2010-04-15

計算機アーキテクチャ 第一 (E)

2. 命令形式, アドレス指定形式

吉瀬 謙二 計算工学専攻
kise_at_cs.titech.ac.jp
W641講義室　木曜日13:20 ー 14:50

Acknowledgement

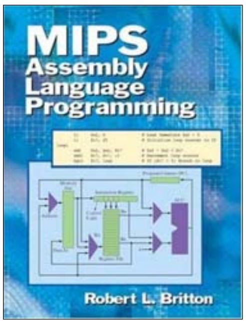
- Lecture slides for Computer Organization and Design, Third Edition, courtesy of **Professor Mary Jane Irwin**, Penn State University
- Lecture slides for Computer Organization and Design, third edition, Chapters 1-9, courtesy of **Professor Tod Amon,** Southern Utah University.

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

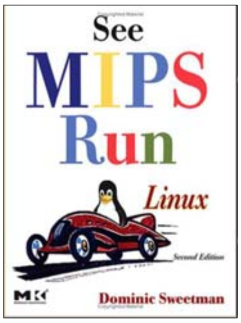
参考書

- **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
- 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, 2008
- コンピュータアーキテクチャ,
  村岡 洋一 著, 近代科学社, 1989
- 計算機システム工学,
  富田 真治, 村上 和彰 著, 昭晃堂, 1988
- コンピュータハードウエア,
  富田 真治, 中島 浩 著, 昭晃堂, 1995
- 計算機アーキテクチャ,
  橋本 昭洋 著, 昭晃堂, 1995

参考書（アセンブラに興味があれば）



MIPSのアセンブラがよくわかります．面白いです．



MIPSとLinuxの関係がわかります．お勧め．

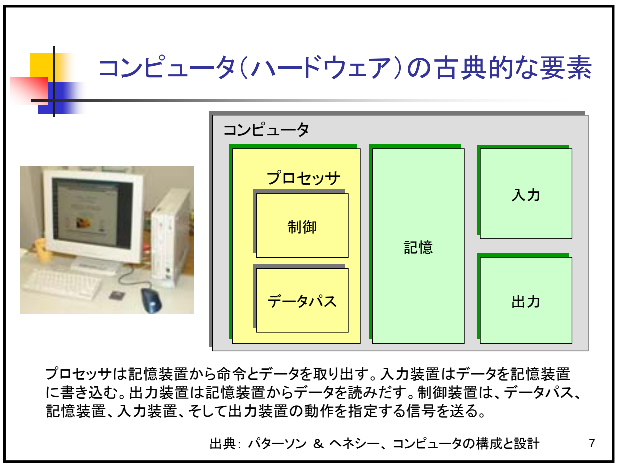
ただしい講義の受け方？

- どんどん質問する！ ＞＞ 活発な講義！
  - 難しい！
- わからない時は．．．
  - わからない顔をする！
- 不満のある時は．．．
  - 不満のある顔をする！
- わかった時は．．．
  - うなずく！

計算機アーキテクチャ 第一 (E)

2. 命令形式, アドレス指定形式

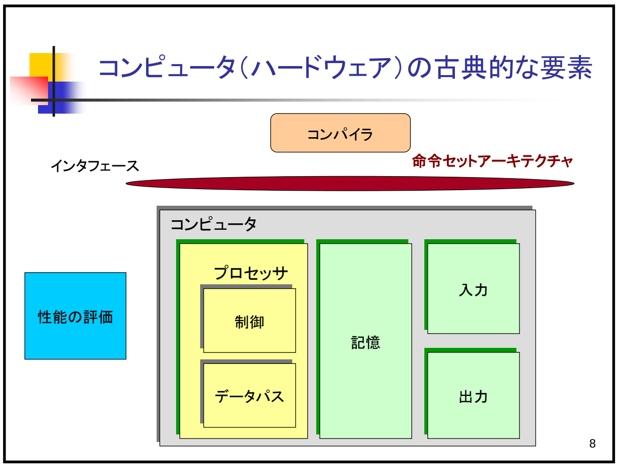
吉瀬 謙二 計算工学専攻
kise_at_cs.titech.ac.jp
W641講義室　木曜日13:20 ー 14:50



