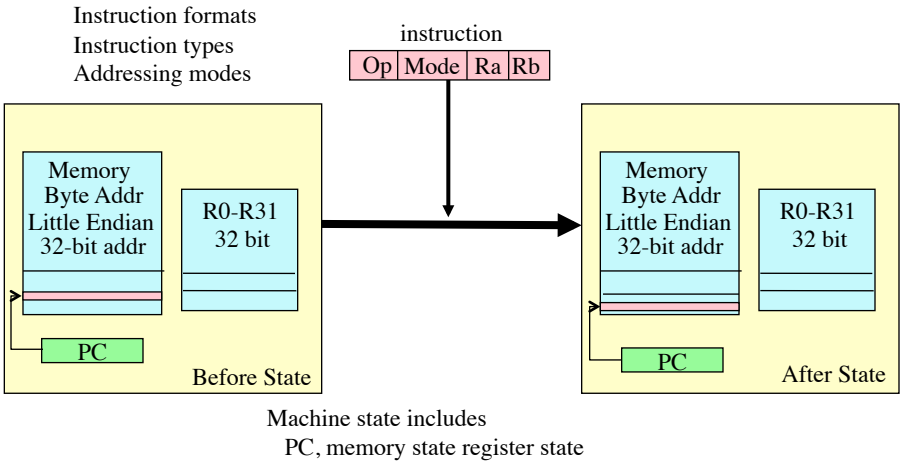
Lecture 6: Instruction Set Architectures III

- Last Time
  - Machine state (memory, computation, control)
  - Design principles
  - Instruction formats
  - Representing and addressing data
- Today
  - Take QUIZ 3 before 11:59pm today over Chapter 2 readings
  - Quiz 2: 100% - 19; 75% - 19; 50% - 13; 25% - 3
  - ISA III
    - In/out of Memory and Registers
    - Program counter (PC) control
    - MIPS assembly connected to high-level programming

ISA Design Principles

1. Simplicity favors regularity
   - e.g., instruction size, instruction formats, data formats
   - eases implementation by simplifying hardware
2. Smaller is faster
   - fewer bits to read, move, & write
   - use/reuse the register file instead of memory
3. Make the common case fast
   - e.g., small constants are common, thus immediate fields can be small
4. Good design demands compromise
   - special formats for important exceptions
   - e.g., a jump far away (beyond a small constant)

ISA Specifies How the Computer Changes State



Machine state includes
PC, memory state register state

Design Considerations:

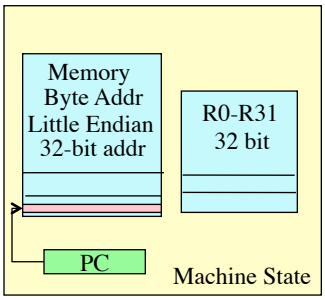
Changes to Machine State Imply Instruction Classes & Design

Changes to Memory State

- **Data representation**
- **ALU Operations**
  - arithmetic (`add, sub, mult, div`)
  - logical (`and, or, xor, srl, sra`)
  - data type conversions (`cvtf2d,cvtf2i`)
- **Data movement**
  - memory reference (`lb, lw, sb, sw`)
  - register to register (`movi2fp, movf`)

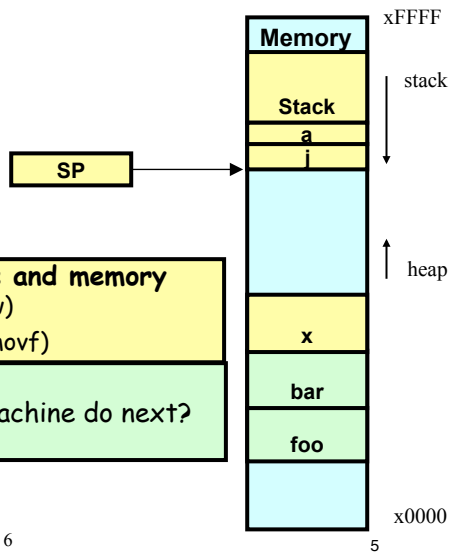
Control: what instruction to do next

- PC = ??
- tests/compare (`slt, seq`)
- branches and jumps (`beq, bne, j, jr`)
- procedure calls (`jal, jalr`)
- operating system entry (`trap`)



High Level Program in C

```
int x [10] ;
void foo(int a) {
  int j ;
  for(j=1; j<10; j++)
    x[j] = a * x [j-1];
  bar(a);
}
```



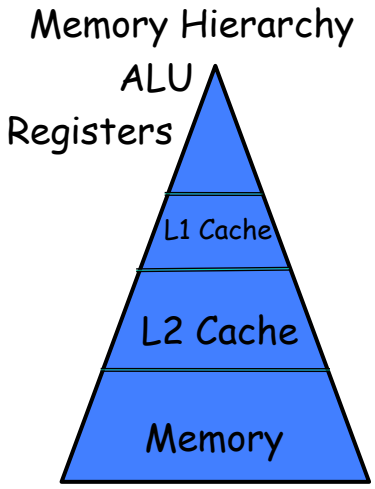
- **Data movement in/out registers and memory**
  - memory reference (lb, lw, sb, sw)
  - register to register (movi2fp, movf)
- **Control**
  - What instruction does the machine do next?

