Course number: CSC.T341

コンピュータ論理設計
Computer Logic Design

8. 命令セットアーキテクチャ:ロードストア命令と分岐命令
Instruction Set Architecture: Load/Store and Branch Instructions

吉瀬 謙二　情報工学系
Kenji Kise, Department of Computer Science
kise _at_ c.titech.ac.jp　www.arch.cs.titech.ac.jp/lecture/CLD/
W621 講義室　　月 10:45-12:15, 木 9:00-12:15

MIPS Memory Access Instructions

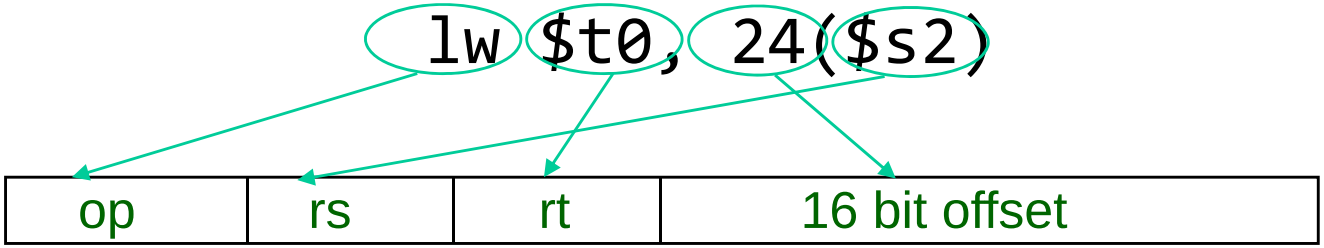
- MIPS has two basic data transfer instructions for accessing memory

```
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 $\pm 2^{13}$ or 8,192 words ($\pm 2^{15}$ or 32,768 bytes) of the address in the base register

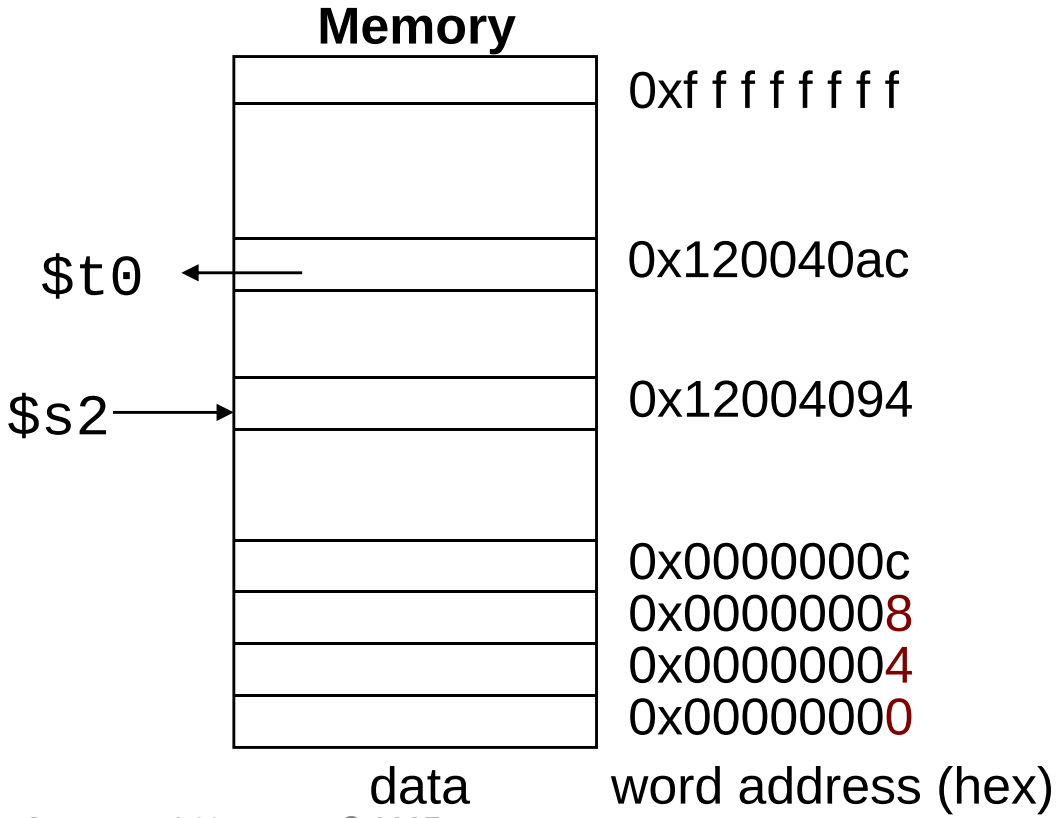
Machine Language - Load Instruction

- Load / Store Instruction Format (I format):



24_{10} + $s2 =

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



Example（例題）

- $g = h + A[8]$

  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．

Answer

- 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
```