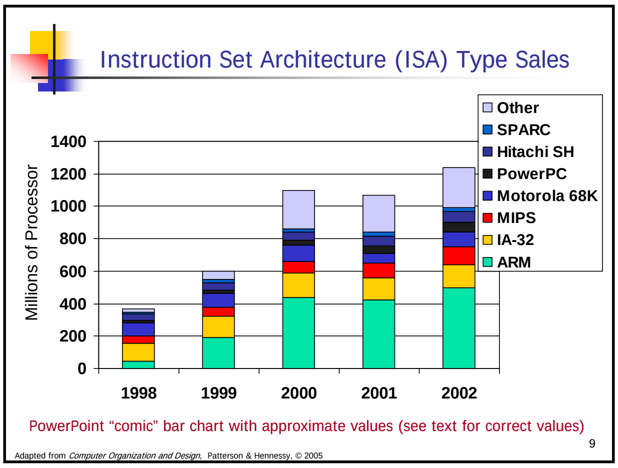
コンピュータ(ハードウェア)の古典的な要素

プロセッサは記憶装置から命令とデータを取り出す。入力装置はデータを記憶装置に書き込む。出力装置は記憶装置からデータを読みだす。制御装置は、データパス、記憶装置、入力装置、そして出力装置の動作を指定する信号を送る。

出典: パターソン & ヘネシー、コンピュータの構成と設計



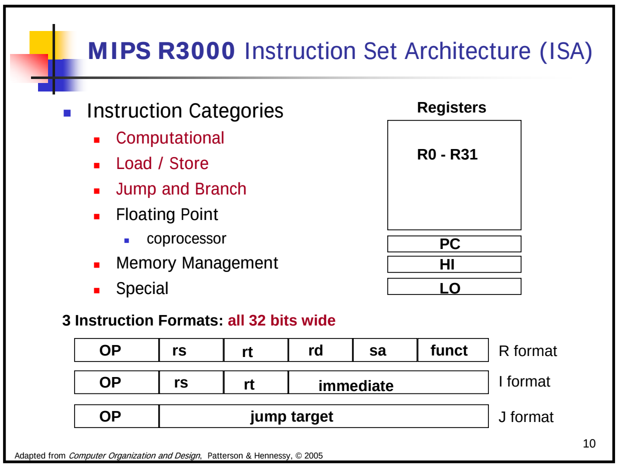
コンピュータ(ハードウェア)の古典的な要素



Instruction Set Architecture (ISA) Type Sales

PowerPoint "comic" bar chart with approximate values (see text for correct values)

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



MIPS R3000 Instruction Set Architecture (ISA)

- Instruction Categories
  - Computational
  - Load / Store
  - Jump and Branch
  - Floating Point
    - coprocessor
  - Memory Management
  - Special

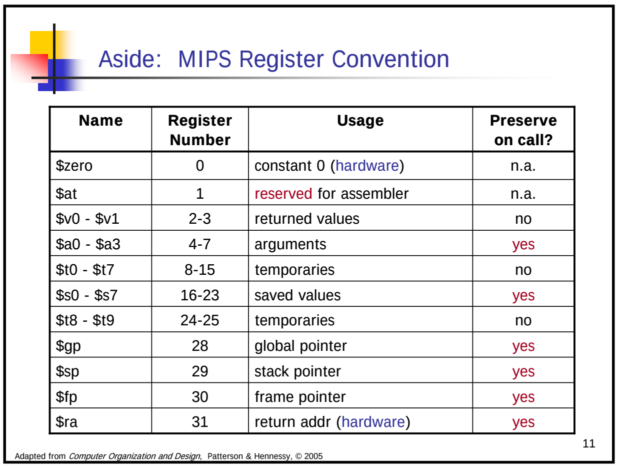
Registers

R0 - R31
PC
HI
LO

3 Instruction Formats: all 32 bits wide

OP	rs	rt	rd	sa	funct	R format
OP	rs	rt	immediate			I format
OP	jump target					J format

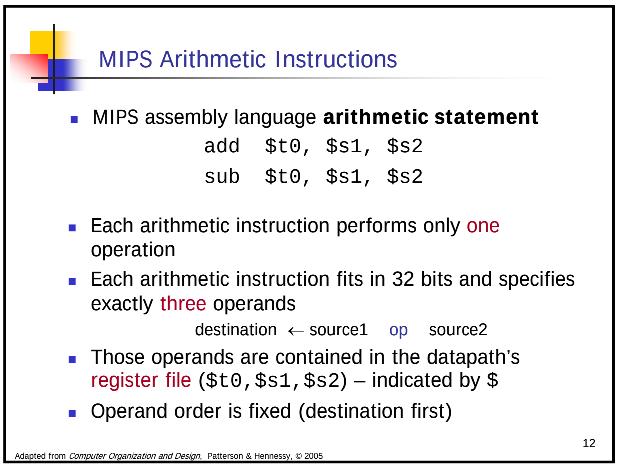
Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005



