

MIPS arithmetic and logic instructions

<u>Instruction</u>	<u>Example</u>	<u>Meaning</u>	<u>Comments</u>
add	add \$1,\$2,\$3	$\$1 = \$2 + \$3$	3 operands; exception possible
subtract	sub \$1,\$2,\$3	$\$1 = \$2 - \$3$	3 operands; exception possible
add immediate	addi \$1,\$2,100	$\$1 = \$2 + 100$	+ constant; exception possible
multiply	mult \$2,\$3	Hi, Lo = $\$2 \times \3	64-bit signed product
divide	div \$2,\$3	Lo = $\$2 \div \3 , Hi = $\$2 \bmod \3	Lo = quotient, Hi = remainder
Move from Hi	mfhi \$1	$\$1 = \text{Hi}$	get a copy of Hi
Move from Lo	mflo \$1	$\$1 = \text{lo}$	

<u>Instruction</u>	<u>Example</u>	<u>Meaning</u>	<u>Comment</u>
and	and \$1,\$2,\$3	$\$1 = \$2 \& \$3$	Logical AND
or	or \$1,\$2,\$3	$\$1 = \$2 \$3$	Logical OR
xor	xor \$1,\$2,\$3	$\$1 = \$2 \oplus \$3$	Logical XOR
nor	nor \$1,\$2,\$3	$\$1 = \sim(\$2 \$3)$	Logical NOR

ECE468 Lec4.3

Adapted from © VC 1997 © & UCB'97

Example

E.g. $f = (g+h) - (i+j)$,
assuming f, g, h, i, j be assigned to \$1, \$2, \$3, \$4, \$5

```

add $7, $2, $3
add $8, $4, $5
sub $1, $7, $8
    
```

ECE468 Lec4.4

Adapted from © VC 1997 © & UCB'97

MIPS data transfer and branch instructions

<u>Instruction</u>	<u>Comment</u>
SW 500(\$4), \$3	Store word
SH 502(\$2), \$3	Store half
SB 41(\$3), \$2	Store byte
LW \$1, 30(\$2)	Load word
LH \$1, 40(\$3)	Load half a word
LB \$1, 40(\$3)	Load byte

Example

Assume A is an array of 100 words, and compiler has associated the variables g and h with the register \$1 and \$2. Assume the base address of the array is in \$3. Translate

$$g = h + A[8]$$

```
lw $4, 8($3); $3 <-- A[8]
add $1, $2, $4
```



```
lw $4, 32($3);
add $1, $2, $4
```

A[12] = h+A[8]

SW \$1, 48(\$3)

Example

Assume A is an array of 100 words, and compiler has associated the variables g, h, and i with the registers \$1, \$2, \$5. Assume the base address of the array is in \$3. Translate

$$g = h + A[i]$$

```
addi $6, $0, 4;    $6 = 4
mult $5, $6;       Hi,Lo = i*4
mflo $6            $6 = i*4

add $4, $3, $6;    $4 = A[i]
```

MIPS jump, branch, compare instructions

<i>Instruction</i>	<i>Example</i>	<i>Meaning</i>
branch on equal	beq \$1,\$2,100	if (\$1 == \$2) go to PC+4+100 <i>Equal test; PC relative branch</i>
branch on not eq.	bne \$1,\$2,100	if (\$1 != \$2) go to PC+4+100 <i>Not equal test; PC relative</i>
<i>beq, bne, blt, blez, bgt, bgez</i>		
set on less than	slt \$1,\$2,\$3	if (\$2 < \$3) \$1=1; else \$1=0 <i>Compare less than; 2's comp.</i>
set less than imm.	slti \$1,\$2,100	if (\$2 < 100) \$1=1; else \$1=0 <i>Compare < constant; 2's comp.</i>
jump	j 10000	go to 10000 <i>Jump to target address</i>
jump register	jr \$31	go to \$31 <i>For switch, procedure return</i>
jump and link	jal 10000	\$31 = PC + 4; go to 10000 <i>For procedure call</i>

Example

```
        if (i==j) go to L1;  
        f = g+ h;  
L1:    f = f - i;
```

Assuming f, g, h, i, j ~ \$1, \$2, \$3, \$4, \$5

```
        beq $4, $5, L1  
        add $1, $2, $3  
L1:    sub $1, $1, $4
```

Example

```
Loop:  g = g +A[i];  
        i = i+ j;  
        if (i != h) go to Loop;
```

Assuming variables g, h, i, j ~ \$1, \$2, \$3, \$4 and base address of array is in \$5

```
Loop:  add $7, $3, $3  
        add $7, $7, $7  
        add $7, $7, $5  
        lw $6, 0($7)  
        add $1, $1, $6; g= g+A[i]  
        add $3, $3, $4  
        bne $3, $2, Loop;
```

Example

while (A[i]==k)

 i = i+j;

Assume i, j, and k ~ \$17, \$18, \$19 and base of A is in \$3

 Loop: add \$20, \$17, \$17

 add \$20, \$20, \$20

 add \$20, \$20, \$3

 lw \$21,0(\$20)

 bne \$21, \$19, Exit

 add \$17, \$17, \$18

 j Loop

 Exit:

Example

while (A[i]==k)

 i = i+j;

Assume i, j, and k ~ \$17, \$18, \$19 and base of A is in \$3

 Loop: add \$20, \$17, \$17

 add \$20, \$20, \$20

 add \$20, \$20, \$3

 lw \$21,0(\$20)

 bne \$21, \$19, Exit

 add \$17, \$17, \$18

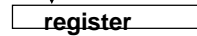
 j Loop

 Exit:

