2010-04-22

計算機アーキテクチャ 第一 (E)

3. 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.

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 眞治, 村上 和彰 著, 昭晃堂, 1988
- コンピュータハードウエア, 富田 眞治, 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

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

Aside: How About Larger Constants?

- We'd also like to be able to load a 32 bit constant into a register, for this we must use two instructions
- a new "load upper immediate" instruction

```
lui $t0, 1010101010101010
```

16	0	8	1010101010101010

- Then must get the lower order bits right, use

```
ori $t0, $t0, 1010101010101010
```

1010101010101010	0000000000000000
0000000000000000	1010101010101010
1010101010101010	1010101010101010

Aside: Loading and Storing Bytes

- MIPS provides special instructions to move bytes

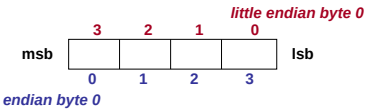
```
lb   $t0, 1($s3)  #load byte from memory
sb   $t0, 6($s3)  #store byte to  memory
```

op	rs	rt	16 bit offset

- What 8 bits get loaded and stored?
  - load byte places the byte from memory in the rightmost 8 bits of the destination register
    - what happens to the other bits in the register?
  - store byte takes the byte from the rightmost 8 bits of a register and writes it to a byte in memory
    - what happens to the other bits in the memory word?

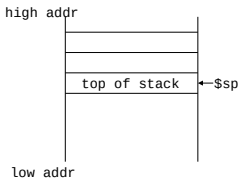
Byte Addresses

- Since 8-bit bytes are so useful, most architectures address individual bytes in memory
  - The memory address of a word must be a multiple of 4 (alignment restriction)
- **Big Endian:**
  - leftmost byte is word address
    IBM 360/370, Motorola 68k, **MIPS**, SPARC, HP PA
- **Little Endian:**
  - rightmost byte is word address
    Intel 80x86, DEC Vax, DEC Alpha (Windows NT)



Aside: **Spilling Registers**

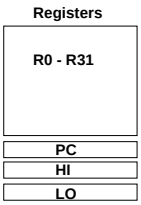
- What if the **callee** needs more registers? What if the procedure is **recursive**?
  - uses a **stack** – a last-in-first-out queue – in memory for passing additional values or saving (recursive) return address(es)



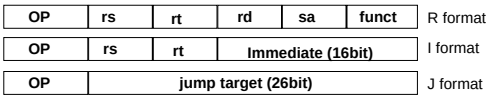
- One of the general registers, **$sp**, is used to address the stack (which "grows" from high address to low address)
  - add data onto the stack – **push**
    $sp = $sp – 4
    data on stack at new $sp
  - remove data from the stack – **pop**
    data from stack at $sp
    $sp = $sp + 4

MIPS R3000 ISA

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



3 Instruction Formats: all 32 bits wide



Instructions for Accessing Procedures

- MIPS procedure call instruction:
  `jal Procedure-Address  #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):

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

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

演習

- アセンブラを示せ.

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

演習

```
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 (shift left logical) $t1, $a1, 2

Exercise 2

```
void max (int v[], int n)
{
  int i;
  for (i = 1; i < n; i +=1){
    if (v[i-1] > v[i]) swap(v,i-1);
  }
}
```


Exercise 3