Aside: MIPS Register Convention

Name	Register Number	Usage	Preserve on call?
$zero	0	constant 0 (hardware)	n.a.
$at	1	reserved for assembler	n.a.
$v0 - $v1	2-3	returned values	no
$a0 - $a3	4-7	arguments	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



MIPS Arithmetic Instructions

- MIPS assembly language **arithmetic statement**

```
add  $t0, $s1, $s2
sub  $t0, $s1, $s2
```

- Each arithmetic instruction performs only one operation
- Each arithmetic instruction fits in 32 bits and specifies exactly three operands

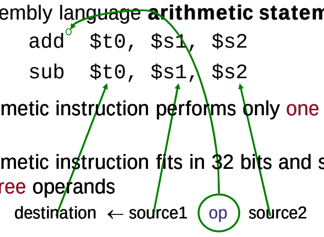
  destination ← source1 op source2

- Those operands are contained in the datapath's register file (`$t0,$s1,$s2`) – indicated by $
- Operand order is fixed (destination first)

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

MIPS Arithmetic Instructions

- MIPS assembly language **arithmetic statement**



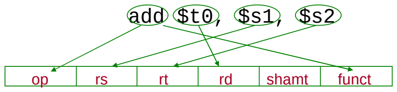
```
add  $t0, $s1, $s2
sub  $t0, $s1, $s2
```

- Each arithmetic instruction performs only one operation
- Each arithmetic instruction fits in 32 bits and specifies exactly three operands

- Operand order is fixed (destination first)
- Those operands are contained in the register file (`$t0,$s1,$s2`) – **indicated by $**

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

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):

op	rs	rt	16 bit immediate	I format

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

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.

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.

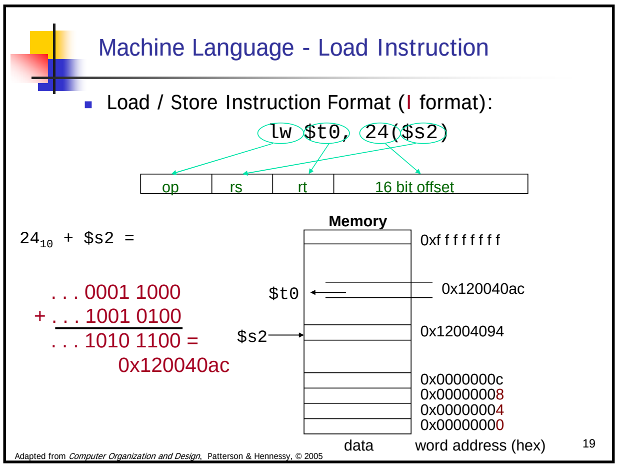
```
add  $t0, $s1, $s2      # $t0 = ( g + h)
add  $t1, $s3, $s4      #
sub  $s0, $t0, $t1      #
```

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
- 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 is limited to memory locations within a region of $\pm2^{13}$ or 8,192 words ($\pm2^{15}$ or 32,768 bytes) of the address in the base register



Machine Language - Load Instruction

- Load / Store Instruction Format (I format):

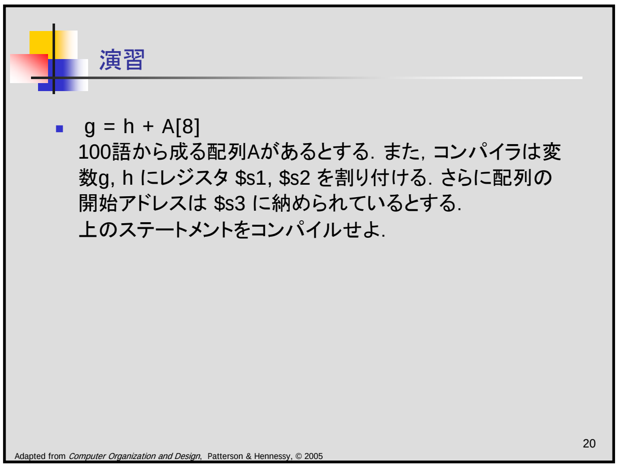
`lw $t0, 24($s2)`

op	rs	rt	16 bit offset

24_{10} + $s2 =

```
  . . . 0001 1000
+ . . . 1001 0100
  . . . 1010 1100 =
        0x120040ac
```

