

## 計算機アーキテクチャ 第二 (O)

### 1. 導入

大学院情報理工学研究科 計算工学専攻  
吉瀬謙二 kise\_at\_cs.titech.ac.jp  
S321講義室 月曜日 5, 6 時限 13:20-14:50

1

## 関連科目・履修条件等

### 4学期: 計算機論理設計

- 計算機を構成するプロセッサとその制御部に関し、具体構成と設計の原理を講義する。特に、レジスタトランസ്ഫア言語を用いて計算機の内部動作を記述し、簡単な計算機の設計を行う。

### 5学期: 計算機アーキテクチャ第一

- CPUを含め、メモリ、チャネル、入出力、通信制御、等の計算機システムを構成する各種装置について、その役割、動作原理について講義する。

### 6学期: 計算機アーキテクチャ第二

- 最新の計算機システムに採り入れられている高速プロセッサ制御方式、構成方式について述べ、これらの技術を駆使したパイプラインプロセッサ、スーパーコンピュータ、超並列計算機、データフロー計算機、等の先端的なアーキテクチャについて講義する。

### 計算機アーキテクチャ特論(大学院)

2

## 講義項目

- コンピュータの性能
- RISC vs CISC
- パイプライン制御
  - ハザードとスケジューリング
  - スーパースカラ、スーパーパイプライン、VLIW
- ベクトル・プロセッサ
- データフロー
- マルチプロセッサ、マルチコアシステム

レポート、演習、期末試験により評価

3

## 参考書(1)

### コンピュータの構成と設計 第3版

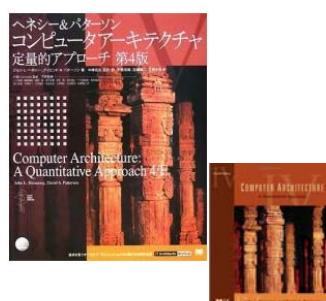
パターソン&ヘネシー  
(成田光彰 訳)、  
日経BP社、2006



4

## 参考書(2)

- コンピュータアーキテクチャ 定量的アプローチ 第4版, 翔泳社
- Computer Architecture, Fourth Edition: A Quantitative Approach, Fourth Edition
  - Publisher: Morgan Kaufmann; 4 edition (September 13, 2006)
  - ISBN-10: 0123704901
  - ISBN-13: 978-0123704900