MIPS Addressing Modes/Instruction Formats

R-format:

Register (direct)

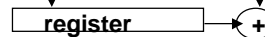
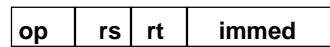


I-format:

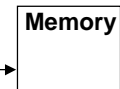
Immediate



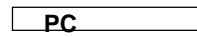
Base+index



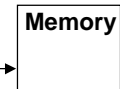
+



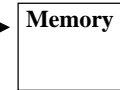
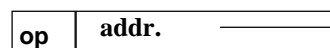
PC-relative



+



J-format:



ECE468 Lec4.13

Adapted from © VC 1997 © & UCB'97

Example

```
while (A[i]==k)
```

```
    i = i+j;
```

Assume i, j, and k ~ \$17, \$18, \$19 and base of A is in \$3

Assume the loop is placed starting at loc 8000

```
Loop: add $20, $17, $17
```

```
      add $20, $20, $20
```

```
      add $20, $20, $3
```

```
      lw $21,0($20)
```

```
      bne $21, $19, Exit
```

```
      add $17, $17, $18
```

```
      j     Loop
```

```
Exit:
```

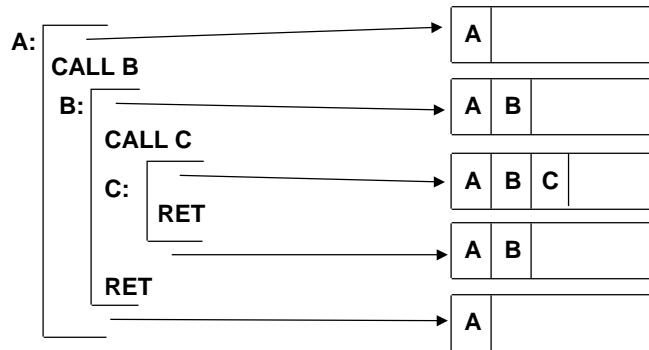
8000:	0	17	17	20	0	32
	0	20	20	20	0	32
	0	20	3	20	0	32
	35	20	21			0
	5	21	19			8
	0	17	18	17	0	32
	2		80000			

ECE468 Lec4.14

Adapted from © VC 1997 © & UCB'97

Procedure Call and Stack

Stacking of Subroutine Calls & Returns and Environments:



Some machines provide a memory stack as part of the architecture (e.g., the VAX)

Sometimes stacks are implemented via software convention (e.g., MIPS)

Example in C: swap

- Assume swap is called as a procedure
- Assume temp is register \$15; arguments v and k ~ \$16 and \$17;
- Write MIPS code

```

swap(int v[], int k)
{
    int temp;
    temp = v[k];
    v[k] = v[k+1];    sll    $18, $17, 2    ; multiply k by 4
                    addu   $18, $18, $16 ; address of v[k]
    v[k+1] = temp;   lw     $15, 0($18)    ; load v[k]
                    lw     $19, 4($18)    ; load v[k+1]
                    sw     $19, 0($18)    ; store v[k+1] into v[k]
                    sw     $15, 4($18)    ; store old v[k] into v[k+1]
}
    
```

Registers \$15, \$16, \$17, \$18, \$19 are occupied by caller ??

Example: Swap

Given a procedure swap(v, j)

Calling swap is as simple as
jal swap

jal --- jump and link
\$31 = PC+4; \$31 always store return address
goto swap

swap: MIPS

swap:

```
addi $sp,$sp, -24 ; Make room on stack for 6 registers
sw   $31, 20($sp) ; Save return address
sw   $15, 16($sp) ; Save registers on stack
sw   $16, 12($sp)
sw   $17, 8($sp)
sw   $18, 4($sp)
sw   $19, 0($sp)
....
lw   $19, 0($sp) ; Restored registers from stack
lw
lw   $15, 16($sp)
lw   $31, 20($sp) ; Restore return address
addi $sp,$sp, 24 ; restore top of stack
jr   $31 ; return to place that called swap
```

Other ISAs

- Intel 8086/88 => 80286 => 80386 => 80486 => Pentium => P6
 - 8086 few transistors to implement 16-bit microprocessor
 - tried to be somewhat compatible with 8-bit microprocessor 8080
 - successors added features which were missing from 8086 over next 15 years
 - product several different intel engineers over 10 to 15 years
 - Announced 1978
- VAX simple compilers & small code size =>
 - efficient instruction encoding
 - powerful addressing modes
 - powerful instructions
 - few registers
 - product of a single talented architect
 - Announced 1977

Machine Examples: Address & Registers

Intel 8086	2^{20} x 8 bit bytes AX, BX, CX, DX SP, BP, SI, DI CS, SS, DS IP, Flags	acc, index, count, quot stack, string code, stack, data segment
VAX 11	2^{32} x 8 bit bytes 16 x 32 bit GPRs	r15-- program counter r14-- stack pointer r13-- frame pointer r12-- argument ptr
MC 68000	2^{24} x 8 bit bytes 8 x 32 bit GPRs 7 x 32 bit addr reg 1 x 32 bit SP 1 x 32 bit PC	
MIPS	2^{32} x 8 bit bytes 32 x 32 bit GPRs 32 x 32 bit FPRs HI, LO, PC	

Homework 2, due 10/4 (Tuesday before class)

- Questions 3.2, 3.3, 3.5, 3.6, 3.7, 3.9, 3.11
- To answer question 3.7, please refer to Figure 3.13 (page 140) for register convention
- To answer 3.11, please refer to sort example in pages 166 for a skeleton of for loop