Registers

- Registers are faster than cache & memory
- Level 1 cache is faster than level 2, & so on
- Memory hierarchies are effective because programs have locality & regularity

Temporal locality: reuse of same location

Spatial locality: reuse of nearby locations



General Registers

Any register may hold values or addresses

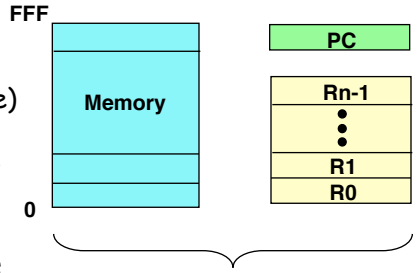
- simpler
- more orthogonal (opcode independent of register usage)
- More fast local storage
- but….addresses and data must be same size (32 bits)

How many?

- More - fewer loads and stores
- But - more instruction bits

MIPS: 32 registers

- **ISA Register conventions**
- **All ALU/Control instructions operate on registers**
- **Must load from memory!**



3-address Instruction Format

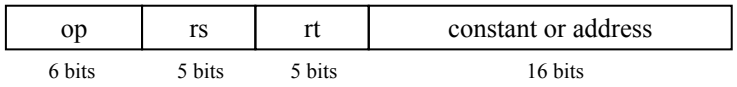
MIPS Register Conventions

$zero:	holds constant 0 (register 0)
$at:	assembler temporary (1)
$v0, $v1:	result values (2, 3)
$a0 – $a3:	arguments (4–7)
$t0 – $t9:	temporaries (8-15, 24, 25)
	All the above can be overwritten by callee
$s0 – $s7:	saved (16-23)
	Callee must save/restore all these registers
$gp:	global pointer for static data (28)
$sp:	stack pointer (29)
$fp:	frame pointer (30)
$ra:	return address (31)

In/out of Memory and Registers on MIPS

(lw, lbu, sw, sb, lhu, llui, ll)

- Uses I format instructions
- Example Load Word
  - lw $t1, 4($t2) means t1 = value at address(t2 + 4)

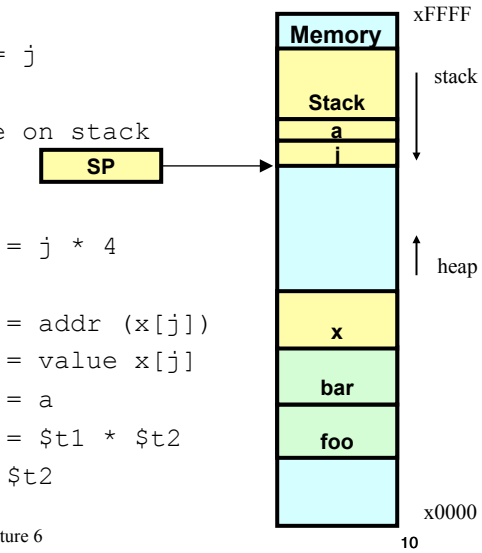


In/out of Memory and Registers on MIPS

(lw, lbu, sw, sb, lhu, llui, ll)

```
// j = j+1
lw $t0, 0($sp)    // $t0 = j
addi $t0, $t0, 1 // j++
sw $t0, 0($sp)    // store on stack

// a = x [j] * a
sll $t1, $t0, 2    // $t1 = j * 4
lw  $t3, 0(&x)
add $t3, $t1, &t3 // $t3 = addr (x[j])
lw $t4, 0($t3)     // $t4 = value x[j]
lw $t2, 4($sp)     // $t2 = a
mul $t2, $t4, $t2 // $t2 = $t1 * $t2
sw $t2, 4($sp)     // a = $t2
```



Control Instructions

Given 32 bit instructions (MIPS)

- Implicit control on each instruction
  - PC ← PC + 4
- Unconditional jumps
  - PC ← X (direct)
  - PC ← PC + X (PC relative)
  - X can be constant or register
- Conditional jumps (branches)
  - PC ← PC + ((cond) ? X : 4)
- Conditions
  - flags
  - in a register
  - fused compare and branch