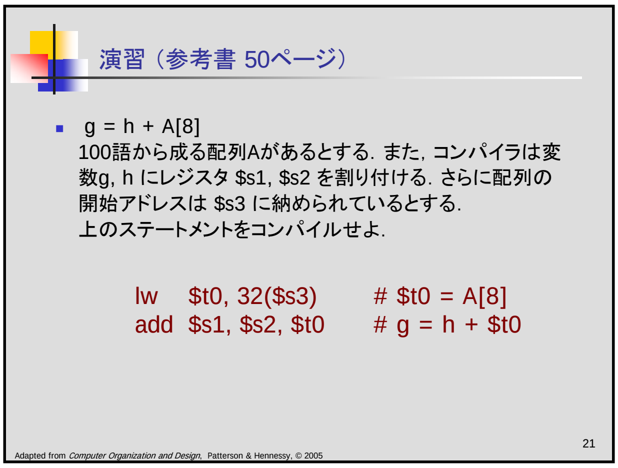
Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005 19



演習

- 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.

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005 20

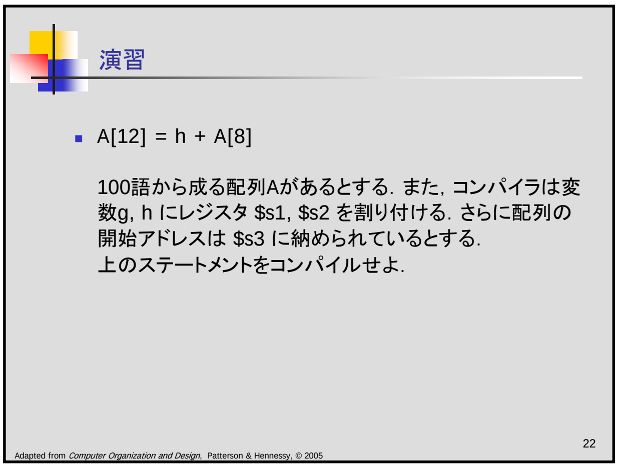


演習（参考書 50ページ）

- 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
```

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005 21

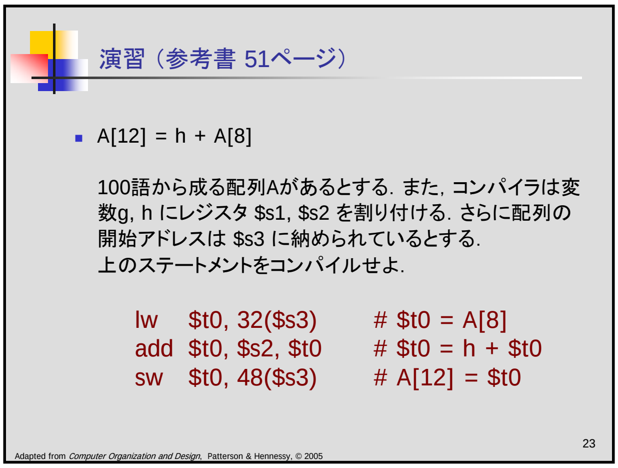


演習

- 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 22



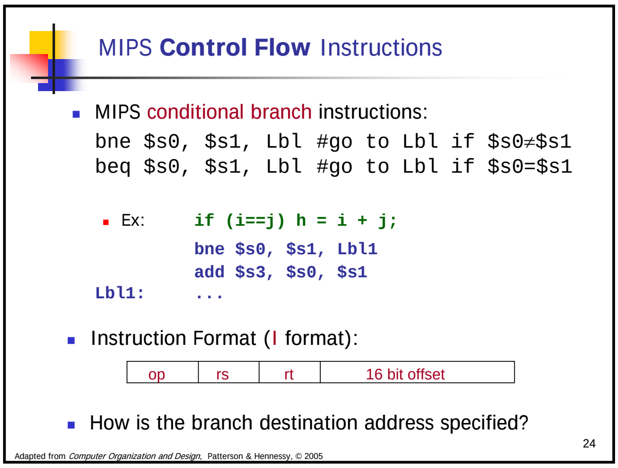
演習（参考書 51ページ）

- 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   $t0, $s2, $t0      # $t0 = h + $t0
sw    $t0, 48($s3)       # A[12] = $t0
```

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005 23



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):

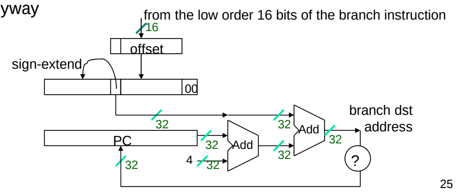
op	rs	rt	16 bit offset

- How is the branch destination address specified?