Example（例題）

- 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．

Answer

- 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
```

MIPS Memory Access Instructions

Load Byte	lb	I	R[rt]={24'b0, M[R[rs]+ZeroExtImm](7:0)}	(3)	20
Load Byte Unsigned	lbu	I	R[rt]={24'b0, M[R[rs]+SignExtImm](7:0)}	(2)	24
Load Halfword	lh	I	R[rt]={16'b0, M[R[rs]+ZeroExtImm](15:0)}	(3)	25
Load Halfword Unsigned	lhu	I	R[rt]={16'b0, M[R[rs]+SignExtImm](15:0)}	(2)	25
~~Load Upper Imm.~~	~~lui~~	~~I~~	~~R[rt]={imm,16'b0}~~		~~f~~
Load Word	lw	I	R[rt]=M[R[rs]+SignExtImm]	(2)	23
~~Load Immediate~~	~~li~~	~~P~~	~~R[rd]=immediate~~		
~~Load Address~~	~~la~~	~~P~~	~~R[rd]=immediate~~		
Store Byte	sb	I	M[R[rs]+SignExtImm] (7:0)=R[rt](7:0)	(2)	28
Store Halfword	sh	I	M[R[rs]+SignExtImm] (15:0)=R[rt](15:0)	(2)	29
Store Word	sw	I	M[R[rs]+SignExtImm]=R[rt]	(2)	2b

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
```

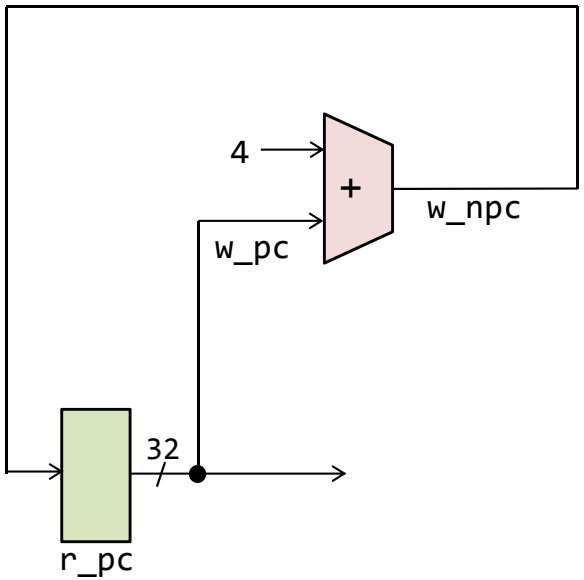
```
Ex:   if (i==j) h = i + j;

      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?

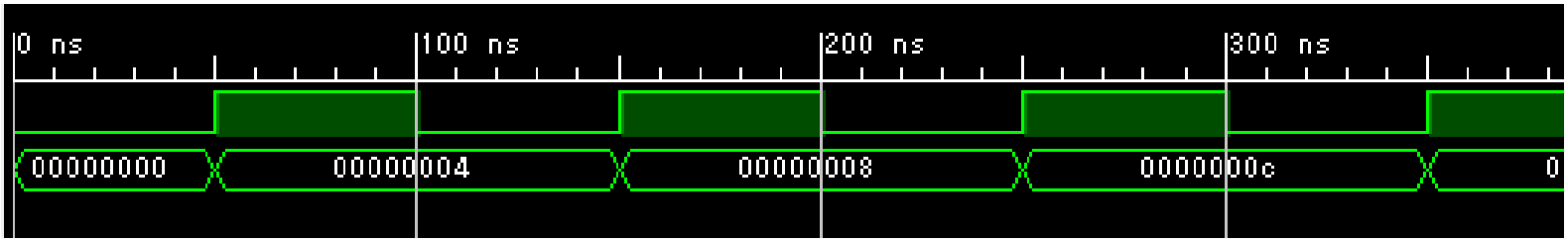
Sample Circuit 1



code081.v

```
module m_top ();
  reg r_clk=0; initial forever #50 r_clk = ~r_clk;
  wire [31:0] w_pc;
  m_main m_main0 (r_clk, w_pc);
  always@(*) #1 $write("%3d %x¥n", $time, w_pc);
endmodule

module m_main (w_clk, w_pc);
  input  wire w_clk;
  output wire [31:0] w_pc;
  reg [31:0] r_pc = 0;

  assign w_pc = r_pc;
  wire [31:0] w_npc = w_pc + 4;
  always@(posedge w_clk) r_pc <= w_npc;
endmodule
```