```
void sort (int v[], int n)
{
  int i, j;
  for (i = 0; i < n; i +=1){
    for (j=i-1; j>=0 && v[j]>v[j+1]; j-=1) swap(v,j);
  }
}
```


計算機アーキテクチャ 第一 (E)

データ形式

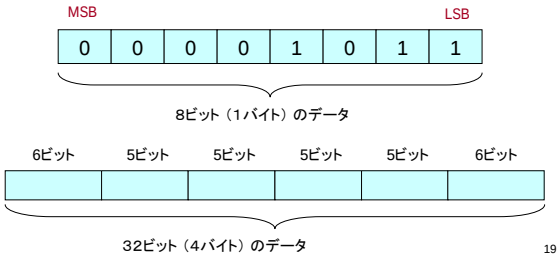
吉瀬 謙二 計算工学専攻
kise_at_cs.titech.ac.jp
W641講義室 木曜日13:20 ー 14:50

整数(integer)の表現

- コンピュータは決まったビット幅を単位としてデータを処理.
  - **例えば, 8ビットコンピュータ は, 8ビット単位で処理**
- nビットの整数表現は, 2^n (2のn乗)種類の整数を表現できる. (しか表現できない!)
  - 8ビットであれば, 2^8 = 256 種類の整数.
  - **表現できる範囲には限りがある.**
  - **効率の良い表現を利用して, 資源を有効に活用する!**
- 整数表現
  - **符号なし表現**
  - **符号つき絶対値表現**
  - **2の補数表現**

データの表現

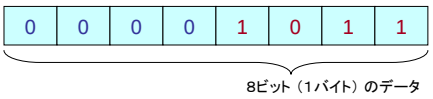
- MSB: Most Significant Bit, 最上位の桁
- LSB: Least Significant Bit, 最下位の桁



19

整数：符号なし表現

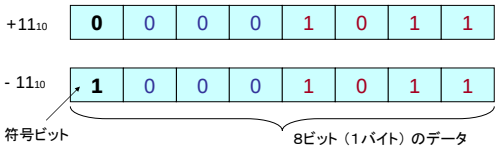
- ある整数mを2進数で表現する.
  - 11_{10} であれば，1011_2 として下位ビットを決める.
  - 上位の残ったビットを0で埋める.
  - 8ビットであれば，0～255までの256個の整数を表現できる.
- 簡潔な表現方法.
- 負数を表現できない！



20

整数：符号つき絶対値表現（1）

- ある整数mを2進数で表現する.
  - 11_{10} であれば，1011_2 として下位ビットを決める.
  - ただし，最上位ビットを用いて符号を表す（**符号ビット**）.
    - mが正ならば，符号ビットを0，負ならば1とする.
  - 残ったビットを0で埋める.
- **符号無し表現の自然な拡張**
- 8ビットであれば，- 127 ～127 までの**255個**の整数を表現できる？



21

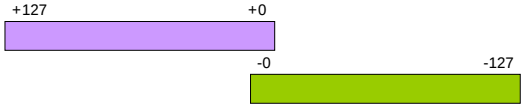
整数：符号つき絶対値表現（2）

- 8ビットであれば，- 127 ～127 までの**255個**の整数を表現できる？
- どうして256個の数を表現できないのか？

22

整数：符号つき絶対値表現（2）

- 8ビットであれば，- 127 ～127 までの**255個**の整数を表現できる？
- どうして256個の数を表現できないのか？
- それは，ゼロに正と負の2つがあるから！
  - プログラマが問題を起こす原因となる.
  - **符号つき絶対値表現が利用されることは少ない！**



23

整数：符号つき絶対値表現（3）

- もう一度，8ビット時の，符号つき絶対値表現を確認

128種類	$0000\ 0000_2 = +0_{10}$	$1000\ 0000_2 = -0_{10}$
	$0000\ 0001_2 = +1_{10}$	$1000\ 0001_2 = -1_{10}$
	$0000\ 0010_2 = +2_{10}$	$1000\ 0010_2 = -2_{10}$
	…	…
	$0111\ 1101_2 = +125_{10}$	$1111\ 1101_2 = -125_{10}$
	$0111\ 1110_2 = +126_{10}$	$1111\ 1110_2 = -126_{10}$
	$0111\ 1111_2 = +127_{10}$	$1111\ 1111_2 = -127_{10}$

符号つき絶対値表現が利用されることは少ない！

24

整数：2の補数表現（1）

- 多くの計算機では2の補数 (two's complement) 表現が利用される.
- 2の補数の利点
  - 最上位ビットのみで正負判定が可能.
  - 正負の反転が容易.
  - ビット幅の異なるデータへの変換が容易.
  - 符号なし整数と同じハードウェアで加算を実装できる.

25

整数：2の補数表現（2）

- その前に，1の補数 (one's complement)
  - 全てのビットを反転することで，マイナスを表現

128種類

$0000\ 0000_2 = +0_{10}$	$1111\ 1111_2 = -0_{10}$
$0000\ 0001_2 = +1_{10}$	$1111\ 1110_2 = -1_{10}$
$0000\ 0010_2 = +2_{10}$	$1111\ 1101_2 = -2_{10}$
…	…
$0111\ 1101_2 = +125_{10}$	$1000\ 0010_2 = -125_{10}$
$0111\ 1110_2 = +126_{10}$	$1000\ 0001_2 = -126_{10}$
$0111\ 1111_2 = +127_{10}$	$1000\ 0000_2 = -127_{10}$

26

整数：2の補数表現（3）

- 2の補数
  - （1の補数で表された数に1を加えたもの）を負の数とする.

$0000\ 0000_2 = +0_{10}$	$1111\ 1111_2 = -0_{10}$	
$0000\ 0001_2 = +1_{10}$	$1111\ 1110_2 = -1_{10}$	$1111\ 1111_2 = -1_{10}$