Adapted from *Computer Organization and Design*, Patterson & Hennessy, © 2005 24

Specifying Branch Destinations

- Use a register (like in lw and sw) added to the 16-bit offset
  - which register? Instruction Address Register (the PC)
    - its use is automatically implied by instruction
    - PC gets updated (PC+4) during the fetch cycle so that it holds the address of the next instruction
  - limits the branch distance to -2^{15} to $+2^{15}-1$ instructions from the (instruction after the) branch instruction, but most branches are local anyway



More Branch Instructions

- We have `beq`, `bne`, but what about other kinds of brances (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

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`)

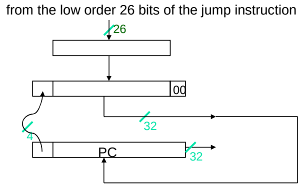
Other Control Flow Instructions

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

```
    j  label        #go to label
```

- Instruction Format (J Format):

op	26-bit address



演習（参考書 64ページ）

- f, g, h, i, j は変数である. それぞれを $s0 から $s4に割り付ける. このコードをコンパイルした結果を示せ.

  if (i == j) f = g + h; else f = g – h;

演習（参考書 64ページ）

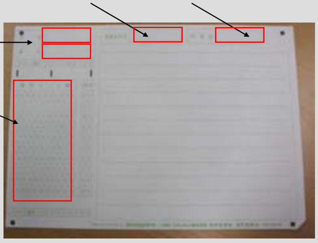
- f, g, h, i, j は変数である. それぞれを $s0 から $s4に割り付ける. このコードをコンパイルした結果を示せ.

  if (i == j) f = g + h; else f = g – h;

```
      bne   $s3, $s4, Else     # if (i!=j) goto Else
      add   $s0, $s1, $s2      # f = g + h
      j     Exit               # goto Exit
Else:
      sub   $s0, $s1, $s2      # f = g - h
Exit:
```

演習

- ループを利用して1から100までの合計値を求めるアセンブラを示せ．



年 月 日

Arch I

氏名，学籍番号，
学籍番号マーク欄(右詰で)

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

MIPS ISA So Far

Category	Instr	Op Code	Example	Meaning
Arithmetic (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 v 6
Data Transfer (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^{16}
Cond. Branch (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. Jump (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

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

今日のまとめ，MIPS R3000 ISA

- Instruction Categories
  - Computational
  - Load / Store
  - Jump and Branch
  - Floating Point
  - Memory Management
  - Special

Registers

R0 - R31

PC

HI

LO

3 Instruction Formats: all 32 bits wide

OP	rs	rt	rd	sa	funct	R format
OP	rs	rt	Immediate (16bit)			I format
OP	jump target (26bit)					J format

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

RISC - Reduced Instruction Set Computer

- **RISC philosophy** ←→ **CISC**
  Complex Instruction Set Computer
  - fixed instruction lengths
  - load-store instruction sets
  - limited addressing modes
  - limited operations
- Sun SPARC, HP PA-RISC, IBM PowerPC, Compaq Alpha, **MIPS**, …

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

講義項目

- 計算機システムの基本構成と動作原理
- (1) 命令形式，アドレス指定形式
- (2) 命令形式，データ形式
- メモリ1：半導体メモリシステム，ファイルメモリシステム
- メモリ2：記憶階層，キャッシュシステム
- メモリ3：仮想記憶システム（セグメンテーション，ページング，等）
- メモリ4：主記憶とファイルメモリの管理，多重仮想記憶，記憶保護
- 割り込み1：割り込みの必要性，割り込みの種類
- 割り込み2：割り込み処理の流れ
- 入出力制御1：チャネル，チャネルプログラム方式
- 入出力制御2：入出力動作の流れ，チャネル動作の効率化
- 入出力制御3：チャネルの種類，通信制御

レポートと期末試験により評価