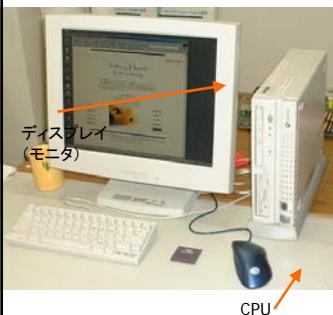
5

## 計算機アーキテクチャ 第一 (E)

### 計算機システムの基本構成

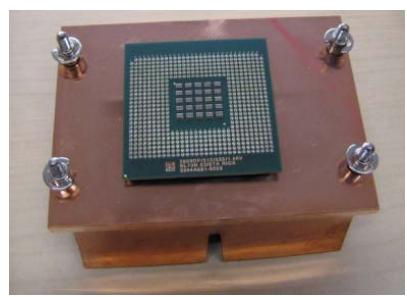
6

## 計算機(デスクトップコンピュータ)



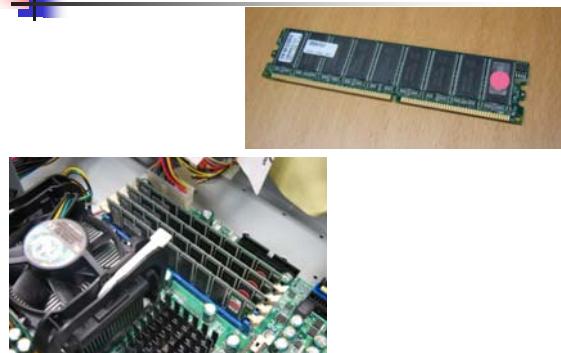
7

## マイクロプロセッサ, CPU



8

## メモリ DRAM (dynamic random access memory)



9

## ディスク, 磁気ディスク



10

## グラフィックカード



11

## ネットワークカード



12

## マザーボード



13

## 計算機



14

## 補足：クラスタ型(並列)計算機



15

## 計算機アーキテクチャとは？

- アーキテクチャ  
Architecture
- 計算機アーキテクチャ  
Computer Architecture

16

## アーキテクチャ(建築) Architecture



パルテノン神殿



世界最大のケフ王のピラミッド  
1個約2.5tのブロックを 230~250万 個  
積み重ねて造られている。

写真は計算機アーキテクチャのホームページから <http://www.cs.wisc.edu/arch/www/>

17

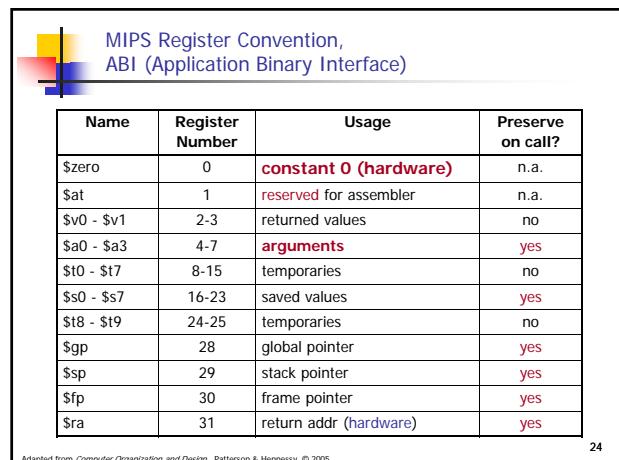
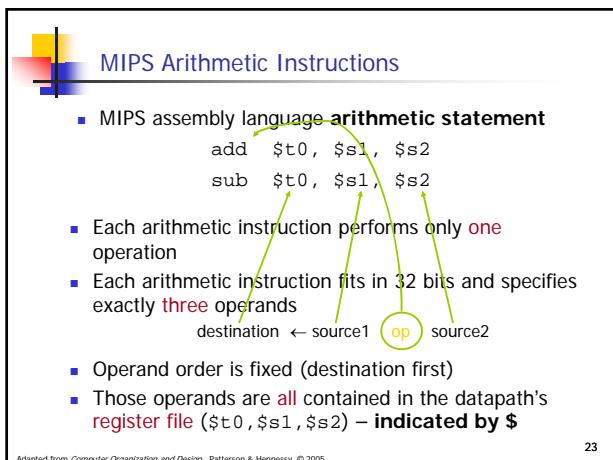
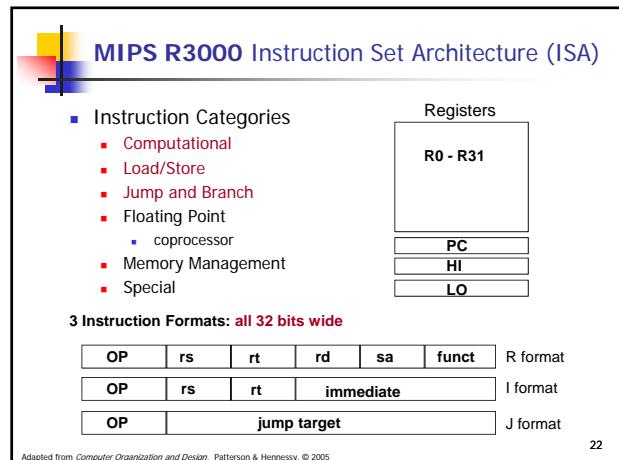
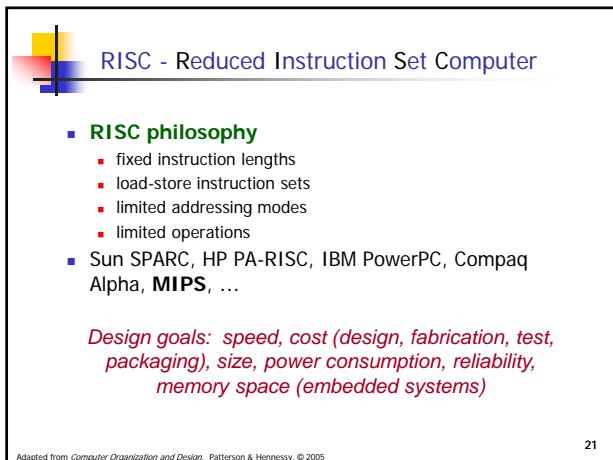
## 計算機アーキテクチャ

### What's Computer Architecture?

**Computer Architecture** is the science and art of selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals. Computer architecture is *not* about using computers to design buildings.