$0000\ 0010_2 = +2_{10}$	$1111\ 1101_2 = -2_{10}$	$1111\ 1110_2 = -2_{10}$
…	…	…
$0111\ 1101_2 = +125_{10}$	$1000\ 0010_2 = -125_{10}$	$1000\ 0011_2 = -125_{10}$
$0111\ 1110_2 = +126_{10}$	$1000\ 0001_2 = -126_{10}$	$1000\ 0010_2 = -126_{10}$
$0111\ 1111_2 = +127_{10}$	$1000\ 0000_2 = -127_{10}$	$1000\ 0001_2 = -127_{10}$
	負の数の1の補数表現	$1000\ 0000_2 = -128_{10}$
		負の数の2の補数表現

2の補数では，−128 ～ 127 までの数を表現できる.

27

整数：2の補数表現（4）

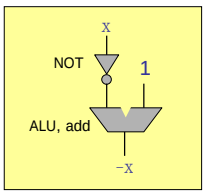
- 2の補数
  - 1の補数で表された数（ビットの反転）に1を加えたものを負の数とする.

$0000\ 0000_2 = +0_{10}$	$1111\ 1111_2 = -0_{10}$	負の数の2の補数表現
$0000\ 0001_2 = +1_{10}$	$1111\ 1110_2 = -1_{10}$	$1111\ 1111_2 = -1_{10}$
$0000\ 0010_2 = +2_{10}$	$1111\ 1101_2 = -2_{10}$	$1111\ 1110_2 = -2_{10}$
…	…	…
$0111\ 1101_2 = +125_{10}$	$1000\ 0010_2 = -125_{10}$	$1000\ 0011_2 = -125_{10}$
$0111\ 1110_2 = +126_{10}$	$1000\ 0001_2 = -126_{10}$	$1000\ 0010_2 = -126_{10}$
$0111\ 1111_2 = +127_{10}$	$1000\ 0000_2 = -127_{10}$	$1000\ 0001_2 = -127_{10}$
		$1000\ 0000_2 = -128_{10}$

28

整数：2の補数表現（5）

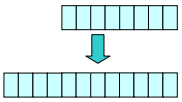
- 2の補数
  - 1の補数で表された数（ビットの反転）に1を加えたものを負の数とする.
- 2の補数表現では，正負の反転を簡潔に実現できる！
  - 正数から負数への変換
    - 2進数表現の1と0を反転する.
    - 得られたデータに1を加える.
  - 負数から正数への変換
    - 2進数表現の1と0を反転する.
    - 得られたデータに1を加える.



29

整数：2の補数表現（6）

- 符号拡張
  - ビット幅の異なるデータへの変換
  - 例：8ビットから12ビットのデータへの変換
- 符号拡張の処理
  - ビット幅を増やすときには，最上位ビットの値で補填すればよい.

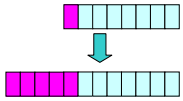


$1111\ 1111_2 = -1_{10}$	符号拡張	$1111\ 1111\ 1111_2 = -1_{10}$
$1111\ 1110_2 = -2_{10}$		$1111\ 1111\ 1110_2 = -2_{10}$
…		…
$1000\ 0011_2 = -125_{10}$		$1111\ 1000\ 0011_2 = -125_{10}$
$1000\ 0010_2 = -126_{10}$		$1111\ 1000\ 0010_2 = -126_{10}$
$1000\ 0001_2 = -127_{10}$		$1111\ 1000\ 0001_2 = -127_{10}$
$1000\ 0000_2 = -128_{10}$		$1111\ 1000\ 0000_2 = -128_{10}$

30

整数：2の補数表現（7）

- 符号拡張
  - ビット幅の異なるデータへの変換
  - 例：8ビットから12ビットのデータへの変換
- 符号拡張の処理
  - ビット幅を増やすときには，最上位ビットの値で補填すればよい．



計算機アーキテクチャ 第一 (E)

3．命令形式，アドレス指定形式

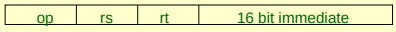
吉瀬 謙二 計算工学専攻
kise_at_cs.titech.ac.jp
W641講義室　木曜日13:20 ー 14:50

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):

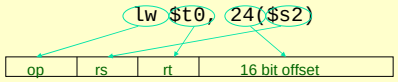


I format

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

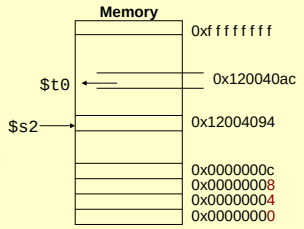
Machine Language - Load Instruction

- Load / Store Instruction Format (I format):



24_{10} + $s2 =

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



MIPS **Control Flow** Instructions

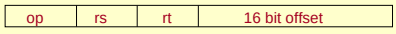
- MIPS conditional branch instructions:

