2012-10-15

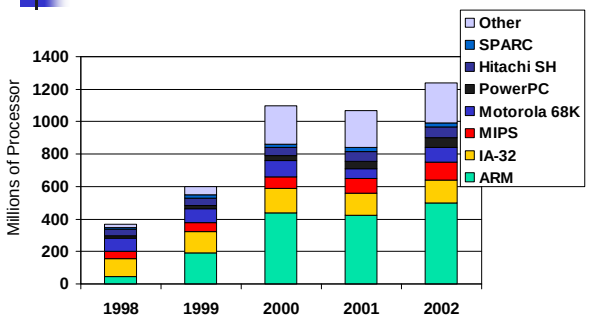
2012年 後学期

計算機アーキテクチャ 第二 (O)

RISC vs. CISC

RISC命令セットの例とその動作

Instruction Set Architecture (ISA) Type Sales



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

RISC - Reduced Instruction Set Computer

- **RISC philosophy**
  - fixed instruction lengths
  - load-store instruction sets
  - limited addressing modes
  - limited operations
- Sun SPARC, HP PA-RISC, IBM PowerPC, Compaq Alpha, **MIPS**, …

Design goals: speed, cost (design, fabrication, test, packaging), size, power consumption, reliability, memory space (embedded systems)

MIPS R3000 Instruction Set Architecture (ISA)

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

Registers

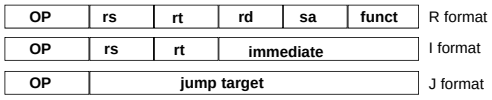
R0 - R31

PC

HI

LO

3 Instruction Formats: all 32 bits wide



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

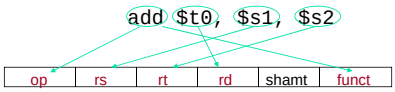
- Operand order is fixed (destination first)
- Those operands are all contained in the datapath's register file (`$t0,$s1,$s2`) – **indicated by $**

MIPS Register Convention, ABI (Application Binary Interface)

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

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

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）

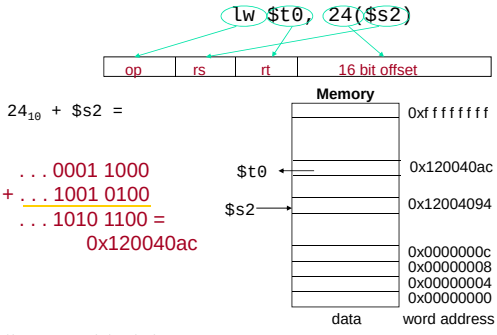
- f = (g + h) – (i + j)

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     #
```


Machine Language - Load Instruction

- Load/Store Instruction Format (I format):



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

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.

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
```


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.

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
```


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?

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
```

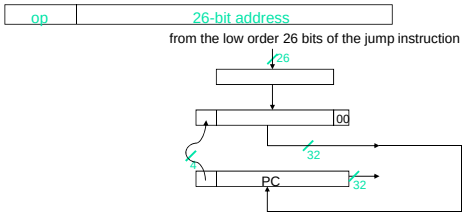
  - less than or equal to `ble $s1, $s2, Label`
  - greater than `bgt $s1, $s2, Label`
  - great than or equal to `bge $s1, $s2, Label`
- Such branches are included in the instruction set as pseudo instructions - recognized (and expanded) by the assembler
  - Its why the assembler needs a reserved register (`$at`)

Other Control Flow Instructions

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

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

- Instruction Format (J Format):



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:
```


Instructions for Accessing Procedures

- MIPS procedure call instruction:

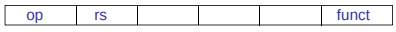
```
jal  ProcedureAddress     #jump and link
```

- 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):

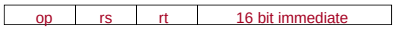


MIPS **Immediate** Instructions

- 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):



I format

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

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

MIPS Register Convention, ABI (Application Binary Interface)

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

ABI Sample

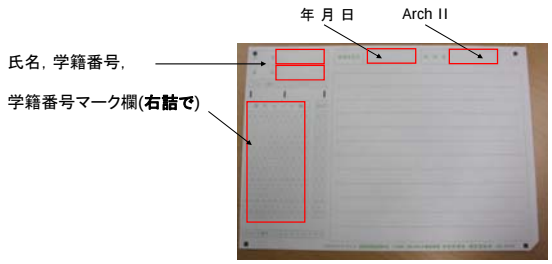
```
int simple_add(int a, int b)
{
  return a + b;
}
```


ABI Sample

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

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


スキャネットシート



Exercise

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


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
```

sll $t1, $a1, 2

CISC - Complex Instruction Set Computer

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

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
- MIPS/SPIM Reference Cardは次回も利用します．