計算機アーキテクチャのホームページから <http://www.cs.wisc.edu/arch/www/>

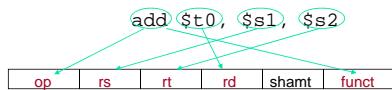
18



## Machine Language - Add Instruction

- Instructions, like registers and words of data, are **32 bits long**

- Arithmetic Instruction Format (R format):



op 6-bits **opcode** that specifies the operation  
 rs 5-bits **register file address of the first source operand**  
 rt 5-bits **register file address of the second source operand**  
 rd 5-bits **register file address of the result's destination**  
 shamt 5-bits **shift amount** (for shift instructions)  
 funct 6-bits **function code augmenting the opcode**

25

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## 演習

- $f = (g + h) - (i + j)$

$f, g, h, i, j$  をそれぞれレジスタ  $\$s0, \$s1, \$s2, \$s3, \$s4$  に割り付けるとする。  
上のステートメントをコンパイルした結果のMIPSアプリケーション・コードはどうなるか。

26

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## 演習 (参考書 48ページ)

- $f = (g + h) - (i + j)$

$f, g, h, i, j$  をそれぞれレジスタ  $\$s0, \$s1, \$s2, \$s3, \$s4$  に割り付けるとする。  
上のステートメントをコンパイルした結果のMIPSアプリケーション・コードはどうなるか。

```

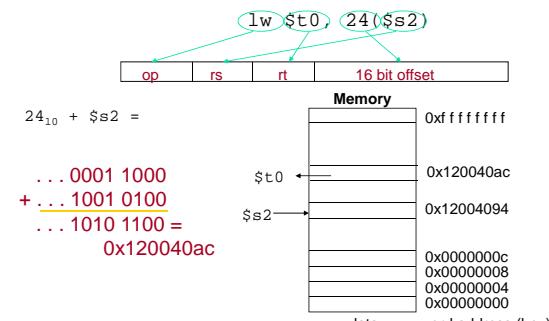
add $t0, $s1, $s2      # $t0 = (g + h)
add $t1, $s3, $s4      #
sub $s0, $t0, $t1      #
    
```

27

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## Machine Language - Load Instruction

- Load/Store Instruction Format (I format):



Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

28

## MIPS Memory Access Instructions

- MIPS has two basic **data transfer** instructions for accessing memory
  - lw \$t0, 4(\$s3) #load word from memory
  - sw \$t0, 8(\$s3) #store word to memory
- The data is loaded into (lw) or stored from (sw) a register in the register file – a 5 bit address
- The memory address – a 32 bit address – is formed by adding the contents of the **base address register** to the **offset** value
  - A 16-bit field meaning access is limited to memory locations within a region of  $\pm 2^{13}$  or 8,192 words ( $\pm 2^{15}$  or 32,768 bytes) of the address in the base register
  - Note that the offset can be positive or negative

29

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## 演習

- $g = h + A[8]$   
100語からなる配列Aがあるとする。また、コンパイラは変数  $g, h$  にレジスタ  $\$s1, \$s2$  を割り付ける。さらに配列の開始アドレスは  $\$s3$  に納められているとする。  
上のステートメントをコンパイルせよ。

30

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## 演習 (参考書 50ページ)

- g = h + A[8]  
100語から成る配列Aがあるとする. また, コンパイラは変数g, h にレジスタ \$s1, \$s2 を割り付ける. さらに配列の開始アドレスは \$s3 に納められているとする.  
上のステートメントをコンパイルせよ.

```
lw $t0, 32($s3)      # $t0 = A[8]
add $s1, $s2, $t0      # g = h + $t0
```

31

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## 演習

- A[12] = h + A[8]

100語から成る配列Aがあるとする. また, コンパイラは変数g, h にレジスタ \$s1, \$s2 を割り付ける. さらに配列の開始アドレスは \$s3 に納められているとする.  
上のステートメントをコンパイルせよ.

