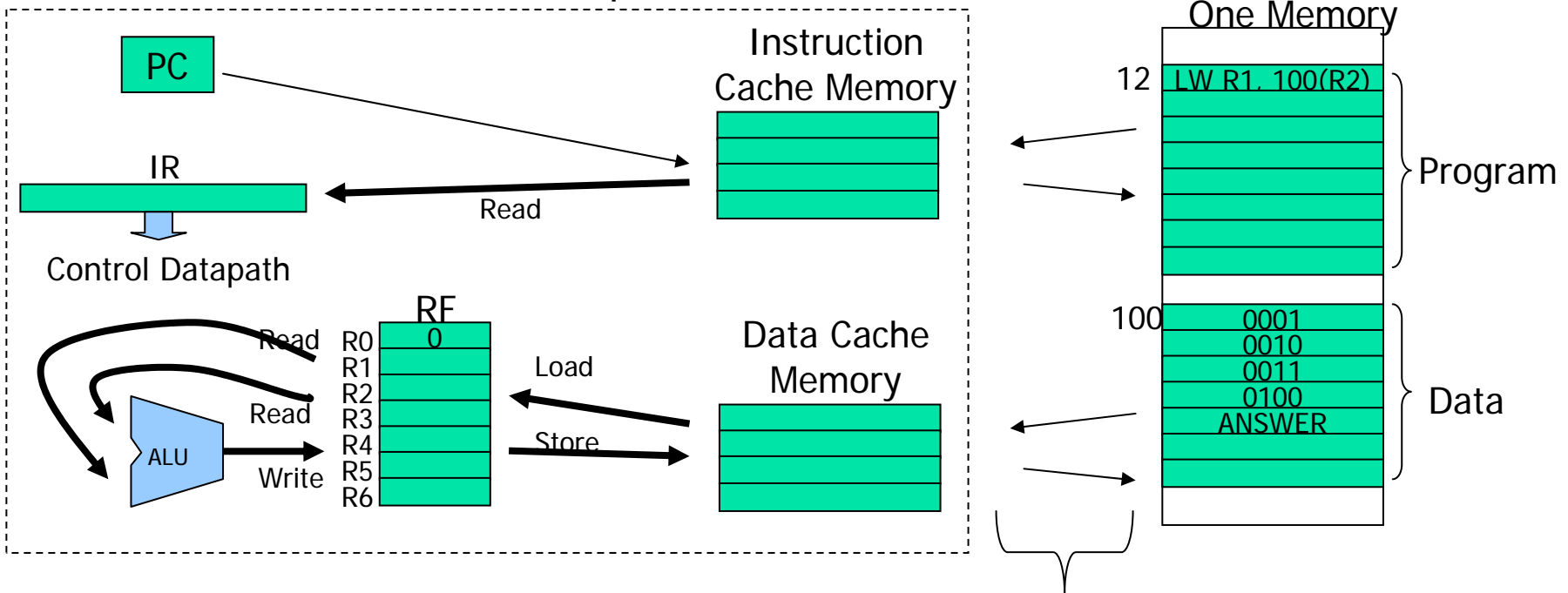
- $PC \leq 12$
- Read memory[12]
- $IR \leq LW\ R1,\ 100(R2)$
- ALU compute $100 + R2$
- Read memory[100 + R2]
- $R1 \leq memory[100 + R2]$

スピード向上の歴史: メモリ vs. CPU



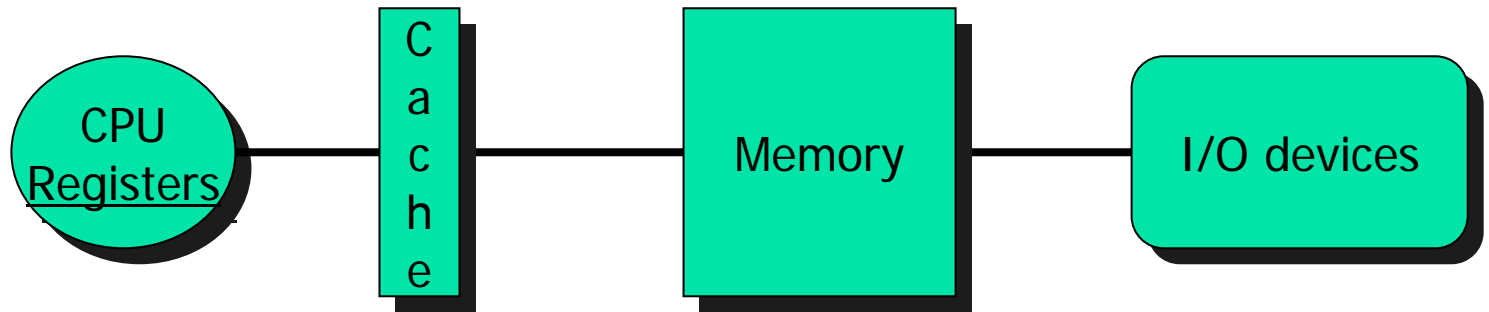
キャッシュメモリが導入 - ハーバードアーキテクチャ -

Micro Processor Chip



メモリ(主記憶)とキャッシュメモリ間のデータの転送は
S/Wでプログラムせず、H/Wでコントロールする。

メモリの階層構成で全体性能をUP



Register
reference

Cache
reference

Memory
reference

Disk Memory
reference

Size: 400Byt

256KByte

128MByt

10GByte

Speed: 2ns

5ns

100ns

5ms