```
  1 00000000
 51 00000004
151 00000008
251 0000000c
351 00000010
451 00000014
551 00000018
651 0000001c
751 00000020
851 00000024
951 00000028
```



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):

More Branch Instructions

- 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`

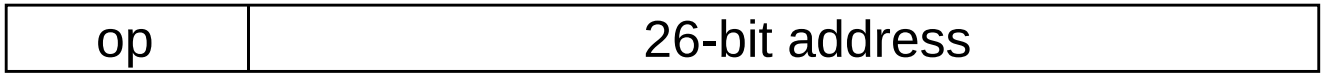
- Such branches are included in the instruction set as pseudo instructions - recognized (and expanded) by the assembler
  - Its why the assembler needs a reserved register (`$at`)

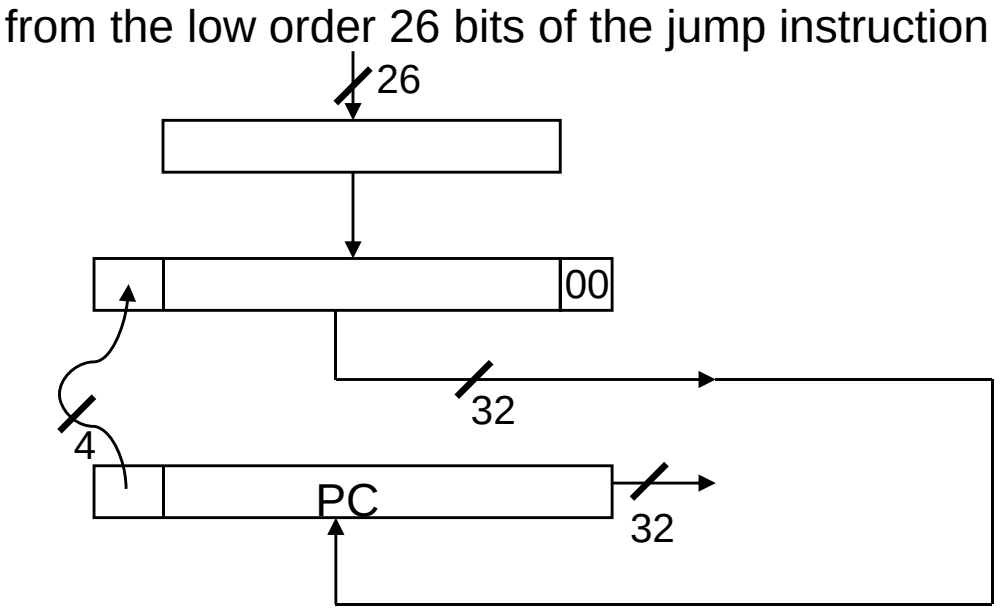
Other Control Flow Instructions

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

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

- Instruction Format (J Format):





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 Register Convention, ABI(Application Binary Interface)

Name	Register Number	Usage	Preserve on call?
$zero	0	The Constant Value 0	N.A.
$at	1	Assembler Temporary	No
$v0 - $v1	2-3	Returned Values	No
$a0 - $a3	4-7	Arguments	No
$t0 - $t7	8-15	Temporaries	No
$s0 - $s7	16-23	Saved Temporaries	Yes
$t8 - $t9	24-25	Temporaries	No
$k0 - $k1	26-27	Reserved for OS Kernel	No
$gp	28	Global Pointer	Yes
$sp	29	Stack Pointer	Yes
$fp	30	Frame Pointer	Yes
$ra	31	Return Address	Yes

MIPS Control Flow Instructions

Set Less Than	slt	R	R[rd]=(R[rs]<R[rt])?1:0		0/2a
Set Less Than Imm.	slti	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)	a
Set Less Than Imm. Unsign.	sltiu	I	R[rt]=(R[rs]<SignExtImm)?1:0	(2)(6)	b
Set Less Than Unsigned	sltu	R	R[rd]=(R[rs]<R[rt])?1:0	(6)	0/2b
Branch On Equal	beq	I	if(R[rs]==R[rt]) PC=PC+4+BranchAddr	(4)	4
Branch On Not Equal	bne	I	if(R[rs]!=R[rt]) PC=PC+4+BranchAddr	(4)	5
Branch Less Than	blt	P	if(R[rs]<R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than	bgt	P	if(R[rs]>R[rt]) PC=PC+4+BranchAddr		
Branch Less Than Or Equal	ble	P	if(R[rs]<=R[rt]) PC=PC+4+BranchAddr		
Branch Greater Than Or Equal	bge	P	if(R[rs]>=R[rt]) PC=PC+4+BranchAddr		
Jump	j	J	PC=JumpAddr	(5)	2
Jump And Link	jal	J	R[31]=PC+4; PC=JumpAddr	(5)	2
Jump Register	jr	R	PC=R[rs]		0/08
Jump And Link Register	jalr	R	R[31]=PC+4; PC=R[rs]		0/09

(1) May cause overflow exception
(2) SignExtImm ={16{immediate[15]},immediate }
(3) ZeroExtImm ={16{1b'0},immediate }
(4) BranchAddr = {14{immediate[15]},immediate,2'b0 }
(4) JumpAddr = {PC[31:28], address, 2'b0 }

Example（例題）

- 次の関数をコンパイルせよ.

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

Answer

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

Example（例題）

- 次の関数をコンパイルせよ.

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