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

32

## 演習 (参考書 51ページ)

- A[12] = h + A[8]  
100語から成る配列Aがあるとする. また, コンパイラは変数 h にレジスタ \$s2 を割り付ける. さらに配列の開始アドレスは \$s3 に納められているとする.  
上のステートメントをコンパイルせよ.

```
lw $t0, 32($s3)      # $t0 = A[8]
add $t0, $s2, $t0      # $t0 = h + $t0
sw $t0, 48($s3)      # A[12] = $t0
```

33

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## MIPS Control Flow Instructions

- MIPS **conditional branch** instructions:  
bne \$s0, \$s1, Lbl #go to Lbl if \$s0≠\$s1  
beq \$s0, \$s1, Lbl #go to Lbl if \$s0==\$s1
  - Ex: `if (i==j) h = i + j;`  
`bne $s0, $s1, Lbl1`  
`add $s3, $s0, $s1`  
`Lbl1: ...`
- Instruction Format (**I** format):  

|    |    |    |               |
|----|----|----|---------------|
| op | rs | rt | 16 bit offset |
|----|----|----|---------------|
- How is the branch destination address specified?

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

34

## More Branch Instructions

- We have beq, bne, but what about other kinds of branches (e.g., branch-if-less-than)? For this, we need yet another instruction, slt
- Set on less than instruction:

```
slt $t0, $s0, $s1      # if $s0 < $s1      then
                         # $t0 = 1          else
                         # $t0 = 0
```

- Instruction format (**R** format):

|    |    |    |    |       |
|----|----|----|----|-------|
| op | rs | rt | rd | funct |
|----|----|----|----|-------|

35

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## More Branch Instructions, Con't

- Can use slt, beq, bne, and the fixed value of 0 in register \$zero to **create** other conditions
  - less than `blt $s1, $s2, Label`  
`slt $at, $s1, $s2 # $at set to 1 if`  
`bne $at, $zero, Label # $s1 < $s2`
  - less than or equal to `ble $s1, $s2, Label`
  - greater than `bgt $s1, $s2, Label`
  - great than or equal to `bge $s1, $s2, Label`
- Such branches are included in the instruction set as pseudo instructions - recognized (and expanded) by the assembler
  - Its why the assembler needs a reserved register (\$at)

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

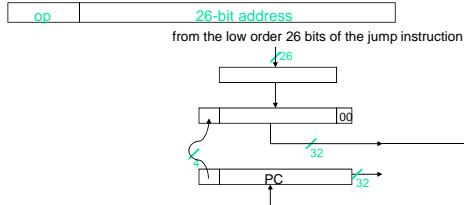
36

## Other Control Flow Instructions

- MIPS also has an **unconditional branch** instruction or **jump** instruction:

**j label #go to label**

- Instruction Format (**J** Format):



Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## Aside: Branching Far Away

- What if the branch destination is further away than can be captured in 16 bits?
- The assembler comes to the rescue – it inserts an unconditional jump to the branch target and inverts the condition

**beq \$s0, \$s1, L1**

becomes

**bne \$s0, \$s1, L2**

**j L1**

**L2:**

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

38

## Instructions for Accessing Procedures

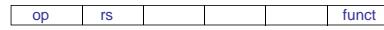
- MIPS **procedure call** instruction:  
**jal ProcedureAddress #jump and link**
- Saves PC+4 in register **\$ra** to have a link to the next instruction for the procedure return
- Machine format (**J** format):



- Then can do procedure **return** with a

**jr \$ra #return**

- Instruction format (**R** format):



39

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## MIPS Immediate Instructions

- Small constants are used often in typical code
- Possible approaches?
  - put “typical constants” in memory and load them
  - create hard-wired registers (like \$zero) for constants like 1
  - have special instructions that contain constants !

**addi \$sp, \$sp, 4 #\$sp = \$sp + 4**  
**slti \$t0, \$s2, 15 #\$t0 = 1 if \$s2 < 15**

- Machine format (**I** format):



- The constant is kept **inside** the instruction itself!
  - Immediate format **limits** values to the range  $+2^{15}-1$  to  $-2^{15}$