Other architectures

- Predicated instructions

```
// basic loop control
//  while (j < 100)

LOOP:   …..
        LW      $t0, 0($sp)
        ADDI    $t0, $t0, 1
        SLTI    $t1, $t0, 100
        BNE     $t1, $zero, LOOP
```


Branch Instructions & Addressing (beq, bne, j, jal, jr)

Opcode, two registers, target address

beq: branch equal

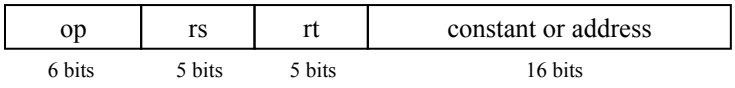
if ($rs = $rt) PC = label

bne: branch not equal

if ($rs != $rt) PC = label

```
// basic loop:  while (j < 100)

LOOP:   …..
        LW      $t0, 0($sp)
        ADDI    $t0, $t0, 1
        SLTI    $t1, $t0, 100
        BNE     $t1, $zero, LOOP
```



Most branch targets are near branch

Forward or backward

PC- relative addressing

Target address = PC + offset * 4

PC implicitly incremented by 4 after every instruction!

12

Branch Pseudo-instructions (blt, bgt, ble, bge)

blt: branch less than

if ($rs < $rt) PC = label

bgt: branch greater than

if ($rs > $rt) PC = label

ble: branch less than or equal

if ($rs <= $rt) PC = label

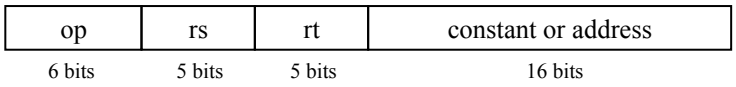
bge: branch greater than or equal

if ($rs >= $rt) PC = label

ELSE PC = PC + 4 // Implicit

```
// loop control: while (j < a)

LOOP:   …..
        LW      $t0, 0($sp)
        LW      $t1, -4($sp)
        BLT     $t0, $t1, LOOP
```



13

Branch Pseudo-instructions (blt, bgt, ble, bge)

You write

```
// loop control: while (j < a)

LOOP:   …..
        LW      $t0, 0($sp)
        LW      $t1, -4($sp)
        BLT     $t0, $t1, LOOP
```

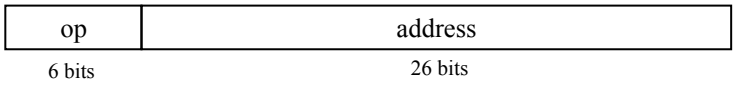
You get from the assembler

```
LOOP:   …..
        LW      $t0, 0($sp)
        LW      $t1, -4($sp)
        SLT     $t3, $t0, $t1
        BNE     $t3, $zero, LOOP
```

14

Jump Addressing

- Jump (`j` and `jal`) targets could be anywhere in text segment
  - Encode full address in instruction



- Direct jump addressing
  - Target address = $PC_{31\ldots28}$: (address × 4)

Target Addressing Example

- Simple Loop code
  - Assume Loop at location 80000

```
Loop: sll  $t1, $s3, 2      80000
      add  $t1, $t1, $s6    80004
      lw   $t0, 0($t1)      80008
      bne  $t0, $s5, Exit   80012
      addi $s3, $s3, 1      80016
      j    Loop             80020
Exit: …                     80024
```

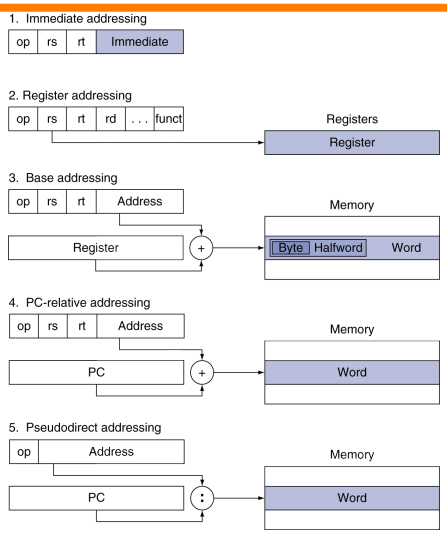
0	0	19	9	4	0	
0	9	22	9	0	32	
35	9	8	0			
5	8	21	2			
8	19	19	1			
2	20000					

Branching Far Away

- If branch target is too far to encode with 16-bit offset, assembler rewrites the code
- Example

```
        beq $s0,$s1, L1
          ↓
        bne $s0,$s1, L2
        j L1
L2:     …
```


MIPS Addressing Modes



Summary

- ISA III
  - Registers, Memory, Control
  - MIPS assembly for control, arrays, basic blocks
  - registers
- Next Time
  - Homework #2 is due 2/9
  - Procedures & more about MIPS assembly
  - Other ISAs
- Reading: P&H 2.16-19