```
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:   ...
```

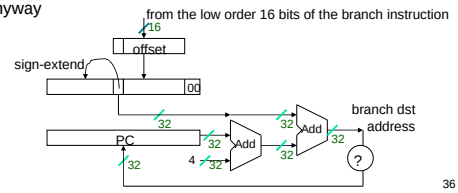
- Instruction Format (I format):

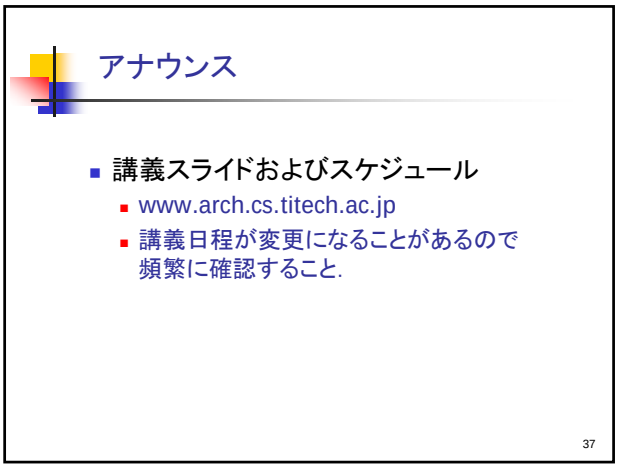


- How is the branch destination address specified?

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




アナウンス
講義スライドおよびスケジュール
www.arch.cs.titech.ac.jp
講義日程が変更になることがあるので
頻繁に確認すること.
37