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

40

## MIPS ISA So Far

| Category                          | Instr                      | Op Code  | Example              | Meaning                                 |
|-----------------------------------|----------------------------|----------|----------------------|---|
| Arithmetic<br>(R & I format)      | add                        | 0 and 32 | add \$s1, \$s2, \$s3 | \$s1 = \$s2 + \$s3                      |
|                                   | subtract                   | 0 and 34 | sub \$s1, \$s2, \$s3 | \$s1 = \$s2 - \$s3                      |
|                                   | add immediate              | 8        | addi \$s1, \$s2, 6   | \$s1 = \$s2 + 6                         |
|                                   | or immediate               | 13       | ori \$s1, \$s2, 6    | \$s1 = \$s2   6                         |
| Data Transfer<br>(I format)       | load word                  | 35       | lw \$s1, 24(\$s2)    | \$s1 = Memory(\$s2+24)                  |
|                                   | store word                 | 43       | sw \$s1, 24(\$s2)    | Memory(\$s2+24) = \$s1                  |
|                                   | load byte                  | 32       | lb \$s1, 25(\$s2)    | \$s1 = Memory(\$s2+25)                  |
|                                   | store byte                 | 40       | sb \$s1, 25(\$s2)    | Memory(\$s2+25) = \$s1                  |
|                                   | load upper imm             | 15       | lui \$s1, 6          | \$s1 = 6 * 2 <sup>16</sup>              |
| Cond. Branch<br>(I & R format)    | br on equal                | 4        | beq \$s1, \$s2, L    | If (\$s1 == \$s2) go to L               |
|                                   | br on not equal            | 5        | bne \$s1, \$s2, L    | If (\$s1 != \$s2) go to L               |
|                                   | set on less than           | 0 and 42 | slt \$s1, \$s2, \$s3 | If (\$s2 < \$s3) \$s1 = 1 else \$s1 = 0 |
|                                   | set on less than immediate | 10       | slti \$s1, \$s2, 6   | If (\$s2 < 6) \$s1 = 1 else \$s1 = 0    |
| Uncond.<br>Jump<br>(J & R format) | jump                       | 2        | j 2500               | go to 10000                             |
|                                   | jump register              | 0 and 8  | jr \$t1              | go to \$t1                              |
|                                   | jump and link              | 3        | jal 2500             | go to 10000: \$ra=PC+4                  |

41

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

## MIPS Register Convention, ABI (Application Binary Interface)

| Name        | Register Number | Usage                        | Preserve on call? |
|-------------|-----------------|------------------------------|-------------------|
| \$zero      | 0               | <b>constant 0 (hardware)</b> | n.a.              |
| \$at        | 1               | reserved for assembler       | n.a.              |
| \$v0 - \$v1 | 2-3             | returned values              | no                |
| \$a0 - \$a3 | 4-7             | <b>arguments</b>             | yes               |
| \$t0 - \$t7 | 8-15            | temporaries                  | no                |
| \$s0 - \$s7 | 16-23           | saved values                 | yes               |
| \$t8 - \$t9 | 24-25           | temporaries                  | no                |
| \$gp        | 28              | global pointer               | yes               |
| \$sp        | 29              | stack pointer                | yes               |
| \$fp        | 30              | frame pointer                | yes               |
| \$ra        | 31              | return addr (hardware)       | yes               |

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

42

## ABI Sample

```
int simple_add(int a, int b)
{
    return a + b;
}
```

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

43

## ABI Sample

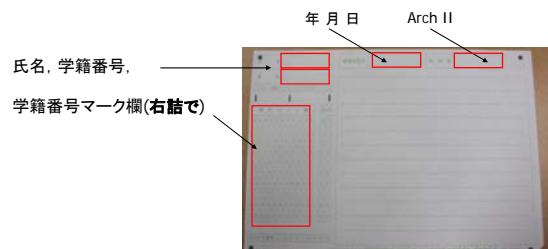
```
int simple_add(int a, int b)
{
    return a + b;
}
```

```
simple_add:
    add $v0, $a0, $a1    #
    jr $ra                 # return
```

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

44

## スキヤネットシート



Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

45

## Exercise

```
swap(int v[], int k)
{
    int temp:
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}
```

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

46

## Exercise 1

```
swap(int v[], int k)
{
    int temp:
    temp = v[k];
    v[k] = v[k+1];
    v[k+1] = temp;
}

swap:
    add $t1, $a1, $a1    #
    add $t1, $t1, $t1    # $t1 = k * 4;
    add $t1, $a0, $t1    # $t1 = &v[k];
    lw  $t0, 0($t1)      # $t0 = v[k];
    lw               #
    sw               #
    sw               #
    jr $ra             # return
```

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

47

## CISC - Complex Instruction Set Computer

- **CISC philosophy**
  - ! fixed instruction lengths
  - ! load-store instruction sets
  - ! limited addressing modes
  - ! limited operations
- DEC VAX11 Intel 80x86, ...

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005

48



## アナウンス

- 講義スライド, 講義スケジュール
  - [www.arch.cs.titech.ac.jp](http://www.arch.cs.titech.ac.jp)
- MIPS/SPIM Reference Cardは次回も利用します.

49

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005